# Zadatak 1.

Napišite program u kojem ćete:

1. Deklarirati dvije varijable tipa float (broj1 i broj2).
2. Inicijalizirati varijablu broj1 vrijednošću 7.3.
3. Deklarirati pokazivač na tip podatka float.
4. Inicijalizirati pokazivač na adresu varijable broj1.
5. Ispisati vrijednost varijable na koju upućuje pokazivač koristeći dereferenciranje pokazivača.
6. Neka pokazivač pokazuje na varijablu broj2. Dodjelite proizvoljnu vrijednost u broj1 preko pokazivača. Ispišite vrijednost broj1 preko pokazivača.
7. Ispisati adresu varijable broj1.

# Zadatak 2.

Napravite program u kome ćete:

1. unijeti dva broja (a i b tipa float).
2. deklarisati pokazivač p1 koji će pokazivati na varijablu a
3. deklarisati pokazivač p2 koji će pokazivati na varijablu b
4. izračunati sljedeće matematičke izraze koristeći pokazivače p1 i p2 (tj. bez korištenja varijable a i b).



# Zadatak 3.

Dat je sljedeći dio koda

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11: | void main()  {  int a;  cout << "Unesite broj: ";  cin >> a;  int\* p1;  p1 = &a;  cout << \*p1;  } |

Prepravite program tako da se dereferenciranje pokazivača p1 i ispis tako dobivene vrijednosti vrši u novoj funkciji void f1. Znači, u funkciji main treba se nalaziti samo varijabla a, dok u funkciji f1 treba se nalaziti samo varijabla (pokazivač) p1, koja prima adresu varijable a iz funkcije main.

|  |
| --- |
| *Pomoć:*   * formalni parametar je: p1 tipa int\* * aktuelni parametar: adresa varijable a |

# Zadatak 4.

Napravite program tako da iskoristite prototip funkcije:

void saberi(int \*, int \*, int \*);

Funkcija treba sabrati vrijednosti prva dva parametra i rezultat smjestiti u treći parametar. U funkciji main trebate nakon poziva funkcije ispisati rezultat sabiranja.

# Zadatak 5.

Prepravite prethodni program (funkciju main) tako da poziv funkcije saberi i ispis vrijednosti sabiranja izvršava sa dodatnim pokazivačem p1 koji pokazuje na varijablu c.

|  |
| --- |
| *Pomoć:* Deklarisat ćemo pokazivač **p1** tako da on pokazuje na varijablu **c**. Neophodno je da treći aktuelni parametar bude pokazivač **p1** - tj.  umjesto **saberi(&a, &b, &c)** trebanapisati saberi**(&a, &b, p1)** jer je p1 == &c.   * Pri pozivu funkcije saberi(&a, &b, &c) vrši se kopiranje (adrese) &c u varijablu (pokazivač) u3, tj. *kao* da se izvršava se naredba u3 = &c * Pri pozivu funkcije saberi(&a, &b, p1)   + ne vrši se kopiranje adrese od varijable p1 u u3, tj.   + pokazivač u3 neće pokazivat na p1, tj. u3 nije *pokazivač na pokazivač*, jer:  1. deklaraciji varijable ne glasi int\*\* u3, nego glasi int\* u3 2. ne izvršava se naredba u3 = &p1, nego se izvršava naredba u3 = p1   Stanje varijabli prije poziva funkcije je dato na narednoj slici:    Nakon izvršenja naredbe u3 = p1 koja se u pozadini izvršava pri pozivu funkcije, sadržaj pokazivača p1 će se iskopirati u u3. To je ilustrovano na slijedećoj slici: |

# Zadatak 6. i 7.

1. Napravite program u kojem će te koristiti funkciju void zamjeni(int\* p1, int\* p2). Funkcija treba zamijeniti vrijednosti varijabli koji su joj proslijeđeni kao aktuelni parametri u pozivu.

|  |  |  |  |
| --- | --- | --- | --- |
| *Pomoć:*  - U funkciji main ćemo deklarisati dvije varijable tipa integer (a=5, b=6)   * + to znači da se na nekoj memorijskoj adresi (pretpostavimo 0x102) rezervisao prostor veličine jednog integera (npr. 4 bajta) i da se memorijski prostor, u koji ćemo smjestit vrijednost 5, dodijelio varijabli a;   + na isti način će se alocirati memorija u RAM-u za varijablu b (pretpostavimo 0x106), to se može grafički prikazati ovako:     - pošto funkcija zamjeni prima dva akuelna parametra u formalne parametre (pokazivači p1 i p2), pri pozivu ove funkcije moramo proslijediti kao prvi akuelni parametar adresu neke varijable (npr: &a) i kao drugi aktuelni parametar adresu neke varijable (npr: &b)   * + u main-u ćemo pozvati funkciju naredbom zamjeni(&a,&b);   + sadržaj varijable (pokazivača) p1 će biti adresa varijable a, tj. 0x102   + sadržaj varijable (pokazivača) p2 će biti adresa varijable b, tj. 0x106   + varijabla (pokazivač) p1 ima, takođe, svoj memorijski prostor u kojem se nalazi adresa varijable a (pretpostavimo za p1: 0x110, za p2: 0x114), ali taj memorijski prostor pokazivača nam nije bitan     - pokazivač p1 pokazuje na a, pokazivač p2 pokazuje na b, to možete zamisliti ovako:     * + sa naredbom cout << &p1; se ispisuje memorijska adresa varijable p1 (pokazivača) tj. 0x110   + sa naredbom cout << p1; ispisujemo vrijednost varijable p1 (pokazivača) tj. u našem slučaju je to memorijska adresa varijable a (0x102)   + sa naredbom cout << \*p1; ispisujemo vrijednost koja se nalazi na memorijskoj lokaciji na koju pokazuje p1, tj. vrijednost 5   + može se zaključiti da vrijede jednakosti (\*p1 == a) i (\*p2 == b)   + u slučaju da smo trebali zamijeniti vrijednost varijabli a i b u funkciji main to bi vršili na slijedeći način:  |  | | --- | | int temp;  temp = a;  a = b;  b = temp; |   - ako bi u funkciji zamjeni zamjenu varijabli (pokušali) vršiti na slijedeći način…   |  | | --- | | int\* temp; //pomoćni pokazivač  temp = p1;  p1 = p2;  p2 = temp; |  * onda bi se samo 'smjer' pokazivača promijenio, ali ne i vrijednosti varijabli a i b * promjena pokazivača nam nije ni od kakve koristi jer će pokazivači ionako 'nestati' nakon završetka funkcije; cilj nam je promijeniti vrijednosti a i b koje će i dalje postojati nakon završetka funkcije * u prethodnom nepoželjnom primjeru imali bi sljedeću situaciju:      * ispravan primjer zamjene vrijednosti varijabli a i b:   + zamjenu vrijednost varijabli a i b, činit ćemo na sličan način kao što bi to činili u funkciji main, ali umjesto a koristit ćemo \*p1 i umjesto b koristit ćemo \*p2:  |  | | --- | | int temp; // sada ovo nije pokazivač  temp = \*p1;  \*p1 = \*p2;  \*p2 = temp; |   *Napomena:*  Obratite pažnju na program sa pokazivačem u funkciji f1 sa stranice 41, gdje su korištene dodatne zagrade za (\*u2)++. U ovom slučaju će se vrijednost na koji pokazuje pokazivač (neki *integer*) inkrementirati. Da smo koristili samo \*u2++; to bi značilo da se vrijednost pokazivača (varijable) u2 inkrementira, tj. adresa će se povećati npr: sa 0x102 na 0x104 (za veličinu *integera*). Navedeno se događa jer operator inkrementa '++' ima veći prioritet nego operator za dereferenciranje '\*'.  U slučaju da nisu korištene dodatne zagrade vrijednost varijable u2 bi se promijenila (inkrementirala), pa bi u2 pokazivao na neku novu lokaciju. To može dovesti do grešaka u radu aplikacije, jer operativni sistem ne bi dozvolio čitanje memorijske adrese koja nije dodijeljena za korištenje vašem programu. |

1. Riješite prethodni zadatak pomoću *reference*.

|  |
| --- |
| *Pomoć:*   * Kod poziva funkcije zamjeni po referenci nećemo prosljeđivati adrese varijabli a i b nego samo vrijednosti varijabli a i b. * Imat ćemo sljedeću situaciju: |

# Zadatak 8.

1. deklarišite pokazivač p1 na *int*
2. alocirajte prostor pokazivaču p1 u dinamičkoj memoriji p1 = new int;
3. učitajte broj sa tastature u prethodno alocirani prostor
4. inkrementirajte prethodno uneseni broj koristeći pokazivač p1 pomoću operatora ++
5. definišite i pozovite funkciju **promjeni\_predznak** čiji će formalni parametar biti pokazivač;

funkcija treba promijeniti predznak broju koji je korisnik unio u koraku **c**

1. deklarirajte varijablu d i neka pokazivač **p1** pokazuje na varijablu d
2. dodijelite varijabli d neku vrijednost koristeći pokazivač **p1**
3. promijenite predznak varijabli d, koristeći funkciju promjeni\_predznak
4. pozovite ponovo funkciju promjeni\_predznak i proslijedite joj kao aktuelni parametar pokazivač p1

# Zadatak 9. - Nizovi

**Napišite program u kojem ćete:**

1. deklarirati niz tipa double koji ima 10 elemenata:

double niz [10]={0.0, 1.1, 2.2, 3.3., 4.4, 5.5, 6.6, 7.7, 8.8, 9.9}

1. provjeriti koliko prostora u memoriji zauzima tip podatka double i koliko prostora u memoriji zauzima cijeli niz (*upotrijebite operator sizeof*)
2. deklarirati pokazivač koji pokazuje na objekt tipa double
3. inicijalizirate pokazivač na adresu prvog elementa niza

*(pokušajte ovu inicijalizaciju napraviti na dva načina)*

1. ispišite 4 element niza na barem 3 različita načina

*(koristite indeksaciju elemenata niza, dereferenciranje pokazivača i operacije nad pokazivačem)*

1. ispišite adrese svih elemenata niza koristeći operator inkrementa na pokazivač

*(vodite računa da ukoliko dekrementirate pokazivač – rezultat je adresa pomaknuta za onoliko bajta koliko zauzima tip podatka na koji upućuje pokazivač)*

1. ispišite elemente niza koristeći dereferenciranje pokazivača.

# Zadatak 10

Napišite program u kojem ćete deklarirari niz od 5 cijelobrojnih elemenata te uz pomoć:

1. funkcije: **void unos (int \*, int);** omogućiti unos elemenata niza;
2. funkcije: **void ispis (int \*, int);** omogućiti ispis elemenata niza;
3. funkcije: **int\* najveci (int \*, int);** vratiti adresu najvećeg elementa u nizu; obavezno provjerite je li funkcija vratila korektnu adresu.
4. funkcije: **int\* arit\_sredina (int \*, int);** vratiti aritmetičku sredinu elemenata u nizu

# Zadatak 11 – Pokazivač na objekat (instancu strukture)

Prepravite naredni program tako da funkcija **doIt** ima sljedeći prototip

void doIt(Trougao\* T1)

|  |
| --- |
| #include <iostream>  using namespace std;  struct Tacka  {  float x;  float y;  };  struct Kruznica  {  Tacka centar;  float poluprecnik;  };  struct Trougao  {  Tacka A;  Tacka B;  Tacka C;  };  void info(Kruznica k)  {  cout << "centar O( " << k.centar.x << ", " << k.centar.y << ")";  cout << "poluprečnik r = " << k.poluprecnik << endl;  }  void doIt(Trougao& T1)  {  cout << "Unesite x i y za tacku A: ";  cin >> T1.A.x;  cin >> T1.A.y;  cout << "Unesite x i y za tacku B: ";  cin >> T1.B.x;  cin >> T1.B.y;  cout << "Unesite x i y za tacku C: ";  cin >> T1.C.x;  cin >> T1.C.y;  //b  Kruznica K1;  K1.centar = T1.A;  //ili  K1.centar.x = T1.A.x;  K1.centar.y = T1.A.y;  //c  cout << "Unesite poluprecnik za K1: ";  cin >> K1.poluprecnik;  //d  Kruznica K3 = K1;  //e  K3.poluprecnik = K3.poluprecnik \* 1.1;  //ili  K3.poluprecnik \*= 1.1;  //f  Kruznica K2 = { T1.B, K3.poluprecnik };  //ili  K2.centar = T1.B;  K2.poluprecnik = K3.poluprecnik;  info(K1);  info(K2);  info(K3);  }  void main()  {  Trougao T1;    doIt(T1);  } |

**Info - Razlika između reference i pokazivača**

Analizirajte slijedeća dva programa.

|  |  |  |
| --- | --- | --- |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15:  16:  17:  18:  19:  20: | #include <iostream>  using namespace std;  void f1(int &);  void main()  {  int a = 6;  f1(a);  cout << a << endl;  }  void f1(int **&**u1) //ovo je referenca  {  u1++;  cout << u1 << endl;  }  // zaključak: a == u1 | #include <iostream>  using namespace std;  void f1(int \*);  void main()  {  int a = 6;  f1(**&**a);  cout << a << endl;  }  void f1(int \*u2)  {  (\*u2)++; // isto kao (\*u2) = (\*u2) + 1;  cout << u2 << endl; // adresa od a  cout << \*u2 << endl; // broj 7  }  // zaključak: a == (\*u2) |

*Važna napomena:*

* znak *ampersand* kod deklaracije varijabli predstavlja referencu (linija br. 14, lijevi primjer)
* znak *ampersand* ispred već deklarisane varijabli predstavlja adresni operator (linija br. 8, desni primjer)

U lijevom primjeru se za varijablu a rezerviše prostor na nekoj slučajnoj memorijskoj lokaciji, pretpostavimo 0x106 (u stvarnosti su memorijske adrese 32 bitne). U memorijsku lokaciju 0x106 se pamti cijeli broj 6. Pri pozivu funkcije prosljeđuje se varijabla a koja se „prima“ kao referenca u1. To znači da varijabla a iz funkcije main i varijabla u1 iz funkcije f1 dijeli isti memorijski prostor. Bilo koja promjena vrijednosti varijable u1 mijenjat će vrijednost varijable a.



U desnom primjeru se, takođe, rezerviše memorijski prostor za varijablu a, ali se ovdje prosljeđuje adresa varijable a (kao aktuelni parametar pri pozivu funkcije). Adresa varijable se u funkciji smješta u pokazivač u2, tj. u2 je formalni parametar. Znači da se za varijablu u2 rezerviše prostor na nekoj slučajnoj memorijskoj lokaciji (ova memorijska adresa nam nije bitna). Sadržaj varijable (pokazivača) u2 je memorijska adresa varijable a, tj. 0x106.

