PROGRAMSKI JEZIK

C

Lazar Ljubenović

Algoritmi i programiranje

Elektronski fakultet, Niš

Glava 1

Uvod u C

1.1 Kratka istorija C-a i osnovni koncepti

C je programski jezik opšte namene tesno povezan sa UNIX operativnim sistemom uz koji je i razvijan. Prvu verziju razvio je Denis Riči (*Dennis MacAlistair Ritchie*) u *Bell Laboratories* u Sjedinjenim Američkim Državama. C je portabilan jezik, što znači da se ne vezuje ni za operativni sistem ni za harver računara. To je strukturni programski jezik, i programi napisani u C-u su efikasni, uglavnom manji i brži od programa napisanih u drugim programskim jezicima. Jako je fleksibilan, i nalazi široku primenu u kreiranju video-igara, komercijalnih sistema, veštačkoj inteligenciji (ekspertni sistemi, robotika); koristi se i za upravljanje procesima.

Za razliku od nekih drugih jezika, u C-u ne postoje operacije za direknu manipulaciju sa složenim objektima kao što su stringovi, liste, polja, itd. Ne postoje ni operacije za manipulaciju sa celim poljem ili stringom. Mada na prvi pogled deluje začuđujuće, ne postoji ni direktna podrška za ulaz i izlaz (ne postoje `read` i `write` naredbe). Slično, nije ugrađen pristup fajlovima, tj. ne postoje metode pristupa. U sledećem poglavlju će biti reči o tome šta je potrebno uraditi da bi ove mogućnosti bile dostupne i u progamima napisanim u C-u.

C nije strogo tipiziran jezik – ne postoji automatska konverzija neusaglašenih tipova podataka. Ipak, to je relazivno mali jezik, opisan kratko i uči se brzo, mada ima reputaciju da je težak za savladavanje. Većina koncepata u C-u se nalazi i u `Pascal`-u.

Azbuku programskog jezika C čine velika i mala slova engleske abecede, dekadne cifre i specijalni znaci. **Reč** je niz znakova koji predstavljaju elementarnu logičku celinu u jednom programskom jeziku.

Vrste reči u C-u su:

- identifikatori,
- ključne reči,
- separatori,
- konstante,

- literali (konstantni znakovni nizovi) i
- operatori.

Osim reči koje imaju svoje određeno značenje u programu, postoje i delovi programa koje kompajler jednostavno ignoriše. Ti delovi programa služe programeru da pojasne kôd – oni se nazivaju **komentari**. U C-u se komentari mogu obeležiti na dva različita načina:

- između simbola `/*` i `*/` i
- počinju simbolom `//` i protežu se do kraja programske linije.

Primer 1

```
1 /* ovo je komentar */
2 // i ovo je komentar
3 /* ovo je komentar */ ovo = nije + komentar;
4 izraz = izraz + /* komentar + */ izraz;
```

Svaka naredba se završava sa tačkom-zarez (`;`). Mada nije obavezno, dobra je praksa da se svaka naredba piše u zasebnoj liniji programskog kôda.

1.2 Identifikatori (simbolička imena)

Identifikator predstavlja niz slova, cifara i znaka "_" (donja crta), u kojem prvi znak ne sme biti cifra. Identifikatori služe da imenuju promenljive, simboličke konstante, funkcije, korisničke tipove podataka i labele u programu.

Identifikator može biti proizvoljne dužine, ali kompajler tretira:

- 31 znak za interna imena (koja se definišu i koriste samo u jednom fajlu),
- 6 znakova za imena koja se koriste u više fajlova.

1.3 Ključne reči

Ključne reči (*keywords*) su reči čije je značenje unapred definisano i ne mogu se koristiti kao identifikatori. Sve ključne reči programskog jezika C su prikazane u nastavku, sortirane po alfabetu.

`auto`	`double`	`int`
`break`	`else`	`long`
`case`	`enum`	`register`
`char`	`extern`	`return`
`const`	`float`	`short`
`continue`	`for`	`signed`
`default`	`goto`	`sizeof`
`do`	`if`	`static`

`struct`	`union`	`volatile`
`switch`	`unsigned`	
`typedef`	`void`	`while`

1.4 Separatori

Separatori su posebni znaci koji imaju određeno unapred definisano sintaksno i semantičko značenje u programu, a ne označavaju nikakvu operaciju. U ovu grupu reči spadaju `{`, `}`, `(`, `)`, `;`, `:`…

1.5 Tipovi podataka, konstante i literali

U programskom jeziku `C` definisani su **samo numerički tipovi podataka**. Ukupno postoji samo **četiri tipa podataka**:

- `int` – celobrojni podatak za čije se predstavljanje koriste 16 ili 32 bita (zavisno od hardvera računara);
- `char` – mali celobrojni podatak (dužine jedan bajt) koji može da primi i kôd jednog znaka pa se koristi i za predstavljanje znakovnih podataka;
- `float` – realni podatak u jednostrukoj tačnosti (32 bita);
- `double` – realni podatak u dvostrukoj tačnosti.

Varijante navedenih tipova podataka se dobijaju dodavanjem ključnih reči `short`, `long`, `signed` ili `unisgned` ispred oznake tipa.

- `short` – memorijski prostor za predstavljanje podatka se smanjuje na polovinu uobičajene vrednosti;
- `long` – memorijski prostor za predstavljanje podatka se udvostručava;
- `unsigned` – naglašava činjenicu da se radi o neoznačenom podatku;
- `signed` – naglašava da se radi o označenom podatku (po difoltu su svi podaci označeni).

Logičke vrednosti se predstavljaju takođe numeričkim podacima, pri čemu nula odgovara logičkoj vrednosti `false`, a svaka vrednost različita od nule logičkoj vrednosti `true`.

Stringovi (znakovni nizovi) se predstavljaju nizom uzastopnih podataka tipa `char` koji se završava simbolom `\0`.

Celobrojne **konstante** mogu biti predstavljene u dekadnom, heksadekadnom ili oktalnom brojevnom sistemu.

> Decimalna celobrojna konstanta je niz dekadnih cifara u kojem prva cifra ne sme biti `0`.
>
> Oktalna celobrojna konstanta je niz otalnih cifara u kojem je prva cifra `0`.
>
> Heksadekadna celobrojna konstanta je niz heksadekadnih cifara koji počinje prefiksom `0x` ili `0X` u kojem heksadekadne cifre mogu biti mala ili velika slova.

Po difoltu, celobrojne konstante su tipa `int`. Dodavanjem sufiksa `l` ili `L` naglašava se da se radi o konstanti tipa `long int`. Slično se dodavnjem sufiksa `u` ili `U` obeležavaju konstante tipa `unsigned int`. Ova dva sufiksa se mogu kombinovati.

Primer 1

`1234` – dekadna konstanta tipa `int`
`9876543ul` – dekadna konstanta tipa `unsigned long int`
`-765` – (negativna) dekadna konastanta tipa `int`
`0498` – oktalna konstanta tipa `int`
`-01` – (negativna) oktalna konstanta tipa `int`
`0x1a2` – heksadekadna konstanta tipa `int`
`0x1a2u` – heksadekadna konstanta tipa `unsigned int`

Što se realnih konstanti tiče, one se uvek pišu u dekadnom brojevnom sistemu i mogu biti predstavljene u fiksnom zarezu ili u eksponencijalnom obliku.

Zapis u fiksnom zarezu obavezno sadrži decimalnu tačku i to tačno tamo gde je njeno stvarno mesto u broju (na primer, `3.14`, `2.0`, `2.`, `0.5`, `.5`). Međutim, ovaj zapis je nepogodan za veoma velike ili veoma male brojne vrednosti, koje nisu retkost u raznim naučnim oblastima.

U tim slučajevima se koristi eksponencijalni zapis. To je zapis oblika `mEe`, gde `m` predstavlja mantisu broja, a `e` eksponent (na primer, `2.23e2`, `7e-23`, `8.55E-12`, `9E-9`).

Po difoltu, realne konstane su tipa `double`. Dodavanjem sufiksa `f` ili `F` se dobijaju konstante tipa `float`, a dodavanjem sufiksa `l` ili `L` – konstante tipa `long double`.

Znakovna konstanta je ceo broj čija je vrednost jednaka kôdu znaka navedenog između jednostrukih navodnika (apostrofa). Na primer, u ASCII skupu znakova, znakovna konstanta `'A'` ima vrednost 65, a znakovna konstanta `'5'` ima vrednost 53.

ASCII tabela sadrži izvestan broj upravljačkih simbola (simboli koji ne ostavljaju trag na ekranu ili štampaču). Za takve znakove su u C-u uvedene određene simboličke oznake, prikazane u tabeli 1.5.

Literal je sažeti način zapisivanja niza znakovnih konstanti. Na primer, `"Ovo je literal"`. C prevodilac svaki znak prevodi u jedan ceo broj tipa `char` (ASCII ekvivalent) i na kraju dodaje još jedan znak sa vrednošću `'\0'`, takozvani *null terminated symbol*.

Primer 2 Literal `"Zdravo"` će u memoriji biti zapisan kao niz sledećih karaktera: `'Z'`, `'d'`, `'r'`, `'a'`, `'v'`, `'o'` i `'\0'`.

kotrolni znaci	simboli u C-u
prelaz u sledeći red	`\n`
horizontalna tabulacija	`\t`
vertikalna tabulacija	`\v`
znak za vraćanje (*backspace*)	`\b`
nova strana (*form feed*)	`\f`
obrnuta kosa crta (*backslash*)	`\\`
znak pitanja	`\?`
apostrof	`\'`
znaci navoda (dvostruki)	`\"`

Tabela 1.1: Simbočke oznake za upravljačke simbole u C-u.

Primer 3 Ispravno korišćenje literala:

`"Elektronski fakultet"`

`"a"` – literal koji sadrži samo jedno slovo (i `\0`);

`""` – literal koji ne sadrži nijedan znak (u memoriji samo `'\0'`);

`" "` .. literal koji sadrži samo blanko znak (i `'\0'`);

`"/* nije komentar */"` – simboli `'/'` i `'*'` se tretiraju kao i svaki drugi simbol;

`"\"navodnici\""` – sadrži (duple) navodnike kao specijalne simbole `'\"'` koji će se na ekranu i štampaču prikazati kao navonici (`"`).

Primer 4 Neispravno korišćenje literala:

`/*"nije literal"*/` – sve između `/*` i `*/` je komentar;

`'nije literal'` – literal **mora** biti pod (duplim) navodnicima, ne pod apostrofima;

`'a'` – ispravno kao karakter (tip `char`), ali ni to nije literal.

1.6 Operatori

Operatori su simboli koji označavaju određenu operaciju i koji povezuju jedan ili više operanada u izraz. Podela operanada se može izvršiti po broju operanada i po funkcionalnosti.

U zavisnosti od broja operanada, operatori u C-u se dele na:

- unarne,
- binarne i
- ternarne.

Grupe operatora u odnosu na funkcionalnost su:

- operatori za pristup članovima polja i struktura,
- operator za poziv funkcije,
- aritmetički operatori,
- relacioni operatori,
- logički operatori,
- operatori za rad sa bitovima,
- operatori dodeljivanja vrednosti,
- operator grananja,
- `sizeof` operator,
- `cast` operator i
- operatori refernciranja i deferenciranja.

1.6.1 Aritmetički operatori

Aritmetički operatori imaju numeričke operande i rezultati su, takođe, numeričkog tipa. Po prioritetu (od najvišeg ka najnižem), to su:

- unarni operatori `+` i `-`,
- operatori inkrementiranja (`++`) i dekrementiranja (`--`),
- operatori `*`, `/`, `%` i
- binarni operatori `+` i `-`.

Unarni operator `-` menja znak operanda koji sledi. Ako je operand tipa `unsigned`, rezultat se računa oduzimanjem operanda od 2^n, gde je n broj bitova u binarnoj predstavi tipa rezultata.

Unarni operator `+` zadržava znak operanda koji sledi, što znači da je on praktično operator bez dejstva.

Operatori **inkrementiranja** i **dekrementiranja** su unarni operatori i oni mogu biti prefiksni i postfiksni. Oni vrše povećanje (odnosno smanjenje) vrednosti svog operanda za 1. Rezultat izraza sa prefiksnim operatorima `++` (i `--`) je nova vrednost operanda, dok je rezultat izraza sa postfiksnim operatorom stara vrednost operanda.

Drugim rečima, ukoliko se vrednost izraza `++a` (odnosno `--a`) koristi u nekom širem izrazu, prvo će se izvršiti promena vrednosti promenljive `a` i ta promenjena vrednost će se iskoristiti za izračunavanje vrednosti šireg izraza.

Ukoliko se vrednost izraza `a++` (odnosno `a--`) koristi u nekom širem izrazu, najpre će se izvršiti izrčunavanje vrednosti šireg izraza sa starom vrednošću promenljive `a`, a nakon toga će se izvršiti promena vrednosti promenljive `a`.

Primer 1 U sledećem isečku koda je ilustrovana primena prefiksne verzije operatora za inkrementiranje.

```
a = 3; b = 5;
c = (++a) + (++b);
```

Dakle, pre nego što se nađe zbir promenljivih `a` i `b`, obe će se uvećati za jedan, pa će `c` biti jednako $4 + 6 = 10$ nakon izvršenja druge linije kôda, dok će `a` i `b` imati vrednosti 4 i 6, respektivno.

Primer 2

```
a = 4; b = 6;
c = (a++) + (b++);
```

U ovom slučaju se najpre izračuna zbir, tako da će `c` imati vrednost $4+6 = 10$, a nakon toga će se `a` i `b` uvećati za jedan, pa će biti jednake 5 i 7, respektivno.

1.6.2 Operatori množenja, deljenja i ostatka pri deljenju

Operator `*` označava množenje.

Operator `/` označava deljenje. Ukoliko se primenjuje nad celobrojnim podacima, predstavlja takozvano celobrojno deljenje, odnosno u rezultatu se vrši odbacivanje cifara iza decimalne tačke.

Primer 3 U prvoj liniji koda su deklarisane tri realne promenljive, `a`, `b` i `c`.

```
float a = 10.0, b = 5.0, c = 1.0;
d = a / b;
e = c / a;
f = c / b;
```

Kako su svi podaci realni, deljenje se obavlja na klasičan način, kao u metematici. Kako je $10 \div 5 = 2$, $1 \div 10 = 0.1$, $1 \div 5 = 0.2$, rezultati su, redom, `2.0`, `0.1`, `0.2`.

Primer 4

```
int a = 10, b = 5, c = 3, d = 2;
x = a / b;
y = a / c;
z = c / d;
p = b / a;
q = d / c;
r = ( c / a ) * a;
```

Jasno je da je vrednost promenljive `x` nakon izvršenja druge linije koda jednaka `2`.

Kod treće i čevrte linije, prilikom računanja navedenih izraza, primenjuje se celobrojno deljenje. Kako je $10 \div 3 = 3.333\ldots$, rezultat ove operacije (i vrednosti promenljive `y`) je `3`. Slično je $3 \div 2 = 1.5$, pa je rezultat `1`.

Kako je $5 \div 10 = 0.5$, rezultat operacije u petoj liniji koda je `0`. Isti rezultat se dobija i u šestoj liniji koda.

Sedma linija koda bi mogla da zavara na prvi pogled, ukoliko se ne obrati pažnja na to da `c/a` predstavlja celobrojno deljenje. Da to nije slučaj, očigledno bi bilo

$$r = \frac{c}{a} \cdot a = \frac{c}{\not{a}} \cdot \not{a} = c.$$

Međutim, celobrojno deljenje će odbaciti sve cifre iza decimalnog zareza. Zato je rezultat razmatrane operacije

$$r = \left\lfloor \frac{3}{10} \right\rfloor \cdot 10 = 0 \cdot 10 = 0,$$

odnosno u `r` će biti smešten broj `0`.

Operator `%` (moduo) označava ostatak deljenja i može se primeniti samo nad celobrojnim podacima.

Primer 5 Kako se pri deljenju 5 sa 2 dobija ostatak 1, rezultat primene operacije `5%2` je `1`.

Primer 6

```
1 a = 7 % 3;
2 b = 9 % 2;
3 c = 3 % 9;
4 d = 14 % 4;
5 e = 14 - (14 / 4);
```

Pri deljenju 7 sa 3 se dobija ostatak 1, pa je rezultat u prvoj liniji koda jednak `1`. Matematički konciznije zapisano, isti rezultat sledi iz jednakosti $7 = 3 \cdot 2 + 1$. Slično je i u drugoj liniji koda ($9 = 2 \cdot 4 + 1$).

Kako je $3 = 9 \cdot 0 + 3$, `c` će imati vrednost `3` nakon izvršenja treće linije koda. U četvrtoj liniji je jasno da je rezultat primene operacije moduo jednak `2`.

U poslednjoj liniji koda je na primeru pokazan alternativan metod izračunavanja ostatka pri deljenju bez korišćenja operacije `%`.

Napomena Treba voditi o računa o primeni operatora `%` nad negativnim brojevima. Programski jezik `C` za rezultat primene ove operacije uzima znak deljenika (levi operand). Na primer, `(-11)%3` vraća rezultat `-2`.

1.6.3 Binarni operatori + i -

Kao što je i očekivano, ovi operatori su identični binarnim operatorima u matematici – `+` označava sabiranje a `-` označava oduzimanje.

1.6.4 Relacioni operatori

Operatori koji se koriste za poređenje vrednosti operanada nazivaju se relacioni operatori. Operandi u ovom slučaju mogu biti bilo kog numeričkog tipa ili pokazivači, a rezultat je uvek logičkog tipa. Međutim, kao što je već rečeno, logički tip u C-u ne postoji, tako da je rezultat relacionih operatora `1` u slučaju da je poređenje tačno, a suprotnom je rezultat `0`.

Ovoj grupi operatora pripadaju

- `<` – manje,
- `<=` – manje ili jednako,
- `>` – veće,
- `>=` – veće ili jednako,
- `==` – jednako i
- `!=` – različito.

Prioritet prva četiri operatora je viši od prioriteta poslednja dva.

Primer 7

```
a = ( 5 > 2 );
b = ( 0 < -4 );
c = ( (1+6) >= (5*7) );
d = ( a < b );
e = ( 'a' < 'b' );
f = ( 5 <= 5 );
g = ( b == 0 );
```

Zbog toga što je $5 > 2$, vrednost izraza `5>2` je `1`. Slično, izraz u drugoj liniji koda nije istinit, pa je rezultat koji vraća izraz `0<-4` jednak `0`.

U trećoj liniji koda se najpre računaju izrazi `1+6` i `5*7`, a onda se dobijene vrednosti upoređuju. Obzirom na to da je izraz $7 \geq 35$ netačan, rezultat operacije će biti `0`.

Četvrta linija koda ilustruje primenu operatora `<` nad promenljivama. Na osnovu rezultata iz prve i druge linije koda, vidimo da se izraz `a<b` svodi na `1<0`. Kako ovo nije tačno, rezultat ove operacije će biti `0`.

U petoj liniji koda se porede dva karaktera. Poređenje karaktera se svodi na poređenje vrednosti koje ti karakteri imaju u ASCII tabeli. Nije potrebno znati ASCII tabelu napamet da bi se našao rezultat ove operacije. Naime, u tabeli su mala i velika slova poređana istim redom kao i u abecedi. Kako je slovo *a* ispred slova *b*, rezultat razmatrane operacije je `1`.

Obzirom na to da je svaki broj manji ili jednak od samog sebe, posle izvršenja šeste linije koda, u promenljivoj `f` će biti smešten rezultat `1`.

Poslednja linija kôda poredi vrednost dobijenu u drugoj liniji koda sa nulom. Kako je izraz $0 = 0$ tačan, rezultat ove operacije je `1`. Treba primetiti da je na ovaj način postavljen upit "da li je izraz `b` netačan?".

1.6.5 Logički operatori

U programskom jeziku `C`, operandi u logičkim operacijama mogu biti numeričkog tipa ili tipa pokazivača. Obzirom da se ovi operatori prirodno (u matematici) primenjuju nad logičkim podacima, pri izvršenju ovih operacija svaka vrednost različita od nule se tretira kao logička vrednost tačno (*true*), a samo nula označava netačno (*false*).

U ovu grupu operatora spadaju:

- `!` – negacija,
- `&&` – konjukcija (logičko "i") i
- `||` – disjunkcija (logičko "ili").

Operatori su poređani po prioritetu, tako da poslednji ima najviši prioritet.

Rezultat operacije je `1` ako je rezultat logičke operacije `true`, u suprotnom je rezultat `0`.

Primer 8

```
a = 1; b = 0;
e = !a;
f = ( b != 0 );
g = ( a && b );
h = ( a || (!b) );
i = !c;
j = ( b || 6 );
k = ( !a && (e+f+g+h) );
```

1.6.6 Operatori za rad sa bitovima

Operatori koji rade nad bitovima unutar celobrojnih podataka definisani su u asemblerskim jezicima, a od viših programskih jezika, jedino ih `C` podržava.

Ovoj grupi operatora pripadaju:

- `~` – nepotpuni (jedinični) komplement (komplementira se svaka cifra ponaosob),
- `&` – pokomponentno logičko "i",
- `^` – pokomponentno logičko "isključivo ili",
- `|` – pokomponentno logičko "ili",

- `<<` – pomeranje ulevo (vrši pomeranje bitova levog operanda za onoliko mesta ulevo kolika je vrednost desnog operanda) i
- `>>` – pomeranje udesno (vrši pomeranje bitova levog operanda za onoliko mesta udesno kolika je vrednost desnog operanda).

Primer 9

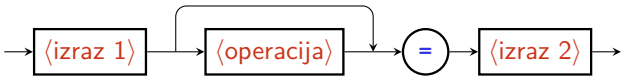
```
int a = 0723; // a = 0000000111010011
int b = ~a;   // b = 1111111000101100
a = a << 4    // a = 0001110100110000
a = a >> 3    // a = 0000001110100110
a = a | 071;  // 071=0000000000111001
              // a = 0000001110111111
```

1.6.7 Operator dodele

U prethodnim odeljcima je prećutno bio korišćen operator `=` prilikom dodele vrednosti u listinzima. Naime, u pitanju je binarni operator koji smešta vrednost desnog operanda na lokaciju čije se ime nalazi sa leve strane operatora. Jasno je da zbog ovoga levi operator mora biti ime memorijske lokacije (ne može se konstantni dodelite vrednost, dok se memorijskoj lokaciji može dodeliti neka konstanta).

Postoje dve grupe ovakvog operatora:

- elementarni operator dodeljivanja (bez ⟨operacija⟩) i
- složeni operator dodeljivanja (sa ⟨operacija⟩).



⟨operacija⟩ može biti jedna od operacija `+`, `-`, `*`, `/`, `%`, `<<`, `>>`, `&`, `|` i `^`.

Izraz `a` ⟨op⟩`= b` se može napisati u ekvivalentnom obliku `a = a` ⟨op⟩ `b`.

Primer 10

```
a += 3;      // isto kao da smo pisali a = a + 3;
b -= a;      // isto kao da smo pisali b = b - a;
p *= a - 5; // isto kao da smo pisali p = p * (a-5)
```

1.6.8 Operator grananja

Ovo je jedini ternarni operator u C-u. Opšti oblik izraza kreiranog pomoću ovog operatora je sledeći.

Najpre se izračunava vrednost izraza ⟨izraz 1⟩. Ako je vrednost ovog izraza različita od nule (`true`), izračunava se vrednost izraza ⟨izraz 2⟩ i njegova vrednost je rezultat operacije. Ako je vrednost izraza ⟨izraz 1⟩ jednaka nuli (`false`), izračunava se vrednost izraza ⟨izraz 3⟩ i njegova vrednost je rezultat operacije.

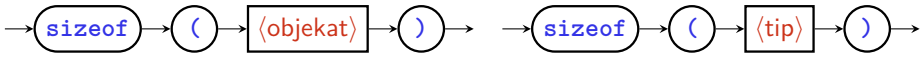
Primer 11

```
int max = ( a > b ) ? a : b;   // ako je a>b, max=a, inace max=b
int abs = ( x >= 0 ) ? x : -x; // ako je x>=0, abs=x, inace abs=-
    x
```

1.6.9 `sizeof` operator

Svaka promenljiva ili konstanta zauzima određeni deo memorijskog prostora. Veličina tog memorijskog prostora zavisi od tipa promenljive (ili konstante), tj. od internog načina predstavljanja pojedinih tipova u memoriji. Operator `sizeof` služi toj svrsi – pomoću njega se može izračunati broj bajtova koji neki podatak ili tip podatka zauzima u memoriji.

To je unarni operator i može se koristiti na dva načina.

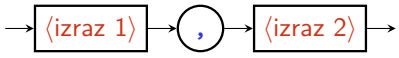


Primer 12

```
int a = sizeof(short int);
int b = sizeof(a);
```

1.6.10 `comma` operator

U pitanju je binarni operator koji služi za formalno spajanje više izraza u jedan. Levi i desni operand operatora `,` su izrazi (⟨izraz 1⟩ i ⟨izraz 2⟩), pri čemu je redosled njihovog izvršavanja sleva nadesno.



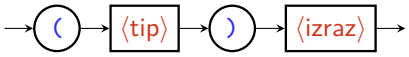
Primenu najčešće nalazi kod poziva funkcija i u okviru `for` petlje.

Primer 13 Inkrementiranje broja `a`, udvostručavanje vrednosti broja `b` i nalaženje zbira ove dve vrednosti i njeno smeštanje u `c` se pomoću operatora `,` može spojiti u jedan izraz na način prikazan u donjem listingu.

```
a++, b *= 2, c = a + b;
```

1.6.11 `cast` operator

Koristi se za eksplicitno pretvaranje promenljivih (ili izraza) jednog tipa u neki drugi tip. Opšti oblik ovog unarnog operatora je prikazan na slecećoj slici,



gde je ⟨tip⟩ – neki od tipova, a ⟨izraz⟩– promenljiva ili neki složeniji izraz čiji tip treba promeniti. Primenjuje se najčešće kod matematičkih funkcija.

Primer 14 Funkcija `sin`, koja se koristi za računanje sinusa, kao parametar uzima promenljivu tipa `double`. Ukoliko je ugao čiji se sinus traži sačuvan u promenljivoj tipa `float`, prilikom poziva funkcije `sin` mora se iskoristiti `cast` operator, kao što je učinjeno u sledećem listingu.

```
float alfa = 3.1415;
double beta = 3.1415;
double x = sin((double)alfa); // argument mora biti tipa double
double y = sin(beta);
```

1.6.12 Prioritet operatora

Kao i u matematici, prioritet operatora određuje redosled izvođenja operacija kada u izrazu figuriše veći broj operatora. U izrazima se uvek najpre izvršavaju operatori najvišeg prioriteta, pa zatim oni nižih. Ovaj redosled je moguće promeniti korišćenjem zagrada (one imaju najviši prioritet).

U donjoj listi se nalaze operatori, poređani po prioritetu (počev od operatora najvišeg prioriteta, prema onim nižih).

1. `(`, `)`, `[`, `]`, `.`, `->`
2. unarni `-`, `~`, `&`, `|`, `++`, `--`, `sizeof`
3. `*`, `/`, `%`
4. `+`, `-`
5. `<<`, `>>`
6. `<`, `>`, `<=`, `>=`
7. `==`, `!=`
8. `&`

9. ^
10. |
11. &&
12. ||
13. ?:
14. =, <op>=
15. ,

Primer 15

```
int a = 1+2*3 // isto kao 1+(2*3)
int b = 1+2-3+4-5 // isto kao (((1+2)-3)+4)-5 (sleva nadesno)
```

Primer 16 U sledećem listingu su prve linije koda ekvivalentne poslednjoj (četvrtoj).

```
b = 2;
c = 3;
a = b + c;
a = ( b = 2 ) + ( c = 3 );
```

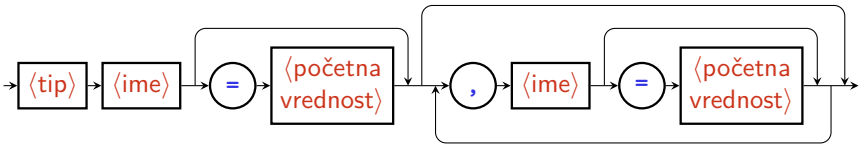
Glava 2

Struktura C programa

C program predstavlja skup definicija promenljivih, simboličkih konstantnti, tipova i funkcija. Obavezan deo svakog C programa je funkcija main. Svaki C program sadrži tačno jednu definiciju funkcije main, koju poziva operativni sistem u trenutku poziva programa.

2.1 Definisanje promenljivih, simboličkih konstanti i tipova

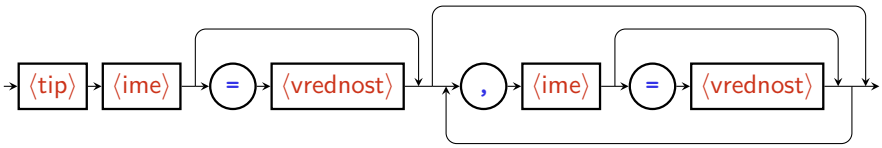
Promenljive se definišu na sledeći način:



Primer 1

```
1 char c;
2 int x = 5;
3 float a, b;
4 int p = 1, q = 2, r = 3;
5 double n = 4.29348, m;
6 int i, j = 0;
```

Konstante se definišu na sledeći način:



Primer 2 U prvoj liniji prikazanog isečka koda su definisane dve konstante, obeležene slovima `a` i `b`. Obe konstante su tipa `int` i imaju vrednosti 3 i 5, respektivno.

```
1 const int a = 3, b = 5;
2 const float pi = 3.14;
```

U drugoj liniji je definisana konstanta `pi` tipa `float` kojoj je dodeljena vrednost `3.14`.

U gornjem isečku koda bi u ostatku koda trebalo koristiti simboličko ime `pi` svaki put kada je potrebno iskoristiti matematičku konstantu π. Postoje dve bitne prednosti zbog čega je ovo dobra praksa.

Pre svega, kôd je mnogo jasniji za čitanje. Pisanjem `3.14` neće uvek biti jasno na šta se taj broj odnosi, ali ukoliko se uvek bude pisalo `pi` biće jasno o čemu se radi (na primer, za izračunavanje površine kruga poluprečnika `r`, treba pisati `pi*r*r` umesto `3.14*r*r`).

Druga velika prednost jeste to što je dovoljno samo na jednom mestu promeniti vrednost `3.14`, i ta promena će se odraziti na ceo kôd. Na primer, nakon pisanja celog programa u kome se intenzivno koristi broj π, programer dolazi do zaključka da mu je bila potrebna veća tačnost – dva decimalna mesta nisu bila dovoljna. Ukoliko je korišćena konstanta `pi`, biće dovoljno u definiciji konstante brojnu vrednost `3.14` promeniti na, recimo, `3.1415`. Ukoliko je na mestima gde je potrebno iskoristiti broj π bila pisana brojna vrednost `3.14`, biće potrebno na svakom mestu gde se ona pojavljuje ispraviti grešku, odnosno dodati dve decimale.

Tipovi se definišu na sledeći način:

Primer 3

```
1 typedef float duzina;
2 typedef char slovo;
```

2.2 Funkcija `main()`

Minimalna definicija funkcije `main` opisana je na sledeći način.

Telo funkcije `main` sadrži definicje promenljivih, konstanti i tipova koji se koriste lokalno (unutar funkcije `main`) i skup izvršnih naredbi (naredbi kojima se vrši obrada podataka). Izvršne naredbe u C-u se dele na elementarne naredbe i strukturne naredbe.

Elementarne izvršne naredbe su naredbe kojima se vrši neka elementarna obrada podatka. Dobijaju se tako što se na kraj jednog izraza doda znak tačka-zarez (`;`).

Primer 1 Sledeći listing ilustruje tri naredbe.

```
1 a+=b*5;
2 Max(a,b);
3 i++;
```

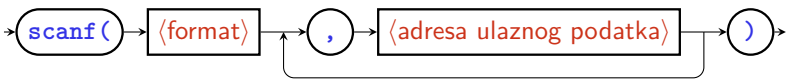
2.2.1 Naredbe za ulaz i izlaz

Kao što je na početku rečeno, u C-u ne postoje posebne naredbe za uzlaz i izlaz podataka. Međutim, postoji skup bibliotečkih funkcija za učitavanje podataka u program i za upis rezultata na standarni izlaz ili u datotetku (`scanf` i `printf`).

Deklaracije funkcija za upravljanje ulaznim i izlaznim resursima u C-u nalaze se u fajlu `stdio.h`. Pre korišćenja funkcija za ulaz i izlaz podataka u fajl sa izvornim kodom, treba uključiti fajl `string.h`. Za uključivanje jednog fajla u drugi koristi se preprocesorska direktiva `include`.

Zato, kada je u programu potrebno koristiti funkcije `scanf` i `printf`, treba navesti `#include <stdio.h>`.

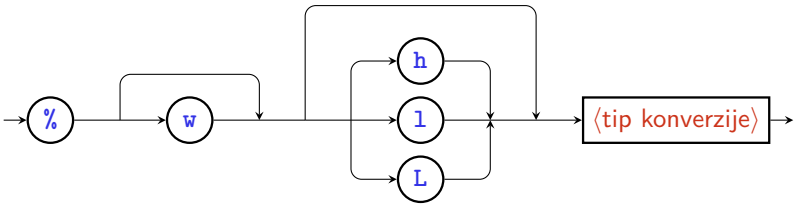
Funkcija za učitavanje podataka sa standardnog ulaza `scanf()` poziva se na sledeći način:



gde je ⟨format⟩ znakovni niz koji predstavlja definiciju konverzija koje treba izvršiti pri unosu podataka. Svi ulazni podaci se učitavaju kao niz ASCII kodova. Format definiše kako će se taj znakovni niz konvertovati u podatke odgovarajućeg tipa. Svaka pojedinačna konverzija u ovom znakovnom nizu počinje znakom `%` i završava se slovom koje označava vrstu konverzije.

Da bi se pristupilo adresi ulaznog podatka, potrebno je koristiti adresni operator `&`. Adresa promenljive `a` bi bila `&a`.

Format pojedinačne ulazne konverzije je sledeći:



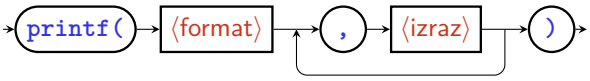
Tip konverzije može biti:

- `d` – dekadna konverzija (rezultat je tipa `int`);
- `u` – dekadna konverzija neoznačenih brojeva (rezultat je tipa `unsigned int`);
- `o` – oktalna konverzija (rezultat je tipa `int`);
- `x` – heksadekadna konverzija (rezultat je tipa `int`);
- `i` – dekadna, oktalna ili heksadekadna konverzija zavisno od zapisa pročitanog broja (rezultat je tipa `int`);
- `f` – konverzija brojeva zapisanih u fiksnom zarezu u podatke tipa `float`;
- `e` – konverzija brojeva zapisanih u eksponencijalnom obliku u podatke tipa `float`;
- `g` – konverzija koja označava da se ulazni znakovni niz konvertuje u podatak tipa `float`; bez obzira da li je na ulazu broj zapisan u fiksnom zarzu ili u eksponencijalnom obliku;
- `c` – znakovna konverzija (rezultat je tipa `char`);
- `s` – konverzija u znakovni niz (niz znakova između dva blanko znaka što znači da učitani niz ne sadržio blanko znake; iza pročitanih slova se dodaje `'\0'`.

Postoji i četiri prefiksa, koji nisu obavezni. To su:

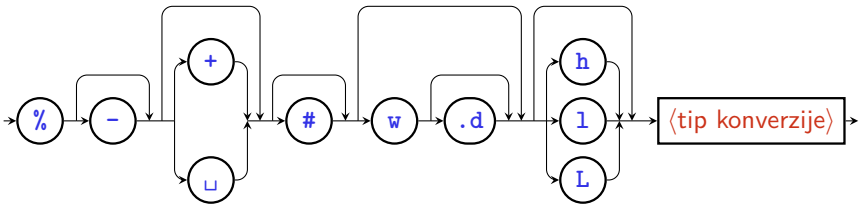
- `w` – ceo broj koji označava dozvoljeni broj znakova za podataka u ulaznom nizu;
- `h` – prefiks koji se koristi ispred celobrojnih konverzija i označava da se vrši konverzija ulaznog znakovnog niza u kratki ceo broj (rezultat je tipa `short int`);
- `l` – prefiks koji se koristi ispred celobrojnih i realnih tipova konverzija i označava da se vrši konverzija ulaznog znakovnog niza u dugi ceo broj, ili u realni broj u dvostrukoj tačnosti (rezultat je tipa `long int` ili `double`);
- `L` – prefiks koji se koristi samo ispred realnih tipova konverzija koji označava da se vrši konvertovanje ulaznog znakovnog niza u podatak tipa `long double`.

Za prikaz rezultata na standardni izlaz koristi se funkcija `printf()`. Poziva se na sledeći način:



gde je ⟨format⟩ znakovni niz koji predstavlja definiciju konverzija koje treba izvršiti pri prikazu rezultata. U ovom slučaju, vrši se konvertovanje memorijskog zapisa u izlazne znakovne nizove. Osim pojedinačnih konverzija, format može sadržati i dodatne komentare koji će pojasniti izlazne rezultate ili uticati na preglednost prikazanih rezultata (recimo, prelazak u novi red prilikom štampanja rezultata na standarndi izlaz).

Format pojedinačne izlazne konverzije je sledeći:



gde prefiksi imaju sledeća značenja:

- `-` – označava da će se, ukoliko je dužina podatka manja od širine polja koje je predviđeno za prikaz podatka, vršiti poravnanje uz levu ivicu (po difoltu se vrši desno poravnanje),
- `+` – može da stoji isključivo ispred numeričkih konverzija i označava da će i ispred pozitivnih brojeva stajati njihov znak (`+`),
- ␣ (blanko) – stoji takođe ispred numeričkih konverzija i označava da će se pri prikazu pozitivnih borjeva, na poziciji predviđenoj za znak broja pojaviti blanko znak,
- `#` – stoji ispred celobrojnih konverzija (`o` ili `x`) i označava da će se uz vrednost broja prikazati i oznaka brojnog sistema koji se koristi za prikaz (tj. ako se koristi `o` konverzija, broj će početi nulom, a ako se koristi heksadekadna konverzija, broj će početi sa `0x`),
- `w` – ceo broj koji predstavlja širinu polja koje će se koristiti za prikaz podatka,
- `d` – ceo broj koji ispred realnih podataka predstavlja broj cifara iza decimalne tačke, a ispred znakovnih nizova predstavlja maksimalan broj karaktera koji će biti ispisani,
- `h`, `l` i `L` – imaju isto značenje kao i kod ulazne konverzije.

Primer 2 Sledeći listing ilustruje primenu nekih od gorenavedenih konverzija.

```
1 int a;
2 unsigned b;
3 float x, y;
4 long double z;
5 // ...
```

```
6 printf("a=%-5d,␣b=%-3u\nx=%-8.4f,␣y=%-+15.7e\nz=%Lf", a,b,x,y,z);
```

Glava 3

Kontrola toka programa

U teoriji programiranja, definisana su tri osnovna tipa programskih struktura:

- sekvenca,
- grananja i
- programske petlje.

U programskom jeziku C,

- **sekvenca** se implementira korišćenjem bloka,
- **grananja** naredbama uslovnog (`if` i `switch`) i bezuslovnog skoka (`goto`, `break`, `continue`), a
- **programske petlje** naredbama `for`, `while` i `do-while`.

Skup programskih naredbi koje se izvršavaju onim redosledom kako su zapisane naziva se **blok naredbi**. Blok počinje simbolom `{`, a završava se simbolom `}`. Dakle, blok u C-u ima sledeći izgled:

Na početku bloka mogu biti navedeni opisi promenljivih i konstanti koje su lokalne za blok.

3.1 Naredbe grananja i skoka

3.1.1 Naredbe uslovnog grananja

`if` naredba

`if` naredbom se implementira osnovni tip selekcije (grananja) kojim se vrši uslovno izvršenje jedne od dve moguće naredbe.

Kôd u C-u ima sledeći izgled.

Kada je vrednost izraza ⟨izraz⟩ različita od nule (odnosno kada je `true`), izvršava se ⟨blok naredbi 1⟩, a ako je nula (tj. `false`), izvršava se ⟨blok naredbi 2⟩.

Primer 1 Sledeći program za dva (različita) uneta cela broja određuje veći od njih.

```
#include <stdio.h>
void main()
{
  int a, b;
  printf("Unesite broj a: ");
  scanf("%d", &a);
  printf("Unesite broj b: ");
  scanf("%d", &b);
  if ( a > b )
    printf("Veci broj je %d.", a);
  else
    printf("Veci broj je %d.", b);
}
```

Primer 2 Program koji za tri uneta cela broja određuje najveći od njih je prikazan u sledećem listingu. Kako bi se izbegle komplikacije, slučajevi kada su bar dva broja međusobno jednaka nije razmatran.

```
#include <stdio.h>
void main()
{
  int a, b, c, max;
  printf("Unesite broj a: ");
  scanf("%d", &a);
  printf("Unesite broj b: ");
  scanf("%d", &b);
  printf("Unesite broj c: ");
  scanf("%d", &c);
  if ( a > b )
    if ( c > a ) // c>a>b
      max = c;
    else // a>b i a>c
      max = a;
  else // b>a
    if ( c > b ) // c>b>a
      max = c;
    else // b>a i b>c
      max = b;
  printf("Najveci broj je %d.", max);
}
```

Jasno je da se slična tehnika može primeniti i za određivanje maksimuma za više od tri uneta broja. Međutim, pisanje takvog programa je zametan

i repetativan posao. Kasnije će biti reči o programu koji može odrediti maksimum za ma koliko unetih brojeva.

Primer 3 Dat je program koji izračunava vrednost funkcije $y = 1/x$, vodeći računa o tome da funkcija nije definisana za $x = 0$.

```
#include <stdio.h>
void main()
{
  float x;
  printf("Unesite vrednost argumenta x: ");
  scanf("%f", &x);
  if ( x == 0 )
    printf("Za x=0 funkcija y=1/x nije definisana.");
  else
    printf("y=%f", 1/x);
}
```

Primer 4 Sledi program koji računa vrednost funkcije

$$y = \begin{cases} x & \text{za } x < 2 \\ 2 & \text{za } 2 \leq x < 3 \\ x - 1 & \text{za } x \geq 2 \end{cases}.$$

```
#include <stdio.h>
void main()
{
  float x, y;
  printf("Unesite vrednost argumenta x: ");
  scanf("%f", &x);
  if ( x < 2 )
    y = x;
  else
    if ( x < 3 )
      y = 2;
    else
      y = x - 1;
  printf("y=%f", y);
}
```

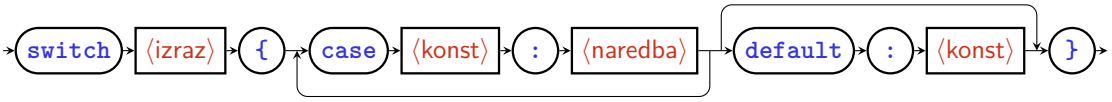
Postoji i skraćeni oblik `if` naredbe. Može se implementirati odluka da li će se jedna naredba u programu izvršiti ili ne (izostavljanjem ključne reči `else` i bloka naredbi ⟨blok naredbi 2⟩).

U tom slučaju, `if` grananje radi na sledeći način: kada je vrednost izraza ⟨izraz 1⟩ različita od nule (tj. `true`), izvršava se navedena naredba ⟨naredba⟩. U slučaju da je vrednost izraza ⟨izraz 1⟩ jednaka nuli (tj. `false`), ne izvršava se nijedna naredba.

`switch` naredba

Korišćenje `switch` naredbe je jedan od načina za realizaciju višestrukog grananja u programu.

Kôd u C-u ima sledeći izgled.



Vrednost izraza upoređuje se sa vrednostima navedenih konstanti. Ukoliko se vrednost izraza poklopi sa nekom od konstanti, izvršiće se niz naredbi iz `switch` strukture od naredbe obeležene tom konstantnom (uključujući i nju), pa sve do kraja strukture.

Ukoliko se vrednost izraza ne poklapa ni sa jednom od ponuđenih vrednosti izvršiće se samo naredba navedena u `default` delu (ukoliko on postoji). Izraz ⟨izraz⟩, kao i konstante ⟨konst⟩ mogu biti tipa `int` ili `char`.

Primer 5 Sledeći program štampa imena meseci u godini počev od tekućeg.

```
#include <stdio.h>
void main()
{
  int tekuci;
  printf("Unesite␣redni␣broj␣tekuceg␣meseca:␣");
  scanf("%d", &tekuci);
  switch(tekuci)
  {
    case 1: printf("Januar");
    case 2: printf("Februar");
    case 3: printf("Mart");
    case 4: printf("April");
    case 5: printf("Maj");
    case 6: printf("Jun");
    case 7: printf("Jul");
    case 8: printf("Avgust");
    case 9: printf("Septembar");
    case 10: printf("Oktobar");
    case 11: printf("Novembar");
    case 12: printf("Decembar");
  }
}
```

3.1.2 Naredbe bezuslovnog skoka

`break` naredba

Naredbom `break` se prekida izvršenje strukture u kojoj se naredba nalazi. Format naredbe je jednostavan:

Ovom naredbom se može prekinuti izvršenje programske petlje (izvršenje programa će se nastaviti od prve naredbe iza programske petlje), ali se najčešće koristi unutar `switch` strukture kako bi se omogućilo izvršenje samo naredbe obeležene odgovarajućom konstantom. Struktura koja se realizije kombinacijom `switch` strukture i `break` naredbe poznata je kao **struktura češlja**.

Primer 6 Sledeći progrm štampa ime tekućeg meseca za redni broj koji koristnik unosi sa tastature.

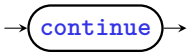
```
#include <stdio.h>
void main()
{
  int tekuci;
  printf("Unesite redni broj tekuceg meseca: ");
  scanf("%d", &tekuci);
  switch(tekuci)
  {
    case 1: printf("Januar"); break;
    case 2: printf("Februar"); break;
    case 3: printf("Mart"); break;
    case 4: printf("April"); break;
    case 5: printf("Maj"); break;
    case 6: printf("Jun"); break;
    case 7: printf("Jul"); break;
    case 8: printf("Avgust"); break;
    case 9: printf("Septembar"); break;
    case 10: printf("Oktobar"); break;
    case 11: printf("Novembar"); break;
    case 12: printf("Decembar"); break;
  }
}
```

Primer 7 Dat je program za štampanje broja dana tekućeg meseca.

```
#include <stdio.h>
main()
{
  int tekuci;
  printf("Unesite redni broj tekuceg meseca: ");
  scanf("%d", &tekuci);
  switch(tekuci)
  {
    case 1: case 3: case 5: case 7: case 8: case 10: case 12:
    printf("31"); break;
    case 2: printf("28"); break;
    case 4: case 6: case 9: case 11: printf("30");
    default: printf("Redni broj meseca nije korektan.");
  }
}
```

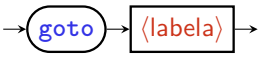
`continue` naredba

Naredba `continue` se može naći samo u tele neke programske petlje i njom se preikda izvršenje tekuće iteracije petlje. Format naredbe:



`goto` naredba

Format naredbe:



Izvršenje programa se nastavlja od naredbe obeležene navedenom labelom ⟨labela⟩. **Labele** se obeležavaju dodavanjem dve tačke posle imena labele (na primer `labela1:`).

3.2 Programske petlje

Programske petlje omogućavaju višestruko ponavljanje određenog dela programa u toku njegovog izvršavanja. Postoje dve osnovne vrste petlji. To su petlje sa konstantnim brojem prolaza (brojačke petlje) i petlje sa promenljivim brojem prolaza.

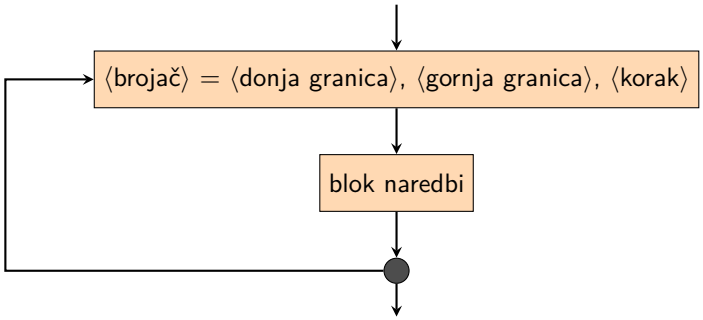
U programskom jeziku `C` postoje tri vrste programskih petlji:

- `for` petlja,
- `while` petlja,
- `do-while` petlja.

`for` petlja

`for` petlja se obično koristi kada u programu treba realizovati takozvanu brojačku petlju. Brojačka petlja podrazumeva da je unapred (pre ulaska u petlju) poznat (može da se izračuna) broj ponavljanja tela petlje.

Strukturni dijagram je prikazan na slici.



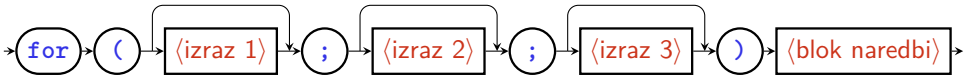
⟨brojač⟩ – ime promenljive koja predstavlja brojač petlje,

⟨donja granica⟩ – početna vrednost brojača petlje,

⟨gornja granica⟩ – konačna vrednost brojača petlje,

⟨korak⟩ – vrednost koja se dodaje brojaču petlje na kraju svake iteracije, ukoliko je izostavljena smatra se da je jednaka jedinici.

U programskom jeziku C, `for` petlja ima nešto širi smisao od navedene definicije brojačkih petlji. Format `for` naredbe u C-u je:



pri čemu:

⟨izraz 1⟩ – vrši inicijalizaciju promenljivih koje se koriste u petlji (što može da bude postavljanje početne vrednosti brojača petlje),

⟨izraz 2⟩ – predstavlja uslov na osnovu koga se odlučuje da li će se telo petlje još izvršavati ili se izvršenje petlje prekida – petlja se izvršava dok je vrednost ovog izraza različita od nule (uslov za izvršenje tela petlje može da bude postizanje gornje granice brojača petlje),

⟨izraz 3⟩ – definiše promenu vrednosti promenljivih koje se koriste u petlji, pri čemu se navedena promena vrši nakon svake iteracije (tu se može definisati kako se menja vrednost brojača petlje nakon svake iteracije).

Bilo koji od izraza može biti izostavljen, ali znak `;` se mora pisati.

Primer 1 Sledeći program štampa prvih n prirodnih brojeva za uneto n.

```
#include <stdio.h>
void main()
{
  int n, i;
  printf("Unesite␣prirodan␣broj.␣")
  scanf("%d", &n);
  for ( i = 1; i <= n; i++ )
    printf("%d␣", i);
}
```

Primer 2 Sledeći program štampa sve parne prirodne brojeve manje ili jednake od n za uneto n.

```
#include <stdio.h>
void main()
{
  int n, i;
  printf("Unesite prirodan broj. ")
  scanf("%d", &n);
  for ( i = 2; i <= n; i=i+2 )
    printf("%d ", i);
}
```

Primer 3 Ovaj program računa proizvod prvih n prirodnih brojeva (faktorijel broja n, tj. $n!$) i prikazuje ga na standardni izlaz.

```
#include <stdio.h>
void main()
{
  int n, i, p;
  p = 0;
  printf("Unesite prirodan broj. ")
  scanf("%d", &n);
  for ( i = 1; i <= n; i++ )
    p *= i; // p = p * i;
  printf("%d! = %d", n, p);
}
```

Primer 4 Sumiranje n brojeva čije se vrednosti unose sa tastature.

```
#include <stdio.h>
void main()
{
  int n, i, k, S;
  printf("Unesite koliko brojeva treba sumirati. ")
  scanf("%d", &n);
  S = 0;
  for ( i = 0; i < n; i++ )
  {
    printf("Unesite sledeci broj. ");
    scanf("%d", &k);
    S += k;
  }
  printf("Suma unetih brojeva je %d.", S);
}
```

Primer 5 U prošlom odeljku su opisani postupci nalaženja maksimalnog elementa od uneta dva ili tri. Šta je sa određivanjem maksimalnog elementa u opštem slučaju, kada se treba odrediti maksimum od n brojeva?

Program najpre od korisnika mora da zahteva da se specificira koliko će brojeva biti uneto (n). Glavna ideja rešenja jeste da se nakon svakog unetog broja proveri da li on najveći od svih prethodno unetih. Ovo je najlakše (i najoptimalnije) realizovati pomoću pomoćne promenljive (max) u kojoj će se u svakom trenutku nalaziti najveći od svih (do tod trenutka) unetih brojeva.

Nakon unosa prvog broja, jasno je da je taj broj istovremeno i najveći od svih. Zato je najbolje njega odmah smestiti u promenljivu max. Nakon toga treba uneti još $n-1$ brojeva, i nakon svakog sledećeg unetog proveriti da li je on veći od trenutnog najvećeg, i, ukoliko jeste, u pomoćnu promenljivu max smestiti njegovu vrednost.

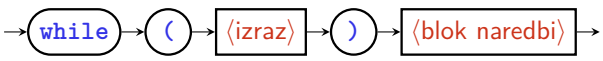
Opisani postpupak je realizovan u sledećem programu.

```
#include <stdio.h>
void main()
{
  int n, max, i, a;
  printf("Unesite broj elemenata: ");
  scanf("%d", &n);
  printf("Unesite 1. broj: ");
  scanf("%d", &max);
  for ( i = 2; i <= n; i++ )
  {
    printf("Unesite %d. broj: ");
    scanf("%d", &a);
    if ( a > max )
      max = a;
  }
  printf("Maksimalni element je %d.", max);
}
```

while petlja

while petlja omogućava ponavljanje bloka naredbi programa sve dok je definisani uslov zadovoljen (tj. dok je vrednost izraza različita od nule).

U C-u se ova naredba može opisati na sledeći način.



while naredba je jako korisna za realizaciju iterativnih procesa (sumiranje beskonačnih redova i slično).

Svaka for petlja se može jednoznačno preslikati u while petlju.

Primer 6 Recimo da treba napisati program u C-u za izračunavanje i štampanje vrednosti funkcije $\cosh x$ primenom sledećeg razvoja u red:

$$\cosh x = \sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!}.$$

Izračunavanje treba prekinuti kada relativna vrednost priraštaja sume postane manja od zadate vrednosti ε.

Jasno je da je na računaru nemoguće izračunati tačnu vrednost ovog reda, jer on predstavlja beskonačnu sumu. Pri računanju vrednosti $\cosh x$ moramo se zadovoljiti određenom tačnošću, a nju zadaje korisnik (vrednost ε). Obzirom na to da ne možemo unapred odrediti koliko članova sume je potrebno sumirati da bi se postigla zadata tačnost, ovde ne možemo koristiti brojačku `for` petlju, jer je kod nje potrebno unapred definisati koliko će se puta ona izvršiti. Zato koristimo `while` petlju, a uslov za izlazak iz petlje računamo.

Svaka sledeća (tačnija) suma dobija se dodavenjem sledećeg člana niza koji je opisan u datom redu, odnosno važi

$$|S_{i+1} - S_i| = |a_{i+1}|.$$

Odatle imamo da za relativnu grešku važi

$$\left|\frac{\Delta S}{S}\right| = \left|\frac{S_{i+1} - S_i}{S}\right| = \left|\frac{a_{i+1}}{S_i}\right| \approx \left|\frac{a_{i+1}}{S_{i+1}}\right|.$$

Ponovno računanje svakog člana niza "iz početka" bi zahtevalo previše vremena. Zato je potrebno naći rekurzivnu vezu, gde se sledeći član niza dobija na osnovu prethodnog člana. Opšti član zadatog niza je očigledno $a_k = \frac{x^{2k}}{(2k)!}$. Kako u ovom izrazu figuriše množenje i deljenje, potražićemo količnik dva susedna člana niza:

$$\frac{a_{k+1}}{a_k} = \frac{\frac{x^{2(k+1)}}{(2(k+1))!)}}{\frac{x^{2k}}{(2k)!}} = \frac{\frac{x^{2k+2}}{(2k+2)!}}{\frac{x^{2k}}{(2k)!}} = \frac{\frac{\cancel{x^{2k}} \cdot x^2}{(2k+2)\cdot(2k+1)\cdot\cancel{(2k)!}}}{\frac{\cancel{x^{2k}}}{\cancel{(2k)!}}} 4 = \frac{x^2}{(2k+1)(2k+2)}.$$

Pomoću dobijenog izraza možemo izračunati svaki sledeći član niza, ukoliko nam je poznat prethodni. Potrebno je samo izračunati prvi član niza (za $k = 0$ jer u datoj formuli sumiranje počinje za tu vrednost k):

$$a_0 = \frac{x^{2\cdot 0}}{(2 \cdot 0)!} = \frac{1}{0!} = \frac{1}{1} = 1.$$

Dakle, početna vrednost promenljive u kojoj ćemo računati svaki sledeći element niza je 1. Kako je očigledno $S_0 = a_0$, i promenljivu u kojoj čuvamo trenutnu vrednost sume ćemo incijalizovati na 1. Računanje sledećeg elementa niza, dodavnje dobijene vrednost elementa niza u sumu i uvećavanje brojača k treba ponavljati sve dok je ispunjen uslov $|a/S| > \varepsilon$. Sada nije teško napisati odgovarjući program u C-u:

```c
#include <stdio.h>
#include <stdlib.h>
void main()
{
  int k;
  float x, a, S, eps;
  printf("Unesite argument x i tacnost eps.");
  scanf("%f, %e", &x, &eps);
  a = 1;
  S = 1;
  k = 0;
  while ( abs(S/a) > eps )
  {
    a *= (x*x)/((2*k+1)*(2*k+2));
    S += a;
    k++;
  }
  printf("cosh(%f)=%f", x, S);
}
```

U drugoj liniji koda je uključena biblioteka `stdlib.h` zahvaljujući kojoj možemo da koristimo funkciju `abs()` koja računa i vraća apsolutnu vrednost izraza (u okviru uslova `while` petlje u dvanaestoj liniji koda).

U osmoj liniji koda smo `eps` pročitali po konverziji `%e` zato što je ε tipično jako mali broj, na primer 10^{-10}. Ovime omogućavamo korisniku da broj unese u obliku `1e-10` umesto `0.0000000001`.

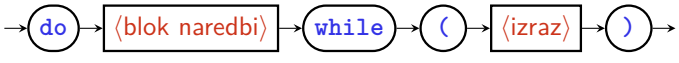
Ovaj kôd je moguće napisati u znatno kraćem (mada, bar na prvi pogled, u nešto komplikovanijem obliku) ukoliko ulogu `while` petlje formalno preuzme `for` petlja. U tom slučaju kôd može izgledati ovako:

```c
#include <stdio.h>
#include <stdlib.h>
void main()
{
  int k;
  float x, a, S, eps;
  printf("Unesite argument x i tacnost eps.");
  scanf("%f, %e", &x, &eps);
  for ( a=1, S=1, k=0; abs(a/S)>eps; a *= (x*x)/((2*k+1)*(2*k+2))
    , S += a, k++ );
  printf("cosh(%f)=%f", x, S);
}
```

`do-while` petlja

`do-while` petlja ima slično dejstvo kao i `while` petlja. Jedina razlika je u tome što se uslov za ponavljanje petlje u kodu nalazi iza njenog tela. Ovo znači da se naredba, koja predstavlja telo petlje, mora izvršiti bar jednom.

Format ove naredbe u programskom jeziku `C` je:



Primer 7 Sledeći program izračunava i štampa vrednost n-tog korena broja $a > 0$, primenom sledećeg iterativnog postupka:

$$x_0 = \frac{a+n-1}{n} \qquad x_{i+1} = \frac{(n-1)x_i + \frac{a}{x_i^{n-1}}}{n}.$$

Izračunavanje će se prekinuti kada je $|x_{i+1} - x_i| \leq \varepsilon$, gde je ε zadata tačnost.

```c
#include <stdio.h>
#include <math.h>
void main()
{
  int n;
  float x1, x2, a, eps, abs;
  printf("Unesite n, a i eps. ");
  scanf("%d %f %e", &n, &a, &eps);
  x2 = (a+n-1)/n;
  do
  {
    x1 = x2;
    x2 = ((n-1)*x1+a/pow(x1,n-1))/n;
    abs = x2-x1>0 ? x2-x1 : x1-x2;
  }
  while ( abs > eps );
  printf("x=%g", x2);
}
```

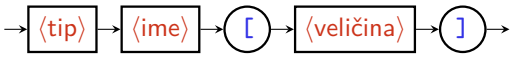
Glava 4

Polja

Polja predstavljaju složenu strukturu podataka sa elementima istog tipa.

4.1 Jednodimenzionalna polja

Jednodimenzionalna polja (nizovi, vektori) se deklarišu na sledeći način:



gde je:

⟨ime⟩ – ime polja,
⟨veličina⟩ – broj elemenata polja,
⟨tip⟩ – tip svakog pojedinačnog elementa polja (neki elementarni tip, tip pokazivača, polja ili strukture).

Prvi element polja ima uvek indeks `0`, drugi – indeks `1`, ..., poslednji element ima indeks ⟨veličina⟩-1.

Primer 1

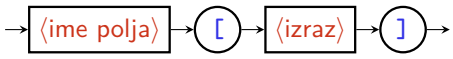
```
1 int a[100], b[50];
2 float x[30];
3 double d[2000];
```

U prvoj liniji su deklarisana dva niza (`a` i `b`), oba tipa `int`, pri čemu prvi ima dužinu 100 (indeksi se protežu počev od 0 zaključno sa 99), a drugi dužinu 50 (indeksi idu od 0 zaključno sa 49).

U drugoj liniji je deifnisan niz `x` dužine 30 čiji su elementi tipa `float`.

Poslednja linija sadrži deklaraciju niza čije je simboličko ime `d`, dužina mu je 2000, a tip svih elemenata je `double`.

Jedan od načina da se pristupi elementima polja je korišćenjem indeksa (tj. pozicije elementa u polju). Za pristup elementima polja pomoću indeksa koristi se operator `[]`. Opšti oblik izraza za pristup elementu polja je prikazan niže.



Primer 2

```
a[5]
b[i]
c[5*j+7]
```

Kako, uslovno rečeno, niz predstavlja skup (u opšten slučaju) različith elemenata, pa je nemoguće učitati ih sve jednim pozivom funkcije `scanf()`, potrebno je najpre učitati dužinu niza, a zatim pomoću (recimo) `for` petlje učitati svaki element ponaosob.

Primer 3 U sledećem isečku koda je prikazan unos elemenata u niz `a` tipa `int`.

```
printf("Unesite broj elemenata niza: ");
scanf("%d, &n");
for ( i = 0; i < n; i++ )
{
  printf("Unesite element sa indeksom %d", i);
  scanf("%d", &a[i]);
}
```

Primer 4 Slična situacija je i kod štampanja niza:

```
for ( i = 0; i < n; i++ )
  printf("Element sa indeksom %d je %d", i, a[i]);
```

Primer 5 Traženje minimalnog elementa celobrojnog niza (koristeći ideju iz prethodnog poglavlja).

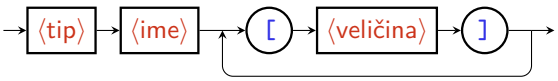
```
min = a[0];
for ( i = 1; i < n; i++ )
  if ( min > a[i] )
    min = a[i];
printf("Minimalni element niza a je %d.", min);
```

Primer 6 Sortiranje niza u neopadajući poredak.

```
for ( i = 0; i < n-1; i++ )
  for ( j = i+1; j < n; j++ )
    if ( a[i] > a[j] )
    {
      pom = a[i];
      a[i] = a[j];
      a[j] = pom;
    }
```

4.2 Višedimenzionalna polja

U definiciji višedimenzionalnog polja, svaka dimenzija se piše unutar jednog para uglastih zagrada. Definicija višedimenzionalnog polja u C-u:



Polje ima onoliko dimenzija koliko se veličina navede.

Najprostiji primer višedimenzionalnog niza jeste dvodimenzionalni niz. Dvodimenzionalni nizovi se još nazivaju i matrice. Matrica može da predstavlja, na primer, šahovsku tablu – osam vrsta i osam kolona.

Primer 1 U sledećem listingu je deklarisano nekoliko dvodimenzionalnih nizova.

```
char sahovska_tabla[8][8];
int sudoku[9][9];
int matrica[20][30];
double vektor[100][1];
float niz[1][100];
```

U slučaju dvodimenzionalnog polja, prva dimenzija predstavlja broj vrsti, a druga broj kolona matrice.

Elementi višeidmenzionalnih polja se u memoriji smeštaju kao jednodimenzionalna polja tako da se poslednji indeks najbrže menja, zatim pretpostlednji, itd. Dakle, u slučaju dvodimenzionalnog polja, elementi se smeštaju "po vrstama".

Pri definisanju jednodimenzionalnog polja, veličina polja može biti izostavljena u tri slučaja:

- ako se u definiciji vrši i inicijalizacija polja,
- ako je polje fiktivni argument funkcije i
- ako polje ima klasu memorije `extern`.

Primer 2 Sledeći program računa proizvod matica $A_{m\times n}$ i $B_{n\times k}$.

```
1  #include <stdio.h>
2  main()
3  {
4    int A[100][100], B[100][100], C[100][100], m, n, k, i, j, l,
       pom;
5    printf("Unesite␣dimenzije␣matrice␣A:␣");
6    scanf("%d␣%d", &m, &n);
7    printf("Unesite␣dimenzije␣matrice␣B:␣");
8    scanf("%d␣%d", &pom, &k);
9    if ( n != pom )
10   {
11     printf("Matrice␣se␣ne␣mogu␣pomnoziti.");
12     goto izlazak;
13   }
14   printf("Unesite␣elemente␣matrice␣A:\n");
15   for ( i = 0; i < m; i++ )
16     for ( j = 0; j < n; j++ )
17       scanf("%d", &A[i][j]);
18   printf("Unesite␣elemente␣matrice␣B:\n");
19   for ( i = 0; i < n; i++ )
20     for ( j = 0; j < k; j++ )
21       scanf("%d", &B[i][j]);
22   for ( i = 0; i < m; i++ )
23     for ( j = 0; j < k; j++ )
24     {
25       C[i][j] = 0;
26       for ( l = 0; l < n; l++ )
27         C[i][j] += A[i][l] * B[l][j];
28     }
29   printf("Proizvod␣matrica␣A␣i␣B␣je:\n");
30   for ( i = 0; i < m; i++ )
31   {
32     for ( j = 0; j < k; j++ )
33       printf("%3d", C[i][j]);
34     printf("\n");
35   }
36   izlazak:;
37 }
```

Štampanje elemenata matrice je moglo biti obavljeno i u okviru petlje u kojoj se računaju njeni elementi, ali je izdvojeno radi preglednosti.

U liniji 12 je iskorišćena `goto` naredba (odgovarajuća labela se nalazi na liniji 36) da se izgled programa ne bi preopteretio. Savetuje se izbegavanje korišćenje ove naredbe.

Lista početnih vrednosti elemenata polja se piše unutar para vitičastih zagrada. Inicijalizacija višedimenzionalnog polja se može vršiti na dva načina.

Primer 3

```
1  char samoglasniciV[5] = {'A', 'I', 'U', 'E', 'O'};
2  char samoglasniciM[ ] = {'a', 'i', 'u', 'e', 'o'};
3  int matrica1[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
4  int matrica2[3][4] = {
```

```
                        {1, 2, 3, 4},
                        {5, 6, 7, 8},
                        {9, 10, 11, 12}
                      };
int matrica3[][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
```

U drugoj linija je izostavljena dimenzija niza `samoglasniciM`, obzirom na to da je niz inicijalizovan pri deklaraciji, pa prevodilac može sam da izračuna broj elemenata.

U poslednjoj liniji je izostavljena prva dimenzija matrice `matrica3`. Pošto je broj kolona poznat, prevodilac može da izračuna broj vrsta na osnovu broja elemenata u inicijalizaciji.

Glava 5

Funkcije

Uobičajeno je da se pri pisanju programa koji treba da reše složenije probleme problemi razlažu na niz jednostavnijih (elementarnih) problema za čije rešavanje se pišu nezavisni programski moduli (funkcije), a osnovni problem se rešava pozivanjem tako definisanih funkcija.

Primer 4 Potrebno je izračunati zapreminu kupe izvodnice s i poluprečnika osnove r.

Iz matematike je poznato da se zapremina računa primenom formule $V = BH/3$, gde je B površina osnove kupe a H visina.

Površinu osnove možemo naći kao $B = \pi r^2$.

Visinu kupe H možemo naći primenom Pitagorine teoreme:

$$s^2 = r^2 + H^2 \Rightarrow H^2 = s^2 - r^2 \Rightarrow H = \sqrt{s^2 - r^2}.$$

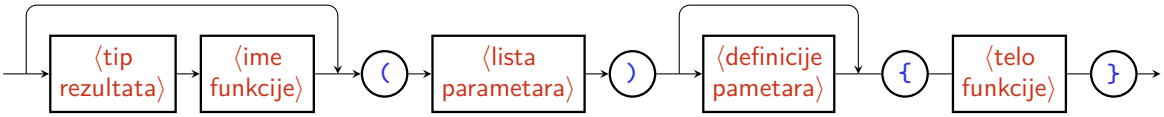
Za računanje visine, dakle, treba naći kvadrate brojeva s, i r, oduzeti ih, a zatim naći i njihov kvadratni koren.

Na ovaj način je prvobitni problem rešen svođenjem na jednostavnije probleme.

Glavni program u C-u je takođe definisan kao funkcija. To je funkcija `main()`.

5.1 Definisanje funkcije

Definicija funkcije u programskom jeziku C ima sledeći format:



gde je:

⟨ime funkcije⟩ – simboličko ime koje u isto vreme predstavlja i adresu funkcije,

⟨telo funkcije⟩ – blok naredbi (može biti i prazno),

⟨lista parametara⟩ – ovi parametri se nazivaju **formalni** ili **fiktivni parametri** (argumenti), oni rezervišu mesta za podatke iz pozivajućeg modula sa kojima će funkcija raditi nakon poziva; izrazi čije se vrednosti dodeljuju fiktivnim parametrima funkcije u trenutku poziva nazivaju se **stvarnim parametrima**,

⟨definicije parametara⟩ – definicija tipova fiktivnih argumenata funkcije, na isti način kako se definišu tipovi bilo koje promenljive,

⟨tip rezultata⟩ – određuje tip vrednosti koju funkcija vraća; ako je tip rezultata (tip funkcije) izostavljen, prevodilac će povratnu vrednost tretirati kao `int`; funkcija može biti i tipa `void` što znači da ne vraća nikakvu vrednost (kao procedura u `Pascal`-u).

Naredbom `return` **prekida se izvršenje funkcije**. Ova naredba može da vrati rezultat funkcije koji preuzima pozivajuća funkcija. Rezultat koji fukcija vraća je izraz navedenog tipa (tipa rezultata ili tipa funkcije). Funkcija može da sadrži nijednu (ukoliko je tipa `void`), jednu ili više naredbi `return`.

Jedan od mogućih načina da pozivajuća funkcija preuzme rezultat funkcije je sledeći:

5.2 Deklaracija funkcije

Svaka funkcija programskog jezika `C` treba da bude poznata kompajleru pre njenog poziva iz neke druge funkcije. Često se dešava da funkciju treba pozvati pre njene definicije. U tom slučaju, pre poziva funkcije, funkciju treba deklarisati.

Deklaracija funkcije omogućava poziv funkcije pre njenog definisanja. Deklaracija funkcije se naziva **prototip fukcije**.

Smisao deklaracjie je da se saopšti prevodiocu da takva funkcija postoji i da će njena definicija biti navdena negde kasnije u izvornom kodu.

Moguće su eksplicitne i implicitne deklaracije. Eksplicitne us one koje navodi progamer, a implicitne su one koje uvodi `C` prevodilac.

Eksplicitna deklaracija funkcije ima sledeći oblik:

Tip rezultata mora da se slaže sa tipom rezultata koji je naveden u kasnijoj definiciji funkcije.

U deklaraciji funkcije mogu se navesti i tipovi argumenata (što signalizira programeru koje argumente treba da dostavi funkciji na pozivu), a mogu se navesti i sama imena argumenata. To samo pomaže čitaocu koda da lakše razume kod, dok prevodilac jendostavno taj deo koda ignoriše.

Funkcija može biti deklarisana izvan tela drugih funkcija (globalni nivo) ili unutar tela neke druge funkcije. U prvom slučaju se kaže da je deklaracija **globalna**, a u drugom da je **lokalna** za tu funkciju u okviru koje je deklarisana.

Ne dozvoljava se definisanje jedne funkcije unutar neke druge, ali se može deklarisati.

Primer 1

```
1  double kvadrat();
2  void main()
3  {
4    double a, b, x, y, koren();
5    x = 4;
6    y = kvadrat(x);
7    a = 64;
8    b = koren(a);
9  }
10 void funkcija()
11 {
12   /* ... */
13 }
14 double kvadrat(double broj)
15 {
16   /* ... */
17 }
18 double koren(double broj)
19 {
20   /* ... */
21 }
```

U prvoj liniji koda deklarisana je funkcija `kvadrat()` na globalnom nivou. To znači da se ona može koristiti u svim funkcijama, uključujući i one koje su definisane pre nego što je ona definisana.

U četvrtoj liniji deklarisana je funkcija `koren()` na lokalnom nivou, u okviru funkcije `main()`. To znači da će funkcija `koren()` moći da se koristi u okviru funkcije `main()` (iako je definisana tek posle nje), ali neće moći da se koristi u, na primer, funkciji `funkcija()` (niti u funkciji `kvadrat()`).

Primer 2 Data je funkcija koja računa celobrojni stepen zadate osnove. U glavnom programu se koristi kreirana funkcija da se izračuna stepen realnog broja, pri čemu se taj broj i stepen unose sa tastature.

Da bi se još jednom istakao smisao i razlika između definicije i deklaracije funkcija, funkcija `stepen()` će biti definisana posle funkcije `main()`.

```
#include <stdio.h>
void main()
{
  int eksponent;
  float osnova, stepen();
  printf("Unesite␣osnovu␣i␣eksponent:␣");
  scanf("%f%d", &osnova, &eksponent);
  if ( eksponent < 0 )
  {
    osnova = 1/osnova;
    eksponent = -eksponent;
  }
  printf("%f^%d␣=␣%f", osnova, eksponent, stepen(osnova,
    eksponent));
}
float stepen(a, n);
float a;
int n;
{
  int i;
  float rez;
  for ( i = 1; i++ <= n; rez *= a);
  return rez;
}
```

5.3 Prenos parametara

Parametar se može prenositi na dva načina:

- po vrednosti (*call by value*) ili
- po referenci (*call by reference*).

U programskom jeziku C, parametri se prenose funkciji **po vrednosti**. Prenos parametara po vrednosti podrazumeva da se pri pozivu funkcije u operativnoj memoriji prave kopije za sve parametre funkcije. Funkcija radi sa tim kopijama i u trenutku završetka rada funkcije, te kopije se brišu iz operativne memorije. To automatski onemogućava da parametar bude promenjen u funkciji, a da to bude vidljivo u pozivajućem modulu.

Primer 1

```
#include <stdio.h>
void main()
{
  int A = -5, abs();
  printf("Vrednost␣A␣pre␣poziva␣funkcije:␣%d\n", A);
```

```
6     abs(A);
7     printf("Vrednost␣A␣posle␣poziva␣funkcije:␣%d", A);
8   }
9   void abs(int x)
10  {
11    x = x > 0 ? x : -x;
12  }
```

Na izlazu će se oba puta pojaviti vrednost `-5`, obzirom na to da se izmena broja `x` u funkciji obavila samo nad kopijama koje su napravljene u memoriji pri pozivu funkcije.

Ovo se može prevazići na sledeći način.

```
1   #include <stdio.h>
2   void main()
3   {
4     int A = -5, abs();
5     printf("Vrednost␣A␣pre␣poziva␣funkcije:␣%d\n", A);
6     A = abs(A);
7     printf("Vrednost␣A␣posle␣poziva␣funkcije:␣%d", A);
8   }
9   int abs(int x)
10  {
11    x = x > 0 ? x : -x;
12    return x;
13  }
```

U ovom slučaju smo u šestoj liniji kôda promenili vrednost promenljive `A`, tako što smo joj dodelili vrednost koju vraća funckija `abs(A)`, a to je (obzirom na liniju 12), vrednost koju ima **kopija** promenljive `x` u toku izvršenja funkcije.

Dakle, vrednost promenljive `A` se nije promenila zbog linije 11, već zbog toga što je u promenljivu `A` (linija 6) upisana povratna vrednost funkcije `abs()` (linija 12).

Prethodni primer ukazuje na jedno od mogućih rešenja problema koje nastaje u situacijama kada treba promeniti vrednost neke promenljive. Međutim, često je potrebno kreirati funkcije koje menjaju vrednosti više promenljivih. Jedan veoma elementaran primer takve funkcije koja se često koristi jeste zamena dve promenljive (na primer, kod sortiranja).

U takvim slučajevima, kada funkcija treba da vrati veći broj izlaznih podataka, jedino rešenje je da se funkciji umesto podataka prenesu **pokazivači na podatke** koje treba u funkciji menjati.

U tom slučaju, u trenutku poziva funkcije kreiraju se kopije za pokazivače, i u funkciji će se menjati sadržaji lokacija na koje ti pokazivači ukazuju, a sami pokazivači se brišu nakon završetka rada funkcije.

▸ **Primer 2** U sledećem listingu je prikazana implementacija koda iz prethodnog primera pomoću pokazivača.

```c
#include <stdio.h>
void main()
{
  int A = -5, abs();
  printf("Vrednost A pre poziva funkcije: %d\n", A);
  abs(&A);
  printf("Vrednost A posle poziva funkcije: %d", A);
}
void abs(int *x)
{
  *x = *x > 0 ? *x : -(*x);
}
```

Treba primetiti da je u šestoj liniji koda argument funkcije `abs()` *adresa promenljive* `A`. Da je navedeno samo ime `A`, tipovi fiktivnih i stvarnih argumenata se ne bi poklapali, i kompajler bi javio grešku.

Primer 3 Dat je kôd u kome je ilustrovana primena pokazivača kao argumenata funkcije na funkciji koja vrši zamenu dve promenljive.

```c
#include <stdio.h>
void izmena(int *x, int *y)
{
  int pomocna = *x;
  *x = *y;
  *y = pomocna;
}
void main()
{
  int a = 1, b = 2;
  printf("Pre poziva funkcije: a = %d, b = %d", a, b);
  izmena(&a, &b);
  printf("Posle poziva funkcije: a = %d, b = %d", a, b);
}
```

5.4 Parametri funkcije `main()`

Za razliku od ostalih funkcija C-a, funkcija `main()` se poziva određenom komandom operativnog sistema. Ona poseduje dva parametra koji se predaju programu na korišćenje prilikom njenog poziva:

- `argc` (***arg****ument* ***c****ount*) – prvi parametar; predstavlja broj parametara koje operativni sistem predaje programu (funkciji `main()`; on uvek ima početnu vrednost `1` jer prvi element niza `argv` pokazuje na ime programa;
- `argv` (***arg****ument* ***v****alue*) – dugi parametar; predstavlja pokazivač na tabelu čiji su elementi znakovni nizovi koji predstavljaju stvarne argumente funkcije `main()`; funkcija `main()`, dakle, može imati proizvoljan broj parametara.

Primer 1 Neka je program poizvan na sledeći način:
`pozdrav petar marija`
Za funkciju `main()`, ime programa je takođe parametar. To znači da ova funkcija ima ukupno tri parametara, odnosno važi: `argc=3`; `argv[0]="pozdrav"`, `argv[1]="petar"` i `argv[2]="marija"`.

Primer 2 Sledeći program štampa parametre funkcije `main()`.

```
#include  <stdio.h>
void main(int argc, char *argv[])
{
  int i;
  for ( i = 0; i < argc; printf("%s", argv[i++]);
}
```

5.5 Rekurzivne funkcije

Matematička definicija rekurentnih formula se može iskazati na sledeći način:

$$U_{n+1} = f(U_n, U_{n-1}, \ldots, U_{n-k+1}), \quad U_1 = a_1, U_2 = a_2, \ldots, U_k = a_k, \quad n \geq k.$$

Vrednost k je **rekurenta dubina**, a vrednosti a_1, ..., a_k su **početne vrednosti**. Za $k = 1$, rekurentnost se naziva **iteracija**.

Rekurzija je definisanje nekog pojma preko samog sebe. Drugim rečima, u definiciju se uključuje i pojam koji se definiše. C dozvoljava korišćenje rekurzivnih funkcija, tj. funkcija koje direktno ili indrektno pozivaju same sebe. Svaka iterativna procedura se može prevesti u rekurzivnu, ali obrnuto (u opštem slčaju) nije moguće. Nerekurzivna funkcija se iz programa može pozvati nerekurzivno i rekurzivno, dok se rekurzivna funkcija se može pozvati samo rekurzivno.

Da bi se izbegao "začarani krug", pored implicitne definicije koja uključuje rekurziju, mora da postoji i eksplicitni deo.

Primer 1 U jednom od primera iz prethodne glave opisan je program koji računa faktorijel unetog prirodnog broja. Suštinski, taj program je mogao da se opiše sledećom formulom:

$$n! = \begin{cases} 1 & n = 0 \\ n \cdot (n-1) \cdot \cdots \cdot 1 & n > 0 \end{cases}.$$

Broj 2! se onda računa preko donjeg izraza (jer je $n = 2 > 0$) kao $2! = 2 \cdot 1 = 2$. Primetimo da je za donji izraz ui programu potrebno napraviti `for` petlju. Promenljiva u kojoj ćemo smestiti rešenje mora imati početnu vrednost 1, i potrebno je definisati promenljivu koja će predstavljati brojač u brojačkoj petlji.

Kodovi rekurzivnih funkcija su znatno elegantniji, jer često nema potrebe za petljama, brojačima i početnim vrednostima (to ne znači da rekurzivne funkcije nemaju svoje nedostatke). Faktorijel prirodnog broja se može definisati rekurzivno na sledeći način:

$$n! = \begin{cases} 1 & n = 0 \\ n \cdot (n-1)! & n > 0 \end{cases}.$$

Izračunajmo 2! pomoću gornje rekurentne definicije.

- Kako je $n = 2 > 0$ koristimo donji izraz u definiciji. Dakle, $2! = 2 \cdot 1!$. Međutim, mi ne znamo koliko je 1! – ovde se u okviru definicije faktorijela broja dva javlja faktorijel broja jedan. Tu vrednost ponovo računamo pomoću definicije.
- Kako je i $n = 1 > 0$, ponovo koristimo donji izraz, odakle vidimo da je $1! = 1 \cdot 0!$. Ponovo se javlja novi faktorijel koji moramo da izračunamo koristeći definiciju.
- Kako je $n = 0$, formula nam kaže da treba da koristimo gornji izraz. Dakle, iz definicije vidimo da je $0! = 1$.
- Sa ovim podatkom ($0! = 1$), možemo da se vratimo računanju broja 1!. Ranije smo dobili da je $1! = 1 \cdot 0!$, i sada, znajući da je $0! = 1$, računamo $1! = 1 \cdot 1 = 1$.
- Sve ovo nam je bilo potrebno da bismo izračunali $2! = 2 \cdot 1!$. U opisanom postupku smo našli $1! = 1$, i sada lako sledi $2! = 2 \cdot 1 = 2$.

U ovom primeru je ilustrovana i važnost postojanja bar dve "grane" u rekurzivnoj formuli. Da nije postojao uslov $n = 0$ za koji se broj $n! = 0!$ ne računa tako što funkcija poziva samu sebe, goreopisani postupak bi se nastavio u beskonačnost.

Prilikom svakog poziva rekurzivne funkcije, za sve formale parametre i lokalne promenljive rezervišu se nova mesta u memoriji. Rekurzivne funkcije su obično kraće i elegantnije, ali je njihovo izvršenje duže i zahtevaju korišćenje znatno većehg dela memorijskog prostora, što ilustruje sledeći primer.

▸ **Primer 2** Fibonačijev niz se najčešće definiše kao niz brojeva kod kojih se svaki sledeći član dobija kao suma prethodna dva, a prva dva ("nulti" i "prvi") elementa su data početnim vrednostima $f_0 = 0$ i $f_1 = 1$.

Ovakva definicija je sama po sebi rekurzivna, i može se predstaviti matematički na sledeći način:

$$f_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ f_{n-1} + f_{n-2} & n \geq 2 \end{cases},$$

pri čemu je f_i oznaka za i-ti član Fibonačijevog niza.

Osnova za rekurziju je korišćenje posebne strukture podataka koja se naziva **stek**. Ova struktura funkcioniše po LIFO principu (***L**ast **I**n – **F**irst **O**ut*). Za svaki poziv rekurzivne funkcije, ulazni i izlazni argumenti, kao i adresa povratka, moraju da se čuvaju na steku.

▸ **Primer 3** U jednom od prethodnih primera je opisano kako suštinski radi rekurzivna funkcija koja računa faktorijel nenegativnog celog broja. Niže je prikazana odgovarajuća funkcija u C-u, a u glavnom programu je iskorišćena za računanje faktorijela unetog broja.

```
#include <stdio.h>
long fakt(int n)
{
  if ( n == 0 )
    return 1;
  else
    return n * fakt(n-1);
}
void main()
{
  int a;
  printf("Unesite broj: ");
  scanf("%d", &a);
  if ( a < 0 )
    printf("Faktorijel je definisan samo za pozitivne brojeve. ")
  else
    printf("%d! = %ld", a, fakt(a));
}
```

Radi poređenja, ekvivalentna nerekurzivna funkcija je prikazana u donjem listingu. Poziva se na isti način i rekurzivna funkcija, pa je glavni program izostavljen.

```
long fakt(int n)
{
  int i;
  long f = 1;
  for ( i = 1; i <= n; f *= i++ );
  return rez;
}
```

▸ **Primer 4** Data je funkcija koja računa najveći zajednički delilac dva broja, a zatim i odgovarajući kôd u C-u.

$$\text{NZD}\,(m,n) = \begin{cases} m & m = n \\ \text{NZD}\,(m-n,n) & m > n \\ \text{NZD}\,(n-m,m) & n < m \end{cases}$$

```
int NZD(int m, int n)
{
  if ( m == n ) return m;
  if ( m > n )
    return NZD(m-n, n);
```

```
  else // m <= n
    return NZD(n-m, m);
}
```

Primer 5 Dat je program koji prevodi prodan broj unet u dekadnom sistemu u proizvoljan brojni sistem. Brojni sistem q unosi koristnik i smatra se da je $q \leq 16$.

```
void stampajcifru(int k)
{
  if ( k < 10 )
    printf("%d", k);
  else
    printf("%c", 'a' + k - 10);
}
void prevedi(n, m)
int n, m;
{
  int i = n, vc[32], k = 0;
  while ( i > m )
  {
    vc[k++] = i % m;
    i /= m;
  }
  vc[k++] = i;
  for ( i = k-1; i >= 0 ; i-- )
    stampajcifru(vc[i]);
}
void main()
{
  int n, q;
  printf("Unesite broj (dekadno) i ciljnu osnovu: ")
  scanf("%d %d", &n, &q);
  prevedi(n, q);
}
```

Primetimo da funkcija `prevedi()` ne vraća prevedeni broj, već ga direktno štampa. Zato je dovoljno samo pozvati funkciju u glavnom programu, i odgovarajući ekvivalent u zadatom brojnom sistemu će biti ispisan na ekranu.

U sledećem listingu je prikazana rekurzivna verzija funkcije `prevedi()`.

```
void prevedi(n, m)
int n, m;
{
  if ( n < m )
    stampajcifru(n);
  else
  {
    prevedi(n/m, m);
    stampajcifru(n%m);
  }
}
```

Primer 6 Funkcija `main()` se ponaša kao i svaka druga funkcija i može se pozvati rekurzivno.

```
#include <stdio.h>
void main()
{
  printf("Zdravo!\n")
  main();
}
```

5.6 Standardna biblioteka C funkcija

C obezbeđuje veliku funkcionalnost kroz skup već realizovanih funkcija iz svoje standardne biblioteke.

Na primer, ako programer želi da koristi gotove matematičke funkcije, mora uključiti odgovarajuću biblioteku: `#include <math.h>`. Na taj način će dobiti na raspolagađu veliki broj funkcija:

- `abs` – apsolutna vrednost za tip `int`,
- `acos` – arkus kosinus (inverzna funkcija od kosinusa, arccos),
- `cos` – kosinus (cos),
- `cosh` – kosisnus hiperbolički (cosh),
- `exp` – eksponencijalna funkcija ($e^x = \exp(x)$),

  ...

Za rad sa znakovnim podacima (stringovima), mora se uključiti sledeći *header* fajl: `#include <string.h>`. Na taj način će se na raspolaganju dobiti veliki broj funkcija za rad sa znakovnim podacima:

- `sprintf`, `_stprintf` – štampanje podataka u string,
- `strcat`, `wcscat` – nadovezivanje stringova,
- `strcmp`, `wcscmp` – poređenje dva stringa,
- `strcpy`, `wcscpy` – kopiranje jednog stringa u drugi,

  ...

Glava 6

Pokazivači, strukture i unije

Izvedenim tipovima podataka u programskom jeziku C pripadaju:

- nizovi (polja),
- pokazivači (pointeri),
- strukture,
- unije i
- znakovni nizovi (stringovi).

6.1 Pokazivači (pointeri)

Pokazivač je promenljiva koja sadrži adresu promenljive ili funkcije. Korišćenje pokazivača u C-u je od posebnog značaja jer se često postiže kompaktniji i efikasniji programski zapis. Sa druge strane, njihovo nekontrolisano korišćenje može da dovede do teško čitljivih ili potpuno nerazumnih programa.

U neposrednoj vezi sa pokazivačima su dva specijalna operatora programskog jezika C:

> & (ampersend) – operator **adresa od** ili operator **referenciranja** daje memorijsku adresu objekta na koga je primenjen;
>
> * (zvezdica) – posredni operator (operator **indirekcije** ili **dereferenciranja**); njime se upravlja memorijskim lokacijama u pokazivačkim promenljivama.

Primer 1

```
1 #include <stdio.h>
2 void main()
3 {
4   int a;
5   int *p;
6   p = &a;
7   a = 5;
```

```
8     printf("%d\n", a);
9     printf("%d\n", *p);
10    printf("%d\n", &a);
11    printf("%x\n", p);
12  }
```

U šestoj liniji koda u promenljivu `p` upisujemo adresu memorijske lokacije promenljive `a`. Na taj način smo obezbedili da pokazivač (pointer) `p` pokazuje na memorijsku lokaciju u kojoj smeštamo vrednosti promenljive `a`.

U sedmoj liniji koda je promenjena vrednost promenljive `a` (u memorijsku lokaciju `&a` je sada upisana vrednost `5`).

Osmom linijom koda štampamo vrednost promenljive `a` (sadržaj memorijske lokacije `&a`).

U devetoj liniji koda funkcija `printf()` je pozvana tako da štampa pokazivač. U šestoj liniji koda smo u promenljivu `p` smestili adresu memorijske lokacije u kojoj se čuva promenljiva `a`. Zato sada `*p` pokazuje na promenljivu `a`. Štampanjem `*p`, dakle, štampamo sadržaj memorijske lokacije `&a`, odnosno sâmu promenljivu `a`.

U desetoj liniji se štampa adresa memorijske lokacije u koju se smešta vrednost `a`.

Nakon izvršenja jedanaeste linije koda, na ekranu će biti odštampan sadržaj memorijske lokacije u koju se smešta vrednost promenljive `p`. Jasno je (najviše na osnovu linije 6) da se u `p` nalazi memorijska lokacija u koju se smešta vrednost promenljive `a`, odnosno `&a`.

Dakle, prve dve linije koje sadrže funkciju `printf()` će štampati `5`, a poslednje dve će štampati ime memorijske lokacije u kojoj je smeštena promenljiva `a`. Ime te lokacije će se razlikovati svaki put kada se program pokrene, jer programer nije taj koji odlučuje gde će se smeštati promenljive, već operativni sistem. Svaki put kada se program pokrene, operativni sistem programu (tj. promenljivama koje se javljaju u programu) dodeli određeni skup memorijskih lokacije koje "sme" da koristi.

Primer 2 Koje su vrednosti promenljivih `x` i `y` na kraju izvršenja ovog dela programa?

```
1  int x = 1, y = 2, z[10];
2  int *ip;
3  ip = &x;
4  y = *ip;
5  x = 2;
6  *ip = 0;
7  ip = &z[0];
8  ip = &x;
9  *ip = *ip + 10;
10 *ip += 1;
11 ++*ip;
```

```
12 (*ip)++;
13 int *iq;
14 iq = ip;
15 int **ir;
16 ir = &iq;
17 **ir += 3;
```

U prvoj liniji koda se deklarišu i istovremeno inicijalizuju promenljive `x`, `y` i `z[]`.

U drugoj liniji se deklariše pointer `*ip`.

Nakon izvršenja treće linije, promenljiva `ip` će imati vrednost `&a`, odnosno u promenljivu `ip` će biti upisano ime memorijske lokacije u koju se smeštaju vrednosti promenljive `a`. Ovime je ostvarena veza između promenljive `a` i pokazivača `*ip`.

U liniji broj 4, `y` dobija vrednost promenljive na koju pokazuje `ip`. Kako `ip` pokatuje na `x` (prethodna linija), u promenljivoj `y` se sada nalazi `1`.

Peta linija je sama po sebi jasna – `x` dobija vrednost `2`.

U šestoj liniji menjamo sadržaj memorijske lokacije na koju pokazuje `ip` na nulu. Kako `ip` i dalje pokazuje na `x`, sada će u `x` biti upisana vrednost `0`. Treba primetiti da sâma promenljiva `ip` nije promenila svoju vrednost, jer ona i dalje pokazuje na `x` – a memorijska lokacija gde se čuva vrednost promenljive `x` je ostala nepromenjena (ona se i ne može promeniti, jer operativni sistem odlučuje o tome gde će čuvati promenljive).

Posle izvršenja sedme linije, pokazivač `*ip` više neće pokazivati na promenljivu `x`, već na prvi element u nizu `z[]` – a to je element sa indeksom nula, odnosno na `z[0]` (u `ip` se smešta adresa memorijske lokacije u kojoj se čuva prvi element niza).

Nakon toga u `ip` ponovo smeštamo adresu memorijske lokacije u kojoj se čuva promenljiva `x`. Sada `ip` ponovo pokazuje na `x`.

U devetoj liniji koda u memorijsku lokaciju na koju pokazuje `ip` (a to je memorijska lokacija čija je adresa `&x`) smeštamo vrednost izraza `*ip+10`, a to je isto što i `x+10`. Kako je u `x` trenutno upisana vrednost `1`, vrednost izraza `x+10` je `10`. Dakle, u adresu `&x` je sada upisana vrednost `10`.

U desetoj liniji koda se vrednost promenljive na koju pokazuje `ip` uvećava za `1` i rezultat smešta u promenljivu `y`. Dakle, `y` sada iznosi `11`.

Jedanaesta i dvanaesta linija koda raste istu stvar kao i deseta – uvećavaju promenljivu na koju pokazuje `ip` (a to je promenljiva `x`) za `1`. Dakle, posle izvršenja linije koda broj 12, `x` ima vrednost `13`.

U trinaestoj liniji koda deklarisan je nov pokazivač na `int`, `iq`.

U četrnaestoj liniji koda u promenljivu `iq` se upisuje vrednost promenljive `ip`. Dakle, promenljiva `iq` sada ima vrednost `&x`. Primetimo da ovime nije promenjena vrednost promenljive `y`, već je samo uklonjen pokazivač.

U petnaestoj liniji koda deklarisan je pokazivač na pokazivač na `int`, `ir`.

Polse izvršenja šesnaeste linije koda, `ir` pokazuje na pokazivač `iq`. To znači da se u promenljivoj `ir` čuva adresa memorijske lokacije promenljive (a ta promenljiva je neki drugi pokazivač) `iq`.

Poslednja linija koda uvećava vrednost promenjive `x` za `3` (pokazivač `ir` pokazuje na `iq`, a on pak pokazuje na `x`).

Dakle, nakon izvršenja i poslednje linije koda, `x` ima vrednost `16`, a `y` ima vrednost `11`.

Napomena Treba voditi računa o prioritetu operacija.

```
ip = &x;
++*ip;   /* ++x */
(*ip)++; /* x++ */
*ip++;   /* *(ip++) */
```

Napomena Nije moguće definisati pokazivač na konstantu ili izraz. Takođe nije moguće promeniti adresu promenljive (jer to ne određuje programer već operativni sistem).

Nijedna od sledećih linija kôda nije ispravna.

```
i = &3;
j = &(k+5);
k = &(a==b);
&a = &b;
&a = 150;
```

Kao što je ranije rečeno, u programskom jeziku `C` parametri se prenose funkciji **po vrednosti**.

Prenos parametara po vrednosti podrazumeva da se pri pozivu funkcije u operativnoj memoriji prave kopije za sve parametre funkcije. Funkcija radi sa tim kopijama i u trenutku završetka rada funkcije, te kopije se brišu iz operativne memorije. To automatski onemogućava da parametar funkcije bude promenjen u funkciji, a da to bude vidljivo u pozivajućem modulu.

Ukoliko funkcija treba da vrati veći broj izlaznih podataka, jedino rešenje je da se koristi **prenos po referenci**, odnosno da se funkciji, umesto podataka, prenesu pokazivači na podatke koje treba u funkciji menjati.

U tom slučaju, u trenutku poziva kreiraju se kopije za pokazivače. U funkciji će se menjati sadržaji lokacija na koje ti pokazivači pokazuju (to su stvarne lokacije koje i želimo da izmenimo), a sami pokazivači se brišu nakon završetka rada funkcije.

Primer 3

```c
#include <stdio.h>
void zamena(int *x, int *y)
{
  int pom = *x;
  *x = *y;
  *y = pom;
}
void main()
{
  int a = 1, b = 2;
  printf("Pre poziva funkcije: a = %d, b = %d\n", a, b);
  zamena(&a, &b);
  printf("Posle poziva funkcije: a = %d, b = %d\n", a, b);
}
```

U C-u, pokazivači i polja su tesno povezani. Samo ime polja je, u stvari, adresa početka polja. U tom smislu, ako definišemo polje `a[10]`, `a` i `&a[0]` su iste stvari. Kako su elementi polja smešteni u sukcesivnim (uzastopnim) memorijskim lokacijama, jasno je da je `a+1` isto što i `&a[1]`. Uopšteno, imamo da je `a+index` isto što i `&a[index]`.

Analogno tome, zaključujemo, u opštem slučaju, da je `a[index]` isto što i `*(a+index)`.

Kako ime polja (`a`) nije promenljiva, izraz `a++` nema smisla. Ako, međutim, uvedemo novi pokazivač (na primer, `*pa`) koji pokazuje na prvi element niza `a`, onda je izraz `pa++` sasvim korektan, jer se radi o posebnom promenljivoj koja je tipa pokazivač.

Programski jezik C dozvoljava da se za vrednost praznog pointera koristi `NULL` vrednost. `NULL` pointer je, u stvari, pokazivač na nultu adresu i kada se pokuša da se pristupi elementu `*p`, (pri čemu je `p=NULL`) javlja se takozvana *run-time* greška u programu. Prevodilac neće prijaviti grešku, već će se greška javiti u toku izvršenja programa.

6.2 Strukture

Strukture predstavljaju kompleksne tipove podataka koji mogu biti promenljive istog ili različitog tipa. One predstavljaju pogodno sredstvo za rad sa podacima koji su u međusobnoj vezi, jer se mogu grupisati pod istim imenom.

Primer 1 Struktura može biti, na primer, student. Svaki student ima svoje ime, prezime, jedinstveni matični broj, broj indeksa, listu predmeta koje je položio, itd.

Primer 2 Kako kompleksni broj ima svoj realan i imaginaran deo, on se može predstaviti kao struktura.

Kompleksni broj se može jednoznačno opisati i pomoću modula i argumenta, pa umesto realnog i imaginarnog dela, moduo i argument mogu biti elementi strukture.

Opšta forma strukture u C-u je:

`struct` – ključna reč koja jedinstveno implicira da će se koristiti struktura;

⟨naziv strukture⟩ – ime strukture koje mora biti jenstveno u celom programskom modulu, dok se članovi strukture specificiraju listom deklaracije promenljivih; oni se nalaze unutar vitičastih zagrada, svaki član je opisan sopstvenom deklaracijom i oni mogu biti bilo koji tip podataka, uključujući i strukture.

Napomena Ne treba zaboraviti cela deklaracija šablona strukture predstavlja jednu naredbu i zato se ona obavezno završava znakom `;`.

Primer 3

```
1 struct tacka
2 {
3   float x;
4   float y;
5 };
```

Nakon što je određen format strukture, čime je programskom prevodiocu saopštena informacija kako da upravlja podacima, strukturne promenljive se kreiraju saglasno pravilima korišćenja strukturnih promenljivih:

Primer 4 Promenljive `a`, `b` i `c` su strukturne promenljive tipa `tacka` (struktura koja je definisana u prethodnom primeru).

```
1 struct tacka a, b, c;
```

Strukturne promenljive se mogu deklarisati i bez eksplicitnog imenovanja strukture.

Primer 5

```
struct
{
  int x;
  int y;
} a, b, c;
```

Gornji listing predstavlja definisanje formata neimenovane strukture i deklaraciju promenljivih a, b i c koji su tipa definisane strukture.

Niže je prikazana slična situacija, osim što je struktura imenovana, pa se i kasnije mogu deklarisati promenljive istog tipa (kao što je učinjeno u poslednjoj liniji sa promenljivama d i e).

```
struct tacka
{
  int x;
  int y;
} a, b, c;
struct tacka d, e;
```

Inicijalizacija članova strukture se vrši isto kao kod nizova.

Primer 6

```
struct tacka koordinatni_pocetak = {0, 0}; //inicijalizacija
    strukture
```

Za pristup članovima strukture se koristi operator člana strukture – tačka (.):

Primer 7

```
struct tacka { float x, y; }; // deficija
struct tacka koordinatni_pocetak; // deklaracija
koordinatni_pocetak.x = 0; //inicijalizacija
koordinatni_pocetak.y = 0; //inicijalizacija
```

C podržava princip ugneždenih struktura (tj. član strukture može takođe biti struktura).

Primer 8

```
struct tacka { float x, y; };
struct pravougaonik
{
  struct tacka dole_levo;
  struct tacka gore_desno;
};
```

```
struct pravougaonik ekran;
  ekran.dole_levo.x = 600;
```

Pokazivači se mogu koristiti kod strukturnih tipova na isti način kao i kod osnovnih tipova. Ukoliko se koriste pokazivači, onda se za pristup članovima strukture može koristiti i operator `->`.

Operacije nad strukturama su kopiranje i dodela vrednosti, uzimanje adrese strukture i pristup članovima strukture. Zato se strukture mogu pojaviti kao argumenti funckije (kopiranje), odnosno kao vrednost koju funkcija vraća (dodela). Strukture se ne mogu porediti.

Primer 9 Sledeći program proverava da li se zadata tačka nalazi unutar zadatog pravougaonika. Funkcija `testirajTacku` vraća vrednost različitu od nule ukoliko se tačka nalazi unutar zadatog pravougaonika.

```
#include <stdio.h>
struct Tacka
{
  float x;
  float y;
}
struct Pravougaonik
{
  struct Tacka dole_levo;
  struct Tacka gore_desno;
}
int testirajTacku(struct Pravougaonik, struct Tacka*);
void main()
{
  struct Tacka testTacka;
  int x1, y1, x2, y2;
  struct Pravougaonik testPravougaonik;
  printf("Unesi dve tacke za pravougaonik: ");
  scanf("%f%f%f%f", &x1, &y1, &x2, &y2);
  testPravougaonik.dole_levo.x = x1;
  testPravougaonik.dole_levo.y = y1;
  testPravougaonik.gore_desno.x = x2;
  testPravougaonik.gore_desno.y = y2;
  printf("Unesi tacku koja se testira: ");
  scanf("%f%f", &x1, &y1);
  testTacka.x = x1;
  testTacka.y = y1;
  if ( testirajTacku(testPravougaonik, &testTacka) )
    printf("Tacka je unutar pravougaonika.");
  else
    printf("Tacka je van pravougaonika");
}
int testirajTacku(struct Pravougaonik pr, struct Tacka *t)
{
  return t->x > pr.dole_levo.x && t->x < pr.gore_desno.x && t->y
    > pr.dole_levo.y && t->y < pr.gore_desno.y;
}
```

Polja struktura se formiraju na isti način kao i polja podataka elementarnih tipova.

Primer 10

```
struct tacka
{
  int x;
  int y;
};
struct tacka tacke[50]; // deklaracija niza struktura
tacke[10].x = 5; // pristup clanu strukture
tacke[10].y = 15; // pristup clanu strukture
```

Nije dozvoljeno da struktura sadrži instancu same sebe, ali je sasvim korektno da sadrži pokazivač na instancu same sebe.

Primer 11 Struktura lančane liste.

```
struct cvor
{
  int sadrzaj;
  cvor *sledeci;
}
```

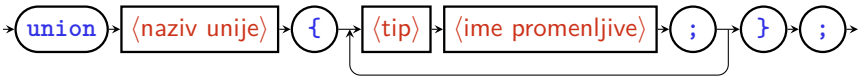
Često se u sprezi sa definisanjem strukture koristi i naredba za definisanje novog tipa.

Primer 12

```
struct tacka
{
  int x, y;
};
typedef struct tacka point; //definisanje novog tipa
point pt1, pt2; //deklaracija promenljivih
pt1.x = 15; //pristup clanu strukture
```

6.3 Unije

Unija kao tip podataka poseduje veoma mnogo sličnosti strukturi, uz neke bitne razlike. Naime, dok je struktura agregatni tip podatka koji okuplja objekte (promenljive) čiji su tipova (u opštem slučaju) različiti, unija je elementarni tip podatka od samo jednog elementa koji u nekom trenutku može da bude samo jedan od specificranih tipova. Naravno, promenljiva je dovoljno velika da može da sadrži najveći od tipova članice unije.



Deklaracija promenljivih tipa unije i pristup elementima unije je potpuno isti kao i kod struktura.

Primer 1

```
union naziv_unije
{
  int ival;
  float fval;
  char *sval;
} unija; //definicija unije i deklaracija promenljive
unija.ival = 5; //pristup clanici unije
unija.fval = 5.5; //pristup clanici unije
/* unije sadrzi samo poslednju postavljenu vrednost*/
```

Glava 7

Dinamička alokacija (zauzimanje) memorije

Zauzimanje memorije se obično vrši pomoću funkcije `malloc()`:

```
void *malloc(size_t size);
```

Funkcija rezerviše prostor u memoriji veličine `size` bajtova, i kao rezultat vraća pokazivač na početak rezervisanog prostora.

Ako se tražena količina memorije ne može rezervisati, kao rezultat se dobije `null` pointer (`null` pokazivač je definisana pokazivačka vrednost, ali ne označava stvarnu adresu; u stvari, njena vrednost je 0).

Primer 2 Sledećom komandom je rezervisano 100 bajtova u memoriji.

```
1 malloc(100);
```

Operator `sizeof` vraća veličinu operanda u bajtovima. Na primer, `sizeof(int);` će vratiti 4 ili 2, u zavisnosti od kompajera, jer je veličina podatka tipa `int` obično 4 ili 2 bajta.

Primer 3

```
1 malloc(sizeof(int));
2 double x;
3 int s1 = sizeof(x);
4 int s2 = sizeof(int);
```

U prvoj liniji koda je rezervisano 4 (odnosno 2) bajta u memoriji. U trećoj liniji je promenljivoj `s1` dodeljena vrednost `8`, a u četvrtoj je u `s2` smešten broj `4`, odnosno `2`.

Pokazivač koji je rezultat funkcije `malloc()` je tipa `void`, što znači da nije definisano za koji tip podatka u njemu može biti smeštena adresa. Međutim, rezultat funkcije se može pridružiti nekom pokazivaču preko `cast` operatora.

Primer 4

```
int *p;
p = (int*) malloc(sizeof(int));
*p = 5;
```

U prvoj liniji koda se rezerviše prostor za promenljivu `p`.

U drugoj liniji se rezerviše prostor za neku promenljivu tipa `int`, i ta adresa se smešta u promenljivu `p`.

Poslednja linija koda predstavlja indirektnu dodelu vrednosti `p` u oslobođenu memorijsku lokaciju.

Primer 5 Operator `sizeof` može da se primeni i na polja i strukture.

```
int ar[25];
int i = sizeof(ar);      // i = 25*4
int j = sizeof(ar[0]);   // j = 4
int len = i/j;           // len = 25
struct st
{
  int i;
  char c;
};
int k = sizeof(st);      // k = 4+4 = 8
```

Zauzimanje memorije se može izvršiti i pomoću funkcije `calloc()`.

```
void *calloc(size_t num, size_t size);
```

Funkcija rezerviše prostor u memoriji (i inicijalizuje je na vrednost `0`) za `num` elemenata veličine `size` bajtova, i kao rezultat vraća pokazivač na početak rezervisanog prostora. Ako se tražena količina memorije ne može rezervisati, kao rezultat se dobije `null`.

Za oslobađanje memorije zauzete funkcijama `malloc()` i `calloc()` se koristi funkcija `free()`.

```
void free(void *memblock);
```

Argument `memblock` je pokazivač na memorijski blok koji treba osloboditi. Funkcija ne vraća nikakvu vrednost.

Primer 6 Alociranje memorije za 40 `long` podatka.

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
void main(void)
{
  long *buffer;
  buffer = (long*) calloc(40, sizeof(long));
  if ( buffer != NULL )
    printf("Alocirano je 40 long int-a.");
  else
    printf("Ne moze se alocirati memorija.");
  free(buffer); //oslobadjanje memorije
}
```

Realociranje (ponovno zauzimanje memorije) se vrši pomoću funkcije `realloc()`.

```
void *realloc(void *memblock, size_t size);
```

Funkcija oslobađa rezervisani blok na koji pokazuje `memblock` i rezerviše novi veličine `size` bajtova, i kao rezultat vraća pokazivač na početak rezervisanog prostora. Ako se tražena količina memorije ne može rezervisati, kao rezultat se dobije `NULL` pointer.

Napomena Česte greške su takozvani "viseći" pointeri i "curenje" memorije (*memory leaks*).

Primer 7 Funkcija koja dodaje novi čvor sa vrednošću `x` u lančanu listu `lista`, odmah iza prvog čvora koji ima vrednost `y`.

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
struct cvor
{
  int sadrzaj;
  cvor sledeci;
};
void dodaj(struct cvor *lista, int x, int y)
{
  struct cvor *p, *novi;
  p = lista;
  while ( p->sadrzaj != y && p != NULL )
    p = p->sledeci;
  if ( p != NULL )
  {
    novi = (struct cvor *)malloc(sizeof(struct cvor));
    novi->sadrzaj = x;
    novi->sledeci = p->sledeci;
    p->sledeci = &novi;
```

```
21    }
22    else
23      printf("Ne postoji element sa zadataim kljucem.");
24  }
```

7.1 Preprocesorske direktive

Preprocesorske direktive u C jeziku omogućavaju jednostavniji razvoj i modifikaciju programa, i, što je posebno bitno, omogućavaju veću prenosivost programa pisanih u C jeziku na različita hardverska i softverska okruženja.

U okviru svakog C programskog prevodioca postoji preprocesor koji je njegov sastavni deo i koji omogućava prihvatanje specijalnih iskaza koji su pisani zajedno sa iskazima C jezika.

C preprocesor vrši analizu teksta programa pre C prevodioca i omogućava identifikaciju preprocesorskih naredbi uključivanjem specijalnog karaktera #. Ovaj karakter mora biti prvi u preprocesorskoj naredbi jer je sintaksa iskaza preprocesora različita od sintakse iskaza C jezika.

Napomena Preprocesorke direktive se obično navode na početku programa i **ne** završavaju se tačkom-zarez (;).

U preprocesorke direktive spadaju:

- direktive za definisanje simboličkih konstanti i makroa,
- direktive za uključivanje datoteka i
- direktive za uslovnu kompilaciju.

Simboličke konstante i makroi se definišu korišćenjem preprocesorke direktive #define.

Primer 1

```
1  #define PI 3.14159
2  #define E 2.71828
3  #define BRZINA_SVETLOSTI 2.99792e8
4  #define PERMEABILNOST_VAKUUMA 8.85418e-12
5  #define MAX_BROJ_POENA 40
6  #define TRUE 1
7  #define FALSE 0
8  #define AND &&
9  #define OR ||
10 #define EQ ==
11 /* ... */
12 povrsina = poluprecnik*poluprecnik*PI;
13 game_over = ( brojPoena > MAX_BROJ_POENA ) ? TRUE : FALSE;
14 do { /* ... */ } while ( !game_over );
15 /* ... */
```

Primer 2 Funkcija koja računa vrednost PDV-a na osnovni iznos.

```
#include <stdio.h>
#define PDV 0.18
void main()
{
  float glavnica;
  float porez;
  glavnica = 72.10;
  porez = glavnica * PDV;
  pritnf("Porez na %.2f iznosi %.2f.", glavnica, porez);
}
```

Preprocesor prvo zamenjuje sve simboličke konstante pre nego što je program preveden, tako da nakon preprocesiranja (a pre prevođenja), program izgleda ovako:

```
#include <stdio.h>
#define PDV 0.18
void main()
{
  float glavnica;
  float porez;
  glavnica = 72.10;
  porez = glavnica * 0.18;
  pritnf("Porez na %.2f iznosi %.2f.", glavnica, porez);
}
```

Napomena

```
#include <stdio.h>
#define PDV 0.18
void main()
{
  float glavnica;
  float porez;
  glavnica = 72.10
  PDV = 0.15; // neispravno
  porez = glavnica * PDV;
  pritnf("Porez na %.2f iznosi %.2f.", glavnica, porez);
}
```

Simboličkoj konstanti se ne može dodeliti vrednost.

Makro je definicija simbola u kojoj se javlja jedan ili više argumenata.

Primer 3

```
#include <stdio.h>
#define KVADRAT(y) y*y
void main()
{
  int a, b;
  scanf("%d", &a);
```

```
7     b = KVADRAT(a); // b = a*a;
8     printf("%d", b);
9  }
```

Napomena Program iz prethodnog primera je ispravan, ali će korišćenje tako definisanog makroa u nekim situacijama dovesti do neočekivanih rezultata.

```
1  #include <stdio.h>
2  #define KVADRAT(y) y*y
3  void main()
4  {
5     int a, b;
6     scanf("%d", &a);
7     b = KVADRAT(a+1); // b = a+1*a+1 = 2a+1 != a^2;
8     printf("%d", b);
9  }
```

Ovo se može izbeći ako se pri definiciji makroa dodaju zagrade na odgovarajućim mestima.

```
1  #include <stdio.h>
2  #define KVADRAT(y) ((y)*(y))
3  void main()
4  {
5     int a, b;
6     scanf("%d", &a);
7     b = KVADRAT(a+1); // b = ((a+1)*(a+1));
8     printf("%d", b);
9  }
```

Primer 4

```
1  #include <stdio.h>
2  #define KVADRAT(x) ((x)*(x))
3  #define KOCKA(x) (KVADRAT(x)*(x))
4  #define MAX(x,y) ((x)>(y)?(x):(y))
5  #define PRINT(x) printf("x je %d.\n", x)
6  void main()
7  {
8     int x, y, veci, zapremina, povrsina;
9     x = 50;
10    y = 100;
11    veci = MAX(x,y);
12    PRINT(veci);
13    ipovrsina = KVADRAT(veci);
14    zapremina = KOKCA(veci);
15 }
```

Definicije simboličkih konstanti i makroa se mogu ukinuti direktivnom `#undef`.

Primer 5

```
#define SIRINA 80
#define SABERI(a, b) ((a)+(b))
/* ... */
#undef SIRINA
#undef SABERI
```

7.2 Uključivanje datoteka

Preprocesor u C jeziku omogućava da se u izvorni kôd programa uključe kompletne datoteke i na taj načina znatno ubrza razvoj programa. Direktiva `#include` uključuje u izvornu datoteku takozvane datoteke zaglavlja (*header file*) koji imaju ekspenziju `.h`.

Postoje dva načina uključivanja datoteke zaglavlja u program:

`#include <ime_zaglavlja.h>` i

`#include "ime_zaglavlja.h"`.

Prvi način pretražuje sistemske direktorijume u potrazi za datotekom zaglavlja, dok drugi način pretražuje radni direktorijum u kome se nalazi program.

U zaglavljima se mogu nalaziti simbolička imena konstanti, definicije makroa i struktura i deklaracije promenljivih i funkcija.

Primer 1 Sledeći listing ilustruje uključivanje četiri standardne datoteke (biblioteke `stdio.h`, `stdlib.h`, `math.h` i `malloc.h`), kao i jednog fajla koji je kreirao sâm programer, `myfile.h`.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>
#include "myfile.h"
```

7.3 Uslovno prevođenje (kompilacija) programa

Koncept uslovnog prevođenja se često koristi u programima namenjenim za prenos na više različitih hardverskih ili softverskih formi.

U okviru C preprocesora, definisane su naredbe koje omogućavaju uključivanje i isključivanje određenih blokova naredbi tokom procesa prevođenja. Takve naredbe su preprocesorkse direktive:

- `#ifdef`,
- `#ifndef`,

- `#else`,
- `#if`,
- `#elif` i
- `#endif`.

Direktiva `#ifdef` omogućava uključenje kôda koji sledi direktivu u proces prevođenja ukoliko je identifikator `identif` prethodno definisan.

Direktiva `#else` omogućava alternativu za prethodni slučaj.

Direktiva `#ifndef identif` omogućava uključivanje kôda koji sledi direktivu u proces prevođenja ukoliko identifikator `identif` nije prethodno definisan.

Direktiva `#if izraz` omogućava uključivanje kôda koji sledi direktivu u proces prevođenja ukoliko je identifikator `izraz` različit od `0`.

Primer 1 `#if defined(ELFAK)` je ekvivalentno sa `#ifdef ELFAK`. Slično, `#if !defined(ELFAK)` je ekvivalentno sa `#ifndef ELFAK`.

Direktiva `#elif` je ekvivalentna izrazu `#else if`.

Direktiva `#endif` zatvara sve prethodno navedene `#if` direktive.

Primer 2 Jedan od čestih uslova na osnovu kojih programer bira koje će se datoteke priključiti programu jeste operativni sistem na kome se program pokreće.

```
#if SYSTEM == UNIX
  #define HDR "unix.h"
#elif SYSTEM == MSDOS
  #define HDR "dos.h"
#elif SYSTEM == MSWIN
  #define HDR "windows.h"
#endif
#include HDR
```

Primer 3 Navedene direktive se mogu koristiti i u toku izvršenja programa. Dok programer testira program, odgovara mu da "vidi" više podataka nego što bi korisnik programa mogao. U tom slučaju je korisno definisati direktivu (u donjem listingu je nazvana `PROVERA`) i sve poruke koje su namenjene programeru pisati pod uslovom da je ta direktiva definisana.

```
#include <stdio.h>
#define PROVERA
void main()
{
  float sirina, duzina, visina, povrsinaOsnove, zapremina;
  printf("Unesite␣stranice␣kvadra:␣");
  scanf("%f%f%f", &sirina, &duzina, &visina);
  povrsinaOsnove = sirina * duzina;
```

```
9     #ifdef PROVERA
10      printf("Povrsina␣osnove␣je␣%f.\n", povrsinaOsnove);
11    #endif
12    zapremina = povrsinaOsnove * visina;
13    printf("Zapremina␣kvadra␣je␣%f.", zapremina);
14  }
```

Ovakav program će na izlazu štampati površinu osnove i zapreminu kvadra.

Naravno, u slučaju velikih projekata ima mnogo međurezultata. Kada programer bude želeo da vidi kako će *korisnik* videti izlazne podatke (samo konačni rezultati), sve što treba da uradi jeste da izbriše direktivu `#define PROVERA`. U tom slučaju se deo kôda o štampanju površine osnove neće izvršiti, i na izlazu će biti ispisana samo informacija o zapremini kvadra.

7.4 Memorijske klase identifikatora

Promenljive i funkcije u C-u imaju dva atributa: **memorijsku klasu** i **tip**.

Memorijska klasa određuje lokaciju i vreme postojanja memorijskog bloka dodeljenog funkciji ili promenljivoj.

Tip određuje značenje memorisane vrednosti u memorijskom bloku.

Uobičajena podela identifikatora je na **lokalne** i **globalne**.

Lokalne promenljive postoje samo unutar određene funkcije koja ih kreira. Nepoznate su drugim funkcijama i glavnom programu.

One prestaju da postoje kada se izvrši funkcija koja ih je kreirala. Ponovo se kreiraju svaki put kada se funkcija poziva ili izvršava.

Globalnim promenljivama može pristupati svaka funkcija iz programa. Ne kreiraju se ponovo ako se funkcija ponovo poziva.

▸ **Primer 1** Sledeći jednostavan program ilustruje razliku između lokalnih i globalnih promenljivih.

```
1  #include <stdio.h>
2  int saberi_brojeve(void);
3  int vrednost1, vrednost2, vrednost3;
4  void main()
5  {
6    int rezultat;
7    vrednost1 = 10;
8    vrednost2 = 20;
9    vrednost3 = 30;
10   rezultat = saberi_brojeve();
11   printf("Suma␣%d␣+␣%d␣+␣%d␣iznosi␣%d.\n", vrednost1, vrenost2,
       vrednost3, rezultat);
12 }
13 int saberi_brojeve(void)
```

```
14 {
15   int rezultat;
16   rezultat = vrednost1 + vrednost2 + vrednost3;
17   return rezultat;
18 }
```

U drugoj liniji kôda je definisan ANSI prototip funkcije.

Treća linija koda sadrži deklaraciju tri promenljive (`vrednost1`, `vrednost2` i `vrednost3`). To su **globalne** promenljive i može im pristupiti svaka funkcija (uključujući, naravno, i funkciju `main()`).

Zatim je, u šestoj liniji, deklarisana promenljiva `rezultat` koja je lokalna za funkciju `main()`.

U petnaestoj liniji je deklarisana još jedna promenljiva sa istim imenom `rezultat`, i ona je lokalna za funkciju `saberi_brojeve`. Ova promenljiva nema nikakve veze sa istoimenom promenljivom definsanom u funkciji `main()`.

U šestanestoj liniji koda se u (lokalnu) promenljivu `rezultat` smešta zbir **globalnih** promenljivih `vrednost1`, `vrednost2` i `vrednost3`.

Dakle, kada se ova funkcija pozove (u desetoj liniji koda), ona će koristiti vrednosti `10`, `20` i `30` koje su inicijalizovane u prethodne tri linije koda. Rezultat koji vraća ta funkcija (ona vraća pomenljivu `rezultat` koja je lokalna za funkciju `saberi_brojeve`, videti 17. liniju koda) se smešta u promenljivu `rezultat` koja je lokalna za funkciju `main()`.

Nakon toga se ta promenljiva `rezultat`, lokalna za `main()`, štampa na standrni izlaz u liniji broj 11.

Generalno, u jeziku `C` postoje četiri osnovne memorijske klase:

- automatska (`auto`),
- eksterna (`extern`),
- statička (`static`) i
- registarska (`register`).

Promenljive deklarisane u funkciji ili na početku bloka su, po definiciji, **automatske**. Nastaju pri ulasku u blok ili funkciju i nestaju kada se blok ili funkcija završe. Deklaracije ovih promenljivih mogu opciono imati ključnu reč `auto`.

Primer 2 Deklaracije u sledeće dve linije koda su ekvivalentne.

```
1 int x, y;
2 auto int x, y;
```

Promenljive koje su deklarisane u jednom modulu (fajlu), a koriste se u drugom modulu suu **eksterne**. Deklaracije ovih promenljivih moraju imati ključnu reč `extern`.

Ove promenljive omogućavaju modularno programiranje, odnosno pisanje programa korišćenjem više datoteka (fajlova). Svaka globalna promenljiva je i eksterna promenljiva. Da bi se ona videla iz drugog modula, mora se koristiti `extern`.

Primer 3 U fajlu `Moduo1.c` se nalazi sledeći deo koda:

```
int spoljasnja;
void main()
{
  /* ... */
}
```

U fajlu `Moduo2.c` se nalazi sledeći deo koda:

```
int funkcija1()
{
  extern int spoljasnja;
  spoljasnja = 5;
  /* ... */
}
```

Promenljiva `spoljasnja` je deklarisana u fajlu `Moduo1.c` kao eksterna promenljiva zato što je globalna. Ona rezerviše memoriju.

U fajlu `Moduo2.c` je deklarisana promenljiva sa istim imenom (`spoljasnja`), i ona je eksterna jer je ispred tipa podatka navedena ključna reč `extern`. Ona ne rezerviše memoriju.

Promenljive koje zadržavaju vrednost kada se blok ili funkcija gde su deklarisane završe su **statičke**. Deklaracije ovih promenljivih moraju imati ključnu reč `static`.

Primer 4

```
#include <stdio.h>
void demo(void);
void demo(void)
{
  auto int autoBroj = 0;
  static int staticBroj = 0;
  printf("autoBroj = %d, staticBroj = %d.\n", autoBroj,
    staticBroj);
  ++autoBroj;
  ++staticBroj;
}
void main()
{
  int i;
  for ( i =0; i <= 2; i++ )
    demo();
}
```

Nakon izvršenja ovog programa, na izlazu će biti ispisano sledeće:

```
autoBroj = 0, staticBroj = 0
autoBroj = 0, staticBroj = 1
autoBroj = 0, staticBroj = 2
```

C jezik omogućava programeru da utiče na efikasnost programa. Naime, ako program često koristi neku promenljivu, programer može sugerisati prevodiocu da tu promenljivu smesti u brze registre centralnog procesora. Ovakve promenljive se zovu **registarske** i u deklaraciji se mora koristiti ključna reč `register`.

Naravno, prevodilac može i ignorisati ovakvu preporuku programera.

Primer 5

```
1 register char x;
```

Glava 8

Stringovi

U programskom jeziku C, string predstavlja niz znakovnih podataka (jedan ili više njih) koji se završavaju `null` znakom (`'\0'`).

Primer 6
```
"Ovo je string"
"Tekst izmedju navodnika"
```

Nad stringovima se mogu izvršavati tri operacije:

- konkatenacija,
- poređenje i
- traženje.

Konkatenacija je nadovezivanje vrednosti stringova.

Primer 7 Rezultat primene konkatencije na stringove `"Elektronski "` i `"fakultet"` je `"Elektronski fakultet"`.

Napomena Treba imati u vidu da operacija konkatenacije nadovezuje stringove takvim kakvi su. U prethodnom primeru, razmak (blanko znak) između reči *Elektronski* i *fakultet* postoji samo zato što se prvi string završava blanko znakom (isti efekat se može postići i ako drugi string počinje blanko znakom, ali ako se istovremeno i prvi završava i drugi počinje razmakom, u rezultujućem stringu će se javiti dva blanko znaka), kao što je ilustrovano u sledećem primeru.

Primer 8 Konkatenacijom stringova `"El"` i `"fak"` dobija se string `"Elfak"`. Od stringova `"El "` i `"fak"` dobija se `"El fak"`. Od stringova

`"El"` i `" fak"` se takođe dobija string `"El fak"`. Ukoliko se konkatenacija primeni na stringove `"El "` i `" fak"`, razultat je `"El  fak"` (sa dva razmaka).

Stringovi se **upoređuju** leksikografski, po engleskoj abecedi. Jedan string je manji (odnosno veći) od drugog, ako je on po leksikografskom uređenju pre (odnosno posle) drugog. Na primer, u rečniku su reči poređanje leksikografskim redosledom.

Primer 9

String `"a"` je pre stringa `"b"`;
string `"za"` je posle stringa `"az"`;
string `"abzz"` je pre stringa `"ac"`;
string `"ab"` je pre stringa `"abaaaa"`;

U C-u ne postoji poseban tip za predstavljanje znakovnih podataka, već se koristi tip `char`. `char` je mali celobrojni podatak (dužine 1 bajt) koji može da primi i kôd jednog znaka, pa se koristi i za predstavljanje znakovnih podataka.

Primer 10 Predstavljanje znakovnih nizova:

```
1 char ime[20];
2 char prezime[20];
```

`ime` i `prezime` su pokazivači na prvi znak u strungu.

Na kraju svakog znakovnog niza stoji simbol nultog znaka, koji se prikazuje kao znakovna konstanta sa vrednošću `'\0'` (poznata kao *null terminated symbol* ili *null character*).

String se može inicijalizovati na dva načina:

- listom individualnih znakova (karaktera) ili
- navođenjem konstantne vrednosti za niz znakova.

Primer 11 U sledećem listingu obe linije rade istu stvar.

```
1 char ime[] = {'E', 'l', 'f', 'a', 'k', '\0'};
2 char ime[] = "Elfak";
```

Primetimo da u oba slučaja nisu dati podaci o veličini niza. Nju određuje prevodilac.

Napomena Kod prvog načina (pomoću individualnih znakova) se **mora** navesti oznaka `'\0'`. U suprotnom se dobija običan niz znakova, a ne string (videti definiciju stirnga na početku glave).

8.1 Ulaz/izlaz sa stringovima

Podsetimo se značenja tipova konverzija `c` i `s` prilikom korišćenja fukcija `scanf()` i `printf()` iz biblioteke `stdio.h`:

> `c` – znakovna klonverzija (rezultat je tipa `char`);
>
> `s` – konverzija u znakovni niz (niz znakova između dva blanko znaka, što znači da učitani niz ne sadrži blanko znake; iza pročitanih znakova se dodaje `'\0'`.

Konverzija `s` označava da se prilikom učitavnja podataka prenosi u operativnu memoriju računara u onom obliku kako je sa tastature prihvaćen (kao niz ASCII simbola), jedino se na kraj tog niza dodaje simbol `'\0'`.

U pozivu funkcije `scanf()` navode se adrese memorijskih lokacija gde će pročitani podaci biti upisani, a imena nizova su ujedno i memorijske adrese prvih članova niza. Zato je pri učitavanju niza dovoljno navesti samo ime niza (bez adresnog operatora `&`).

Konverzija `s` se koristi na isti način i pri pozivu funkcije `printf()`. Na standardni izlaz se prenosi niz karaktera iz operativne memorijse od prvog elementa u navedenom nizu do simbola `'\0'`.

Primer 1

```
char ime[20], prezime[30];
scanf("%s␣%s", ime, prezime);
printf("%s␣%s\n", ime, prezime);
```

C nema dobru podršku za rad sa stringovima. Stringovi praktično ne postoje – oni se predstavljaju kao polje karaktera.

Primer 2 Poslednje tri linije sledećeg isečka koda nisu ispravne. Ove naredbe nisu dozvoljene u C-u (jer je string samo niz karaktera).

```
char ime[50];
char prezime[50];
char punoIme[100];
ime = "Arnold";              /* nije dozovljeno */
prezime = "schwarzenegger";  /* nije dozvoljeno */
punoIme = ime + prezime;     /* nije dozvoljeno */
```

U C-u je, dakle, nemoguće:

dodeljivanje vrednosti jednog stringa drugom (`s1 = s2`);
upoređivanje stringova (`s1 < s2`);
izvršiti konkatenaciju stringova (`s1 + s2`);
formirati funkciju koja kao rezultat vraća string.

8.2 Biblioteka `string.h`

Da bi se olakšao rad sa stringovima, treba uključiti biblioteku `string.h`, odnosno navesti `#include <stdio.h>` na početku programa.

8.2.1 Spisak osnovnih funkcija

Kopiranje (dodela vrednosti stringu):

- `char *strcpy(const char *string1, const char *string 2)`
  Kopira `string1` u `string2`, uključujući oznaku kraja stringa.
- `char *strncpy(const char *string1, const char *string 2, size_t n)`
  Kopira prvih `n` karaktera iz stringa `string2` u `string1`.

Konkatenacija (spajanje stringova u jedan):

- `char *strcat(const char *string1, const char *string 2)`
  Dodaje `string2` iza stringa `string1`.
- `char *strncat(const char *string1, const char *string 2, size_t n)`
  Dodaje `n` karaktera iz stringa `string2` u `string1`.

Funkcije poređenja:

- `char *strcmp(const char *string1, const char *string 2)`
  Upoređuje `string1` i `string2` za određivanje leksičkog redosleda.
- `char *strncmp(const char *string1, const char *string 2, size_t n)`
  Upoređuje (leksički) prvih `n` karaktera dva stringa. Kao i `strcmp()`, vraća `0` ako su stringovi jednaki, vrednosti manju od nule ako `string1` prethodi stringu `string2`, u suprotnm vraća vrednost veću od nule (ako `string2` prethodi stringu `string1`).
- `char *strcasecmp(const char *string1, const char *string 2)`
  Case inseneitive verzija za `strcmp()`.
- `char *strncasecmp(const char *string1, const char *string 2, size_t n)`
  Case insensitive verzija za `strncmp()`.

Ostale funkcije:

- `char *strerror(int errnum)`
  Poruka o grešci za zadati broj greške.
- `int strlen(const char *string)`
  Vraća dužinu stringa `string`.

Primer 1 Sledeći program određuje koliko slova ima uneta reč.

```
#include <stdio.h>
#include <string.h>
void main()
{
  int duzina;
  char rec[100];
  scanf("%s", rec);
  duzina = strlen(rec);
  printf("Uneta rec \"%s\" ima %d slova.\n", rec, duzina);
}
```

Napomena Funkcija `strlen()` vraća dužinu bez oznake kraja.

Primer 2 Pomoću funkcije `strcpy()` je moguće:

- upisati tekst u string, tj. inicijalizovati ga (kao što je učinjeno u petoj liniji koda),
- kopirati ceo sadržaj jednog stringa u drugi (šesta linija) i
- kopirati ostatak jednog stringa (počev od nekog karaktera) u drugi (sedma linija).

```
#include <stdio.h>
#include <string.h>
void main()
{
  char s1[100], s2[100], s3[100], s4[100];
  strcpy(s1, "Ovo je Elektronski fakultet");
  strcpy(s2, "Elfak");
  strcpy(s3, s2);
  strcpy(s4,  &s1[7]);
  printf("s1: %s\ns2: %s\ns3: %s\ns4: %s\n", s1, s2, s3, s4);
}
```

Navedeni program štampa na ekran:

```
s1: Ovo je Elektronski fakultet
s2: Elfak
s3: Elfak
s4: Elektronski fakultet
```

Primer 3

```
#include <stdio.h>
#include <string.h>
void main()
{
  char s1[100], s2[100], s3[100], s4[100];
  strcpy(s1, "Elektronski fakultet ELFAK");
  strncpy(s2, s1, 11);
```

```
8     s2[11] = 0;
9     strncpy(s3, &s1[21], 6);
10    s3[6] = 0;
11    strncpy(s4,  &s1[12], 8);
12    s4[8] = 0;
13    printf("s1:␣%s\ns2:␣%s\ns3:␣%s\ns4:␣%s\n", s1, s2, s3, s4);
14 }
```

Nakon izvršenja navedenog programa, na ekran će biti ispisano:

```
s1: Elektronski fakultet ELFAK
s2: Elektronski
s3: ELFAK
s4: fakultet
```

Napomena Nepažljivo korišćenje funkcije `strncpy()` može dovesti do neočekivanih rezultata.

```
1  #include <stdio.h>
2  #include <string.h>
3  void main()
4  {
5     char s2[15], s1[30];
6     strcpy(s1, "Elektronski␣fakultet␣ELFAK");
7     strncpy(s2, s1, 11);
8     printf("s1:␣%s\ns2:␣%s\n", s1, s2);
9  }
```

Kako se string `str2` ne završava *null-terminated* simbolom, na ekran se štampa sledeće (ili nešto tome slično):

```
s1: Elektronski fakultet ELFAK
s2: Elektronski     Đ~>ś˙8
```

Primer 4

```
1  #include <stdio.h>
2  #include <string.h>
3  void main()
4  {
5     char s1[10] = "abc";
6     char s2[10] = "aab";
7     int diff;
8     diff = strcmp(s1, s2);
9     printf("%d␣␣", diff);
10    diff = strcmp(s2, s1);
11    printf("%d␣␣", diff);
12    diff = strcmp(s1, s1);
13    printf("%d␣␣", diff);
14    diff = strcmp(&s1[1], &s2[1]);
15    printf("%d␣␣", diff);
16    diff = strncmp(s1, s2, 1);
17    printf("%d", diff);
18 }
```

Na izlazu će se javiti sledeći podaci:

```
1  -1  0  1  0
```

8.2.2 Funkcije za traženje

- `char *strchr(const char *string, int c)`
  Nalazi prvo pojavljivanje karaktera u stringu.
- `char *strrchr(const char *string, int c)`
  Pronalazi poslednje pojavljivanje karaktera `c` u stringu.
- `char *strstr(const char *s1, const char *s2)`
  Locira prvo pojavljivanje stringa `s2` u stringu `s1`.
- `char *strpbrk(const char *s1, const char *s2)`
  Vraća pointer na prvo pojavljivanje u stringu `s1` nekog karaktera iz stringa `s2`, ili `null` pointer ako nema takvog karaktera.
- `size_t strspn(const char *s1, const char *s2)`
  Vraća broj karaktera na početku `s1` koji se poklapaju sa `s2`.
- `size_t strcspn(const char *s1, const char *s2)`
  Vraća broj karaktera na početku `s1` koji se **ne** poklapaju sa `s2`.
- `char *strtok(char *s1, const char *s2)`
  Deli string `s1` u sekvencu tokena, svaki od njih je ograničen jednim ili više karaktera iz stringa `s2`.
- `char *strtok_r(char *s1, const char *s2, char **lasts)`
  Kao `strtok()`, osim što pointer na string mora biti zadat od strane pozivne funkcije.

Primer 5

```
1 char *str1 = "Hello";
2 char *ans;
3 and = strchr(str1, 'l');
```

Nakon izvršenja ovog isečka koda, `ans` pokazuje na lokaciju `str1 + 2`.

Primer 6 Generalnija funkcija od `strch()` je `strpbrk()`. Ona traži prvo pojavljivanje bilo koje grupe karaktera.

```
1 char *str1 = "Hello";
2 char *ans;
3 ans = strpbrk(str1, 'aeiou');
```

Posle izvršenja treće linije koda, `ans` će pokazivati na lokaciju `str1 + 1`, tj. na lokaciju slova `e` iz stringa `Hello` (jer je `e` prvo slovo u `Hello` koje se nalazi i u `aeiou`).

Primer 7 Funkcija `strstr()` vraća pointer na specificirani string za traženje, ili `null` pointer ako taj string nije nađen. Ako `s2` ukazuje na string dužine nula (odnosno string), funkcija vraća `s1`.

```
char *str1 = "Hello";
char *ans;
ans = strstr(str1, 'lo');
```

Ovde će `ans` pokazivati na `str1 + 3`.

Primer 8 Korišćenje nekih navedenih funkcija za traženje.

```
#include <stdio.h>
#include <string.h>
void main()
{
  char linija[100], *deoTeksta;
  strcpy(linija, "Zdravo, ja sam string.");
  printf("Linija: %s\n", linija);
  strcat(linija, " Ko si ti?");
  printf("Linija: %s\n", linija);
  printf("Duzina linije: %d\n", (int) strlen(linija));
  if ( (deoTeksta = strchr(linija, 'K')) != NULL )
    printf("String koji pocinje sa \'K\': %s\n", deoTeksta);
}
```

Na izlazu će biti ispisano sledeće.

```
Linija: Zdravo, ja sam string.
Linija: Zdravo, ja sam string. Ko si ti?
Duzina linije: 32
String koji pocinje sa 'K': Ko si ti?
```

Primer 9 Realizacija i primena funkcije za određivanje dužine stringa (bez korišćenja funkcija iz biblioteke `string.h`).

```
#include <stdio.h>
#define DIM 100
int duzinaStringa(char str[])
{
  int n = 0;
  while ( str[n] != '\0' )
    n++;
  return n;
}
void main()
{
  char str[DIM];
  printf("Unesite string: ");
  scanf("%s", str);
  printf("Duzina unetog stringa je %d.", duzinaStringa(str));
}
```

Primer 10 Sledeći program nalazi najkraću reč u stringu.

Učitava se reč po reč sa tastature, računa se njihova dužina i poredi se sa dužinom do tada najkraće unete reči. Ukoliko je dužina tekuće reči manja od dužine najkraće, tekuća reč će se kopirati u najkraću reč. Na početku će se za dužinu najkraće reči uzeti vrednost veća od maksimalne moguće dužine reči.

Program je realizovan bez korišćenja bibliotečkih funkcija iz fajla `string.h`. U komentarima je naveden gde se delovi koda mogu zameniti odgovarjućim funkcijama iz ove biblioteke.

```
#include <stdio.h>
void main()
{
  char rec[20], minRec[20];
  int i, j, n, duzina, minDuzina;
  printf("Unesite broj reci: ");
  scanf("%d", &n);
  minDuzina = 20;
  for ( i = 0; i < n; i++ )
  {
    printf("Unesite rec br. %d: ", i+1);
    scanf("%s", rec);
    for ( duzina = 0; rec[duzina]!=0; duzina++ ); // strlen
    if ( duzina < minDuzina )
    {
      minDuzina = duzina;
      j = 0;
      do                          //
        minRec[j] = rec[j];       // strcpy
      while ( rec[j++] != 0 ); //
    }
  }
  printf("Najkraca rec je: %s", minRec);
}
```

Primer 11 Štampanje stringa izjedna (koristeći `%s` konverziju) i kao polje karatekra.

```
#include <stdio.h>
#include <string.h>
#define DIM 80
void main()
{
  char str[DIM];
  int duzina, i;
  str[0] = 'a'; str[1] = 'b'; str[2] = 'c';
  str[3] = 'd'; str[4] = 'e'; str[5] = 'f';
  str[6] = '0'; str[7] = 0;
  duzina = strlen(str);
  printf("str: %s\n", str);
  printf("duzina: %d\n", duzina);
  printf("\nString unapred: \n");
  for ( i = 0; i < duzina; i++ )
    printf("str[%d] = %c\n", i, str[i]);
```

```
17    printf("\nString unazad: \n");
18    for ( i = duzina; i >= 0; i-- )
19      printf("str[%d] = %c\n", i, str[i]);
20  }
```

Glava 9

Datoteke (fajlovi)

Programi čitaju podatke sa spoljašnjih uređaja i ispisuju podatke na druge spoljašnje uređaje. `printf()`, odnosno `scanf()`, piše, odnosno čita, podatke na ekran, odnosno sa tastature – to je prolazni ulaz/izlaz.

Trajni ulaz/izlaz omogućavaju datoteke (fajlovi, *files*). Standardni `C` ne smatra ulazno-izlazne operacije svojim delom: postoje mnoge razlike (na primer, operativni sistem), ali zato postoje biblioteke sa funkcijama i makroima za ulaz/izlaz.

Postoje dva nivoa bibliotekâ `C` funkcijâ:

- ulaz/izlaz na nivou toka (viši nivo) – skrivaju se detalji fizičkih uređaja; lakša prenosivost na druga radna okruženja; baferovanje se vrši automatski i
- ulaz/izlaz na sistemskom nivou (niži nivo) – blisko vezan sa detaljima konkretne implementacije; za baferovanje je odgovoran programer.

Ulaz/izlaz na nivou toka:

standardni tok (*steam*) – model toka je izveden iz UNIX radnog okruženja; `C` standard prepoznaje oba tipa tokova (tekstualni i binarni blok); postoje UNIX-ov i DOS-ov model standardnih tokova.

- UNIX-ov model – postoje tri standardna toka:

  `stdin` – standardni ulaz,

  `stdout` – standardni izlaz i

  `stderr` – standardni tok za greške;

- DOS-ov model – postoje tri standardna toka (isti kao kod UNIX-ovog modela) i još dva dodatna:

  `stdprn` – standardni priključak na štampač i

  `stdaux` – standardni priključaj za serijsku komunikaciju.

`stdin` se standardno vezuje za tastaturu. On baferuje u redove, što znači da proces ne prima ništa sve do pritiska tastera *Enter*.

`stdout` je, standardno, korisnikov ekran. On je befarenovan (da bi se izbeglo zadržavanje poruka o greškama u baferu).

Fajl ili **datoteka** je skup bajtova kome je dodeljeno neko ime. Čuvaju se na nekom spoljenjem uređaju (disk, traka ili slično). U fajlovima se podaci čuvaju trajnije nego na standardni ulaz/izlaz, a omogućen im je brz i lak pristup.

Da bi se neki spoljašnji fajl pridružio u `C` programu, mora mu se pridružiti neki **tok**. Ovo podrazumeva

- deklarisanje promenljive zvane **pokazivač na fajl** i
- davanje vrednosti toji promenljivoj.

Vrednost koja se dodeljuje dobija se pozivom funkcije koja pokušava da otvori specificirani fajl.

Gledano sa strane koristnika, fajl ima svoje ime i (verovatno) neki sadržaj. Sa strane programera, fajl je **tok bajtova** kome se pristupa preko pokazivača na fajl.

Svim operacijama koje pristupaju sadržaju fajla ili koje obavljaju određene operacije nad fajlovima pristupa se preko pokazivača na fajl.

`stdio.h` je standardna biblioteka koja sadrži tipove i funkcije za ulaz i izlaz sa fajlovima. Ako je trebno koristiti ih, ona se mora priključiti preprocesorskom direktivom `#include <stdio.h>`.

U okviru biblioteke `stdio.h` definisan je tip **pointer na fajl**: `FILE *`. Ovaj tip se može shvatiti kao apstraktni tip podataka kome se pristupa preko `C` fukcija iz biblioteke.

> **Napomena** `FILE` predstavlja neku strukturu koja sadrži informacije o fajlu. Pri tome se mora koristiti pointer `FILE *`, pošto neke funkcije menjaju sadržaj nekih elemenata strukture.

Što se funkcija tiče, rad sa fajlovima (čitanje ili upis) zahteva tri koraka:

- otvaranje datoteke (pristup),
- čitanje ili upis i
- zatvaranje datoteke.

9.1 Predefinisani tokovi, tipovi i vrednosti

- `FILE` – tip podataka (struktura) koja čuva sve informacije o fajlu (koje se koriste, na primer, kod otvaranja fajl);
- `FILE *stdin` – `stdin` je povezan sa tokom za standardni ulaz za podatke;
- `FILE *stdout` – analogno, `stdout` je povezan sa standardnim tokom koji se koristi za izlazne podatke programa;
- `FILE *stderr` – `stderr` je povezan sa tokom za greške;

- `EOF` – vrednost koja označava kraj fajla (***E**nd **O**f **F**ile*); za ANSI `C` to je negativna celobrojna konstanta, čija je vrednost tradicionalno i uglavnom `-1`.
- `NULL` – vrednost nultog pointera (konstanta `0`);
- `BUFSIZ` – celobrojna konstanta (`int`) koja specificira "odgovarajuću" veličinu bafera preko kojih se vrši ulaz/izlaz za fajlove;
- `size_t` – `unsigned` tip podatka čija je veličina takva da može da čuva bilo koju vrednost koju može da vrati `sizeof`.

Primer 1 **Standardni ulazni tok** se koristi kada se navodi `scanf()`. Drugim rečima, sledeće dve linije koda su ekvivalentne.

```
scanf("%d", &vrednost);
fscanf(stdin, "%d", &vrednost);
```

Slično je i sa funkcijom za ispisivanje na ekran (standardni izlaz).

```
printf("%d", vrednost);
fprintf(stdout, "%d", vrednost);
```

Napomena Standardni ulazni tok je povezan sa tastaturom, a standardni izlazni tok sa ekranom, osim ako nije urađena redirekcija.

Primer 2 Standardni tok za greške omogućava da se na njega šalju greške.

```
fprintf(stderr, "Ne mogu da otvorim ulazni fajl in.list!\n");
```

Napomena Tok za greške je asociran sa standardnim izlaznim tokom. Međutim, ukoliko je izvršena redirekcija za izlazni tok, to ne važi za tok za greške.

9.2 Funkcije za rad sa fajlovima

Za **otvaranje fajlova** mogu se koristiti dve funkcije:

- `fopen` i
- `freopen`.

`fopen()` se koristi na sledeći način:

pri čemu je:

⟨ime fajla⟩ – string koji sadrži ime fajla na disku (uključujući i putanju);
⟨mod⟩ – string koji predstavlja način otvaranja, tj. pristupa fajlu.

```
FILE *fopen(const char *path, const char *mode);
```

pri čemu:

`fopen` otvara fajl sa navedenim imenom (string `*path` koji sadrži, pored imena, i putanju do fajla), i asocira ga sa odgovarajućim tokom (*steam*);
`mode` – string koji predstavlja način otvaranja, tj. pristupa fajlu; može imati sledeće vrednosti:

- `r` – otvara postojeći fajl za čitanje, počinje čitanje od početka fajla;
- `w` – kreira novi fajl i otvara ga za upis, od početka fajla (ako fajl već postoji, briše sav sadržaj fajla);
- `a` – otvara ili kreira fajl za dodavanje na kraj tekstualnog fajla;
- `r+` – otvara fajl za čitanje i upis, počev od startne pozicije (ažuriranje);
- `w+` – isto kao `w`, s tim da se fajl otvara za čitanje i upis (ažuriranje);
- `a+` – isto kao `a`, s tim da se fajl otvara za čitanje i upis (ažuriranje);
- može da sadrži i `b` kao drugi ili treći karakter, s namerom da označi da se radi o binarnom fajlu;
- može sadržati i druge karaktere koji se mogu koristiti u procesu implementacije.

Napomena Ako se fajl otvara za ažuriranje (modovi sa znakom `+`), operacija izlaza (upisa) ne može da sledi iza operacije ulaza (čitanja) bez operacije flašinga (*flushing*) za bafer (`fflush()`) ili repozicioniranja (`fseek()`, `fsetpost`, `rewind`).

Takođe, operacija čitanja ne može da sledi iza operacije upisa bez flašinga bafera ili repozicioniranja, osim ako se nije stiglo do kraja fajla.

`fopen()` vraća `FILE *` koji se koristi za pristup fajlu. Ukoliko fajl ne može da se otvori iz nekog razloga (privilegije, fajl ne postoji i sl.), `fopen()` vraća `NULL` i setuje se `errno`.

Funkcija `freopen()` je definisana na sledeći način.

```
FILE *freopen(const char *pathname, const char *mode, FILE *stream)
```

`freopen()` radi kao `open()` s tim da asocira fajl sa nekim tokom, umesto da kreira novi. Koristi se primarno da se fajl asocira sa nekim standardnim tokom za tekstualne podatke (`stdin`, `stdout` ili `stderr`).

Primer 1

```
FILE *ifp, *ofp;
char *mode = "r";
char outputFilename[] = "out.list";
ifp = fopen("in.list", mode);
if ( ifp == NULL )
{
  fprintf(stderr, "Ne mogu da otvorim ulazni fajl in.list!\n");
  exit(1);
}
ofp = fopen(outputFilename, "w");
if ( ofp == NULL )
{
  fprintf(stderr, "Ne mogu da otvorim izlazni fajl %s!\n",
    outputFilename);
  exit(1);
}
```

Napomena Fajl koji se otvara za čitanje (`r`) mora da postoji.

Kod upisa (`w`), ako program pokušava da otvori fajl koji ne postoji, kreiraće novi sa zadatim imenom. Ako takav fajl, međutim, postoji, njegov kompletan sadržaj će biti izbrisan.

Nakon uspešnog otvaranja fajla moguće je:

čitati fajl korišćenjem `fscanf()` i
upisivanje u fajl korišćenjem `fprintf()`.

Ove funkcije rade na isti način kao i `printf()` i `scanf()`, osim što zahtevaju dodatni parametar `FILE *` koji specificira fajl u koji se piše ili iz kojeg se čitaju podaci.

Funkcija `fscanf()` (kao i `scanf()`) vraća broj vrednosti koje može da pročita. Ako dođe do kraja fajla, vraća specijalnu vrednost `EOF`. Obrada celog fajla može da se obavlja u okviru neke petlje, gde je uslov za izlazak dosezanje `EOF`-a.

Međutim, šta će se dogoditi ako se dođe do greške u formatu podatka, pa se, na primer, umesto očekivanog slova pročita broj? U tom slučaj, funkcija `fscanf()` ne može da pročita tu liniju i samim tim ne prelazi u sledeću – u ovom slučaju `fscanf()` nikada neće vratiti `EOF`. Zbog ovakvih greški se ne može pročitati ostatak fajla i program ulazi u beskonačnu petlju.

Primer 2 U datoteci se nalaze prezimena studenata (maksimalne dužine 20) i broj poena sa ispita, odvojeni jednim blanko znakom. Treba napisati program koji formira novi fajl identičnog formata, s tim što uvećava broj poena za 10.

⟨Primer fajla:⟩
```
Smit 98
```

```
Džonson 70
Vilijams 100
Braun 40
```

```
char prezime[21];
int rezultat;
/* ... */
while ( fscanf(ifp, "%s␣%d", ime, &rezultat) != EOF )
  fprintf(ofp, "%s␣%d\n", ime, rezultat+10)
/* ... */
```

Međutim, ako se navedeni kod izvrši nad sledećim fajlom, program će ući u beskonačnu petlju jer nikad neće stići do kraja fajla.

⟨Primer fajla sa greškom u formatu:⟩

```
Smit 98
Džonson 70
Vilijams A+
Braun 40
```

Jedino rešenje problema kod čitanja podataka pogrešnog formata je testiranje broja vrednosti koje treba da se pričitaju sa `fscanf()`. Pošto je navedeni format `"%s␣%d"`, očekuje se čitanje dve vrednosti, uslov provere bi bio:

```
while ( fscanf(ifp, "%s␣%d", ime, &rezultat) == 2 )
```

Prolazak kroz fajl se može realizovati i korišćenjem funkcije `feof()` iz biblioteke `stdio.h`. Ova funkcija kao argument ima pointer na fajl, a vraća `true` ili `false` u zavisnosti od toga da li je dosegnut kraj fajla ili ne.

Napomena Kao i kod testiranja `!= EOF`, ova funkcija može prourokovati beskonačnu petlju ako je format ulaznog podatka iz fajla neodgovarajući. Zato treba dodati i kod za proveru da li su pročitane dve vrednosti.

Primer 3

```
while ( !feof(ifp) )
{
  if ( fscanf(ifp, "%s␣%d", ime, &rezultat) != 2 )
    break;
  fprintf(ofp, "%s␣%d", ime, rezultat+10);
}
```

9.3 Spisak funkcija za čitanje i upis

- Čitanje karaktera iz fajla:
  - *int* `fgetc(`*FILE* `*stream)`

Čita sledeći karakter iz ulaznog toka i vraća ga kao `int` (kôd karktera).

  - `int getc(FILE *stream)`

    Radi kao `fgetc` s tim da je obično implementiran kao makro.

  - `int getchar(void)`

    Čita sledeći karakter sa `stdin` (obično implementiran kao `getc(stdin)`, zapravo je makro).

- Upis karaktera u fajl:

  - `int fputc(int c, FILE *stream)`

    Upisuje `c` na izlazni tok kao `unsigned char` i vraća karkater kao `int` (ako dođe do greške, vraća se `EOF` i setuje se `errno`).

  - `int putc(int c, FILE *stream)`

    Identična funkcija kao `fputc` ali implementirana kao makro.

  - `int putchar(int c)`

    Upisuje `c` na `stdout`, implementirana kao `putc(stdout)` (makro).

- Čitanje stringa iz fajla:

  - `char *fgets(char *s, int n, FILE *stream)`

    Čita karaktere iz toka `stream` i smešta ih u string na koji ukazuje `s`;

    čitanje se prekida kod `newline` karaktera, dostizanja kraja fajla ili je pročitano $n-1$ karaktera, i oznaka kraja stringa (`'\0'`) je dodata stringu `c` (nakon svakog `newline` karaktera);

    ako je dosegnut kraj fajla, bez pročitanog karaktera, `fgets` vraća `NULL` i sadržaj stringa `s` ostaje nepromenjen;

    ako dođe do greške tokom čitanja, vbraća `NULL` i sadržaj stringa `s` je nedefinisan;

    inače vraća `s` (pointer na pročitani string).

  - `*gets(char *s, FILE *stream)`

    Slično kao `fgets`, s tim što ne pamti `newline` karakter i podrazumeva da je `s` neodređeno velik string, što može izazvati dosta problema.

- Upis stringa u fajl:

  - `int fputs(const char *s, FILE *stream)`

    Upisuje *null-terminated* string `s` na izlazni tok `steam` i ako dođe do greške vraća `EOF`, inače vraća nenegativni ceo broj.

  - `int puts(const char *s)`

    Upisuje *null-terminated* string `s`, iza koga sledi `newline` karakter, na standardni izlazni tok `stdout`;

- Čitanje iz binarne datoteke:

  - `size_t fread(void *ptr, size_t siz, size_t num, FILE *stream)`

    Čita najviše `num` objekata, gde je svaki veličine `siz` bajtova, sa ulaznog toga `stream` i smešta ih u memorijsku lokaciju na koju ukazuje pokazivač `ptr`;

    vraća broj pročitanih objekata, a ako dođe do greške vraća nulu;

    ako dođe do kraja fajla, vrednost koju vrati biće manja od `num` (može biti i nula), pri čemu se mogu koristiti `feof` ili `ferror` da bi se video razlog greške.

- Upis u binarni fajl:
  - `size_t fwrite(const void *ptr, size_t siz, size_t num, FILE *stream)`

    Upisuje najviše `num` objekata, gde je svaki veličine `siz` bajtova, iz memorijske lokacije na koju ukazuje `ptr` na izlazni tok `stream`;

    vbraća broj upisanih objekata, a ako dođe do greške, vraća nulu.

- Upis formatiranog izlaza:
  - `printf(const char *format, ...)`

    Piše na standardni izlazni tok `stdout`.
  - `fprintf(FILE *stream, const char *format, ...)`

    Piše na izlazni tok `stream`.
  - `sprintf(const char *str, const char *format, ...)`

    Piše svoj izraz u string `str` (završen oznakom kraja `'\0'`).

  Sve tri funkcije vraćaju broj upisanih karaktera (bez `'\0'` za `sprintf`)

- Čitanje formatiranog ulaza:
  - `int scanf(const char *format, ...)`
  - `int fscanf(FILE *stream, const char *format, ...)`
  - `int sscanf(const char *str, const char *format, ...)`

- Provera statusa fajla:
  - `int feof(FILE *stream)`

    Proverava indikator kraja fajla za navedeni tok `stream` i vraća broj različit od nule ako je setovan. Oznaka kraja fajla se nalazi iza poslednjeg karaktera u fajlu.
  - `int ferror(FILE *stream)`

    Proverava indikator greške za tok `stream` i vraća broj različit od nule ako je setovan.
  - `void clearerr(FILE *stream)`

    Briše vrednost indikatora za kraj fajla i indikatora za greške za navedeni tok `stream`. (Kada se jednom setuju ova dva indikatora, oni se ne resutuju sve do poziva `clearerr`, osim ako se repozicioniranjem ne obriše oznaka kraja fajla, tj. vrednost odgovarajućeg indikatora.)

– `int fgetpos(FILE *stream, fpos_t *pos);`

  Smešta vrednost tekuće pozicije indikatora za tok `stream` u `pos`;

  `pos` je *implementation-defined* tip koji može da ima i kompleksnu strukturu;

  ako dođe do greške, vraća broj veći od nule i setuje se `errno`.

– `int fsetpos(FILE *stream, fpos_t *pos);`

  Postavlja infdikator pozicije u fajlu za `stream` na poziciju definisanu u `pos`;

  ako dođe do greške, vraća broj veći od nule i setuje se `errno`;

  kod uspešnog izvršenja, indikator *end-of-file* se briše.

– `void rewind(FILE *stream)`

  Postavlja indikator pozicije na početak fajla.

– `int fseek(FILE *stream, long offset, int whence`

  Postavlja indikator pozicije za `stream`. Nova pozicija se određuje dodavanjem vrednosti pomeraja `offset` u odnosu na poziciju zadatu u `whence`:

  · ako je `whence` postavljeno na `SEEK_CUR`, pomeraj se računa u odnosu na tekuću poziciju u fajlu;

  · ako je `whence` postavljeno na `SEEK_SET`, pomeraj se računa u odnosu na početak fajla;

  · ako je `whence` postavljeno na `SEEK_END`, pomeraj se računa u odnosu na kraj fajla.

  Obično se primenljuje na binarne fajlove.

– `long ftell(FILE *stream)`

  Vrača tekuću poziciju u fajlu sa `stream`.

  Za binarne fajlove, vrednost koju vrati je broj bajtova od početka fajla do tekuće pozicije;

  za tekstualne fajlove, vrednost je *implementation-defined*.

> ⚠ **Napomena** Fajl mora biti zatvoren nakon korišćenja. Zatvaranje fajlova je veoma važno, naročito za izlazne fajlove.
>
> Razlog ovome je činjenica da je izlaz baferovan. U C-u, ako se nešto upisuje u fajl, ne znači da će se navedeni sadržaj upisati u fajl na disku računara, već da će taj sadržaj završiti u baferu u memoriji koja se koristi za pristup fajlul.
>
> Taj bafer čuva tekst privremeno, do zatvaranja fajla. Tek se nakon zatvaranja fajla sadržaj bafera upisuje na disk.

Funkcija za zatvaranje fajla je `fclose()`.

```
int fclose(FILE *stream)
```

Baferovani sadržaj se upisuje na disk i zatvara se tok `stream`. Svako naknadno korišćenje istog toka `steam` (bez prethodnog ponovnog otvaranja pomoću `freopen`) će prouzrokovati grešku. Ako je operacija zatvaranja uspešna , `fclose` vraća `0`, inače vraća `EOF` i postavlja se kôd greške u `errno`.

Primer 1 U prethodnom primeru treba dodati sledeće dve linije koda da bi se oba fajla zatvorila.

```
1 fclose(ifp);
2 fclose(ofp);
```

Sadržaj