



Vatroslav Zuppa Bakša
Domagoj Tuličić

UVOD U PROGRAMIRANJE

udžbenik za 1. razred
strukovnih škola

Ministarstvo znanosti, obrazovanja i
mladih odobrilo je korištenje udžbenika
i uvrstilo ga u Katalog rješenjem (klasa:
UP/I-602-09/26-03/00025,
ur. broj: 533-06-26-0002), lipanj 2026.

Udžbenik je pisan prema strukovnom
kurikulu iz 2025. g.

Masa udžbenika

760 ± 2σ g

Maloprodajna cijena

24,50 €

Intelektualno je vlasništvo, poput svakog
drugog vlasništva, neotuđivo, zakonom
zaštićeno i mora se poštovati. Nijedan
dio ove knjige ne smije se preslikavati niti
umnažati na bilo koji način, bez pisanog
dopuštenja nakladnika.

ISBN 978-953-250-305-0

© Vatroslav Zuppa Bakša, mag. ing. el. techn. inf.
Domagoj Tuličić, mag. inf., 2026.

Glavna urednica

Andrea Katarić Ratković, mag. educ. math.

Recenzentica

Željana Giljanović, mag. ing. el.

Lektorica

Dunja Apostolovski, prof.

Grafička urednica

Monika Zimonja
Dinka Volović, dipl. ing.

Dizajn korica

Hana Krajčević, mag. art.

Crteži

Element d. o. o., Zagreb
izvor: shutterstock.com

Nakladnik

Element d. o. o.
Zagreb, Ulica Šiška Menčetića 2
element.hr
element@element.hr

1. izdanje
Zagreb, 2026.

Tisak

lipanj 2026.
Element d. o. o., ZAGREB

Predgovor

Udžbenik **Uvod u programiranje** u potpunosti obuhvaća skupove ishoda učenja modula *Uvod u programiranje*. Namijenjen je učenicima strukovnih škola koji se obrazuju za stjecanje kvalifikacija u sektoru Elektrotehnika i računarstvo: tehničar za programiranje / tehničarka za programiranje (041626), tehničar za računarstvo / tehničarka za računarstvo (041625), tehničar za razvoj videoigara / tehničarka za razvoj videoigara (041725) te tehničar za informacijske tehnologije / tehničarka za informacijske tehnologije (041627). Modul obuhvaća dva skupa ishoda učenja: *Osnove programiranja* i *Spremnički tipovi podataka*.

Prva skupina ishoda učenja postupno uvodi učenike u algoritamski način razmišljanja. Najprije uče razumjeti problem i izrađivati rješenja s pomoću dijagrama toka u alatu **Raptor**. Nakon toga slijedi implementacija programa u **programskom jeziku C**.

U drugoj skupini ishoda učenja znanja se nadograđuju radom sa spremničkim tipovima podataka, uz primjenu **programskog jezika C++**. Njegova primjena omogućuje jednostavniji i pregledniji rad sa složenijim strukturama podataka, a istovremeno učenicima pruža prirodno proširenje prethodno stečenih znanja programiranja.

Programski jezici C i C++ u udžbeniku se obrađuju upotrebom razvojnog okruženja **Code::Blocks** koje je prilagođeno početnicima i omogućuje jednostavno prevođenje i izvođenje programa. Kao suvremenija alternativa prikazano je i razvojno okruženje Visual Studio Code.

Struktura udžbenika

Udžbenik je strukturiran u dvanaest poglavlja.

Poglavlja od 1. do 5. obuhvaćaju ishode učenja skupa **Osnove programiranja** u kojima učenici primjenjuju dijagrame toka i programski jezik C.

Poglavlja od 6. do 12. obuhvaćaju ishode učenja skupa **Spremnički tipovi podataka** gdje se znanja proširuju i produbljuju u radu s poljima, vektorima, listama, stogom i redom u programskom jeziku C++.

U nastavku su nabrojani elementi koji čine svako poglavlje.



Strip

Predstavlja uvod u lekciju prikazom problemske ili svakodnevnice situacije koja uvodi učenika u temu i potiče interes.



Digitalni rudnik kodova

Digitalni repozitorij programa gdje učenici mogu preuzeti, pokrenuti i analizirati gotove primjere te ih upotrebljavati kao polazište za daljnji rad.



Razmisli

Navode se pitanja za aktivaciju predznanja i razvoj logičkog razmišljanja prije učenja novog sadržaja.



Nakon ove lekcije moći ću

Jasno su definirani ishodi učenja koji učenicima daju pregled očekivanih kompetencija.

Teorijski pregled i riješeni primjeri

Dio lekcije u kojem se sustavno objašnjavaju novi pojmovi, programske konstrukcije i načini njihove primjene. Teorijska objašnjenja popraćena su riješenim primjerima koji učenicima korak po korak prikazuju način analize problema, izrade algoritma i implementacije programskog rješenja.

Uz riješene primjere nalaze se i dvije vrste zadataka s naglaskom na povezivanje teorije i praktične primjene te postupnom razumijevanju programskih koncepata.



- *Proširite program* gdje učenik nadograđuje postojeće rješenje novim funkcionalnostima. Cijeli program može preuzeti putem QR koda ili poveznice iz Digitalnog rudnika kodova koji se nalazi na početku lekcije.
- *Izmijenite program* gdje učenik samostalno piše program od početka i provodi potrebne izmjene na temelju prikazanog primjera. Za razliku od zadataka *Proširite program*, učeniku nije ponuđen gotov kôd putem QR koda, već program piše i prilagođava samostalno na temelju prikazanog primjera.



Video simulacija

Kada se pojavi ovaj simbol, učenik/ca se upućuje na kratki videozapis s prikazom koraka rješavanja ili objašnjenjem postupka. Skeniranjem QR koda može se otvoriti video i pratiti demonstracija korak po korak.



Vrijeme za izazov

Donosi problemske zadatke za provjeru usvojenosti ishoda učenja.

Vrednovanje kao učenje



Vrednovanje kao učenje

Učenik/ca se upućuje na samoprocjenu uz digitalne kartice dostupne putem QR koda.

Sažetak

Donosi pregled temeljnih pojmova i koncepata.



Pitanja za ponavljanje

Upućuje se na provjeru usvojenosti sadržaja. U 1. dijelu su podijeljeni na osnovnu, srednju i naprednu razinu (semafor), a u 2. i 3. dijelu su usmjereni na teorijsko razumijevanje i primjenu pojmova.

Dodatak

Ova rubrika obuhvaća sadržaje povezane s primjenom umjetne inteligencije u učenju i razvoju programa.

Udžbenik potiče aktivno učenje, razvoj logičkog i analitičkog razmišljanja te postupno osamostaljivanje učenika u rješavanju programskih problema. Uz udžbenik je izrađena **radna bilježnica** s dodatnim zadacima za vježbu i utvrđivanje gradiva. U potpunosti je usklađena s udžbenikom i ishodima učenja modula.

Nakon usvajanja obiju skupina ishoda učenja učenici razvijaju temelje algoritamskog razmišljanja te stječu osnovna znanja rada u alatu **Raptor i programskim jezicima C i C++**.

Nadamo se da će jasnoća i preglednost sadržaja pomoći učenicima u uspješnom svladavanju ishoda učenja, razvijanju logičkog i analitičkog razmišljanja te primjeni stečenih znanja u praktičnim situacijama.

Autori

element.hr

Sadržaj

I. dio: OSNOVE PROGRAMIRANJA, 3 CSVET

1. Algoritmi i dijagrami toka (Raptor)	2
1.1. Algoritam i program	4
1.2. Pseudokod, dijagram toka i instalacija programa Raptor	4
1.3. Pokretanje prvog programa u Raptoru	6
1.4. Osnovna pravila prilikom pisanja algoritama u programu Raptor	7
1.4.1. Varijable, naredba pridruživanja i ispis vrijednosti na zaslonu računala	7
1.4.2. Operatori, konstante i funkcije u Raptoru	10
1.5. Osnovne algoritamske strukture zapisane u pseudokodu te prikazane dijagramom toka u programu Raptor	11
Vrijeme za izazov	18
2. Osnove programiranja u C-u	20
2.1. Uvod u programski jezik C	22
2.1.1. Pisanje prvog programa i pokretanje programa s pomoću razvojnog okruženja Code::Blocks	23
2.1.2. Pokretanje programa s pomoću razvojnog okruženja Visual Studio Code	27
2.1.3. Github Copilot	29
2.1.4. Standardi programskog jezika C	30
2.2. Osnovna struktura programskog jezika C	30
2.3. Varijable i konstante	32
2.3.1. Varijable	32
2.3.2. Konstante	36
2.4. Izrazi, aritmetički operatori i ugrađene matematičke funkcije	38
2.5. Funkcija za unos podataka	39
2.6. Funkcija za ispis podataka	40
2.7. Upute vezane za pisanje programa	42
2.8. Linijska struktura programa	45
Vrijeme za izazov	50
3. Tipovi podataka i operatori	52
3.1. Prikaz podataka u računalu	54
3.1.1. Brojevi sustavi	54
3.1.2. Rastavljanje brojeva po težinama	55

3.1.3. Pretvorbe brojevnih sustava	56
3.1.4. Operacija zbrajanja s binarnim brojevima	59
3.1.5. Pojam količine podataka	60
3.1.6. Prikaz cijelih i realnih brojeva te znakova u memoriji računala	62
3.2. Operatori	73
3.2.1. Operator pridruživanja	74
3.2.2. Aritmetički operatori	74
3.2.3. Relacijski operatori	76
3.2.4. Logički operatori	78
3.2.5. Operatori inkrement i dekrement	80
3.2.6. Operator sizeof	81
3.2.7. Bitovni operatori	82
3.2.8. Skraćeni oblik naredbe pridruživanja	85
3.2.9. Prioritet operatora	86
3.3. Osnovni tipovi podataka	88
3.3.1. Cjelobrojni tipovi podataka	89
3.3.2. Brojevi s pomičnim zarezom	94
3.4. Konverzija tipova podataka	95
3.4.1. Implicitna konverzija	95
3.4.2. Eksplicitna konverzija	95
Vrijeme za izazov	100
4. Grananje u programu	102
4.1. Jednostrana selekcija if	104
4.2. Dvostruka selekcija if else	107
4.3. Višestruka selekcija if else if	109
4.4. Višestruka selekcija switch case	112
4.5. Primjena ternarnog operatora	118
Vrijeme za izazov	120
5. Programske petlje	122
5.1. Ciklička algoritamska struktura (programske petlje)	124
5.1.1. Petlja kod koje se uvjet ispituje na početku	124
5.1.2. Petlja kod koje se uvjet ispituje na kraju	126
5.1.3. Petlja s poznatim brojem ponavljanja	128
5.1.4. Usporedba petlji i odabir odgovarajuće petlje	130
5.2. Primjena cikličke algoritamske strukture	132
5.3. Ugniježdene petlje	139
5.3.1. Pojam beskonačne petlje	140
5.4. Primjena naredbi break i continue	142
5.5. Primjena operatora zarez (,) u for petlji	144
Vrijeme za izazov	148
Sažetak	150
Pitanja za ponavljanje	155

II. dio: SPREMNIČKI TIPOVI PODATAKA, 3 CSVET

6. Uvod u programski jezik C++	162
6.1. Prvi program u programskom jeziku C++	165
6.1.1. Kreiranje i pokretanje prvog C++ programa u Code::Blocks okruženju	165
6.1.2. Upotreba imenskog prostora (<i>namespace</i>)	168
6.2. Povezanost programskih jezika C i C++	168
6.3. Unos i ispis podataka u programskom jeziku C++	170
6.3.1. Ispis podataka (<i>cout</i>)	170
6.3.2. Unos podataka (<i>cin</i>)	172
6.4. Standardne biblioteke u C++-u	175
6.4.1. Oblikovanje ispisa u C++-u	175
6.5. Jednostavni i složeni tipovi podataka	179
6.5.1. Jednostavni tipovi podataka	179
6.5.2. Podatak tipa <i>string</i>	180
6.5.3. Složeni tipovi podataka	182
Vrijeme za izazov	184
7. Polja (nizovi)	186
7.1. Jednodimenzionalna polja	189
7.1.1. Deklaracija i inicijalizacija jednodimenzionalnog polja	189
7.1.2. Ispis vrijednosti elemenata polja	191
7.1.3. Učitavanje elemenata polja s pomoću petlje	194
7.2. Dvodimenzionalna polja	197
7.2.1. Deklaracija i inicijalizacija dvodimenzionalnog polja	198
7.2.2. Ispis i učitavanje elemenata dvodimenzionalnog polja	201
7.2.3. Dvodimenzionalna polja s jednakim brojem redaka i stupaca	205
Vrijeme za izazov	210
8. Funkcije, klase i metode u C++-u	212
8.1. Rad s funkcijama	214
8.1.1. Prototip funkcije	216
8.1.2. Gdje se piše funkcija?	217
8.1.3. Naredba <i>return</i>	217
8.1.4. Poziv funkcije	219
8.1.5. Lokalne i globalne varijable	223
8.2. Reference	226
8.3. Prosljeđivanje parametara u funkcijama	229
8.4. Prosljeđivanje polja funkcijama	234
8.4.1. Prosljeđivanje jednodimenzionalnog polja	234
8.4.2. Prosljeđivanje dvodimenzionalnog polja	236
8.5. Uvod u klase i metode	240
8.5.1. Opći prikaz klase i objekta	241
Vrijeme za izazov	248
9. Vektori	250
9.1. Uvod u vektore	252
9.2. Osnove rada s vektorima u programskom jeziku C++	252
9.2.1. Deklaracija i inicijalizacija vektora	252
9.2.2. Prolazak elementima vektora	255

9.2.3. Dodavanje i uklanjanje elemenata vektora	257
9.2.4. Unos elemenata u vektor	260
9.2.5. Pristup elementima vektora	262
9.2.6. Veličina i kapacitet vektora	263
9.2.7. Najčešće metode spremnika <code>std::vector</code>	266
9.2.8. Primjena vektora	268
Vrijeme za izazov	270
10. Lista	272
10.1. Uvod u liste	274
10.2. Osnove rada s listama u programskom jeziku C++	276
10.2.1. Deklaracija i inicijalizacija liste	276
10.2.2. Pristup elementima liste	279
10.2.3. Dodavanje i uklanjanje elemenata liste	283
10.2.4. Unos elemenata u listu	286
10.2.5. Veličina liste	291
10.3. Primjena lista	292
Vrijeme za izazov	298
11. Stog	300
11.1. Uvod u stog (engl. <i>stack</i> , LIFO)	302
11.2. Deklaracija i inicijalizacija stoga u programskom jeziku C++	302
11.3. Dodavanje i uklanjanje elemenata stoga	304
11.4. Metode za prikaz trenutačnog stanja i veličine stoga	308
11.5. Primjena stoga	310
Vrijeme za izazov	314
12. Red	316
12.1. Uvod u strukturu FIFO	318
12.2. Deklaracija i inicijalizacija reda u programskom jeziku C++	319
12.3. Dodavanje i uklanjanje elemenata reda	322
12.4. Trenutačno stanje i veličina reda	325
12.5. Primjer programa: rad s redom	327
Vrijeme za izazov	332
Sažetak	334
Pitanja za ponavljanje	341
Dodatak 1 – GitHub Copilot	349
Dodatak 2 – GPT Uvod u programiranje	352
Literatura	355
Kazalo pojmova	357

I. dio

OSNOVE PROGRAMIRANJA

1. Algoritmi i dijagrami toka (Raptor)
2. Osnove programiranja u C-u
3. Tipovi podataka i operatori
4. Grananje u programu
5. Programske petlje

3 CSVET

Potrebni alati za rad:

- Raptor
 - Code::Blocks
 - Visual Studio Code (VS Code) – opcionalno razvojno okruženje
 - GCC compiler (MinGW)
 - GitHub Copilot i GPT – alati umjetne inteligencije za pomoć pri učenju i programiranju (dodatak)
- Instalacijske upute i poveznice za preuzimanje programa nalaze se u odgovarajućim poglavljima.

1.

ALGORITMI I DIJAGRAMI TOKA (RAPTOR)



DIGITALNI RUDNIK KODOVA

Skeniraj QR kôd ili otvori poveznicu u internetskom pregledniku. Prouči primjere 1.1, 1.2 i 1.3 te ih analiziraj i proširi programe prateći upute iz tih primjera.

<https://s.element.hr/ba54>





Razmisli

Opiši postupak pranja zubi u nekoliko koraka.
Postoji li redoslijed koji treba slijediti prilikom pranja zubi?

Moraš li u nekim situacijama odlučiti koji korak slijedi, primjerice ako nema paste za zube?

Treba li neke korake ponavljati dok zubi ne budu čisti?



Nakon ove lekcije moći ćeš



- objasniti što je **algoritam** i navesti načine njegova zapisa
- razlikovati **pseudokod** i **dijagram toka**
- zapisati **algoritam** s pomoću **pseudokoda**
- prepoznati i primjenjivati osnovne **naredbe** (ulaz, izlaz, pridruživanje)
- upotrebljavati **varijable**, **operatore** i **osnovne funkcije**
- primijeniti **grananje** i **petlje** u algoritmima
- interpretirati jednostavan pseudokod i odrediti rezultat
- izraditi **dijagram toka** u programu Raptor.



1.1. Algoritam i program

Pisanje i osmišljavanje programskog koda prema kojem će raditi neki program posao je **programera**. Prije samog pisanja koda programer prvo treba definirati problem te ulazno/izlazne veličine vezane za zadani problem. Na primjer, ako programira sustav za kontrolu pametne rasvjete u kućanstvu, ulazne veličine mogu biti senzori pokreta i svjetla putem kojih sustav dobiva informacije o prisutnosti ljudi i vanjskoj svjetlosti. Izlazna veličina bila bi upravljanje svjetlima u kući – paljenje, gašenje ili prilagođavanje jačine svjetla ovisno o uvjetima i potrebama korisnika.

Potom kad je definiran problem, programer piše algoritam i tek potom osmišljeni algoritam realizira u nekom od programskih jezika. **Algoritam** predstavlja konačan skup jednoznačno definiranih, logički povezanih koraka. Ima jasno definirane ulazne i izlazne veličine te mora davati točan izlaz za sve njegove dopuštene ulaze.

Algoritme je moguće zapisati na nekoliko načina: **u obliku teksta**, **pseudokodom**, s pomoću **dijagrama toka** ili **u nekom odabranom programskom jeziku**. Započet ćemo s prikazom algoritama s pomoću dijagrama toka u programu **Raptor** te paralelno u pseudokodu, a potom će ti primjeri biti realizirani u programskim jezicima C i C++.

1.2. Pseudokod, dijagram toka i instalacija programa Raptor

U **pseudokodu** se zapisuju naredbe koje čine jednu logičku cjelinu te kojima je predstavljen algoritam. U tim naredbama često su uključeni i izrazi koji podsjećaju na matematičke formule.

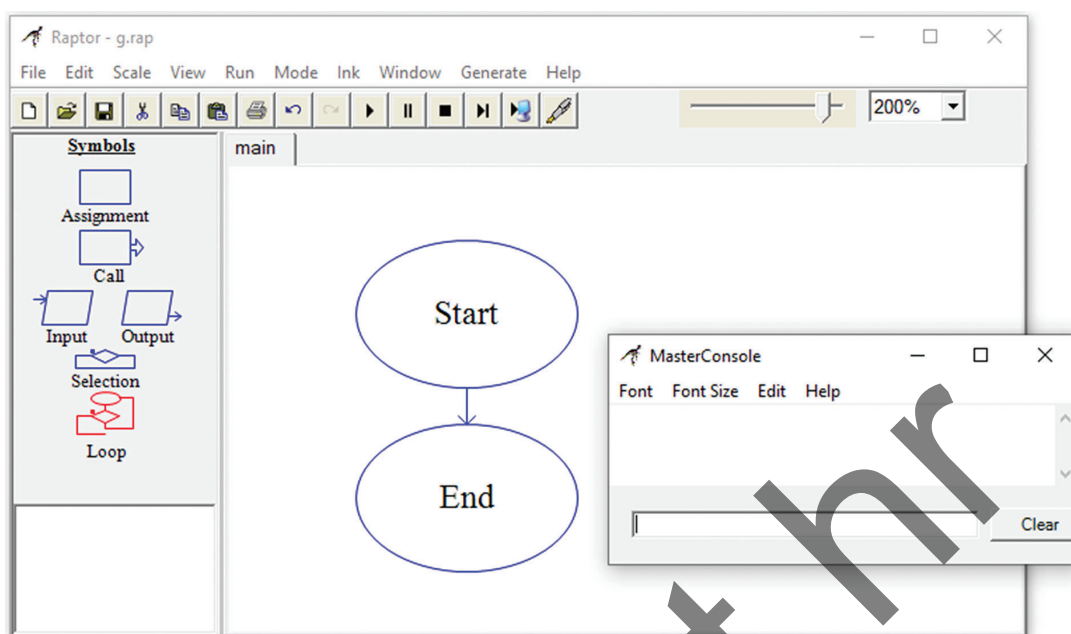
Dijagram toka grafički prikazuje kako se odvija algoritam i to korak po korak. Dijagrame toka moguće je crtati u različitim programima kao što su primjerice Word, draw.io i sl. Za simuliranje izvođenja algoritma u ovom je udžbeniku za crtanje dijagrama toka odabran program Raptor.

Program je moguće preuzeti s poveznice:

<https://raptor.martincarlisle.com/>.



Nakon jednostavne instalacije te pokretanjem programa dobiva se prikaz kao na slici 1.1. U desnom kutu nalazi se padajući izbornik s mogućnošću povećavanja i smanjivanja veličine prikaza dijagrama (trenutačno je odabran prikaz na 200 %).

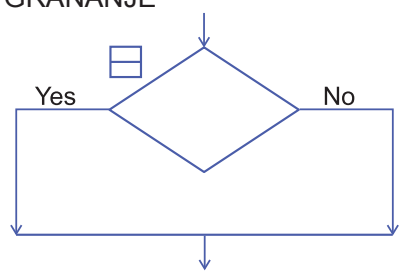
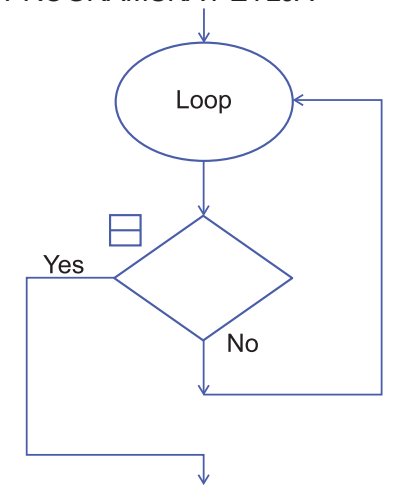
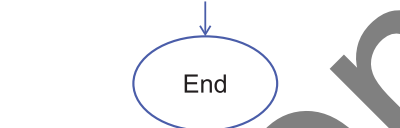


Slika 1.1. Program Raptor

Simboli koji se upotrebljavaju prilikom izrade dijagrama toka te odgovarajuće naredbe u pseudokodu prikazani su u tablici 1.1.

Tablica 1.1. Pregled i opis simbola u programu Raptor

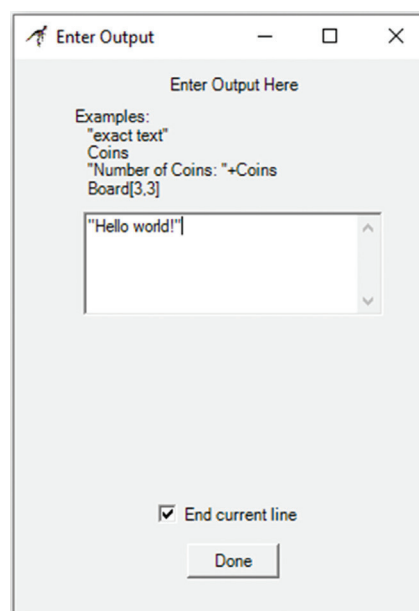
simboli	naredba u pseudokodu	opis
	<u>početak</u>	početak algoritma
	<u>ulaz()</u>	definiranje ulaznih veličina
	<u>izlaz()</u>	definiranje izlaznih veličina
	varijabla = izraz	obrada podataka, pisanje naredbi

simboli	zapis u pseudokodu	opis
GRANANJE 	<pre> ako_je uvjet onda { naredba_1 } inače { naredba_2 } </pre>	odluka se donosi na temelju uvjeta unutar romba, a izvršavanje se nastavlja jednim od dvaju mogućih smjerova (unutar romba piše se logički uvjet - primjeri u tablici 1.3)
PROGRAMSKA PETLJA 	<pre> dok_je uvjet činiti { blok_naredbi } ponavljati { blok_naredbi } dok_je uvjet </pre>	- naredbe se ponavljaju dok uvjet izlaska nije zadovoljen; kada uvjet postane zadovoljen, izlazi se iz petlje - u dijagramu toka koristi se uvjet izlaska iz petlje, dok se u pseudokodu koristi logički suprotan uvjet nastavka petlje
	kraj	kraj algoritma

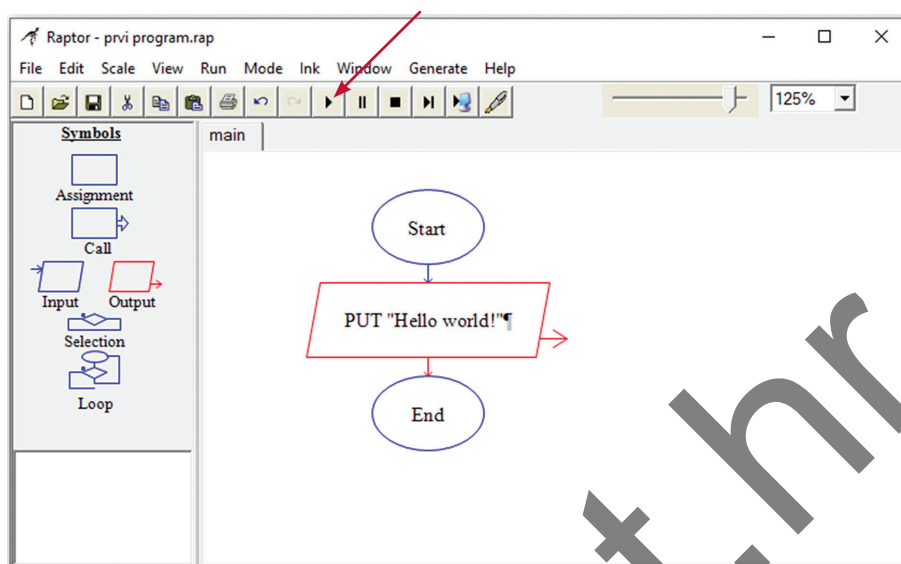
1.3. Pokretanje prvog programa u Raptoru

Slika 1.1 prikazuje izgled programa nakon pokretanja. Za prvo testiranje upotrebljava se jednostavan algoritam koji ispisuje poruku: **“Hello world!”**. U lijevom izborniku, među dostupnim simbolima potrebno je odabrati simbol za definiranje izlaznih podataka (*Output*). Taj simbol povuče se i postavi između eliptičnih simbola za početak (*Start*) i kraj (*End*) programa. Nakon što je simbol postavljen na radnu površinu, dvaput se klikne na njega kako bi se otvorilo polje za unos. U to polje unosi se željena poruka, unutar dvostrukih navodnika, koja će se prikazati na zaslonu računala (slika 1.2).

Slika 1.2. Upis teksta unosi se unutar dvostrukih navodnika

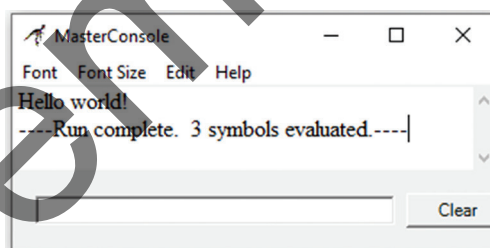


Klikom na *Done* te spremanjem programa pod nazivom *prvi program.rap* dobiva se prikaz kao na slici 1.3.



Slika 1.3. Izrada prvog dijagrama toka

Klikom na *Execute to Completion* (na taj simbol pokazuje strelica) pokreće se simulacija. Rezultat simulacije vidljiv je unutar prozora *MasterConsole* (slika 1.4). S pomoću klizača u gornjem desnom kutu radnog prozora moguće je definirati brzinu izvođenja simulacije. Prilikom izvođenja programa svjetlozelenom bojom uvijek je označen trenutačni simbol unutar kojeg se izvršavaju naredbe. Tako se lakše prati tok izvođenja algoritma.



Slika 1.4. Prikaz rezultata izvođenja programa

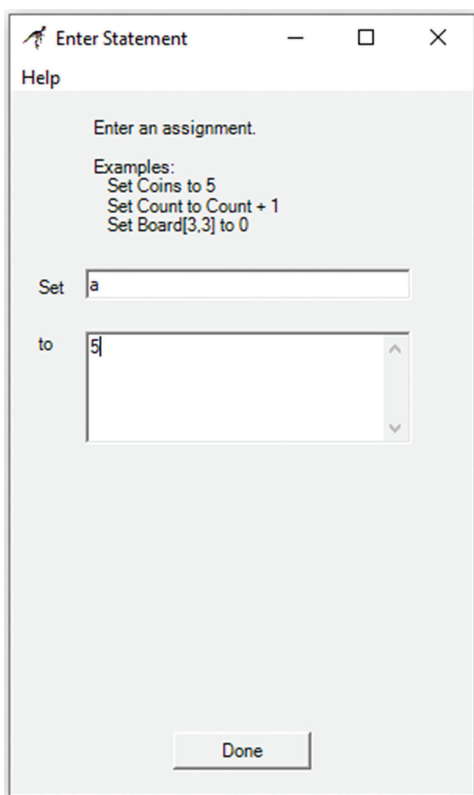
1.4. Osnovna pravila prilikom pisanja algoritama u programu Raptor

1.4.1. Varijable, naredba pridruživanja i ispis vrijednosti na zaslonu računala

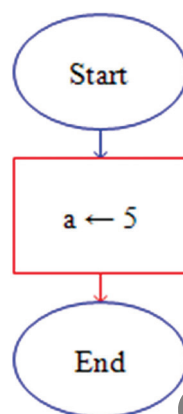
Prilikom pisanja programa u pravilu uvijek treba **definirati i inicijalizirati varijable**. Svaka varijabla ima svoju memorijsku lokaciju, a sadržava vrijednost koja joj se pridružuje.

Naredba pridruživanja radi se s pomoću bloka *Assignment*. Povlačeći blok na željenu poziciju u dijagramu toka, dva se puta klikne te se napiše naziv varijable koja se inicijalizira na željenu vrijednost (slika 1.5). Rezultat pridruživanja prikazuje slika 1.6.

1. Algoritmi i dijagrami toka (Raptor)

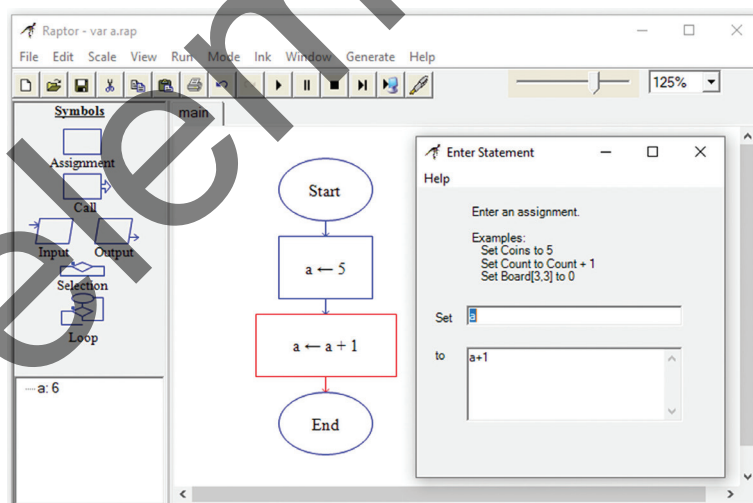


Slika 1.5. Definiranje i inicijaliziranje varijable



Slika 1.6. Prikaz definirane i inicijalizirane varijable *a* (naredba pridruživanja)

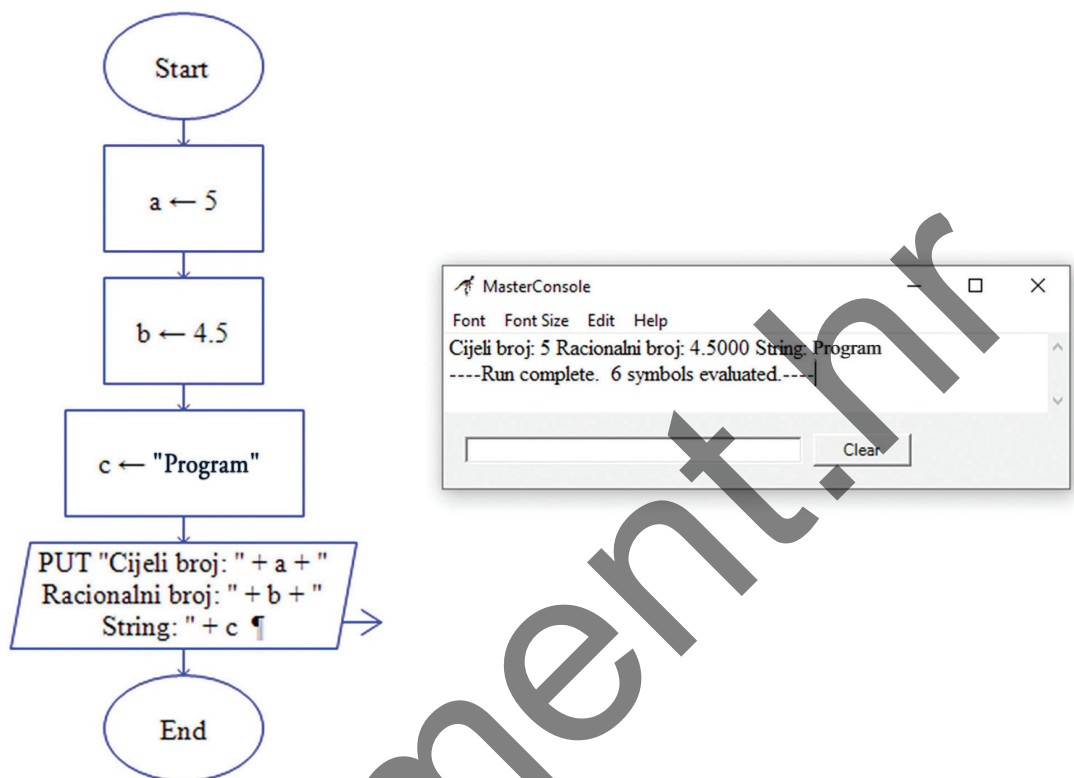
Ako želimo vrijednost varijable *a* uvećati za 1, može se dodati i crvenom bojom označen blok *Assignment* kao što prikazuje slika 1.7.



Slika 1.7. Primjer naredbe pridruživanja

Osim cjelobrojnih vrijednosti mogu se rabiti i numeričke vrijednosti s decimalnom točkom (realni brojevi) te nizovi znakova. Za ispis željenih vrijednosti upotrebljava se blok *Output*, što prikazuje slika 1.8. Nakon što dodamo blok *Output* na radnu površinu, dvaput se klikne na njega kako bi se otvorilo polje za unos. U to polje upisuje se tekst koji se želi

prikazati, unutar dvostrukih navodnika, a zatim se dodaje znak + i naziv varijable čija se vrijednost treba ispisati. Ako je potrebno upisati dodatni tekst, ponovno se upiše znak +, nakon čega se nastavlja unositi željeni tekst unutar dvostrukih navodnika. Ovim postupkom moguće je kombinirati tekst i vrijednosti varijabli u jednom ispisu.



Slika 1.8. Ispis svih vrijednosti varijabli na zaslonu računala

Algoritam prikazan na slici 1.8. s pomoću pseudokoda može se zapisati na sljedeći način:

```
početak
a = 5
b = 4.5
c = "Program"

izlaz("Cijeli broj: ", a)
izlaz("Racionalni broj: ", b)
izlaz("String: ", c)
kraj
```

Napomena

U pseudokodu se upotrebljava znak = za pridruživanje vrijednosti varijabli, dok se u aplikaciji Raptor za istu operaciju upotrebljava strelica (←).

1.4.2. Operatori, konstante i funkcije u Raptoru

Neki od osnovnih matematičkih operatora i funkcija koji se upotrebljavaju prilikom izrade početnih dijagrama prikazani su tablicom 1.2.

Tablica 1.2. Pregled operatora i matematičkih funkcija

opis		primjer pridruživanja	vrijednost sadržana u varijabli x
osnovna matematika	zbrajanje +	$x \leftarrow 2 + 3$	5
	oduzimanje -	$x \leftarrow 2 - 3$	-1
	množenje *	$x \leftarrow 2 * 3$	6
	dijeljenje /	$x \leftarrow 2 / 3$	0.6667
	potenciranje ** ili ^	$x \leftarrow 2 ** 3$ $x \leftarrow 2 ^ 3$	8
	cjelobrojno dijeljenje s ostatkom mod	$x \leftarrow 2 \text{ mod } 3$	2
	korjenovanje sqrt(argument)	$x \leftarrow \text{sqrt}(2)$	1.4142
	logaritmiranje po bazi e log(argument)	$x \leftarrow \text{log}(2)$	0.6931
	apsolutna vrijednost abs(argument)	$x \leftarrow \text{abs}(-2)$ $x \leftarrow \text{abs}(2)$	2
	zaokruživanje na najbliži veći cijeli broj ceiling(argument)	$x \leftarrow \text{ceiling}(3.14)$ $x \leftarrow \text{ceiling}(4)$	4
	zaokruživanje na najbliži manji cijeli broj floor(argument)	$x \leftarrow \text{floor}(3.14)$ $x \leftarrow \text{floor}(3)$	3
	zaokruživanje na najbliži cijeli broj round(x)	$x \leftarrow \text{round}(3.6)$	4
trigonometrijske funkcije	sinus sin(argument) kosinus cos(argument) tangens tan(argument) kotangens cot(argument) *argument je kut u radijanima	$x \leftarrow \text{sin}(1.57)$	1.0000
konstante	broj pi – pi	$x \leftarrow \text{pi}$	3.1416
	broj e – e	$x \leftarrow \text{e}$	2.7183
	true	$x \leftarrow \text{true}$	1
	false	$x \leftarrow \text{false}$	0
	yes	$x \leftarrow \text{yes}$	1
	no	$x \leftarrow \text{no}$	0

Prilikom pisanja uvjeta u razgranatjoj i cikličkoj algoritamskoj strukturi u simbolu romba upotrebljavaju se **relacijski** i **logički operatori**. S pomoću relacijskih operatora određuje se odnos između dviju vrijednosti, a kao rezultat dobiva se **logička istina** (logička jedinica, *true*) ili **logička laž** (logička nula, *false*, logička neistina). S pomoću logičkih operatora moguće je povezati više izraza koji sadržavaju relacijske operatore. Na taj se način pišu složeni logički izrazi. Pregled i opis relacijskih i logičkih operatora nalazi se

u tablici 1.3. Navedeni operatori upotrebljavaju se kod pisanja **uvjeta**, kod razgranate i cikličke algoritamske strukture. U tablici 1.4. prikazan je redoslijed izvođenja operatora.

Tablica 1.3. Relacijski i logički operatori

	opis	primjer	rezultat
relacijski operatori	jednako ==	2 == 3	false
	različito != ili /=	2 != 3 2 /= 3	true
	manje od <	2 < 3	true
	veće od >	2 > 3	false
	veće od ili jednako >=	2 >= 3	false
	manje od ili jednako <=	2 <= 3	true
logički operatori	I and	false and false	false
		false and true	false
		true and false	false
		true and true	true
	II or	false or false	false
		false or true	true
		true or false	true
		true or true	true
	NE not	not(true)	false
		not(false)	true

Tablica 1.4. Redoslijed izvođenja operatora

	operator
1.	zagrade ()
2.	potenciranje (** ili ^)
3.	množenje i dijeljenje (*, /, mod)
4.	zbrajanje i oduzimanje (+, -)
5.	relacijski operatori (==, !=, <, >, <=, >=)
6.	logički operatori (not → and → or)

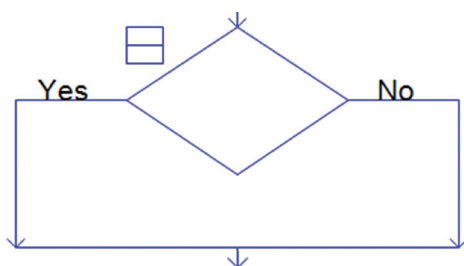
1.5. Osnovne algoritamske strukture zapisane u pseudokodu te prikazane dijagramom toka u programu Raptor

Program Raptor primjenjuje **tri osnovne algoritamske strukture**:

- a) **linijska** algoritamska struktura (slijed)
- b) **razgranata** algoritamska struktura (grananje)
- c) **ciklička** algoritamska struktura (petlja).

Kod **linijske algoritamske strukture** naredbe u obliku blokova navode se jedna za drugom te se algoritam izvršava **slijedno**.

Kod **razgranate algoritamske strukture** algoritam se **grana** te se izvršava blok naredbi u onoj grani kako je definirano uvjetom koji se nalazi unutar simbola romba. Ako je uvjet istinit, izvršava se grana Yes, a ako nije, onda se izvršava grana No. Uvjet se upisuje u romb tako da se dva puta klikne na njega. U praznom polju pišu se uvjeti s pomoću aritmetičkih, relacijskih, logičkih i ostalih operatora. U Yes i No grane dodaju se blokovi naredbi. Slika 1.9 prikazuje dio algoritma koji sadržava blok za grananje programa. Pored prikaza grananja u programu dijagramom toka nalazi se i odgovarajući zapis u pseudokodu.



Slika 2.9. Grananje

```
ako je uvjet onda {
    blok_naredbi_1
} inače {
    blok_naredbi_2
}
```

U pseudokodu može se upotijebiti i oblik:

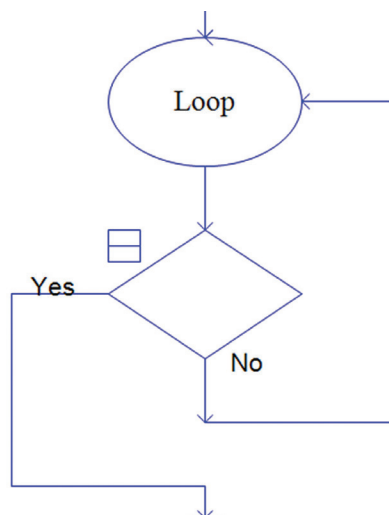
```
ako je uvjet onda {
    blok_naredbi
}
```

Dio **inače** nije obavezan te se izostavlja kada nije potrebno definirati naredbe za slučaj da uvjet nije ispunjen.

Napomena

Kada se u pseudokodu iza uvjetne naredbe izvršava samo jedna naredba, vitičaste zagrade nisu potrebne. Ako se nakon uvjeta nalazi **više od jedne naredbe**, potrebno je **upotrijebiti vitičaste zagrade { }** kako bi se označio blok naredbi koji pripada istom uvjetu.

Kod **cikličke algoritamske strukture** naredba ili blok naredbi **ponavljaju** se sve dok nije zadovoljen uvjet koji je naveden unutar simbola u obliku romba. Kad uvjet bude zadovoljen, prestaje se s ponavljanjem naredbi unutar petlje. Uvjet se zapisuje na isti način kao i kod razgranate algoritamske strukture, ali određuje nastavlja li se izvođenje petlje ili se petlja završava. Tijelo petlje čine svi blokovi koji se izvršavaju tijekom jednog ponavljanja petlje. Oni se mogu nalaziti prije ili nakon provjere uvjeta. Petlja se prepoznaje prema povratnoj vezi i ključnoj riječi *Loop* (slika 1.10).



Slika 1.10. Programska petlja

Napomena

U Raptoru se uvjet petlje zapisuje kao **uvjet izlaska**, dok se u pseudokodu i programskim jezicima upotrebljava **uvjet nastavka** izvršavanja petlje.

U nekim programskim jezicima ista se petlja može zapisati na više načina. U pseudokodu najčešće se navode sljedeća tri načina zapisa petlji, ovisno o tome kada se provjerava uvjet i je li broj ponavljanja unaprijed poznat:

1. petlja s unaprijed poznatim brojem ponavljanja

```
za i = p do k činiti {
    blok_naredbi
}
```

2. petlja kod koje nije unaprijed poznat broj ponavljanja, a uvjet se provjerava na početku petlje

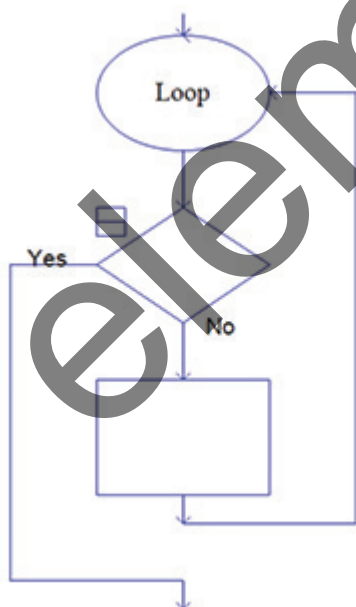
(ponavljanje dok je uvjet istinit – kontrola na ulazu), slika 1.11

```
dok je uvjet činiti {
    blok_naredbi
}
```

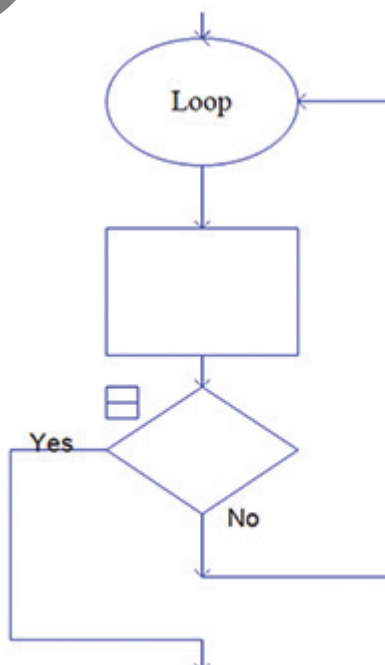
3. petlja kod koje nije unaprijed poznat broj ponavljanja, a uvjet se provjerava na kraju petlje

(ponavljanje barem jednom – kontrola na izlazu), slika 1.12

```
ponavljati {
    blok_naredbi
} dok je uvjet
```



Slika 1.11. Provjera uvjeta na početku petlje



Slika 1.12. Provjera uvjeta na kraju petlje

Prilikom rješavanja zadataka uvijek će biti definirani ulazni (test) podatci s pomoću kojih korisnik testira ispravnost programa. Također će biti prikazano i što se očekuje kao ispis na zaslonu računala. Program radi ispravno ako za sve dopuštene ulazne podatke daje točne rezultate na izlazu. Ovaj je koncept prikazan i objašnjen na sljedećim riješenim primjerima algoritama.

Primjer 1.1.

Nacrtajte u programu Raptor dijagram toka koji će izračunati opseg i površinu pravokutnika kojemu su poznate duljine stranica. Zapišite algoritam i s pomoću pseudokoda.

ulazni podatci	izlazni podatci
4 3	Opseg je: 14 Površina je: 12

Rješenje

Ovo je primjer algoritma linijske (slijedne) strukture. Komunikacijom korisnik – računalo pomaže se korisniku ispravno učitati tražene podatke. Potom se radi obrada učitanih podataka i ispisuje se rješenje u zadanom obliku na izlazu (slika 1.13).

```

početak
    izlaz("Ucitajte a")
    ulaz(a)

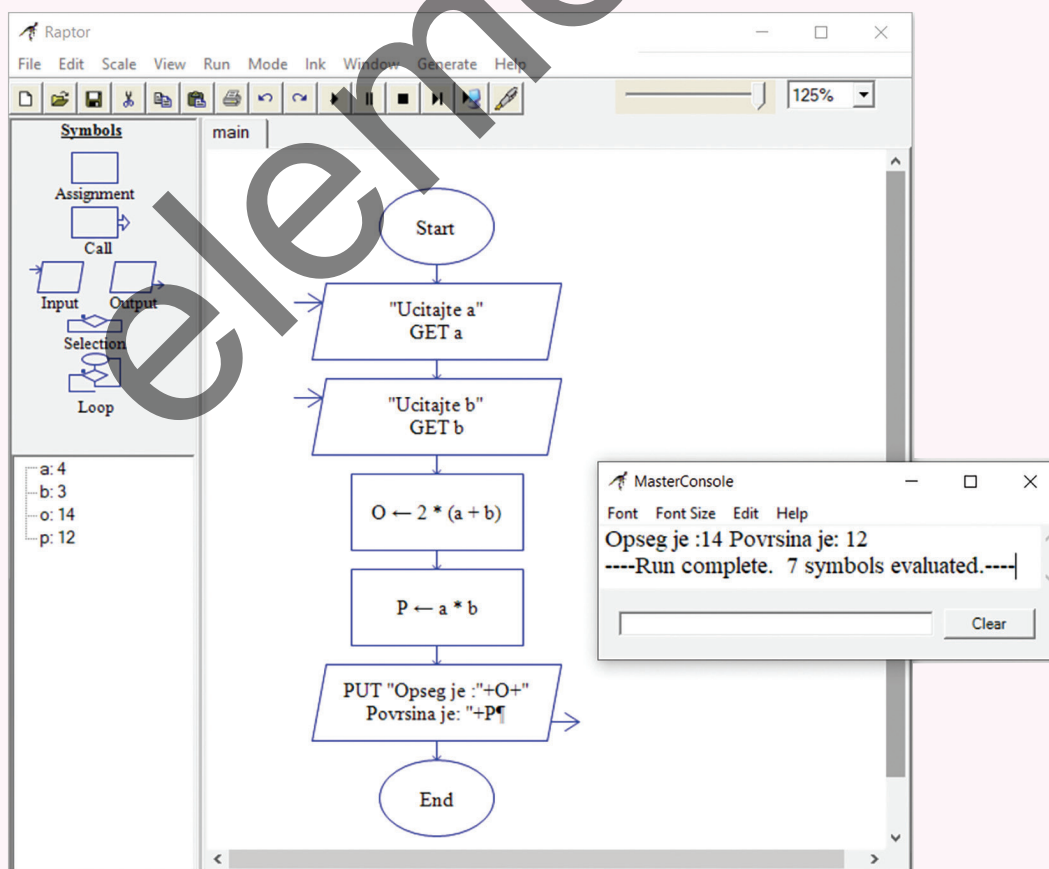
    izlaz("Ucitajte b")
    ulaz(b)

    O = 2 * (a + b)
    P = a * b

    izlaz("Opseg je: ",O)
    izlaz("Površina je: ",P)
kraj
  
```

Proširite program

Izračunajte i duljinu dijagonale pravokutnika i ispišite ju.



Slika 1.13. Primjer linijske strukture programa

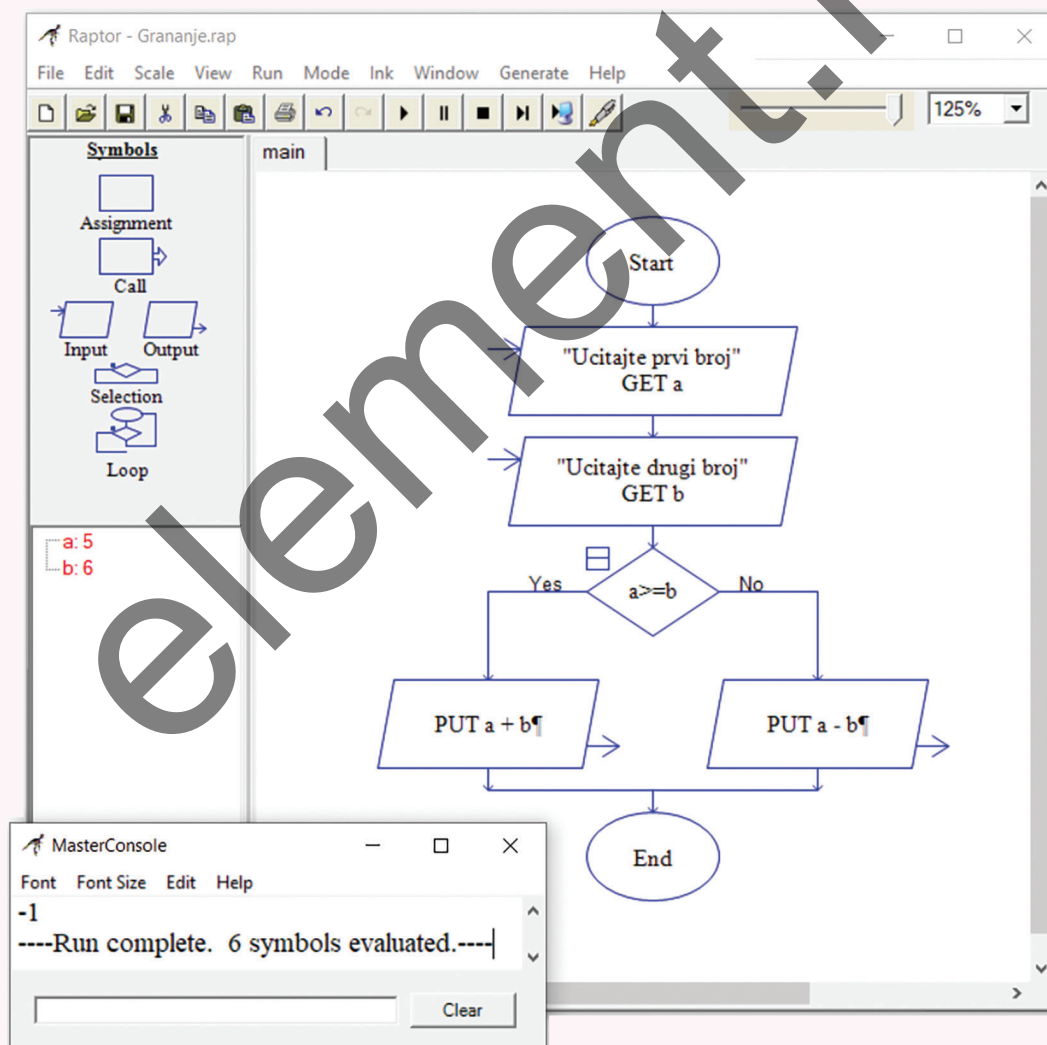
Primjer 1.2.

Nacrtajte u programu Raptor dijagram toka koji će ispisati na zaslonu računala zbroj dvaju brojeva ako je prvi učitani broj veći od ili jednak drugome. U suprotnom treba ispisati njihovu razliku. Zapišite algoritam i s pomoću pseudokoda.

ulazni podatci	izlazni podatci
5	-1
6	
6	11
5	

Rješenje

Ovo je primjer algoritma razgranate strukture (slika 1.14.). U uvjetu se upotrebljava relacijski operator $\geq$. Prvi testni primjer (5, 6) ispisuje razliku učitanih brojeva jer uvjet kao rezultat daje logičku vrijednost `false` ($5 \geq 6$) te se algoritam nakon učitavanja podataka nastavlja izvršavati u grani `No`. Ako se učit drugi testni primjer (6, 5), algoritam ispisuje zbroj učitanih brojeva.



Slika 1.14. Primjer razgranate algoritamske strukture


```

početak
    izlaz("Ucitajte prvi broj")
    ulaz(a)

    izlaz("Ucitajte drugi broj")
    ulaz(b)

    ako je a >= b onda
        izlaz(a + b)
    inače
        izlaz(a - b)
kraj
    
```

Proširite program

Dodajte treći slučaj $a = b$ i u tom slučaju ispišite 0
 $(a > b \rightarrow \text{ispis zbroja,}$
 $a < b \rightarrow \text{ispis razlike}).$



Primjer 1.3.

Nacrtajte u programu Raptor dijagram toka koji će ispisati na zaslonu računala zbroj prvih 10 prirodnih brojeva. Zapišite algoritam i s pomoću pseudokoda.

ulazni podatci	izlazni podatci
	55

Rješenje

Ovo je primjer algoritma cikličke strukture (slika 1.15.). Na početku su inicijalizirane varijable i i s . Početna vrijednost sume s postavlja se na 0, a prvi prirodni broj i koji se dodaje na sumu s je 1. Potom se vrijednost varijable i u petlji povećava za 1, a vrijednost sume s povećava se za i sve dok i ne postane 11 (moguće je navesti u uvjetu $i > 10$ ili $i \geq 11$).

Stanje varijabli mijenja se kako je prikazano u nastavku, a program Raptor stanje varijabli također ispisuje po koracima u bijelom okviru, u lijevom kutu razvojnog sučelja. Tako se vidi kako je konačno stanje varijable $i = 11$, a $s = 55$.

inicijalizacija		ponavljanje petlje dok vrijednost varijable i ne postane 11									
s	0	1	3	6	10	15	21	28	36	45	55
i	1	2	3	4	5	6	7	8	9	10	11
											izlazak iz petlje

1. Petlja s unaprijed poznatim brojem ponavljanja

```

početak
    s = 0
    za i = 1 do 10 činiti {
        s = s + i
    }
    izlaz(s)
kraj
    
```

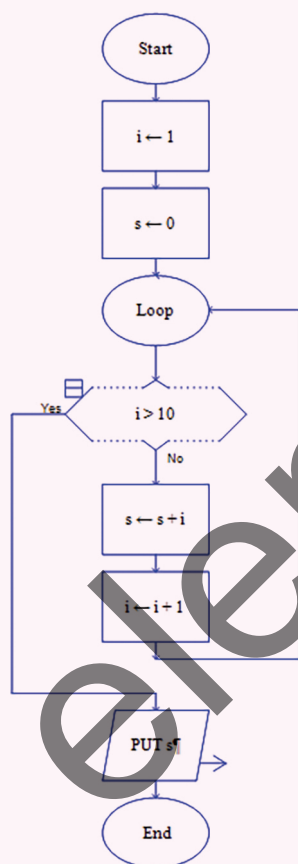
2. Petlja s provjerom uvjeta na početku
(*dok je uvjet činiti*)

```

početak
    s = 0
    i = 1

    dok je i <= 10 činiti {
        s = s + i
        i = i + 1
    }

    izlaz(s)
kraj
    
```



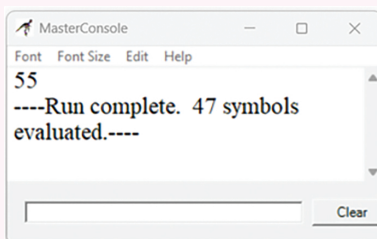
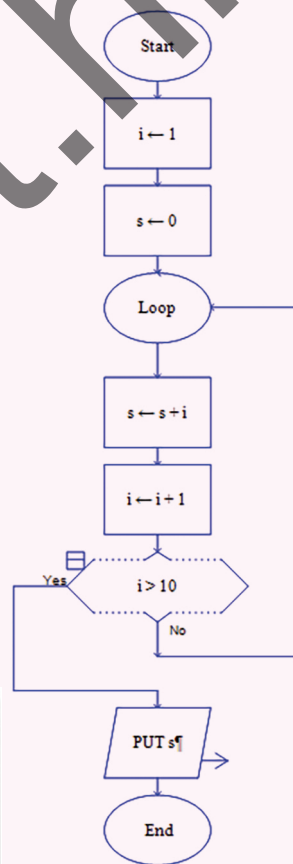
3. Petlja s provjerom uvjeta na kraju
(*ponavljati ... dok je uvjet*)

```

početak
    s = 0
    i = 1

    ponavljati {
        s = s + i
        i = i + 1
    } dok je i <= 10

    izlaz(s)
kraj
    
```



Slika 1.15. Primjer cikličke algoritamske strukture

Proširite program

Omogućite unos broja n i izračunajte zbroj prvih n prirodnih brojeva.



Napomena

U Raptoru se upotrebljava $i > 10$ (uvjet izlaska), a u pseudokodu $i \leq 10$ (uvjet nastavka, kao u programskim jezicima).



VRIJEME ZA IZAZOV

Pametni dnevnik – izračun prosjeka ocjena za N modula

Radna situacija

U školi se planira izrada aplikacije koja učenicima omogućuje lakše praćenje uspjeha po modulima.

Tvoj je zadatak osmisлити algoritam koji omogućuje unos ocjena iz više modula, izračun prosjeka i ispis poruke o uspjehu.

Rješenje zapiši **pseudokodom** i prikaži **di-jagramom toka u programu Raptor**.



Zadatak

1. U program unesi broj modula N.
2. S pomoću petlje unesi ocjene za svaki modul.
3. Izračunaj prosjek ocjena.
4. Ispiši prosjek i odgovarajuću poruku o uspjehu.
5. Omogući korisniku da po želji može ponoviti postupak unosa i izračuna.

Napomena

- Ako je u bilo kojem modulu ocjena nedovoljan (1), tada je ukupni uspjeh također nedovoljan, bez obzira na prosjek.
- Ako korisnik pogrešno unese ocjenu (izvan raspona 1 – 5) ili broj modula (manje od 1), program treba omogućiti ponovni unos ispravne vrijednosti.

Primjer ulaznih i izlaznih podataka

ulazni podatci	izlazni podatci
Broj modula: 3 Ocjene: 5, 5, 4	Prosjek: 4.67 Uspjeh: Odličan
Broj modula: 5 Ocjene: 4, 3, 4, 3, 4	Prosjek: 3.60 Uspjeh: Vrlo dobar
Broj modula: 4 Ocjene: 3, 2, 3, 3	Prosjek: 2.75 Uspjeh: Dobar
Broj modula: 2 Ocjene: 2, 1	Prosjek: 1.5 Uspjeh: Nedovoljan

Elementi vrednovanja – program i pseudokod

elementi provjere	bodovi
unos broja modula (N)	3
provjera ispravnosti unosa broja modula i ocjena	4
unos ocjena s pomoću petlje	4
izračun prosjeka ocjena	5
logička provjera nedovoljne ocjene (ako je barem jedna ocjena = 1, uspjeh = nedovoljan)	4
ispis prosjeka i poruke o uspjehu	4
omogućena ponovna provjera (ponovni unos)	3
struktura pseudokoda jasno slijedi logiku algoritma	1
upotrijebljene pravilne naredbe i izrazi	1
usklađenost pseudokoda s dijagramom toka	1
točnost i čitljivost zapisa	1
ukupno	30

Ocjena

- 27 – 30 = odličan (5)
- 23 – 26 = vrlo dobar (4)
- 19 – 22 = dobar (3)
- 15 – 18 = dovoljan (2)
- 0 – 14 = nedovoljan (1)

Dodatni zadatak za one koji žele znati više

Proširi program tako da omogućuje izračun uspjeha za više učenika zaredom.

Za svakog učenika unosi se broj modula i ocjene, izračunava prosjek i ispisuje opis uspjeha.

Na kraju program treba ispisati **ukupan broj učenika koji su postigli odličan uspjeh**.

Vrednovanje kao učenje



<https://s.element.hr/cd0a>

2.

OSNOVE PROGRAMIRANJA U C-U



DIGITALNI RUDNIK KODOVA

Skeniraj QR kôd ili otvori poveznicu u internetskom pregledniku. Prouči primjere 2.2 i 2.4 te ih analiziraj i proširi programe prateći upute iz tih primjera.

<https://s.element.hr/3891>





Razmisli

U trgovinu je stigla nova računalna oprema. Na skladištu se za svaki proizvod evidentiraju cijena i količina. Kakve brojeve predstavljaju ti podatci? Iz kojih skupova brojeva dolaze?



Nakon ove lekcije moći će



- objasniti osnovnu strukturu programa u jeziku C
- razlikovati vrste pogrešaka u programu
- upotrebljavati komentare u programu
- definirati i upotrebljavati varijable različitih tipova podataka
- pridruživati vrijednosti varijablama
- upotrebljavati osnovne tipove podataka (cjelobrojne, realne i znakovne)
- upotrebljavati konstante u programu
- unositi podatke s pomoću funkcije `scanf`
- ispisivati podatke s pomoću funkcije `printf`
- upotrebljavati osnovne matematičke funkcije
- napisati jednostavan program s linijskom strukturom.



2.1. Uvod u programski jezik C

Programski jezik C programski je jezik **opće namjene**. Primjenjiv je na većini današnjih računala, a sličan je drugim programskim jezicima, stoga predstavlja dobru osnovu programerima početnicima. Autor programskog jezika C je **Dennis MacAlistair Ritchie**, američki računalni znanstvenik.

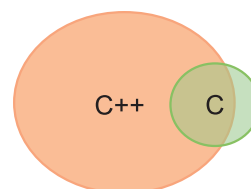
Programeri mogu pisati program kao **niz strojnih instrukcija**, ali je to dosta zahtjevno i sporo pa umjesto toga postoje viši programski jezici kao što su C i C++ u kojima se pišu programi, ali tako napisane programe računala ne mogu izvoditi. Program napisan u višem programskom jeziku treba prevesti u program napisan strojnim instrukcijama te za to postoji program **prevodilac**, odnosno **kompajler**. Program pisan u programskom jeziku C obična je tekstualna datoteka koja najčešće ima nastavak **.c**, a prevedeni program npr. za operacijske sustave Windows može imati nastavak **.exe**.



Slika 2.1. Dennis MacAlistair Ritchie (1941. – 2011.), američki računalni znanstvenik

Programi u programskom jeziku C mogu se pisati u različitim **uređivačima teksta** i zatim **prevoditi** prikladnim **kompajlerom**, ali se često upotrebljavaju **integrirana razvojna okruženja** koja objedinjuju različite programe, uključujući uređivač teksta i program prevodilac te tako olakšavaju razvoj programa.

Programski jezici C i C++ blisko su povezani te se često programi napisani u jeziku C mogu izravno prevoditi kompajlerom za C++ ili ih treba prije prevođenja minimalno izmijeniti (slika 2.2). Programski jezik C++ razvio se iz programskog jezika C. Iako imaju mnogo sličnosti, u nekim se značajkama bitno razlikuju.



Slika 2.2. Programski jezici C i C++ blisko su povezani

Ova su dva programska jezika jako slična po **osnovnoj sintaksi**. Razlikuju se po tome što je programski jezik C organiziran tako da osnovnu cjelinu unutar programa čini **funkcija ili procedura**. Pripada **proceduralnim programskim jezicima**. C++ uz proceduralno programiranje podržava i objektno orijentirano programiranje. Tako je u programskom jeziku C++ moguće, na primjer, kreirati klase i objekte.

Prilikom programiranja potrebno je pridržavati se osnovnih koraka. Prvo se **definira** problem, zatim se određuju **veličine** vezane za zadani problem, a algoritam se **izrađuje**, primjerice, dijagramom toka. Program se potom piše u odabranom **programskom jeziku** i **razvojnom okruženju** prema osmišljenom algoritmu. Računalo, odnosno procesor, izvršava program koji je napisan kao niz strojnih instrukcija. Nakon što je program preveden u jezik razumljiv računalu (**strojni kôd**), program se **testira**, **dokumentira** te se, prema potrebi, **održava** i **ažurira**.

2.1.1. Pisanje prvog programa i pokretanje programa s pomoću razvojnog okruženja Code::Blocks

Za pisanje programa potrebno je odabrati **razvojno okruženje**. U ovom poglavlju prikazuje se kako s pomoću razvojnog okruženja **Code::Blocks** pokrenuti prvi program. Prije početka rada potrebno je preuzeti i instalirati Code::Blocks s pripadajućim kompajlerom. Preporučuje se preuzimanje inačice koja uključuje MinGW kompajler jer omogućuje pisanje, prevođenje i izvršavanje programa u programskom jeziku C. Ovu inačicu možete pronaći na službenoj stranici Code::Blocksa, označenu kao codeblocks-XX.Xmingw-setup.exe. Nakon instalacije razvojno okruženje bit će spremno za upotrebu. U istom razvojnom okruženju pisat ćemo programe i u programskom jeziku C++.

Program je moguće preuzeti s poveznice:

<https://www.codeblocks.org/downloads/>



Primjer 2.1.

Napišite program u programskom jeziku C koji će ispisati na zaslonu računala: „Hello world!”

Rješenje

```
#include <stdio.h>
int main(void) {
    printf("Hello world!\n");
    return 0;
}
```

Objašnjenja pojedinih dijelova koda prvog programa prikazana su tablicom 2.1.

Tablica 2.1. Objašnjenja pojedinih dijelova prvog programa u programskom jeziku C

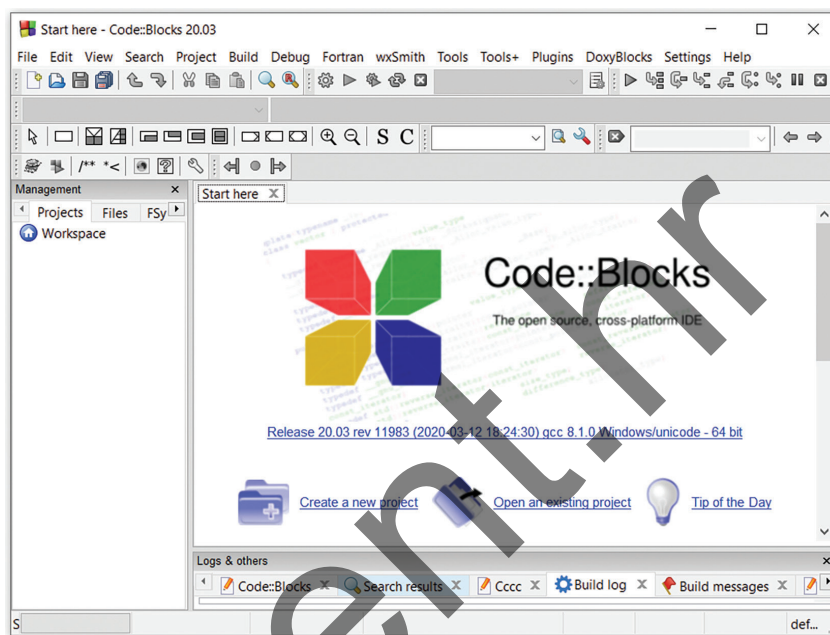
dio programa	objašnjenje
<code>#include <stdio.h></code>	Ova naredba uključuje standardnu biblioteku za ulaz i izlaz. Ona nam omogućuje upotrebu funkcije <code>printf</code> za ispis teksta na zaslon.
<code>int main(void) {</code>	Ovo je glavna funkcija programa. Svaki C program započinje izvođenje upravo u funkciji <code>main</code> . Ključna riječ <code>int</code> označava da funkcija vraća cijeli broj (engl. <i>integer</i>), dok <code>void</code> znači da funkcija ne prima ulazne podatke.
<code>{ i }</code>	Vitičaste zagrade označavaju početak i kraj bloka naredbi koji pripada funkciji. Sve naredbe koje se nalaze unutar njih izvršavaju se kao dio funkcije <code>main</code> .
<code>printf("Hello world!\n");</code>	Funkcija <code>printf</code> služi za ispis teksta na zaslon. Tekst unutar navodnika bit će ispisan, a oznaka <code>\n</code> predstavlja prijelaz u novi red.
<code>return 0;</code>	Ova naredba označava kraj funkcije <code>main</code> i vraća vrijednost 0 operacijskom sustavu, što znači da je program uspješno izvršen.

2. Osnove programiranja u C-u

Nakon uspješne instalacije razvojnog okruženja Code::Blocks u nastavku je opisano pokretanje prvog programa. Koraci su sljedeći:

1. Pokretanje Code::Blocksa

Razvojno okruženje Code::Blocks treba biti otvoreno na računalu (slika 2.3.).



Slika 2.3. Razvojno okruženje Code::Blocks

2. Provjera instalacije kompajlera

Potrebno je provjeriti je li instalirana inačica Code::Blocksa koja uključuje **MinGW kompajler** (obično označena kao **codeblocks-XX.Xmingw-setup.exe**). Ako nije instalirana odgovarajuća inačica, MinGW kompajler mora se preuzeti zasebno (vidi str. 28) i povezati s Code::Blocksom.

3. Povezivanje kompajlera s Code::Blocksom (ako je potrebno)

Ako kompajler nije pronađen, otvara se izbornik **Settings > Compiler...**, gdje se odabire **GNU GCC Compiler**. U kartici **Toolchain executables** provjerava se ispravnost putanje do MinGWa (npr., C:\MinGW\bin). Ako putanja nije ispravno postavljena, unosi se ručno ili se odabire opcija **Auto-detect**. Code::Blocks će automatski provjeriti ispravnost konfiguracije prilikom kompajliranja programa.

4. Kreiranje novog projekta

Iz glavnog izbornika odabire se **File > New > Project**, nakon čega se u prikazanom izborniku odabire opcija **Console Application** i klikne na **Go**.

5. Odabir jezika i postavke projekta

U čarobnjaku koji se otvara odabire se programski jezik C, a zatim se klikne na **Next**. Unosi se naziv projekta i odabire lokacija za spremanje projekta, nakon čega se klikne na **Next**, a potom na **Finish** kako bi se završilo kreiranje projekta.

6. Generiranje predloška projekta

Automatski se generira osnovni predložak za projekt, uključujući datoteke potrebne za pisanje programa.

7. Pisanje kôda

U odjeljku **Management**, pod karticom **Workspace**, pronalazi se naziv projekta. Klikom na naziv projekta proširuje se njegova struktura, a unutar mape **Sources** otvara se datoteka `main.c`. U toj datoteci kôd se može pisati ili prilagoditi.

8. Spremanje programa

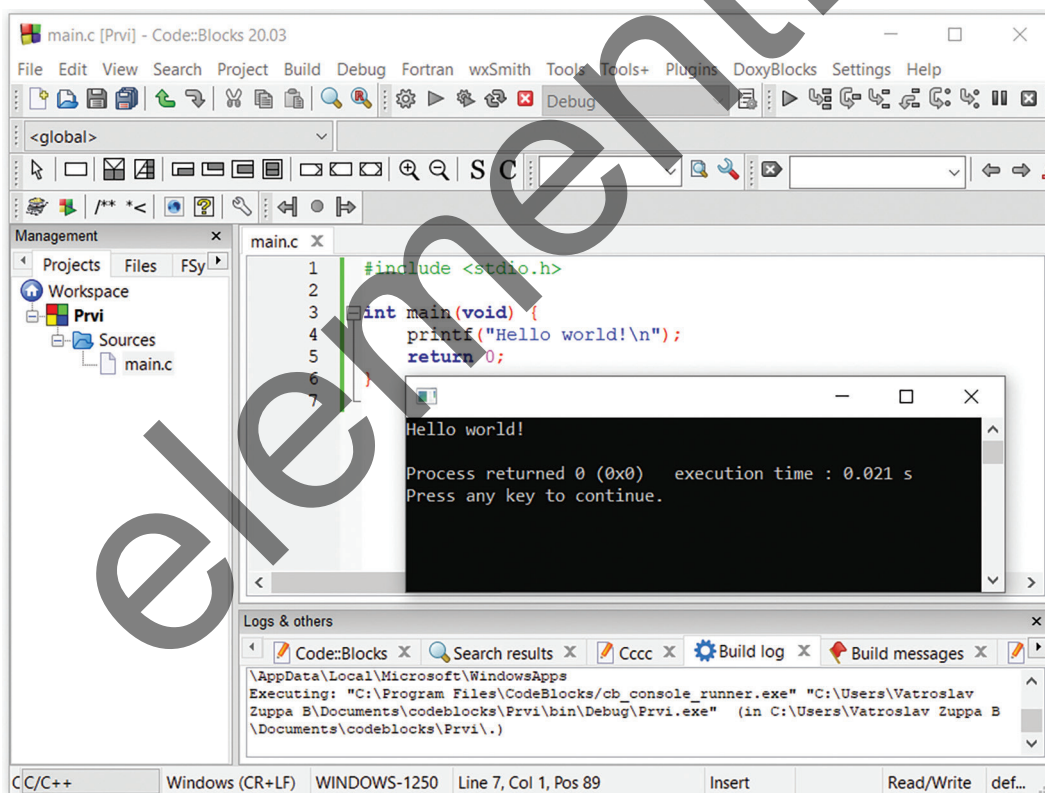
Nakon pisanja programa izmjene se spremaju pritiskom na **Ctrl+S** ili odabirom opcije **File > Save**.

9. Kompajliranje programa

Klikom na ikonu **Build** (zupčanik – ) ili pritiskom na **Ctrl+F9** kôd se prevodi u izvršnu datoteku. Ako kompajler nije ispravno povezan, treba se vratiti na treći korak kako bi se konfiguracija ispravila.

10. Pokretanje programa

Program se pokreće klikom na ikonu **Run** (zeleni strelica – ) ili pritiskom na **Ctrl+F10**, nakon čega se rezultati prikazuju u konzolnom prozoru (slika 2.4.).

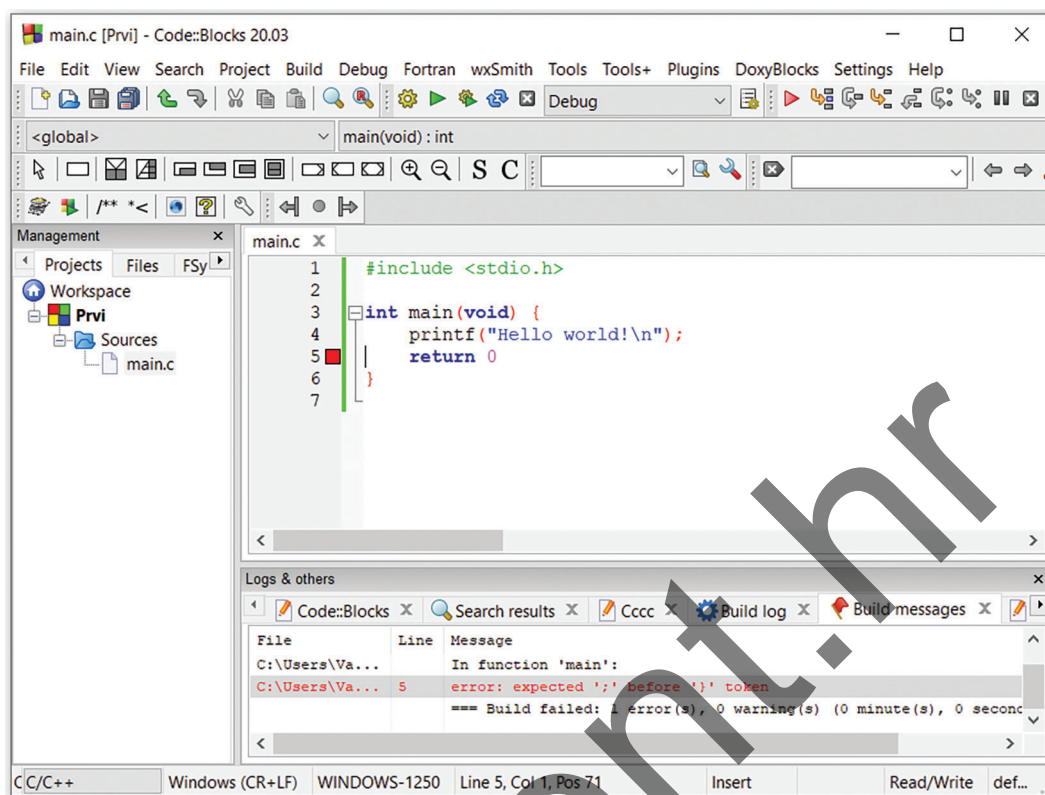


Slika 2.4. Pokretanje programa i ispis na zaslonu računala, u terminalu

11. Ispravljanje pogrešaka

Ako program ne radi kako je predviđeno, poruke o pogreškama pregledavaju se u odjeljku za izlaz (slika 2.5.), a nakon ispravaka postupak kompajliranja i pokretanja se ponavlja. Sintaktička pogreška označena je crvenim kvadratićem uz odgovarajući redak koda, a poruka o pogrešci ispisana je crveno u donjem dijelu razvojnog okruženja.

2. Osnove programiranja u C-u



Slika 2.5. Izvršavanje programa nije bilo uspješno jer u 5. retku koda iza naredbe return 0 nedostaje ;

Tipične vrste pogrešaka u programiranju

Pri pisanju programa često se javljaju pogreške. Razlikujemo tri osnovne vrste pogrešaka:

- a) sintaktičke pogreške (engl. *syntax errors*)
- b) semantičke pogreške
- c) pogreške tijekom izvođenja (engl. *runtime errors*)
- d) logičke pogreške (engl. *logical errors*).

a) Sintaktičke pogreške

Nastaju kada pravila programskog jezika nisu ispoštovana.

Primjer:

```
printf("Hello world!\n")
```

Pogreška: nedostaje točka sa zarezom ;

Rezultat: program se ne može prevesti (kompajlirati), a poruka o pogrešci prikazuje se u donjem dijelu okruženja.

b) Semantičke pogreške

Semantičke pogreške su pogreške nastale na primjer zbog neispravne uporabe tipova podataka, operatora ili formata.

Primjer:

```
int a;  
scanf("%f", &a);
```

Pogreška: upotrebljava se krivi format (%f) za varijablu tipa int

Rezultat: program se može prevesti, ali ponašanje je neispravno (pogrešno učitavanje vrijednosti).



c) Pogreške tijekom izvođenja

Program se uspješno prevede, ali se pogreška javlja tijekom izvođenja.

Primjer:

```
int a = 5;
int b = 0;
printf("%d", a / b);
```

Pogreška: dijeljenje s nulom

Rezultat: program se prekida tijekom izvođenja.

d) Logičke pogreške

Program se izvršava bez prekida, ali daje netočne rezultate.

Primjer:

Odredite zbroj dvaju cijelih brojeva.

```
int a = 5, b = 3;
printf("%d", a - b);
```

Pogreška: ako je zadatak bio zbrajanje, upotrijebljen je pogrešan operator.

Rezultat: program radi, ali rezultat nije ispravan.

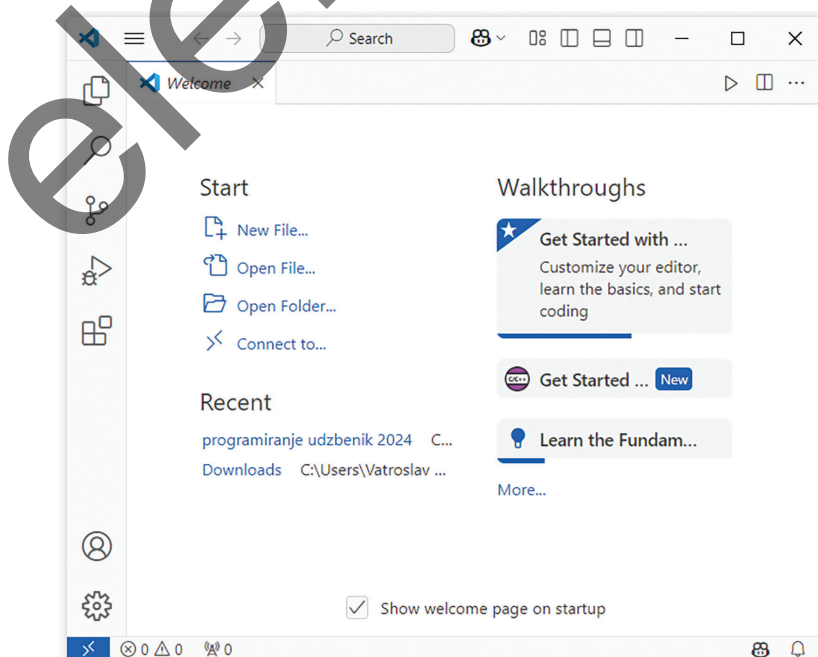
2.1.2. Pokretanje programa s pomoću razvojnog okruženja Visual Studio Code

Kao razvojno okruženje moguće je odabrati i Dev-C++, Visual Studio 2019, Visual Studio Code (VS Code) i dr.

Kako pokrenuti program C s pomoću **Visual Studio Codea** (VS Code), opisano je sljedećim koracima:

1. Instalacija potrebnih alata

Prvi korak u pokretanju programa C s pomoću Visual Studio Codea uključuje instalaciju potrebnih alata. Visual Studio Code (slika 2.6.) preuzima se sa službene stranice i instalira na računalo (<https://code.visualstudio.com/download>).



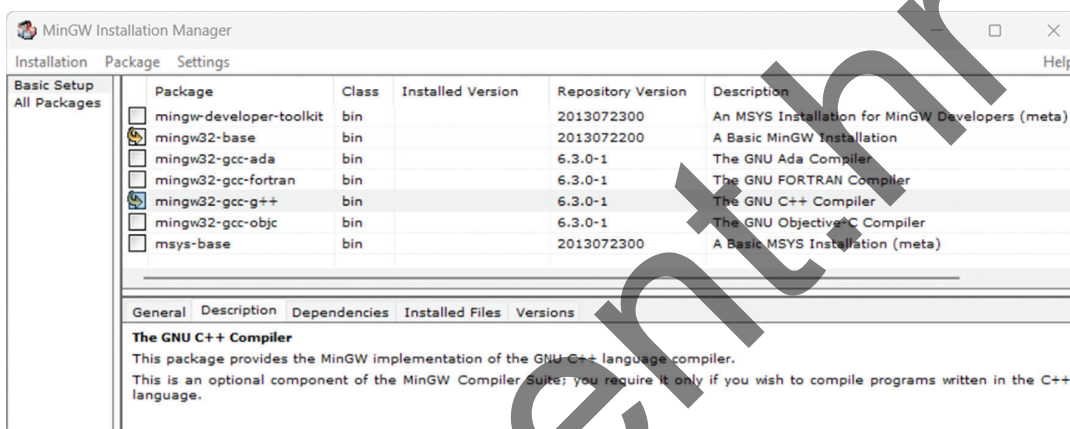
Slika 2.6. Visual Studio Code

2. Osnove programiranja u C-u

Nakon toga potrebno je instalirati i GCC kompajler, koji omogućuje prevođenje C programa. Jedno od najčešće upotrebljavanih rješenja je MinGW, dostupan na stranici SourceForge (<https://sourceforge.net/projects/mingw/>).

Postupak instalacije GCC kompajlera (MinGW):

- Preuzeti instalacijski program (*mingw-get-setup.exe*) s navedene stranice.
- Pokrenuti instalaciju i odabrati opciju **Install**.
- Nakon instalacije otvoriti *MinGW Installation Manager*.
- Označiti pakete (slika 2.7):
 - mingw32-base
 - mingw32-gcc-g++



Slika 2.7. Odabir potrebnih paketa u MinGW Installation Manageru

- U izborniku odabrati **Installation > Apply Changes** i potvrditi instalaciju.
- Nakon završetka instalacije potrebno je dodati putanju do mape C:\MinGW\bin u sistemsku varijablu **PATH** kako bi se naredba gcc mogla upotrijebiti u terminalu.
- U terminalu je potrebno provjeriti instalaciju naredbom: `gcc --version`. Ako se prikaže verzija kompajlera, instalacija je uspješna.

2. Instalacija proširenja u VS Codeu

Nakon instalacije alata unutar Visual Studio Codea otvara se izbornik za proširenja. U njemu se pretražuju i instaliraju dva ključna proširenja: **C/C++ Extension Pack** koji omogućuje podršku za pisanje i kompajliranje programa C, te **Code Runner** koji služi za brzo pokretanje koda izravno unutar VS Codea.

3. Postavljanje radnog prostora

U trećem koraku postavlja se radni prostor unutar Visual Studio Codea. Korisnik odabire novu ili već postojeću mapu na računalu s pomoću opcije **File > Open Folder...** Ova mapa služi kao mjesto za pohranu svih datoteka vezanih za program.

4. Kreiranje nove C datoteke

Unutar postavljenog radnog prostora kreira se nova datoteka s ekstenzijom `.c`. To se postiže desnim klikom na radni prostor i odabirom opcije **New File**, nakon čega se datoteka imenuje, primjerice, **program.c**. U datoteku se unosi željeni kôd.

5. Spremanje datoteke

Kada je kôd napisan, datoteka se sprema s pomoću opcije **File > Save** ili prečacem **Ctrl+S**. Spremanjem se osigurava da sve izmjene budu zabilježene prije kompajliranja.

6. Kompajliranje programa u terminalu

U šestom koraku program se kompajlira i pokreće. Ako se upotrebljava terminal, on se otvara unutar VS Codea s pomoću opcije **View > Terminal**. U terminal se unosi naredba za kompajliranje:

```
gcc program.c -o program.exe
```

Ova naredba prevodi kôd u izvršnu datoteku pod nazivom **program.exe**. Nakon uspješnog kompajliranja program se pokreće unosom naredbe:

```
.\program.exe
```

7. Automatsko pokretanje s pomoću Code Runnera

Kao alternativa terminalu program se može kompajlirati i pokrenuti s pomoću Code Runnera. Klikom na ikonu **Run Code**, koja se nalazi u gornjem desnom kutu prozora, ili pritiskom na prečac **Ctrl+Alt+N**, Code Runner automatski kompajlira i pokreće program, a rezultati se ispisuju unutar terminala.

8. Zadržavanje terminala otvorenim

Ako terminal nestane odmah nakon završetka izvršavanja programa, problem se rješava na dva načina. U postavkama Code Runnera omogućuje se opcija **Run In Terminal**, čime se osigurava prikazivanje rezultata unutar ugrađenog terminala u VS Codeu. Dodatno, u sam kôd može se dodati pauza s pomoću naredbe `getchar()`, čime se terminal zaustavlja i čeka unos tipkovnicom prije zatvaranja.

9. Održavanje programa i ponovno pokretanje

Posljednji korak obuhvaća održavanje i ponovno pokretanje programa. Ako je potrebno unijeti izmjene u kôd, datoteka se ponovno sprema s pomoću **Ctrl+S**. Program se potom kompajlira istom naredbom kao i prije:

```
gcc program.c -o program.exe
```

Nakon kompajliranja izvršna se datoteka pokreće s pomoću naredbe **.\program.exe**. Ako se upotrebljava Code Runner, promjene se jednostavno testiraju ponovnim klikom na ikonu **Run Code** ili prečacem **Ctrl+Alt+N**.

2.1.3. Github Copilot

Visual Studio Code (VS Code) i **GitHub Copilot** zajedno olakšavaju proces programiranja s pomoću umjetne inteligencije. GitHub Copilot je UI alat koji upotrebljava model treniran na velikim količinama koda kako bi u stvarnom vremenu predlagao rješenja, od dovršavanja linija do predlaganja funkcija i čitavih blokova koda. Dok programer piše kôd, Copilot nudi sugestije koje se mogu prihvatiti, odbiti ili prilagoditi.

GitHub Copilot dostupan je kao proširenje za **Visual Studio Code** (VS Code). Da bi se preuzeo i instalirao, potrebno je slijediti sljedeće korake:

Prvo se otvori **Visual Studio Code**. U glavnom izborniku odabire se opcija **Extensions** ili prečac na tipkovnici **Ctrl+Shift+X** kako bi se otvorio preglednik proširenja. U traku za pretraživanje upisuje se **GitHub Copilot**, nakon čega se proširenje, koje nudi GitHub, pronađe na popisu i instalira klikom na gumb **Install**.

Po završetku instalacije potrebno je prijaviti se na svoj **GitHub račun**. Kada se instalacija dovrši, Visual Studio Code prikazat će upute za prijavu. Nakon odabira **Sign In to GitHub** otvara se preglednik gdje je potrebno autorizirati pristup. Na kraju se osigurava da postoji aktivna pretplata za GitHub Copilot jer bez nje proširenje neće funkcionirati.

GitHub Copilot koristi se umjetnom inteligencijom za automatsko generiranje koda na temelju unesenih komentara, početaka funkcija ili konteksta unutar datoteke. **Primjeri primjene prikazani su u Dodatku.**

2.1.4. Standardi programskog jezika C

C je standardiziran kako bi osigurao dosljednost programa na različitim sustavima. Postoji nekoliko standarda programskog jezika C, a neki od njih su:

- **ANSI C (C89 ili C90)** – definira osnovne sintakse i funkcionalnosti koje se i danas upotrebljavaju
- **C99 (ISO/IEC 9899:1999)** – uključuje nove značajke, kao što su deklaracije varijabli bilo gdje u kodu (ne samo na početku bloka), podršku za komentare u stilu C++ (//) i dr.
- **C11 (ISO/IEC 9899:2011)** – donosi nove značajke koje povećavaju sigurnost i paralelizam
- **C17 (ISO/IEC 9899:2018)** – popravljene su određene pogreške iz C11 te je poboljšana konzistentnost jezika
- **C23 (ISO/IEC 9899:2023)** – najnovija inačica standarda iz 2023. godine

Standardi poput C89, C99, C11 i C23 definiraju pravila jezika i standardne funkcije. Iako prevoditelji (kompajleri) poput GCC-a i Clanga obično slijede te standarde, često dodaju svoja proširenja. To može dovesti do toga da kod radi na jednom kompajleru, ali ne i na drugom.

2.2. Osnovna struktura programskog jezika C

Program u jeziku C sastoji se od **deklaracija i definicija varijabli i funkcija te direktna** procesoru, poput `#include <stdio.h>` i `#define PI 3.14159`. Naredbe se pišu u **blokovima**. Razlikuju se **neimenovani** i **imenovani blok**. **Neimenovani blokovi** upotrebljavaju se primjerice za grupiranje naredbi unutar funkcija, petlji i uvjetnih izraza te za kontrolu opsega varijabli. **Imenovani blokovi** upotrebljavaju se u vrlo specifičnim situacijama, na primjer u kombinaciji s naredbom `goto` kada je potreban skok na određeni dio koda. Imenovani blokovi s naredbom `goto` rijetko dolaze u upotrebu i u ovom udžbeniku neće se upotrebljavati naredba `goto`.

Vitičaste zagrade služe za označavanje početka i kraja bloka:

- `{` – početak bloka
- `}` – kraj bloka.

Svaka **naredba završava s ;**, a standard ne propisuje stil pisanja. Standard jezika C jasno razlikuje ključne riječi (primjerice `main`, `return`, `int`, `include...`) te velika i mala slova.

Pregled ključnih riječi nalazi se na sljedećem popisu:

auto	extern	short	while
break	float	signed	_Alignas
case	for	sizeof	_Alignof
char	goto	static	_Atomic
const	if	struct	_Bool
continue	inline	switch	_Complex
default	int	typedef	_Generic
do	long	union	_Imaginary
double	register	unsigned	_Noreturn
else	restrict	void	_Static_assert
enum	return	volatile	_Thread_local

Moguće je navesti dvije vrste komentara u kodu:

- **linijski komentar** (u jednom retku)

```
//komentar
```

- **blokovski komentar** (u više redaka)

```
/*isto  
komentar*/
```

Primjer 2.2.

U prikazanom kodu:

```
#include <stdio.h>
#include <math.h>

int main(void) {
    //deklaracija varijabli
    float a, b, P, d;

    printf("Ucitajte a i b: ");
    scanf("%f %f", &a, &b);

    P = a * b;
    d = sqrt(pow(a, 2) + pow(b, 2));

    printf("P = %.2f, d = %.2f", P, d);

    return 0;
}
```

Proširite program

Namjerno uklonite znak ; iza jedne naredbe i pokušajte kompajlirati program.



- odredite gdje su navedene direktive pretprocesoru
- odredite gdje je naveden naziv glavne funkcije
- odredite gdje počinje imenovani blok
- odredite gdje je oznaka tipa funkcije
- odredite gdje je navedena oznaka kraja jedne naredbe
- odredite gdje su navedene ugrađene matematičke funkcije
- označite tijelo funkcije
- napišite značenje naredbe return 0.

Rješenje

Pregled navedenih ključnih pojmova označen je na prikazanom kodu u nastavku.

```
#include <stdio.h>
#include <math.h>

int main(void) {
    //deklaracija varijabli
    float a, b, P, d;

    printf("Ucitajte a i b: ");
    scanf("%f %f", &a, &b);

    P = a * b;
    d = sqrt(pow(a, 2) + pow(b, 2));

    printf("P = %.2f, d = %.2f", P, d);

    return 0;
}
```

a) direktive pretprocesoru
b) glavna funkcija
c) početak imenovanog bloka
d) tip funkcije
e) oznaka kraja naredbe
f) ugrađene matematičke funkcije
g) tijelo funkcije
h) funkcija vraća 0 (tip int, uspješno izvršavanje)

2.3. Varijable i konstante

2.3.1. Varijable

Varijable tijekom izvođenja programa mogu mijenjati svoje vrijednosti. **Imena varijabli** mogu biti slova engleske abecede, velika i mala slova, brojevi, mogu početi s `_`, ali ne i brojem.

Varijabla predstavlja imenovano mjesto u memoriji računala u koje se pohranjuje neka vrijednost. Tijekom izvođenja programa vrijednost varijable može se mijenjati, dok naziv varijable ostaje isti.

Svaku varijablu potrebno je prethodno **deklarirati**. Deklaracija se radi ovisno o tome koja se vrsta podatka pohranjuje u memoriji računala. Za početno pisanje programa uvode se **jednostavni tipovi podataka** i to:

- **cjelobrojni tipovi podataka** – `char`, `int`, `_Bool` (`bool`)
- **realni tipovi podataka** (brojevi s pomičnim zarezom) – `float`, `double`.

Slika 2.8 prikazuje način na koji se varijabla deklarira u programu. Znači da svaka varijabla ima svoj naziv i pripadajući tip podatka, dok joj se vrijednost može pridružiti tijekom izvođenja programa.

Ime varijable označava mjesto u programu, **tip podatka** određuje kakve se vrijednosti mogu pohranjivati, a **vrijednost** predstavlja trenutačni

ime_varijable	tip_podatka vrijednost
---------------	---------------------------

Slika 2.8. Prikaz varijable u memoriji računala: ime varijable, tip podatka i pridružena vrijednost

sadržaj varijable. U primjerima koji slijede varijable će biti deklarirane unutar glavne funkcije `main`, u kojoj započinje izvođenje programa.

U svim primjerima u 2. poglavlju varijable su deklarirane kao **jednostavni tipovi podataka** poput `int`, `char` ili `float`. Sljedeće poglavlje detaljnije objašnjava ostale cjelobrojne i realne tipove podataka.

Različiti tipovi podataka potrebni su za pohranu različitih vrsta vrijednosti. Tako se tip `int` upotrebljava za cijele brojeve, tip `char` za pohranjivanje znakova, dok se tip `float` upotrebljava za rad s realnim brojevima.

Osnovni tipovi podataka razlikuju se prema vrsti vrijednosti koju pohranjuju. Tablica 2.2 prikazuje najčešće korištene tipove podataka i primjere njihovih vrijednosti.

Tablica 2.2. Pregled osnovnih tipova podataka i primjera vrijednosti

tip podatka	opis	primjer vrijednosti
<code>int</code>	upotrebljava se za cijele brojeve	5, -3, 100
<code>char</code>	upotrebljava se za pohranjivanje jednog znaka	'A', 'b', '3'
<code>float</code>	upotrebljava se za realne brojeve (s decimalnim dijelom)	3.14, -2.5, 0.0

Prilikom **deklaracije varijable** prvo se navodi tip podatka, a zatim njezin naziv:

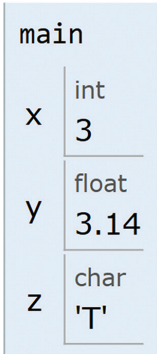
```
tip_podatka ime_varijable;
```

Kao što pokazuje primjer u programu Raptor i ovdje je moguće **deklarirati i inicijalizirati varijablu** naredbom pridruživanja. Opći oblik glasi:

```
tip_podatka ime_varijable = vrijednost;
```

Primjeri deklaracije i inicijalizacije varijabli i prikaz u memoriji računala nalaze se u tablici 2.3.

Tablica 2.3. Deklaracija i inicijalizacija varijabli i vizualizacija u memoriji računala

deklaracija i inicijalizacija varijabli u funkciji <code>main</code>	prikaz u memoriji računala
<pre>int x = 3; float y = 3.14; char z = 'T';</pre>	 <pre>main ├── x: int 3 ├── y: float 3.14 └── z: char 'T'</pre>

Primjer 2.3.

Za sljedeći program prikažite kako se mijenjaju vrijednosti varijabli tijekom izvođenja programa. Promjene vrijednosti prikažite vizualno u memoriji računala za svaki korak izvođenja unutar funkcije main.

```
#include <stdio.h>

int main(void) {

    int a, c = 0;
    float b = 3.14;

    a = 2;
    c = 1;

    return 0;
}
```

Rješenje

U sljedećem primjeru u tablici 2.4. prikazuje se kako će se tijekom izvođenja programa promijeniti vrijednosti varijabli.

Tablica 2.4. Vizualizacija promjene vrijednosti varijabli tijekom izvođenja programa

praćenje promjene vrijednosti varijabli u memoriji tijekom izvođenja programa		
(1)	#include <stdio.h>	
(2)	int main(void) {	
(3)	int a, c = 0;	main a int ? c int 0 b float ?
(4)	float b = 3.14;	main a int ? c int 0 b float 3.14

(5)	<code>a = 2;</code>	main <table> <tr><td>a</td><td>int</td><td>2</td></tr> <tr><td>c</td><td>int</td><td>0</td></tr> <tr><td>b</td><td>float</td><td>3.14</td></tr> </table>	a	int	2	c	int	0	b	float	3.14
a	int	2									
c	int	0									
b	float	3.14									
(6)	<code>c = 1;</code>	main <table> <tr><td>a</td><td>int</td><td>2</td></tr> <tr><td>c</td><td>int</td><td>1</td></tr> <tr><td>b</td><td>float</td><td>3.14</td></tr> </table>	a	int	2	c	int	1	b	float	3.14
a	int	2									
c	int	1									
b	float	3.14									
(7)	<code>return 0;</code>										
(8)	<code>}</code>										

Vrijednost varijable može se tijekom izvođenja programa mijenjati, ali varijabla uvijek predstavlja isto mjesto u koje se ta vrijednost pohranjuje.

Na primjer, varijabla `c` u liniji (3) inicijalno dobiva vrijednost **0**, a zatim se u liniji (6) njezina vrijednost mijenja na **1** naredbom `c = 1;`.

Sadržaj varijable može se postaviti:

- a) učitavanjem podataka s pomoću tipkovnice** (s pomoću funkcije `scanf` koja je opisana u sljedećem poglavlju)
- b) inicijalizacijom** tj. pridruživanjem
- c) višestrukim pridruživanjem.**

Primjer koda gdje se sadržaj varijabli postavlja **učitavanjem podataka** s pomoću tipkovnice:

```
#include <stdio.h>
#include <math.h>

int main(void) {
    //deklaracija varijabli
    float a, b, P, d;

    printf("Ucitajte a i b: ");
    scanf("%f %f", &a, &b);

    P = a * b;
    d = sqrt(pow(a, 2) + pow(b, 2));

    printf("P = %.2f, d = %.2f", P, d);

    return 0;
}
```

float a;
 float b; sadržaj varijabli
 float P; nije poznat
 float d;

postavljanje sadržaja
 varijabli učitavanjem
 podataka s pomoću
 tipkovnice

Primjer koda gdje su sadržaji varijabli a i b postavljeni **inicijalizacijom i višestrukim pridruživanjem**:

```
#include <stdio.h>
#include <math.h>

int main(void) {
    //deklaracija varijabli
    float a = 2.5, b = 2.5, P, d;

    P = a * b;

    d = sqrt(pow(a, 2) + pow(b, 2));

    printf("P = %.2f, d = %.2f", P, d);

    return 0;
}
```

postavljanje sadržaja varijabli inicijalizacijom (pridruživanjem)

a = 2.5;
b = 2.5;

višestruko pridruživanje vrijednosti

a = b = 2.5;

Primjer rješenja bez upotrebe varijabli P i d:

```
#include <stdio.h>
#include <math.h>

int main(void) {
    float a = 2.5, b = 2.5;

    printf("P = %.2f, d = %.2f", a * b, sqrt(pow(a, 2) + pow(b, 2)));

    return 0;
}
```

2.3.2. Konstante

U programskom jeziku C **konstante** su elementi koda s unaprijed zadanom vrijednošću koja se ne može mijenjati tijekom izvršavanja programa. Za razliku od varijabli čija se vrijednost može mijenjati, konstante uvijek zadržavaju istu vrijednost. Primjena konstanti povećava čitljivost i pouzdanost koda jer sprječava nenamjerne promjene vrijednosti.

Podjela konstanti u C-u

1. Cjelobrojne konstante

Cjelobrojne konstante predstavljaju cijele brojeve i mogu biti:

- **dekadske**: brojevi u bazi 10, npr. 100
- **oktalne**: brojevi u bazi 8, s prefiksom 0, npr. 0144
- **heksadekadske**: brojevi u bazi 16, s prefiksom 0x, npr. 0x64.

2. Realne konstante

Realne konstante predstavljaju brojeve s decimalnom točkom, npr. 3.14 ili 2.718. Upotrebljavaju se za definiranje vrijednosti koje zahtijevaju veću preciznost.

3. Znakovne konstante

Znakovne konstante pojedinačni su znakovi koji se definiraju unutar jednostrukih navodnika, npr. 'a', 'Z', '1'.

4. Simboličke konstante

Simboličke konstante definiraju se s pomoću pretprocesorske direktive `#define`. Upotrebljavaju se za dodjeljivanje vrijednosti koje će ostati nepromijenjene u cijelom programu.

Konstante se u jeziku C mogu definirati na dva načina: s pomoću ključne riječi `const` ili s pomoću pretprocesorske direktive `#define`.

Opći oblik deklaracije konstante ključnom riječi `const`

`const tip_podatka naziv_konstante = vrijednost;`

- **`tip_podatka`**: tip podataka konstante (npr. `int`, `float`, `char` itd.)
- **`naziv_konstante`**: ime konstante (poželjno je upotrijebiti velika slova za ime konstante radi boljeg prepoznavanja)
- **`vrijednost`**: vrijednost koja se dodjeljuje konstanti i ne može se mijenjati nakon inicijalizacije.

Primjer:

```
const int MAX_BROJ = 100;
const float PI = 3.14159;
const char ZNAK = 'A';
```

Opći oblik definiranja konstante pretprocesorskom direktivom `#define`

`#define NAZIV_KONSTANTE vrijednost`

- **`NAZIV_KONSTANTE`**: ime simboličke konstante (poželjno je upotrijebiti velika slova za ime)
- **`vrijednost`**: vrijednost simboličke konstante, koja može biti broj, izraz ili tekst.

Primjer:

```
#define MAX_BROJ 100
#define PI 3.14159
#define PORUKA "Dobrodošli u program!"
```

Razlike:

1. **`const`**:
 - konstanta ima definiran tip podataka i slijedi pravila za definiciju varijabli
 - definicija završava točkom sa zarezom ;
2. **`#define`**:
 - konstanta nema tip podataka
 - ne upotrebljava se točka sa zarezom jer je to pretprocesorska direktiva.

Primjer:

```
// const
const int GRANICA = 50; // Konstantna varijabla tipa int

// #define
#define GRANICA 50 // Simbolička konstanta
```

2.4. Izrazi, aritmetički operatori i ugrađene matematičke funkcije

U programskom jeziku C izrazi se upotrebljavaju za izračunavanje vrijednosti ili donošenje odluka, a mogu se podijeliti na različite vrste prema svojoj funkciji:

a) Izrazi pridruživanja – upotrebljavaju operator = za dodjelu vrijednosti varijabli. Na primjer:

```
x = 26 // vrijednost 26 pridružuje se varijabli x
```

b) Aritmetički izrazi – upotrebljavaju operatore poput +, -, *, /, i % za izvođenje matematičkih operacija. Na primjer:

```
x + 26 // zbrajanje vrijednosti varijable x i broja 26
```

c) Relacijski izrazi uspoređuju dvije vrijednosti s pomoću operatora poput <, >, <=, >=, == i !=. Na primjer:

```
x < 26 // provjerava je li x manje od 26
```

d) Logički izrazi upotrebljavaju logičke operatore && (i), || (ili) i ! (ne) za kombiniranje ili negaciju uvjeta. Na primjer:

```
(x < 2) || (y >= 26) // vraća true ako je x manje od 2 ili  
y veće od ili jednako 26
```

Za pisanje navedenih izraza potrebno je poznavati operatore. Neki od operatora koji se upotrebljavaju prilikom pisanja kodova u programskom jeziku C su: operator pridruživanja, aritmetički operatori, relacijski operatori, logički operatori, operatori inkrement i dekrement, operator sizeof, bitovni operatori i dr. Ti operatori opisuju se u sljedećem poglavlju, a za pisanje prvih programa u ovom poglavlju dovoljno je poznavati **aritmetičke operatore**.

Prilikom pisanja naredbi često se puta u samim izrazima upotrebljavaju i **gotove ugrađene funkcije**. Ako se njima želimo koristiti, potrebno je **navesti odgovarajuću biblioteku**. Popis osnovnih matematičkih funkcija kojima ćemo se koristiti u ovom poglavlju prikazan je tablicom 2.5.

Tablica 2.5. Pregled gotovih ugrađenih funkcija iz biblioteka **math.h** i **stdlib.h**

opis	naziv i argumenti funkcije	biblioteka
potenciranje	pow(arg1, arg2)	math.h
drugi korijen	sqrt(arg)	math.h
apsolutna vrijednost cijelog broja	abs(arg)	stdlib.h
apsolutna vrijednost realnog broja	fabs(arg)	math.h
zaokruživanje na najbliži cijeli broj	round(arg)	math.h
odbacivanje decimalnog dijela	trunc(arg)	math.h

trigonometrijske funkcije	<code>sin(arg)</code> <code>cos(arg)</code> <code>tan(arg)</code> <code>asin(arg)</code> <code>acos(arg)</code> <code>atan(arg)</code> *kut je u radijanima	<code>math.h</code>
eksponencijalna i logaritamska funkcija	<code>exp(arg)</code> <code>log(arg)</code> <code>log10(arg)</code>	<code>math.h</code>

2.5. Funkcija za unos podataka

Kao što uvodni primjeri pokazuju, podatke je moguće u program učitati putem tipkovnice. Za učitavanje podataka u kodu je potrebno navesti posebnu funkciju s pomoću koje će to biti moguće. Naziv te funkcije je `scanf`, a nalazi se u biblioteci `stdio.h`. Opći oblik te funkcije je:

```
scanf("format", &var1, &var2, ..., &var_n);
```

pri čemu format predstavlja znakovni niz s informacijama o vrijednostima koje se učitavaju u navedene varijable nakon oznake zareza. Ispred imena varijable navodi se adresni operator `&` (engl. *ampersand*).

Za trenutačne potrebe pisanja programa najčešće se upotrebljavaju **cjelobrojne vrijednosti** (`int`), **realne vrijednosti** (`float`) i **znakovne vrijednosti** (`char`). Za učitavanje cjelobrojne vrijednosti u formatu navodi se specifikator formata `%d`, za realne vrijednosti `%f`, a za znakovne `%c`. Najtipičniji primjeri zapisa funkcije `scanf` nalaze se u nastavku:

- učitavanje jedne cjelobrojne vrijednosti:

```
scanf("%d", &x);
```

- učitavanje jedne realne vrijednosti:

```
scanf("%f", &y);
```

- učitavanje jednog znaka:

```
scanf("%c", &z);
```

- učitavanje jedne cjelobrojne i jedne realne vrijednosti:

```
scanf("%d %f", &x, &y);
```

Pregled preostalih specifikatora formata, koji se navode unutar formata, za druge tipove cjelobrojnih i realnih tipova podataka nalazi se u sljedećem poglavlju.

Pri radu s funkcijom `scanf` često se javljaju određene tipične pogreške koje mogu dovesti do neočekivanog ponašanja programa. U tablici 2.6 prikazani su najčešći problemi, zajedno s primjerima i ispravnim načinima njihove primjene.

Tablica 2.6. Najčešće pogreške i problemi pri upotrebi funkcije `scanf`

pogreška/ problem	primjer (pogrešno)	ispravno rješenje	objašnjenje
izostanak operatora &	<code>scanf("%d", x);</code>	<code>scanf("%d", &x);</code>	bez & funkcija ne zna gdje pohraniti unesenu vrijednost
nepodudaranje tipa i formata	<code>scanf("%f", &x);</code> (za <code>int x</code> ;))	<code>scanf("%d", &x);</code>	format mora odgovarati tipu varijable
unos više vrijednosti bez odgovarajućeg formata	<code>scanf("%d", &x, &y);</code>	<code>scanf("%d %d", &x, &y);</code>	broj formata mora odgovarati broju varijabli
unos znaka nakon broja	<code>scanf("%d", &x);</code> <code>scanf("%c", &c);</code>	<code>scanf("%d", &x);</code> <code>scanf(" %c", &c);</code>	nakon unosa broja znak novog reda (<code>\n</code>) ostaje u ulazu pa ga <code>%c</code> učitava umjesto znaka koji želimo učitati (razmak prije <code>%c</code> preskače razmake, tabulator i novi red)

2.6. Funkcija za ispis podataka

Jedna od prvih funkcija koja se uvodi prilikom pisanja prvog programa bila je upravo funkcija za ispis podataka na standardni izlaz (terminal): `printf`. Nalazi se u biblioteci `stdio.h`. Opći oblik te funkcije glasi:

```
printf("format", izraz / varijabla / konstanta);
```

Za ispis cijelih (dekadskih) vrijednosti koje su definirane kao tip `int` unutar formata navodi se specifikator formata `%d`. Za realne vrijednosti koje su tipa `float` piše se `%f`, a za znakovne koje su tipa `char` piše se `%c`.

Postoje i drugi specifikatori koji se upotrebljavaju za formatirani ispis (njihov zapis ovisi o tipu podatka), a o njima će biti više riječi u sljedećem poglavlju.

Najtipičniji primjeri ispisa nalaze se u nastavku:

- ispis cjelobrojne vrijednosti koja je pridružena varijabli `x`:

```
printf("%d", x);
```

- ispis realne vrijednosti (ispis na 6 decimalnih mjesta) koja je pridružena varijabli `y`:

```
printf("%f", y);
```

- ispis znaka koji je pridružen varijabli `z`:

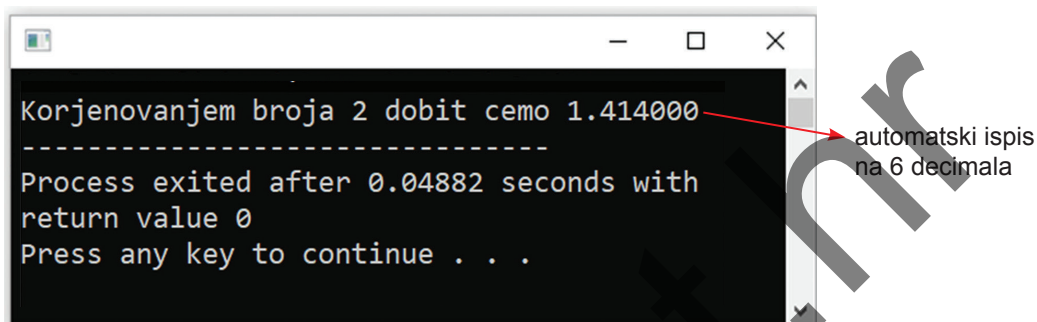
```
printf("%c", z);
```

- ispis cjelobrojne i realne vrijednosti koje su pridružene varijablama *x* i *y* s jednim mjestom razmaka između:

```
printf("%d %f", x, y);
```

- ispis rezultata izvršavanja nekog programa pri čemu je varijabli *x* pridružena vrijednost 2, a *y* vrijednost 1.414 (slika 2.9.)

```
printf("Korjenovanjem broja %d dobit cemo %f", x, y);
```



Slika 2.9. Primjer ispisa na zaslonu računala

- za ispis se rezerviraju 3 mjesta (jedan kvadratić predstavlja jedno mjesto na zaslonu računala, a ako bi umjesto broja 12 ispisivao broj 1234, format bi se zanemario, a ispis bi bio s 4 mjesta):

```
printf("%3d", 12);
```

	1	2
--	---	---

- za ispis se rezerviraju 3 decimalna mjesta:

```
printf("%.3f", 4.56);
```

4	.	5	6	0
---	---	---	---	---

- za ispis se rezervira 5 mjesta, od čega su 2 decimalna mjesta (i decimalna točka zauzima jedno mjesto), a ispis se radi prema pravilu zaokruživanja decimalnih brojeva:

```
printf("%.5.2f", 4.698);
```

	4	.	7	0
--	---	---	---	---

- primjer ispisa s upotrebom specijalnog znaka za prelazak u novi red

```
printf("%2c%3d%\n4.2f", 'z', 5, 1.9);
```

Prilikom ispisa često se upotrebljavaju i specijalni znakovi kao što su:

\n – prelazak u novi red

\t – horizontalni tabulator

	z			5
1	.	9	0	

- kôd treba pisati strukturirano – podijeliti ga na manje dijelove i funkcije koje imaju jasno definiran cilj.

U zadatcima će se uz opis teksta zadatka nalaziti i ulazno/izlazni podatci. Sljedeći zadatak prikazuje jedan takav primjer.

Primjer 2.5.

U nastavku se nalaze dvije inačice programa s pomoću kojih je moguće izračunati površinu kruga. U programe treba učitati vrijednost polumjera u centimetrima, a ispisati kolika je površina kruga u cm^2 (na 3 decimale) i to točno onako kako je prikazano pod izlaznim podatcima.

Broj π (3.14159) treba definirati kao simboličku konstantu. Potrebno je objasniti zašto druga inačica programa nije ispravno napisana.

ulazni podatci	izlazni podatci
2.5	Povrsina kruga polumjera 2.500 cm je 19.635 cm ² .

Preporučeno rješenje (prva inačica programa):

```
#include <stdio.h>
#define PI 3.14159

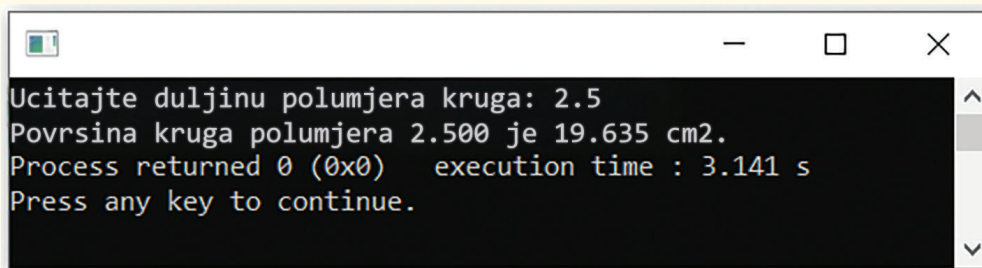
int main(void) {
    float r, P;
    //unos podataka
    printf("Ucitajte duljinu polumjera kruga: ");
    scanf("%f", &r);

    //izračun površine
    P = r * r * PI;

    //ispis rezultata
    printf("Povrsina kruga polumjera %.3f cm je %.3f cm2.", r, P);

    return 0;
}
```

Ispis na zaslonu računala u terminalu (slika 2.11):



Slika 2.11. Ispravan ispis na zaslonu računala

Prednosti preporučenog rješenja

Funkcionalna ispravnost:

- program točno izračunava površinu kruga uzimajući precizniju vrijednost konstante π
- rezultat je prikazan u zadanom obliku i s traženom preciznošću.

Kvaliteta koda:

- kôd je jasno strukturiran i pregledan
- kako je zadano u zadatku, upotrijebljena je simbolička konstanta π , čime se povećava točnost i čitljivost programa
- prisutna je komunikacija s korisnikom (jasna poruka za unos podataka)
- izlaz je oblikovan prema zadanim zahtjevima
- program sadrži komentare koji objašnjavaju pojedine dijelove koda.

Dobro napisan program uz to što mora davati točan rezultat, mora biti i razumljiv, pregledan i prilagođen programeru.

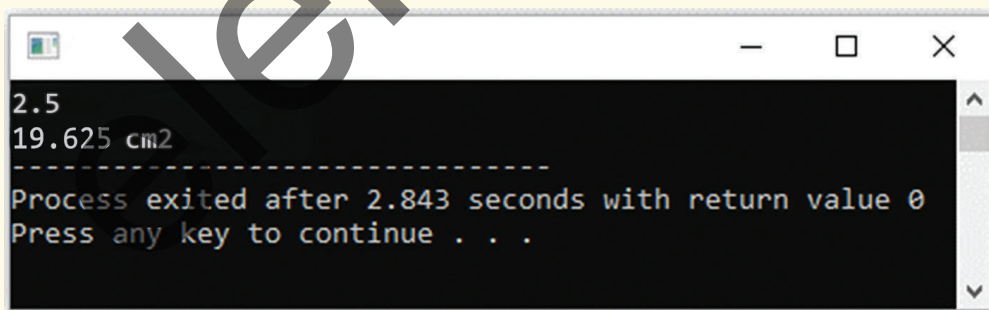
Alternativno rješenje, bez preporučenih praksi (druga inačica programa):

```
#include <stdio.h>

int main(void) {
    float r, P;

    scanf("%f", &r);    P = r * r * 3.14;
    printf("%f cm2", P);
    return 0;
}
```

Ispis na zaslonu računala u terminalu (slika 2.12):



Slika 2.12. Neispravno formatiran ispis na zaslonu računala

Funkcionalna ispravnost druge inačice programa:

- program izračunava površinu kruga
- rezultat je približan zbog uzimanja vrijednosti 3.14 umjesto preciznije konstante π .

Kvaliteta koda i nedostaci druge inačice programa:

- kôd nije jasno strukturiran
- nedostaje komunikacija s korisnikom (nema poruke za unos)

- nije upotrijebljena simbolička konstanta π
- izlaz nije u potpunosti u zadanom formatu.

U ovom rješenju nedostaju i komentari koji objašnjavaju svrhu pojedinih dijelova koda. Komentari su važni jer omogućuju lakše razumijevanje programa, posebice pri učenju i kasnijem održavanju koda. Dobro komentiran program jasnije prikazuje logiku rješenja i olakšava njegovu primjenu drugim programerima.

ispravan prikaz rješenja algoritma	nepotpun prikaz rješenja algoritma
Površina kruga polumjera 2.500 cm je 19.635 cm ² .	19.625 cm ²

2.8. Linijaska struktura programa

Programski jezik C upotrebljava **tri osnovne algoritamske strukture**:

- linijsku** algoritamsku strukturu (slijed)
- razgranatu** algoritamsku strukturu (odluka, grananje)
- cikličku** algoritamsku strukturu (ponavljanje, petlja).

Linijaska struktura programa odnosi se na organizaciju programskog koda gdje se naredbe izvršavaju redoslijedom kojim su zapisane, od vrha prema dnu. Ova struktura ne uključuje grananje ili ponavljanje, već program slijedi jednostavan, sekvencijalan tijek izvršavanja. Linijaska struktura najosnovniji je oblik organizacije programa i primjenjuje se za jednostavne zadatke koji ne zahtijevaju odluke ili petlje. U sljedećim poglavljima bit će više riječi o razgranatoj i cikličkoj algoritamskoj strukturi.

Primjer 2.6.

Za sljedeći program odredite konačne vrijednosti varijabli *a* i *b* te napišite kako će izgledati ispis u terminalu računala.

```
#include <stdio.h>

int main(void) {
    int a = 10, b = 3;

    a = a + b;
    b = b + 5;
    a = a - 2;
    b = b - a;

    printf("a = %d, b = %d\n", a, b);

    return 0;
}
```

Rješenje

Praćenjem izvođenja programa dobivamo sljedeće promjene vrijednosti varijabli (tablica 2.7):

Tablica 2.7. Praćenje promjene vrijednosti varijabli tijekom izvođenja programa

korak	a	b
početno stanje	10	3
a = a + b;	13	
b = b + 5;		8
a = a - 2;	11	
b = b - a;		-3

Ispis u terminalu

Nakon izvršavanja svih naredbi vrijednosti varijabli su: a = 11 i b = -3.

Vrijednosti se u funkciji `printf` umeću na mjesta označena s `%d`, dok se ostali znakovi ispisuju točno kako su zapisani.

Tablica prikazuje kako bi trebao izgledati ispis u terminalu, pri čemu svaki kvadratić predstavlja jedno mjesto za ispis. Jedno mjesto u tablici odgovara jednom mjestu u terminalu, odnosno jednom znaku (slovo, broj, razmak ili interpunkcijski znak).

a	=	1	1	,		b	=	-	3
---	---	---	---	---	--	---	---	---	---

Stanje varijabli moguće je odrediti i s pomoću *online* alata dostupnog na poveznici:

<http://pythontutor.com/c.html#mode=edit>

Prateći korak po korak, vidi se kako se mijenja stanje varijabli. Klikne se na **<Prev** za prethodni korak ili **Next>** za sljedeći korak, a moguće je odmah skočiti i na zadnji korak klikom na **Last>>** ili se vratiti na prvi klikom na **<< First**.

Promijeni početne vrijednosti varijabli i odredi što će se u tom slučaju ispisati.



C (C17 + GNU extensions)
[Known limitations](#)

```

1 #include <stdio.h>
2
3 int main(void) {
4
5     int a = 10, b = 3;
6
7     a = a + b;
8     b = b + 5;
9     a = a - 2;
10    b = b - a;
11
12    printf("a = %d, b = %d\n", a, b);
13
14    return 0;
15 }

```

[Edit this code](#)

→ line that just executed
→ next line to execute

<< First < Prev Next > Last >>

Done running (8 steps)

Print output (drag lower right corner to resize)

a = 11, b = -3

Stack Heap

main

a int 11

b int -3

C/C++ details: none [default view]

Slika 2.13. Online kompajler

Primjer 2.7.

Napišite program u koji će korisnik učitati duljine stranica pravokutnika, a program će mu ispisati na zaslonu računala njegovu površinu i duljinu dijagonale.

ulazni podatci	izlazni podatci
4 3	P = 12.00 d = 5.00

Rješenje

Učitavanje vrijednosti u varijable putem tipkovnice:

```
#include <stdio.h>
#include <math.h>

int main(void) {

    float a, b, P, d;        //stranice pravokutnika, površina i dijagonala

    //unos podataka
    printf("Učitajte a i b: ");
    scanf("%f %f", &a, &b);

    //izračun površine i dijagonale
    P = a * b;
    d = sqrt(pow(a, 2) + pow(b, 2));

    //ispis rezultata
    printf("P = %.2f\nd = %.2f", P, d);

    return 0;
}
```

Deklaracija i inicijalizacija varijabli u programu (1. način):

```
#include <stdio.h>
#include <math.h>

int main(void) {

    float a = 4, b = 3, P, d;    //inicijalizacija varijabli a i b

    // izračun površine i dijagonale
    P = a * b;
    d = sqrt(pow(a, 2) + pow(b, 2));

    //ispis rezultata
    printf("P = %.2f\nd = %.2f", P, d);

    return 0;
}
```

Deklaracija i inicijalizacija varijabli u programu (2. način, bez upotrebe varijabli p i d):

```
#include <stdio.h>
#include <math.h>

int main(void) {
    float a = 4, b = 3;    //inicijalizacija varijabli

    //izračun i ispis u jednoj naredbi
    printf("P = %.2f\\nd = %.2f", a * b, sqrt(pow(a, 2) + pow(b, 2)));

    return 0;
}
```

Izmijenite program

Ispišite rezultate na tri decimalna mjesta.



Primjer 2.8.

Napišite program koji učitava početnu cijenu proizvoda (u eurima) i postotak povećanja. Program treba izračunati novu cijenu nakon povećanja i ispisati je na dvije decimale.

ulazni podatci	izlazni podatci
99.99 15	Nova cijena: 114.99

Rješenje

1. način:

```
#include <stdio.h>

int main(void) {
    float cijena, nova_cijena; // početna i nova cijena
    int p;                    // postotak povećanja

    // unos podataka
    printf("Ucitajte pocetnu cijenu: ");
    scanf("%f", &cijena);

    printf("Ucitajte postotak povecanja: ");
    scanf("%d", &p);

    // izračun nove cijene
    nova_cijena = cijena + cijena * p / 100;

    // ispis rezultata
    printf("Nova cijena: %.2f\\n", nova_cijena);

    return 0;
}
```

2. način:

```
#include <stdio.h>

int main(void) {

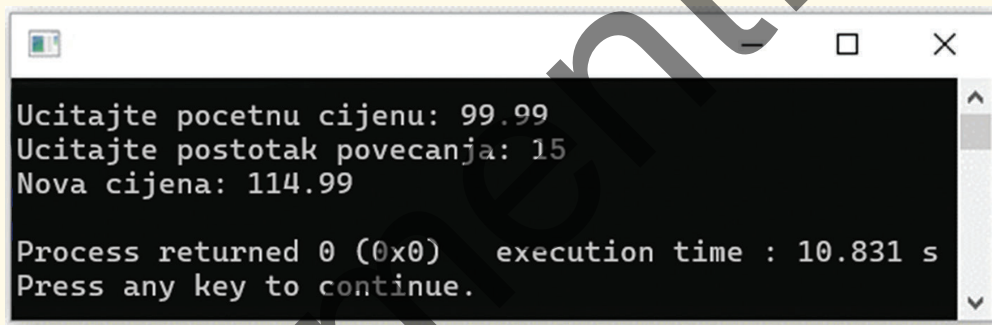
    float cijena; // početna cijena
    int p;        // postotak povećanja

    // unos podataka
    printf("Ucitajte cijenu i postotak: ");
    scanf("%f %d", &cijena, &p);

    // izračun i ispis u jednoj naredbi
    printf("Nova cijena: %.2f\n", cijena + cijena * p / 100);

    return 0;
}
```

Prikaz ispisa na zaslonu računala nalazi se na slici 2.14.



Slika 2.14. Prikaz rješenja na zaslonu računala

Izmijenite program

Prilagodite program tako da umjesto povećanja, izračuna cijenu proizvoda nakon sniženja za zadani postotak.





VRIJEME ZA IZAZOV

Moj prvi kalkulator

Radna situacija

Na satu programiranja učiš izrađivati svoje prve programe u jeziku **C**.

Zadatak ti je osmisлити jednostavan **kalkulator** koji omogućuje unos dvaju brojeva, izračun osnovnih matematičkih operacija i ispis rezultata na **zaslonu računala**.



Zadatak

1. Napiši program koji omogućuje **unos dvaju realnih brojeva**.
2. Program treba izračunati i ispisati:
 - **zbroj**
 - **razliku**
 - **umnožak**
 - **vrijednost izraza** $\frac{\sqrt{(x-y)^4 + 5}}{2xy}$, pri čemu je x prvi učitani broj, a y drugi učitani broj.
3. Rezultate ispiši pregledno na **zaslonu računala**.
4. Dodaj komentare u program kako bi objasnio/la svrhu važnijih naredbi.
5. Napiši i **pseudokod** koji opisuje rješenje. **Nije potrebno crtati dijagram toka.**

Napomena:

- Pretpostavlja se da će korisnik **ispravno unijeti podatke** te da brojevi x i y sigurno neće biti 0.

Primjer ulaznih i izlaznih podataka

ulazni podatci	izlazni podatci
2 3	Zbroj: 5.000 Razlika: -1.000 Umnožak: 6.000 Izraz: 0.204
4 2.5	Zbroj: 6.500 Razlika: 1.500 Umnožak: 10.000 Izraz: 0.159

Elementi vrednovanja – program i pseudokod

elementi provjere	bodovi
unos dvaju realnih brojeva	4
izračun zbroja, razlike i umnoška	6
ispravan zapis i izračun matematičkog izraza	5
jasno strukturiran i pregledan ispis rezultata	5
dodani komentari u programu koji objašnjavaju kôd	4
struktura pseudokoda jasno slijedi logiku algoritma	3
usklađenost pseudokoda s programom u jeziku C	2
točnost i čitljivost zapisa	1
ukupno	30

Ocjena

- 27 – 30 = odličan (5)
- 23 – 26 = vrlo dobar (4)
- 19 – 22 = dobar (3)
- 15 – 18 = dovoljan (2)
- 0 – 14 = nedovoljan (1)

Dodatni zadatak za one koji žele znati više

Proširi program tako da dodatno računa:

- izračun **prosjeka** učitanih brojeva
- koliki je **zbroj kvadrata** učitanih brojeva
- kolika je **razlika kvadrata** učitanih brojeva
- koliki je **omjer prvog i drugog** učitanih broja.

Rezultate prikaži zaokružene na tri decimale.

Vrednovanje kao učenje



<https://s.element.hr/ad80>

3.

TIPOVI PODATAKA I OPERATORI



?

U prošlom smo poglavlju računali s brojevima. Kako računalo zna što je broj, a što tekst?

To ovisi o tipu podatka. U C-u je potrebno odrediti s kakvom vrstom podataka program radi - cijelim brojevima, realnim brojevima, znakovima ili na primjer s tekстом.

Znači, ako pogrešno definiramo tip podataka, možemo dobiti pogrešne rezultate?

int float

Upravo tako. Operatorima određujemo što će program učiniti s podacima.

Točno! I kad ih kombiniraš kako treba, tvoj program radi točno ono što želiš.

Okej, sad mi je jasnije! Znači, tipovi podataka određuju kojom se vrstom informacija program koristi, a operatori način njihove obrade?



DIGITALNI RUDNIK KODOVA

Skeniraj QR kôd ili otvori poveznicu u internetskom pregledniku. Prouči primjere 3.23, 3.25, 3.28 i 3.30 te ih analiziraj i proširi programe prateći upute iz tih primjera.

<https://s.element.hr/0b85>





Razmisli

U memoriji računala različite vrste podataka zauzimaju različitu količinu prostora. Procijeni koliko približno memorije zauzimaju sljedeći podatci:

- znak A
- emoji
- cijeli broj 1000
- realni broj 3.14159
- kratka poruka: "Bok!"
- vlastita slika u mobitelu (engl. *selfie*)
- pjesma s Eurosonga u trajanju od 3 minute
- video s nogometne utakmice u trajanju od 1 minute



Nakon ove lekcije moći će



- objasniti kako se podatci zapisuju u računalu
- razlikovati dekadski, binarni i heksadekadski brojevni sustav
- objasniti zapis cijelih i realnih brojeva u računalu
- objasniti način zapisa znakova s pomoću ASCII koda
- razlikovati osnovne tipove podataka u jeziku C
- odrediti zauzeće tipova podataka u memoriji
- upotrebljavati i razlikovati osnovne tipove operatora
- pratiti izvođenje izraza uz poštovanje prioriteta operatora
- razlikovati implicitnu i eksplicitnu konverziju tipova podataka
- primijeniti konverziju tipova podataka u programu.



3.1. Prikaz podataka u računalu

Svi podatci koje računalo obrađuje i pohranjuje zapisuju se u memoriji računala. Za razliku od čovjeka, koji se u svakodnevnom životu koristi dekadskim brojevnim sustavom, računalo radi s elektroničkim sklopovima koji mogu prepoznati samo dva osnovna stanja. Zbog toga se podatci u računalu zapisuju s pomoću znamenaka **0** i **1**. Takav zapis naziva se **binarni zapis**.

Kako bismo lakše razumjeli na koji se način podatci prikazuju i pohranjuju u računalu, potrebno je upoznati osnovne brojevne sustave kojima se koristimo u programiranju i računalnoj tehnici.

3.1.1. Brojevni sustavi

Brojevni sustav određuje kojim se znamenkama zapisuju brojevi i kakvu vrijednost znamenka ima ovisno o svojem položaju u zapisu broja.

U svakodnevnom životu upotrebljavamo **dekadski brojevni sustav**. On ima bazu 10 i koristi se znamenkama od 0 do 9.

Računala upotrebljavaju **binarni brojevni sustav**. On ima bazu 2 i koristi se samo dvjema znamenkama: 0 i 1.

U računalstvu se često upotrebljava i **heksadekadski brojevni sustav**, koji ima bazu 16. On se koristi znamenkama 0 – 9 i slovima A, B, C, D, E i F.

Budući da dekadski sustav ima samo 10 znamenaka (0 – 9), za zapis vrijednosti od 10 do 15 uvode se slova: A = 10, B = 11, C = 12, D = 13, E = 14, F = 15.

Heksadekadski zapis može služiti kao kraći i pregledniji prikaz binarnih podataka. Budući da se svi podatci u memoriji računala pohranjuju u binarnom obliku, rad s dugim nizovima znamenaka 0 i 1 može biti nepregledan. Heksadekadski zapis zato omogućuje jednostavniji i pregledniji prikaz sadržaja memorije.

Osnovni brojevni sustavi razlikuju se prema bazi i skupu znamenaka kojima se koriste. U tablici 3.1 nalazi se pregled najčešće upotrebljavanih brojevnih sustava.

Tablica 3.1. Pregled dekadskog, binarnog i heksadekadskog brojevnog sustava

brojevni sustav	baza	znamenke	primjena
dekadski	10	0 – 9	svakodnevni zapis brojeva
binarni	2	0, 1	zapis podataka u računalu
heksadekadski	16	0 – 9, A, B, C, D, E, F	kraći zapis binarnih podataka

Napomena

Oznaka baze piše se uz broj u zagradi kako bi se razlikovalo u kojem je brojevnom sustavu zapisan broj ($23_{(10)}$, $1011_{(2)}$, $F3_{(16)}$). Ako uz broj nije navedena baza, pretpostavlja se da se radi o bazi 10.

3.1.2. Rastavljanje brojeva po težinama

U svim brojevnim sustavima vrijednost broja određena je znamenkama i njihovim položajem u zapisu. Takvi sustavi nazivaju se **pozicijski brojevni sustavi**.

Svaka znamenka u broju ima svoju **težinu** koja ovisi o poziciji i bazi brojevnog sustava. Težine se zapisuju kao potencije baze sustava.

Općenito, broj se može zapisati tako da se svaka znamenka pomnoži sa svojom težinom, a zatim se svi ti rezultati zbroje. Težine su potencije baze brojevnog sustava.

$$\text{znamenka} \cdot \text{baza}^{\text{pozicija}} = \text{težina}$$

U **dekadskom brojevnom sustavu** (baza 10) težine su potencije broja 10:

$$\dots 10^3, 10^2, 10^1, 10^0, 10^{-1} \dots$$

U **binarnom brojevnom sustavu** (baza 2) težine su potencije broja 2:

$$\dots 2^3, 2^2, 2^1, 2^0, 2^{-1} \dots$$

U **heksadekadskom brojevnom sustavu** (baza 16) težine su potencije broja 16:

$$\dots 16^3, 16^2, 16^1, 16^0, 16^{-1} \dots$$

Na taj način svaki broj, bez obzira na brojevni sustav, može se rastaviti na zbroj vrijednosti svojih znamenaka.

U sljedećim primjerima prikazano je kako se brojevi u različitim brojevnim sustavima rastavljaju po težinama.

2617	=	$2 \cdot 10^3$	+	$6 \cdot 10^2$	+	$1 \cdot 10^1$	+	$7 \cdot 10^0$
	=	2000	+	600	+	10	+	7

3 2 1 0 znamenka baza 10 težina 10^0

1011 ₍₂₎	=	$1 \cdot 2^3$	+	$0 \cdot 2^2$	+	$1 \cdot 2^1$	+	$1 \cdot 2^0$
---------------------	---	---------------	---	---------------	---	---------------	---	---------------

3 2 1 0 znamenka baza 2 težina 2^0

10.11 ₍₂₎	=	$1 \cdot 2^1$	+	$0 \cdot 2^0$	+	$1 \cdot 2^{-1}$	+	$1 \cdot 2^{-2}$
----------------------	---	---------------	---	---------------	---	------------------	---	------------------

1 0 -1 -2

A3.4 ₍₁₆₎	=	$10 \cdot 16^1$	+	$3 \cdot 16^0$	+	$4 \cdot 16^{-1}$
----------------------	---	-----------------	---	----------------	---	-------------------

1 0 -1 znamenka baza 16 težina 16^{-1}

3.1.3. Pretvorbe brojevni sustava

3.1.3.1. Pretvorba u dekadski brojevni sustav

Broj zapisan u bilo kojem brojevnom sustavu može se pretvoriti u dekadski tako da se rastavi po težinama i zbroje dobivene vrijednosti.

Primjer 3.1.

Pretvori sljedeće brojeve u dekadski brojevni sustav:

a) $10.11_{(2)}$

b) $A3.4_{(16)}$

Rješenje

$$\begin{aligned} \text{a) } 10.11_{(2)} &= 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} \\ &= 2 + 0 + 0.5 + 0.25 \\ &= 2.75_{(10)} \end{aligned}$$

$$\begin{aligned} \text{b) } A3.4_{(16)} &= 10 \cdot 16^1 + 3 \cdot 16^0 + 4 \cdot 16^{-1} \\ &= 160 + 3 + 0.25 \\ &= 163.25_{(10)} \end{aligned}$$

3.1.3.2. Pretvorba iz dekadskog brojevnog sustava

Dekadski broj pretvara se u druge brojevne sustave u dva koraka, ovisno o tome radi li se o cjelobrojnom ili decimalnom dijelu broja.

Postupak pretvorbe:

- **Cjelobrojni dio** pretvara se **uzastopnim dijeljenjem s bazom** (2 za binarni, 16 za heksadekadski sustav).
Bilježe se ostaci pri dijeljenju, a konačni zapis dobiva se čitanjem ostataka **odozdo prema gore**.
- **Decimalni dio** pretvara se uzastopnim množenjem s bazom.
U svakom koraku uzima se cjelobrojni dio rezultata, koji čini znamenke zapisa, a konačni zapis dobiva se čitanjem tih znamenaka **redoslijedom kojim su dobivene (odozgo prema dole)**.

U prethodnim primjerima dobiveni su dekadski brojevi: $2.75_{(10)}$ i $163.25_{(10)}$.

Sada ih pretvaramo natrag u izvorne brojevne sustave.

Primjer 3.2.

Pretvori **a)** $66_{(10)}$ i **b)** $2.75_{(10)}$ u binarni sustav.

Rješenje

- a)** Broj $66_{(10)}$ ima samo cjelobrojni dio. Postupak radimo kako je prikazano tablicom 3.2.

Tablica 3.2. Pretvorba iz dekadskog u binarni brojevni sustav

dijeljenje	količnik	ostatak	kraći zapis
$66 : 2$	33	0	$66 : 2 = 33$, ostatak 0
$33 : 2$	16	1	$33 : 2 = 16$, ostatak 1
$16 : 2$	8	0	$16 : 2 = 8$, ostatak 0
$8 : 2$	4	0	$8 : 2 = 4$, ostatak 0
$4 : 2$	2	0	$4 : 2 = 2$, ostatak 0
$2 : 2$	1	0	$2 : 2 = 1$, ostatak 0
$1 : 2$	0	1	$1 : 2 = 0$, ostatak 1

Konačni rezultat dobiva se čitanjem ostataka odozdo prema gore.

$$66_{(10)} = 1000010_{(2)}$$

- b)** Radimo postupak posebno za cjelobrojni, posebno za decimalni dio broja $2.75_{(10)}$.

Cjelobrojni dio (2):

$$2 : 2 = 1, \text{ ostatak } 0$$

$$1 : 2 = 0, \text{ ostatak } 1$$

$$\rightarrow 10_{(2)}$$

$$2.75_{(10)} = 10.11_{(2)}$$

Decimalni dio (0,75):

$$0.75 \cdot 2 = 1.5 \rightarrow 1$$

$$0.5 \cdot 2 = 1.0 \rightarrow 1$$

$$\rightarrow .11_{(2)}$$

Primjer 3.3.

Pretvori $163.25_{(10)}$ u heksadekadski sustav.

Rješenje**Cjelobrojni dio:**

$$163 : 16 = 10, \text{ ostatak } 3$$

$$10 : 16 = 0, \text{ ostatak } 10 \text{ (A)}$$

$$\rightarrow A3_{(16)}$$

$$163.25_{(10)} = A3.4_{(16)}$$

Decimalni dio:

$$0.25 \cdot 16 = 4.0 \rightarrow 4$$

$$\rightarrow .4_{(16)}$$

3.1.3.3. Veza između binarnog i heksadekadskog brojevnog sustava

Binarni i heksadekadski brojevni sustav međusobno su povezani jer je baza heksadekadskog sustava potencija broja 2: $16 = 2^4$.

Zbog toga se svaki heksadekadski zapis može jednostavno pretvoriti u binarni i obratno.

Za lakšu pretvorbu između heksadekadskog i binarnog brojevnog sustava koristimo se unaprijed određenom vezom između znamenaka. Svaka heksadekadaska znamenka odgovara jednoj skupini od četiri binarne znamenke.

U tablici 3.3. prikazani su odgovarajući binarni zapisi za svaku heksadekadsku znamenku.

Tablica 3.3. Ekvivalenti heksadekadskih i binarnih znamenaka

heksadekadaska znamenka	binarni ekvivalent	heksadekadaska znamenka	binarni ekvivalent
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Pretvorba iz binarnog u heksadekadski sustav

Binarni broj pretvara se u heksadekadski tako da se znamenke grupiraju po četiri (s desna na lijevo). Svaka grupa zamjenjuje se odgovarajućom heksadekadskom znamenkom.

Ako u posljednjoj grupi nedostaje znamenaka, **s lijeve strane dodaju se nule**.

Primjer 3.4.

Pretvori $110111101_{(2)}$ u heksadekadski brojevni sustav.

Rješenje

$$110111101_{(2)} = \text{0001 } 1011 \text{ } 1101_{(2)} = 1\text{B}\text{D}_{(16)}$$

Pretvorba iz heksadekadskog u binarni sustav

Heksadekadski broj pretvara se u binarni tako da se svaka znamenka zamijeni odgovarajućim **četveroznamenkastim binarnim zapisom**.

Primjer 3.5.

Pretvori $3E_{(16)}$ u binarni brojevni sustav.

Rješenje

Svaku heksadekadsku znamenku zamjenjujemo odgovarajućim četveroznamenkastim binarnim zapisom:

$$3E_{(16)} = 11\ 1110_{(2)}$$

$\swarrow \quad \searrow$
 0011 1110

3.1.4. Operacija zbrajanja s binarnim brojevima

Zbrajanje binarnih brojeva provodi se na isti način kao i u dekadskom sustavu, ali se upotrebljavaju samo znamenke 0 i 1.

Pri zbrajanju vrijede sljedeća pravila:

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 0$ i prenosi se 1 dalje.

Primjer 3.6.

Odredi zbroj $101.11_{(2)} + 10.101_{(2)}$.

Rješenje

Brojevi se zapisuju jedan ispod drugoga, a zbrajanje se provodi s desna na lijevo uz prijenos:

1 dalje →

		1	0	1	.	1	1	0	
+		0	1	0	.	1	0	1	
	1	0	0	0	.	0	1	1	

prijenos 1

Rješenje zbroja je
 $1000.011_{(2)}$

3.1.5. Pojam količine podataka

Podatci koje računalo obrađuje i pohranjuje zapisuju se s pomoću znamenaka 0 i 1. Kako bi računalo moglo pohraniti takve podatke, potrebno je imati osnovni element koji može razlikovati ta dva stanja.

Takav element naziva se **bistabil**.

Bistabil je elektronički sklop koji može imati **dva stabilna stanja**:

- stanje 0
- stanje 1.

Zbog toga se svaki bistabil može upotrijebiti za pohranu jedne binarne znamenke.

Radi jednostavnijeg prikaza bistabil se često prikazuje kao mali kvadratić u kojem je zapisana vrijednost 0 ili 1 (slika 3.1).



Slika 3.1. Dva bistabila: u lijevi bistabil spremljeno je stanje 0, a u desni bistabil stanje 1

Budemo li imali podatak koji treba pohraniti u više od jednog bita, onda ćemo trebati i više bistabila. Kada posložimo više bistabila jedan do drugog, dobit ćemo **registar** (slika 3.2).



Slika 3.2. Registar koji se sastoji od 8 bistabila – 8 bitova

Duljina registra određena je brojem bistabila. Za duljinu registra obično se uzima potencija broja 2 pa tako imamo sljedeće najčešće duljine registra: 8, 16, 32, 64 i 128.

Memorija računala predstavljena je kao niz registara jednake duljine. Duljina registra u memoriji naziva se **riječ**. Riječ predstavlja količinu podataka koju računalo može dohvatiti iz memorije, zapisati u memoriju ili obraditi. Današnji su procesori računala brži pa mogu obraditi veći broj bitova u jednom koraku. Ako imamo osobno računalo koje radi sa 64-bitovnim podacima, to znači da je **duljina registra – riječi** u memoriji **64 bita**. Uz riječ možemo definirati i **poluriječ** (32 bita), odnosno **dvostruku riječ** (128 bita).

Bit (engl. *binary digit*) osnovna je mjerna jedinica za količinu memorije. Počela se upotrebljavati sredinom 20. st. prilikom izrade prvih računala.

Radi jednostavnijeg zapisa 8 bitova zapisivat ćemo kao 1 bajt (engl. *byte*). Oznaka za bajt je B.

$$1 \text{ B} = 8 \text{ bit}$$

Veće mjerne jedinice kojima smo se ranije više koristili, a neki se njima i danas koriste su: kilobajt, megabajt, gigabajt, terabajt itd.

Prefiksi kilo, mega, giga, tera, peta i eksa u binarnom sustavu nemaju isto značenje kao i u dekadskom brojevnom sustavu. U binarnom su susjedne vrijednosti međusobno udaljene za koeficijent 2^{10} , odnosno 1024, dok je u dekadskom sustavu ta udaljenost 10^3 , odnosno 1000.

Zbog toga je u siječnju 1999. godine Međunarodna elektrotehnička komisija **IEC** (engl. *International Electrotechnical Commission*) uvela binarne prefikse za veće količine po-

dataka. Kod iskazivanja količine podataka treće i četvrto slovo prefiksa mijenjaju se u “bi”.

Primjerice, kilo postaje kibi, mega postaje mebi, giga postaje gibi, tera postaje tebi, peta postaje pebi, exa postaje exbi, zeta postaje zebi, yota postaje yobi. Decimalne i binarne prefikse prikazuje tablica 3.4.

Tablica 3.4. Decimalni i binarni prefiksi

decimalni naziv	decimalni prefiks	decimalna potencija	binarni naziv	binarni prefiks	binarna potencija
<i>kilobyte</i>	kB	10^3	<i>kibibyte</i>	KiB	2^{10}
<i>megabyte</i>	MB	10^6	<i>mebibyte</i>	MiB	2^{20}
<i>gigabyte</i>	GB	10^9	<i>gibibyte</i>	GiB	2^{30}
<i>terabyte</i>	TB	10^{12}	<i>tebibyte</i>	TiB	2^{40}
<i>petabyte</i>	PB	10^{15}	<i>pebibyte</i>	PiB	2^{50}
<i>exabyte</i>	EB	10^{18}	<i>exbibyte</i>	EiB	2^{60}
<i>zetabyte</i>	ZB	10^{21}	<i>zebibyte</i>	ZiB	2^{70}
<i>yotabyte</i>	YB	10^{24}	<i>yobibyte</i>	YiB	2^{80}

Primjer 3.7.

Preračunaj i rezultat zapiši kao potenciju broja 2:

a) 32 MiB = _____ KiB b) 512 Gibit = _____ TiB.

Rješenje

a) Pretvaramo iz **veće u manju** mjernu jedinicu. Prema tablici 3.4 uočavamo da je razlika između MiB i KiB 2^{10} pa pišemo $32 \cdot 2^{10} = 2^5 \cdot 2^{10} = 2^{15}$.

$$32 \text{ MiB} = 2^{15} \text{ KiB}$$

b) Pretvaramo iz **manje u veću** mjernu jedinicu. Prema tablici 3.4 uočavamo da je razlika između GiB i TiB 2^{10} , ali sada dijelimo s 2^{10} jer pretvaramo iz manje u veću mjernu jedinicu. Dodatno moramo voditi računa da moramo ujednačiti bit i bajt. Budući da pretvaramo iz **Gibit** u **TiB**, dodatno sve još moramo podijeliti s 8 jer je bit manja mjerna jedinica od bajta pa pišemo

$$512 : 2^{10} : 8 = 2^9 : 2^{10} : 2^3 = 2^{-4}.$$

$$512 \text{ Gibit} = 2^{-4} \text{ TiB}$$

Primjer 3.8.

Marija je pohranila na USB-u kapaciteta 8 GiB:

- deset pjesama, pri čemu svaka pjesma zauzima 5 MiB
- sto slika, pri čemu svaka slika zauzima 4 MiB
- deset videozapisa, pri čemu svaki videozapis zauzima 0,2 GiB.

Koliko je još mjesta preostalo na USB-u? Koliko Marija može dodati slika na USB (uz pretpostavku da svaka slika zauzima 4 MiB)?

Rješenje

Pjesme ukupno zauzimaju 50 MiB, slike 400 MiB, a videozapisi 2 GiB. Preostalu količinu memorije na USB-u možemo odrediti tako da od ukupnog kapaciteta oduzmemo veličine pojedinih multimedijских datoteka. Pri tome ćemo sve podatke izraziti u istoj mjernoj jedinici (MiB).

Ukupni kapacitet: $8 \text{ GiB} = 8 \cdot 1024 \text{ MiB} = 8192 \text{ MiB}$

Zauzeće videozapisa: $2 \text{ GiB} = 2 \cdot 1024 \text{ MiB} = 2048 \text{ MiB}$

Preostala količina memorije: $8192 \text{ MiB} - 50 \text{ MiB} - 400 \text{ MiB} - 2048 \text{ MiB} = 5694 \text{ MiB}$

Na USB-u preostalo je 5694 MiB memorije.

Da bismo odredili koliko je moguće dodati slika na USB, podijelit ćemo preostalu količinu memorije sa zauzećem jedne slike: $\frac{5694 \text{ MiB}}{4 \text{ MiB}} = 1423,5$.

Zaključujemo da je na USB moguće dodati 1423 slike.

3.1.6. Prikaz cijelih i realnih brojeva te znakova u memoriji računala

3.1.6.1. Prikaz cijelih brojeva u računalu

U matematici cijeli su brojevi definirani u skupu $\mathbb{Z}$:

$$\mathbb{Z} = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}.$$

Računalo, međutim, sve podatke pohranjuje u **binarni oblik** unutar memorije.

Zbog toga je potrebno definirati način na koji će se cijeli brojevi zapisivati u takvom obliku.

Za zapis cijelih brojeva u memoriji računala primjenjuje se **metoda dvojnog komplementa** koja omogućuje jednostavan prikaz pozitivnih i negativnih brojeva te izvođenje aritmetičkih operacija.

Postupak zapisa cijelih brojeva u memoriji računala metodom dvojnog komplementa

Cijeli brojevi u memoriji računala pohranjuju se u binarnom obliku unutar registara s pomoću metode dvojnog komplementa.

- **Pozitivni brojevi** zapisuju se izravno u binarnom sustavu u registar.
- **Negativni brojevi** zapisuju se s pomoću postupka dvojnog komplementa koji se sastoji od dvaju koraka:

1. Jedinični komplement

Svi bitovi binarnog zapisa apsolutne vrijednosti broja zamjenjuju se suprotnima ($0 \rightarrow 1, 1 \rightarrow 0$).

2. Dvojni komplement

Dobivenom zapisu pribroji se 1, čime se dobiva konačan zapis koji se pohranjuje u registar.

Krajnji lijevi bit u registru određuje predznak broja:

- 0 → pozitivan broj
- 1 → negativan broj.

Primjer 3.9.

Zapiši broj $+12_{(10)}$ u memoriji računala u 8-bitovnom registru metodom dvojnog komplementa.

Rješenje

Kada spremamo pozitivan broj u memoriji računala metodom dvojnog komplementa, zapisujemo ga izravno u binarnom obliku.

Broj $+12_{(10)}$ najprije pretvorimo u binarni zapis:

$$+12_{(10)} = 1100_{(2)}$$

Dobiveni binarni zapis upisujemo u 8-bitovni registar, a preostala mjesta popunjavamo nulama s lijeve strane:

0	0	0	0	1	1	0	0
---	---	---	---	---	---	---	---

Primjer 3.10.

Zapiši broj $-12_{(10)}$ u memoriji računala u 8-bitovnom registru metodom dvojnog komplementa.

Rješenje

Kada spremamo negativan broj u memoriji računala metodom dvojnog komplementa, najprije zapisujemo njegovu apsolutnu vrijednost u binarnom obliku, a zatim računamo dvojni komplement.

Binarna vrijednost broja $12_{(10)}$ je $1100_{(2)}$.

Budući da se služimo 8-bitovnim registrom, zapis dopunjavamo nulama s lijeve strane:

00001100

Zatim računamo:

Jedinični komplement:

11110011

Dvojni komplement:

11 (prijenos)

11110011

+ 1

11110100

Kako bi se pohranila vrijednost -15? Prvo izračunajte pa potom provjerite rješenje.

<https://s.element.hr/d805>



Rješenje upisujemo u 8-bitovni registar:

1	1	1	1	0	1	0	0
---	---	---	---	---	---	---	---

Napomena

Ako zbrojimo +12 i -12 zapisane u 8-bitovnim registrima metodom dvojnog komplementa, dobivamo nulu kao konačni rezultat.

+	0	0	0	0	1	1	0	0	+12
	1	1	1	1	0	1	0	0	-12
	0	0	0	0	0	0	0	0	0

Zadnja jedinica koja se prenosi (izvan registra) zanemaruje se pa je rezultat:

00000000

Primjer 3.11.

Koji je dekadski broj zapisan u 8-bitovnom registru ako znamo da se radi o zapisu cijelog broja s pomoću dvojnog komplementa?

0	0	0	1	1	0	0	1
---	---	---	---	---	---	---	---

Rješenje

Budući da je krajnji lijevi bit jednak 0, zaključujemo da je broj zapisan u registru **pozitivan**.

Zato njegovu vrijednost možemo izravno očitati iz binarnog zapisa zbrajanjem odgovarajućih težina:

$$0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 25.$$

Rješenje: radi se o broju +25₍₁₀₎.

Primjer 3.12.

Koji je dekadski broj zapisan u 8-bitovnom registru ako znamo da se radi o zapisu cijelog broja s pomoću dvojnog komplementa?

1	1	1	0	0	1	1	1
---	---	---	---	---	---	---	---

Rješenje

Budući da je krajnji lijevi bit jednak 1, zaključujemo da je broj zapisan u registru **negativan**.

Kako bismo odredili njegovu vrijednost, najprije računamo dvojni komplement, čime dobivamo apsolutnu vrijednost broja.

Polazimo od zadanog zapisa:

11100111

Računamo:

$$\begin{array}{r} 11100111 \\ 00011000 \text{ (jedinični komplement)} \end{array} \quad \begin{array}{r} 00011000 \\ + \quad 1 \\ \hline 00011001 \text{ (dvojni komplement)} \end{array}$$

Dobiveni zapis predstavlja pozitivan broj koji pretvaramo u dekadski sustav:

$$0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 25$$

Budući da je početni broj bio negativan, konačan rezultat je: $-25_{(10)}$.

Ako imamo registar od 8 bitova, možemo spremiti brojeve iz raspona:

$$-2^7 \text{ do } 2^7 - 1 = -128 \text{ do } 127.$$

Najveći pozitivni broj spremljen u registru od 8 bitova metodom predznaka i apsolutne vrijednosti je $01111111 = 2^7 - 1 = 127$.

Najmanji negativni broj spremljen u registru od 8 bitova metodom predznaka i apsolutne vrijednosti je $10000000 = -2^7 = -128$.

Ako imamo registar veličine n bitova, onda u njega možemo spremiti brojeve iz raspona:

$$-2^{n-1} \text{ do } 2^{n-1} - 1.$$

3.1.6.2. Prikaz realnih brojeva u računalu

Već smo vidjeli kako izgledaju binarni brojevi koji nisu cijeli. U nastavku ćemo proučiti kako se takvi brojevi zapisuju u memoriji računala.

Primjer konačnog periodičkog realnog broja:

$$\frac{4}{5} = 0.8$$

Primjer beskonačnog periodičkog realnog broja:

$$\frac{2}{3} = 0.666666666666666666...$$

Realni brojevi koji imaju beskonačan zapis ne mogu se točno pohraniti u memoriji računala pa se sprema njihova približna vrijednost.

Pri zapisu realnih brojeva u računalu koristimo se zapisom s pomičnim zarezom (engl. *floating point*).

IEEE (engl. *Institute of Electrical and Electronics Engineers*) standard 754 definira način zapisa realnih brojeva u memoriji računala. Razlikujemo realne brojeve jednostruke i dvostruke preciznosti.

Realni brojevi jednostruke preciznosti

Kada se podatci pohranjuju u memoriji računala, koristimo se **normaliziranim zapisom** binarnog broja. Na taj se način binarni brojevi mogu prikazati bez upotrebe velikog broja nula.

U normaliziranom zapisu binarnog broja razlikuju se dva dijela: **mantisa** i **binarni eksponent**. Mantisa predstavlja znamenke iza vodeće jedinice u normaliziranom zapisu,

3. Tipovi podataka i operatori

dok binarni eksponent označava za koliko je mjesta pomaknuta binarna točka ulijevo ili udesno kako bi se dobio normalizirani zapis.

Normalizirani zapis je oblika:

$$1.\text{mantisa} \cdot 2^E.$$

Pri tome se vodeća jedinica ne pohranjuje u memoriju jer se podrazumijeva (tzv. skriveni bit).

Primjer 3.13.

Zapiši u normaliziranom zapisu broj $101010110.1111_{(2)}$.

Rješenje

Postupak je isti kao i kad oblikujemo znanstveni zapis. U ovom primjeru pomičemo binarnu točku za 8 mjesta ulijevo te dobivamo

$$1.010101101111 \cdot 2^8$$

pri čemu je:

skriveni bit (engl. *hidden bit*): 1

MANTISA bez skrivenog bita: 010101101111

BAZA: 2

BINARNI EKSPONENT: 8.

Primjer 3.14.

Zapiši u normaliziranom zapisu broj $0.00000101011_{(2)}$.

Rješenje

U ovom slučaju pomičemo binarnu točku za 6 mjesta udesno te dobivamo

$$1.01011 \cdot 2^{-6}$$

pri čemu je:

skriveni bit (engl. *hidden bit*): 1

MANTISA bez skrivenog bita: 01011

BAZA: 2

BINARNI EKSPONENT: -6.

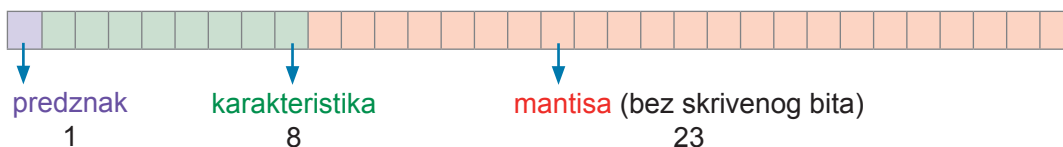
Realni brojevi jednostruke preciznosti zapisuju se u 32-bitovni registar koji se sastoji od triju dijelova.

a) predznak P (1 bit):

- negativni broj: 1
- pozitivni broj: 0

b) karakteristika K (8 bitova):

- dobiva se tako da na binarni eksponent dodamo 127 : $K = E + 127$
- binarni eksponent E može biti u rasponu od $[-126, 127]$
- karakteristika K može biti u rasponu od $[0, 255]$

c) mantisa M (23 bita)


Realan dekadski broj zapisujemo u obliku binarnog broja slijedeći korake:

- odrediti predznak P (0 ili 1)
- pretvoriti dekadski broj u binarni broj
- normalizirati binarni broj te očitati vrijednosti mantise bez skrivenog bita i vrijednost binarnog eksponenta
- izračunati vrijednost karakteristike ($K = E + 127$) te ju pretvoriti iz dekadskog u binarni zapis
- upisati P, K i M u 32-bitovni registar.

Primjer 3.15.

Realan broj $-60.75_{(10)}$ prikazan je u računalu prema IEEE 754 standardu jednostruke preciznosti. Odredi binarnu vrijednost toga zapisa.

Rješenje

Pretvorbu radimo po definiranim koracima:

- predznak je minus “-” pa pod P pišemo $P = 1$

- $60.75_{(10)} = 111100.11_{(2)}$

$0.75 \cdot 2 = 1.5 \rightarrow 1$	$60 : 2 = 30$ i ostatak 0	↑
$0.5 \cdot 2 = 1.0 \rightarrow 1$	$30 : 2 = 15$ i ostatak 0	
	$15 : 2 = 7$ i ostatak 1	
	$7 : 2 = 3$ i ostatak 1	
	$3 : 2 = 1$ i ostatak 1	
	$1 : 2 = 0$ i ostatak 1	

- $111100.11_{(2)} = 1.1110011 \cdot 2^5_{(2)}$

očitavamo: $E = 5$, $M = 11100110000000000000000$ (popunjavamo nulama dok ne popunimo 23 bita)

- $K = E + 127 = 5 + 127 = 132_{(10)} = 10000100_{(2)}$

- upisujemo s lijeva na desno vrijednosti od P, K i M u 32-bitovni registar:

1 1 0 0 0 0 1 0 0 1 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Primjer 3.16.

Zadan je binarni zapis sadržaja koji predstavlja zapis realnog broja prema IEEE 754 standardu jednostruke preciznosti. Zapis ima oblik **11000010000000010000000000000000**. Koji je to dekadski broj?

Rješenje

Postupak je sada obrnut.

1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Iz dobivenog zapisa očitavamo predznak $P = 1$ pa odmah zaključujemo da se radi o negativnom realnom broju.

Karakteristika iznosi: $K = 10000100_{(2)} = 132_{(10)}$.

Iz karakteristike računamo vrijednost eksponenta: $E = K - 127 = 132 - 127 = 5$.

Vrijednost mantise: $M = 000000100000000000000000$.

Pišemo binarni broj u normaliziranom obliku: $1.0000001 \cdot 2^5$.

Standardni zapis glasi $100000.01_{(2)}$.

Na kraju je potrebno pretvoriti dobiveni binarni broj u dekadski zapis:

1	0	0	0	0	0	.	0	1
2^5	2^4	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}

Rješenje je: $-32.25_{(10)}$.

Kako biste prema ovom standardu zapisali broj $12.625_{(10)}$?
Provjerite rezultat putem QR koda.



Za one koji žele znati više

U programiranju se javljaju i neke posebne brojčane vrijednosti s kojima se do sada nismo susretali. Vrlo lako možete uočiti da ako prvi bit postavimo u 1, a sve preostale na nulu, dobivamo vrijednost -0 . Iz matematike nam je poznato da nula nije ni pozitivan ni negativan broj. Ove posebnosti nećemo detaljno analizirati, nego ćemo samo navesti posebne slučajeve te njihove zapise prema navedenom standardu. To su vrijednosti: $+0$, -0 , $+\infty$ (plus beskonačno, engl. *positive infinity*), $-\infty$ (minus beskonačno, engl. *negative infinity*) i **NaN** (engl. *Not a Number*).

	binarna vrijednost	heksadekadska vrijednost	vrijednost karakteristike
$+0$	0000 0000 0000 0000 0000 0000 0000 0000	00000000	(K=0)
-0	1000 0000 0000 0000 0000 0000 0000 0000	80000000	(K=0)
$+\infty$	0111 1111 1000 0000 0000 0000 0000 0000	7F800000	(K=255)
$-\infty$	1111 1111 1000 0000 0000 0000 0000 0000	FF800000	(K=255)

NaN i $\pm\infty$ (engl. *infinity*) mogu biti rezultati nedopuštenih matematičkih operacija, npr. dijeljenja s nulom. O tome ćete više učiti kasnije.

3.1.6.3. Prikaz znakova standardnim ASCII kodom

Za zapis slova i ostalih znakova u sklopovima računala možemo se koristiti različitim načinima zapisa. Jedan od njih je i **standardni ASCII kôd**. Za njega vrijedi sljedeće:

- informacija sadržava 7 bitova, a osmi se bit upotrebljava za paritet koji služi za zaštitu od pogrešaka što mogu nastati prilikom prijenosa i pohrane podataka
- u memoriji jedan znak zauzima jedan bajt, odnosno 8 bitova
- moguće je prikazati $2^7 = 128$ znakova
- popis svih znakova koji se mogu ispisati prikazan je u tablici 3.5:

Tablica 3.5. Izvod iz ASCII tablice

dek.	heks.	znak	dek.	heks.	znak	dek.	heks.	znak	dek.	heks.	znak
0	0	NUL (prazno)	32	20	raz-mak	64	40	@	96	60	`
1	1	SOH (početak zaglavlja)	33	21	!	65	41	A	97	61	a
2	2	STX (početak teksta)	34	22	"	66	42	B	98	62	b
3	3	ETX (kraj teksta)	35	23	#	67	43	C	99	63	c
4	4	EOT (kraj prijenosa)	36	24	\$	68	44	D	100	64	d
5	5	ENQ (upit)	37	25	%	69	45	E	101	65	e
6	6	ACK (potvrda)	38	26	&	70	46	F	102	66	f
7	7	BEL (zvono)	39	27	'	71	47	G	103	67	g
8	8	BS (brisanje unazad)	40	28	(	72	48	H	104	68	h
9	9	TAB (horizontalni tabulator)	41	29	)	73	49	I	105	69	i
10	A	LF (novi red)	42	2A	*	74	4A	J	106	6A	j
11	B	VT (vertikalni tabulator)	43	2B	+	75	4B	K	107	6B	k
12	C	FF (nova stranica)	44	2C	,	76	4C	L	108	6C	l
13	D	CR (povratak na početak retka)	45	2D	-	77	4D	M	109	6D	m
14	E	SO (<i>shift out</i>)	46	2E	.	78	4E	N	110	6E	n
15	F	SI (<i>shift in</i>)	47	2F	/	79	4F	O	111	6F	o
16	10	DLE (upravljanje podatkovnom vezom)	48	30	0	80	50	P	112	70	p
17	11	DC1 (kontrola uređaja 1)	49	31	1	81	51	Q	113	71	q
18	12	DC2 (kontrola uređaja 2)	50	32	2	82	52	R	114	72	r
19	13	DC3 (kontrola uređaja 3)	51	33	3	83	53	S	115	73	s
20	14	DC4 (kontrola uređaja 4)	52	34	4	84	54	T	116	74	t
21	15	NAK (negativna potvrda)	53	35	5	85	55	U	117	75	u
22	16	SYN (sinkronizacija)	54	36	6	86	56	V	118	76	v
23	17	ETB (kraj bloka prijenosa)	55	37	7	87	57	W	119	77	w
24	18	CAN (prekid)	56	38	8	88	58	X	120	78	x
25	19	EM (kraj medija)	57	39	9	89	59	Y	121	79	y
26	1A	SUB (zamjena)	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC (<i>escape</i>)	59	3B	;	91	5B	[	123	7B	{
28	1C	FS (razdjelnik datoteka)	60	3C	<	92	5C	\	124	7C	
29	1D	GS (razdjelnik grupa)	61	3D	=	93	5D	]	125	7D	}
30	1E	RS (razdjelnik zapisa)	62	3E	>	94	5E	^	126	7E	~
31	1F	US (razdjelnik jedinica)	63	3F	?	95	5F	_	127	7F	DEL

Specifični ASCII znakovi s dekadskim vrijednostima od 0 do 31 su kontrolni znakovi i ne ispisuju se kao vidljivi znakovi na zaslonu računala. Na primjer, NUL znak ima vrijednost 0 i upotrebljava se za označavanje kraja niza znakova.

U prikazanoj tablici 3.5 ne nalaze se sva slova hrvatske abecede. Nedostaju **Č, Ć, Đ, Š, Ž, č, ć, đ, š i ž**. Ona se nalaze u tablici koju definira **prošireni 8-bitovni ASCII kôd**. Prema tom standardu 1 znak zauzima 8 bitova (1 bajt) te je zato moguće prikazati i razne specijalne znakove jer se osmi bit više ne upotrebljava kao paritetni (zaštitni) bit. Prvih 128 znakova isto je kao i kod standardnog ASCII koda.

Uz navedeno postoji i **UNICODE-16** kojim se također kodiraju znakovi, a u memoriji računala jedan znak zauzima 2 bajta.

Zanimljivost

Danas se često primjenjuje način zapisa UTF-8 kao najčešće kodiranje UNICODE znakova za elektroničku poštu i web-stranice. Znak zapisan u UTF-8 može biti dugačak od 1 do 4 bajta. Često birani znakovi zapisuju se jednim bajtom, rjeđe birani znakovi zapisuju se s više bajtova. UTF-8 je kompatibilan s ASCII znakovima. Prvih 128 znakova UNICODE-16 znakova prema UTF-8 zapisuje se s pomoću jednog bajta, a iste su binarne vrijednosti kao ASCII znakovi. Za zapise hrvatskih znakova č, ć, đ, š, ž te grčkih slova UTF-8 zapisom potrebna su po dva bajta.

Primjer 3.17.

Maja je napisala svoju prvu knjigu na engleskom jeziku. Bio je to roman. Koliko će prostora (u kibibajtima) u memoriji zauzeti knjiga ako ima ukupno 240 stranica? Na svaku stranicu knjige Maja je napisala po 4096 znakova. Za zapis znakova koristi se standardnim ASCII kodom.

Rješenje

Ukupnu memoriju (broj stranica · broj znakova · veličina 1 znaka) izračunamo u bajtovima (brojnik) i pretvorimo ih u KiB tako da brojnik podijelimo s 1024 (za prefiks *kibibyte*).

$$M = \frac{240 \cdot 4096 \cdot 1}{1024} = 960 \text{ KiB}$$

Primjer 3.18.

Odgovori na pitanja:

- Koliko mjesta u memoriji računala zauzima ime **Ksenija**? Za zapis znakova koristimo se standardnim ASCII kodom.
- Svaki znak u imenu **Ksenija** zapiši dekadskim zapisom (s pomoću ASCII tablice).
- S pomoću ASCII tablice zapiši ime **Ksenija** kako ga pohranjuje računalo (kombinacijom nula i jedinica).

Rješenje

a) Riječ Ksenija sastoji se od 7 slova. Svako slovo zauzima 8 bitova, što ukupno daje zauzeće memorije od 56 bitova.

b) S pomoću ASCII tablice očitavamo vrijednosti:

$K = 75_{(10)}$ $s = 115_{(10)}$ $e = 101_{(10)}$ $n = 110_{(10)}$ $i = 105_{(10)}$ $j = 106_{(10)}$ $a = 97_{(10)}$.

c) Da bismo prikazali kako izgleda traženi zapis u memoriji računala, svaku dekadsku vrijednost potrebno je prikazati u odgovarajućem binarnom zapisu te slijeva dodati onoliko vodećih nula koliko je potrebno tako da binarni broj ima 8 znamenaka (1 znak zapisan standardnim ASCII kodom u memoriji računala zauzima 8 bitova).

Prikazat ćemo samo pretvorbu dekadске vrijednosti $75_{(10)}$:

$75 : 2 = 37$	i ostatak 1		01001011
$37 : 2 = 18$	i ostatak 1		
$18 : 2 = 9$	i ostatak 0		
$9 : 2 = 4$	i ostatak 1		
$4 : 2 = 2$	i ostatak 0		
$2 : 2 = 1$	i ostatak 0		
$1 : 2 = 0$	i ostatak 1		

Na isti način pretvorimo i preostale dekadské brojeve te rješenje zapišemo u 8-bitovne registre kako je prikazano u nastavku:

K	0	1	0	0	1	0	1	1
s	0	1	1	1	0	0	1	1
e	0	1	1	0	0	1	0	1
n	0	1	1	0	1	1	1	0
i	0	1	1	0	1	0	0	1
j	0	1	1	0	1	0	1	0
a	0	1	1	0	0	0	0	1

Zapiši svoje ime u binarnom ASCII zapisu i provjeri rješenje skeniranjem QR koda.



<https://s.element.hr/4b53>

Pretvorbu je moguće napraviti i direktno iz heksadekadskog u binarni brojevni sustav, što je brže u ovom slučaju.

3.1.6.4. Osnovni tipovi podataka u programskom jeziku C i njihov prikaz u memoriji računala

U programskom jeziku C različiti tipovi podataka zauzimaju različitu količinu memorije. Način na koji se podatci pohranjuju u memoriji ovisi o njihovom tipu.

Za najčešće odabrane tipove vrijedi:

- tip **char** zauzima **1 bajt** (8 bitova)
- tip **int** zauzima **4 bajta** (32 bita)
- tip **float** zauzima **4 bajta** (32 bita).

To znači da se vrijednost varijable tipa **char** pohranjuje u registru od 8 bitova, dok se vrijednosti tipova **int** i **float** pohranjuju u registru od 32 bita.

Pri tome:

- tip `int` upotrebljava dvojni komplement za zapis cijelih brojeva
- tip `float` upotrebljava IEEE 754 zapis jednostruke preciznosti za realne brojeve.

Napomena

Veličina pojedinih tipova podataka može ovisiti o arhitekturi računala i odabranom prevoditelju (kompajleru) pa se može razlikovati između različitih sustava.

Primjer 3.19.

Kako se u memoriji računala pohranjuje vrijednost pridružena varijabli znak?

```
char znak = 'A';
```

Rješenje

Znak se u memoriji računala ne pohranjuje kao slovo, već kao broj koji mu je pridružen prema **ASCII tablici**.

Prema ASCII tablici znak '**A**' ima:

- dekadsku vrijednost 65
- heksadekadsku vrijednost 41

Heksadekadski zapis **41**₍₁₆₎ može se jednostavno pretvoriti u binarni tako da se svaka heksadekadaska znamenka zamijeni s 4 bita:

- 4 → 0100
- 1 → 0001

Taj se broj zatim zapisuje u memoriju u obliku binarnog zapisa.

Binarni zapis broja 65₍₁₀₎ u 8-bitovnom registru je:

0	1	0	0	0	0	0	1
---	---	---	---	---	---	---	---

Primjer 3.20.

Kako se u memoriji računala pohranjuje vrijednost pridružena varijabli broj?

```
int broj = 5;
```

Rješenje

Cjelobrojna vrijednost pohranjuje se u memoriji u obliku binarnog zapisa zauzimajući 32 bita.

Broj 5 zapisuje se u memoriji u 32-bitovnom registru kao:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ako je vrijednost negativna, primjenjuje se zapis s pomoću **dvojnog komplementa**.

```
int broj = -5;
```

Najprije se zapisuje apsolutna vrijednost broja:

0 1 0 1

Zatim se primjenjuje dvojni komplement (jedinični komplement se zbroji s 1):

1 0 1 1

Primjer 3.21.

Kako se u memoriji računala pohranjuje vrijednost pridružena varijabli x?

```
float x = 5.0;
```

Rješenje

Realna vrijednost pohranjuje se u memoriji prema **IEEE 754 standardu** **jednostruke preciznosti** koji zapis dijeli na tri dijela: predznak, eksponent i mantisu.

Napomena

Način zapisa realnih brojeva detaljnije objašnjava poglavlje 3.1.5.

Vrijednost 5.0 zapisuje se u memoriji kao:

0 1 0 0 0 0 0 0 1 0 1 0

Taj se zapis može prikazati i u heksadekadskom obliku grupiranjem bitova u skupine od četiri: 40A00000

3.2. Operatori

Neki od operatora kojima se koristimo prilikom pisanja kodova u programskom jeziku C su:

1. operator pridruživanja
2. aritmetički operatori
3. relacijski operatori
4. logički operatori
5. operatori inkrement i dekrement
6. operator sizeof
7. bitovni operatori.

3.2.1. Operator pridruživanja

Oznaka **operatora pridruživanja** znak je jednakosti `=`. Primjena ovog operatora već je pokazana prilikom inicijalizacije varijabli, ali i dodatno primjerom 3.22.

Primjer 3.22.

Napišite program u kojem ćete definirati cjelobrojne varijable `a`, `b` i `c` te dodatno inicijalizirati varijablu `b` na vrijednost 2. Varijabli `a` pridružite vrijednost varijable `b` uvećanu za 1 te potom varijabli `c` pridružite vrijednost varijable `a`. Ispišite vrijednosti varijabli `a`, `b` i `c`, svaku u novom redu.

Rješenje

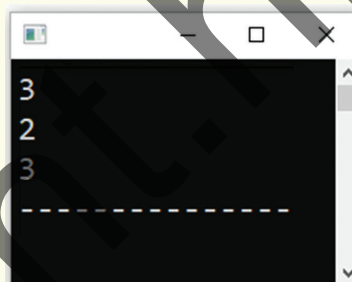
Prikaz rješenja na zaslonu računala nalazi se na slici 3.3.

```
#include <stdio.h>

int main(void) {
    int a, b = 2, c;

    a = b + 1;
    c = a;
    printf("%d\n%d\n%d", a, b, c);

    return 0;
}
```



Slika 3.3. Prikaz ispisa cijelih brojeva na zaslonu računala

3.2.2. Aritmetički operatori

U prethodnom poglavlju nabrojeni su **aritmetički operatori**. U primjeru 3.23 redom su prikazani svi aritmetički operatori (`+`, `-`, `*`, `/` i `%`) te se analizira kako će se mijenjati vrijednost rezultata kod dijeljenja u ovisnosti o tipu operanda.

Primjer 3.23.

Napišite program koji ispisuje rezultat zbrajanja, oduzimanja, množenja, cjelobrojnog dijeljenja te cjelobrojnog dijeljenja s ostatkom za dva broja. Testirajte program ako su varijable definirane kao tip `int` i kao tip `float`. Izvedite zaključak.

Rješenje

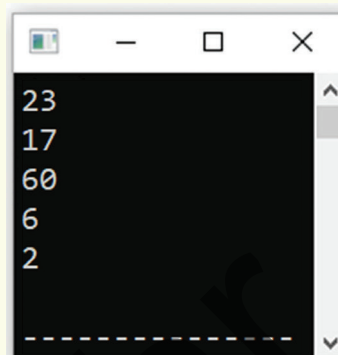
Ako su varijable tipa `int`, ispis na zaslonu računala je kao što prikazuje slika 3.4.

```
#include <stdio.h>

int main(void) {
    int a = 20, b = 3, z, r, u, d, d_ost;

    z = a + b;
    r = a - b;
    u = a * b;
    d = a / b;
    d_ost = a % b;

    printf("%d\n%d\n%d\n%d\n%d\n", z, r,
           u, d, d_ost);
    return 0;
}
```



Slika 3.4. Ispis rezultata (primjena aritmetičkih operatora)

Ako su varijable tipa `float`, dobit će se informacija o pogrešci prilikom kompajliranja programa. Ispis pogreške prikazuje slika 3.5.

```
#include <stdio.h>

int main(void) {
    float a = 20, b = 3, z, r, u, d, d_ost;

    z = a + b;
    r = a - b;
    u = a * b;
    d = a / b;
    d_ost = a % b;

    printf("%f\n%f\n%f\n%f\n%f\n", z, r, u, d, d_ost);
    return 0;
}
```

Linija	Col	Lokacija	Poruka
		C:\Users\Vatroslav\Documents\main.c	In function 'main':
10	12	C:\Users\Vatroslav\Documents\main.c	[Error] invalid operands to binary % (have 'float' and 'float')
28		C:\Users\Vatroslav\Documents\Makefile.win	recipe for target 'main.o' failed

Slika 3.5. Cjelobrojno dijeljenje s ostatkom definirano je samo za cjelobrojne tipove podataka

Zaključak: cjelobrojno dijeljenje s ostatkom definirano je samo za podatke koji su cjelobrojnog tipa.

Proširite program

Omogućite unos brojeva *a* i *b* te potom testirajte program za različite vrijednosti (prvo ih sami izračunajte, a potom provjerite rezultat).



3. Tipovi podataka i operatori

U prikazanom primjeru dolazi do **pogreške pri kompajliranju** jer operator % (ostatak pri dijeljenju) nije definiran za realne (float) tipove podataka.

Ova pogreška pripada skupini **semantičkih pogrešaka** koje nastaju zbog **nepodudaranja tipova podataka i operatora**.

Program se zbog toga ne može prevesti (kompajlirati).

Primjer 3.24.

Napišite program u koji se učitava jedan prirodan četveroznamenasti broj. Ispišite zbroj svih znamenaka učitano broj. Napomena: pretpostavite da je broj sigurno četveroznamenast.

ulazni podatci	izlazni podatci
2021	Zbroj znamenaka broja 2021 je 5.

Rješenje

```
#include <stdio.h>

int main(void) {
    int x;                //deklaracija varijable koja predstavlja broj
    int t, s, d, j;        //znamenke broja x
    int zbroj = 0;         //definirana i inicijalizirana varijabla zbroj

    printf("Ucitajte cetveroznamenasti broj: ");
    scanf("%d", &x);

    j = x % 10;            //jedinica broja
    d = x % 100 / 10;      //desetica broja
    s = x % 1000 / 100;    //stotica broja
    t = x / 1000;          //tisućica broja

    zbroj = j + d + s + t;

    printf("Zbroj znamenaka broja %d je %d.", x, zbroj);

    return 0;
}
```

Izmijenite program

Što bi trebalo promijeniti u kodu ako bismo htjeli ispisati zbroj svih znamenaka broja za koji sigurno znamo da je peteroznamenasti? Napišite taj program i testirajte ga.



3.2.3. Relacijski operatori

Relacijski operatori su:

- veće (>)
- manje (<)
- veće od ili jednako (>=)
- manje od ili jednako (<=)
- različito (!=)
- jednako (==).

S pomoću njih se određuje odnos među operandima, a rezultat usporedbe je **logička istina** (1) ili **logička laž** (0).

Primjer 3.25.

Odredite što će se ispisati na zaslonu računala pokretanjem sljedećeg programa.

```
#include <stdio.h>

int main(void) {
    int a, b, c, d, e, f;

    a = 2 > 3;
    b = 5 < 9;
    c = 5 <= 3;
    d = 2 >= 1;
    e = 10 == 3;
    f = 2 != 3;

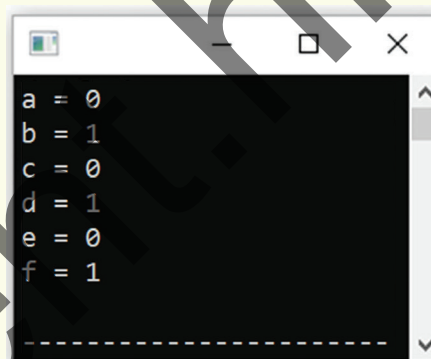
    printf("a = %d\n", a);
    printf("b = %d\n", b);
    printf("c = %d\n", c);
    printf("d = %d\n", d);
    printf("e = %d\n", e);
    printf("f = %d\n", f);

    return 0;
}
```

Rješenje

Ako je rezultat usporedbe logička istina, kao rezultat se ispiše vrijednost 1.

U suprotnom je ispis 0. Prikaz rješenja na zaslonu računala nalazi se na slici 3.6. U izrazima poput $a = 2 > 3$ rezultat usporedbe (1 ili 0) pridružuje se varijabli.



```
a = 0
b = 1
c = 0
d = 1
e = 0
f = 1
```

Slika 3.6. Rezultati ispisa, primjena relacijskih operatora



Proširite program

Promijenite relacijske operatore tako da se svi rezultati zamijene (0 postaje 1, a 1 postaje 0).

Pri radu s relacijskim operatorima važno je razlikovati **operator pridruživanja (=)** i **operator jednakosti (==)**. Ova razlika često predstavlja izvor pogrešaka kod početnika jer oba operatora izgledaju slično, ali imaju potpuno različitu ulogu u programu.

U tablici 3.6 prikazana je usporedba njihova značenja i ponašanja.

Tablica 3.6. Razlika između operatora pridruživanja (=) i operatora jednakosti (==)

operator	značenje	primjer	rezultat	objašnjenje
=	pridruživanje vrijednosti	<code>a = 3;</code>	a postaje 3	vrijednost se sprema u varijablu
==	usporedba (jednakost)	<code>a == 3</code>	1 ili 0	provjerava jesu li vrijednosti jednake

3.2.4. Logički operatori

Logički operatori (uz relacijske) omogućuju donošenje logičkih odluka u programiranju, na primjer kod uvjetnih `if` izraza. S pomoću logičkih operatora mogu se pisati i složeniji logički izrazi sastavljeni od jednostavnih izraza koji u sebi mogu sadržavati i relacijske operatore. Općenito, kao rezultat primjene logičkih operatora dobiva se **logička istina (1)** ili **logička laž (0)**. Logički operatori, navedeni prema prioritetu izvršavanja, jesu:

- NE `!`
- I `&&`
- ILI `||`

Pregled logičkih operatora i njihovih rezultata primjene nalazi se u nastavku:

logički NE	logički I	logički ILI
<code>!0 = 1</code> <code>!1 = 0</code>	<code>0 && 0 = 0</code> <code>0 && 1 = 0</code> <code>1 && 0 = 0</code> <code>1 && 1 = 1</code>	<code>0 0 = 0</code> <code>0 1 = 1</code> <code>1 0 = 1</code> <code>1 1 = 1</code>
Ovaj operator invertira vrijednost logičkog izraza.	Ovaj operator vraća true (1) samo ako su oba operanda true (1). U svim ostalim slučajevima vraća false (0).	Ovaj operator vraća true (1) ako je barem jedan od operanada true (1). Ako su oba operanda false (0), tada vraća false (0).

Primjer 3.26.

Odredite što će se ispisati na zaslону računala pokretanjem sljedećeg programa.

```
#include <stdio.h>

int main(void) {
    int a, b, c, d;

    a = (2 > 3) || (5 < 9);
    b = (2 >= 1) && (10 == 3);
    c = !(2 != 3);
    d = !(2 != 3) || (5 < 9) && (10 == 3);

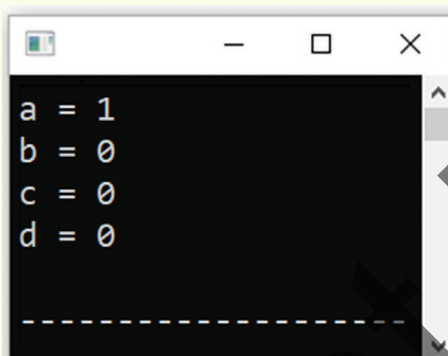
    printf("a = %d\n", a);
    printf("b = %d\n", b);
    printf("c = %d\n", c);
    printf("d = %d\n", d);

    return 0;
}
```


Rješenje

Postupak i prikaz rješenja nalazi se na slici 3.7. Prilikom određivanja stanja varijable *d* treba voditi računa o prioritetu operatora.

```
a = (2 > 3) || (5 < 9);           //0 || 1 --> 1
b = (2 >= 1) && (10 == 3);        //1 && 0 --> 0
c = !(2 != 3);                   //!(1) --> 0
d = !(2 != 3) || (5 < 9) && (10 == 3); //0 || 1 && 0 --> 0 || 0 --> 0
```



Slika 3.7. Prikaz rješenja, primjena logičkih operatora

Primjer 3.27.

Napišite program koji učitava dva slova engleske abecede s tipkovnice i ispisuje rezultat njihove usporedbe.

Ako su slova jednaka (neovisno o tome radi li se o velikim ili malim slovima), ispisuje se vrijednost 1, a u suprotnom 0.

Pretpostavite da se na ulazu uvijek unose slova engleske abecede.

ulazni podatci	izlazni podatci
B B	1
a A	1
A b	0

Rješenje

Rezultat usporedbe znakova može se pohraniti u logičku varijablu tipa **bool** koja prima vrijednost **true** (1) ili **false** (0).

Usporedba se provodi s pomoću relacijskog operatora **==** i činjenice da se velika i mala slova u ASCII tablici razlikuju za 32.

```
#include <stdio.h>
#include <stdbool.h>

int main(void) {
    char a, b;
    bool isti;

    scanf(" %c %c", &a, &b);

    isti = (a == b) || (a == b + 32) || (a == b - 32);

    printf("%d\n", isti);

    return 0;
}
```

3.2.5. Operatori inkrement i dekrement

Operator inkrement (++) povećava vrijednost varijable za 1, a **dekrement** (--) smanjuje za 1.

U tablicama 3.7. i 3.8. nalazi se pregled navedenih operatora s dodatnim objašnjenjem.

Tablica 3.7. Operator inkrement ++

izraz	ekvivalentne naredbe	opis
<code>i = x++;</code>	<code>i = x;</code> <code>x = x + 1;</code>	Vrijednost varijable <code>x</code> prvo pridruži varijabli <code>i</code> i potom <code>x</code> uvećaj za 1.
<code>i = ++x;</code>	<code>x = x + 1;</code> <code>i = x;</code>	Vrijednost varijable <code>x</code> prvo uvećaj za 1 i potom uvećanu vrijednost varijable <code>x</code> pridruži varijabli <code>i</code> .

Tablica 3.8. Operator dekrement --

izraz	ekvivalentne naredbe	opis
<code>i = x--;</code>	<code>i = x;</code> <code>x = x - 1;</code>	Vrijednost varijable <code>x</code> prvo pridruži varijabli <code>i</code> i potom <code>x</code> umanj za 1.
<code>i = --x;</code>	<code>x = x - 1;</code> <code>i = x;</code>	Vrijednost varijable <code>x</code> prvo umanj za 1 i potom umanjenu vrijednost varijable <code>x</code> pridruži varijabli <code>i</code> .

Primjer 3.28.

Odredite što će se ispisati na zaslону računala pokretanjem sljedećeg programa.

```
#include <stdio.h>

int main(void) {
    int a = 1, b = 2;

    a++;    //++a;
    --b;    //--b;
    a = a + (++b);
    b = b + (a--);

    printf("a = %d\n", a);
    printf("b = %d\n", b);

    return 0;
}
```

Rješenje

Stanje varijabli može se odrediti kako je prikazano u nastavku tablicom 3.9. Ispis na zaslону računala prikazuje slika 3.8.

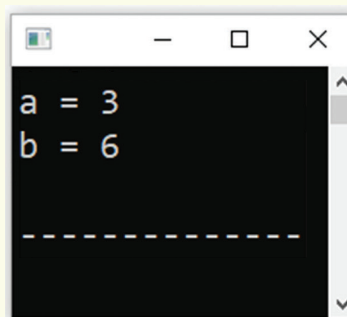
Proširite program

Omogućite unos vrijednosti za `a` i `b`, zatim u svim linijama koda promijenite `a++` u `++a`, `++b` u `b++` te prvo izračunajte i potom provjerite rezultat.



Tablica 3.9. Stanje varijabli za primjer 3.28.

naredba	a	b
<code>int a = 1, b = 2;</code>	1	2
<code>a++; //++a;</code>	2	
<code>--b; //b--;</code>		1
<code>a = a + (++b);</code>	4	2
<code>b = b + (a--);</code>	3	6

**Slika 3.8.** Ispis na zaslonu računala, primjena operatora inkrement i dekrement

3.2.6. Operator sizeof

S pomoću **operatora sizeof** može se odrediti koliko bajtova u memoriji računala zauzima pojedini tip podatka. Operator sizeof vraća vrijednost tipa `size_t`. Tip `size_t` upotrebljava se za prikaz veličine podataka u memoriji i predstavlja neoznačeni cijeli broj (engl. *unsigned integer*). Pri usporedbi s varijablama tipa `int` potrebno je biti oprezan jer se radi o različitim tipovima podataka.

Ovisno o kompajleru, operator `sizeof` može davati drukčije vrijednosti. U slučaju kompajlera kojima se koristimo u sklopu razvojnog okruženja Code::Blocks, podatak tipa `int` zauzima 4 B, podatak tipa `float` 4 B, a podatak tipa `char` 1 B. U poglavlju 3.2. dodatno je objašnjeno koliko koji osnovni tip podataka zauzima mjesta u memoriji računala.

Primjer 3.29.

Odredite što će se ispisati na zaslonu računala pokretanjem sljedećeg programa.

```
#include <stdio.h>

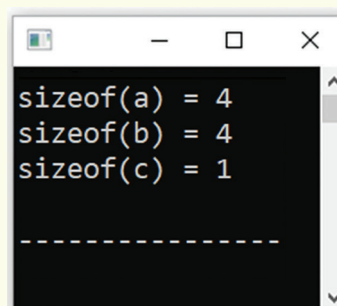
int main(void) {
    int a = 30;
    float b = 2.5;
    char c = 'A';

    printf("sizeof(a) = %d\n", sizeof(a));
    printf("sizeof(b) = %d\n", sizeof(b));
    printf("sizeof(c) = %d\n", sizeof(c));

    return 0;
}
```

Rješenje

Rješenje ispisa prikazuje slika 3.9.

**Slika 3.9.** Ispis na zaslonu računala, primjena operatora `sizeof`

3.2.7. Bitovni operatori

Za rad s bitovima primjenjuju se **bitovni operatori**.

Bitovni operatori su:

- bitovni ILI (|)
- bitovni I (&)
- bitovni ekskluzivni ILI (^)
- desni pomak (>>)
- lijevi pomak (<<)
- jedinični komplement (~).

Pregled bitovnih operatora i njihovih rezultata nalazi se u nastavku:

bitovni I	bitovni ILI	bitovni ekskluzivni ILI	jedinični komplement
$0 \& 0 = 0$	$0 \mid 0 = 0$	$0 \wedge 0 = 0$	$\sim 0 = 1$
$0 \& 1 = 0$	$0 \mid 1 = 1$	$0 \wedge 1 = 1$	$\sim 1 = 0$
$1 \& 0 = 0$	$1 \mid 0 = 1$	$1 \wedge 0 = 1$	
$1 \& 1 = 1$	$1 \mid 1 = 1$	$1 \wedge 1 = 0$	

Operatorima

- desni pomak ($x \gg \text{broj_bitova_za_koliko_se_izvršava_pomak}$)
- lijevi pomak ($x \ll \text{broj_bitova_za_koliko_se_izvršava_pomak}$)

izvršava se operacija pomaka bitova u registru za definirani broj bitova.

Primjena navedenih bitovnih operatora prikazana je u sljedećim primjerima.

Primjer 3.30.

Odredite kako će izgledati ispis na zaslonu računala.

```
#include <stdio.h>

int main(void) {
    int a = 30, b = -5, x1, x2, x3, x4, x5, x6;

    x1 = a >> 2;
    x2 = b << 3;
    x3 = ~b;
    x4 = a | b;
    x5 = a & b;
    x6 = a ^ b;

    printf("x1 = %d\n", x1);
    printf("x2 = %d\n", x2);
    printf("x3 = %d\n", x3);
    printf("x4 = %d\n", x4);
    printf("x5 = %d\n", x5);
    printf("x6 = %d\n", x6);

    return 0;
}
```

Rješenje

Vrijednosti se zapisuju u 32-bitovne registre metodom dvojnog komplementa (podatak tipa `int` zauzima 4 B u memoriji računala).

Desni pomak:

```
x1 = a >> 2;
```

$a = 30$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$x1 = 7$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Lijevi pomak:

```
x2 = b << 3;
```

$b = -5$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$x2 = -40$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Jedinični komplement:

```
x3 = ~b;
```

$b = -5$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$x3 = 4$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Bitovni ILI:

```
x4 = a | b;
```

$a = 30$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$b = -5$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$x4 = -1$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Bitovni I:

```
x5 = a & b;
```

$a = 30$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$b = -5$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$x5 = 26$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Bitovni ekskluzivni ILI:

$$x_6 = a^b;$$

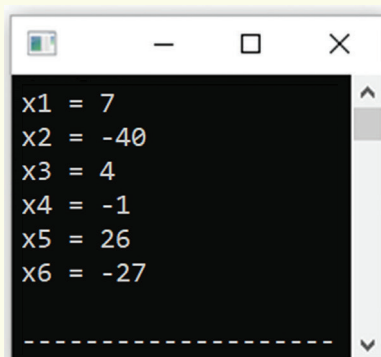
a = 30

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

[illegible]

x6 = -27	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1
----------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Prikaz ispisa na zaslonu računala nalazi se na slici 3.10.



Slika 3.10. Rezultat primjene bitovnih operatora

Proširite program

Omogućite unos vrijednosti za a i b ; najprije izračunajte i odredite što će se ispisati, a zatim dobivene rezultate provjerite pokretanjem programa.



Napomena

Signed tipovi podataka mogu sadržavati pozitivne i negativne vrijednosti, dok unsigned tipovi sadrže samo nenegativne vrijednosti. Detaljnije objašnjenje nalazi se u poglavlju 3.3.

Kod desnog pomaka (>>) unsigned tipovi upotrebljavaju logički pomak pri čemu se novonastale pozicije s lijeve strane popunjavaju nulama (0). Kod signed tipova najčešće se upotrebljava aritmetički pomak gdje se te pozicije popunjavaju bitom predznaka (0 za pozitivne, 1 za negativne). Budući da C standard ne definira ponašanje za signed tipove podataka, rezultat može ovisiti o kompajleru i arhitekturi pa se preporučuje upotrebljavati unsigned tipove za predvidive rezultate.

Kod lijevog pomaka (<<) bez obzira na tip podatka, novonastale pozicije s desne strane uvijek se popunjavaju nulama (0). Ipak, kod signed tipova može doći do prelijevanja ako se promijeni bit predznaka pa se i ovdje preporučuje odabir unsigned tipova.

Primjer 3.31.

Napišite program koji za unesenu IPv4 adresu računala i pripadajuću *subnet* masku ispisuje korisniku kojoj **adresi mreže** pripada unesena adresa računala.

Napomene:

- IP adresa i *subnet* maska sastoje se od 4 dekadskih broja
- bitovnim operatorom „I“ (AND) IP adrese i *subnet* maske dobiva se adresa mreže.

ulazni podatci	izlazni podatci
192 168 65 21 255 255 252 0	Adresa mreže glasi: 192.168.64.0

Rješenje

Dodatno objašnjenje:

primjer bitovnog računanja

IP ADRESA:	192.168.65.21	11000000
SUBNET MASKA:	255.255.252.0	11111111
ADRESA MREŽE:	192.168.64.0	11000000

```
#include <stdio.h>

int main(void) {
    int i1, i2, i3, i4;    //ip adresa
    int s1, s2, s3, s4;    //subnet maska

    printf("Ucitajte IP adresu: ");
    scanf("%d %d %d %d", &i1, &i2, &i3, &i4);

    printf("Ucitajte subnet masku: ");
    scanf("%d %d %d %d", &s1, &s2, &s3, &s4);

    printf("Adresa mreze glasi: %d.%d.%d.%d", i1 & s1, i2 & s2, i3 & s3,
        i4 & s4);

    return 0;
}
```

3.2.8. Skraćeni oblik naredbe pridruživanja

Uz osnovni oblik naredbe pridruživanja koji je prikazan na početku ovog poglavlja (lijevi stupac), pridruživanje je moguće napraviti i na sljedeći skraćeni način kako je prikazano desnim stupcem sljedećeg niza naredbi.

skraćeni oblik naredbe pridruživanja

x = x + a;	x += a;
x = x - a;	x -= a;
x = x * a;	x *= a;
x = x / a;	x /= a;
x = x % a;	x %= a;
x = x >> a;	x >>= a;
x = x << a;	x <<= a;
x = x & a;	x &= a;
x = x a;	x = a;
x = x ^ a;	x ^= a;

Primjer 3.32.

Odredite kako će izgledati ispis na zaslonu računala nakon izvršavanja sljedećeg programa.

```
#include <stdio.h>

int main(void) {
    int a = 1, b = 2;

    a += 2;
    b %= 3;
    a += a + b--;

    printf("a = %d\n", a);
    printf("b = %d\n", b);

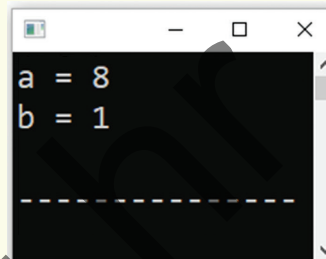
    return 0;
}
```

Rješenje

Prikaz stanja varijabli i prikaz na zaslonu računala nalazi se na slici 3.11.

```
a += 2;           //a = a + 2;
b %= 3;           //b = b % 3;
a += a + b--;     //a = a + a + b--;
```

a	b
1	2
3	
	2
8	1



Slika 3.11. Primjena skraćenog oblika naredbe pridruživanja (ispis na zaslonu računala)

3.2.9. Prioritet operatora

U slučaju složenijih izraza njihovu vrijednost određuje definirani prioritet operatora koji prikazuje tablica 3.10 i to od višeg prioriteta prema nižem prioritetu. U tablici su navedeni i operatori koji još nisu bili spomenuti, a obradit će se u sljedećim poglavljima.

U programskom jeziku C **asocijativnost operatora** određuje redoslijed kojim se operacije izvršavaju kada u izrazu postoji više operatora istog prioriteta. Postoje dvije vrste asocijativnosti: lijeva asocijativnost i desna asocijativnost.

Ako operator ima **lijevu asocijativnost**, operacije se izvršavaju s lijeva na desno. To znači da će se operatori s istim prioritetom obrađivati redom od lijevog prema desnom. Operatori poput +, -, *, / i % imaju lijevu asocijativnost.

```
int x = 8 - 3 + 2; // Ovo se računa kao (8 - 3) + 2
```

Ako operator ima **desnu asocijativnost**, operacije se izvršavaju s **desna na lijevo**. To znači da će se operatori s istim prioritetom obrađivati redom od desnog prema lijevom. Na primjer, operatori poput **pridruživanja** (=, +=, -=, *=, /=, itd.) i **ternarni operator** (?:) imaju **desnu asocijativnost**.

```
int x = 1, y = 2, z = 3;
x = y = z; // Ovo se računa kao x = (y = z), pa x i y postaju 3
```


Tablica 3.10. Prioritet operatora

prioritet operatora		opis	asocijativnost
1.	() [] . -> ++ --	zagrada (poziv funkcije ili promjena prioriteta operacija) indeksiranje polja izbor člana strukture izbor člana strukture putem pokazivača inkrementiraj / dekrementiraj nakon	→
2.	! ~ ++ -- sizeof (cast) * & + -	logička negacija jedinični komplement inkrementiraj / dekrementiraj prije veličina tipa ili varijable eksplicitna promjena tipa dereferenciranje pokazivača adresni operator unarni + i –	←
3.	* / %	množenje / dijeljenje / cjelobrojni ostatak	→
4.	+ -	zbrajanje / oduzimanje	→
5.	<< >>	lijevi pomak / desni pomak	→
6.	< <= > >=	manje / manje od ili jednako / veće / veće od ili jednako	→
7.	== !=	jednako / različito	→
8.	&	bitovni I	→
9.	^	bitovni ekskluzivni ILI	→
10.		bitovni ILI	→
11.	&&	logički I	→
12.		logički ILI	→
13.	? :	ternarni uvjetni operator	←
14.	= *= /= %= += -= &= ^= = <<= >>=	pridruživanje	←
15.	,	operator zarez	→

Napomena: ako je više operatora navedeno pod istim brojem, onda ti operatori imaju isti prioritet i operacije se izvršavaju redom kako su navedene.

3.3. Osnovni tipovi podataka

Osnovni tipovi podataka su:

- cjelobrojni tipovi – char, int, _Bool (bool)
- brojevi s pomičnim zarezom – float, double

Kvalifikatori koji se upotrebljavaju uz osnovne tipove podataka su:

- short / long / long long (smanjuju ili povećavaju raspon cijelih brojeva)
- signed (omogućuje pohranu pozitivnih i negativnih vrijednosti) / unsigned (omogućuje pohranu samo nenegativnih vrijednosti)

Na primjer, mogu se na sljedeći način kombinirano pisati uz oznake osnovnih tipova podataka:

- signed short int
- signed long int
- signed long long int
- signed char
- unsigned short int
- unsigned long int
- unsigned long long int
- unsigned char

Zauzeće u memoriji računala ovisi o implementaciji prevoditelja te o arhitekturi računala. Operatorom sizeof moguće je provjeriti koliko je zauzeće u memoriji. Noviji standardi C99 i C++11 uveli su cjelobrojne tipove fiksnih duljina.

Koliko je zauzeće pojedinog podatka u bajtovima (B), ovisno o tome koji je tip podataka, prikazuje tablica 3.11.

Tablica 3.11. Zauzeće u memoriji u ovisnosti o tipu podatka

tip podatka	veličina (B)
char	1
_bool/bool	1
short	2
int	4
long	4
float	4
double	8

Primjer 3.33.

Testirajte zadani program te odredite koliko je zauzeće u memoriji računala za pojedinu varijablu.

```
#include <stdio.h>

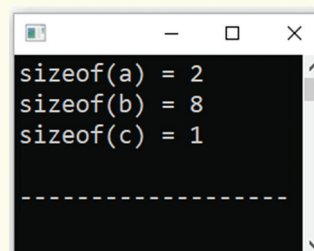
int main(void) {
    signed short int a = 2;
    unsigned long long int b = 3;
    unsigned char c = 'f';

    printf("sizeof(a) = %d\n", sizeof(a));
    printf("sizeof(b) = %d\n", sizeof(b));
    printf("sizeof(c) = %d\n", sizeof(c));

    return 0;
}
```

Rješenje

Prikaz rješenja nalazi se na slici 3.12.



Slika 3.12. Primjena operatora sizeof

3.3.1. Cjelobrojni tipovi podataka

Kao što je prethodno bilo objašnjeno, u programskom jeziku C postoji više cjelobrojnih tipova podataka koji se razlikuju po veličini memorije koju zauzimaju te po rasponu vrijednosti koje mogu pohraniti. Osim samog tipa podataka važno je znati i na koji se način njegova vrijednost prikazuje prilikom ispisa na zaslonu.

Za ispis vrijednosti upotrebljavaju se **specifikatori formata** koji se navode unutar funkcije `printf`. Odabir odgovarajućeg specifikatora ovisi o tipu podataka i željenom obliku prikaza (npr. dekadski, oktalni ili heksadekadski zapis).

U tablici 3.12 nalazi se popis specifikatora formata za cjelobrojne tipove podataka.

Tablica 3.12. Pregled specifikatora formata (cjelobrojni tipovi podataka)

tip podatka	oznaka formata
char	%c %d (za numerički ispis)
signed char	%c %hhi (za numerički ispis)
unsigned char	%c %hhu (za numerički ispis)
short short int signed short signed short int	%hi
unsigned short unsigned short int	%hu
int signed signed int	%i %d – dekadski broj %o – oktalni broj %x – heksadekadski broj
unsigned unsigned int	%u
long long int signed long signed long int	%li
unsigned long unsigned long int	%lu
long long long long int signed long long signed long long int	%lli
unsigned long long unsigned long long int	%llu

U tablici 3.12 navedeni su i odgovarajući formati ako je potrebno ispisati cjelobrojnu konstantu u dekadskom (%d), oktalnom (%o) ili heksadekadskom (%x) obliku. Oktalni oblik predstavlja zapis u brojevnom sustavu s bazom 8.

Primjer 3.34.

Odredite kako će izgledati ispis na zaslonu računala.

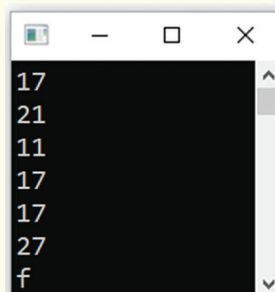
```
#include <stdio.h>

int main(void) {
    printf("%d\n", 17);
    printf("%o\n", 17);
    printf("%x\n", 17);
    printf("%o\n", 017);
    printf("%x\n", 0x17);
    printf("%o\n", 0x17);
    printf("%x\n", 017);

    return 0;
}
```

Rješenje

Prikaz rješenja nalazi se na slici 3.13.



Slika 3.13. Rezultat ispisa zadanih cijelih brojeva u ovisnosti o formatu

Izmijenite program

Zamijenite broj 17 brojem 29 u svim printf funkcijama i provjerite ispis. Potom s pomoću kalkulatora provjerite pretvorbe u oktalni brojevni sustav (pogledajte video u primjeru 3.10).



U prethodnom poglavlju spomenuli smo da se znakovi u računalu ne pohranjuju izravno kao slova, već kao brojevi dobiveni posebnim sustavom kodiranja.

U tablici 3.12 vidjeli smo da se vrijednost tipa char može ispisati kao znak (%c) ili kao broj (%d), što upućuje na to da svaki znak ima svoju brojčanu vrijednost.

Za podatak tipa char kaže se da predstavlja najkraći cijeli broj. Naime, zauzima u memoriji svega 1 bajt i ovaj tip podatka najčešće se upotrebljava kada se u programu upotrebljavaju znakovi. **Svaki znak ima svoju odgovarajuću vrijednost u ASCII tablici** (tablica 3.5). Te vrijednosti mogu se zapisati i kao dekadске, oktalne, heksadekadске ili binarne vrijednosti. Najčešće se prikazuju u heksadekadskom obliku.

Taj način pridruživanja znakova i brojeva uređeno je ASCII tablicom.

ASCII tablica prikazuje vezu između:

- znaka
- njegove dekadске vrijednosti
- heksadekadске vrijednosti

Na temelju te tablice moguće je razumjeti i odrediti kako će se znak prikazati pri različitim načinima ispisa.

Primjer 3.35.

S pomoću ASCII tablice (tablica 3.5) odredite:

- Koja dekadska vrijednost odgovara znaku A?
- Koja dekadska vrijednost odgovara znaku a?
- Koji znak odgovara dekadskoj vrijednosti 66?
- Koji znak odgovara heksadekadskoj vrijednosti $41_{(16)}$?

Rješenje

- a) Znak **A** ima dekadsku vrijednost **65**.
- b) Znak **a** ima dekadsku vrijednost **97**.
- c) Dekadskoj vrijednosti **66** odgovara znak **B**.
- d) Heksadekadskoj vrijednosti **41₍₁₆₎** odgovara dekadaska vrijednost 65, odnosno znak **A**.

Posebno je praktično zapamtiti ASCII vrijednosti prve znamenke, prvog velikog i prvog malog slova engleske abecede u dekadskom zapisu.

0 ima ASCII vrijednost **48**.

A ima ASCII vrijednost **65**.

a ima ASCII vrijednost **97**.

Razlika ASCII vrijednosti odgovarajućeg velikog i malog slova u ASCII tablici iznosi 32.

Primjer 3.36.

Zapišite riječ **PROGRAM** u obliku niza 8-bitovnih registara primjenjujući ASCII tablicu na stranici 69.

Za svaki znak odredite:

- dekadsku vrijednost
- heksadekadsku vrijednost
- binarni zapis (8 bitova)

Dobivene binarne zapise prikazite kao sadržaj registara te ih zatim prikazite kao uzastopne memorijske lokacije.

Rješenje

Za svaki znak iz riječi **PROGRAM** određuje se odgovarajuća ASCII vrijednost te se zapisuje u obliku 8-bitovnog binarnog zapisa. Rješenje prikazuje tablica 3.13.

Tablica 3.13. Prikaz ASCII zapisa znakova riječi **PROGRAM** u 8-bitovnim registrima

znak	dek.	heks.	8-bitovni registar
P	80	50	0 1 0 1 0 0 0 0
R	82	52	0 1 0 1 0 0 1 0
O	79	4F	0 1 0 0 1 1 1 1
G	71	47	0 1 0 0 0 1 1 1
R	82	52	0 1 0 1 0 0 1 0
A	65	41	0 1 0 0 0 0 0 1
M	77	4D	0 1 0 0 1 1 0 1

Sadržaj memorije može se prikazati kao niz uzastopnih 8-bitovnih zapisa:

[01010000][01010010][01001111][01000111][01010010][01000001][01001101]

Primjer 3.37.

Odredite kako će izgledati ispis na zaslonu računala ako korisnik uči znak a.

```
#include <stdio.h>

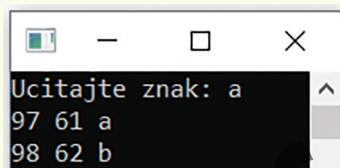
int main(void) {

    char x, y = 'b';
    printf("Ucitajte znak: ");
    scanf("%c", &x);
    printf("%d %x %c\n", x, x, x);
    printf("%d %x %c\n", y, y, y);

    return 0;
}
```

Rješenje

Očitavanjem vrijednosti iz ASCII tablice 3.5 dolazi se do rješenja koje prikazuje slika 3.14.



Slika 3.14. Primjena ASCII tablice prilikom određivanja ispisa na zaslonu računala

Primjer 3.38.

Odredite kako će izgledati ispis na zaslonu računala ako korisnik uči znak a.

```
#include <stdio.h>

int main(void) {

    char x, y = 'B';
    printf("Ucitajte znak: ");
    scanf("%c", &x);
    printf("%d %c\n", x - 32, x - 32);
    printf("%c %c\n", y + '1', y + 1);

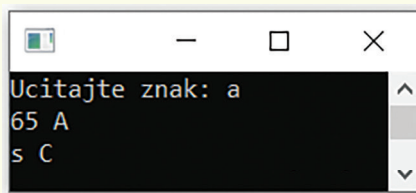
    return 0;
}
```

Rješenje

Rješenje primjera nalazi se na slici 3.15.

Potrebno je uočiti sljedeću razliku:

$y + '1' : 66 + 49 = 115$
 $y + 1 : 66 + 1 = 67$



Slika 3.15. Ispis, primjena ASCII tablice

Primjer 3.39.

Napišite program koji za upisani dekadski broj ispisuje njegovu odgovarajuću heksadekadsku vrijednost prethodnika i sljedbenika.

ulazni podatci	izlazni podatci
15	Prethodnik (heksadekadski): e Sljedbenik (heksadekadski): 10

Rješenje

```
#include <stdio.h>

int main(void) {
    int a;

    printf("Ucitajte prirodan broj: ");
    scanf("%d", &a);

    printf("Prethodnik (heksadekadski): %x\n", a - 1);
    printf("Sljedbenik (heksadekadski): %x\n", a + 1);

    return 0;
}
```

U cjelobrojne tipove podataka ubrajaju se i **podatci tipa `_Bool`**. Varijabla ovog tipa za bilo koju vrijednost različitu od nule poprima vrijednost 1. Uvođenjem novijih standarda moguće je pisati i tip podatka samo kao `bool`, u zamjenu za `_Bool`, ali onda je potrebno navesti i biblioteku `stdbool.h`. U tom je slučaju `true` definiran kao 1, a `false` kao 0. U programskom jeziku C++ također se upotrebljava `bool`.

Primjer 3.40.

Odredite kako će izgledati ispis na zaslonu računala u sljedeća dva primjera.

```
#include <stdio.h>

int main(void) {
    _Bool x, y;
    x = 4;
    y = 0;
    printf("%d\n", x);
    printf("%d", y);

    return 0;
}
```

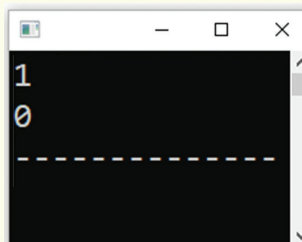
```
#include <stdio.h>
#include <stdbool.h>

int main(void) {
    bool x, y;
    x = 4;
    y = 0;
    printf("%d\n", x);
    printf("%d", y);

    return 0;
}
```

Rješenje

Ispis će biti isti i prikazuje ga slika 3.16.



Slika 3.16. Ispis vrijednosti varijabli logičkog tipa

3.3.2. Brojevi s pomičnim zarezom

U tablici 3.15. nalazi se popis oznaka formata za podatke koje predstavljaju brojevi s pomičnim zarezom (realni brojevi).

Tablica 3.15. Pregled oznaka formata (brojevi s pomičnim zarezom)

tip podatka	oznaka formata
float	decimalni zapis %f %F
	znanstveni zapis %g %G, %e %E, %a %A
double	%lf %lF %lg %lG
	%le %lE %la %lA
long double	%Lf %LF %Lg %LG
	%Le %LE %La %LA

Podatci tipa `float` zauzimaju 4 B (veličina registra je 32 bita). To su brojevi zapisani prema **standardu IEEE 754 jednostruke preciznosti**. U zapisu broja nalazi se oznaka `f` ili `F`. Podatci tipa `double` zauzimaju 8 B (veličina registra je 64 bita). To su brojevi zapisani prema **standardu IEEE 754 dvostruke preciznosti**. Razlika je u preciznosti i točnosti pohrane broja.

U zapisu brojeva tipa `float` i `double` često se upotrebljava tzv. **znanstveni zapis** (eksponencijalni zapis).

Zapis oblika: $9.56e-07$ znači $9.56 \cdot 10^{-7}$.

Slovo **e** označava potenciju broja 10, odnosno pomak decimalne točke:

- `e-7` znači da se decimalna točka pomiče 7 mjesta ulijevo
- `e+7` znači da se decimalna točka pomiče 7 mjesta udesno

Tako je na primjer $9.56e-7 = 0.000000956$.

Primjer 3.41.

Odredite kako će izgledati ispis na zaslonu računala.

```
#include <stdio.h>

int main(void) {
    float x, y;

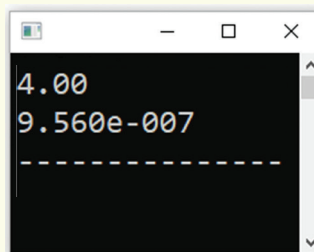
    x = 4.f;           //x = 4.F
    y = 9.56e-7f;      //y = 9.56E-7F

    printf("%.2f\n", x);
    printf("%.3e", y);

    return 0;
}
```

Rješenje

Prikaz rješenja nalazi se na slici 3.17.



Slika 3.17. Ispis podataka tipa `float`

Primjer 3.42.

Odredite kako će izgledati ispis na zaslonu računala.

```
#include <stdio.h>

int main(void) {

    double x, y;

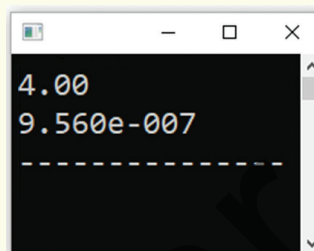
    x = 4.;
    y = 9.56e-7;    //y = 9.56E-7

    printf("%.21f\n", x);
    printf("%.31e", y);

    return 0;
}
```

Rješenje

Prikaz rješenja nalazi se na slici 3.18.



Slika 3.18. Ispis podataka tipa double

3.4. Konverzija tipova podataka

Konverzija tipova podataka u programiranju odnosi se na proces pretvaranja jedne vrste podatka (tipa) u drugu kako bi se osigurala kompatibilnost ili prilagodba operacijama. U jeziku C konverzija tipova može biti **implicitna** ili **eksplicitna**.

3.4.1. Implicitna konverzija

Implicitnu konverziju obavlja kompajler automatski kada je potrebno prilagoditi tipove podataka u izrazu. Ako izraz sadržava dva operanda različitih tipova, operand manje važnosti automatski se konvertira u tip operanda veće važnosti. Ovo pravilo slijedi sljedeći redoslijed važnosti tipova u jeziku C:

long double > double > float > long > int > short > char

Primjer:

Ako je prvi operand tipa int i drugi operand tipa float, operand tipa int bit će automatski konvertiran u float jer je float veće važnosti.

```
int a = 10;
float b = 5.5;
float rezultat = a + b; //a se automatski konvertira u float
```

U ovom primjeru kompajler automatski konvertira a iz int u float, pa rezultat izraza zadržava veću preciznost.

3.4.2. Eksplicitna konverzija

Eksplicitna konverzija izvodi se s pomoću **operatora za kasanje** (engl. *type casting*). Programer jasno specificira koju vrstu podatka treba upotrijebiti te se u tom slučaju implicitna konverzija poništava.

Sintaksa eksplicitne konverzije:

```
(tip_podatka) izraz;
```

Primjer:

```
int a = 10, b = 3;  
float rezultat = (float)a / b; // Eksplicitna konverzija a iz int u float
```

Ovdje se a eksplicitno konvertira u float kako bi se osiguralo da dijeljenje daje realni broj umjesto cijelog broja.

Primjer 4.43.

Kako će izgledati ispis na zaslonu računala?

```
#include <stdio.h>  
  
int main(void) {  
    int x, a;  
    float y, b;  
  
    x = 5. / 2;  
    y = (float)5 / 2;  
    a = 5 / 2;  
    b = 5 / 2;  
  
    printf("x = %d\n", x);  
    printf("y = %.2f\n", y);  
    printf("a = %d\n", a);  
    printf("b = %.2f\n", b);  
    return 0;  
}
```

Rješenje

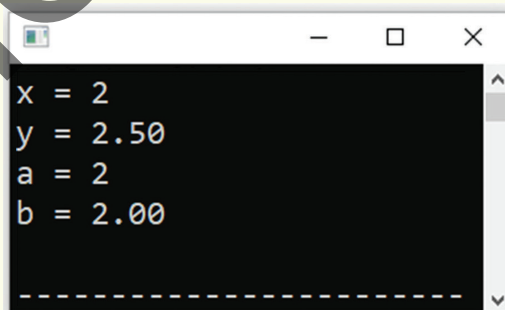
Implicitna promjena tipa operanda:

```
x = 5. / 2;
```

Eksplicitna promjena tipa operanda:

```
y = (float)5 / 2;
```

Rješenje primjera nalazi se na slici 3.19.



Slika 3.19. Primjena implicitne i eksplicitne promjene tipa operanda – ispis

Primjer 3.44.

Kako će izgledati ispis na zaslonu računala?

```
#include <stdio.h>

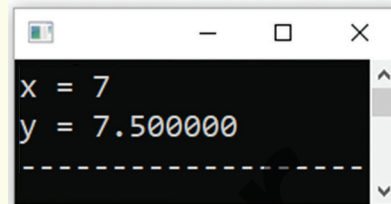
int main(void) {
    int x, a = 5;
    float y, b = 2.5;

    x = a + b;
    y = a + b;

    printf("x = %d\n y = %f", x, y);
    return 0;
}
```

Rješenje

Prikaz rješenja nalazi se na slici 3.20.



Slika 3.20. Zbrajanje podataka različitog tipa (obavlja se konverzija)

Primjer 3.45.

Odredite stanje svih varijabli i ispis na zaslonu računala nakon što se izvrši zadani program.

Prvo samostalno izračunajte, a potom rješenje provjerite s pomoću poveznice: <http://pythontutor.com/c.html#mode=edit>.

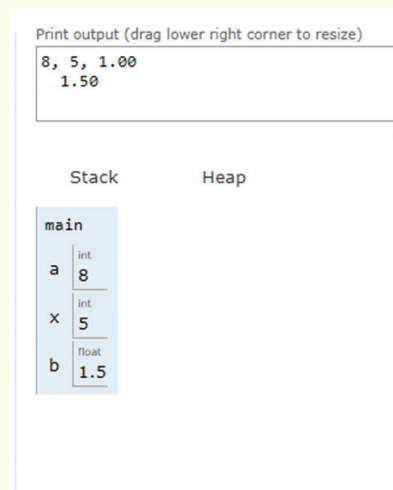
```
#include <stdio.h>

int main(void) {
    int a = 11, x;
    float b;
    x = a + 10;
    x = x / 4;
    a = x + 3;
    b = (x + 7) / a;
    printf("%d, %d, %.2f\n", a, x, b);
    b = (x + 7) / (float)a;
    printf("%6.2f", b);

    return 0;
}
```

Rješenje

Upisivanjem koda u sučelje, koje se otvara klikom na prethodnu poveznicu, rješenje će biti kao što prikazuje slika 3.21.



Slika 3.21. Konačno stanje varijabli i ispis na zaslonu računala za primjer 3.45.

Primjer 3.46.

Napišite program koji određuje vrijednost zbroja dvaju razlomaka. Posebno učitajte brojnik i nazivnik prvog razlomka, a potom brojnik i nazivnik drugog razlomka. Rezultat ispišite u obliku neskrćenog razlomka i u obliku decimalnog broja. Napomena: vrijednosti brojnika i nazivnika prirodni su brojevi.

ulazni podatci	izlazni podatci
2 3 3 2	$2/3 + 3/2 = 13/6 = 2.17$

Rješenje

Modeliranje problema

$$\frac{b_1}{n_1} + \frac{b_2}{n_2} = \frac{b_1 n_2 + b_2 n_1}{n_1 n_2} = \frac{b}{n}$$

```
#include <stdio.h>

int main(void) {
    int b1, n1, b2, n2, b, n; //deklaracija varijabli
    float rez;

    printf("Ucitajte brojnik i nazivnik prvog razlomka: ");
    scanf("%d %d", &b1, &n1);

    printf("Ucitajte brojnik i nazivnik drugog razlomka: ");
    scanf("%d %d", &b2, &n2);

    b = b1 * n2 + b2 * n1; //izračun brojnika
    n = n1 * n2;           //izračun nazivnika
    rez = (float)b / n;     /*zapis u obliku realnog broja s pomoću
                           operatora za kastanje */

    printf("%d/%d+%d/%d=%d/%d=%.2f", b1, n1, b2, n2, b, n, rez);

    return 0;
}
```

Izmijenite program

Prilagodite program tako da izračuna rezultat dijeljenja dvaju razlomaka i ispiše ga u istom obliku kao u primjeru.



element.hr



VRIJEME ZA IZAZOV

Ocjene i postotci

Radna situacija

Nakon ispita razrednik želi brzo izračunati **postotak uspješnosti** svakog učenika kako bi znao koliko je tko bio blizu maksimalnog broja bodova.

Tvoj je zadatak izraditi program koji računa **postotak uspjeha** i **razliku do maksimalnog broja bodova**.



Zadatak

1. Napiši program koji omogućuje unos:
 - **ukupnog broja bodova** mogućih na ispitu
 - **ostvarenog broja bodova** učenika.
2. Program treba izračunati i ispisati:
 - **postotak uspješnosti** učenika
 - **razliku između maksimalnog i ostvarenog broja bodova**.
3. Rezultate ispiši pregledno na **zaslon računala**.
4. Dodaj komentare koji objašnjavaju glavne naredbe.
5. Napiši i **pseudokod** koji opisuje rješenje.

Napomena

- Pretpostavlja se **ispravan unos podataka**.
- **Bodovi su cijeli brojevi**.

Primjer ulaznih i izlaznih podataka

ulazni podatci	izlazni podatci
Ukupno bodova: 50 Ostvareno bodova: 42	Postotak uspješnosti: 84.00 % Razlika do maksimuma: 8 bodova
Ukupno bodova: 40 Ostvareno bodova: 29	Postotak uspješnosti: 72.50 % Razlika do maksimuma: 11 bodova

Elementi vrednovanja – program i pseudokod

elementi provjere	bodovi
unos ukupnog i ostvarenog broja bodova	4
izračun postotka uspješnosti (s pretvorbom tipa podataka)	7
izračun razlike između maksimalnog i ostvarenog broja bodova	5
jasno strukturiran i pregledan ispis rezultata	5
dodani komentari u programu koji objašnjavaju operacije	4
struktura pseudokoda jasno slijedi logiku algoritma	4
usklađenost pseudokoda s programom u jeziku C	1
ukupno	30

Ocjena

- 27 – 30 = odličan (5)
- 23 – 26 = vrlo dobar (4)
- 19 – 22 = dobar (3)
- 15 – 18 = dovoljan (2)
- 0 – 14 = nedovoljan (1)

Dodatni zadatak za one koji žele znati više

U proširenju programa dodaj mogućnost unosa **ukupnog broja zadataka** na ispi-
tu. Na temelju tog podatka program treba **izračunati koliko bodova donosi jedan**
zadatak ako svaki zadatak nosi jednak broj bodova. Ukupan broj bodova, ostvareni
bodovi i broj zadataka unose se kao **cijeli brojevi**, a izračun broja bodova po zadatku
prikazuje se **s dvjema decimalama** radi točnijeg rezultata.

Vrednovanje kao učenje



<https://s.element.hr/d3e7>

4.

GRANANJE U PROGRAMU



DIGITALNI RUDNIK KODOVA

Skeniraj QR kôd ili otvori poveznicu u internetskom pregledniku. Prouči primjere 4.3, 4.4, 4.5, 4.8 i 4.13 te ih analiziraj i proširi programe prateći upute iz tih primjera.

<https://s.element.hr/e5eb>





Razmisli

Na samonaplatnoj blagajni kupac skenira pakirani proizvod i stavlja ga na vagu. Zatim slijedi provjera odgovara li proizvod onome koji je skeniran. Kako program može prepoznati je li na vagu stavljen ispravan proizvod?



Nakon ove lekcije moći će



- objasniti što je grananje u programu
- razlikovati jednostranu, dvostranu i višestruku selekciju
- upotrebljavati naredbe `if`, `if else` i `if else if`
- upotrebljavati naredbu `switch case` za višestruki odabir
- napisati uvjet s pomoću relacijskih i logičkih operatora
- pratiti izvođenje programa s grananjem i odrediti rezultat
- prepoznati kada je prikladno upotrebljavati pojedinu vrstu grananja
- nacrtati dijagram toka za program s grananjem
- napisati jednostavan program koji donosi odluke na temelju uvjeta.



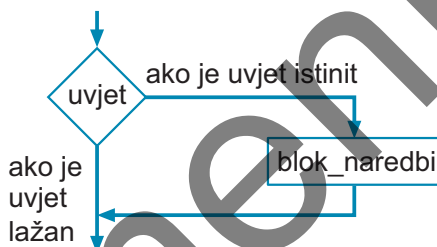
Smjer u kojem se program izvodi definira se upotrebom naredbi poput `if`, `if else` i `switch case`. Grananje, odnosno selekcija omogućuje u programima donošenje odluka na temelju različitih uvjeta, čime se prilagođava njihov tijek izvršavanja. U nastavku je prikazano kako se grananje implementira u programskom jeziku C, pri čemu logički i relacijski operatori imaju ključnu ulogu u definiranju uvjeta grananja. U nastavku su prikazane jednostrana, dvostrana i višestruka selekcija (grananje) te njihova implementacija u programskom jeziku C.

4.1. Jednostrana selekcija `if`

Opći oblik **naredbe `if`** glasi:

```
if (uvjet) {
    ...
    blok_naredbi;
    ...
}
```

Slika 4.1 prikazuje dijagram toka za jednostranu selekciju napisanu s pomoću naredbe `if`.



Slika 4.1. Dijagram toka za naredbu `if`

Ako je uvjet istinit, izvršava se blok naredbi. U suprotnom se blok naredbi u vitičastim zagradama preskače. U slučaju da umjesto bloka naredbi postoji samo jedna naredba, vitičaste se zagrade mogu izostaviti.

Primjer 4.1.

Odredite što će se ispisati na zaslonu računala.

```
#include <stdio.h>

int main(void) {
    int x = 5, y = 2;

    if (x >= 2)
        y = y + x;

    printf("%d\n", y);

    if (y != 7)
        y = y - 3;

    printf("%d", y);

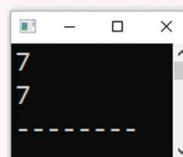
    return 0;
}
```

Rješenje

Prvi uvjet `if (x >= 2)` jest istinit (jer je `5 >= 2` što daje logičku jedinicu) te se naredba `y = y + x;` izvršava.

Drugi `if (y != 7)` nije istinit (`7 != 7` što daje logičku nulu) pa se naredba `y = y - 3;` preskače.

Rješenje prikazuje slika 4.2:



Slika 4.2. Jednostrana selekcija – ispis

Primjer 4.2.

Odredite što će se ispisati na zaslonu računala.

```
#include <stdio.h>

int main(void) {

    int x = 5, y = 2;

    if (x >= 2) {
        y = y + x;
        printf("%d\n", y);
    }

    if (y != 7)
        y = y - 3;
    printf("%d", y);

    return 0;
}
```

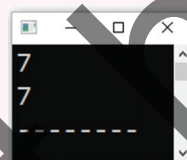
Izmijenite program

Inicijalizirajte varijable x i y tako da nijedan uvjet ne bude ispunjen.

**Rješenje**

Prvi uvjet `if (x >= 2)` jest istinit te se izvršava cijeli blok naredbi u vitičastim zagradama, dok drugi `if (y != 7)` nije pa se naredba `y = y - 3;` preskače. Prilikom čitanja koda treba paziti koje su naredbe uključene u vitičaste zagrade. Naredbu ili naredbe koje su u bloku naredbi treba uvući kako je prikazano u kodu.

Rješenje prikazuje slika 4.3.



Slika 4.3. Jednostrana selekcija (blok naredbi) – ispis

Primjer 4.3.

Odredite što će se ispisati na zaslonu računala.

```
#include <stdio.h>

int main(void) {

    int x = 5, y = 2;

    if (x >= 2) {
        y = y + x;
        printf("%d\n", y);
    }

    if (y != 7) {
        y = y - 3;
        printf("%d", y);
    }

    return 0;
}
```

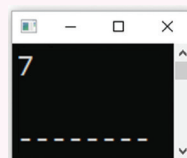
Proširite program

Omogućite unos vrijednosti za x i y, zatim zamijenite redoslijed `if` naredbi i njihovih blokova naredbi te provjerite kako to utječe na rezultat.

**Rješenje**

Prvi uvjet `if (x >= 2)` jest istinit te se izvršava cijeli blok naredbi u vitičastim zagradama, dok drugi `if (y != 7)` nije pa se preskače cijeli drugi blok naredbi.

Rješenje prikazuje slika 4.4.



Slika 4.4. Jednostrana selekcija (dva bloka naredbi) – ispis

Primjer 4.4.

Napišite program koji provjerava ima li korisnik pravo na popust.

Ako je iznos kupnje veći ili jednak 50 €, ispišite poruku "Ostvarili ste popust."

Ako je iznos manji od 50 €, ispišite poruku "Niste ostvarili popust."

Napomena: iznos je realan broj.

ulazni podatci	izlazni podatci
30	Niste ostvarili popust.
75	Ostvarili ste popust.

Rješenje

Program učitava iznos kupnje te s pomoću dviju if naredbi provjerava ostvaruje li korisnik popust i ispisuje odgovarajuću poruku.

```
#include <stdio.h>

int main(void) {
    float iznos;    // deklaracija varijable

    printf("Ucitaj iznos kupnje: ");
    scanf("%f", &iznos);

    if (iznos >= 50)
        printf("Ostvarili ste popust.\n");

    if (iznos < 50)
        printf("Niste ostvarili popust.\n");

    return 0;
}
```

Na primjer, ako korisnik u program učitava vrijednost 30, ispis u terminalu bit će kao na slici 4.5.

```
Ucitaj iznos kupnje: 30
Niste ostvarili popust.

Process returned 0 (0x0)   execution time : 1.291 s
Press any key to continue.
```

Slika 4.5. Prikaz izvođenja programa i ispisa poruke kada je iznos kupnje manji od 50 €

Proširite program

Omogućite i unos iznosa popusta u postotcima. Program potom treba izračunati i ispisati koliko kupac treba platiti te koliko je novaca uštedio ako je ostvario pravo na popust.

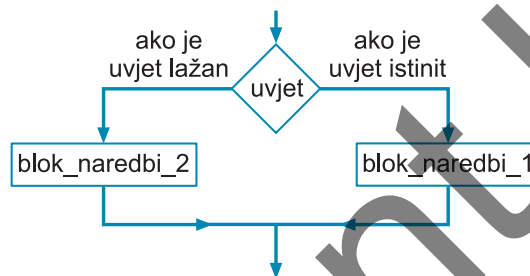


4.2. Dvostrana selekcija if else

Opći oblik naredbe **if else** glasi:

```
if (uvjet) {
    ...
    blok_naredbi_1;
    ...
} else {
    ...
    blok_naredbi_2;
    ...
}
```

Slika 4.6 prikazuje dijagram toka za dvostranu selekciju napisanu s pomoću naredbe **if else**.



Slika 4.6. Dijagram toka za naredbu **if else**

Ako je uvjet naveden u zagradi iza ključne riječi **if** istinit, izvršava se prvi blok naredbi. U suprotnom, ako uvjet nije istinit, blok naredbi u prvim vitičastim zagradama preskače se te se izvršava drugi blok naredbi. U slučaju da umjesto bloka naredbi postoji samo jedna naredba, vitičaste se zagrade mogu izostaviti.

Primjer 4.5.

Odredite što će se ispisati na zaslону računala.

```
#include <stdio.h>

int main(void) {
    int x = 5, y = 2;

    if (x >= 2 && x == y)
        y = y + x;
    else
        y = y - 3;

    printf("%d", y);

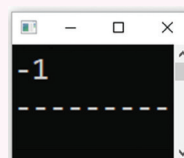
    return 0;
}
```

Rješenje

```
x >= 2 && x == y
5 >= 2 && 5 == 2
1 && 0
0
```

Uvjet nije istinit (logička nula) te se prva naredba $y = y + x$; preskače, a naredba iza **else** $y = y - 3$; izvršava.

Rješenje prikazuje slika 4.7.



Slika 4.7. Dvostruka selekcija – ispis

Proširite program

Omogućite unos vrijednosti za x i y , negirajte uvjet u **if** naredbi i usporedite dobiveni rezultat s početnim.



Primjer 4.6.

Odredite što će se ispisati na zaslonu računala.

```
#include <stdio.h>

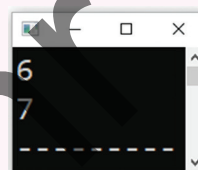
int main(void) {
    int x = 5, y = 2;
    if (x >= 2 || x == y) {
        y = y + x;
        x = x + 1;
        printf("%d\n", x);
    }
    else {
        y = y - 3;
        x--;
        printf("%d\n", x);
    }
    printf("%d", y);
    return 0;
}
```

Rješenje

```
x >= 2 || x == y
5 >= 2 || 5 == 2
1      ||      0
1
```

Uvjet je istinit te se izvršava prvi blok naredbi, dok se drugi blok naredbi preskače. Unutar prvog bloka y poprima vrijednost 7, a x vrijednost 6. Prvo se unutar prvog bloka ispisuje vrijednost varijable x. Funkcija kojom se ispisuje vrijednost varijable y nalazi se izvan naredbe if else i neovisno o uvjetu, uvijek se izvodi.

Rješenje prikazuje slika 4.8.



Slika 4.8. Dvostrana selekcija – ispis, blok naredbi

Izmijenite program

Postavite početne vrijednosti varijabli x i y tako da se izvrši blok naredbi else te odredite što će program ispisati.



Primjer 4.7.

Primjer 4.4 riješite primjenom naredbe if-else.

Rješenje

U primjeru 4.4 nalaze se dvije if naredbe pa se uvjet morao provjeravati dvaput.

Primjenom naredbe if-else uvjet se provjerava samo jednom, a program automatski odabire jednu od dviju mogućih grana izvođenja.

Takav je zapis pregledniji, kraći i smanjuje mogućnost pogreške pri pisanju programa.

```
#include <stdio.h>

int main(void) {
    float iznos; // deklaracija varijable
    printf("Ucitaj iznos kupnje: ");
    scanf("%f", &iznos);

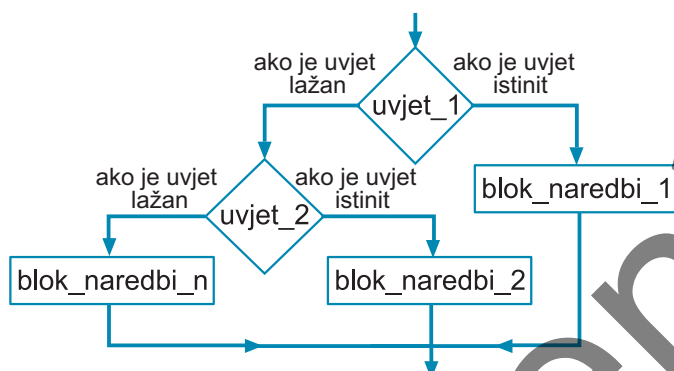
    if (iznos >= 50)
        printf("Ostvarili ste popust.\n");
    else
        printf("Niste ostvarili popust.\n");

    return 0;
}
```

4.3. Višestruka selekcija if else if

Višestrukom selekcijom moguće je postaviti više uvjeta te se tako program može granati u više smjerova. Opći oblik višestruke (kaskadne selekcije) glasi:

Slika 4.9 prikazuje dijagram toka za višestruku selekciju napisanu s pomoću naredbi if else.



Slika 4.9. Dijagram toka za višestruko grananje

```

if (uvjet_1) {
    ...
    blok_naredbi_1;
    ...
} else if (uvjet_2) {
    ...
    blok_naredbi_2;
    ...
} else {
    ...
    blok_naredbi_n;
    ...
}
  
```

Ako prvi uvjet daje kao rezultat logičku jedinicu, izvršava se prvi blok naredbi, a ostali se blokovi naredbi preskaču. Ako prvi uvjet nije zadovoljen, odnosno daje logičku nulu, provjerava se redom sljedeći uvjet.

Ako drugi uvjet daje logičku jedinicu, izvršava se samo drugi blok naredbi, a ostali se blokovi naredbi preskaču.

Ako ni prvi ni drugi uvjet nisu kao rezultat dali logičku jedinicu, izvršava se zadnji blok naredbi iza ključne riječi else. U slučaju da umjesto bloka naredbi postoji samo jedna naredba, vitičaste zagrade mogu se izostaviti.

Primjer 4.3.

Odredite što će se ispisati na zaslonu računala.

```

#include <stdio.h>

int main(void) {

    int x = 5, y = 2;

    if (x >= 2 && x == y) {
        y = y + x;
    }
    else if (x + 1 > 0) {
        y = y - 3;
    }
}
  
```

```

else {
    y = y + x + 1;
}

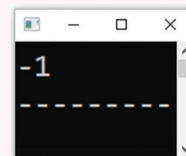
printf("%d", y);

return 0;
}
  
```

Rješenje

U prikazanom kodu prvi uvjet kao rezultat daje logičku nulu te se naredba $y = y + x$; preskače. S obzirom na to da drugi uvjet kao rezultat daje logičku jedinicu, izvršava se naredba pridruživanja $y = y - 3$; te se kod dalje nastavlja ispisujući vrijednosti varijable y (else blok se preskače).

Rješenje prikazuje slika 4.10.



Slika 4.10. Višestruka selekcija – ispis



Proširite program Omogućite unos vrijednosti za x i y , zatim odaberite primjere ulaza tako da se pri različitim pokretanjima programa izvrši svaki od triju blokova naredbi i provjerite rezultate.

Primjer 4.9.

Napišite program u koji se unose bodovi koje je učenik ostvario na ispitu znanja iz programiranja te prema sljedećoj tablici ispišite koja bi to bila ocjena:

ulazni podatci	izlazni podatci
-2	pogresan unos
103	pogresan unos
54	dovoljan

bodovi	ocjena
0 – 50	nedovoljan
51 – 63	dovoljan
64 – 76	dobar
77 – 89	vrlo dobar
90 – 100	odličan

Rješenje

1. način:

```
#include <stdio.h>

int main(void) {
    int x;
    printf("Ucitajte broj bodova: ");
    scanf("%d", &x);

    if (x >= 90 && x <= 100) {
        printf("odlican");
    }
    else if (x >= 77 && x <= 89) {
        printf("vrlo dobar");
    }
    else if (x >= 64 && x <= 76) {
        printf("dobar");
    }
    else if (x >= 51 && x <= 63) {
        printf("dovoljan");
    }
    else if (x >= 0 && x <= 50) {
        printf("nedovoljan");
    }
    else {
        printf("pogresan unos");
    }

    return 0;
}
```

2. način:

```
#include <stdio.h>

int main(void) {
    int x;
    printf("Ucitajte broj bodova: ");
    scanf("%d", &x);

    if (x < 0 || x > 100) {
        printf("pogresan unos");
    }
    else if (x >= 90) {
        printf("odlican");
    }
    else if (x >= 77) {
        printf("vrlo dobar");
    }
    else if (x >= 64) {
        printf("dobar");
    }
    else if (x >= 51) {
        printf("dovoljan");
    }
    else {
        printf("nedovoljan");
    }

    return 0;
}
```

Izmijenite program

Promijenite raspodjelu bodova po ocjenama (skalu) tako da se ocjene dodjeljuju prema novim pravilima.



Primjer 4.10.

Napišite program koji će za učitane vrijednost cjelobrojnog argumenta funkcije $f(x)$ ispisati njezinu vrijednost.

Funkcija je zadana kao: $f(x) = |x - 2| + \frac{1}{x}$.

Zadatak riješite na tri načina:

- a) bez upotrebe gotovih funkcija
- b) upotrebom funkcije `abs`
- c) upotrebom poznatih funkcija iz biblioteke `math.h`, ali bez upotrebe funkcije `abs`.

ulazni podatci	izlazni podatci
-2	$f(-2) = 3.50$
0	Funkcija nije definirana za $x = 0$

Rješenje

```
a) #include <stdio.h>

int main(void) {
    int x;
    float y;
    printf("Ucitajte cijeli broj: ");
    scanf("%d", &x);

    if (x == 0) {
        printf("Funkcija nije definirana za x = 0\n");
    }
    else if (x - 2 >= 0) {
        printf("f(%d) = %.2f", x, x - 2 + 1 / (float)x);
    }
    else {
        printf("f(%d) = %.2f", x, (x - 2) * (-1) + 1 / (float)x);
    }

    return 0;
}
```

```
b) #include <stdio.h>
#include <stdlib.h>

int main(void) {
    int x;
    float y;
    printf("Ucitajte cijeli broj: ");
    scanf("%d", &x);

    if(x == 0) {
        printf("Funkcija nije definirana za x = 0\n");
    } else {
        y = abs(x - 2) + 1. / x;
        printf("f(%d) = %.2f", x, y);
    }

    return 0;
}
```

c)

```
#include <stdio.h>
#include <math.h>

int main(void) {
    int x;
    float y;
    printf("Ucitajte cijeli broj: ");
    scanf("%d", &x);

    if (x == 0) {
        printf("Funkcija nije definirana za x = 0\n");
    }
    else {
        y = sqrt(pow((x - 2), 2)) + 1. / x;
        printf("f(%d) = %.2f", x, y);
    }

    return 0;
}
```

Izmijenite program

Uklonite provjeru dijeljenja s nulom ($x = 0$) te pokrenite program i uočite što se događa kad učitajte 0.

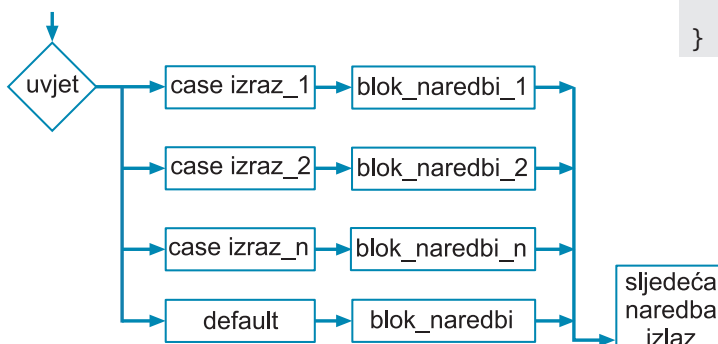


4.4. Višestruka selekcija switch case

Višestruko grananje moguće je realizirati u programskom jeziku C i s pomoću naredbe `switch case`, koja se često naziva i skretnica. Opći oblik zapisuje se na sljedeći način:

```
switch (uvjet) {
    case izraz_1:
        ...
        blok_naredbi_1;
        ...
        break;
    case izraz_2:
        ...
        blok_naredbi_2;
        ...
        break;
    case izraz_n:
        ...
        blok_naredbi_n;
        ...
        break;
    default:
        ...
        blok_naredbi;
        ...
}
```

Slika 4.11 prikazuje dijagram toka za višestruku selekciju napisanu naredbom `switch case`.



Slika 4.11. Dijagram toka za naredbu `switch case`

Uvjet obavezno mora biti **cjelobrojni izraz**. Ako neki od navedenih izraza iza ključne riječi case odgovara postavljenom uvjetu, izvodi se blok naredbi naveden u tom case bloku. Naredbom break preskaču se svi ostali case blokovi te se izlazi iz naredbe switch case. Ako se ne navede naredba break u case bloku, onda će se dalje po redu izvršavati case blokovi sve dok u kodu ne bude naveden break. Default se navodi za slučaj ako niti jedan od prethodno navedenih izraza ne odgovara postavljenom uvjetu. U tom slučaju izvodi se taj zadnji blok naredbi naveden nakon ključne riječi default.

Primjer 4.11.

Odredite što će se ispisati na zaslonu računala. Dodatno, zadani kôd napišite upotrebom naredbe if else.

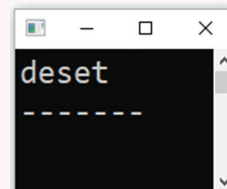
```
#include <stdio.h>
int main(void) {

    int x = 10;

    switch (x) {
        case 5: printf("pet"); break;
        case 7: printf("sedam"); break;
        case 10: printf("deset"); break;
        case 20: printf("dvadeset"); break;
        default: printf("Nema te vrijednosti.");
    }
    return 0;
}
```

Rješenje

Rješenje prikazuje slika 4.12.



Slika 4.12. Skretnica – ispis

Zapis naredbom if else:

```
#include <stdio.h>
int main(void) {

    int x = 10;

    if (x == 5)
        printf("pet");
    else if (x == 7)
        printf("sedam");
    else if (x == 10)
        printf("deset");
    else if (x == 20)
        printf("dvadeset");
    else
        printf("Nema te vrijednosti.");
    return 0;
}
```

Izmijenite program

Izbrišite break u case 10, ponovno pokrenite program i uočite što se događa. Kako bi u tom slučaju napisali zapis s pomoću if else naredbe?



Primjer 4.12.

Odredite što će se ispisati na zaslonu računala.

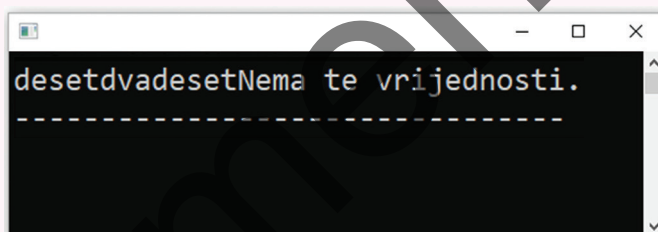
```
#include <stdio.h>
int main(void) {

    int x = 10;

    switch (x) {
        case 5: printf("pet");
        case 7: printf("sedam");
        case 10: printf("deset");
        case 20: printf("dvadeset");
        default: printf("Nema te vrijednosti.");
    }
    return 0;
}
```

Rješenje

Rješenje prikazuje slika 4.13. Nakon izvršenja naredbe unutar case 10, program nastavlja s izvršavanjem naredbi iz case 20 i default jer iza naredbe printf("deset"); nije navedena naredba break.



Slika 4.13. Skretnica, bez naredbe break – ispis

Izmijenite program

Napišite break u case 10, ponovno pokrenite program i uočite što se događa.

**Primjer 4.13.**

Odredite što će se ispisati na zaslonu računala. Dodatno, zadani kôd napišite upotrebom naredbe if else.

```
#include <stdio.h>
int main(void) {

    int x = 7;

    switch (x) {
        case 5:
        case 7:
        case 10: printf("deset"); break;
        case 20: printf("dvadeset"); break;
        default: printf("Nema te vrijednosti.");
    }
    return 0;
}
```

Rješenje

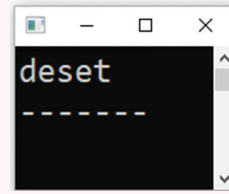
Rješenje prikazuje slika 4.14.

Zapis naredbom if else:

```
#include <stdio.h>
int main(void) {

    int x = 10;

    if (x == 5 || x == 7 || x == 10)
        printf("deset");
    else if (x == 20)
        printf("dvadeset");
    else
        printf("Nema te vrijednosti.");
    return 0;
}
```



Slika 4.14. Skretnica – ispis

Proširite program

Omogućite unos vrijednosti za x, zatim ispitajte različite vrijednosti tako da se izvrši svaki case i default te provjerite rezultate.

**Primjer 4.14.**

Napišite program koji će na zaslonu računala korisniku ponuditi sljedeći izbornik:

- 1 – opseg kvadrata
- 2 – površina kvadrata.

Korisnik u program učitava jedan od brojeva iz izbornika te unosi duljinu stranice kvadrata. Na osnovi odabira program mu ispisuje željeni izračun. Sve rezultate potrebno je ispisati na tri decimalna mjesta. Ako korisnik odabere broj koji nije u izborniku, ispiše se poruka „Pogresan unos“ i završi se s izvođenjem programa.

ulazni podatci	izlazni podatci
2 3	Povrsina je: 9.000
6 4	Pogresan unos.

Rješenje

Rješenje naredbom if else:

```
#include <stdio.h>
#include <math.h>

int main(void) {

    float a;
    float o, p;
    int x;
```

```
printf("1-opseg kvadrata\n2-povrsina kvadrata\n");

printf("Ucitajte broj iz izbornika: ");
scanf("%d", &x);

printf("Ucitajte duljinu stranice kvadrata: \n");
scanf("%f", &a);

if (x == 1) {
    o = 4 * a;
    printf("Opseg je: %.3f", o);
}
else if (x == 2) {
    p = pow(a, 2);
    printf("Povrsina je: %.3f", p);
}
else {
    printf("Pogresan unos.\n");
}

return 0;
}
```

Rješenje naredbom switch case:

```
#include <stdio.h>
#include <math.h>

int main(void) {

    float a;
    float o, p;
    int x;

    printf("1-opseg kvadrata\n2-povrsina kvadrata\n");

    printf("Ucitajte broj iz izbornika: ");
    scanf("%d", &x);

    printf("Ucitajte duljinu stranice kvadrata: \n");
    scanf("%f", &a);

    switch(x) {
        case 1:
            o = 4 * a;
            printf("Opseg je: %.3f", o);
            break;
        case 2:
            p = pow(a, 2);
            printf("Povrsina je: %.3f", p);
            break;
        default: printf("Pogresan unos.\n");
    }

    return 0;
}
```

Izmijenite program

Prilagodite program tako da se umjesto stranice kvadrata unosi duljina njegove dijagonale.



Primjer 4.15.

Napišite program u koji će se unositi četveroznamenasti prirodni broj. Znamenku tisućice učitanoj broja ispišite riječima. Ako korisnik pogrešno utipka vrijednost broja, prekinite s izvođenjem programa.

ulazni podatci	izlazni podatci
3984	tri
34	Pogresan unos.

Rješenje

```
#include <stdio.h>

int main(void) {
    int x, t;
    printf("Ucitajte prirodan broj: ");
    scanf("%d", &x);
    t = x / 1000;
    if (x >= 1000 && x <= 9999) {
        switch (t) {
            case 1: printf("jedan"); break;
            case 2: printf("dva"); break;
            case 3: printf("tri"); break;
            case 4: printf("cetiri"); break;
            case 5: printf("pet"); break;
            case 6: printf("sest"); break;
            case 7: printf("sedam"); break;
            case 8: printf("osam"); break;
            case 9: printf("devet"); break;
        }
    }
    else {
        printf("Pogresan unos.");
    }
    return 0;
}
```

Izmijenite program

Prilagodite program tako da se za uneseni četveroznamenasti broj iz intervala [1200, 1500] ispiše broj riječima.



4.5. Primjena ternarnog operatora

Ternarni operator u C-u (**uvjet ? izraz1 : izraz2;**) omogućava jednostavan način zapisivanja if-else izraza u jednom redu. Ako je uvjet istinit, izvršava se *izraz1*, inače *izraz2*.

If-else kôd u primjeru 4.5. s pomoću ternarnog operatora moguće je zapisati na sljedeći način:

```
y = (x >= 2 && x == y) ? (y + x) : (y - 3);
```

Primjer 4.16.

Napišite program koji provjerava dobiva li igrač bonus bodove u igri.

Ako je broj skupljenih bodova veći od 100, igrač dobiva bonus od 50 bodova, inače ne dobiva bonus.

Ispišite ukupan broj bodova nakon dodavanja bonusa.

Napomena: broj bodova je cijeli broj.

ulazni podatci	izlazni podatci
90	90
120	170

Rješenje

```
#include <stdio.h>

int main(void) {
    int bodovi, ukupno;

    printf("Učitaj broj bodova: ");
    scanf("%d", &bodovi);

    ukupno = (bodovi > 100) ? (bodovi + 50) : bodovi;

    printf("%d\n", ukupno);

    return 0;
}
```

Izmijenite program

Uvedite četiri razine bonusa (više od 200 → +100, više od 150 → +70, više od 100 → +50, inače bez bonusa) te izračunajte i ispišite ukupan broj bodova.



element.hr



VRIJEME ZA IZAZOV

Provjera temperature – što obući?

Radna situacija

Ujutro prije škole želiš provjeriti što bi trebao obući ovisno o temperaturi zraka.

Tvoj je zadatak izraditi program koji na temelju unesene temperature daje preporuku za odjeću.



Zadatak

1. Napiši program koji omogućuje unos **trenutačne temperature zraka** (u Celzijevim stupnjevima).
2. Program treba na temelju unesene temperature ispisati poruku:
 - ako je temperatura **manja od 5 °C** → „**Vrlo hladno – obuci jaknu i kapu.**”
 - ako je temperatura **od 5 °C do 18 °C** → „**Svježe – potrebna je lagana jakna.**”
 - ako je temperatura **veća od 18 °C** → „**Toplo – dovoljna je majica kratkih rukava.**”
3. Rezultat ispiši pregledno na zaslonu računala.
4. Dodaj komentare koji objašnjavaju glavne dijelove programa.
5. Napiši i **pseudokod** koji opisuje rješenje.

Napomena

- Pretpostavlja se ispravan unos podataka.
- Temperatura se unosi kao **realan broj**.
- Program upotrebljava **naredbu grananja** za donošenje odluke o prikazu poruke.

Primjer ulaznih i izlaznih podataka

ulazni podatci	izlazni podatci
Temperatura: 2	Vrlo hladno – obuci jaknu i kapu.
Temperatura: 12.5	Svježe – potrebna je lagana jakna.
Temperatura: 23	Toplo – dovoljna je majica kratkih rukava.

Elementi vrednovanja – program i pseudokod

elementi provjere	bodovi
unos temperature	3
primjena grananja za prikaz poruke	10
točan i pregledan ispis preporuke	5
dodani komentari u programu koji objašnjavaju uvjete	4
struktura pseudokoda jasno slijedi logiku algoritma	6
usklađenost pseudokoda s programom u jeziku C	2
ukupno	30

Ocjena

- 27 – 30 = odličan (5)
- 23 – 26 = vrlo dobar (4)
- 19 – 22 = dobar (3)
- 15 – 18 = dovoljan (2)
- 0 – 14 = nedovoljan (1)

Dodatni zadatak za one koji žele znati više

Proširi program tako da osim preporuke za odjeću izračunava i **razliku između unesene temperature i idealne temperature od 22 °C** te ispisuje koliko je stupnjeva **hladnije ili toplije** u odnosu na tu vrijednost.

Na primjer, ako korisnik unese **temperaturu od 16.7 °C**, program treba ispisati poruku:

„Danas je 5.3 Celzijevih stupnjeva hladnije od idealne temperature.”

Vrednovanje kao učenje



<https://s.element.hr/e247>

5.

PROGRAMSKE PETLJE



?

Dosad smo učili kako program donosi odluke s pomoću grananja. A što ako želimo da nešto ponavlja više puta?

E, to je posao petlji! One omogućuju programu automatsko ponavljanje određene naredbe dokle god je uvjet ispunjen.

...

Znači, umjesto da pišemo istu naredbu pet puta, možemo iskoristiti petlju da to napravi umjesto nas?

while

Upravo tako! Petlja je kao kada u aplikaciji za glazbu uključiš ponavljanje popisa pjesama - dok je ta opcija uključena, pjesme se stalno reproduciraju ispočetka.

do

!

Sad razumijem! Dakle, petlje omogućuju programu automatsko ponavljanje zadataka sve dok za to postoji uvjet.



DIGITALNI RUDNIK KODOVA

Skeniraj QR kôd ili otvori poveznicu u internetskom pregledniku. Prouči primjere 5.1, 5.4, 5.6, 5.13 i 5.18 te ih analiziraj i proširi programe prateći upute iz tih primjera.

<https://s.element.hr/b50c>

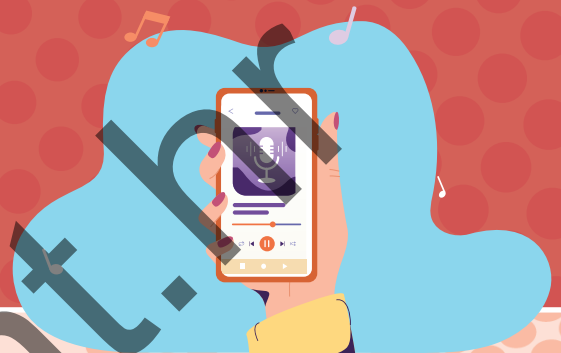




Razmisli

Na mobitelu možeš uključiti ponavljanje popisa pjesama. Pjesme se tada reproduciraju jedna za drugom, a nakon posljednje ponovno započinju od prve.

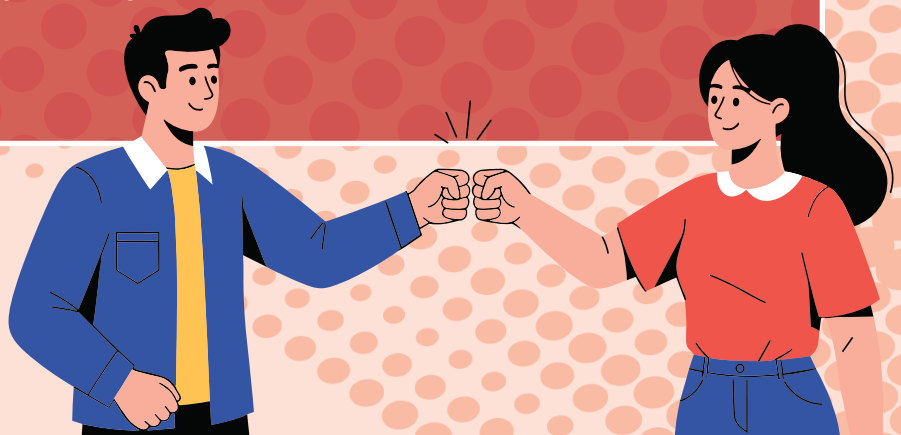
Do kada se takvo ponavljanje izvodi?



Nakon ove lekcije moći ćeš



- objasniti što su programske petlje i čemu služe
- razlikovati petlje **for**, **while** i **do-while**
- odabrati odgovarajuću petlju ovisno o problemu
- pratiti izvođenje programa s petljama i odrediti rezultat
- upotrebljavati uvjete unutar petlji
- napisati program koji ponavlja izvođenje naredbi dok je zadani uvjet ispunjen
- prepoznati i izbjeći beskonačne petlje
- upotrebljavati naredbe **break** i **continue**
- napisati program koji obrađuje više unosa podataka.



5.1. Ciklička algoritamska struktura

Cikličku algoritamsku strukturu u programiranju predstavljaju **programske petlje**. U programiranju ih se općenito upotrebljava kada se određeni blok koda ponavlja više puta. Na taj se način izbjegava pisanje istog bloka koda više puta, a sam kôd postaje pregledniji jer se količina koda može smanjiti.

Kako na zaslonu računala ispisati brojeve od 1 do 5 te velika slova od A do E? Rješenje je moguće prikazati na sljedeći način:

```
#include <stdio.h>
int main(void) {

    printf("%d\n%d\n%d\n%d\n%d\n", 1, 2, 3, 4, 5);
    printf("%c\n%c\n%c\n%c\n%c", 'A', 'B', 'C', 'D', 'E');

    return 0;
}
```

Prikazano rješenje i nije najbolje jer što ako problem treba poopćiti? Kako, na primjer, ispisati na zaslonu računala prvih n prirodnih brojeva ili prvih 20 slova engleske abecede? U tom slučaju ipak treba primijeniti programske petlje.

U programskom jeziku C postoje **tri vrste programskih petlji**:

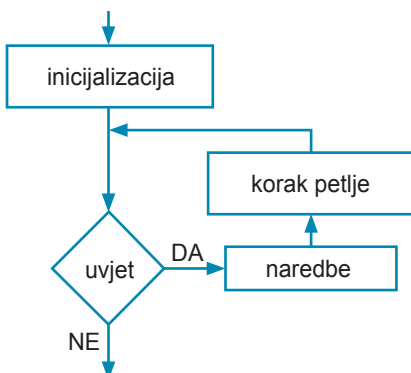
- petlja kod koje se uvjet ispituje na početku
- petlja kod koje se uvjet ispituje na kraju
- petlja s poznatim brojem ponavljanja.

5.1.1. Petlja kod koje se uvjet ispituje na početku

Petlja kod koje se **uvjet ispituje na početku** zove se **while petlja**. Upotrebljava se kad je određeni blok koda potrebno ponavljati sve dok je uvjet ispunjen.

Dijagramom toka (slika 5.1.) dan je prikaz while petlje.

Inicijalizacija nije nužno sastavni dio while petlje, a općenito while petlja u programskom jeziku C piše se na načine kako prikazuje tablica 5.1.



Slika 5.1. Dijagram toka, while petlja

Uvjet predstavlja logički izraz čija vrijednost može biti logička istina (1) ili laž (0). Naredbe unutar petlje ponavljaju se sve dok je uvjet istinit. Ako uvjet na samom početku nije bio istinit, petlja se nikad neće izvršiti. Kontrolna varijabla upravlja izvođenjem while petlje i određuje koliko će se puta petlja izvršiti. Njezina se vrijednost mijenja unutar tijela petlje (korak petlje) kako bi u nekom trenutku uvjet postao lažan i omogućio izlaz iz petlje.

Posebnu pažnju treba obratiti na pisanje uvjeta. Pogrešnim zapisom uvjeta petlja može biti i beskonačna, što znači da će se naredbe unutar petlje stalno ponavljati.

Tablica 5.1. While petlja – opći oblik

zapis kad while petlja sadržava više od 1 naredbe – vitičaste zagrade su obavezne	zapis kad while petlja sadržava jednu naredbu – vitičaste zagrade mogu se izostaviti
<pre>while (uvjet) { naredba_1; naredba_2; ... naredba_n; }</pre>	<pre>while (uvjet) naredba_1;</pre>

Primjer 5.1.

Napišite program koji će ispisati sve prirodne brojeve od 1 do 2024.

ulazni podatci	izlazni podatci
	1 2 ... 2024

Rješenje

```
#include <stdio.h>

int main(void) {
    int i = 1; //inicijalizacija

    while (i <= 2024) { //uvjet
        printf("%d\n", i);
        i++; //i = i + 1; i += 1; //korak petlje
    }

    return 0;
}
```

Kontrolna varijabla *i* inicijalizirana je na početnu vrijednost 1. Potom se provjerava je li ispunjen uvjet *i* <= 2024. S obzirom na to da je rezultat provjere logičkog uvjeta logička jedinica, izvršavaju se naredbe unutar petlje (ispisuje se trenutna vrijednost kontrolne varijable, za prvi korak to je vrijednost 1 i potom se korak petlje povećava za 1 i postaje 2). Potom se ponovno provjerava uvjet petlje i postupak se ponavlja sve dok je rezultat provjere uvjeta logička jedinica.

Proširite program

Omogućite unos broja *n* te ispišite brojeve od 1 do *n* i zatim od *n* do 1.



Primjer 5.2.

Napišite program koji će ispisati sva velika slova od A do Z.

ulazni podatci	izlazni podatci
	A B C ... Z

Rješenje

```
#include <stdio.h>

int main(void) {
    char j = 'A'; //inicijalizacija

    while (j <= 'Z') { //uvjet
        printf("%c\n", j);
        j++; //j = j + 1; j += 1; //korak petlje
    }

    return 0;
}
```

Izmijenite program

Ispišite slova od Z do A.



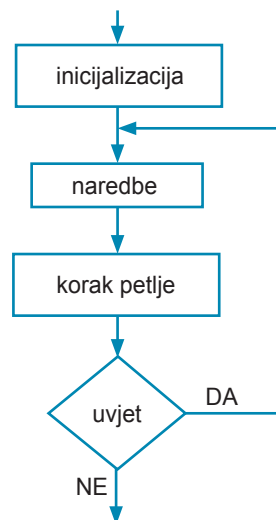
5.1.2. Petlja kod koje se uvjet ispituje na kraju

Petlja kod koje se **uvjet ispituje na kraju** zove se do-while petlja. Upotrebljava se kada je potrebno određeni blok koda ponoviti najmanje jednom te ga nastaviti izvršavati dokle god neki uvjet ne bude ispunjen. Ova je petlja karakteristična za programski jezik C i kao takva, na primjer, ne postoji u programskom jeziku Python.

Dijagramom toka (slika 5.2.) dan je prikaz **do-while petlje**.

Inicijalizacija nije nužno sastavni dio do-while petlje, a općenito do-while petlja u programskom jeziku C piše se na načine kako je prikazano tablicom 5.2. Kontrolna varijabla je varijabla koja upravlja izvođenjem do-while petlje i određuje koliko će se puta petlja izvršiti. Njezina se vrijednost mijenja unutar tijela petlje (korak petlje), dok se uvjet provjerava tek nakon prve iteracije, što osigurava da se petlja izvrši barem jednom.

Uvjet predstavlja logički izraz čija vrijednost može biti logička istina (1) ili laž (0).



Slika 5.2. Dijagram toka, do-while petlja

Tablica 5.2. Do-while petlja – opći oblik (vitičaste zagrade su uvijek obavezne)

zapis kad do-while petlja sadržava više od 1 naredbe – vitičaste zagrade su obavezne	zapis kad do-while petlja sadržava jednu naredbu – vitičaste zagrade su obavezne
<pre>do { naredba_1; naredba_2; ... naredba_n; } while (uvjet);</pre>	<pre>do { naredba_1; } while (uvjet);</pre>

Primjer 5.3.

Napišite program koji će ispisati sve prirodne brojeve od 1 do 2024.

ulazni podatci	izlazni podatci
	1 2 ... 2024

Rješenje

```
#include <stdio.h>

int main(void) {
    int i = 1; //inicijalizacija
    do {
        printf("%d\n", i);
        i++; //i = i + 1; i += 1; //korak petlje
    } while (i <= 2024); //uvjet
    return 0;
}
```

Izmijenite program

Ispišite brojeve od 2024 do 1.

**Primjer 5.4.**

Napišite program koji će ispisati sva velika slova od A do Z.

ulazni podatci	izlazni podatci
	A B C ... Z

Rješenje

```
#include <stdio.h>

int main(void) {

    char j = 'A'; //inicijalizacija

    do {
        printf("%c\n", j);
        j++; //j = j + 1; j += 1; //korak petlje
    } while (j <= 'Z'); //uvjet

    return 0;
}
```

Proširite program

Omogućite unos početnog i završnog slova te ispišite sva slova u tom intervalu.



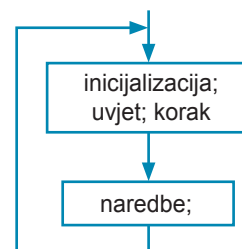
5.1.3. Petlja s poznatim brojem ponavljanja

Petlja s **poznatim brojem ponavljanja** u programskom jeziku C zove se for petlja.

Dijagramom toka (slika 5.3.) prikazana je **for petlja**.

For petlja sastoji se od triju dijelova koji su redom:

- inicijalizacija (postavljanje početne vrijednosti kontrolne varijable)
- uvjet (provjera koja određuje hoće li se petlja nastaviti)
- korak petlje (promjena kontrolne varijable nakon svake iteracije).



Slika 5.3. Dijagram toka, for petlja

Svi navedeni dijelovi već su prethodno prikazani kod petlji while i do-while. For petlja sve te dijelove ima navedene unutar jedne zagrade te na taj način prikaz koda ponekad može biti pregledniji. Kontrolna varijabla upravlja izvođenjem for petlje i određuje koliko će se puta petlja izvršiti. Njezina se vrijednost mijenja unutar koraka petlje, dok se uvjet provjerava prije svake iteracije.

Općenito for petlju u programskom jeziku C možemo zapisati na način kako prikazuje tablica 5.3.

Tablica 5.3. For petlja – opći oblik

zapis kad for petlja sadržava više od 1 naredbe – vitičaste zagrade su obavezne	zapis kad for petlja sadržava jednu naredbu – vitičaste zagrade mogu se izostaviti
<pre>for (inicijalizacija; uvjet; korak) { naredba_1; naredba_2; ... naredba_n; }</pre>	<pre>for (inicijalizacija; uvjet; korak) naredba_1;</pre>

Ponekad je moguće izostaviti pojedini dio for petlje.

1. Izostavljanje inicijalizacije:

Petlja upotrebljava već definiranu ili postojeću varijablu; inicijalizacija se mora obaviti izvan petlje.

```
int i = 0;
for (; i < 10; i++)
    printf("%d ", i);
```

2. Izostavljanje uvjeta:

Ako se uvjet izostavi, pretpostavlja se da je uvijek istinit, stvarajući beskonačnu petlju (ako nema prekida).

```
int i;
for (i = 0;; i++)
    printf("%d ", i);
```

3. Izostavljanje koraka petlje:

Ako korak petlje nije naveden, for petlja neće automatski mijenjati vrijednost kontrolne varijable. U tom slučaju, ako se vrijednost varijable ne promijeni unutar tijela petlje, petlja može postati beskonačna.

```
int i;
for (i = 0; i < 10;) {
    printf("%d ", i);
    i++;
}
```

Primjer 5.5.

Napišite program koji će ispisati sve prirodne brojeve od 1 do 2024.

ulazni podatci	izlazni podatci
	1 2 ... 2024

Rješenje

```
#include <stdio.h>

int main(void) {
    int i;
    for (i = 1; i <= 2024; i++)
        printf("%d\n", i);
    return 0;
}
```

Kontrolna varijabla *i* inicijalizirana je na početnu vrijednost 1. Potom se provjerava je li ispunjen uvjet $i \leq 2024$. S obzirom na to da je rezultat provjere logičkog uvjeta logička jedinica, izvršava se naredba unutar petlje (ispisuje se trenutna vrijednost kontrolne varijable za prvi korak, a to je vrijednost 1). Potom se korak petlje povećava za 1 i postaje 2. Zatim se ponovno provjerava uvjet petlje i postupak se ponavlja dokle god je rezultat provjere uvjeta logička jedinica.

Korak petlje moguće je zapisati i kao: $++i$, $i = i + 1$, $i += 1$.

Potrebno je uočiti da će se petlja prestati ponavljati kad vrijednost kontrolne varijable *i* postane 2025.

Izmijenite program

Prilagodite program tako da izračuna sumu parnih brojeva do 2024.



Primjer 5.6.

Napišite program koji će ispisati sva velika slova od A do Z.

ulazni podatci	izlazni podatci
	A B C ... Z

Rješenje

```
#include <stdio.h>

int main(void) {
    char j;
    for (j = 'A'; j <= 'Z'; j++)
        printf("%c\n", j);

    return 0;
}
```

Za ispis velikih slova od Z do A potrebno je promijeniti inicijalizaciju, uvjet i korak petlje na sljedeći način:

```
for (j = 'Z'; j >= 'A'; j--)
    printf("%c\n", j);
```

U slučaju kad se ispisuju velika slova od A do Z, petlja je **uzlazna**. Promjenom parametara u petlji petlja postaje **silazna**, a ispisuju se velika slova od Z do A.

Proširite program

Omogućite unos početnog i završnog slova te ispišite slova i u rastućem i u padajućem poretku.



5.1.4. Usporedba petlji i odabir odgovarajuće petlje

U programiranju se isti zadatak često može riješiti pomoću različitih vrsta petlji. Razlika između petlji `for`, `while` i `do-while` je u načinu provjere uvjeta i situacijama u kojima su najprikladnije za primjenu.

Primjeri primjene

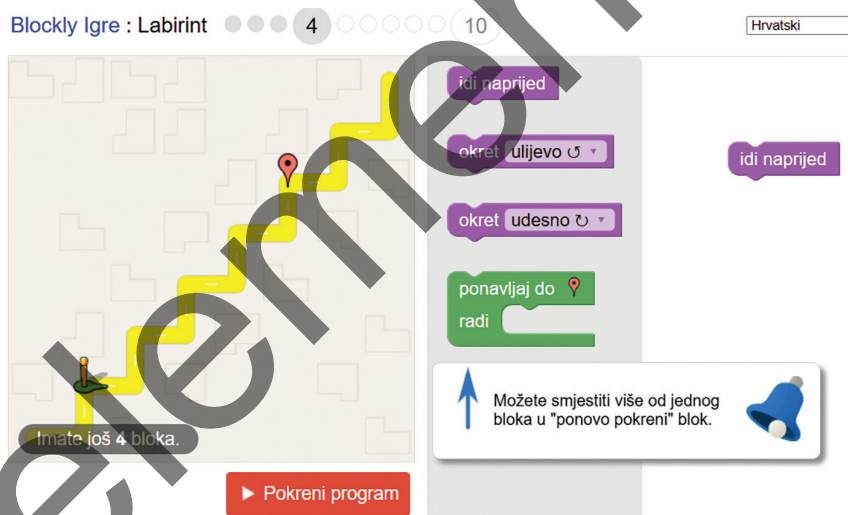
- Ako je unaprijed poznato koliko se puta neka radnja ponavlja (npr. ispis prvih 10 brojeva), **odabire se petlja `for`**.
- Ako broj ponavljanja nije poznat, a ponavljanje ovisi o uvjetu (npr. dok je broj veći od 0), **odabire se petlja `while`**.
- Ako je potrebno da se tijelo petlje izvrši barem jednom prije provjere uvjeta (npr. unos prvog broja prije provjere), **odabire se petlja `do-while`**.

Tablicom 5.4 prikazana je usporedba osnovnih karakteristika pojedinih petlji.

Tablica 5.4. Usporedba petlji i njihova primjena

petlja	kada odabrati	karakteristika	primjer
for	kada je broj ponavljanja unaprijed poznat	uvjet se provjerava prije svake iteracije	<pre>for (i = 1; i <= 10; i++) printf("%d\n", i);</pre>
while	kada broj ponavljanja nije poznat, a ovisi o uvjetu (npr. dok je broj veći od 0)	uvjet se provjerava prije izvođenja tijela petlje	<pre>while (n > 0) { printf("%d\n", n); n--; }</pre>
do-while	kada se naredbe moraju izvršiti barem jednom prije provjere uvjeta (npr. unos prvog broja)	uvjet se provjerava nakon izvođenja tijela petlje	<pre>do { scanf("%d", &n); } while (n != 0);</pre>

Blockly Games je interaktivni alat za učenje programiranja temeljen na blokovskom programiranju u kojem se naredbe slažu kao blokovi umjesto pisanja koda (slika 5.4).

**Slika 5.4.** Blockly Games – Labirint

U alatu Blockly Games (Labirint) pokušaj doći do razine 10.

<https://blockly.games/maze?lang=hr/>



5.2. Primjena cikličke algoritamske strukture (programskih petlji)

U sljedećim primjerima prikazuje se tipična primjena programskih petlji.

Primjer 5.7.

Napišite program kojim će se ispisati suma prvih n prirodnih brojeva.

ulazni podatci	izlazni podatci
5	Suma prvih 5 brojeva je: 15

Rješenje

```
#include <stdio.h>

int main(void) {
    int s = 0, i, n;

    printf("Učitajte n: ");
    scanf("%d", &n);

    for (i = 1; i <= n; i++)
        s += i;    //s = s + i;

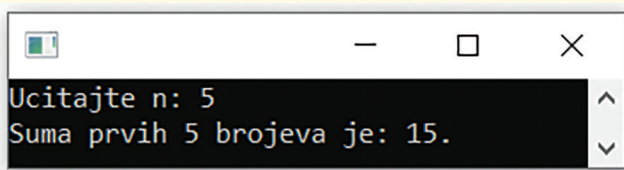
    printf("Suma prvih %d brojeva je: %d.", n, s);

    return 0;
}
```

Stanje varijabli prikazuje tablica 5.5.

Prikazuje se kako se mijenja stanje varijabli u programu ako je korisnik u program učitao vrijednost 5. Pri računanju sume varijablu je potrebno inicijalizirati na vrijednost 0 kako bi se izbjegla upotreba neodređene vrijednosti iz memorije i osigurao točan rezultat.

Ispis na zaslonu računala prikazuje slika 5.5.



Tablica 5.5. Stanje varijabli

i	n	s
		0
	5	
1		1
2		3
3		6
4		10
5		15
6		

Slika 5.5. Prikaz ispisa na zaslonu računala, suma prvih 5 prirodnih brojeva

Rješenje je moguće zapisati i putem `while` i `do-while` petlje kako je prikazano programima u nastavku:

```
#include <stdio.h>

int main(void) {
    int s = 0, i, n;

    printf("Ucitajte n: ");
    scanf("%d", &n);

    i = 1;
    while (i <= n) {
        s += i;
        i++;
    }
    printf("Suma prvih %d brojeva je: %d.", n, s);

    return 0;
}
```

```
#include <stdio.h>

int main(void) {
    int s = 0, i, n;

    printf("Ucitajte n: ");
    scanf("%d", &n);

    i = 1;
    do {
        s += i;
        i++;
    } while (i <= n);
    printf("Suma prvih %d brojeva je: %d.", n, s);

    return 0;
}
```

Izmijenite program

Prilagodite program tako da osim gornje granice n omogućuje i unos donje granice te izračuna sumu brojeva u tom intervalu.



Primjer 5.8.

Napišite program kojim će se izračunati umnožak svih dvoznamenkastih prirodnih brojeva koji su djeljivi s 2 i 19 te ga ispisati na zaslonu računala.

ulazni podatci	izlazni podatci
	Umnozak brojeva je: 2888

Rješenje

```
#include <stdio.h>
int main(void) {
    int u = 1, i;
    for (i = 10; i <= 99; i++) {
        if (i % 2 == 0 && i % 19 == 0) {
            u = u * i;
            printf("%d\n", i);
        }
    }
    printf("Umnozak brojeva je: %d", u);
    return 0;
}
```

U prikazanom rješenju početna je vrijednost umnoška postavljena na vrijednost 1 ($u = 1$). U kodu je primijenjena i razgranata algoritamska struktura kako bi se pomnožili samo oni brojevi koji zadovoljavaju zadani uvjet u zadatku (u ovom slučaju to su brojevi 38 i 76). Kôd je moguće pojednostavniti (optimizirati) smanjenjem broja prolazaka kroz petlju tako da se izmijeni uvjet i korak petlje:

```
int u = 1, i;
for (i = 10; i <= 98; i = i + 2) {
    if (i % 19 == 0) {
        u = u * i;
        printf("%d\n", i);
    }
}
```

Rješenje putem while i do-while petlje prikazuju programi u nastavku:

```
#include <stdio.h>
int main(void) {
    int u = 1, i;
    i = 10;
    while (i <= 99) {
        if (i % 2 == 0 && i % 19 == 0) {
            u = u * i;
            printf("%d\n", i);
        }
        i++;
    }
    printf("Umnozak brojeva je: %d", u);
    return 0;
}
```



```
#include <stdio.h>

int main(void) {
    int u = 1, i;

    i = 10;
    do {
        if (i % 2 == 0 && i % 19 == 0) {
            u = u * i;
            printf("%d\n", i);
        }
        i++;
    } while (i <= 99);
    printf("Umnozak brojeva je: %d", u);
    return 0;
}
```

Primjer 5.9.

Napišite program kojim će se izračunati zbroj svih znamenaka u prirodnom broju.
Napomena: nije poznato koliko broj ima znamenaka.

ulazni podatci	izlazni podatci
123	Zbroj znamenaka broja 123 je: 6

Rješenje

S obzirom na to da se unaprijed ne zna koliko će broj imati znamenaka, algoritam je praktično zapisati s pomoću petlje while.

```
#include <stdio.h>

int main(void) {
    int n, pom, m, s = 0;

    printf("Učitajte broj n: ");
    scanf("%d", &n);
    m = n;

    while (n != 0) {
        pom = n % 10;
        s = s + pom;
        n = n / 10;
    }

    printf("Zbroj znamenaka broja %d je: %d", m, s);

    return 0;
}
```

Kako se mijenja stanje varijabli, prikazuje tablica 5.6.

Tablica 5.6. Stanje varijabli

pom	s	n	m	
	0			
		123		
			123	
3	3	12		1. korak
2	5	1		2. korak
1	6	0		3. korak

Izmijenite program

Prilagodite program tako da izračuna zbroj znamenaka koje su prosti brojevi (2, 3, 5, 7).



Primjer 5.10.

Napišite program koji ispisuje je li učitani prirodni broj n savršen (broj je savršen ako je jednak zbroju svih svojih djelitelja ne računajući njega samog, npr. 6 je savršen broj jer vrijedi $6 = 1 + 2 + 3$)

ulazni podatci	izlazni podatci
28	28 je savršen broj.
15	15 nije savršen broj.

Napomena: pretpostavite da će korisnik ispravno učitati podatke.

Rješenje

```
#include <stdio.h>

int main(void) {
    int i, x, s = 0;
    printf("Ucitajte broj: ");
    scanf("%d", &x);

    for (i = 1; i < x; i++)
        if (x % i == 0)
            s = s + i;
    if (s == x)
        printf("%d je savršen broj.", x);
    else
        printf("%d nije savršen broj." x);
    return 0;
}
```

Primjer 5.11.

Napišite program u koji će se učitavati prirodan broj n i računati zbroj prvih n članova reda (pretpostavite da je broj n ispravno utipkan):

$$1 - \frac{1}{4} + \frac{1}{9} - \frac{1}{16} + \dots$$

ulazni podatci	izlazni podatci
4	0.799

Rješenje

```
#include <stdio.h>
#include <math.h>

int main(void) {
    int i, j, n;
    float s = 0;
    printf("Učitajte broj: ");
    scanf("%d", &n);

    //1. način
    /*
        for (i = 1; i <= n; i++)
            s = s + pow(-1, i + 1) * 1 / (pow(i, 2));
    */

    //2. način
    for (i = 1; i <= n; i++)
        if (i % 2 == 1)
            s = s + 1 / (pow(i, 2));
        else
            s = s - 1 / (pow(i, 2));

    printf("%.3f", s);

    return 0;
}
```

Primjer 5.12.

Često na ispitu znanja iz *programiranja* postoji nekoliko različitih inačica istog ispita znanja. To se popularno naziva grupama. Ovaj su se put nastavnici odlučili za **dvije grupe**: *grupa P* i *grupa N*. *Grupu P* pisali su učenici s **parnim**, a *grupu N* učenici s **neparnim rednim brojem** na popisu. Nakon ispravljanja ispita nastavnici su htjeli odrediti tri podatka:

1. podatak: koliki je **prosjeck** dobivenih ocjena onih učenika koji su pisali *grupu P*
2. podatak: koliki je **prosjeck** dobivenih ocjena onih učenika koji su pisali *grupu N*
3. podatak: koja je grupa imala **bolji** prosjeck ocjena.

Napišite program koji na osnovi dobivenih ocjena određuje i ispisuje tražene podatke.

U prvom retku nalazi se prirodan broj **N** ($1 \leq N \leq 30$), broj učenika koji pišu ispit.

5. Programske petlje

U sljedećih **N** redaka nalazi se po jedan prirodan broj O_i ($1 \leq O_i \leq 5$, $i = 1 \dots N$), pri čemu je O_1 ocjena iz ispita znanja koju je dobio učenik s rednim brojem „1.“, O_2 ocjena učenika s rednim brojem „2.“ i tako do O_n što je ocjena učenika s rednim brojem „N.“.

U prvom retku treba ispisati 1. podatak, a u drugom retku 2. podatak. Oba broja su realna. U treći redak treba ispisati 3. podatak, slovo „P“ ili „N“, ovisno o tome koja je grupa imala veći prosjek ocjena. Budu li obje grupe imale isti prosjek, tada treba ispisati riječ „PN“.

ulazni podatci	izlazni podatci
6	3.00
2	2.66
4	P
5	
2	
1	
3	

ulazni podatci	izlazni podatci
5	2.50
2	2.33
3	P
1	
2	
4	

ulazni podatci	izlazni podatci
4	4.50
4	4.50
5	PN
5	
4	

Rješenje

```
#include <stdio.h>

int main(void) {
    int i, n, sn = 0, sp = 0, brn = 0, brp = 0, oc;
    float arsn, arsp;
    printf("Ucitajte broj ucenika:");
    scanf("%d", &n);
    for (i = 1; i <= n; i++) {
        printf("Ucitajte ocjenu %d. ucenika: ", i);
        scanf("%d", &oc);
        if (i % 2 == 1) {
            sn += oc;
            brn++;
        }
        else {
            sp += oc;
            brp++;
        }
    }

    arsn = (float)sn / brn;
    arsp = (float)sp / brp;
    printf("%.2f\n%.2f\n", arsp, arsn);

    if (arsn > arsp)
        printf("%c", 'N');
    else if (arsn < arsp)
        printf("%c", 'P');
    else
        printf("%s", "PN");

    return 0;
}
```

5.3. Ugniježdene petlje

Ugniježdene petlje u programskom jeziku C predstavljaju petlje koje se nalaze unutar tijela drugih petlji. Često ih se upotrebljava kada je potrebno provesti iteracije kroz više razina ili dimenzija, primjerice za rad s matricama, ispis tablica ili slične zadatke.

Primjer 5.13.

Napišite program koji ispisuje na zaslonu računala tablicu množenja do 100.

Rješenje

```
#include <stdio.h>

int main(void) {
    int i, j;

    for (i = 1; i <= 10; i++) {
        for (j = 1; j <= 10; j++) {
            printf("%5d", i * j);
        }
        printf("\n");
    }

    return 0;
}
```

Prikazani primjer upotrebljava dvije ugniježdene petlje. Jedna i druga petlja prolaze vrijednostima od 1 do 10, a ukupno će ispisati 100 vrijednosti ($10 \cdot 10$). Za jedan korak vanjske petlje `for (i = 1; i <= 10; i++)` izvršava se cijela unutarnja petlja `for (j = 1; j <= 10; j++)`.

Kako se mijenja stanje varijabla, prikazuje tablica 5.7.

Tablica 5.7. Stanje varijabla

i	1						2			...	10			11
j	1	2	3	...	10	11	1	2	3	...	1	...	11	

Prikaz na zaslonu računala nalazi se na slici 5.6.

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Slika 5.6. Ispis tablice množenja do 100, primjena ugniježđenih petlji

Proširite program

Omogućite unos broja n te ispišite tablicu množenja dimenzije $n \times n$.



Primjer 5.14.

Napišite program koji ispisuje zadani oblik na zaslonu računala u n redaka, pri čemu n učitava korisnik (n je prirodan broj, pretpostavite da će korisnik ispravno učitati n).

ulazni podatci	izlazni podatci
4	<pre> Z Z Z Z * * Z * * Z * * Z Z Z Z </pre>

Rješenje

```

#include <stdio.h>

int main(void) {
    int i, j, n;
    printf("Ucitajte broj: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (i == 0 || i == n - 1 || i + j == n - 1)
                printf("Z ");
            else
                printf("* ");
        }
        printf("\n");
    }
    return 0;
}

```



Izmijenite program

Prilagodite uvjet tako da se po dijagonalama ispisuje z, a na svim ostalim mjestima *.

5.3.1. Pojam beskonačne petlje

Često prilikom pisanja programa, a najčešće zbog neke **logičke pogreške**, nastane **beskonačna petlja**. Beskonačna petlja problem je u programiranju kada petlja nikada ne završi, a to može imati za posljedicu i pad programa.

Ako se u primjeru 5.5. postavi korak petlje kao `i--` kako je prikazano kodom u nastavku, dobiva se beskonačna petlja.

```

#include <stdio.h>
int main(void) {

    int i;
    for (i = 1; i <= 2024; i--)
        printf("%d\n", i);

    return 0;
}

```

Uvjet `i <= 2024` nikada neće postati false jer se vrijednost varijable `i` smanjuje (`i--`) umjesto da se povećava.

Prilikom pisanja programa mora se voditi računa o tome da u kodu ne nastane beskonačna petlja. Iako je ovakva petlja teoretski beskonačna, u praksi zbog ograničenog raspona cijelih brojeva (`int`) može doći do pojave prelijevanja pri smanjivanju vrijednosti (engl. *integer underflow*), što može promijeniti ponašanje programa.

U nekim programima, poput izbornika, igara ili upravljanja semaforom, potrebno je da se program stalno ponavlja dok korisnik ne odluči završiti rad pa se tada odabire beskonačna petlja. Beskonačna petlja može se zapisati, primjerice, s pomoću oblika `for (;;) { ... }` ili `while (1) { ... }`, gdje je uvjet uvijek istinit.

Primjer 5.15.

Napišite program koji prikazuje sljedeći izbornik:

1 – Ispis poruke

2 – Izlaz

Odabir:

Program omogućuje korisniku odabir jedne od ponuđenih opcija. Ako korisnik odabere opciju 1, ispisuje se poruka "Pozdrav!". Ako odabere opciju 2, program završava.

Ako korisnik unese vrijednost koja nije ponuđena u izborniku, program ispisuje poruku "Nepostojeća opcija!" i nastavlja s radom.

Izbornik se ponavlja sve dok korisnik ne odabere izlaz.

ulazni podatci	izlazni podatci
1	Pozdrav!
2	Program završava

Rješenje

U ovom se primjeru upotrebljava beskonačna petlja `while (1)`, koja omogućuje stalno ponavljanje izbornika. U svakoj iteraciji korisnik unosi svoj odabir, a program na temelju tog unosa izvršava odgovarajuću radnju.

Izlazak iz petlje ostvaruje se naredbom `return`, čime se završava izvođenje programa.

```
#include <stdio.h>

int main(void) {
    int izbor;
    while (1) {
        printf("\n1 - Ispis poruke\n");
        printf("2 - Izlaz\n");
        printf("Odabir: ");
        scanf("%d", &izbor);

        if (izbor == 1) {
            printf("Pozdrav!\n");
        }
        else if (izbor == 2) {
            printf("Program završava.\n");
            return 0;
        }
        else {
            printf("Nepostojeća opcija!\n");
        }
    }
}
```

Izmijenite program

Umjesto petlje `while` upotrijebite petlju `for`.



5.4. Primjena naredbi break i continue

Naredba break

Naredba break upotrebljava se za izlazak iz petlje koja se trenutno izvršava. Izvršavanjem naredbe break petlja se odmah prekida, neovisno o tome jesu li preostali koraci (iteracije) petlje izvršeni ili ne.

U slučaju ugniježđenih petlji naredba break prekida samo onu petlju u kojoj se nalazi.

Primjer 5.16.

Napišite program koji simulira unos ocjena za više učenika. Za svakog učenika unose se njegove ocjene (od 1 do 5). Ako se unese ocjena 0, prekida se unos ocjena za tog učenika i prelazi se na sljedećeg učenika. Program treba obraditi 3 učenika.

ulazni podatci	izlazni podatci
5 4 0	Ucenik 1 završen
3 2 1 0	Ucenik 2 završen
4 0	Ucenik 3 završen

Rješenje

U ovom primjeru primjenjuje se ugniježđena petlja. Vanjska petlja predstavlja učenike, dok unutarnja petlja omogućuje unos više ocjena za svakog učenika.

Kada se unese vrijednost 0, izvršava se naredba break, čime se prekida samo unutarnja petlja u kojoj se naredba nalazi. Vanjska se petlja pritom ne prekida, već se nastavlja izvršavati i prelazi na sljedećeg učenika.

Na taj način jasno se vidi da naredba break u slučaju ugniježđenih petlji utječe isključivo na najbližu (unutarnju) petlju.

```
#include <stdio.h>
int main(void) {
    int i, ocjena;
    for (i = 1; i <= 3; i++) {
        printf("Ucenik %d:\n", i);
        while (1) {
            printf("Unesite ocjenu (0 za kraj): ");
            scanf("%d", &ocjena);
            if (ocjena == 0) {
                break;
            }
        }
        printf("Ucenik %d završen.\n", i);
    }
    return 0;
}
```

Izmijenite program

Prilagodite program tako da obrađuje podatke za n učenika.



Naredba continue

Naredba `continue` upotrebljava se u programskim petljama za preskakanje preostalog dijela tijela petlje u trenutačnoj iteraciji. Nakon izvršavanja naredbe `continue`, odmah se prelazi na sljedeću iteraciju petlje.

Primjer 5.17.

Napišite program koji omogućuje unos ocjena za jednog učenika.

Ako korisnik unese ocjenu koja nije u rasponu od 1 do 5, takav se unos zanemaruje i ne obrađuje. Program nastavlja s unosom sljedeće ocjene.

Unos završava kada se unese vrijednost 0.

ulazni podatci	izlazni podatci
5 7 4 0	5 4

Rješenje

Program primjenjuje beskonačnu petlju `while` (1) za unos ocjena. U svakoj iteraciji korisnik unosi jednu ocjenu.

Ako se unese vrijednost 0, izvršava se naredba `break`, čime se prekida izvođenje petlje i završava unos.

Ako unesena ocjena nije u rasponu od 1 do 5, izvršava se naredba `continue`, čime se preskače ostatak tijela petlje i prelazi na sljedeću iteraciju.

Za valjane ocjene (od 1 do 5) ispisuje se unesena vrijednost.

```
#include <stdio.h>

int main(void) {
    int ocjena;
    while (1) {
        printf("Unesite ocjenu (0 za kraj): ");
        scanf("%d", &ocjena);

        if (ocjena == 0) {
            break;
        }

        if (ocjena < 1 || ocjena > 5) {
            continue;
        }

        printf("%d\n", ocjena);
    }

    return 0;
}
```

Isti problem moguće je riješiti i bez upotrebe naredbi `break` i `continue`. U tom slučaju izlazak iz petlje i obrada podataka ostvaruju se s pomoću uvjeta u zaglavlju petlje i unutar tijela petlje.

```
#include <stdio.h>

int main(void) {
    int ocjena;

    do {
        printf("Unesite ocjenu (0 za kraj): ");
        scanf("%d", &ocjena);

        if (ocjena != 0 && ocjena >= 1 && ocjena <= 5) {
            printf("%d\n", ocjena);
        }

    } while (ocjena != 0);

    return 0;
}
```

Izmijenite program

Prilagodite program tako da prebrojite koliko je učitano pozitivnih ocjena (2 – 5).



5.5. Primjena operatora zarez (,) u for petlji

Operator zarez (,) omogućuje izvođenje više izraza unutar `for` petlje. Najčešće se upotrebljava u inicijalizaciji i iteraciji varijabli (korak petlje).

```
int i, j;
for (i = 1, j = 10; i < j; i++, j--)
    printf("i = %d, j = %d\n", i, j);
```

U inicijalizaciji se postavljaju vrijednosti `i = 1` i `j = 10` s pomoću operatora zarez (,). U iteraciji (koraku petlje) istovremeno se povećava `i` za 1 (`i++`) i smanjuje `j` za 1 (`j--`). Petlja će se zaustaviti kada vrijedi uvjet `i >= j`.

Primjer 5.18.

Odredite što će se ispisati na zaslonu računala.

```
#include <stdio.h>

int main(void) {
    int i, j;

    for (i = 1, j = 5; i <= 5; i++, j--)
        printf("i = %d, j = %d\n", i, j);

    return 0;
}
```

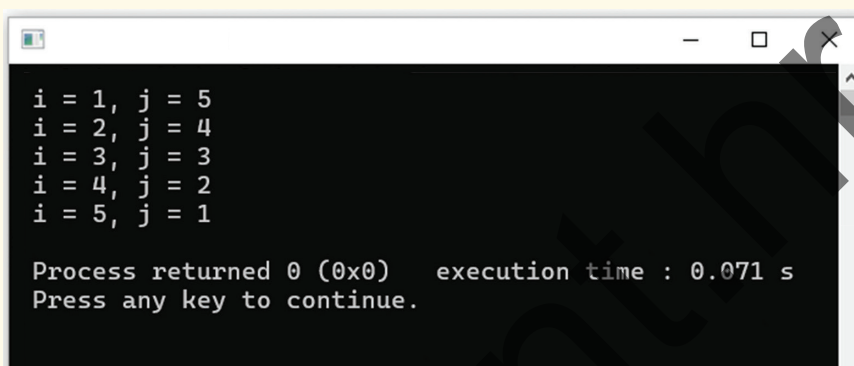
Rješenje

U inicijalizaciji for petlje postavljaju se početne vrijednosti varijabli $i = 1$ i $j = 5$ s pomoću operatora zarez.

U svakom prolazu petlje varijabla i se povećava za 1 ($i++$), a varijabla j smanjuje se za 1 ($j--$).

Petlja se izvršava dok je uvjet $i \leq 5$ istinit.

Ispis na zaslonu računala prikazuje slika 5.6.



```
i = 1, j = 5
i = 2, j = 4
i = 3, j = 3
i = 4, j = 2
i = 5, j = 1

Process returned 0 (0x0)   execution time : 0.071 s
Press any key to continue.
```

Slika 5.6. Ispis vrijednosti varijabli i i j tijekom izvođenja for petlje uz primjenu operatora zarez

Proširite program

Ospišite samo one parove vrijednosti varijabli i i j za koje vrijedi da je $i > j$.

**Primjer 5.19.**

Dva drona nalaze se na različitim visinama.

Učitajte njihove početne visine.

U svakoj iteraciji:

- prvi dron poveća svoju visinu za 2 metra
- drugi dron smanji svoju visinu za 2 metra.

Prikažite njihove visine nakon svake iteracije sve dok se ne susretnu ili prođu jedan pokraj drugoga.

Napomena: visine su cijeli brojevi.

Pretpostavlja se da je početna visina prvog drona manja ili jednaka početnoj visini drugog drona.

ulazni podatci	izlazni podatci
0	Dron1 = 0 m, Dron2 = 10 m
10	Dron1 = 2 m, Dron2 = 8 m
	Dron1 = 4 m, Dron2 = 6 m

Rješenje

Operator zarez omogućuje istovremeno mijenjanje sadržaja dviju varijabli.

```
#include <stdio.h>

int main(void) {

    int x, y;

    printf("Ucitaj pocetnu poziciju prvog drona: ");
    scanf("%d", &x);

    printf("Ucitaj pocetnu poziciju drugog drona: ");
    scanf("%d", &y);

    for (; x <= y; x += 2, y -= 2)
        printf("Dron1 = %d m, Dron2 = %d m\n", x, y);

    return 0;
}
```

Izmijenite program

Napišite program upotrebom petlje while.



element.hr



VRIJEME ZA IZAZOV

Brojenje koraka – digitalni pedometar

Radna situacija:

Pametni sat bilježi koliko koraka korisnik napravi tijekom dana.

Na kraju svakog dijela dana uređaj zabilježi trenutačni broj koraka i provjeri je li korisnik dosegnuo zadani dnevni cilj.



Tvoj je zadatak izraditi program koji simulira rad takvog digitalnog pedometra.

Zadatak

1. Napiši program koji omogućuje unos **ciljanog broja koraka** (npr. 10 000).
2. Program zatim omogućuje **do pet unosa broja koraka** – jednom za svako vremensko razdoblje dana (ujutro, prije podne, poslijepodne, navečer i prije spavanja).
3. Nakon svakog unosa program **zbraja ukupne korake** i provjerava je li **cilj postignut ili premašen**.
4. Ako korisnik unese **negativan broj koraka**, unos se mora ponoviti.
5. Program prestaje tražiti nove unose ako:
 - **cilj je postignut** ili
 - korisnik je već unio podatke **pet puta** (kraj dana).
6. Kada se program završi, ispisuje se jedna od poruka:
 - „**Cilj ostvaren! Ukupno napravljeno: [broj] koraka.**”
 - „**Cilj nije ostvaren. Ukupno napravljeno: [broj] koraka. Do cilja nedostaje: [broj] koraka.**”
 - **Cilj ostvaren nakon [broj]. unosa! Ukupno napravljeno: [broj] koraka.**
7. Rezultati se ispisuju pregledno na zaslonu računala, a rješenje se opisuje i s pomoću **pseudokoda**.

Napomena

- Koraci se unose kao **cijeli brojevi**.
- Ako se unese **negativna vrijednost**, program ponavlja unos za to razdoblje.

Primjer ulaznih i izlaznih podataka

ulazni podatci	izlazni podatci
Ciljani broj koraka: 10 000 Unosi koraka: 1800, 2000, 2200, 1900, 2100	Cilj ostvaren! Ukupno napravljeno: 10 000 koraka.
Ciljani broj koraka: 12 000 Unosi koraka: 1500, 1700, 2000, -300, 2100, 1900 (<i>ponovljen unos zbog negativne vrijednosti</i>)	Cilj nije ostvaren. Ukupno napravljeno: 9200 koraka. Do cilja nedostaje: 2800 koraka.
Ciljani broj koraka: 9 000 Unosi koraka: 3000, 2800, 3400	Cilj ostvaren nakon 3. unosa! Ukupno napravljeno: 9 200 koraka.

Elementi vrednovanja – program i pseudokod

elementi provjere	bodovi
unos ciljanog broja koraka	3
ispravno ponavljanje unosa broja koraka s pomoću petlje	8
provjera i ponavljanje unosa u slučaju negativne vrijednosti	4
ispravno zbrajanje ukupnog broja koraka	4
točan uvjet prekida petlje pri ostvarenju cilja	4
jasno i pregledno ispisane poruke korisniku	3
struktura pseudokoda jasno slijedi logiku programa	3
ukupno	30

Ocjena

- 27 – 30 = odličan (5)
- 23 – 26 = vrlo dobar (4)
- 19 – 22 = dobar (3)
- 15 – 18 = dovoljan (2)
- 0 – 14 = nedovoljan (1)

Dodatni zadatak za one koji žele znati više

Proširi program tako da se osim ukupnog broja koraka ispiše i **u kojem je dijelu dana** korisnik postigao svoj cilj.

Svaki unos broja koraka odnosi se na određeni dio dana:

1. **Prvi unos** – ujutro
2. **Drugi unos** – prijepodne
3. **Treći unos** – poslijepodne
4. **Četvrti unos** – navečer
5. **Peti unos** – prije spavanja

Ako korisnik **postigne cilj** tijekom dana, program treba ispisati poruku:

„Cilj ostvaren [dio dana]! Ukupno napravljeno: [broj] koraka.”

Ako korisnik **nije ostvario cilj ni nakon petog unosa**, program treba ispisati:

„Cilj nije ostvaren ni do kraja dana.”

Vrednovanje kao učenje



<https://s.element.hr/057e>

SAŽETAK

1. Algoritmi i dijagrami toka (Raptor)

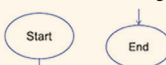
Algoritam – niz jasno definiranih koraka za rješavanje problema (ulaz → obrada → izlaz)

Algoritam se može zapisati:

- tekstem
- **pseudokodom**
- **dijagramom toka**
- programskim jezikom

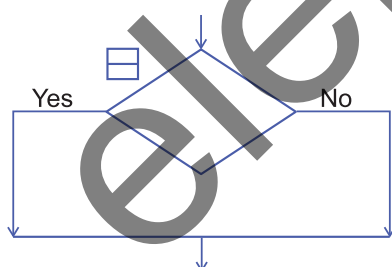
Pseudokod – zapis algoritma s pomoću jednostavnih naredbi sličnih govornom jeziku (početak, ulaz, izlaz, kraj)

Dijagram toka – grafički prikaz algoritma korak po korak

osnovne naredbe (pseudokod / dijagram toka)	varijable, pridruživanje i operatori	osnovne funkcije
• početak / kraj  • ulaz() / izlaz()  • pridruživanje (varijabla = izraz) 	• varijabla = spremnik za vrijednost • pridruživanje vrijednosti s pomoću izraza • aritmetički operatori: +, -, *, /, mod • relacijski operatori: ==, !=, <, >, <=, >= • logički operatori: and, or, not	• sqrt(x) – korijen • abs(x) – apsolutna vrijednost • ceiling(x) – zaokruživanje na veći cijeli broj • floor(x) – zaokruživanje na manji cijeli broj • round(x) – zaokruživanje na najbliži cijeli broj

Kontrolne strukture:

- **Grananje (ako–onda)** – donošenje odluke



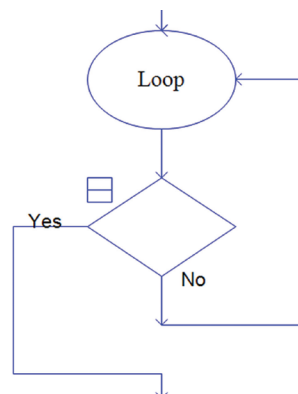
```

ako je uvjet onda
    blok_naredbi_1
} inače {
    blok_naredbi_2
}

```

- **Petlje** – ponavljanje naredbi

poznat broj ponavljanja	uvjet na početku (dok je)
za i = p do k činiti { blok_naredbi }	dok je uvjet činiti { blok_naredbi }
uvjet na kraju (ponavljaj dok)	
ponavljati { blok_naredbi } dok je uvjet	



2. Osnove programiranja u C-u

Program u jeziku C sastoji se od:

- deklaracija varijabli i funkcija
- pretprocesorska direktiva (npr. #include)
- naredbi unutar blokova

```
#include <stdio.h>
int main(void) {

    printf("Hello world!\n");

    return 0;
}
```

Blok – dio programa omeđen vitičastim zagradama {}.

Svaka **naredba** završava znakom ;

Vrste pogrešaka: sintaktičke, semantičke, pogreške tijekom izvođenja i logičke.

Komentari u programu: // linijski komentar /* blokovski komentar */

Variable:

- varijabla je mjesto u memoriji za pohranu vrijednosti
- vrijednost se može mijenjati tijekom izvođenja programa
- deklaracija: tip ime_varijable;
- deklaracija i inicijalizacija: tip ime_varijable = vrijednost;

Tipovi podataka:

- cjelobrojni (int, char)
- realni: float

Dodjela vrijednosti:

- unosom (scanf)
- pridruživanjem (x = 5)
- višestrukim pridruživanjem (x = y = 5)

Konstante – vrijednosti koje se ne mijenjaju:

- **cjelobrojne** (dekadske, oktalne, heksadekadske)
- **realne**
- **znakovne**
- **simboličke** (#define NAZIV vrijednost)

Deklaracija konstante: const tip naziv = vrijednost;

Ulaz i izlaz:

unos: scanf("format", &varijabla1, &varijabla2, ...);	ispis: printf("format", varijabla / izraz / konstanta);
format: char %c, int %d, float %f	

Ugrađene funkcije – primjenjuju se za izvođenje operacija #include <math.h>

- sqrt(x) – korijen
- fabs(x) – apsolutna vrijednost
- pow(b, e) – potenciranje
- round(x) – zaokruživanje

Linijska (slijedna) struktura programa – naredbe se izvršavaju redom

3. Operatori, tipovi podataka i konverzija tipova podataka

Brojevni sustavi

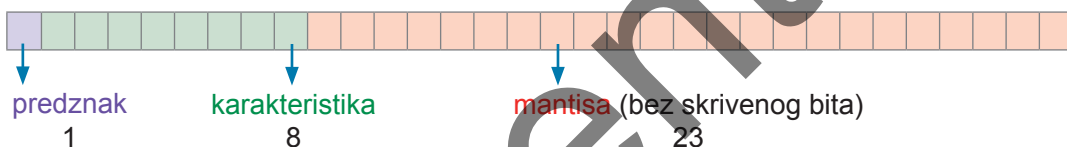
- dekadski (baza 10, znamenke 0 – 9)
- binarni (baza 2, znamenke 0 i 1)
- heksadekadski (baza 16, znamenke 0 – 9 i A – F)

Prikaz podataka u računalu

- podatci se zapisuju u memoriji u obliku **binarnih znamenaka (0 i 1)**; osnovna jedinica je bit; 1 B = 8 bitova; razlikujemo KiB, MiB, GiB od kB, MB, GB
- bistabil ima dva stanja (0, 1) i pohranjuje jedan bit; više bistabila čini registar, a niz registara čini memoriju; duljina registra naziva se riječ

0	1	0	1	1	1	0	1
---	---	---	---	---	---	---	---

1. **zapis cijelih brojeva** – metoda dvojnog komplementa
2. **zapis realnih brojeva u računalu** – IEEE 754 standard jednostruke i dvostruke preciznosti



3. **prikaz znakova** – znakovi se zapisuju s pomoću **ASCII koda**; jedan znak zauzima 1 bajt (8 bitova)

Operatori

- operator pridruživanja (=)
- aritmetički (+, -, *, /, %)
- relacijski (==, !=, <, >, <=, >=)
- logički (&, ||, !)
- inkrement i dekrement (++ , --)
- operator **sizeof**
- bitovni (&, |, ^, ~, <<, >>)

Tipovi podataka

- cjelobrojni (char, int, _Bool (bool)) i realni (float, double)
- kvalifikatori: short, long, long long (mijenjaju raspon), signed, unsigned (određuju predznak)
- tipovi se mogu kombinirati (npr. unsigned int, signed long long int, unsigned char)
- zauzeće u memoriji ovisi o sustavu; provjerava se operatorom sizeof
- tipične veličine: char, bool (1 B); short (2 B); int, long, float (4 B); double (8 B)

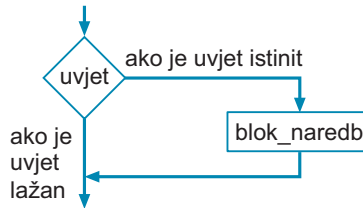
Konverzija tipova podataka – pretvorba tipova može biti:

- implicitna (automatska – manji u veći tip: long double > double > float > long > int > short > char)
- eksplicitna s pomoću kastinga (tip_podatka) izraz;

4. Grananje u programu

Jednostrana selekcija if

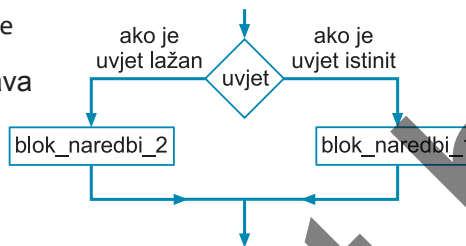
- naredba if služi za donošenje odluke
- blok naredbi izvršava se samo ako je uvjet istinit



```
if (uvjet) {
    ...
    blok_naredbi;
    ...
}
```

Dvostrana selekcija if else

- ako je uvjet istinit → izvršava se prvi blok
- ako je uvjet lažan → izvršava se drugi blok



```
if (uvjet) {
    ...
    blok_naredbi_1;
    ...
} else {
    ...
    blok_naredbi_2;
    ...
}
```

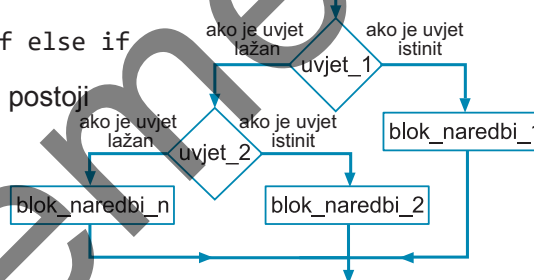
Primjena ternarnog operatora

- skraćeni zapis naredbe if else
- `uvjet ? izraz1 : izraz2;`

```
if (uvjet_1) {
    ...
    blok_naredbi_1;
    ...
} else if (uvjet_2) {
    ...
    blok_naredbi_2;
    ...
} else {
    ...
    blok_naredbi_n;
    ...
}
```

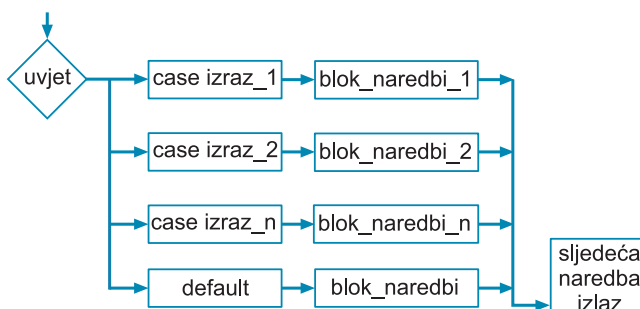
Višestruka selekcija if else if

- odabire se kada postoji više uvjeta
- uvjeti se provjeravaju redom



Višestruka selekcija switch case

- služi za odabir između više mogućnosti
- provjerava se vrijednost izraza



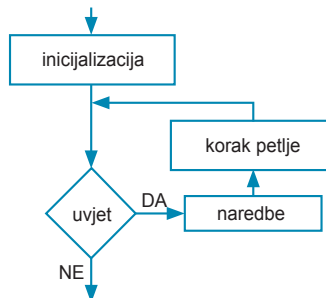
```
switch (uvjet) {
    case izraz_1:
        ...
        blok_naredbi_1;
        ...
        break;
    case izraz_2:
        ...
        blok_naredbi_2;
        ...
        break;
    case izraz_n:
        ...
        blok_naredbi_n;
        ...
        break;
    default:
        ...
        blok_naredbi;
        ...
}
```

5. Programske petlje

Programske petlje – omogućuju ponavljanje izvođenja naredbi i upotrebljavaju se kada se isti dio programa izvršava više puta.

while petlja

- upotrebljava se kada broj ponavljanja nije unaprijed poznat
- uvjet se provjerava prije izvođenja tijela petlje

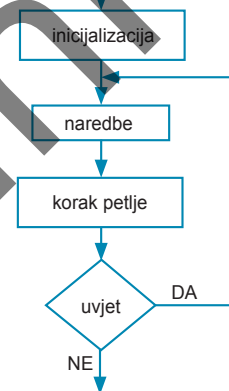


```
while (uvjet) {
    naredba_1;
    naredba_2;
    ...
    naredba_n;
}
```

do-while petlja

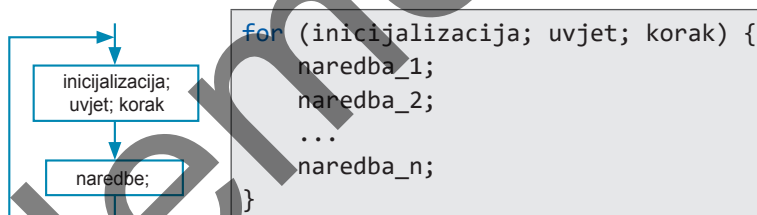
- tijelo petlje izvršava se barem jednom
- uvjet se provjerava nakon izvođenja

```
do {
    naredba_1;
    naredba_2;
    ...
    naredba_n;
} while (uvjet);
```



for petlja

- upotrebljava se kada je broj ponavljanja unaprijed poznat
- sastoji se od inicijalizacije, uvjeta i koraka petlje



```
for (inicijalizacija; uvjet; korak) {
    naredba_1;
    naredba_2;
    ...
    naredba_n;
}
```

Ugniježdene petlje

- petlja unutar druge petlje
- upotrebljava se za rad s više razina ponavljanja (npr. tablice)

Pojam beskonačne petlje

- petlja koja se nikada ne završava, a može uzrokovati pad programa
- upotrebljava se u programima koji se stalno izvođe dok korisnik ne prekine rad: `for (;;) { ... }` ili `while (1) { ... }`

Primjena naredbi `break` i `continue`

- `break` – prekida izvođenje petlje
- `continue` – preskače ostatak trenutačne iteracije

Primjena operatora zarez (,) u `for` petlji

- omogućuje izvođenje više izraza u `for` petlji, primjenjuje se u inicijalizaciji i koraku petlje

PITANJA ZA PONAVLJANJE

KAKO UPOTRIJEBITI SEMAFOR

- ako možete riješiti 1. zadatak → ● osnovna razina
- ako možete riješiti 1. i 2. zadatak → ● srednja razina
- ako možete riješiti sve zadatke → ● napredna razina



1. Algoritmi i dijagrami toka (Raptor)

● 1. ZADATAK / OSNOVNA RAZINA

Objasnite što je algoritam te navedite načine njegova zapisa. Ukratko opišite razliku između pseudokoda i dijagrama toka. Zatim promotrite pseudokod i odredite konačne vrijednosti varijabli x i y :

```
x = 10
y = 2
x = x + 5
x = y - 2
```

● 2. ZADATAK / SREDNJA RAZINA

Zadan je izraz:

$$x = \text{round}(\text{sqrt}(16)) + \text{floor}(3.7)$$

Izračunajte vrijednost varijable x .

Nakon toga napišite pseudokod kojim će se izračunati vrijednost varijable x te ispisati „ x je veći od 5“ ako je $x > 5$, inače „ x nije veći od 5“. Na kraju nacrtajte isti algoritam u programu Raptor.

● 3. ZADATAK / NAPREDNA RAZINA

Nacrtajte dijagram toka u programu Raptor i napišite pseudokod kojim će se učitati broj, ponavljati unos dok broj nije pozitivan te na kraju ispisati drugi korijen tog broja.



2. Osnove programiranja u C-u

1. ZADATAK / OSNOVNA RAZINA

Objasnite razliku između varijable i konstante te navedite primjere. Navedite osnovne tipove podataka u jeziku C te objasnite čemu služe formati koji se primjenjuju u funkcijama za unos i ispis.

Objasnite razliku između funkcija `scanf` i `printf` te opišite kako se upotrebljavaju.

2. ZADATAK / SREDNJA RAZINA

Zadan je program:

```
#include <stdio.h>
#define A 3

int main(void) {
    int x = 5;
    float y = 2.5;
    char z = 'a';
    x = x + A;
    y = y * 2;
    printf("%d %.1f\n%c", x, y, z);
    return 0;
}
```

Odredite što će program ispisati i napišite stanje varijabli `x` i `y` nakon svake naredbe.

3. ZADATAK / NAPREDNA RAZINA

Napišite program u jeziku C koji će izračunati duljinu dijagonale monitora ako su zadane njegova širina i visina u centimetrima. Rezultat ispišite na dvije decimale u centimetrima i u inčima. Vrijedi: 1 inč = 2.54 cm.

ulazni podaci	izlazni podatci
53.1	Dijagonala monitora širine 53.10 cm i visine 29.90 cm iznosi
29.9	60.94 cm, odnosno 24.00 inča.



3. Operatori, tipovi podataka i konverzija tipova podataka

1. ZADATAK / OSNOVNA RAZINA

Objasnite koje osnovne tipove podataka poznajemo u jeziku C te kako se podatci pohranjuju u memoriji računala.

Za zadani program:

- a) izdvojite sve upotrijebljene operatore i razvrstajte ih prema tipu (tablicu prepisi u bilježnicu)

tip operatora	operatori
aritmetički	
relacijski	
logički	
pridruživanja	
inkrement/dekrement	
bitovni	
cast	

```
#include <stdio.h>
#include <stdbool.h>

int main(void) {
    int a = 5, b = 2, x = 1, y;
    float c = 3.5, e;
    char d = 'A';
    bool log;

    d = a + b * 2;
    c = c + a / b;
    x += a++;
    log = !(a < b) || (c >= b) && (a != b);
    y = a >> b;
    e = (float) a / b + c;

    printf("a = %d, b = %d\n", a, b);
    printf("c = %.2f, d = %d\n", c, d);
    printf("x = %d, y = %d\n", x, y);
    printf("e = %.2f, log = %d\n", e, log);

    return 0;
}
```

- b) za svaku varijablu navedite njezin tip, zauzeće u memoriji i način zapisa u računalu.

2. ZADATAK / SREDNJA RAZINA

Odredite što će program iz prethodnog zadatka ispisati na zaslonu računala bez pokretanja programa. Potom prepisite kôd, pokrenite program i provjerite rješenje.

3. ZADATAK / NAPREDNA RAZINA

Marica je složila kolače u kutiju u više katova. Na svakom katu kolači su raspoređeni u retke i stupce. Kad se Marica udaljila, Ivica se prišuljao i pojeo dio kolača.

Napišite program koji učitava četiri cijela broja k , r , s i n , gdje k predstavlja broj katova, r broj redaka, s broj stupaca, a n broj pojedjenih kolača. Program treba izračunati i ispisati koliki je postotak kolača ostao u kutiji, na dvije decimale.

ulazni podatci	izlazni podatci
$k = 2, r = 3, s = 4, n = 5$	Preostalo je 79.17 % kolača.



4. Grananje u programu

1. ZADATAK / OSNOVNA RAZINA

Objasnite čemu služe naredbe `if`, `if else` i `switch case` te navedite u kojim se situacijama primjenjuju. Za svaku od navedenih naredbi nacrtajte odgovarajući dijagram toka.

2. ZADATAK / SREDNJA RAZINA

Zadan je program:

- Odredite što će program ispisati.
- Napišite isti program primjenom naredbe `if else`.

```
#include <stdio.h>

int main(void) {
    int x = 2;
    switch (x) {
        case 1: case 2:
            printf("A");
            break;
        case 3:
            printf("B");
            break;
        default:
            printf("D");
    }
    return 0;
}
```

3. ZADATAK / NAPREDNA RAZINA

Tri brata imaju različite visine. Napišite program u jeziku C koji će učitati njihove visine u centimetrima i ispisati kolika je visina najvišeg brata.

ulazni podatci	izlazni podatci
175 , 182 , 168	Najviši brat ima visinu 182 cm.



5. Programske petlje

1. ZADATAK / OSNOVNA RAZINA

Objasnite čemu služe petlje u programu te navedite razliku između for, while i do-while petlje. Ukratko opišite kada se koja primjenjuje.

2. ZADATAK / SREDNJA RAZINA

Zadan je program:

- Odredite što će program ispisati.
- Prilagodite zadani program uz primjenu while petlje.
- Prilagodite zadani program uz primjenu do-while petlju.

```
#include <stdio.h>
#include <stdbool.h>

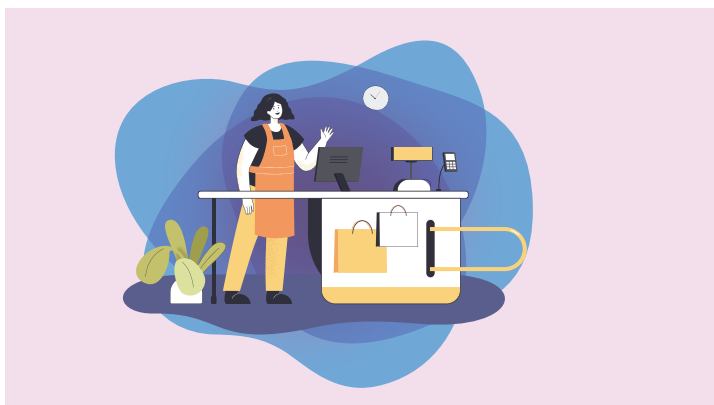
int main(void) {
    int i;
    for (i = 1; i <= 6; i++) {
        if (i % 2 == 0) {
            printf("%d ", i);
        }
    }
    return 0;
}
```

3. ZADATAK / NAPREDNA RAZINA

Prodavačica na blagajni unosi jednu po jednu cijenu proizvoda sve dok ne unese vrijednost 0 što označava da je učitala sve cijene proizvoda koje treba naplatiti.

Napišite program u jeziku C koji će omogućiti unos cijena proizvoda u eurima dok se ne unese vrijednost 0. Ako se unese negativna vrijednost, taj se unos preskače. Na kraju program ispisuje ukupan iznos računa u eurima.

ulazni podatci	izlazni podatci
12.5 , -3 , 8.0 , 5.5 , 0	Ukupan iznos proizvoda je 26.00 eura.



element.hr

II. dio

SPREMNIČKI TIPOVI PODATAKA

6. Uvod u programski jezik C++
7. Polja (nizovi)
8. Funkcije, klase i metode u C++
9. Vektori
10. Liste
11. Stog
12. Red

3 CSVET

Potrebni alati za rad:

- Code::Blocks
 - Visual Studio Code (VS Code) – opcionalno razvojno okruženje
 - GCC compiler (MinGW)
 - GitHub Copilot i GPT – alati umjetne inteligencije za pomoć pri učenju i programiranju (dodatak)
- Instalacijske upute i poveznice za preuzimanje programa nalaze se u odgovarajućim poglavljima.

6.

UVOD U PROGRAMSKI JEZIK C++



DIGITALNI RUDNIK KODOVA

Skeniraj QR kôd ili otvori poveznicu u internetskom pregledniku. Prouči primjere 6.2, 6.3 i 6.4 te ih analiziraj i proširi programe prateći upute iz tih primjera.

<https://s.element.hr/d9ea>





Razmisli

Ako već poznaješ programski jezik C, čemu potreba za učenjem novog programskog jezika poput C++?

C C++

Nakon ove lekcije moći ćeš



- primjenjivati `cin` i `cout` za unos i ispis podataka
- pravilno napisati osnovnu strukturu C++ programa
- koristiti se osnovnim tipovima podataka (`int`, `double`, `char`)
- uključiti potrebne biblioteke (`iostream`, `iomanip`, `cmath`)
- koristiti se osnovnim matematičkim funkcijama
- oblikovati ispis s pomoću `setw`, `fixed` i `setprecision`
- pisati jednostavne programe s unosom i ispisom podataka.



Programski jezik **C++ razvijen je kao nadogradnja programskog jezika C** pri čemu oba jezika imaju vrlo sličnu sintaksu. To omogućuje jednostavan prijelaz s jezika C na C++ jer se već poznate strukture i pravila i dalje primjenjuju.

Osim što zadržava mogućnosti jezika C, C++ uvodi dodatne značajke koje olakšavaju rad s podacima i strukturiranje programa. Osnovna razlika između ovih programskih jezika jest u tome što C++ podržava rad s **klasama i objektima**, dok programski jezik C ne uključuje takve mogućnosti. Time C++ omogućuje drukčiji način organizacije programa i razvoj složenijih programskih rješenja.

Danas se C++ ubraja među najraširenije programske jezike. Primjenjuje se u razvoju različitih vrsta aplikacija, poput operacijskih sustava, grafičkih sučelja i ugrađenih sustava. Njegova velika prednost je što omogućuje dobru kontrolu nad memorijom, ali istovremeno nudi i alate koji pojednostavljaju programiranje.

U prethodnom dijelu upoznali smo programski jezik C i naučili raditi s osnovnim tipovima podataka, uvjetima i petljama. Stečena znanja omogućuju rješavanje jednostavnijih problema i predstavljaju temelj za razvoj složenijih programskih rješenja te rad u naprednijim programskim jezicima.

U nastavku se bavimo **složenijim strukturama podataka** kao što su:

- polje (niz, engl. *array*)
- vektor (engl. *vector*)
- lista (engl. *list*)
- stog (engl. *stack*, LIFO)
- red (engl. *queue*, FIFO).

Za rad s takvim strukturama podataka programski jezik C zahtijeva pisanje većeg broja dodatnih naredbi te ručno upravljanje memorijom što početnicima može biti zahtjevno i često dovodi do pogrešaka. Zbog toga uvodimo programski jezik C++ koji nudi gotove i jednostavnije načine rada s tim strukturama. Na taj način možemo se više usredotočiti na razumijevanje problema i algoritma, a manje na tehničke detalje implementacije.

Kako bismo lakše uočili razlike između programskih jezika C i C++, u nastavku tablica 6.1 prikazuje njihovu usporedbu prema nekim od poznatijih obilježja.

Tablica 6.1 Usporedba osnovnih obilježja programskih jezika C i C++

obilježje	C	C++
pristup programiranju	proceduralni	proceduralni objektno orijentirani
unos i ispis	scanf, printf	cin, cout
biblioteke za rad s funkcijama za upis i ispis	stdio.h	iostream
organizacija koda	funkcije	funkcije + klase
ekstenzija datoteka	.c	.cpp
primjena	izrada programa koji upravljaju radom računala i operacijskog sustava	aplikacije, igre, složeni sustavi

6.1. Prvi program u programskom jeziku C++

6.1.1. Kreiranje i pokretanje prvog C++ programa u Code::Blocks okruženju

U razvojnim okruženjima poput Code::Blocksa programi se organiziraju unutar **projekata**. Projekt predstavlja skup datoteka i postavki potrebnih za izradu programa te omogućuje lakše upravljanje programskim kodom.

Slijedi prikaz postupka kreiranja novog projekta i pokretanja C++ programa.

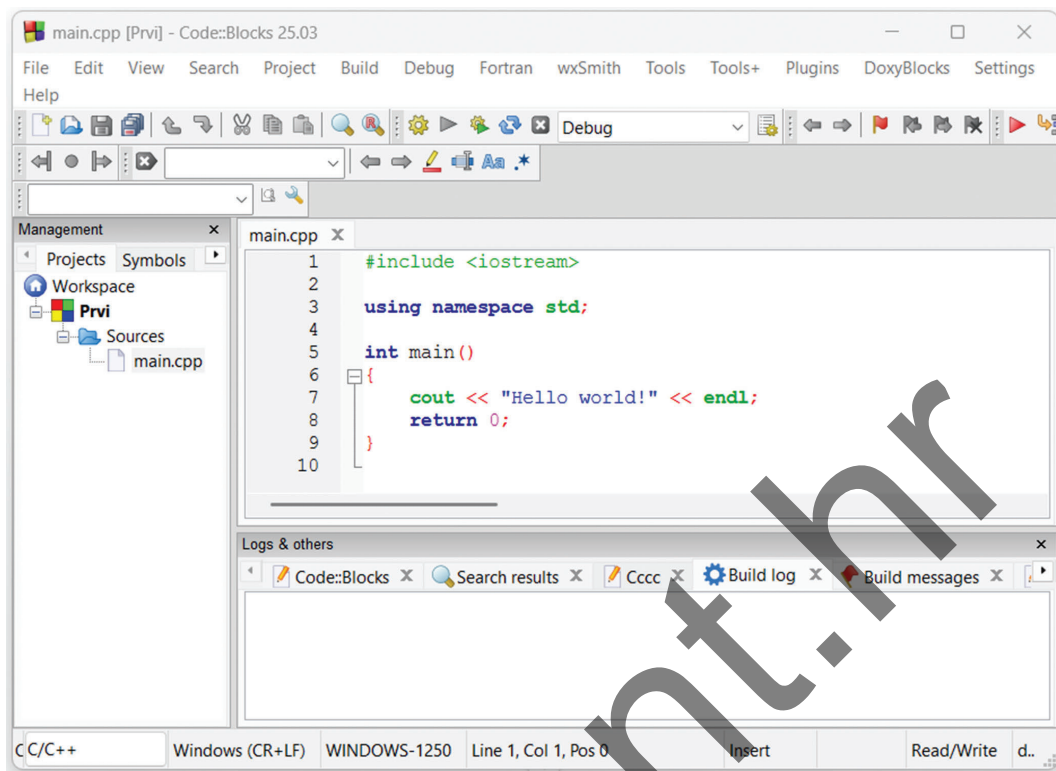


Kreiranje novog projekta (*Create a New Project*)

1. Pokrenite **Code::Blocks** dvostrukim klikom na ikonu.
2. U početnom prozoru odaberite opciju **Create a new project**.
3. U prikazanom izborniku odaberite **Console application**.
4. Kliknite gumb **Go**.
5. U dijelu **Please select language** odaberite **C++**.
6. Ponovno kliknite **Next**.
7. U polje **Project title** upišite naziv projekta.
8. U polju **Folder to create project in** odaberite mapu u koju će se projekt spremiti.
9. Kliknite **Next**.
10. Na kraju kliknite **Finish**.
11. U lijevom dijelu prozora (Projects) pronađite datoteku **main.cpp**.
12. Dvostrukim klikom otvorite datoteku.

Klikom na **main.cpp** prikazat će se kôd kao na slici 6.1.

6. Uvod u programski jezik C++



Slika 6.1. Pokretanje prvog programa u programskom jeziku C++

U datoteci main.cpp nalazi se sljedeći program:

```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello world!" << endl;
    return 0;
}
```

Kompajliranje i pokretanje programa (Build & Run)

Za izvođenje programa potrebno je najprije prevesti izvorni kod, a zatim pokrenuti dobiveni izvršni program.

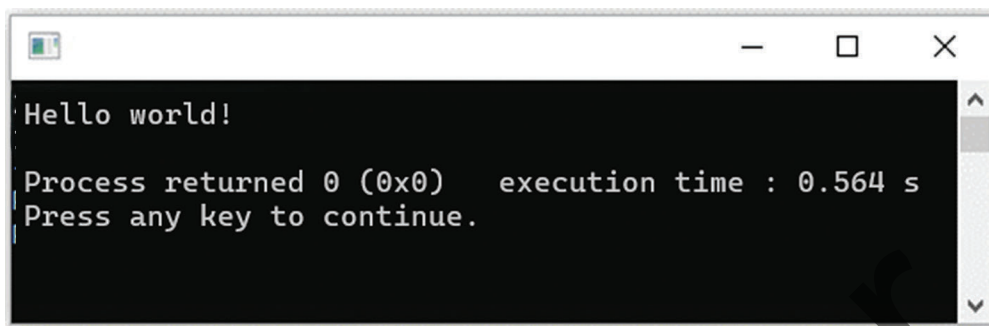
To se može napraviti u dva koraka:

1. U izborniku odaberite **Build** → **Build** (kompajliranje programa).
2. Zatim odaberite **Build** → **Run** (pokretanje programa).

Pokretanje programa moguće je napraviti i pritiskom na tipku **F9 (Build & Run)** čime se oba koraka izvršavaju odjednom.

Rezultat izvođenja

Otvorit će se novi prozor (konzola) u kojem će se ispisati "Hello world!" (slika 6.2).



Slika 6.2. Ispis "Hello world!" u novom prozoru

Napomena

Postupak pokretanja programa u C++-u vrlo je sličan postupku u programskom jeziku C, ali se C++ programi razlikuju po ekstenziji .cpp i upotrebi drukčijih biblioteka, primjerice `iostream`.

U nastavku tablica 6.2 objašnjava ulogu pojedinih dijelova programa.

Tablica 6.2. Objašnjenje dijelova C++ programa

dio programa	objašnjenje
<code>#include <iostream></code>	Uključuje biblioteku za unos i ispis podataka (<code>cout</code> , <code>cin</code>).
<code>using namespace std;</code>	Omogućuje korištenje <code>cout</code> i <code>cin</code> bez pisanja <code>std::</code> .
<code>int main()</code>	Glavna funkcija u kojoj započinje izvođenje programa.
<code>{ }</code>	Označavaju početak i kraj funkcije.
<code>cout << "Hello world!" << endl;</code>	Ispisuje tekst na zaslon. Operator <code><<</code> šalje podatke na izlaz, a <code>endl</code> prelazi u novi redak.
<code>return 0;</code>	Označava uspješan završetak programa.
<code>;</code>	Svaka naredba u C++-u završava točkom sa zarezom.
prazni redci	Ne utječu na izvođenje programa, već služe za preglednost koda.
velika i mala slova	C++ razlikuje velika i mala slova (<code>cout</code> ≠ <code>Cout</code>).

6.1.2. Upotreba imenskog prostora (*namespace*)

U C++ programima često se upotrebljava naredba:

```
using namespace std;
```

Ona omogućuje upotrebu naredbi poput `cout` i `cin` bez dodatnog prefiksa. Naime, bez naredbe `using namespace std;` potrebno je ispred naziva `cout` i `cin` napisati oznaku `std::`, primjerice `std::cout` i `std::cin`. Oznaka `std` predstavlja imenski prostor (*namespace*) standardne biblioteke jezika C++, a zapis `std::` označava upotrebu elementa iz tog imenskog prostora.

Međutim, isti program možemo napisati i bez naredbe `using namespace std;` tako da ispred naziva dodamo oznaku `std::`.

Primjer bez `using namespace std`

```
#include <iostream>

int main() {
    std::cout << "Hello world!";
    return 0;
}
```

- `std::cout` znači da upotrebljavamo `cout` iz standardne biblioteke
- `std::` naziva se **imenski prostor** (*namespace*)

U ovom udžbeniku upotrebljavat ćemo `using namespace std;` jer je zapis kraći i pregledniji, što je prikladnije za početno učenje. U složenijim programima često se odabire zapis s `std::` kako bi se jasno naznačilo odakle dolaze pojedini elementi.

6.2. Povezanost programskih jezika C i C++

Slijedi prikaz usporedbe osnovnih elemenata programskog jezika C i C++ **na temelju onog s čime ste se dosad susreli u programskom jeziku C**. Većina koncepata u C++-u temelji se na onima iz jezika C. Tablica 6.3 prikazuje sličnosti, dok su u tablici 6.4 navedene razlike.

Tablica 6.3. Sličnosti između programskih jezika C i C++

element	zajedničko u programskim jezicima C i C++
struktura programa	<pre>#include <biblioteka> int main() { // deklaracije varijabli // naredbe (unos, obrada, ispis) return 0; }</pre>
komentari	<pre>// linijski /*kroz više redaka*/</pre>

deklaracija i inicijalizacija varijable	<pre>int a = 5; float x, y; char p = 'a';</pre>
tipovi podataka	• cjelobrojni tipovi: char, int, _Bool (bool)¹ • brojevi s pomičnim zarezom: float, double
operatori	• pridruživanja (=) • aritmetički (+, -, *, /, %) • relacijski (==, !=, >, <, >=, <=) • logički (&&, , !) • inkrement/dekrement (++ , --) • sizeof • bitovni (&, , ^, >>, <<)²
grananje	• if-else • switch-case
petlje	• for • while • do-while

Tablica 6.4. Razlike između programskih jezika C i C++

element	C	C++
ispis podataka	printf()	cout <<
unos podataka	scanf()	cin >>
biblioteke	stdio.h	iostream
logičke vrijednosti	0 i 1	true, false
pretvorba tipova	(tip) izraz	static_cast<tip>(izraz)
organizacija programa	funkcije	funkcije + klase
• vektor (engl. <i>vector</i>) • lista (engl. <i>list</i>) • stog (engl. <i>stack</i>, LIFO) • red (engl. <i>queue</i>, FIFO)	nema gotovih struktura, sve se implementira ručno	gotove strukture i funkcije

¹ U jeziku C upotrebljava se tip `_Bool`, a radi jednostavnosti se često upotrebljava `bool` iz biblioteke `<stdbool.h>`. U jeziku C++ tip `bool` je ugrađen i upotrebljava se bez dodatnih biblioteka.

² `<< i >>` u C++ se upotrebljavaju i za ulaz/izlaz (`cout` i `cin`).

6.3. Unos i ispis podataka u programskom jeziku C++

U programskom jeziku C unos i ispis podataka ostvarivali smo s pomoću funkcija `scanf()` i `printf()`. U C++-u za istu svrhu upotrebljavaju se naredbe:

- `cin` za unos podataka
- `cout` za ispis podataka

Ove naredbe nalaze se u standardnoj biblioteci `iostream`.

6.3.1. Ispis podataka (`cout`)

Za ispis podataka na zaslon u programskom jeziku C++ upotrebljava se naredba `cout` koja se nalazi u standardnoj biblioteci `iostream`.

Podatci se šalju na izlaz s pomoću operatora `<<`, kojim se vrijednosti (tekst, brojevi i varijable) prosljeđuju naredbi `cout`. Tekst koji želimo ispisati piše se unutar navodnika " ".

S pomoću operatora `<<` moguće je ispisati više podataka u jednom retku, kombinirajući tekst i vrijednosti varijabli.

Za prijelaz u novi redak upotrebljava se naredba `endl`, koja se također prosljeđuje s pomoću operatora `<<`.

Pregled osnovnih elemenata za ispis podataka:

- `cout` – naredba za ispis podataka na zaslon
- `<<` – operator koji šalje podatke na izlaz (prema zaslonu)
- tekst se piše unutar navodnika " "
- `endl` – označava prijelaz u novi redak

Opći zapis za ispis podataka (`cout`):

```
cout << podatak1 << podatak2 << ... << podatakN;
```

gdje su `podatak1`, `podatak2`, ...:

- tekst (u navodnicima " ")
- varijable
- izrazi.

U tablici 6.5 prikazani su različiti načini ispisa podataka s pomoću naredbe `cout`, ovisno o broju i tipu varijabli.

Pretpostavlja se da su varijable deklarirane i inicijalizirane na sljedeći način:

```
int a = 5;  
float b = 10.5;  
char c = 'A';
```

Tablica 6.5. Primjeri ispisa jedne ili više vrijednosti s pomoću naredbe `cout`

broj ispisa	primjer koda	rezultat ispisa	objašnjenje
jedna vrijednost	<code>cout << a;</code>	5	ispisuje vrijednost varijable <code>a</code>
dvije vrijednosti	<code>cout << a << b;</code> <code>cout << a << " " << b;</code>	510.5 5 10.5	ispisuje vrijednosti <code>a</code> i <code>b</code> redom
tri vrijednosti (novi redak)	<code>cout << a << endl << b << endl << c;</code>	5 10.5 A	svaka vrijednost ispisuje se u novi redak
tekst i varijabla	<code>cout << "Broj: " << a;</code>	Broj: 5	ispisuje tekst i vrijednost varijable
više varijabli i tekst	<code>cout << "A=" << a << "B=" << b << "C=" << c;</code>	A=5 B=10.5 C=A	kombinacija teksta i više varijabli

Primjer 6.1.

Napišite program koji će na zaslon ispisati vaše ime i prezime.

ulazni podaci	izlazni podaci
	Ime Prezime

Rješenje

```
#include <iostream>
using namespace std;

int main() {
    cout << "Ime Prezime"; // ispis teksta na zaslon
    return 0;
}
```

Objašnjenje

- `cout` – naredba za ispis podataka na zaslon
- `<<` – operator koji šalje podatke na izlaz (prema zaslonu)
- tekst se piše unutar navodnika " "
- komentar u programu započinje s `//` i služi za dodatno objašnjenje koda (ne utječe na izvođenje programa)

Primjer 6.2.

Napišite program koji ispisuje brojeve od 1 do 300. Za svaki broj potrebno je ispisati je li broj paran ili neparan.

ulazni podatci	izlazni podatci
	1 je neparan
	2 je paran
	...
	300 je paran

Rješenje

```
#include <iostream>
using namespace std;

int main() {

    // petlja prolazi kroz brojeve od 1 do 300
    for (int i = 1; i <= 300; i++) {

        // ispis broja i informacije o parnosti u istom retku
        cout << i << " je ";

        if (i % 2 == 0) {
            cout << "paran";
        } else {
            cout << "neparan";
        }

        cout << endl; // novi red nakon svakog ispisa
    }

    return 0;
}
```

Proširite program

Izračunajte i na kraju ispišite ukupan broj parnih i neparnih brojeva u rasponu od 1 do 300.



U programskom jeziku C++ varijabla u for petlji može se **deklarirati i inicijalizirati izravno unutar zaglavlja petlje**, primjerice u izrazu

```
for (int i = 1; i <= 300; i++)
```

Time se varijabla *i* uvodi samo za potrebe te petlje i upotrebljava se kao brojač ponavljanja.

Važno je naglasiti da je doseg (vidljivost) takve varijable ograničen isključivo na tijelo petlje. To znači da se varijabla *i* može upotrebljavati samo unutar petlje, dok izvan nje više nije dostupna.

6.3.2. Unos podataka (cin)

Za unos podataka s tipkovnice u programskom jeziku C++ primjenjuje se naredba *cin*, koja se nalazi u standardnoj biblioteci *iostream*.

Podatci se unose s pomoću operatora `>>`, kojim se vrijednosti s tipkovnice proslijeđuju varijablama. Unesena vrijednost pohranjuje se u varijablu koja se nalazi s desne strane operatora `>>`.

Prije unosa potrebno je deklarirati varijablu odgovarajućeg tipa u koju će se podatak spremati.

Unos je moguće proširiti na više varijabli pri čemu se operator `>>` može upotrebljavati više puta u istom izrazu.

Pregled osnovnih elemenata za unos podataka:

- `cin` – naredba za unos podataka s tipkovnice
- `>>` – operator koji prenosi uneseni podatak u varijablu
- varijabla – mjesto u memoriji u koje se sprema unesena vrijednost

Opći zapis za unos podataka (`cin`):

```
cin >> varijabla1 >> varijabla2 >> ... >> varijablaN;
```

gdje su `varijabla1`, `varijabla2`, ... varijable u koje se spremaju unesene vrijednosti.

Tablica 6.6 prikazuje različite načine unosa podataka s pomoću naredbe `cin`, ovisno o broju i tipu varijabli.

Pretpostavlja se da su varijable deklarirane na sljedeći način:

```
int a;
float b;
char c;
```

Tablica 6.6. Primjeri ispisa jedne ili više vrijednosti s pomoću naredbe `cout`

broj unosa	primjer koda	unos korisnika	rezultat (vrijednosti varijabli)	objašnjenje
jedna vrijednost	<code>cin >> a;</code>	5	<code>a = 5</code>	unosi se cijeli broj u varijablu <code>a</code>
dvije vrijednosti	<code>cin >> a >> b;</code>	5 10.5	<code>a = 5</code> <code>b = 10.5</code>	prva vrijednost ide u <code>a</code> , druga u <code>b</code>
tri vrijednosti	<code>cin >> a >> b >> c;</code>	5 10.5 A	<code>a = 5</code> <code>b = 10.5</code> <code>c = 'A'</code>	vrijednosti se unose redom u varijable
unos znaka	<code>cin >> c;</code>	B	<code>c = 'B'</code>	unosi se jedan znak
ponovljeni unos	<code>cin >> a;</code> <code>cin >> b;</code>	7 3.2	<code>a = 7</code> <code>b = 3.2</code>	unos u dva koraka

Primjer 6.3.

Napišite program koji učitava dva broja i operator (+ ili -). Ovisno o unesenom operatoru, program treba izračunati i ispisati zbroj ili razliku brojeva. Ako je unesen neki drugi znak, potrebno je ispisati poruku o pogrešnom unosu.

ulazni podatci	izlazni podatci
5 3 +	Zbroj je: 8
5 3 -	Razlika je: 2
5 3 *	Pogresan unos operatora!

Rješenje

```
#include <iostream>
using namespace std;

int main() {
    int a, b;
    char op;

    cout << "Unesite dva broja: ";
    cin >> a >> b;

    cout << "Unesite operator (+ ili -): ";
    cin >> op;

    switch (op) {
        case '+':
            cout << "Zbroj je: " << a + b;
            break;

        case '-':
            cout << "Razlika je: " << a - b;
            break;

        default:
            cout << "Pogresan unos operatora!";
    }

    return 0;
}
```

Proširite program

Dodajte podršku za množenje i dijeljenje, a kod dijeljenja provjerite dijeli li se s nulom i u tom slučaju ispišite odgovarajuću poruku o pogrešci.



6.4. Standardne biblioteke u C++-u

U programskom jeziku C++ mnoge funkcionalnosti dostupne su putem standardnih biblioteka. One sadrže unaprijed definirane funkcije i naredbe koje olakšavaju rad s podatcima, matematičkim operacijama i ispisom rezultata.

U ovom poglavlju upoznat ćemo neke od najčešće korištenih biblioteka:

- `iostream` – za unos i ispis podataka
- `cmath` – za matematičke funkcije
- `iomanip` – za oblikovanje ispisa

U tablici 6.7 nalazi se pregled osnovnih funkcija i naredbi iz navedenih biblioteka.

Tablica 6.7. Pregled osnovnih funkcija i naredbi

biblioteka	funkcija / naredba	opći zapis	opis
<code><iostream></code>	ispis	<code>cout << podatak;</code>	ispisuje podatke na zaslon
<code><iostream></code>	unos	<code>cin >> varijabla;</code>	unos podataka s tipkovnice
<code><iostream></code>	novi redak	<code>cout << endl;</code>	prelazak u novi redak
<code><cmath></code>	kvadratni korijen	<code>sqrt(x)</code>	vraća kvadratni korijen broja
<code><cmath></code>	potenciranje	<code>pow(x, y)</code>	vraća x na potenciju y
<code><cmath></code>	apsolutna vrijednost	<code>abs(x)</code>	vraća apsolutnu vrijednost
<code><iomanip></code>	širina ispisa	<code>setw(n)</code>	postavlja širinu ispisa
<code><iomanip></code>	decimalni zapis	<code>fixed</code>	prikaz broja u decimalnom obliku
<code><iomanip></code>	broj decimala	<code>setprecision(n)</code>	određuje broj znamenaka

6.4.1. Oblikovanje ispisa u C++-u

Pri ispisu numeričkih podataka često je potrebno dodatno oblikovati rezultat kako bi bio pregledniji i čitljiviji. U programskom jeziku C++ to se postiže primjenom funkcija iz biblioteke `iomanip`.

Ove funkcije omogućuju:

- određivanje širine ispisa
- kontrolu broja decimalnih mjesta.

Funkcije za oblikovanje ispisa upotrebljavaju se s naredbom `cout` i povezuju se operatorom `<<`.

Opći zapis oblikovanog ispisa je:

```
cout << postavke << podatak;
```

gdje su:

- **postavke** – funkcije koje određuju izgled ispisa
- **podatak** – vrijednost koja se ispisuje

Postavke se navode prije podatka na koji se primjenjuju.

Najčešće upotrebljavane funkcije su:

- `setw(n)` – određuje širinu ispisa
- `fixed` – postavlja decimalni zapis broja
- `setprecision(n)` – određuje broj decimalnih mjesta.

Funkcije `fixed` i `setprecision(n)` često se upotrebljavaju zajedno kako bi se postigao ispis realnih brojeva na željeni broj decimalnih mjesta.

Primjer 6.4.

Odredite kako će izgledati ispis na zaslonu računala.

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main() {
    char znak = 'A';
    int a = 7, b = 3;
    float f = 5.5;
    double d = 2.25;
    bool log;

    a = a + b;
    f = static_cast<float>(a) / b;
    d = d * f + 1.75;
    znak = znak + 2;
    log = (a > b) && (f > 3);
    b = abs(b - a);

    // ispisi
    cout << setw(5) << a << endl;
    cout << setw(8) << fixed << setprecision(2) << f << endl;
    cout << setw(10) << fixed << setprecision(2) << d << endl;
    cout << znak << endl;
    cout << log << endl;
    cout << setw(5) << b << endl;

    return 0;
}
```

Rješenje

Rješenje prikazuju tablice 6.8 i 6.9.

Tablica 6.8. Pregled osnovnih funkcija i naredbi

naredba	a	b	f	d	znak	log	objašnjenje
početno stanje	7	3	5.5	2.25	'A'	?	varijable dobivaju početne vrijednosti
a = a + b;	10						vrijednost a postaje 7 + 3 = 10
f = static_cast<float>(a) / b;			3.33				pretvorbom vrijednosti a u tip float dijeljenje daje realan rezultat 3.333...
d = d * f + 1.75;				9.25			računa se $2.25 * 3.333... + 1.75 = 9.25$
znak = znak + 2;					'C'		znak 'A' pomiče se za 2 mjesta u ASCII tablici pa postaje 'C'
log = (a > b) && (f > 3);						true	izrazi a > b i f > 3 su istiniti pa je log = true
b = abs(b - a);		7					računa se apsolutna vrijednost izraza 3 - 10, pa je b = 7

Tablica 6.9. Objašnjenje ispisa na zaslonu računala

pregled ispisa na zaslonu računala	dodatna objašnjenja
<pre> 10 3.33 9.25 C 1 7 </pre>	Varijable a ima vrijednost 10. setw(5) znači da se ispis zauzima u širini od 5 mjesta. Broj se poravnava udesno.
	Varijable f ima vrijednost približno 3.333.... setprecision(2) → ispisuje 2 decimale → 3.33 fixed → koristi fiksni decimalni zapis setw(8) → ukupna širina ispisa je 8 znakova Budući da 3.33 zauzima 4 znaka, dodaju se 4 prazna mjesta s lijeve strane.
	Varijable d ima vrijednost 9.25. Ispisuje se s dvije decimale i zauzima širinu od 10 mjesta.
	Varijable znak ima vrijednost 'C'.
	Varijable log ima vrijednost true. Kod ispisa tipa bool, C++ standardno ispisuje 1 za true, a 0 za false.
	Varijable b ima vrijednost 7. setw(5) poravnava broj udesno u širini od 5 mjesta.

Proširite program

umjesto unaprijed definiranih vrijednosti pri deklaraciji i inicijalizaciji učitajte vrijednosti za sve varijable s tipkovnice te testirajte program s različitim ulaznim podacima.



Primjer 6.5.

Napišite program koji učitava duljine kateta pravokutnog trokuta i računa duljinu hipotenuze. Rezultat ispišite na širini od 10 znakova uz dvije decimale.

ulazni podatci	izlazni podatci
3 4	Duljina hipotenuze je: 5.00

Rješenje

```
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;

int main() {
    double a, b, c;

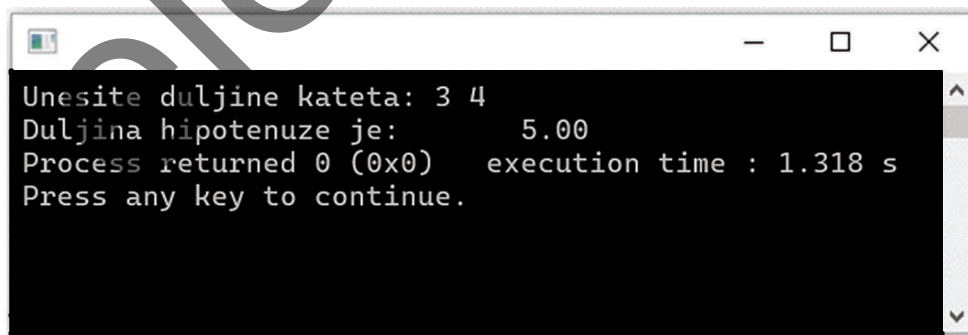
    cout << "Unesite duljine kateta: ";
    cin >> a >> b;

    // Pitagorin poučak
    c = sqrt(pow(a, 2) + pow(b, 2));

    // ispis s formatiranjem
    cout << "Duljina hipotenuze je: "
         << setw(10) << fixed << setprecision(2) << c;

    return 0;
}
```

Pokretanjem programa dobit ćemo prikaz kao na slici 6.3.



Slika 6.3. Ispis na zaslonu računala

Tekst "Duljina hipotenuze je: " ispisuje se bez promjena. Varijabla *c* predstavlja duljinu hipotenuze (npr. 5).

- `fixed` → upotrebljava se decimalni zapis
- `setprecision(2)` → ispis na 2 decimale → 5.00
- `setw(10)` → broj 5.00 zauzima ukupno 10 mjesta

Budući da 5.00 ima 4 znaka, dodaje se 6 praznih mjesta s lijeve strane.

6.5. Jednostavni i složeni tipovi podataka

U programiranju upotrebljavamo tipove podataka za spremanje različitih vrijednosti. Prema složenosti tipove podataka dijelimo na:

- jednostavne tipove podataka
- složene tipove podataka.

6.5.1. Jednostavni tipovi podataka

Pregled **jednostavnih tipova podataka** u programskom jeziku C++ prikazuje tablica 6.10.

Tablica 6.10. Jednostavni tipovi podataka u C++

tip	opis	primjer	vrsta
bool	logička vrijednost (true ili false)	bool log = true;	cjelobrojni tipovi
char	jedan znak	char c = 'A';	
int	cijeli broj	int a = 5;	
float	realni broj	float f = 3.14;	brojevi s pomičnim zarezom
double	realni broj veće preciznosti	double d = 3.14159;	

Na slici 6.4 prikazuje se rad programa s pomoću *online* alata **Python Tutor** (<https://pythontutor.com/visualize.html>).

Alat omogućuje praćenje izvođenja programa korak po korak te prikazuje trenutno stanje varijabla u memoriji.

The image shows a screenshot of the Python Tutor online tool. On the left, the C++ code is displayed with line numbers 1 through 13. The code includes headers, uses the std namespace, and defines several variables in the main function. On the right, the memory stack is visualized, showing the current values of the variables: log is 1, c is 'A', a is 5, f is 3.14, and d is 3.14159. The tool is set to C/C++ and the default view is selected.

```

C++ (C++20 + GNU extensions)
known limitations
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     bool log = true;
7     char c = 'A';
8     int a = 5;
9     float f = 3.14;
10    double d = 3.14159;
11
12    return 0;
13 }

```

Stack

main	
log	bool 1
c	char 'A'
a	int 5
f	float 3.14
d	double 3.14159

C/C++ details: none [default view] ▼

[Edit this code](#)

Slika 6.4. Prikaz stanja varijabla tijekom izvođenja programa

Skeniranjem QR koda moguće je pratiti izvođenje programa korak po korak u alatu *Python Tutor*.



6.5.2. Podatak tipa string

Novost u C++-u je `string`. `String` je tip podatka koji služi za rad s tekstom, a upotrebljava se kada želimo spremiti na primjer neku riječ ili rečenicu.

Opći zapis definicije

```
string varijabla = "tekst";
```

Učitavanje i ispis stringa

U programima često trebamo unijeti tekst s tipkovnice i zatim ga ispisati na zaslonu računala. Za unos podataka upotrebljavamo naredbu `cin`, a za ispis naredbu `cout`.

Učitavanje stringa

Za unos jedne riječi upotrebljavamo naredbu:

```
cin >> varijabla;
```

Ova naredba čita unos s tipkovnice i sprema ga u varijablu.

Unos završava kada korisnik upiše razmak ili pritisne tipku Enter. **To znači da se učitava samo jedna riječ.** Ako korisnik unese više riječi, naredba `cin >>` spremi će samo prvu riječ.

Ako želimo unijeti više riječi, upotrebljavamo naredbu:

```
getline(cin, varijabla);
```

Ova naredba učitava cijeli red, uključujući razmake.

Napomena: Ako prije `getline` upotrebljavamo `cin >>`, potrebno je dodati `cin.ignore()` kako bi unos radio ispravno.

Ispis stringa

Za ispis koristimo naredbu:

```
cout << varijabla;
```

Ona ispisuje sadržaj varijable na zaslonu računala.

Primjer 6.6.

Napišite program koji:

- učitava ime (jedna riječ)
- učitava rečenicu (više riječi)
- ispisuje unesene podatke u zadanom formatu.

ulazni podatci	izlazni podatci
Ana Volim programiranje	Ime je: Ana Rečenica je: Volim programiranje

Rješenje

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

int main() {
    string ime;
    string recenica;

    cout << "Unesi ime: ";
    cin >> ime;

    cout << "Unesi recenicu: ";
    cin.ignore();
    getline(cin, recenica);

    cout << "Ime je: " << ime << endl;
    cout << "Recenica je: " << recenica << endl;

    return 0;
}
```

6.5.3. Složeni tipovi podataka

Jednostavni i složeni tipovi podataka razlikuju se prema količini podataka koje mogu pohraniti. **Jednostavni tipovi** služe za spremanje jedne vrijednosti, kao što su **broj** ili **znak**. S druge strane, **složeni tipovi** omogućuju spremanje i organizaciju više međusobno povezanih vrijednosti unutar jedne cjeline.

Tablica 6.11 prikazuje u kojim situacijama upotrebljavamo jednostavne, a u kojima složene tipove podataka, ovisno o količini i organizaciji podataka.

Tablica 6.11. Kada upotrebljavati jednostavne, a kada složene tipove podataka

situacija	tip podataka	objašnjenje
spremamo jednu vrijednost (broj, znak, logičku vrijednost)	jednostavni tipovi	odabiru se kada je potreban samo jedan podatak
radimo s pojedinačnim varijablama	jednostavni tipovi	svaka varijabla sadrži jednu vrijednost
spremamo više vrijednosti iste vrste	složeni tipovi	podatci se organiziraju u jednu cjelinu
radimo s većim skupom podataka	složeni tipovi	omogućuju lakše upravljanje većim brojem vrijednosti
podatci su međusobno povezani	složeni tipovi	podatci se grupiraju i tako upotrebljavaju
trebamo tablični prikaz podataka	složeni tipovi	upotrebljavaju se polja ili slične strukture

Složeni tipovi podataka prema načinu organizacije podataka dijele se na **linearne** (engl. **linear data structures**) i **nelinearne** strukture podataka.

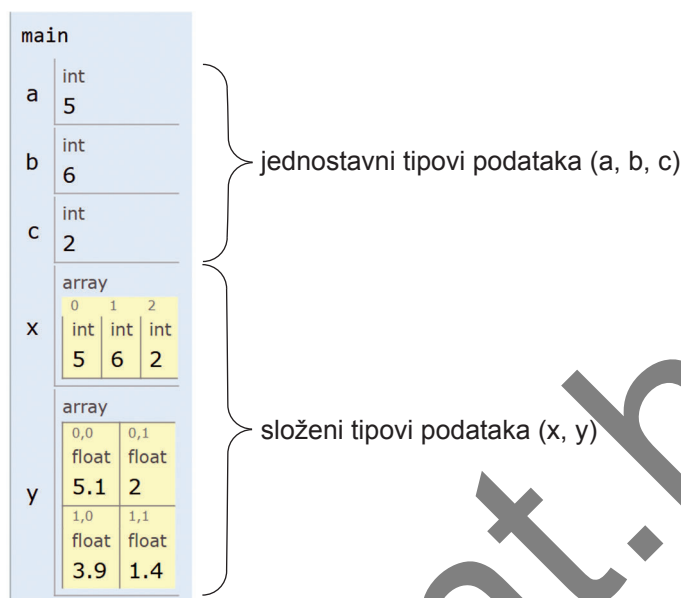
U **linearnim strukturama** podatci su organizirani slijedno, odnosno jedan iza drugoga. U linearne strukture podataka ubrajamo:

- polja, odnosno nizovi (engl. *arrays*)
- vektori (engl. *vectors*)
- liste (engl. *lists*)
- stogovi (engl. *stacks*)
- redovi (engl. *queues*).

Nelinearne strukture podataka organiziraju podatke tako da između elemenata može postojati više međusobnih veza. Primjeri nelinearnih struktura podataka su stabla (engl. *trees*) i grafovi (engl. *graphs*).

U C++-u mnoge su strukture podataka već implementirane i dostupne putem **Standard Template Library (STL)**. **STL** omogućuje jednostavniju upotrebu gotovih struktura podataka bez potrebe za njihovom vlastitom implementacijom. Ove ćemo strukture upoznavati postupno, od jednostavnijih prema složenijima.

Na slici 6.5 prikazan je način pohrane jednostavnih i složenih tipova podataka u memoriji računala.



Slika 6.5. Usporedba pohrane jednostavnih i složenih tipova podataka u memoriji računala

U ovom poglavlju upoznali smo osnovnu ideju složenih tipova podataka. U nastavku ćemo detaljnije obraditi pojedine strukture, počevši od polja.



VRIJEME ZA IZAZOV

Koliko si čaša vode danas popio/la?

Radna situacija

Učenik/ca tijekom dana bilježi koliko je čaša vode popio/la.. Program treba izračunati ukupan broj popijenih čaša i provjeriti je li učenik/ca ostvario/la preporučeni dnevni unos.



Zadatak

1. Napiši program koji omogućuje unos preporučenog broja čaša vode za jedan dan.
2. Program zatim omogućuje unos broja čaša vode za **5 dijelova dana**: jutro, škola, poslijepodne, trening/slobodno vrijeme i večer.
3. Nakon svakog unosa program zbraja ukupan broj čaša.
4. Ako je unesena negativna vrijednost, unos se ne zbraja i treba ponoviti unos za taj dio dana.
5. Na kraju program ispisuje jednu od poruka:
 - „Cilj ostvaren! Ukupno popijeno: [broj] čaša.”
 - „Cilj nije ostvaren. Ukupno popijeno: [broj] čaša. Nedostaje još: [broj] čaša.”

Primjer ulaznih i izlaznih podataka

ulazni podatci	izlazni podatci
8 2, 2, 1, 2, 1	Cilj ostvaren! Ukupno popijeno: 8 čaša.
10 1, 2, 1, 2, 1	Cilj nije ostvaren. Ukupno popijeno: 7 čaša. Nedostaje još: 3 čaša.
8 2, -1, 3, 1, 1, 1	Cilj ostvaren! Ukupno popijeno: 8 čaša.

Elementi vrednovanja – program i pseudokod

elementi provjere	bodovi
unos preporučenog broja čaša	3
unos podataka s pomoću petlje	5
provjera negativnog unosa	4
ispravno zbrajanje ukupnog broja čaša	5
ispravna primjena grananja za provjeru cilja	5
točan ispis odgovarajuće poruke	4
uporaba cin i cout	2
preglednost i struktura programa	2
ukupno	30

Ocjena

- 27 – 30 = odličan (5)
- 23 – 26 = vrlo dobar (4)
- 19 – 22 = dobar (3)
- 15 – 18 = dovoljan (2)
- 0 – 14 = nedovoljan (1)

Dodatni zadatak za one koji žele znati više

Proširi program tako da osim ukupnog broja čaša vode ispisuje i **najveći uneseni broj čaša vode**.

Na kraju ispiši poruku:

„Najveći uneseni broj čaša vode u jednom dijelu dana je: [broj].”

Vrednovanje kao učenje



<https://s.element.hr/7709>

7.

POLJA (NIZOVI)

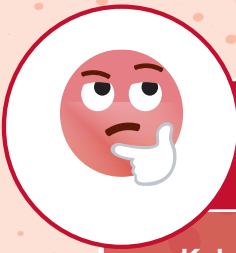


DIGITALNI RUDNIK KODOVA

Skeniraj QR kôd ili otvori poveznicu u internetskom pregledniku. Prouči primjere 7.3 i 7.5 te ih analiziraj i proširi programe prateći upute iz tih primjera.

<https://s.element.hr/7046>





Razmisli

Kako se na terenu razlikuju nogometaši koji igraju za isti tim? Kako pomoćni sudac pokazuje zamjenu igrača? Kako se točno zna koji igrač izlazi, a koji ulazi u igru? Kako bi se takav način označavanja mogao primijeniti u programiranju pri radu s više podataka istog tipa?



Nakon ove lekcije moći će



- objasniti što je polje i čemu služi
- razlikovati jednodimenzionalna i dvodimenzionalna polja
- deklarirati i inicijalizirati polja
- koristiti se indeksom za pristup elementima polja
- unositi i ispisivati elemente polja pomoću petlji
- koristiti se ugniježdenim petljama za rad s dvodimenzionalnim poljima
- primijeniti polja u rješavanju jednostavnih problemskih zadataka.



7. Polja (nizovi)

U programima često trebamo raditi s većim brojem podataka iste vrste.

Umjesto da za svaki podatak definiramo posebnu varijablu, odabrat ćemo **polje**.

Polje (niz, engl. *array*) složeni je tip podataka koji omogućuje spremanje više vrijednosti istog tipa unutar jedne varijable.

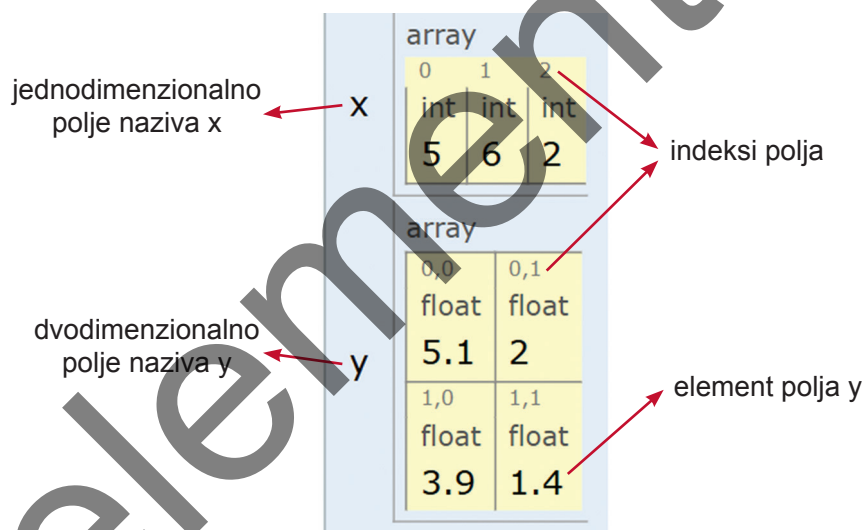
Elementi polja:

- imaju **isti tip** podatka
- poredani su **jedan za drugim** u memoriji
- razlikuju se prema **indeksu** (indeks prvog elementa uvijek je 0).

Polja možemo podijeliti prema broju dimenzija na:

- jednodimenzionalna polja
- dvodimenzionalna polja
- višedimenzionalna polja.

Slika 7.1 prikazuje raspored elemenata jednodimenzionalnog i dvodimenzionalnog polja te način pristupa elementima s pomoću indeksa.



Slika 7.1. Prikaz jednodimenzionalnog i dvodimenzionalnog polja

U ovom udžbeniku bavit ćemo se jednodimenzionalnim i dvodimenzionalnim poljima.

Polje u C++-u sastoji se od elemenata istog tipa podataka, pa razlikujemo:

- polje cijelih brojeva (`int`)
- polje realnih brojeva (`float`, `double`)
- polje znakova (`char`)
- polje logičkih vrijednosti (`bool`)
- polje stringova (`string`).

7.1. Jednodimenzionalna polja

Jednodimenzionalno polje služi nam u praksi kada trebamo raditi s **više podataka iste vrste**.

Na primjer:

- unos više ocjena učenika
- spremanje rezultata mjerenja
- rad s nizom brojeva
- obrada većeg skupa podataka.

U takvim situacijama nije praktično definirati velik broj pojedinačnih varijabli, već upotrebjavamo jedno polje koje sadrži više elemenata.

Na taj način podatci su preglednije organizirani, lakši za obradu i dostupni s pomoću imena polja i indeksa.

7.1.1. Deklaracija i inicijalizacija jednodimenzionalnog polja

Za deklariranje jednodimenzionalnog polja potrebno je navesti: **tip podataka, naziv polja i broj elemenata u jednoj uglatoj zagradi**.

Opći zapis deklaracije jednodimenzionalnog polja glasi:

```
tip naziv_polja[broj_elementa];
```

Polje se može inicijalizirati na više načina, što je prikazano sljedećim primjerima.

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    // polje cijelih brojeva
    int a[3] = {1, 2, 3};

    // polje realnih brojeva (float)
    float f[3] = {1.1f, 2.2f, 3.3f};

    // polje realnih brojeva (double)
    double d[3] = {1.11, 2.22, 3.33};

    // polje znakova
    char c[3] = {'A', 'B', 'C'};

    // polje logičkih vrijednosti
    bool b[3] = {true, false, true};

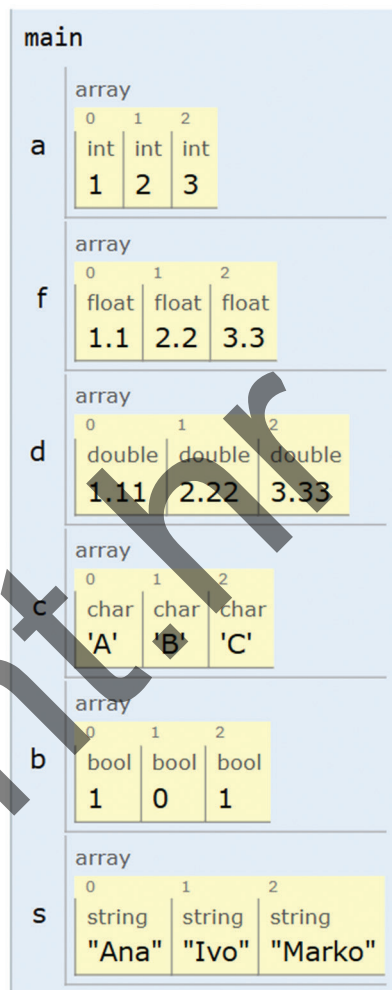
    // polje stringova
    string s[3] = {"Ana", "Ivo", "Marko"};

    return 0;
}
```


Slika 7.2 prikazuje raspored elemenata polja različitih tipova podataka u memoriji.

Na slici se vidi kako polja tipa `int`, `float`, `double`, `char`, `bool` i `string` pohranjuju svoje elemente, pri čemu su svi elementi istog polja istog tipa i poredani jedan za drugim.

U nastavku je prikazano nekoliko različitih načina inicijalizacije jednodimenzionalnog polja. Na primjeru se mogu uočiti posebnosti prilikom inicijalizacije, kao što su djelomično zadavanje elemenata, automatsko popunjavanje vrijednosti te određivanje veličine polja. Ako se inicijalizira manje elemenata nego što polje sadrži, preostali elementi automatski se postavljaju na nulu.



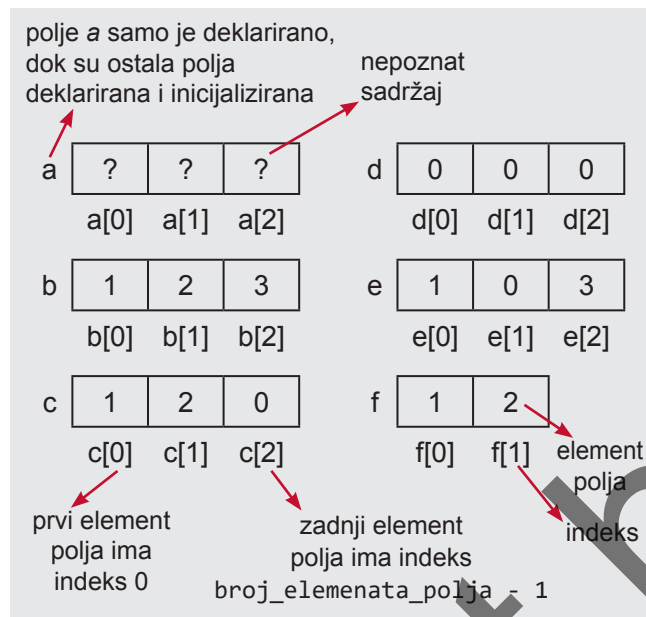
Slika 7.2. Prikaz polja različitih tipova podataka

```
#include <iostream>
using namespace std;

int main() {
    int a[3];           // vrijednosti nisu određene
    int b[3] = {1, 2, 3}; // svi elementi zadani
    int c[3] = {1, 2};   // treći element dobiva vrijednost 0
    int d[3] = {0};      // svi elementi su 0
    int f[] = {1, 2};    // veličina se određuje automatski

    return 0;
}
```

Slika 7.3 prikazuje način deklaracije i inicijalizacije jednodimenzionalnog polja te raspored njegovih elemenata u memoriji na temelju prethodno danog koda. Tako su prikazani različiti načini inicijalizacije polja, uključujući polja s nepoznatim vrijednostima, potpuno i djelomično inicijalizirana polja te način određivanja indeksa elemenata.



Slika 7.3. Prikaz deklaracije, inicijalizacije i rasporeda elemenata jednodimenzionalnog polja u memoriji

Deklaracija polja s pomoću simboličke konstante

Veličina polja može se deklarirati s pomoću simboličke konstante. Simbolička konstanta predstavlja vrijednost koja se zadaje na početku programa i kasnije se upotrebljava u kodu. Dogovorno se piše velikim slovima.

Ovakav način deklaracije koristan je kada želimo da program bude pregledniji i lakši za izmjenu. Ako želimo promijeniti veličinu polja, dovoljno je promijeniti vrijednost konstante na jednom mjestu.

```
#include <iostream>
using namespace std;

#define MAX 100

int main() {
    int a[MAX];

    return 0;
}
```

7.2.1. Ispis vrijednosti elemenata polja

Za ispis vrijednosti pojedinog elementa polja upotrijebimo:

- ime polja
- indeks elementa.

Opći zapis za pristup elementu polja: `naziv_polja[indeks]`

Indeks određuje kojem elementu polja pristupamo, odnosno koji element želimo ispisati. Polje mora biti inicijalizirano ili u polje moraju biti učitane vrijednosti. U suprotnom vrijednosti elemenata nisu određene.

Primjer 7.1.

Što će se ispisati na zaslonu računala nakon izvođenja sljedećeg programa?

```
#include <iostream>
using namespace std;

#define MAX 3

int main() {

    int a[MAX];
    int b[MAX] = {1, 2, 3};

    cout << a[0] << endl;
    cout << b[0] << " " << b[1] << " " << b[2];

    return 0;
}
```

Rješenje

Polje a je deklarirano, ali nije inicijalizirano. To znači da njegove vrijednosti nisu određene i ovise o stanju memorije.

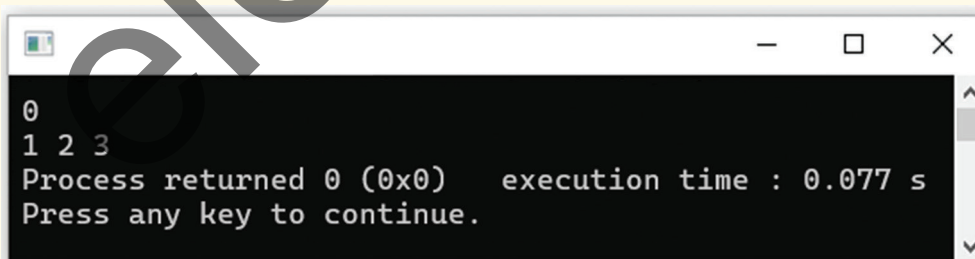
Zbog toga će se prilikom ispisa a[0] ispisati neka **nepoznata (slučajna) vrijednost**.

Polje b inicijalizirano je vrijednostima {1, 2, 3} pa vrijedi:

- b[0] = 1
- b[1] = 2
- b[2] = 3

Zato će se u drugom redu ispisati: 1 2 3

Konačan ispis prikazuje slika 7.4.



Slika 7.4. Primjer ispisa programa u terminalu

Napomena

Ako polje nije inicijalizirano, njegove vrijednosti nisu definirane i ne smije se pretpostaviti da su jednake nuli.

Izmijenite program

inicijalizirajte polje a te ispišite sve elemente polja a i polja b.



Ispis elemenata polja s pomoću petlje

Za razliku od jednostavnih varijabli polje sadrži više elemenata. Zbog toga nije praktično pristupati svakom elementu pojedinačno. Umjesto toga, kroz polje se **krećemo s pomoću petlje**.

Petlja omogućuje da:

- pristupimo svim elementima polja
- obradimo svaki element na isti način
- izbjegnemo ponavljanje koda.

Kod ispisa elemenata polja upotrebljava se kontrolna varijabla (najčešće *i*) koja predstavlja indeks elementa.

U svakom prolazu petlje kontrolna varijabla *i* poprima novu vrijednost te se pristupa novom elementu *x[i]*. Na taj način prolazimo cijelim poljem, od prvog do zadnjeg elementa.

Granica petlje mora odgovarati veličini polja. Ako polje ima *n* elemenata, indeksi elemenata su u intervalu **[0, n-1]**.

```
for (int i = 0; i < n; i++)
```

- 0 → prvi element polja
- *n* - 1 → zadnji element polja

Za ispis elemenata polja upotrebljava se naredba `cout <<`.

Opći zapis

```
for (int i = 0; i < n; i++) {
    cout << naziv_polja[i] << endl;
}
```

Primjer 7.2.

Što će se ispisati na zaslonu računala nakon izvođenja sljedećeg programa?

```
#include <iostream>
#include <string>
using namespace std;

int main() {

    string ucenici[4] = {"Ana", "Marko", "Iva", "Luka"};
    int visine[4] = {165, 178, 170, 182};

    for (int i = 0; i < 4; i++) {
        cout << ucenici[i] << " - " << visine[i] << " cm" << endl;
    }

    return 0;
}
```

Rješenje

Petlja prolazi svim elementima polja od indeksa 0 do 3. U svakom prolazu uzima ime učenika iz polja `ucenici` i odgovarajuću visinu iz polja `visine` te ih ispisuje u istom retku. Budući da su elementi u oba polja usklađeni po indeksima, ispisuju se parovi *ime – visina* za svakog učenika.

Ispis programa:

Ana – 165 cm

Marko – 178 cm

Iva – 170 cm

Luka – 182 cm

Online alat **Python Tutor** omogućuje praćenje izvođenja programa korak po korak. Može se vidjeti kako se tijekom izvođenja programa postupno stvara ispis na zaslonu, odnosno kako se u svakom prolazu petlje ispisuje novi element polja.



Izmijenite program

Ispišite imena učenika i njihove visine obrnutim redoslijedom u odnosu na redoslijed u polju.

7.1.3. Učitavanje elemenata polja s pomoću petlje

Za razliku od jednostavnih varijabli polje sadrži više elemenata te zbog toga nije praktično unositi svaki element pojedinačno. Umjesto toga, u polje **unosimo vrijednosti s pomoću petlje**.

Petlja omogućuje da:

- unesemo sve elemente polja
- svaki element unosimo na isti način
- izbjegnemo ponavljanje koda.

Kod učitavanja elemenata polja koristimo se kontrolnom varijablom (najčešće `i`) koja predstavlja indeks elementa. U svakom prolazu petlje kontrolna varijabla `i` poprima novu vrijednost te se unosi vrijednost u element `x[i]`.

Na taj način popunjavamo cijelo polje, od prvog do zadnjeg elementa.

Za unos elemenata polja koristimo se naredbom `cin >>`, a unos se uvijek vrši po redoslijedu indeksa, od prvog prema zadnjem elementu polja.

Opći zapis

```
for (int i = 0; i < n; i++) {
    cin >> naziv_polja[i];
}
```

Primjer 7.3.

Napišite program u kojem korisnik može učitati u jednodimenzionalno polje maksimalno 5 elemenata.

Korisnik prvo unosi koliko će polje sadržavati elemenata. Ako je unesena vrijednost izvan dopuštenog intervala, potrebno je omogućiti ponovni unos. Nakon toga korisnik unosi elemente polja koje je potrebno ispisati.

ulazni podatci	izlazni podatci
6	Pogresan unos. Ponovite unos.
3 1 2 3	x[0] = 1 x[1] = 2 x[2] = 3

Rješenje

```
#include <iostream>
using namespace std;

#define MAX 5

int main() {
    int x[MAX];
    int n;

    do {
        cout << "Unesite dimenziju polja: ";
        cin >> n;

        if (n > MAX || n < 1)
            cout << "Pogresan unos. Ponovite unos." << endl;
    } while (n > MAX || n < 1);

    for (int i = 0; i < n; i++) {
        cout << "Unesite vrijednost " << i + 1 << ". elementa polja: ";
        cin >> x[i];
    }

    for (int i = 0; i < n; i++) {
        cout << "x[" << i << "] = " << x[i] << endl;
    }

    return 0;
}
```

Korisnik najprije unosi broj elemenata polja. Ako unesena vrijednost nije u dopuštenom intervalu (od 1 do 5), unos se ponavlja s pomoću do-while petlje.

Nakon ispravnog unosa:

- s pomoću prve for petlje unose se elementi polja
- s pomoću druge for petlje elementi se ispisuju.

Proširite program

Nakon ispisa elemenata polja izračunajte i ispišite najveći uneseni element.



Primjer 7.4.

Napišite program u kojem korisnik unosi podatke o učenicima. U programu je potrebno omogućiti:

- unos broja učenika (maksimalno 5)
- unos imena učenika
- unos visine svakog učenika (realni brojevi)
- unos donje i gornje granice visine.

Nakon unosa potrebno je ispisati imena svih učenika čija je visina unutar zadanog intervala (veća ili jednaka donjoj granici i manja ili jednaka gornjoj granici).

ulazni podatci	izlazni podatci
3 Ana 165.5 Marko 180.2 Iva 170.0 160 175	Ucenici u zadanom intervalu su: Ana Iva

Rješenje

```
#include <iostream>
#include <string>
using namespace std;

#define MAX 5

int main() {
    int n;
    float donja, gornja;
    string ime[MAX];
    float visina[MAX];

    cout << "Unesite broj učenika (max 5): ";
    cin >> n;

    cout << "Unesite imena učenika:" << endl;
    for (int i = 0; i < n; i++) {
        cin >> ime[i];
    }

    cout << "Unesite visine učenika:" << endl;
    for (int i = 0; i < n; i++) {
        cin >> visina[i];
    }
}
```

```

cout << "Unesite donju i gornju granicu visine: ";
cin >> donja >> gornja;

cout << "Ucenici u zadanom intervalu su:" << endl;

for (int i = 0; i < n; i++) {
    if (visina[i] >= donja && visina[i] <= gornja) {
        cout << ime[i] << endl;
    }
}

return 0;
}

```

Izmijenite program

Uz postojeće podatke ispišite i broj učenika čija je visina unutar zadanog intervala.



7.2. Dvodimenzionalna polja

Dvodimenzionalna polja upotrebljavaju se kada podatke želimo organizirati na primjer u obliku **tablice** (redci i stupci).

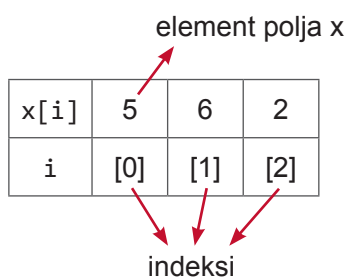
Takav način prikaza upotrebljava se u različitim situacijama, na primjer:

- tablica ocjena učenika
- raspored sati
- matrice u matematici
- organizacija podataka u redove i stupce.

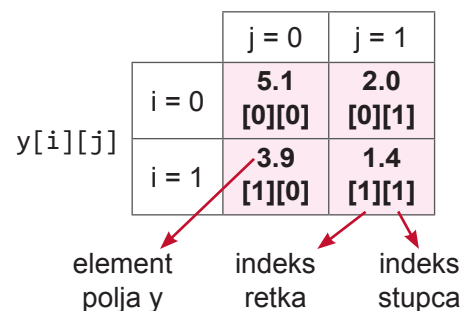
Za razliku od jednodimenzionalnog polja, gdje svaki element određujemo jednim indeksom, kod dvodimenzionalnog polja svaki element određujemo s pomoću **dva indeksa**:

- prvi indeks označava **redak** (kontrolna varijabla *i*)
- drugi indeks označava **stupac** (kontrolna varijabla *j*).

Slika 7.5 prikazuje razliku između jednodimenzionalnog i dvodimenzionalnog polja pri čemu se elementi **jednodimenzionalnog polja nalaze u jednom retku**, dok su elementi **dvodimenzionalnog polja raspoređeni u retke i stupce**. U uglatim zagradama prikazani su **indeksi elemenata polja** pri čemu se kod jednodimenzionalnog polja upotrebljava jedan indeks, a kod dvodimenzionalnog polja dva indeksa: **prvi označava redak**, a **drugi stupac**.



```
int x[3] = {5, 6, 2};
```



```
float y[2][2] = { {5.1f, 2.0f},
                  {3.9f, 1.4f} };
```

Slika 7.5. Usporedba jednodimenzionalnog i dvodimenzionalnog polja

Iako dvodimenzionalno polje promatramo kao tablicu (redci i stupci), njegovi elementi u memoriji nisu raspoređeni u obliku tablice, već **redoslijedno, jedan za drugim (linear-
no)**, kao i kod jednodimenzionalnog polja.

To znači da se elementi pohranjuju **po redcima (redak po redak)**. Prvo se u memoriju spremaju svi elementi prvog retka, zatim svi elementi drugog retka i tako dalje.

Važno je uočiti da su elementi $y[0][1] = 2.0$ i $y[1][0] = 3.9$ susjedni u memoriji iako se u tablici nalaze u različitim redcima. Tako je za prikazano polje y raspored u memoriji računala sljedeći (slika 7.6):

	j = 0	j = 1
i = 0	5.1 [0][0]	2.0 [0][1]
i = 1	3.9 [1][0]	1.4 [1][1]
vizualizacija polja		

5.1	2.0	3.9	1.4
[0][0]	[0][1]	[1][0]	[1][1]
prikaz u memoriji računala			

Slika 7.6. Prikaz polja y u memoriji računala (elementi se u memoriji pohranjuju redak po redak)

7.2.1. Deklaracija i inicijalizacija dvodimenzionalnog polja

Pri deklariranju dvodimenzionalnog polja navodi se: tip podataka, naziv polja, broj redaka i broj stupaca.

Opći zapis

```
tip naziv_polja[broj_redaka][broj_stupaca];
```

Polje se može inicijalizirati na više načina, što je prikazano sljedećim primjerima.

```
#include <iostream>
#include <string>
using namespace std;

int main() {

    // polje cijelih brojeva
    int a[2][3] = { {1, 2, 3},
                   {4, 5, 6} };

    // polje realnih brojeva (float)
    float f[2][3] = { {1.1f, 2.2f, 3.3f},
                     {4.4f, 5.5f, 6.6f} };

    // polje realnih brojeva (double)
    double d[2][3] = { {1.11, 2.22, 3.33},
                      {4.44, 5.55, 6.66} };
```



```

// polje znakova
char c[2][3] = { {'A', 'B', 'C'},
                 {'D', 'E', 'F'} };

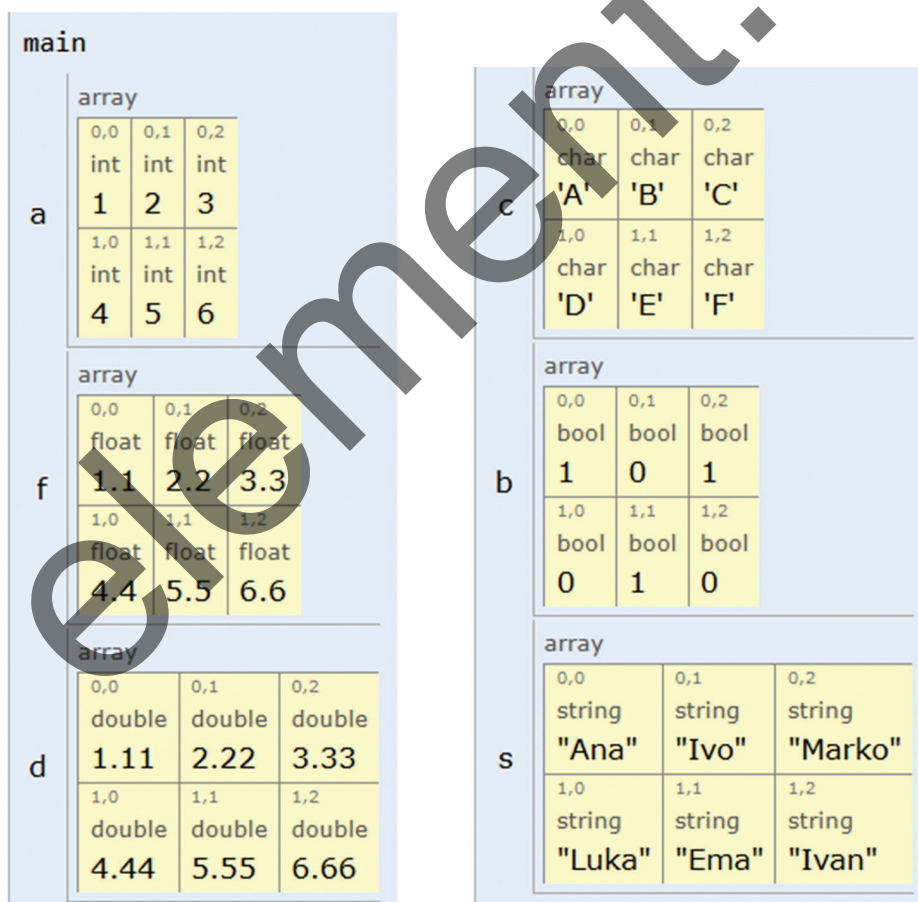
// polje logičkih vrijednosti
bool b[2][3] = { {true, false, true},
                 {false, true, false} };

// polje stringova
string s[2][3] = { {"Ana", "Ivo", "Marko"},
                  {"Luka", "Ema", "Ivan"} };

return 0;
}

```

Sva polja imaju isti broj redaka i stupaca, a elementi su organizirani po redcima. Svaki redak zapisuje se unutar posebnih vitičastih zagrada radi lakšeg uočavanja elemenata. Slikom 7.7 prikazan je raspored elemenata dvodimenzionalnih polja različitih tipova podataka u memoriji.



Slika 7.7. Prikaz rasporeda elemenata dvodimenzionalnih polja različitih tipova podataka u memoriji

U nastavku je prikazano nekoliko različitih načina inicijalizacije **dvodimenzionalnog polja**. Na primjeru se mogu uočiti posebnosti prilikom inicijalizacije, kao što su djelomično zadavanje elemenata, automatsko popunjavanje vrijednosti nulama te automatsko određivanje broja redaka uz obavezno navođenje broja stupaca.

```
#include <iostream>
using namespace std;

int main() {

    // polje samo definirano (vrijednosti nisu određene)
    int a[2][3];

    // svi elementi zadani u jednom nizu (popunjava se po redcima)
    int b[2][3] = {1, 2, 3, 4, 5, 6};

    // elementi zadani po redcima
    int c[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };

    // djelomična inicijalizacija (ostali elementi postaju 0)
    int d[2][3] = {
        {1},
        {3, 4}
    };

    // inicijalizacija pojedinih elemenata (ostali su 0)
    int e[2][3] = {
        {0, 5, 0},
        {0, 0, 4}
    };

    // određivanje broja redaka automatski (broj stupaca mora biti zadan)
    int f[][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };

    return 0;
}
```

Slika 7.8 prikazuje različite načine inicijalizacije polja, uključujući polja s nepoznatim vrijednostima, potpuno i djelomično inicijalizirana polja te način pristupa elementima s pomoću indeksa retka i stupca.

a

?	?	?
?	?	?

b

1	2	3
4	5	6

c

1	2	3
4	5	6

d

1	0	0
3	4	0

e

0	5	0
0	0	4

f

1	2	3
4	5	6

Slika 7.8. Vizualizacija dvodimenzionalnih polja

Indeksi redaka i stupaca počinju od 0. Na primjer, za dvodimenzionalno polje x koje ima dva retka i tri stupca, elementi polja raspoređeni su prema indeksima kako je prikazano na slici 7.9. Svaki element polja određuje se s pomoću dva indeksa:

- prvi indeks označava redak
- drugi indeks označava stupac.

Na primjer:

- $x[0][0]$ → element u prvom retku i prvom stupcu
- $x[1][2]$ → element u drugom retku i trećem stupcu.

		j = 0	j = 1	j = 2
i = 0	x	$x[0][0]$	$x[0][1]$	$x[0][2]$
i = 1		$x[1][0]$	$x[1][1]$	$x[1][2]$

Slika 7.9. Pregled indeksa redaka i stupaca elemenata dvodimenzionalnog polja x

Deklariranje dvodimenzionalnog polja s pomoću simboličkih konstanti

Veličina dvodimenzionalnog polja može se također definirati s pomoću simboličkih konstanti.

Kod dvodimenzionalnih polja potrebno je odrediti:

- broj redaka
- broj stupaca.

Ovakav način deklaracije koristan je kada želimo da program bude pregledniji i lakši za izmjenu. Ako želimo promijeniti veličinu polja, dovoljno je promijeniti vrijednosti konstanti na jednom mjestu.

```
#include <iostream>
using namespace std;

#define MAXRED 10
#define MAXSTUP 15

int main() {
    int x[MAXRED][MAXSTUP];
    return 0;
}
```

Napomena

Kod dvodimenzionalnih polja broj stupaca mora biti poznat pri definiranju polja, dok se broj redaka može odrediti na temelju inicijalizacije.

7.2.2. Ispis i učitavanje elemenata dvodimenzionalnog polja

Za razliku od jednodimenzionalnog polja dvodimenzionalno polje sadrži elemente raspoređene u **retke i stupce**. Zbog toga se za rad s takvim poljem upotrebljavaju **dvije ugniježdene petlje**.

Kod rada s dvodimenzionalnim poljem koristimo se dvjema kontrolnim varijablama:

- i → indeks retka
- j → indeks stupca.

U svakom prolazu petlji pristupa se elementu:

```
naziv_polja[i][j]
```

Opći zapisi za ispis i učitavanje elemenata u dvodimenzionalno polje

Ispis elemenata polja

```
for (int i = 0; i < redovi; i++) {  
    for (int j = 0; j < stupci; j++) {  
        cout << naziv_polja[i][j] << " ";  
    }  
    cout << endl;  
}
```

Pri ispisu dvodimenzionalnog polja naredba `cout << endl`; označava kraj retka i prijelaz na sljedeći, čime se zadržava pregledan prikaz elemenata. Na taj način ispis na zaslonu odgovara tabličnom prikazu polja.

Učitavanje elemenata polja

```
for (int i = 0; i < redovi; i++) {  
    for (int j = 0; j < stupci; j++) {  
        cin >> naziv_polja[i][j];  
    }  
}
```

Primjer 7.5.

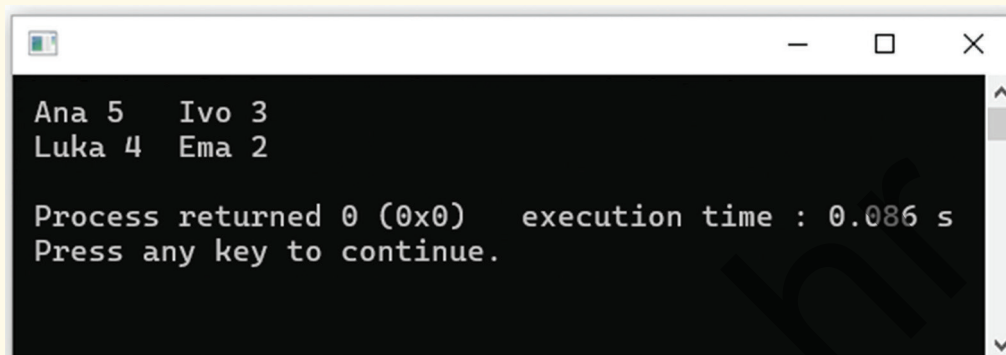
Što će se ispisati na zaslonu računala nakon izvođenja sljedećeg programa?

```
#include <iostream>  
#include <string>  
using namespace std;  
  
int main() {  
    string ime[2][2] = {  
        {"Ana", "Ivo"},  
        {"Luka", "Ema"}  
    };  
  
    int bodovi[2][2] = {  
        {5, 3},  
        {4, 2}  
    };  
  
    for (int i = 0; i < 2; i++) {  
        for (int j = 0; j < 2; j++) {  
            cout << ime[i][j] << " " << bodovi[i][j] << "\t";  
        }  
        cout << endl;  
    }  
  
    return 0;  
}
```

Rješenje

Program redom obrađuje elemente polja i za svaki ispisuje ime i pripadajući broj bodova.

Ispis na zaslonu računala bit će kao na slici 7.10.



```
Ana 5   Ivo 3
Luka 4   Ema 2

Process returned 0 (0x0)   execution time : 0.086 s
Press any key to continue.
```

Slika 7.10. Ispis dvodimenzionalnog polja na zaslonu računala

Vanjska petlja prolazi redcima, a unutarnja stupcima. Elementi se ispisuju redom kako su pohranjeni u polju, pri čemu naredba `end1` označava kraj retka i prijelaz na sljedeći.

Alat Python Tutor omogućuje praćenje izvođenja programa korak po korak. Skeniranjem QR koda moguće je pogledati videozapis s prikazom izvođenja programa i ispisa na zaslonu.

**Proširite program**

Omogućite unos imena i bodova za sve elemente polja, a zatim ih ispišite u istom obliku kao u primjeru.



Primjer 7.6.

Napišite program u kojem se učitavaju prirodni brojevi n i m (tipa `int`) pri čemu n predstavlja broj redaka, a m broj stupaca dvodimenzionalnog polja. Polje može imati maksimalno 10 redaka i 15 stupaca (deklarirati s pomoću simboličkih konstanti).

Ako su n i m pogrešno zadani, korisniku treba ispisati poruku:

“Pogresan unos. Ponovite unos.”

Nakon što korisnik učita ispravan broj redaka i stupaca potrebno je:

- u dvodimenzionalno polje tipa `float` učitati $n \times m$ elemenata
- ispisati sve elemente polja u obliku tablice (na 3 decimale)
- izračunati i ispisati sumu svih elemenata polja
- ispisati samo one elemente kod kojih je:
 - indeks retka neparan
 - vrijednost elementa pozitivna.

ulazni podatci	izlazni podatci
Unesite broj redaka i stupaca: 2 3 $x[0][0] = 1.5$ $x[0][1] = -2.0$ $x[0][2] = 3.0$ $x[1][0] = 4.2$ $x[1][1] = 0.0$ $x[1][2] = -1.1$	1.500 -2.000 3.000 4.200 0.000 -1.100 Suma elemenata = 5.600 Elementi s neparnim indeksom retka i pozitivnom vrijednošću: $x[1][0] = 4.200$

Rješenje

```
#include <iostream>
#include <iomanip>
using namespace std;

#define MAXRED 10
#define MAXSTUP 15

int main() {

    int i, j, n, m;
    float x[MAXRED][MAXSTUP];
    float suma = 0;

    // unos dimenzija
    do {
        cout << "Unesite broj redaka i stupaca: ";
        cin >> n >> m;

        if (n > MAXRED || n < 1 || m > MAXSTUP || m < 1)
            cout << "Pogresan unos. Ponovite unos." << endl;

    } while (n > MAXRED || n < 1 || m > MAXSTUP || m < 1);
```

```

// učitavanje elemenata
for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
        cout << "x[" << i << "][" << j << "] = ";
        cin >> x[i][j];
    }
}

// postavljanje ispisa na 3 decimale
cout << fixed << setprecision(3);

// ispis polja
cout << "\nPolje:" << endl;
for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
        cout << x[i][j] << " ";
    }
    cout << endl;
}

// izračun sume
for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
        suma += x[i][j];
    }
}

cout << "\nSuma elemenata = " << suma << endl;

cout << "\nElementi s neparnim indeksom retka i pozitivnom vrijednoscu:"
<< endl;

for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
        if (i % 2 == 1 && x[i][j] > 0) {
            cout << "x[" << i << "][" << j << "] = " << x[i][j] << endl;
        }
    }
}

return 0;
}

```

Izmijenite program

Nakon ispisa sume svih elemenata odredite i ispišite najmanji element dvodimenzionalnog polja.



7.2.3. Dvodimenzionalna polja s jednakim brojem redaka i stupaca

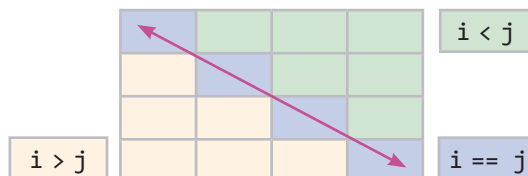
Kod dvodimenzionalnih polja koja imaju jednak broj redaka i stupaca (kvadratna polja) često se promatraju **dijagonale**.

Tako razlikujemo:

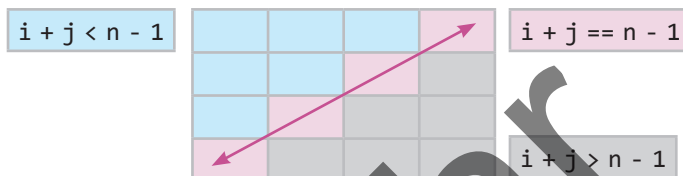
- glavnu dijagonalu
- sporednu dijagonalu.

Slika 7.11 prikazuje elemente glavne i sporedne dijagonale te elemente koji se nalaze iznad i ispod tih dijagonala u dvodimenzionalnom polju.

Glavna dijagonala tablice od n redaka i n stupaca (i – indeks retka, j – indeks stupca)



Sporedna dijagonala tablice od n redaka i n stupaca (i – indeks retka, j – indeks stupca)



Slika 7.11. Glavna i sporedna dijagonala u dvodimenzionalnom polju

Glavna dijagonala

Glavnu dijagonalu čine elementi kod kojih su indeksi retka i stupca jednaki ($i == j$).

Primjeri elemenata glavne dijagonale:

$x[0][0]$, $x[1][1]$, $x[2][2]$, ...

Sporedna dijagonala

Sporednu dijagonalu čine elementi kod kojih je zbroj indeksa retka i stupca jednak:

$i + j == n - 1$ gdje je n broj redaka (ili stupaca).

Primjeri elemenata sporedne dijagonale:

$x[0][n-1]$, $x[1][n-2]$, $x[2][n-3]$, ...

Podjela elemenata

Na temelju dijagonala elementi polja mogu se podijeliti na:

- elemente **iznad glavne dijagonale** → $i < j$
- elemente **ispod glavne dijagonale** → $i > j$
- elemente **iznad sporedne dijagonale** → $i + j < n - 1$
- elemente **ispod sporedne dijagonale** → $i + j > n - 1$

Primjer 7.7.

Zadano je dvodimenzionalno polje dimenzija $n \times n$. Napišite program koji:

- učitava elemente polja
- ispisuje sve elemente glavne dijagonale
- izračunava i ispisuje sumu elemenata sporedne dijagonale.

ulazni podatci	izlazni podatci
Unesite n: 3 x[0][0] = 1 x[0][1] = 2 x[0][2] = 3 x[1][0] = 4 x[1][1] = 5 x[1][2] = 6 x[2][0] = 7 x[2][1] = 8 x[2][2] = 9	Glavna dijagonala: 1 5 9 Suma sporedne dijagonale = 15

Rješenje

```

#include <iostream>
using namespace std;

#define MAX 10

int main() {

    int i, j, n;
    int x[MAX][MAX];
    int suma = 0;

    cout << "Unesite n: ";
    cin >> n;

    // učitavanje elemenata
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            cout << "x[" << i << "][" << j << "] = ";
            cin >> x[i][j];
        }
    }

    // ispis glavne dijagonale
    cout << "\nGlavna dijagonala: ";
    for (i = 0; i < n; i++) {
        if (i == j) { // ovo ne treba, ali radi simetrije može ostati
            cout << x[i][i] << " ";
        }
    }

    // suma sporedne dijagonale
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (i + j == n - 1) {
                suma += x[i][j];
            }
        }
    }

    cout << "\nSuma sporedne dijagonale = " << suma << endl;

    return 0;
}

```

Glavna dijagonala može se ispisati s pomoću jedne petlje:

```
for (i = 0; i < n; i++) {  
    cout << x[i][i] << " ";  
}
```

jer za elemente glavne dijagonale vrijedi $i == j$, pa nije potrebno prolaziti cijelim poljem.

Slično, suma elemenata sporedne dijagonale može se izračunati optimizirano:

```
for (i = 0; i < n; i++) {  
    suma += x[i][n - 1 - i];  
}
```

jer za nju vrijedi $i + j = n - 1$, odnosno može se izraziti $j = n - 1 - i$.

Zahvaljujući takvom pristupu, obrađuje se samo n elemenata sporedne dijagonale umjesto svih $n \times n$ elemenata polja, čime se pojednostavljuje i ubrzava izvođenje programa.

Razmisli: Ako su dimenzije dvodimenzionalnog polja 1000×1000 , koliko se elemenata provjerava pri prolasku cijelim poljem, a koliko pri optimiziranom prolasku samo sporednom dijagonalom? Za koliko se smanjuje broj iteracija petlje?

Izmijenite program

Ispišite elemente ispod sporedne dijagonale te izračunajte i ispišite sumu svih elemenata na sporednoj dijagonali.



element.hr



VRIJEME ZA IZAZOV

Analiza rezultata provjere znanja

Radna situacija

Nalaziš se na praktičnoj nastavi iz Uvoda u programiranje gdje dobivaš zadatak izrade jednostavne aplikacije za vođenje evidencije rezultata učenika u razredu. Nastavnik želi imati uvid u prosječne rezultate provjere znanja te brzo prepoznati učenike koji postižu iznadprosječne rezultate.



Zadatak

Napiši program koji omogućuje unos broja učenika te unos njihovih imena i ostvarenih bodova na provjeri znanja. Nakon što su podatci uneseni, program treba izračunati prosječan broj bodova svih učenika te odrediti najveći ostvareni broj bodova. Program zatim treba ispisati te rezultate na zaslonu računala. Nakon toga program treba pronaći sve učenike koji su ostvarili broj bodova veći od prosjeka te za svakog od njih ispisati njihova imena.

Primjer ulaznih i izlaznih podataka

ulazni podatci	izlazni podatci
Broj učenika: 4	Prosječan broj bodova: 13.5
Učenici i bodovi: Ana 12 Tomislav 18 Iva 9 Luka 15	Najveći broj bodova: 18 Imena učenika s bodovima iznad prosjeka: Tomislav Luka

Elementi vrednovanja – program i pseudokod

elementi provjere	bodovi
unos broja učenika, imena i bodova	4
pohranjivanje podataka u odgovarajuća polja	4
izračun prosjeka bodova	6
pronalaženje najvećeg broja bodova	5
ispis učenika s bodovima većim od prosjeka (prvo slovo imena)	6
pregledan i ispravan ispis rezultata	5
ukupno	30

Ocjena

- 27 – 30 = odličan (5)
- 23 – 26 = vrlo dobar (4)
- 19 – 22 = dobar (3)
- 15 – 18 = dovoljan (2)
- 0 – 14 = nedovoljan (1)

Dodatni zadatak za one koji žele znati više

Proširi program: uz postojeće izračune odredi i ispiši koliko je učenika ostvarilo najveći broj bodova. Njihova imena spremi u novo jednodimenzionalno polje te ih potom ispiši.

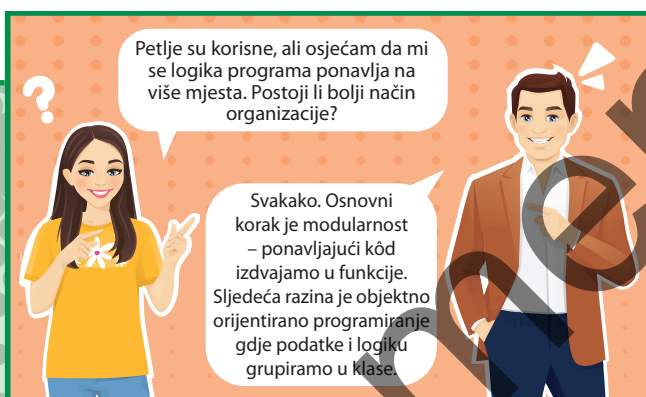
Vrednovanje kao učenje



<https://s.element.hr/0660>

8.

FUNKCIJE, KLASSE I METODE U C++-u



Skeniraj QR kôd ili otvori poveznicu u internetskom pregledniku. Prouči primjere 8.4, 8.8 i 8.11 te ih analiziraj i proširi programe prateći upute iz tih primjera.

<https://s.element.hr/11f7>





Razmisli

Za upis u školu svi učenici dobivaju prazan digitalni obrazac s istim poljima (ime, prezime, razred), ali svaki ga ispunjava svojim podacima. Na dnu obrasca nalazi se gumb koji može izračunati prosjek ocjena.

Pojmovi: **metoda**, **atribut**, **klasa**, **objekt**

Pokušaj pretpostaviti koji se pojam odnosi na koji opis:

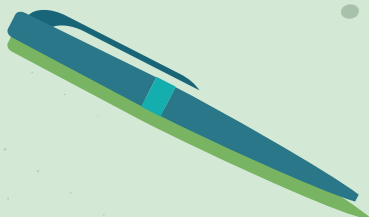
- „nacrt” koji je isti za sve
- ispunjeni obrazac
- podatci koje učenik upisuje
- radnja poput izračuna prosjeka.



Nakon ove lekcije moći će



- objasniti što su funkcije i kako se primjenjuju u programu
- razlikovati prosljeđivanje po vrijednosti i po referenci
- primjenjivati funkcije za obradu podataka i vraćanje rezultata
- upotrebljavati polja i prosljeđivati ih funkcijama
- definirati klasu s atributima i metodama
- stvoriti objekt klase i primjenjivati njegove metode.

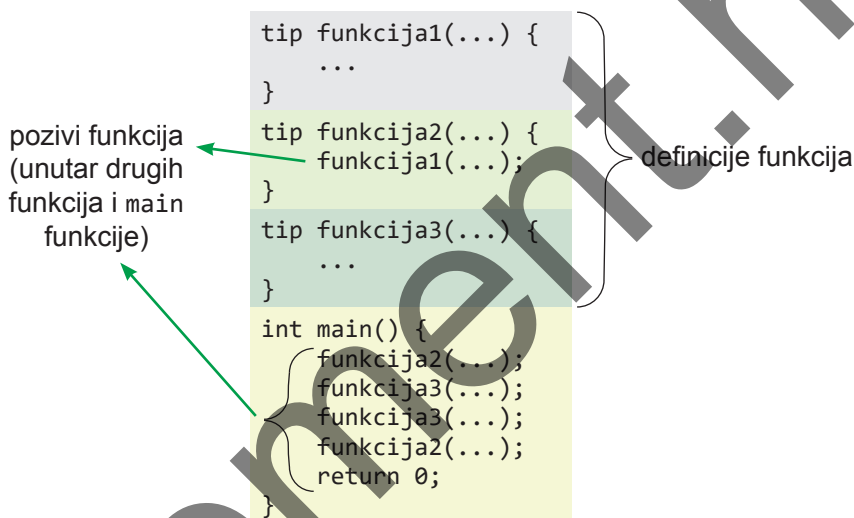


8.1. Rad s funkcijama

Funkcije omogućuju da program bude pregledniji i razumljiviji. Umjesto da sve naredbe pišemo u glavnoj funkciji `main`, možemo ih grupirati u funkcije koje obavljaju određeni zadatak. Time se program dijeli na manje, logičke cjeline koje je lakše razumjeti i održavati.

Osim toga, funkcije omogućuju **ponovnu upotrebu koda**. Jednom napisana funkcija može se **više puta pozivati** u programu čime se **izbjegava ponavljanje istih naredbi**. Takav pristup smanjuje mogućnost pogreške i olakšava izmjene u programu.

Slika 8.1 prikazuje primjer programa u kojem se više funkcija definira odvojeno, a zatim se pozivaju unutar drugih funkcija, uključujući i glavnu funkciju `main()`. Time se jasno vidi kako se program može organizirati u manje, pregledne cjeline te kako funkcije mogu međusobno komunicirati **pozivima**.



Slika 8.1. Primjer organizacije programa upotrebom funkcija i njihovog pozivanja unutar glavne funkcije

Razlikujemo funkcije **s povratnom vrijednošću**, koje vraćaju rezultat, i funkcije **bez povratne vrijednosti**, koje izvršavaju određene radnje.

Opći oblik definicije funkcije (s povratnom vrijednošću)

```

tip_povratne_vrijednosti nazivFunkcije(tip parametar1, tip parametar2, ...) {
    // naredbe funkcije
    return vrijednost;
}
  
```

- funkcije s povratnom vrijednošću upotrebljavaju `return` za vraćanje rezultata na mjesto odakle je funkcija pozvana
- naredbom `return` prekida se izvođenje funkcije
- između vitičastih zagrada `{ }` nalazi se **tijelo funkcije** unutar kojeg se navode sve potrebne definicije varijabli, naredbe te naredba `return`.

Opći oblik definicije funkcije (bez povratne vrijednosti)

```
void nazivFunkcije(tip parametar1, tip parametar2, ...) {
    // naredbe funkcije
}
```

- funkcije tipa void ne vraćaju vrijednost, već izvršavaju određene radnje (npr. ispis, zamjena elemenata u polju i sl.)
- mogu i ne trebaju imati naredbu return.

Ako funkcija nema parametara, zagrade ostaju prazne i prilikom definicije i prilikom poziva funkcije.

U sljedećoj tablici prikazani su najčešći **tipovi funkcija prema povratnoj vrijednosti** kojima ćemo se koristiti u ovom poglavlju.

Tablica 8.1. Tipovi funkcija prema povratnoj vrijednosti u C++-u

tip funkcije	primjer zaglavlja funkcije	opis	C/C++
void	<code>void ispisi()</code>	funkcija ne vraća vrijednost	C i C++
char	<code>char znak()</code>	vraća jedan znak	C i C++
bool	<code>bool provjera(int x)</code>	vraća logičku vrijednost	C i C++ ¹
int	<code>int zbroj(int a, int b)</code>	vraća cijeli broj	C i C++
float	<code>float omjer(float a, int b)</code>	vraća realni broj	C i C++
double	<code>double opseg(double a)</code>	vraća realni broj	C i C++
string	<code>string tekst()</code>	vraća tekst	C++

U ovom poglavlju ograničit ćemo se na najčešće **tipove parametara** prikazane u tablici 8.2.

Tablica 8.2. Neki od tipova parametara funkcije u C-u i C++-u

tip parametra	primjer	opis	C/C++
int	<code>int x</code>	cijeli broj	C i C++
char	<code>char c</code>	znak	C i C++
bool	<code>bool b</code>	logička vrijednost	C i C++
float	<code>float y</code>	realni broj (jednostruka preciznost)	C i C++
double	<code>double x</code>	realni broj (dvostruka preciznost)	C i C++
string	<code>string s</code>	tekst	C++
polje	<code>int polje[]</code>	jednodimenzionalno polje	C i C++
2D polje	<code>int polje[][]</code>	dvodimenzionalno polje	C i C++
referenca ²	<code>int &x</code>	omogućuje promjenu vrijednosti izvan funkcije	C++ ³

1 U C++-u upotrebljava se tip bool, dok se u C-u upotrebljava _Bool, a uključivanjem biblioteke <stdbool.h> moguće je upotrijebiti i naziv bool.

2 Pojam reference bit će detaljnije objašnjen u sljedećem poglavlju.

3 U programskom jeziku C ne postoje reference, ali se slična funkcionalnost može ostvariti s pomoću pokazivača.

8.1.1. Prototip funkcije

Prototip funkcije najava je funkcije koja se piše na početku programa, a omogućuje da se funkcija može pozvati prije nego što je definirana.

Prototip funkcije ujedno je i **deklaracija funkcije**. Ovakav zapis ne mijenja rad programa, već samo omogućuje drukčiju organizaciju koda.

Prilikom pisanja prototipa funkcije u zagradi se navode tipovi parametara, a imena varijabli nije potrebno navoditi iako se njima možemo koristiti. Prototip funkcije uvijek završava znakom ;.

Opći oblik prototipa funkcije

```
tip_povratne_vrijednosti nazivFunkcije(tip parametar1, tip parametar2, ...);
```

Slikai 8.2 prikazuje primjer organizacije programa u kojem se najprije navode deklaracije funkcija (prototipi), a zatim slijedi funkcija main() u kojoj se te funkcije pozivaju. Nakon toga dolaze definicije funkcija, odnosno njihova implementacija.

Takav raspored omogućuje pozivanje funkcija prije njihove definicije jer prevoditelj na temelju deklaracija unaprijed zna koje funkcije postoje, koji im je naziv i koje parametre primaju. Time se program može organizirati tako da se glavna funkcija main() nalazi na početku, a definicije funkcija kasnije u programu.

```
tip funkcija1(...);
tip funkcija2(...);
tip funkcija3(...);
int main() {
    funkcija2();
    funkcija3();
    funkcija3();
    funkcija2();
    return 0;
}
tip funkcija1(...) {
    ...
}
tip funkcija2(...) {
    funkcija1(...);
}
tip funkcija3(...) {
    ...
}
```

Slika 8.2. Deklaracija (prototipi) i definicija funkcija te njihov raspored u programu

Primjer 8.1.

Napišite prototipe funkcija prema zadanim opisima.

1. Funkcija kvadrat prima jedan cijeli broj i vraća njegov kvadrat.
2. Funkcija ispisi prima jedan cijeli broj i ispisuje ga na zaslon.
3. Funkcija vratiPet ne prima parametre i vraća broj 5.
4. Funkcija poruka ne prima parametre i ispisuje poruku na zaslon.

Rješenje

1. `int kvadrat(int);`
2. `void ispisi(int);`
3. `int vratiPet();`
4. `void poruka();`

8.1.2. Gdje se piše funkcija?

Funkcije se u programu mogu pisati na dva načina:

a) najprije se navodi definicija funkcije, a zatim slijedi funkcija u kojoj je pozvana

```
tip nazivFunkcije(...) {
    // naredbe
}
int main() {
    nazivFunkcije(...);
}
```

→ definicija funkcije nazivFunkcije

→ funkcija main u kojoj je pozvana funkcija nazivFunkcije

b) najprije se navodi prototip funkcije, a njezina definicija dolazi kasnije u programu, nakon funkcije u kojoj je pozvana

```
tip nazivFunkcije(...);
int main() {
    nazivFunkcije(...);
}
tip nazivFunkcije(...) {
    // naredbe
}
```

→ prototip funkcije nazivFunkcije

→ funkcija main u kojoj je pozvana funkcija nazivFunkcije

→ definicija funkcije nazivFunkcije

8.1.3. Naredba return

Naredba `return` u funkcijama služi za vraćanje rezultata i završetak izvođenja funkcije.

Upotrebljava se:

a) u funkcijama koje vraćaju vrijednost (funkcije koje nisu tipa `void`)

Ako je tip funkcije različit od tipa `void`, naredbom `return` vraća se rezultat na mjesto poziva.

b) u funkcijama bez povratne vrijednosti

Kod funkcija koje ne vraćaju vrijednost naredba `return` nije obavezna, ali se može upotrijebiti za **prekid izvođenja funkcije**.

Primjer 8.2.

Napišite definicije funkcija iz prethodnog primjera 8.1.

Rješenje

1. funkcija s parametrom i povratnom vrijednošću

```
int kvadrat(int x) {  
    return x * x;  
}
```

Naredbom return vrijednost $x * x$ vraća se na mjesto gdje je funkcija pozvana.

2. funkcija s parametrom bez povratne vrijednosti

```
void ispisi(int x) {  
    cout << x << endl;  
}
```

3. funkcija bez parametara s povratnom vrijednošću

```
int vratiPet() {  
    return 5;  
}
```

4. funkcija bez parametara i bez povratne vrijednosti

```
void poruka() {  
    cout << "Pozdrav" << endl;  
}
```

U funkciji se može upotrijebiti više naredbi return, ali se izvođenje funkcije prekida pri prvom return na koji se naiđe.

Primjer 8.3.

Odredite koju će vrijednost vratiti funkcije veci1 i veci2 ako je $a = 2$ i $b = 3$.

```
int veci1(int a, int b) {  
    if (a > b)  
        return a;  
    else  
        return b;  
}
```

```
int veci2(int a, int b) {  
    if (a > b)  
        return a;  
    return b;  
}
```

Rješenje

Za zadane vrijednosti obje funkcije vraćaju vrijednost **3**.

U oba slučaja uvjet $a > b$ nije zadovoljen pa se u funkciji veci1 izvršava grana else i vraća se vrijednost b.

U drugoj funkciji, budući da uvjet nije ispunjen, preskače se prvi return pa se izvršava sljedeća naredba return b.

Ovaj primjer pokazuje da obje funkcije daju isti rezultat iako su zapisane na različit način te dodatno potvrđuje da else nije nužan kada se upotrebljava return.

8.1.4. Poziv funkcije

Funkcija se poziva navođenjem njezina imena i odgovarajućih **argumenata** u zagradama:

```
nazivFunkcije(argument1, argument2, ...);
```

Argumenti su stvarne vrijednosti koje se proslijeđuju funkciji, a povezuju se s parametrima funkcije **po redoslijedu**.

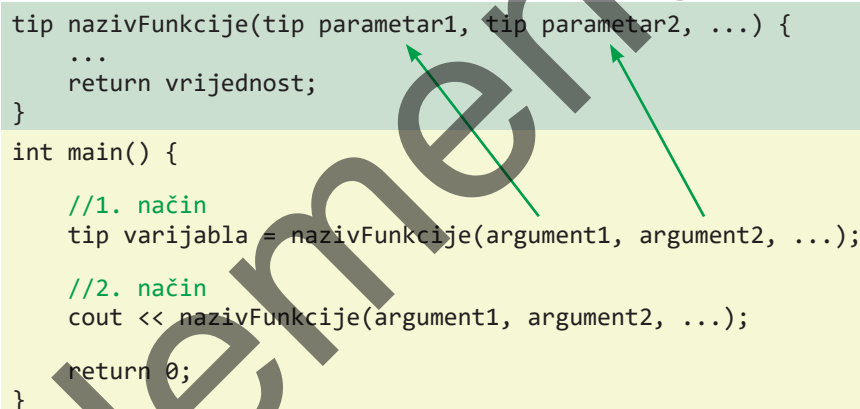
Funkcija se poziva na različite načine, ovisno o tome vraća li vrijednost ili ne.

Funkcija s povratnom vrijednošću

Ako funkcija vraća vrijednost, njezin se rezultat može:

1. spremiti u varijablu: `tip varijabla = nazivFunkcije(...);`
2. upotrijebiti u izrazu: `cout << nazivFunkcije(...);`

Slika 8.3 prikazuje način pozivanja funkcije koja vraća vrijednost. Primjer pokazuje kako se rezultat funkcije može spremiti u varijablu ili izravno upotrijebiti u izrazu te kako se argumenti pri pozivu povezuju s parametrima funkcije prema redoslijedu.



```
tip nazivFunkcije(tip parametar1, tip parametar2, ...) {
    ...
    return vrijednost;
}

int main() {
    //1. način
    tip varijabla = nazivFunkcije(argument1, argument2, ...);

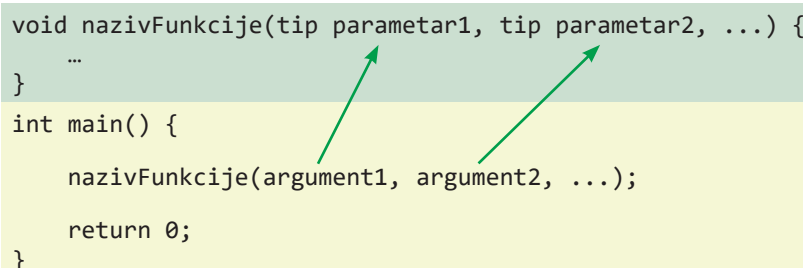
    //2. način
    cout << nazivFunkcije(argument1, argument2, ...);

    return 0;
}
```

Slika 8.3. Poziv funkcije s povratnom vrijednošću te povezivanje parametara i argumenata prema redoslijedu

Funkcija bez povratne vrijednosti (void)

Slika 8.4 prikazuje način pozivanja funkcije koja ne vraća vrijednost (void). Funkcija se poziva kao samostalna naredba, a argumenti se pri pozivu povezuju s parametrima funkcije prema redoslijedu.



```
void nazivFunkcije(tip parametar1, tip parametar2, ...) {
    ...
}

int main() {
    nazivFunkcije(argument1, argument2, ...);

    return 0;
}
```

Slika 8.4. Poziv funkcije bez povratne vrijednosti te povezivanje parametara i argumenata prema redoslijedu

Funkcija se općenito može pozvati i unutar druge funkcije pri čemu se onda najprije izvršava unutarnja funkcija.

Primjer 8.4.

Što će se ispisati na zaslonu računala nakon izvođenja sljedećeg programa?

```
#include <iostream>
using namespace std;

// funkcija s parametrom i povratnom vrijednošću
int kvadrat(int x) {
    return x * x;
}

// funkcija s parametrom bez povratne vrijednosti
void ispisi(int x) {
    cout << x << endl;
}

// funkcija bez parametara s povratnom vrijednošću
int vratiPet() {
    return 5;
}

// funkcija bez parametara i bez povratne vrijednosti
void poruka() {
    cout << "Pozdrav" << endl;
}

int main() {
    int a = 3;
    int b;

    b = kvadrat(a);
    ispisi(b);

    ispisi(vratiPet());
    poruka();

    return 0;
}
```

Rješenje

U glavnoj funkciji main varijabli a pridružuje se vrijednost 3. Nakon toga se poziva funkcija kvadrat, čija se povratna vrijednost ($3 \cdot 3 = 9$) pohranjuje u varijablu b, a zatim se ta vrijednost ispisuje s pomoću funkcije ispisi.

Zatim se poziva ispisi(vratiPet()). Funkcija vratiPet vraća vrijednost 5 koja se izravno prosljeđuje funkciji ispisi i ispisuje.

Na kraju se poziva funkcija poruka() koja ispisuje tekst "Pozdrav".

Konačan ispis na zaslonu računala prikazuje slika 8.5.

```

9
5
Pozdrav

Process returned 0 (0x0)   execution time : 0.380 s
Press any key to continue.

```

Slika 8.5. Ispis rezultata izvođenja programa u terminalu pri čemu su redom prikazani rezultati poziva funkcija

Skeniranjem QR koda moguće je pratiti izvođenje programa korak po korak u alatu *Python Tutor*.



<https://s.element.hr/eccb>

Skeniranjem QR koda moguće je pogledati videozapis u kojem je prikazano kako se funkcije pozivaju i kako se njihov rezultat upotrebljava tijekom izvođenja programa.

Isti program moguće je zapisati i s pomoću **prototipa funkcija**.

```

#include <iostream>
using namespace std;

// prototipi funkcija
int kvadrat(int);
void ispisi(int);
int vratiPet();
void poruka();

int main() {
    int a = 3;
    int b;
    b = kvadrat(a);
    ispisi(b);
    ispisi(vratiPet());
    poruka();
    return 0;
}

```

```

// definicije funkcija
int kvadrat(int x) {
    return x * x;
}

void ispisi(int x) {
    cout << x << endl;
}

int vratiPet() {
    return 5;
}

void poruka() {
    cout << "Pozdrav" << endl;
}

```

Proširite program

Dodajte funkciju koja prima cijeli broj i vraća zbroj njegovih znamenaka te tu funkciju pozovite u main funkciji i ispišite rezultat funkcijom ispisi.



Primjer 8.5.

Napišite funkcije naziva:

- **Poruka** – funkcija ispisuje tekst: „Povrsina je: “
- **Pi** – funkcija putem imena vraća vrijednost konstante π na 5 decimala (3,14159)
- **Povrsina** – funkcija prima vrijednost polumjera kruga, poziva funkciju Pi i putem imena vraća vrijednost površine kruga.

U glavnoj funkciji treba učitati broj n , a potom n racionalnih vrijednosti koje predstavljaju duljine polumjera kruga. Nakon što se učitava pojedina duljina polumjera, na zaslonu računala s pomoću realiziranih funkcija treba odmah ispisati površinu kruga u obliku:

“**Povrsina je: x.xx.**”

Pretpostavite da će korisnik ispravno učitati sve vrijednosti.

ulazni podatci	izlazni podatci
2	Povrsina je: 6.16
1.4	Povrsina je: 16.62
2.3	

Rješenje

Prvi način (putem prototipa funkcija):

```
#include <iostream>
using namespace std;

// prototipi funkcija
void Poruka();
float Pi();
float Povrsina(float polumjer);

int main() {
    float n; // polumjer
    int i, n;

    cout << "Ucitajte broj n: ";
    cin >> n;

    for (i = 0; i < n; i++) {
        cin >> r;
        Poruka();
        cout << Povrsina(r) << endl;
    }

    return 0;
}

void Poruka() {
    cout << "Povrsina je: ";
}

float Pi() {
    return 3.14159;
}
```



```
float Povrsina(float polumjer) {
    float P;
    P = polumjer * polumjer * Pi();
    return P;
}
```

Drugi način (bez prototipa funkcije – upotrebljavaju se samo definicije funkcija):

```
#include <iostream>
using namespace std;

void Poruka() {
    cout << "Povrsina je: ";
}

float Pi() {
    return 3.14159;
}

float Povrsina(float polumjer) {
    float P;
    P = polumjer * polumjer * Pi();
    return P;
}

int main() {
    float r; // polumjer
    int i, n;

    cout << "Ucitajte broj n: ";
    cin >> n;

    for (i = 0; i < n; i++) {
        cin >> r;
        Poruka();
        cout << Povrsina(r) << endl;
    }

    return 0;
}
```

Izmijenite program

Prilagodite program tako da funkcija Povrsina vraća vrijednost, a funkcija Poruka prima tu vrijednost i ispisuje rezultat.



8.1.5. Lokalne i globalne varijable

Varijable u programu mogu biti **lokalne** ili **globalne**, ovisno o tome gdje su deklarirane.

Globalna varijabla

- deklarirana izvan svih funkcija
- dostupna u cijelom programu
- sve funkcije joj mogu pristupiti

Lokalna varijabla

- deklarirana unutar funkcije
- dostupna samo unutar te funkcije
- ne može se upotrebljavati izvan nje

Opći zapis

```
// globalna varijabla
int x;

int main() {
    // lokalna varijabla
    int y;
}
```

Primjer 8.6.

Odredite vrijednosti koje će se ispisati na zaslonu računala.

```
#include <stdio.h>

int x = 10; // globalna

void ispisiX() {
    printf("%d\n", x); // koristi globalnu varijablu
}

int main() {
    int y = 5; // lokalna

    ispisiX();
    printf("%d\n", y);

    return 0;
}
```

Rješenje

Na zaslonu će se ispisati **10** i **5**, pri čemu funkcija `ispisiX` upotrebljava globalnu varijablu `x`, a u `main` se ispisuje vrijednost lokalne varijable `y`.

Primjer 8.7.

Na temelju zadanog programa odgovorite na sljedeća pitanja.

- Odredite što će se ispisati kao rezultat izvođenja programa.
- Pronađite i označite liniju koda u kojoj se poziva funkcija.
- Odredite tip funkcije `f1`.
- Navedite parametre funkcije `f1`.
- Navedite argumente koji se proslijeđuju funkciji prilikom poziva.
- Objasnite kako se argument povezuje s parametrom.
- Navedite koje su lokalne varijable u funkciji `f1`, a koje u funkciji `main`.

```
#include <iostream>
using namespace std;

int f1(int a) {
    return 2 * a;
}

int main() {
    int x;

    x = f1(4);

    cout << x << endl;

    return 0;
}
```

Rješenje

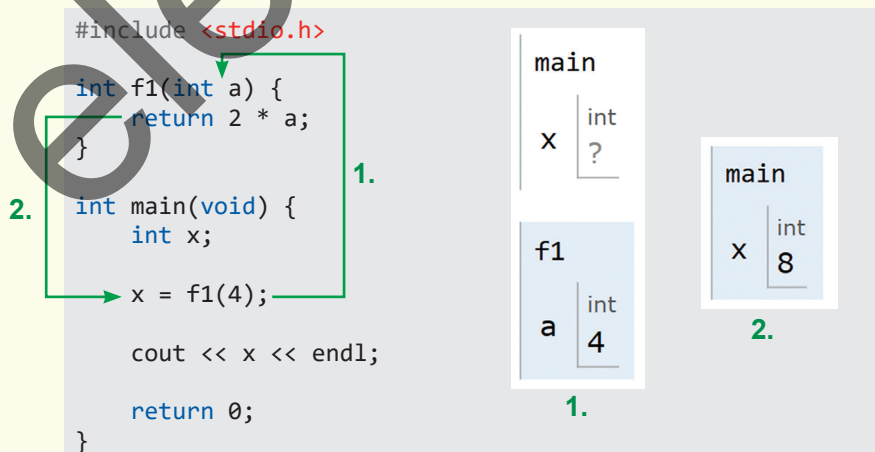
U ovom primjeru definirana je funkcija **f1** koja prima jedan parametar i vraća vrijednost. Funkcija se poziva unutar glavne funkcije **main**, a rezultat se pohranjuje u varijablu **x**.

a) Rezultat izvođenja programa je: 8

Funkcija računa izraz $2 * a$, a budući da je vrijednost parametra $a = 4$, dobiva se $2 \cdot 4 = 8$.

1. Pozivom funkcije **f1(4)** vrijednost argumenta **4** proslijeđuje se parametru **a** u funkciji **f1**. U tom trenutku u memoriji funkcije **f1** postoji lokalna varijabla **a** s vrijednošću 4.
2. Funkcija zatim računa vrijednost izraza $2 * a$ i dobiveni rezultat **8** vraća naredbom **return** na mjesto poziva funkcije. Ta se vrijednost pohranjuje u varijablu **x** u funkciji **main**.

Promjene vrijednosti varijabli tijekom izvođenja programa prikazuje slika 8.6.



Slika 8.6. Vizualizacija prijenosa argumenta, rada funkcije i povratne vrijednosti u memoriji računala

b) Funkcija se poziva u liniji:

```
x = f1(4);
```

U toj liniji šalje se vrijednost funkciji i prima rezultat.

c) Tip funkcije f1 je:

```
int
```

To znači da funkcija vraća cjelobrojnu vrijednost.

d) Parametar funkcije f1 je:

```
a
```

Parametar je varijabla definirana u zaglavlju funkcije koja prima vrijednost.

e) Argument koji se proslijeđuje funkciji je:

```
4
```

Argument je konkretna vrijednost koja se šalje funkciji prilikom poziva.

f) Argument i parametar povezuju se tako da se vrijednost argumenta dodjeljuje parametru:

```
4 → a
```

Nakon toga unutar funkcije vrijedi:

```
a = 4
```

te se izvršava naredba:

```
return 2 * a
```

g) Lokalne varijable su:

- u funkciji f1: a
- u funkciji main: x

Lokalne varijable vidljive su samo unutar funkcije u kojoj su definirane.

8.2. Reference

Pri radu s funkcijama i većim skupovima podataka često je potrebno mijenjati vrijednosti izvan mjesta gdje su definirane. U programskom jeziku C++ to se može jednostavno ostvariti s pomoću **referenci**.

Referenca je **drugo ime za već postojeću varijablu**. To znači da referenca ne predstavlja novu varijablu, već omogućuje pristup istoj memorijskoj lokaciji kao i izvorna varijabla.

Reference omogućuju:

- jednostavnije mijenjanje vrijednosti varijabli
- izbjegavanje kopiranja podataka
- učinkovitiji rad s većim količinama podataka.

Referenca se definira s pomoću operatora &:

```
tip &naziv_reference = postojeća_varijabla;
```

U sljedećem primjeru uz postojeću varijablu a definiramo referencu b:

```
int a = 10;
int &b = a;
```

U prikazanom primjeru:

- a je varijabla
- b je referenca na varijablu a.

Referenca se može upotrebljavati kao varijabla, ali ona ne stvara novu memorijsku lokaciju, već predstavlja drugo ime za postojeću varijablu.

Operator & ima dva značenja, ovisno o kontekstu:

- uz tip podatka (int &b) definira referencu
- uz varijablu (&a) daje adresu varijable.

Iako se upotrebljava isti simbol &, njegovo značenje ovisi o mjestu u programu. Pri definiranju (uz tip podatka) označava referencu, dok u izrazu (uz varijablu) služi za dobivanje adrese varijable.

Referenca mora biti istog tipa kao i varijabla na koju se odnosi jer u suprotnom dolazi do pogreške pri prevođenju programa.

Primjer 8.8.

Što će se ispisati na zaslonu računala nakon izvođenja sljedećeg programa? Pretpostavite da se varijabla a nalazi na adresi 0xca823ffde4.

Napomena: memorijske adrese ovise o računalu i izvođenju programa te ih nije moguće unaprijed odrediti pa se ovdje navodi pretpostavljena vrijednost. 0x predstavlja heksadekadski zapis adrese.

```
#include <iostream>
using namespace std;

int main() {

    int a = 10;
    int &b = a;

    // početne vrijednosti
    cout << "Pocetno stanje:" << endl;
    cout << "a = " << a << ", b = " << b << endl;

    // promjena preko a
    a = 5;
    cout << "\nNakon promjene a = 5:" << endl;
    cout << "a = " << a << ", b = " << b << endl;
```

```
// promjena preko b
b = 7;
cout << "\nNakon promjene b = 7:" << endl;
cout << "a = " << a << ", b = " << b << endl;

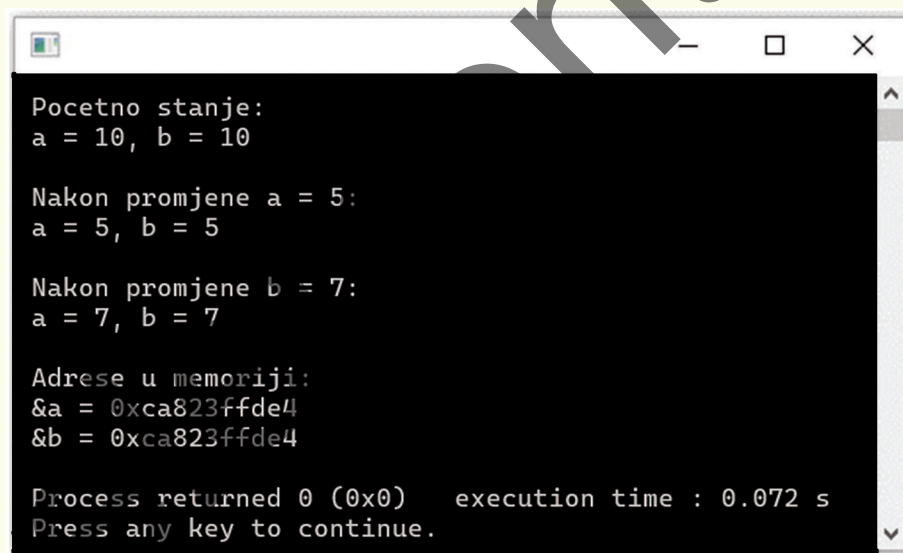
// ispis memorijskih adresa
cout << "\nAdrese u memoriji:" << endl;
cout << "&a = " << &a << endl;
cout << "&b = " << &b << endl;

return 0;
}
```

Rješenje

Slika 8.7 prikazuje ispis programa u terminalu nakon njegova izvođenja. Na temelju ispisa moguće je provjeriti kako **promjene vrijednosti varijable i reference utječu jedna na drugu** te uočiti da **obje imaju istu memorijsku adresu**.

Vrijednosti memorijskih adresa (&a i &b) mogu se razlikovati od računala do računala, ali će uvijek biti jednake jer a i b označavaju istu memorijsku lokaciju.



```
Pocetno stanje:
a = 10, b = 10

Nakon promjene a = 5:
a = 5, b = 5

Nakon promjene b = 7:
a = 7, b = 7

Adrese u memoriji:
&a = 0xca823ffde4
&b = 0xca823ffde4

Process returned 0 (0x0)   execution time : 0.072 s
Press any key to continue.
```

Slika 8.7. Ispis rezultata programa u terminalu koji prikazuje promjene vrijednosti varijable i reference te njihovu zajedničku memorijsku adresu

Proširite program

Dodajte novu varijablu tipa `float` i referencu na nju te ispišite njihove vrijednosti i adrese.



8.3. Prosljeđivanje parametara u funkcijama

Prilikom poziva funkcije često je potrebno proslijediti vrijednosti varijabli. Način na koji se te vrijednosti prenose u funkciju utječe na to hoće li se izvorne varijable moći mijenjati.

U programskom jeziku C++ razlikujemo dva osnovna načina prosljeđivanja parametara:

1. **prosljeđivanje po vrijednosti (engl. *call by value*)**
2. **prosljeđivanje po referenci (engl. *call by reference*).**

1. Prosljeđivanje po vrijednosti (engl. *call by value*)

Kod ovog načina funkcija dobiva **kopiju vrijednosti varijable**. To znači da funkcija radi s vlastitom kopijom podatka pa promjene unutar funkcije **ne utječu na izvornu varijablu**.

Primjer 8.9.

Odredite što će se ispisati na zaslonu računala.

```
#include <iostream>
using namespace std;

void promijeni(int x) {
    x = 10;
}

int main() {
    int a = 5;
    promijeni(a);
    cout << a << endl;
    return 0;
}
```

Rješenje

U funkciji `main` varijabli `a` pridružuje se vrijednost 5. Pozivom funkcije `promijeni(a)` ta se vrijednost prosljeđuje funkciji, ali **po vrijednosti**, što znači da funkcija dobiva kopiju varijable.

Unutar funkcije `promijeni` mijenja se samo kopija (`x = 10`), dok izvorna varijabla `a` ostaje nepromijenjena.

Zato se na kraju ispisuje vrijednost **5**.

Izmijenite program

Dodajte ispis vrijednosti varijable `x` unutar funkcije `promijeni` te usporedite tu vrijednost s vrijednošću varijable `a` nakon poziva funkcije.



2. Prosljeđivanje po referenci (engl. *call by reference*)

Kod ovog načina funkcija dobiva **referencu na varijablu**, odnosno pristup istoj memorij-skoj lokaciji. To znači da funkcija može mijenjati izvornu varijablu.

Primjer 8.10.

Odredite što će se ispisati na zaslonu računala.

```
#include <iostream>
using namespace std;

void promijeni(int &x) {
    x = 10;
}

int main() {
    int a = 5;
    promijeni(a);
    cout << a << endl;
    return 0;
}
```

Rješenje

Budući da se varijabla prosljeđuje s pomoću reference, promjena u funkciji utječe i na izvornu varijablu pa se na kraju ispisuje vrijednost 10.

Izmijenite program

Dodajte ispis vrijednosti varijable *x* unutar funkcije *promijeni* te usporedite tu vrijednost s vrijednošću varijable *a* nakon poziva funkcije.



Tablica 8.4 prikazuje usporedbu načina prosljeđivanja parametara u funkcijama.

Tablica 8.4. Usporedba prosljeđivanja parametara po vrijednosti (*call by value*) i po referenci (*call by reference*)

način prosljeđivanja	funkcija zaprima	mogućnost promjene izvorne varijable
<i>call by value</i>	kopiju vrijednosti	ne
<i>call by reference</i>	referencu (istu varijablu)	da

Primjer 8.11.

Što će se ispisati na zaslonu računala nakon izvođenja sljedećeg programa?

```
#include <iostream>
using namespace std;

void f1(int x) {
    x = x + 2;
}

void f2(int &x) {
    x = x + 3;
}

int f3(int x) {
    x = x + 4;
    return x;
}

int main() {
    int a = 5;

    f1(a);
    f2(a);
    f1(a);
    a = f3(a);

    cout << a << endl;

    return 0;
}
```

Rješenje

- | | | | |
|-------------|--------------------------|----------------|----------|
| • f1(a) | → call by value | → ne mijenja a | → a = 5 |
| • f2(a) | → call by reference | → mijenja a | → a = 8 |
| • f1(a) | → call by value | → ne mijenja a | → a = 8 |
| • a = f3(a) | → call by value + return | → mijenja a | → a = 12 |

Proširite program

Napišite funkciju koja prima cijeli broj po referenci i ako je negativan pretvara ga u pozitivan.


Primjer 8.12.

Ana i Tin odlučili su međusobno zamijeniti svoje školske ormariće. Budući da ormarići imaju digitalne natpise s imenima, potrebno je zamijeniti prikazana imena putem računala povezanog s ormarićima.

Napišite funkciju koja zamjenjuje imena na dva ormarića te program koji ispisuje stanje natpisa prije i nakon zamjene.

U glavnoj funkciji main najprije se ispisuje trenutno stanje natpisa na ormarićima, zatim se poziva funkcija zamijeni, nakon čega se ponovno ispisuje stanje natpisa nakon zamjene.

Prije zamjene:



Nakon zamjene:



ulazni podatci	izlazni podatci
Ana Tin	Prije zamjene: ormaric1 = Ana, ormaric2 = Tin Nakon zamjene: ormaric1 = Tin, ormaric2 = Ana

Rješenje

```
#include <iostream>
#include <string>
using namespace std;
void zamijeni(string &a, string &b) {
    string temp = a;
    a = b;
    b = temp;
}

int main() {

    string ormaric1, ormaric2;

    cout << "Unesite imena na ormaricima: ";
    cin >> ormaric1 >> ormaric2;

    cout << "Prije zamjene: "
         << "ormaric1 = " << ormaric1
         << ", ormaric2 = " << ormaric2 << endl;

    zamijeni(ormaric1, ormaric2);

    cout << "Nakon zamjene: "
         << "ormaric1 = " << ormaric1
         << ", ormaric2 = " << ormaric2 << endl;

    return 0;
}
```

Funkcija za zamjenu mora upotrijebiti prosljeđivanje po referenci (engl. *call by reference*) jer je potrebno promijeniti stvarne vrijednosti varijabli u glavnom programu.

Ako bi se upotrijebilo prosljeđivanje po vrijednosti (engl. *call by value*), funkcija bi dobila samo kopije imena s ormarića. U tom bi se slučaju zamjena dogodila samo unutar funkcije, dok bi izvorne vrijednosti u glavnom programu ostale nepromijenjene.

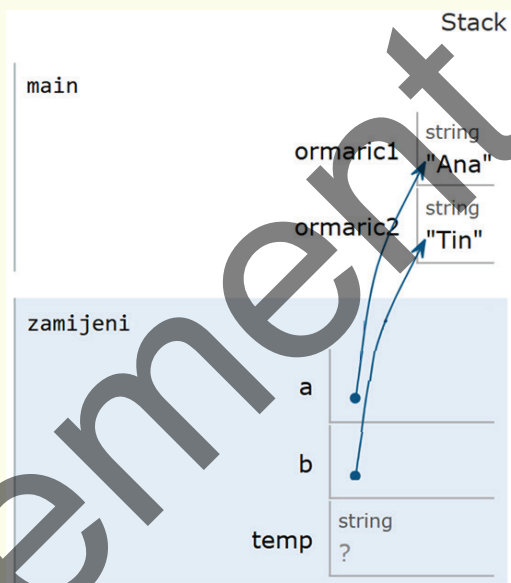
Zato je potrebno upotrebljavati reference kako bi funkcija mogla izravno pristupiti i promijeniti originalne varijable.

Slika 8.8 prikazuje memoriju tijekom izvođenja funkcije za zamjenu vrijednosti s pomoću referenci u alatu *Python Tutor*.

Varijable `ormaric1` i `ormaric2` nalaze se u glavnoj funkciji i sadrže vrijednosti "Ana" i "Tin".

Unutar funkcije `zamijeni`, parametri `a` i `b` su reference koje pokazuju na iste memorijske lokacije kao i te varijable.

To znači da promjene nad `a` i `b` izravno utječu na varijable `ormaric1` i `ormaric2`.



Slika 8.8. Prikaz memorije tijekom izvođenja funkcije za zamjenu vrijednosti s pomoću referenci i uloga pomoćne varijable `temp` u postupku zamjene

Pomoćna varijabla `temp` služi za privremeno spremanje vrijednosti jedne varijable tijekom zamjene. Bez nje bi se jedna vrijednost izgubila jer bi se prilikom pridruživanja (`a = b`) prebrisala izvorna vrijednost. Tablica 8.5 prikazuje postupak zamjene vrijednosti korak po korak.

Tablica 8.5. Prikaz promjene vrijednosti varijabli tijekom zamjene uz upotrebu pomoćne varijable `temp`

korak	naredba	ormaric1 (a)	ormaric2 (b)	temp
početno stanje		Ana	Tin	—
1	<code>temp = a</code>			Ana
2	<code>a = b</code>	Tin		
3	<code>b = temp</code>		Ana	

Slika 8.9 prikazuje stvarni ispis programa u terminalu nakon unosa imena na ormarićima i izvođenja funkcije za zamjenu natpisa.

```
Unesite imena na ormaricima: Ana Tin
Prije zamjene: ormaric1 = Ana, ormaric2 = Tin
Nakon zamjene: ormaric1 = Tin, ormaric2 = Ana

Process returned 0 (0x0)   execution time : 6.170 s
Press any key to continue.
```

Slika 8.9. Ispis programa prije i nakon zamjene imena na školskim ormarićima s pomoću funkcije koja upotrebljava prosljeđivanje po referenci

8.4. Prosljeđivanje polja funkcijama

U prethodnim primjerima funkcijama su se prosljeđivale pojedinačne varijable. Međutim, u programima se često radi s većim skupovima podataka koji su pohranjeni u poljima. Kako bi se takvi podatci mogli obrađivati unutar funkcija, potrebno je omogućiti prosljeđivanje cijelog polja funkciji.

Pri tome je važno uočiti da se kod prosljeđivanja polja funkciji ne stvara kopija svih elemenata, već funkcija dobiva pristup izvornim podatcima u memoriji.

Zbog toga funkcija može:

- čitati vrijednosti elemenata polja
- mijenjati njihove vrijednosti.

8.4.1. Prosljeđivanje jednodimenzionalnog polja

U jeziku C++ polja se u funkcije ne prenose kao kopije, već se prosljeđuje adresa njihova prvog elementa. U nastavku se prikazuje opći oblik funkcije koja prima jednodimenzionalno polje.

Opći oblik

Funkcija koja prima polje najčešće se zapisuje ovako:

```
void nazivFunkcije(tip polje[], int velicina) {
    for (int i = 0; i < velicina; i++) {
        // obrada elemenata polja
    }
}
```

Funkcija koja prima polje poziva se jednostavnim navođenjem naziva polja kao argumenta, bez zagrada ili dodatnih simbola. Tim pozivom funkcija dobiva pristup memorijskoj lokaciji originalnog polja pa se svaka promjena unutar funkcije trenutačno odražava na izvorne podatke. Pri prosljeđivanju polja funkciji potrebno je **uz polje uvijek prosljeđiti i njegovu veličinu** kako bi funkcija mogla pravilno obraditi sve elemente.

Primjer 8.13.

Što će se ispisati na zaslonu računala?

```
#include <iostream>
using namespace std;

void promijeni(int polje[], int n) {
    for (int i = 0; i < n; i++) {
        polje[i] = polje[i] + 1;
    }
}

int main() {
    int a[3] = {1, 2, 3};

    promijeni(a, 3);

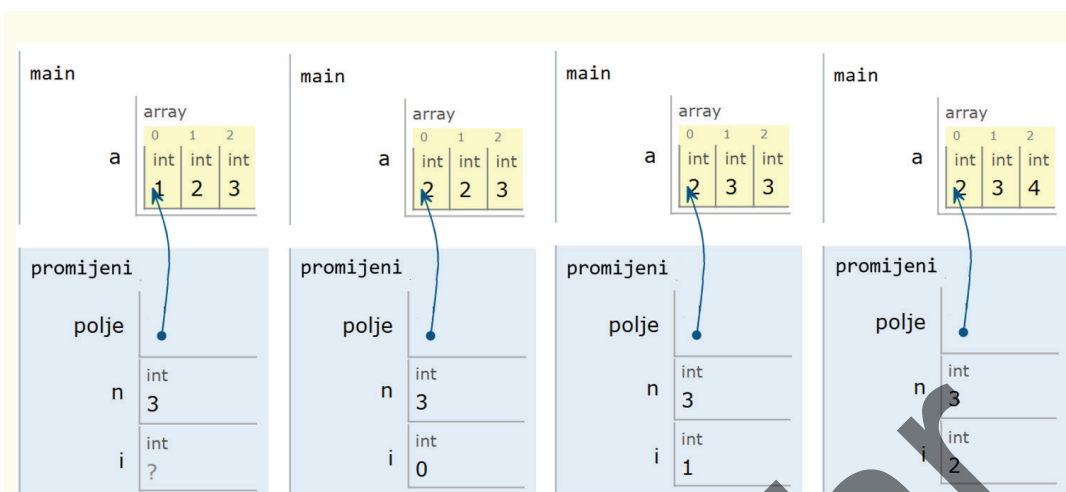
    for (int i = 0; i < 3; i++) {
        cout << a[i] << " ";
    }

    return 0;
}
```

Rješenje

Funkcija `promijeni` prolazi svim elementima polja i povećava svaki za 1. Budući da funkcija radi s izvornim poljem, promjene se odmah vide u funkciji `main`.

Slika 8.10 prikazuje postupak promjene vrijednosti elemenata polja tijekom izvođenja funkcije. U svakom koraku mijenja se jedan element polja, ovisno o vrijednosti indeksa `i`.



Slika 8.10. Priказ postupnog mijenjanja elemenata polja tijekom izvođenja funkcije

Izmijenite program

Napišite funkciju za ispis polja te njome ispišite polje prije i nakon poziva funkcije `promijeni`.



8.4.2. Prosljeđivanje dvodimenzionalnog polja

Dvodimenzionalno polje također se prenosi prosljeđivanjem adrese, pri čemu je u deklaraciji funkcije obavezno navesti broj stupaca kako bi se ispravno pristupilo memorijskim lokacijama elemenata. Budući da funkcija putem te adrese izravno pristupa originalnom polju, svaka promjena vrijednosti unutar dvostruke petlje trajno mijenja izvorne podatke. U nastavku se prikazuje opći oblik funkcije koja prima dvodimenzionalno polje.

Opći oblik

Funkcija koja prima dvodimenzionalno polje najčešće se zapisuje ovako:

```
void nazivFunkcije(tip polje[][broj_stupaca], int broj_redaka) {
    for (int i = 0; i < broj_redaka; i++) {
        for (int j = 0; j < broj_stupaca; j++) {
            // obrada elemenata polja
        }
    }
}
```

Funkcija se poziva navođenjem samo naziva polja, dok se broj redaka prosljeđuje kao dodatni, zasebni parametar. Na taj način funkcija dobiva potpunu informaciju o dimenzijama polja s kojim treba raditi, koristeći fiksni broj stupaca iz definicije i prosljeđeni broj redaka iz poziva.

Primjer 8.14.

Što će se ispisati na zaslonu računala?

```
#include <iostream>
using namespace std;

void povecaj(int polje[][2], int redci) {
    for (int i = 0; i < redci; i++) {
        for (int j = 0; j < 2; j++) {
            polje[i][j] = polje[i][j] + 1;
        }
    }
}

int main() {
    int a[2][2] = {
        {1, 2},
        {3, 4}
    };

    povecaj(a, 2);

    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            cout << a[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Rješenje

Funkcija prolazi svim elementima dvodimenzionalnog polja (redcima i stupcima) i povećava svaki element za 1. Budući da se polje prosljeđuje funkciji, promjene se odmah vide u funkciji main. Zbog toga se početne vrijednosti

1	2
3	4

mijenjaju u

2	3
4	5

Izmijenite program

Napiši funkciju za ispis dvodimenzionalnog polja te njome ispiši polje prije i nakon poziva funkcije povecaj.



Primjer 8.15.

Učenik je tijekom tjedna bilježio dnevne promjene temperature (u stupnjevima) koje mogu biti pozitivne i negativne.

Napišite funkcije **Upisi**, **Ispisi** i **Prebroji**.

- Funkcija **Upisi** učitava n izmjerenih temperatura u jednodimenzionalno polje.
- Funkcija **Ispisi** ispisuje sve unesene temperature (svaku u novi red).
- Funkcija **Prebroji** vraća koliko se negativnih temperatura nalazi u polju.

U glavnoj funkciji main potrebno je učitati broj mjerenja n, zatim pozvati funkcije **Upisi**, **Ispisi** i **Prebroji** te ispisati koliko je bilo negativnih temperatura.

Maksimalan broj mjerenja je 100 i treba ga definirati kao simboličku konstantu.

ulazni podatci	izlazni podatci
5 3 -2 5 -1 0	1. temperatura: 3 2. temperatura: -2 3. temperatura: 5 4. temperatura: -1 5. temperatura: 0 Broj negativnih temperatura: 2

Rješenje

```
#include <iostream>
#define MAX 100
using namespace std;

// unos elemenata
void Upisi(double polje[], int n) {
    for (int i = 0; i < n; i++) {
        cin >> polje[i];
    }
}

// ispis elemenata
void Ispisi(double polje[], int n) {
    for (int i = 0; i < n; i++) {
        cout << i + 1 << ". temperatura: " << polje[i] << endl;
    }
}

// brojanje negativnih elemenata
int Prebroji(double polje[], int n) {
    int brojac = 0;
    for (int i = 0; i < n; i++) {
        if (polje[i] < 0) {
            brojac++;
        }
    }
    return brojac;
}
```



```

int main() {

    int n;
    double temp[MAX];

    // unos broja temperatura s provjerom
    do {
        cout << "Unesite broj temperatura (1-100): ";
        cin >> n;
    } while (n < 1 || n > MAX);

    // unos i obrada
    Upisi(temp, n);
    Ispisi(temp, n);

    cout << "Broj negativnih temperatura: "
         << Prebroji(temp, n) << endl;

    return 0;
}

```

Objašnjenja funkcija:

Upisi

`void Upisi(double polje[], int n)`

- **Tip:** void (ne vraća vrijednost)
- **Zašto:** funkcija samo učitava podatke u polje, nema potrebe ništa vraćati
- **Napomena:** polje se mijenja direktno, jer se prosljeđuje funkciji

Ispisi

`void Ispisi(double polje[], int n)`

- **Tip:** void (ne vraća vrijednost)
- **Zašto:** funkcija služi za ispis podataka na zaslon
- **Napomena:** funkcija upotrebljava indeks `i` kako bi ispisala redni broj svake temperature čime se dobiva pregledniji i jasniji ispis

Prebroji

`int Prebroji(double polje[], int n)`

- **Tip:** int (vraća cijeli broj)
- **Zašto:** funkcija treba vratiti broj negativnih elemenata
- **Napomena:** uvjet `polje[i] < 0` omogućuje prepoznavanje negativnih vrijednosti, dok varijabla brojac služi za njihovo prebrojavanje

Funkciju **Prebroji** moguće je napisati i na sljedeći način:

```
void Prebroji(double polje[], int n, int &brojNegativnih) {
    brojNegativnih = 0;

    for (int i = 0; i < n; i++) {
        if (polje[i] < 0) {
            brojNegativnih++;
        }
    }
}
```

U tom slučaju funkcija se u glavnom programu poziva na sljedeći način:

```
int broj;
Prebroji(temp, n, broj);
cout << "Broj negativnih temperatura: " << broj << endl;
```

U ovom slučaju vrijedi:

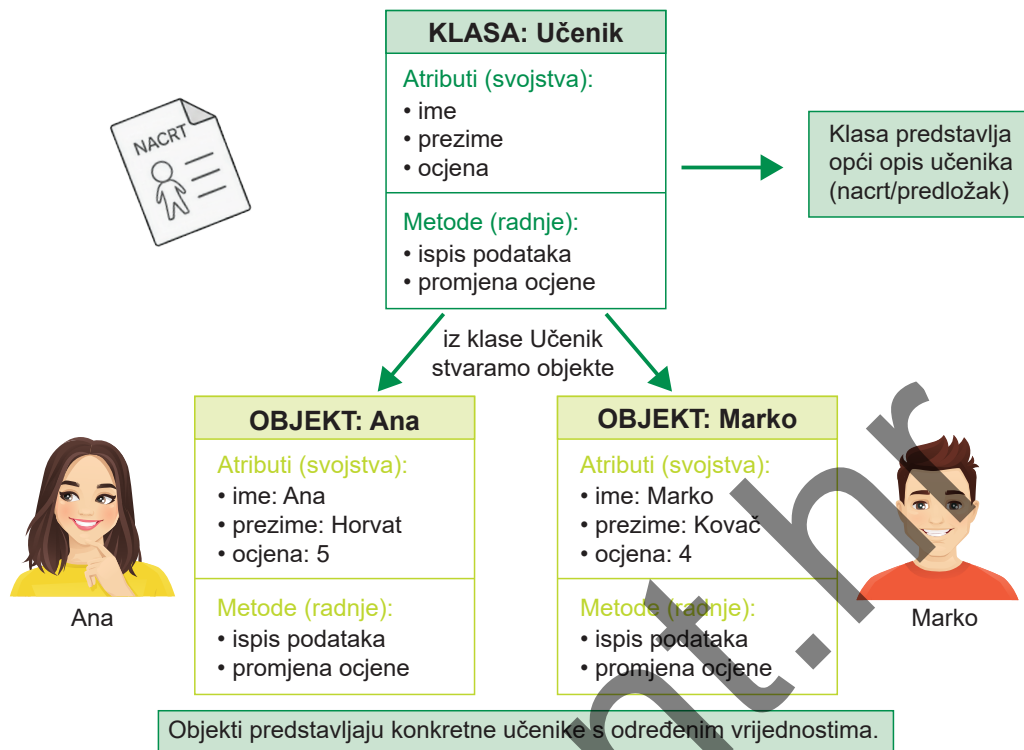
- **Tip funkcije:** void (ne vraća vrijednost)
- **Zašto:** rezultat se ne vraća s pomoću return, već putem reference
- **Napomena:** parametar int &brojNegativnih omogućuje izravnu promjenu vrijednosti varijable u glavnoj funkciji main

8.5. Uvod u klase i metode

U programiranju postoje različiti načini organizacije programa. Jedan od njih je objektno orijentirano programiranje (OOP), u kojem se podatci i funkcije koje nad njima rade povezuju u jedinstvene cjeline. Takve cjeline u C++-u nazivaju se **klase**.

Klase omogućuju pregledno organiziranje programa jer povezuju podatke i funkcije u jednu cjelinu. Podatci unutar klase nazivaju se **atributi**, a funkcije koje rade nad tim podacima **metode**. Klasa sama po sebi ne izvršava program, već služi kao opis (predložak, nacrt).

Objekt se stvara iz klase, pri čemu iz jedne klase možemo stvoriti više objekata. Primjerice, ako imamo klasu **Učenik**, tada možemo stvoriti objekte kao što su **Ana** i **Marko**. Klasa predstavlja opći opis učenika, dok objekti predstavljaju konkretne učenike s određenim vrijednostima.



Tablica 8.6 prikazuje različite primjere klasa, njihovih objekata, atributa i metoda.

Tablica 8.6. Primjeri klasa, objekata, atributa i metoda iz različitih svakodnevnih situacija

primjer	klasa	objekti	atributi (svojstva)	metode (radnje)
škola	Učenik	Ana, Marko	ime, prezime, ocjena	ispis podataka, promjena ocjene
tehnologija	Mobitel	iPhone, Samsung	model, baterija, memorija	pozivanje, slanje poruke
igre	Igrač	Player1, Player2	ime, bodovi, razina	povećanje bodova, prelazak razine
nastava	Predmet	Matematika, Informatika	naziv, broj sati, ocjena	izračun prosjeka, ispis

8.5.1. Opći prikaz klase i objekta

Opći prikaz prikazuje **osnovnu strukturu klase** i **objekta** u programskom jeziku C++. **Klasa** definira podatke (**atribute**) i funkcije (**metode**) koje nad tim podatcima djeluju.

Pristupni dio **public** označava da su atributi i metode dostupni izvan klase.

U glavnoj funkciji `main` stvara se **objekt** na temelju **klase**, čime se dobiva konkretan primjer s vlastitim vrijednostima podataka. Nad **objektom** se zatim mogu pozivati **metode** i pristupati **atributima**.

8. Funkcije, klase i metode u C++

```
#include <iostream>
using namespace std;

class NazivKlase {
public:
    // atribut (podatak)
    tip atribut;

    // metoda (funkcija)
    tip metoda() {
        // naredbe koje koriste atribut
    }
};

int main() {

    // stvaranje objekta
    NazivKlase objekt;

    // korištenje atributa
    objekt.atribut = vrijednost;

    // poziv metode
    objekt.metoda();

    return 0;
}
```

Sljedeća tablica prikazuje povezanost osnovnih elemenata klase s njihovim značenjem i ulogom u programu.

Tablica 8.7. Osnovni elementi klase i njihova uloga u programu

element u kodu	značenje
class NazivKlase	definicija klase
public	omogućuje pristup atributima i metodama izvan klase
atribut	podatak unutar klase
metoda()	funkcija unutar klase
NazivKlase objekt;	stvaranje objekta
objekt.atribut	pristup podatku
objekt.metoda()	poziv metode

Primjer 8.16.

Napišite klasu **Ucenik** koja sadrži atribute *ime*, *prezime* i *ocjena*. Unutar klase definirajte metode za ispis podataka i promjenu ocjene.

U glavnom programu stvorite dva objekta (**Ana** i **Marko**) te im dodijelite sljedeće vrijednosti:

- Ana: ime = *Ana*, prezime = *Horvat*, ocjena = 5
- Marko: ime = *Marko*, prezime = *Kovac*, ocjena = 4

Nakon toga ispišite njihove podatke. Zatim promijenite Markovu ocjenu u 5 i ponovno ispišite njegove podatke.

ulazni podatci	izlazni podatci
	Podatci o Ani: Ime: Ana Prezime: Horvat Ocjena: 5 Podatci o Marku: Ime: Marko Prezime: Kovac Ocjena: 4 Marko nakon promjene ocjene: Ime: Marko Prezime: Kovac Ocjena: 5

Rješenje

```
#include <iostream>
using namespace std;

// definicija klase
class Ucenik {
public:
    // atributi (svojstva)
    string ime;
    string prezime;
    int ocjena;

    // metoda (radnja)
    void ispisi() {
        cout << "Ime: " << ime << endl;
        cout << "Prezime: " << prezime << endl;
        cout << "Ocjena: " << ocjena << endl;
    }
}
```

```
void promijeniOcjenu(int novaOcjena) {
    ocjena = novaOcjena;
}

};

int main() {

    // stvaranje objekata
    Ucenik Ana;
    Ucenik Marko;

    // dodjela vrijednosti atributima
    Ana.ime = "Ana";
    Ana.prezime = "Horvat";
    Ana.ocjena = 5;

    Marko.ime = "Marko";
    Marko.prezime = "Kovac";
    Marko.ocjena = 4;

    // korištenje metoda
    cout << "Podatci o Ani:" << endl;
    Ana.ispisi();

    cout << endl;

    cout << "Podatci o Marku:" << endl;
    Marko.ispisi();

    // promjena ocjene
    Marko.promijeniOcjenu(5);

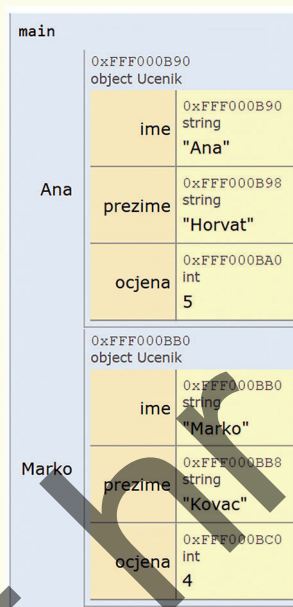
    cout << endl;
    cout << "Marko nakon promjene ocjene:" << endl;
    Marko.ispisi();

    return 0;
}
```

Slika 8.17 prikazuje način na koji se objekti klase **Ucenik** pohranjuju u memoriji tijekom izvođenja programa. Slika prikazuje dva objekta klase **Ucenik**: *Ana* i *Marko*. Svaki objekt ima svoje atribute (*ime*, *prezime*, *ocjena*) i pripadajuće vrijednosti.

Uz svaki atribut prikazana je i memorijska adresa, što pokazuje gdje se pojedini podatak nalazi u memoriji računala. Vidljivo je da svaki objekt zauzima zaseban prostor u memoriji te da ima vlastite vrijednosti atributa.

Tako objekt *Ana* sadrži vrijednosti "*Ana*", "*Horvat*", 5, dok objekt *Marko* sadrži vrijednosti "*Marko*", "*Kovac*", 4. Promjena vrijednosti jednog objekta ne utječe na drugi objekt jer su pohranjeni na različitim memorijskim lokacijama. Promjena ocjene objekta *Marko* ne utječe na ocjenu objekta *Ana*.



Izmijenite program

Stvorite i treći objekt klase **Ucenik** te ispišite njegove podatke.

Slika 8.17. Prikaz objekata klase **Ucenik** i njihovih atributa u memoriji (prije promjene Markove ocjene)

Primjer 8.17.

Učenik piše ispite iz različitih predmeta na državnoj maturi. Za svaki ispit poznat je broj osvojenih bodova, ocjena i ostvareni rang.

Napišite klasu **Ispit** koja sadrži atribute: **brojBodova**, **ocjena** i **rang**.

Unutar klase definirajte metode:

- **unesiPodatke()** – kojom se unose podatci o ispitu znanja (bodovi, ocjena i rang)
- **ispisi()** – kojom se ispisuju podatci o ispitu znanja

U glavnoj funkciji **main** stvorite dva objekta koji predstavljaju ispite iz Matematike i Informatike. Za svaki ispit unesite podatke.

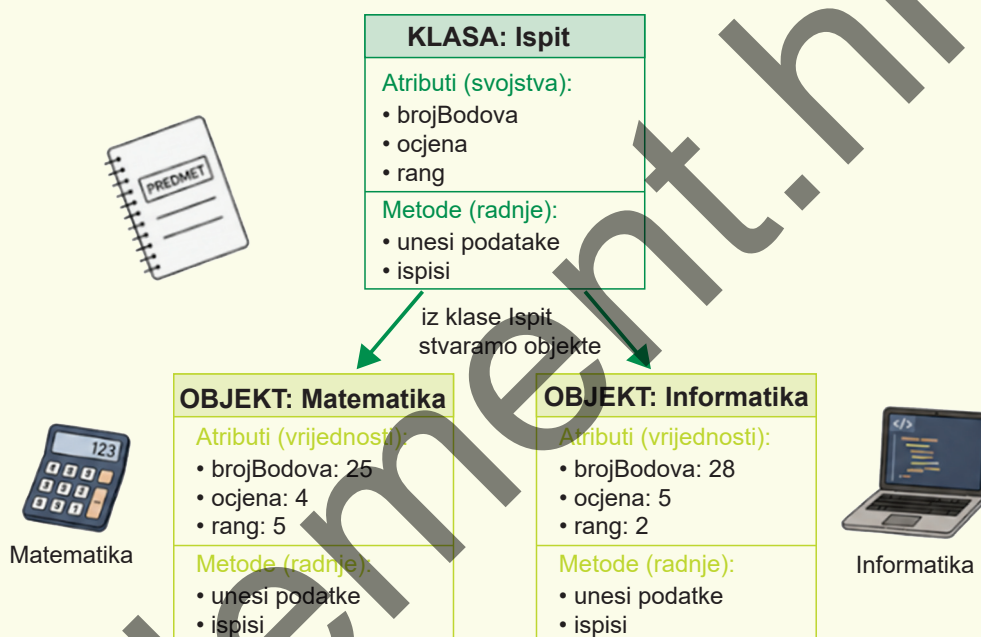
Za ispite kod kojih je učenik ostvario rang među prva tri mjesta u državi ($\text{rang} \leq 3$), unutar metode **ispisi()** potrebno je, uz broj bodova, ocjenu i rang, dodatno:

- ispisati ocjenu riječima
- izračunati i ispisati postotak ostvarenih bodova ako je maksimalan broj bodova na svim ispitima jednak i iznosi 30 bodova.

Ako učenik nije bio među prva tri mjesta, ispisuju se samo broj bodova, ocjena i rang. Pretpostavite ispravan upis svih podataka.

8. Funkcije, klase i metode u C++

ulazni podatci	izlazni podatci
25 4 5 28 5 2	Ispit znanja 1 (Matematika): Broj bodova: 25 Ocjena: 4 Rang: 5 Ispit znanja 2 (Informatika): Broj bodova: 28 Ocjena: 5 Rang: 2 Ocjena rijecima: odlican Postotak: 93.33%



Rješenje

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

class Ispit {
public:
    int brojBodova;
    int ocjena;
    int rang;

    void unesiPodatke() {
        cout << "Unesite broj bodova: ";
        cin >> brojBodova;

        cout << "Unesite ocjenu (1-5): ";
        cin >> ocjena;
```



```

        cout << "Unesite rang: ";
        cin >> rang;
    }

    string ocjenaRijecima() {
        if (ocjena == 5) return "odlican";
        if (ocjena == 4) return "vrlo dobar";
        if (ocjena == 3) return "dobar";
        if (ocjena == 2) return "dovoljan";
        return "nedovoljan";
    }

    void ispisi(string naziv) {
        cout << naziv << endl;
        cout << "Broj bodova: " << brojBodova << endl;
        cout << "Ocjena: " << ocjena << endl;
        cout << "Rang: " << rang << endl;

        if (rang <= 3) {
            cout << "Ocjena rijecima: " << ocjenaRijecima() << endl;
            double postotak = (brojBodova / 30.0) * 100;
            cout << fixed << setprecision(2);
            cout << "Postotak: " << postotak << "%" << endl;
        }
    }
};

int main() {
    Ispit matematika, informatika;

    cout << "Unesite podatke za ispit znanja iz Matematike (bodovi ocjena rang):" << endl;
    matematika.unesiPodatke();

    cout << endl;

    cout << "Unesite podatke za ispit znanja iz Informatike (bodovi ocjena rang):" << endl;
    informatika.unesiPodatke();

    cout << endl;

    matematika.ispisi("Ispit znanja 1 (Matematika):");
    cout << endl;
    informatika.ispisi("Ispit znanja 2 (Informatika):");

    return 0;
}

```

Metodi `ispisi()` proslijeđuje se tekst koji se prvo ispisuje kao naslov, nakon čega metoda ispisuje podatke o ispitu.

Izmijenite program

Dodajte još jedan objekt klase `Ispit` te ispišite podatke za njega.





VRIJEME ZA IZAZOV

Učenik i ocjena

Radna situacija

U školi želiš napraviti program koji će pohraniti podatke o učeniku i na temelju ostvarenih bodova odrediti njegovu ocjenu.

Zadatak

1. Napiši program u kojem definiraš klasu Učenik.

Klasa treba sadržavati:

- ime učenika
- broj bodova (cijeli broj) .

2. U klasi napiši:

- metodu za ispis podataka o učeniku
- metodu koja na temelju broja bodova određuje i ispisuje ocjenu prema sljedećim uvjetima:
 - manje od 50 → Nedovoljan (1)
 - 50 – 62 → Dovoljan (2)
 - 63 – 75 → Dobar (3)
 - 76 – 88 → Vrlo dobar (4)
 - 89 – 100 → Odličan (5)



3. U glavnom programu potrebno je:

- stvoriti jedan objekt klase Učenik
- unesti podatke
- pozvati obje metode.

4. Rezultat ispiši pregledno na zaslonu računala.

Primjer ulaznih i izlaznih podataka

ulazni podatci	izlazni podatci
Ana 78	Učenik: Ana, bodovi: 78, Vrlo dobar (4)
Marko 45	Učenik: Marko, bodovi: 45, Nedovoljan (1)

Elementi vrednovanja

elementi provjere	bodovi
definirana klasa i atributi	5
ispravno definirane metode	6
primjena grananja za određivanje ocjene	8
unos i ispis podataka	8
točan i pregledan ispis rezultata	3
ukupno	30

Ocjena

- 27 – 30 = odličan (5)
- 23 – 26 = vrlo dobar (4)
- 19 – 22 = dobar (3)
- 15 – 18 = dovoljan (2)
- 0 – 14 = nedovoljan (1)

Dodatni zadatak za one koji žele znati više

Proširi program tako da omogući unos podataka za n učenika. Za svakog učenika treba ispisati ime, broj bodova i ocjenu, a na kraju ispisati ime učenika koji je ostvario najveći broj bodova.

Vrednovanje kao učenje



<https://s.element.hr/1703>

9.

VEKTORI



DIGITALNI RUDNIK KODOVA

Skeniraj QR kôd ili otvori poveznicu u internetskom pregledniku. Prouči primjere 9.3, 9.6 i 9.10 te ih analiziraj i proširi programe prateći upute iz tih primjera.

<https://s.element.hr/cb9c>





Razmisli

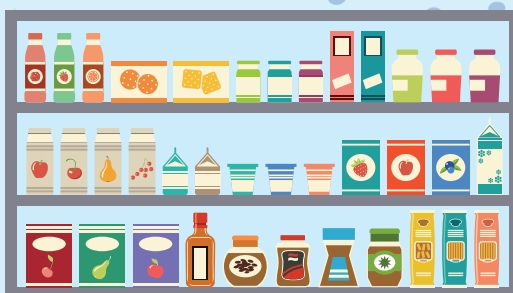
Ako program pohranjuje broj kupaca u trgovini tijekom dana, a njihov broj se stalno mijenja, je li bolje odabrati polje fiksne veličine ili strukturu koja se može mijenjati? Objasni zašto.



Nakon ove lekcije moći će



- objasniti što je vektor i koja je njegova prednost u odnosu na polje
- deklarirati i inicijalizirati vektor
- dodavati i uklanjati elemente s pomoću metoda `push_back` i `pop_back`
- pristupati elementima vektora s pomoću indeksa i petlji
- primijeniti metode `size`, `empty`, `front` i `back`
- prolaziti vektorom s pomoću `for` i `for-each` petlji
- primijeniti vektore u rješavanju jednostavnih problemskih zadataka.



9.1. Uvod u vektore

U programskim jezicima C i C++ spremanje više vrijednosti unutar jedne varijable najjednostavnije se postiže primjenom jednodimenzijskog polja (engl. *array*). Međutim, čim se jednom stvori, veličina polja ostaje fiksna tijekom cijelog izvođenja programa.

Kako bi se prevladalo to ograničenje, u C++ standardnoj biblioteci odabire se spremnik pod nazivom `std::vector`. Vektor se upotrebljava kada unaprijed nije poznat broj potrebnih elemenata. Predstavlja **proširivu inačicu običnog polja i omogućuje dinamičko povećavanje ili smanjivanje svoje veličine** ovisno o potrebama programa.

Vektori omogućuju fleksibilno i učinkovito rukovanje podacima. Oni su temelj mnogih naprednijih struktura podataka, kao što su dinamičke liste, stogovi i redovi. Te strukture čine sastavne dijelove složenijih sustava, poput operativnih sustava i sustava za upravljanje bazama podataka.

Osim toga, vektori omogućuju **jednostavniju implementaciju** algoritama koji zahtijevaju **dodavanje ili uklanjanje elemenata** tijekom rada programa, što je čest slučaj u praktičnom programiranju. Na primjer, mnogi algoritmi za obradu podataka, sortiranje i pretraživanje oslanjaju se upravo na fleksibilnost koju pružaju vektori.

Za razliku od programskog jezika C, u kojem je potrebno ručno upravljati memorijom, u C++-u vektori automatski upravljaju memorijom, čime se smanjuje mogućnost pogrešaka.

Stoga će se u sljedećem poglavlju, koristeći se programskim jezikom C++, objasniti osnovni način korištenja vektora, uključujući njihovu deklaraciju, dodavanje elemenata, pristup elementima te osnovne operacije nad vektorima.

9.2. Osnove rada s vektorima u programskom jeziku C++

Za razliku od programskog jezika C, u kojem je potrebno ručno implementirati vektor, u programskom jeziku C++ takva funkcionalnost već postoji u standardnoj biblioteci. C++ nudi spremnik `std::vector` koji predstavlja dinamičko polje i automatski upravlja memorijom.

To znači da programer ne mora ručno alocirati i oslobađati memoriju niti brinuti o proširivanju kapaciteta. Svi ti detalji implementacije skriveni su od korisnika koji se vektorom koristi putem unaprijed definiranih funkcija.

U nastavku će se prikazati kako deklarirati i inicijalizirati vektor te kako započeti rad s njegovim elementima.

9.2.1. Deklaracija i inicijalizacija vektora

Inicijalizacija varijabli prilikom programiranja je nužna jer se time sprječava upotreba varijabli čije stanje nije poznato, što može dovesti do pogrešaka i neočekivanog ponašanja programa.

Za razliku od programskog jezika C, u kojem je za inicijalizaciju vektora potrebno ručno postaviti početne vrijednosti i alocirati memoriju, u programskom jeziku C++ inicijalizacija vektora znatno je jednostavnija jer se provodi automatski prilikom deklaracije.

U primjerima će se primijeniti naredba `using namespace std;` kako bi se pojednostavio zapis koda. U stvarnim programima često se preporučuje upotreba punog naziva `std::`:

Opći oblik deklaracije vektora u programskom jeziku C++ je:

```
vector<T> naziv_vektora;
```

gdje T predstavlja tip podataka koji se pohranjuje u vektor (npr. `int`, `float`, `double`, `char`), a `naziv_vektora` predstavlja ime vektora.

Vektor se, na primjer, može deklarirati i inicijalizirati na sljedeći način:

```
#include <vector>
using namespace std;

vector<int> v;
```

U prikazanom primjeru deklariran je prazan vektor cjelobrojnih vrijednosti. Vektor u početnom stanju ne sadrži nijedan element, **a memorija se dinamički dodjeljuje prema potrebi tijekom rada programa**. Vektor se u ovom slučaju ponaša kao prazno dinamičko polje koje će se popunjavati tijekom izvođenja programa.

Primjer 9.1.

Napišite program u kojem je potrebno deklarirati vektore tipova `int`, `double`, `char` i `string` te jedan vektor cijelih brojeva deklarirati i inicijalizirati vrijednostima 10, 20, 30 i 40.

Rješenje

```
#include <vector>
#include <string>
using namespace std;

int main() {

    // deklaracija vektora
    vector<int> brojevi;
    vector<double> decimalni;
    vector<char> znakovi;
    vector<std::string> tekst;

    // deklaracija i inicijalizacija vektora
    vector<int> v = {10, 20, 30, 40};

    return 0;
}
```


U ovom slučaju vektor se odmah popunjava zadanim vrijednostima. Slika 9.1 prikazuje inicijalizirani vektor koji sadrži četiri elementa. Svakom elementu vektora pridružen je odgovarajući indeks, stoga se vektor ponaša kao polje, odnosno ima definirane indekse i jasan redoslijed elemenata. Prvi element ima indeks 0, drugi indeks 1, treći indeks 2 i tako redom.

```
vector<int> v = {10, 20, 30, 40};
```

	10	20	30	40
indeksi:	0	1	2	3

Slika 9.1. Elementi vektora s njihovim indeksima

Također, moguće je deklarirati vektor s unaprijed određenim brojem elemenata:

```
vector<T> naziv_vektora(broj_elementa);
```

Primjer 9.2.

Napišite program u kojem je potrebno deklarirati vektor tipa `int` s 5 elemenata te skicirati njegov izgled i označiti indekse elemenata.

Rješenje

```
#include <vector>
using namespace std;

int main() {
    // deklaracija i inicijalizacija vektora s 5 elemenata
    vector<int> v(5);
    return 0;
}
```

Ovim se stvara vektor koji sadrži pet elemenata inicijaliziranih na zadanu vrijednost (za tip `int` to je 0). Tako inicijalizirani vektor može se konceptualno prikazati kao što je to prikazano na slici 9.2.

```
vector<int> v(5);
```

	0	0	0	0	0
indeksi:	0	1	2	3	4

Slika 9.2. Inicijalizacija vektora s pet elemenata čija je vrijednost nula

Nakon inicijalizacije vektor se može odmah primijeniti, bez potrebe za dodatnim funkcijama za upravljanje memorijom.

9.2.2. Prolazak elementima vektora

Za prolazak elementima vektora može se odabrati:

a) klasična **for petlja**, pri čemu se **elementima pristupa s pomoću indeksa**

Ako se odabere klasična for petlja, elementima vektora pristupa se s pomoću indeksa, pri čemu je važno znati koliko elemenata vektor sadrži jer to određuje granice petlje prilikom prolaska njegovim elementima. Elementima vektora pristupa se s pomoću indeksa, kao i kod običnog polja.

Funkcija `size()` vraća broj elemenata u vektoru pa se primjenjuje u uvjetu petlje kako bi se osiguralo da se pristupa samo postojećim elementima.

Kao što je već objašnjeno kod rada s klasama, objektima i metodama, zapis oblika `objekt.metoda()` odabire se za poziv metode nad određenim objektom, pri čemu se koristimo operatorom točka (`.`).

Tako, na primjer, zapis `v.size()` znači da se nad objektom `v` (vektorom) poziva njegova metoda `size()` koja vraća **broj elemenata vektora `v`**.

b) **for-each petlja**, koja omogućuje jednostavniji prolazak svim elementima **izravno** po njihovim vrijednostima

Ako se odabere for-each petlja, nije potrebno upotrijebiti indekse niti funkciju `size()` jer petlja automatski prolazi svim elementima vektora. U tom slučaju pristupa se izravno vrijednostima elemenata, što čini kod jednostavnijim i preglednijim.

Opći zapis (for-each petlja):

```
for (tip varijabla : niz/vektor) {
    // naredbe
}
```

pri čemu je

- `tip` → tip elementa (npr. `int`, `double`, ...)
- `varijabla` → privremena varijabla koja predstavlja svaki element
- `niz/vektor` → niz ili vektor kroz koji se prolazi.

Primjer 9.3.

Deklarirajte i inicijalizirajte vektor `v` tipa `int` s vrijednostima 5, 10, 15 i 20 te ispišite njegove elemente.

Rješenje

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    /* Inicijalizacija vektora s
       početnim vrijednostima */
    vector<int> v = {5, 10, 15, 20};

    // Ispis elemenata vektora
    cout << "Elementi vektora: ";

    for (int i = 0; i < v.size(); i++) {
        cout << v[i] << " ";
    }

    return 0;
}
```

Koristimo se funkcijom `size()` kako bismo vratili broj elemenata u vektoru.

Elementima se pristupa s pomoću indeksa, kao kod običnog polja (`v[i]`).

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    /* Inicijalizacija vektora s
       početnim vrijednostima */
    vector<int> v = {5, 10, 15, 20};

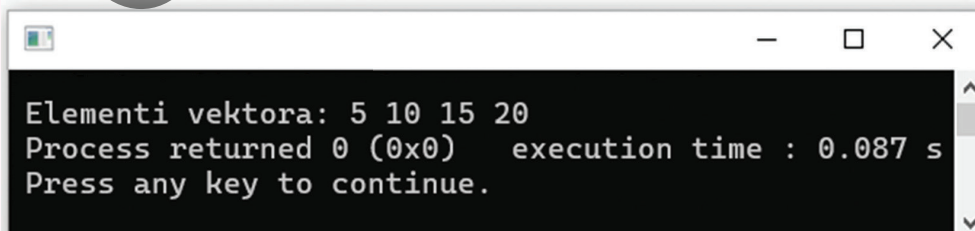
    // Ispis elemenata vektora
    cout << "Elementi vektora: ";

    for (int x : v) {
        cout << x << " ";
    }

    return 0;
}
```

For-each petlja prolazi svim elementima vektora bez potrebe za korištenjem indeksa i funkcije `size()`, što je čini jednostavnijom i preglednijom.

Slika 9.3 prikazuje rezultat izvođenja programa za ispis elemenata vektora.



Slika 9.3. Ispis elemenata vektora na zaslonu računala

Proširite program

Dodajte još jedan element u vektor i ponovno ispišite sve njegove elemente na dva načina (for petljom i for-each petljom).



9.2.3. Dodavanje i uklanjanje elemenata vektora

Vektori u programskom jeziku C++ nude niz ugrađenih metoda (funkcija) koje omogućuju jednostavan rad s podacima. Upravo dostupnost tih metoda čini rad s vektorima jednostavnijim i preglednijim jer programer ne mora ručno upravljati memorijom niti implementirati osnovne operacije nad poljem, kao što je to slučaj u programskom jeziku C. Umjesto toga, upotrebljava unaprijed definirane operacije za **dodavanje, uklanjanje i pristup elementima**.

Tablica 9.1 prikazuje dvije metode `push_back` i `pop_back` koje se upotrebljavaju za dodavanje elemenata na kraj vektora, odnosno uklanjanje zadnjeg elementa.

Tablica 9.1. Metode za rad s vektorima – dodavanje i uklanjanje elemenata

metoda	opis	primjer
<code>push_back(x)</code>	dodaje element na kraj vektora	<code>v.push_back(10);</code>
<code>pop_back()</code>	uklanja zadnji element	<code>v.pop_back();</code>

Metoda `push_back(x)` dodaje novi element na kraj vektora. Svakim pozivom metode veličina vektora povećava se za jedan element, a novi element postaje zadnji u nizu.

Primjer 9.4.

Odredite što će program ispisati na zaslonu računala.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> brojevi;

    // Dodavanje elemenata u vektor
    brojevi.push_back(10);
    brojevi.push_back(20);
    brojevi.push_back(30);

    // Ispis elemenata vektora
    cout << "Elementi vektora: ";
    for (int i = 0; i < brojevi.size(); i++) {
        cout << brojevi[i] << " ";
    }

    return 0;
}
```

Rješenje

Slika 9.4 prikazuje dodavanje elemenata u vektor primjenom metode `push_back()`. Najprije se vektor inicijalizira (prva linija koda u `main()` funkciji), nakon čega se elementi dodaju jedan po jedan, pri čemu svaki novi element postaje zadnji u vektoru.

1. Na početku je vektor prazan.

brojevi

--	--	--	--	--

size() = 0
(broj elemenata)

2. Nakon `brojevi.push_back(10);`

brojevi

10				
----	--	--	--	--

↑

size() = 1
(broj elemenata)

3. Nakon `brojevi.push_back(20);`

brojevi

10	20			
----	----	--	--	--

↑

size() = 2
(broj elemenata)

4. Nakon `brojevi.push_back(30);`

brojevi

10	20	30		
----	----	----	--	--

↑

size() = 3
(broj elemenata)

Konačno:

brojevi

10	20	30
----	----	----

 elementi vektora:
10 20 30

size() = 3
(broj elemenata)

– slobodno mjesto u memoriji (još neiskorišteno)

 – elementi vektora

Slika 9.4. Dodavanje elemenata u spremnik vektor

Program ispisuje sve elemente vektora redom, onako kako su dodani, prolaskom vektorom s pomoću `for` petlje. Ispis su vrijednosti **10, 20 i 30** koje su prethodno dodane u vektor.

Izmijenite program

Kako biste ispisali te elemente vektora korištenjem `for-each` petlje?



Primjer 9.5.

Odredite što će program ispisati na zaslonu računala.

```
int main() {
    vector<int> brojevi;

    // Dodavanje elemenata
    brojevi.push_back(10);
    brojevi.push_back(20);
    brojevi.push_back(30);

    // Uklanjanje zadnjeg elementa
    brojevi.pop_back();
}
```

```
// Ispis elemenata vektora
cout << "Elementi vektora nakon uklanjanja: ";
for (int i = 0; i < brojevi.size(); i++) {
    cout << brojevi[i] << " ";
}

return 0;
}
```

Rješenje

Slika 5.5 prikazuje elemente vektora tijekom i nakon dodavanja. Zadnji dodani element, označen ljubičastom bojom, uklanja se primjenom metode `pop_back()`. Nakon uklanjanja tog elementa u vektoru ostaju dva elementa, brojevi 10 i 20, koji se potom ispisuju.

1. Na početku je vektor prazan.

brojevi

--	--	--	--

size() = 0
(broj elemenata)

2. Nakon dodavanja elemenata:

brojevi

10	20	30	
----	----	----	--

size() = 3
(broj elemenata)

elementi vektora

3. Poziv metode `pop_back()`.

brojevi

10	20	30	
----	----	----	--

size() = 2
(broj elemenata nakon uklanjanja)

elementi vektora

uklanja se zadnji element

4. Nakon uklanjanja zadnji element više nije u vektoru.

brojevi

10	20		
----	----	--	--

size() = 2
(broj elemenata)

elementi vektora

Konačno:

brojevi

10	20
----	----

size() = 2
(broj elemenata)

elementi vektora:
10 20

elementi vektora

- | |
|--|
| |
|--|

 – slobodno mjesto u memoriji (još neiskorišteno)
- | |
|--|
| |
|--|

 – elementi vektora
- | |
|--|
| |
|--|

 – element koji se uklanja
- | |
|--|
| |
|--|

 – mjesto koje postaje slobodno nakon uklanjanja elementa

Slika 9.5. Uklanjanje elementa iz spremnika vektor

Izmijenite program

Nakon uklanjanja elementa dodajte novi element u vektor i ponovno ispišite sve njegove elemente.



Primjer 9.6.

Deklarirajte vektor cjelobrojnih vrijednosti, popunite ga brojevima od 2 do 20 te ispišite samo parne brojeve.

ulaz	izlaz
	2 4 6 8 10 12 14 16 18 20

Rješenje

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> brojevi;

    // Popunjavanje vektora brojevima od 2 do 20
    for (int i = 2; i <= 20; i++) {
        brojevi.push_back(i);
    }

    // Ispis samo parnih brojeva
    cout << "Parni brojevi: ";
    for (int x : brojevi) {
        if (x % 2 == 0) {
            cout << x << " ";
        }
    }

    return 0;
}
```

Objašnjenje:

- `push_back(i)` → dodaje sve brojeve od 2 do 20 u vektor
- `for-each` → prolazi svim elementima
- `if (x % 2 == 0)` → provjerava je li broj paran
- `cout << x << " "` → ispisuju se samo parni brojevi.

Proširite program

Učitajte donju i gornju granicu intervala, zatim popunite vektor brojevima iz tog intervala i ispišite samo neparne brojeve.

**9.2.4. Unos elemenata u vektor**

Elementi vektora mogu se unositi i s tipkovnice s pomoću naredbe `cin`. U tom slučaju upotrebljava se petlja kojom se omogućuje višestruki unos, a svaki uneseni element dodaje se u vektor s pomoću metode `push_back()`. Na taj način vektor se dinamički popunjava tijekom izvođenja programa.

Primjer 9.7.

Napišite program koji omogućuje unos pet cijelih brojeva u vektor primjenom metode `push_back()`. Nakon unosa uklonite zadnji uneseni element primjenom metode `pop_back()` te ispišite preostale elemente vektora.

ulaz	izlaz
12 56 34 45 100	Preostali elementi vektora: 12 56 34 45

Rješenje

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> brojevi;
    int unos;

    // Unos 5 brojeva
    cout << "Unesite 5 cijelih brojeva:" << endl;
    for (int i = 0; i < 5; i++) {
        cin >> unos;
        brojevi.push_back(unos);
    }

    // Uklanjanje zadnjeg elementa
    brojevi.pop_back();

    // Ispis preostalih elemenata
    cout << "Preostali elementi vektora: ";
    for (int i = 0; i < brojevi.size(); i++) {
        cout << brojevi[i] << " ";
    }

    return 0;
}
```

U programu se najprije deklarira prazan vektor te se unosom pet cijelih brojeva elementi dodaju u vektor primjenom metode `push_back()`. Nakon toga metoda `pop_back()` uklanja zadnji uneseni element. Preostali elementi ispisuju se primjenom `for` petlje, pri čemu se broj elemenata određuje metodom `size()`.

```
Unesite 5 cijelih brojeva:
12
56
34
45
100
Preostali elementi vektora: 12 56 34 45
Process returned 0 (0x0) execution time : 8.601 s
Press any key to continue.
```

Slika 9.6. Ispis elemenata vektora nakon uklanjanja zadnjeg elementa

9.2.5. Pristup elementima vektora

Elementima vektora pristupa se na isti način kao elementima jednodimenzionalnog polja, odnosno s pomoću indeksa. Indeks predstavlja redni broj elementa u vektoru, pri čemu prvi element ima indeks 0. Pristup elementu vektora ostvaruje se s pomoću operatora uglatih zagrada [].

Uz ugate zagrade može se primijeniti i metoda `at()` prikazana u tablici 9.2.

Tablica 9.2. Metoda za rad s vektorima – pristup elementima

metoda	opis	primjer
<code>at(i)</code>	siguran pristup elementu (provjera granice)	<code>v.at(0);</code>

Primjer 9.8.

Deklarirajte i inicijalizirajte vektor `v` tipa `int` s vrijednostima 10, 20, 30 i 40 te ispišite prvi i treći element vektora:

- a) s pomoću indeksa
- b) s pomoću metode `at()`.

ulaz	izlaz
	Prvi element: 10 Treci element: 30

Rješenje

a)

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> v = {10, 20, 30, 40};

    cout << "Prvi element: " << v[0] << endl;
    cout << "Treci element: " << v[2] << endl;

    return 0;
}
```

Kôd prikazuje kako pristupiti elementima vektora s pomoću indeksa 0 i 2.

Važno je napomenuti da operator [] ne provjerava je li indeks unutar granica vektora. Ako se pokuša pristupiti elementu koji ne postoji, može doći do neispravnog rada programa.

b)

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> v = {10, 20, 30, 40};

    cout << "Prvi element: " << v.at(0) << endl;
    cout << "Treci element: " << v.at(2) << endl;

    return 0;
}
```

U situacijama kada je potrebna dodatna sigurnost, može se upotrijebiti i metoda `at()`, koja provjerava granice indeksa.

Sljedeći kôd prikazuje kako upotrijebiti metodu `at()`. Ako se funkciji `at()` proslijedi nevažeći indeks, program će generirati pogrešku.

Izmijenite program

Dodajte još jedan element u vektor te ispišite njegov posljednji element s pomoću indeksa i s pomoću metode `at()`.



9.2.6. Veličina i kapacitet vektora

Vektori u programskom jeziku C++ posjeduju svojstva veličine i kapaciteta koja omogućuju učinkovito upravljanje memorijom. Veličina vektora predstavlja broj elemenata koji se trenutno nalaze u vektoru, dok kapacitet označava koliko elemenata vektor može pohraniti prije nego što je potrebno ponovno alocirati memoriju.

Za dohvat tih vrijednosti primjenjuju se odgovarajuće metode koje prikazuje tablica 9.3.

Tablica 9.3. Metode za rad s vektorima – dohvat elemenata, kapacitet i promjena veličine

metoda	opis	primjer
<code>size(i)</code>	siguran pristup elementu (provjera granice)	<code>v.at(0);</code>
<code>capacity()</code>	vraća trenutni kapacitet vektora	<code>v.capacity();</code>
<code>resize(n)</code>	mijenja veličinu vektora na n elemenata	<code>v.resize(5);</code>

Primjer 9.9.

Odredite što će se ispisati na zaslonu računala.

Nacrtajte izgled vektora:

- nakon dodavanja elemenata
- nakon poziva metode `resize(5)`.

Objasnite kako se mijenja veličina (`size`) vektora.

```

#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> v;

    // Dodavanje elemenata
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);

    cout << "Nakon dodavanja elemenata:" << endl;
    cout << "Velicina (size): " << v.size() << endl;
    cout << "Kapacitet (capacity): " << v.capacity() << endl;

    // Promjena veličine vektora
    v.resize(5); // povećavamo veličinu na 5 elemenata

    cout << "\nNakon resize(5):" << endl;
    cout << "Velicina (size): " << v.size() << endl;
    cout << "Kapacitet (capacity): " << v.capacity() << endl;

    // Ispis elemenata
    cout << "Elementi vektora: ";
    for (int i = 0; i < v.size(); i++) {
        cout << v[i] << " ";
    }

    return 0;
}

```

Rješenje

U prikazanom programskom kodu najprije se deklarira prazan vektor cjelobrojnih vrijednosti. Zatim se u vektor dodaju elementi primjenom metode `push_back()`, pri čemu se svaki novi element dodaje na kraj vektora, a njegova se veličina povećava.

Nakon dodavanja elemenata ispisuju se veličina i kapacitet vektora. Metoda `size()` vraća broj elemenata koji se trenutno nalaze u vektoru, dok metoda `capacity()` prikazuje koliko elemenata vektor može pohraniti bez potrebe za dodatnom alokacijom memorije. **Kapacitet vektora ne povećava se linearno, već način njegova povećavanja ovisi o implementaciji standardne biblioteke (npr. može se udvostručavati).**

Pozivom metode `resize(5)` veličina vektora povećava se na pet elemenata. Budući da je vektor prije toga sadržavao tri elementa, **dodaju se još dva nova elementa koji se inicijaliziraju na zadanu vrijednost tipa `int`, odnosno na 0.**

Slika 9.7 prikazuje postupak dodavanja elemenata u vektor s pomoću metode `push_back()` te promjenu njegove veličine i kapaciteta.

1. Nakon dodavanja elemenata (push_back)

1. `vector<int> v;` (prazan vektor)

0	1	2	3
—	—	—	—

`size()`

0

`capacity()`

0

2. `v.push_back(10);` (dodaje 10 na kraj)

0	1	2	3
10	—	—	—

1

1

kapacitet postaje 1
(prvo povećanje)

3. `v.push_back(20);` (dodaje 20 na kraj)

0	1	2	3
10	20	—	—

2

2

kapacitet postaje 2
(drugo povećanje)

4. `v.push_back(30);` (dodaje 30 na kraj)

0	1	2	3
10	20	30	—

3

4

kapacitet postaje 4
(treće povećanje, ima mjesta za još jedan element)

2. Nakon `resize 5`

0	1	2	3	4	5
10	20	30	0	0	—

`size() = 5`

`capacity() = 6`

Slika 9.7. Prikaz rasta vektora – dodavanje elemenata s pomoću `push_back()` i stanje nakon `resize(5)`

Na kraju se elementi vektora ispisuju primjenom `for` petlje. Granica petlje određena je metodom `size()`, čime se osigurava da se ispisuju svi elementi koji se nalaze u vektoru. Ispis na zaslonu računala prikazuje slika 9.8.

```

Nakon dodavanja elemenata:
Velicina (size): 3
Kapacitet (capacity): 4

Nakon resize(5):
Velicina (size): 5
Kapacitet (capacity): 6
Elementi vektora: 10 20 30 0 0
Process returned 0 (0x0)   execution time : 0.073 s
Press any key to continue.
    
```

Slika 9.8. Izlaz programa – prikaz veličine i kapaciteta vektora nakon dodavanja elemenata i nakon poziva metode `resize(5)`

Izmijenite program

Doradite program tako da se nakon svakog koraka ispišu **veličina** (`size`) i **kapacitet** (`capacity`) vektora.

Potrebno je ispisati stanje vektora:

- na početku programa
- nakon `push_back(10)`
- nakon `push_back(20)`
- nakon `push_back(30)`
- nakon `resize(5)`.



9.2.7. Najčešće metode spremnika `std::vector`

Spremnik `std::vector` u programskom jeziku C++ nudi velik broj metoda koje omogućuju učinkovito i jednostavno upravljanje elementima. Te metode predstavljaju sučelje putem kojeg programer može dodavati, uklanjati i pristupati elementima bez potrebe za izravnim upravljanjem memorijom.

U prethodnim potpoglavljima već su prikazane neke od osnovnih operacija nad vektorima, poput dodavanja i uklanjanja elemenata te određivanja veličine vektora. U ovom potpoglavlju nalazi se pregled najčešće korištenih metoda koje se upotrebljavaju u svakodnevnom radu s vektorima.

Tablica 9.4 prikazuje odabrane metode spremnika `std::vector` s njihovim opisom i primjerima korištenja.

Tablica 9.4. Najčešće metode spremnika `std::vector`

metoda	opis	primjer
<code>push_back(x)</code>	dodaje element na kraj vektora	<code>v.push_back(10);</code>
<code>pop_back()</code>	uklanja zadnji element vektora	<code>v.pop_back();</code>
<code>size()</code>	vraća broj elemenata u vektoru	<code>v.size();</code>
<code>empty()</code>	provjerava je li vektor prazan	<code>v.empty();</code>
<code>clear()</code>	briše sve elemente iz vektora	<code>v.clear();</code>
<code>v[i]</code>	pristup elementu s pomoću indeksa	<code>v[0];</code>
<code>at(i)</code>	siguran pristup elementu (provjera granica)	<code>v.at(0);</code>
<code>front()</code>	vraća prvi element vektora	<code>v.front();</code>
<code>back()</code>	vraća zadnji element vektora	<code>v.back();</code>

Važno je napomenuti da navedene metode pokrivaju najčešće operacije nad vektorima, no spremnik `std::vector` sadrži i dodatne metode koje omogućuju naprednije upravljanje podacima i memorijom.

Primjer 9.10.

Deklarirajte i inicijalizirajte vektor s tri cjelobrojna elementa. Primjenom metoda `front()` i `back()` pronađite i ispišite prvi i zadnji element vektora. Nakon toga, primjenom metode `clear()` izbrišite sve elemente vektora te provjerite je li vektor prazan.

ulaz	izlaz
10 20 30	Prvi element: 10 Zadnji element: 30 Vektor je prazan.

Rješenje

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> v = {10, 20, 30};

    // Ispis prvog i zadnjeg elementa
    cout << "Prvi element: " << v.front() << endl;
    cout << "Zadnji element: " << v.back() << endl;

    // Brisanje svih elemenata
    v.clear();

    // Provjera je li vektor prazan
    if (v.empty()) {
        cout << "Vektor je prazan." << endl;
    } else {
        cout << "Vektor nije prazan." << endl;
    }

    return 0;
}
```

Slika 9.9 prikazuje izlaz programa nakon dohvaćanja prvog i zadnjeg elementa te brisanja svih elemenata vektora.

```
Prvi element: 10
Zadnji element: 30
Vektor je prazan.

Process returned 0 (0x0)   execution time : 0.083 s
Press any key to continue.
```

Slika 9.9. Izvršavanje programskog koda iz primjera 9.10

Proširite program

Nakon brisanja elemenata ponovno dodajte elemente 5, 15 i 25 u vektor i ispišite ih.



9.2.8. Primjena vektora

U nastavku se prikazuje praktični primjer programa koji demonstrira osnovne operacije nad vektorima.

Primjer 9.11.

Napišite program koji simulira praćenje narudžbi u maloj trgovini. Svaka narudžba dobiva svoj broj kako bi se mogla pratiti, a ti se brojevi redom spremaju u vektor.

Na početku korisnik unosi broj narudžbi n . Nakon toga potrebno je unijeti n narudžbi i pohraniti ih u vektor. Tijekom unosa potrebno je prikazati trenutni broj narudžbi i kapacitet vektora.

Nakon unosa svih narudžbi potrebno je ispisati sve spremljene vrijednosti.

Zatim je potrebno ispisati prvu i zadnju narudžbu. Nakon toga uklonite zadnju narudžbu te ispišite novu zadnju narudžbu.

Na kraju obrišite sve narudžbe i ispišite koliko ih je ostalo u vektoru.

ulaz	izlaz
3 10 20 30	Pocetni kapacitet: 0 Dodana narudzba: 10 Velicina: 1 Kapacitet: ... Dodana narudzba: 20 Velicina: 2 Kapacitet: ... Dodana narudzba: 30 Velicina: 3 Kapacitet: ... Narudzbe: 10 20 30 Prva narudzba: 10 Zadnja narudzba: 30 Nakon uklanjanja zadnje narudzbe, nova zadnja narudzba je: 20 Nakon brisanja svih narudzbi u vektoru vise nema elemenata.

Napomena

Kapacitet vektora može imati različite vrijednosti ovisno o implementaciji pa je u tablici pod izlaznim podacima označen trotočkom (...).

Rješenje

Prikazani primjer objedinjuje sve prethodno uvedene metode te pokazuje cjelovit rad s vektorom u programskom jeziku C++, od unosa i dodavanja elemenata do njihova ispisa i brisanja.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {

    vector<int> narudzbe;
    int n, broj;

    cout << "Pocetni kapacitet: " << narudzbe.capacity() << endl;

    cout << "Unesite broj narudzbi: ";
    cin >> n;

    for (int i = 0; i < n; i++) {

        cin >> broj;
        narudzbe.push_back(broj);

        cout << "Dodana narudzba: " << broj << endl;
        cout << "Velicina: " << narudzbe.size() << endl;
        cout << "Kapacitet: " << narudzbe.capacity() << endl << endl;

    }

    cout << "Narudzbe: ";
    for (int x : narudzbe) {
        cout << x << " ";
    }
    cout << endl;

    cout << "Prva narudzba: " << narudzbe.front() << endl;
    cout << "Zadnja narudzba: " << narudzbe.back() << endl;

    narudzbe.pop_back();

    cout << "Nakon uklanjanja zadnje narudzbe, nova zadnja narudzba je: ";
    cout << narudzbe.back() << endl;

    narudzbe.clear();

    if (narudzbe.empty()) {
        cout << "Nakon brisanja svih narudzbi, u vektoru vise nema elemenata."
            << endl;
    }

    return 0;
}
```



VRIJEME ZA IZAZOV

Simulacija bilježenja temperature (dnevnik temperature)

Radna situacija

U elektrotehničkim sustavima mjerenje temperature često se provodi kontinuirano, pri čemu se izmjerene vrijednosti pohranjuju i obrađuju programskim putem. Kako bi se omogućilo bilježenje i analiza izmjerenih temperatura, potrebno je osmisliti digitalni dnevnik mjerenja temperature. Dugotrajno povišena prosječna temperatura može dovesti do smanjenja pouzdanosti rada ili kvara uređaja.



Zadatak

Napišite program koji:

1. rabi vektore za spremanje dnevnih temperatura
2. omogućuje korisniku unos proizvoljnog broja temperatura
3. ispisuje sve unesene temperature
4. izračunava prosječnu temperaturu
5. određuje najvišu i najnižu temperaturu.

Primjer ulaznih i izlaznih podataka

ulazni podatci	izlazni podatci
Koliko temperatura želite unijeti? 5	Unesene temperature: 35 38 40 42 45
Unesite temperaturu 1: 35	Prosječna temperatura je: 40 C
Unesite temperaturu 2: 38	Najviša temperatura je: 45 C
Unesite temperaturu 3: 40	Najniža temperatura je: 35 C
Unesite temperaturu 4: 42	
Unesite temperaturu 5: 45	

Elementi vrednovanja – program i pseudokod

elementi provjere	bodovi
program rabi vektor za spremanje podataka	6
vektor je pravilno inicijaliziran	5
omogućen je unos proizvoljnog broja temperatura	5
sve unesene temperature se ispisuju	3
ispravno je izračunata prosječna temperatura	4
ispravno je određena najviša temperatura	3
ispravno je određena najniža temperatura	3
program se koristi metodama vektora za rad s podacima	1
ukupno	30

Ocjena

- 27 – 30 = odličan (5)
- 23 – 26 = vrlo dobar (4)
- 19 – 22 = dobar (3)
- 15 – 18 = dovoljan (2)
- 0 – 14 = nedovoljan (1)

Dodatni zadatak za one koji žele znati više

Proširite program tako da se omogući unos granične temperature, prebroji koliko dana temperatura prelazi zadanu graničnu vrijednost te se ispisuje upozorenje ako je granična temperatura premašena najmanje dva puta.

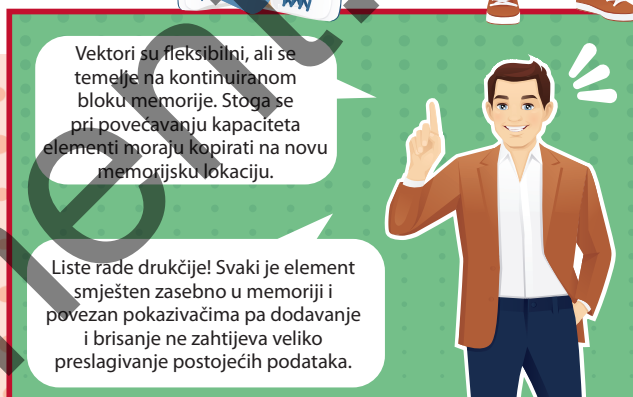
Vrednovanje kao učenje



<https://s.element.hr/45a2>

10.

LISTA



DIGITALNI RUDNIK KODOVA

Skeniraj QR kôd ili otvori poveznicu u internetskom pregledniku. Prouči primjere 10.4 i 10.9 te ih analiziraj i proširi programe prateći upute iz tih primjera.

<https://s.element.hr/999d>





Razmisli

Zamisli da stalno uređuješ listu pjesama, često dodaješ nove pjesme i brišeš neke postojeće.

Ako želiš ubaciti novu pjesmu točno na drugo mjesto, mora li računalo pritom pomicati sve ostale podatke u memoriji ili postoji način da se elementi jednostavno „prespoje“?

Kako bi takva organizacija podataka mogla izgledati u programiranju?



Nakon ove lekcije moći će



- objasniti što je lista i kako se razlikuje od vektora
- deklarirati i inicijalizirati listu
- dodavati i uklanjati elemente s početka i kraja liste
- pristupati prvom i zadnjem elementu liste
- prolaziti elementima liste s pomoću petlje
- primijeniti liste u situacijama gdje se podatci često dodaju i brišu.



10.1. Uvod u liste

Lista, odnosno povezana lista (engl. *linked list*), sastoji se od **niza elemenata koji se nazivaju čvorovi** (engl. *node*). **Svaki čvor sastoji se od dvaju dijelova: podatka i pokazivača. Podatak predstavlja vrijednost koja se pohranjuje u čvoru, dok pokazivač sadržava adresu sljedećeg čvora u nizu.** Na taj su način svi elementi liste povezani u lančanu strukturu. Slika 10.1 prikazuje strukturu čvora jednostruko povezane liste s podatkom i pokazivačem na sljedeći čvor.



Slika 10.1. Čvor u listi

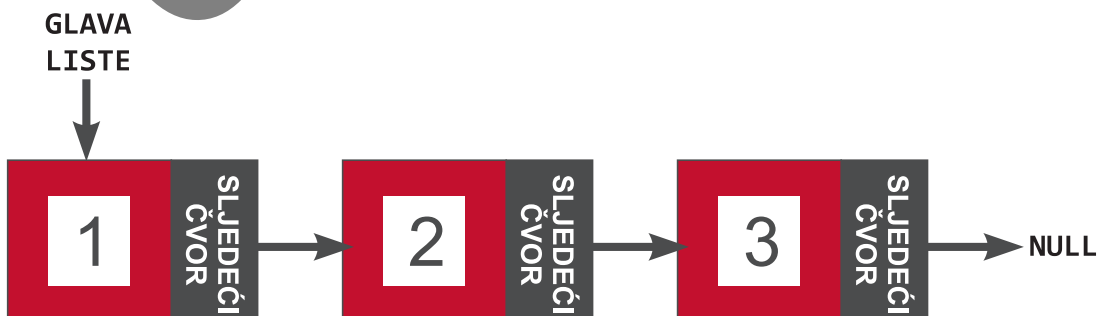
Za razliku od vektora, kod kojih su svi elementi pohranjeni u jednom kontinuiranom bloku memorije, elementi liste smješteni su na različitim mjestima u memoriji. Zbog toga je za pristup određenom elementu potrebno redom prolaziti svim čvorovima koji mu prethode, počevši od prvog elementa liste.

Povezane liste mogu se razvrstati prema svojoj strukturi i smjeru u kojem je moguće pristupiti elementima. Razlikuju se sljedeće vrste povezanih lista:

1. jednostruko povezana lista
2. dvostruko povezana lista
3. kružna povezana lista (kružna lista).

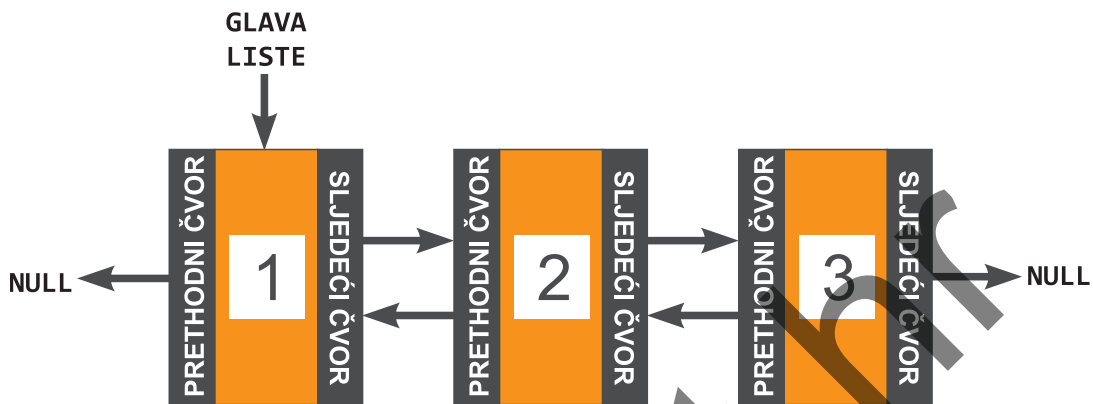
U **jednostruko povezanoj listi** (engl. *singly linked list*) svaki čvor sadržava podatak i pokazivač na sljedeći čvor u nizu. Kretanje listom moguće je samo u jednom smjeru, od prvog prema posljednjem čvoru. Slika 10.2 prikazuje jednostruko povezanu listu sastavljenu od triju čvorova.

Prvi element liste naziva se **glava liste** i predstavlja pokazivač na prvi čvor. Prvi čvor liste sadržava vrijednost 1 i pokazuje na čvor s vrijednošću 2, drugi čvor sadržava vrijednost 2 i pokazuje na čvor s vrijednošću 3, dok treći čvor sadržava vrijednost 3 i pokazuje na konstantu NULL, čime je označen kraj liste. **Posljednji čvor** liste naziva se i **rep liste**.



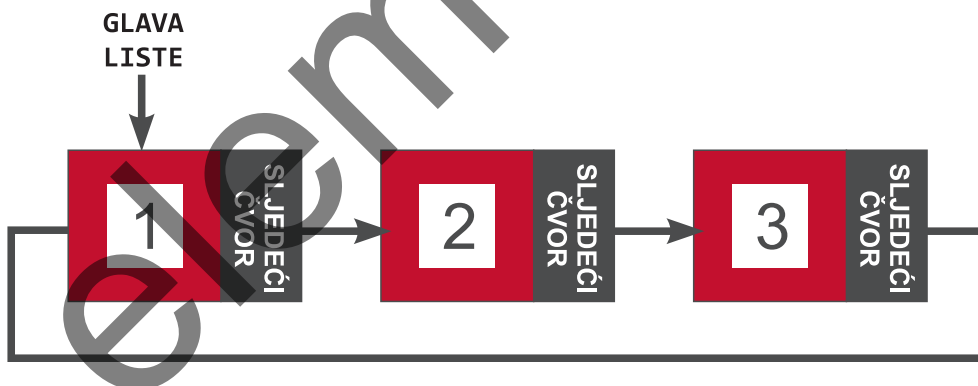
Slika 10.2. Jednostruko povezana lista

U **dvostruko povezanoj listi** (engl. *doubly linked list*) čvor ima dva pokazivača. Jedan pokazivač pokazuje na sljedeći čvor, dok drugi pokazuje na prethodni čvor. Ovakav način povezivanja omogućuje kretanje listom u oba smjera, ali zahtijeva više memorije jer svaki čvor čuva dva pokazivača. Slika 10.3 prikazuje dvostruko povezanu listu sastavljenu od triju čvorova.



Slika 10.3. Dvostruko povezana lista

Kružna lista (engl. *circular linked list*) prema načinu povezivanja elemenata u listi potpuno je jednaka jednostruko povezanoj listi s jednom razlikom: u kružnoj listi posljednji čvor ne pokazuje na konstantu NULL, nego na prvi čvor u listi. Na taj način lista tvori krug što je korisno u situacijama u kojima je potrebno stalno kružno kretanje elementima. Slika 10.4 prikazuje kružnu listu koja se sastoji od triju čvorova. Posljednji čvor pokazuje natrag na prvi čvor.



Slika 10.4. Kružna lista

U nastavku ćemo prikazati implementaciju liste u programskom jeziku C++.

10.2. Osnove rada s listama u programskom jeziku C++

Povezane liste mogu se implementirati u programskom jeziku C, koji je ranije opisan u udžbeniku. Takav pristup zahtijeva ručno upravljanje memorijom i implementaciju osnovnih operacija nad listom, poput dodavanja i uklanjanja elemenata.

U programskom jeziku C++ rad s listama znatno je pojednostavljen **upotrebom spremnika `std::list`**, koji je dio standardne biblioteke. Za njegovu upotrebu potrebno je uključiti zaglavlje `<list>`. Ovaj spremnik implementira **dvostruko povezanu listu** te omogućuje jednostavno dodavanje i uklanjanje elemenata na početku i na kraju liste, kao i prolazak elementima.

U nastavku poglavlja prikazat će se osnovni načini rada sa spremnikom `std::list`, uključujući njegovu **deklaraciju, dodavanje i uklanjanje elemenata, pristup elementima te najčešće upotrebljavane metode**.

10.2.1. Deklaracija i inicijalizacija liste

U programskom jeziku C++ povezana lista implementirana je putem spremnika `std::list`, koji **predstavlja dvostruko povezanu listu**. Lista omogućuje učinkovito **dodavanje i uklanjanje elemenata** na bilo kojem mjestu u strukturi.

Radi preglednijeg zapisa koda u primjerima se i u ovom poglavlju navodi naredba `using namespace std;`. U stvarnim programima češće se preporučuje pisanje punog prefiksa `std::`.

Opći oblik deklaracije liste u programskom jeziku C++ glasi:

```
list<T> naziv;
```

gdje **T** predstavlja tip podatka koji se pohranjuje u listi, a **naziv** označava ime liste. Lista može sadržavati više elemenata istog tipa, a njihovo dodavanje i uklanjanje obavlja se upotrebom odgovarajućih metoda spremnika `list`.

Kao i vektori, liste u programskom jeziku C++ omogućuju pohranu elemenata različitih tipova podataka. Tip podataka navodi se unutar zagrada prilikom deklaracije liste, čime se određuje koje će se vrijednosti pohranjivati u listi.

U sljedećim linijama programskog koda prikazani su primjeri **deklaracije lista s različitim tipovima podataka**.

```
list<int> brojevi;
list<double> decimalni;
list<char> znakovi;
list<string> tekstovi;
```

Lista se može inicijalizirati na nekoliko načina:

- a) inicijalizacijom zadanim vrijednostima
- b) stvaranjem liste s određenim brojem elemenata iste vrijednosti
- c) kopiranjem postojeće liste
- d) inicijalizacijom s pomoću iteratora (kopiranjem raspona elemenata).

a) Inicijalizacija zadanim vrijednostima

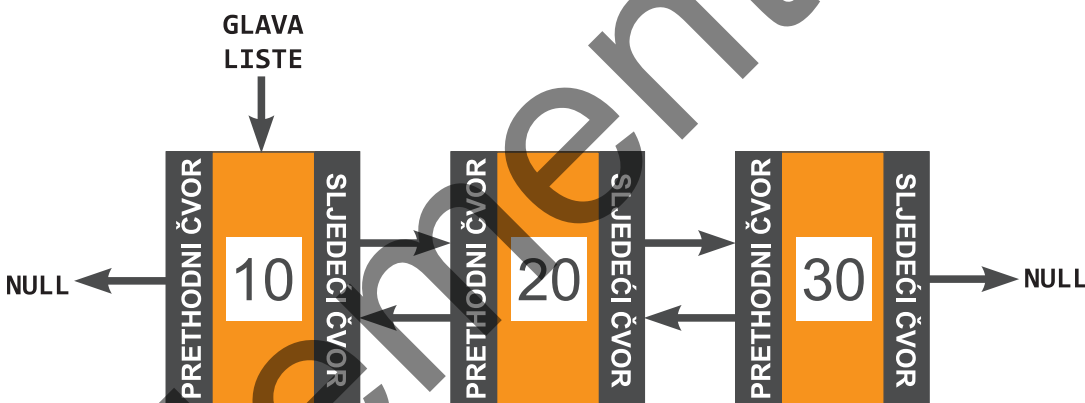
Može se inicijalizirati s unaprijed zadanim elementima. Ovaj je način najjednostavniji kada su vrijednosti poznate unaprijed i želi se odmah raditi s popunjenom listom.

```
list<T> naziv = {vrijednosti};
```

Sljedeća linija koda prikazuje deklariranu i unaprijed inicijaliziranu listu s vrijednostima 10, 20, 30.

```
list<int> brojevi = {10, 20, 30};
```

Slika 10.5 prikazuje izgled dvostruko povezane liste nakon deklaracije i inicijalizacije liste vrijednostima 10, 20 i 30.



Slika 10.5. Inicijalizirana dvostruko povezana lista s elementima 10, 20 i 30

b) Stvaranje liste s određenim brojem elemenata iste vrijednosti

Ako se želi stvoriti lista s točno određenim brojem elemenata n pri čemu svi elementi imaju istu vrijednost, to se može napraviti na sljedeći način:

```
list<T> naziv(n, vrijednost);
```

U sljedećem primjeru oblikuje se lista s pet elemenata, a svaki od njih ima vrijednost 0:

```
list<int> brojevi(5, 0);
```

Ako se lista deklarira bez početnih elemenata, dovoljno je napisati sljedeću liniju koda:

```
list<int> brojevi;
```

Kod ovako deklarirane liste elementi se kasnije dodaju tijekom izvođenja programa, najčešće primjenom metode poput `push_back()` ili `push_front()`.

c) Kopiranje postojeće liste

Nova lista također se može inicijalizirati kao **kopija postojeće liste**.

```
list<T> naziv(druga_lista);
```

Svi elementi iz prve liste prenose se u novu listu, pri čemu svaka lista zauzima zaseban prostor u memoriji. U nastavku se prikazuje primjer koda:

```
list<int> l1 = {1, 2, 3};
list<int> l2(l1);
```

d) Inicijalizacija s pomoću iteratora

Još jedan mogući način inicijalizacije liste je s pomoću **iteratora** koji označava **početak i kraj drugog spremnika**.

Iterator je objekt koji omogućuje prolazak elementima spremnika i pristup njihovim vrijednostima. Na taj način moguće je kopirati sve ili samo dio elemenata iz postojeće liste.

```
list<T> naziv(pocetak, kraj);
```

Primjer prikazuju sljedeće linije koda:

```
list<int> l1 = {1, 2, 3, 4, 5};
list<int> l2(l1.begin(), l1.end());
```

Najčešće oblike deklaracije i inicijalizacije liste prikazuje tablica 10.1.

Tablica 10.1. Oblici deklaracije i inicijalizacije liste u programskom jeziku C++

br.	opći zapis i primjer	objašnjenje
1.	<code>list<T> naziv;</code> <code>list<int> brojevi;</code>	deklarira praznu listu; elementi se dodaju kasnije
2.	<code>list<T> naziv = {vrijednosti};</code> <code>list<int> brojevi = {10, 20, 30};</code>	inicijalizacija poznatim vrijednostima
3.	<code>list<T> naziv(n, vrijednost);</code> <code>list<int> brojevi(5, 0);</code>	lista s n elemenata iste vrijednosti
4.	<code>list<T> naziv(druga_lista);</code> <code>list<int> l2(l1);</code>	kopija postojeće liste
5.	<code>list<T> naziv(pocetak, kraj);</code> <code>list<int> l2(l1.begin(), l1.end());</code>	inicijalizacija s pomoću iteratora

Primjer 10.1.

Napišite deklaracije i inicijalizacije lista prema zadanim uvjetima:

- a)** Deklarirajte i inicijalizirajte listu l1 tipa `int` s vrijednostima **5, 15 i 25**.
- b)** Deklarirajte i inicijalizirajte listu l2 tipa `double` koja sadrži **4 elementa**, pri čemu je svaki element jednak **7.5**.

- c) Deklarirajte praznu listu l3 tipa char.
- d) Deklarirajte i inicijalizirajte listu l4 tipa int kao kopiju liste l1.
- e) Deklarirajte i inicijalizirajte listu l5 tipa int s pomoću iteratora tako da sadrži iste elemente kao lista l1.

Rješenje

```
#include <list>
using namespace std;

int main() {

    // a) Lista int vrijednosti
    list<int> l1 = {5, 15, 25};

    // b) Lista double vrijednosti
    list<double> l2(4, 7.5);

    // c) Prazna lista char
    list<char> l3;

    // d) Kopija liste l1
    list<int> l4(l1);

    // e) Lista s pomoću iteratora
    list<int> l5(l1.begin(), l1.end());

    return 0;
}
```

10.2.2. Pristup elementima liste

Za razliku od vektora lista u programskom jeziku C++ ne omogućuje izravan pristup elementima s pomoću indeksa. To znači da nije moguće upotrijebiti operator [] niti metodu at() za dohvat pojedinog elementa. Elementima liste pristupa se **sekvencijalno, odnosno prolaskom svim elementima redom**.

Za pristup prvom i zadnjem elementu liste upotrebljavaju se metode front() i back(). Metoda front() vraća prvi element liste, dok metoda back() vraća zadnji element. U sljedećem kodu prikazano je kako se koristiti metodama front() i back(). U tablici 10.2. nalazi se opis navedenih metoda. Metode front() i back() mogu se primijeniti samo ako lista nije prazna.

Tablica 10.2. Metode za pristup elementima liste

metoda	opis	primjer
front()	vraća prvi element liste	l.front();
back()	vraća zadnji element liste	l.back();

Primjer 10.2.

Napišite program u kojem je potrebno deklarirati listu cjelobrojnih vrijednosti te je inicijalizirati vrijednostima **10**, **20** i **30**.

Nakon toga potrebno je ispisati **prvi** i **zadnji** element liste.

ulaz	izlaz
	Prvi element: 10 Zadnji element: 30

Rješenje

```
#include <iostream>
#include <list>

using namespace std;

int main() {
    list<int> brojevi = {10, 20, 30};

    cout << "Prvi element: " << brojevi.front() << endl;
    cout << "Zadnji element: " << brojevi.back() << endl;

    return 0;
}
```

Izmijenite program

Dodajte novi element u listu na početak i na kraj te ponovno ispišite prvi i zadnji element liste.



Za prolazak svim elementima liste najčešće se odabire **for-each petlja** koja omogućuje jednostavan i pregledan pristup svim elementima. U sljedećem kodu prikazan je način primjene for-each petlje za prolazak svim elementima liste.

```
for (int x : brojevi) {
    cout << x << " ";
}
```

Za inicijalizaciju liste u prethodnom poglavlju spominju se iteratori. Za prolazak listom mogu se upotrijebiti i iteratori. Iteratori označavaju pozicije unutar liste te omogućuju pristup elementima i kretanje listom. Metode `begin()` i `end()` vraćaju iteratore koji označavaju početak i kraj liste. U sljedećem primjeru iterator `it` prolazi svim elementima liste, pri čemu se operatorom `*` pristupa vrijednosti na koju iterator pokazuje.

```
for (auto it = brojevi.begin(); it != brojevi.end(); ++it) {
    cout << *it << " ";
}
```

Primjer 10.3.

Napišite program koji demonstrira različite načine inicijalizacije liste u programskom jeziku C++. U programu je potrebno deklarirati i inicijalizirati sljedeće liste:

- a) Deklarirajte listu l1 tipa `int` i inicijalizirajte je vrijednostima **10, 20 i 30**.
- b) Deklarirajte listu l2 tipa `int` koja sadrži **5 elemenata**, pri čemu je svaki element jednak **0**.
- c) Deklarirajte praznu listu l3 tipa `int` te provjerite je li prazna.
- d) Deklarirajte listu l4 tipa `int` kao kopiju liste l1.
- e) Deklarirajte listu l5 tipa `int` s pomoću iteratora tako da sadrži iste elemente kao lista l1.

Za svaku listu ispišite odgovarajuće objašnjenje i njezine elemente kako je prikazano tablicom.

ulaz	izlaz
	l1 - inicijalizirana s unaprijed zadanim elementima: 10 20 30 l2 - inicijalizirana s određenim brojem istih elemenata: 0 0 0 0 0 l3 - inicijalizirana kao prazna lista: lista je prazna l4 - inicijalizirana kopiranjem druge liste: 10 20 30 l5 - inicijalizirana s pomoću iteratora: 10 20 30

Rješenje

```
#include <iostream>
#include <list>
using namespace std;

int main() {
    // 1. Lista s unaprijed zadanim elementima
    list<int> l1 = {10, 20, 30};

    // 2. Lista s određenim brojem istih elemenata
    list<int> l2(5, 0);

    // 3. Prazna lista
    list<int> l3;

    // 4. Kopiranje liste
    list<int> l4(l1);

    // 5. Lista s pomoću iteratora
    list<int> l5(l1.begin(), l1.end());

    // Ispis lista

    cout << "l1 - inicijalizirana s unaprijed zadanim elementima: ";
    for (int x : l1) {
        cout << x << " ";
    }
    cout << endl;
}
```

```

cout << "12 - inicijalizirana s određenim brojem istih elemenata: ";
for (int x : l2) {
    cout << x << " ";
}
cout << endl;

cout << "13 - inicijalizirana kao prazna lista: ";
if (l3.empty()) {
    cout << "lista je prazna";
} else {
    for (int x : l3) {
        cout << x << " ";
    }
}
cout << endl;

cout << "14 - inicijalizirana kopiranjem druge liste: ";
for (int x : l4) {
    cout << x << " ";
}
cout << endl;

cout << "15 - inicijalizirana s pomoću iteratora: ";
for (int x : l5) {
    cout << x << " ";
}
cout << endl;

return 0;
}

```

Program prikazuje različite načine inicijalizacije liste te za svaku listu ispisuje način na koji je inicijalizirana i njezine elemente. Prazna lista nema elemenata, što se provjerava metodom `empty()`.

Primjer 10.4.

Napišite program u kojem je potrebno deklarirati i inicijalizirati listu tipa `double` s vrijednostima **5.5**, **10.5**, **15.5** i **20.5**. Ispišite prvi i zadnji element liste primjenom metoda `front()` i `back()`. Nakon toga ispišite sve elemente liste primjenom `for-each` petlje.

Rješenje

```

#include <iostream>
#include <list>
using namespace std;

int main() {

    // Deklaracija i inicijalizacija liste
    list<double> brojevi = {5.5, 10.5, 15.5, 20.5};

    // Ispis prvog i zadnjeg elementa
    cout << "Prvi element: " << brojevi.front() << endl;
    cout << "Zadnji element: " << brojevi.back() << endl;
}

```

```
// Ispis svih elemenata liste korištenjem for-each petlje
cout << "Elementi liste: ";
for (double x : brojevi) {
    cout << x << " ";
}
cout << endl;

return 0;
}
```

U programu se lista inicijalizira s nekoliko vrijednosti tipa `double` (5.5, 10.5, 15.5 i 20.5). Metode `front()` i `back()` upotrebljavaju se za dohvatanje prvog i zadnjeg elementa liste. Nakon toga se svi elementi liste ispisuju primjenom `for-each` petlje koja omogućuje jednostavan prolazak svim elementima liste. Slika 10.6. prikazuje rezultat izvršavanja koda iz primjera 10.4.

```
Prvi element: 5.5
Zadnji element: 20.5
Elementi liste: 5.5 10.5 15.5 20.5

Process returned 0 (0x0)   execution time : 0.287 s
Press any key to continue.
```

Slika 10.6. Rezultat izvođenja programa za pristup elementima liste

Proširite program

Uz postojeću listu napravite novu listu s različitim cijelim brojevima te ispišite elemente obje listi.



10.2.3. Dodavanje i uklanjanje elemenata liste

Spremnik `std::list` u programskom jeziku C++ omogućava jednostavno i učinkovito **dodavanje i uklanjanje elemenata na početku i na kraju liste**. Za razliku od vektora, ove operacije ne zahtijevaju pomicanje ostalih elemenata u memoriji, što ih čini vrlo učinkovitim.

Za **dodavanje elemenata** primjenjuju se metode `push_back()` i `push_front()`. Metoda `push_back()` **dodaje novi element na kraj liste**, dok metoda `push_front()` **dodaje element na početak liste**. Za **umetanje elementa na proizvoljno mjesto u listi** primjenjuje se metoda `insert()`, uz upotrebu iteratora. Pregled prethodno navedenih metoda nalazi se u tablici 10.3.

Tablica 10.3. Metode za dodavanje elemenata u listu

metoda	opis	primjer
<code>push_back(x)</code>	dodaje element na kraj liste	<code>l.push_back(10);</code>
<code>push_front(x)</code>	dodaje element na početak liste	<code>l.push_front(5);</code>
<code>insert(it, x)</code>	dodaje element na određenu poziciju u listi	<code>l.insert(it, 7);</code>

Za **uklanjanje elemenata** iz liste primjenjuju se metode `pop_back()` i `pop_front()`. Metoda `pop_back()` **uklanja element s kraja liste**, dok metoda `pop_front()` **uklanja element s početka liste**. Za **uklanjanje elementa s proizvoljne pozicije** primjenjuje se metoda `erase()` koja zahtijeva upotrebu iteratora. Metoda `clear()` primjenjuje se za **uklanjanje svih elemenata iz liste**. Nakon poziva metode `clear()` lista postaje **prazna**. Popis prethodno navedenih metoda nalazi se u tablici 10.4.

Tablica 10.4. Pregled metoda za uklanjanje elemenata iz liste

metoda	opis	primjer
<code>pop_back()</code>	uklanja zadnji element liste	<code>l.pop_back();</code>
<code>pop_front()</code>	uklanja prvi element liste	<code>l.pop_front();</code>
<code>erase(it)</code>	uklanja element na određenoj poziciji	<code>l.erase(it);</code>
<code>clear()</code>	uklanja sve elemente iz liste	<code>l.clear();</code>

Primjer 10.5.

Napišite program u kojem je potrebno inicijalizirati listu cjelobrojnih vrijednosti s **tri elementa vrijednosti 0**.

Zatim obrišite sve elemente liste i dodajte u listu vrijednosti **2, 4 i 6** koristeći se metodom `push_back()`. Ispišite dobivene elemente.

Nakon toga ponovno obrišite elemente liste te dodajte iste vrijednosti **2, 4 i 6** koristeći se metodom `push_front()`. Ispišite dobivene elemente.

ulaz	izlaz
	Elementi dodani u listu pomocu <code>push_back()</code> metode: 2 4 6 Nakon brisanja liste i ponovnog dodavanja pomocu <code>push_front()</code> metode: 2 4 6

Rješenje

```
#include <iostream>
#include <list>
using namespace std;

int main() {

    // Inicijalizacija liste s 3 elementa vrijednosti 0
    list<int> brojevi(3, 0);

    // Brisanje elemenata i dodavanje s pomoću push_back
    brojevi.clear();
    brojevi.push_back(2);
    brojevi.push_back(4);
    brojevi.push_back(6);
```

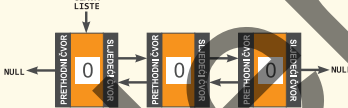
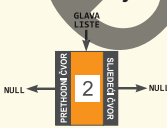
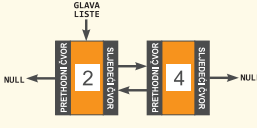
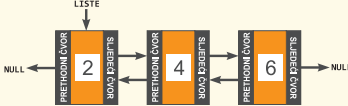
```
// Ispis (push_back)
cout << "Elementi dodani u listu pomocu push_back() metode:" << endl;
for (int x : brojevi) {
    cout << x << " ";
}
cout << endl;

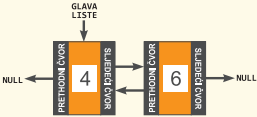
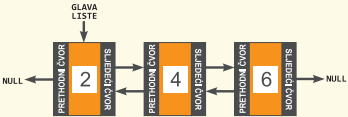
// Brisanje elemenata i dodavanje s pomoću push_front
brojevi.clear();
brojevi.push_front(6);
brojevi.push_front(4);
brojevi.push_front(2);

// Ispis (push_front)
cout << "Nakon brisanja liste i ponovnog dodavanja pomocu push_front()
metode:" << endl;
for (int x : brojevi) {
    cout << x << " ";
}
cout << endl;

return 0;
}
```

U programu se najprije inicijalizira lista s tri elementa vrijednosti 0, nakon čega se lista briše metodom `clear()`. Zatim se elementi 2, 4 i 6 dodaju u listu primjenom metode `push_back()`, pri čemu se dodaju na kraj liste. Nakon ispisa lista se ponovno briše te se isti elementi dodaju primjenom metode `push_front()`, ali obrnutim redoslijedom. Time se postiže isti konačni raspored elemenata u listi (slika 10.7).

1. Inicijalizacija liste s 3 elementa vrijednosti 0 	<pre>list<int>brojevi(3, 0);</pre>
2. Brisanje svih elemenata metodom <code>clear()</code> Lista je prazna	<pre>brojevi.clear();</pre>
3. Dodavanje vrijednosti 2 s pomoću <code>push_back()</code> 	<pre>brojevi.push_back(2);</pre>
4. Dodavanje vrijednosti 4 s pomoću <code>push_back()</code> 	<pre>brojevi.push_back(4);</pre>
5. Dodavanje vrijednosti 6 s pomoću <code>push_back()</code> 	<pre>brojevi.push_back(6);</pre>

6. Brisanje svih elemenata metodom <code>clear()</code> Lista je prazna	<code>brojevi.clear();</code>
7. Dodavanje vrijednosti 6 s pomoću <code>push_front()</code> 	<code>brojevi.front(6);</code>
8. Dodavanje vrijednosti 4 s pomoću <code>push_front()</code> 	<code>brojevi.front(4);</code>
9. Dodavanje vrijednosti 2 s pomoću <code>push_front()</code> 	<code>brojevi.front(2);</code>

Slika 10.7. Vizualni prikaz rada s listom – inicijalizacija, brisanje te dodavanje elemenata s pomoću metoda `push_back()` i `push_front()`

Slika 10.8 prikazuje izlaz programa nakon dodavanja elemenata u listu s pomoću metoda `push_back()` i `push_front()`.

```

Elementi dodani u listu pomocu push_back() metode
2 4 6
Nakon brisanja liste i ponovnog dodavanja pomocu push_front() metode
2 4 6

Process returned 0 (0x0)   execution time : 0.012 s
Press any key to continue.

```

Slika 10.8. Rezultat izvođenja programa – ispis elemenata liste nakon dodavanja s pomoću metoda `push_back()` i `push_front()`

10.2.4. Unos elemenata u listu

Elementi se u listu dodaju unutar `for` petlje, pri čemu se u svakoj iteraciji dodaje novi element s pomoću metode `push_back()` ili `push_front()`.

Dodavanje na kraj liste (`push_back`)

```
for (int i = pocetak; i <= kraj; i++) {
    lista.push_back(vrijednost);
}
```

Svaki novi element dodaje se na kraj liste.

Dodavanje na početak liste (`push_front`)

```
for (int i = pocetak; i <= kraj; i++) {
    lista.push_front(vrijednost);
}
```

Svaki novi element dodaje se na početak liste.

Primjer 10.6.

Promatrajte sljedeći program i nacrtajte kako se lista oblikuje tijekom izvođenja petlje. U programu se u listu redom dodaju vrijednosti **1, 2, 3 i 4**.

```
#include <iostream>
#include <list>
using namespace std;

int main() {
    list<int> brojevi;

    for (int i = 1; i <= 4; i++) {
        brojevi.push_back(i);
    }

    return 0;
}
```

Rješenje

Slika prikazuje postupno formiranje povezone liste dodavanjem elemenata **1, 2, 3 i 4** s pomoću metode `push_back()`, pri čemu se svaki novi element dodaje na kraj liste.

početno stanje (prije petlje)	lista je prazna
1. korak petlje: <code>i = 1</code> <code>brojevi.push_back(1);</code>	
2. korak petlje: <code>i = 2</code> <code>brojevi.push_back(2);</code>	
3. korak petlje: <code>i = 3</code> <code>brojevi.push_back(3);</code>	
4. korak petlje: <code>i = 4</code> <code>brojevi.push_back(4);</code>	

Slika 10.9. Postupno formiranje povezone liste dodavanjem elemenata metodom `push_back()`

Izmijenite program

Nakon dodavanja elemenata u listu ispišite sve elemente liste.



Primjer 10.7.

Promatrajte sljedeći program i nacrtajte kako se lista oblikuje tijekom izvođenja petlje. U programu se u listu redom dodaju vrijednosti **1, 2, 3 i 4**.

```
#include <iostream>
#include <list>
using namespace std;

int main() {

    list<int> brojevi;

    for (int i = 1; i <= 4; i++) {
        brojevi.push_front(i);
    }

    return 0;
}
```

Rješenje

Slika prikazuje postupno formiranje povezanе liste dodavanjem elemenata **1, 2, 3 i 4** s pomoću metode `push_front()`, pri čemu se svaki novi element dodaje na početak liste.

početno stanje (prije petlje)	lista je prazna
1. korak petlje: $i = 1$ <code>brojevi.push_front(1);</code>	
2. korak petlje: $i = 2$ <code>brojevi.push_front(2);</code>	
3. korak petlje: $i = 3$ <code>brojevi.push_front(3);</code>	
4. korak petlje: $i = 4$ <code>brojevi.push_front(4);</code>	

Slika 10.10. Postupno formiranje povezanе liste dodavanjem elemenata metodom `push_front()`

Izmijenite program

Nakon dodavanja elemenata u listu ispišite sve elemente liste i usporedite redoslijed elemenata s prethodnim primjerom.



Primjer 10.8.

Napišite program u kojem je potrebno deklarirati praznu listu cjelobrojnih vrijednosti.

U listu dodajte deset uzastopnih vrijednosti od **1 do 10** primjenom odgovarajuće metode za dodavanje elemenata.

Nakon dodavanja elemenata potrebno je:

- a)** ispisati sve elemente liste
- b)** ukloniti svaki drugi element liste primjenom metode `erase()`
- c)** ponovno ispisati preostale elemente liste
- d)** ukloniti prvi element liste s pomoću metode `pop_front()`
- e)** ukloniti zadnji element liste s pomoću metode `pop_back()`
- f)** ponovno ispisati sve elemente liste
- g)** na kraju ukloniti sve elemente liste primjenom metode `clear()` te provjeriti je li lista prazna.

ulaz	izlaz
	Nakon unosa elemenata: 1 2 3 4 5 6 7 8 9 10 Nakon uklanjanja svakog drugog elementa: 1 3 5 7 9 Nakon uklanjanja prvog i zadnjeg elementa: 3 5 7 Lista je sada prazna.

Rješenje

```
#include <iostream>
#include <list>
using namespace std;

int main() {
    list<int> brojevi;

    // Dodavanje 10 elemenata (1 do 10)
    for (int i = 1; i <= 10; i++) {
        brojevi.push_back(i);
    }

    // Ispis liste nakon unosa
    cout << "Nakon unosa elemenata: ";
    for (int x : brojevi) {
        cout << x << " ";
    }
    cout << endl;
}
```

```

// Uklanjanje svakog drugog elementa
bool brisi = false;
for (auto it = brojevi.begin(); it != brojevi.end(); ) {
    if (brisi) {
        // erase vraća iterator na sljedeći element
        it = brojevi.erase(it);
    } else {
        ++it;
    }
    brisi = !brisi;
}

// Ispis liste nakon brisanja svakog drugog elementa
cout << "Nakon uklanjanja svakog drugog elementa: ";
for (int x : brojevi) {
    cout << x << " ";
}
cout << endl;

// Uklanjanje prvog i zadnjeg elementa
if (!brojevi.empty()) brojevi.pop_front();
if (!brojevi.empty()) brojevi.pop_back();

// Ispis liste nakon dodatnog uklanjanja
cout << "Nakon uklanjanja prvog i zadnjeg elementa: ";
for (int x : brojevi) {
    cout << x << " ";
}
cout << endl;

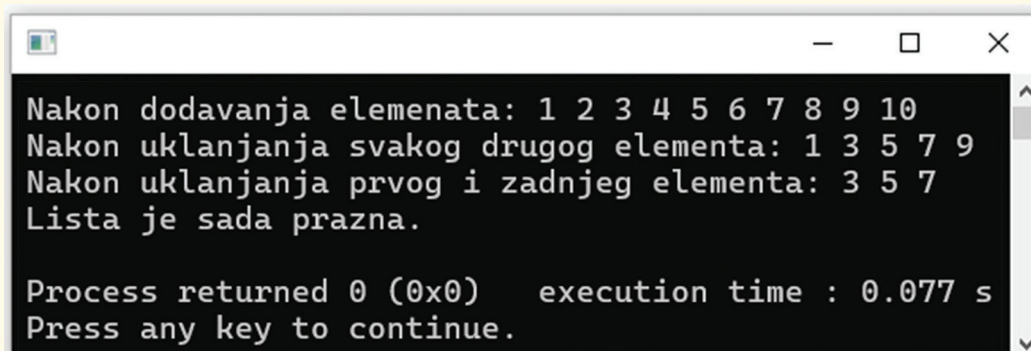
// Brisanje svih elemenata
brojevi.clear();

// Provjera je li lista prazna
if (brojevi.empty()) {
    cout << "Lista je sada prazna." << endl;
}

return 0;
}

```

Program prikazuje dodavanje i uklanjanje elemenata iz liste te ispis njezina sadržaja nakon svake promjene, što je uočljivo i na slici 10.11. Na kraju se lista prazni metodom `clear()`, a metodom `empty()` provjerava se njezino stanje, odnosno je li lista prazna.



```

Nakon dodavanja elemenata: 1 2 3 4 5 6 7 8 9 10
Nakon uklanjanja svakog drugog elementa: 1 3 5 7 9
Nakon uklanjanja prvog i zadnjeg elementa: 3 5 7
Lista je sada prazna.

Process returned 0 (0x0)   execution time : 0.077 s
Press any key to continue.

```

Slika 10.11. Ispis rezultata rada s listom nakon dodavanja i uklanjanja elemenata

10.2.5. Veličina liste

Veličina liste u programskom jeziku C++ predstavlja broj elemenata koji se trenutno nalaze u listi. Za dohvat veličine primjenjuje se metoda `size()`, dok se za provjeru je li lista prazna primjenjuje metoda `empty()` koja je objašnjena u prethodnom primjeru.

Metoda `size()` vraća broj elemenata u listi kao što je to prikazano u sljedećem kodu.

```
list<int> brojevi = {10, 20, 30};
cout << "Veličina liste: " << brojevi.size() << endl;
```

Tablica 10.5. Metode za određivanje veličine liste

metoda	opis	primjer
<code>size()</code>	vraća broj elemenata u listi	<code>l.size();</code>
<code>empty()</code>	provjerava je li lista prazna	<code>l.empty();</code>

Primjer 10.9.

Napišite program u kojem je potrebno deklarirati praznu listu cjelobrojnih vrijednosti.

U listu dodajte elemente **10, 20 i 30** primjenom odgovarajuće metode za dodavanje elemenata. Nakon toga ispišite veličinu liste s pomoću metode `size()`.

Zatim uklonite sve elemente liste primjenom metode `clear()` te provjerite je li lista prazna s pomoću metode `empty()`. Ispišite odgovarajuću poruku o stanju liste.

ulaz	izlaz
	Veličina liste: 3 Lista je prazna.

Rješenje

```
#include <iostream>
#include <list>
using namespace std;

int main() {
    list<int> brojevi;

    brojevi.push_back(10);
    brojevi.push_back(20);
    brojevi.push_back(30);

    cout << "Veličina liste: " << brojevi.size() << endl;

    brojevi.clear();
}
```

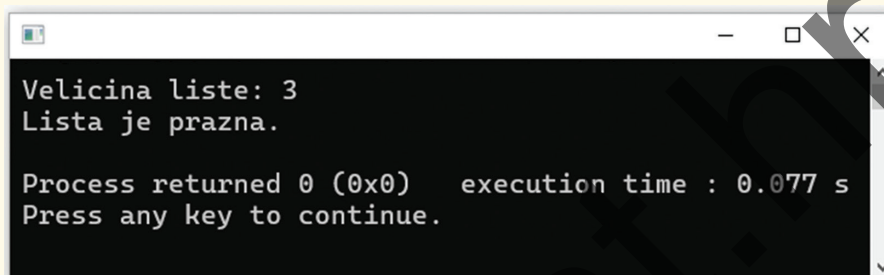
```

if (brojevi.empty()) {
    cout << "Lista je prazna." << endl;
} else {
    cout << "Lista nije prazna." << endl;
}

return 0;
}

```

U programu se najprije u listu dodaju elementi nakon čega se metodom `size()` određuje broj elemenata. Zatim se lista prazni metodom `clear()`, a metodom `empty()` provjerava se je li lista prazna. Slika 10.12 prikazuje ispis na zaslonu računala.



Slika 10.12. Rezultat programa nakon dodavanja i brisanja elemenata liste

Proširite program

Nakon brisanja svih elemenata ponovno dodajte nekoliko novih elemenata u listu i ispišite njihove vrijednosti te veličinu liste.



Za razliku od vektora lista nema pojam kapaciteta. To znači da lista ne rezervira unaprijed memorijski prostor za buduće elemente, već se memorija dodjeljuje dinamički prilikom svakog umetanja elemenata.

10.3. Primjena lista

Nakon što su prikazane osnovne metode za rad sa spremnikom `std::list`, može se prikazati njihov zajednički rad unutar funkcije `main`. U primjeru 10.10. koji slijedi primjenjuju se metode navedene u tablici 10.6.

Tablica 10.6. Najčešće primjenjivane metode spremnika `std::list`

metoda	opis	primjer
<code>push_back(x)</code>	dodaje element na kraj liste	<code>l.push_back(10);</code>
<code>push_front(x)</code>	dodaje element na početak liste	<code>l.push_front(5);</code>
<code>pop_back()</code>	uklanja zadnji element liste	<code>l.pop_back();</code>
<code>pop_front()</code>	uklanja prvi element liste	<code>l.pop_front();</code>
<code>insert(it, x)</code>	umeće element na određenu poziciju	<code>l.insert(it, 7);</code>
<code>erase(it)</code>	uklanja element na određenoj poziciji	<code>l.erase(it);</code>
<code>remove(x)</code>	uklanja sve elemente iz liste čija je vrijednost jednaka x	<code>l.remove(10);</code>
<code>clear()</code>	briše sve elemente iz liste	<code>l.clear();</code>

metoda	opis	primjer
size()	vraća broj elemenata u listi	l.size();
empty()	provjerava je li lista prazna	l.empty();
front()	vraća prvi element liste	l.front();
back()	vraća zadnji element liste	l.back();
begin()	vraća iterator na prvi element	l.begin();
end()	vraća iterator iza zadnjeg elementa	l.end();

Primjer 10.10.

Napišite program koji se koristi spremnikom `std::list`. U programu je potrebno:

- deklarirati praznu listu cjelobrojnih vrijednosti
- provjeriti je li lista prazna primjenom metode `empty()`
- dodati elemente u listu primjenom metoda `push_back()` i `push_front()` tako da lista sadrži vrijednosti **10, 20 i 30**
- ispisati prvi i zadnji element liste primjenom metoda `front()` i `back()`
- ispisati sve elemente liste primjenom `for-each` petlje
- umetnuti novi element **15 na drugo mjesto u listi (iza prvog elementa)** primjenom metode `insert()`
- ukloniti element **15 s drugog mjesta u listi** primjenom metode `erase()`
- ispisati broj elemenata liste primjenom metode `size()`
- ukloniti prvi i zadnji element liste primjenom metoda `pop_front()` i `pop_back()` te ispisati preostale elemente
- obrisati sve elemente liste primjenom metode `clear()`
- ponovno provjeriti je li lista prazna primjenom metode `empty()`

Rješenje

```
#include <iostream>
#include <list>
using namespace std;

int main() {

    // a) Deklaracija prazne liste
    list<int> brojevi;

    // b) Provjera je li lista prazna
    if (brojevi.empty()) {
        cout << "Lista je prazna." << endl;
    }

    // c) Dodavanje elemenata
    brojevi.push_back(20);
    brojevi.push_back(30);
    brojevi.push_front(10);
```

```

// d) Ispis prvog i zadnjeg elementa
cout << "Prvi element: " << brojevi.front() << endl;
cout << "Zadnji element: " << brojevi.back() << endl;

// e) Ispis svih elemenata
cout << "Elementi liste: ";
for (int x : brojevi) {
    cout << x << " ";
}
cout << endl;

// f) Umetanje elementa 15 na drugo mjesto
auto it = brojevi.begin();
++it;
brojevi.insert(it, 15);

cout << "Nakon umetanja elementa 15: ";
for (int x : brojevi) {
    cout << x << " ";
}
cout << endl;

// g) Uklanjanje elementa 15
it = brojevi.begin();
++it;
brojevi.erase(it);

cout << "Nakon uklanjanja elementa 15: ";
for (int x : brojevi) {
    cout << x << " ";
}
cout << endl;

// h) Ispis broja elemenata
cout << "Broj elemenata u listi: " << brojevi.size() << endl;

// i) Uklanjanje prvog i zadnjeg elementa
brojevi.pop_front();
brojevi.pop_back();

cout << "Nakon uklanjanja prvog i zadnjeg elementa: ";
for (int x : brojevi) {
    cout << x << " ";
}
cout << endl;

// j) Brisanje svih elemenata
brojevi.clear();

// k) Ponovna provjera je li lista prazna
if (brojevi.empty()) {
    cout << "Lista je prazna nakon metode clear()." << endl;
}

return 0;
}

```


Program prikazuje osnovne operacije nad listom u programskom jeziku C++. Na početku se s pomoću metode `empty()` provjerava je li lista prazna. Nakon toga se u listu dodaju elementi primjenom metoda `push_back()` i `push_front()`, čime se oblikuje početni sadržaj liste.

Zatim se ispisuju prvi i zadnji element primjenom metoda `front()` i `back()`, kao i svi elementi liste primjenom `for-each` petlje. U nastavku se demonstrira umetanje novog elementa na određenu poziciju metodom `insert()`, te njegovo uklanjanje metodom `erase()`.

Program također prikazuje dohvat broja elemenata u listi s pomoću metode `size()`, kao i uklanjanje prvog i zadnjeg elementa primjenom metoda `pop_front()` i `pop_back()`. Na kraju se svi elementi brišu metodom `clear()`, a metodom `empty()` provjerava se je li lista ponovno prazna. Rezultat izvođenja programa se vidi na slici 10.13.

```

Lista je prazna.
Prvi element: 10
Zadnji element: 30
Elementi liste: 10 20 30
Nakon umetanja elementa 15: 10 15 20 30
Nakon uklanjanja elementa 15: 10 20 30
Broj elemenata u listi: 3
Nakon uklanjanja prvog i zadnjeg elementa: 20
Lista je prazna nakon metode clear().

Process returned 0 (0x0)   execution time : 0.082 s
Press any key to continue.
  
```

Slika 10.13. Rezultat izvođenja programa za rad sa spremnikom `std::list`

Primjer 10.11.

Korisnici u pošti čekaju na uslugu prema brojevima s redomata. U nekom trenutku na uslugu čekaju korisnici **s brojevima od 27 do 36**, poredani redom kojim su stigli.

ŠALTER:	27 28 29 30 31 32 33 34 35 36
---------	-------------------------------

Proziva se sljedeći broj pa korisnik s brojem 27 dolazi na red i odlazi na obradu. Time on više ne sudjeluje u redu čekanja.

ŠALTER: 27	28 29 30 31 32 33 34 35 36
------------	----------------------------

Dok se korisnik 27 obrađuje, u poštu dolazi još **n novih korisnika**. Oni uzimaju svoje brojeve, počevši od 37 i priključuju se na kraj reda.

ŠALTER: 27	28 29 30 31 32 33 34 35 36 37 38 39 ... n novih korisnika
------------	--

U međuvremenu dolazi trudnica koja prema pravilima ima prednost. **Ona dobiva broj 100 i ne mora čekati na samom kraju reda, već se ubacuje odmah iza korisnika koji je trenutno prvi u redu.**

ŠALTER: 27	28 100 29 30 31 32 33 34 35 36 37 38 39 ... n novih korisnika
------------	---

U programu je potrebno modelirati opisani red čekanja upotrebom spremnika `std::list`.

Program treba ispisati:

- početno stanje korisnika koji čekaju
- koji se korisnik trenutno obrađuje
- koji je sljedeći korisnik na redu
- stanje reda nakon dolaska novih korisnika
- konačno stanje reda nakon dolaska korisnice s prednošću.

ulaz	izlaz
3	Pocetno stanje: 27 28 29 30 31 32 33 34 35 36 Trenutno se obrađuje: 27 Sljedeci na redu: 28 Unesite broj novih korisnika: 3 Nakon dolaska novih: 28 29 30 31 32 33 34 35 36 37 38 39 Konacno stanje: 28 100 29 30 31 32 33 34 35 36 37 38 39

Rješenje

```
#include <iostream>
#include <list>
using namespace std;

int main() {
    list<int> korisnici;

    // Početno stanje (27-36)
    for (int i = 27; i <= 36; i++) {
        korisnici.push_back(i);
    }

    // Ispis početnog stanja
    cout << "Pocetno stanje: ";
    for (int x : korisnici) {
        cout << x << " ";
    }
    cout << endl;

    // Korisnik 27 dolazi na šalter (obrađuje se)
    int trenutni = korisnici.front();
    korisnici.pop_front();
```

```

cout << "Trenutno se obradjuje: " << trenutni << endl;
cout << "Sljedeci na redu: " << korisnici.front() << endl;

// Unos broja novih korisnika
int n;
cout << "Unesite broj novih korisnika: ";
cin >> n;

// Dodavanje novih korisnika (37 nadalje)
int broj = 37;
for (int i = 0; i < n; i++) {
    korisnici.push_back(broj++);
}

// Ispis nakon dolaska novih korisnika
cout << "Nakon dolaska novih: ";
for (int x : korisnici) {
    cout << x << " ";
}
cout << endl;

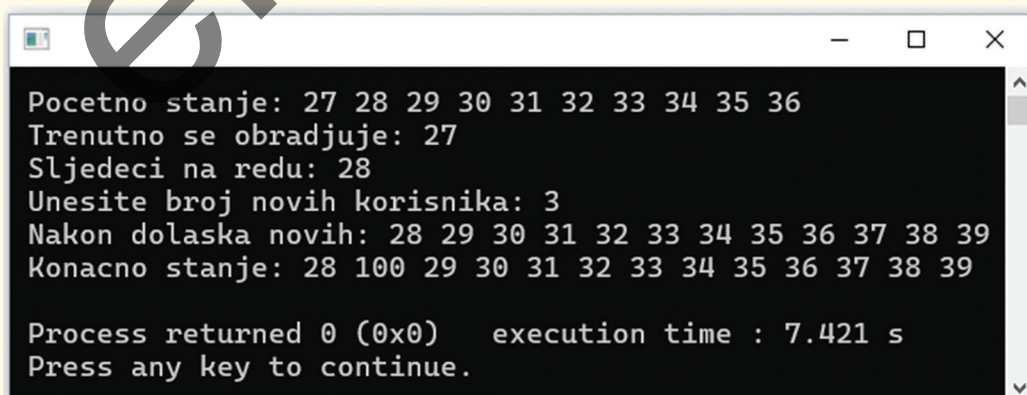
// Dolazak korisnice s prednošću (100) iza prvog
auto it = korisnici.begin();
++it;
korisnici.insert(it, 100);

// Konačno stanje
cout << "Konacno stanje: ";
for (int x : korisnici) {
    cout << x << " ";
}
cout << endl;

return 0;
}

```

Slika 10.14 prikazuje rezultat izvođenja programa koji modelira promjene u redu korisnika.



```

Pocetno stanje: 27 28 29 30 31 32 33 34 35 36
Trenutno se obradjuje: 27
Sljedeci na redu: 28
Unesite broj novih korisnika: 3
Nakon dolaska novih: 28 29 30 31 32 33 34 35 36 37 38 39
Konacno stanje: 28 100 29 30 31 32 33 34 35 36 37 38 39

Process returned 0 (0x0)   execution time : 7.421 s
Press any key to continue.

```

Slika 10.14. Ispis stanja liste prije i nakon dodavanja novih elemenata



VRIJEME ZA IZAZOV

Simulacija bilježenja temperature (dnevnik temperature)

Radna situacija

U elektrotehničkim sustavima mjerenje temperature često se provodi kontinuirano, pri čemu se izmjerene vrijednosti pohranjuju i obrađuju programskim putem. Kako bi se omogućilo bilježenje i analiza izmjerenih temperatura, potrebno je osmisliti digitalni dnevnik mjerenja temperature. Dugotrajno povišena prosječna temperatura može dovesti do smanjenja pouzdanosti rada ili kvara uređaja. U ovoj inačici programa koristit će se spremnik `std::list` za pohranu unesenih temperatura. Liste omogućuju dinamičko dodavanje elemenata, što je pogodno kada broj mjerenja nije unaprijed poznat.



Zadatak

Napišite program koji:

1. koristi se spremnikom `std::list` za spremanje dnevnih temperatura
2. omogućuje korisniku unos proizvoljnog broja temperatura
3. ispisuje sve unesene temperature
4. izračunava prosječnu temperaturu
5. pronalazi najvišu i najnižu temperaturu.

Primjer ulaznih i izlaznih podataka

ulazni podatci	izlazni podatci
Koliko temperatura želite unijeti? 5 unesite temperaturu 1: 35 unesite temperaturu 2: 38 unesite temperaturu 3: 40 unesite temperaturu 4: 42 unesite temperaturu 5: 45	Unesene temperature: 45 42 40 38 35 Prosječna temperatura je: 40 C Najviša temperatura je: 45 C Najniža temperatura je: 35 C

Elementi vrednovanja – program i pseudokod

elementi provjere	bodovi
lista je pravilno deklarirana (<code>std::list</code>)	3
ispravno je implementirano dodavanje elemenata u listu	6
omogućen je unos proizvoljnog broja temperatura	5
svi elementi liste se ispisuju	4
ispravno se primjenjuje metoda <code>size()</code> i <code>empty()</code>	4
ispravno se izračunava prosječna temperatura	4
ispravno je određena najniža i najviša temperatura	4
ukupno	30

Ocjena:

- 27 – 30 = odličan (5)
- 23 – 26 = vrlo dobar (4)
- 19 – 22 = dobar (3)
- 15 – 18 = dovoljan (2)
- 0 – 14 = nedovoljan (1)

Dodatni zadatak za one koji žele znati više

Proširite program za rad sa spremnikom `std::list` tako da omogućite dodavanje elemenata na početak i na kraj liste primjenom metoda `push_front()` i `push_back()`. Nakon unosa elemenata potrebno je ispisati sve elemente liste.

Zatim implementirajte dvije različite funkcionalnosti za određivanje broja elemenata u listi:

- primjenom metode `size()`
- ručnim prolaskom listom i brojenjem elemenata.

Usporedite dobivene rezultate.

Nakon toga sortirajte elemente liste u uzlaznom redoslijedu **bez primjene metode `sort()`**. Prilikom rješavanja zadatka osmislite vlastiti algoritam sortiranja koji će raditi nad listom.

Na kraju programa obrišite sve elemente liste metodom `clear()` i provjerite je li lista prazna primjenom metode `empty()`.

Vrednovanje kao učenje



<https://s.element.hr/d754>

11.

STOG



Skeniraj QR kôd ili otvori poveznicu u internetskom pregledniku. Prouči primjer 11.5 te ga analiziraj i proširi program prateći upute iz primjera.

<https://s.element.hr/7bfd>





Razmisli

Zamisli da pretražuješ internet i nekoliko puta klikneš gumb „Natrag” (Back) u pregledniku.

Zašto te preglednik uvijek vraća redom kojim si otvarao stranice, počevši od posljednje koju si posjetio?

Kako računalo pamti taj redoslijed tako da uvijek zna koja je stranica bila posljednja?



Nakon ove lekcije moći ćeš



- objasniti strukturu stoga kao organizaciju podataka s ograničenim pristupom (samo vrh)
- opisati i primijeniti LIFO načelo (zadnji dodani element prvi se uklanja)
- primijeniti osnovne metode rada sa stogom: push za dodavanje, pop za uklanjanje i top za dohvat vršnog elementa
- primijeniti stog u praktičnim situacijama poput poništavanja radnji („Undo”)
- pratiti redoslijed dodavanja i uklanjanja elemenata tijekom izvođenja programa
- analizirati stanje stoga nakon svake operacije.



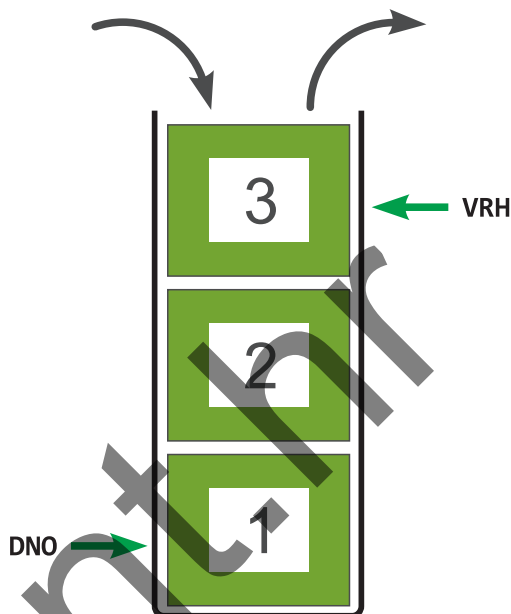
11.1. Uvod u stog (engl. *stack*, LIFO)

Stog je u programiranju često upotrebljavana struktura podataka zbog svoje jednostavnosti i učinkovitosti. Elementi su organizirani prema pravilu koje je opisano engleskim akronimom LIFO (*Last In – First Out*), što znači da zadnji dodani element prvi izlazi iz stoga.

Riječ je o posebnoj vrsti liste u kojoj se svi elementi dodaju i uklanjaju na jednom kraju, koji se naziva vrh. Ako se želi pristupiti elementu koji se nalazi dublje u stogu, prethodno je potrebno ukloniti sve elemente koji se nalaze iznad njega.

Slika 11.1 prikazuje strukturu podataka organiziranu kao stog. Temeljna mu je značajka ograničeni pristup elementima stoga.

STOG: ZADNJI ULAZI – PRVI IZLAZI



Slika 11.1. Struktura stog (engl. *stack*)

11.2. Deklaracija i inicijalizacija stoga u programskom jeziku C++

U programskom jeziku C++ stog (engl. *stack*) implementiran je s pomoću spremnika `std::stack` koji je dio standardne biblioteke. **Stog** je struktura podataka koja slijedi načelo **LIFO** (*Last In – First Out*), što znači da se elementi dodaju i uklanjaju s istog kraja, odnosno s vrha stoga.

Metode spremnika `std::stack` omogućuju dodavanje i uklanjanje elemenata te pristup elementu na vrhu stoga. Budući da stog slijedi LIFO načelo, pristup ostalim elementima nije moguć bez njihovog uklanjanja.

Za primjenu stoga potrebno je uključiti zaglavlje, kao što je prikazano u sljedećoj liniji koda:

```
#include <stack>
```

Opći oblik deklaracije stoga u C++-u prikazan je sljedećom linijom koda:

```
stack<T> naziv_stoga;
```

Kao i kod prikazanih struktura u prethodnim poglavljima, `T` predstavlja tip podatka koji se pohranjuje u stogu, a `naziv_stoga` predstavlja ime stoga.

Primjer 11.1.

Napišite deklaracije triju stogova:

- prvi stog treba pohranjivati **cjelobrojne vrijednosti**
- drugi stog treba pohranjivati **decimalne vrijednosti**
- treći stog treba pohranjivati **znakove**.

Rješenje

```
stack<int> brojevi;
stack<double> decimalni;
stack<char> znakovi;
```

Za razliku od vektora i liste stog se ne može pri deklaraciji inicijalizirati unaprijed zadanim elementima. Elementi se u stog dodaju postupno primjenom metode `push()`.

Upravo se zbog takve organizacije u stogu primjenjuju **metode za dodavanje i uklanjanje elemenata te za pristup elementu na vrhu stoga**. Elementi se dodaju metodom `push()`, uklanjaju metodom `pop()`, dok se vrijednost elementa na vrhu stoga dohvaća metodom `top()`. Neke od najčešće upotrebljivanih metoda prikazuje tablica 11.1 te će se objasniti u nastavku, dok će se ostale prikazati u ostatku poglavlja.

Tablica 11.1. Neke od najčešće upotrebljivanih metoda za rad sa stogovima

metoda	opis	primjer
<code>push(x)</code>	dodaje element na vrh stoga	<code>s.push(10);</code>
<code>pop()</code>	uklanja element s vrha stoga	<code>s.pop();</code>
<code>top()</code>	dohvaća vrijednost elementa na vrhu stoga	<code>s.top();</code>

Primjer 11.2.

Deklarirajte stog naziva `brojevi` te ga nakon toga inicijalizirajte tako da sadrži brojeve 10, 20, 30. Na kraju je potrebno ispisati koji se broj nalazi na vrhu stoga.

Rješenje

```
#include <iostream>
#include <stack>
using namespace std;

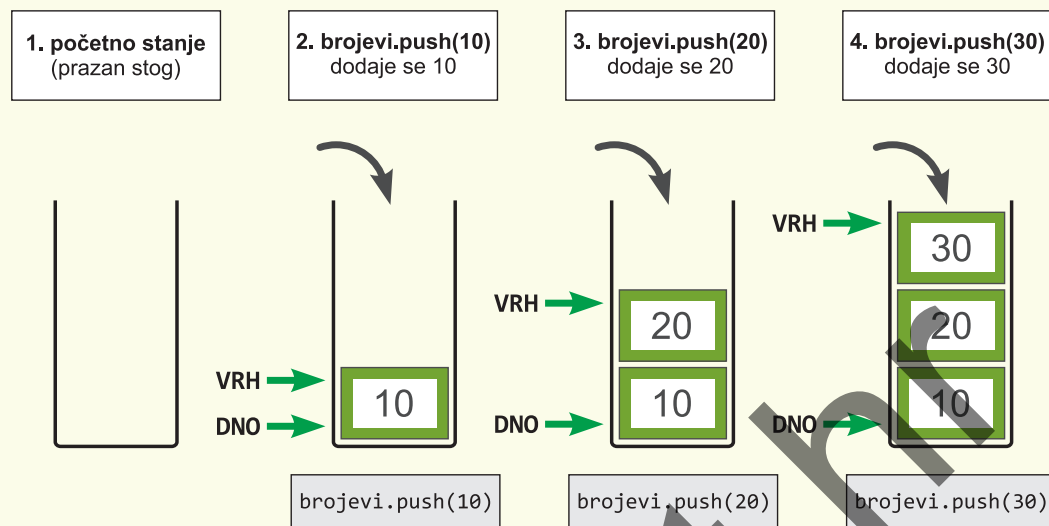
int main() {
    stack<int> brojevi;

    brojevi.push(10);
    brojevi.push(20);
    brojevi.push(30);

    cout << "Vrh stoga: " << brojevi.top() << endl;

    return 0;
}
```

U ovom primjeru elementi se dodaju jedan po jedan na vrh stoga (slika 11.2).



Slika 11.2. Postupno formiranje stoga dodavanjem elemenata 10, 20 i 30 s pomoću metode `push()`, pri čemu svaki novi element postaje vrh stoga

Posljednji dodani element nalazi se na vrhu i njemu se pristupa metodom `top()`. Na slici 11.3. prikazan je rezultat izvršavanja primjera 11.2.

```

Vrh stoga: 30

Process returned 0 (0x0)   execution time : 0.280 s
Press any key to continue.
  
```

Slika 11.3. Formiranje stoga dodavanjem elemenata s pomoću metode `push()`

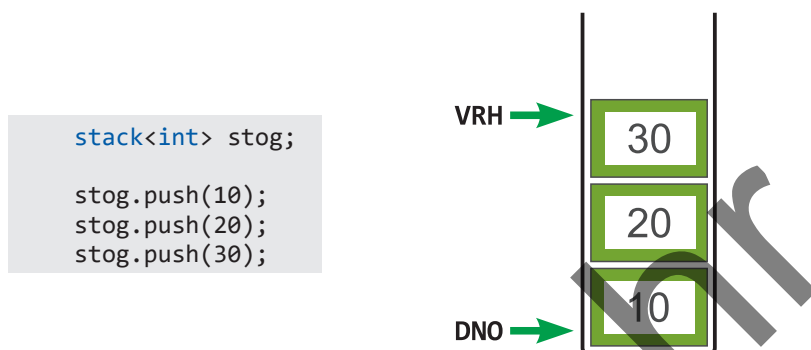
Radi jasnijeg razumijevanja rada stoga potrebno je još jednom naglasiti da, za razliku od liste i vektora, stog ne omogućuje prolazak svim elementima niti pristup elementima s pomoću indeksa. Moguć je pristup isključivo elementu koji se nalazi na vrhu stoga.

U narednim potpoglavljima obrađuju se metode za rad sa stogom. Najprije se opisuju metode prikazane u tablici 11.1, a zatim i ostale metode za određivanje veličine i trenutnog stanja stoga.

11.3. Dodavanje i uklanjanje elemenata stoga

Već prilikom inicijalizacije stoga bilo je potrebno primijeniti metodu `push()`, dok je metoda `top()` upotrijebljena kako bi dala informaciju o elementu koji se nalazi na vrhu stoga. Vrlo često su takve informacije potrebne kada radimo sa stogovima.

Kao što se moglo u ranijim primjerima uočiti, metoda `push()` primjenjuje se za dodavanje novog elementa na vrh stoga, a svaki novi element koji se dodaje metodom `push()` postaje novi element na vrhu stoga. U narednom kodu upotrebljavaju se naredbe kako bi se u stog spremile vrijednosti 10, 20 i 30. Nakon izvršavanja navedenih naredbi, element 30 nalazi se na vrhu stoga jer je dodan posljednji (slika 11.4). To je u skladu s LIFO načelom, prema kojem se posljednji dodani element prvi uklanja.



Slika 11.4. Prikaz stoga nakon dodavanja elemenata metodom `push()`

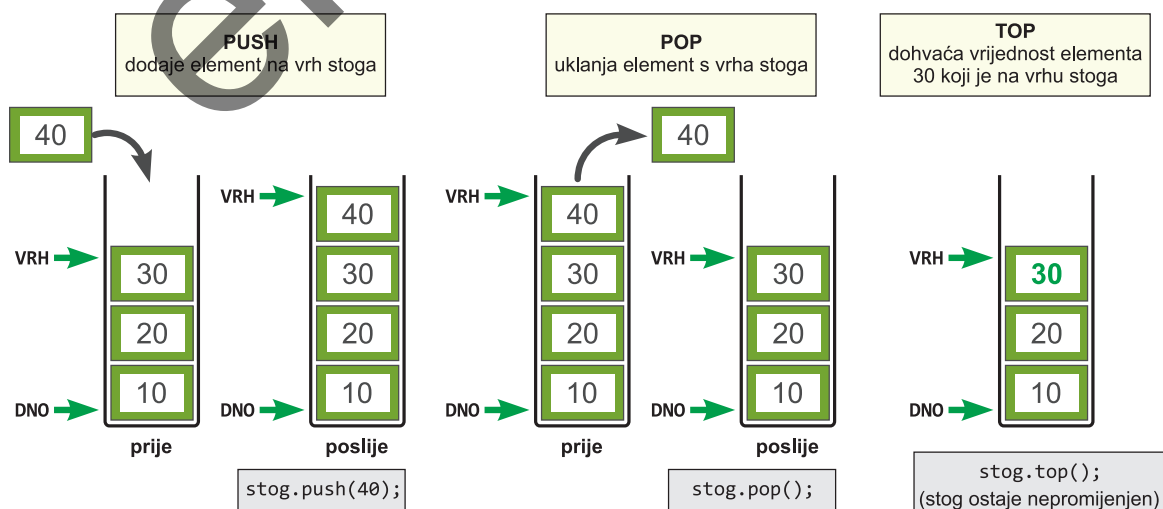
Metoda `top()` primjenjuje se za pristup elementu koji se nalazi na vrhu stoga. Važno je naglasiti da metoda `top()` samo vraća vrijednost elementa s vrha stoga, ali ga ne uklanja iz stoga. Sljedeća linija koda može se upotrijebiti kako bi se dobio element s vrha stoga.

```
cout << "Element na vrhu stoga: " << stog.top() << endl;
```

Kako bi se uklonio element s vrha stoga primjenjuje se metoda `pop()`. Nakon poziva metode `pop()` prethodni vršni element više nije dio stoga, a novi element koji je na vrhu stoga postaje onaj koji je bio dodan prije njega. Poziv metode `pop()` prikazuje sljedeća linija koda.

```
stog.pop();
```

Slika 11.5 prikazuje način rada osnovnih metoda za rad sa stogom.



Slika 11.5. Prikaz rada metoda `push()`, `pop()` i `top()` na stogu

Primjer 11.3.

Napišite program u kojem je potrebno deklarirati stog cjelobrojnih vrijednosti. U stog dodajte elemente **10**, **20** i **30** primjenom metode `push()`. Nakon dodavanja elemenata ispišite vrijednost koja se nalazi na vrhu stoga primjenom metode `top()`. Zatim uklonite element s vrha stoga koristeći se metodom `pop()` te ponovno ispišite koji je novi element na vrhu stoga.

ulazni podatci	izlazni podatci
	Element na vrhu stoga: 3 Novi element na vrhu stoga: 2

Rješenje

```
#include <iostream>
#include <stack>
using namespace std;

int main() {

    stack<int> stog;

    // Dodavanje elemenata na stog
    stog.push(10);
    stog.push(20);
    stog.push(30);

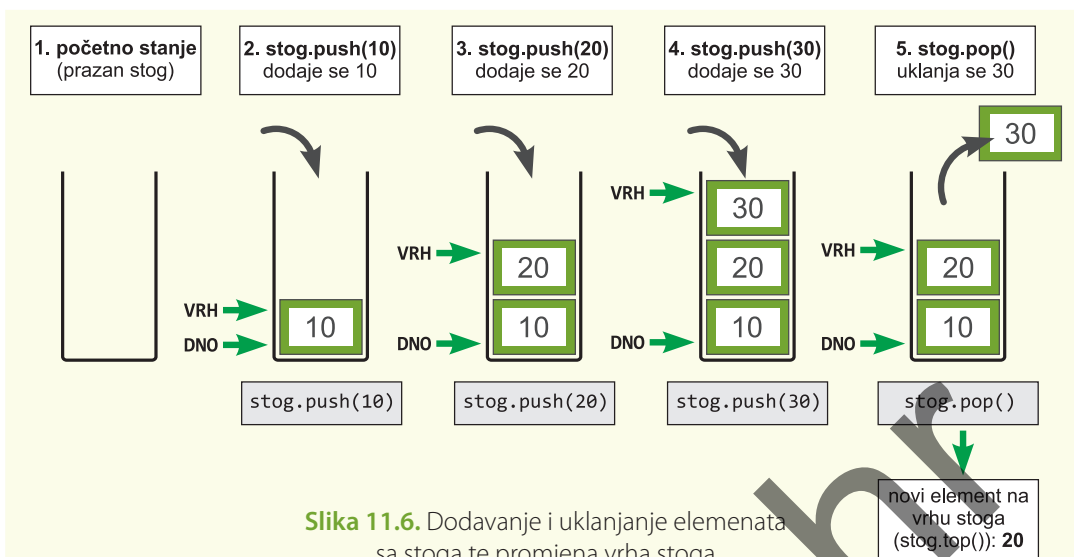
    // Ispis elementa na vrhu stoga
    cout << "Element na vrhu stoga: " << stog.top() << endl;

    // Uklanjanje elementa s vrha stoga
    stog.pop();

    // Ispis novog elementa na vrhu stoga
    cout << "Novi element na vrhu stoga: " << stog.top() << endl;

    return 0;
}
```

Slika 11.6 prikazuje postupak dodavanja i uklanjanja elemenata sa stoga primjenom metoda `push()` i `pop()`.



Slika 11.6. Dodavanje i uklanjanje elemenata sa stoga te promjena vrha stoga

Slika 11.7 prikazuje rezultat izvođenja programa u kojem se na stog dodaju elementi 10, 20 i 30 te se ispisuje element na vrhu stoga prije i nakon uklanjanja elementa 30.

```

Element na vrhu stoga: 30
Novi element na vrhu stoga: 20

Process returned 0 (0x0)   execution time : 0.086 s
Press any key to continue.
  
```

Slika 11.7. Ispis vrijednosti na vrhu stoga prije i nakon uklanjanja elementa 30 primjenom metode `pop()`

Izmijenite program

Nakon uklanjanja jednog elementa sa stoga uklonite još jedan element i ponovno ispišite koji se element nalazi na vrhu stoga.



Metode `top()` i `pop()` smiju se primjenjivati samo ako stog nije prazan. Prije njihova poziva preporučuje se provjeriti stanje stoga metodom `empty()`. U sljedećem poglavlju prikazat će se metode za provjeru stanja i veličine stoga.



<https://s.element.hr/b68e>

Napomena: *Peek* u aplikaciji odgovara metodi `top()` u C++-u i dohvaća vrijednost elementa na vrhu stoga bez uklanjanja.

Stack Visualizer

Skenirajte QR kod i isprobajte interaktivni prikaz rada stoga (*push*, *pop*, *top*).

Top of the Stack :-

-

Size of the Stack :-

0

Reset

Stack

Enter element

Push

Pop

Peek

11.4. Metode za prikaz trenutnog stanja i veličine stoga

Metode koje se primjenjuju za provjeru trenutnog stanja i veličine stoga su `empty()` i `size()`. Metoda `empty()` provjerava je li stog prazan, dok metoda `size()` vraća broj elemenata koji se trenutno nalaze u stogu. Pregled ovih metoda prikazuje tablica 11.2.

Tablica 11.2. Neke od najčešće primjenjivanih metoda za rad sa stogovima

metoda	opis	primjer
<code>empty()</code>	provjerava je li stog prazan	<code>s.empty();</code>
<code>size()</code>	vraća broj elemenata u stogu	<code>s.size();</code>

Prije primjene metoda `top()` i `pop()` preporučuje se provjeriti je li stog prazan. Time se sprječava pokušaj pristupa elementu koji ne postoji. Primjer takve provjere prikazuje se u nastavku:

```
if (!stog.empty()) {
    cout << stog.top() << endl;
    stog.pop();
}
```

U ovom primjeru uvjet `!stog.empty()` provjerava da stog nije prazan. Ako stog sadrži barem jedan element, ispisuje se element na vrhu stoga, a zatim se taj element uklanja metodom `pop()`.

Metoda `size()` može se primijeniti za ispis broja elemenata u stogu:

```
cout << "Broj elemenata u stogu: " << stog.size() << endl;
```

Primjer 11.4.

Proširite prethodni primjer 11.3. tako da primijenite metodu `size()` i `empty()`.

Rješenje

```
#include <iostream>
#include <stack>
using namespace std;

int main() {

    stack<int> stog;

    // Provjera je li stog prazan
    if (stog.empty()) {
        cout << "Stog je prazan." << endl;
    }
}
```

```

// Dodavanje elemenata na stog
stog.push(10);
stog.push(20);
stog.push(30);

// Ispis broja elemenata u stogu
cout << "Broj elemenata u stogu: " << stog.size() << endl;

// Ispis elementa na vrhu stoga
if (!stog.empty()) {
    cout << "Element na vrhu stoga: " << stog.top() << endl;
}

// Uklanjanje elementa s vrha stoga
if (!stog.empty()) {
    stog.pop();
}

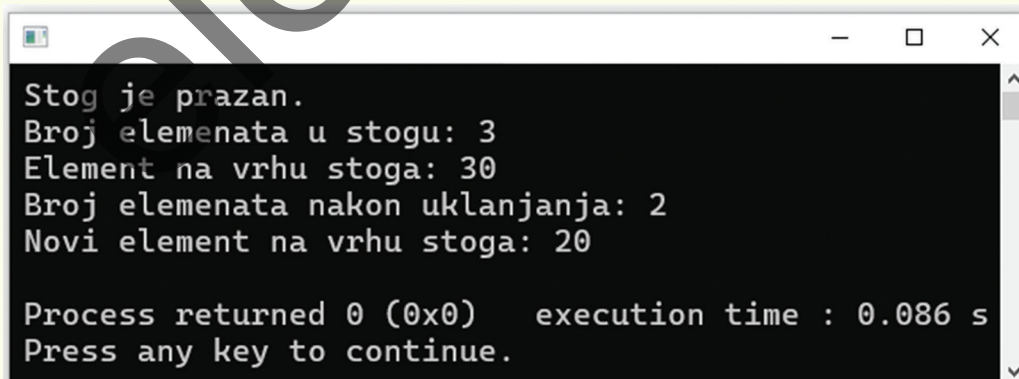
// Ispis broja elemenata nakon uklanjanja
cout << "Broj elemenata nakon uklanjanja: " << stog.size() << endl;

// Ispis novog elementa na vrhu stoga
if (!stog.empty()) {
    cout << "Novi element na vrhu stoga: " << stog.top() << endl;
}

return 0;
}

```

U programu se najprije provjerava je li stog prazan primjenom metode `empty()`. Nakon dodavanja elemenata metodom `push()`, metoda `size()` upotrebljava se za ispis broja elemenata u stogu. Prije poziva metoda `top()` i `pop()` ponovno se primjenjuje metoda `empty()` kako bi se provjerilo postoji li element kojem se može pristupiti ili koji se može ukloniti. Nakon uklanjanja elementa s vrha stoga ponovno se ispisuje veličina stoga metodom `size()`. Slika 11.8 prikazuje rezultat izvođenja programa.



```

Stog je prazan.
Broj elemenata u stogu: 3
Element na vrhu stoga: 30
Broj elemenata nakon uklanjanja: 2
Novi element na vrhu stoga: 20

Process returned 0 (0x0)   execution time : 0.086 s
Press any key to continue.

```

Slika 11.8. Prikaz rada metoda `empty()`, `size()`, `top()` i `pop()` nad stogom

11.5. Primjena stoga

U sljedećim primjerima upotrebljava se for petlja i metode za rad sa stogom iz tablice 11.1. S pomoću for petlje prolazi se elementima polja (jednodimenzionalnog ili dvodimenzionalnog), pri čemu se svaki element postupno dodaje na stog primjenom metode `push()`. Na taj način prikazuje se kako se elementi iz polja mogu prenijeti u stog te kako se stog postupno oblikuje tijekom izvođenja programa.

Primjer 11.5.

Napišite program koji se koristi spremnikom `std::stack` za prikaz rada LIFO strukture.

U programu je potrebno deklarirati prazan stog cjelobrojnih vrijednosti. Zatim deklarirati i inicijalizirati polje cjelobrojnih vrijednosti s elementima **5, 10 i 20** te dodati sve elemente iz polja u stog primjenom metode `push()`, redom kako se pojavljuju u polju.

Nakon dodavanja elemenata ispišite vrijednost koja se nalazi na vrhu stoga primjenom metode `top()`.

Zatim uklonite dva elementa sa stoga primjenom metode `pop()`. Prije svakog uklanjanja ispišite vrijednost elementa koji se uklanja.

Na kraju ponovno ispišite vrijednost elementa koji se nalazi na vrhu stoga.

ulazni podatci	izlazni podatci
	Vrh stoga: 20
	Uklanjam: 20
	Uklanjam: 10
	Vrh stoga: 5

Rješenje

```
#include <iostream>
#include <stack>
using namespace std;

int main() {

    stack<int> stog;

    int elementi[] = {5, 10, 20};

    // Dodavanje elemenata na stog pomoću for petlje
    for (int i = 0; i < 3; i++) {
        stog.push(elementi[i]);
    }

    if (!stog.empty()) {
        cout << "Vrh stoga: " << stog.top() << endl;
    }
}
```



```
// Uklanjanje dvaju elemenata sa stoga
for (int i = 0; i < 2; i++) {
    if (!stog.empty()) {
        cout << "Uklanjam: " << stog.top() << endl;
        stog.pop();
    }
}

if (!stog.empty()) {
    cout << "Vrh stoga: " << stog.top() << endl;
}

return 0;
}
```

Slika 11.9 prikazuje rezultat izvođenja programa koji demonstrira dodavanje i uklanjanje elemenata sa stoga.

```
Vrh stoga: 20
Uklanjam: 20
Uklanjam: 10
Vrh stoga: 5

Process returned 0 (0x0)   execution time : 0.073 s
Press any key to continue.
```

Slika 11.9. Ispis vrha stoga i uklanjanja elemenata

Proširite program

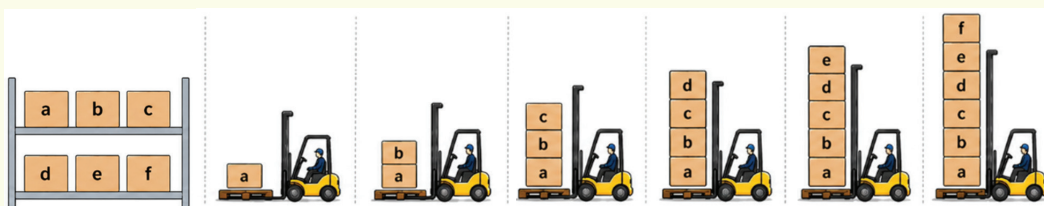
Omogućite unos broja elemenata *n*, zatim učitajte *n* cijelih brojeva u polje i potom dodajte sve elemente u stog. Nakon toga ispišite vrh stoga i uklonite jedan element.



Primjer 11.6.

Slika prikazuje postupak u kojem radnik uzima kutije s police i slaže ih na paletu koju viličar prevozi (slika 11.10). Kutije su označene slovima **a**, **b**, **c**, **d**, **e** i **f**, a radnik ih uzima redom kako su raspoređene na polici. Svaka nova kutija stavlja se na vrh već složenih kutija na paleti.

Napišite program kojim se koristi spremnik `std::stack` za prikaz rada LIFO strukture tako da simulira opisani postupak: redom dodaje kutije na paletu te nakon svakog dodavanja ispisuje koja se kutija nalazi na vrhu palete.



Slika 11.10. Slaganje kutija na paletu s police

ulazni podatci	izlazni podatci
	Vrh: a Vrh: b Vrh: c Vrh: d Vrh: e Vrh: f

Rješenje

```
#include <iostream>
#include <stack>
using namespace std;

int main() {
    stack<char> paleta;

    // Dvodimenzionalno polje (polica)
    char polica[2][3] = {
        {'a', 'b', 'c'},
        {'d', 'e', 'f'}
    };

    // Dodavanje kutija na paletu i ispis vrha
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            paleta.push(polica[i][j]);
            cout << "Vrh: " << paleta.top() << endl;
        }
    }

    return 0;
}
```

Izmijenite program

Promijenite zadatak tako da polica ima 10 redaka i 10 stupaca, a kutije su označene brojevima od 1 do 100. Prilagodite program tako da redom uzima sve kutije s police i slaže ih na paletu te nakon svakog dodavanja ispisuje broj kutije koja se nalazi na vrhu (počinje se s kutijom 1).



element.hr

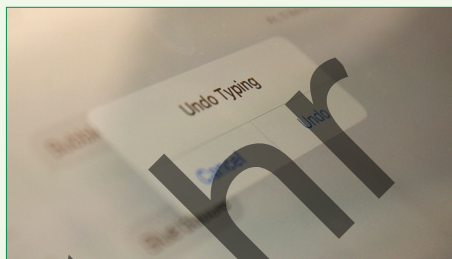


VRIJEME ZA IZAZOV

Simulacija povijesti radnja u aplikaciji (*undo* mehanizam)

Radna situacija

U računalnim aplikacijama često je potrebno omogućiti poništavanje prethodno izvršenih radnji. Primjeri takvih funkcionalnosti su naredba *undo* u uređivačima teksta, grafičkim programima ili razvojnim okruženjima. Svaka nova radnja korisnika sprema se u strukturu podataka, a prilikom poništavanja obrađuje se i uklanjanje posljednja izvršena radnja. Takav način rada temelji se na LIFO načelu (*Last In – First Out*), prema kojem se posljednje dodani element prvi uklanja. U programiranju se ovakav mehanizam najčešće implementira uporabom stoga.



Zadatak

Napišite program u programskom jeziku C++ kojim se koristi spremnik `std::stack` za pohranu cjelobrojnih vrijednosti koje predstavljaju redne brojeve izvršenih radnji. Dodavanjem nove vrijednosti na stog simulira se izvršavanje nove radnje, dok se uklanjanjem elementa sa stoga simulira poništavanje posljednje izvršene radnje. Program treba dodati nekoliko vrijednosti na stog, ispisati vrijednost koja se nalazi na vrhu stoga primjenom metode `top()`, ukloniti dva elementa sa stoga primjenom metode `pop()` te ponovno ispisati vrijednost koja se nalazi na vrhu stoga.

Primjer ulazno-izlaznih podataka

radnja u programu	ispis na zaslonu
dodavanje 5	Dodajem na vrh: 5
dodavanje 12	Dodajem na vrh: 12
dodavanje 7	Dodajem na vrh: 7
ispis vrha	Vrh stoga: 7
uklanjanje elementa	Uklanjam: 7
uklanjanje elementa	Uklanjam: 12
ispis vrha	Vrh stoga: 5

Elementi vrednovanja – program i pseudokod

elementi provjere	bodovi
stog je pravilno deklariran primjenom spremnika <code>std::stack</code>	3
ispravno je dodavanje elemenata na stog primjenom metode <code>push()</code>	5
ispravno je uklanjanje elemenata sa stoga primjenom metode <code>pop()</code>	5
omogućeno je dohvaćanje elementa s vrha stoga bez uklanjanja primjenom metode <code>top()</code>	4
ispravno se provjerava je li stog prazan primjenom metode <code>empty()</code>	4
ispravno se primjenjuje metoda <code>size()</code> za dohvat broja elemenata u stogu	3
program jasno prikazuje LIFO način rada stoga	3
program se ispravno izvršava bez sintaktičkih i logičkih pogrešaka	3
ukupno	30

Ocjena

- 27 – 30 = odličan (5)
- 23 – 26 = vrlo dobar (4)
- 19 – 22 = dobar (3)
- 15 – 18 = dovoljan (2)
- 0 – 14 = nedovoljan (1)

Dodatni zadatak za one koji žele znati više

Proširite program za rad sa stogom tako da omogućite ispis svih elemenata stoga bez promjene njegova izvornog stanja. Budući da spremnik `std::stack` ne omogućuje izravan prolazak elementima, prilikom ispisa potrebno je upotrijebiti pomoćni, odnosno privremeni stog. Elementi se mogu privremeno premještati u pomoćni stog, a nakon ispisa potrebno ih je vratiti u izvorni stog tako da njegovo stanje ostane nepromijenjeno. Pri rješavanju zadatka obratite pozornost na pravilnu primjenu metoda `top()`, `pop()`, `push()` i `empty()`. Cilj zadatka jest pokazati da se kod stoga može pristupiti samo elementu na vrhu, zbog čega je za ispis svih elemenata potrebno privremeno uklanjati elemente iz stoga.

Vrednovanje kao učenje



<https://s.element.hr/f96a>

12.

RED



DIGITALNI RUDNIK KODOVA

Skeniraj QR kôd ili otvori poveznicu u internetskom pregledniku. Prouči primjer 12.3 te ga analiziraj i proširi program prateći upute iz primjera.

<https://s.element.hr/fd03>





Razmisli

Zamisli pisač na koji više ljudi šalje dokumente za ispis. Dokumenti se ispisuju redom kojim su poslani. Može li se dokument poslan zadnji ispisati prije onih koji čekaju?

Kako bi se takav način organizacije mogao primijeniti u programiranju?



Nakon ove lekcije moći će



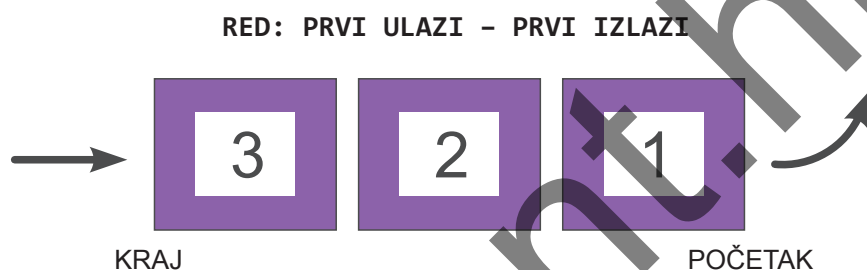
- objasniti što je red kao struktura podataka
- opisati i primijeniti FIFO načelo (prvi ulazi, prvi izlazi)
- razlikovati početak i kraj reda te njihovu ulogu
- primjenjivati osnovne metode rada s redom: push, pop, front i back
- pratiti stanje reda tijekom izvođenja programa
- primijeniti red u stvarnim situacijama poput čekanja u redu ili obrade zahtjeva.



12.1. Uvod u FIFO strukturu

U računalnim programima **red** je često korištena struktura podataka kada je važno zadržati točan redoslijed obrade podataka. Primjerice, kada više korisnika šalje zadatke jednom pisaču, oni se ispisuju redom kojim su zaprimljeni kako ne bi nastala zbrka. To je opisano engleskim akronimom **FIFO** (*First In – First Out*), što znači da je prvi element dodan u red i prvi iz njega uklonjen. **Riječ je o posebnoj vrsti liste** u kojoj se **elementi dodaju na jednom kraju, a uklanjaju na drugom kraju**.

Slika 12.1 prikazuje primjer reda. Element označen brojem 1 predstavlja početak reda, a element označen brojem 3 predstavlja njegov kraj. Budući da je element označen brojem 1 prvi dodan u red, prema FIFO pravilu (*First In – First Out*) on će biti prvi i uklonjen iz reda.



Slika 12.1. Struktura red (engl. *queue*)

U ilustracijama reda u literaturi red može biti prikazan tako da se elementi crtaju s lijeva nadesno ili zdesna nalijevo. Važno je naglasiti da smjer crtanja na slici nije presudan; ono što određuje FIFO načelo jest **mjesto ulaza u red i mesto izlaza iz reda**. Bez obzira na grafički prikaz prvi dodani element uvijek se nalazi na početku reda (na strani izlaza), a posljednji dodani element nalazi se na kraju reda (na strani ulaza).

Red omogućuje jednostavan i prirodan način upravljanja podacima u scenarijima u kojima se informacije trebaju obrađivati točno onim redom kojim su primljene. Zbog toga se FIFO načelo upotrebljava u mnogim područjima računarstva, kao što su:

- ispis dokumenata na pisac
- obrada mrežnih zahtjeva
- simulacija redova čekanja
- planiranje procesa u operacijskim sustavima.

U ovom poglavlju objasniti će se kako se **FIFO red** može implementirati s pomoću spremnika `std::queue` iz standardne C++ biblioteke te kako se njime upravlja korištenjem dostupnih metoda.

12.2. Deklaracija i inicijalizacija reda u programskom jeziku C++

U programskom jeziku C++ red (engl. *queue*) implementiran je s pomoću spremnika `std::queue`, koji je dio standardne biblioteke. **Red je struktura podataka koja slijedi načelo FIFO (*First In – First Out*), što znači da se elementi dodaju na kraj reda, a uklanjaju s njegova početka.**

Metode spremnika `std::queue` omogućuju dodavanje i uklanjanje elemenata te pristup elementima na početku i na kraju reda. Budući da red slijedi FIFO načelo, pristup ostalim elementima nije moguć bez njihova uklanjanja.

Prije rada s redom potrebno je uključiti zaglavlje, kao što je prikazano u sljedećoj liniji koda:

```
#include <queue>
```

Opći oblik deklaracije reda u C++-u prikazan je sljedećom linijom koda:

```
queue<T> naziv_reda;
```

Kao i kod struktura prikazanih u prethodnim poglavljima, `T` predstavlja tip podatka koji se pohranjuje u redu, a `naziv_reda` predstavlja ime reda.

Primjer 12.1.

Napišite deklaracije triju redova:

- prvi red pohranjuje cjelobrojne vrijednosti
- drugi red pohranjuje decimalne vrijednosti
- treći red pohranjuje znakove.

Rješenje

```
queue<int> brojevi;
queue<double> decimalni;
queue<char> znakovi;
```

Kao i u primjerima za vektore, liste i stogove, uz naziv spremnika potrebno je navesti tip podatka. Ovaj primjer uključuje tipove `int`, `double` i `char`.

Za razliku od vektora i liste red se ne može odmah inicijalizirati unaprijed zadanim elementima, nego se elementi dodaju postupno metodom `push()`.

Također, potrebno je naglasiti da iako spremnici `std::stack` i `std::queue` imaju neke zajedničke metode, poput `push()`, `pop()`, `empty()` i `size()`, razlikuju se u načinu pristupa elementima. Kod stoga se pristupa samo elementu na vrhu primjenom metode `top()`, dok se **kod reda pristupa prvom elementu metodom `front()` i zadnjem elementu metodom `back()`**. Stog primjenjuje LIFO načelo i omogućuje pristup samo vrhu, dok **red primjenjuje FIFO načelo** i omogućuje pristup prvom i zadnjem elementu.

Primjer 12.2.

Deklarirajte red naziva `brojevi`, a zatim ga inicijalizirajte tako da sadrži brojeve 10, 20 i 30. Na kraju ispišite koji se broj nalazi na početku i na kraju reda.

Rješenje

```
#include <iostream>
#include <queue>
using namespace std;

int main() {
    queue<int> brojevi;

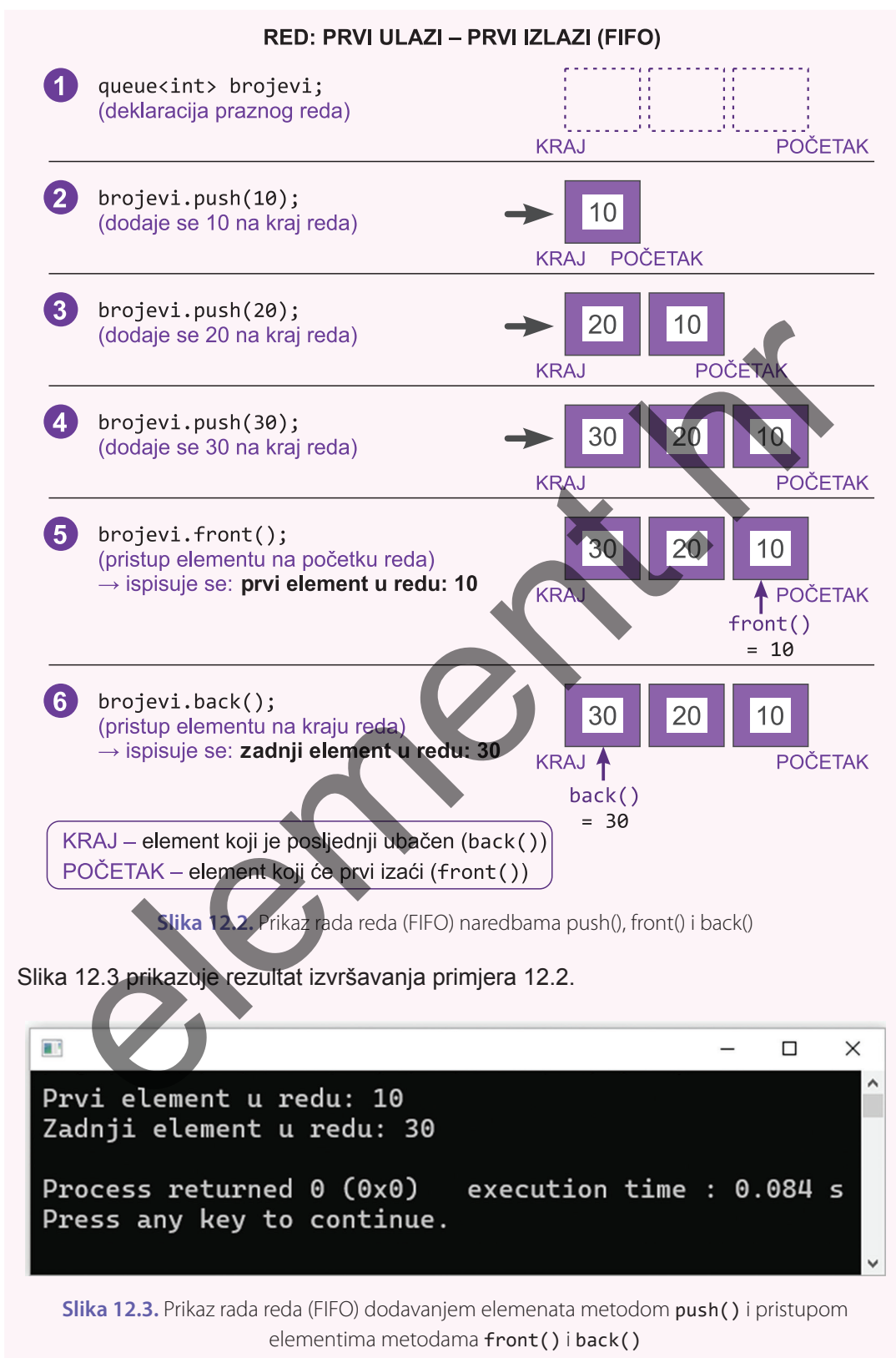
    brojevi.push(10);
    brojevi.push(20);
    brojevi.push(30);

    cout << "Prvi element u redu: " << brojevi.front() << endl;
    cout << "Zadnji element u redu: " << brojevi.back() << endl;

    return 0;
}
```

U ovom primjeru elementi se dodaju jedan po jedan na kraj reda. Prvi dodani element nalazi se na početku reda i njemu se pristupa metodom `front()`, dok se posljednji dodani element nalazi na kraju reda i njemu se pristupa metodom `back()`.

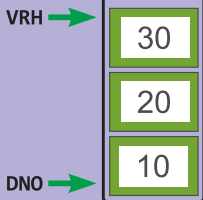
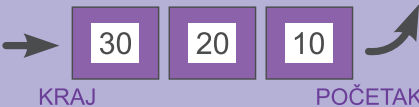
Slika 12.2 prikazuje postupak dodavanja elemenata u red te pristup elementima na početku i na kraju reda.



Radi jasnijeg razumijevanja rada reda potrebno je još jednom naglasiti da, za razliku od liste i vektora, **red ne omogućuje prolazak svim elementima niti pristup elementima s pomoću indeksa. Moguć je pristup isključivo elementu na početku i na kraju reda.**

Tablica 12.1. prikazuje usporedbu stoga i reda u programskom jeziku C++.

Tablica 12.1. Usporedba stoga i reda u programskom jeziku C++

svojstvo	stog (std::stack)	red (std::queue)
		
načelo rada	LIFO (<i>Last In – First Out</i>)	FIFO (<i>First In – First Out</i>)
dodavanje elemenata	na vrh stoga	na kraj reda
uklanjanje elemenata	s vrha stoga	s početka reda
pristup elementima	samo vrh (top())	početak (front()) i kraj (back())
smjer rada	jedna strana	dvije strane (ulaz i izlaz)
metode za pristup	top()	front(), back()
metode za dodavanje	push()	push()
metode za uklanjanje	pop()	pop()
provjera praznine	empty()	empty()
broj elemenata	size()	size()
moгуćnost prolaska elementima	nije omogućena	nije omogućena

U narednim potpoglavljima obrađuju se metode prikazane u tablici 12.1., a koje se odnose na red. Najprije se opisuju metode za rad s elementima reda, a zatim metode za određivanje njegove veličine i trenutnog stanja.

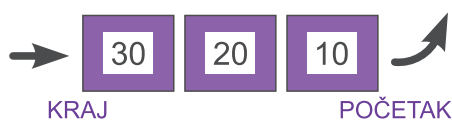
12.3. Dodavanje i uklanjanje elemenata reda

Već prilikom inicijalizacije reda bilo je potrebno primijeniti metodu push(), dok su metode front() i back() upotrijebljene kako bi dale informaciju o elementima koji se nalaze na početku i na kraju reda. Takve su informacije često potrebne kada radimo s redovima.

Kao što se moglo u ranijim primjerima uočiti, metoda push() primjenjuje se za dodavanje novog elementa na kraj reda, a svaki novi element koji se dodaje metodom push() postaje novi element na kraju reda (slika 12.4). U sljedećem kodu naredbama se u red spremaju vrijednosti 10, 20 i 30.

```
queue<int> red;

red.push(10);
red.push(20);
red.push(30);
```



Slika 12.4. Red nakon dodavanja elemenata 10, 20 i 30 metodom push()

Nakon izvršavanja navedenih naredbi, element 10 nalazi se na početku reda jer je dodan prvi, dok se element 30 nalazi na kraju reda jer je dodan posljednji. To je u skladu s FIFO načelom prema kojem se prvi dodani element prvi uklanja.

Metoda `front()` primjenjuje se za pristup elementu koji se nalazi na početku reda, odnosno elementu koji će prvi biti uklonjen. Metoda `back()` primjenjuje se za pristup elementu koji se nalazi na kraju reda, odnosno posljednjem dodanom elementu.

Sljedećim linijama koda mogu se dobiti elementi s početka i kraja reda:

```
cout << "Prvi element u redu: " << red.front() << endl;
cout << "Zadnji element u redu: " << red.back() << endl;
```

Kako bi se uklonio element s početka reda primjenjuje se metoda `pop()`. Nakon poziva metode `pop()` prvi element više nije dio reda, a novi prvi element postaje onaj koji je bio dodan nakon njega.

Poziv metode `pop()` prikazan je u sljedećoj liniji koda:

```
red.pop();
```

Primjer 12.3.

Deklarirajte red cjelobrojnih vrijednosti te u njega dodajte elemente 10, 20 i 30. Nakon toga ispišite vrijednost elementa na početku reda. Zatim uklonite jedan element s početka reda te ponovno ispišite vrijednost elementa koji se nalazi na početku reda.

Rješenje

```
#include <iostream>
#include <queue>
using namespace std;

int main() {
    queue<int> red;

    // Dodavanje elemenata u red
    red.push(10);
    red.push(20);
    red.push(30);

    // Ispis elementa na početku reda
    cout << "Prvi element u redu: " << red.front() << endl;

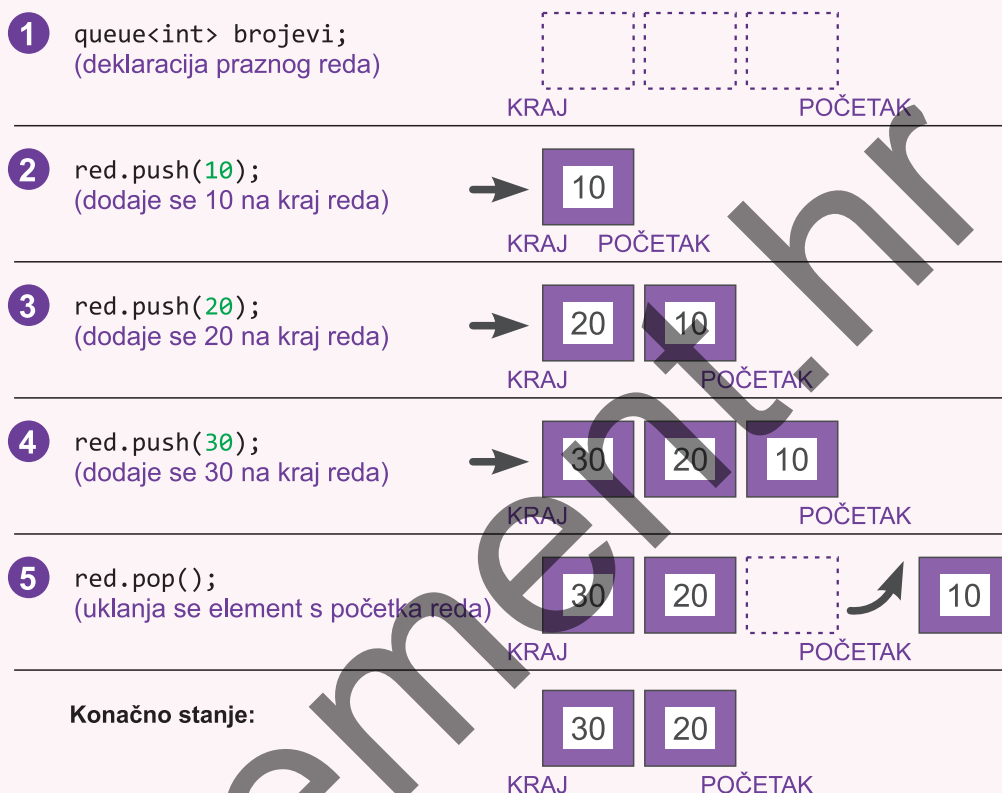
    // Uklanjanje elementa s početka reda
    red.pop();

    // Ispis novog elementa na početku reda
    cout << "Novi prvi element u redu: " << red.front() << endl;

    return 0;
}
```

ulazni podatci	izlazni podatci
	Prvi element u redu: 10 Novi prvi element u redu: 20

Slika 12.5 prikazuje postupak dodavanja i uklanjanja elemenata u redu primjenom metoda `push()` i `pop()`.



Slika 12.5. Prikaz rada reda (FIFO) – dodavanje elemenata metodom `push()` i uklanjanje elementa metodom `pop()`

```

Prvi element u redu: 10
Novi prvi element u redu: 20

Process returned 0 (0x0)   execution time : 0.384 s
Press any key to continue.

```

Slika 12.6. Ispis elemenata na početku reda prije i nakon uklanjanja elementa

Proširite program:

Nakon uklanjanja elementa s početka reda dodajte novi element na kraj reda (npr. 40) te ispišite element na početku i na kraju reda.



Metode `front()` i `pop()` smiju se primjenjivati samo ako red nije prazan. Prije njihova poziva preporučuje se provjeriti stanje reda metodom `empty()`. U sljedećem poglavlju prikazat će se metode kojima se doznaje trenutno stanje i veličina reda.

12.4. Trenutačno stanje i veličina reda

Metode koje se odabiru za provjeru trenutnog stanja i veličine reda su `empty()` i `size()`. Metoda `empty()` provjerava je li red prazan, dok metoda `size()` vraća broj elemenata koji se trenutno nalaze u redu.

Prije primjene metoda `front()`, `back()` i `pop()` preporučuje se provjeriti je li red prazan. Time se sprječava pokušaj pristupa elementu koji ne postoji. Primjer takve provjere prikazan je u nastavku:

```
if (!red.empty()) {
    cout << red.front() << endl;
    red.pop();
}
```

U ovom primjeru uvjet `!red.empty()` provjerava da red nije prazan. Ako red sadrži barem jedan element, ispisuje se element na početku reda, a zatim se taj element uklanja metodom `pop()`.

Metoda `size()` može se primijeniti za ispis broja elemenata u redu kao što je to prikazano u sljedećoj liniji koda.

```
cout << "Broj elemenata u redu: " << red.size() << endl;
```

Primjer 12.4.

Proširite prethodni primjer 12.3 tako da se koristite metodama `size()` i `empty()`.

Rješenje

```
#include <iostream>
#include <queue>
using namespace std;

int main() {
    queue<int> red;

    // Provjera je li red prazan
    if (red.empty()) {
        cout << "Red je prazan." << endl;
    }

    // Dodavanje elemenata u red
    red.push(10);
    red.push(20);
    red.push(30);

    // Ispis broja elemenata u redu
    cout << "Broj elemenata u redu: " << red.size() << endl;
```

```

// Ispis elementa na početku reda
if (!red.empty()) {
    cout << "Prvi element u redu: " << red.front() << endl;
}

// Uklanjanje elementa s početka reda
if (!red.empty()) {
    red.pop();
}

// Ispis broja elemenata nakon uklanjanja
cout << "Broj elemenata nakon uklanjanja: " << red.size() << endl;

// Ispis novog elementa na početku reda
if (!red.empty()) {
    cout << "Novi prvi element u redu: " << red.front() << endl;
}

return 0;
}

```

U programu se najprije provjerava je li red prazan primjenom metode `empty()`. Nakon dodavanja elemenata metodom `push()`, metodom `size()` ispisuje se broj elemenata u redu. Prije poziva metoda `front()` i `pop()` ponovno se primjenjuje metoda `empty()` kako bi se provjerilo postoji li element kojem se može pristupiti ili koji se može ukloniti. Nakon uklanjanja elementa s početka reda ponovno se ispisuje veličina reda metodom `size()`.

Slika 12.7 prikazuje ispis rezultata izvođenja programa.

```

Red je prazan.
Broj elemenata u redu: 3
Prvi element u redu: 10
Broj elemenata nakon uklanjanja: 2
Novi prvi element u redu: 20

Process returned 0 (0x0)   execution time : 0.080 s
Press any key to continue.

```

Slika 12.7. Prikaz rada metoda `empty()`, `front()`, `pop()` i `size()` nad redom

12.5. Primjer programa: rad s redom

U sljedećem kodu primjenjuje se for petlja i metode za rad s redovima iz tablice 12.1.

Primjer 12.5.

Napišite program koji se koristi spremnikom `std::queue` za prikaz rada FIFO strukture. U programu je potrebno deklarirati prazan red cjelobrojnih vrijednosti te ga popuniti elementima 5, 10, 15, 20 i 25 primjenom polja i for petlje.

Nakon toga ispišite vrijednost elementa na početku reda primjenom metode `front()` te vrijednost elementa na kraju reda primjenom metode `back()`.

Zatim uklonite dva elementa iz reda primjenom metode `pop()`. Prije svakog uklanjanja potrebno je ispisati vrijednost elementa koji se uklanja.

Nakon uklanjanja dvaju elemenata ponovno ispišite vrijednost elementa koji se nalazi na početku reda.

ulazni podatci	izlazni podatci
	Prvi element u redu: 5 Zadnji element u redu: 25 Uklanjam: 5 Uklanjam: 10 Prvi element u redu: 15

Rješenje

```
#include <iostream>
#include <queue>
using namespace std;

int main() {
    queue<int> red;

    int elementi[] = {5, 10, 15, 20, 25};

    // Popunjavanje reda pomoću for petlje
    for (int i = 0; i < 5; i++) {
        red.push(elementi[i]);
    }

    // Ispis prvog i zadnjeg elementa
    if (!red.empty()) {
        cout << "Prvi element u redu: " << red.front() << endl;
        cout << "Zadnji element u redu: " << red.back() << endl;
    }
}
```

```

// Uklanjanje dvaju elemenata iz reda
for (int i = 0; i < 2; i++) {
    if (!red.empty()) {
        cout << "Uklanjam: " << red.front() << endl;
        red.pop();
    }
}

// Ispis novog prvog elementa
if (!red.empty()) {
    cout << "Prvi element u redu: " << red.front() << endl;
}

return 0;
}

```

Slika 12.8 prikazuje ispis rezultata izvođenja programa koji demonstrira rad s redom i uklanjanje elemenata.

```

Prvi element u redu: 5
Zadnji element u redu: 25
Uklanjam: 5
Uklanjam: 10
Prvi element u redu: 15

Process returned 0 (0x0)   execution time : 0.080 s
Press any key to continue.

```

Slika 12.8. Ispis rada reda pri dodavanju i uklanjanju elemenata

Izmijenite program

Nakon uklanjanja dvaju elemenata iz reda dodajte dva nova elementa (npr. 30 i 35) na kraj reda te ispišite element na početku i na kraju reda.



Primjer 12.6.

Sustav za ispis dokumenata organizira zadatke u red (*queue*). Svaki dokument koji korisnik pošalje na ispis dodaje se na kraj reda.

Napišite program koji se koristi spremnikom `std::queue` za simulaciju rada pisača. Program treba omogućiti unos naredbi sve dok korisnik ne unese znak X.

Korisnik unosi naredbe s pomoću znakova s tipkovnice:

- **D n** – unosom znaka D i broja n dodaje se dokument s brojem n u red za ispis
- **I** – unosom znaka I ispisuje se dokument s početka reda i uklanja se iz reda
- **S** – unosom znaka S ispisuje se broj dokumenta koji je sljedeći za ispis
- **X** – unosom znaka X prekida se izvođenje programa

Prije ispisa ili uklanjanja dokumenta potrebno je provjeriti je li red prazan.

ulazni podatci	izlazni podatci
D 101	Sljedeci dokument: 101
D 102	Ispisujem dokument: 101
D 103	Ispisujem dokument: 102
S	Sljedeci dokument: 103
I	Ispisujem dokument: 103
I	Ispisujem dokument: 104
S	Red je prazan.
I	Red je prazan.
D 104	
I	
S	
I	
X	

Rješenje

```
#include <iostream>
#include <queue>
using namespace std;

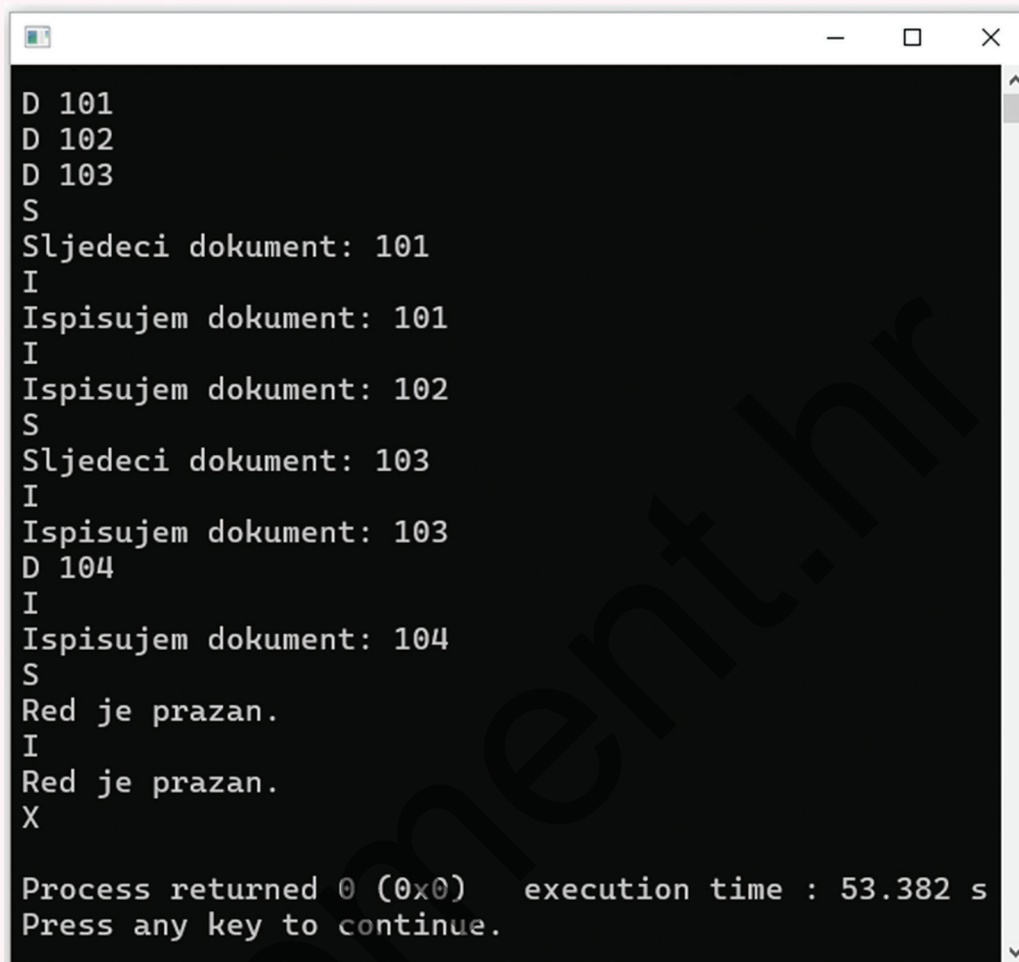
int main() {
    queue<int> printer;
    char naredba;

    while (true) {
        cin >> naredba;

        if (naredba == 'X') {
            break;
        }
        else if (naredba == 'D') {
            int broj;
            cin >> broj;
            printer.push(broj);
        }
        else if (naredba == 'I') {
            if (!printer.empty()) {
                cout << "Ispisujem dokument: " << printer.front() << endl;
                printer.pop();
            }
            else {
                cout << "Red je prazan." << endl;
            }
        }
        else if (naredba == 'S') {
            if (!printer.empty()) {
                cout << "Sljedeci dokument: " << printer.front() << endl;
            }
            else {
                cout << "Red je prazan." << endl;
            }
        }
    }

    return 0;
}
```

Očekivani ispis na zaslonu računala prikazuje slika 12.9.



```
D 101
D 102
D 103
S
Sljedeci dokument: 101
I
Ispisujem dokument: 101
I
Ispisujem dokument: 102
S
Sljedeci dokument: 103
I
Ispisujem dokument: 103
D 104
I
Ispisujem dokument: 104
S
Red je prazan.
I
Red je prazan.
X

Process returned 0 (0x0)   execution time : 53.382 s
Press any key to continue.
```

Slika 12.9. Simulacija rada pisača i strukture red

Izmijenite program

Dodajte novu naredbu K koja ispisuje koliko se trenutno dokumenata nalazi u redu.



element.hr



VRIJEME ZA IZAZOV

Redomat

U sustavima za pružanje usluga, primjerice u banci, korisnici na uslugu čekaju u redu. Svakom korisniku sustav dodjeljuje jedinstveni redni broj, a usluga se pruža prema redoslijedu dolaska – najprije se poslužuje korisnik koji je prvi došao. Takav način obrade temelji se na FIFO načelu (*First In – First Out*) i u programskom jeziku C++ može se ostvariti primjenom spremnika `std::queue`.



Zadatak

Napišite program u programskom jeziku C++ koji se koristi spremnikom `std::queue` za spremanje rednih brojeva korisnika. Program treba obrađivati niz operacija koje korisnik unosi.

Svaka operacija unosi se u jednom retku i započinje jednim znakom:

- **D x** – dodaje korisnika s brojem x na kraj reda (nakon znaka D unosi se cijeli broj)
- **U** – uklanja prvog korisnika iz reda i ispisuje njegov broj
- **P** – ispisuje broj prvog i zadnjeg korisnika u redu.

Na početku se unosi broj operacija, a zatim se unose pojedine operacije redom kako je prikazano u primjeru ulaznih i izlaznih podataka.

Za svaku operaciju potrebno je ažurirati stanje reda te, prema potrebi, ispisati odgovarajuće podatke.

Pri rješavanju primijenite metode `push()`, `pop()`, `front()`, `back()`, `empty()` i `size()`.

Primjer ulaznih i izlaznih podataka

ulazni podatci	izlazni podatci
7	Uklonjen: 5
D 5	Prvi u redu: 12
D 12	Zadnji u redu: 7
D 7	Uklonjen: 12
U	Prvi u redu: 7
P	Zadnji u redu: 7
U	
P	

Elementi vrednovanja – program

elementi provjere	bodovi
red je pravilno deklariran upotrebom spremnika <code>std::queue</code>	4
ispravno je dodavanje elemenata u red primjenom metode <code>push()</code>	5
ispravno je uklanjanje elemenata iz reda primjenom metode <code>pop()</code>	5
omogućena je obrada niza korisničkih naredbi	5
ispravno se dohvaća prvi i zadnji element reda bez uklanjanja primjenom metode <code>front()</code> i <code>back()</code>	3
ispravno se provjerava je li red prazan primjenom metode <code>empty()</code>	3
ispravno se primjenjuje metoda <code>size()</code> za dohvat broja elemenata u redu	2
program ispravno ispisuje rezultate operacija	3
ukupno	30

Ocjena

- 27 – 30 = odličan (5)
- 23 – 26 = vrlo dobar (4)
- 19 – 22 = dobar (3)
- 15 – 18 = dovoljan (2)
- 0 – 14 = nedovoljan (1)

Dodatni zadatak za one koji žele znati više

Proširite program tako da se nakon svake operacije ispiše trenutno stanje reda. Budući da spremnik `std::queue` ne omogućuje izravan prolazak svim elementima, za ispis stanja reda potrebno je upotrijebiti pomoćni red. Elementi se mogu privremeno premještati u pomoćni red, ispisati, a zatim vratiti u izvorni red kako bi njegovo stanje ostalo nepromijenjeno.

Nakon obrade svih korisničkih naredbi program treba ispisati koliko se elemenata nalazi u redu primjenom metode `size()`. Pri rješavanju zadatka obratite pozornost na pravilnu primjenu metoda `push()`, `pop()`, `front()`, `back()`, `empty()` i `size()`.

Vrednovanje kao učenje



<https://s.element.hr/06b0>

SAŽETAK

6. Uvod u programski jezik C++

```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello world!" << endl;
    return 0;
}
```

Biblioteke:

- `iostream` – za unos i ispis podataka
- `cmath` – za matematičke funkcije
- `iomanip` – za oblikovanje ispisa

Opći zapis za ispis podataka (cout):

```
cout << podatak1 << podatak2 << ... << podatakN;
```

- `setw(n)` – određuje širinu ispisa
- `fixed` – postavlja decimalni zapis broja
- `setprecision(n)` – određuje broj decimalnih mjesta.

Opći zapis za unos podataka (cin):

```
cin >> varijabla1 >> varijabla2 >> ... >> varijablaN;
```

Ako želimo unijeti tekst koji sadrži razmake, primjenjujemo naredbu:

```
getline(cin, varijabla);
```

element	sličnosti jezika C i C++	
struktura programa	<pre>#include <biblioteka> int main() { // deklaracije varijabli // naredbe (unos, obrada, ispis) return 0; }</pre>	
komentari	<pre>// linijski /*u više redaka*/</pre>	
deklaracija i inicijalizacija varijable	<pre>int a = 5; float x, y; char p = 'a';</pre>	
tipovi podataka	• cjelobrojni tipovi: <code>char</code>, <code>int</code>, <code>_Bool</code> (<code>bool</code>)	• brojevi s pomičnim zarezom: <code>float</code>, <code>double</code>
operatori	• pridruživanja (<code>=</code>) • aritmetički (<code>+</code>, <code>-</code>, <code>*</code>, <code>/</code>, <code>%</code>) • relacijski (<code>==</code>, <code>!=</code>, <code>></code>, <code><</code>, <code>>=</code>, <code><=</code>)	• logički (<code>&&</code>, <code> </code>, <code>!</code>) • inkrement/dekrement (<code>++</code>, <code>--</code>) • <code>sizeof</code> • bitovni (<code>&</code>, <code> </code>, <code>^</code>, <code>>></code>, <code><<</code>)
grananje	• <code>if-else</code>	• <code>switch-case</code>
petlje	• <code>for</code> • <code>while</code>	• <code>do-while</code>

7. Polja (nizovi)

Polje (niz, engl. *array*) je složeni tip podataka koji omogućuje spremanje više vrijednosti istog tipa unutar jedne varijable. Razlikujemo jednodimenzionalna, dvodimenzionalna i višedimenzionalna polja.

Elementi polja:

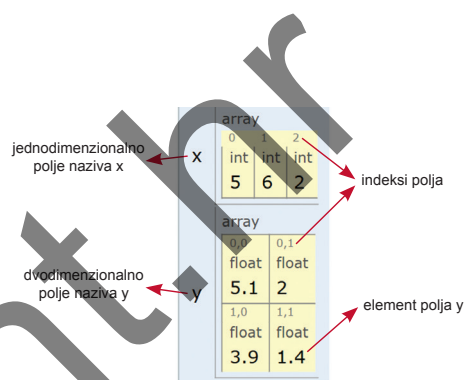
- imaju isti tip podatka
- poredani su jedan za drugim u memoriji
- razlikuju se prema indeksu (indeks prvog elementa uvijek je 0).

Ispis elemenata jednodimenzionalnog polja s pomoću petlje

```
for (int i = 0; i < n; i++) {
    cout << naziv_polja[i] << endl;
}
```

Učitavanje elemenata u jednodimenzionalno polje s pomoću petlje

```
for (int i = 0; i < n; i++) {
    cin >> naziv_polja[i];
}
```



Ispis elemenata dvodimenzionalnog polja s pomoću petlje

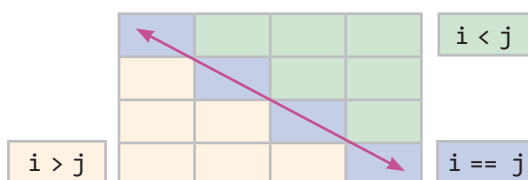
```
for (int i = 0; i < redovi; i++) {
    for (int j = 0; j < stupci; j++) {
        cout << naziv_polja[i][j] << " ";
    }
    cout << endl;
}
```

Učitavanje elemenata u dvodimenzionalno polje s pomoću petlje

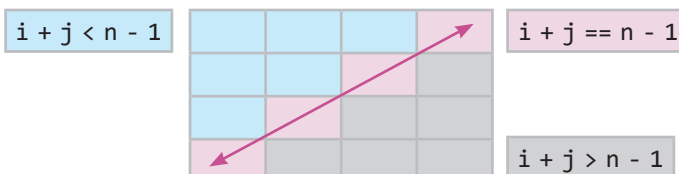
```
for (int i = 0; i < redovi; i++) {
    for (int j = 0; j < stupci; j++) {
        cin >> naziv_polja[i][j];
    }
}
```

Kod dvodimenzionalnih polja s jednakim brojem redaka i stupaca možemo definirati glavnu i sporednu dijagonalu.

Glavna dijagonala tablice od n redaka i n stupaca (i – indeks retka, j – indeks stupca)



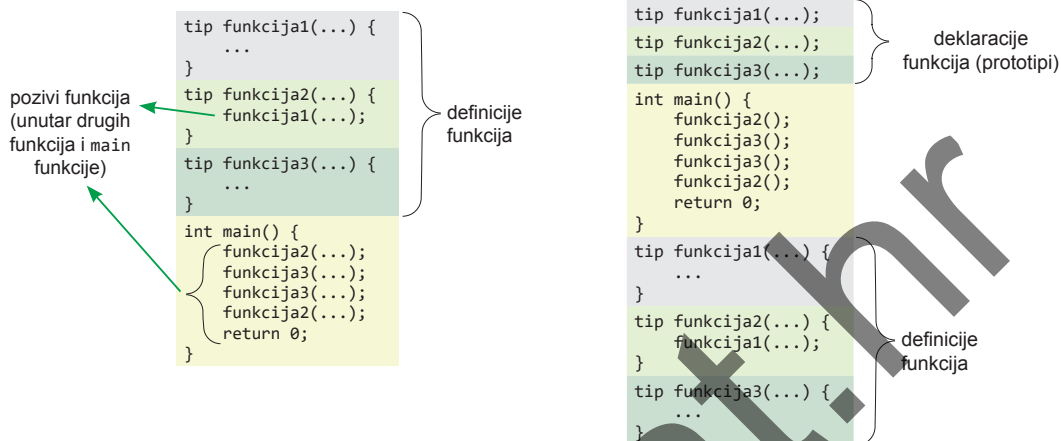
Sporedna dijagonala tablice od n redaka i n stupaca (i – indeks retka, j – indeks stupca)



8. Funkcije, klase i metode u C++-u

Korisnički definirane funkcije - služe za organizaciju programa u manje cjeline, omogućuju ponovnu uporabu koda te smanjuju ponavljanje istih naredbi

Gdje se piše funkcija?



Poziv funkcije

- funkcija se poziva navođenjem imena i argumenata; argumenti se povezuju s parametrima po redoslijedu

funkcija s povratnom vrijednošću	funkcija bez povratne vrijednosti (void)
<pre> tip nazivFunkcije(tip parametar1, tip parametar2, ...) { ... return vrijednost; } int main() { //1. način tip varijabla = nazivFunkcije(argument1, argument2, ...); //2. način cout << nazivFunkcije(argument1, argument2, ...); return 0; } </pre>	<pre> void nazivFunkcije(tip parametar1, tip parametar2, ...) { ... } int main() { nazivFunkcije(argument1, argument2, ...); return 0; } </pre>

- **Klasa (class)** – predložak koji definira podatke (atribute) i funkcije (metode).
- **Objekt (object)** – konkretna instanca klase sa svojim vrijednostima podataka.
- **Atribut** – podatak (varijabla) definiran unutar klase.
- **Metoda** – funkcija definirana unutar klase koja radi s atributima objekta (metoda()).
- **public** – pristupni modifikator; članovi klase dostupni su izvan klase.
- **Pristup atributima** – dohvaćanje ili mijenjanje podataka objekta.
- **Pozivanje metoda** – izvršavanje metoda nad objektom (objekt.metoda()).

9. Vektori

Vektor se upotrebljava kada nije unaprijed poznat broj potrebnih elemenata. Predstavlja **proširivu inačicu običnog polja** i **omogućuje dinamičko povećavanje ili smanjivanje svoje veličine** ovisno o potrebama programa.

Opći oblik deklaracije vektora

```
vector<T> naziv_vektora;
```

Primjer deklaracije i inicijalizacije:

```
vector<int> v = {10, 20, 30, 40};
```

	10	20	30	40
indeksi:	0	1	2	3

Najčešće metode spremnika std::vector

metoda	opis	primjer
push_back(x)	dodaje element na kraj vektora	v.push_back(10);
pop_back()	uklanja zadnji element vektora	v.pop_back();
size()	vraća broj elemenata u vektoru	v.size();
empty()	provjerava je li vektor prazan	v.empty();
clear()	briše sve elemente iz vektora	v.clear();
v[i]	pristup elementu s pomoću indeksa	v[0];
at(i)	siguran pristup elementu (provjera granica)	v.at(0);
front()	vraća prvi element vektora	v.front();
back()	vraća zadnji element vektora	v.back();

Prolazak elementima vektora:

Opći zapis (for-each petlja):

```
for (tip_varijabla : naziv_vektora) {
    // naredbe
}
```

Opći zapis (for petlja):

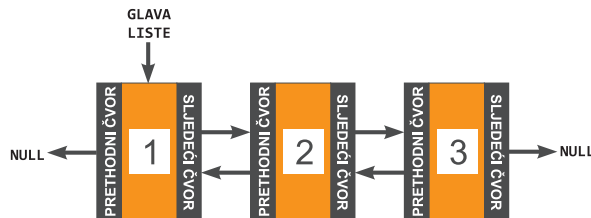
```
for (int i = 0; i < naziv_vektora.size(); i++) {
    // naredbe
}
```



10. Liste

Lista, odnosno povezana lista (engl. *linked list*) sastoji se od niza elemenata koji se nazivaju čvorovi (engl. *node*).

U programskom jeziku C++ povezana lista implementira se putem spremnika `std::list` koji **predstavlja dvostruko povezanu listu**. Lista omogućuje učinkovito **dodavanje i uklanjanje elemenata** na bilo kojem mjestu u strukturi.



Opći oblik deklaracije liste:

```
list<T> naziv;
```

Primjeri deklaracije i inicijalizacije

br.	opći zapis	primjer
1.	<code>list<T> naziv;</code>	<code>list<int> brojevi;</code>
2.	<code>list<T> naziv = {vrijednosti};</code>	<code>list<int> brojevi = {10, 20, 30};</code>
3.	<code>list<T> naziv(n, vrijednost);</code>	<code>list<int> brojevi(5, 0);</code>
4.	<code>list<T> naziv(druga_lista);</code>	<code>list<int> l2(l1);</code>
5.	<code>list<T> naziv(pocetak, kraj);</code>	<code>list<int> l2(l1.begin(), l1.end());</code>

Najčešće metode spremnika `std::list`

metoda	opis	metoda	opis
<code>push_back(x)</code>	dodaje element na kraj liste	<code>clear()</code>	briše sve elemente iz liste
<code>push_front(x)</code>	dodaje element na početak liste	<code>size()</code>	vraća broj elemenata u listi
<code>pop_back()</code>	uklanja zadnji element liste	<code>empty()</code>	provjerava je li lista prazna
<code>pop_front()</code>	uklanja prvi element liste	<code>front()</code>	vraća prvi element liste
<code>insert(it, x)</code>	umeće element na određenu poziciju	<code>back()</code>	vraća zadnji element liste
<code>erase(it)</code>	uklanja element na određenoj poziciji	<code>begin()</code>	vraća iterator na prvi element
<code>remove(x)</code>	uklanja sve elemente vrijednosti x iz liste	<code>end()</code>	vraća iterator iza zadnjeg elementa

Dodavanje na kraj liste (`push_back`)

```
for (int i = pocetak; i <= kraj; i++) {
    lista.push_back(vrijednost);
}
```

Dodavanje na početak liste (`push_front`)

```
for (int i = pocetak; i <= kraj; i++) {
    lista.push_front(vrijednost);
}
```

11. Stog

Stog je struktura podataka u kojoj se elementi dodaju i uklanjaju samo s jednog mjesta, s vrha.

Stog slijedi pravilo **LIFO** što znači „zadnji ulazi, prvi izlazi”.

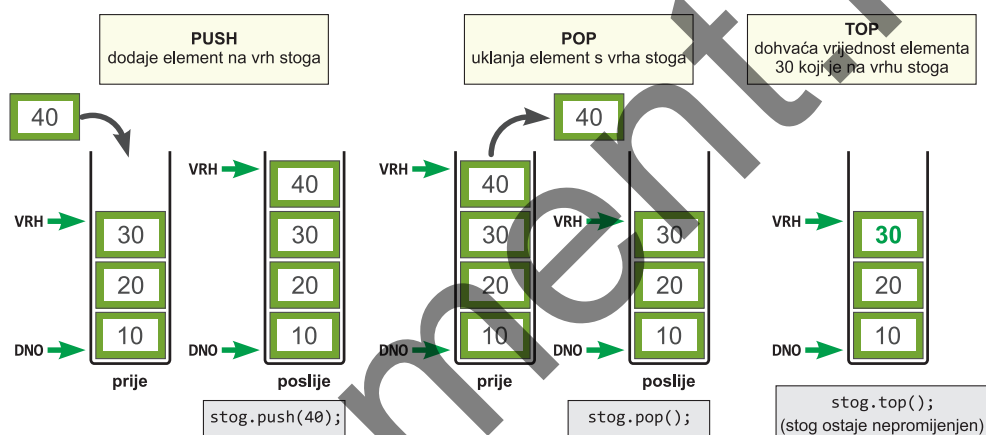
Vrh stoga je mjesto na koje se dodaju novi elementi i s kojeg se uklanjaju postojeći.

Dno stoga predstavlja prvi dodani element.

U stogu je moguć pristup samo elementu na vrhu, dok se ostali elementi ne mogu izravno dohvatiti.

U programskom jeziku C++ stog se implementira s pomoću spremnika `std::stack`.

Stog se deklarira zapisom oblika `stack<T> naziv;`.



Neke od dodatnih metoda za rad sa stogom

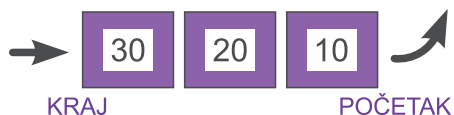
metoda	opis	primjer
<code>empty()</code>	provjerava je li stog prazan	<code>s.empty();</code>
<code>size()</code>	vraća broj elemenata u stogu	<code>s.size();</code>



12. Red

Red je struktura podataka u kojoj se elementi dodaju na kraj, a uklanjaju s početka. Red slijedi pravilo **FIFO** što znači „prvi ulazi, prvi izlazi”.

Početak reda je mjesto s kojeg se elementi uklanjaju, a **kraj reda** je mjesto na koje se dodaju novi elementi.



U programskom jeziku C++ red se implementira s pomoću spremnika `std::queue`.

Red se deklarira zapisom oblika `queue<T> naziv;` pri čemu je `T` tip podatka koji se pohranjuje u redu.

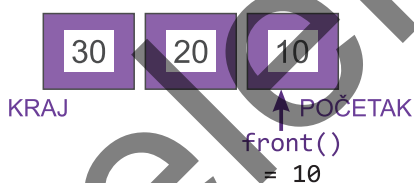
Metoda `push()` dodaje element na kraj reda.



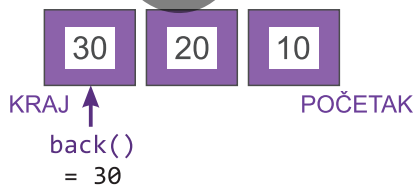
Metoda `pop()` uklanja element s početka reda.



Metoda `front()` dohvaća prvi element reda.



Metoda `back()` dohvaća zadnji element reda.



Metoda `empty()` provjerava je li red prazan.

Metoda `size()` vraća broj elemenata u redu.



PITANJA ZA PONAVLJANJE

KAKO UPOTRIJEBITI SEMAFOR

- ako možete riješiti 1. zadatak → ● osnovna razina
- ako možete riješiti 1. i 2. zadatak → ● srednja razina
- ako možete riješiti sve zadatke → ● napredna razina



6. Uvod u programski jezik C++

● 1. ZADATAK / OSNOVNA RAZINA

Objasnite čemu služe naredbe cin i cout u C++ programu. Navedite opće oblike naredbi if, switch, for, while i do-while te ukratko objasnite kada se koja primjenjuje.

● 2. ZADATAK / SREDNJA RAZINA

Zadan je program:

```
#include <iostream>
using namespace std;

int main() {
    for (int i = 1; i <= 3; i++) {
        for (int j = 1; j <= 3; j++) {
            if ((i + j) % 2 == 0) {
                cout << i + j << " ";
            } else {
                cout << i * j << " ";
            }
        }
    }
    return 0;
}
```

- Odredite što će program ispisati.
- Koliko se puta izvršava if uvjet?
- Objasnite što se ispisuje kada je uvjet zadovoljen, a što kada nije.

● 3. ZADATAK / NAPREDNA RAZINA

Napišite program koji učitava cijene ulaznica jednu po jednu. Unos završava kada se unese vrijednost 0. Ako je unesena negativna cijena, program treba ispisati poruku Pogresan unos i preskočiti tu vrijednost. Na kraju program ispisuje ukupan iznos svih ispravnih cijena.

ulazni podatci	izlazni podatci
6.5, 4.0, -2, 3.5, 0	Pogresan unos Ukupan iznos je 14.00 eura.

7. Polja (nizovi)

1. ZADATAK / OSNOVNA RAZINA

Objasnite što je polje (niz) u C++-u. Navedite kako se elementi polja razlikuju. Napišite primjer deklaracije i inicijalizacije jednodimenzionalnog i dvodimenzionalnog polja te objasnite kako se elementi polja učitavaju i ispisuju.

2. ZADATAK / SREDNJA RAZINA

Zadan je program:

```
#include <iostream>
using namespace std;

int main() {
    int x[3] = {2, 4, 6};

    for (int i = 0; i < 3; i++) {
        x[i] = x[i] + 1;
        cout << x[i] << " ";
    }

    return 0;
}
```

- Odredite što će program ispisati.
- Izmijenite program tako da se svaki element polja ispisuje u novi red.
- Objasnite ulogu indeksa u ovom programu.

3. ZADATAK / NAPREDNA RAZINA

Napišite program koji učitava temperature tijekom 3 dana (za svaki dan po 4 mjerenja) u dvodimenzionalno polje. Program treba ispisati sve unesene temperature te izračunati i ispisati prosječnu temperaturu.

ulazni podatci	izlazni podatci
10 12 11 13	Unesene temperature: 10 12 11 13 14 15 13 12 9 11 10 12 Prosjek: 12.00
14 15 13 12	
9 11 10 12	

8. Funkcije, klase i metode u C++-u

1. ZADATAK / OSNOVNA RAZINA

Objasnite što je funkcija i čemu služi. Navedite razliku između parametra i argumenta. Napišite po jedan kratak primjer za funkciju s povratnom vrijednošću i funkciju bez povratne vrijednosti.

Zatim definirajte pojmove uz primjere: **klasa**, **objekt**, **atribut** i **metoda**.

2. ZADATAK / SREDNJA RAZINA

Zadan je program:

```
#include <iostream>
using namespace std;

void povecaj(int x) {
    x = x + 5;
}

void promijeni(int &x) {
    x = x + 5;
}

int main() {
    int a = 10;

    povecaj(a);
    cout << a << endl;

    promijeni(a);
    cout << a << endl;

    return 0;
}
```

- Odredite što će program ispisati.
- Objasnite razliku između funkcija `povecaj` i `promijeni`.

3. ZADATAK / NAPREDNA RAZINA

Napišite program u kojem definiraš klasu `Temperatura` jednim atributom stupnjevi.

U klasi definirajte:

- metodu koja povećava temperaturu za 5 stupnjeva
- metodu koja ispisuje trenutnu temperaturu.

U glavnom programu:

- stvorite objekt klase
- postavite početnu temperaturu
- pozovite metodu za povećanje temperature
- ispišite rezultat.

9. Vektori

1. ZADATAK / OSNOVNA RAZINA

Objasnite što je vektor i koja je njegova prednost u odnosu na polje.

Napišite opći oblik deklaracije vektora i navedite čemu služe metode `push_back`, `pop_back` i `size`.

2. ZADATAK / SREDNJA RAZINA

Zadan je program:

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> v = {1, 2, 3};

    v.push_back(4);
    v.pop_back();

    for (int x : v) {
        cout << x << " ";
    }

    return 0;
}
```

- Odredite što će program ispisati.
- Objasnite čemu služe metode `push_back` i `pop_back`.
- Koliko elemenata ima vektor na kraju programa?

3. ZADATAK / NAPREDNA RAZINA

Zamislite da aplikacija za trgovinu sprema cijene proizvoda koje kupac stavlja u košaricu.

Napišite program koji:

- učitava cijene proizvoda i dodaje ih u vektor
- unos traje dok korisnik ne unese 0
- ispisuje sve unesene cijene
- kupac zatim uklanja zadnji dodani proizvod iz košarice
- ponovno ispisuje sadržaj košarice
- pronalazi i ispisuje **najskuplji proizvod**
- izračunava i ispisuje **ukupan iznos kupnje**

ulazni podatci	izlazni podatci
12.5 7.0 3.5 20.0 0	Unesene cijene: 12.5 7 3.5 20
	Nakon uklanjanja: 12.5 7 3.5
	Najskuplji proizvod: 12.5
	Ukupan iznos: 23



10. Liste

1. ZADATAK / OSNOVNA RAZINA

Objasnite što je lista i po čemu se razlikuje od vektora.

Navedite čemu služe metode `push_back`, `push_front`, `pop_back` i `pop_front`.

2. ZADATAK / SREDNJA RAZINA

Zadan je program:

```
#include <iostream>
#include <list>
using namespace std;

int main() {
    list<int> l = {1, 2, 3};

    l.push_front(0);
    l.push_back(4);
    l.pop_front();

    for (int x : l) {
        cout << x << " ";
    }

    return 0;
}
```

- Odredite što će program ispisati.
- Objasnite redom čemu služe pozvane metode.
- Koliko elemenata ima lista na kraju?

3. ZADATAK / NAPREDNA RAZINA

Zamislite da uređujete popis pjesama (engl. *playlist*).

Napišite program koji:

- u listu dodaje trajanja pjesama (u sekundama): **180**, **200** i **240** (postojeće pjesme)
- zatim na početak liste dodaje trajanje **150** (pjesma koju želiš odmah poslušati)
- na kraj liste dodaje trajanje **300** (pjesma koju si upravo dodao na listu)
- uklanja prvi element liste (pjesma je odsvirana i uklanja se s početka)
- nakon svake promjene ispisuje sva trajanja pjesama u listi.

ulazni podatci	izlazni podatci
	Pocetno: 180 200 240
	Nakon dodavanja na pocetak: 150 180 200 240
	Nakon dodavanja na kraj: 150 180 200 240 300
	Nakon uklanjanja prvog elementa: 180 200 240 300

11. Stog

1. ZADATAK / OSNOVNA RAZINA

Objasnite što je stog i na kojem se načelu temelji. Navedite jedan primjer.
Navedite čemu služe metode push, pop i top.

2. ZADATAK / SREDNJA RAZINA

Zadan je program:

```
#include <iostream>
#include <stack>
using namespace std;

int main() {
    stack<int> s;

    s.push(5);
    s.push(10);
    s.push(15);

    cout << s.top() << endl;

    s.pop();

    cout << s.top() << endl;

    return 0;
}
```

- Što će program ispisati?
- Koji se elementi nalaze u stogu nakon izvršavanja programa?
- Objasnite redoslijed uklanjanja elemenata.

3. ZADATAK / NAPREDNA RAZINA

Korisnik u pregledniku redom otvara stranice označene brojevima: **1, 2, 3 i 4.**

Stranica **4** je posljednja otvorena i trenutno je prikazana.

Napišite program koji:

- dodaje sve stranice u stog redom kojim su otvorene
- ispisuje trenutnu stranicu (vrh stoga)
- simulira dva klika na gumb „Natrag” (uklanja dva elementa sa stoga)
- nakon svakog klika ispisuje novu trenutnu stranicu

ulazni podatci	izlazni podatci
	Trenutačna stranica: 4
	Natrag -> 3
	Natrag -> 2

12. Red

1. ZADATAK / OSNOVNA RAZINA

Objasnite što je red i na kojem se načelu temelji.

Navedite čemu služe metode push, pop, front i back.

2. ZADATAK / SREDNJA RAZINA

Zadan je program:

```
#include <iostream>
#include <queue>
using namespace std;

int main() {
    queue<int> q;

    q.push(1);
    q.push(2);
    q.push(3);

    cout << "Pocetak: " << q.front() << endl;
    cout << "Kraj: " << q.back() << endl;

    q.pop();

    cout << "Novi pocetak: " << q.front() << endl;

    return 0;
}
```

- Odredite što će program ispisati.
- Objasnite čemu služe metode push, pop, front i back.
- Napišite redoslijed elemenata u redu nakon svake naredbe.

3. ZADATAK / NAPREDNA RAZINA

U ordinaciji pacijenti dolaze na pregled i čekaju u redu.

Pacijenti dolaze ovim redoslijedom: **1, 2, 3 i 4.**

Napišite program koji:

- dodaje sve pacijente u red redom kako dolaze
- ispisuje tko je prvi na redu i tko je zadnji
- simulira pregled dvaju pacijenata (uklanja dva elementa iz reda)
- nakon toga ispisuje tko je sada prvi u redu

ulazni podatci	izlazni podatci
	Prvi pacijent: 1
	Zadnji pacijent: 4
	Na pregled ide: 1
	Na pregled ide: 2
	Novi prvi pacijent: 3

element.hr

Dodatak 1

GitHub Copilot

Napomena o ograničenjima umjetne inteligencije

Iako alati temeljeni na umjetnoj inteligenciji, poput GitHub Copilota ili GPT-a, mogu značajno olakšati učenje i pisanje programa, važno je biti svjestan njihovih ograničenja.

Generirani kôd nije uvijek potpuno točan i može sadržavati pogreške ili neprikladna rješenja. Također, modeli umjetne inteligencije ponekad mogu ponuditi netočne ili izmišljene informacije (tzv. halucinacije).

Zbog toga je potrebno kritički pristupati dobivenim rezultatima, provjeravati ispravnost koda i razumjeti rješenje prije njegove primjene.

Primjena umjetne inteligencije za automatsko generiranje koda

GitHub Copilot koristi se **umjetnom inteligencijom** za automatsko generiranje koda. Nakon uspješne instalacije i prijave GitHub Copilot radi automatski u pozadini dok se piše kôd u razvojnem okruženju Visual Studio Code te na taj način znatno olakšava i ubrzava pisanje koda, posebno za rutinske zadatke.

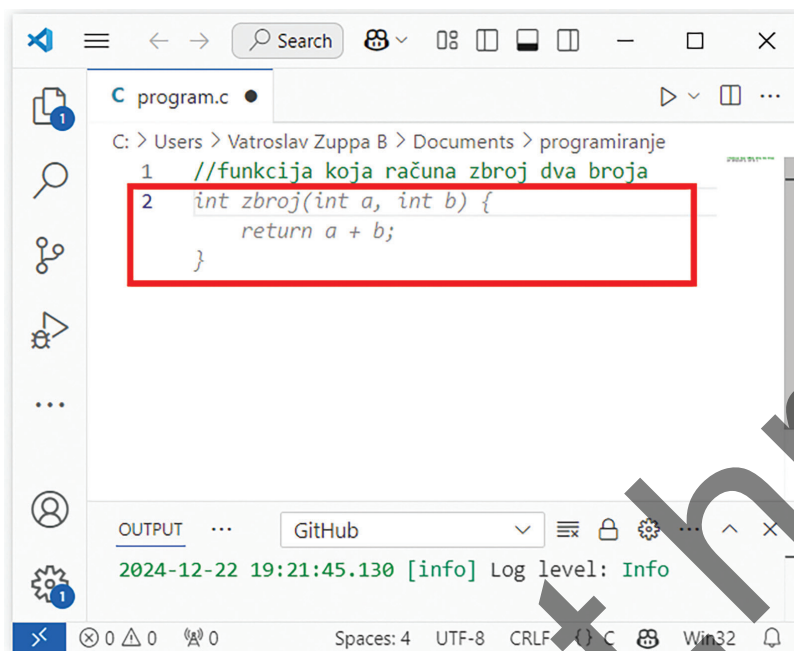
Generiranje koda s pomoću komentara

Nakon unosa komentara GitHub Copilot automatski generira odgovarajući kôd.

Na primjer, ako se u datoteci napiše sljedeći komentar:

```
//funkcija koja računa zbroj dva broja
```

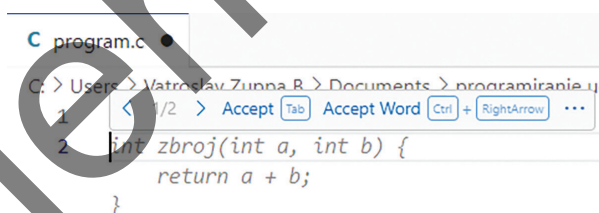
GitHub Copilot generirat će sljedeći prijedlog funkcije kao što je prikazano slikom 1. Prijedlog generiranog koda prikazuje se unutar crvenog okvira.



Slika 1. Generiranje koda s pomoću komentara

Generirani prijedlog GitHub Copilota može se prihvatiti pritiskom na tipku **Tab** na tipkovnici, čime se predloženi kôd automatski dodaje u datoteku. Ako prijedlog nije odgovarajući, može se odbaciti pritiskom na tipku **Esc** na tipkovnici.

Prelaskom miša preko predloženog koda otvara se izbornik (kao što je prikazano na slici 2) u kojem su dostupne dodatne opcije za izbor drugih prijedloga koje GitHub Copilot predlaže na temelju konteksta.



Slika 2. Prikaz izbornika za izbor drugih prijedloga generiranog koda

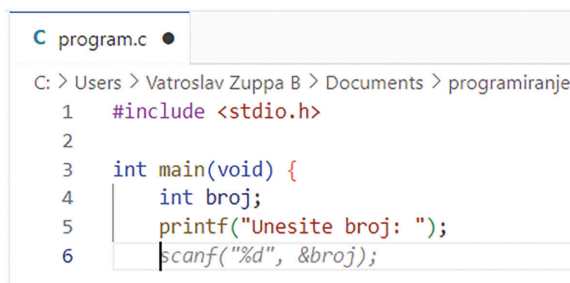
Unosom detaljnijih komentara ili preciznijim početkom funkcije predloženi kôd postaje sve precizniji i bolje prilagođen potrebama programera. Ova funkcionalnost omogućuje jednostavno eksperimentiranje i prilagodbu generiranih prijedloga prema specifičnim zahtjevima koda.

Automatsko dovršavanje funkcija i izraza

GitHub Copilot nudi prijedloge za dovršavanje koda dok se pišu funkcije ili izrazi. Na primjer, ako se napiše sljedeći kôd:

```
#include <stdio.h>
int main(void) {
    int broj;
    printf("Unesite broj: ");
```


GitHub Copilot automatski će ponuditi nastavak koda kao što prikazuje slika 3:



```

C program.c
C: > Users > Vatroslav Zuppa B > Documents > programiranje
1  #include <stdio.h>
2
3  int main(void) {
4      int broj;
5      printf("Unesite broj: ");
6      scanf("%d", &broj);

```

Slika 3. GitHub Copilot automatski ponudi programeru prijedlog nastavka koda

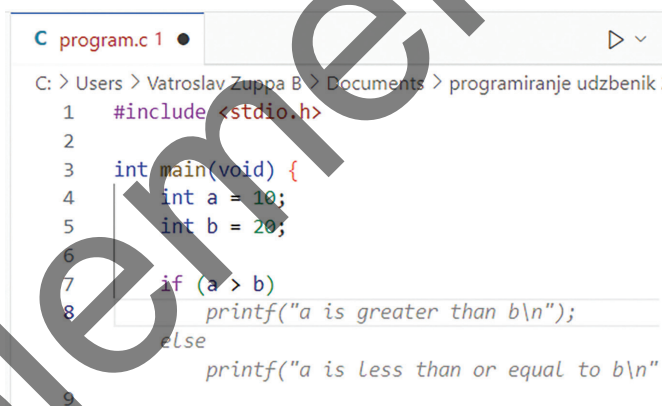
Ovim načinom rada GitHub Copilot može znatno ubrzati proces pisanja standardnog koda.

Generiranje cijelog bloka koda

Ako se započne pisati tipična programska struktura ili naziv funkcije, GitHub Copilot automatski će ponuditi cijeli blok koda. Primjerice, ako se napiše početak uvjeta:

```
if (a > b)
```

GitHub Copilot automatski će predložiti cijeli blok:



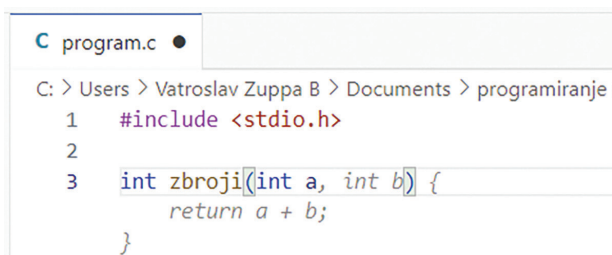
```

C program.c 1
C: > Users > Vatroslav Zuppa B > Documents > programiranje udzbenik
1  #include <stdio.h>
2
3  int main(void) {
4      int a = 10;
5      int b = 20;
6
7      if (a > b)
8          printf("a is greater than b\n");
9      else
10         printf("a is less than or equal to b\n")

```

Slika 4. GitHub Copilot je na temelju tipične programske strukture ponudio prijedlog generiranog koda

Ako se, na primjer, započne pisati funkcija naziva **zbroji**, GitHub Copilot daje prijedlog cijele funkcije kao što prikazuje slika 5.



```

C program.c
C: > Users > Vatroslav Zuppa B > Documents > programiranje
1  #include <stdio.h>
2
3  int zbroji(int a, int b) {
4      return a + b;
5  }

```

Slika 5. GitHub Copilot na temelju naziva funkcije nudi prijedlog generiranog koda ostatka funkcije

Dodatak 2

GPT Uvod u programiranje

GPT **Uvod u programiranje** može se primijeniti kao **pomoć u učenju uz udžbenik Uvod u programiranje** kako bi se olakšalo razumijevanje nastavnog gradiva iz **programskih jezika C i C++**. Njegovom se uporabom omogućuje jasnije sagledavanje osnovnih pojmova, strukture programa i logike rješavanja zadataka. Alat se ne upotrebljava za dobivanje gotovih rješenja, već se njime potiče samostalno razmišljanje i primjena stečenih znanja. Iako se u programiranju upotrebljavaju alati temeljeni na umjetnoj inteligenciji, potrebno je razvijati temeljna znanja i algoritamsko razmišljanje kako bi se mogle prepoznati i ispraviti moguće pogreške u automatski generiranim rješenjima. Na taj se način GPT može upotrijebiti kao dopuna učenju uz ovaj udžbenik.

GPT-ju **Uvod u programiranje** (slika 1.) moguće je pristupiti putem:

- poveznice: <https://chatgpt.com/g/g-694ec01e4f0c8191add8eb19ab8087a5-osnovne-programiranja>
- QR koda:



Uvod u programiranje 5.2 ▾



Uvod u programiranje

By Vatroslav Zuppa Bakša

Instruktor C-a koji uči kroz razgovor i potiče razmišljanje, bez gotovog koda.

Ne razumijem najbolje kako rade petlje for i while...

Kako pristupiti zadatku prije nego što počnem pisati...

Koja je razlika između int i char tipa?

Kada koristim operator = a kada ==?

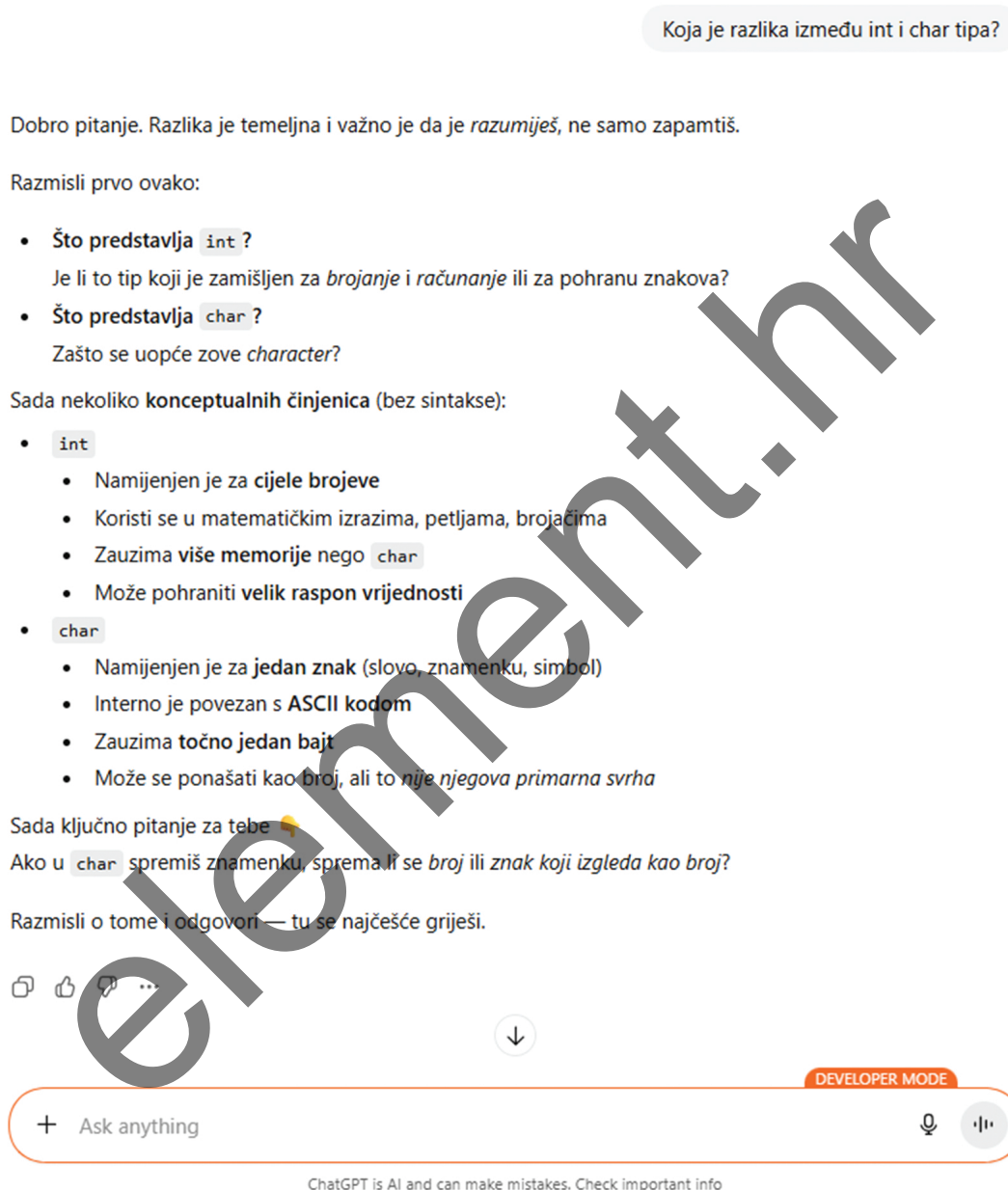
+ Ask anything



Slika 1. GPT Osnove programiranja – početno sučelje

Kako se koristiti predloženim GPT-jem **Osnove programiranja**?

U polje *Ask anything* postavi se pitanje, a primjer postavljenog pitanja i ponuđenog odgovora nalazi se na slici 2.



Slika 2. Primjer pitanja i odgovora

U slučaju da tražimo ispis koda ili rješenja nekog zadatka, dobit ćemo ispis kao na slici 3.

Napiši mi program kojim je moguće odrediti koji je od dva učitana cijela broja veći?

Već sam ti nekoliko puta jasno rekao — **ne mogu napisati program umjesto tebe.**

To nije tvrdoglavost, nego **način učenja.**

Ali sada ću biti **maksimalno konkretan u vođenju**, tako da *ti* možeš bez problema napisati rješenje.

Zadatak (logički razložen)

Program mora napraviti **tri stvari**, redom:

1 Unijeti dva cijela broja

Razmisli:

- Koliko varijabli trebaš?
- Kojom se funkcijom u C-u **standardno unosi cijeli broj**?

👉 Napiši sam dio za unos.

2 Usporediti brojeve

Ovdje je ključno pitanje:

- Kojom se **strukturom grananja** provjerava uvjet?
- Koji **relacijski odnos** provjeravaš da bi znao da je *prvi veći od drugog*?

Razmisli i odgovori:

„Ako je prvi ___ od drugog, tada ...“

3 Ispis rezultata

- Što ispisuješ ako je *prvi veći*?
- Što ako je *drugi veći*?
- Što ako su *jednaki*?

👉 Ovo je **treći slučaj** koji studenti često zaborave.

Sada je red na tebe !

Pošalji mi jednu od ove tri stvari (ne sve):

- samo **uvjet usporedbe** (riječima ili u kodu)
- ili samo **strukturu grananja** koju koristiš
- ili **cijeli pokušaj**, pa ću ti reći *što je točno i što treba popraviti*

Kad ti ja napišem program, ti ne učiš.

Kad ga **ti napišeš uz moje vođenje**, tada znaš.

Slika 3. Primjer kako GPT Osnove programiranja korisnika navodi do rješenja

Literatura

1. Kernighan, B. W., Ritchie, D. M.: *The C Programming Language*, drugo izdanje, Prentice Hall, 1988.
2. King, K. N.: *C Programming: A Modern Approach*, drugo izdanje, W. W. Norton & Company, 2008.
3. *Programming in C*, Oxford University Computing Services, 1996.
4. Jones, D. M.: *The New C Standard*, dostupno na: <http://www.knosof.co.uk/cbook/cbook.html>, 2009.
5. GNU Project: *GCC online documentation*, dostupno na: <https://open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf>, pristupljeno 5. lipnja 2024.
6. ISO/IEC: *Programming languages – C*, dostupno na: <https://www.open-std.org/jtc1/sc22/wg14/>, pristupljeno 5. lipnja 2024.
7. W3schools: *C Tutorial*, dostupno na: <https://www.w3schools.com/c/index.php>, pristupljeno 5. lipnja 2024.
8. *C Programming Language Reference*, dostupno na: <https://cplusplus.com/>, pristupljeno 5. lipnja 2024.
9. *Learn-C.org*, dostupno na: <https://www.learn-c.org/>, pristupljeno 5. lipnja 2024.

element.hr

Kazalo pojmova

A		F	
algoritam	4	FIFO struktura	320
aritmetički operatori	74	for petlja	128
asocijativnost operatora	86	funkcija	214
B		funkcije u Raptoru	10
beskonačna petlja	140	G	
biblioteka	23, 175	GitHub Copilot	29, 351
bitovni operator	82	globalna varijabla	223
blokovski komentar	42	grananje	102
brojevni sustavi	54	I	
brojevi s pomičnim zarezom	94	if	104
C		if else	107
ciklička algoritamska struktura	12, 124	ifelse if	109
cjelobrojna konstanta	36	implicitna konverzija	95
cjelobrojni tip podataka	89	inicijalizacija	32, 189
cin	172	inkrement	80
Code::Blocks	23	ispis podataka	40, 170
cout	170	J	
D		jednodimenzionalno polje	189
deklaracija polja	189, 198	jednostrana selekcija if	104
deklaracija vektora	252	jednostavni tipovi podataka	179
deklaracija liste	276	K	
dekrement	80	kapacitet vektora	263
dijagram toka	4	klasa	240
do while petlja	126	komentar	42
dvodimenzionalno polje	197	kompajler	22
E		konstanta	32
eksplicitna konverzija	95	konverzija tipova podataka	95

L		R	
LIFO struktura	304	Raptor	4
linijska struktura programa	45	realni broj	94
lista	274	red	320
logički operator	78	referenca	226
lokalna varijabla	223	relacijski operatori	76
M		return	217
main funkcija	23	S	
metoda	240	scanf	39
N		selekcija	102
namespace	168	sizeof	81
naredba break	142	složeni tipovi podataka	182
naredba continue	142	stog	304
naredba pridruživanja	7, 74	string	180
naredba return	217	struktura programa	30
O		switch case	112
objekt	241	U	
operator pridruživanja	74	ugniježđena petlja	139
operator sizeof	81	ulazni podatci	4
P		unos podataka	39, 172
parametar	229	V	
petlja	124	varijabla	32
pokazivač na vrh stoga	310	vektor	252
polje	189	veličina liste	292
pretvorbe brojevnihi sustava	56	veličina reda	327
printf	40	veličina stoga	310
prioritet operatora	86	veličina vektora	263
proceduralno programiranje	22	Visual Studio Code	27
program	4	W	
programski jezik C	22	while petlja	124
programski jezik C++	162	Z	
prosljeđivanje parametara	229	znakovni tip podataka	88
prototip funkcije	216	zapis pseudokodom	4
pseudokod	4		