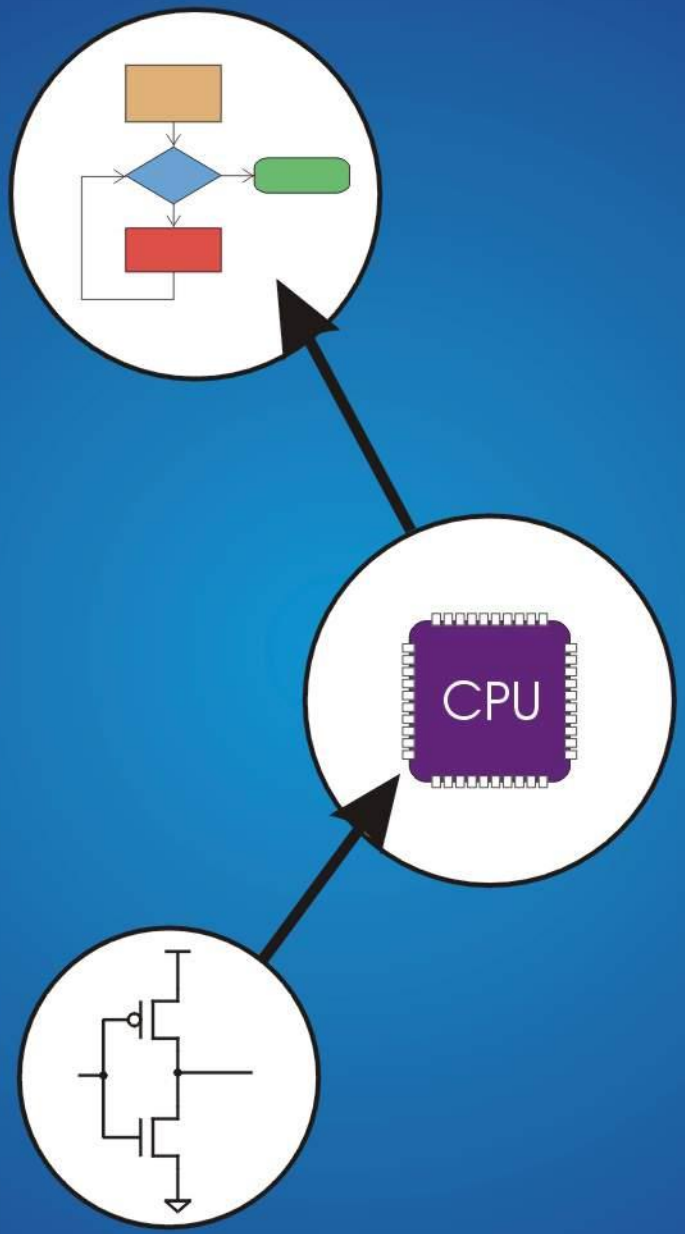


UNIVERZITET UNION
Nikola Tesla
Poslovni i pravni fakultet

UVOD U Programski jezik C

Docent prof. dr Borivoje Milošević

Uvod

Programski jezik C dizajnirao je Dennis Ritchie iz Bell Laboratories ranih 1970.

Oslonjen je na

- ALGOL 60 (1960),
- CPL (Cambridge, 1963),
- BCPL (Martin Richard, 1967),
- B (Ken Thompson, 1970)

Tradicionalno se koristi za sistemsko i aplikativno programiranje, i predstavlja podskup C++ jezika

Tradicionalni C je opisan u knjizi:

- *The C Programming Language*, by Brian Kernighan and Dennis Ritchie, 2nd Edition, Prentice Hall

Dennis Ritchie

Born	September 9, 1941 Bronxville, New York, U.S.
Died	c. October 12, 2011 (aged 70) Berkeley Heights, New Jersey, U.S.
Nationality	American
Alma mater	Harvard University (Ph.D., 1968)
Known for	ALTRAN B BCPL C Multics Unix
Awards	Turing Award (1983) National Medal of Technology (1998) IEEE Richard W. Hamming Medal (1990) Computer Pioneer Award (1994) Computer History Museum Fellow (1997)[1] Harold Pender Award (2003) Japan Prize (2011)
	Scientific career
Fields	Computer science
Institutions	Lucent Technologies Bell Labs

Standard C

Standardizovan je 1989 od ANSI (American National Standards Institute) poznat kao ANSI C

Internacionalni standard (ISO) 1990. godine usvojen je od strane ANSI organizacije i poznat je kao *C89*

Kao deo normalnog procesa evaluacije standard je apdejtovan godije 1995. (*C95*) i 1999 (*C99*) C++ i C

- C++ proširenje C sada uključuje podršku za Object Oriented Programming i mnoge druge funkcije koje omogućuju sučeljavanje sa velikim razvojnim projektima
- C nije striktno podskup jezika C++, ali moguće je pisati takozvani "*Clean C*" koji uključuje oba standarda i za C++ i za C.

Za i protiv?

Prilikom standardizovanja C jezika programeri koji su radili na mašinski zavisnim jezicima kritikovali su ovaj jezik visokog nivoa rečima, da u njegovoj formi postoji "crna kutija" koja ne dozvoljava korisnicima da sagledaju sve detalje izvršavanja jezika.

C jezk je dizajniran tako da omogući pristup svim resursima mašine, kako bi i neiskusnim programerima dao prostor za pisanje programa:

- bez ulaženja u detalje kako će se on izvršavati na računaru
- bez nužnog poznavanja rada procesora
- bez nužnog poznavanja operativne memorije i adresa
- bez nužnog poznavanja rada ulazno-izlaznih jedinica

Sve te neophodne operacije za izvršavanje programa sadržane su sada u u kompajleru i linkeru programskog jezika C

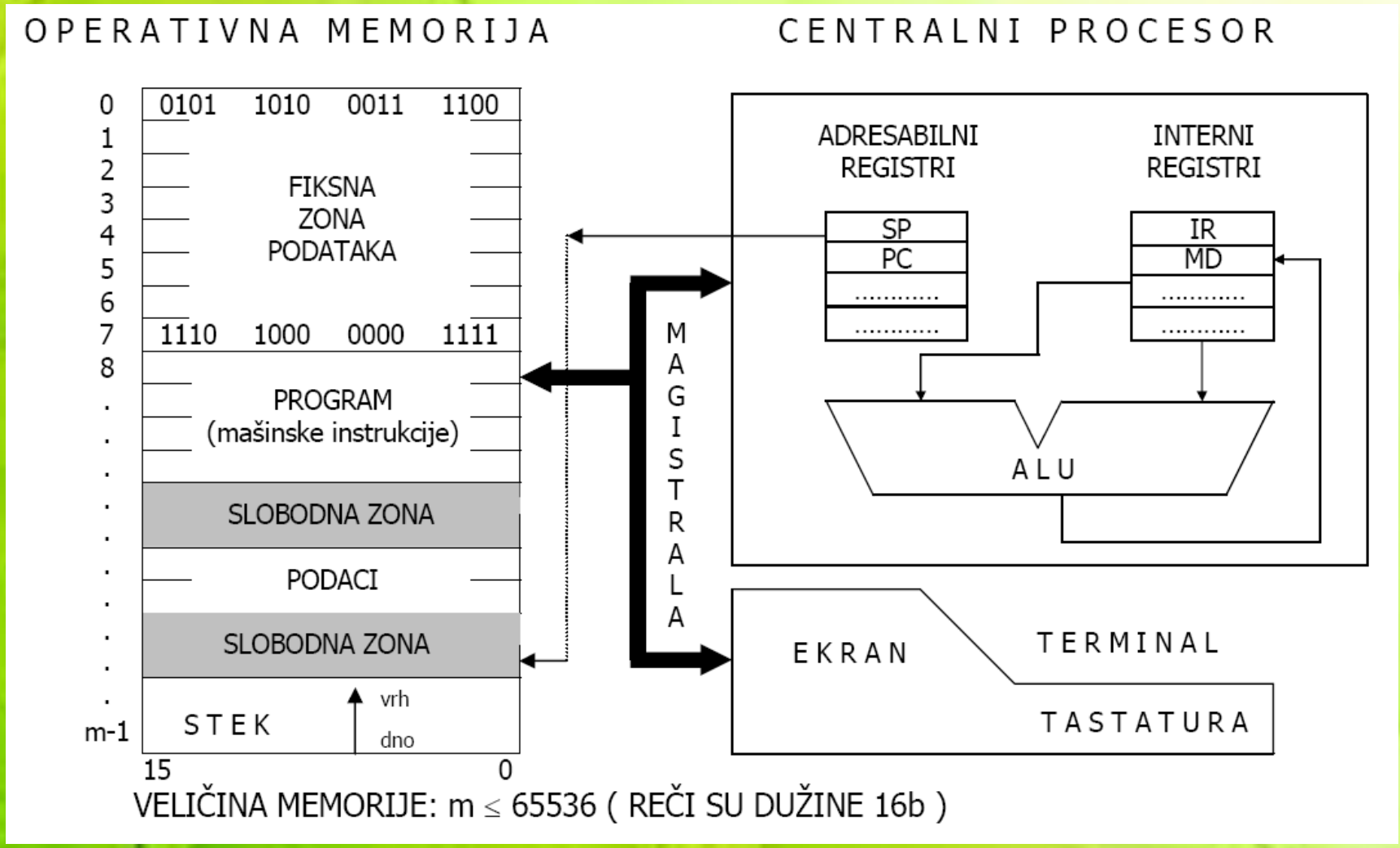
Za i protiv?

Zbog toga je programiranje na jeziku C kompajlerske prirode, pa se C program izvršava na sledeći način:

- Prvo, program se piše u okviru editora, koji je sastavni deo paketa Visual C++. Ova forma programa se naziva "izvorni kod programa" ili "source code". Kompjuter ne može direktno izvršiti izvorni kod programa.

- Drugo, kompletni izvorni kod se procesira kompajlerom, koji tada generiše novi kod programa u obliku mašinskog jezika i koji se zove izvršni fajl "executable file" (ili datoteka) sa ekstenzijom *.exe.
- Kažemo da je izvršni kod nastao kompajliranjem izvornog koda.
- Pokretanje izvršnog programa vrši se kucanjem njegovog imena ili direktno u prozoru opcijom RUN.

SASTAVNI DELOVI I ORGANIZACIJA pC-a



Kompilacija vs. Interpretacija

Različiti načini za prevođenje high-level jezika

Interpretacija

- interpreter = program koji izvršava programske naredbe
- generalno jedna linija/komanda u vremenu
- limitirano procesiranje
- lako vršenje debug opcija, promena, sagledavanja međurezultata
- jezici: BASIC, LISP, Perl, Java, Matlab, C-shell

Kompilacija

- prevodi naredbe u mašinski jezik
  - ne izvršava, već kreira izvršni program
- obezbeđuje optimizaciju preko velikog broja naredbi
- menja neophodne uslove za rekompilaciju
  - teže se sprovodi debug
  - jezici: C, C++, Fortran, Pascal

Kompilacija C Programa

Kada se program kompajlira u mašini, kompajler izvršava operacije po uredjenom redosledu koje se nazivaju "passes". Ove sekvence operacija se mogu opisati kao:

- Prvo kompajler čita izvorni kod, u nekim slučajevima generišući pseudo (medju) kod ili "pseudo code" koji pojednostavljuje izvršavanje izvornog koda po sekvencama.

- Drugo, kompajler konvertuje izvorni ili pseudo kod u objektni kod (formirajući tada datoteku sa ekstenzijom *.obj) koja je u obliku mašinskog jezika, ali koja još nije u izvršnoj formi.

- Na kraju, kompajler pokreće linker. Linker povezuje formirani objektni fajl sa svim bibliotekama programa potrebnim za samo izvršavanje programa, ubacuje ih u objektni kod generišući sada izvršnu verziju programa spremnu za pokretanje.

Kompilacija C Programa

Čitav mehanizam se naziva "compiler"

Predprocesor

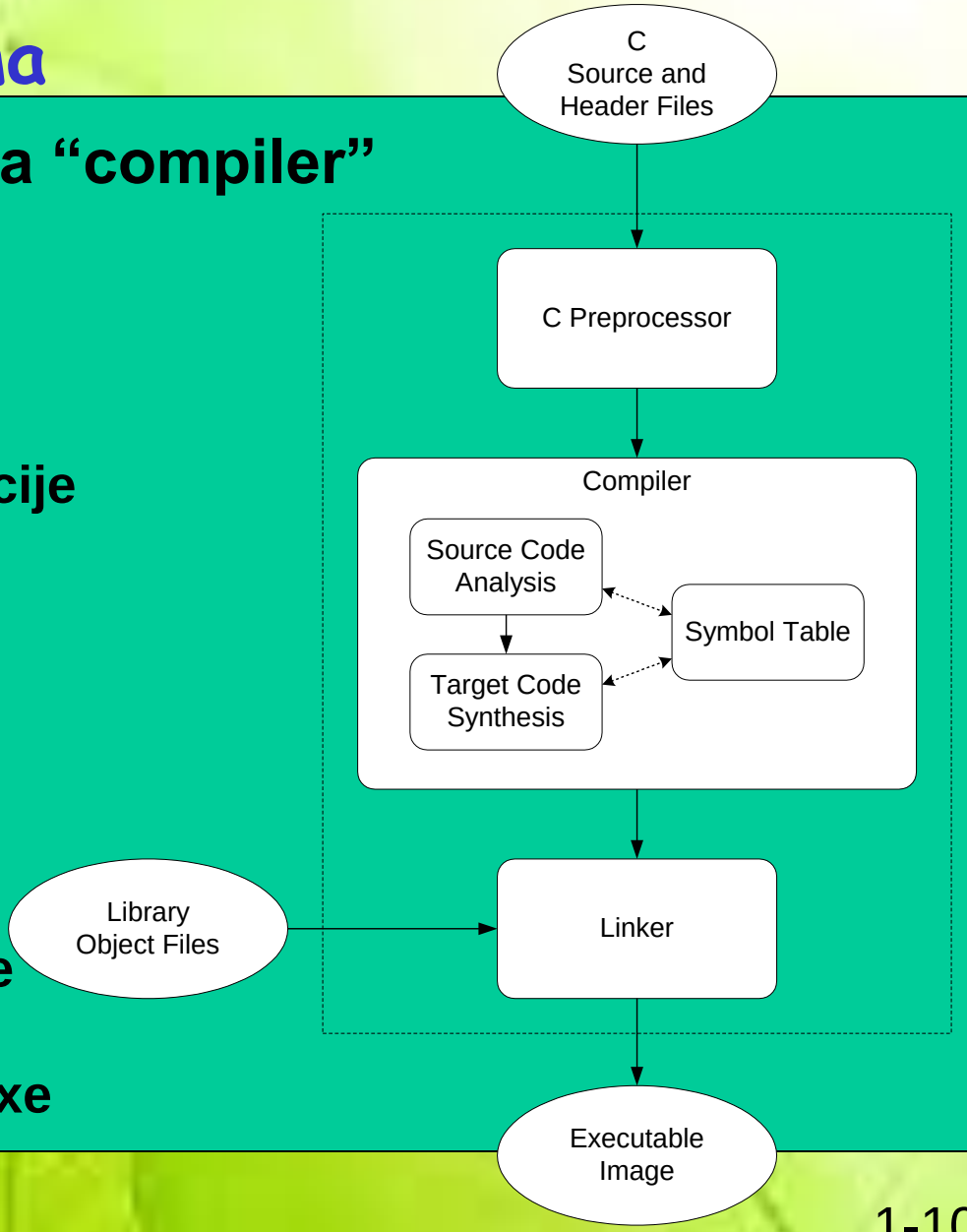
- **uključenje makroa**
- **uslovna kompilacija**
- **"source-level" transformacije**
  - **izlaz je još uvek C**

Kompajler

- **generiše objektni file**
  - **mašinske instrukcije**

Linker

- **kombinuje objektne fajlove (uključujući biblioteke) u izvršnu sliku – image *.exe**



Kompajler

Analiza Source Coda

- “front end”
- parsovanje programa za identifikaciju njegovih delova
  - promenljive, izrazi, naredbe, funkcije, itd.
- u zavisnosti od jezika (ne od mašine)

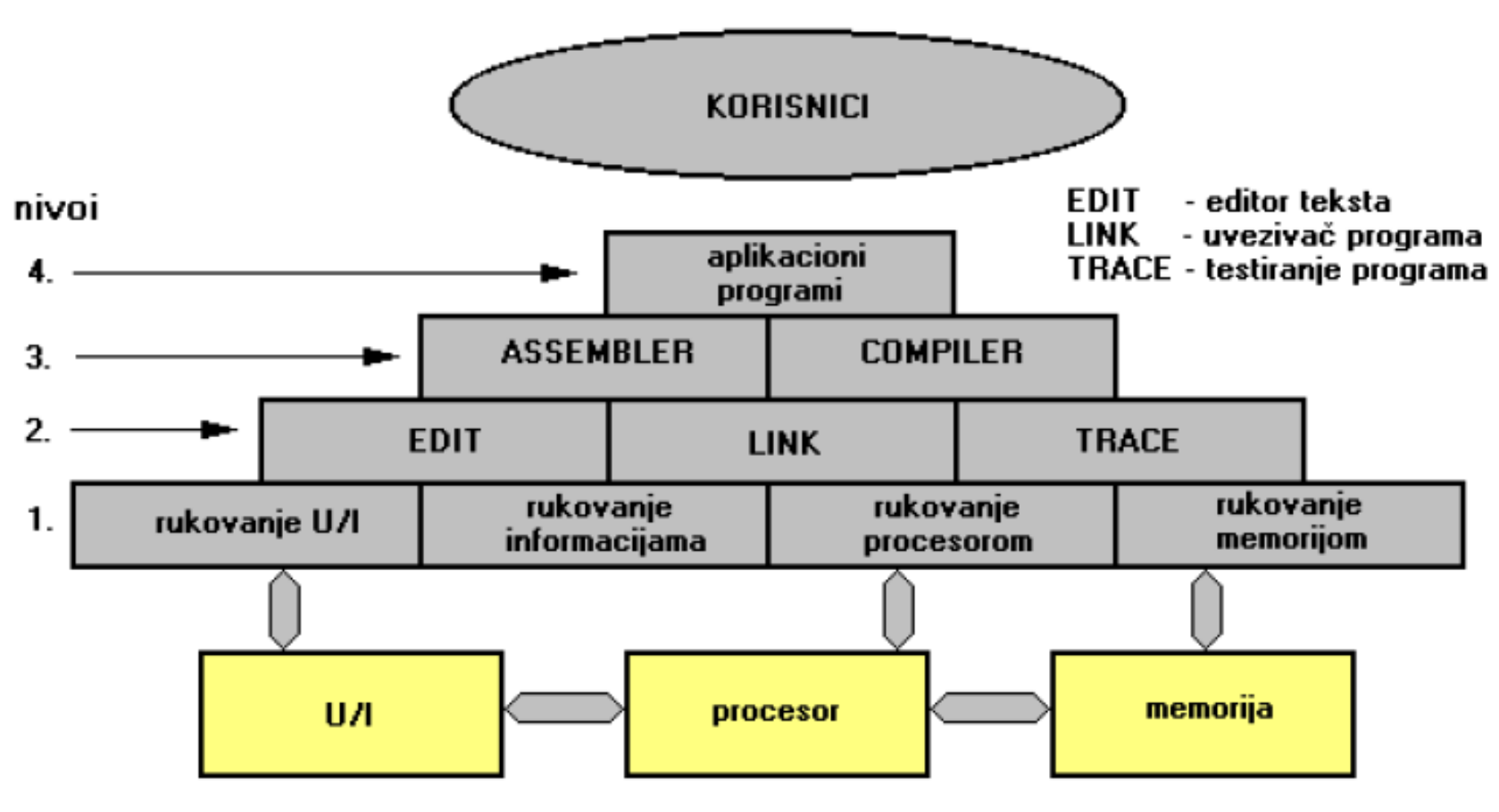
Generisanje koda

- “back end”
- generisanje mašinskog koda iz analiziranog izvornog programa
- optimizacija mašinskog koda
- zavisno od određene mašin

Tabela simbola

- mapiranje između simboličkih imena i stavki
- kao asembler, ali sa mnogo više informacija

Hijerarhijski nivoi programske podrške računarskog sistema



Nivoi su:

1. najniži nivo, tu su programi koji upravljaju resursima računarskog sistema, kod PC računara ovaj nivo je BIOS (Basic Input Output System),

2. na ovom nivou su programi za punjenje programa sa periferskih jedinica u operativnu memoriju, tekst editori i trace ili debugger programi za otklanjanje grašaka,

3. na ovom nivou su programi za prevođenje izvornog koda (programa) u mašinski kod, asembler i kompajleri viših programskih jezika (C, Pascal,itd.),

4. na ovom nivou je skup aplikacionih programa, preko kojih korisnik komunicira sa računarom.

Nivoi su:

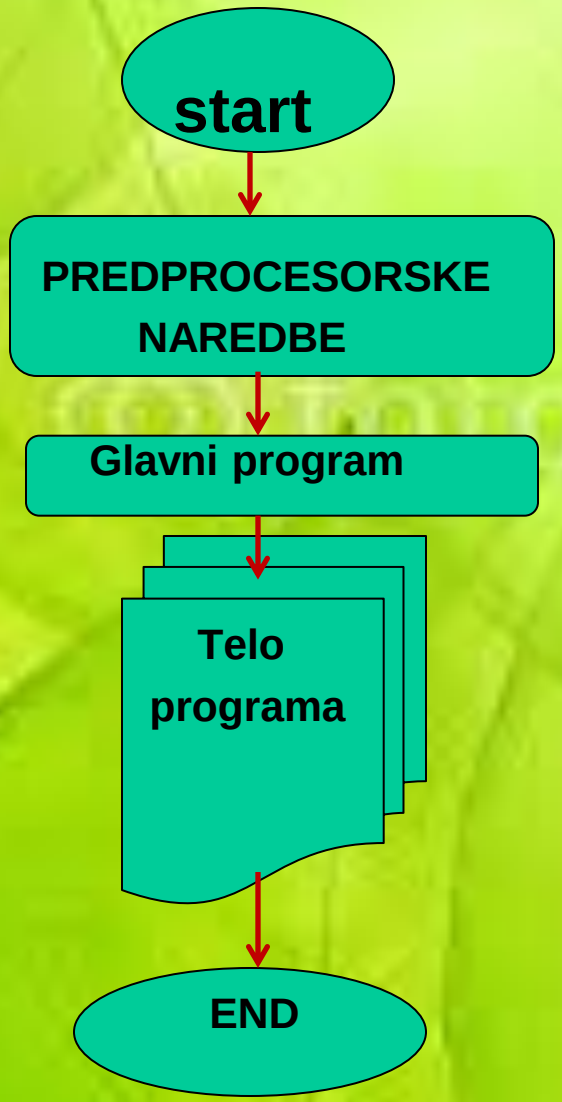
Na trećem nivou razlikujemo:

- **prevođenje izvornog koda u mašinski jezik pomoću prevodilaca, gde su faze prevođenja i izvršenja vremenski nezavisne**
- **i interpretaciju izvornog koda, gde su faze prevođenja i izvršenja vremenski zavisne.**

Prevodilac (compiler, translator): Svaki program koji prevodi neki drugi program iz jednog oblika u drugi. Prevodilac je program koji prevodi izvorni program, napisan na nekom višem programskom jeziku, u mašinski program.

Interpretator: Specijalna vrsta prevodioca programskih jezika, koji svaku naredbu jezika zasebno prevodi i odmah izvršava.

Pisanje programa



PREDPROCESORSE NAREDBE
Source i Header fajlovi

Header files (*.h) uvode zajedničke definicije za interfejse:

- function prototypes, data types, macros, inline functions i druge common deklaracije

Ne treba stavljati source kod (npr. definitions) u regione header file, sem kod nekoliko izuzetaka:

- inline'd code
- class definitions
- const definitions

C predrocesor (cpp) se koristi da insertuje zajedničke definicije u source files

Ali postoje i drugi razlozi zbog kojih pišemo naredbe u regionu predprocesora

C Standardni Header Files koje možemo koristiti

- stdio.h – file i console (takođe i file) IO: *perror, printf, open, close, read, write, scanf,* itd.
- stdlib.h - common utility functions: *malloc, calloc, strtol, atoi,* etc
- string.h - string i byte manipulatcija: *strlen, strcpy, strcat, memcpy, memset,* etc.
- ctype.h – tipovi karaktera: *isalnum, isprint, isupport, tolower,* itd.
- errno.h – definisanje *errno* korišćenjem alata reporting system errors
- math.h – matematičke funkcije: *ceil, exp, floor, sqrt,* itd.
- signal.h – signalne funkcije: *raise, signal,* itd
- stdint.h – standardni integer tip: *intN_t, uintN_t,* itd
- time.h – funkcije vremena: *asctime, clock, time_t,* itd.

Predprocesor

C predprocesor nam dozvoljava da definišemo jednostavne makroe koji se uvode i evalorizuju pre kompilacije.

Komande počinju sa '#'. Skraćena lista može biti:

- #define : definicija makroa
- #undef : uklanjanje definicije makroa
- #include : insertovanje teksta iz datoteke
- #if : uslov zasnovan na vrednosti izraza
- #ifdef : uslov zasnovan makro definiciji
- #ifndef : uslov zasnovan za nedefinisane makroe
- #else : alternativa
- #elif : uslovna alternativa
- defined() : predrocesorska funkcija: 1 ako je definisano ime, u suprotnom 0

Predrocesor: Makroi

Korišćenje makro funkcija:

- primer: `#define mymult(a,b) a*b`
  - Izvorno: `k = mymult(i-1, j+5);`
  - Post predrocesirano: `k = i - 1 * j + 5;`
- bolje: `#define mymult(a,b) (a)*(b)`
  - Izvorno : `k = mymult(i-1, j+5);`
  - Post predrocesirano : `k = (i - 1)*(j + 5);`

Biti obazriv na *efekat strane*, naprimer

- Macro: `#define mysq(a) (a)*(a)`
- flawed usage:
  - Izvorno : `k = mysq(i++)`
  - Post predrocesirano : `k = (i++)*(i++)`

Alternativno je korišćenje inline funkcija

- `inline int mysq(int a) {return a*a};`
- `mysq(i++)` radi kao što je opisano.

C: High-Level Language

Dopušta simbolička imena i vrednosti

- ne mora se poznavati koji se registar ili memorija koristi za smeštanje podataka

Obezbeđuje apstrakciju prema hardveru

- operacije ne zavise od seta instrukcija

Obezbeđuje široku izražajnost jezika

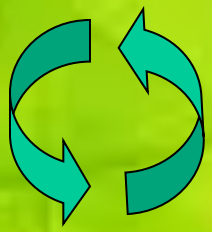
- korišćenje sadržajnih simbola sa prenosnim značenjem
- jednostavni izrazi za blokove zajedničke kontrole (if-then-else)

Sopstveni resursi za borbu protiv grešaka

Pisanje i pokretanje programa

```
#include <stdio.h>
/* The simplest C Program */
int main( )
{
  printf("Hello World\n");
  return 0;
}
```

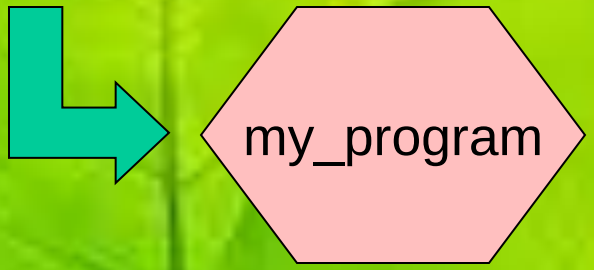
1. Napisati teks programa (source code) korišćenjem editora kao što je Visual C++, i zapamtiti fajl kao npr.. my_program.c



2.Pokrenuti kompajler da bi konvertovali program u jednu "executable" ili "binary" verziju:

```
$ gcc -Wall -g my_program.c -o my_program
tt.c: In function `main':
tt.c:6: parse error before `x'
tt.c:5: parm types given both in parmlist and separately
tt.c:8: `x' undeclared (first use in this function)
tt.c:8: (Each undeclared identifier is reported only once
tt.c:8: for each function it appears in.)
tt.c:10: warning: control reaches end of non-void function
tt.c: At top level:
tt.c:11: parse error before `return'
```

3-N. Kompajler će nam ukazati na greške - errors ili upozoranja - warning; editovati source file, ispraviti graške i izvršiti re-kompilaciju



N. Pokrenuti programi videti kako radi. ☺

. / ?

Šta ako ne radi?

C Sintaksa i pozdrav Hello World

#include insertuje druge fajlove. ".h" fajlove nazivamo "header" files. Oni sadrže stvari potrebne da bi se povezale biblioteke i kod programa.

Da li program može imati više .c fajlova?

Šta znači? <... >

Ovo je komentar. Kompajler ga ignoriše.

```
#include <stdio.h>
/* The simplest C Program */
int main()
{
  printf("Hello World\n");
  return 0;
}
```

main() funkcija je uvek na samom početku programa.

Blok koda ("lexical scopes") odvojen je zagradama { … }

Return '0' iz ove funkcije

Štampanje poruke. '\n' znači "new line".

Značenje naredbi

#include <stdio.h>

Naredbom se od prevodioca traži da uključi (include) u program datoteku stdio.h koja sadrži informacije nužne za korišćenje funkcije printf i mnogih drugih. Datoteke sa ekstenzijom .h nazivaju se datoteke zaglavlja (eng. header files) i njihovo stavljanje u oštre zagrade < > informiše prevodilac da se radi o

standardnim datotekama zaglavlja, koje se nalaze na unapred određenim mestima u sistemu datoteka.

Sledeća linija predstavlja deklaraciju funkcije main:

int main(void)

Funkcija može uzimati jedan ili više argumenata i obično vraća neku vrednost. Deklaracijom se uvodi ime funkcije, broj i tip argumenata koje uzima i tip povratne vrednosti. U našem primeru ime funkcije je main i ona ne uzima ni jedan argument. To je deklarisano tako što je u male zagrade stavljena ključna rec void (eng. void=prazan). Povratna vrednost je tipa int, što je oznaka za celi broj.

Iza malih zagrada dolaze velike zagrade koje omeđuju telo programa.

Telo funkcije - programa je sastavljeno od deklaracija promenljivih i izvršnih naredbi. Sve deklaracije promenljivih dolaze pre prve izvršne naredbe. U našem primeru nemamo deklaraciju promenljivih. Telo sadrži samo dve naredbe: poziv funkcije printf i return naredbu:

```
{
printf(“Hello World.\n");
return 0;
}
```

Još o kompajleru

```
#include <stdio.h>
/* The simplest C Program */
int main(int argc, char
**argv)
{
  printf("Hello World\n");
  return 0;
}
```

Preprocess

Kompilacija se izvodi u dva prolaza:
"Preprocessing" i "Compiling"

Zašto ?

```
__extension__ typedef  unsigned long long
int   __dev_t;
__extension__ typedef  unsigned int
__uid_t;
__extension__ typedef  unsigned int
__gid_t;
__extension__ typedef  unsigned long int
__ino_t;
__extension__ typedef  unsigned long long
int   __ino64_t;
__extension__ typedef  unsigned int
__nlink_t;
__extension__ typedef  long int   __off_t;
__extension__ typedef  long long int
__off64_t;
extern void flockfile (FILE *__stream)  ;
extern int ftrylockfile (FILE *__stream)  ;
ex     void funlockfile                   ;
in    in(int argc, ch
{
      o wo
  return 0
}
```

U Predprocesiranju, source code je "expanded" u formu koja je jednostavnija kompajleru za razumevanje. Svaka linija koja počinje sa '#' je linija koju Predprocesor mora interpretirati.

- Includovani fajlovi su "pasted in" (#include)
- Makroi su "expanded" (#define)
- Komentari su ''stripped out'' (/* */ , //)

\ ?

my_program

Compile

Kompajler onda konvertuje dobijeni tekst u binarni kod, koji CPU može direktno pokrenuti.

Šta je "Memorija"?

Memorija je kao velika tabela numerisanih mesta, gde bitovi podataka mogu biti zapamćeni.

Broj mesta je njegova Adresa.
Samo jedan byte vrednosti može biti zapamćen na jednom mestu.

Neke vrednosti "logical" podataka zahtevaju više od jednog mesta, kao npr. karakter string "Hello\n"

Tip označava logičku veličinu podatka za smeštaj u memoriji. Neki jednostavni tipovi podataka su:

```
char
char [10]
int
float
int64_t
```

a single character (1 slot)
an array of 10 characters
signed 4 byte integer
4 byte floating point
signed 8 byte integer

ne uvek…

Šta je Signed?…

Adr	Value
0	
1	
2	
3	72?
4	'H' (72)
5	'e' (101)
6	'l' (108)
7	'l' (108)
8	'o' (111)
9	'\n' (10)
10	'\0' (0)
11	
12	1-25

Šta je promenljiva?

symbol table?

Promenljiva imenuje mesto u memoriji gde se pamti Vrednost za nju određenog Tipa.

Prvo moramo da definišemo promenljivu dajući joj ime i specificirajući njen tip, a opciono i njenu inicijalnu vrednost.

declare vs define?

```
char x;
char y='e';
```

Inicfijalna vrednost x nije definisana

Inicijalna vrednost

Ime

Koja imena su legalna?

Tip je jedan karakter (char)

extern? static? const?

Kompajler je postavlja negde u memoriji.

Symbol	Addr	Value
	0	
	1	
	2	
	3	
x	4	?
y	5	'e' (101)
	6	
	7	
	8	
	9	
	10	
	11	
	12	

Tipovi promenljivih

Tipovi promenljivih:

char jedan znak,
short \kratki" celi broj,
int celi broj,
long \dugi" celi broj,
float realan broj jednostruke preciznosti,
double realan broj dvostruke preciznosti.

Ključne reči

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

Logicki podaci

✓U ANSI C-u (standard C90) ne postoji poseban logički tip. Svaki celobrojni tip može preuzeti ulogu logičkog tipa tako što se vrednost različita od nule interpretira kao istina, a nula kao laž.

✓Standard C99 uvodi poseban logički tip Bool koji prima samo vrednosti 0 i 1 (‘’laž" i ‘’istina"). I nadalje se celobrojne promenljive mogu koristiti za prezentaciju logičkih vrednosti, ali upotreba novog tipa može učiniti program jasnijim. Ključna reč Bool je izabrana da se izbegnu konflikti sa aplikacijama koje su implementirale svoj tip bool. Korisnik može uključiti datoteku zaglavlja <stdbool.h> u kojoj se donira bool kao sinonim za Bool tip podatka, kada se uvode simbolička imena *true* i *false*.

Promenljive i konstante

Konstanta je specifična vrednost koja se ne može modifikovati u programu.

Specijalne konstante korišćene sa stringom:

\n new line
\t tabulator
\r carriage return
\b backspace
\" escape double quote
\0 end string

Simboli definisani predprocesorskim naredbama:

#define ESC '\033' /* ASCII escape */

Promenljive specificiraju mesto u memoriji koje sadrži vrednost datu svojim tipom koja se može modifikovati kroz program. Mogu biti globalne i lokalne.

```
short x;
long y;
unsigned a,b;
long double lb;
unsigned short z;
```

- **sizeof() je funkcija koja vraća veličinu promenljive u bajtima.**

- **Promenljive se moraju inicijalizovati pre nego se koriste u programu, jer će drugačije dobiti proizvoljnu vrednost.**

Promenljive i konstante

Celobrojne dekadne: **1; 50; 15+55; -55**	Realne u fiksnom zarezu: **3.14; +3.0; -0.314;**	Realne u pokretnom zarezu: **3.14 E 0; -0.314E-2**	Kompleksne konstante: **(real,imag)** **(3.14, 0.13)**
Binarne konstante: **10111; 000l**	Oktalne konstante: **0567; 0753; 0104**	Heksadecimalne konstante: **0XIF; 0XAA; 0Xll**	Logičke konstante: **true; false**
Znakovne konstante: **' A ' ; ' B '**	String konstante: **"Beograd";"Alfa l"**	Binarne string konstante: **'10001111' '11000011'**	Simboličke konstante: **ZERO; SPACE**

KONVERZIJA TIPOVA

- **Celobrojni izrazi se automatski konvertuju u realne ukoliko se upotrebe u kontekstu u kojem se očekuje realan izraz, npr. ukoliko želimo dodeliti neki celobrojni izraz realnoj promenljivoj (*promocija*).**

```
(int)(2.541 * 3.17);
```

- **Realan izraz često možemo upotrebiti u kontekstu u kojem se očekuje celobrojni izraz, npr. kao argument neke funkcije koja očekuje celobrojni argument, ili pri dodeli realnog izraza celobrojnoj promenljivoj. Takve konverzije praćene su *gubitkom informacija* (u konkretaom primeru *odsecanjem decimali*), i obično ih nazivamo (*degradacija*).**

```
float broj_l, broj_2;
(int)broj_l / broj_2;
```

Izrazi i njihovo izvođenje

Izrazi kombinuju vrednosti korišćenjem operatora, saglasno određenim prioritetima izvršavanja operacija.

```
1 + 2 * 2     → 1 + 4     → 5
(1 + 2) * 2   → 3 * 2     → 6
```

Simbolima pridružujemo njihove vrednosti pre nego što vršimo njihovu kombinaciju.

```
int x=1;
int y=2;
x + y * y     → x + 2 * 2   → x + 4   → 1 + 4   → 5
```

Operatori poređenja se koriste da bi uporedili vrednosti odrteđenih veličina.
U jeziku C, 0 znači “false”, a *bilo koja druga vrednost* znači “true”.

```
int x=4;
(x < 5)              → (4 < 5)            → <true>
(x < 4)              → (4 < 4)            → 0
((x < 5) || (x < 4))  → (<true> || (x < 4))  → <true>
```

Tablice istinitosti i,ili,ni,nili ?

Operatori poređenja i matematički operatori

```
== equal to
<  less than
<= less than or equal
>  greater than
>= greater than or equal
!= not equal
&& logical and
|| logical or
!  logical not
```

```
+  plus
-  minus
*  mult
/  divide
%  modulo
```

```
&  bitwise and
|  bitwise or
^  bitwise xor
~  bitwise not
<< shift left
>> shift right
```

Pravila prioriteta su striktno definisana ali često su teška za pamćenje ili neintuitivna. Ako sumnjamo u to, dodavaćemo zagrade da bi ih učinili eksplicitnim. U svakom slučaju, kompajler će nam dati upozorenje "Suggest parens around …" – do it!

Oprez za deljenje:

- Ako je drugi argument integer, rezultat će biti integer tipa :
  5 / 10 → 0 *ali je* 5 / 10.0 → 0.5
- Deljenje sa 0 će prouzrokovati FPE

Ne mešati & and &&..
1 & 2 → 0 *ali je* 1 && 2 → <true>

Označavanje operatora

```
x = y  assign y to x
x++    postfiksni-increment x
++x    pre fiksni -increment x
x--    post fiksni -decrement x
--x    pre fiksni -decrement x
```

```
x += y    dodeljuje (x+y) na x      x=(x+y)
x -= y    dodeljuje (x-y) na x      x=(x-y)
x *= y    dodeljuje (x*y) na x      x=(x*y)
x /= y    dodeljuje (x/y) na x      x=(x/y)
x %= y    dodeljuje (x%y) t na x    x=(x%y)
```

Zapamtiti razliku između ++x and x++:

```
int x=5;
int y;
y = ++x;
/* x == 6, y == 6 */
```

```
int x=5;
int y;
y = x++;
/* x == 6, y == 5 */
```

Ne praviti konfuziju sa = i ==! Kompajler će upozoriti“suggest parens”.

```
int x=5;
if (x==6)   /* false */
{
  /* ... */
}
/* x is still 5 */
```

```
int x=5;
if (x=6)   /* always true */
{
  /* x is now 6 */
}
/* ... */
```

preporučuje se

Skraćeni prikaz operatora

C dopušta skraćeni prikaz operatora koji ima isti efekat kao i standardno.

Ako ++ ili – koristimo kao prefiksne, promenljive se modifikuju u razvoju izraza:

```
a = 3;
b = ++a + 3;        /* b = 4 + 3 = 7 i a = 4  */
```

Ako ++ ili -- koristimo kao postfiksne, promenljive se prvo koriste a onda modifikuju:

```
a = 3;
b = a++ + 3;        /* b = 3 + 3 = 6 i a = 4 */
```

Operatori mogu biti kombinovani:

```
a += b;             /* a = a + b */
a *= b;             /* a = a * b */
```

Uslovni operator

Uslovni operator **(?:) je specifičan ternarni operator jezika C. lma tri operanda i piše se u obliku npr. Ne postoji u drugim programskim jezicima i zamenjuje tri linije koda.**

x=(a?b:c)

Prvo se izračunava vrednost izraza a i ako ima vrednost logičke istine (različito od 0), izračunava se vrednost izraza *b* i to predstavlja vrednost celog izraza. Ako a ima vrednost logičke neistine (=0), izračunava se vrednost izraza c i to predstavlja rezultat celog izraza. Bitno je da se od izraza *b* i c uvek izračunava samo jedan.

Adresni operatori

***Adresni operatori* služe za manipulisanje sa adresama podataka. U užem smislu, tu spadaju:**

ampersand

(&) - unarni operatori za nalaženje adrese datog podatka i

(*) - za posredni pristup podatku pomoću pokazivača - koji se naziva i operatorom indirektnog adresiranja.

Aritmetika

Aritmetička izračunavanja

- Koristiti `*` za množenje i `/` za deljenje
- Integer deljenje zanemaruje ostatak
  - `7 / 5` odgovara sa `1`
- Operator deljernja po modulu(`%`) vraća ostatak deljenja
  - `7 % 5` odgovara sa `2`

Prioriteti operacija

- Neki aritmetički operatori izvršavaju se pre nego drugi (npr, množenje pre sabiranja)
  - Koristiti zagrade uvek kada su potrebne
- Primer: Naći srednju vrednost tri promenljive `a`, `b` i `c`
  - **NE KORISTITI:** `a + b + c / 3`
  - **KORISTITI:** `(a + b + c ) / 3`

Aritmetika

Aritmetički operatori:

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	`+`	$f + 7$	`f + 7`
Subtraction	`-`	$p - c$	`p - c`
Multiplication	`*`	bm	`b * m`
Division	`/`	x / y	`x / y`
Modulus	`%`	$r \bmod s$	`r % s`

Pravila prioriteta:

Operatori	Operacije	Redosled izvršavanje
`()`	Zagrade	Prvo se izračunava. Ako su zagrade ugnježdene, izraz u unutrašnjem paru se prvi izračunava Ako imamo nekoliko pari zagrada "na istom nivou " (npr., neugnježdene), izvršavaju se sa leva u desno.
`*`, `/`, or `%`	Množenje, deljenje, moduli	Izračunavaju se kao druge. Ako ih ima nekoliko,izvršavaju se sa leva u desno.
`+` or `-`	Sabiranje i oduzimanje	Zadnje se izračunavaju. Ako ih ima nekoliko,izvršavaju se sa leva u desno.

ODLUKE: Relacioni operatori i operatori jednakosti

Standardni algebarski operatori jednakosti ili relacioni operatori	C izrazi za relacione operatore	Primeri C uslova	Značenje C uslova
Algebarski operatori			
=	==	`x == y`	**x** is equal to **y**
not =	!=	`x != y`	**x** is not equal to **y**
Relacioni operatori			
>	>	`x > y`	**x** is greater than **y**
<	<	`x < y`	**x** is less than **y**
>=	>=	`x >= y`	**x** is greater than or equal to **y**
<=	<=	`x <= y`	**x** is less than or equal to **y**

Logički operatori

Logički operatori kao operande uzimaju logičke promenljive, a to su tačno (true) i netačno (false) odnosno u C-u je to celobrojni tip pri čemu se vrednost 0 tumači kao netačno, a sve različito od nula kao tačno. Postoje tri logička operatora, to su i, ili i negacija.

Operator	Opis	Primer, A=1, B=0
&&	logički operator i (and), tačan je ako su oba operanda tačna, odnosno u C-u veća od 0	(A && B) je 0
||	logički operator ili (or), tačan ako je bar jedan od operanada tačan	(A || B) je 1
!	logička negacija (not), tačan ako je operand netačan i netačan ako je operand tačan	!(A && B) je 1

Logički i operatori poredjenja

Operatori poredjenja vraćaju 1 ili 0 u zavisnosti od toga kakav je rezultat komparacije. Operatori poredjenja u jeziku C su:

< (manje od)
> (veće od)
<= (manje ili jednako)
>= (veće ili jednako)
== (jednako)
!= (nejednako)

&& i || su logički operatori ''I'' AND i ''ILI'' OR

(a != b) && (c > d)
(a < b) || (c > d)

Operatori nad bitovima

Programski jezik C podržava operacije koje se izvršavaju direktno nad bitovima. Kao operatori ovakvih operacija najčešće se koriste celobrojni tipovi podataka koji se zapisuju u heksadecimalnom ili oktalnom formatu, a operacije se izvršavaju direktno nad bitovima kojima se predstavljaju ovi brojevi.

Operacije nad bitovima se mogu podeliti u dve grupe:

- **logičke operacije i**
- **operacije pomeranja bitova.**

Sve operacije sa objašnjenjem date su u tabeli na primeru dve celobrojne vrednsoti A = 60, B = 13, njihova binarna reprezentacija je

A = 0011 1100, B = 0000 1101.

Operator	Opis	Primer A = 0011 1100, B = 0000 1101
&	bitovna konjunkcija	(A & B) je 0000 1100
|	bitovna disjunkcija	(A | B) je 0011 1101
^	bitovna ekskluzivna disjunkcija - XOR, bit je setovan ako su dva ulazna bita različite vrednosti	(A ^ B) je 0011 0001
~	promena vrednosti bit-a	(~A) je 1100 0011
<<	šiftovanje bitova ulevo, iza znaka se zadaje broj za koliko se bitovi pomeraju ulevo, za upražnjena mesta upisuje se 0	A << 2 je 1111 0000
>>	šiftovanje bitova udesno, iza znaka se zadaje broj za koliko se bitovi pomeraju udesno, za upražnjena mesta upisuje se 0	A >> 2 je 0000 1111

Jednostavan C Program

```
#include <stdio.h>
#define STOP 0

/* Funkcija: main */
/* Opis: brojač na dole od korisničkog ulaza do naredbe
STOP */
main()
{
  /* deklarisanje promenljivih */
  int counter;  /* brojač tipa integer - celobrojni*/
  int startPoint; /* startna tačka za brojač*/
  /* zahtev korisniku za unos startne tačke*/
  printf(„Ukucajte jedan pozitivni broj: ");
  scanf("%d", &startPoint);  /* čitanje startPoint */
  /* brojanje nadole do naredbe STOP*/
  for (counter=startPoint; counter >= STOP; counter--)
    printf("%d\n", counter); /* prikaz brojača*/
}
```

Predprocesorske direktive

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

- **Pre kompajliranja, kopira sadržaj header file (stdio.h) u oblast izvornog koda - source code.**
- **Header files tipično sadrži opis funkcija i promenljivih potrebnih programu.**
  - **nema restrikcije – može biti bilo koji C source code**

```
#define STOP 0
```

- **Pre kompajliranja, zamenjuje sve instance stringa "STOP" sa stringom "0"**
- **Pozivanje *macro***
- **Koristi se za vrednosti koje se ne menjaju tokom izvršenja, ali se mogu promeniti ako ih program ponovo koristi. (Mora se izvršiti rekompilacija.)**

Komentari

Počinju sa `/*` i završavaju sa `*/`

Počinju sa `//` . . . . . samo u jednoj liniji

Mogu biti sačinjeni iz velikog broja linija.

Ne može biti komentar u komentaru.

Komentari se ne prepoznaju u okviru stringa.

- **Primer: "my/*don't print this*/string"**
  biće prezentovano kao: my/*don't print this*/string

Kao što je opisano, koristiti komentare da pomognu programeru razumevanje programa, njegovu namenu, module, promenljive itd…

main Funkcija

Svaki C program mora imati funkciju `main()`.

Ovo je kod koji se izvršava kada pokrenemo program.

Kod funkcije se piše unutar velikih zagrada:

```
int main()
{
  /* ovde se piše kod programa */
return 0;}
```

Deklaracija promenljivih

Promenljive se koriste kao imena korišćenih podataka. Svaka promenljiva ima svoj *tip*, koji govori kompajleru kako će se interpretirati podaci i koliko mesta u memoriji traže, svaki od njih pojedinačno.

TIPOVI PODATAKA predstavljaju jedan od najznačajnijih pojmova u okviru programskih jezika. Sve promenljive koje se koriste u jednom programu imaju :

- ***atribut tipa*, što određuje skup vrednosti koji im se može dodeljivati,**
- ***format* predstavljanja ovih vrednosti u memoriji računara,**
- ***skup osnovnih operacija* koje se nad njima mogu izvršavati i veze sa drugim tipovima podataka.**

```
int counter;
int startPoint;
```

`int` je reč za predstavku integer tipa podatka u jeziku C.

Ulaz i Izlaz

Postoji mnoštvo I/O funkcija u *C Standardnoj biblioteci*.

Moramo napisati `#include <stdio.h>` da bi ih koristili.

```
printf("%d\n", counter);
```

- **String sadrži print naredbu za štampanje sadržaja i direktive za formatiranje promenljive counter.**
- **Ova programska rečenica kaže da se odštampa promenljiva `counter` kao decimalni integer, praćena sa prelaskomu novi red - linefeed (`\n`). Ako nije drugačije predvidjeno u programu, prikazaće se na monitoru.**

```
scanf("%d", &startPoint);
```

- **String sadrži direktive za formatiranje očekivanog ulaza.**
- **Ova programska rečenica čita decimalni integer i dodeljuje ga promenljivoj `startPoint`.**

Još o Izlazu

Mogu se štampati proizvoljni izrazi a ne samo promenljve:

```
printf("%d\n", startPoint - counter);
```

Štampanje više izraza jednom naredbom:

```
printf("%d %d\n", counter,
        startPoint - counter);
```

Različite opcije formatiranja:

- `%d` **decimalni integer**
- `%x` **heksadecimalni integer**
- `%c` **ASCII karakter**
- `%f` **realni broj (floating-point)**

Primeri

Kod:

```
printf("%d je prvi broj.\n", 43);
printf("43 plus 59 decimalno je %d.\n", 43+59);
printf("43 plus 59 hex je %x.\n", 43+59);
printf("43 plus 59 kao karakter je %c.\n", 43+59);
```

produkuje izlaz:

```
43 je prvi broj.
43 + 59 decimalno je 102.
43 + 59 hex je 66.
43 + 59 kao karakter je f.
```

Primeri ulaza

Mnogo različitih formata može biti upotrebljeno za unos podataka:

```
scanf("%c", &nextChar);
```

- Čita karakter sa ulaza (tastatura) i pamti ga u promenljivoj nextChar

```
scanf("%f", &radius);
```

- Čita realni broj sa ulaza (floating point number) i pamti ga u promenljivoj radius

```
scanf("%d %d", &length, &width);
```

- Čita dva decimalna integer broja (odvojena blanko mestom), pamti prvi u promenljivoj length a drugi u promenljivoj width.

Mora se koristiti ampersend (&) za postavljanje adrese gde će podatak, koji smo sa tastature uneli, biti smešten u memoriji.

Primer jednostavnog programa s=a+b+c

```
#include <stdio.h>
#include <math.h>
main( )
{
int a, b, c, sum;
a = 1;  b = 2;  c = 3;
sum = a + b + c;
printf("suma brojeva a,b ic je: %d", sum);
return 0;
}
```

Programski jezik C

nastavak

Grananje programa i petlje

Naredbe grananja

NAREDBE ?

Naredbe

C raspolaže malim skupom naredbi

Svaka naredba (iskaz) završava sa ; (terminator iskaza)

Osnovne grupe naredbi:

- naredbe grananja i izbora: `if, switch`
- naredbe petlji: `while, do while, for`
- naredbe za nasilnu promenu toka: `break, continue, goto, return`

Komponovana (složena) naredba

jednu ili više naredbi grupišemo korištenjem velikih zagrada { }

opšti oblik:

```
{
        iskaz1;
        iskaz2;
           …
        iskazN;
}
```

Složena naredba izvršava se tako što se izvršavaju jedna po jedna prosta naredba (iskaz) od kojih je sastavljena

Ekvivalentna je BLOKU naredbi u PASCALU BEGIN ... END

if naredba

Sintaksa if naredbe je sledeća:

if(uslov)

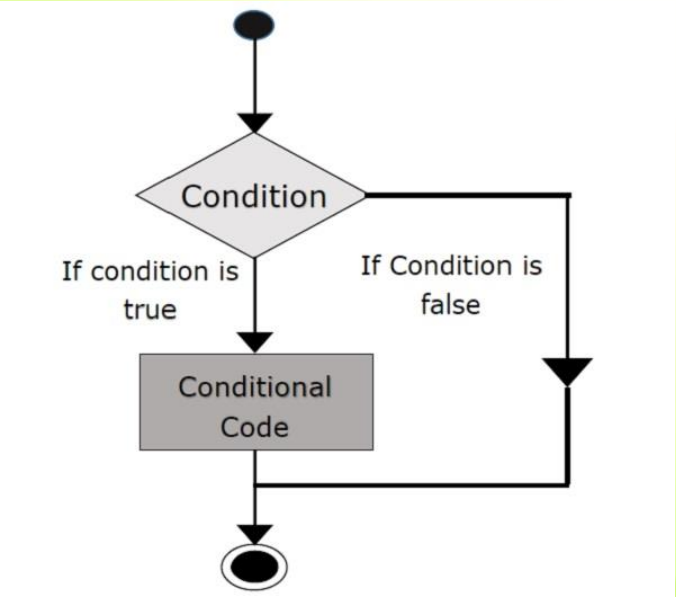
{ blok naredbi koje će se izvršiti ako je uslov tačan (if grana) }

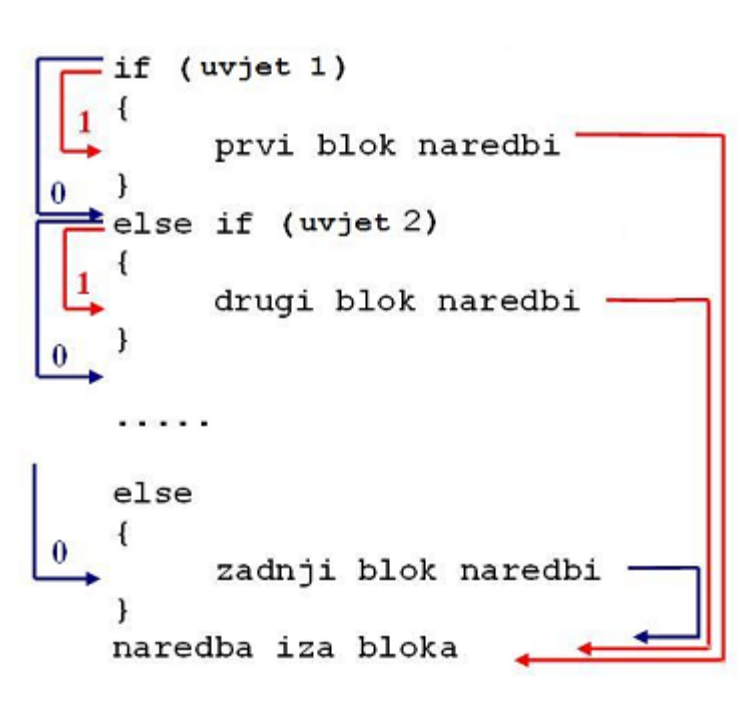
else{ blok naredbi koje će se izvršiti ako uslov nije tačan (else grana) }

if . then . else naredbe se mogu koristiti sa ili bez naredbe else. Tri forme naredbe izgledaju:

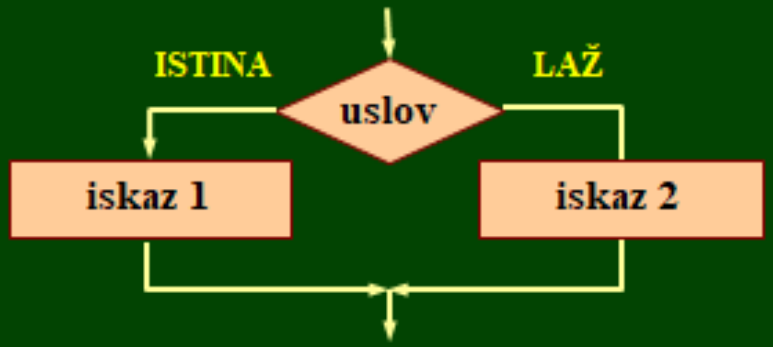
```
if (izraz)
naredba1

if (izraz)
naredba 1
else
naredba 2

if (izraz)
naredba 1
else if
naredba 2
else
.....
```

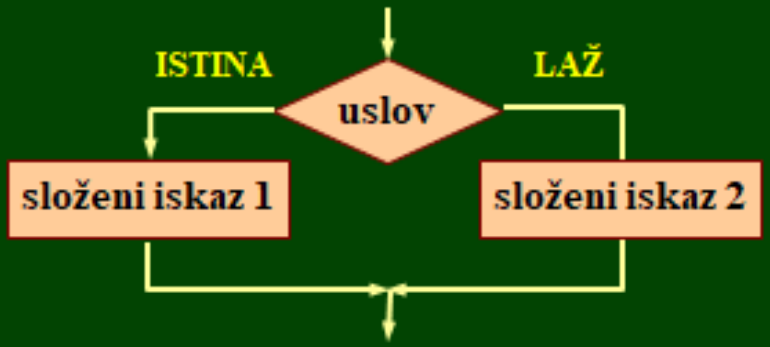
U sva tri slučaja, kada je izraz istinit, izvršava se natedba 1. Ako je izraz neistinit, onda u prvom slučaju naredba 1 se preskače i nastavlja dalje sa programom, i u drugom slučaju, naredba 2 u okviru else se izvršava.





Dvoblokovsko grananje

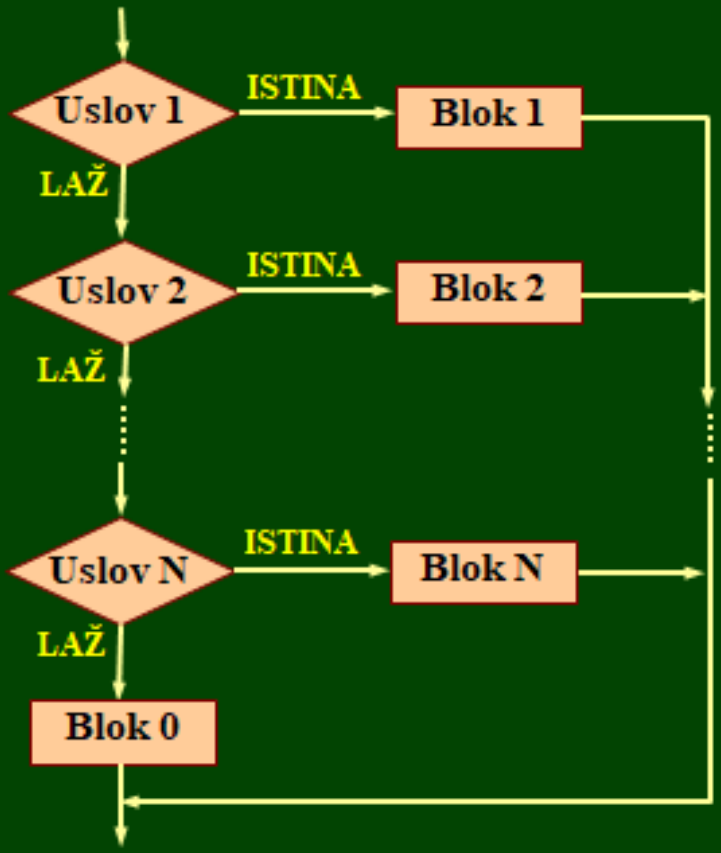





```
if (uslov)
   iskaz1;
else
   iskaz2;
```

```
if (uslov)
{
   iskaz1;
    ...
   iskazN;
}
else
{
   iskaz1;
    ...
   iskazN;
}
```

Višeblokovsko grananje

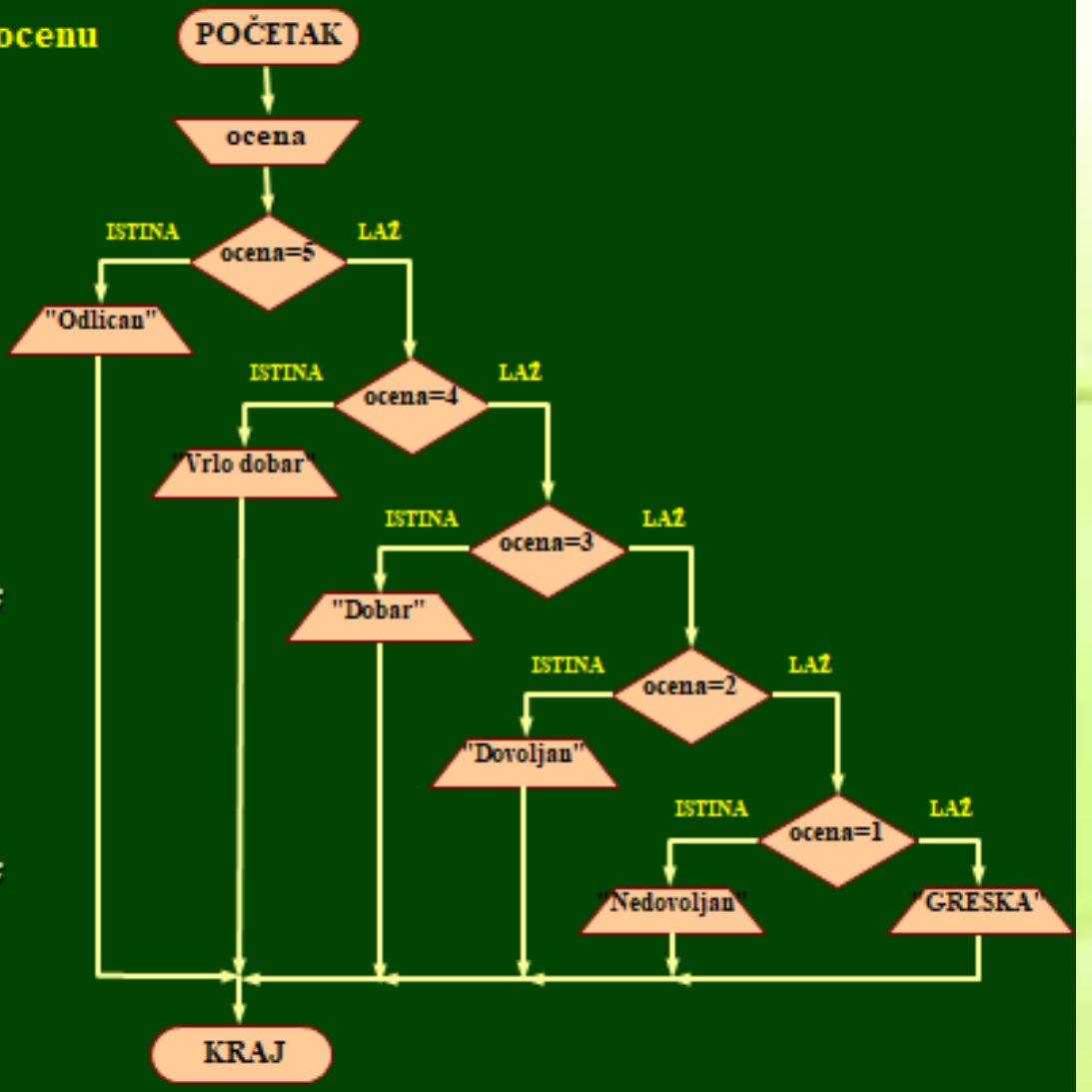



```
if (uslov1)
   iskaz1;
else if (uslov2)
   iskaz2;
else if (uslov3)
   iskaz3;
   ...
else if (uslovN)
   iskazN;
[else
   iskaz0;]
```

Primer

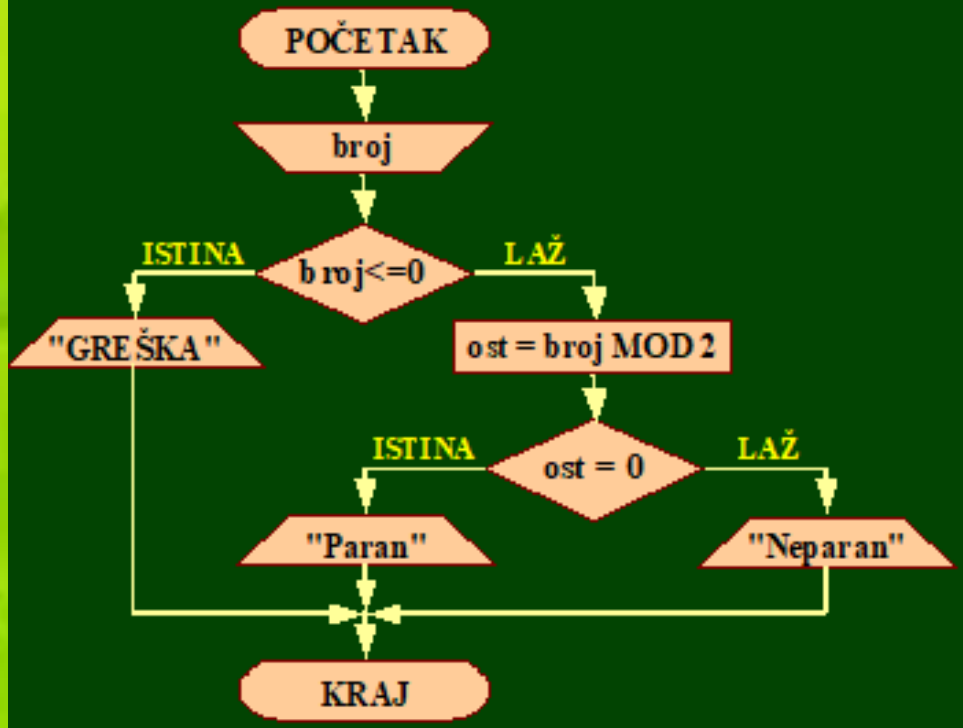
Program koji za učitanu numeričku ocenu ispisuje opisnu ocenu.

```
#include <stdio.h>
main()
{
   int ocena;
   printf( "Ocena? " );
   scanf( "%d", &ocena );
   if (ocena==5)
      printf("Odlican\n");
   else if (ocena==4)
      printf("Vrlo dobar\n");
   else if (ocena==3)
      printf("Dobar\n");
   else if (ocena==2)
      printf("Dovoljan\n");
   else if (ocena==1)
      printf("Nedovoljan\n");
   else
      printf("GRESKA\n");
}
```



Primer

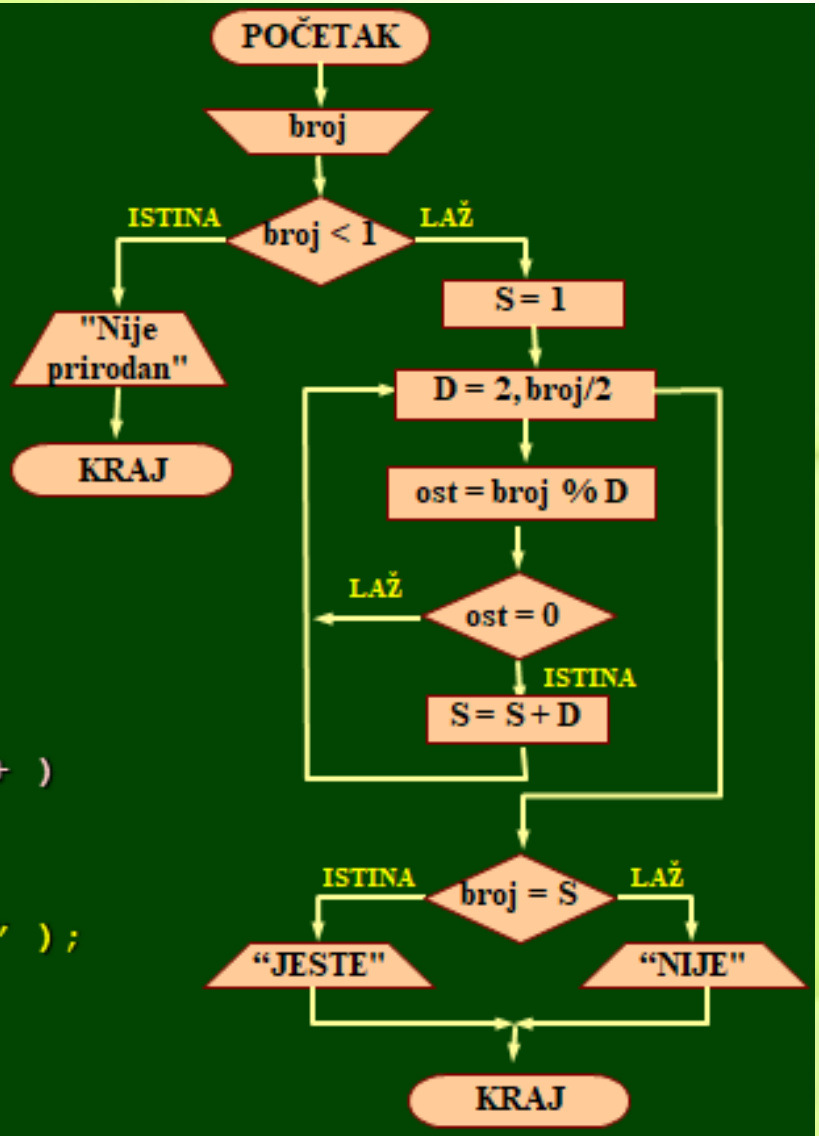
Program koji proverava da li je učitani prirodni broj paran ili neparan.




```
#include <stdio.h>
main()
{
    int broj, ost;
    printf( "Unesite broj: " );
    scanf( "%d", &broj );
    if ( broj<=0 )
       printf("Nije prirodan\n");
    else
    {
       if (broj % 2 == 0)
            printf("Paran\n");
       else
            printf("Neparan\n");
    }
}
```

Primer: **Učitati prirodan broj, pa proveriti da li je on savršen.**

```
#include <stdio.h>
main()
{
   int broj, d, s;
   printf("Unesite prirodan broj:");
   scanf("%d",&broj);
   if (broj<1)
      printf("Nije prirodan!");
   else
   {
      for ( s=1, d=2 ; d<=broj/2 ; d++ )
         s += (broj%d) ? 0 : d;
      printf("%d %s savrsen", broj,
             (broj==s) ? "je" : "nije" );
   }
}
```



Primer:

Najčešći način za implementaciju grananja u programima je if-else naredba. Implementacija zadatka koji učitava dva broja i ispisuje koji je veći u C-u bi izgledala ovako:

```
int main()
{ int a, b;
printf("Unesite prvi broj:");
scanf("%d", &a);
printf("Unesite drugi broj:");
scanf("%d", &b);
      if(a>b)
      { printf("Veci broj je %d", a); }
      else{ printf("Veci broj je %d", b); }
return 0; }
```

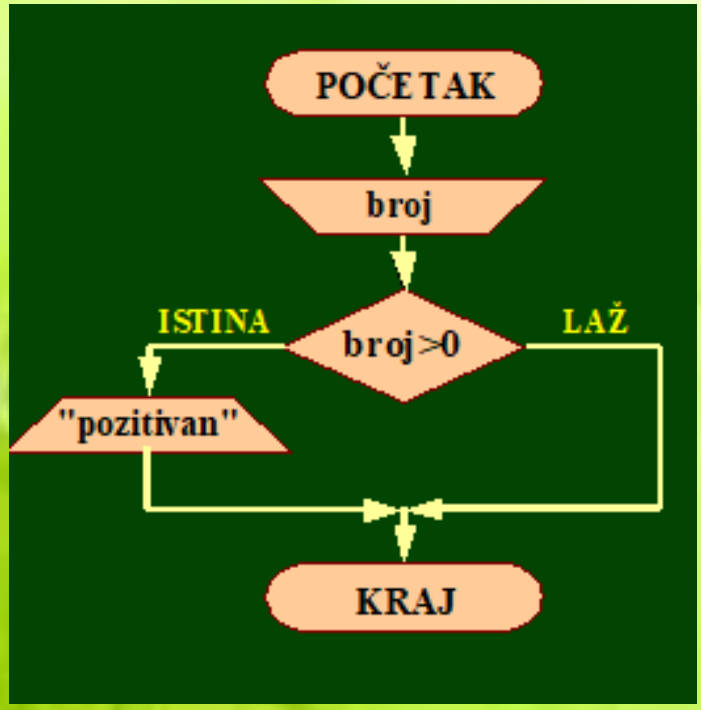
Ukoliko imamo više uslova od kojih nam zavisi tok izvršavanja programa koristimo izraz else if kome se, kao i if naredbi, zadaje uslov.

Sintaksa je sledeća:

```
if(uslov1){ blok naredbi 1 }
else if(uslov2){ blok naredbi 2 }
else if(uslov3){ blok naredbi 3 }
else{ blok naredbi 4 }
```

Program koji proverava da li je učitani broj pozitivan?

```
#include <stdio.h>
main()
{
     float broj;
     printf( "Unesite broj: " );
     scanf( "%f", &broj );
     if ( broj>0 )
        printf( "Broj je
   pozitivan\n", );
}
```



Naredba for

Opšti oblik

```
for (poc_izraz; uslov; izraz_petlje)
   iskaz;
```

ili

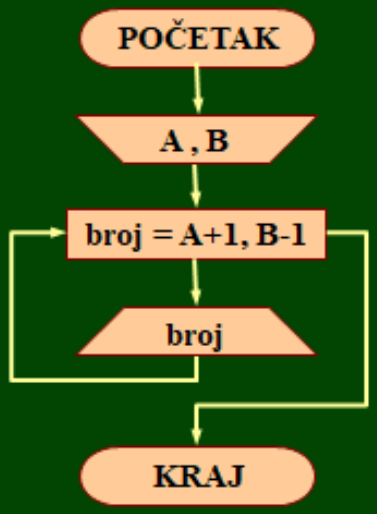
```
for (poc_izraz; uslov; izraz_petlje)
 {
    iskaz1;
     ...
    iskazN;
 }
```

Ekvivalentno sledećoj `while` petlji:

```
pocetni_izraz;
while (uslov)
{
    iskaz; izraz_petlje;
}
```

for ?

Primer: Program koji ispisuje cele brojeve iz intervala <A,B>.

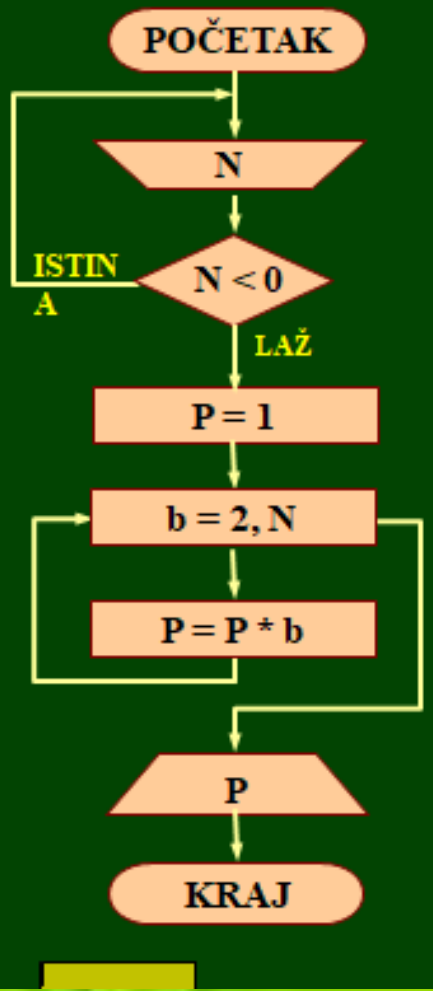



```
#include <stdio.h>
main()
{
   int a, b, broj;
   printf("Unesite granice intervala: ");
   scanf("%d %d",&a, &b);
   for ( broj=a+1 ; broj<=b-1 ; broj++ )
      printf(" %d", broj);
}
```

izraz petlje izostavljen
ali je implementiran u sledećem
iskazu!!!
`printf(" %d", broj++);`

```
#include <stdio.h>
main()
{
   int a, b, broj;
   printf("Unesite granice intervala: ");
   scanf("%d %d",&a, &b);
   for ( broj=a+1 ; broj<=b-1 ; )
      printf(" %d", broj++);
}
```

Primer: **Program koji računa n!**




```
#include <stdio.h>
main()
{
    int n, p;
    do
    {  printf("n="); scanf("%d",&n); }
    while (n<0);
    for ( p=1 ; n ; p*=n-- );
    printf("%d", p);
}
```

```
#include <stdio.h>
main()
{
    int n, b, p;
    do
    {  printf("n="); scanf("%d",&n); }
    while (n<0);
    for ( p=1, b=2 ; b<=n ; b++ )
       p=p*b;
    printf("%d!=%d", n, p);
}
```

dva početna izraza
međusobno odvojeni zapetom

for naredba

For naredba omogućuje način za iterativni postupak u odredjenom rangu.

for (initialization; termination; increment)
{ statement }

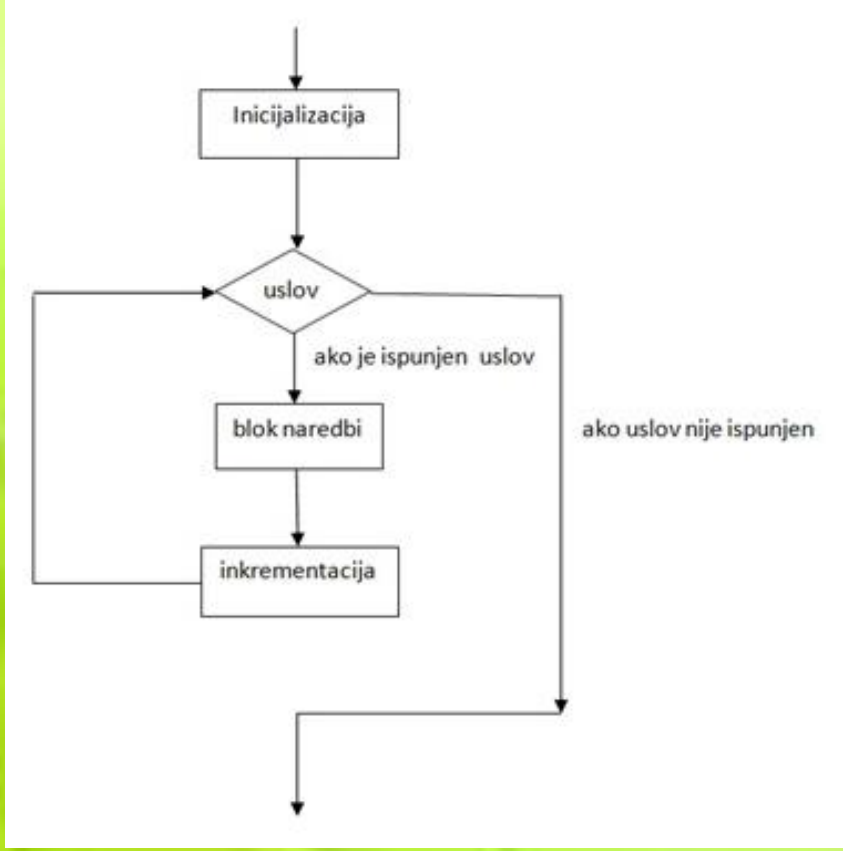
Svi elementi u petlji su opcionalni ali se moraju odvajati sa ;

for (; ;);

Naredba break se može koristiti za prekid petlje bez čekanja da se završi test uslova i evaluira sa istinom

Naredba continue se može koristiti da preskoči izvršavanje tela petlje i reevaluira završavanje uslova

```
for (int x =0; x < 100; x = x + 3)
{
if (x == 27) continue;
else printf(.%d.,x);
}
```



for petlja

Prvo se izvrši inicijalizacija, i zatim se proverava uslov petlje.

Ukoliko je njegova vrednost tačna (odnosno u C-u veća od 0), ulazi se u blok naredbi koji je naveden u petlji. Kada se izvrše sve naredbe izvršava se inkrementacija i tok programa se vraća na ponovno ispitivanje uslova.

Opisani proces se ponavlja sve dok je ispunjen uslov. Ako uslov nije ispunjen izlazi se iz petlje i nastavlja se sa izvršavanjem naredbi koje su navedene posle for petlje.

Deo za inkrementaciju ne mora uvek da znači povećanje vrednosti neke promenljive za jedan, to može da bude proizvoljno povećanje ili smanjenje vrednosti promenljive, a najčešće se menja vrednost promenljive koja se pojavljuje u uslovu.

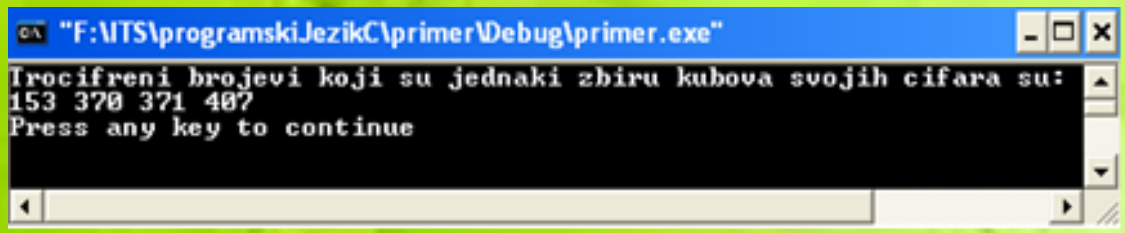
Primer:

Ako bismo posmatrali problem izračunavanja zbira svih parnih brojeva u intervalu od nula, do nekog zadatog broja n, ovaj zadatak bi se mogao jednostavno implementirati korišćenjem for naredbe.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ int i, n; int zbir = 0;
printf("Unesite broj n:\n");
scanf("%i", &n);
for(i=0; i<=n; i=i+2)
    { zbir = zbir+ i; }
    printf("Zbir parnih brojeva od 0 do %d jednak je %d.", n, zbir);
return 0; }
```

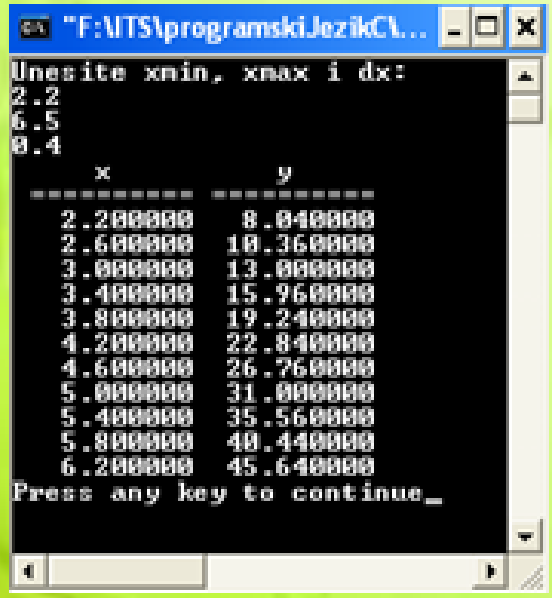
Primer: Program za nalazenje svih trocifrenih brojeva koji su jednaki zbiru kubova svojih cifara, abc = a^3+ b^3+c^3

```
#include <stdio.h>
    #include <math.h>
    #define KUB 3
main()
    {
      int a,b,c,m;
      double k;
      printf("Trocifreni brojevi koji su jednaki zbiru kubova svojih cifara su:\n");
      //petlja za prolazak kroz sve trocifrene brojeve
      for(m=100;m<999;m++)
      {
        a = m/100; //cifra ne mestu stotica
        b = m%100/10; //cifra ne mestu desetica
        c = m%10; //cifra ne mestu jedinica
        k = pow(a,KUB) + pow(b,KUB) + pow(c,KUB); //izracunavanje kubova
        if(m == k)
          //stampaj brojeve
          printf("%d ",m);  }
      printf("\n"); }
```



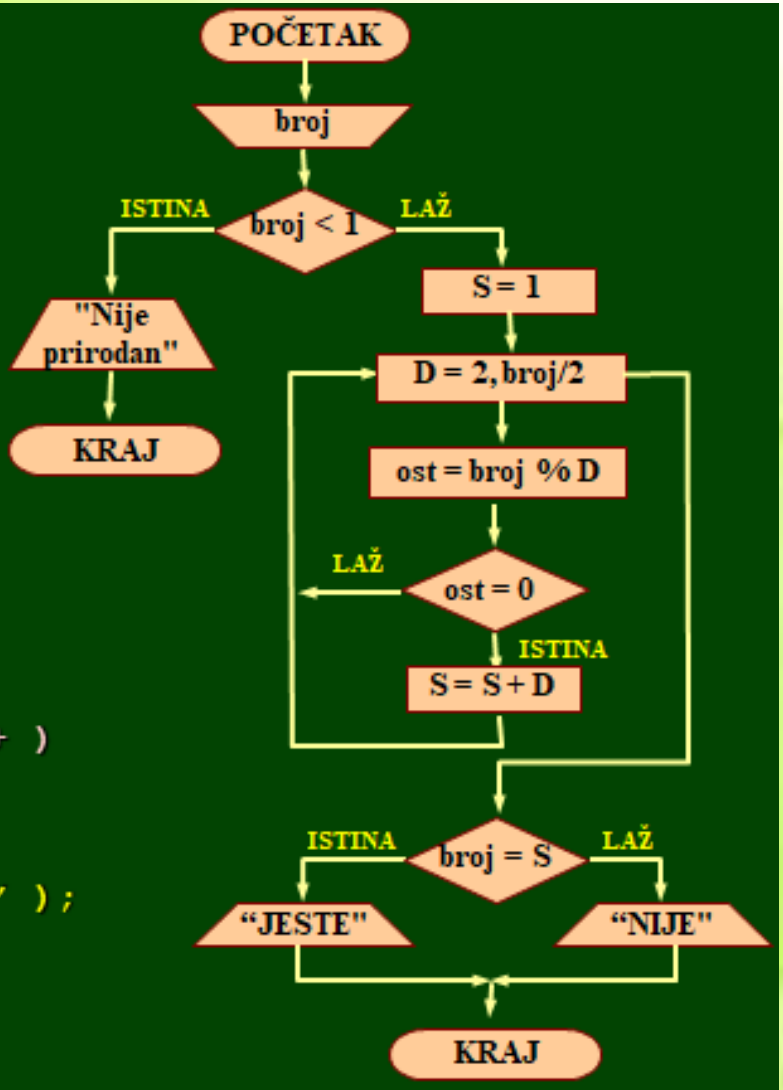
Primer: Program za tabeliranje funkcije: $y = (1+x+x^2)$ u opsegu od xmin do xmax sa korakom dx

```
#include <stdio.h>
    main()
    {
      double x, xmin, xmax,dx,y;
      // unos parametara
      printf("Unesite xmin, xmax i dx:\n");
      scanf("%lf%lf%lf",&xmin,&xmax,&dx);
      // radi lepseg prikaza
      printf("     x          y\n");
      printf(" ========== ==========\n");
      // tabeliranje funkcije
      for(x=xmin;x<xmax;x+=dx)
      {
        y= 1+x+x*x;
        printf(" %10.6f %10.6f\n",x,y);
      }
    }
```



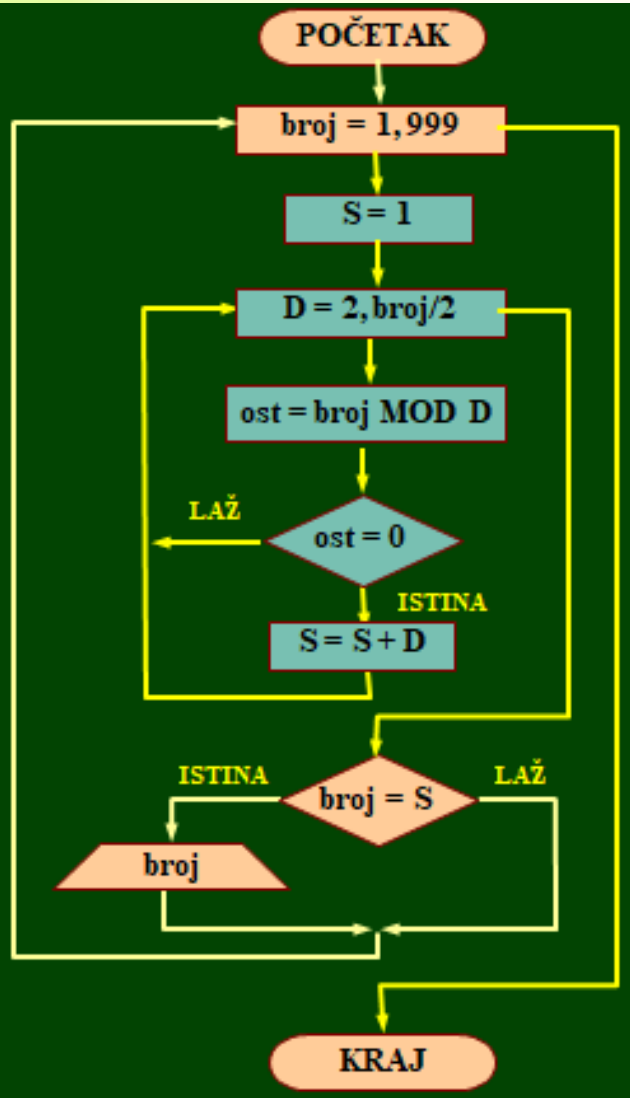
Primer: **Učitati prirodan broj, pa proveriti da li je on savršen.**

```
#include <stdio.h>
main()
{
   int broj, d, s;
   printf("Unesite prirodan broj:");
   scanf("%d",&broj);
   if (broj<1)
      printf("Nije prirodan!");
   else
   {
      for ( s=1, d=2 ; d<=broj/2 ; d++ )
         s += (broj%d) ? 0 : d;
      printf("%d %s savrsen", broj,
             (broj==s) ? "je" : "nije" );
   }
}
```



Primer: **Ispisati sve savršene prirodne brojeve manje od 1000.**

```
#include <stdio.h>
main()
{
   int broj, d, s;
   for ( broj=1 ; broj<1000 ; broj++ )
   {
      for ( s=1, d=2 ; d<=broj/2 ; d++ )
         s += (broj%d) ? 0 : d;
      if (broj==s)
         printf(" %d", broj);
   }
}
```



Primer: **Program koji ispisuje prvih n redova sledećeg trougla:**

```
*
**
***
****
*****
.
.
.
```

```
#include <stdio.h>
main()
{
    int n, red, broj;

    do
    {  printf("n=? "); scanf("%d",&n); }
    while (n<1 || n>20);

    for ( red=1 ; red<=n ; red++ )
    {
       for ( broj=1 ; broj<=red ; broj++ )
          printf("*");

       printf("\n");

    }
}
```

Primer: **Program koji ispisuje prvih n redova sledećeg trougla:**

```
1
12
123
1234
.
.
.
123456789
1234567890
12345678901
.
.
.
```

```
#include <stdio.h>
main()
{
    int n, red, broj;

    do
    {  printf("n=? "); scanf("%d",&n); }
    while (n<1 || n>20);

    for ( red=1 ; red<=n ; red++ )
    {
        for ( broj=1 ; broj<=red ; broj++ )
            printf("%d", broj%10);

        printf("\n");

    }
}
```

Primer: **Program koji ispisuje prvih n redova sledeće piramide cifara:**

```
#include <stdio.h>
main()
{
   int n, red, broj;
   do
   {  printf("n="); scanf("%d",&n); }
   while (n<1 || n>20);
   for ( red=1 ; red<=n ; red++ )
   {
      for ( broj=1 ; broj<=n-red ; broj++ )
         printf(" ");
      for ( broj=red ; broj<=2*red-1 ; broj++ )
         printf("%d", broj%10);
      for ( broj=2*red-2 ; broj>=red ; broj-- )
         printf("%d", broj%10);
      printf("\n");
   }
}
```

```
          1
         232
        34543
       4567654
          .
          .
          .
  890123454321098
 90123456765432109
          .
          .
          .
```

Zadaci:

1. Napisati program koji koji učitava prirodan broj, a zatim ispisuje sve parne brojeve manje od njega. Zadatak rešiti korišćenjem a) for petlje; b) while petlje; c) do…while petlje.
2. Napisati program koji tabelarno ispisuje vrednosti funkcije $f(x)=2x+3$, za vrednosti argumenta $x \in [A,B]$, s korakom Δx. Zadatak rešiti korišćenjem a) for petlje; b) while petlje; c) do…while petlje.
3. Učitati *n* brojeva, a zatim ispisati koliko među njima ima pozitivnih, kao i njihovu aritmetičku sredinu.
4. Učitati n brojeva, a zatim ispisati najvećeg.
5. Učitati *n* brojeva, a zatim ispisati najmanjeg, kao i njegov redni broj.
6. Ispisati sve dvocifrene proste brojeve.
7. Ispisati prvih n prostih brojeva.
8. Ispisati prvih n savršenih brojeva.
9. Učitati broj pa proveriti da li je Armstrongov. Broj je Armstrongov ako je jednak zbiru kubova svojih cifara.
10. Učitati dva prirodna broja pa proveriti da li oni predstavljaju par prijateljskih brojeva. Za dva broja kažemo da su prijateljski ako je prvi jednak zbiru delilaca drugog, a drugi jednak zbiru delilaca prvog broja.

Ternarni uslovni operator

Još jedan način grananja u C-u je korišćenjem ternarnog uslovnog operatora koji ima sledeću sintaksu:

```
Izraz1 ? Izraz2 : Izraz3;
```

Vrednost izračunavanja operatora se dobija tako što se prvo izračuna vrednost izraza Izraz1 i ako je njegova vrednost tačna (odnosno u C-u veća od nule), kao rezultat operacije uzima se Izraz2, ako je vrednost netačna, rezultat je Izraz3.

Na primer, ako posmatramo sledeći niz naredbi

```
int a = 5; int c = (a>0) ? 10 : -10;
```

Nakon izvršavanja ovog niza naredbi vrednost promenljive c će biti 10, jer je vrednost izraza a>0 tačno i ternarni uslovni operator vraća vrednost prvog izraza iza upitnika, koji iznosi 10.

Na mestu izraza Izraz2 i Izraz3 mogu se navesti i naredbe, na primer

```
int a = -5; (a>0) ? printf("%d\n", a) : printf("%d\n", 0-a);
```

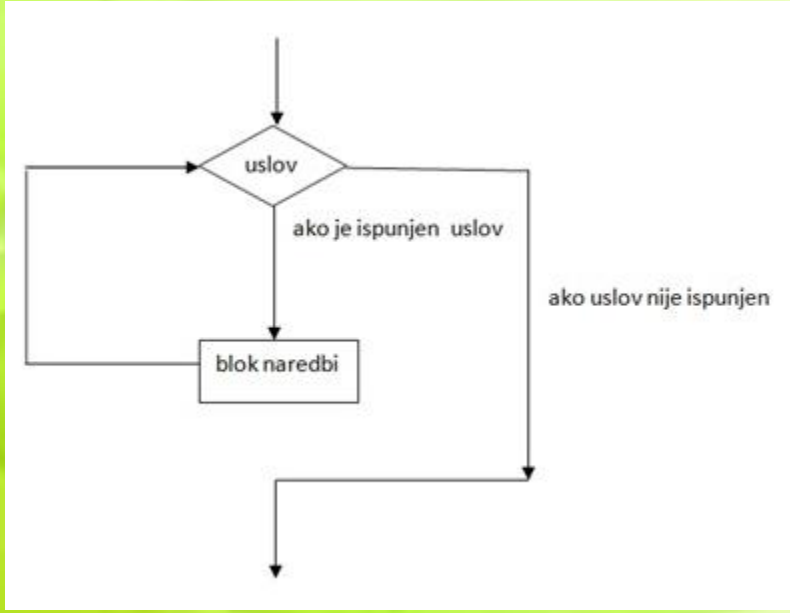
while naredba

Sintaksa naredbe while je:

while (uslov)
{ blok naredbi; }

Dijagram toka koji ilustruje izvršavanje while naredbe:

Prvo se proverava ispunjenje zadatog uslova, ukoliko je uslov ispunjen, izvršava se blok naredbi i ide se ponovo na proveru uslova. Ukoliko uslov nije ispunjen izlazi se iz while petlje i nastavlja sa izvršavanjem naredbi posle while naredbe. Ovde možemo zaključiti da se blok naredbi u while-u ne mora izvršiti ni jednom. Uslov while naredbe je logičkog tipa, odnosno u C-u celobrojnog tipa.



Naredba while

Omogućava realizaciju petlje sa izlazom na vrhu

Ciklus će se izvršavati sve dok je uslov ispunjen

(ne mora ni jedanput)

Opšti oblik

```
while (izraz)
   iskaz;
```

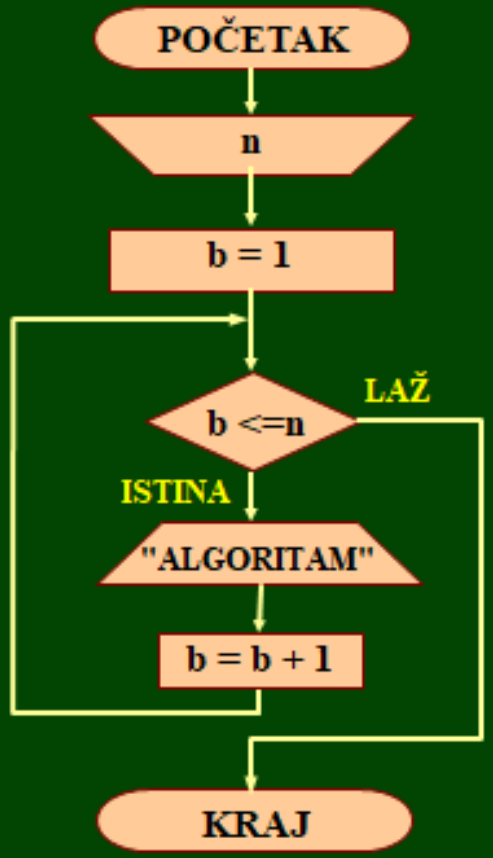
ili

```
while (izraz)
{
   iskaz1;
    ...
   iskazN;
}
```

Sve dok je izraz ISTINIT (!=0) izvršava se (složena) naredba

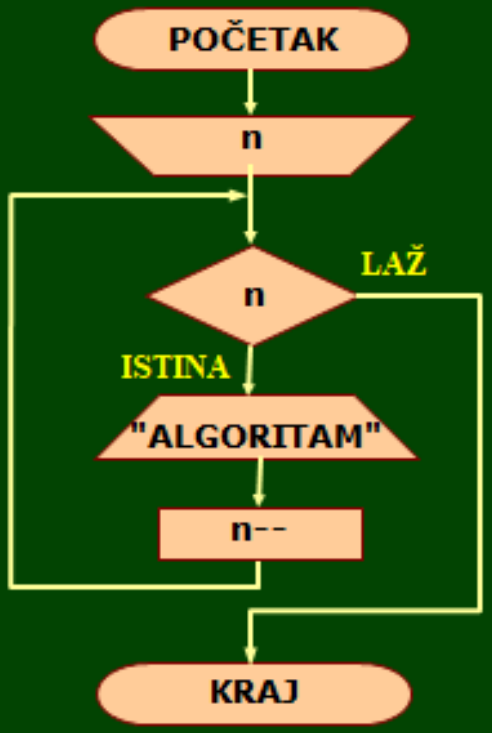
Kad izraz postane LAŽ (==0) prekida se izvršavanje petlje

Primer: **Program koji N puta ispisuje reč "ALGORITAM“.**




```
#include <stdio.h>
main()
{
    int n, b=1;
    printf("Unesite N: ");
    scanf("%d",&n);
    while ( b<=n )
    {
       printf("Algoritam\n");
       b++;
    }
}
```

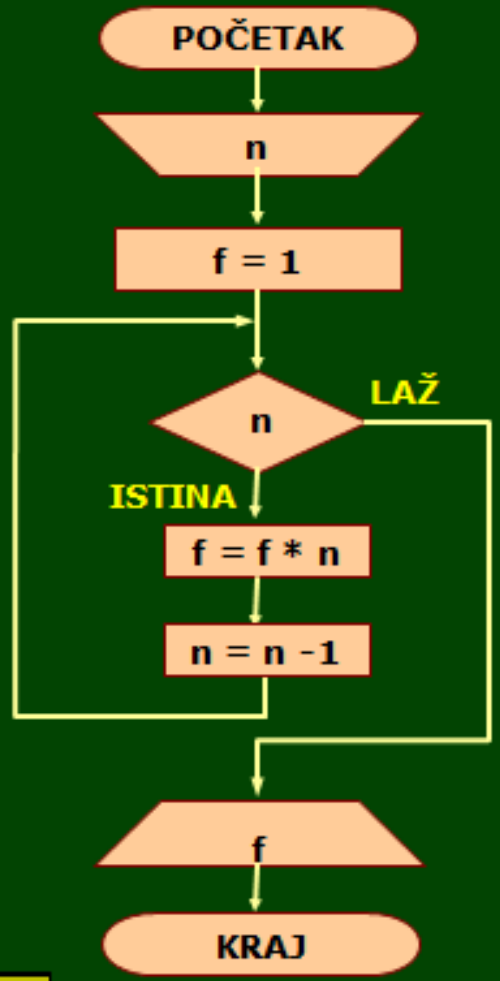
Primer: **Program koji N puta ispisuje reč "ALGORITAM".**
(bez korišćenja pomoćnog brojača)




```
#include <stdio.h>
main()
{
    int n;
    printf("Unesite N: "); scanf("%d",&n);
    while (n)
    {
        printf("ALGORITAM\n");
        n--;
    }
}
```

```
#include <stdio.h>
main()
{
    int n;
    printf("Unesite N: "); scanf("%d",&n);
    while (n--) printf("ALGORITAM\n");
}
```

Primer: **Program koji učitava *n*, a zatim izračunava i ispisuje *n*!**

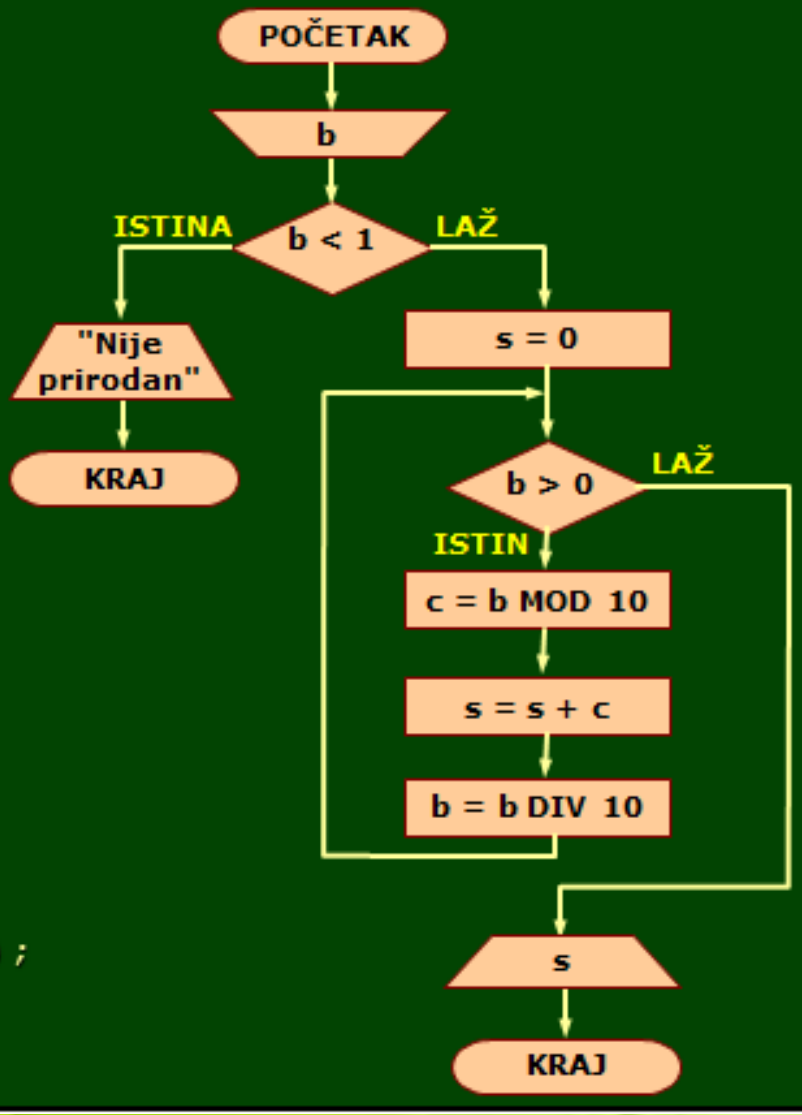



```
#include <stdio.h>
main()
{
    int n, f=1;
    printf("Unesite N: ");
    scanf("%d",&n);
    printf("%d!=",n);
    while (n)
        f *= n--;
    printf("%d",f);
}
```

Primer:

Program koji učitava prirodan broj, a zatim ispisuje zbir njegovih cifara.

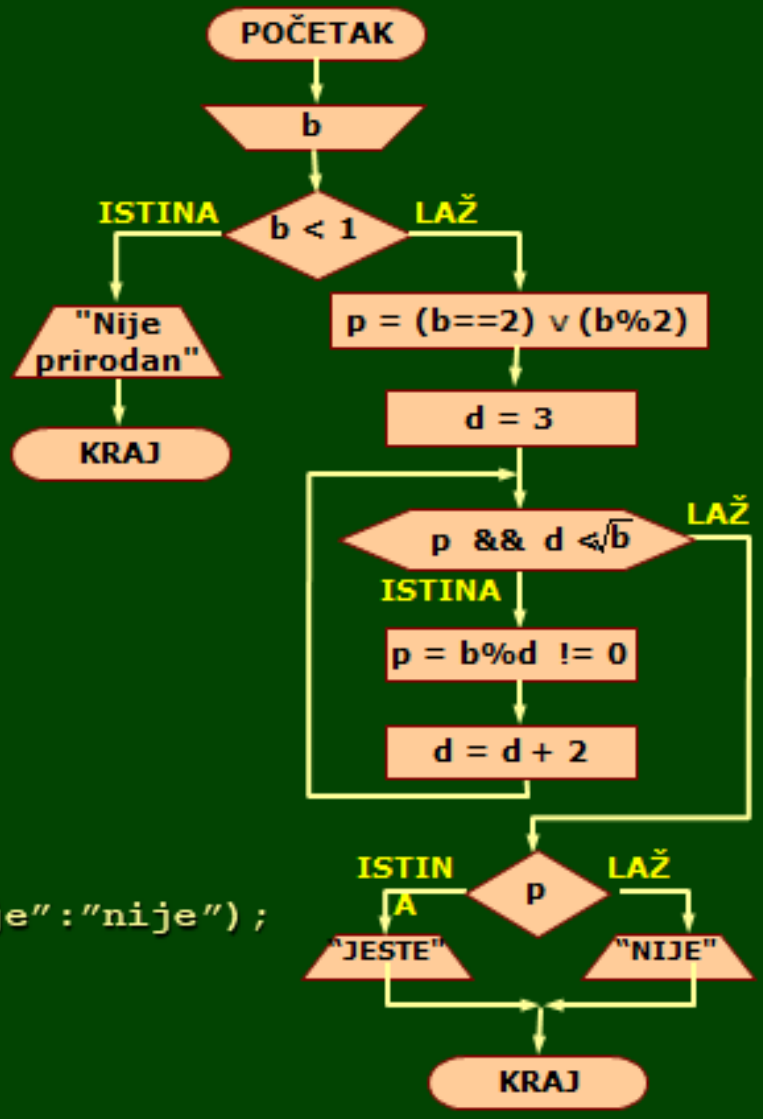
```
#include <stdio.h>
main()
{
    int b, s=0;
    printf("Unesite broj: ");
    scanf("%d",&b);
    if (b<1)
       printf("Nije prirodan!");
    else
    {
       while (b)
       {
          s += b % 10;
          b /= 10;
       }
       printf("Zbir cifara je:%d",s);
    }
}
```



Primer:

Program koji učitava prirodan broj, a zatim ispisuje da li je on prost ili nije.

```
#include <stdio.h>
#include <math.h>
main()
{
    int b, p, d=1;
    printf("Unesite broj: ");
    scanf("%d",&b);
    if (b<1)
       printf("Nije prirodan!");
    else
    {
       p = (b == 2) | (b % 2);
       while (p && (d+=2)<=sqrt(b))
          p = b % d;
       printf("%d %s prost", b, p ? "je":"nije");
    }
}
```



Primer:

Primer izračunavanja faktorijela za neki broj koji unosi korisnik. Faktorijel nekog broja se računa na sledeći način:

0! = 1 n! = n*(n-1)*(n-2)*...*1

Implementacija ovog programa korišćenjem while naredbe:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ int n;
printf("Unesite broj: ");
scanf("%i", &n);
int faktorijel = 1;
while(n>0)
    { faktorijel = faktorijel * n; --n; }
printf("Vrednost faktorijela je %d", faktorijel);
return 0; }
```

do..while naredba

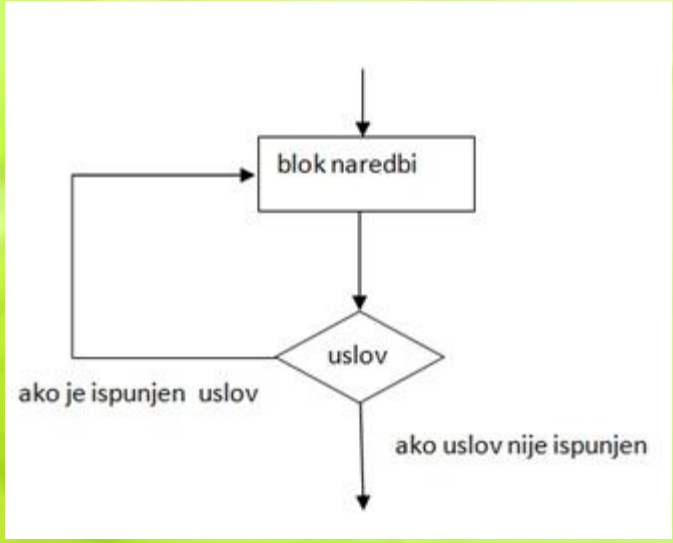
Sintaksa naredbe do..while je:

do

{ blok naredbi; }

while(uslov)

Dijagram toka koji ilustruje izvršavanje naredbe do...while:



Naredba se izvršava tako što se prvo izvrši blok naredbi, nakon čega se proverava ispunjenost uslova.

Ukoliko je ispunjen uslov, ponovo se izvršava blok naredbi, ukoliko uslov nije ispunjen izlazi se iz petlje i nastavlja se sa izvršavanjem naredbi koje su navedene posle do..while naredbe.

Osnovna razlika između naredbi while i do..while je što će se kod naredbe do..while blok naredbi sigurno izvršiti barem jednom.

Naredba do...while

Omogućava realizaciju petlje sa izlazom na dnu

Opšti oblik

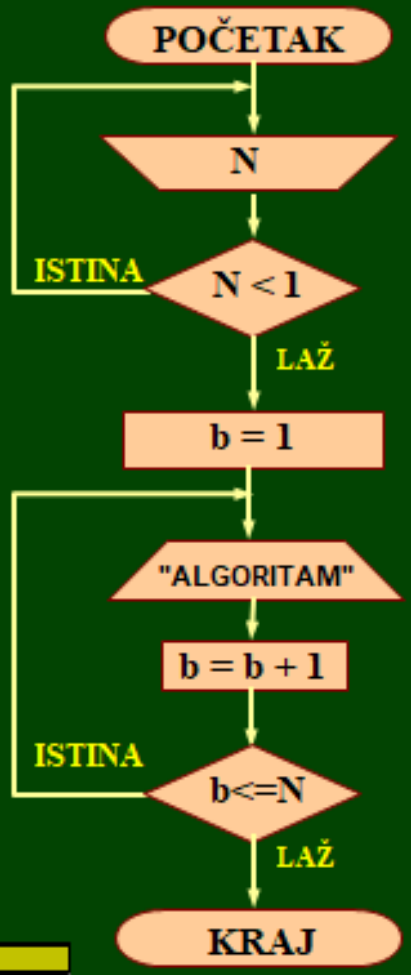
```
do
  iskaz;
while (izraz);
```

ili

```
do
{
   iskaz1;
    ...
   iskazN;
}
while (izraz);
```

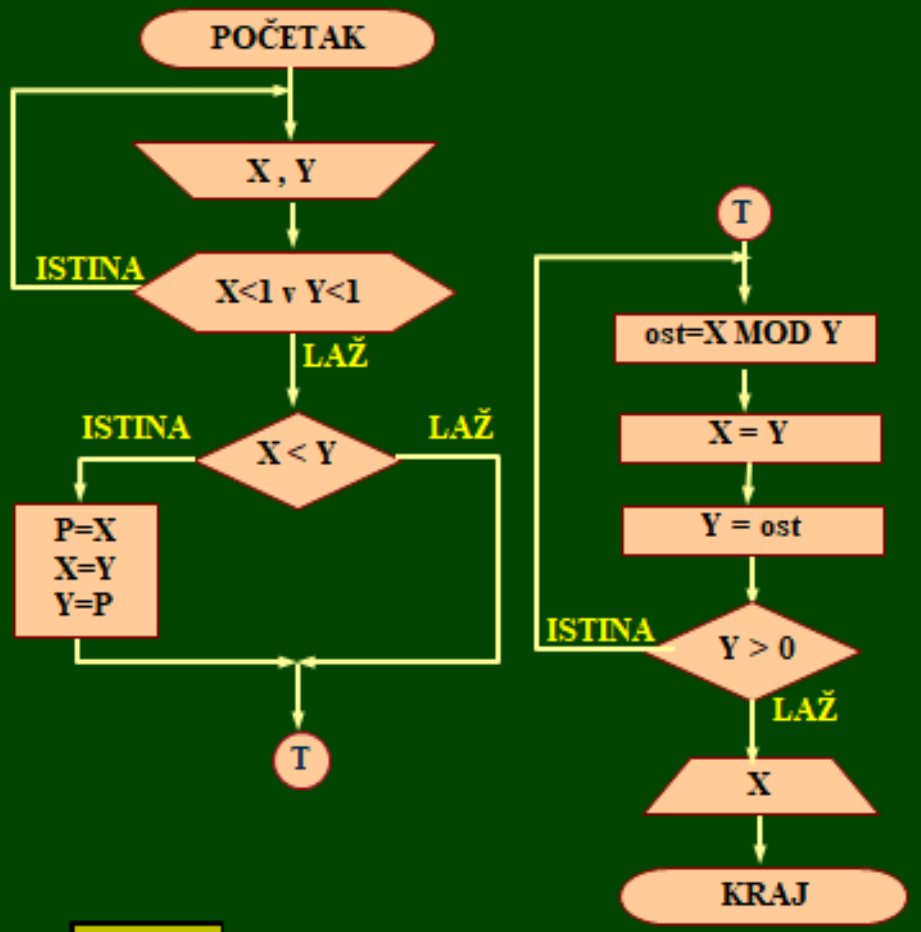
1. Izvršavaju se iskazi iza do
2. Računa se vrednost izraza
3. Ako je izraz ISTINIT (!=0) idi na korak 1
4. Ako je izraz LAŽ (==0) prekida se izvršavanje petlje i prelazi na sledeći iskaz iza while

Primer: **Program koji N puta ispisuje reč "ALGORITAM".**




```
#include <stdio.h>
main()
{
   int n, b=1;

   do
   {
      printf("Unesite N: ");
      scanf("%d",&n);
   }
   while (n<1);

   do
      printf("Algoritam\n");
   while ( ++b <= n );
}
```

Primer: **Euklidovim algoritmom odrediti NZD dva prirodna broja.**




```
#include <stdio.h>
main()
{
  int x,y,p,ost;
  do
  {
    printf("Unesite X i Y:");
    scanf("%d %d",&x,&y);
  }
  while (x<1 || y<1);
  if (x<y)
  {  p=x; x=y; y=p; }
  do
  {
     ost = x%y;
     x = y;
     y = ost;
  }
  while (y);
}
```

Primer: **Podeliti dva prirodna broja na proizvoljan broj decimala.**
(koristeći pravilo za ručno deljenje)

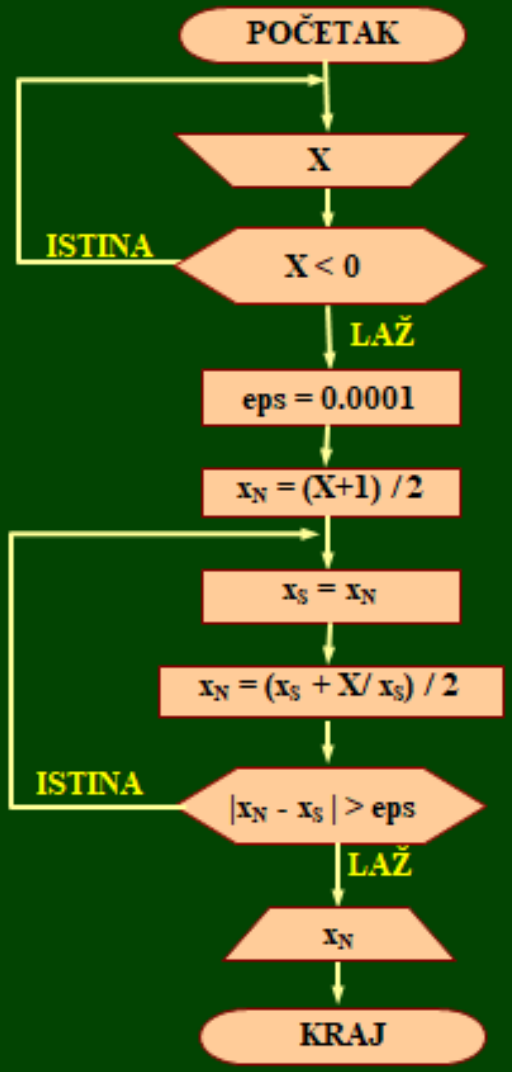
```
#include <stdio.h>
main()
{
  int x,y,bd;
  do
  {  printf("Unesite X i Y:"); scanf("%d %d",&x,&y); }
  while (x<1 || y<1);
  do
  {  printf("Broj decimala:"); scanf("%d",&bd); }
  while (bd<1);
  printf("%d:%d=%d.", x, y, x/y);
  do
  {
     x = x%y * 10;
     printf("%d", x/y);
  }
  while (--bd);
}
```

Primer:

Program koji računa drugi koren iz pozitivnog realnog broja *X* na četiri decimale koristeći Njutnovu iterativnu formulu.

$$x_0 = \frac{X+1}{2}, \quad x_{n+1} = \frac{x_n + \frac{X}{x_n}}{2}, \quad n = 0, 1, 2\ldots$$

```
#include <stdio.h>
#include <math.h>
#define EPS 0.0001
main()
{
    float x,xs,xn;
    do
    {  printf("X="); scanf("%f",&x);}
    while (x<0);
    xn = (x+1)/2;
    do
    {  xs = xn; xn = (xs + x/xs) / 2;}
    while (fabs(xn-xs)>EPS);
    printf("Korijen je: %8.4f",xn);
}
```



Primer:

Posmatrajmo sada zadatak da treba učitati broj koji je deljiv sa tri i dok korisnik ne unese takav broj ponovo ispisujemo isti zahtev. Ovakav zadatak je najbolje implementirati korišćenjem do..while naredbe:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ int broj;
    do
    { printf("unesite broj deljiv sa tri: \n");
    scanf("%i", &broj); }
            while(broj%3!=0);
            printf("%i / 3 = %i", broj, broj/3);
return 0; }
```

break i continue

Postoje dve ugrađene naredbe koje mogu da promene tok izvršavanja petlji i mogu se koristiti u kombinaciji sa bilo kojom od tri navedene naredbe ponavljanja.

Prva je naredba break, koja prekida proces izvršavanja petlje, a druga je naredba continue koja prekida izvršavanje jedne iteracije petlje.

Način na koji naredba break funkcioniše ako se pozove u petlji može se ilustovati na sledeći način:

Ovde se naredba break koristi u while petlji, ali na isti način se može pozvati i sa ostalim naredbama ponavljanja. Ako se toku izvršavanja petlje u određenoj grani pozove naredba break, na tom mestu se prekida izvršavanje petlje i prelazi se na sledeću naredbu posle petlje.

Naredba `break`

Dejstvo:

u naredbi `switch`:

- nasilni prekid izvršavanja naredbe switch
- prelazak na prvi iskaz iza naredbe switch

u petljama:

- nasilni prekid izvršavanja petlje
- prelazak na prvi iskaz iza petlje

u ostalim slučajevima:

- nema dejstva

Naredba return

Dejstvo: izlazak iz funkcije

- prekida izvršavanje date funkcije

  (ignorišu se svi iskazi iza return)

- ako se nalazimo u main funkciji – prekid programa

Omogućava da funkcija vrati neku vrednost

Opšti oblik

```
return (vrednost);
```

ili

```
return (izraz);
```

Naredba goto

Omogućava bezuslovni skok

Opšti oblik

```
goto labela;
```

Labela (oznaka) neke naredbe u istoj funkciji

Primjer

```
#include <stdio.h>
main()
{
   int broj;
   lab1: printf("Unesite prirodan broj: ");
   scanf("%d",&broj);
   if (broj<1)
   {  printf("Nije prirodan!\n"); goto kraj; }
   printf ("%d %s paran\n", broj, (broj%2)? "nije" : "je");
   goto lab1;
   kraj: printf("KRAJ...");
}
```

Bezuslovni skok na iskaz označen sa kraj

Bezuslovni skok na iskaz označen sa lab1

Primer:

Zadatak sa učitavanjem broja koji je deljiv sa 3 koji smo prethodno uradili korišćenjem do..while naredbe može se implementirati korišćenjem while petlje i naredbe break:

```
#include <stdio.h>
#include <stdlib.h>
 int main()
 { int broj;
     while(1)
{ printf("unesite broj deljiv sa tri: \n");
scanf("%i", &broj);
if(broj%3==0) break; }
printf("%i / 3 = %i", broj, broj/3);
return 0; }
```

Ovde smo izabrali da napravimo beskonačnu petlju (uslov za naredbu while je uvek tačan), ali čim dobijemo odgovarajući broj izlazimo iz petlje naredbom break i program nastavlja sa naredbom posle while-a, odnosno ispisuje rezultat deljenja unetog broja sa 3.

Primer:

Naredba continue se može iskoristiti u implementaciji programa koji sabira 10 brojeva koji se unose preko konzole, ali izostavlja negativne brojeve.

```
# include <stdio.h>
int main()
{ int i;
double broj, suma = 0.0;
for(i=1; i <= 10; ++i)
    { printf("Unesite broj %d: ",i);
    scanf("%lf",&broj); // ako korisnik unese negativan broj, preskacemo ga
    if(broj < 0.0)
    { continue; }
suma = suma+ broj; }
printf("Suma = %.2lf",suma);
    return 0; }
```

Ugnježdene naredbe ponavljanja

Naredbe ponavljanja se mogu kombinovati tako da se jedna petlja poziva unutar druge, može biti neograničen broj ovako ugnježdenih petlji, a mogu se i kombinovati različite vrste petlji, for, while ili do..while.

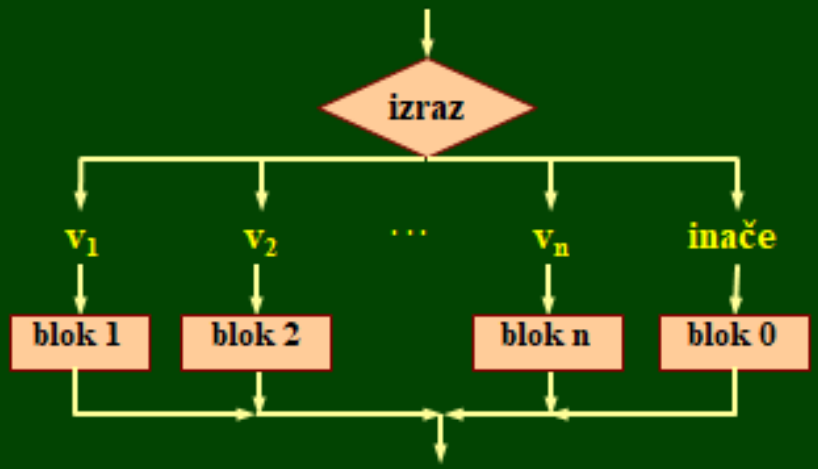
Ako bismo hteli da napravimo program koji zahteva da se unese 5 parnih brojeva preko konzole, ne dozvoljava da se unese sledeći broj dok se ne unese parni i na kraju ispisuje najveći od unetih 5 brojeva. Ovaj zadatak se može rešiti korišćenjem ugnježdene petlje.

```
#include <stdio.h>
 #include <stdlib.h>
 int main() { int i, max, br; max = 0;
for(i = 0; i<5; i++)
      { do
              { printf("Unesite %i. paran broj: \n", i+1);
                        scanf("%i", &br); }
      while(br%2!=0);
if(br>max) max = br; }
printf("Najveci uneti parni broj je %i\n", max);
return 0; }
```

Ovde imamo primer for petlje koja sadrži do..while petlju. For petlja će se izvršiti 5 puta za svaki traženi parni broj. U svakoj iteraciji for petlje izvrava se jedna ili više iteracija do..while petlje u kojoj tražimo da se unese paran broj i vrtimo se u petlji sve dok se to ne desi. Kada se učita paran broj, završava se do..while u jednoj iteraciji for petlje i prelazi se na sledeću iteraciju for petlje.

Naredba switch

Omogućava realizaciju selektivnog višeblokovskog grananja




```
switch (izraz)
{
   case v1:
      iskaz1;
       ...
      iskazN;
   case v2:
      iskaz1;
       ...
      iskazN;
   .
   .
   .
   case vN:
      iskaz1;
       ...
      iskazN;
   [default:
      iskaz0;]
}
```

Naredba switch

Ako bismo dobili zadatak da napišemo program da za uneti redni broj dana u nedelji ispišemo njegov naziv i ako bismo pokušali da ga implementiramo korišćenjem if-else naredbe, morali bismo da koristimo više else-if grana. Kod ovakvih slučajeva, kada imamo uslov koji podrazumava neke višestruke vrednosti, koristi se naredba switch. Naredba switch ima sledeću sintaksu:

```
switch(izraz)
{ case konstanta1 : blok naredbi 1
case konstanta2 : blok naredbi 2
...
case konstantan : blok naredbi n
default: blok naredbi }
```

Pri čemu izraz mora biti nekog celobrojnog tipa.

Naredba switch se izvršava tako što se izračuna vrednost izraza i pronalazi se case grana čija je vrednost konstante jednaka vrednosti izračunatog izraza.

Ukoliko se pronađe takva case grana izvršavaju se svi blokovi naredbi u svim case granama počevši od case grane čija konstanta se poklapa sa izrazom, pa do kraja switch naredbe. Ukoliko se vrednost izraza ne poklapa ni sa jednom case granom, izvršava se blok označen sa default, ukoliko postoji.

Primer

Program koji za učitanu numeričku ocenu ispisuje opisnu ocenu koristeći selektivno višeblokovsko grananje.

```
#include <stdio.h>
main()
{
   int ocena;
   printf("Ocena? "); scanf("%d",&ocena);
   switch (ocena)
   {
      case 5:
         printf("Odlican\n"); break;
      case 4:
         printf("Vrlo dobar\n"); break;
      case 3:
         printf("Dobar\n"); break;
      case 2:
         printf("Dovoljan\n"); break;
      case 1:
         printf("Nedovoljan\n"); break;
      default:
         printf("GRESKA\n");
   }
}
```

Primer: **Program koji simulira rad kalkulatora celobrojnih operacija**

```
#include <stdio.h>
main()
{
   int x,y,rez;
   char op;
   printf("Unesite izraz u obliku op1 operator op2: \n");
   scanf("%d %c %d",&x,&op,&y);
   switch (op)
   {
      case '+':
         rez=x+y; break;
      case '-':
         rez=x-y; break;
      case '*':
         rez=x*y; break;
      case '/':
         rez=x*y; break;
      default:
         printf("Ne poznajem tu operaciju!\n");
         return (0);
   }
   printf ("%d %c %d = %d", x,op,y,rez);
}
```

Nasilni izlaz iz programa

Naredba switch

selektorski izraz
ne može biti **float** ili **double**

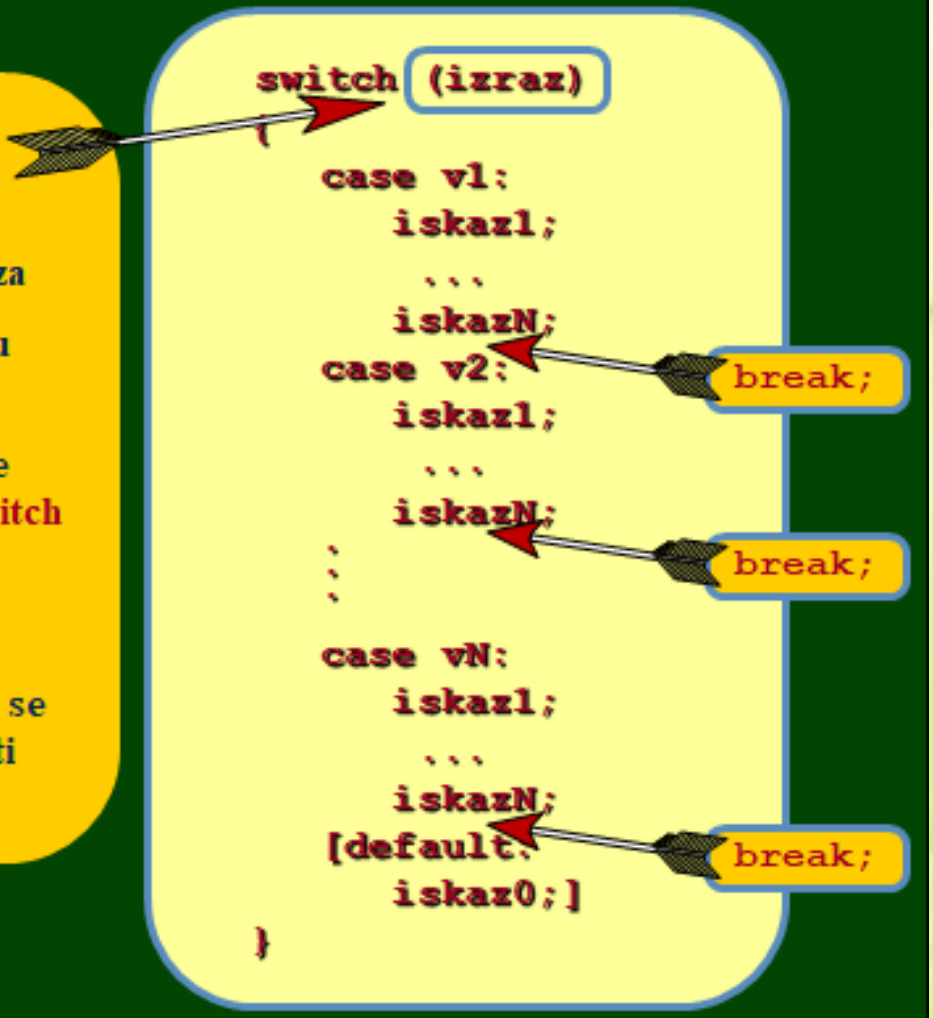
Na početku se izračunava vrednost izraza

Izračunata vrednost pronalazi se među konstantama v1 .. vN

Ako se vrednost pronađe, izvršavaju se **svi iskazi od pronađenog case do kraja switch**

Ako se vrednost ne pronađe, **prelazi se na iskaze iza default:**

Ako se želi izlazak iz naredbe switch kad se završi odgovarajući case, treba koristiti naredbu **break**



Primer:

Ispisivanje imena izabranog dana u nedelji:

```
#include <stdio.h>
int main()
{ int i; printf("Unesite broj dana u nedelji od 1 do 7: \n");
scanf("%d", &i);
     switch(i)
{ case 1: printf("Ponedeljak"); break;
case 2: printf("Utorak"); break;
case 3: printf("Sreda"); break;
case 4: printf("Cetvrtak"); break;
case 5: printf("Petak"); break;
case 6: printf("Subota"); break;
case 7: printf("Nedelja"); break;
default: printf("Pogresan broj"); }
return 0; }
```

Naredba exit ()

exit() završava egzekuciju programa izbacujući pri tome kod greške koji je tipa integer:

0 -> no error,
1 -> not found,
99 -> crash

Naredba exit() je vrlo primitivni način završavanja programa i nudi vrlo limitiranu šansu da se program izbori sa greškom. Neki programski jezici koriste izuzetke (exceptions) i njihove mehanizme indicirajući nastalu grešku bez potrebe da se program

```
if (!(buf = AllocMem (BufSize);))
{
printf("Nema dovoljno memorije");
exit(NO_MEM);
}
```

Rad sa karakterima (stringovima) getchar() i putchar()

Pored funkcija scanf i printf za učitavanje podataka sa konzole, odnosno standardnog ulaza i ispis podataka na konzolu odnosno standardni izlaz u C-u postoje i druge funkcije koje čitaju i ispisuju na konzolu.

Funkcija *getchar* čita jedan karakter sa konzole, nema ulaznih parametara i vraća tip int koji predstavlja ASCII kod karaktera.

int getchar(void)

Funkcija *putchar* ispisuje jedan karakter na konzolu, ima jedan ulazni parametar tipa int koji je ASCII kod karaketra i ima isti taj karaketer (ASCII kod) kao povratnu vrednost.

int putchar(int c)

Obe ove funkcije učitavaju i ispisuju po jedan karakter i da bi se pročitalo i ispisalo više karaktera pozivaju se u petlji i ovde će biti primenjeni za ilustraciju rada sa petljama u zadacima koji čitaju i obrađuju ulazni niz karaktera, na primer izbacuju, zamenjuju, filtriraju, dupliraju karaktere prema određenim pravilima, pronalaze neke zakonitosti ili nepravilnosti.

Stringovi

String je jednodimenzionalni znakovni niz

To je niz znakova koji završava nul-znakom (znak čiji je kod 0) `'\0'`

Deklaracija stringa:

```
char ime[duzina] = "string" ;
```

ili

```
char ime[duzina] = { znak, znak, … , znak } ;
```

Primer:

```
char grad1[] = "BANJA LUKA" ;
char grad2[] = { 'D','O','B','A','R','\0' };
```

Označavanje znakovnih i string konstanti:

```
'D'   - ovo je znak D
"D"   - ovo je string 'D','\0'
```

Manipulacija stringom

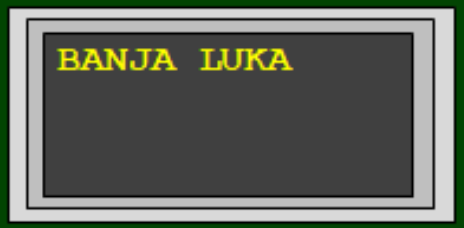
Ispisivanje stringa

ispisivanje pomoću funkcije

```
printf("%s", string);
```

Primer:

```
#include <stdio.h>
main()
{
   char grad[]= "BANJA LUKA";
   printf("%s\n", grad);
}
```

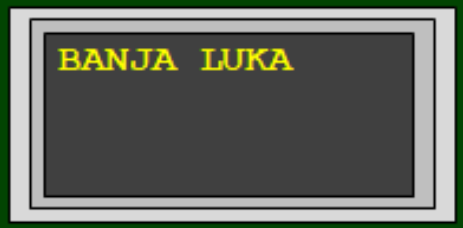


ispisivanje pomoću funkcije

```
puts(string);
```

Primer:

```
#include <stdio.h>
main()
{
   char grad[]= "BANJA LUKA";
   puts(grad);
}
```



Manipulacija stringom

Učitavanje stringa

učitavanje pomoću funkcije

```
scanf("%s", string);
```

Primer:

```
#include <stdio.h>
main()
{
   char tekst[50];
   printf("Unesi recenicu:\n");
   scanf("%s", tekst);
   printf("%s", tekst);
}
```

```
Unesi recenicu:
BANJA LUKA
BANJA
```

učitavanje pomoću funkcije

```
gets(string);
```

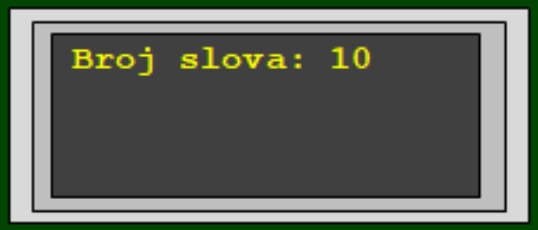
Primer:

```
#include <stdio.h>
main()
{
   char tekst[50];
   printf("Unesi recenicu:\n");
   gets(tekst);
   printf("%s", tekst);
}
```

```
Unesi recenicu:
BANJA LUKA
BANJA LUKA
```

Primer: **Program koji učitava string, a zatim ispisuje njegovu dužinu.**

```
#include <stdio.h>

int strlen ( char *s )
      {
         int i;
         for (i=0; *s!='\0'; i++);
         return(i);
      }

main()
{
   char grad[]="BANJA LUKA";
   printf("Broj slova: %d", strlen(grad) );
}
```



Primer: **Program koji učitava string i pravi kopiju tog stringa.**

```
#include <stdio.h>

void strcopy ( char *s1, char *s2 )
     {
        while ( *s2++ = *s1++ );
     }

main()
{
   char original[100]="BANJA LUKA";
   char kopija[100];
   strcopy(original, kopija);
   printf("%s\n", original);
   printf("%s\n", kopija);
}
```

```
BANJA LUKA
BANJA LUKA
```

Primer:

Ako bismo hteli da napišemo program koji na osnovu ulaznog niza karaktera koji se unosi preko konzole, a koji se završava oznakom za novi red, pravi izlazni niz karaktera koji sadrži samo brojeve iz ulaznog niza (iz ulaznog teksta izbaciti sve osim brojeva). Jedno rešenje ovog zadatka bi izgledalo ovako:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ char ch;
      while((ch = getchar()) != '\n')
      { if(ch>='0' && ch<='9') putchar(ch); }
return 0; }
```

Stringu možemo dodeliti vrednost na dva načina:

```
char str[7] = {'Z', 'd', 'r', 'a', 'v', 'o', '\0'}; // poslednji element mora biti null karakter
char str[7] = "Zdravo"; // string je niz karaktera u dvostrukim navodnicima,
```

kod ovog slučaja kompajler će napraviti niz od 7 karaktera, od kojih će prvih 6 biti popunjeno vidljivim karakterima koji su dodeljeni stringu, a na poslednje mesto će se postaviti null karakter. Ovde možemo primetiti da je veličina niza jednaka duzini stringa plus jedan.

Primer:

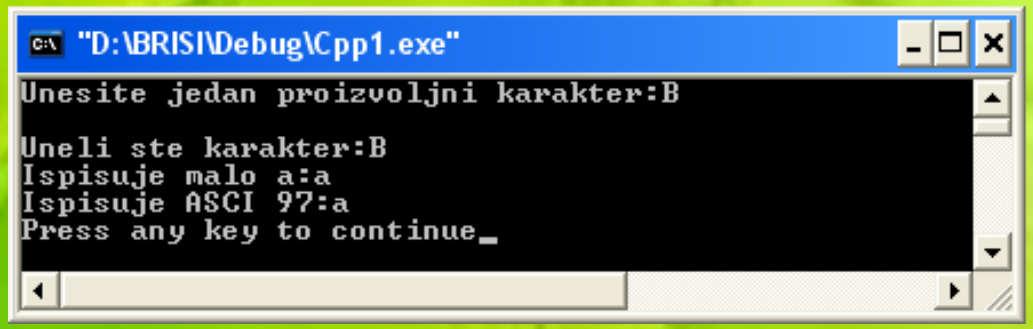
Potrebno je napisati program korićenjem while petlje koji iz ulaznog niza karaktera izbacuje sva dupla slova, odnosno ako dva ista slova stoje jedan pored drugog, ispisuje samo jedno, na primer za aabbdd, treba da ispiše adb. Jedno rešenje ovog zadatka je:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ char karakter;
char prethodniKarakter;
prethodniKarakter = getchar();
putchar(prethodniKarakter);
      while((karakter = getchar()) != '\n')
      { if(prethodniKarakter!=karakter) putchar(karakter);
      prethodniKarakter = karakter; }
return 0; }
```

U rešenju ovog zadatka čitamo ulazni niz karaktera i poredimo dva susedna slova, zbog toga imamo dve promenljive tipa char koje označavaju prethodni karakter i trenutni karakter koji se posmatrama. Prvi karakter ćemo ispisati nakon učitavanja (jer on sigurno nije jednak sa prethodnim, jer ni nema prethodnog), to je pre izvršavanja while naredbe i to će nam biti prvi prethodni karakter. U while naredbi proveravamo da li je trenutni karakter različit od prethodnog i samo ako jeste ispisujemo ga. Uslov za izlazak iz while petlje je učitavanje karaktera za kraj reda.

Program čita jedan karakter i ispisuje ga.

```
#include <stdio.h>
void main()
{
int c; /* Karakter - obratiti paznju na int */
printf("Unesite jedan proizvoljni karakter:");
c = getchar(); /* cita karakter sa standardnog ulaza */
printf("\nUneli ste karakter:");
putchar(c); /* pise karakter zapamćen u promenljivoj c na standardni izlaz */
putchar('\n'); /* prelazak u novi red */
printf("Ispisuje malo a:");
putchar('a'); /* ispisuje malo a */
printf("\nIspisuje ASCI 97:");
putchar(97); /* ekvivalentno prethodnom */
putchar('\n');/* prelazak u novi red */
}
```



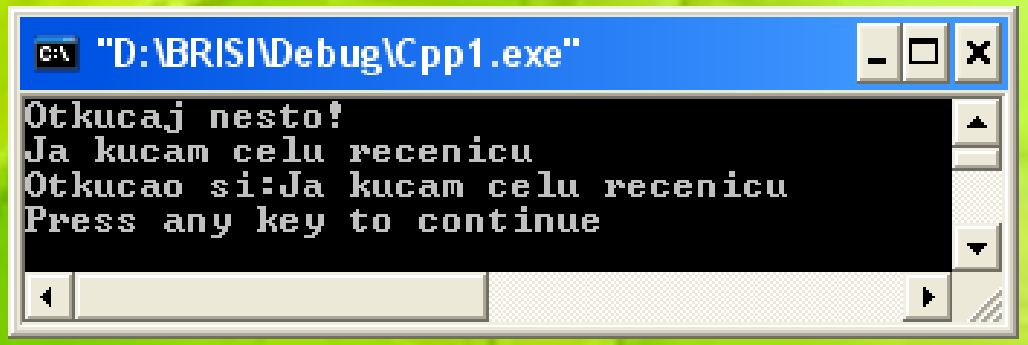
Primer funkcija puts() i gets()

Pošto je funkcionalnost učitavanja stringa potrebna u mnogim programima, u programski jezik C kao deo biblioteke stdio ugrađene su funkcije za čitanje i ispis celog stringa, to su funkcije gets(str) - učitava ceo string str sa istemskog ulaza, puts(str) - ispisuje string str na sistemski izlaz.

```
//Unos i štampanje karaktera

#include<stdio.h>
void main( ) {
        char c[30];
       printf("Unesite niz proizvoljnih karaktera:");
        gets(c);
       printf("\n");//Prelazak u novi red
       printf("Uneli ste karaktere:");
        puts(c);//Stampanje unetih karaktera
       printf("\n");//Prelazak u novi red
```

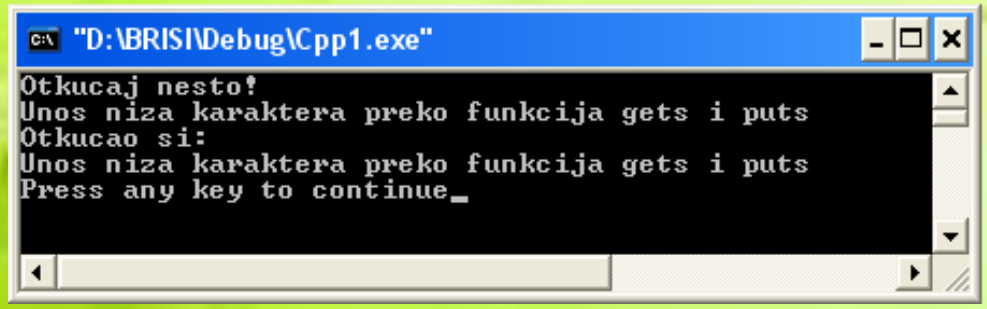
Program čita ceo izraz i ispisuje ga.

```
#include <stdio.h>
int main()
{  char my_string[50];
printf("Otkucaj nesto!\n");
    gets(my_string);
printf("Otkucao si:%s\n", my_string);
return 0;    }
```



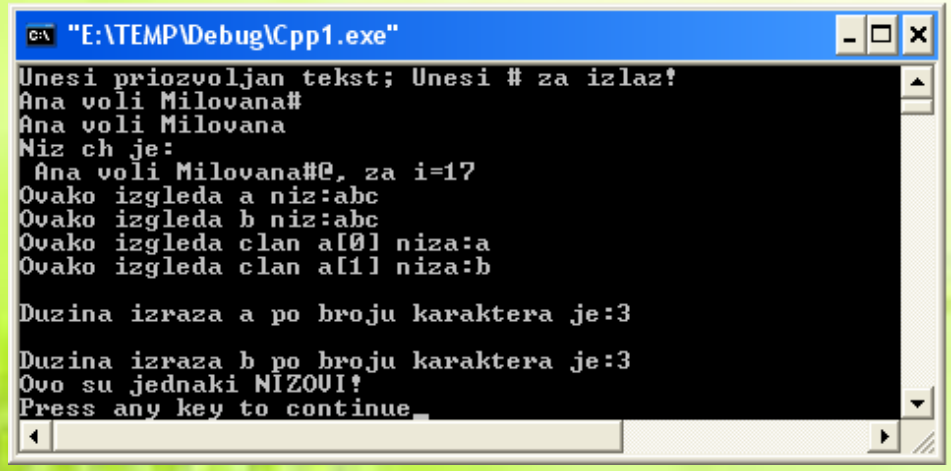
Program čita ceo izraz i ispisuje ga pomoću funkcija gets i puts.

```
#include <stdio.h>

 int main()
 {
  char my_string[500];
  printf("Otkucaj nesto!\n");
     gets(my_string);
  printf("Otkucao si:\n");
     puts(my_string);
 return 0;
 }
```



PROGRAM za unos i štampanje karaktera koji testira duzinu i jednakost dva stringa

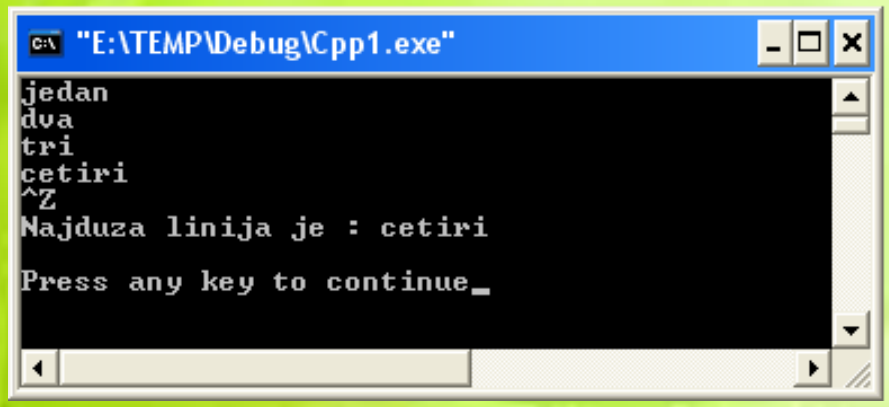
```
#include <stdio.h>
#include <string.h>
void main()
{
char a[3], b[3], ch[4];
int i=0;
printf("Unesi priozvoljan tekst; Unesi # za izlaz!\n");
ch[i]=getchar();
while(ch[i]!='#')//izlazak iz petlje kada je uneto #
{//ispisuje trenutno uneti karakter direktno na izlaz
putchar(ch[i]);i=i+1; ch[i]=getchar();}
printf("Niz ch je: \n %s, za i=%d\n",ch,i) ;
strcpy(a, "abc");//kopiranje niza karaktera u vektor a
strcpy(b, "abc");//kopiranje niza karaktera u vektor b
                printf("Ovako izgleda a niz:%s\n",a) ;
                printf("Ovako izgleda b niz:%s\n",b) ;
                printf("Ovako izgleda clan a[0] niza:%c\n", a[0]);
                printf("Ovako izgleda clan a[1] niza:%c\n", a[1]);
printf("\nDuzina izraza a po broju karaktera je:%d\n",strlen(a)) ;//izracunavanje duzine niza
printf("\nDuzina izraza b po broju karaktera je:%d\n",strlen(b)) ;//izracunavanje duzine niza
if (strcmp(a,b) == 0) //komparacija-uporedjivanje dva niza
{ printf("Ovo su jednaki NIZOVI!\n") ;}
 else
printf("Ovo nisu jednaki NIZOVI!\n");
}
```



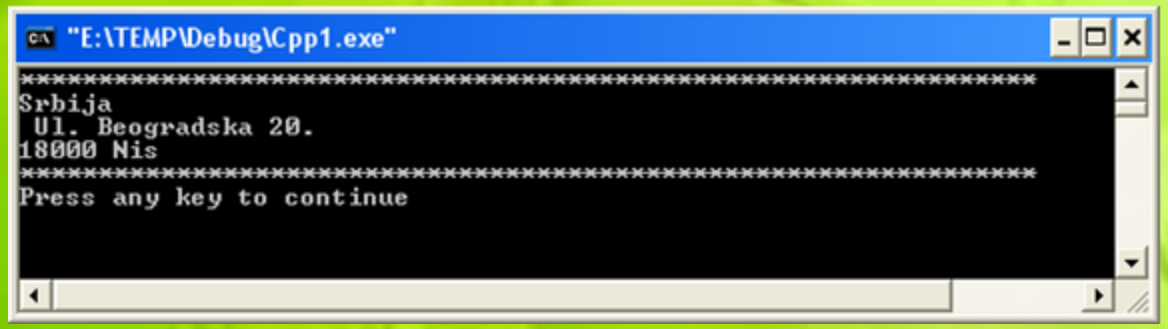
PROGRAM koji pronalazi najduzu liniju sa učitanu sa ulaza. Nije poznat ukupan broj linija, ali svaka linija nema vise od 80 karaktera (izlaz CTRL/Z).

```
#include <stdio.h>
#include <string.h>
#define MAX_DUZINA_LINIJE 81
main()
{
char linija[MAX_DUZINA_LINIJE];
char najduza_linija[MAX_DUZINA_LINIJE];
int duzina_linije = 0;
int duzina_najduze_linije = 0;
najduza_linija[0] = '\0';
while( fgets(linija, MAX_DUZINA_LINIJE, stdin) != NULL)
{
duzina_linije=strlen(linija);
if (duzina_linije>duzina_najduze_linije)
{strcpy(najduza_linija, linija);
duzina_najduze_linije = duzina_linije;}
}
printf("Najduza linija je : %s\n",najduza_linija);
}
```

Dužina linije plus 1 za null karakter

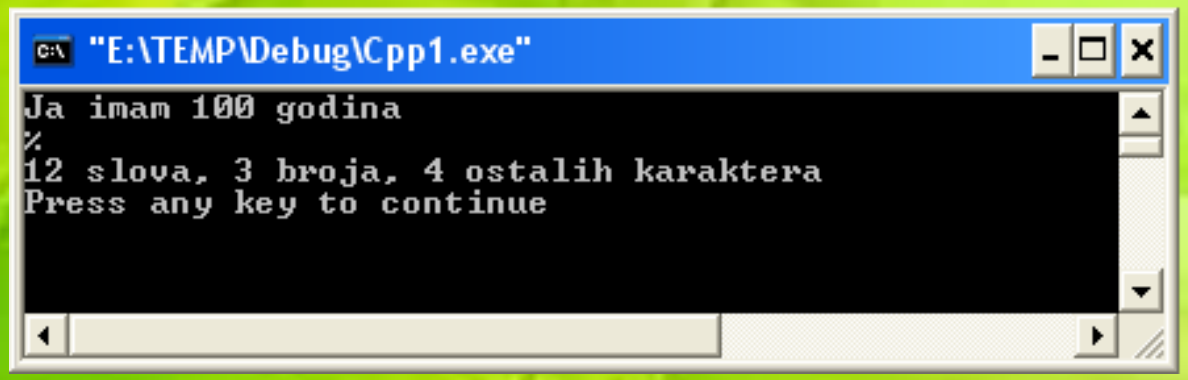


```
//PROGRAM  za formatiranje izlaza
#include <stdio.h>
#define IME "Srbija"
#define ADRESA "Ul. Beogradska 20."
#define MESTO "18000 Nis"
#define LIMIT 65
void zvezde(void);/*deklaracija funkcije bez argumenata*/

void main()
{ zvezde(); /*poziv korisnicke funkcije */
printf("%s \n ",IME) ; /*poziv funkcije iz standardne biblioteke*/
printf("%s\n", ADRESA);
printf("%s\n", MESTO) ;
zvezde ( ) ;} /*poziv korisnicke funkcije */

void zvezde() /*funkcija nema argumenata*/
{ int brojac;
for(brojac=1;brojac<=LIMIT;brojac++)
putchar ( '*' );
putchar ( '\n' ) ; }
```



Program broji karaktere, brojeve i praznine u tekstu. Završava se unosom %.

```
#include <stdio.h>
        void main( ) {
         int let, dig, other, c;
         let = dig = other = 0;
while( (c=getchar( )) != '%' )
   if( ('A'<=c && c<='Z') || ('a'<=c &&  c<='z') )
                ++let;
          else if( '0'<=c && c<='9' ) ++dig;
                else  ++other;
printf("%d slova, %d broja, %d ostalih karaktera\n", let, dig,
other); }
```



Ugrađene funkcije za rad sa stringovima

Funkcija	Značenje
strcpy(s1, s2)	kopira string s2 u string s1
strcat(s1, s2)	konkatenacija stringova, string s2 se dodaje na kraj stringa s1
strlen(s1)	vraća dužinu stringa s1, vraća se broj karaktera u stringu bez null karaktera
strcmp(s1, s2)	poređenje stringova, vraća 0 ako su s1 i s2 jednaki, manje od 0 ako je s1<s2 i veće od 0 ako je s1>s2
strchr(s1, ch)	vraća pokazivač na prvo pojavljivanje karaktera ch u string s1
strstr(s1, s2)	vraća pokazivač na prvo pojavljivanje stringa s2 u stringu s1
strlwr(s1)	sva slova u stringu s1 pretvara u mala slova
strupr(s1)	sva slova u stringu s1 pretvara u velika slova

FUNKCIJE

Postoje mnogo razloga za korišćenje funkcija:

a.) Deo koda se može puno puta koristiti u različitim delovima programa.

b.) Program se deli na separatne blokove. Svaki blok ima za zadatak da odradi neki od specijalnih zadataka. Razumevanje i dizajniranje programa je tada jednostavnije u svakom slučaju.

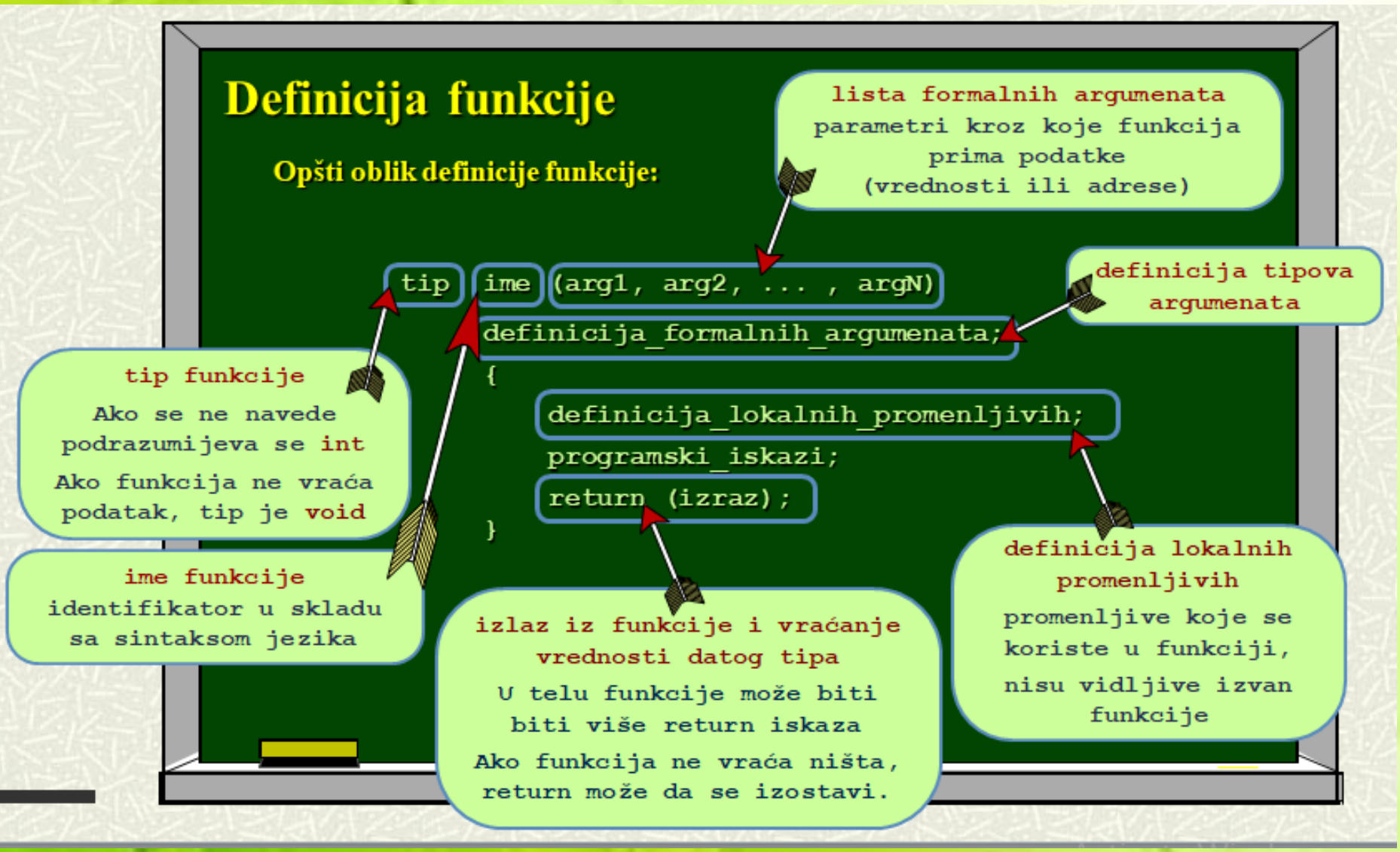
c.) Deo koda se može izvršavati sa različitim brojem inicijalnih parametara. Ovi parametri se pridružuju funkciji svojim argumentima.

Veliki računski zadaci mogu se razbiti u manje delove i time se omogućava programerima da iskoriste ono što su neki drugi već uradili, umesto da počinju sve od početka. Odgovarajuće funkcije skrivaju detalje postupka od delova programa i time čine ceo program jasnijim i jednostavnijim za menjanje.

Za sada funkcije će se pisati u nastavku glavnog programa !!!

Definicija funkcije

Opšti oblik definicije funkcije:



Definicija funkcije

Alternativni oblik definicije funkcije:

```
tip   ime ( tip1 arg1, tip2 arg2, ... , tipN argN )
      {
          definicija_lokalnih_promenljivih;
          programski_iskazi;
          return (izraz);
      }
```

definicija tipova argumenata sadržana je u listi argumenata

Poziv funkcije

Opšti oblik poziva funkcije:

```
ime ( sarg1, sarg2, ... , sargN )
```

lista stvarnih argumenata
parametri koji se šalju u funkciju (vrednosti ili adrese)
Zovu se stvarni zato što su to stvarni, realni podaci

Deklaracija funkcije (prototip)

Prototip ili deklaracija funkcije predstavlja njen "opis za spoljašnji svet"

Iz prototipa se vidi kako se komunicira sa datom funkcijom, jer sadrži:

- tip funkcije,
- ime funkcije,
- tipove (a može i imena) argumenata.

Opšti oblik prototipa:

```
tip  ime ( tip1, tip2, ... , tipN );
```

Primer:

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

```
int zbir( int, int );
```

```
void poruka ()
    {  cout<<"Zdravo!"<<endl;
}
```

```
void poruka ();
```

Struktura C programa

Uobičajena je sledeća struktura:

```
pretprocesorske direktive
prototip funkcija
main
definicija funkcija
```

Alternativno je moguće i:

```
pretprocesorske direktive
definicija funkcija
main
```

Pogodno kad imamo malen broj funkcija

Ako funkcija zove neku drugu funkciju, funkcija koja se poziva mora biti prethodno definisana

Primer:

```
#include <stdio.h>
void poruka ();

main()
{
    poruka();
}

void poruka ()
{
    printf("Zdravo!");
}
```

Prenos parametara u funkciju putem VREDNOSTI

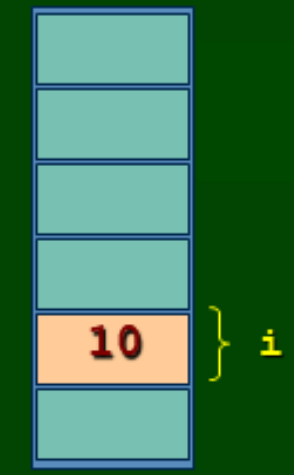
U prethodnim primerima korišćen je prenos parametara putem VREDNOSTI:

1. prilikom poziva funkcije šalje se vrednost
   - kao stvarni argument može da se navede:
     `ime promenljive, neka konstanta ili izraz`
   - izračunava se vrednost izraza i ta vrednost šalje u funkciju
2. funkcija prihvata vrednosti u formalne argumente
   - formalni i stvarni argumenti moraju da se slažu u:
     `broju, redosledu i tipu`
   - formalni argumenti (promenljive) automatski nastaju prilikom ulaska u funkciju i preuzimaju prosleđene vrednosti
3. po izlasku iz funkcije formalni argumenti automatski nestaju
   - automatski nastaju i nestaju pa se nazivaju automatske promenljive

U funkciji se kreira slika stvarnih argumenata i koriste te slike, a ne stvarne promenljive iz glavnog programa, zato nakon izlaska iz funkcije promenljive u glavnom programu ostaju nepromenjene!!!

Primer: **Ilustracija prenosa parametara putem vrednosti.**

```
#include <stdio.h>
void f (int);
main()
{
  int i=10;
  printf("main: %d\n",i);
  f(i);
  printf("main: %d\n",i);
}
void f (int k)
{
  printf("f: %d\n",k);
  k=20;
  printf("f: %d\n",k);
}
```




```
main: 10
f: 10
f: 20
main: 10
```

VEKTOR (polje) kao argument funkcije

Prenos vektora (polja) vrši se putem adrese

1. prilikom poziva funkcije šalje se adresa niza (vektora)
   - kao stvarni argument navodi se ime niza
     (ime niza je početna adresa niza)
2. funkcija prihvata adresu niza
   - formalni argument u definicji funkcije mora da sadrži [ ]
     broj elemenata niza ne mora da se navodi
   - ne stvara se kopija niza nego se manipuliše stvarnim podacima

U funkciji se manipuliše stvarnim podacima,
a ne kopijom niza. Zato funkcija može da promeni niz,
pa po povratku iz funkcije nemamo originalni nego
modifikovani niz !!!

Primjer: **Program koji pronalazi najveći element u nizu.**

```
#include <stdio.h>
int max ( int niz[], int n);
main()
{
  int a[] = { 2,15,5,0,8 };
  int n = 5;
  printf("Najveci: %d\n", max( a,n ) );
}
int max ( int niz[], int n )
{
  int i, m=niz[0];
  for (i=1; i<n; i++)
     if (niz[i]>m) m=a[i];
  return (m);
}
```

```
Najveci: 15
```

Poziv funkcije
šalje se adresa niza

Formalni argument
deklaracija niza bez dimenzije
moglo je i int niz[5]

Primjer: **Program koji sortira i ispisuje niz korišćenjem funkcija.**

```
#include <stdio.h>
void sort ( int niz[], int n);
void pisi ( int niz[], int n);
main()
{
   int a[] = { 2,15,5,0,8 };
   printf("Prije:"); pisi( a,n );
   sort( a,n );
   printf("Poslije:"); pisi( a,n );
}
void pisi ( int niz[], int n )
{
   int i;
   for (i=0; i<n; i++)
     printf(" %d", niz[i]);
   printf("\n");
}
```

```
void sort ( int niz[], int n )
{
  int i,j,rb,pom;
  for (i=0; i<n-1; i++)
  {
    for (rb=i, j=i+1; j<n; j++)
      if (niz[j]<niz[rb]) rb=j;
    if (rb!=i)
    {
       pom=niz[i];
       niz[i]=niz[rb];
       niz[rb]=pom;
    }
  }
}
```

```
Prije: 2 15 5 0 8
Poslije: 0 2 5 8 15
```

VIŠEDIMENZIONALNO polje kao argument funkcije

Prenos višedimenzionalnog polja vrši se putem adrese

U deklaraciji formalnih parametara:

- prva dimenzija ne mora da se navede (može samo [ ])
- ostale dimenzije moraju da se navedu

Primer:

```
int funkcija (int a[][10][10], int p, int q)
```

ili

```
int funkcija (int a[10][10][10], int p, int q)
```

U funkciji se manipuliše stvarnim podacima, a ne kopijom polja. Zato po povratku iz funkcije nemamo originalno nego modifikovano polje !!!

Primer: **Program koji transponuje kvadratnu matricu.**

```
#include <stdio.h>
void tran ( int m[10][10], int n);
void pisi ( int m[10][10], int n);
main()
{
  int a[10][10] = { {1,2},{3,4} };
  printf("Prije:\n"); pisi(a,n);
  tran( a,n );
  printf("Poslije:\n"); pisi(a,n);
}
void pisi (int m[10][10], int n)
{
   int i,j;
   for (i=0; i<n; i++)
   {
     for (j=0; j<n; j++)
       printf(" %4d", m[i][j]);
     printf("\n");
   }
}
```

```
void tran (int m[10][10], int n)
{
  int i,j,pom;
  for (i=0; i<n; i++)
    for (j=i+1; j<n; j++)
    {
       pom=m[i][j];
       m[i][j]=m[j][i];
       m[j][i]=pom;
    }
  }
}
```

```
Prije:
    1    2
    3    4
Poslije:
    1    3
    2    4
```

Rekurzivne funkcije (REKURZIJE)

Rekurzivna funkcija je funkcija koja sama sebe poziva!!!

Primer:

faktorijel: $n! = n * (n-1)!, n>0; \quad 0!=1$

stepenovanje: $x^n = x * x^{n-1}, n>0; \quad x^0=1$

Fibonačijev niz: $f_n = f_{n-1} * f_{n-2}, n>1; \quad f_0=0, f_1=1$

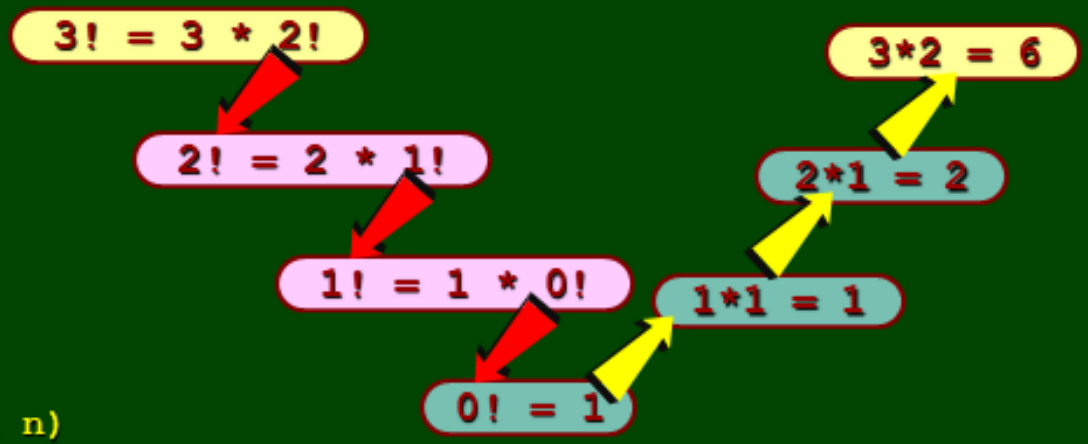
Opšte karakteristike rekurzivnih (rekurentnih) rešenja:

- problem se svodi na rešavanje istog problema, ali jednostavnijeg
- funkcija poziva samu sebe sve dok se problem ne pojednostavi do trivijalnog
- rekurzivna rešenja su manje efikasna (veći zahtev za memorijom)
- treba ih izbjegavati, ali postoje problemi koji su po prirodi rekurzivni i koje je gotovo nemoguće rešiti bez rekurzije

Primer: **Rekurzivno izračunavanje faktorijela.**

Matematski opis rekurzije:

$$n! = n * (n-1)!,\ n>0;\quad 0!=1$$




```
int fakt (int n)
{
  if (n == 0)
    return (1);
  else
    return (n * fakt(n-1));
}
```

```
int fakt (int n)
{
  return (n==0 ? 1 : n*fakt(n-1));
}
```

Primer: **Rekurzivna funkcija za konverziju dekadskog u binarni brojni sistem.**

$19_{10} = ?_2$

19 : 2	
9	1
4	1
2	0
1	0
0	1

$19_{10} = 10011_2$

```
19
cif=1
b=9
kon(9) => cif=1
          b=4
          kon(4) => cif=0
                    b=2
                    kon(2) => cif=0
                              b=1
                              kon(1) => cif=1
                                        b=0
                                        "1"
                                  "0"   <=
                             "0"   <=
                        "1"   <=
                   "1"   <=
```

```
void kon (int b)
{
     int cif;
     cif = b % 2;
     b /= 2;
     if (b>0) kon (b);
     printf("%d",cif);
}
```

Podrazumevane vrednosti argumenata u funkciji

U definiciji funkcije mogu da se navedu podrazumevane vrednosti za formalne argumente.

Podrazumevane vrednosti formalnih argumenata koriste se kad u pozivu funkcije nedostaju stvarni argumenti

Ako se za neki argument podrazumeva vrednost, onda mora da se podrazumeva i za sve preostale argumente deklarisane iza njega

Prilikom poziva funkcije mogu da se izostave samo posljednji argumenti (nema preskakanja)

Dozvoljeno je da se svi argumenti podrazumevaju

Format za definisanje funkcija

povratna vrednost-type ime-funkcije(lista-parametara)
{ deklaracije i promenljive }

povratna vrednost-type : tip rezultata (default int)

ime-funkcije: ispravan identifikator

void – označava da funkcija ne vraća vrednost

lista-parametara: lista parametara razdvojenih zarezima, deklariše parametre

deklaracije i promenljive : telo funkcije (blok)

Promenljive mogu biti definisane unutar bloka (ugnježdavanje). Funkcije ne mogu biti definisane unutar drugih funkcija.

Povratak iz funkcije:

Ako ne vraća vrednost, povratak iz funkcije u glavni program vrši se naredbom return(0);
Ако funkcija vraća vrednost naredba je return izraz;
Tip izraza se pretvara u tip rezultata ako je to neophodno.
Takođe, iza return ne mora uopšte da se navede neki izraz – u tom slučaju se nijedna vrednost ne vraća pozivaocu.
Tip mora biti eksplicitno naveden za svaki parametar, osim ako je tip int

Prototip funkcije

Obezbeđuje naziv funkcije.

Obezbeđuje parametre – šta funkcija “uzima”

Obezbeđuje tip rezulata – tip koji funkcija vraća (default int)

Prototip se koristi za validaciju funkcije

Prototip je neophodan samo ako se definicija funkcije pojavljuje posle upotrebe funkcije u programu

Funkcija sa prototipom

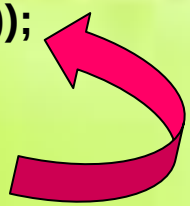
```
int maximum( int x, int y, int z );
```

Ima 3 cela broja kao argumente a vraća int vrednost

Funkcija mora biti deklarisana pre mesta na kome se poziva kako bi prevodilac znao broj i tip argumenata funkcije i tip rezultata koji vraća. Stoga je prirodno definiciju funkcije staviti pre funkcije main.

Naći minimum od dva cela broja. U glavnom programu obezbediti štampanje, a u funkciji njihovo poređenje.

```
#include<stdio.h>
int imin(int n,int m); /* prototip funkcije*/
void main() /* glavni program */
{
        int broj1,broj2,vrati;
        int imin(int,int);
        printf("Unesite dva cela broja\n");
        vrati=scanf("%d %d",&broj1,&broj2);
        if(vrati==2)
        printf("Manji od %d i %d je %d\n",broj1,broj2,imin(broj1,broj2));
}
```



```
int imin(int n,int m) /* funkcija */
   {
        int min;
          if (n<m)
            min=n;
          else
            min=m;
          return min;
                    }
```

MNOZENJE I SABIRANJE DVA REALNA BROJA PREKO FUNKCIJE

```
#include<stdio.h>
#include<math.h>
void mno(); /* prototip funkcije 1*/
void sab();  /* prototip funkcije 2*/
void main() /* glavni program */
{
mno();
sab();    }

void mno() /* funkcija 1*/
{float a,b;
printf("\nUnesite vrednosti za a i b\n");
scanf("%f%f",&a,&b);
printf("a*b=%f",a*b);}

void sab() /* funkcija 2*/
{float a,b;
printf("\nUnesite vrednosti za a i b\n");
scanf("%f%f",&a,&b);
printf("a+b=%f",a+b);}
```

Glavni program se sastoji samo od dve programske linije za poziv funkcija

Suma kvadrata celih brojeva preko funkcije

```
/*Program za izracunavanje sume
kvadrata celih brojeva od 1 do n*/

#include <stdio.h>
void suma_kvadrata(int n); /* prototip funkcije sa argumentom*/
void main( ) /* glavni program */
        {
        suma_kvadrata(10);
        suma_kvadrata(15);
        suma_kvadrata(20);
        }

void suma_kvadrata(int n)          /*f ja za izracunavanje*/
                                   /*sume kvadrata*/
        {
        int i;
        long suma=0;
        for (i=1; i<=n; suma+=(long)i*i, ++i);
        printf("Suma kvadrata od 1 do %d je %ld\n", n, suma);
        }
```

Napisati funkciju koja vrši sabiranje dva cela broja i program koji testira rad ove funkcije. Funkcija je definisana u okviru svog prototipa.

```
#include <stdio.h>
/* Definicija funkcije */
int zbir(int a, int b)
{
return a+b;
}
int main() //pozivajuca funkcija
{
/* Poziv funkcije zbir */
int rezultat = zbir(3,5);
printf("%d\n", rezultat);
return 0;
}
```

Program računa vrednost Ojlerove funkcije pozitivnog celog broja unetog sa ulaza. Pod Ojlerovom funkcijom Φ(n) podrazumevamo broj brojeva m, takvih da je 1 ≤ m < n i da su m i n uzajamno prosti. Ojlerova funkcija, kao i NZD(m, n) se računaju u posebnim funkcijama.

```
#include <stdio.h>
/* Deklaracije funkcija (prototipovi) */
int nzd(int n, int m);
int euler(int n);
/* main */
int main()
{
int n;
/* Unosimo ceo broj, a zatim izracunavamo
njegovu Ojlerovu funkciju, i ispisujemo
vrednost na izlaz */
printf("Uneti pozitivan ceo broj: ");
scanf("%d", &n);
printf("euler(%d) = %d\n", n, euler(n));
return 0;
}
```

```
/* Ojlerova funkcija broja n */
int euler(int n)
{ int br = 0, m;
/* Prolazimo kroz sve prirodne brojeve m koji su manji od
n, i za svaki koji je uzajamno prost sa n uvecavamo brojac */
for(m = 1; m < n; m++)
if(nzd(n,m) == 1)
br++;
/* Vracamo broj uzajamno prostih brojeva sa brojem n */
return br; }
```

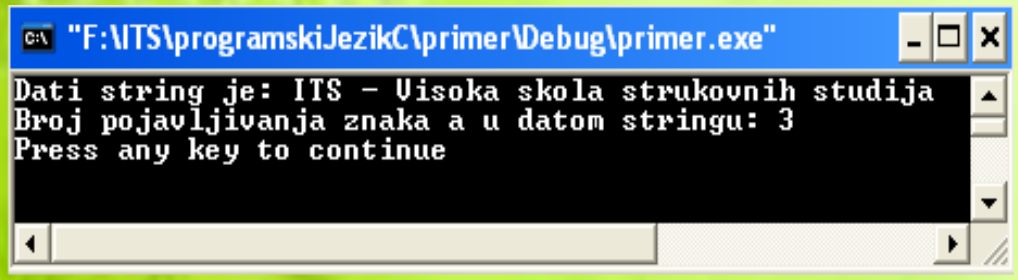
```
/* Funkcija racuna NZD dva pozitivna broja
primenom euklidovog algoritma. */
int nzd(int n, int m)
{ int r;
/* Dokle god je m!=0, svodimo nzd(n,m) na
nzd(m,r), gde je r = n % m. */
while(m)
{ r = n % m;
n = m;
m = r; }
/* Nakon sto je m postalo 0, jasno je da je
nzd(n,0) = n */
return n; }
```

Napisati funkciju koja izračunava zbir n-tih stepena brojeva od 1 do zadate granice i program koji ilustruje rad ove funkcije.

```
#include <stdio.h>
long zbir_stepena(int n, int granica);
int main()
{
printf("Zbir 2. stepena od 1 do 5 jeste %ld\n",
zbir_stepena(2,5));
printf("Zbir 3. stepena od 1 do 5 jeste %ld\n",
zbir_stepena(3,5));
printf("Zbir 4. stepena od 1 do 10 jeste %ld\n",
zbir_stpena(4,10));
return 0;
}
lont zbir_stepena(int n, int granica)
{
int i, j; /* brojaci u petljama */
long zbir = 0;
long stepenovan;
/* Spoljasnji for ciklus obavlja sumiranje */
for(i = 1; i <= granica; zbir= zbir + stepenovan, ++i)
/* Unutrasnji for ciklus obavlja stepenovanje */
for(stepenovan = 1, j = 1; j <= n; stepenovan= stepenovan *i, ++j);
return zbir;
}
```

Primer: Program koji u datom stringu utvrdjuje broj pojavljivanja datog znaka 'a', primenom funkcije

```
#include <stdio.h>
int brojanjeZnaka(char znak, char *s);
main()
    {
      int ukupno;
      char izraz[ ] = "ITS - Visoka skola strukovnih studija";
      printf("Dati string je: %s\n", izraz);
      ukupno = brojanjeZnaka('a',izraz);
      printf("Broj pojavljivanja znaka a u datom stringu je: ");
      printf("%d\n",ukupno);
    }
int brojanjeZnaka(char znak, char *s)
    {
      int suma = 0;
      while(*s)
      {
        if(*s == znak) // poredjenje trenutnog znaka za datim znakom
          suma++;
        s++; // predji na sledeci karakter stringa
      }
      return suma;
```



Složeni tipovi podataka

C++ raspolaže sledećim složenim tipovima podataka:

polja (indeksirane promjenljive)

- jednodimenzionalno = NIZ,
- dvodimenzionalno = MATRICA,
- višedimenzionalno

✓ strukture

✓ unije

Nizovi

Niz ili vektor je jednodimenzionalno polje

To je skup podataka istog tipa

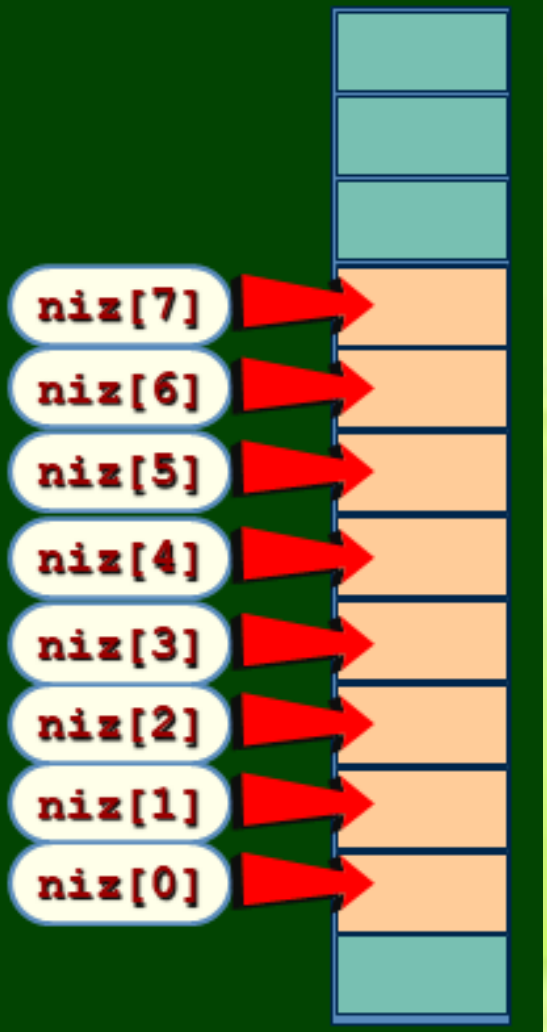
U memoriji se smešta u niz uzastopnih mem. lokacija

Svi podaci u nizu imaju jedno ime = IME NIZA

Ime niza predstavlja početnu adresu niza u memoriji

Svaki element u nizu određen je:

- ✓ imenom niza, i
- ✓ indeksom (pomerajem u odnosu na početak niza)



Deklaracija niza

Opšti oblik deklaracije:

```
tip ime_niza [broj_elemenata] = {lista vrednosti}
```

tip niza
(tip podataka u nizu)

ime niza
(identifikator u skladu sa sintaksom jezika, važe sva pravila kao i za skalarne promjenljive)

broj elemenata u nizu
(celobrojna konstanta)
(rezerviše se potreban broj bajtova za memorisanje deklarisanog broja elemenata)

vrednosti elemenata u nizu
(inicijalizacija niza)
(nije obavezno inicijalizovanje)
(vrednosti se razdvajaju zapetama)

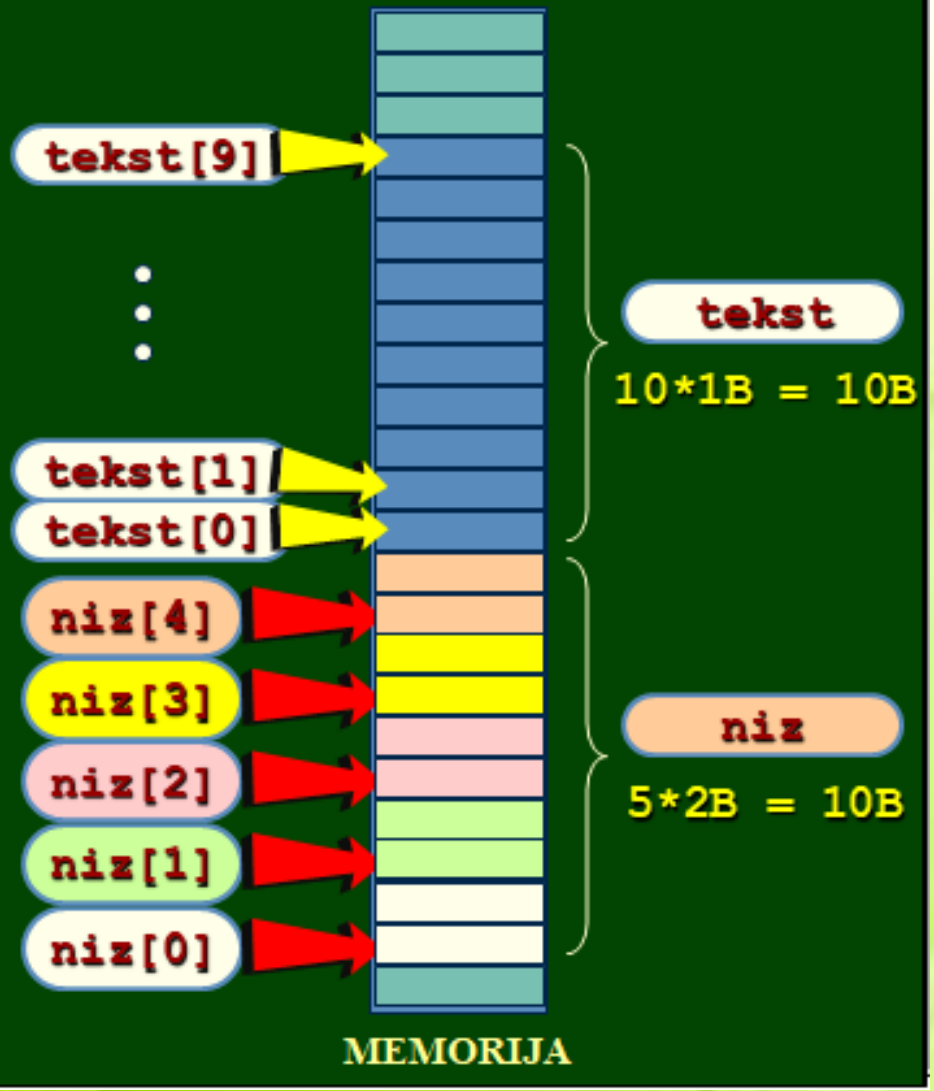
Deklaracija niza

Primer deklaracije:

```
int niz[5];
char tekst[10];
```

Primer:

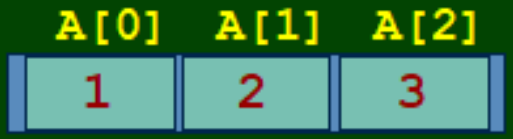
```
#define MAX 5
#define LENG 10
main()
{
   int niz[MAX];
   char tekst[LENG];
   ...
}
```



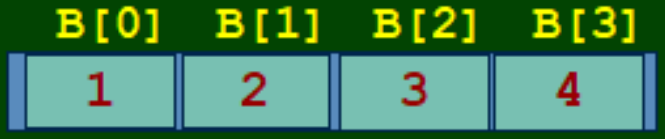
Deklaracija niza

Primer deklaracije sa inicijalizacijom:

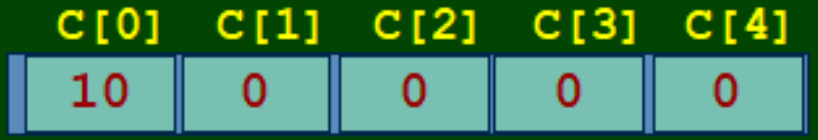
```
int A[3]={1,2,3};
```



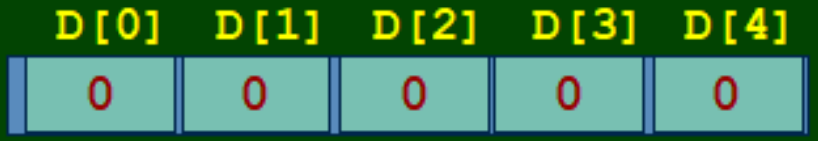

```
int B[]={1,2,3,4};
```




```
int C[5]={10};
```

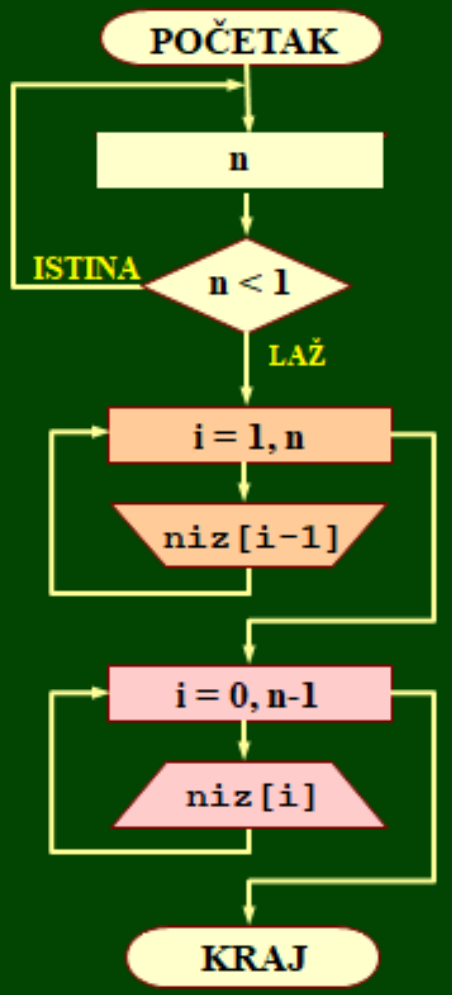



```
int D[5]={0};
```



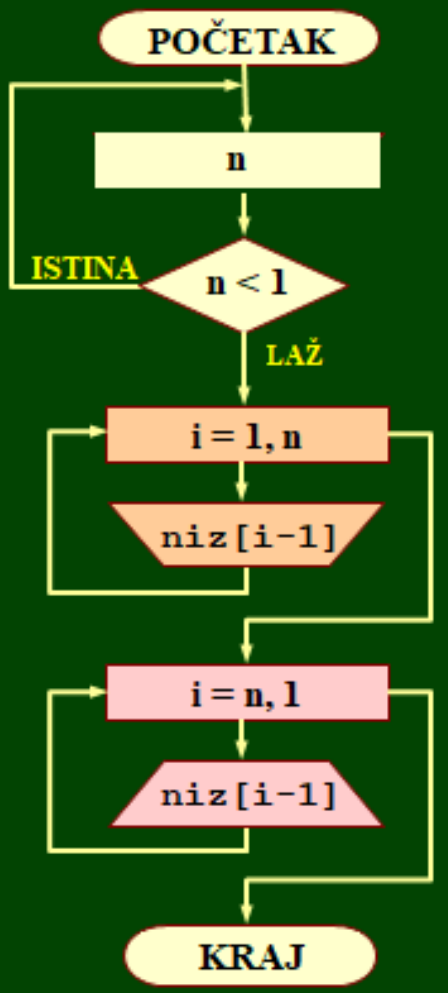
Primer: Učitati niz od *n* celih brojeva, pa ih ispisati redosledom kojim su učitani.

```
#include <stdio.h>
#define MAX 100
main()
{
    int niz[MAX];
    int i,n;
    do
    {  printf("n="); scanf("%d", &n); }
    while (n<1);
    for ( i=1 ; i<=n ; i++ )
    {  printf("Unesite %d. broj:");
       scanf("%d", &niz[i-1]); }
    printf("Uneli ste: ");
    for ( i=0 ; i<n ; i++ )
       printf(" %d", niz[i]);
}
```

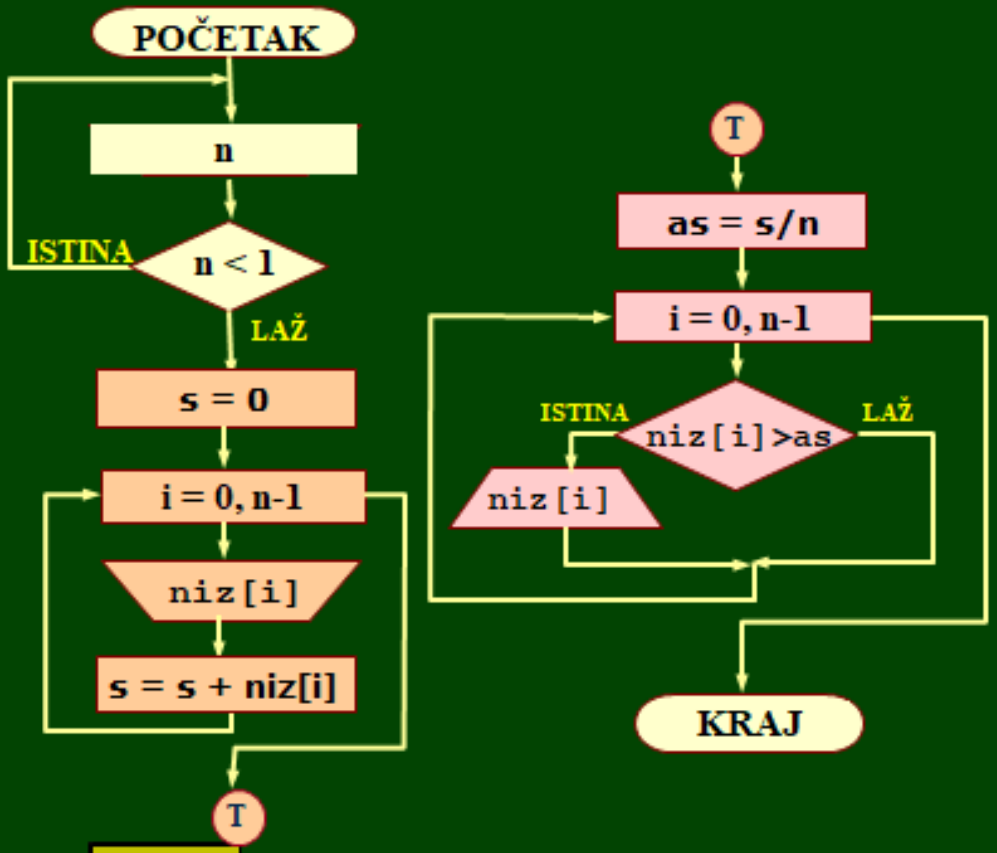


Primer: **Učitati niz od *n* celih brojeva, pa ih ispisati obrnutim redosledom.**

```
#include <stdio.h>
#define MAX 100
main()
{
    int niz[MAX];
    int i,n;
    do
    {  printf("n="); scanf("%d", &n); }
    while (n<1);
    for ( i=1 ; i<=n ; i++ )
    {  printf("Unesite %d. broj:");
       scanf("%d", &niz[i-1]); }
    printf("Uneli ste: ");
    for ( i=n ; i>=1 ; i++ )
       printf(" %d", niz[i-1]);
}
```



Primer: Učitati niz od *n* celih brojeva, pa ispisati njihovu aritmetičku sredinu i one učitane brojeve koji su veći od sredine.




```
#include <stdio.h>
#define MAX 100
main()
{
  int niz[MAX];
  int i,n,s=0;
  float as;
  do
  {  printf("n=");
     scanf("%d", &n); }
  while (n<1);
  for (i=0;i<=n-1;i++)
  { printf("Unesite broj:");
    scanf("%d", &niz[i]);
    s+=niz[i]; }
  as=(float)s / n;
  printf("Sredina: %7.3f\n",as);
  printf("Veci su: ");
  for ( i=0 ; i<n ; i++ )
    if (niz[i]>as)
       printf(" %d", niz[i]);
}
```

Primer: **Učitati niz od *n* celih brojeva, pa ispisati njihovu aritmetičku sredinu i onaj učitani broj koji joj je najbliži.**

```
#include <stdio.h>
#include <math.h>
#define MAX 100
main()
{  int niz[MAX], i, rb=0, n, s=0;
   float as;
   do
   {  printf("n="); scanf("%d", &n); }
   while (n<1);
   for ( i=0; i<=n-1; i++ )
   {  printf("Unesite broj:"); scanf("%d", &niz[i]);
      s+=niz[i]; }
   as=(float)s / n;
   for ( i=1 ; i<n ; i++ )
     if ( fabs(niz[i]-as) < fabs(niz[rb]-as) ) rb=i;
   printf(" Sredina je: %7.3f\n", as);
   printf(" Najblizi je %d. po redu\n", rb+1);
   printf(" To je broj: %d", niz[rb]);
}
```

Primer: **Učitati niz A od *n* celih brojeva, pa zatim formirati niz B, tako da je element b_i jednak najvećoj cifri elementa a_i.**

```
#include <stdio.h>
#define MAX 100
main()
{  int a[MAX], b[MAX], i, n, pom, c;
   do
   {  printf("n="); scanf("%d", &n); }
   while ((n<1) || (n>MAX));
   for ( i=0; i<=n-1; i++ )
   {  printf("Unesite broj:"); scanf("%d", &a[i]); }
   for ( i=0 ; i<n ; i++ )
   {  b[i]=0; pom=a[i];
      do
      {  c=pom%10; pom/=10; if (c>b[i]) b[i]=c; }
      while (pom);
   }
   printf("\n A:"); for ( i=0 ; i<n ; i++ ) printf(" %d", a[i]);
   printf("\n B:"); for ( i=0 ; i<n ; i++ ) printf(" %d", b[i]);
}
```

Primer: **Program koji učitava koeficijente dva polinoma *A* i *B*, stepena *m* i *n*, respektivno, (*m* , *n* < 10), a zatim izračunava i ispisuje proizvod ta dva polinoma.**

Polinom A:

$$A_m(x) = a_m x^m + a_{m-1} x^{m-1} + \ldots + a_1 x + a_0$$

Polinom B:

$$B_n(x) = b_n x^n + b_{n-1} x^{n-1} + \ldots + b_1 x + b_0$$

Množenje dva monoma:

$$a_i x^i * b_j x^j = a_i * b_j x^{i+j}$$

Proizvod polinoma A i polinoma B:

$$C_{m+n}(x) = c_{m+n} x^{m+n} + c_{m+n-1} x^{m+n-1} + \ldots + c_1 x + c_0$$

Primer: **Program koji učitava koeficijente dva polinoma *A* i *B*, stepena *m* i *n*, respektivno, (*m* , *n* < 10), a zatim izračunava i ispisuje proizvod ta dva polinoma.**

```
#include <stdio.h>
#define MAX 10
main()
{  int a[MAX], b[MAX], c[2*MAX]={0};
   int i,j,m,n;
   printf("Polinom A:\n");
   do
   {
       printf("Stepen polinoma A (m) je:");
       scanf("%d", &m);
   }
   while ((m<0) || (m>=MAX));
   for ( i=m; i>=0; i-- )
   {
       printf("Unesite koeficijent a%d: ",i);
       scanf("%d", &a[i]);
   }
```

Nastavak

```
    printf("Polinom B:\n");
    do
    {
        printf("Stepen polinoma B (n) je:");
        scanf("%d", &n);
    }
    while ((n<0) || (n>=MAX));
    for ( i=n; i>=0; i-- )
    {
        printf("Unesite koeficijent b%d: ",i);
        scanf("%d", &b[i]);
    }

    for ( i=0; i<=m; i++ )
       for ( j=0; j<=n; j++ )
          c[i+j] += a[i] * b[j];

    printf("Proizvod:\n");
    for ( i=m+n; i>=0; i-- )
        printf(" %da%d ", c[i], i);
}
```

Primer: **Program koji učitava prirodan broj *n* (*n*<100), a zatim ispisuje *n*!**

Napomena:
Standardni prosti tipovi (int, long) omogućavaju maksimalno
12!=479001600

Rešenje problema?
Problemi "velikih brojeva" rešavaju se tako što se broj posmatra kao niz cifara!!!

	f[9]	f[8]	f[7]	f[6]	f[5]	f[4]	f[3]	f[2]	f[1]	f[0]
1!=	0	0	0	0	0	0	0	0	0	1
2!=	0	0	0	0	0	0	0	0	0	2
3!=	0	0	0	0	0	0	0	0	0	6
4!=	0	0	0	0	0	0	0	0	2	4
5!=	0	0	0	0	0	0	0	1	2	0

Primer: **Program koji učitava prirodan broj *n* (*n*<100), a zatim ispisuje *n*!**

```
#include <stdio.h>
#define MAXCIF 1000
main()
{  int f[MAXCIF]={1};
   int i,j,n, pom, prenos;
   do
   { printf("n="); scanf("%d", &n); }
   while ((n<1) || (n>100));
   for ( i=2; i<=n; i++ )
       for ( j=0, prenos=0; j<MAXCIF; j++ )
       {
           pom = f[j]*i + prenos;
           f[j] = pom % 10;
           prenos = pom / 10;
       }
   printf("%d!=",n);
   for ( j=MAXCIF-1; f[j]==0; j-- ); /* preskace vodece nule */
   for (; j>=0; j-- )
       printf("%d",f[j]);       /* ispisuje cifre */
}
```

Primer: **Program koji učitava niz celih brojeva,**
a zatim ih ispisuje u rastućem redosledu.

Sortiranje niza:

Sortiranje je postupak kojim se menja poredak elemenata u nizu, kako bi se podaci doveli u željeni redosled:

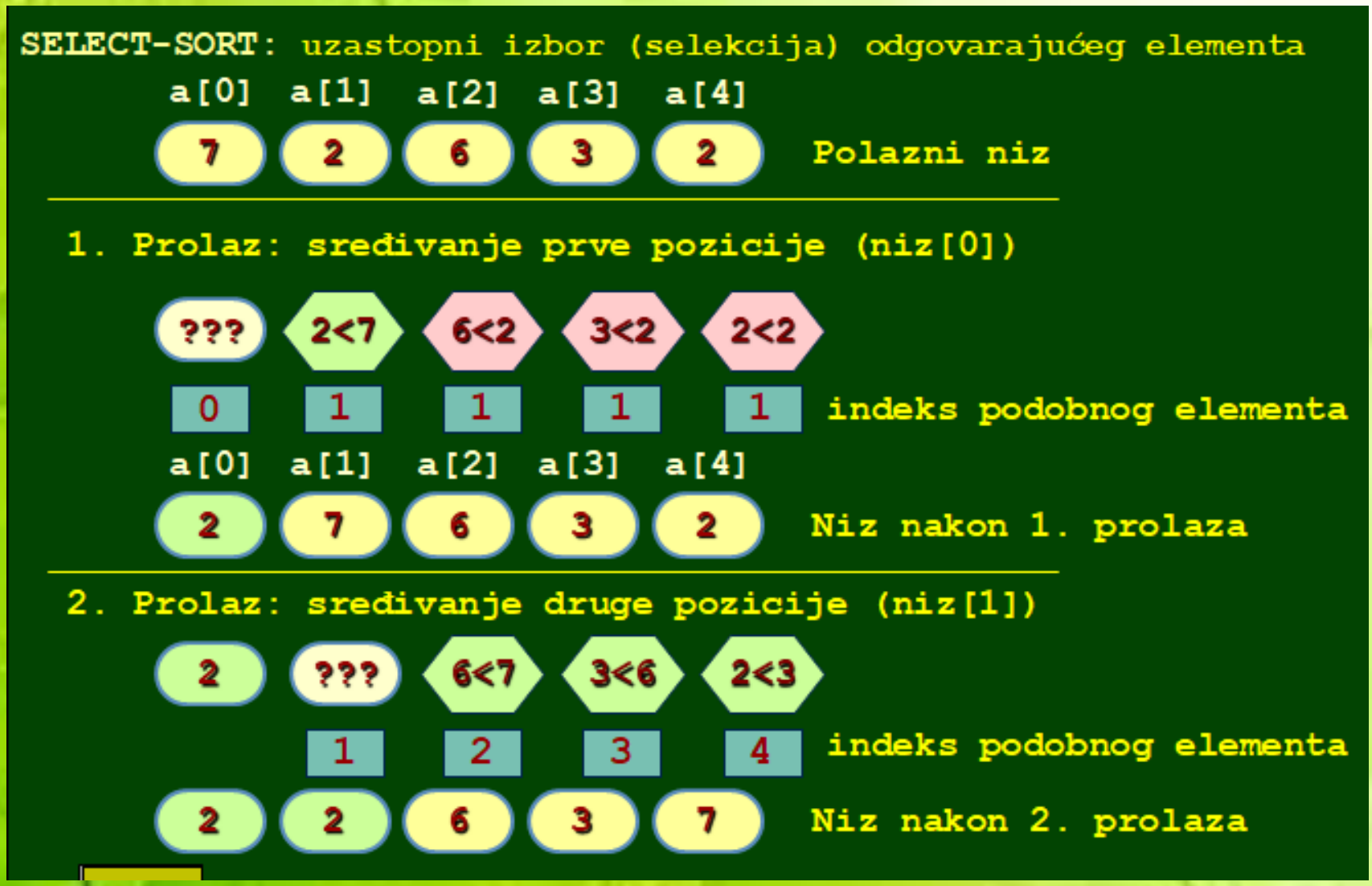
- rastući (ascending)
- opadajući (descending)

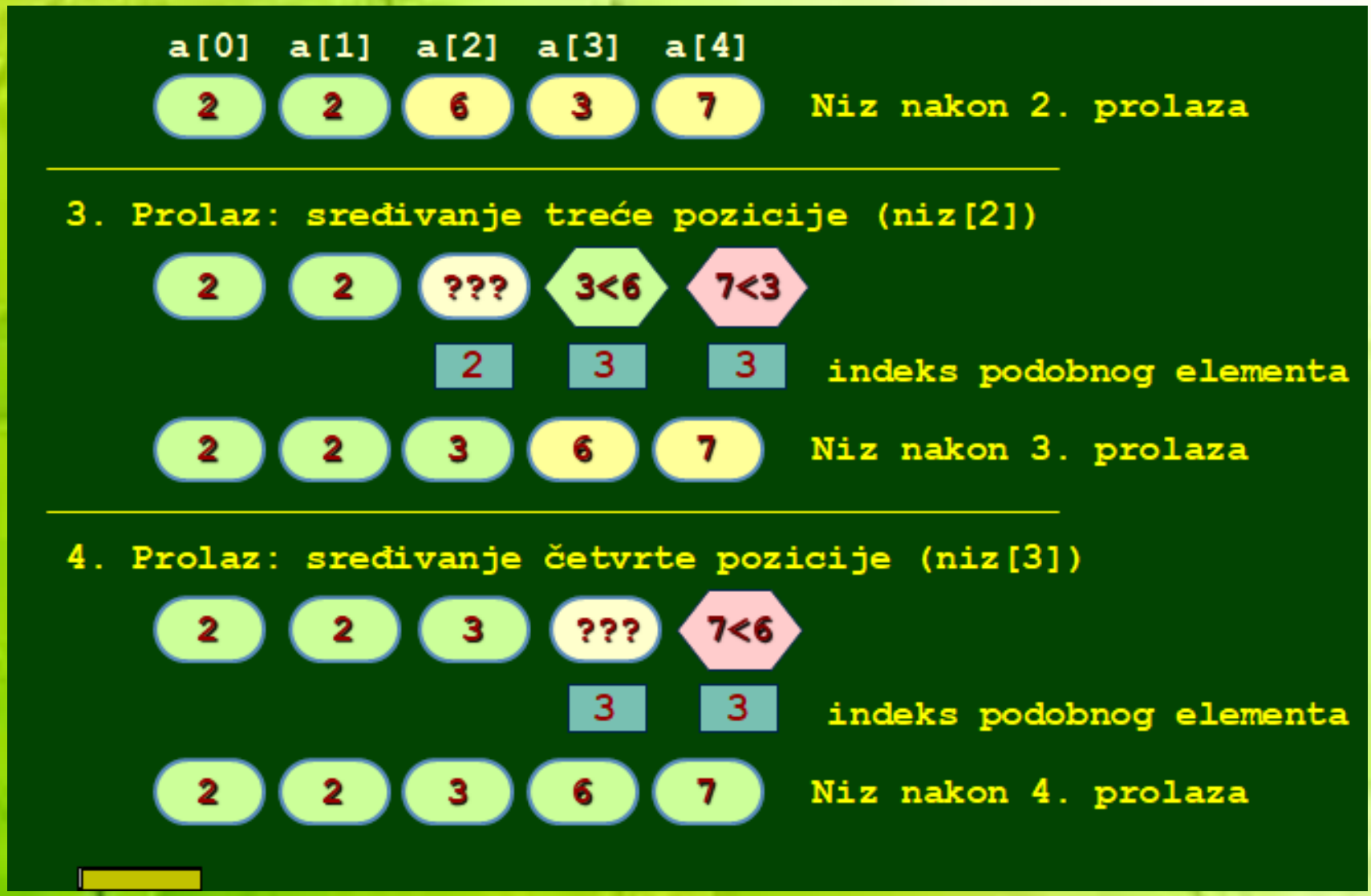
Rešenje problema?

Postoji velik broj algoritama za sortiranje

Najpopularniji algoritmi za sortiranje:

- select-sort
- bubble-sort
- quick-sort
- shell-sort
- stack-sort
- ...


SELECT-SORT: uzastopni izbor (selekcija) odgovarajućeg elementa
a[0] a[1] a[2] a[3] a[4]
7 2 6 3 2
Polazni niz
1. Prolaz: sređivanje prve pozicije (niz[0])
??? 2<7 6<2 3<2 2<2
0 1 1 1 1
indeks podobnog elementa
a[0] a[1] a[2] a[3] a[4]
2 7 6 3 2
Niz nakon 1. prolaza
2. Prolaz: sređivanje druge pozicije (niz[1])
2 ??? 6<7 3<6 2<3
1 2 3 4
indeks podobnog elementa
2 2 6 3 7
Niz nakon 2. prolaza


a[0] a[1] a[2] a[3] a[4]
2 2 6 3 7
Niz nakon 2. prolaza
3. Prolaz: sređivanje treće pozicije (niz[2])
2 2 ??? 3<6 7<3
2 3 3
indeks podobnog elementa
2 2 3 6 7
Niz nakon 3. prolaza
4. Prolaz: sređivanje četvrte pozicije (niz[3])
2 2 3 ??? 7<6
3 3
indeks podobnog elementa
2 2 3 6 7
Niz nakon 4. prolaza

Primer: **Program koji učitava niz celih brojeva, a zatim ih ispisuje u rastućem redosledu.**

```
#include <stdio.h>
#define MAX 100
main()
{  int niz[MAX];
   int i,j,n, rb, pom;
   do
   {  printf("n="); scanf("%d", &n); }
   while ((n<1) || (n>100));
   for ( i=1; i<=n; i++ )
   {  printf("Unesite broj:"); scanf("%d", &niz[i-1]); }
   for ( i=0; i<n-1; i++ )
   {
      for ( j=i+1, rb=i; j<n; j++ )
         if (niz[j] < niz[rb]) rb=j;
      if (rb!=i) { pom=niz[i]; niz[i]=niz[rb]; niz[rb]=pom; }
   }
   printf("Sortirani niz:");
   for ( i=0; i<n; i++ ) printf(" %d",niz[i]);
}
```

Tip podataka niz

Nizovi su specijalni tipovi podataka koji predstavljaju kolekciju promenljivih istog tipa. Ta kolekcija je najčešće fiksne, unapred definisane veličine.

Umesto da definišemo više promenljivih na primer broj0, broj1, broj2, broj3,..broj99, možemo definisati niz pod imenom nizBrojeva veličine 100 čiji su elementi brojevi. U tom slučaju elementima niza pristupamo sa nizBrojeva[0], nizBrojeva[1],... nizBrojeva[99].

Brojevi u uglastoj zagradi nazivaju se indeksi. Prvi element niza ima indeks 0. Elementi niza mogu se u programu koristiti kao i bilo koja druga promenljiva. Može im se dodeliti ili ispisati vrednost, mogu se koristiti kao operandi u operacijama, mogu se prosleđivati kao argumenti funkcija.

Niz je u memoriji predstavljen kao sekvenca susednih memorijskih lokacija.

Deklaracija niza

Nizovi se u programima deklarišu na sledeći način:

tip imeNiza [velicinaNiza]

Ovo je primer deklaracije jednodimenzionalnog niza, mogu se deklarisati i nizovi većih dimenzija o čemu će biti reči u sledećoj temi.

Veličina niza se izražava pozitivnim celim brojem, i ne može bit nula. Tip predstavlja tip podataka elemanata niza i može biti bilo koji tip koji postoji u programskom jeziku C. Veličina niza označava koliko će se memorijskog prostora rezervisati za promenljivu i on je jednak proizvodu veličine niza i memorije potrebne za skladištenje tipa koji predstavlja tip elementa niza.

Primer deklaracije niza veličine 10 čiji elementi su tipa double: double niz[10]. Ova deklaracija će rezervisati 10 X 8 = 80 bajtova memorije (veličina tipa double je 8 bajtova).

Inicijalizacija niza

Inicijalizacija niza predstavlja dodelu vrednosti elementima niza. Inicijalizacija se može vršiti na dva načina, dodela vrednosti svim elementima niza odjednom i dodela vrednosti jednom po jednom elementu.
Svim elementima niza vrednost se dodeljuje navodjenjem vrednosti u vitičastim zagradama razdvojenih zarezom, kao na primer

```
double niz[5] = {100.0, 2.0, 5.0, 4.0, 6.0};
```

Broj elemenata u vitičastoj zagradi mora biti jednak ili manji od zadate veličine niza. Ukoliko je broj dodeljenih vrednosti manji od veličine niza, na preostala mesta postavljaju se podrazumevane vrednosti koje zavise od tipa, na primer za int je to 0, za float i double 0.0.

Moguće je dodeliti elemente nizu bez zadavanje veličine niza:

```
double niz[ ] = {100.0, 2.0, 5.0, 4.0, 6.0};
```

Ovom naredbom kreira se niz čija veličina odgovara broju dodeljenih elemenata. Dodela vrednosti jednom elementu niza izgleda ovako:

```
niz[4] = 23.0;
```

Primer programa koji ilustruje dodelu vrednosti elementima niza i ispis elemanata niza. Za obradu nizova najčešće se koristu for petlja sa brojačem i koji redom uzima vrednosti indeksa niza, znači počinje od 0 i ide do veličine-1 (uslov je i<velicinaNiza).

```
#include <stdio.h>
int main ()
{ int niz[10]; /* deklarisemo niz od 10 integera */
int i,j; /* inicijalizacija vrednosti elemenata niza */
for (i = 0; i<10; i++)
{ niz[i] = i + 100; /* dodela vrednosti i-tom elementu*/ }
/* ispis vrednosti elemenata niza*/
for (j = 0; j < 10; j++ )
{ printf("niz[%d] = %d\n", j, niz[j] ); }
return 0; }
```

Multidimenzionalni nizovi

Multidimenzionalni nizovi su nizovi većih dimenzija, što znači da imamo nizove čiji su elementi nizovi, koji opet kao elemente mogu imati nizove.

Multidimenzionalni nizovi deklarišu se na sledeći način:

tip naziv[velicina1][velicina2][velicina3]..[velicinaN];

Matrice:

Od multidimenzionalnih nizova najčešće se koristi dvodimenzionalni niz koji se još naziva i matrica, a to je niz čiji su elementi jednodimenzionalni nizovi i deklariše se na sledeći način:

tip dvodimenzionalniNiz [m][n];

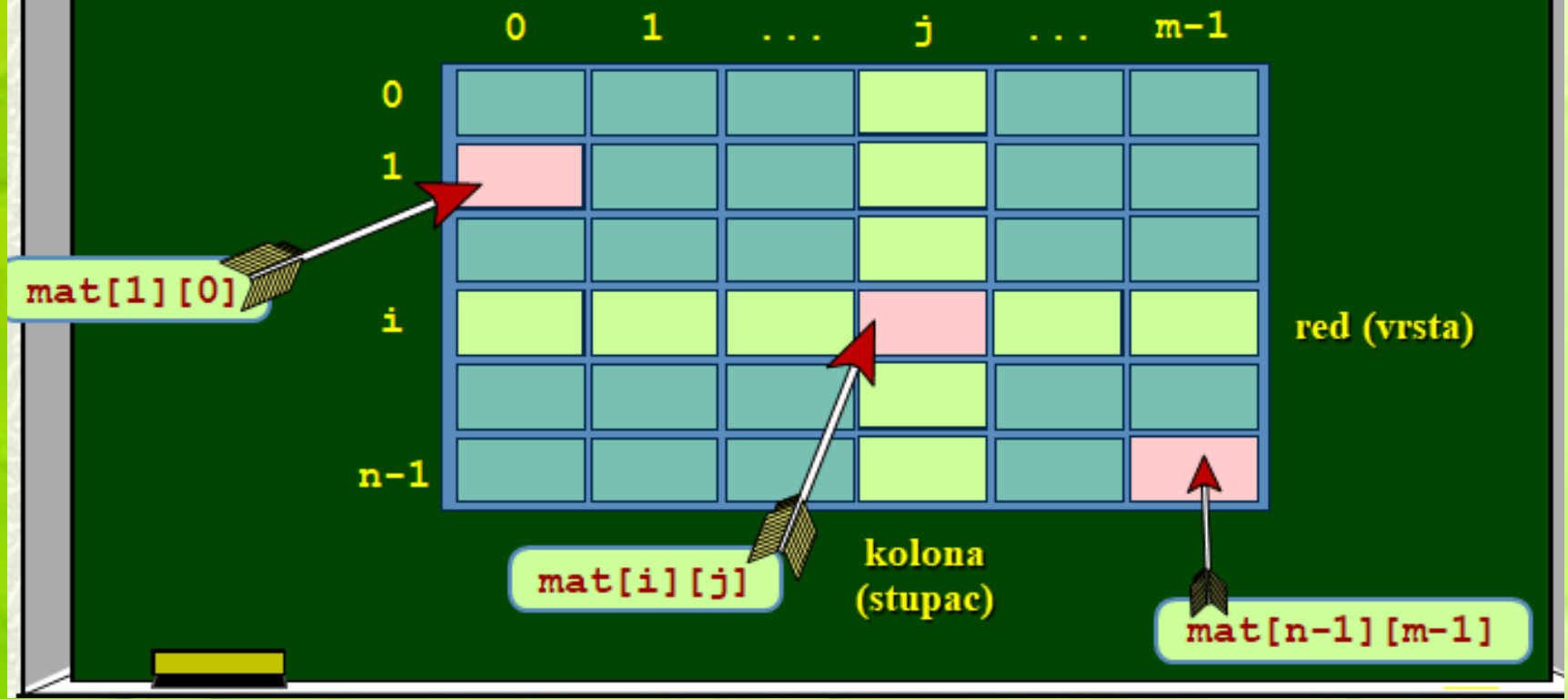
Ovakav dvodimenzionalni niz se može predstaviti i kao tabela koja ima m vrsta i n kolona.
Ako bismo imali matricu a[3][4], njene vrednosti bi se tabelarno mogle predstaviti na sledeći način:

	Kolona 0	Kolona 1	Kolona 2	Kolona 3
Vrsta 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Vrsta 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Vrsta 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Dvodimenzionalna polja

Element niza može da bude novi niz

tako dobijamo dvodimenzionalno polje (matrica)



Deklaracija matrice

Opšti oblik deklaracije:

```
tip ime_mat [D1][D2] = { {L1},{L2},...,{Ln-1} }
```

tip matrice
(tip podataka)

ime matrice

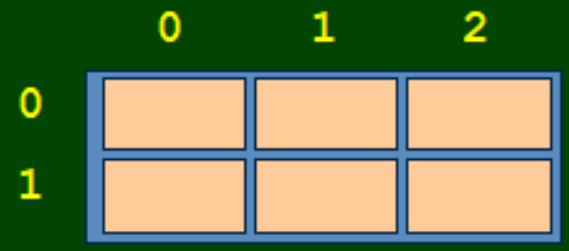
vrednosti elemenata u matrici
inicijalizacija matrice
nije obavezno inicijalizovanje

dimenzije matrice
D1 = broj redova
D2 = broj kolona
rezerviše se potreban broj bajtova
za memorisanje D1*D2 elemenata

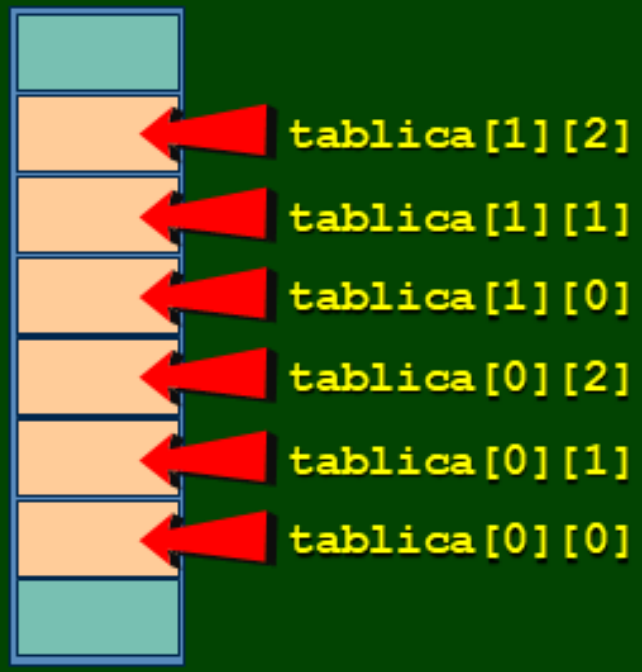
Deklaracija matrice

Primer deklaracije:

```
char tablica[2][3];
```



logička predstava

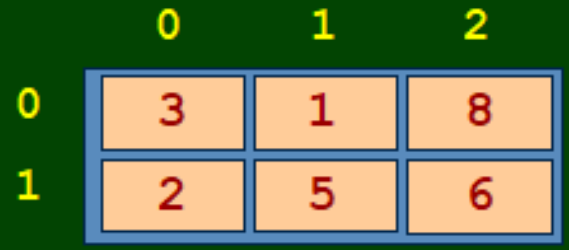


fizička predstava

Deklaracija matrice

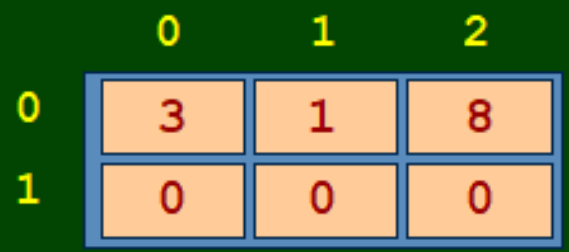
Primer deklaracije sa inicijalizacijom:

```
int mat[2][3] = { {3,1,8}, {2,5,6} };
```



Primer deklaracije sa inicijalizacijom:

```
int mat[2][3] = { {3,1,8} };
```



Manipulacija matricom

Primer učitavanja matrice:

```
for ( red=1; red<=n; red++ )
  for ( kol=1; kol<=m; kol++ )
  {
     printf("mat[%d][%d]=",red-1,kol-1);
     scanf("%d", &mat[red-1][kol-1]);
  }
```

ili:

```
for ( red=0; red<n; red++ )
  for ( kol=0; kol<m; kol++ )
  {
     printf("mat[%d][%d]=",red,kol);
     scanf("%d", &mat[red][kol]);
  }
```

Manipulacija matricom

Primer ispisivanja matrice:

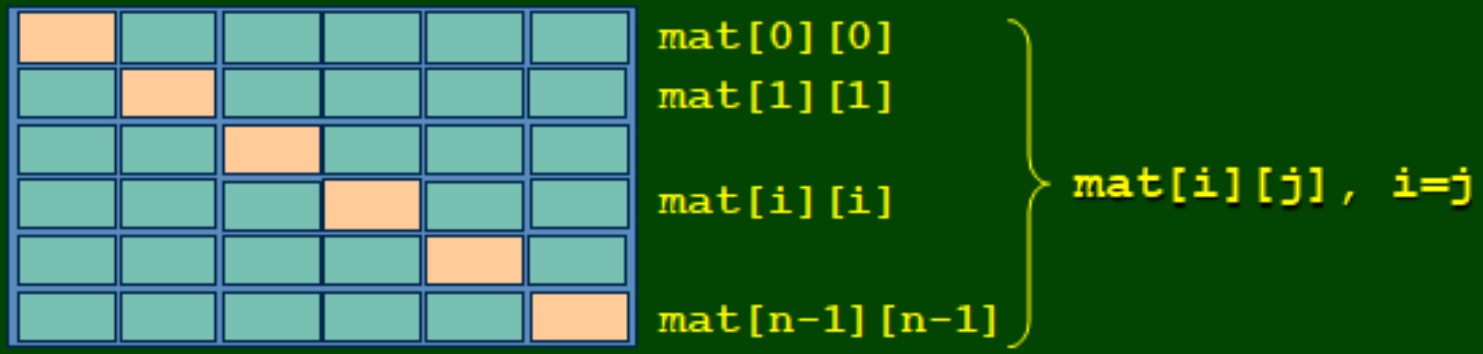
```
for ( red=1; red<=n; red++ )
{
   for ( kol=1; kol<=m; kol++ )
      printf(" %4d",mat[red-1][kol-1]);
   printf("\n");
}
```

ili:

```
for ( red=0; red<n; red++ )
{
   for ( kol=0; kol<m; kol++ )
      printf(" %4d",mat[red][kol]);
   printf("\n");
}
```

Manipulacija matricom

Primer manipulacije glavnom dijagonalom:



Ispis elemenata na glavnoj dijagonali:

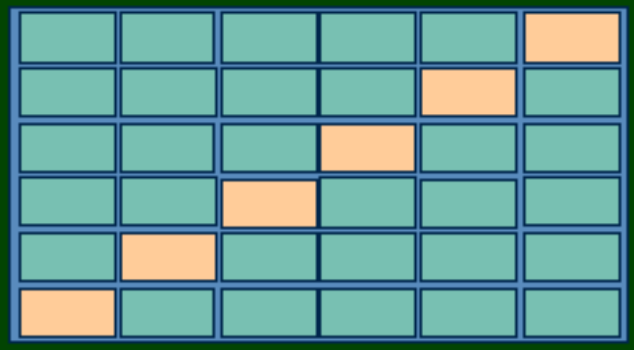
```
printf("Glavna dijagonala: ");
for ( i=0; i<n; i++ ) printf(" %d", mat[i][i]);
```

Suma elemenata na glavnoj dijagonali:

```
for ( s=i=0; i<n; i++ ) s+=mat[i][i];
printf("Suma elemenata na GD: %d",s);
```

Manipulacija matricom

Primer manipulacije sporednom dijagonalom:



Ispis elemenata na sporednoj dijagonali:

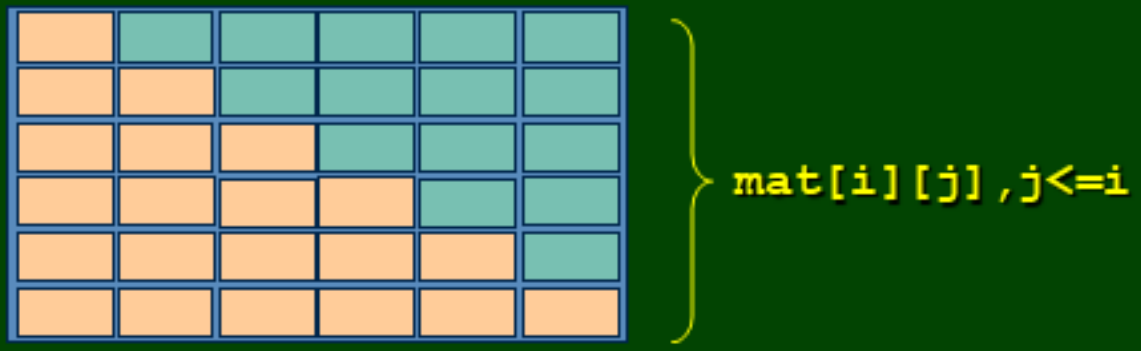
```
printf("Sporedna dijagonala: ");
for ( i=n-1; i>=0; i-- ) printf(" %d", mat[i][n-1-i]);
```

Suma elemenata na sporednoj dijagonali:

```
for ( s=i=0; i<n; i++ ) s+=mat[i][n-1-i];
printf("Suma elemenata na SD: %d",s);
```

Manipulacija matricom

Primer manipulacije donjom trougaonom matricom:

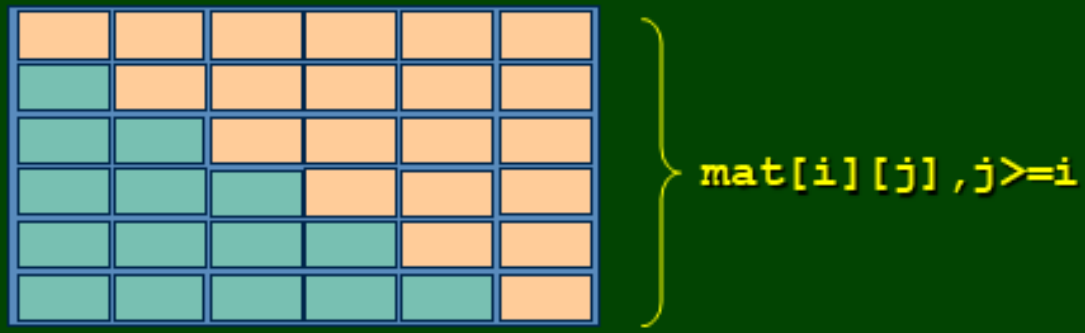


Ispis donje trougaone matrice:

```
printf("Donja trougaona matrica:\n ");
for ( i=0; i<n; i++ )
{
    for ( j=0; j<=i; j++ )
        printf(" %4d", mat[i][j]);
    printf("\n");
}
```

Manipulacija matricom

Primer manipulacije gornjom trougaonom matricom:

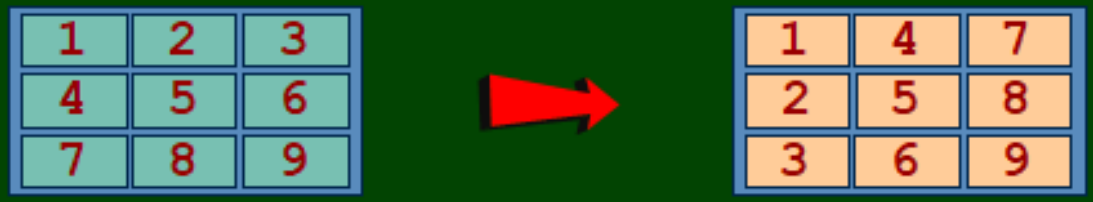


Ispis gornje trougaone matrice:

```
printf("Gornja trougaona matrica:\n ");
for ( i=0; i<n; i++ )
{
    for ( j=1; j<=i; j++ ) printf("%5c",' ')
    for ( j=i; j<n; j++ ) printf(" %4d", mat[i][j]);
    printf("\n");
}
```

Manipulacija matricom

Primer transponovanja matrice



polazna matrica

transponovana matrica

Ispis transponovane matrice na osnovu originalne matrice:

```
printf("Transponovana:\n ");
for ( i=0; i<n; i++ )
{
  for ( j=0; j<n; j++ )
    printf(" %4d", mat[j][i]);
  printf("\n");
}
```

Transponovanje matrice:

```
for ( i=0; i<n; i++ )
  for ( j=i+1; j<n; j++ )
  {
      pom=mat[i][j]);
      mat[i][j]=mat[j][i];
      mat[j][i]=pom;
  }
```

Manipulacija matricom

Primer množenja dvije matrice

Množenje matrica:

```
for ( i=0; i<m; i++ )          /* red matrice C */
  for ( j=0; j<p; j++ )        /* kolona matrice C */
     for ( k=0; k<n; k++ )
        c[i][j]+=a[i][k]*b[k][j];
```

Primer:

Primer formiranja elemenata matrice int val = a[5][2]

```
#include <stdio.h>
int main ()
{ /* polje matrice sa 5 redova i 2 kolone*/
 int a[5][2] = { {0,0},{1,2},{2,4},{3,6},{4,8}};
int i, j; /* izlazne vrednosti elemenata matrice*/
for ( i = 0; i < 5; i++ )
    { for ( j = 0; j < 2; j++ )
    { printf("a[%d][%d] = %d\n", i,j, a[i][j] ); } }
return 0; }
```

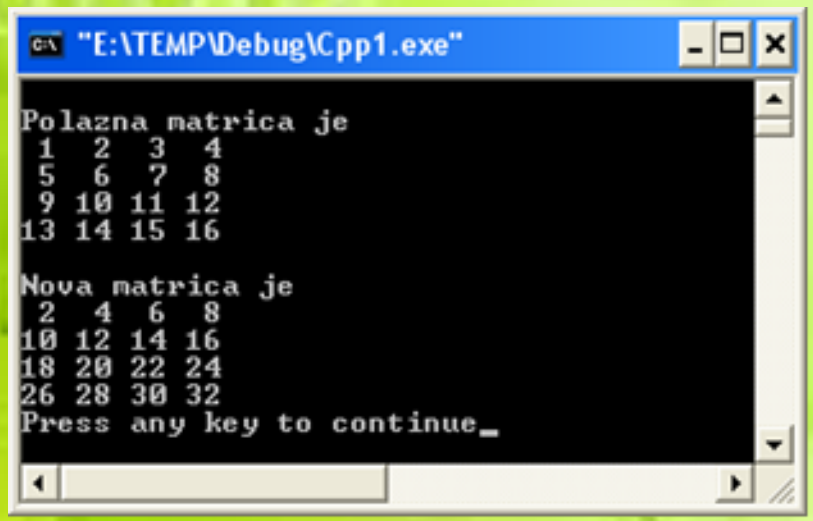
```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

Neka je data kvadratna matrica u formatu:

$$\begin{vmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{vmatrix}$$

Članove matrice uneti u program kao konstantne celobrojne veličine, a kao izlaz iz programa predvideti štampanje ove matrice u pokazanom formatu kao polazne matrice i štampanje nove matrice u istom obliku ali čiji su svi članovi sada pomnoženi skalarom 2.

```
 /* 1 STAMPANJE CLANOVA MATRICE I CLANOVA MATRICE POMNOZENE
     SKALAROM*/
#include<stdio.h>
#include<math.h>
void main()
{int mat[4][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
int i,j;
printf("\nPolazna matrica je\n");
for(i=0;i<4;i++)
{
 for(j=0;j<4;j++)
{printf("%2d ",mat[i][j]);}printf("\n");}
 printf("\nNova matrica je\n");
for(i=0;i<4;i++)
{
 for(j=0;j<4;j++)
{printf("%2d ",2*mat[i][j]);}printf("\n"); } }
```



Program na C jeziku koji izračunava sumu svih elemenata u svakom redu pojedinačno, pravougaone matrice [4x4].

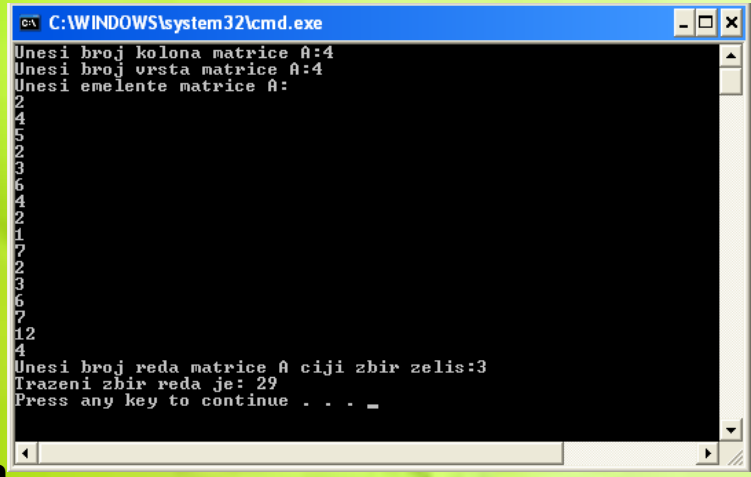

```
#include<stdio.h>
#include<math.h>
void main()
{int mat[4][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
int i,j;
printf("\nPolazna matrica je\n");
for(i=0;i<4;i++)
{           for(j=0;j<4;j++)
            {printf("%2d ",mat[i][j]);}printf("\n");}
 printf("\nNova matrica je\n");
for(i=0;i<4;i++)
{           for(j=0;j<4;j++)
{printf("%2d ",mat[i][j]);}printf("\n");}}
```

Program na C++ jeziku koji izračunava sumu svih elemenata u svakom redu pojedinačno, pravougaone matrice [4x4].


```
#include<stdio.h>
#include<math.h>
void main()
{int mat[4][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
int i,j,s=0;
printf("\nPolazna matrica je\n");
for(i=0;i<4; i++)
{
 for(j=0;j<4; j++)
{printf("%2d ",mat[i][j]);}printf("\n");}

for(i=0;i<4; i++)
{printf("\nSuma %d reda je:",i+1);
 for(j=0;j<4; j++)

 {s=s+mat[i][j];}
 printf("%2d\n",s);s=0;}
return 0;}
```

Primer: Program za izračunavanje bilo kojeg reda unete matrice A.

```
#include<stdio.h>
#include<math.h>
main()
{
// deklarisanje promenljivih
int i,j,n,m,red,zbir=0;
// deklarisanje matrice
int A[20][20];
printf("Unesi broj kolona matrice A:"); scanf("%d",&n);
printf("Unesi broj vrsta matrice A:"); scanf("%d",&m);
//unos elemenata matrice
printf("Unesi emelente matrice A:\n");
for(i=0;i<m;i++)
     for(j=0;j<n;j++)
              scanf("%d",&A[i][j]);
zbir=0;
printf("Unesi broj reda matrice A ciji zbir zelis:");
              scanf("%d",&red);
for(j=0;j<n;j++) zbir=zbir+A[red][j];
              printf("Trazeni zbir reda je: %d\n",zbir);}
```



Višedimenzionalna polja

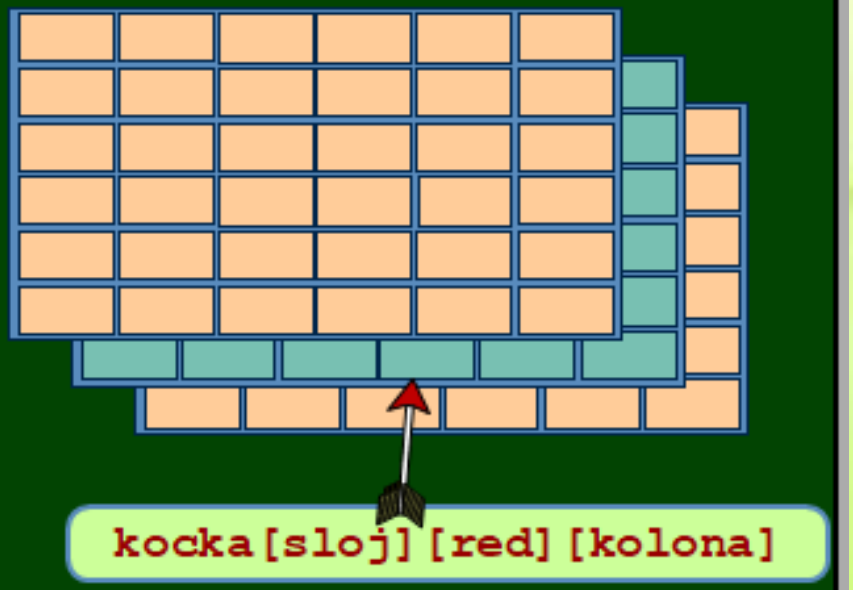
C omogućava i manipulaciju višedimenzionalnim poljima

Trodimenzionalno polje

```
tip ime[d1][d2][d3];
```

Primer deklaracije:

```
int kocka[3][6][6];
```



Primer 4-dim. polja:

```
char P4[d1][d2][d3][d4];
```

Stek i slobodno skladište

Pomenućemo pet područja memorije:

1. **prostor globalnih imena,**
2. **slobodno skladište,**
3. **registri,**
4. **prostor za kod i**
5. **stek.**

Globalne promenljive se nalaze u prostoru globalnih imena. Registri se koriste za interne domaćinske funkcije. Kod se nalazi u prostoru za kod, naravno, kao što je pamćenje vrha steka i pokazivača instrukcija.

Lokalne promenljive su na steku, zajedno sa parametrima funkcije. Skoro sva preostala memorija se dodeljuje slobodnom skladištu. Problem sa lokalnim promenljivama je taj što one nisu trajne: Po povratku iz funkcije, one se odbacuju. Globalne promenljive rešavaju taj problem po cenu neograničenog pristupa širom programa, što vodi do kreiranja koda koji je težak za razumevanje i održavanje. Stavljanje podataka u slobodno skladište rešava oba ova problema. O slobodnom skladištu možete razmišljati kao o masivnoj sekciji memorije, u kojoj hiljade sekvencijalno označenih kockica leže, čekajući vaše podatke.

U računarstvu, **stek** je privremeni apstraktni tip podataka i struktura podataka, baziran na principu LIFO (LIFO - Last In First Out - poslednji koji ulazi, prvi izlazi). Stekovi se u velikoj meri koriste na svim nivoima modernog računarskog sistema. Stek-mašina je računarski sistem koji privremene podatke skladišti prvenstveno u stekovima, umesto u hardverskim registrima.

Stek i slobodno skladište

Tipičan stek skladišti lokalne podatke i informacije o pozivu za ugnežđene procedure. Ovaj stek raste na dole od početka. Pokazivač na stek pokazuje na trenutni vrh steka. Operacija *push* umanjuje pokazivač, i kopira podatke na stek; operacija *pop* vraća podatke sa steka, a zatim pokazivač pokazuje na novi vrh steka. Svaka procedura pozvana unutar programa skladišti svoje povratne informacije (žuto) i lokalne podatke (u drugim bojama) gurajući ih na stek. Ovakva implementacija steka je izrazito česta, ali je ranjiva na napade prekoračenja bafera. Tipičan stek je deo u memoriji računara sa fiksiranim početkom, i promenljivom veličinom. Na početku, veličina steka je nula. Pokazivač na stek, obično u vidu hardverskog registra, pokazuje na poslednju iskorišćenu adresu na steku; kada je stek veličine nula, pokazivač pokazuje na početak steka.

Dve operacije primenljive na sve stekove su:

***push* operacija, u kojoj se predmet postavlja na lokaciju na koju pokazuje pokazivač na stek, a adresa u pokazivaču se prilagođava za veličinu tog predmeta;**

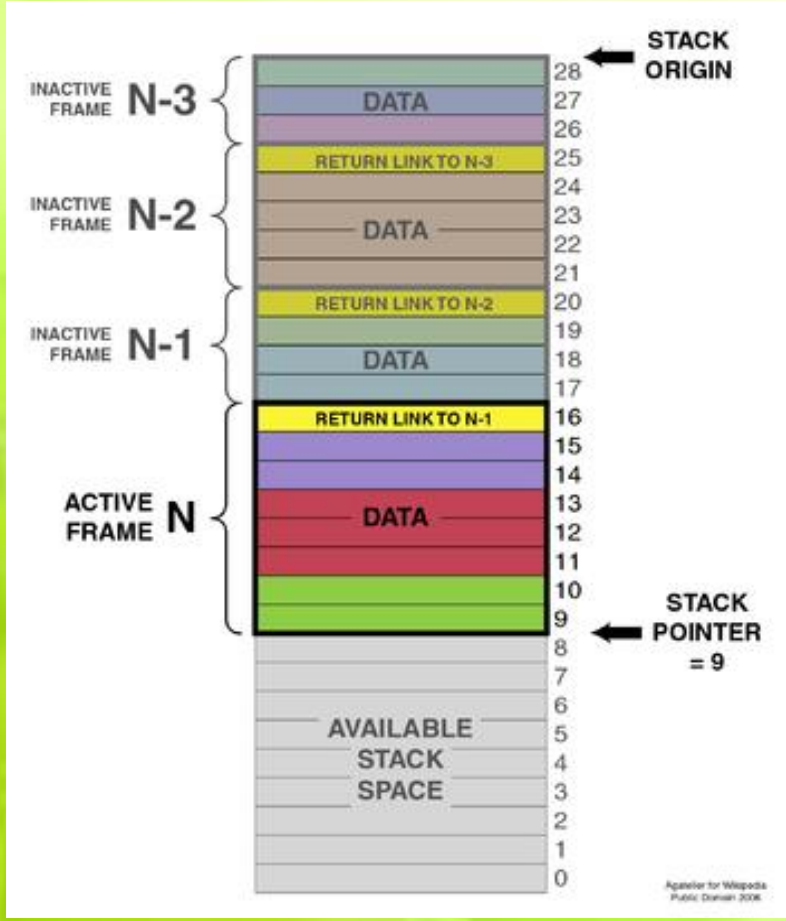
***pop* ili *pull* operacija: predmet na trenutnoj lokaciji na koju pokazuje pokazivač se uklanja, a adresa u pokazivaču se prilagođava za veličinu tog predmeta**

Stek i slobodno skladište

Dve operacije primenljive na sve stekove su:

***push* operacija, u kojoj se predmet postavlja na lokaciju na koju pokazuje pokazivač na stek, a adresa u pokazivaču se prilagođava za veličinu tog predmeta;**

***pop* ili *pull* operacija: predmet na trenutnoj lokaciji na koju pokazuje pokazivač se uklanja, a adresa u pokazivaču se prilagođava za veličinu tog predmeta**



Stek i slobodno skladište

Stek se automatski čisti po povratku iz funkcije. Sve lokalne promenljive izlaze iz opsega i uklanjaju se sa steka.

• Slobodno skladište se ne čisti sve dok se vaš program ne završi i vaš je zadatak da oslobodite svaki memorijski prostor koji ste rezervisali. Prednost slobodnog skladišta je što memorija koju rezervišete ostaje raspoloživa, dok je eksplicitno ne oslobodite. Ako rezervišete memorju na slobodnom skladištu u funkciji, memorija je još uvek raspoloživa po povratku iz funkcije. Prednost pristupanja memoriji na ovaj način, umesto korišćenja globalnih promenljivih, je to da samo funkcije sa pristupom pokazivaču imaju pristup podacima.

U aplikacionim programima pisanim u višim programskim jezicima, stek se može efikasno primeniti bilo pomoću nizova bilo pomoću povezanih listi. U jeziku Lisp nema potrebe da se implementira stek, jer su funkcije push i pop dostupne za svaku listu. AdobePoscript je takođe dizajniran oko steka koga programer direktno može da vidi, i da njime manipuliše.

Stek i slobodno skladište

- operatori new i delete -

NEW : memoriju na slobodnom skladištu u C alocirate korišćenjem ključne reči new. Posle nje sledi tip objekta koji želite da alocirate tako da kompajler zna koliko se memorije zahteva. New unsigned short int alocira dva bajta na slobodnom skladištu, a new long alocira četiri. Povratna vrednost iz new je memorijska adresa. Ona se mora dodeliti pokazivaču. Da biste kreirali unsigned short na slobodnom skladištu, napišite:

unsigned short int *pPointer;

pPointer = new unsigned short int;

U svakom slućaju, pPointer sada pokazuje na unsigned short int na slobodnom skladištu. Njega možete korisriti kao i svaki drugi pokazivač na promenljivu i dodeliti vednost području memorije.

UPOZORINJE: Svaki put kada alocirate memoriju, korišćenjem ključne reći new, morate proveriti da biste bili sigurni da pokazivač nije nula.

Stek i slobodno skladište

Primer, izraz: ((1 + 2) * 4) + 3 može biti zapisan na sledeći način u postfiksnoj notaciji uz prednost da nisu potrebna pravila prednosti i zagrade:

1 2 + 4 * 3 +

Izraz se izračunava sleva na desno korišćenjem steka:

- *push* kad se dođe do operanda
- *pop* dva operanda i njihovo računanje, kad se dođe do operacije praćeno *push* rezultatom.

Улаз	Операција	Стек
1	*Push* операнд	1
2	*Push* операнд	1, 2
+	Сабирање и Pop последња два операнда и Push резултат	3
4	*Push* операнд	3, 4
*	Множење и Pop последња два операнда и Push резултат	12
3	*Push* операнд	12, 3
+	Сабирање и Pop последња два операнда и Push резултат	15

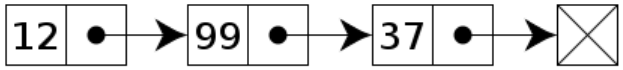
Povezana lista

Povezana lista je struktuira podataka, koja je u osnovi predstavljena kao vektor parova (element, pokazivač), pri čemu pokazivač sadrži adresu narednog para. Tako postavljeni parovi nazivaju se čvorovima. Prolazak kroz listu moguć je jedino linearno - redom od početka, element po element, prateći pokazivače.

Nedostaci ove strukture podataka su u tome što zahteva dodatni prostor u memoriji (uz svaki element ide i pokazivač), a *k-tom* (k>1) elementu se može pristupiti samo preko svih predhodnih. Prednost povezane liste je u tome što se upis i brisanje lako realizuju, potrebno je menjanje samo pokazivača (jednog u slučaju brisanja, dva u slučaju upisivanja)

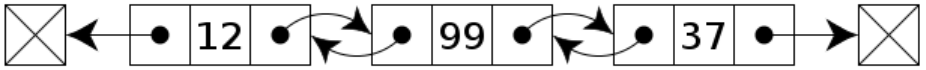
Jednostruko povezane liste

Jednostruke povezane liste imaju čvorove koji sadrže samo vrednost i pokazivač na sledeći čvor.



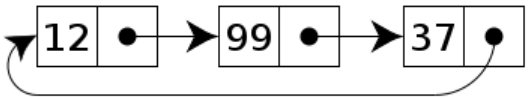
Dvostruko povezane liste

U **dvostruko povezanim listama** svaki čvor osim vrednosti i pokazivača na sledeći čvor sadrži još jedan pokazivač, na prethodni element.



Ciklične liste

Kod **cikličnih listi** poslednji čvor ne pokazuje na nil, već na glavu liste.



Ako je lista dvostruko povezana osim što poslednji čvor pokazuje na glavu, i glava svojim pokazivačem na predhodi element pokazuje na poslednji čvor.