INSTALACIJA RAZVOJNOG OKRUŽENJA DEV-C++

Duplim klikom miša na program ***dev cpp dem c++*** pokrenuti instalaciju. Odabrati engleski jezik i pritisnuti ***I Agree***. Odabrati punu instalaciju sa ***Full***. Ostalo sve ići sa ***OK*** i ***Next***.

POKRETANJE PRVOG PROGRAMA HELLO WORLD

Nakon instalacije razvojnog okruženja Dev-C++ potrebno je napisati svoj prvi program što je za većinu programera na svijetu bio Hello World:

```
#include <iostream>
using namespace std;
main() {
cout<<"Hello World!!!"<<endl;
}
```

Idemo sa File->New->Source File unijeti gornji tekst, koristiti *Copy-Paste* alat. Radi preglednosti i kasnije lakše upotrebe programe snimiti u poseban direktorij nazvan C++ na podatkovnoj particiji hard diska (npr. D). Potrebno je namjestiti izlazni direktorij na D/C++ u varijablama okoline: Tools/Environment Options/Files&Dir/User's Default Directory mora biti D:\C++\.

Zatim je program potrebno kompajlirati što će odraditi Dev-C++ pritiskom na Ctrl+F9. Kompajler je generirao **`helloworld.exe`** tj. izvršni programski fajl kojeg moramo pokrenuti, a da bi to uradili moramo se nalaziti u našem direktoriju D:\C++\ unutar Windows konzole.

Pokrenuti ***Command Prompt*** sa /*Start->All Programs->Accesories->Command Prompt/* Windows konzolu s kojom ćemo testirati napisane programe. Još samo treba naučiti kako doći do programa koristeći **DOS** komande. Sa **CD..** se izlazi iz nekog direktorija, a sa **D:** npr. ulazi na D particiju hard diska. **DIR** komanda će izlistati sve što se nalazi u nekom direktoriju, a sa **CD *ime direktorija*** ćemo ući u isti. Nakon što smo došli do našeg direktorija D:\C++\ potrebno je pokrenuti program tako što ćemo upisati ime programa npr. helloworld i pritisnuti ENTER. Dolazak na D:\C++\helloworld ide na sljedeći način u Command Promptu (nakon svake komande pritisnuti ENTER):

d:
cd c++
helloworld

Ako je program ispisao redak sa Hello World!!! znači da je sve postavljeno kako treba i da možemo početi sa programiranjem. Prozor *Command Prompt* ostaviti uključen dok god se programira i testiraju programi.

Programski kod mora biti pregledan i poželjno je dodati što tačnije komentare nakon linija koda tako da program mogu razumjeti i drugi, a i sam autor nakon nekog vremena. Komentari se dodaju sa dvije kose linije **`//`**. Sve nakon toga u istom redu kompajler ignoriše. Duži komentari koji zauzimaju više redaka počinju se sa **`/*`**, a završavaju sa **`*/`**. Sad ćemo iskomentarisati programski kod **`helloworld`** u cilju njegovog objašnjenja:

```
#include <iostream>
using namespace std; /* uključivanje koda biblioteke
standardnih ulazno-izlaznih funkcija u naš program */
main() {  // uvijek pozivamo glavnu funkciju main
cout<<"Hello World!!!"<<endl; // endl za novi red
/* ispis poruke na ekran korištenjem funkcije cout koja se
nalazi u biblioteci iostream.h, endl označava kraj linije */
}    // uvijek zatvaramo glavnu funkciju main
```

PROGRAM ZA SABIRANJE DVA BROJA

Uradićemo jednostavan program za sabiranje dva cijela broja x+y=z. Od korisnika ćemo tražiti da unese x pa zatim y i ispisaćemo na ekranu rezultat kao z. Kod je sljedeći:

```
#include <iostream>
using namespace std;
main() {

     int x, y, z;
     cout << "Unesite x!" << endl;
     cin >> x;
     cout << endl;
     cout << "Unesite y!" << endl;
     cin >> y;
     cout << endl;
     z = x + y;
     cout << "Zbir z=" << z << "." << endl << endl;
}
```

Prvo smo uključili **iostream** koja sadržava gotove funkcije za unos podataka sa tastature i ispis podataka na ekran. Zatim smo deklarirali tri varijable **x,y** i **z** kao **integer** (cijeli broj). Potom od korisnika tražimo da unese **x** koju sa **cin** funkcijom dodjeljujemo varijabli **x**. Sa funkcijom **endl** pravimo jedan red razmaka i tražimo od korisnika da unese **y**, te se stvar ponavlja. U retku **z=x+y** pridružili smo varijabli **z** zbir varijabli **x** i **y** te ćemo je zatim ispisati na ekran. Zapamtite: **=** je operator pridruživanja, a ne jednakosti !!!
Obratiti pažnju na sintaksu za ispis: **cout<<"Zbir z="<<z<<"."<<endl;** kako se kombinira poruka u slovima pod navodnicima **"poruka"**sa varijablom iz programa unutar znakova za manje i veće **<< varijabla >>**. Vrlo jednostavno možemo kombinirati jedne sa drugim unoseći razmake i znakove interpunkcije.

DOMAĆA ZADAĆA

Za domaću zadaću instalirati *Dev-C++* razvojno okruženje. Program je besplatan i može se skinuti na **www.bloodshed.net**. Zatim podesiti kompajler te napisati i pokrenuti program helloworld u obliku hello***tvojeime***. Nakon toga uraditi par programa koji obavljaju osnovne računske operacije nad cjelim brojevima koje su prethodno zatražili od korisnika da ih unesu i ispisuju rezultat na ekranu. Ispis mora sadržavati kombinaciju poruke i varijabli te mora biti uredan.

UKLJUČIVANJE MAKRODEFINICIJA

Makro je riječ čije sve pojave u izvornom tekstu programa pretprocesor zamjenjuje specificiranim nizom riječi, npr:

```
#define Pi 3.14 // nema ; na kraju !!!
```

će svaki Pi u programu zamjeniti sa vrijednošću 3.14. Ovo je vrlo značajna mogućnost pretprocesora koja nam omogućuje da neke definirane ili konstantne vrijednosti u programu možemo zamjeniti na jednom mjestu, u samom zaglavlju programa, tako da ih ne moramo mjenjati na svakom mjestu u kodu gdje se pojavljuju. Definisani makro se može ukinuti tako da nakon naredbe **#undef Pi** kompajler ne pridodaje više Pi vrijednost 3.14.

NAREDBE U C++ ZA RAČUNANJE MATEMATIČKIH OPERACIJA

Ako se koriste druge osim osnovne 4 matematičke operacije **(+ - / *)** i operator modulus (**%**) u program potrebno je uključiti biblioteku matematičkih funkcija **cmath**.

Uradićemo jednostavan program za stepenovanje broja tj. baze x sa eksponentom y, programski kod je sljedeći: **pow(x,y)** gdje je **x** baza, a **y** eksponent.

```
#include <iostream>
#include <cmath>
using namespace std;

main() {
       float x, y;
       cout << "Unesite bazu:" << endl;
       cin >> x;
       cout << "Unesite exponent:" << endl;
       cin >> y;
       x = pow(x, y);
       cout << "Rezultat je: " << x << “.“ << endl;
       }
```

Napomena: Koristite tačke umjesto zareza kad unosite vrijednosti u program.

DOMAĆA ZADAĆA:

1. Matematičke izraze napisati u obliku prihvatljivom za C++:

a) $y = (\frac{a}{b})^{n-1}$

b) $y = \frac{ax^2 + d}{3b} - \frac{ab}{c+2} - 1$

c) $y = \frac{\frac{ax-5}{3x} + \exp(4x+8)}{2+\sin(x)} + b$

2. Ispitati rad modulus (**%**) operatora uvrštavanjem sljedećih izraza u program:

a) **9%2**

b) **1%2**

c) **4%2**

OPERATORI

U C++ izrazi igraju važnu ulogu pa C++definiše veći broj operatora od većine drugih jezika. Operatori obavljaju operacije nad operandima, i mogu biti sljedećeg tipa:

Aritmetički

+ (sabiranje) **-** (oduzimanje) * (množenje) **/**(djeljenje)

% (operator modulus daje ostatak pri djeljenju, npr. **9%2** će dati **1**)

Relacioni

> >= < <=

= = (jednako) **!=** (nije jednako)

Logički

p	q	AND p&&q	OR p||q	NOT !p
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Bitski (operatori nad bitovima)

& AND
| OR
^ XOR
~ komplement
>> pomjeranje udesno
<< pomjeranje ulijevo

Inkrement i dekrement

++a ili **a++** za inkrement (uvećanje za 1 ili **a+1**)
--a ili **a--** za dekrement (umanjenje za 1 ili **a-1**)

NAREDBE U C++ ZA GRANANJE TOKA PROGRAMA

if

Naredba **if** omogućava uvjetno grananje toka programa ovisno o tome da li je ili nije zadovoljen uvjet naveden iza ključne riječi **if**. Najjednostavniji oblik naredbe za uvjetno grananje je:

```
if (logički izraz)
     {
     blok naredbi;
     }
```

Ako je vrijednost izraza iza riječi **if** logička istina **(true)**, izvodi se blok naredbi koje slijede iza izraza. U protivnom se taj blok preskače i izvođenje nastavlja od prve naredbe iza bloka. Na primjer:

```
if (a<0)
     {
     cout << „Broj a je negativan!“ << endl;
     }
```

Ako želimo da se neovisno o rezultatu izraza u **if** uvjetu izvode dva nezavisna programska dijela, primjenićemo **if – else** oblik uvjetnog grananja:

```
if   (logički izraz)
     {
     blok naredbi;
     }

else
     {
     blok naredbi;
     }
```

Kod ovog bloka, ako izraz u **if** uvjetu daje kao rezultat logičku istinu izvešće se prvi blok naredbi. Po završetku bloka, izvođenje programa se nastavlja iza **else** bloka. Ako izraz daje logičku neistinu **(false)** preskače se prvi blok iza **if** i izvršava se onaj pod **else** nakon čega program nastavlja izvođenje naredbi koje slijede.

Zadatak: Napisati program koji će ispitati da li je uneženi broj cijeli broj manji od nule, jednak nuli ili veći od nule, te rezultat kao poruku ispisati na ekran.

if – else if - else

Blokovi **if** mogu se nadovezivati na sljedeći način:

```
if        (prvi logički izraz)
          {prvi blok naredbi}

else if   (drugi logički izraz)
          {drugi blok naredbi}
```

```
else if   (treći logički izraz)
          {treći blok naredbi}
...

else      {zadnji blok naredbi}
```

Ilustriraćemo upotrebu ove strukture primjerom u kojem tražimo kakvi su korijeni kvadratne jednačine ispitivanjem diskriminante:

```
#include <iostream>
using namespace std;

int main()
{
float a,b,c;

cout << "Unesite koeficijente kvadratne jednačine:"
     << endl;
cout << "a=";
cin  >> a;
cout << "b=";
cin  >> b;
cout << "c=";
cin  >> c;

float D=b*b-4*a*c;  // određivanje diskriminante

cout << "Jednačina ima ";     // ispis prvog dijela poruke

/* ispis drugog dijela poruke ovisno o uvjetu koji
diskriminanta ispunjava */

if        (D==0)
          {cout << "dvostruki realni korijen." << endl;}

else if   (D>0)
          {cout << "dva realna korijena." << endl;}

else
          {cout << "dva kompleksna korijena." << endl;}
}
```

for petlja

Tamo gdje se ponavljaju djelovi koda koristimo **for** petlju u sljedećem obliku:

```
for  (početni izraz; uvjet izvođenja; izraz prirasta)
     {blok naredbi}
```

Početni izraz je najčešće početna vrijednost brojača, uvjet izvođenja mora biti istinit da bi se izveo blok naredbi, a na kraju se računa izraz prirasta tj. povećanje ili smanjenje brojača. Ilustriraćemo upotrebu **for** petlje kroz sljedeći primjer, program koji računa sumu svih

parnih brojeva od 1 do nekog broja N uključujući i taj broj. Da li je broj paran ili ne određujemo tako što cijeli broj **i** ispitujemo sa aritmetičkim operatorom modulus **%** za ostatak pri djeljenju sa 2 koji mora biti 0 za svaki paran broj.

PROGRAM KOJI UČITAVA PRIRODNI BROJ N I RAČUNA SUMU SVIH PARNIH BROJEVA OD 1 DO N UKLJUČUJUĆI I N

```
#include <iostream>
using namespace std;

main()
{

int N, i;
int k=0;        /* varijabla k se mora inicijalizirati, u
                protivnom možemo imati grešku u programu */

cout << "Unesite N!" << endl;
cin >> N;
cout << endl;

for(i=0; i<=N; i++)
                {
                if(i%2==0) {k=k+i;}
/* ako je ostatak pri djeljenju sa 2 jednak 0 broj je paran pa
uvećavamo rezultat k za i - broj koji se obrađuje u tom
momentu u petlji; rezultat k ispisijemo na izlasku iz petlje
*/
                }
cout << "Rezultat je: " << k << "." << endl;
}
```

do-while petlja

- se koristi za izvršavanje određenog bloka naredbi sve dok se određeni uslov ne izvrši, sintaksa je:

```
do
     {
     blok naredbi;
     }
while (logički izraz);
```

DOMAĆA ZADAĆA: Napisati program koji će tražiti od korisnika da pogodi neko slovo i ispisivati tu poruku na ekran sve dok korisnik ne pogodi to slovo. Ulazni podatak deklarirati kao **`char x = 'b';`** gdje je **b** slovo koje se pogađa. Kad korisnik pogodi slovo ispisati prikladnu poruku.

switch – case struktura za kontrolu toka programa

- koristi se umjesto **if–else if-else** strukture u situacijama gdje postoji više od tri različita rezultata ispitivanog logičkog iskaza. Općenita sintaksa izgleda ovako:

```
switch (cjelobrojni_izraz) {
case: konstantan_izraz1:     // prvi blok naredbi
case: konstantan_izraz2:     // drugi blok naredbi
```

```
break;
case: konstantan_izraz3:
case: konstantan_izraz4:      // treći blok naredbi
break;
default:                      // četvrti blok naredbi
}
```

Upotrebu **switch-case** strukture vidjećemo u sljedećem programu:

```
// mjesec.cpp
#include <iostream>
using namespace std;
main () {
int mjesec;
cout << "Unesite broj 1 do 12 za mjesec: ";
cin >> mjesec;
switch (mjesec) {
  case 1:{cout<<"januar";
  break;}
  case 2:{cout<<"februar";
  break;}
  case 3:{cout<<"mart";
  break;}
  case 4:{cout<<"april";
  break;}
  case 5:{cout<<"maj";
  break;}
  case 6:{cout<<"juni";
  break;}
  case 7:{cout<<"juli";
  break;}
  case 8:{cout<<"avgust";
  break;}
  case 9:{cout<<"septembar";
  break;}
  case 10:{cout<<"oktobar";
  break;}
  case 11:{cout<<"novembar";
  break;}
  case 12:{cout<<"decembar";
  break;}
  default:{cout<<"pogresan unos";
  break;}
}
}
```

UGRAĐENI ILI OSNOVNI TIPOVI PODATAKA U C++

Tip	Značenje	Širina u bajtima	Opseg
char	Karakter (slovo)	1	-128 do 127
int	Cijeli broj	2	-32768 do 32767
long int	Cijeli broj	4	-2147483648 do 2147483648
float	Broj u pokretnom zarezu (7 decimala)	4	3.4e-38 do 3.4e+38
double	Brojevi u pokretnom zarezu dvostruke preciznosti (15 dec.)	8	-1.7e-308 do 1.7e+308
long double	Brojevi u pokretnom zarezu dvostruke preciznosti (18 dec.)	10	3.4e-4932 do 3.4e+4932

Ako je prije tipa navedemo **unsigned** npr. **unsigned char** to znači da radimo samo sa pozitivnim brojevima pa se opseg povećava za 2 puta jer bit za predznak koristimo za prikaz podatka, pa bi npr. **unsigned char** imao opseg 0-255 umjesto -128 do 127.

Da ispitamo koliko memorije zauzimaju pojedini tipovi na računaru, što zavisi od arhitekture računara koji koristimo, primjenićemo unarni operator **sizeof()**. Na primjer:

```
float f;
cout << sizeof(f) << endl; // dužina tipa float
```

ZADATAK: Ispitati sve osnovne tipove podatka koristeći **sizeof()**.

OPERATOR DODJELE TIPA `static_cast`

Probajmo sljedeći jednostavni primjer djeljenja dva broja izvesti na računaru:

```
#include <iostream>
using namespace std;
main() {
int x = 1;
int y = 2;
float z = x / y;
cout << z << endl;
}
```

Očekujemo da će program ispisati tačan rezultat **`0.5`** jer smo isti deklarirali kao **`float`**, međutim program je podjelio dva cijela broja prije ispisa rezultata i pridružio cjelobrojni rezultat tipu **`float`** pa je isti **`0`** što je netačno, tako da smo ustvari zakasnili sa deklaracijom. U ovom slučaju bilo je potrebno promijeniti tip podatka varijabli koje će se dijeliti iz **`int`** u **`float`** što činimo sa operatorom dodjele tipa **`static_cast`** na sljedeći način:

```
#include <iostream>
using namespace std;

main() {
int x = 1;
int y = 2;
float z = static_cast<float>(x) /
static_cast<float>(y);
cout << z << endl;
}
```

Opći oblik primjene operatora **`static_cast`** je: **`static_cast<tip>(izraz)`**. Bitno je napomenuti da varijable nakon upotrebe ovog operatora u programu i dalje ostaju tipa **`int`**. Ako ovaj operator više puta koristimo u programu nad istim podacima znači da tipovi podataka za te varijable nisu dobro projektovane u tom programu.

UGNJEŽĐAVANJE FOR PETLJI

Ugnježđene **for** petlje se vrlo često koriste pa ćemo pokazati kako na sljedećem primjeru programa koji računa i ispisuje tablicu množenja:

```
#include <iostream>
#include <iomanip>
using namespace std;

main() {
       for (int red=1; red<=10; red++) {
           for (int kolona=1; kolona<=10; kolona++)
           cout << setw(5) << red*kolona;
           cout << endl;
       }
}
```

Za svaku vrijednost vanjskog brojača „red“ izvodi se cjelokupna unutrašnja petlja „kolona“ :
red=1;
kolona=1, 2, 3, 4, 5, 6, 7, 8, 9, 10.
red=2;
kolona=1, 2, 3, 4, 5, 6, 7, 8, 9, 10.
itd...

U program je uključena biblioteka **iomanip** koja posjeduje korištenu funkciju **setw()** za razmak između brojeva pri ispisu, parametar unutar zagrada određuje širinu razmaka.

FUNKCIJE U C++

Funkcije se prave da se izbjegne ponavljanje koda i smanji mogućnost greške. Funkcije omogućavaju da se program "razbije" na više manjih djelova. Programski kod postaje razumljiviji i pregledniji. Opći oblik deklaracije ili zaglavlja funkcije (prototip) je sljedeći:

```
izlazni_tip    ime_funkcije(parametar1, parametar2, ...)
```

Funkcija se deklariše poslije zaglavlja **`#include`** i definicija **`#define`**, a prije poziva funkcije **`main()`**. Tijelo funkcije se stavlja ispod završetka funkcije **`main }`**. Uvođenjem funkcija programi dobijaju sljedeći oblik:

```
#include ...
#define ...

deklaracija funkcija koje će se pozivati u main

main{
... programski kod
poziv funkcije ...
... programski kod
... programski kod
poziv funkcije, itd...
}

definicije (tijela) funkcija koje su se pozivale u main
```

U definiciji funkcije je sadržan programski kod (tijelo) funkcije, ono izgleda ovako:

```
izlazni_tip ime_funkcije(parametar1, parametar2, ...)
{
deklaracije lokalnih varijabli;
linije koda koji se izvršava unutar funkcije;
return izlazna_varijabla;
}
```

Ako nema izlazne varijable izlazni tip je definiran kao **`void`**, a **`return`** poziv na kraju se ne navodi. Funkcije se mogu pozivati po vrijednosti i po referenci. Varijable unutar funkcije su lokalne za funkciju što znači da nisu vidljive van funkcije, nastaju pozivom funkcije, a nestaju kad se izvrši **`return`**.

```
// Primjer poziva funkcije po vrijednosti – faktorijel
#include <iostream>
using namespace std;

double fakt(int n);      // deklaracija funkcije
// može samo int bez n ali je razumljivije ovako
main() {
  int n;
  cout << "Unesite broj manji od 171:" << endl;
  cin >> n;    // 171 već unosi preljev za tip double
  cout << "Faktorijel od " << n << " je: " << fakt(n) << endl;
} // fakt(n) je poziv funkcije, predan joj je unešeni n
```

```
double fakt(int n) {          // tijelo funkcije
  double rezultat=1;          // obavezna inicijalizacija
  for (int i=1; i<=n; i++)
  {rezultat*=i;}
  return rezultat;            // rezultat izlazi iz funkcije
}
```

S obzirom da su varijable unutar svake funkcije lokalne nemoguće je npr. napraviti funkciju za zamjenu dva broja pozivom po vrijednosti. Varijable će se zamjeniti unutar funkcije, a u programu će ostati iste. U ovom slučaju moramo koristiti poziv funkcije po referenci. Ova se funkcija često koristi kao pomoćna funkcija u algoritmima sortiranja kad je potrebno zamijeniti pozicije dva elementa u nizu koji se sortira:

```
void zamjeni(int& x, int& y) // x i y su ulazni parametri
{
int z=x;  // x stavljam u privremeni spremnik z
x=y;      // y stavljam u x
y=z;      // z stavljam u y i zamjena je izvršena
}
```

& *(ampersand)* iza tipa varijable znači da se funkciji prosljeđuje adresa varijable. Ovdje je izlazni tip **`void`** jer funkcija ne vraća ništa, ona samo vrši zamjenu već postojećeg.

ZADATAK: Za sljedeći program napisati funkciju koja će odrediti srednju vrijednost za unešena tri decimalna broja:

```
#include <iostream>
using namespace std;

float srednja3(float a, float b, float c);

main () {
     float a, b, c;
     cout << "Koristite tacke umjesto zareza!" << endl;
     cout << "Unesite a: " << endl;
     cin >> a;
     cout << "Unesite b: " << endl;
     cin >> b;
     cout << "Unesite c: " << endl;
     cin >> c;
     cout << "Srednja vrijednost je: " << srednja3(a, b, c);
     }
```

DOMAĆA ZADAĆA: Koristeći funkciju za faktorijel napisati program za računanje binominalnog koeficijenta prema sljedećoj formuli:

$$\frac{n!}{k!(n-k)!}$$

IZRADA MENIJA PROGRAMA

Zašto se koristi **atoi()** funkcija pri pravljenju menija programa? Probaćemo funkcionalnost ovog menija bez upotrebe **atoi** funkcije u situaciji kad korisnik umjesto broja unese slovo.

```
#include <iostream>
using namespace std;
main() {
int meni; // briše se poslije
//char opcija[40];
cout<<"MENI PROGRAMA"<<endl;
cout<<"1.Kocka"<<endl;
cout<<"2.Valjak"<<endl;
cout<<"3.Kugla"<<endl;
cout<<"4.Izlaz iz programa"<<endl;
cout<<"Izaberite opciju: ";
cin>>meni; // briše se poslije
//cin>>opcija;
//int meni=atoi(opcija);
switch(meni) {
case 1: cout<<"Kocka";
break;
case 2: cout<<"Valjak";
break;
case 3: cout<<"Kugla";
break;
case 0: exit(0);
default: exit(0); // ništa nas ne košta ostaviti default...
}
}
```

Kako radi **atoi()** funkcija? Pretvara string (niz karaktera) u cijeli broj ako je na početku stringa cijeli broj, a ako nije izlaz je **0**. Testirajte je na sljedeće stringove: kd, 1gfd, 125ftz...

```
#include <iostream>
using namespace std;
main()
{
char s[20];
cout << "Unesite string za atoi(): ";
cin >> s;
cout << endl;
cout << "Pretvorena vrijednost stringa a je " << atoi(s);
}
```

Upotrebom **atoi()** funkcije ne može se desiti da umjesto potrebne cjelobrojne vrijednosti u **switch-case** izrazu dobijemo karakter. U kombinaciji sa **case 0: exit(0);** svaki pogrešan unos ili izabrana 0 osigurava izlaz iz programa.

ZAJEDNIČKI PROJEKT „GEOMETRIJSKA TIJELA“

Značaj funkcija možemo vidjeti u ovom jednostavnom projektu koji će napraviti grupa učenika. Potrebno je napraviti zaglavlje **`geot.h`** u kojem se nalazi zbirka funkcija za izračun površine i zapremine raznih geometrijskih tijela npr. kocka, prizma, kugla, itd (vidi logaritamske tablice). U glavnom programu napraviti meni (izbornik) koji će na početku izvođenja programa ponuditi korisniku spisak geometrijskih tijela pod rednim brojevima, npr:

1. Kocka
2. Prizma
3. Kugla, itd...
0. Izlaz

Meni rješiti strukturom **`switch-case`**. Odabir se vrši upisivanjem broja tijela (1, 2, 3, itd), za **`default`** slučaj i **`case 0`** koristiti funkciju **`exit(0)`**. Koristiti funkciju **`atoi()`** pri izradi menija za osiguranje od pogrešnog unosa. Meni staviti u beskonačnu petlju. Unutar pojedinih slučaja u meniju pozivaju se odgovarajuće funkcije za to tijelo, npr. **`case 1: Kocka();`**. Svaka funkcija traži od korisnika potrebne varijable i vrši izračun, te ispisuje rezultate.

2 učenika:
U Dev-C++ napraviti novi projekt, izabrati *Console Application*, ime projekta staviti **`gtijela`**, projekt sadrži **`gtijela.cpp`** i **`geot.h`**. Sve funkcije prikupiti i staviti u zaglavlje **`geot.h`**, uraditi meni unutar **`gtijela.cpp`**. Na kraju projekta kompajlirati i testirati program za svako pojedino tijelo.

Ostali učenici:
Izabrati jedno geometrijsko tijelo (osim kocke) iz logaritamskih tablica i uraditi za njega odgovarajuće funkcije za izračun površine i zapremine. Funkcije imenovati na sljedeći način, npr. za kocku: **`Kocka`**. Ulazni tip podataka za sve funkcije je **`void`** jer funkcije ne vraćaju ništa u glavni program, pozivaju se bez argumenata, znači i argumenti su tipa **`void`**.

```
// gtijela.cpp
#include <iostream>
#include "geot.h"
/* navodnici govore kompajleru da trazi tu datoteku u istom
direktoriju gdje je gtijela.cpp - naš radni direktorij C++ */
using namespace std;

extern void Kocka(void);
extern void Valjak(void);
extern void Kugla(void);
/* ovdje se dalje dodaju prototipovi funkcija, extern govori
kompajleru da je funkcija definirana u nekoj vanjskoj
datoteci,
kod nas je to geot.h */

main () {
     char s[20];
     cout << "PROGRAM ZA IZRACUN POVRSINE I ZAPREMINE
     GEOMETRIJSKIH TIJELA" << endl;
     cout << "1. Kocka" << endl;
     cout << "2. Valjak" << endl;
     cout << "3. Kugla" << endl;
     cout << "0. Izlaz" << endl;
```

```
     // ovdje se dalje dodaje listing geometrijskih tijela
     for(;;){  // meni je unutar beskonačne petlje
     cout << "Izaberite opciju: ";
     cin >> s;
      int meni=atoi(s);
     switch (meni) {
          case 1: Kocka();
          break;
          case 2: Valjak();
          break;
          case 3: Kugla();
          break;
          case 0: exit(0);
          break;
  // ovdje se dalje samo dodaju opcije menija...
          default: exit(0);
     }
     }
}

// geot.h
#include <cmath>
using namespace std;

void Kocka(void){
float a;
cout<<"Unesite duzinu stranice a: ";
cin>>a;
float P = 6 * a *a;
float Z = pow(a, 3);
cout<<"Povrsina kocke je: "<<P<<endl;
cout<<"Zapremina kocke je: "<<Z<<endl;
}

void Valjak(void){cout<<"Funkcija-za-valjak"<<endl;}
void Kugla(void) {cout<<"Funkcija-za-kuglu"<<endl;}

// ovdje se dalje samo dodaju funkcije za druga tijela...
```

JEDNODIMENZIONALNI NIZOVI U C++

Niz (*engl. array*) je lista promjenljivih istog tipa na koje se može ukazivati preko istog imena. Pojedinačna promjenjljiva u nizu naziva se elemenat niza. Opšta forma deklaracije jednodimenzionalnog niza u C++ je:

```
tip_promjenljive ime_niza[veličina_niza];
```

Jednodimenzionalni niz od 5 cijelih brojeva deklarisaćemo na sljedeći način:

```
int niz[5];    // niz sadrži 5 cijelih brojeva
```

U C/C++ je definisano da prvi elemenat u nizu ima indeks 0, što znači da zadnji elemenat prethodno definiranog niza mora imati indeks 4. Indeksi su bitni zbog toga što pristupamo pojedinačnim elementima niza pomoću njih. Npr, ako želimo da dodjelimo vrijednost 56 trećem elementu našeg niza uradićemo to pomoću njegovog indeksa:

```
niz[2] = 56;
```

Niz se može i inicijalizirati pri pisanju programa koristeći sljedeću sintaksu:

```
int niz[] = {15, 6, 12, 3, -1};
```

gdje su unutar vitičastih zagrada inicijalizirane početne vrijednosti niza.

Ako nisu inicijalizirani u toku pisanja programa, prethodno deklarisani nizovi se upisuju u toku izvođenja programa pomoću tastature korištenjem **`cin`** funkcije i **`for`** petlje. Analogno tome, **`cout – for`** bi se koristio za ispis niza.

Sljedeći programski kod omogućava upis i ispis niza od 8 cijelih brojeva korištenjem gore navedenog:

```
// Ime programa: nizunos.cpp
#include <iostream>
#include <iomanip>
using namespace std;

#define N 8 // maksimalni broj elemenata niza

int main()
{
int i, A[N];
cout << "Unesite " << N << "-clani niz: " << endl;
cout<<"(nakon svakog broja pritisnite ENTER)" << endl;
for (i=0; i<N; i++) {cin >> A[i];} // upis niza
cout << "Uneseni niz je: ";
for (i=0; i<N; i++) {cout << A[i] << setw(5);} // ispis
cout<<endl;
}
```

PROGRAM KOJI UTVRĐUJE DA LI JE NIZ OPADAJUĆI ILI RASTUĆI

```
// Ime programa: kakavjeniz.cpp
#include <iostream>
using namespace std;

main()
{
int niz[]={1,2,5,11};       // niz je inicijaliziran

if(niz[1]>niz[0])  // ako je drugi član veći od prvog
     {cout<<"Niz je rastuci."<<endl;}
else
cout<<"Niz je opadajuci.";
}
```

Promijenite vrijednosti elemenata niza da niz bude opadajući i ispitajte ispravnost programa.

DOMAĆA ZADAĆA

a) Napraviti program koji će tražiti od korisnika da unese 5-člani niz cijelih brojeva.
b) Ispisati taj niz s lijeva na desno i obrnuto**.**
c) Zamjeniti prvi i zadnji član niza korištenjem gotove funkcije **`zamjeni`**.

DODJELJIVANJE DINAMIČKE MEMORIJE

Do sada su nizovi kojima smo radili bili inicijalizirani ili je njihova veličina morala biti određena u toku pisanja programa. C++ naravno ima mogućnost dodjeljivanja dinamičke memorije u toku samog izvođenja programa što ćemo vidjeti u sljedećem primjeru gdje se koristi ključna riječ **`new`** za alokaciju dinamičke memorije i koristi se pokazivač (*engl. pointer*) **`*`**:

```
// Ime programa: alokacija.cpp
#include <iostream>
#include <iomanip>
using namespace std;

main()
{
int n;
cout << “Unesite duzinu niza:“ << endl;
cin >> n;
int *niz = new int[n];        // kreiranje niza
for (int i=0; i<n; i++)       // upis
     {
     cout << “Unesi broj:“;
     cin >> niz[i];
     }
cout << “Uneseni niz je: “ << endl << endl; // ispis
for (int i=0; i<n; i++) {cout << niz[i] << setw(5);}
cout<<endl;
}
```

SORTIRANJE ELEMENATA NIZA

Za sortiranje elemenata nekog niza koristićemo najjednostavniji algoritam *selection-sort* koji se još naziva i „izbor uzastopnih minimuma“, a sastoji se iz sljedećih koraka:

1. Pronađi najmanji elemenat u nizu i zamjeni ga sa prvim
2. Pronađi drugi najmanji element u nizu i zamjeni ga sa drugim
3. To ponavljaj dok se niz ne sortira

```
// Ime programa: ssort.cpp
#include <iostream>
#include <iomanip>
using namespace std;

void zamjeni(int& x, int& y); // funkcija za zamjenu
void ssort(int niz[], int n); // funkcija selection-sort

main()
{
int n;
cout << "Unesite duzinu niza:" << endl;
cin >> n;
int *niz = new int[n];
for (int i=0; i<n; i++){
     cout << "Unesite element niza: ";
     cin >> niz[i];
}
cout << "Uneseni niz je: ";
for (int i=0; i<n; i++) {
cout << niz[i] << setw(5);
}
cout<<endl;
ssort(niz, n);
cout << "Sortirani niz je: ";
for (int x=0; x<n; x++){
cout<<niz[x]<<setw(5);
}
}

void zamjeni(int& x, int& y)  // funkcija uzima dvije adrese
{
int z=x; x=y; y=z;
}

void ssort(int niz[], int n)
// funkcija uzima niz i duzinu tog niza
{
int x, y;
for (x=0; x<n; x++)
for (y=x+1; y<n; y++)
          if (niz[x]>niz[y]) zamjeni(niz[x], niz[y]);
/* unutar funkcije ssort poziva se funkcija zamjeni, a u
argumentu poziva ne koristimo & nego dajemo dva elementa niza
*/
}
```