1
UKRATKO O PROGRAMSKOM JEZIKU C

Programski jezik C je nastao sedamdesetih godina. Prethodilo mu je nekoliko dosta sličnih programskih jezika, ali se u praksi samo on zadržao. U toku razvoja bilo je postavljeno više zahteva, koje je novi jezik trebalo da zadovolji, ali su najbitniji sledeći:

- razviti viši programski jezik koji raspolaže barem nekim mogućnostima mašinskog jezika (direktan pristup memoriji, veći skup definisanih operatora itd.);
- razviti programski jezik sa vrlo malim skupom naredbi i relativno velikom bibliotekom funkcija.

Jezik koji zadovoljava prvo svojstvo bi bio udoban za rad, ali bi imao izražajne mogućnosti mašinskog jezika, što za većinu prethodnih viših programskih jezika nije bilo karakteristično.

Jezik koji zadovoljava drugo svojstvo ima mali skup naredbi i time je pojednostavljen prevodilac.

1.1 Azbuka programskog jezika C

Kao svaki programski jezik, i programski jezik C ima azbuku koja se sastoji od:

- velikih slova engleske abecede;
- malih slova engleske abecede;
- cifara dekadnog brojnog sistema;
- specijalnih znakova:

```
( ) + –  * ~ ! # % ^ & _ | = ' " : ; > < , . / \ ?
```

1.2 Leksičke konstrukcije u programskom jeziku C

Kombinovanjem znakova iz azbuke po određenim sintaksnim pravilima dobijamo leksičke konstrukcije u programskom jeziku. Te konstrukcije delimo na:

- proste i
- složene.

1.3 Proste leksičke konstrukcije

U proste se ubrajaju:

- imena,
- konstante,

dok složene konstrukcije čine:

- izrazi,
- naredbe,
- funkcije.

1.3.1 Imena

Imena čine reči koje se sastoje od:

- slova,
- cifara i
- specijalnog znaka _

tako da:

- prvi znak ne može biti cifra i
- ime može imati najviše 31 znak.

Napomenimo da se mala i velika slova razlikuju. Tako reči ABC i abc predstavljaju različita imena.

Imena služe da se identifikuju (tj. imenuju):

- promenljive,
- simboličke konstante,
- obeležja,
- funkcije.

1.3.2 Konstante

Konstante se dele na

- celobrojne,
- mešovite (razlomljene) i
- znakovne.

1.3.2.1 Celobrojne konstante

Celobrojna konstanta može biti zapisana u jednom od tri brojna sistema:

- dekadnom (osnova 10),
- oktalnom (osnova 8) ili
- heksadekadnom (osnova 16).

Ispred konstante može biti naveden znak – ili + (obično konstante u dekadnom sistemu mogu biti označene).

Konstanta u dekadnom brojnom sistemu se sastoji od cifara dekadnog brojnog sistema.

Konstantu u oktalnom sistemu čini niz cifara iz skupa {0, 1, 2, 3, 4, 5, 6, 7} pri čemu prva cifra mora biti 0 (to je indikator da je konstanta u oktalnom sistemu).

Konstanta u heksadekadnom sistemu se sastoji od prefiksa *0x* iza koga sledi niz cifara iz skupa {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}. Znakovi A, B, C, D, E i F odgovaraju ciframa koje imaju, redom, vrednost 10, 11, 12, 13, 14 i 15 u dekadnom sistemu. Umesto velikih slova mogu biti navedena odgovarajuća mala slova (a, b, c, d, e ili f).

1.3.2.2 Mešovite konstante

Mešovite brojne konstante se zapisuju u dekadnom brojnom zapisu. Mogu imati znak + ili – (kaže se da su označene), koji se navodi kao prvi simbol u konstanti. Iza toga sledi mešovita konstanta bez znaka. Mešovita konstanta bez znaka ima dva moguća oblika i javlja se kao:

- mešovita konstanta bez eksponenta i
- mešovita konstanta sa eksponentom.

Mešovita konstanta bez eksponenta se sastoji od niza cifara koji čini celobrojni deo, decimalne tačke (.) i niza cifara koji čini decimalni deo.

Mešovita konstanta sa eksponentom se sastoji od mešovite konstante bez eksponenta za kojom slede slovo e (ili E) i eksponent. Eksponent je celobrojna konstanta sa ili bez znaka, zapisana u dekadnom sistemu. Na primer:

```
1.2e2 ili 0.3e–5.
```

Vrednost ovakve konstante se računa tako što se broj ispred simbola *e* (mantisa mešovite konstante) pomnoži brojem koji se dobije nakon stepenovanja broja 10 brojnom konstantom iza simbola *e*, tj. gore zapisane konstante imaju, redom, vrednosti:

$1.2*10^{2}$ i $0.3*10^{-5}$.

1.3.2.3 Znakovne konstante

Znakovne konstante su konstrukcije čija je vrednost neki od znakova koje je moguće proizvesti na datom računaru (učitati ili prikazati na izlaznom uređaju). To su slova, cifre, interpunkcijski znaci itd. Najjednostavniji oblik zapisivanja takvih konstanti je navođenjem znaka između apostrofa: npr. 'a' ili 'Z' ili '!' itd.

Međutim, osim ovakvih znakova koji su, uslovno rečeno, pogodni za zapisivanje postoje i znaci koje je nešto teže prikazati. Na primer, takav je znak za prelazak u novi red, vraćanje na početak reda, vraćanje jedno mesto unazad, itd. To su tzv. neprintabilni znaci i znakovne konstante koje odgovaraju tim znacima moraju biti na drugi način zapisane. Tako su u programskom jeziku C uvedene i sledeće (specijalne) znakovne konstante:

'\n' znak za nov red;
'\t' znak za horizontalno pomeranje;
'\r' znak za pozicioniranje na početak reda;
'\a' znak čijim se štampanjem proizvodi zvučni efekat (engl. *beep*);
'\b' znak za vraćanje jedno mesto unazad (engl. *backspace*);
'\f' znak koji proizvodi prelazak na sledeću stranu (engl. *formfeed*);
'\"' jednostruki znak navoda;
'\"' dvostruki znak navoda;
'\\' znak \ (engl. *backslash*).

Ovakav zapis znakovne konstante je vrlo zgodan zato što podseća na odgovarajući znak. Drugi način da se zapiše neki znak je navođenjem *ASCII koda* datog znaka na jedan od sledeća dva načina:

'\ooo' gde pojedinim znakovima *o* odgovaraju oktalne cifre (iz skupa {0, 1, 2, 3, 4, 5, 6, 7}) tako da trocifreni ceo broj predstavlja *ASCII kôd* znaka koji želimo da zapišemo;

'\xhh' gde pojedinim pojavama znaka *h* odgovaraju cifre iz heksadekadnog brojnog sistema ({0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}). Tako je sa dve cifre zapisan heksadekadni broj koji predstavlja *ASCII kôd* željenog znaka.

1.4 Podatak

Podatak je svaki objekat koji se obrađuje (nad kojim se izvršavaju neke radnje (manipulacije)). Pojedini podaci se razlikuju po tome šta može biti njihova vrednost i šta sve možemo s njima da radimo. Tako je jedan podatak potpuno određen:

- skupom mogućih vrednosti koje može imati, i
- operacijama (radnjama, manipulacijama) koje možemo izvršavati nad njim.

Skup vrednosti i operacije određuju tip podataka. Svaki tip ima svoje ime.

1.5 Promenljive

Promenljive u programskom jeziku su objekti koji imaju neku vrednost, pri čemu se vrednost može menjati u toku izvršavanja programa. Pri tome to ne može biti bilo koja vrednost, već samo vrednost jednog tipa.

Svaka promenljiva koja se koristi mora biti deklarisana. Deklaracijom se određuju:

- ime promenljive (to je ime prema sintaksi jezika C),
- tip promenljive (odnosno tip kome pripada vrednost date promenljive) i
- početna vrednost (vrednost u početnom trenutku) – neobavezno.

Deklaracija promenljive ima sledeći zapis:

```
imet imep [= vrednost];
```

gde je *imet* tip promenljive, *imep* ime promenljive, a *vrednost* (koja može ali ne mora biti navedena) početna vrednost date promenljive.

Na primer, deklaracijama:

```
int i, j, n=10;
float a, e=2.71, pi=3.14;
char c, d='?';
```

su deklarisane:

- promenljive celobrojnog tipa *i, j* i *n*;
- promenljive realnog tipa *a*, *e* i *pi*;
- promenljive znakovnog tipa *c* i *d*.

Promenljive *n*, *e*, *pi* i *d* imaju, redom, vrednosti 10, 2.71, 3.14 i '?'.

1.6 Prosti tipovi podataka

Programski jezik C raspolaže tipovima podataka sličnim onima koji postoje u većini viših programskih jezika. Možemo ih podeliti u dve grupe: prosti i složeni (komponovani) tipovi podataka.

Od prostih tipova podataka postoje:

- znakovni tip podataka (char),
- celobrojni tip (short, int, long),
- mešoviti tip podataka (float, double).

1.6.1 Znakovni tip

Kako većina ovih tipova postoji u drugim višim programskim jezicima, nema potrebe da se detaljno opisuju.

Promenljiva tipa *char* može imati vrednost iz skupa znakova koje možemo prikazati na konkretnom računaru (velika i mala slova, cifre, specijalni znaci itd.).

1.6.2 Celobrojni tip

Promenljiva nekog od celobrojnih tipova može imati kao vrednost neki ceo broj. Međutim, opseg u kome se može kretati data vrednost je određen tipom koji smo koristili (*short, int, long*), kao i konkretnom implementacijom jezika C. Naime, pojedini od ovih celobrojnih tipova se razlikuju po broju bajtova iskorišćenih za registrovanje podataka i to je obično 1 (ili 2) bajt za podatak tipa *short*, 2 (ili 4) bajta za podatak tipa *int* i 4 bajta za podatak tipa *long*. U zavisnosti od toga koliko je bajtova iskorišćeno, menja se i opseg u kome se nalazi vrednost podatka, i to je:

- od – 128 do 127 ako je iskorišćen 1 bajt;
- od – 32768 do 32767 ako je iskorišćeno 2 bajta, ili
- od -2^{31} do $2^{31}-1$ ako se koriste 4 bajta.

Svakom od navedenih tipova može biti dodat prefiks *unsigned*. To znači da će se podatak tog tipa tretirati kao neoznačen broj (podatak bez znaka ili nenegativan). Time se menja opseg u kome se može nalaziti vrednost podatka tog tipa, i to je:

- 0 do 255 ako je iskorišćen 1 bajt;
- 0 do 65535 ako su iskorišćena 2 bajta;
- 0 do $2^{32}-1$ ako su iskorišćena 4 bajta.

Da bi se naglasilo da je neka celobrojna konstanta tipa *long*, iza niza cifara koji čine tu konstantu, dodaje se sufiks *l* (ili *L*). Da bi se naglasilo da je konstanta neoznačen broj, slično se dodaje sufiks *u* (ili *U*).

1.6.3 Logički tip

Za razliku od nekih viših programskih jezika, u programskom jeziku C ne postoji logički tip podataka (*boolean*). Promenljive tog tipa bi imale vrednosti iz skupa {tačno (istina, engl. *true*), netačno (laž, engl. *false*)}. Ipak, kao što ćemo videti kasnije, postoje neke logičke operacije: negacija, konjunkcija, disjunkcija. Naime, podatak bilo kog od navedenih tipova može biti interpretiran kao logička vrednost (tačno ili netačno). Uobičajeno je da se za to koriste celobrojni tipovi podataka. Naime, ako je vrednost nekog podatka tipa *int* različita od nule i ako ga tretiramo kao logički podatak (izvodimo logičke operacije), njegova vrednost će biti *tačno*. Ako je vrednost tog podatka jednaka nuli i ako ga tretiramo kao logički podatak, onda će imati vrednost *netačno*.

Ako rezultat određenih logičkih operacija pridružujemo promenljivoj nekog od navedenih tipova, onda će toj promenljivoj biti pridružena vrednost:

- 1 ako je rezultat logičke operacije *tačno* (istina);
- 0 ako je rezultat logičke operacije *netačno* (laž).

1.6.4 Mešoviti tip

Pored navedenih, u proste tipove podataka se ubrajaju i tipovi *float* i *double*. To su mešoviti tipovi podataka, tj. podaci tog tipa su realni brojevi. Napomenimo da to ne odgovara matematičkom poimanju realnih brojeva. Podatak ovog tipa ne može imati za vrednost bilo koji realni broj (već samo brojeve iz određenog podskupa i to je podskup skupa racionalnih brojeva [razlomaka]). Tipovi *float* i *double* se razlikuju po veličini prostora za registrovanje podataka (obično su to 4 bajta za podatak tipa float, odnosno 8 bajtova za podatak tipa double). Razlikuju se takođe po broju značajnih cifara koje možemo izdvojiti (to je obično 6–7 cifara za podatak tipa float, odnosno 14–15 cifara za podatak tipa double).

1.7 Definisane operacije

U daljem tekstu koristi se termin *l-vrednost*. Pod *l-vrednošću* se podrazumeva objekat kome može biti promenjena vrednost. Za sada su to promenljive a kasnije ćemo videti još neke objekte sa sličnim svojstvom.

Nad navedenim tipovima podataka definisane su neke operacije. Većina operacija je definisana za sve navedene proste tipove podataka, dok su neke definisane samo nad nekim tipovima.

1.7.1 Aritmetički operatori

Prvu grupu operatora čine aritmetički operatori:

Operator	**Naziv**	**Tipovi nad kojima je definisana**	**Vrsta operatora**
++	Uvećavanje za 1 (*inkrementiranje*)	char, int	unarni
––	Umanjivanje za 1 (*dekrementiranje*)	char, int	unarni
–	Promena znaka		unarni
*	Množenje		binarni
/	Deljenje		binarni
+	Sabiranje		binarni
–	Oduzimanje		binarni
%	Ostatak pri deljenju	char, int	binarni
=	Dodeljivanje		binarni

Operatori uvećavanja i umanjivanja se ne pojavljuju u većini drugih programskih jezika. To su unarni operatori (imaju jedan operand). Operand na koji se

primenjuju je znakovnog ili celobrojnog tipa i mora biti *l-vrednost*. Operator može biti naveden pre operanda (*prefiks operator*) ili posle operanda (*postfiks operator*). U oba slučaja dovodi do uvećavanja ili umanjivanja vrednosti operanda za 1 (znači, izraz *x++* [odnosno *++x*] je ekvivalentan izrazu *x=x+1* i, slično, izraz *x––* [odnosno ––x] je ekvivalentan izrazu *x=x–1*). Međutim, u slučaju primene *prefiks operatora*, prvo se izvršava uvećavanje ili umanjivanje vrednosti operanda i ta nova vrednost se koristi u preostalim operacijama u kojima učestvuje taj operand. Pri upotrebi *postfiks operatora* je obratno: trenutna vrednost tog operanda se koristi za ostale operacije u kojima se pojavljuje kao operand, a nakon toga se izvrši uvećanje ili umanjenje.

Na primer:

izraz,

```
y=x*++z
```

je ekvivalentan izrazu,

```
z=z+1, y=x*z,
```

a izraz,

```
y=x*z++
```

izrazu,

```
y=x*z, z=z+1.
```

Može se reći da je vrednost izraza:

```
++op
```

jednaka vrednosti izraza *op+1*, a da je vrednost izraza,

```
op++
```

jednaka sa *op*. Međutim, oba izraza imaju i dodatni efekat, a to je uvećavanje vrednosti objekta *op* za 1.

1.7.2 Složeni operatori dodeljivanja nastali kombinovanjem operatora dodeljivanja i nekog aritmetičkog operatora

Drugu grupu operatora čine složeni operatori dodeljivanja. Ovi operatori su nastali kombinovanjem aritmetičkih operatora i operatora dodeljivanja.

- **+=** Dodaje vrednost operanda sa desne strane operatora vrednosti operanda sa leve strane (levi operand je obavezno l-vrednost) i to postaje nova vrednost levog operanda (npr. *x+=3* je ekvivalentno izrazu *x=x+3*).
- **–=** Oduzima vrednost operanda sa desne strane operatora od vrednosti operanda sa leve strane i to postaje nova vrednost levog operanda (npr. *x–=3* je ekvivalentno izrazu *x=x–3*).

***=** Vrednost izraza sa desne strane množi se vrednošću operanda na levoj strani i to je nova vrednost levog operanda.

/= Vrednost operanda sa leve strane se deli vrednošću operanda sa desne strane i to je nova vrednost levog operanda.

%= Vrednost operanda sa leve strane se deli vrednošću operanda sa desne strane i ostatak tog deljenja je nova vrednost levog operanda.

1.7.3 Relacioni operatori

Treću grupu operatora čine relacioni operatori:

< Manje od.

<= Manje ili jednako.

== Jednako.

>= Veće ili jednako.

> Veće.

!= Različito.

Svi operatori iz ove grupe su binarni. Rezultat njihove primene je logičkog tipa. Na primer, vrednost izraza $x<y$ je *istina* (tačno, true) ako je x manje od y, u suprotnom je *laž* (netačno, false).

1.7.4 Logički operatori

Četvrtu grupu čine logički operatori:

Operator	**naziv**	**vrsta**
!	negacija	unarni
&&	konjunkcija	binarni
||	disjunkcija	binarni

Operandi se interpretiraju kao logičke vrednosti i nad takvim vrednostima se izvršava zadata operacija.

1.7.5 Operatori nad bitovima i operatori pomeranja

Petu grupu čine operatori nad bitovima i operatori pomeranja. Za operatore nad bitovima je karakteristično da se izvršavaju nad parovima odgovarajućih bitova operanada (ako je binarni operand) ili nad pojedinim bitovima operanda (ako je unarni operator).

<< Pomeranje ulevo. Levi operand je znakovnog ili celobrojnog tipa, a desni ceo broj. Vrednost izraza x<<y je jednaka proizvodu broja x i broja 2^y. Na tehničkom nivou, efekat izvršenja ove operacije je da se u reprezentaciji broja x na računaru (to je binarni broj, tj. broj u binarnom brojnom sistemu) izvrši pomeranje svih cifara za y mesta ulevo i dopiše y nula (0) na kraj broja. Ukoliko tako dobijeni broj više ne može da se predstavi u računaru (ima previše cifara), odbacuje se višak cifara s leve strane.

>> Pomeranje udesno. Levi operand je znakovnog ili celobrojnog tipa, a desni ceo broj. Vrednost izraza x>>y je jednaka količniku broja x i broja 2^y. Na tehničkom nivou efekat izvršenja ove operacije je da se u reprezentaciji broja x na računaru izvrši pomeranje svih cifara za y mesta udesno. Nakon tog pomeranja y najtežih cifara će biti izjednačeno sa;

- nulom, ako je operand *x* neoznačen (*unsigned*);
- najtežim bitom broja *x* pre izvršenja operacije, ako je *x* označen (*signed*).

~ Negacija nad bitovima. Ovo je unarni operator. Operand je celobrojnog ili znakovnog tipa. Operacija će se izvršiti tako što će se svaki bit u reprezentaciji tog podatka u računaru negirati (tj. ako je vrednost bita jedan (1) promeniće se u nulu (0) i obratno).

& Konjukcija nad bitovima. Ovo je binarni operator. Operandi su celobrojnog ili znakovnog tipa. Kao i kod negacije nad bitovima izvršava se (logička) konjunkcija nad parovima odgovarajućih bitova u operandima, i to će biti vrednost odgovarajućeg bita u rezultatu. Pri tome se 1 tretira kao *istina* (true), dok se 0 (nula) tretira kao *laž*. Na primer: *0x54 & 0x97 = 0x14*.

| Disjunkcija nad bitovima. Ovo je binarni operator. Operandi su celobrojnog ili znakovnog tipa. Izvršava se disjunkcija nad odgovarajućim bitovima operanada i to je vrednost odgovarajućeg bita u rezutatu.

^ Ekskluzivna disjunkcija nad bitovima. Operandi su znakovnog ili celobrojnog tipa. Izvršava se ekskluzivna disjunkcija nad odgovarajućim bitovima operanada i to je vrednost odgovarajućeg bita u rezultatu.

1.7.6 Složeni operatori dodeljivanja nastali kombinovanjem operatora dodeljivanja i operatora nad bitovima

Operatori nad bitovima se mogu kombinovati sa operatorom dodeljivanja. Tako se dobija još jedna grupa složenih operatora dodeljivanja:

|= Izvršava se disjunkcija nad odgovarajućim bitovima operanada sa leve i desne strane operatora i rezultat se dodeljuje levom operandu.

&= Izvršava se konjukcija nad bitovima nad odgovarajućim bitovima operanada sa leve i desne strane operatora i rezultat se dodeljuje levom operandu.

<<= Izraz *x*<<=*y* je ekvivalentan izrazu *x*=*x*<<*y*.

>>= Izraz *x*>>=*y* je ekvivalentan izrazu *x*=*x*>>*y*.

Za sve navedene složene operatore dodeljivanja levi operand mora biti l-vrednost.

1.7.7 Uslovni operator

Pomenimo, na kraju, i jedan ternarni operator (ima tri operanda). To je takozvani uslovni operator (?:). Prvi operand je uslovni (logički ili bulovski) izraz, dok su druga dva operanda proizvoljnog (ali istog tipa). Prvi i drugi operand su razdvojeni znakom pitanja (?), a drugi i treći dvotačkom (:). Dejstvo je sledeće:

- izračunava se vrednost prvog operanda;
- ako je vrednost prvog operanda istina, onda se računa vrednost drugog operanda i to je vrednost celog izraza;
- ako je vrednost prvog operanda laž, onda se računa vrednost trećeg operanda i to je vrednost celog izraza.

Na primer:

```
max=(x>y)? x: y; ili
min=(x>y)? y: x;
```

1.7.8 Operator veličine

Pomenimo i unarni operator *sizeof*. Operand za ovaj operator se navodi iza reči sizeof i to je ime promenljive ili tipa. Rezultat primene tog operatora je veličina memorijskog prostora (izražena u bajtovima) koji zauzima:

- data promenljiva (ako je operand promenljiva) ili
- promenljiva datog tipa (ako je operand tip).

1.8 Konverzije

Kao i u svim programskim jezicima, tako je i u programskom jeziku C poželjno da operandi u operacijama budu istog tipa (to se odnosi na binarne operacije, osim na operatore pomeranja). Dozvoljeno je, ipak, da operandi budu različitih tipova. U tom slučaju programski jezik C vrši konverziju pojedinih podataka. Konverzija podrazumeva promenu tipa podatka, ali tako da podatak zadrži istu vrednost (ako je to moguće). Postoje situacije kada nije moguće izvesti konverziju podataka iz jednog tipa u drugi tako da on zadrži vrednost. Navešćemo par primera konverzija iz jednog tipa u drugi tip.

1.8.1 Konverzija iz jednog celobrojnog tipa u drugi

Pri konverziji iz označenog u neoznačeni (ali isti) celobrojni tip, mogu nastupiti dva slučaja:

- ako je označeni broj bio pozitivan, vrednost ostaje ista;
- ako je označeni broj bio negativan, onda se konverzijom dobija najmanji pozitivan broj koji ima isti ostatak kao i polazni broj, pri deljenju brojem za jedan većim od najvećeg neoznačenog broja tog tipa koji se može predstaviti na računaru (tj. sa 2^8 ili 2^{16} ili 2^{32} ako se broj predstavlja pomoću 1, 2 ili 4 bajta, redom).

Ako vršimo konverziju iz neoznačenog u označen ceo broj istog tipa, brojevi veći od najvećeg označenog celog broja koji se može predstaviti, biće negativni posle konverzije. Dobijeni broj će imati isti ostatak pri deljenju brojem za jedan većim od najvećeg neoznačenog broja tog tipa koji se može predstaviti na računaru.

Na primer, ako je neoznačen broj 240 predstavljen pomoću jednog bajta, nakon konverzije u označeni broj dobićemo –16. To je zato što brojevi 240 i –16 imaju isti ostatak pri deljenju sa $256=2^8$.

Ako se vrši konverzija iz celog broja, koji se predstavlja pomoću većeg broja memorijskih lokacija, u ceo broj predstavljen sa manje memorijskih lokacija, onda se konverzija izvodi odbacivanjem potrebnog broja značajnijih (težih) cifara. Dobijeni broj ima isti ostatak kao i polazni broj pri deljenju brojem za 1 većim od najvećeg neoznačenog broja ciljnog tipa. Pod ciljnim tipom podrazumevamo tip u koji se vrši konverzija.

Ako se vrši konverzija iz celog broja, koji je predstavljen pomoću manjeg broja memorijskih lokacija, u ceo broj predstavljen sa više memorijskih lokacija, onda konverzija zavisi od toga da li su brojevi označeni ili neoznačeni:

- ako je broj neoznačen, konverzija se izvodi dopisivanjem potrebnog broja cifara sleva, pri čemu se dopisuju nule (0);
- ako je broj označen, konverzija se izvodi dopisivanjem potrebnog broja cifara sleva, pri čemu se dopisuju:
  - **0** (nule, ako je polazni broj bio pozitivan);
  - **1** (jedinice, ako je polazni broj bio negativan).

Dopisane cifre biće, dakle, identične najtežoj cifri u predstavljanju datog broja na računaru.

1.8.2 Konverzija iz mešovitog u celobrojni tip

Konverzija se izvršava odbacivanjem dela iza decimalne tačke. Posle odbacivanja, dobijeni ceo broj može biti izvan opsega brojeva koji mogu da se predstave. U tom slučaju efekat je nedefinisan i zavisi od implementacije. Obično se dobijeni broj deli brojem za jedan većim od najvećeg odgovarajućeg neoznačenog celog broja koji se može predstaviti, a nakon toga se odgovarajući neoznačeni broj konvertuje u označeni (ako se izvodi konverzija u označeni ceo broj).

1.8.3 Konverzije u toku izračunavanja vrednosti izraza

Programski jezik C ima linearno uređenje među tipovima, pa je tako:

```
char < unsigned char < short < unsigned short < int <
unsigned int < long < unsigned long < float < double.
```

Ako su operandi nad kojima se obavlja neka operacija različitog tipa, tada se vrednost operanda, uslovno rečeno, manjeg tipa konvertuje u odgovarajuću vrednost koja će biti istog tipa kao i drugi operand, posle čega se izvršava operacija nad tim podacima.

Ovakvo uređenje među tipovima dovodi ponekad do neočekivanih rezultata. Tako će rezultat poređenja *–1<1U* biti *laž* (netačno) zato što je desni operand tipa *unsigned int*, pa kako je to stariji tip, broj *–1* se konvertuje u odgovarajući neoznačen ceo broj (–1 je predstavljen kao *0xFFFF*, tj. 65535 i to je vrednost nakon konverzije) koji je veći od 1. Međutim, vrednost izraza – *1L<1U* je istina jer je levi operand starijeg tipa u hijerarhiji (*long*) pa se desni operand konvertuje u tip *long*; i tada poredimo brojeve –1 i 1.

Za aritmetičke operacije je karakteristično da je rezultat istog tipa kao i operandi. Tako će, na primer, izraz 5/3 imati vrednost 1 zato što su operandi celi brojevi.

Međutim, u nekim slučajevima je neophodno da dobijemo tačan rezultat deljenja 5/3. To postižemo tako što jedan operand konvertujemo u neki drugi tip (recimo *float*). Konverzija se postiže korišćenjem operatora promene tipa ili konverzije (tzv. *cast* operator). To je unarni operator. Primer korišćenja tog operatora je:

(tip) vrednost,

gde je *tip* ime tipa u koji želimo da konvertujemo vrednost koja sledi iza operatora (zagrade su obavezne). Na primer:

(float) 5/3

proizvodi da se broj 5 konvertuje u odgovarajući mešoviti broj (5.0) i nakon toga izvede deljenje brojeva 5.0 i 3. Kako je jedan operand tipa *float* i tip *float* je veći od tipa *int* (tip drugog operanda), to će i drugi operand biti konvertovan u tip *float*. Tako će se izvesti deljenje 5.0/3.0 i dobiti rezultat 1.666667.

1.9 Nabrojivi tip podataka

Sama reč kaže da se nabrojivi tip podataka dobija nabrajanjem svih vrednosti. Na primer:

```
enum dani {pon, uto, sr, cet, pet, sub, ned};
```

ili

```
enum meseci {jan, feb, mar, apr, maj, jun, jul, avg, sep, okt,
             nov, dec};
```

Prvim zapisom je definisan nabrojivi tip *dani* koji ima ukupno sedam vrednosti. Sporedni efekat je da su definisane simboličke konstante: *pon, uto, sr, cet, pet, sub* i *ned* koje imaju vrednosti, redom 0, 1, 2, 3, 4, 5 i 6.

Slično, drugim zapisom je definisan nabrojivi tip *meseci* sa ukupno 12 vrednosti i 12 simboličkih konstanti čije su vrednosti brojevi od 0 do 11.

Između velikih zagrada se navode imena razdvojena zarezima. Ako se drugačije ne naglasi, prvom imenu se dodeljuje vrednost 0, a svako sledeće ima za jedan veću vrednost. Iza nekog imena može biti dodat znak = (jednako) za kojim sledi ceo broj. Time je datom imenu dodeljena vrednost (to je broj iza znaka =), da bi sledeća imena opet dobijala za po jedan veću vrednost (ako im nije dodeljena).

Na primer, zapisom:

```
enum boje (crna, plava=2, zelena, zuta, crvena=6, ljubicasta};
```

definišemo tip *boje* i konstante *crna, plava, zelena, zuta, crvena* i *ljubicasta* čije su vrednosti, redom, 0, 2, 3, 4, 6 i 7.

Promenljivu nabrojivog tipa *dani* deklarišemo zapisom:

```
enum dani x;
```

Datoj promenljivoj mogu biti dodeljene vrednosti iz skupa *{pon, uto, sr, cet, pet, sub, ned}*.

1.10 Složeni tipovi podataka

Osim navedenih standardnih tipova podataka postoje i složeni tipovi, tj. oni koji se dobijaju komponovanjem prostih tipova podataka.

1.10.1 Nizovi u programskom jeziku C

Prvi od složenih tipova je niz. Promenljiva tipa niz se deklariše na sledeći način:

```
tip ime[BROJ_ELEMENATA];
```

Promenljiva *ime* ima ukupno *BROJ_ELEMENATA* vrednosti i sve su tipa *tip*. Svakoj od pojedinih vrednosti može se pristupiti pomoću *indeksa*, tj. *rednog broja* te vrednosti u nizu: prvoj se pristupa pomoću zapisa *ime[0]*, drugoj sa *ime[1]*, poslednjoj sa *ime[BROJ_ELEMENATA-1]*.

Umesto reči *BROJ_ELEMENATA* može stajati konkretan pozitivan ceo broj ili ime neke simboličke konstante poznate u fazi prevođenja programa. Taj broj određuje koliko će vrednosti (elemenata) imati niz i prema tome, koliko će prostora biti rezervisano za njegovo predstavljanje u računaru.

Programski jezik C dozvoljava da pojedinačni elementi niza opet budu nizovi tako da se može deklarisati:

```
tip ime[BROJ_ELEMENATA1]...[BROJELEMENATAK];
```

Ovim je deklarisan niz koji će imati *k* indeksa (kažemo da je to *k-dimenzionalni* niz). Svaki od indeksa se navodi u posebnom paru uglastih zagrada. Na primer, zapisom:

```
int x[10][20][30];
```

je deklarisan niz *x* koji ima tri indeksa (kaže se još da je to trodimenzionalan niz). Pojedinim elementima se pristupa tako što se navode tri indeksa, svaki u zasebnom paru zagrada:

x[0][0][0] ili *x[2][4][6]* ili *x[9][19][29]*.

Zapisom *x[4][4]* je određen jednodimenzionalni niz sa ukupno *30* elemenata, a zapisom *x[8]* dvodimenzionalan niz dimenzija *20×30*.

Kako se nizovi predstavljaju u računaru? U memoriji računara niz se predstavlja tako što se koristi određen broj uzastopnih memorijskih lokacija (bajtova). Broj iskorišćenih bajtova je određen brojem elemenata niza (BROJ_ELEMENATA1 * BROJ_ELEMENATA2 * ... * BROJ_ELEMENATAK) i brojem lokacija (bajtova) potrebnih za predstavljanje jednog elementa niza. U rezervisanom delu memorije, memoriše se prvo element čiji su svi indeksi jednaki 0. Zatim se memorišu redom elementi kod kojih se poslednji indeks uvećava postepeno za po jedan. Kada poslednji indeks dostigne najveću vrednost, onda se pretposlednji indeks uveća za jedan, dok poslednji kreće ponovo od nule (0) da bi se postepeno uvećavao, itd.

1.10.2 Strukturni tip

Osim nizova u programskom jeziku C mogu se definisati strukture. Jedna promenljiva tipa strukture ima više vrednosti koje ne moraju obavezno biti istog tipa. Na primer:

```
struct imet{
 tip1 ime1;
 tip2 ime2;
 ...
 tipk imek;
} imep;
```

Promenljiva sa imenom *imep* je tipa *struct imet*. Ona ima ukupno k vrednosti koje su redom tipova tip_1, tip_2, ..., tip_k. Svakoj od tih vrednosti može se pristupiti pomoću imena te vrednosti navedenog iza imena promenljive i tačke (.). Tako se prvoj pristupa zapisom $imep.ime_1$, drugoj sa $imep.ime_2$ itd., poslednjoj sa $imep.ime_k$.

Pojedine vrednosti u okviru strukture zovu se *polja*. Naravno, pojedina polja takođe mogu biti strukturnog tipa. Ovde srećemo još jedan operator, operator . (element strukture).

1.10.3 Unije

Promenljiva tipa unije ima više vrednosti koje mogu biti različitih tipova, ali je za njih karakteristično da su zapamćene u istom memorijskom prostoru. Pošto je za registrovanje iskorišćen isti memorijski prostor (tj. prostor od iste adrese), pojedine vrednosti nisu u potpunosti nezavisne. Naime, promena jedne od vrednosti može dovesti do promene i ostalih.

Na primer, zapisom:

```
union pru {
 char a;
 int b;
 float c;
} x;
```

je deklarisana promenljiva *x*. Promenljiva *x* ima tri vrednosti. Te vrednosti ćemo zvati polja. Svakom od polja se pristupa navođenjem njegovog imena iza imena promenljive (*x*) i tačke: *x.a*, *x.b*, *x.c*.

Memorijski prostor potreban za registrovanje promenljive tipa unije određen je memorijskim prostorom potrebnim za registrovanje najdužeg polja (tj. polja za čije je registrovanje potreban najveći prostor). U prethodnom primeru, za polje *a* potreban je 1 bajt, za polje *b* 2 bajta i za polje *c* 4 bajta. Stoga je za registrovanje promenljive *x* potrebno 4 bajta.

Sva polja u okviru unije se registruju od iste memorijske lokacije – to je lokacija sa najnižom adresom u okviru memorijskog prostora rezervisanog za promenljivu tipa unije. Naravno, ne moraju sva polja da iskoriste celokupni memorijski prostor rezervisan za registrovanje unije.

1.10.4 Polja bitova

Podatak ovog tipa sadrži više vrednosti, ali za predstavljanje pojedinih vrednosti ne mora biti iskorišćen ceo bajt ili registar (dva bajta), već deo bajta ili registra. Na primer, deklaracijom:

```
struct pb {
 int x:3, y:4;
 unsigned z:2, w:5;
} u;
```

definisana je promenljiva u. Promenljiva u ima četiri vrednosti, koje se zovu polja. Pojedinim poljima se pristupa navođenjem imena iza prefiksa koji se sastoji od imena promenljive i tačke (u.x, u.y, u.z i u.w). Za registrovanje polja x se koriste 3 bita, polja y 4 bita itd. Polja x i y će biti registrovana u okviru istog bajta zato što je za njihovo registrovanje potrebno ukupno 7 bitova. Za sledeća polja se rezerviše navedeni broj bitova, počev od prvog najlakšeg neiskorišćenog. Polja x i y se interpretiraju kao celi označeni brojevi. Tako će vrednost polja x biti u intervalu $[-4, 3]$, a vrednost polja y u intervalu $[-8=-2^3, 7=2^3-1]$. Polja z i w se interpretiraju kao neoznačeni brojevi. Tako će se vrednost polja z kretati u opsegu $[0, 3=2^2-1]$, a vrednost polja w u opsegu $[0, 31=2^5-1]$.

Polja bitova povećavaju izražajne mogućnosti programskog jezika C i približavaju ga mašinskom jeziku. Naime, često se sadržaj registra deli na manje celine koje za sebe imaju određeno značenje. To se posebno odnosi na tzv. statusne i kontrolne registre. Na primer, u okviru statusnog registra centralnog procesora se mogu izdvojiti delovi kojima su zapisane sledeće informacije: znak rezultata poslednje aritmetičke operacije, da li je rezultat poslednje izvršene operacije jednak nuli, da li je u toku izvođenja došlo do prekoračenja itd. Takvi registri se na najprirodniji način modeliraju poljima bitova.

Napomenimo da se prostor za polja rezerviše od lakših ka težim bitovima.

1.11 Pokazivači

U programskom jeziku C je omogućen rad sa pokazivačima. Promenljiva tipa pokazivač se deklariše na sledeći način:

```
tip *ime;
```

gde je *tip* ime nekog tipa (standardnog ili nekog koji je definisao korisnik). Promenljiva *ime* je tipa pokazivač (ili pokazivačkog tipa). Njena vrednost je obeležje (obično adresa) memorijske lokacije (prostora) na kojoj može da se zapiše vrednost tipa *tip*. Toj memorijskoj lokaciji pristupamo zapisom **ime*. Zapis **ime* sa stanovišta C-a ima gotovo sva svojstva kao i obična promenljiva (može joj biti dodeljena vrednost ili može biti operand u izrazu).

Neka imamo sledeću deklaraciju:

```
tip *ime1, *ime2;
```

Možemo zapisati sledeće izraze:

```
ime1 = ime2;
*ime1 = *ime2;
```

Prvim izrazom izjednačavamo vrednosti pokazivačkih promenljivih *ime1* i *ime2* tako da one pokazuju na istu memorijsku lokaciju (to je lokacija na koju je pre izvršavanja ovog izraza pokazivala promenljiva *ime2*). Drugim izrazom samo se izjednačava sadržaj memorijske lokacije na koju pokazuje pokazivačka promenljiva *ime1* sa sadržajem lokacije na koju pokazuje promenljiva *ime2*.

U bliskoj vezi sa pokazivačima je i unarni operator & (*adresa*). Vrednost izraza &*ime* je obeležje memorijske lokacije rezervisane za registrovanje promenljive sa imenom *ime*. Takva vrednost može biti dodeljena promenljivoj tipa pokazivač. Naime, ako imamo deklaraciju:

```
tip *ime1, ime2;
```

onda zapisom,

```
ime2 = *ime1;
```

promenljivoj *ime2* dodeljujemo vrednost koju sadrži memorijska lokacija na koju pokazuje pokazivačka promenljiva *ime1*. Zapisom,

```
ime1=&ime2;
```

menjamo vrednost promenljive *ime1* tako da je njena vrednost jednaka obeležju (adresi) lokacije rezervisane za promenljivu *ime2*.

Osim na ovaj način (korišćenjem operatora adresa), vrednost pokazivačke promenljive može da se inicijalizuje i korišćenjem standardne funkcije *alloc*. Data funkcija ima samo jedan argument i to je ceo pozitivan broj (*n*). Funkcija rezerviše prostor od *n* bajtova i vraća obeležje (adresu) početka tog prostora. Preporučljivo je jednom rezervisani prostor (korišćenjem funkcije *alloc*) osloboditi pre završetka izvršavanja programa. To se postiže pozivanjem funkcije *afree*. Nedostatak ovih standardnih funkcija je u tome što oslobađanje rezervisanog memorijskog prostora mora biti izvršeno u obrnutom redosledu od rezervisanja. Zbog toga je naknadno razvijen par funkcija *malloc* i *free*, koji to nije zahtevao.

Za razliku od nekih programskih jezika, pokazivačka promenljiva ne mora obavezno pokazivati na memorijsku lokaciju na kojoj je registrovana jedna vrednost, već na memorijsku lokaciju od koje je registrovan niz vrednosti istog tipa. Ako imamo deklaraciju:

```
tip *ime;
```

onda prvoj od vrednosti pristupamo sa **ime*, sledećoj sa **(ime+1)*, zatim **(ime+2)* itd. Međutim, tim vrednostima možemo pristupiti i sa *ime[0]*, *ime[1]*, *ime[2]* itd. Tako pokazivači pomalo podsećaju na nizove. Ako imamo deklaraciju:

```
tip *ime1, ime2[NN];
```

onda možemo zapisati *ime1[0]*, *ime1[1]*, *ime1[2]* itd., i slično *ime2[0]*, *ime2[1]*, *ime2[2]* itd. Ove dve promenljive razlikuju se u sledećem: možemo napisati,

```
ime1=...
```

ali ne možemo napisati,

```
ime2=....
```

Drugim rečima, ime pokazivačke promenljive može stajati na levoj strani operatora dodeljivanja, dok ime niza ne može. Obično se kaže da su nizovi *konstantni pokazivači*.

1.11.1 Operatori definisani nad pokazivačima

Promenljive tipa pokazivač mogu biti operandi u nekim aritmetičkim operacijama. Pri tome se pokazivači tretiraju kao celi brojevi (jer adrese su isto tako celi brojevi). Tako se mogu koristiti:

- unarni operatori uvećavanja (inkrementiranja) i umanjivanja (dekrementiranja); ako napišemo *ime*++ onda će se vrednost promenljive *ime* uvećati za jedan (1), a to znači da će pokazivati na lokaciju na kojoj je registrovan sledeći podatak tipa na koji pokazuje pokazivačka promenljiva *ime*. Na primer, ako imamo deklaraciju,

  ```
  int *x;
  ```

  onda će se izrazom,

  ```
  x++;
  ```

uvećati vrednost promenljive *x* za 2 jer u većini implementacija celi brojevi se registruju korišćenjem 2 bajta. U opštem slučaju, ako je *x* pokazivačka promenljiva, zapisom,

```
x++;
```

vrednost promenljive *x* biće uvećana za broj memorijskih lokacija koje zauzima podatak na koji pokazuje promenljiva *x* (odnosno *x* će biti uvećano za *sizeof *x*).

- binarni operatori sabiranja i oduzimanja, pri čemu je drugi operand ceo broj; ako imamo promenljivu deklarisanu sa:

```
tip *x;
```

onda izraz *x+3* ima vrednost istog tipa kao i promenljiva *x* (pokazivač *tip **) i to je obeležje lokacije na kojoj se nalazi treća vrednost tipa *tip* u odnosu na vrednost *x* (odnosno obeležje lokacije na kojoj je *x[3]*). Drugim rečima, vrednost *x* se uvećava za proizvod broja 3 i broja bajtova potrebnih za registrovanje jedne vrednosti tipa *tip (sizeof *x)*.

Ako je deklarisan pokazivač na strukturu:

```
struct t *x;
```

tada se pojedinim poljima strukture *struct t* ne pristupa zapisom,

```
*x.imep;
```

kao što bi se očekivalo, već zapisom,

```
x->imep;
```

gde je *imep* ime polja u okviru strukture. Tako stižemo do još jednog operatora:

-> je binarni operator (polje strukture).

1.12 Prioriteti i asocijativnost operatora

Navedimo na kraju prioritete pojedinih operatora, kao i *vrstu asocijativnosti*. Kažemo da je operator * *levo asocijativan* ako je zapis,

```
x*y*z
```

ekvivalentan zapisu,

```
(x*y)*z.
```

Slično, operator * je *desno asocijativan* ako je zapis,

```
x*y*z
```

ekvivalentan zapisu

```
x*(y*z).
```

Operatorima će kao prioritet biti dodeljeni celi brojevi. Manji broj označava da je operator većeg prioriteta:

Prioritet 1	**Opis**	**Asocijativni**
++	postfiks	levo
––	postfiks	
–>	polje strukture	
.	polje strukture	
Prioritet 2		
++	prefiks	desno
––	prefiks	
!	logička negacija	
~	negacija nad bitovima	
–	promena znaka	
&	adresa	
*	pokazana promenljiva	
sizeof	operator veličine	
(tip)	operator konverzije	
Prioritet 3		
*	množenje	levo
/	deljenje	
Prioritet 4		
+	sabiranje	levo
–	oduzimanje	levo
Prioritet 5		
<<	pomeranje ulevo	levo
>>	pomeranje udesno	
Prioritet 6		
<	manje	levo
<=	manje ili jednako	
>	veće	
>=	veće ili jednako	
Prioritet 7		
==	jednako	levo
!=	različito	
Prioritet 8		
&	konjukcija nad bitovima	levo
Prioritet 9		
^	isključiva disjunkcija nad bitovima	levo

Prioritet 10	**Opis**	**Asocijativni**
|	disjunkcija nad bitovima	levo
Prioritet 11		
&&	konjukcija	levo
Prioritet 12		
||	disjunkcija	levo
Prioritet 13		
:?	uslovni operator	desno
Prioritet 14		
=	dodeljivanje	desno
*=, /=		
%=, +=		
–=, <<=		
>>=, &=		
^=, |=	složeni operatori dodeljivanja	
Prioritet 16		
,	zarez	levo

1.13 Izrazi

U programskom jeziku C izrazi se formiraju od konstanti, imena promenljivih, imena funkcija i navedenih operatora. Pri tome treba voditi računa o prioritetu pojedinih operatora, kao i o asocijativnosti operatora (leva ili desna asocijativnost).

Primetimo da je za razliku od nekih programskih jezika i dodeljivanje (=) operand. Tako je ono što je u nekim jezicima naredba dodeljivanja ili aritmetička naredba, u jeziku C *izraz*.

1.14 Naredbe

Napomenuli smo već da je skup naredbi u programskom jeziku C vrlo mali. Najjednostavnije naredbe su:

- izraz,
- prazna naredba i
- poziv funkcije.

Napomenimo takođe da se svaka naredba u programskom jeziku C završava znakom tačka i zarez (;). Naime, u programskom jeziku C ovaj znak služi kao oznaka za kraj naredbe (engl. *terminator*), za razliku od nekih programskih jezika u kojima je znak koji razdvaja dve naredbe (engl. *separator*).

Naredba može (ali ne mora) biti obeležena. Ako je naredba obeležena, neposredno ispred nje je postavljeno obeležje. Obeležje je neko ime proizvedeno po sintaksnim pravilima programskog jezika C. Obeležavanje naredbe se postiže tako što se navede obeležje i iza njega dvotačka (:). Naglasimo da se u telu jedne funkcije ne može više puta koristiti isto obeležje za obeležavanje.

Osim navedenih, postoje i sledeće grupe naredbi:

- složena (komponovana) naredba;
- naredbe grananja i izbora;
- naredbe petlji (ciklusa);
- naredbe za nasilnu promenu toka.

1.14.1 Komponovana naredba

Komponovanu naredbu dobijamo od jedne ili više narebi koje grupišemo tako što ispred prve stavimo otvorenu veliku (vitičastu) zagradu ({), a iza poslednje zatvorenu veliku (vitičastu) zagradu (}). To ovako izgleda:

```
{
 naredba1
 naredba2
 ...
 naredbak
}
```

Komponovana naredba se izvršava tako što se jedna za drugom izvršavaju naredbe od kojih je ona sastavljena.

1.14.2 Naredbe grananja i izbora

Ovu grupu naredbi čine naredbe *if* i *switch*. Naredba *if* ima sledeću sintaksu:

```
if (izraz) naredba [else naredba]
```

Naredba počinje službenom rečju *if* iza koje se navodi izraz (obavezno u zagradama). Nakon toga navodi se naredba iza koje može biti navedena službena reč *else*, za kojom sledi naredba (reč *else* i naredba iza nje obrazuju *else-deo* naredbe i mogu se izostaviti). Dejstvo naredbe je sledeće:

- izračunava se vrednost izraza;
- ako je dobijena vrednost istina (različita od nule), izvršava se naredba koja neposredno sledi iza izraza;
- ako je vrednost izraza laž (vrednost je nula) i ako postoji *else*-deo, onda se izvršava naredba iza reči *else*.

Pored naredbe *if* postoji i naredba *switch*. Njena sintaksa je sledeća:

```
switch (izraz) {
 case v1:
  naredba11
  naredba12
  ...
```

```
  naredba1k1
 case v2:
  naredba21
  naredba22
  ...
  naredba2k2
 . . .
 case vl:
  naredbal1
  naredbal2
  ...
  naredbalkl
[ default:
  naredba1
  naredba2
  ...
  naredbak
]
}
```

Kao što se vidi, naredba počinje službenom rečju *switch* iza koje se navodi izraz (obavezno u zagradama). Izraz je nekog od standardnih tipova različitog od *float* i *double*. Preostali deo naredbe se navodi u zagradama ({}) i čine ga delovi oblika:

```
case vi:
 naredbai1
 naredbai2
 ...
 naredbaiki
```

gde je *case* službena reč, a *vi* konstanta istog tipa kao i izraz. Osim toga, može postojati (ali ne mora) deo:

```
default:
 naredba1
 naredba2
 ...
 naredbak
```

Efekat izvršenja naredbe je sledeći:

- izračunava se vrednost izraza;
- pronalazi zapis oblika *case v:* unutar bloka između zagrada tako da je konstanta *v* jednaka izračunatoj vrednosti izraza. Izvršavaju se sve naredbe koje se nalaze između pronađenog *case v* i kraja naredbe *switch*;
- ako, pak, vrednost dobijena izračunavanjem izraza nije pronađena uz neko *case*, a postoji deo *default*, onda se izvršavaju sve naredbe od tog obeležja pa do kraja naredbe *switch*.

Izvršavanje naredbe *switch* se nasilno prekida ako se pojavi naredba *break*. Naime, ova naredba proizvodi prekid izvršavanja naredbe *switch* i prouzrokuje prelazak na prvu naredbu iza naredbe *switch*.

1.14.3 Naredbe petlji

Prva od ovih naredbi je naredba *while*. Ona ima sledeću sintaksu:

```
while (izraz) naredba
```

gde je *while* službena reč. I u ovom slučaju izraz se obavezno navodi u zagradama. Semantika je sledeća:

- izračunava se vrednost izraza;
- ako je dobijena vrednost istina, izvršava se naredba koja sledi i ponavlja se postupak od početka; u suprotnom se prelazi na sledeću naredbu.

Druga naredba je:

```
do naredba while (izraz)
```

Njena semantika je sledeća:

- izvršava se naredba iza reči *do*;
- zatim se računa vrednost izraza iza reči *while*;
- ako je vrednost tog izraza istina, onda se sve ponavlja; u suprotnom se prelazi na sledeću naredbu.

Treća naredba je:

```
for (izraz1; izraz2; izraz3) naredba
```

Izvršavanje ove naredbe sastoji se od sledećih radnji:

(i) izračunava se *izraz1*;
(ii) izračunava se *izraz2*;
(iii) ako je *izraz2* tačan, izvršava se naredba i izračunava *izraz3*; zatim se ponovo izračunava *izraz2*, tj. ponovo se izvršava korak (ii);
(iv) ako je *izraz2* netačan, prekida se izvršavanje naredbe *for* i prelazi na sledeću naredbu.

Može se reći da je naredba *for* ekvivalentna nizu naredbi:

```
izraz1;
while (izraz2) {
 naredba;
 izraz3;
}
```

1.14.4 Naredbe promene toka

Već smo se sreli sa jednom od naredbi promene toka. To je naredba *break*. Efekat naredbe *break* zavisi od mesta na kome se ona pojavljuje:

- ako se koristi u telu naredbe *switch*, dovodi do nasilnog prekida izvršavanja naredbe *switch* i prelaska na prvu naredbu iza naredbe *switch*.
- ako se koristi u naredbi petlje, dovodi do prekida izvršavanja petlje i prelaska na prvu naredbu iza naredbe petlje.
- u ostalim slučajevima naredba *break* je bez efekta.

Druga naredba promene toka je *continue*. Koristi se u naredbama petlji tako što proizvodi preskakanje preostalih naredbi koje čine telo petlje. Po izvršavanju naredbe *continue* odmah se prelazi na ispitivanje uslova za završetak izvršavanja naredbe petlje (odnosno računanje izraza kojim se kontroliše izvršavanje naredbe petlje).

Treća naredba iz ove grupe je naredba *goto*. Za razliku od preostalih, upravo navedenih naredbi, ova naredba ima nešto složeniju sintaksu:

```
goto obeležje;
```

gde je obeležje neko od obeležja koje postoji u funkciji u kojoj se nalazi data naredba *goto*. Efekat ove naredbe je nastavljanje izvršavanja funkcije od naredbe ispred koje stoji navedeno obeležje.

U grupu ovih naredbi može se uvrstiti i naredba *return*. Njena sintaksa je sledeća:

```
return [izraz];
```

Efekat ove naredbe ogleda se u prekidu izvršavanja funkcije koje je trenutno u toku i povratku u funkciju iz koje je pozvana. Nastavlja se izvršavanjem naredbe neposredno iza poziva funkcije.

Kao rezultat svog izvršavanja, funkcija može proizvoditi neku vrednost. Ta vrednost je vrednost izraza koji se navodi iza reči *return*. Ukoliko se ništa ne navede, a funkcija treba da vrati neku vrednost, vraćena vrednost je nedefinisana.

1.15 Funkcije

Funkcije su složene leksičke konstrukcije. Sastoje se od određenog broja deklaracija promenljivih i niza naredbi koji čine logičku celinu. Izračunavanje određeno nizom naredbi može proizvesti jedan rezultat. Tip rezultata je obično neki od standardnih tipova ili pokazivač. Tip rezultata je istovremeno i tip funkcije. U slučaju kada funkcija nema jedan karakterističan rezultat, funkcija je tipa *void*.

Naime, može se desiti da je broj rezultata (izračunatih vrednosti) veći od jedan. U tom slučaju rezultati se prenose kroz *argument listu*.

Argument lista je mehanizam ne samo za prenošenje rezultata, već i za prenošenje argumenata (ulazni podaci) funkcije. Broj argumenata nije ograničen (0 ili više). Takođe ne postoje ograničenja u pogledu tipa argumenata (osim da ne mogu biti strukturnog tipa).

Na primer:

```
int f1 (int a, int b, int *c) {
...
}
```

predstavlja funkciju *f1* koja ima tri argumenta:

- dva su tipa *int* (*a* i *b*) i
- treći je tipa pokazivač na podatak tipa *int* (c).

Funkcija kao rezultat daje jednu vrednost tipa *int*.

Funkcija *f1* može da se pozove iz neke funkcije tako što se pojavi u nekom izrazu:

```
x = ... f1(u, v, &w)...
```

Kao što se vidi, u pozivu se navode argumenti funkcije (to su stvarni argumenti funkcije). Primetimo da je na treći argument primenjen operator adresiranja (jer je odgovarajući argument funkcije pokazivač na ceo broj).

U toku izvršavanja funkcije prvi i drugi argument (*a* i *b*) mogu se menjati (biti levi operand nekog operatora dodeljivanja). Međutim, te promene se neće odraziti na vrednost odgovarajućih stvarnih argumenata (u ovom slučaju to su promenljive *u* i *v*). Ovo zato što se u toku izvršavanja funkcije za te argumente rezerviše prostor na sistemskom steku (kopije odgovarajućih stvarnih argumenata) i sve radnje nad tim argumentima se registruju na tom mestu.

Za treći argument je karakteristično da se može pristupiti odgovarajućem stvarnom argumentu. To se ostvaruje korišćenjem unarnog operatora * (izrvšavanje izraza **c = ...*, dovodi do promene vrednosti stvarnog argumenta čija je adresa stavljena u argument listu).

Tako prva dva argumenta odgovaraju *vrednosnim*, a treći *promenljivim* argumentima funkcije.

Prethodno navedeni zapis:

```
int f1(int a, int b, int *c) {
...
}
```

predstavlja definiciju funkcije *f1*. Osim definicije funkcije, u programskom jeziku C postoji deklaracija funkcije. Deklaracija funkcije se razlikuje od definicije po tome što se ne navodi telo funkcije (deklaracije promenljivih i niz naredbi od kojih se sastoji funkcija). Za konkretnu funkciju deklaracija ima sledeći zapis:

```
int f1 (int, int, int *);
```

Kao što se vidi, deklaracija funkcije se sastoji od tipa funkcije za kojim slede ime funkcije i, u zagradama, tipovi pojedinih argumenata. Imena argumenata nisu obavezna.

Za svaku deklarisanu funkciju na nekom mestu mora biti navedena definicija. Namena deklaracije je da prevodiocu naglasi da postoji funkcija sa odgovarajućim imenom, tipom i argumentima. Te informacije prevodilac koristi, ako se u toku prevođenja pojavi poziv neke funkcije pre njene definicije (radi kontrole da li se tipovi argumenata i rezultata slažu).

1.16 Program na jeziku C

Program na programskom jeziku C čini određen broj funkcija. Pri tome obavezno mora postojati funkcija sa imenom *main*. Funkcija sa imenom *main* je glavna. Izvršavanje programa napisanog na jeziku C predstavlja pozivanje (odnosno, izvršavanje) funkcije *main*.

Primer jednog programa na jeziku C:

```
# include <stdio.h>
# define INC 2.54
main () {
 int i, n;
 printf ("n = "):
 scanf ("%d", &n);
 for (i = 1; i <= n; i++)
  printf ("%5d%5x%15.2f\n", i, i, i*INC);
}
```

Prve dve linije predstavljaju takozvane *preprocesorske direktive*. Sama reč kaže da su to instrukcije koje će biti obrađene pre samog prevođenja programa. Sve preprocesorske direktive počinju znakom #. Najčešće se koriste direktive *include* i *define*.

Direktiva *include* ima zapis:

```
# include <ime> ili
# include "ime".
```

Ovom direktivom u program se uključuje sadržaj datoteke čije se ime navodi između zagrada ili znakova navoda. U toj datoteci najčešće su navedene deklaracije funkcija. U ovom slučaju, u datoteci *stdio.h* se nalaze funkcije za ulazno-izlazne operacije (učitavanje, izdavanje podataka, otvaranje i zatvaranje datoteka, čitanje i upis, itd.).

Direktiva *define* ima zapis:

```
# define ime tekst
```

Ovom direktivom se definiše simbolička konstanta *ime* kojoj odgovara navedeni *tekst*. Efekat je da će svako pojavljivanje te konstante biti zamenjeno tim tekstom. Najčešće se koristi za definisanje simboličkih konstanti i makroa.

U našem primeru funkcija *main* radi sledeće:

- pozivom funkcije *printf* na ekranu se ispisuje tekst između znakova navoda (*n* =);
- pozivom funkcije *scanf* učitava se ceo broj *n*;
- u petlji se za sve brojeve *i* od 1 do *n* ispisuje:
  - broj *i* u dekadnom brojnom sistemu;
  - broj *i* u heksadekadnom brojnom sistemu;
  - proizvod broja *i* i konstante *INC* (2.54).

1.17 Memorijske klase

U prethodnom poglavlju smo videli kako se deklarišu promenljive. Međutim, od mesta na kome je promenljiva deklarisana zavise neke njene karakteristike: gde se sve može koristiti, promene vrednosti u toku izvršavanja itd. Na osnovu tih karakteristika, promenljive su podeljene u *memorijske klase*.

Prva klasa je klasa *automatskih promenljivih*. To su sve promenljive definisane u okviru neke od funkcija ili na početku komponovane naredbe. One se vide samo iz funkcije (ili komponovane naredbe) u okviru koje su definisane. Po završetku izvršavanja funkcije (ili komponovane naredbe) više ne postoje. Pri ponovnom izvršavanju funkcije, vrednost je proizvoljna (neodređena, zavisi od trenutka).

Drugu klasu čine *registarske promenljive*. Definišu se tako što se ispred imena tipa promenljive doda prefiks *register*. Za njih je karakteristično da se u fazi prevođenja povezuju sa nekim registrom iz centralnog procesora. Vrednost promenljive se čuva sve vreme izvršavanja u tom registru. Time se povećava brzina pristupa takvoj promenljivoj. Promenljiva registarskog tipa se čuva u registru ako je to moguće, tj. ako ima raspoloživih registara. Ako nije moguće, te promenljive se tretiraju kao automatske.

Treću klasu čine *statične lokalne promenljive*. Ove promenljive su definisane u okviru neke funkcije tako što je ispred naziva tipa stavljen prefiks *static* (npr. *static int n, m*). Ove promenljive su vidljive samo u funkciji u kojoj su definisane (samo se tu mogu koristiti), ali pri ponovnom izvršavanju funkcije imaju vrednost koju su imale na kraju prethodnog izvršavanja (čuva se vrednost između pojedinih izvršavanja).

Četvrtu klasu čine *statične globalne promenljive*. Ove promenljive se vide iz svih funkcija koje čine jedan modul (nalaze se u jednoj datoteci). Kao i statične lokalne promenljive, definišu se tako što se doda prefiks *static*, ali su definisane pre svih funkcija (izvan funkcija).

Petu klasu čine *globalne promenljive*. Dobijaju se tako što se ispred naziva tipa doda prefiks *extern*. Definišu se izvan funkcija i vidljive su iz svih funkcija koje čine program (tj. vidljive su iz svih modula). Međutim, obavezno mora postojati modul u kome je ova promenljiva definisana izvan tela funkcija (*i nije statična*). U suprotnom smatra se da promenljiva nije definisana. Na taj način dobijamo klasu promenljivih koje su vidljive iz više modula i postoje sve vreme izvršavanja programa.

Četvrta i peta klasa promenljivih obezbeđuju mehanizam razmene podataka između funkcija: istina, četvrta klasa između funkcija u okviru jednog modula, a peta između funkcija iz različitih modula.

1.18 Složene deklaracije u programskom jeziku C

Programski jezik C dozvoljava vrlo složene deklaracije. Tako mogu postojati deklaracije:

```
char **argv;
```

argv je pokazivač na pokazivač na podatak tipa *char*;

```
char (*x)[13]
```

x je pokazivač na niz od 13 elemenata tipa *char*;

```
int *x[13]
```

x je niz od 13 pokazivača na podatke tipa *int*;

```
int *f()
```

f je funkcija koja vraća pokazivač tipa *int*;

```
int (*x)()
```

x je pokazivač na funkciju koja vraća rezultat tipa *int*;

```
int (*(*x())[])()
```

x je funkcija koja vraća pokazivač na niz čiji su elementi pokazivači na funkcije koje vraćaju podatke tipa *int*;

```
int (*(*x[3])())[5]
```

x je niz dužine 3 čiji su elementi pokazivači na funkcije koje vraćaju pokazivače na niz dužine 5 čiji su elementi tipa *int*;

Složene deklaracije se čitaju pomoću sintaksnih pravila za deklaracije:

```
decl::=ime dcl;
dcl::=[*]*dir-dcl
dir-dcl::=ime |
          (dcl) |
          dir-dcl() |
          dir-dcl[[BR_EL]]
```

Znak | razdvaja pojedine varijante za građenje konstrukcije (tj. *dir-dcl* je ime ili *dcl* u zagradama ili *dir-dcl,* za kojim sledi par malih zagrada ili *dir-dcl*, za kojim sledi par uglastih zagrada, pri čemu ne mora biti naveden broj članova niza).

Prvo pravilo kaže da se deklaracija nekog objekta sastoji od imena (ime tipa) za kojim sledi konstrukcija *dcl*.

Po drugom pravilu *dcl* sa sastoji od određenog broja zvezdica (nula ili više) za kojim sledi konstrukcija *dir-dcl* (to je zapravo *dcl* bez zvezdica koje se nalaze na početku).

Slika 1.1 prikazuje kako se čita deklaracija:

```
(*pfa[])();
```

U sledećem odeljku napisaćemo program koji čita deklaraciju i proizvodi rečenicu na prirodnom jeziku, koja opisuje tip promenljive deklarisane tom deklaracijom.

Slika 1.1 Primer čitanja deklaracije; deklaracija je (*pfa[]) ();