Programski jezik C

Istorijski razvoj i evolucija C-a

- Projektovan 1970. godine kao sastavni deo operativnog sistema UNIX
- 1978. godine – “The C programming Language” D.M.Ritchi, B.W.Kernighan
  Definisan kao jezik opšte namene
- 1983. godine osnovan ANSI komitet za standardizaciju C jezika (otklonjene mnoge neodredjenosti i izmenjeni neki koncepti)
- 1989. godine usvojen standard
- Viši programski jezik sa nekim specifičnostima koje ga približavaju asemblerskim jezicima
- Dozvoljava operacije niskog nivoa:
  - nedostatak (slabi pouzdanost sistema)
  - prednost (sprega niskog nivoa sa resursima sistema)

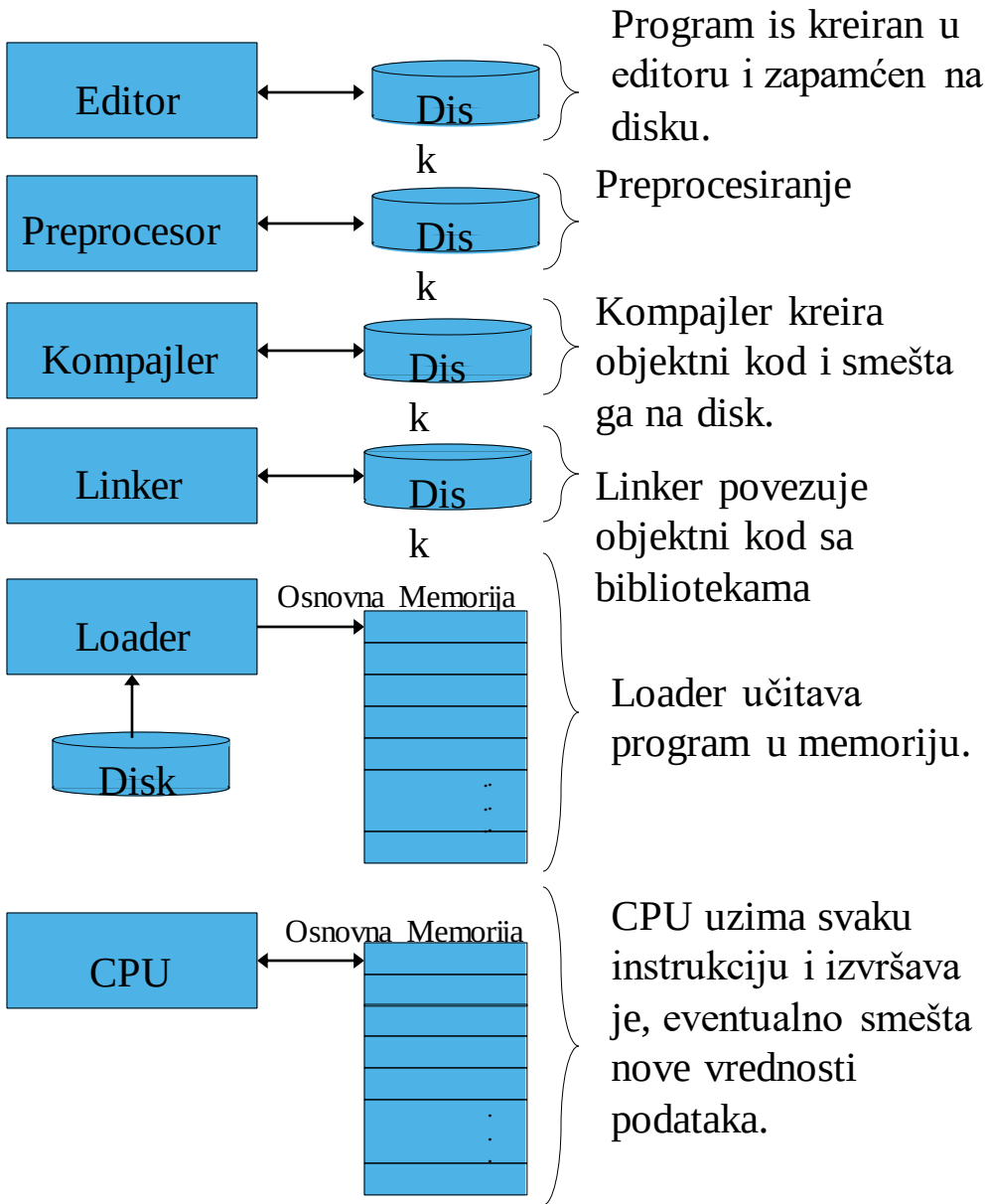
- C je jezik opšte namene tesno povezan sa UNIX OS uz koji je razvijan.
- Prvu verziju razvio Denis Ritchie u Bell Laboratories, USA
- C je portabilan, ne vezuje se za OS ili hardware.
- Strukturni programski jezik
- Programi napisani u C-u su efikasni, uglavnom manji i brži od programa pisanih u drugim programskim jezicima.
- C je fleksibilan jezik sa širokom primenom:
  - CAD, igre, komercijalni sistemi, Veštačka inteligencija (ekspertni sistemi, robotika)
  - Upravljanje procesima, real-time sistemi
  - Sistemski softver (operativni sistemi, kompajleri, linkeri itd.)

- Ne postoje operacije za direktnu manipulaciju sa složenim objektima (stringovi, liste, polja)
  - Ne postoji operacija za manipulaciju sa celim poljem ili stringom
- Ne postoji direktna podrška za ulaz i izlaz -
  - Ne postoje READ i WRITE naredbe
- Nije ugradjen pristup fajlovima tj. ne postoje metode pristupa
- C nije strogo tipiziran jezik
- Ne postoji automatska konverzija neusaglašenih tipova
- C je relativno mali jezik, opisan kratko i uči se brzo.
- Ima reputaciju da je težak za učenje.
- **Većina koncepata u C-u se nalaze i u Pascalu.**
  - **Funkcije, Povezane liste, polja, parametri**

Razvoj C Programa

Faze:

1. *Editovanje*
2. *Preprocesiranje*
3. *Kompilacija*
4. *Linkovanje*
5. *Učitavanje(Load)*
6. *Izvršenje*



Azbuka C-a i struktura C programa

- **Azbuka programskog jezika C:**
  - velika i mala slova engleske azbuke,
  - dekadne cifre i
  - specijalni znaci.

- **Tokeni jezika**

  Reč je niz znakova koji predstavljaju elementarnu logičku celinu u jednom programskom jeziku.

  Vrste reči u C-u:
  - identifikatori,
  - ključne reči,
  - separatori,
  - konstante,
  - literali (konstantni znakovni nizovi),
  - operatori.

• Osim reči koje imaju svoje odredjeno značenje u programu, postoje i delovi programa koje kompajler jednostavno ignoriše, a koji služe programeru da pojasne kod.

• To su **komentari**.

• Komentari se u programskom jeziku C pišu:
- izmedju simbola **/*** i ***/**
- počinju simbolom **//** i protežu se do kraja programske linije.

• Svaka naredba se završava sa **;**

• Jedna naredba po liniji programskog koda

Identifikatori (simbolička imena)

- Identifikator - niz slova, cifara i '_' u kojem prvi znak ne može biti cifra.

- Identifikatori služe da imenuju sledeće elemente programa:
  - promenljive,
  - simboličke konstante,
  - funkcije,
  - korisničke tipove podataka i
  - labele.

- 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.

Ključne reči

Reči čije je značenje unapred definisano i ne mogu se koristiti kao identifikatori

auto	extern	sizeof
break	float	static
case	for	struct
char	goto	switch
const	if	typedef
continue	int	union
default	long	unsigned
do	register	void
double	return	volatile
else	short	while
enum	signed	

Ključne reči u programskom jeziku C pišu se **malim slovima**.

Separatori

Separatori su posebni specijalni znaci koji imaju odredjeno (unapred definisano) sintaksno i semantičko značenje u programu, a ne označavaju nikakvu operaciju.

U ovu grupu reči spadaju:

{
}
(
)
;
:
…

Tipovi podataka

U programskom jeziku C definisani su samo numerički tipovi podataka

Tipovi za predstavljanje celobrojnih podataka

- int – celobrojni podatak za čije se predstavlajnje koriste 16 ili 32 bita (zavisno od hardvera računara).
- char – mali celobrojni podatak (dužine 1 bajt) koji može da primi i kod jednog znaka pa se koristi i za predstavljanje znakovnih podataka.

Tipovi za predstavljanje realnih vrednosti

- float – realni podatak u jednostrukoj tačnosti (32 bita)
- double – realni podatak u dvostrukoj tačnosti (64 bita)

Varijante navedenih tipova podataka se dobijaju dodavanjem ključnih reči
- **short**,
- **long**,
- **signed** ili
- **unsigned**

ispred oznake tipa.

short - memorijski prostor za predstavljanje podatka se smanjuje na pola,

long - memorijski prostor za predstavljanje podatka se udvostručava,

unsigned - kaže da se radi o neoznačenom podatku,

signed – kaže da se radi o označenom podatku
(podrazumeva se (po *default*-u) svi su podaci označeni).

Logičke vrednosti se predstavljaju takođe numeričkim podacima.

- 0 odgovara logičkoj vrednosti *false*, a
- svaka vrednost različita od nule logičkoj vrednosti *true*.

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

Konstante

Celobrojne konstante:

- dekadni brojni sistem
- heksadekadni
- oktalni

- Decimalna celobrojna konstanta je niz dekadnih cifara u kojem prva cifra ne sme biti 0.

- Oktalna celobrojna konstanta je niz oktalnih cifara u kojem je prva cifra 0.

- Heksadekadna celobrojna konstanta je niz heksadekadnih cifara koji počinje prefiksom '0x' ili '0X' u kojem heksa cifre mogu biti mala ili velika slova.

Po *default*-u celobrojne konstante su tipa int.

Dodavanjem sufiksa l ili L naglašava se da se radi o konstanti tipa long int, a dodavanjem sufiksa u ili U obeležavaju se konstante tipa unsigned int.

Ova dva sufiksa se mogu kombinovati.

Primeri:

```
123, 987, -765        //decimalne konstante tipa int
0456, -0547           //oktalne konstante tipa int
0x1a2, 0XA0F          //hexadekadne konstante tipa int
0x1a2u                //hexa konstanta tipa unsigned int
8987655ul             //dekadna konstanta tipa unsigned long
```

Realne konstante

- Realne konstante se pišu u dekadnom brojnom sistemu i mogu biti predstavljene u:
  - fiksnom zarezu
  - eksponencijalnom obliku.

- Zapis u fiksnom zarezu obavezno sadrži decimalnu tačku i to tačno tamo gde je njeno stvarno mesto u broju (npr. 34.67, 56., .98).

- Za veoma velike ili veoma male brojeve koristi se eksponencijalni zapis.
- Eksponencijalni zapis je oblika:

  *m*E*e* gde *m* predstavlja mantisu broja, a *e* eksponent
  (npr. 2.23e2, -0.5, .2E-2, 7e-23)

- Po *default*-u realne konstante su tipa double.
- Dodavanjem sufiksa:

  F ili f dobijaju se konstante tipa float,
  L ili l, konstante tipa long double.

Znakovne konstante

Znakovna konstanta je ceo broj čija je vrednost jednaka kodu znaka navedenog između jednostrukih navodnika (npr. 'a', '>',...).

Na primer, u ASCII skupu znakova, znakovna konstanta ‘5’ ima vrednost 53, ‘A’ ima vrednost 65.

kontrolni 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)	\"

Literali (konstantni znakovni niz)

- Literal je sažeti način zapisivanja niza znakovnih konstanti.

  Npr. **"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 (‘\0’), tzv. *null terminated symbol*.

Npr. literal **"Zdravo"** će se u memoriji zapisati kao **‘Z’ ‘d’ ‘r’ ‘a’ ‘v’ ‘o’ ‘\0’**

korektno

```
" a string of text"
""
"   "
"a = b+c;"
"/* this is not a comment  */"
"a string with double quotes \" within"
"a single backslash \\ is in this string"
"abd" "efg"
```

nekorektno

```
/* "this is not a string" */
"and
Neither is this "
'dgashgahg'
```

Operatori

- Operatori su simboli koji označavaju određenu operaciju i koji povezuju jedan ili više operanada u izraz.

- Podela po:
  - broju operanada
  - po funkcionalnosti

- U zavisnosti od **broja operanada** operatori u C-u se dele na:
  - unarne,
  - binarne,
  - ternarne.

Grupe operatora u odnosu na funkcionalnost:

- 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
- comma operator
- cast operator
- operatori referenciranja i dereferenciranja

Aritmetički operatori

- Imaju numeričke operande i rezultati su, takođe, numeričkog tipa.

Po prioritetu:

- Unarni operatori + i –
- Operatori inkrementiranja (++) i dekrementiranja (--)
- Operatori *, /, %
- Binarni operatori + i -

Unarni operatori + i -

- Unarni operator **-** menja znak operanda koji sledi.
- Ako je operand 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 (to znači da je on praktično operator bez dejstva).

Operatori inkrementiranja (++) i dekrementiranja (--)

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

To znači da:

Ukoliko se vrednost izraza ++a (ili --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++ (ili a--) koristi u nekom širem izrazu, izvršiće se izračunavanje vrednosti šireg izraza sa starom vrednošću promenljive a i nakon toga će se izvršiti promena vrednosti promenljive a.

Operatori *, /, %

***** - označava množenje;

/ - označava deljenje (u slučaju primene nad celobrojnim podacima predstavlja celobrojno deljenje, tj. u rezultatu se vrši jednostavno odbacivanje cifara iza decimalne tačke);

5 / 2 = 2

% - (po modulu) označava ostatak deljenja i primenjuje se isključivo nad celobrojnim podacima.

5 % 2 = 1

Binarni operatori + i -

+ - označava sabiranje,
– - označava oduzimanje

Primeri:

```
int a,b,c,d;
a=3; b=6;
c=a*--b;  //ekvivalentne naredbe: b=b-1; c=a*b;
d=a*b--;  //ekvivalentne naredbe: d=a*b; b=b-1;
```

Relacioni operatori

• Ovo su operatori za poređenje vrednosti operanada.

• Operandi u ovom slučaju mogu biti bilo kog numeričkog tipa ili pokazivači, a rezultat je logičkog tipa.

• Kako logički tip u C-u nije definisan, ukoliko je poređenje tačno (tj. ukoliko je relacija zadovoljena), rezultat je 1, u suprotnom rezultat je 0.

Ovoj grupi operatora pripadaju:

- **<** (manje),
- **<=** (manje ili jednako),
- **>** (veće),
- **>=** (veće ili jednako),
- **==** (jednako),
- **!=** (različito).

Prioritet prva 4 operatora je viši od prioriteta poslednja dva.

Logički operatori

U programskom jeziku C, operandi u logičkim operacijama mogu biti:

- numeričkog tipa
- tipa pokazivača.

S obzirom da su 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*), jedino 0 označava netačno (*false*).

U ovu grupu operatora spadaju:

!	(negacija),
&&	(konjunkcija, tj. logičko I),
||	(disjunkcija, tj. logičko ILI).

prioritet

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

Operatori za rad sa bitovima

Izvođenje operacija nad bitovima unutar celobrojnih podataka.
Ovakvi operatori su definisanu u asemblerskim jezicima,
a od viših programskih jezika, jedino ih C podržava.

Ovoj grupi operatora pripadaju:

~ (nepotpuni (jedinični) komplement,) – komplementira se svaki bit ponaosob,

& (pokomponento logičko I),

^ (pokomponentno isključivo ILI),

| (pokomponentno logičko ILI),

<< (pomeranje ulevo) - vrši pomeranje bitova levog operanda za onoliko mesta ulevo kolika je vrednost desnog operanda,

>> (pomeranje udesno) - vrši pomeranje bitova levog operanda za onoliko mesta udesno kolika je vrednost desnog operanda.

Primeri:

```
int a,b;
a = 0723;          // mem. lok a: 0000000111010011
b = ~a;            // b: 1111111000101100
a = a << 4;        // a: 0001110100110000
a = a >> 3;        // a: 0000001110100110
b = a | 071;       // b: 0000001110111111)
```

Operator dodele

- Ovo je binarni operator koji smešta vrednost desnog operanda na lokaciju čije se ime nalazi sa leve strane operatora.
- Levi operand mora biti ime memorijske lokacije.

Postoje dve grupe ovakvog operatora:

- elementarni operator dodeljivanja (=),
- složeni operatori dodeljivanja (opšti oblik ovih operatora je **<op>=**)

Operator <op> može biti jedan od sledećih:

+, -, *, /, %, <<, >>, &, |, ^.

Opšti izraz a <op> = b se može napisati i u obliku a = a <op> b

Primeri:

```
a += 3;                    //ekvivalntno:    a = a+3;
b -= a;                    //ekvivalentno:   b = b-a;
proizvod *= a-5;           // proizvod = proizvod*(a-5);
```

Operator grananja

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

<izraz1> ?< izraz2> : <izraz3>

Dejstvo operatora:

- Izračunava se najpre vrednost izraza <izraz1>.
- Ako je vrednost ovog izraza **različita od nule** (*true*), izračunava se vrednost izraza <izraz2> i njegova vrednost je rezultat operacije.
- Ako je vrednost izraza <izraz1> **jednaka nuli** (*false*), izračunava se vrednost izraza <izraz3> i njegova vrednost je rezultat operacije.

Primeri:

```
maximum = ( a >= b)? a : b;
abs_x = ( x >= 0 ) ? x : -x;
```

sizeof operator

Svaka promenljiva ili konstanta zauzima određeni deo memorijskog prostora.

Veličina tog memorijskog prostora zavisi od tipa promenljive (konstante), tj. od internog načina predstavljanja podataka pojedinih tipova u memoriji.

sizeof - izračunava broj bajtova koji zauzima neki podatak ili tip u memoriji.

Ovaj operator je unaran i njegov opšti oblik je:

sizeof (<objekat>),

ili

sizeof (tip).

Primeri:

```
int broja, brojb;
broja = sizeof (short int);
brojb = sizeof (broja);
```

comma operator

- Ovo je binarni operator.
- Služi za formalno spajanje više izraza u jedan izraz.
- Levi i desni operand operatora "**,**" su izrazi, pri čemu je redosled njihovog izvršavanja s leva na desno. (Uglavnom se koristi kod poziva funkcija)

Primer:

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

cast operator

Koristi se za eksplicitno pretvaranje promenljive (ili izraza) jednog tipa u neki drugi tip.

Operator je unaran i opšti oblik izraza za konverziju tipa je:

(<tip>) <izraz>

gde je: <tip> - neki od tipova,
<izraz> - je promenljiva ili neki složeniji izraz čiji tip treba promeniti.

Primeri:

```
int   a;
float x;
double y;
... (char) a ...
y=sin((double) x);     //parametar f-je sin mora da bude tipa double
```

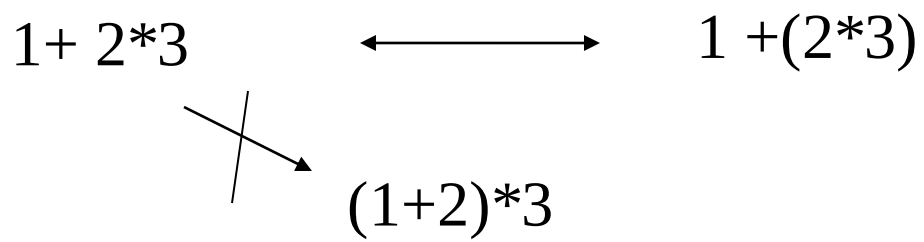
Prioritet operatora

- Prioritet operatora odredjuje redosled izvodjenja operacija kada u izrazu figuriše veći broj operatora.
- U izrazima se uvek izvršavaju operatori od operatora najvišeg prioriteta prema operatorima nižih prioriteta.
- Ovaj redosled se može promeniti korišćenjem zagrada.

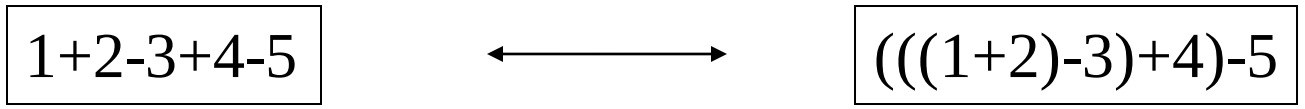
U tabeli su poređani operatori programskog jezika C
po njihovom prioritetu (počev od operatora najvišeg prioriteta
prema operatorima nižih prioriteta).

Prioriteti operatora u programskom jeziku C

1. ( ) [ ] . ->
2. unarni - ~ & | ++ -- sizeof
3. * / %
4. + -
5. << >>
6. < > <= >=
7. == !=
8. &
9. ^
10. |
11. &&
12. ||
13. ? :
14. = *op*=
15. ,



- S leva u desno (Left-to-right) kao pravilo



Operator	pravilo
() ++(postfix) --(postfix)	s leva u desno
+ (unary) – (unary) ++ (prefix) – (prefix)	s desna u levo
* / %	s leva u desno
+ -	s leva u desno
= += -= *= /= %= itd.	s desna u levo

-a*b-c ⟷ ((-a)*b)-c

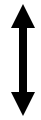
```
777++
++(a*b-1)
```

```
++i
cnt--
```

```
++a*b-c--                    ((++a)*b)-(c--)
7—b*++d                      7-((-b)*(++d))
```

```
variable=right_side (expr);

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



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

```
a=b=c=0;
```

```
a=(b=(c=0));
```

Struktura C programa

- C program predstavlja skup definicija:
  - promenljivih,
  - simboličkih konstanti,
  - 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.

Definisanje promenljivih

<tip> <ime> [=<poc_vred>] [,<ime> [=<poc_vred>]] ... ;

Primeri:

```
int a, b=5;
long int i;
double a, b=45.988675654;
```

Definisanje simboličkih konstanti

const <tip> <ime> = <vrednost> [,<ime> = <vrednost>] ... ;

Primeri:

```
const int a=3, b=5;
const float Pi=3.14;
```

Definisanje tipova

typedef <poznati_tip> <novo_ime>;

Primeri:

```
typedef float duzina;
typedef char slovo;
```

Funkcija main()

Minimalna definicija funkcije main je:

```
main()
{
        <telo_funkcije_main>
}
```

Telo funkcije main sadrži:
- definicije 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

- To su naredbe kojima se vrši neka elementarna obrada podataka.
- Dobijaju se tako što se na kraj jednog izraza doda znak ';'.

Primeri:

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

Naredbe za ulaz i izlaz

- U C-u ne postoje posebne naredbe za ulaz/izlaz podataka.
- Postoji skup bibliotečkih funkcija za učitavanje podataka u program i za upis rezultata na standardni izlaz ili u datoteku (scanf i printf).
- Deklaracije funkcija za upravljanje ulaznim/izlaznim resursima u C-u nalaze se u fajlu stdio.h
- Pre korišćenja funkcija za ulaz/izlaz podataka u fajl sa izvornim kodom treba uključiti fajl stdio.h

- Za uključivanje jednog fajla u drugi koristi se pretprocesorska direktiva include

Format include direktive:

#include <ime_fajla> ili #include "ime_fajla"

Konkretno, kad god se u programu koriste funkcije scanf ili printf, treba navesti:

#include <stdio.h>

Funkcija za učitavanje podataka sa standardnog ulaza scanf()

Poziva se na sledeći način:

scanf(<format>,<adr_ul_podatka1> [,<adr_ul_podatka2>]…)

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.

& - adresni operator

&a – adresa promenljive **a**

Format pojedinačne ulazne konverzije:

%[w][h|l|L]<tip_konverzije>

Tip konverzije može biti:

d – decimalna konverzija (rezultat je tipa int),
u – decimalna konverzija neoznačenih brojeva (rezultat je tipa unsigned int),
o – oktalna konverzija (rezultat je tipa int),
x – heksadekadna konverzija (rezultat je tipa int),
i – decimalna, 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 – konv. brojeva zapisanih u eksponencijalnom obliku u podatke tipa float,
g – konv. koja označava da se ulazni znakovni niz konvertuje u podatak tipa float, bez obzira da li je na ulazu broj zapisan u fiksnom zarezu ili u eksponencijalnom obliku,
c – znakovna konverzija (rezultat je tipa char),
s – konverzija u znakovni niz (niz znakova između dva bela znaka što znači da učitani niz ne sadrži bele znake. Iza pročitanih znakova dodaje se '\0')

Značenja prefiksa

w – ceo broj koji označava dozvoljeni broj znakova za podatak 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 broja 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**.

Primeri:

```
int a;
unsigned b;
float x, y;
long double z;
scanf("%d%u%f%e%Lf", &a, &b, &x, &y, &z);
```

Funkcija za prikaz rezultata na standardni izlaz printf()

Poziva se na sledeći način:

printf(<format>[,<izraz1>][,<izraz>]…)

gde je:

<format> - Znakovni niz koji predstavlja definiciju konverzija koje treba izvršiti pri prikazu rezultata.

- U ovom slučaju, vrši se konvertovanje memorijskog zapisa podataka u izlazne znakovne nizove.
- Osim pojedinačnih konverzija, format može sadržati i dodatne komentare koji će pojasniti izlazne rezultate ili uticati na preglednost prikazanih rezultata.

Format pojedinačne izlazne konverzije:

%[-][+|][#][w[.d]][h|l|L]<tip_konverzije>

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 *default*-u vrši se 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 brojeva, 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 ce 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 maksimalni broj karaktera koji će biti ispisani.

h, l, i L – imaju isto značenje kao i kod ulazne konverzije.

Primer:

```
int a;
unsigned b;
float x, y;
long double z;
        ...
printf("a=%-5d, b=%-3u\nx=%-8.4f, y=%-+15.7e\n z=%Lf", a, b, x, y, z);
```

Primer kompletnog C programa

```
#include <stdio.h>
main()
{
  int a,b,c;
  unsigned j,k;
  float f, g=5.0*f;
  double z;
  printf("unesite vrednosti promenljivih a,b,j,f,z\n");
  scanf("%d%o%u%f%lf", &a,&b,&j,&f,&z );
  c=a+b;
  k=2*j++;
  g*=f;
  z--;
  printf("c=%d, k=%u, j=%u, g=%f, z=%lf\n", c,k,j,g,z);
}
```

Kontrola toka programa

Kontrola toka programa – strukturnim naredbama

U teoriji programiranja definisane su tri osnovna tipa programskih struktura:

- **sekvenca**,
- **grananja**,
- **programske petlje**.

U programskom jeziku C:

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

Blok naredbi

- Skup programskih naredbi koje se izvršavaju onim redosledom kako su zapisane.
- **Blok:** počinje simbolom **{**

  završava se simbolom **}**

Tj. blok u C-u ima sledeći izgled:

```
{
        <naredba1>
        <naredba2>
        …
        <naredbaN>
}
```

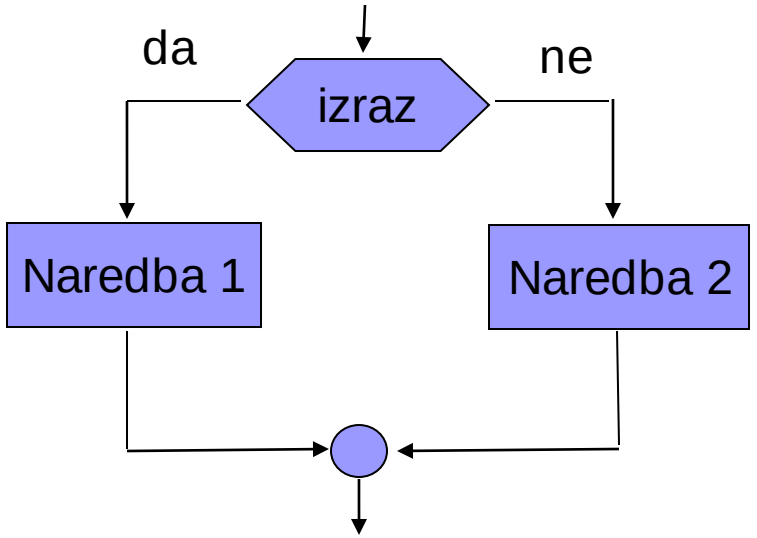
- Na početku bloka mogu biti navedeni **opisi promenljivih i konstanti** koje su **lokalne za blok**.

Naredbe uslovnog grananja

If - naredba

If-naredbom se implementira osnovi tip selekcije (grananja) kojom se vrši uslovno izvršenje jedne od dve moguće naredbe.

Standardni dijagram toka strukture grananja



Ekvivalentan kod u C-u je:

```
if (<izraz>)
   <naredba1>
else
   <naredba2>
```

Dejstvo: Kada je vrednost izraza:

- različita od nule (tj. **true**), izvršava se naredba <naredba1>,
- nula (tj **false**), izvršava se naredba <naredba2>.

Zadatak: **Izračunavanje vrednosti funkcije**

Napisati program na C-u za izračunavanje vrednosti funkcije **y = 1 / x**

Rešenje:

Funkcija **y = 1/x** nije definisana za **x=0**. Zbog toga se u rešenju koristi struktura grananja kojom se odlučuje da li se vrednost funkcije računa ili ne.

```
#include <stdio.h>
main()
{
      float x;

      printf("Unesite vrednost argumenta x \n");
      scanf("%f",&x);
      if ( x == 0 )
        printf("Za x=0 funkcija y=1/x nije definisana\n");
      else
        printf( "y=%f\n", 1/x );
}
```

Zadatak: Napisati program na C-u za izračunavanje vrednosti funkcije:

$$y = \begin{cases} x, & \text{za } x < 2 \\ 2, & \text{za } 2 \le x < 3 \\ x - 1 & \text{za } x \ge 3 \end{cases}$$

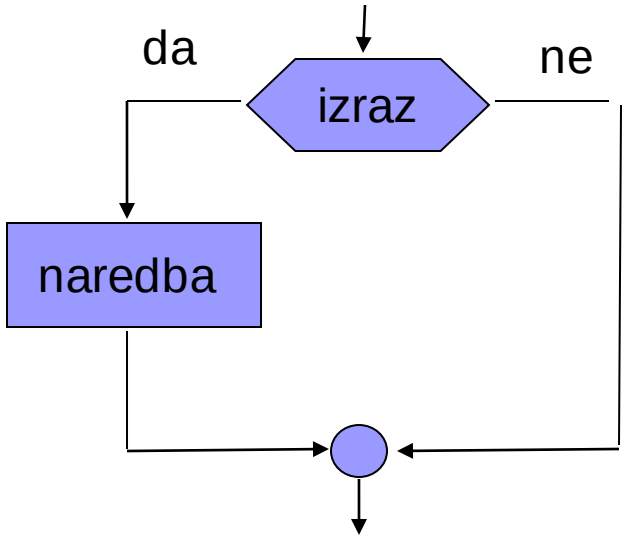
Rešenje:

```
#include <stdio.h>
main()
{
        float x,y;

        printf("Unesite vrednost argumenta x \n");
        scanf("%f",&x);
        if ( x < 2 )
                y = x;
        else if ( x < 3 )
                y = 2;
        else
                y = x – 1;
        printf( "y=%f\n", y );
}
```

Skraćeni oblik If-naredbe

Selekcijom se može implementirati i odluka da li će se jedna naredba u programu izvršiti ili ne.
Dijagram toka strukture skraćenog grananja



Ekvivalentan kod u C-u je:

```
if (<izraz>)
   <naredba>
```

Dejstvo:
Kada je vrednost izraza:

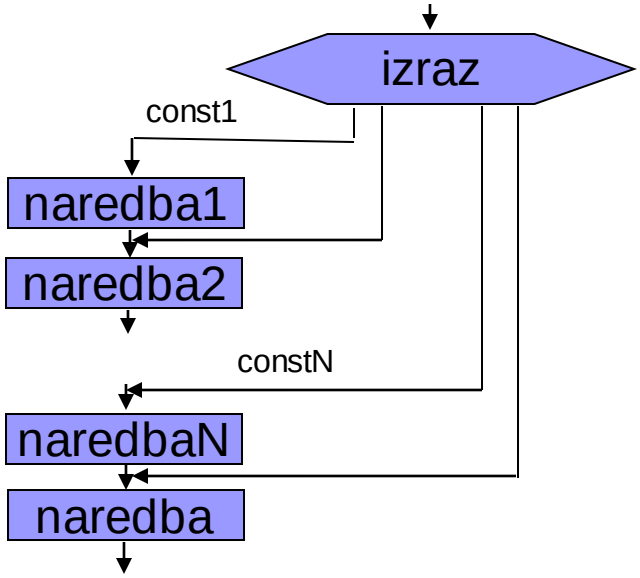
- različita od nule (tj. **true**), izvršava se navedena **<naredba>**
- jednaka nuli (tj. **false**), ne izvršava se ni jedna naredba.

switch naredba

• Jedan od načina za realizaciju višestrukog grananja u programu

Ekvivalentan kod u C-u je:

```
switch (<izraz>)
{
        case <const1>: <naredba1>
        case <const2>: <naredba2>
        ...
        case <constN>: <naredbaN>
        [default: <naredba>]
}
```



Dejstvo: Vrednost izraza upoređuje sa vrednostima navedenih konstanti.

Ukoliko se vrednost izraza poklopi sa nekom od konstanti izvršiće se niz naredbio iz *switch*-strukture od naredbe obeležene tom konstantom do kraja strukture.

Ukoliko se vrednost izraza ne poklapa ni sa jednom od ponuđenih vrednosti izvršiće se samo naredba navedena u *default*-delu (ukoliko *default*-deo postoji).

Izraz (kao i konstante const1,…,constN) mogu biti tipa int ili char.

Zadatak:

Napisati program na C-u za štampanje imena meseci u godini počev od tekućeg.

Rešenje:

```
#include <stdio.h>
main()
{
        int tekuci;
        printf("unesite redni br. tekuceg meseca\n");
        scanf("%d",&tekuci);
        switch( tekuci )
        {
                case 1: printf("januar\n");
                case 2: printf("februar\n");
                case 3: printf("mart\n");
                case 4: printf{"april\n");
                case 5: printf("maj\n");
                case 6: printf("jun\n");
                case 7: printf{"jul\n");
                case 8: printf("avgust\n");
                case 9: printf("septembar\n");
                case 10: printf("oktobar\n");
                case 11: printf("novembar\n");
                case 12: printf("decembar");
        }
```

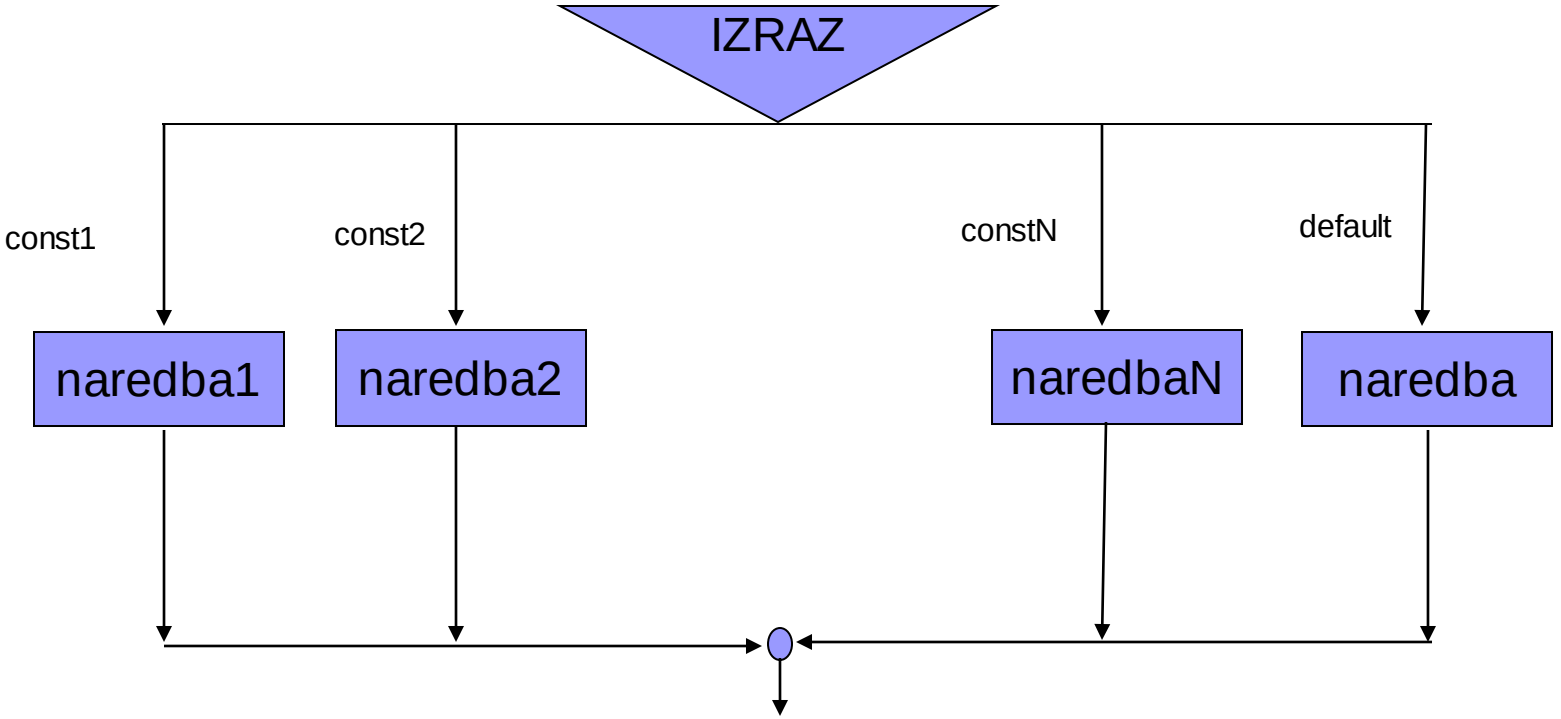
Naredbe bezuslovnog skoka

break naredba

- break naredbom se prekida izvršenje strukture u kojoj se naredba nalazi.
- Format naredbe je:

```
break;
```

- 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 realizuje kombinacijom switch-strukture i *break*-naredbe, poznata kao struktura **«češlja»**



Ekvivalentan kod u C-u:

```
switch (<izraz>)
{
        case <const1>: <naredba1>  break;
        case <const2>: <naredba2>  break;
        ...
        case <constN>: <naredbaN>  break;
        [default: <naredba>]
}
```

Zadatak:

Napisati program na C-u za štampanje imena tekućeg meseca.

Rešenje:

```
#include <stdio.h>
main()
{
        int tekuci;
        printf("unesite redni br. tekuceg meseca\n");
        scanf("%d",&tekuci);
        switch( tekuci )
        {
            case 1: printf("januar\n");  break;
            case 2: printf("februar\n");  break;
            case 3: printf("mart\n");  break;
            case 4: printf("april\n");  break;
            case 5: printf("maj\n");  break;
            case 6: printf("jun\n");  break;
            case 7: printf("jul\n");  break;
            case 8: printf("avgust\n");  break;
            case 9: printf("septembar\n");  break;
            case 10: printf("oktobar\n");  break;
            case 11: printf("novembar\n");  break;
            case 12: printf("decembar\n");
        }
}
```

Zadatak:

Napisati program na C-u za štampanje broja dana tekućeg meseca.

Rešenje:

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

continue naredba

continue naredba se može naći samo u telu neke programske petlje i njom se prekida izvršenje tekuće iteracije petlje.

Format naredbe:

```
continue;
```

goto naredba

Format naredbe:

```
goto <labela>
```

Dejstvo: Izvršenje programa se nastavlja od naredbe obeležene navedenom labelom.

Primer labele:

```
lab:
```

Programske petlje

Programske petlje omogućavaju višestruko ponavljanje određenog dela programa u toku njegovog izvršavanja.

Vrste petlji:

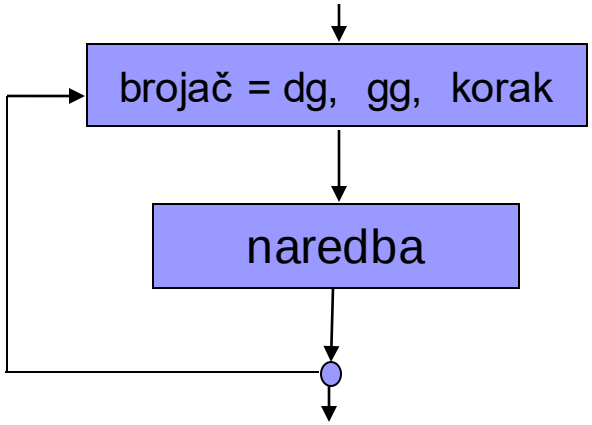
a) sa konstantnim brojem prolaza (brojačka petlja),
b) sa promenljivim brojem prolaza:
- while tip
- repeat until tip

U programskom jeziku C postoje tri vrste programskih petlji:

1) for - petlja
2) *while* - petlja,
3) *do-while* - petlja i

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.



<brojač> - ime promenljive koja predstavlja brojač petlje,
<dg> - početna vrednost brojača petlje,
<gg> - konačna vrednost brojača petlje,
<korak> - vrednost koja se dodaje brojaču petlje na kraju svake iteracije.

For petlja u programskom jeziku C ima širi smisao od navedene definicije brojačkih petlji.

Format **for**-naredbe u C-u je:

for (<izraz1>; <izraz2>; <izraz3>)
<naredba>

gde:

<izraz1> - vrši inicijalizaciju promenljivih koje se koriste u petlji (što može da bude postavljanje početne vrednosti brojača petlje),

<izraz2> - 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 dok brojač petlje ne postigne svoju gornju granicu

<izraz3> – definiše promenu vrednosti promenljivih koje se koriste u petlji. Navedena promena se 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 se znak ';' mora pisati.

Zamenjena while naredbom, for petlja ima oblik:

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

Ako je <izraz2> **izostavljen**, smatra se da je uslov for petlje logički tačan, pa se petlja ponaša kao beskonačna petlja.

Zadatak: Sumiranje N brojeva čije se vrednosti unose sa tastature.

Rešenje:

Ovaj primer pokazuje način realizacije klasične brojačke petje for-petljom u C-u. Brojačkom petljom u ovom slučaju definiše se deo koda koji će se izvršiti tačno N puta.

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

Zadatak: Napisati program na C-u za izračunavanje i štampanje vrednosti funkcije ***ch x*** korišćenjem *for*-petlje.

Rešenje:

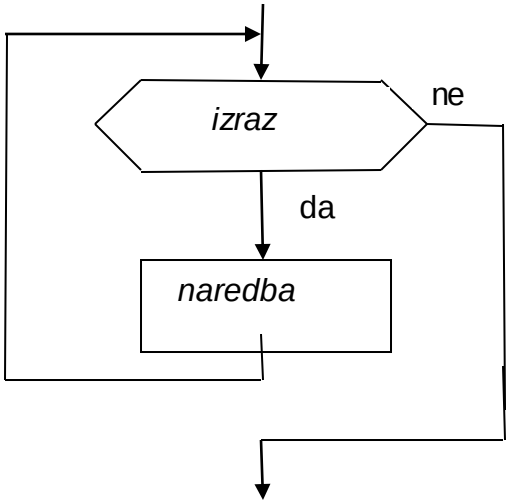
U ovom slučaju for petlju koristimo za realizaciju petlji koje se ponavljaju dok je odredjeni uslov zadovoljen.

- Praktično, ***for***-petlja sada ima ulogu ***while***-petlje.

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

WHILE petlja

While petlja omogućava ponavljanje naredbe programa dok je definisan uslov zadovoljen (tj. dok je vrednost izraza različita od nule).



Ekvivalentan kod u C-u je:

```
while (<izraz>)
        <naredba>
```

While-naredba služi za realizaciju iterativnih procesa (sumiranje beskonačnih redova i sl.

Zadatak:
Napisati program na C-u za izračunavanje i štampanje vrednosti funkcije ***ch x*** primenom sledećeg razvoja u red:
Izračunavanje prekinuti kada relativna vrednost priraštaja sume postane manja od zadate vrednosti ε.

Napomena: Ovaj red važi za $|x|<4$.

Rešenje:

- **Vrednost beskonačne sume *S* se zamenjuje konačnom sumom**
- **Koliko je članova dovoljno sumirati?**
- **Kriterijum za prekid sumiranja:**
  - **apsolutna vrednost priraštaja sume manja od neke zadate tačnosti**
  - **relativna vrednost priraštaja sume manja od neke zadate tačnosti.**

$$\left|S_{i+1}-S_i\right|=\left|a_{i+1}\right| \qquad \left|\left(S_{i+1}-S_i\right)/S\right|\approx\left|a_{i+1}/S_{i+1}\right|$$

- **Rekurentnu formulu za izračunavanje vrednosti proizvoljnog člana sume.**

$$a_k=\frac{x^{2k}}{(2k)!} \qquad a_{k+1}=\frac{x^{2(k+1)}}{(2(k+1))!}=\frac{x^{2k}x^2}{(2k+2)!}=\frac{x^{2k}x^2}{(2k+2)(2k+1)2k!}=\frac{x^2}{(2k+2)(2k+1)}a_k$$

- **Potrebno je odrediti vrednost prvog člana.**

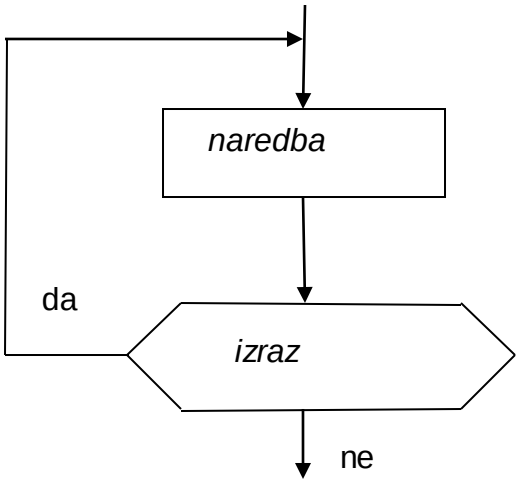
Prvi član sume je:

$$a_0=1$$

```
#include <stdio.h>
#include <stdlib.h>
main()
{
        int k;
        float x,a,S,c,eps;
        printf("unesite argument x i tacnost eps\n");
        scanf("%f,%e",&x,&eps);
        a=1;
        S=1;
        k=0;
        while ( abs(a/S)>eps )
        {
                a*=x*x/((2*k+2)*(2*k+1));
                S+=a;
                k++;
        }
        printf("ch(%f)=%e\n",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 tako da se naredba, koja predstavlja telo petlje, mora izvršiti bar jednom.



Format odgovarajuće naredbe u programskom jeziku C je:

```
do
    <naredba>
while (<izraz>);
```

Zadatak:

Napisati program koji izračunava i štampa vrednost *n*-tog korena iz *a* (pri čemu je a>0), primenom sledećeg iterativnog postupka:

$$x_0 = (a + n - 1)/n \qquad x_{i+1} = ((n\text{-}1)x_i + a/x_i^{\,n-1})/n$$

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

Rešenje:

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

POLJA

Polja predstavljaju složenu strukturu podataka, sa elementima istog tipa

Jednodimenziona polja

Definicija jednodimenzionalnog polja

<tip> <ime> [<veličina>];

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:

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

- 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:

```
<ime_polja>[<izraz>]
```

Primer

```
… a[5] …
… b[i] …
… x[5*j+7]
```

Zadatak: Nalaženje maksimalnog elementa u nizu

Napisati program na C-u kojim se određuje maksimalni element u celobrojnom vektoru **A** od **n** komponenata.

Rešenje:

```
#include <stdio.h>
main()
{
        int a[100], n, max, i;
        printf("unesite broj elemenata vektora x\n");
        scanf("%d",&n);
        printf("unesite elemente vektora\n");
        for ( i=0; i<n; i++ )
                scanf( "%d", &a[i] );
        max=a[0];
        for ( i=1, i<n; i++ )
                if ( max < a[i] )
                        max = a[i];
        printf("maksimalni element vektora a je: %d\n",
    max);
}
```

Zadatak: Uredjivanje niza A u neopadajući redosled

```
#include <stdio.h>
main()
{       int n, imin, i, j;
        float a[100], pom;
        printf("unesite broj elemenata vektora x\n");
        scanf("%d",&n);
        for ( i=0; i<n-1; i++ )
        {       imin=i;
                for ( j=i+1, j<n; j++ )
                        if ( a[imin] > a[j] )
                                imin = j;
                if ( imin != i )
                {       pom = a[imin];
                        a[imin] = a[i];
                        a[i] = pom;
                }
        }
        for ( i=0; i<n; i++ )
                printf("%f, ", a[i] );
        printf("\n");
}
```

Višedimenzionalna polja

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

<tip> <ime>[<vel1>][<vel2>]…[<velN>];

Primer:

char sahtabla[8][8];

- U slučaju dvodimenzionalnog polja, prva dimenzija predstavlja **vrste**, a druga **kolone** dvodimenzionog polja (tj. matrice).
- Elementi višedimenzionalnih polja se u memoriji smeštaju kao jednodimenzionalna polja tako da se poslednji indeks najbrže menja, zatim pretposlednji, … (“po vrstama”)

- Pri definisanju jednodimenzionalnog polja, veličina polja može biti izostavljena u 3 slučaja:
  1. ako se u definiciji vrši i inicijalizacija polja;
  2. ako je polje fiktivni argument funkcije;
  3. ako polje ima klasu memorije extern.
- Kod definisanja višedimenzionalnih polja moguće je izostaviti samo veličinu prve dimenzije.

Zadatak: Množenje matrica

Napisati program na C-u za množenje matrica A i B reda *mxn* i *nxk*, respektivno.

```
#include <stdio.h>
main()
{       int A[100][100],B[100][100],C[200][200], m,n,k,i,j,l;
        printf("unesite m, n i k");
        scanf("%d%d%d",&m, &n, &k);
        printf("unesite elemente matrice A\n");
        for ( i=0; i<m; i++ )
                for ( j=0; j<n; j++ )
                        scanf( "%d", &A[i][j] )
        printf("unesite elemente matrice B\n");
        for ( i=0; i<n; i++ )
                for ( j=0; j<k; j++ )
                        scanf( "%d", &B[i][j] )
        for ( i=0; i<m; i++ )
                for ( j=0; j<n; j++ )
                {
                        C[i][j]=0;
                        for (l=0; l<k; l++ )
                                C[i][j] += A[i][l]*B[l][j];
                }
```

```
        printf( “proizvod matrica A i B je:\n” );
        for ( i=0; i<m; i++ )
        {
                for ( j=0; j<k; j++ )
                        printf( “%d, “, C[i][j] );
                printf(“\n”);
        }
}
```

Inicijalizacija polja

Lista početnih vrednosti elemenata polja se piše unutar para vitičastih zagrada.

Inicijalizacija višedimenzionalnog polja se može vršiti na dva načina,

Primeri:

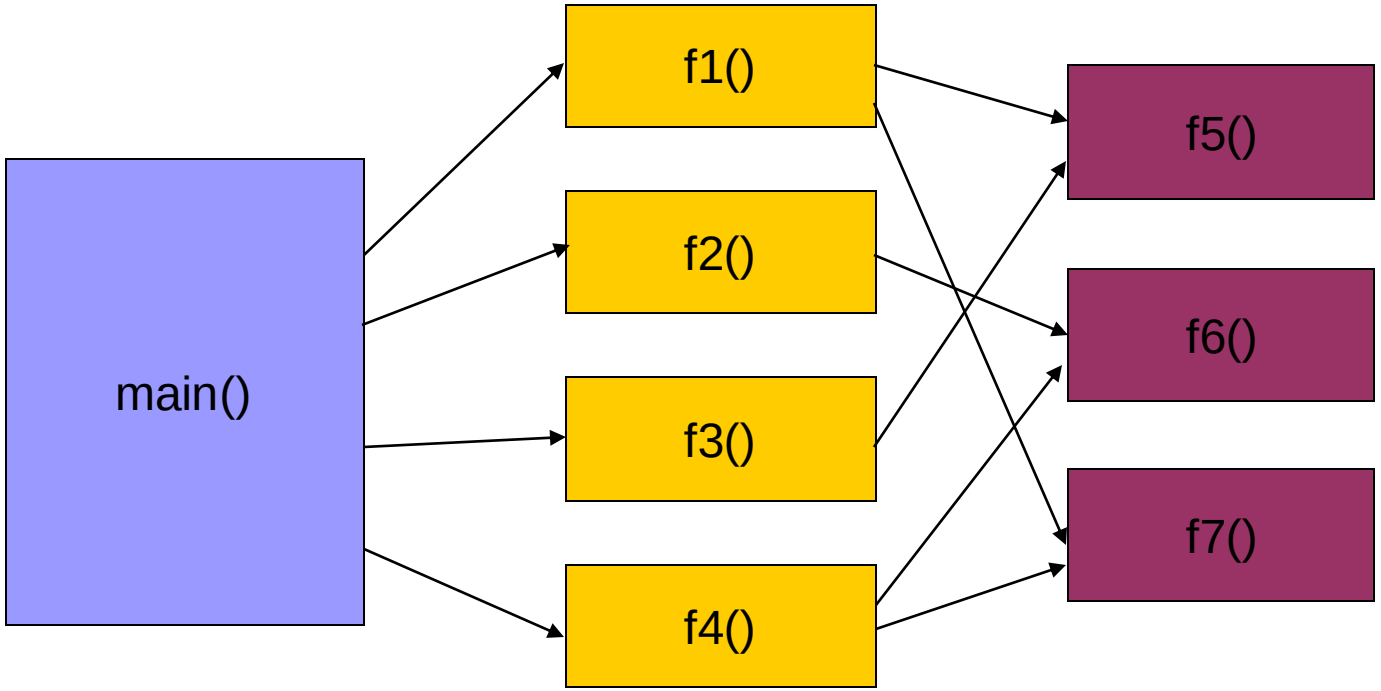
```
char samoglasnik[5]={ 'A', 'E', 'I', 'O', 'U' };
char m_samoglasnik[ ]={ 'a', 'e', 'i', 'o', 'u' };
/*prevodilac sam izračunava broj elemenata */

int matrica1[3][4]={25,26,27,28,29,24,23,21,20,10, 11, 12};
int matrica2[3][4] = {
                                    {25, 26, 27, 28}
                                    {29, 24, 23, 21}
                                    {20, 10, 11, 12}
                    };
int matrica3[ ][4] = {25,26,27,28,29,24,23,21,20,10, 11, 12};
/*posto je broj kolona poznat prevodilac može da izračuna broj vrsta na
osnovu broja elemenata u inicijalizaciji*/
```

Dekompozicija

- Uobičajeno 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.

- Glavni program u C-u definisan kao funkcija (main).



Definisanje funkcije

Definicija funkcije u programskom jeziku C ima sledeći format:

```
[<tip_rezultata>] <ime_funkcije>([<lista_parametara>])
[<definicije_parametara>]
{
        <telo_funkcije>
}
```

```
int fun1(float a, int b)
{
   . . .
}
```

```
int fun1(a,b)
float a;
int b
{
   . . .
}
```

gde je:

<ime_funkcije> - simboličko ime koje u isto vreme predstavlja i adresu funkcije

<telo_funkcije> - blok naredbi (može biti i prazno)

<lista_parametara> - formalni parametri, 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.

Naredba RETURN

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

Postoje dva oblika ove naredbe:

```
return (<pov_vred>);  // pov_vred je rezultat funkcije
return;     //ne vraca se nista, za tip funkcije void
```

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

```
<promenljiva>=<ime_funkcije>(<stvarni_parametri>);
```

Deklaracija funkcije

Svaka funkcija programskog jezika C treba da bude poznata kompilatoru 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 naziva se - prototip funkcije*

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

- Moguće su:
  - eksplicitne (ako ih navodi programer) i
  - implicitne (ako ih uvodi C prevodilac) deklaracije funkcija.

- Ukoliko programer nije definisao, niti deklarisao funkciju pre njenog poziva, C prevodilac je implicitno deklariše i dodeljuje joj tip int.

Eksplicitna deklaracija funkcije

[<tip_rezultata>] <ime_funkcije>();

- 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 u pozivu), a mogu se navesti i sama imena argumenata. To samo pomaže čitaocu koda da lakše razume kod, dok prevodilac jednostavno taj deo koda ignoriše.
- Funkcija može biti deklarisana izvan tela drugih funkcija (globalni nivo), ili unutar tela neke druge funkcije i tada se kaže da je **deklaracija globalna** ili **lokalna** za tu funkciju.
- Ne dozvoljava se definisanje jedne funkcije unutar neke druge, ali je deklaracija dozvoljena.

Primer:

```
double kvadrat();  // ovo je deklaracija na globalnom nivou   ili   double kvadrat(double)
main()                                                              ili   double kvadrat(double x)
{
  double a,b,x,y,koren();  //deklaracija unutar funkcije
  x=12.3;
  y=kvadrat(x);
  a=156.98;
  b= koren(a);
}
void funk1()
{
/* ovde se moze pozvati funkcija kvadrat, ali ne i koren koja je deklarisana u main */
…
}
double kvadrat(brojA)
.....
double koren(brojB)
.....
```

Zadatak: Izračunavanje celobrojnog stepena zadate osnove

Napisati funkciju za izračunavanje pozitivnog celobrojnog stepena realne osnove.

U funkciji main(), korišćenjem kreirane funkcije izračunati celobrojni stepen realne osnove pri čemu se i osnova i stepen unose sa tastature.

Rešenje:

Najpre će biti definisana funkcija main. Zbog toga će u njenoj definiciji biti navedena deklaracija funkcije za izračunavanje stepena,

a definicija ove funkcije biće navedena na kraju.

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

```
float stepen(float a, int n)
{
    int i;
    float rez;
    for (i=1, rez=1; i<=n; i++, rez*=a);
        return (rez);
}
```

Prenos parametara

- Vrste prenosa:
  - po vrednosti (call by value)
  - 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 funkcije bude promenjen u funkciji, a da to bude vidljivo u pozivajućem modulu.

Primer:

```
main()
{
        float a, fun();
        a=23.123;
        printf("\n Vrednost a pre poziva %f",a);
        fun(a);
        printf("\n Vrednost a posle poziva %f",a);
        /* Odstampana je ista vrednost*/
}

float fun(float a)
{
        a= a*a + 234.12;
        return (a);
}
```

Da bi se dobio tačan rezultat može se izvršiti sledeća modifikacija u funkciji main:

```
a=fun(a); //a menja vrednost na vracenu iz fun
```

Ukoliko 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 kreiraju se **kopije za pokazivače**, u funkciji će se menjati sadržaji lokacija na koje ti pokazivači ukazuju, a sami pokazivači se brišu nakon završetka rada funkcije.

Primer

Realizovati funkciju za zamenu vrednosti dveju promenljivih.

```
izmena(int *x, int *y)
{
        int pomocna;
        pomocna=*x;
        *x=*y;
        *y=pomocna;
}
main ()
{
          int a=8,b=3;
          izmena(&a,&b);
          printf("Brojevi posle poziva funkcije %d, %d", a,b);
}
```