C++

Primjer C++ programa:

Izvorni kod:

```
#include <iostream.h>
int main() {
    cout << " Ovo je program u C++";
    return 0;
}
```

Struktura programa:

`#include <iostream.h>`	Predprocesorsla naredba (sve takve naredbe započinju sa #). Od prevoditelja zahtjeva da u program uključi biblioteku <iostream.h> koja omogućava ispis podataka na ekrenu ili čitanje sa ekrana

`int main() {`
`….`
`}`
	Svaki program u C-u mora imati jednu main() funkciju. Tip int ispred funkcije određuje tip vrijednosti koju će po završetku izvođenja vratiti funkcija main() sa naredbom return. Ako ne vraća ništa napiše se **void**. Tijelo funkcije omeđeno je vitičastim zagradama { }.

`cout << " Ovo je program u C++";`	Svaka naredba mora završavati sa znakom **;**. cout je ime izlaznog toka iz datoteke iostream.h. Operatorom **<<** upućuje se podatak koji slijedi na ekran računala (izlazni tok).

Osnovni elementi

1. Komentari

- djelovi izvornog koda koji se ne prevodi u izvršni kod
- vrste komentara:
  a) /* Sav tekst sadržan između ovih oznaka je komentar
  nezavisno od toga koliko redaka obuhvaća */
  b) if (ime = = null) // nastavak retka tretira se kao komentar.

2. Identifikatori

- prvi znak mora biti slovo ili znak podcrtavanja
- može biti sastavljen od kombinacije slova engleskog alfabeta (A-Z, a-z), brojeva (0-9) i znaka za podcrtavanje (_);
- ne smije biti jednak nekoj ključnoj riječi ili nekoj od alternativnih oznaka operatora
- C++ razlikuje mala i velika slova (Program <> program)

3. Literali

- Konstantne vrijednosti (brojevi, decimalni brojevi, znakovi, stringovi ili logičke vrijednosti)
- vrste literara:

a) Brojevni literali

- cijeli brojevi:	dekadski	(-2, 50 , 222219)	- tretiraju se kao int
	long int	(645642L, -258748l)	-dodaje se sufiks **L** ili **l**
	unsigned int	(60000U, 45000u)	- dodaje se sufiks **U** ili **u**
	heksadecimalni	(0x419, 0xA)	- imaju prefikd **0x** ili **0X**
	oktalni	(0614, 0066)	- imaju prefiks **0**
- brojevi sa pomičnim zarezom:		(154.45, .5, 1.6e2, 5.99E-5, 5.)	- tretiraju se kao double
	float	(12.08F, 55.121f)	- dodaje se sufiks **F** ili **f**
	long double	(12.08L, 55.121l)	- dodaje se sufiks **L** ili **l**

b) Logičke vrijednosti: **true i false** (mogu biti dodjeljene **boolean** varijablama)

c) Znakovni literali: - 'a', 'A', '?' , '6' pojedinačni znak naveden u jednostrukim znakovima navođenja
- posebne sekvence (\ i oznaka znaka kojeg treba prikazati)

Prekidni niz	Značenje	Prekidni niz	Značenje
\	Nastavak u novom redu	\\	Kosa crta - \
\n	Prelazak u novi red	\?	Upitnik - ?
\t	Horizontalni tabulator	\'	Jednostuki navodnik - '
\b	Pomak unazad (backspace)	\"	Dvostruki navodnik - "
\r	Povratak na početak reda	\nnn	Oktalni broj - nnn
\f	Prelazak na novu stranicu	\xnn	Heksadecimalni broj - nn
\a	Zvučni signal (alert)	\0	završetak znakovnog niza

d) Nizovni literali - niz znakova naveden između dvostrukih znakova navodnika: "C je programski jezik"

4. Osnovni tipovi podataka

Tip određuje koju vrstu vrijednosti varijabla može poprimiti

Tip podataka	Naziv tipa	Bajtova	Raspon	Točnost
Cjelobrojni tipovi	short int	2	-32768 -32767	
	int	2 (4)	-32768 -32767 (-2.1E9 - 2.1E9)	
	long int	4	-2.1E9 - 2.1E9	
Tipovi za brojeve s pomičnim zarezom	float	4	-3.4E38 - 3.4E38	7 decimala
	double	8	-1.7E308 - 1.7 E308	15 decimala
	long double	10	-1.1E4932 - 1.1 E4932	18 decimala
Logički tipovi	bool	1	true, false	
Znakovni tipovi	char	1		

- navedeni tipovi mogu biti deklarirani i bez predznaka sa **unsigned** čime se udvostručuje najveća moguča vrijednost u odnosu na varijable sa predznakom, ali se ograničava samo na pozitivne vrijednosti: npr. unsigned int i = 60000;
- logički tip je kasno uvršten u standard C++ pa se u starijim prevoditeljima može simulirati sa pobrojanim tipom: **enum bool {false, true};**
- kod znakovnog tipa moguće je uspoređivati konstante i varijable relacijskim operatorima i primjenjivato aritmetičke operatore.
- ključna riječ **typedef** omogućava uvođenje novog imena za već postojeći ugrađeni ili definirani tip: npr. typedef float broj; // identifikator broj postaje sinonim za float pa ga se može ravnopravno koristiti: broj pi =3.14159;

5. Varijable

- logička imena lokacija u memoriji sustava, u koje se pohranjuju i/ili preuzima vrijednosti
- deklaracija: int i, godinaRođenja, brojRačunala;
  float pi;
  char znak;
- inicijalizacija: int i; i =55; ili int i =55;
  int x = 2*i;
- područje važenja: - varijable deklarirane u bloku vidljive su samo unutar njega;
  - lokalna varijabla zakloniti će istoimenu varijablu deklariranu izvan bloka;

Npr.

```
#include <iostream.h>
void  main() {
      int b = 1;
      {                                          // blok naredbi
         int a =1, b=5;                          // lokalna varijabla u bloku
         cout <<" a = " << a <<",  b= "<< b;     // ispisuje  a=1, b= 5
      }
cout << " a=" << a ;                    // GREŠKA !! varijabla a više ne postoji
cout << " b= "<< b;                     // ispisuje  b=1
}
```

6. Konstante

- varijabla koja će služiti kao konstnta označava se ili sa:
  - kvalifikatorom **const** koji spriječava njenu promjenu (npr**.** **const** double pi =3.14159;)
  - predprocesorskom naredbom #**define** (npr**. #define** PI =) koji pojave znaka PI zamjenjuje drugim nizom znakova 3.14159
- vrijednost simboličke konstante mora biti inicijalizirana prilikom deklaracije !

7. Izrazi i operatori

Operator pridruživanja (=)

- mijenja se vrijednost nekog objekta pri čemu tip objekta ostaje nepromjenjen

Npr: i=5; i=i+5; a=b=c=0; x=y=w+z;

Aritmetički operatori

Operator	Namjena	Rezultati
+	Zbrajanje	2+3 =5; "2"+"3" ="23"; 'A'+1 ='B'
-	Oduzimanje	
*	Množenje	
/	Dijeljenje (cjelobrojno za cijele brojeve)	12 /5 = 2; 12**.** /5 = 2,4
%	Modulo - ostatak cjelobrojnog dijeljenja	12 %7 = 5; 12.44 % 7 = 5.44

Dodjeljivanje vrijednosti

Operator	Namjena	Primjer	Sa operatorima
=	Dodjeljivanje vrijednosti	a = 2; x = y = z = 1;	
+=	Zbrajanje i dodjela vrijednosti	x = x + y	x += y
-=	Oduzimanje i dodjela vrijednosti	x = x - y	x -= y
*=	Množenje i dodjela vrijednosti	x = x * y	x *= y
/=	Dijeljenje i dodjela vrijednosti	x = x / y	x /= y
%=	Modulo i dodjela vrijednosti	x = x % y	x %= y

Tip rezultata binarne operacije na dva broja:

- oba operanda istog tipa - rezultat tog tipa
- operandi različitog tipa - svođenje na zajednički tip
- Svođenje na zajednički tip:
  - char > short int > int > long int > float > double > long double
  - bool > int (true = 1 tj. cijeli broj<>0, false =0)
  - ako je jedan od operanada unsigned, tada se i drugi operand pretvara u unsigned
  - ako je jedan od operanada long, tada se i drugi operand pretvara u long

Npr.

```
int i=3, j = 6;
float a =.5;
char x, y = 'h';
cout << (a*i);      // ispisati će se 1.5 jer je float složeniji tip
int c= a*i;
cout << c;          // ispisati će se samo cjeli dio rezultata tj. 1.
```

```
float k = i/j;        // varijable su cjelobrojne pa je djeljenje cjelobrojno !
cout << k;            // ispisati će 0 zbog cjelobrojnog djeljenja.
cout << (i/j/1.);     // vrijednost 1. je float pa će rezultat biti 0.5
cout << (y+1);        // zbog operacije sa cijelim brojem uzima se ASCI kod 'h' (104) pa će se ispisati 105
x=y+1;                // broj 105 se pridružuje znakovnoj varijabli pa se konvertira u znak
cout << x;            // ispisati će se 'i'
```

Operator dodjele tipa: (odredišni tip) varijabla

- eksplicitna promjena tipa varijable u podudarni tip; nakon primjene operatora tip varijable ostaje nepromjenjen

Npr.

```
int b =1, n =3;
float k= (float)b / n;        // vrijednost varijable b pretvara se u decimalnu pa će rezultat biti točan
int l = (int)k;               // cjelobrojne znamenke decimalnog broja
float k = b/n;                // varijable su cjelobrojne pa je djeljenje cjelobrojno
```

Inkrementiranje/ Dekrementiranje

Operator	Namjena	Primjer	Sa operatorima
++	Povećavanje vrijednosti za 1	x = x + 1	++x (uvećaj prije) x++ (uvećaj nakon)
--	Smanjivanje vrijednosti za 1	x = x - 1	-- x (umanji prije) x-- (umanji nakon)

U složenim izrazima:

y = x++; varijabli y dodjeljuje se vrijednost varijable x, a zatim se vrijednost varijable x povećava za 1

y = ++x; varijabla x poveća vrijednost za 1, a zatim se ta vrijednost dodjeljuje varijabli y.

```
npr.   x =20;  y = x++;        // (rezultat:   y=20; x=21)
       x =20;  y = ++x;        // (rezultat:   y=21; x=21)
```

char se u izrazima pretvara u int pa vrijede osnovni operatori:

```
char  zn1 = 'a', zn2 = 'b';
zn1++; zn2 = zn2 +2;  // (rezultat:   zn1 = 'b'; zn2 = 'd')
```

Relacijski operatori

= =	jednako	x = = y
!=	različito	x != y
<	manje	x < y
>	veće	x > y
<=	manje ili jednako	x <= y
>=	veće ili jednako	x >= y

- operator "=" **dodjeljuje** vrijednost izraza (sa desne strane) varijabli sa lijeve strane operatora;
- operator "= =" **uspoređuje** da li je vrijednost izraza sa lijeve strane **jednaka** vrijednosti izraza sa desne strane. Rezultat uspoređivanja je uvijek tipa **boolean**; dakle, **true** ili **false**.

Logički operatori

Izrazi kod kojih je rezultat izvršenja tipa boolean, mogu se vezivati u složenije izraze pomoću **logičkih operatora** AND, OR, XOR, i NOT.

Operator	Značenje	Primjer
!	Negacija	!true=false
&&	AND	true && true = true
| |	OR	false || false = false

Operator razdvajanja (zarez)

- koristi se za razdvajanje izraza u naredbama; izrazi se izvode postepeno sa lijeva na desno

npr. i = 10, i+5; // varijabli i biti će pridružena vrijednost 15

Prioriteti:

1. :: /*područje -scope*/
2. [] /*indeksiranje*/, ( ) /*poziv funkcije*/, ++ /*uvećaj nakon*/, -- /*umanji nakon*/
3. ++ /*uvećaj prije*/, -- /*umanji prije*/, !, +,- /*predznaci*/, new, delete, () /*dodjela tipa*/
4. ->*.* /*pokazivač na član*/
5. *, /, %
6. +, -
7. <, <=, >, >=
8. = =, !=
9. &&
10. ||
11. ?: /*uvjetni operator*/
12. =, *=, /=, %=
13. , /*razdvajanje*/

Strukture za upravljanje tokom procesa

Osnovni konstrukti koji omogućavaju upravljanje tokom procesa izvođenja programa su **selekcija/izbor** (if-else, switch) i **iteracija/ponavljanje** (for, while, do..while).

1. Naredba if

```
if  (logički izraz)                    // ako je logički izraz istinit (true)
        blok_naredbi1;                 // izvodi se blok_naredbi1
[else                                  // ako je definiran else i izraz je neistinit
        blok_naredbi2];                // izvodi se blok_naredbi2
```

gdje je:

blok_naredbi1 = naredba | { naredba_1; naredba_2; …. naredba_n;}

Kombinacija if - else -if

```
if  (logički izraz) {
        …                              // kada treba provjeravati više uvjeta
   }                                   // preporuka je koristiti naredbu swich
else if  (logički izraz) {             // else pripada najbližem if koji nema svoj else
        …
   }
else if  (logički izraz) {
        …
   }
```

Uvjetni (ternarni) operator (Conditional operator) - ?:
(svojevrsni skraćeni oblik naredbe **if)**

izraz ? izraz-je-istini : izraz-nije-istinit

Na primjer,

```
int manji = (x < y) ? x : y;       /* ako je (x < y) istina, varijabli manji dodjeljuje se vrijednost x;
                                      u suprotnom, toj se varijabli dodjeljuje vrijednost y. */
```

2. Naredba switch

```
switch (cjelobrojni_izraz) {       // Izraz može sadržavati samo cjelobrojne tipove
case  konstanta_1:                 // Vrijednost izraza uspoređuje se sa svakom od konstanti
        [naredba_1;]               // ako se podudaraju izvodi se odgovarajuća naredba
        [break;]                   // izlazak iz swich, nakon }
        ….
case konstanta_n:
        [naredba_n;]
        [break;]
[default:]                         // ako nema podudarnosti niti sa jednom konstantom
        [naredba_m;]               // izvodi se naredba pod default (ako je naveden)
}                                  // ili se izlazi iz switch
```

3. for petlja

Petlja **for** omogućava da se odabrani blok naredbi izvede određeni broj puta.

```
for (inicijalizacija; logički izraz; korak) {
        blok-stavaka
}
```

gdje je :

inicijalizacija - inicijalizacija kontrolnih varijabli | deklariranje i inicijaliziranje kontrolnih varijabli
logički izraz - ako je **false** prekida se izvršavanje petlje
korak - povećava ili smanjuje vrijednost kontrolnih varijabli

Primjer:

```
*   for (int a=1; a<10; a++) {        // kontrolne varijable
        …..
    }
*   for (int a=1, b=5; a<b; a++, b--) {      // dvije kontrolne varijable
        …..
    }

*   int i,n; long faktor;
    for (i=1, fakt=1; i<=n; faktor*=i++);   // od bloka je ostala prazna naredba pa je obavezna ;


*   for (int i=20; i<=20;) {                /* korak može biti izostavljen ali vrijednost kontrolne
        …..                                 varijable se mora mijenjati u petlji */
        i--;
    }

*   for ( ; ;  )                            // može biti bez inicijalizacije, izraza i koraka, samo sa ;
```

4. while petlja

```
while (logički izraz) {          // Logički izraz ispituje se prilikom ulaska u petlju, ako je false
    naredbe;                     // petlja se neće izvesti niti jednom;
}
```

Izvršava se dok je navedeni logički izraz ispunjen (**true)**.

```
Npr.  int i = 5 ,j =20;
      while (i< j) {    // Izvođenje petlje završava se kada uvjet postane false da bi do toga došlo,
          …..
          i++;     // u petlji se moraju mijenjati vrijednosti varijabli koje utječu na ispunjenost uvjeta
       }
```

5. do-while petlja

```
do {
        blok_naredbi;
} while (logički izraz);        /* Ispunjenost uvjeta provjerava se nakon prolaska kroz petlju
                                   petlja se izvodi barem jednom */
```

Ako je po prolasku kroz petlju, uvjet **true**, petlja se izvodi ponovno; u suprotnom, petlja se više ne izvodi.

```
Npr.  int i=1, max =25;
      do {
```

```
        ….;
        i++;
    } while (i < max);
```

Izlaz iz petlje

Naredba **break**

- prekid petlje u kojoj se nalazi i nastavak izvođenja prve slijedeće naredbe/ vanjske petlje

Npr:
```
for (int i=0; i<100; i++)
        for (int j=0; j< 30; j++) {
                if (i==1) break;
        }
```

Naredba **continue**

- prekida tekuće izvođenje petlje i nastavlja izvođenje sa slijedećom iteracijim

Npr:
```
for (int i = 0; i < 10; i++) {
        if (i= = 7)  continue;    // ispisati će sve brojeve od 0 do 10 osim 7
        cout << i;
}
```

Naredba **goto labela;**

- bezuvjetni skok na neku drugu naredbu unutar iste funkcije ispred koje se stavlja labela

Npr:
```
if (i < 0) goto negativan;
 …..
negativan:      // naredba na koju se skače
```

Naredba **return -** koristi se u funkcijama kako bi se tijek programa vratio u onaj dio programa od kuda je funkcija bila pozvana

Polja

- niz konačnog broja istovrsnih podataka

- Jednodimenzionalna polja - niz podataka nekog tipa

Deklaracija:

```
tip_pod     ime_var[n];          // deklarira polje ime_var koji sadrži n elemenata tipa tip_pod
```

Npr.

```
int a[7];                        /* deklariranje varijable a osigurava se prostor u memoriji za pet
                                    podataka tipa int; elementi polja nisu inicirani ! */

int bod[] = {10,20,30,40};       /*deklaracija i inicijalizacija polja bod; duljina polja određena je
                                    inicijalizacijskom listom pa ju ne treba navoditi */
float d[10] = {1.5, 2.5};        /* broj elemenata inicijalizacijske liste može biti manji od duljine
                                    polja; elementi polja bez inicijalizatora postaju jednaki nuli */
```

Elementu niza pristupamo tako da u zagradama navedemo njegov indeks (redni broj); indeks prvog elementa je 0, pa je indeks zadnjeg elementa duljina niza -1;

[4]	Indeks	[0]	[1]	[2]	[3]
int **bod**	Sadržaj polja	10	20	30	40

Svakom pojedinom članu niza može se pristupati i mjenjati ga.

Npr.

```
float a = d[2];                  // pridruživanje vrijednosti trećeg član polja
x[2] = x[1] + x[0];              // aritmetičke operacije nad članovima polja i dodjeljivanje
x[3]++;                          // inkrementiranje četvrtog člana niza
```

- Višedimenzionalna polj - niz nizova

```
int matrica[3][4];               // prvi indeks je broj redaka, a drugi broj stupaca

int matrica[3][4] =  { {1,2,3,4},    // inicijalizacija niza prilikom deklaracije
                       {5,6,7,8}};   // treći redak inicirati će se na nulu
```

[3] [4]	Indeks	[0][0]	[0] [1]	[0] [2]	[0] [3]
int **matrica**	Sadržaj polja	1	2	3	4
	Indeks	[1][0]	[1] [1]	[1] [2]	[1] [3]
	Sadržaj polja	5	6	7	8
	Indeks	[2][0]	[2] [1]	[2] [2]	[2] [3]
	Sadržaj polja	0	0	0	0

Pokazivači (pointers)

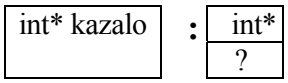
- objekti koji pokazuju na druge objekte
- može se deklarirati tako da pokazuje na bilo koji tip podatka

deklaracija:

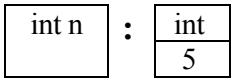
tip_pod *ime_pokazivača; // tip_pod je tip podatka na koji pokazivač pokazuje
tip_pod ime_pokazivača = &var; /* inicijalizacija pokazivača prilikom deklaracije; varijabla **var** na koju se pokazivač inicijalizira mora postojati u memoriji */

npr.

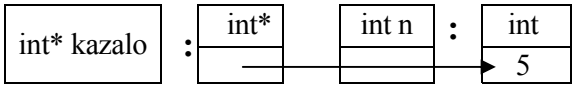
```
int *kazalo;     // deklariran je pokazivač
                 // kazalo na neki broj tipa int
```




```
int n=5;
```




```
kazalo = &n;   // usmjerava pokazivač na
               // memorijsku adresu na
               // kojoj je pohranjena
               // varijabla n
```




```
cout << "*kazalo = " << *kazalo;      // ispisuje 5
cout << "kazalo = " << kazalo;        // ispisuje memorijsku adresu varijable n (npr. 0x219)
```

Operacije sa pokazivačima: sve operacije koje su dozvoljene na cjelobrojnoj varijabli

npr.

```
*kazalo +=5;                  // isto koa n +=5;      (vrijednost od n je sada 10)
n = *kazalo -2;               // isto kao n = n-2;    (vrijednost od ne sada je 8)

kazalo +=5                    // preusmjerava pokazivač; uvećava memorijsku adresu za 5

int m=3; kazalo =&m;          // preusmjeravanje pokazivača na varijablu m

int x =10;  int *px = &x;     // deklaracija i inicijalizacija pokazivača;
float y =25.15; float *py =&y;
*px = *py                     // zbog pravila konverzije n=25;
py = px                       /* pokazivač na decimalnu varijablu preusmjerava se na
py = &x                       adresu cjelobrojne varijable; upitan rezultat - GREŠKA*/
```

Pokazivač tipa void:

```
void *nesvrstan;              // pokazivač tipa void pokazuje na neodređen tip; može se
nesvrstan = &x;               // preusmjeriti na objekt bilo kojeg tipa
nesvrstan = py;

nesvrstan = &x;               // prije ispisa pokazivaču tipa void treba dodjeliti tip
cout << *(int*)nesvrstan;     // pokazivača int (int*)da bi prevoditelj znao pravilno prčitati
                              // sadržaj memorije na koju pokazivač pokazuje
```

Veza pokazivača i polja:

```
npr. float x[5];                    /*deklararira jednodimenzionalno polje od pet članova tipa float;
                                    samo ime x ima smisao pokazivača na prvi član polja x[0] ali u
                                    memoriji nije nigdje alociran*/
npr.    int b[] = {10,20,30};
        cout << b;                  // adresa početnog člana
        cout << *b;                 // vrijednost početnog člana
        cout << b+1;                // adresa člana b[1]
        cout << &(b[1]);            // adresa člana b[1]
cout<<b[2];
cout<<*(b+2);

npr.    float b[] = {10,20,30};
        float a[3];
        *a = *b;                    // pridruživanje prvog člana a[0] = b[0]
```

Pokazivač na višedimenzionalna polja:

```
npr. float a[3][4];                 // može se shvatiti kao niz od tri jednodimenzionalna polja od
                                    // kojih svako sadrži četri elementa
```

a[0], a[1] a[2] su pritom pokazivači na početne članove svakog od tih jednodimenzionalnih polja jer

a[0] je sinonim za a[0] [0]
a[1] je sinonim za a[1] [0]
a[2] je sinonim za a[2] [0]

```
npr.    float a[3][4] = { {1.1, 1.2, 1.3, 1.4},    {2.1, 2.2, 2.3, 2.4},    {3.1, 3.2, 3.3, 3.4}}
        cout << *a[0] <<", \t2 << *a[1] << ",\t" << * a[2];    // ispisati će 1.1, 2.1, 3.1 - poč. članove nizova
```

Aritmetičke operacije sa pokazivačima:

- rezultat zbrajanja ili oduzimanja pokazivača sa konstantom (n) biti će pokazivač koji pokazuje na memorijsku lokaciju udaljenu od početne za nevedeni broj objekata (n) dotičnog tipa.

```
npr.    int a;
        int* poka = &a;                 //pokazivač na int
        cout<< poka << endl;            // memorijska lokacija od a;
        cout << poka - 3 << endl;       // pokazivač (poka-3) pokazuje na mem. lokaciju koja je za
                                        // 3*sizeof(int) manja od početne
npr.    float x[10];
        float* px = &x[3];              // pokazivač usmjeren na x[3]
        float x2 = *(px-1);             // x[2] - predhodni član
        float x4 = *(px+1);             // x[4] - slijedeći član
```

- dozvoljeno je inkrementiranje i dekrementiranje, i operatri skraćenog pridruživanja
- moguće ih je uspoređivati (uspoređuju se memorijske adrese na koje pokazivači pokazuju)

```
npr.    int n = 9;
        int polje[20]  { 10, 20 , 30, 40, 50, 60, 70, 80, 90};    // inicijalizacija sortirane liste
        int* p = polje + n;                 //pokazivač na lokaciju iza zadnjeg broja
        int novi = 15;
        while (*(--p) > novi) {     //kreće od zadnjeg najvećeg broja i uspoređuje ga sa novim
              if  (p < polje)       // prošao je početak polja pa se zato
              break;                // prekida petlja
              *(p+1) =  *p;         //vrijednost pohranjenu na adresi na koju pokazuje pokazivač p pohranjuje
                                    // u memoriju na adresu koja je za jedan memorijski blok veća od adrese p
        }                           // tj. pomiće brojeve koji su veći od novog
```

```
*(p+1) = novi;                    // umeće novi broj u sortiranu listu
n++;                               // povećava broj članova niza
```