Programski jezik C

Uvod

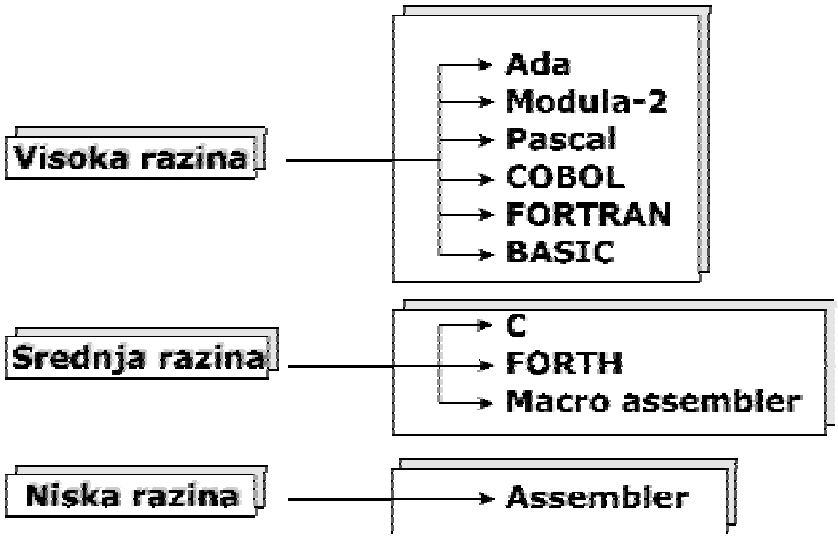
Kratka povijest programskog jezika C

Programski jezik C vuče korijen iz programskih jezika BCPL i B koje je Ken Thompson u AT&T Bell Laboratories razvio za potrebu implementacije novog operacijskog sustava UNIX. Dennis Ritchie je 1971. na temeljima programskog jezika B počeo razvoj novog programskog jezika, jednostavno nazvanog C, na računalima DEC PDP-11. 1973. UNIX je u cijelosti prepisan u programskom jeziku C. Može se kazati da nema UNIX-a bez C-a i obratno. 1983. Godine ANSI (American National Standards Institute) ustanovljuje komitet za standardizaciju definicija u programskom jeziku C. Konačno je 1988. završena prva standardizacija koja se jednostavno naziva ANSI-C. Većina današnjih prevodilaca podržava ovaj standard.

Osnovne karakteristike programskog jezika C

C je programski jezik srednje razine (Slika 1.). U odnosu na ostale programske jezike karakterizira ga:

- Veličina. Vrlo mali broj ključnih riječi (32). C nema uključene naredbe za ulaz ili izlaz, manipulaciju sa stringovima i sl. Sve takve funkcije definirane su u bibliotekama koje dolaze uz C prevodilac.
- Naredbe za kontrolu toka: **for** petlje, **if-else** konstrukcije, **case** (**switch**) naredbe i **while** petlje.
- Bitwise kontrola operatori za direktno upravljanje stanjima BIT-ova i BYTE-ova.
- Pointeri. Varijable koje su pokazivači na adrese drugih varijabli. Programski jezik C uključuje operatore za manipuliranje adresama varijabli (pointer arithmetic, pointer manipulation)



Slika 1. Razine programskih jezika

Struktura jezika

C je strogo deklarativan programski jezik koji se sastoji od 32 ključne riječi. Osnovu čini funkcija **main**, naredbe pretprocesoru (uključivanje pojedinih datoteka, itd), deklaracija funkcija koje se koriste u programu (ANSI standard) te deklaracija globalnih varijabli i konstanti. Na kraju svake izvršne naredbe potrebno je upisati znak **;** . Početak i kraj svake funkcije se označuje vitičastim zagradama **{** (početak) i **}** (kraj funkcije). Komentari se pišu između oznaka **/*** (početak komentara) i ***/** (kraj komentara)

```
# pretprocesorske_direktive

definicije_funkcija;

deklaracije_globalnih_varijabli_i_konstanti;

tip_podataka main (lista_argumenata)

{

        /* komentar */

        naredbe;

}
```

Primjer :

```
# include <stdio.h>

float func (float a);

float pi = 3.14;

void main (int argc, char **argv)

{

        int b; /* deklaracija varijable b */

        b = func (20.4);

}

int func (float a)

{

        return (a * pi);

}
```

Tipovi podataka

Programski jezik C podržava 5 osnovnih tipova podataka: znakovni (**char**), cjelobrojni (**int**), realni (**float**, **double**), bez vrijednosti (**void**). Osim navedenih moguće je definirati i vlastite tipove podataka uporabom naredbe **typedef**.

Osnovni tipovi podataka:

Tip	*Veličina u bitovima*	*Područje vrijednosti*
char	8	-128 do 127
int	16	-32.768 do 32.767
float	32	3.4E-38 do 3.4E+38
double	64	1.7E-308 do 1.7E+308

Pomoću prilagodnika moguće je promijeniti osnovne tipove podataka tako da bolje odgovaraju potrebama. Kao prilagodnici se koriste sljedeće ključne riječi: signed, unsigned, long, short.

Dodatni tipovi podataka:

Tip	*Veličina u bitovima*	*Područje vrijednosti*
unsigned char	8	0 do 255
signed char	8	-128 do 127
unsigned int	16	0 do 65.535
signed int	16	-32768 do 32767
short int	16	-32768 do 32767
unsigned shor int	8	0 do 65535
signed short int	8	-32768 do 32767
long int	32	-2147483648 do 2147483647
signed long int	32	-2147483648 do 2147483647
unsigned long int	32	0 do 4294967295
long double	128	1.7E-308 do 1.7E+308

Konstante

Konstante su svi podaci koji se ne mijenjaju tijekom izvođenja programa, tj. podaci na koje program ne može utjecati.

Konstante osnovnih tipova podataka:

Tip podataka	*Primjer*
int	1, 123, 21000, -234
long int	35000L, -34L
short int	10, -12, 90
unsigned int	10000U, 987U, 40000
float	123.23F, 4.34e-3F
double	123.23, 12312333, -0.9876324
long double	1001.2L

Heksadecimalne i oktalne konstante: 0**x**80, **O**12

Znakovne konstante:

Bilo koji skup znakova koji se navede između dvostrukih navodnika smatra se znakovnom konstantom ili stringom (npr. ?ovo je tekst?). Za razliku od stringa karakter konstante se zapisuju između jednostrukih navodnika (npr. ?c?), a pri zapisu znakovne konstante ne može biti više od jednog znaka.

Posebne konstante:

Kod	*Značenje*	*Kod*	*Značenje*
\b	Backspace	**\?**	Jednostruki navodnik
\f	Sljedeća stranica	**\0**	Nula
\n	Novi redak	****	Backslash
\r	Carriage return	**\v**	Vertikalni tab
\t	Horizontalni tab	**\a**	Upozorenje (alert)
\?	Dvostruki navodnik	**\N**	Oktalna konstanta

Varijable

Imena varijabli moraju započeti slovom, maksimalan broj znamenki u imenu varijable može biti 31 (no pojedini kompajleri dozvoljavaju i više), velika i mala slova imaju utjecaj

prilikom imenovanja varijabli (promjer != Promjer != pRomjer, itd.). Sve varijable koje se koriste u programu moraju se deklarirati.

Način deklaracije varijabli:

tip_podatka lista_varijabli;

Primjer:

```
int i;

double a, promjer, radius;
```

Lokalne varijable

Lokalne varijable se deklariraju unutar funkcije tako da je njihova vrijednost ograničena na granice funkcije u kojoj su deklarirane. Vrijednost lokalne varijable nakon izlaska iz funkcije se poništava.

Primjer:

```
# include <stdio.h>

void main ()

{

        int x;

        x = 10;

        printf ("x = %d\n",x);

}
```

Globalne varijable

Za razliku od vrijednosti lokalnih varijabli, vrijednosti globalnih varijabli su dostupne iz bilo kojeg dijela programa, bilo da se radi o dijelu koda koji se nalazi u istoj datoteci (gdje su globalne varijable deklarirane) ili se radi o dijelovima koda zapisanog u drugim datotekama.

Primjer:

```
# include <stdio.h>

int gbroj; /* deklaracija globalne varijable */

void main ()

{
```

```
        gbroj = 3;

        printf ("gbroj = %d\n",gbroj);

}
```

Konstante

Konstante se deklariraju pomoću ključne riječi **const**. Vrijednost varijabli deklariranih kao konstantne ne može se mijenjati tijekom izvođenja programa.

Primjer:

```
# include <stdio.h>

void main ()

{

        const float pi = 3.14;

        printf ("pi = %f\n",pi);

        pi = 20.8; /* greska */

}
```

Vanjske varijable

Ključna riječ **extern** određuje globalne varijable koje su deklarirane u nekoj drugoj datoteci. Datoteka koja sadrži delaraciju globalne varijable će biti (nakon kompajliranja) povezana zajedno s ostalim programskim kodom.

Primjer:

```
extern int radius;
```

Statičke varijable

Statičke varijable su one varijable koje zadržavaju svoju vrijednost i nakon što program "napusti" funkciju unutar koje je varijabla deklarirana. Statičke varijable se deklariraju uporabom ključne riječi **static**.

Primjer:

```
#include <stdio.h>

void func1 (void)

{
```

```
static int c;
        printf ("c = %d\n",c);
        c = 45;
        printf ("c = %d\n",c);
}
void main (void)
{
    func1 ();
        func1 ();
}
```

Ukoliko je globalna varijabla deklarirana kao statička, vrijednost te varijable je poznata samo unutar datoteke u kojoj je deklarirana. Što znači da, iako je varijabla deklarirana kao globalna, nije dostupna iz funkcija koje su zapisane u drugim datotekama.

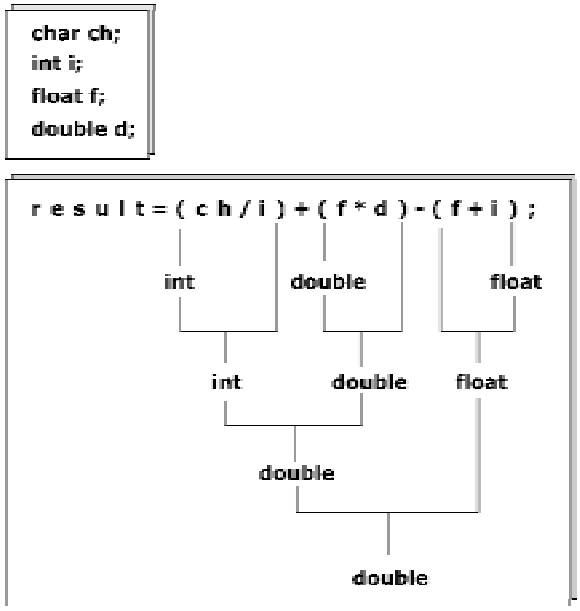
Slobodne varijable

Uporabom ključne riječi **volatile** mogu se deklarirati varijable čija se vrijednost može mijenjati neovisno o programu (npr. adresa nekih parametara operativnog sustava).

Primjer:

```
const volatile unsigned char *port = 0x30;
```

Konverzija tipova podataka



Slika 2. Konverzija tipova podataka

Operatori

Operator dodijeljivanja

Operator dodijeljivanja je **=** , dodijeljivanje vrijednosti se izvršava s desna na lijevo.

Primjer:

```
a = 42;

b = c = d = 18.342;
```

Aritmetički operatori

+	zbrajanje
-	oduzimanje, unarni minus
*****	množenje
/	dijeljenje
%	modul
--	dekrement
++	inkrement

Primjer:

```
a = b + c;

k = 100 ? m;

J = 20 / c;

M = -1 * 30;

P = x % y;


x = 10;  /* x = 11 */

y = ++x;  /* y = 11 */


x = 10; /* x = 11 */

y = x++; /* y = 10 */
```

Operatori uspoređivanja

>	veće od
>=	veće ili jednako
<	manje od
<=	manje ili jednako
==	jednako
!=	različito

Logički operatori

&&	konjunkcija (AND)
||	disjunkcija (OR)
!	negacija (NOT)

U programskom jeziku C istina (true) je bilo koja vrijednost različita od nule, dok je laž (false) vrijednost jednaka nuli.

Primjer:

```
#include <stdio.h>

int xor (int a, int b);

void main (void)

{

   printf ("%d\n", xor(0,0));

   printf ("%d\n", xor(1,0));

   printf ("%d\n", xor(0,1));

   printf ("%d\n", xor(1,1));

}

int xor (int a, int b)

{

   return ((a || b) && !(a && b));

}
```

Bitwise operatori

Budući da je C jezik srednje razine, što u nekim slučajevima znači da može zamijeniti strojni jezik (assembler), posjeduje skup operatora koji se mogu uporabiti za operacije nad bitovima.

&	konjunkcija (AND)
|	disjunkcija (OR)
^	XOR
~	komplement
<<	pomak desno (right shift)
>>	pomak lijevo (left shift)

Operator zarez

Uporabom operatora zarez moguće je stvoriti složene izraze u jednoj liniji.

Primjer:

```
x = (y = 3, y + 1);
```

Prvo se vrijednost 3 dodijeli varijabli y, zatim se vrijednost varijable y (3) poveća za jedan, te se ta vrijednost (4) dodijeli varijabli x.

Ulazno izlazne naredbe

Programski jezik C ima jedinstveni način pristupa unosu i ispisu podataka. Naime, u jeziku nisu definirane nikakve ulazno izlazne naredbe već se ulazno izlazne operacije obavljaju putem funkcija deklariranih i definiranih u bibliotekama. **stdio.h** je biblioteka koja se mora uključiti da bi se ulazno/izlazne funkcije mogle rabiti u programu (#include <stdio.h>).

Ispisivanje i učitavanje znakova

Učitavanje i ispisivanje jednog znaka ostvaruje se uporabom naredbi **getchar()** i **putchar()**. Funkcije su deklarirane na sljedeći način:

```
int getchar (void);

int putchar (int c);
```

Funkcija **getchar()** vraća cjelobrojnu vrijednost, no low-order byte sadrži učitani znak. Unatoč činjenici da je argument funkciji **putchar()** deklariran kao cjelobrojna vrijednost, low-order byte sadrži vrijednost znaka koji se želi ispisati. Funkcija **putchar()** nakon ispisa

znaka vraća znak, dok u slučaju greške vraća **EOF** kod (vrijednost ?1). Cjelobrojne vrijednosti se koriste zbog kompatibilnosti sa izvornom verzijom jezika.

Primjer:

```
#include <stdio.h>

void main (void)

{

        char ch;

        printf ("Upisite znak: \n");

        ch = getchar ();

        printf ("Upisali ste znak ");

        putchar (ch);

        putchar ('\n');

}
```

Potencijalni problemi pri uporabi naredbe **getchar()** leže u tome što se znakovi upisuju u "buffer" sve dok se ne pritisne tipka *ENTER,* što može predstavljati problem kod nekih interaktivnih aplikacija. Osim funkcije **getchar()**, u svrhu učitavanja znakova, moguće je uporabiti i sljedeće funkcije **getch()** i **getche()** (ove funkcije nisu podržane prema ANSI standardu). Funkcije su deklarirane na sljedeći način:

```
int getche(void);

int getch(void);
```

Funkcija **getch()** reagira na svaki unos znaka ne čekajući ENTER, tijekom unosa znakovi se ne prikazuju na ekranu. Funkcija **getche()** se ponaša na isti način s tom razlikom što prilikom unosa prikazuje uneseni znak na ekranu računala.

Pimjer:

```
#include <stdio.h>

#include <conio.h>

void main (void)

{

        char ch;

        printf ("1. Unesite znak:");
```

```
        ch = getch();

        printf ("2. Unesite znak:");

        ch = getche();

}
```

Uporabom funkcije **gets()** moguće je učitati skup znakova sve dok se ne pritisne tipka ENTER. Znak za novi redak ne postaje dio učitanog stringa već se na kraj stringa upisuje znak "null" koji označava kraj učitanog skupa znakova. String ili skup znakova se na ekran ispisuje uporabom funkcije **puts()**. Funkcija **puts()** vraća ispisani broj znakova. Funkcije su deklarirane na sljedeći način:

```
char *gets(char *str);

int puts(char *str);
```

Primjer:

```
#include <stdio.h>

void main (void)

{

        char str[80];

        puts ("Unesite string:");

        gets (str);

        puts ("Unijeli ste sljedeci string:");

        puts (str);

}
```

Ispis i unos podataka pomoću formata

U tablici su prikazani načini određivanja formata za učitavanje ili ispisivanje pojedinih tipova podataka.

%c	znakovi
%d	cjelobrojne vrijednosti (signed)
%i	cjelobrojne vrijednosti (signed)
%e	eksponencijalni prikaz
%E	eksponencijalni prikaz

%f	realne vrijednosti
%g	realne vrijednosti (uporaba %f ili %e)
%G	realne vrijednosti (uporaba %F ili %E)
%o	oktalne vrijednosti
%s	stringovi
%u	cjelobrojne vrijednosti (unsigned)
%x	heksadecimalne vrijednosti (mala slova)
%X	heksadecimalne vrijednosti (velika slova)
%p	prikaz pointera
%%	ispis znaka %

Funkcija **printf()** se koristi za formatirani ispis podataka. Nakon ispisa funkcija vraća broj ispisanih znakova ili vrijednost -1, što znači da je došlo do greške prilikom ispisa. Funkcija je deklarirana na sljedeći način:

```
int printf(char *format,lista);
```

Primjer:

```
#include <stdio.h>

void main (void)

{

    double br;

    printf ("Crtaona %c%d je %s \n",'S',20,"zauzeta");

    br = 1000000;

    printf ("br= %g\n",br);

    br = 10000000;

    printf ("br= %g\n",br);

    br = 10.1234;

    printf ("br= %f\n",br);

    printf ("br= %10f\n",br);
```

```
        printf ("br= %012f\n",br);

        printf ("br= %5.2f\n",br);

        printf ("br= %15.5f\n",br);

}
```

Učitavanje podataka se ostvaruje uporabom funkcije **scanf()**. Funkcija vrača broj učitanih znakova ili EOF znak (-1) u kom slućaju se radi o grešci prilikom učitavanja. Prilikom upisa liste varijabli čije se vrijednosti žele učitati mora se imati na umu da se funkciji kao argument mora proslijediti adresa varijable, a ne ime varijable. Funkcija je deklarirana na sljedeći način:

```
int scanf(char *format,lista);
```

Primjer:

```
#include <stdio.h>

void main(void)

{

        char str[80];

        printf ("Upisite string: ");

        scanf ("%s",str);

        printf ("\nUpisali ste sljedeci string: %s\n",str);

}
```

```
#include <stdio.h>

void main(void)

{

        int a, b;

        printf (" a = ");

        scanf ("%d",&a);

        putchar ('\n');
```

```
        printf (" b = ");
        scanf ("%d",&b);
        printf ("\n %d + %d = %d \n",a,b,a + b);
}

#include <stdio.h>
void main (void)
{
        char str[80];
        printf ("Upisite skup znakova: ");
        scanf ("%[abcdefg]",str);
        printf ("\nUcitani znakovi su: %s\n",str);
}
```

Polja i stringovi

Polje je skup varijabli istog tipa na koje se može referencirati preko istog imena. Svakom pojedinačnom elementu polja može se pristupiti preko indeksa. Jedan od najčešće korištenih i najjednostavnijih polja je string tj. skup znakova. U C-u prvi element polja ima indeks 0. Polja se deklariraju na sljedeći način:

```
tip ime_varijable[broj_znakova];
```

Primjer:

```
int broj[10];
float coord[20][34];
```

Stringovi se deklariraju pomoću ključne riječi **char**. Svaki string kao posljednji znak ima znak NULL ('\0') koji označava kraj stringa. Zbog navedenog svaki string se mora deklarirati jedan znak više od maksimalnog broja znakova kojeg želimo upisati. Stringovi se deklariraju na isti način kao i polja:

```
char var[broj_znakova];
```

Primjer:

```
char prezime[20];
```

```
char imena[10][20];
```

Polja i stringovi se inicijaliziraju na sljedeći način:

```
tip var[broj_znakova]={lista_elemenata};
```

Primjer:

```
int i[10]={1,2,3,4,5,6,7,8,9,0};

char ime[7]="Zlatko";

char tmp_ime[5]={'p','e','r','o','\0'};

char prezime[]="Filipovic";
```

Kontrola tijeka programa

if naredba

Opai oblik **if** naredbe je:

```
if (logički_izraz) naredba1;

else naredba2;
```

Logički izraz je bilo koji izraz koji sadrži kombinaciju varijabli i/ili konstanti i/ili logičkih operatora. Uporaba ključne riječi nije obvezatna već ovisi o strukturi programa.

U slučaju potrebe izvršavanja više naredbi koristi se **if** blok. Način uporabe **if** bloka je:

```
if (logički_izraz) {

   prvi_blok_naredbi;

} else {

   drugi_blok_naredbi;

}
```

Jedan **if** blok može sadržavati proizvoljan broj drugih **if** blokova.

Primjer:

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

```
# include <stdlib.h>
void main (void)
{
   int broj;
   int slucajni;
   slucajni = rand ();
   printf (" Pogodite broj: ");
   scanf ("%d",&broj);
   if (broj == slucajni) printf ("\n*** Tocno ***\n");
   else printf (" \n*** Netocno***\n");
}
# include <stdio.h>
# include <stdlib.h>
void main (void)
{
   int broj;
   int slucajni;
   slucajni = rand();
   printf ("Pogodite broj: ");
   scanf ("%d", &broj);
   if (broj == slucajni)
   {
      printf ("\n*** Tocno ***\n");
      printf ("Slucajni broj je %d\n",slucajni);
   }
   else
```

```
    {
        printf ("\n*** Netocno ***\n);
        if (slucajni > broj) puts ("Broj je premalen");
        else puts ("Broj je prevelik");
    }
}
```

Kao alternativa **if** naredbi moguće je uporabiti operator **?** na sljedeći način:

```
izraz1 ? izraz2 : izraz3;
```

U slueaju da je izraz1 istinit izvršit će se izraz2, a u suprotnom tj. ako je izraz1 neistinit izvršit će se izraz3.

Primjer:

```
x = 10;
y = x > y ? 100 : 200;
```

što je identično izrazu:

```
if (x > y) y = 100;
else y = 200;
```

```
# include <stdio.h>
# include <math.h>
void main (void)
{
    double kor,i;
    printf ("Upisite broj:");
    scanf ("%lf",&i);
    kor = i > 0 ? sqrt (i) : puts (?Broj je negativan.?);
    if (i > 0)
        printf (?\n Korijen broja %lf iznosi %lf\n",i,kor);
```

```
}


# include <stdio.h>

int f1 (int n);

void main (void)

{

    int t;

    printf ("Upisite broj: ");

    scanf ("%d",&t);

    t ? f1(t) : puts ("\n Upisali ste nulu\n");

}


int f1(int n)

{

    printf ("Upisali ste %d", n);

    return 0;

}
```

switch naredba

switch naredba omogućuje višestruko grananje u programu ovisno o vrijednosti izraza koji se ispituje. U **switch** naredbi izraz se može testirati jedino na jednakost. Unutar tijela naredbe ista se vrijednost konstante smije pojaviti samo jednom. Opći oblik **switch** naredbe je:

```
switch (izraz){

    case konstanta1:

        blok_naredbi;

        break;

    case konstanta2:
```

```
        blok_naredbi;
        break;
        .
        .
        .
    default:
        blok_naredbi;
    }
```

Primjer:

```
# include <stdio.h>
void main (void)
{
    char ch;
    printf ("1. Zagreb\n");
    printf ("2. Rijeka\n");
    printf ("3. Split\n");
    printf ("Izberite odrediste: ");
    ch = getchar ();
    switch (ch){
    case '1':
        puts ("Izabrali ste Zagreb");
        break;
    case '2':
        puts ("Izabrali ste Rijeku");
        break;
    case '3':
```

```
            puts ("Izabrali ste Split");
            break;
        default:
            puts ("Niti jedno odrediste nije odabrano");
        }
    }
```

Petlje

for petlja

Opći oblik **for** petlje je:

```
for (inicijalizacija;uvijet;naredba) naredba;
```

Primjer:

```
# include <stdio.h>
void main (void)
{
    int x;
    float z;
    for (x = 1; x <= 10; x++) printf ("<< %d >>\n",x);
    for (x = 100; x != 65; x -= 5)
    {
        z = (float) x * x; /* cast operator (float)*/
        printf (" %d na kvadrat iznosi %f \n", x, z);
    }
}
```

Beskonačna **for** petlja određuje se na taj način da se izostavi inicijalicija, uvijet i inkrement:

```
for (;;) naredba;
```

while petlja

Broj izvršavanja **while** petlje određen je logičkim uvijetom. Opći oblik određen je na sljedeći način:

```
while (logicki_uvijet) naredba;
```

Naredba ili blok naredbi će se izvršavati dok je logički uvijet istinit. Istina je određena kao bilo koja vrijednost veća od nule. Logički uvijet se ispituje prilikom ulaska u petlju te na početku svakog prolaza tj. izvođenja petlje.

Primjer:

```
# include <stdio.h>

void main (void)

{

   double broj, suma = 0.0;

   int brojac = 0;

   printf ("Upisite broj\n");

   printf ("Negativan broj prekida unos!\n");

   printf ("broj %d = ", brojac++);

   scanf ("%lf",&broj);

   while (broj >= 0.0)

   {

      suma += broj;

      printf ("\nbroj %d = ",brojac++);

      scanf ("%lf",&broj);

   }

   if (suma > 0)

      printf ("Prosjecna vrijednost = %lf\n",suma/brojac);

   else
```

```
        puts ("Vrijednosti nisu unesene.");
    }
```

do while petlja

do while petlja je slična **while** petlji s tom razlikom što se logički uvijet u **do while** petlji ispituje na kraju petlje, što znači da će naredba ili blok naredbi unutar **do while** petlje biti izvršeni najmanje jednom. Opći oblik **do while** petlje je:

```
do {
   blok_naredbi;
} while (logicki_uvijet);
```

Primjer:

```
# include <stdio.h>
# define UP 10.0
# define DOWN 1.0
void main (void)
{
   double var;
   do {
      printf ("Upisite broj izmedju %lf i %lf:",UP,DOWN);
      scanf ("%lf", &var);
      putchar ('\n');
      printf (">> %lf - DOWN = %lf\n",var, var - DOWN);
      printf (">> UP - %lf = %lf\n",var, UP - var);
      } while ((var < UP) && (var > DOWN));
   puts ("*** Kraj ***");
}
```

Naredba **define** je direktiva pretprocesoru kojom se mogu odrediti vrijednost i naziv konstanti u programu. Kompajler umjesto definiranih naziva konstanti koristi njihovu vrijednost.

Funkcije

Funkcije čine osnovu C programskog jezika. Ovisno o načinu uporabe, funkcije se mogu ponašati kao potprogrami (povrat više podataka) ili kao funkcije (povrat jednog podatka). Svaka funkcija se mora obvezatno deklarirati (tj. odrediti tip podatka koji funkcija vraća).

```
tip_podatka naziv_funkcije (lista_argumenata)

{

   naredba1;

   naredba2;

   ...

}
```

Definicija funkcije može ujedno biti i njena deklaracija, ukoliko se tijelo funkcije napiše prije funkcije **main**. U suprotnom potrebno je uz definiciju funkcije, funkciju i deklarirati na isti način kao i varijable, prije funkcije **main**.

Primjer:

```
# include <stdio.h>

# include <stdlib.h>

int zbroj (int a, int b);

void poruka (char *msg)

{

   puts (msg);

}

void main (void)

{

   printf ("Zbroj = %d \n", zbroj (2,7));

   poruka ("Gotovo");

}
```

```
int zbroj (int a, int b)
{
   return (a + b);
}
```

Funkcija može, ali i ne mora vratiti podatke. U slučaju da funkcija ne vraća podatke mora se deklarirati kao tip **void**. Ukoliko funkcija ne treba argumente tada se lista agrumenata ostavlja prazna, no okrugle zagrade moraju se navesti prilikom deklaracije, definicije i poziva funkcije.

Funkciji se argumenti mogu proslijediti na dva načina: prijenosom vrijednosti (Call by value) i prijenosom adrese (Call by reference). Prilikom prenosa argumenata po vrijednosti podaci se kopiraju u lokalne varijable koje se potom obračuju unutar funkcije. Promjena vrijednosti lokalnih kopija nema utjecaj na vrijednosti varijabli pozivajuće funkcije.

Primjer:

```
#include <stdio.h>
int sqr (int x);
void main (void)
{
   int t = 10;
   printf ("sqr (%d) = %d\n", t, sqr(t));
}
sqr (int x)
{
   x = x * x;
   return (x);
}
```

Prijenos argumenata po referenci ostvaruje se uporabom pointera. U tom slučaju se funkciji kao argument ne šalje vrijednost podataka već vrijednost memorijske adrese na kojoj se podatak nalazi. Na taj način svaka lokalna (unutar funkcije) promjena vrijednosti podataka utiče i na vrijednost podataka u pozivajućem dijelu programa.

Primjer:

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

```
void swap (int *x, int *y)
{
    int tmp;
    tmp = *x;
    *x = *y;
    *y = tmp;
}
void main (void)
{
    int x, y;
    x = 10;
    y = 20;
    printf ("Prije funkcije x = %d, y = %d \n", x,y);
    swap (&x,&y);
    printf ("Poslije funkcije x = %d, y = %d \n", x,y);
}
#include <stdio.h>
#include <ctype.h>
void print_up( char *str);
void main (void)
{
    char str[80];
    gets (str);
    print_up(str);
}
void print_up (char *str)
```

```
{
   register int t;
   for (t = 0;str[t];++t)
   {
      str[t] = toupper (str[t]);
      putchar (str[t]);
   }
   putchar ('\n');
}
```

Programski jezik C dozvoljava prosljeđivanje podataka i glavnoj funkciji **main**.

Primjer:

```
#include <stdio.h>
#include <stdlib.h>
void main (int argc, char *argv[])
{
   if (argc != 2)
   {
      puts ("Zaboravili ste upisati svoje ime!");
      exit(1)
   }
   printf ("Dobar dan %s\n", argv[1]);
}
```

argc sadrži broj argumenata u komandnoj liniji. Polje **argv** sadrži vrijednosti argumenata navedenih u komandnoj liniji. **argv[0]** najčešće je naziv samog programa.

Primjer:

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

```
#include <ctype.h>
#include <string.h>
void main (int argc, char *argv[])
{
   int disp, count;
   if (argc < 2)
   {
      puts ("Uporaba: naziv_programa broj");
      exit (1);
   }
   if (argc == 3 && !strcmp (argv[2],"vvvvvvvv")) disp = 1;
   else disp = 0;
   for (count = atoi (argv[1]); count; --count)
      if (disp) printf (">%03d<\n",count);
   putchar (7);
   puts (?Gotovo.?);
}
```

Postoje dva načina na koja funkcija može završiti izvođenje i vratiti podatke funkciji. Prvi način je kada se izvrši zadnja naredba u funkciji.

Pimjer:

```
#include <stdio.h>
#include <string.h>
void unazad (char *str);
void main (void)
{
   unazad ("Ovo je tekst.");
}
```

```
void unazad (char *str)
{
   register int t;
   for (t = strlen (str) ? 1;t >= 0;t--)
      putchar (str[t]);
   putchar ('\n');
}
```

Drugi način je uporabom ključne riječi return. Na ovaj način se prenose podaci iz aktivne funkcije u pozivajuću, te se ujedno i završava aktivna funkcija. Ključna riječ **return** može se pojaviti proizvoljan broj puta u funkciji.

Primjer:

```
#include <stdio.h>
int nadji (char *s1, char *s2)
{
   register int t;
   char *p, *p2;
   for (t = 0;s1[t];t++)
   {
      p = &s1[t];
      p2 = s2;
      while (*p2 && *p2 == *p)
      {
         p++;
         p2++;
      }
      if (!*p2) return t;
   }
```

```
    return -1;
}
void main (void)
{
    if (nadji ("Ovo je tekst","je") != -1)
        puts ("Podstring je nadjen.");
}
```