0. Programski jezik C – ponavljanje

U uvodnom ćemo poglavlju ponoviti osnove programskog jezika C s kojima smo se susreli u 1. razredu. U kratkom pregledu navedeni su operatori (aritmetički, relacijski i logički), neke od funkcija iz biblioteka **math.h** i **stdlib.h** te popis osnovnih naredbi. Uz funkcije za unos i ispis podataka, navedene su naredbe za grananje (`if`, `if-else` i `switch-case`) te naredbe za ponavljanje (`for`, `while` i `do-while`). Na kraju pregleda pojašnjeno je kako se deklarira, unosi i ispisuje jednodimenzionalno polje, dvodimenzionalno polje te niz znakova – string.

Aritmetički operatori

Opis	Pseudojezik	C
zbrajanje	+	+
oduzimanje	-	-
množenje	*	*
dijeljenje	/	/
cjelobrojno dijeljenje	DIV	/
ostatak cjelobrojnog dijeljenja	MOD	%

Relacijski operatori

Opis	Pseudojezik	C
manje	<	<
manje ili jednako	<=	<=
veće	>	>
veće ili jednako	>=	>=
jednako	=	==
različito	<>	!=

Logički operatori prema prioritetu

Opis	Pseudojezik	C
logički NE	NE	!
logički I	I	&&
logički ILI	ILI	||

Osnovne definirane funkcije

Opis	Pseudojezik	C
apsolutna vrijednost realnoga broja	`abs(x)`	`abs(x)` – za cijele brojeve `fabs(x)` – za realne brojeve
kvadrat broja	`sqr(x)`	`pow(x, 2)`
drugi korijen realnog broja	`sqrt(x)`	`sqrt(x)`
zaokruživanje realnog broja na najbliži cijeli broj	`round(x)`	`round(x)`
cijeli dio realnog broja x	`trunc(x)`	`trunc(x)`

Prioritet operatora (u tablici su navedni operatori iz C jezika)

Prioritet operatora	Operatori
1	( )
2	!
3	* / %
4	+ -
5	< <= > >=
6	== !=
7	&&
8	||

Popis osnovnih pojmova

Opis	Pseudojezik	C
unos	`ulaz`	`scanf`
ispis	`izlaz`	`printf`
pridruživanje	`=`	`=`
blok naredbi	`{` `}`	`{` `}`

grananje – jednostruko	`ako je(uvjet) onda{`
	`...`
	`naredbe;`
	`...`
	`}`
		`if(uvjet){`
		`...`
		`naredbe;`
		`...`
		`}`

<table>
<tr>
<td>grananje – dvostruko</td>
<td>

```
ako je(uvjet) onda{
     ...
     naredbe;
     ...
}
inače{
     ...
     naredbe;
     ...
}
```

</td>
<td>

```
if(uvjet){
     ...
     naredbe;
     ...
}
else{
     ...
     naredbe;
     ...
}
```

</td>
</tr>
<tr>
<td>grananje – višestruko</td>
<td>

```
skretnica(uvjet){
     slučaj izraz_1:
          ...
          naredbe;
          ...
     slučaj izraz_2:
          ...
          naredbe;
          ...
     ...
     slučaj izraz_N:
          ...
          naredbe;
          ...
     inače:
          ...
          naredbe;
          ...
}
```

</td>
<td>

```
switch(uvjet){
     case izraz_1:
          ...
          naredbe;
          ...
          break;
     case izraz_2:
          ...
          naredbe;
          ...
          break;
     ...
     case izraz_N:
          ...
          naredbe;
          ...
          break;
     default:
          ...
          naredbe;
          ...
}
```

</td>
</tr>
<tr>
<td>petlja s unaprijed poznatim brojem ponavljanja</td>
<td>

```
za i = p do k činiti{
     naredbe;
}
```

</td>
<td>

```
for(inicijalizacija; uvjet;
                korak petlje){
     naredbe;
}
```

</td>
</tr>
</table>

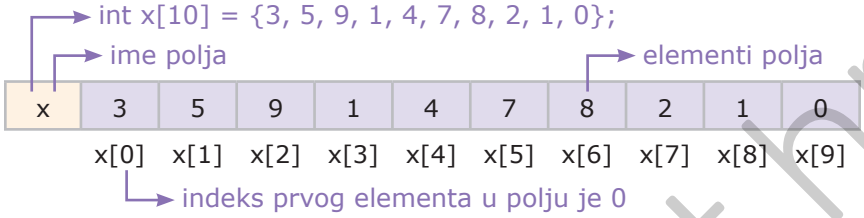
petlja kod koje nije unaprijed poznat broj ponavljanja, a uvjet se provjerava na početku petlje	`dok je(uvjet) činiti{ naredbe; }`	`while(uvjet){ naredbe; }`
petlja kod koje nije unaprijed poznat broj ponavljanja, a uvjet se provjerava na kraju petlje	`ponavljati{ naredbe; }dok je(uvjet);`	`do{ naredbe; }while(uvjet);`

Inicijalizacija polja (jednodimenzionalnog, dvodimenzionalnog i niza znakova)

inicijalizacija jednodimenzionalnog polja

```
tip_polja ime_polja[broj_elemenata_polja];
tip_polja ime_polja[broj_elemenata_polja]
                    = {elementi_polja};
```

```
int x[10] = {3, 5, 9, 1, 4, 7, 8, 2, 1, 0};
```



inicijalizacija dvodimenzionalnog polja

```
tip_polja ime_polja[broj_redaka][broj_stupaca];
tip_polja ime_polja[broj_redaka][broj_stupaca]
                    = {elementi_polja};
```

inicijalizacija niza znakova (stringa)

```
char x[] = "Algoritam";
char x[] = {'A','l','g','o','r','i',t','a','m','\0'};
char x[9+1] = {'A','l','g','o','r','i',t','a','m','\0'};
```

x[i]	A	l	g	o	r	i	t	a	m	\0	→ Elementi polja su znakovi
i	0	1	2	3	4	5	6	7	8	9	→ Indeksi polja

Učitavanje i ispis elemenata iz polja (jednodimenzionalnog, dvodimenzionalnog i niza znakova)

Učitavanje *n* cjelobrojnih elemenata u polje:	`for(i = 0; i < n; i++)` `  scanf("%d", &polje[i]);`
Ispis *n* cjelobrojnih elemenata iz polja:	`for (i = 0; i < n; i++)` `  printf("%d", polje[i]);`

Učitavanje *n* × *m* cjelobrojnih elemenata u dvodimenzionalno polje:	`for(i = 0; i < n; i++){`
	`  for(j = 0; j < m; j++)`
	`  scanf("%d", &polje[i][j]);`
	`}`

Ispis *n* × *m* cjelobrojnih elemenata iz dvodimenzionalnog polja:	`for(i = 0; i < n; i++){`
	`  for(j = 0; j < m; j++)`
	`  printf("%d\t", polje[i][j]);`
	`  printf("\n");`
	`}`

Učitavanje niza znakova (stringa)	**1.** način: funkcija
	`gets(ime_stringa);`
	2. način: funkcija
	`scanf("%s", ime_stringa);`

Ispis niza znakova (stringa)	**1.** način: funkcija
	`puts(ime_stringa);`
	2. način: funkcija
	`printf("%s", ime_stringa);`
	3. način (string se zove s):
	`for(i = 0; i < strlen(s); i++)`
	`  printf("%c", s[i]);`
	4. način (string se zove s):
	`for(i = 0; s[i] != '\0'; i++)`
	`  printf("%c", s[i]);`

1. Funkcije

Što su to funkcije i kako se njima koristimo u programima?

Do sada smo se prilikom pisanja programa koristili gotovim funkcijama. Za upotrebu gotovih funkcija uvijek bismo prvo trebali napisati pretprocesorsku naredbu `include` unutar koje bismo naveli ime biblioteke. U svakoj biblioteci nalazi se gotov skup funkcija kojima se onda možemo koristiti. Tako primjerice ako se želimo koristiti funkcijama za unos i ispis podataka, trebamo navesti biblioteku `stdio.h`. Najavom ove biblioteke koristimo se **gotovim funkcijama**:

```
scanf("format", &varijabla);
printf("format", varijabla);
```

Za upotrebu **matematičkih funkcija** koristili smo se bibliotekama `math.h` i `stdlib.h`. Ponekad prilikom pisanja programa koji je sadržavao neku gotovu matematičku funkciju možda ste zaboravili uključiti biblioteku `math.h`. U tom slučaju prevodilac bi vam dojavio da je došlo do pogreške. Nakon što biste najavili izostavljenu biblioteku, program bi radio.

Ponovimo

Prisjetimo se kako bismo napisali program koji računa vrijednost funkcije $f(e) = b^e$, pri čemu za bazu i eksponent vrijedi: $b, e \in Q$. Uz pomoć funkcije potrebno je izračunati koliko je $27^{\frac{1}{3}}$ i $\sqrt[3]{27}$ te ispisati rezultate na zaslonu računala.

Rješenje:

Gotova funkcija za potenciranje u C-u je: `double pow(double b, double e);`. To znači da funkcija za potenciranje kao rezultat vraća decimalni broj – rezultat potenciranja, a baza i eksponent također mogu biti decimalni brojevi. Ova funkcija sadrži dva parametra (`b` i `e`) koja je obavezno navesti u zagradi funkcije. Ako navedemo manje ili više parametara od predviđenog broja, doći će do pogreške prilikom prevođenja koda.

```
double pow(double b, double e);
```

rezultat funkcije – decimalni broj (dvostruke preciznosti)

parametri funkcije (baza i ekesponent)

Prema pravilima konverzije podataka kod poziva funkcije, kao parametre možemo koristiti i cijele brojeve (`int`) i decimalne brojeve jednostruke preciznosti (`float`).

Program:

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

int main(void){

    float y, b, e;
    printf("Utipkaj vrijednost baze i eksponenta: ");
    scanf("%f %f", &b, &e);

//poziv gotove funkcije pow iz biblioteke math.h
    y = pow(b, e);

    printf("%.2f", y);

    return 0;
}
```

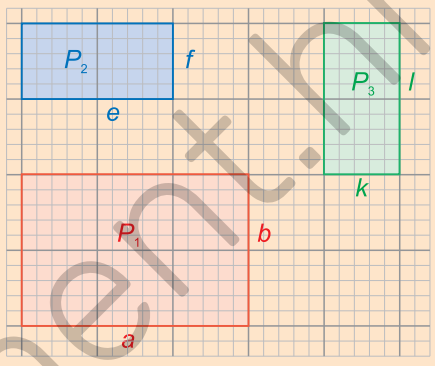
Ako želimo izračunati 27^3, u program upisujemo brojeve 27 i 3. Za broj $\sqrt[3]{27}$ potrebno je redom upisati 27 i $\frac{1}{3}$ jer je $\sqrt[3]{27} = 27^{\frac{1}{3}}$.

U ovom ćemo poglavlju naučiti kako možemo sami pisati vlastite funkcije koje nisu implementirane u standardnim bibliotekama programskog jezika C. To će nam olakšati pisanje složenih programa. Također ćemo pokazati kako možemo sami napisati funkciju za potenciranje koja će računati vrijednost funkcije $f(e) = b^e$ bez upotrebe već ugrađene funkcije `pow`.

Primjer 1.1.

Površina školskih objekata

Napiši program koji će računati koliku površinu zauzima školska zgrada, pripadajuća dvorana za tjelesni te školsko nogometno igralište prikazano slikom 1.1. Treba ispisati i ukupnu površinu. Sve dimenzije su dane u metrima.



Slika 1.1. Tlocrt škole (površina P1), tlocrt sportske dvorane (površina P2) te nogometno igralište (površina P3)

ULAZNI PODATCI

- duljina školske zgrade *a*
- širina školske zgrade *b*
- duljina dvorane za tjelesni *e*
- širina dvorane za tjelesni *f*
- duljina nogometnog igrališta *k*
- širina nogometnog igrališta *l*

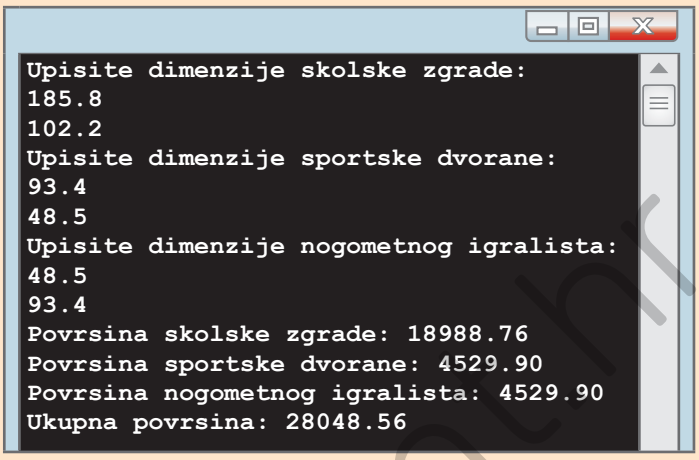
IZLAZNI PODATCI

- površina P1
- površina P2
- površina P3
- zajednička površina školske zgrade, sportske dvorane i nogometnog igrališta

PRIMJER TEST-PODATAKA

Ulaz	Izlaz
185.8 102.2 93.4 48.5 48.5 93.4	Površina školske zgrade: 18988.76 Površina sportske dvorane: 4529.90 Površina nogometnog igrališta: 4529.90 Ukupna površina: 28048.56

Ispis na zaslonu računala:



Kôd programa

```
#include <stdio.h>

int main(void){

    float a, b, e, f, k, l, P1, P2, P3, P;

    printf("Upisite dimenzije skolske zgrade: \n");
    scanf("%f %f", &a, &b);

    printf("Upisite dimenzije sportske dvorane: \n");
    scanf("%f %f", &e, &f);
```

```
    printf("Upisite dimenzije nogometnog igralista: \n");
    scanf("%f %f", &k, &l);

    P1 = a * b;
    P2 = e * f;
    P3 = k * l;

    P = P1 + P2 + P3;

    printf("Povrsina skolske zgrade: %.2f\n", P1);
    printf("Povrsina sportske dvorane: %.2f\n", P2);
    printf("Povrsina nogometnog igralista: %.2f\n", P3);
    printf("Ukupna povrsina: %.2f", P);

    return 0;
}
```

Funkcijama se općenito koristimo kada se isti programski odsječak ponavlja više puta. U 1. primjeru uočavamo da smo površinu pravokutnika u glavnom dijelu programa 3 puta trebali računati istom formulom (samo smo mijenjali varijable – dimenzije). Općenito je preporuka da za dobivanje na preglednosti programa izračunamo tu površinu u funkciji te s pomoću nje izračunamo tražene površine.

```
P1 = a * b;
P2 = e * f;
P3 = k * l;
```

U ovom slučaju zadani program nije bilo teško napisati, ali općenito gledano problem pojavljivanja istog koda te upotrebe istih formula rješavamo tako da kod koji smo više puta pisali u programu napišemo na jednom mjestu te ga potom pozivamo onoliko puta koliko je to potrebno.

Razdvajanjem koda na manje logičke cjeline cjelokupni kod činimo preglednijim, a broj programskih linija smanjujemo te otklanjamo mogućnost pojavljivanja pogreške.

Da bismo znali napisati primjer 1.1 uz pomoć funkcije za računanje površine, prvo moramo definirati opći oblik funkcije.

1.1. Definiranje funkcije

Opći oblik funkcije definiran je na sljedeći način:

```
tip_funkcije ime_funkcije(tip_parametra parametar...) {
    tijelo_funkcije;   → deklaracija varijabli i naredbe
    return rez;        → vrijednost koju vraća funkcija (na mjesto poziva)
}
```

Definicijom funkcije navodimo kojeg će tipa biti rezultat funkcije (`tip_funkcije`), kojim imenom se poziva funkcija, prema potrebi navodimo parametre funkcije (tip i naziv), a pod tijelo funkcije pišemo sve potrebne deklaracije i navodimo potrebne naredbe. Na kraju možemo navesti i naredbu koja vraća vrijednost, tj. rezultat izvođenja funkcije. Ovu naredbu u određenim slučajevima nije potrebno navesti.

Funkcija treba biti deklarirana prije njezina prvog korištenja i zato koristimo prototipove funkcije. Prilikom pisanja programa nije nužno koristiti prototip funkcije, već se funkcija može samo definirati. U tom slučaju njezina definicija ujedno predstavlja i deklaraciju.

1.1.1. Prototip funkcije

Prototip funkcije sastoji se od sljedećih dijelova:

```
tip_funkcije ime_funkcije(tip_parametra parametar...);
```

1.1.2. Tipovi funkcija

Funkcije mogu biti sljedećih tipova: `char`, `short`, `int`, `long`, `float`, `double`, `struct` i `void`. Kada definiramo funkciju, potrebno je navesti i kojeg je tipa. Ako definiramo funkciju koja vraća rezultat tipa `float`, onda za funkciju kažemo da je tipa `float`.

Ako ne navedemo kojeg je tipa funkcija, pretpostavljeni tip funkcije je `int`.

1.1.3. Naredba `return`

Naredbom `return` vraćamo vrijednost funkcije – rezultat (`rez`) u glavni program.

```
tip_funkcije1 ime_funkcije1(parametri_funkcije1) {
    tijelo_funkcije1;
    return rez;
}              → vrijednost funkcije
```

Naredbom `return` moguće je vratiti samo jednu vrijednost, a moguće je da funkcija ne vraća niti jednu vrijednost. Pod "rez" možemo navesti bilo koji tip podatka osim polja. Nakon što se vrijednost funkcije vrati u glavni program, program dalje nastavlja s naredbama. Ako je funkcija tipa `void`, nije potrebno pisati naredbu `return`.

Unutar funkcije moguće je koristiti i više naredbi `return`. Osim što naredbom `return` definiramo vrijednost koju funkcija vraća, njome možemo prekinuti i rad funkcije.

1.1.4. Određivanje imena funkcije

Uz pomoć imena funkcije, funkciju pozivamo u glavni program. Svaka funkcija mora imati svoje ime. Pravila za određivanje imena funkcije su ista kao i za određivanje imena varijabli. U sljedećim primjerima imena vlastitih funkcija pisat ćemo velikim slovima, a imena varijabli, konstanti i polja malim slovima.

1.1.5. Parametri funkcije – prijenos parametara

Parametri mogu biti formalni i stvarni.

U zaglavlju funkcije pišemo **formalne** parametre funkcije te navodimo kojeg su tipa. Možemo ih navesti više pa su zato navedene tri točkice u općem zapisu zaglavlja. Formalni parametri mogu biti osnovnog tipa, pokazivači i strukture, a ne mogu biti tipa polja. U slučaju da funkcija nema formalnih parametara, onda u zagradi pišemo ključnu riječ `void`.

Formalni se parametri povezuju sa **stvarnim** parametrima. Stvarne parametre navodimo prilikom poziva funkcije tako da napišemo njihov naziv, a ako ih ima više, nazive odvojimo zarezima. Stvarni parametri mogu biti osnovni tipovi podataka, pokazivači i strukture, ali ne mogu biti polja. Kada povezujemo stvarne parametre s formalnim parametrima, trebaju si odgovarati po broju, redoslijedu i tipu. U slučaju da si ne odgovaraju po tipu, dolazi do konverzije podataka. U slučaju da funkcija nema formlanih parametara, tada na mjestu stvarnih parametara ne pišemo ništa.

Stvarni parametri se u funkciju mogu prenositi na dva načina:

a) *call by value* (prijenos po vrijednosti) – u funkciju se predaju **kopije vrijednosti** stvarnih parametara.

b) *call by reference* (prijenos po referenci) – u funkciju se predaju **adrese** stvarnih parametara. O ovome će biti više riječi u poglavlju o pokazivačima.

Primjeri nekih prototipova funkcija s pripadajućim tipom i imenom funkcije:

```
int brojnost(int n);
int usporedi(int a, int b);
float opseg(float a, float b, float c);
int prost(int a);
```

tip funkcije

ime funkcije

formalni parametar te kojeg je tipa

Program općenito pišemo na sljedeći način:

- prvo navodimo potrebne biblioteke
- navodimo definicije funkcija
- pišemo glavnu funkciju `main`.

Napomena: Kod prototipa funkcije nije nužno pisati imena formalnih parametara. Primjerice, ispravan prototip bio bi i `int brojnost (int);`.

```
#include <stdio.h>

tip_funkcije1 Ime_funkcije1(tip_parametra_x parametar_x){
    tijelo_funkcije1;
    return rez1;
}

tip_funkcije2 Ime_funkcije2(tip_parametra_y parametar_y){
    tijelo_funkcije2;
    return rez2;
}

int main(void){
    ...
    x = Ime_funkcije1(parametar_a);
    y = Ime_funkcije2(parametar_b);
    return 0;
}
```

formalni parametri

vrijednosti koje funkcije vrate potrebno je pohraniti u varijablu koja je istog tipa kao i sama funkcija

stvarni parametri

Glavna funkcija `main` je tipa `int`, a njezin prototip smo naučili još u 2. razredu. Dogovorno glavna funkcija `main` kao povratnu vrijednost naredbom `return` vraća cijeli broj 0.

Napomena: Dio koda kojim su u varijable `x` i `y` pridružene vrijednosti koje su vratile funkcije `Ime_funkcije1` i `Ime_funkcije2` ne vrijedi za funkcije tipa `void`. Poziv funkcije tipa `void` objašnjen je u cjelini 1.3.

Program možemo pisati i tako da prvo napišemo samo **prototipove funkcija**. Tada prije glavne funkcije `main` navodimo samo prototipove funkcija, a iza glavne funkcije `main` navodimo definicije funkcije sa zaglavljem i tijelom funkcije.

```
#include <stdio.h>

Ime_funkcije1(tip_parametra_x parametar_x);
Ime_funkcije2(tip_parametra_y parametar_y);

int main(void){
    ...
    x = Ime_funkcije1(parametar_a);
    y = Ime_funkcije2(parametar_b);
    return 0;
}

tip_funkcije1 Ime_funkcije1(tip_parametra_x parametar_x){
    tijelo_funkcije1;
    return rez1;
}

tip_funkcije2 Ime_funkcije2(tip_parametra_y parametar_y){
    tijelo_funkcije2;
    return rez2;
}
```

prototipovi funkcija

Prema prototipu funkcije prevodilac može provjeriti jesu li ispravno definirane funkcije.

1.1.6. Tijelo funkcije

U tijelu funkcije pišemo sve potrebne lokalne varijable i naredbe koje su nam potrebne za izvedbu željene funkcije. **Lokalne varijable** su sve varijable deklarirane unutar tijela funkcije te se koriste samo unutar bloka funkcije unutar kojeg su deklarirane, pa stoga njihove vrijednosti nisu vidljive u ostalim dijelovima programa.

Za razliku od lokalnih varijabli postoje i **globalne varijable**. Globalne varijable deklariramo na samom početku, prije glavne funkcije `main` te svih ostalih funkcija koje su navedene u programu (globalne se varijable deklariraju izvan svih funkcija). Na taj način je globalna varijabla vidljiva u svim funkcijama te svaka funkcija može promijeniti vrijednost globalne varijable. Ovako možemo povezati funkcije bez upotrebe formalnih parametara.