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 ću

- 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++.

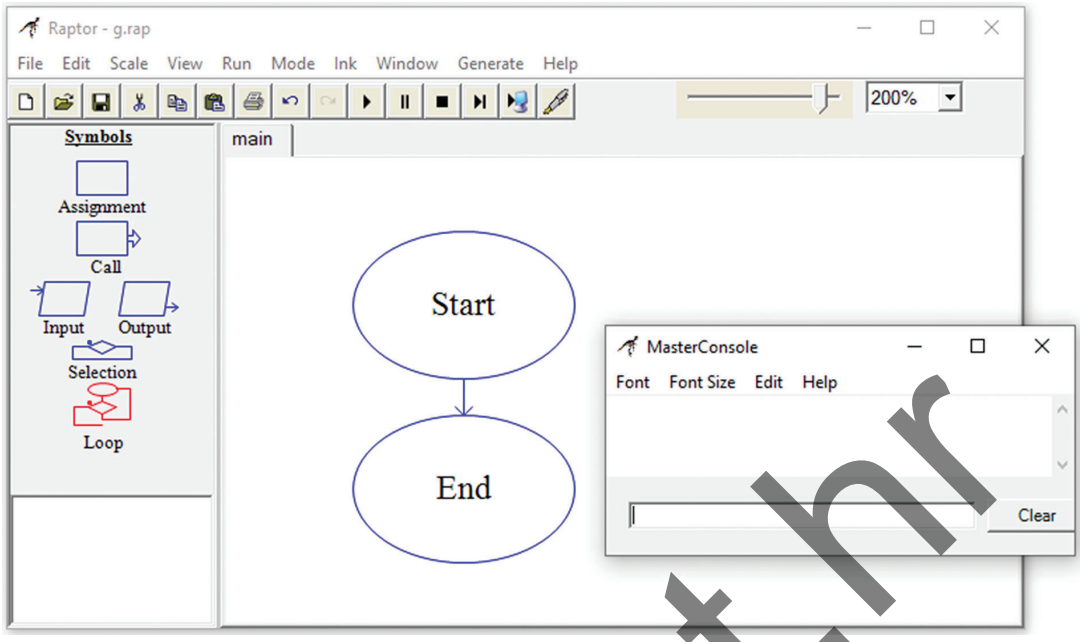
1.2. Pseudokod, dijagram toka i instalacija programa Raptor

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/.

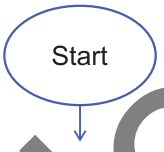
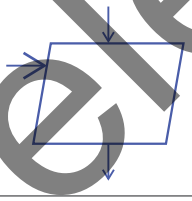
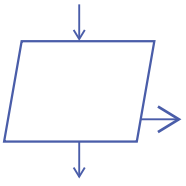
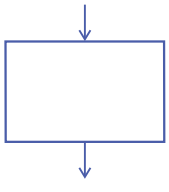
Nakon jednostavne instalacije te pokretanjem programa dobiva se prikaz kao na slici 1.1. U desnom kutu nalazi se padajući izbornik s mogućnošću povećavanja i smanjivanja veličine prikaza dijagrama (trenutačno je odabran prikaz na 200 %).



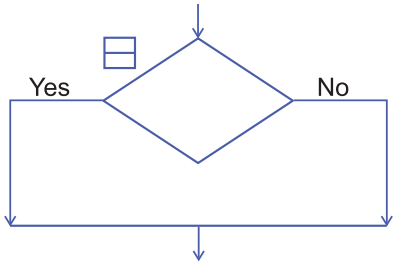
Slika 1.1. Program Raptor

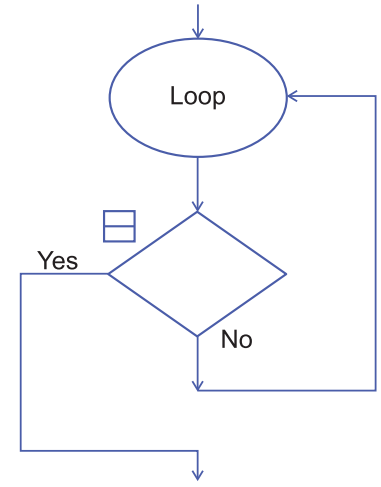
Simboli koji se upotrebljavaju prilikom izrade dijagrama toka te odgovarajuće naredbe u pseudokodu prikazani su u tablici 1.1.

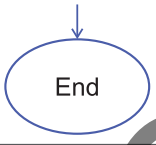
Tablica 1.1. Pregled i opis simbola u programu Raptor

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

simboli	zapis u pseudokodu	opis

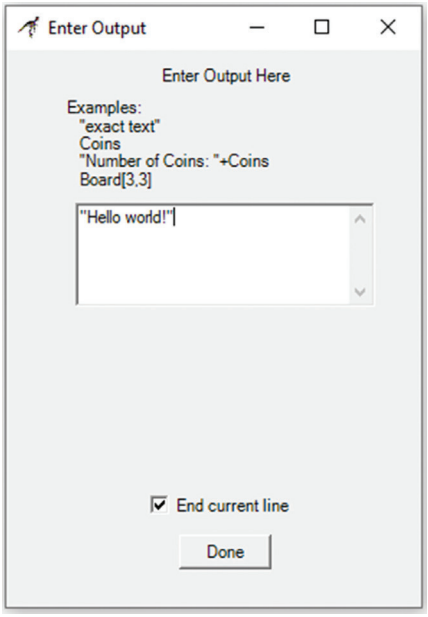
GRANANJE 	ako je uvjet onda {
	  naredba_1
	} inače {
	  naredba_2
	}
		odluka se donosi na temelju uvjeta unutar romba, a izvršavanje se nastavlja jednim od dvaju mogućih smjerova (unutar romba piše se logički uvjet - primjeri u tablici 1.3)

PROGRAMSKA PETLJA 	dok je uvjet činiti {
	  blok_naredbi
	}
	
	ponavljati {
	  blok_naredbi
	} dok je uvjet
		• 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

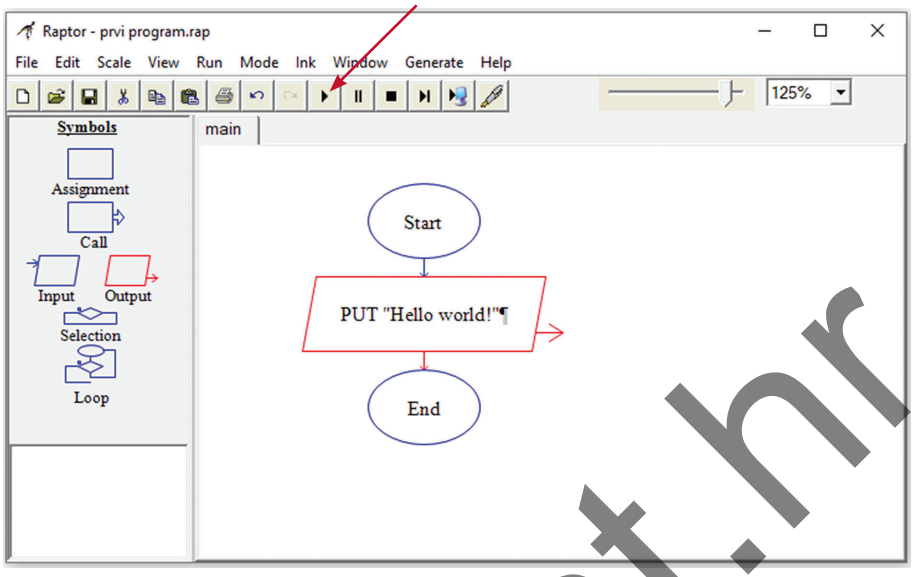
1.3. Pokretanje prvog programa u Raptoru

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).



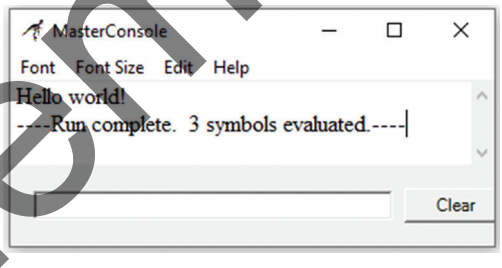
Slika 1.2. Upis teksta unosi se unutar dvostrukih navodnika

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.



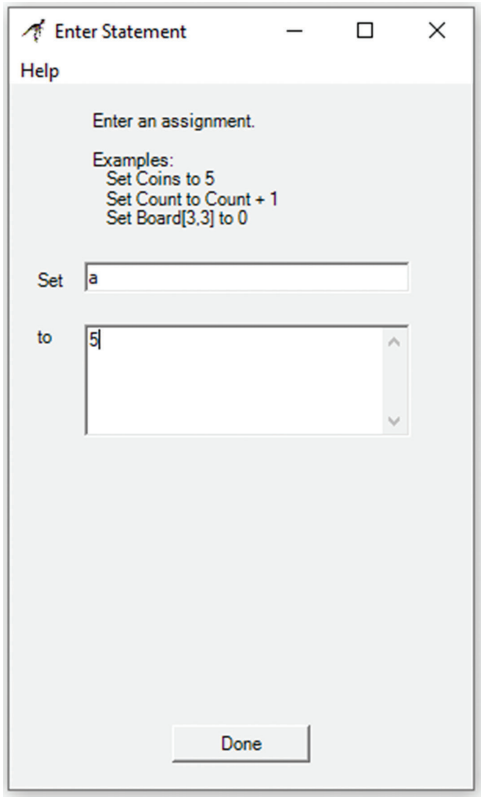
Slika 1.4. Prikaz rezultata izvođenja programa

1.4. Osnovna pravila prilikom pisanja algoritama u programu Raptor

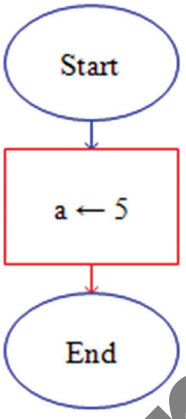
1.4.1. Varijable, naredba pridruživanja i ispis vrijednosti na zaslonu računala

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

Naredba pridruživanja radi se s pomoću bloka *Assignement*. 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.

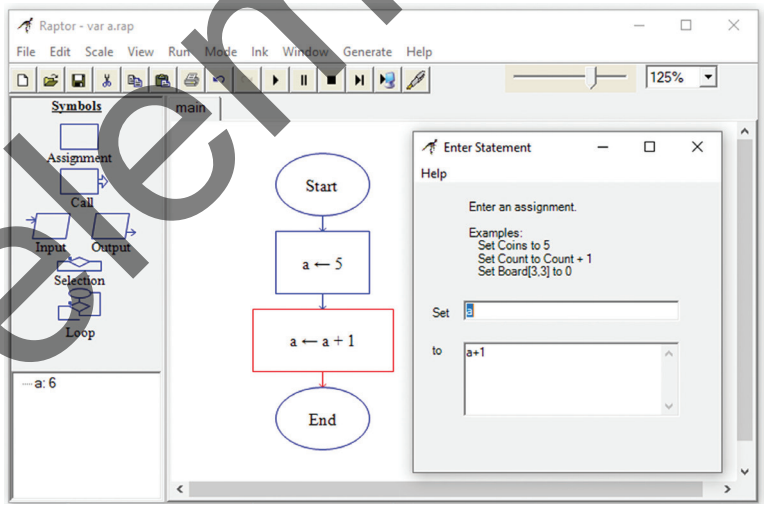


Slika 1.5. Definiranje i inicijaliziranje varijable



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

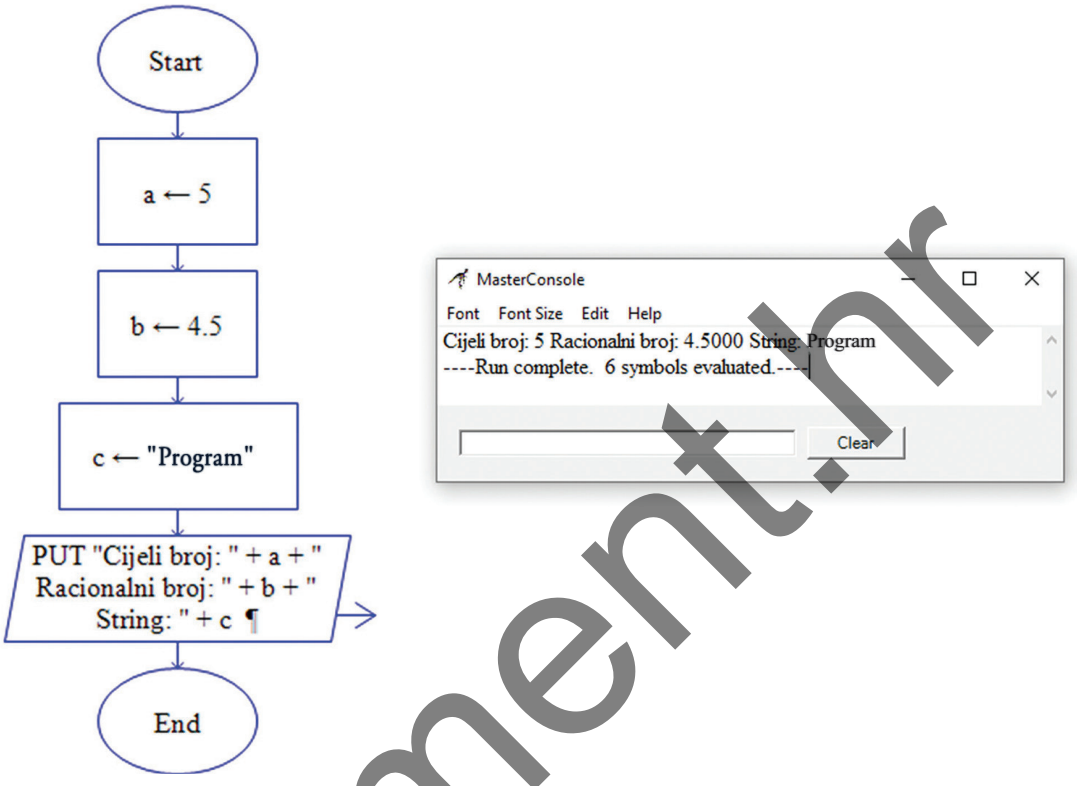
Ako želimo vrijednost varijable *a* uvećati za 1, može se dodati i crvenom bojom označen blok *Assignment* kao što prikazuje slika 1.7.



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

Prilikom pisanja uvjeta u razgranatoj 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 and true`
		`true and false`
		`true and true`
			`false`
			`false`
			`false`
			`true`

	ILI or	`false or false`
		`false or true`
		`true or false`
		`true or true`
			`false`
			`true`
			`true`
			`true`

	NE not	`not(true)` `not(false)`	`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**:

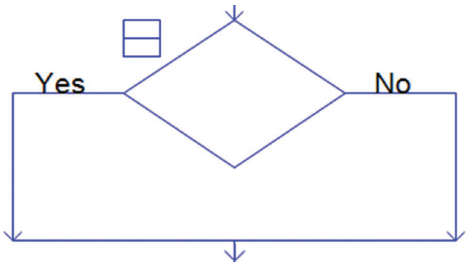
a) **linijska** algoritamska struktura (slijed)

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:

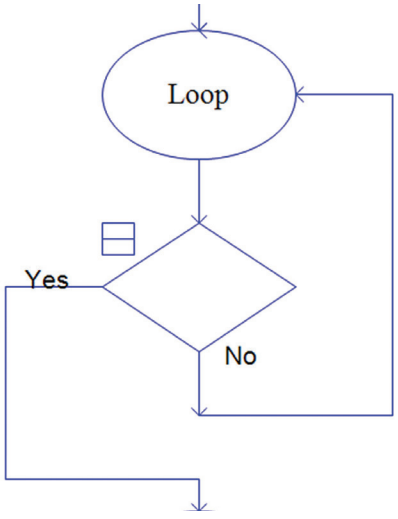
```
ako je uvjet onda {
    blok_naredbi
}
```

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**

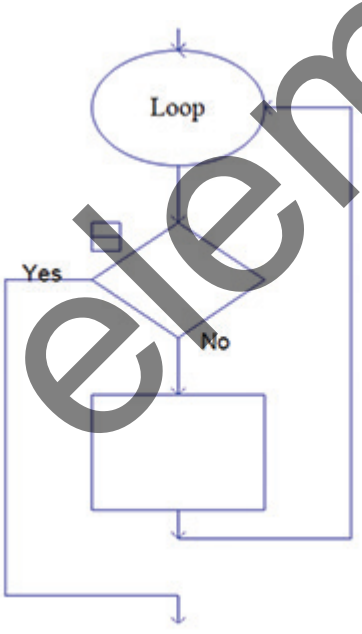
(*ponavljanje dok je uvjet istinit – kontrola na ulazu*), slika 1.11

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

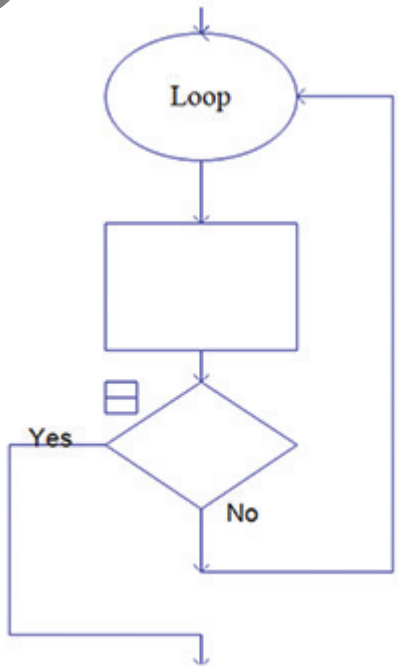
3. **petlja kod koje nije unaprijed poznat broj ponavljanja, a uvjet se provjerava na kraju petlje**

(*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

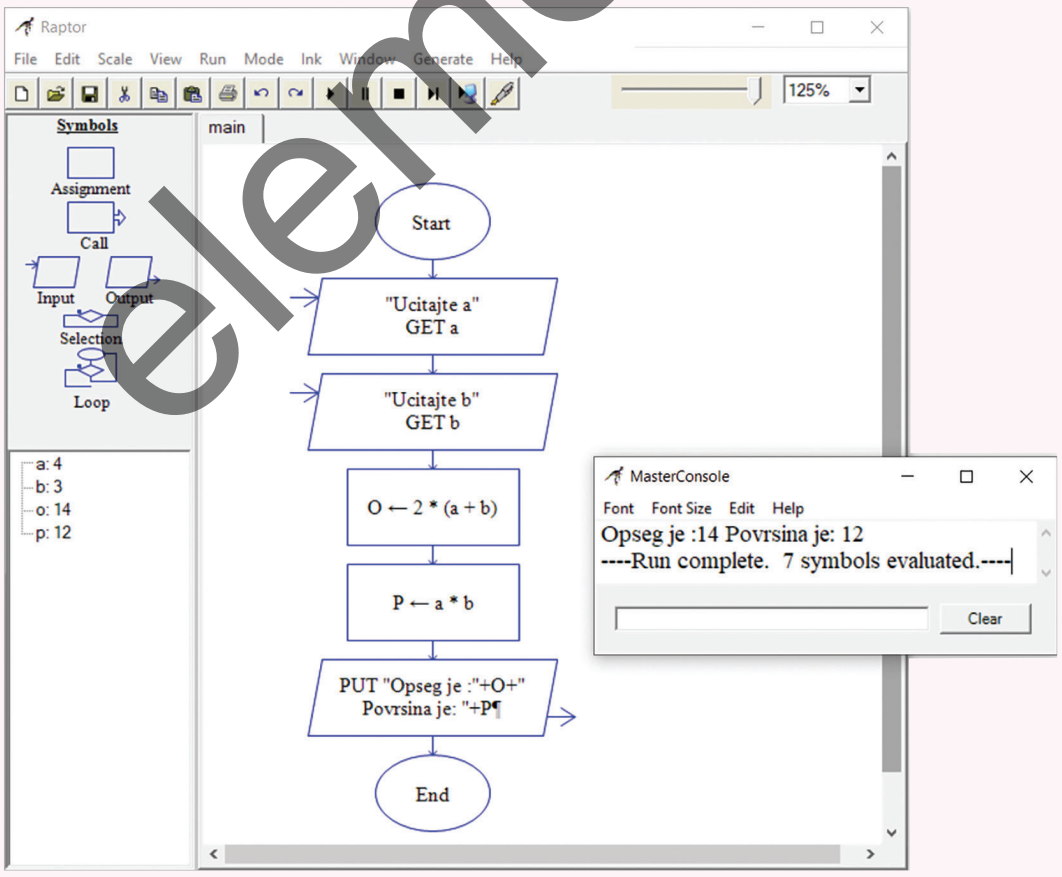
Rješenje

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

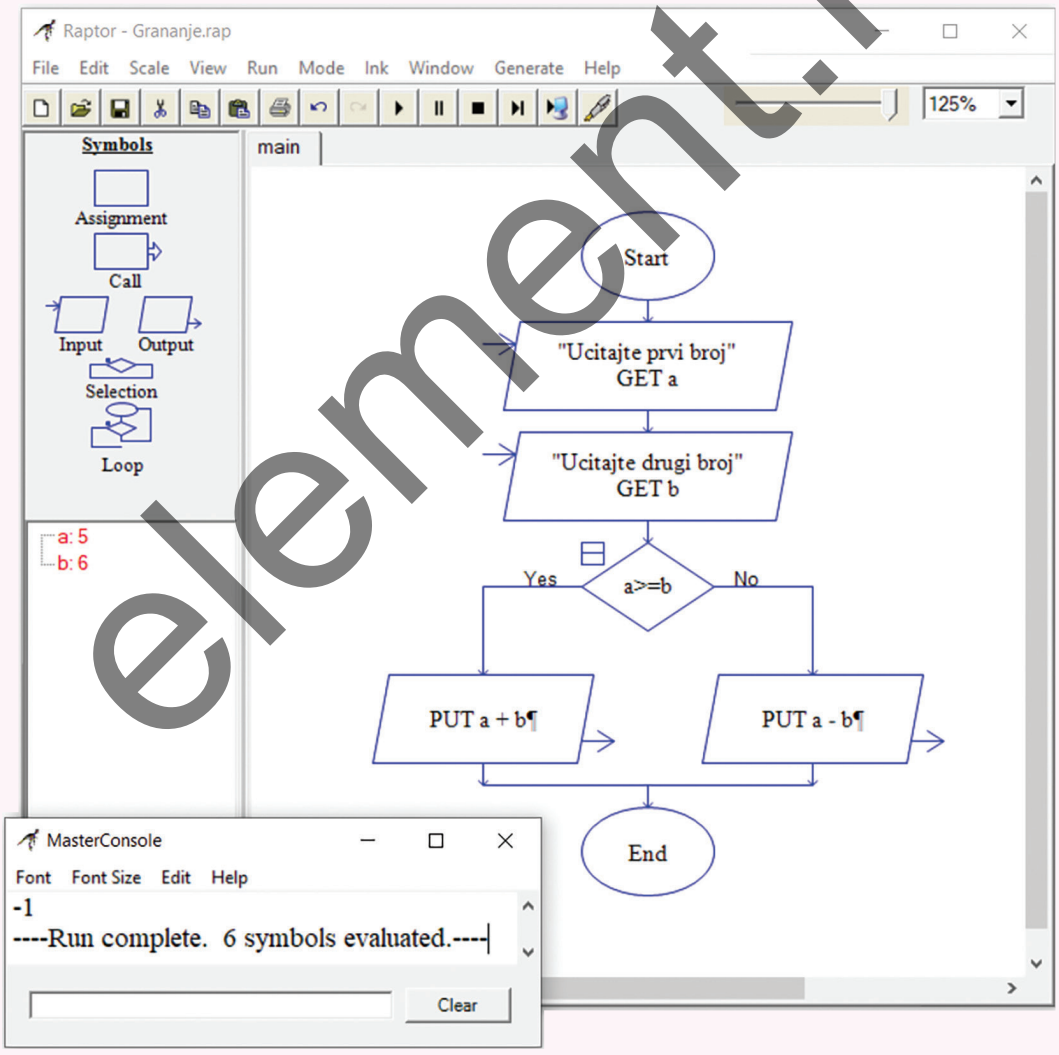
Primjer 1.2.

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 6	–1
6 5	11

Rješenje

Ovo je primjer algoritma razgranate strukture (slika 1.14.). U uvjetu se upotrebljava relacijski operator >=. Prvi testni primjer (5, 6) ispisuje razliku učitanih brojeva jer uvjet kao rezultat daje logičku vrijednost `false` (5 >= 6) te se algoritam nakon učitavanja podataka nastavlja izvršavati u grani *No*. Ako se učita 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 → ispis zbroja,
a < b → 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 >= 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

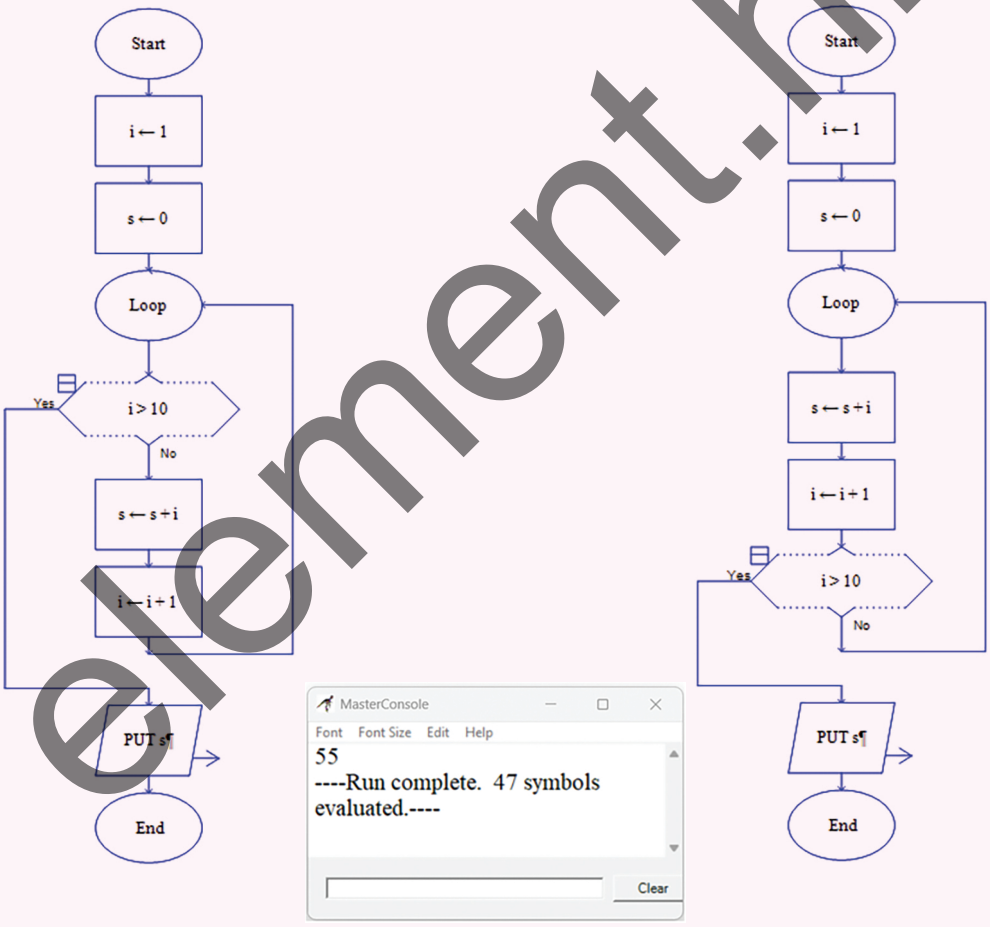
```
početak
    s = 0
    za i = 1 do 10 činiti {
        s = s + i
    }
    izlaz(s)
kraj
```

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 <= 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 osmisliti algoritam koji omogućuje unos ocjena iz više modula, izračun prosjeka i ispis poruke o uspjehu.

Rješenje zapiši **pseudokodom** i prikaži **dijagramom 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