Uvod u C++ programiranje

C++ je objektno orjentirani programski jezik kojim su pisani mnogi današnji programi koje srećete u svakodnevnom radu na računalu.

Bez puno okolišanja i priče o samom C++ idemo odmah učiti.

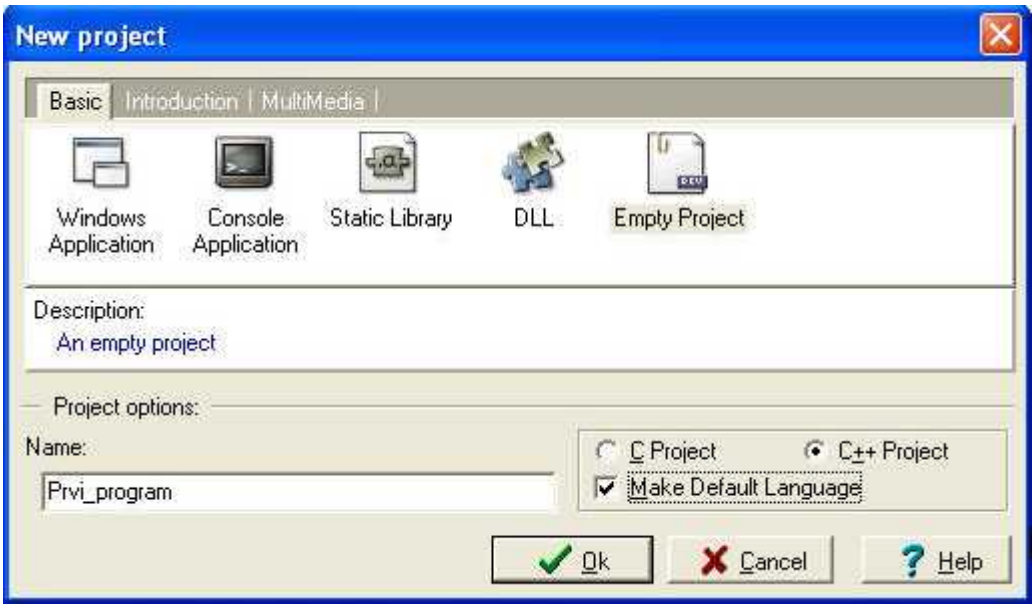
Compilers (kompajleri/prevoditelji)

Kod programiranja u C++ često ćete vidjeti engleske riječi kao naredbe. U tu svrhu nam koriste compileri da bi te naredbe razumljive čovjeku pretvorile u izvedbeni kod.

U izboru imate mnogo compilera za izabrati. Neki od njih se plaćaju (i to dosta novaca) pa vam ja preporučujem besplatno razvojno okruženje i compiler Dev-C++

Prvi projekt

Kada ste instalirali Dev-C++ pokrenite ga i idite na File -> New -> Project. Odaberite "Empty Project" te ga imenujte kao "Prvi_program" (tako bez razmaka). Označite dolje desno C++ Project i uključite kvačicu na "Make Default Language". Kliknite na OK, zatim na Save .



Kada ste to napravili, idite na File -> New -> Source File i kliknite na Yes. S time ste dobili file gdje ćete pisati vaš C++ kod.

Pa napravimo jednostavan "Hello World" program za probu.
Upišite ovaj kod:

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

```
return 0;
}
```

Primjetite da svaka naredba u C++ mora završavati sa sa znakom ;

Sada taj kod treba iskompajlirati. To ćete učiniti tako da pritisnete tipku F9 na tipkovnici. Program će se kompajlirati i pokrenuti.

Program će se pojaviti i odmah nestati tako da nećete uspjeti vidjeti ništa. To možete riješiti tako da između cout << ”Hello World’!’; i return 0; upišete

```
system("pause");
```

ili da program pokrenete iz Command Prompta.

Ovako pomoću system("pause"); zadržavamo program otvorenim na ekranu.

Pa da objasnimo sada dijelove cijelog koda ovog jednostavnog programa.

```
#include <iostream>
```

Program zahtjeva od prevoditelja da u program uključi biblioteku *iostream* koja je standardna ulazno/izlazna biblioteka koja nam omogućuje ispis na ekranu.

NAPOMENA: #include nije naredba u C++ nego se radi o pretprocesorskoj naredbi

```
using namespace std;
```

Svi elementi standardne C++ biblioteke su deklarirani u ovome što piše *namespace* sa imenom std.

```
int main ()
```

Svaki program mora imati ni manje ni više nego jednu main funkciju. Sav kod unutar main zagrada se izvršava. Int pretstavlja Integer (cijeli broj) što govori da će program pri završetku izvođenja programa vratiti cijeli broj.

```
cout << "Hello World";
```

Ovo ispisuje Hello World! na ekran. Cout pretstavlja standardni ispisni tok. Mogli ste taj kod napisati i ovako cout << ”Hello World“ <<endl;
A možete i jednostavno nastaviti rečenicu u novi red tako da napišete

```
cout << "Hello World!" <<endl <<"Ja sam programer";
```

Endl pretstavlja End Line (kraj linije) odnosno ispis u novi red. Tako bi svaka rečenica koju napišete bila u redu ispod.

```
return 0;
```

Tom naredbom glavni program javlja operacijskom sustavu da je program uspješno završen.

Komentari

Kod je potrebno ponekad komentirati kako se nebi izgubili u kodu, ili jednostavno kao potsjetnik na nečega.

Ovako možete komentirati kod bez da taj tekst utječe na izvršavanje programa.

```
/* Ovo je moj prvi program, sa ovim načinom komentiranja
mogu svoje komentare pisati u više redova, tako da tu može
biti svega... */

#include <iostream>
using namespace std;
int main ()
{
cout << "Hello World!"; // komentiram samo jednu liniju

system("pause");
return 0;
}
```

Varijable

U prošlom tutorijalu smo napisali kako ispisati rečenicu na ekran. U ovom tutorijalu ćemo govoriti o varijablama.

Šta su varijable?
Varijabla je mjesto u memoriji rezervisano za pohranu podatka. Svaka varijabla mora biti određena simboličkim imenom i oznakom tipa podatka koji će u nju biti pohranjen. Varijable postoje u svakom programskom jeziku. U varijablama pohranjujete nekakve vrijedosti, brojeve, slova, i sve druge znakove.
Evo primjera kako rečenicu iz prošlog tutorijala upisati u varijablu i zatim pomoć varijable ispisati istu na ekran.

```
#include<iostream>
using namespace std;
main()
{
string recenica;
recenica = "Hello World!";
cout << recenica;

system("pause");
return 0;
}
```

Dakle sa

```
string recenica
```

deklariramo da će varijabla ”recenica” biti string odnosno skup nekakvih znakova. String je tip podatka. O tome više u idućem tutorijalu. Zapamtite samo da se svaka varijabla mora deklarirati prije pridruživanja vrijedosti.

```
recenica = ’’Hello World’’;
```

Sa ovime varijabli ”recenica” pridružujemo vrijednost ”Hello World!”

```
cout << recenica;
```

Sa ovim ispisujemo varijablu. Primjetite da nema navodnika. Ako želite nešto pored toga napisati što nije sadržano u varijabli možete to napisati ovako.

```
cout << recenica << " What’s up?";
```

Dakle prvo ide varijabla ”recenica” bez navodnika, zatim opet stavljamo strelice za ispis i u navodnike pišemo nastavak rečenice. Primjetite razmak između navodnika i slova W u drugoj

rečenici. To smo napravili tako da riječi ne budu spojene jer će se ispisati u isti red. Prisjetite se da ako želite u novi red ispisati nešto možete to napraviti sa ”<<endl” Također u novi red možete ići ako napišete negdje u navodnicima **n**, npr.

```
cout << recenica << "\n What’s up?";
```

Idemo sada napraviti mali program koji će izračunavati dva broja.

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

main()
{
int a, b, rezultat;
a = 7;
b = 12;
rezultat = a + b;
cout << "Zbir ta dva broja je " << rezultat;

system("pause");
return 0;
}
```

Kao što string označava niz znakova, tako int označava *integer* odnosno cijeli broj. Znači sve što će biti pridodano varijablama a, b i rezultat će biti cijeli broj (nikakvi decimalni brojevi, slova, drugi znakovi nego samo cijeli broj).

Primjetite da smo ovdje protiv nestajanja prozora koristili drugu varijablu da ne bi došlo do konflikta.

Deklarirati varijable možete ovako kao u primjeru a možete i svaku posebno na ovaj način:

```
int a;
int b;
int rezultat;
```

Nakon deklariranja varijabli slijedi naravno pridodavanje vrijednosti tim varijablama. Pridodajemo varijabli a broj 7, varijabli b broj 12 a varijabla rezultat će pohraniti onaj zbir koji daju varijable a i b. U ovom slučaju 7 + 12 = 19. Znači varijabli "rezultat" se pridružuje vrijednost 19.

Varijable možete također deklarirati i pridružiti im vrijednost odjednom. Dakle ovako:

```
int a = 7;
int b = 12;
int rezultat = a + b;
```

Probajte sada napisati ovakav program

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

main()
{
int a, b, rezultat;
cout << "Unesite prvi broj: ";
cin >> a;
cout << "Unesite drugi broj: ";
cin >> b;
rezultat = a + b;
cout << "Zbir ta dva broja je " << rezultat;

system("pause");
return 0;
}
```

Dakle, ovo je skoro isto kao i prethodni primjer samo što ovdje imamo

```
cin >> a;
```

Program očekuje da ćete napisati neki broj i kada vi napišete taj broj on će ga spremiti u varijablu "a". Isto tako i dvije linije poslje za varijablu "b".

Signed i Unsigned
Ovo koristimo za varijable sa predznakom i bez predznaka.
Ako stavite

```
unsigned int a;
```

onda varijabli "a" nećete moći pridružiti broj sa predznakom (npr. -12) a ako umjesto unsigned napišete signed ili ostavite bez toga onda ćete moći upisati predznak.

Doseg varijabli

Varijable mogu biti globalne ili lokalne. Globalne varijable su one varijable koje su deklarirane u glavnom tijelu source koda, izvan svih funkcija dok su lokalne varijable one varijable koje su deklarirane u funkciji ili bloku.

Vjerujem da će vam biti jasnije ako vam to objasnim primjerom.

Primjer lokalnih varijabla

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

main()
{
int a, b, rezultat;
a = 2;
b = 5;
rezultat = a + b;
cout << rezultat;

system("pause");
return 0;
}
```

Primjer globalnih varijabli

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

int a, b, rezultat;

main()
{ a = 2;
b = 5;
rezultat = a + b;
cout << rezultat;

system("pause");
return 0;
}
```

Globalnim varijablama se može pristupiti bilo gdje u kodu, čak i u funkcijama dok je lokalnim varijablama moguće pristupiti samo unutar vitičastih zagrada { i }

NAPOMENA: sve varijable **moraju** početi sa slovom ili donjom crticom _ . Ne smiju početi brojevima niti ičim drugim. Osim toga, sve varijable u svom nazivu smiju imati samo slova, brojeve (od drugog mjesta na dalje), i crtice (_), a ne smiju sadržavati razmake, navodnike i slične simbole.

Također bitno je to da vam se varijable ne smiju zvati sljedećim imenima:

asm, auto, bool, break, case, catch, char, class, const, const_cast, continue, default, delete, do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, operator, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_cast, struct, switch, template, this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile,

wchar_t, while, and, and_eq, bitand, bitor, compl, not, not_eq, or, or_eq, xor, xor_eq

Jer su to ključne riječi i operatori jezika C++.

Još jedna napomena: C++ je case sensitive jezik, odnosno osjetljiv na velika i mala slova. Tako da recenica i RecEnica nije ista varijabla.

Kao što smo već rekli, u sljedećem tutorijalu ćemo se pozabaviti tipovima podataka (to vam je ono int, char, string i ostali...)

Tipovi podataka

Ovaj tutorijal se nadovezuje na varijable, jer kako smo u prošlom tutorijalu napisali da *int*, *string* i *char* označavaju tip podakta koji će biti pridružen varijabli. Pa ćemo sada detaljnije objasniti te i ostele tipove podataka.

Kada programiramo, spremamo varijable u memoriju kompjutera, ali kompjuter mora znati što ćemo (kakav tip podatka) spremiti u varijablu. Jedan jednostavan broj neće zauzeti istu količinu memorije kao jedan veliki tekst ili veliki broj, i neće biti interpretirano na isti način.

Memorija kompjutera je organizirana po bajtovima (bytes). Jedan bajt je minimalna količina memorije koju možemo sačuvati u C++. Jedan bajt može pohraniti relativno mali broj podataka. Jedno slovo ili jedan cijeli broj od 0 do 255.

Osim toga, C++ može upravljati mnogo kompleksnijim tipovima podataka koje dolaze grupiranjem bajtova. Kao što su dugački brojevi (long numbers) i slično.

Evo jedne tablice sa osnovnim tj. bitnim tipovima podataka.

Ime	Opis	Veličina	Domet
char	Character - jedan znak ili mali broj	1 bajt	S predznakom od -128 do 127 Bez predznaka od 0 do 255
short int (short)	Short integer - kratki cijeli broj	2 bajta	S predznakom: od - 32768 do 32767 Bez predznaka: od 0 do 65535
int	Integer - cijeli broj	4 bajta	S predznakom: - 2147483648 do 2147483647 Bez predznaka: od 0 do 4294967295
long int (long)	Long integer - dugački cijeli broj	4 bajta	S predznakom: - 2147483648 do 2147483647 Bez predznaka: od 0 do 4294967295
bool	Bolean - istina ili neistina	1 bajt	Istina ili neistina
float	Floating point number	4 bajta	3.4e +/- 38 (7 brojeva)
double	Double precision floating point number	8 bajtova	1.7e +/- 308 (15 brojeva)
long double	Long double precision floating point number	8 bajtova	1.7e +/- 308 (15 brojeva)
wchar_t	Wild character - "divlji" znak	1 bajt	Npr. japansko slovo

Zaboravio sam napisati, Floating Point Numbers su vam brojevi sa pomičnim zarezom.

Konstante (nepromjenjive)

Varijable su promjenjive dok su konstante su izrazi sa fiksnim, nepromjenjivim vrijednostima.

Morate inicirati konstantu kada ju napravite, i ne možete joj prirdružiti novu vrijednost kasnije. Poslije kada je konstanta inicirana njezina vrijednost je nepromjenjiva.

Const float pi = 3.14;

Literal Constants (konkretne konstante)

C++ ima dva tipa konstanti: konkretne i slimbolične (literal i symbolic)

Literal konstanta je vrijednost upisana direktno u program kada god je to potrebno. Npr.

```
int godine = 24;
```

godine je varijabla tipa integer, a 24 je literal konstanta. Ne možete pridružiti vrijednost na 24, i ta vrijednost ne može biti promijenjena.

Symbolic constants (simbolične konstante)
Simbolična konstanta je konstanta koja je pretstavljena po imenu, baš kao i varijabla. Ali nakon što je konstanta inicirana, njezina vrijednost ne može biti promjenjena. Ako imate jednu integer varijablu imenovanu "studenti" i drugu imenovanu "ucionica", možete procijeniti koliko studenata imate, i broj učionica, te ako znate da je 15 studenata po razredu.

```
sudenti = ucionice * 15;
```

U ovom primjeru, broj 15 je Literal konstanta. Kod bi bio puno jednostavniji za čitanje i jednostavniji za održavati ako biste tu vrijednost zamijenili simboličnom konstantom.

```
studenti = ucionice * ucenika_po_ucionici;
```

Ako kasnije odlučite promijeniti broj studenata po ucionice, mozete to uciniti tako da definirate konstantu ucenika_po_ucionici bez potrebe za izmjenama u svakom dijelu koda gdje ste koristili tu vrijednost.

Definiranje konstanti sa #define

Da biste definirali konstantu na sa staromodnim, i lošim načinom unest ćete

```
#define ucenika_po_razredu 15;
```

Primjetite da ucenika_po_ucionici nema konkretni tip (int, char itd.). Svaki puta kad preprocesor vidi riječ ucenika_po_razredu upisat će broj 15 u tekst.

Zbog toga što se preprocesor pokreće prije kompajlera, kompjuter nikad ne vidi vašu konstantu, ono vidi broj 15.

Definiranje konstanti sa const
Iako #define radi, postoji bolji, ukusniji način definiranja konstanti u C++

```
const unsigned int ucenika_por_ucionici = 15;
```

U ovom primjeru se isto deklarira simbolična konstanta imenovana ucenika_po_ucionici, ali ovaj puta ucenika_po_ucionici je napisano kao unsigned int (bez predznaka, cijeli broj)

S ovime imate više za tipkati ali nudi nekoliko prednosti.Najveća razlika je u tome što ova konstanta ima tip i kompajler može prisliti da to bude korišteno po svom tipu.

Aritmetički operatori
Operatori su simboli koji predstavljaju (zamjenjuju) određene funkcije.
Operator je simbol koji predstavlja specifičnu akciju. Već smo kod primjera zbrajanja koristili operator "+". Osim operatora + imamo još nekoliko aritmetičkih operatora.

Evo tablice za pregled operatora.

Operator	Namjena	Primjer	Rezultat
+	**Sabiranje**	5 + 6	11
-	Oduzimanje	7-3	4
*	Množenje	4 * 4	16
/	Dijelenje	12 / 6	2
%	Dijeljenje (s ostatkom)	5 % 2	1

% operator se naziva još i *modulus operator* .

Aritmetički operatori jednako dobro rade sa negativnim brojevima kao i sa pozitivnim, sa iznimkom modulus operatora, rade sa cijelim brojevima jednako dobro kao i sa brojevima sa pomičnim zarezom.

Neki programski jezici imaju eksponent operator, što nije slučaj kod C++. Umjesto toga C++ ima ugrađenu funkciju *pow* koja je definirana u standardnoj biblioteci *cmath*

pow funkcija ima dva arumenta. Prvi argument je baza (glavni broj), a drugi broj je eksponent.

Pogledajmo primjer:

```
#include <iostream>
#include <cmath> // ukljucili smo biblioteku cmath
using namespace std;

int main()
{
double baza, eksponent, rezultat;
cout << "Unesite broj ";
cin >> baza;
cout << "Unesite eksponent ";
cin >> eksponent;
rezultat = pow(baza, eksponent);
cout << "Rezultat = " << rezultat;

system("pause");
return 0;
}
```

Prvo novo što morate primjetiti je to da smo uključili biblioteku cmath. Zatim smo deklarirali broj, eksponent i rezultat kao double tip. (Zbog ogromnih brojeva ako npr. upišete 10 na 10-u). Zatim smo im sa cin >> pridružili vrijednosti, te su te vrijednosti zapravo parametri u funkciji *pow* .

Odnosni (relacijski) operatori

U programskom jeziku C++ kao i u mnogim drugim programskim jezicima postoje odnosni operatori. Takvi operatori se npr. koriste ako želite napraviti program koji će iz baze podataka odrediti osobe sa više od 30 godina i slično.

Evo tih odnosnih operatora:

Operator	Značenje
>	Više od...
<	Manje od...
>=	Više ili jednako
<=	Manje ili jednako
==	Jednako
!=	Različito (nejednako)

Odnosne naredbe

Kao i aritmetički operatori, odnosni operatori su binarni tj. uspoređuju dva operanda. Naredba sa dva operanda i odnosnim operatorom zove se odnosna naredba (eng. *relational expression*.)
Rezultat odnosne naredbe je Boolean vrijednost odnosno istinu ili laž (*true* ili *false*). S ovom tablicom možete vidjeti kako to funkcionira

Odnosna naredba	Vraćena vrijednost
4 == 4	Istina
4 < 4	Laž
4 <= 4	Istina
4 > 4	Laž
4 != 4	Laž
4 == 5	Laž
4 < 5	Istina
4 >= 5	Laž
4 != 5	Istina

U ovoj tablici se koriste konkretne (*literal*) vrijednosti koje ne mogu biti promijenjene. 4 je konkretna vrijednost (konstanta), i ona se ne može mijenjati (moglo bi se mijenjati da je umjesto konstanta koristimo varijable)

Isprobajmo sada ovaj kod koji umjesto konstanti koristi varijable

```
#include <iostream>
using namespace std;
int main()
{
int a = 4, b = 5;
```

```
cout << (a > b) << endl;
cout << (a >= b) << endl;
cout << (a == b) << endl;
cout << (a <= b) << endl;
cout << (a < b) << endl;

system("pause");
return 0;
}
```

Ovaj program će ispisati:
0
0
0
1
1

S time da 0 predstalja laž (*false*) a 1 istinu (*true*).

Logički operatori

Za rad s logičkim podacima, postoje **logičke funkcije**. Logičke se funkcije zapisuju logičkim operatorima.

Logički operatori		
Oznaka operatora	Funkcija	Operator
!	Negacija (unarni operator koji 1 pretvara u 0 i obratno)	NOT
&&	Logički i (binarni operator)	AND
||	Logicki ILI (binarni operator)	OR

Naredba IF

Naredbu IF koristimo onda kada želimo izvršiti neki kod samo ako je vrijednost nekog odnosnog izraza istinita.

Evo primjera:

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

main()
{
string password;
cout << "Unesite password: ";
cin >> password;
```

```
if(password=="G2105Z") {
cout << "Password tocan!"; }

system("pause");
return 0;
}
```

Definirali smo string password, sa cin naredbom upisali ono što korisnik upiše u varijablu password, zatim slijedi provjera sa IF naredbom.

Sintaksa IF naredbe je zapravo:
if(uvjet) { kod koji se izvšrava }

Naravno nije IF naredba ograničena samo na provjeravanje točnosti. Možete koristiti sve odnosne operatore koje smo objasnili u prošlom tutorijalu. Npr:

```
#include <iostream>
using namespace std;
int main()
{
int godine;
cout << "Koliko imate godina? ";
cin >> godine;
if (godine < 18 ) {
cout << "Maloljetni ste!"; }

system("pause");
return 0;
}
```

U svrhu učenja isprobajte sve odnosne operatore. Npr. probajte napisati mali program koji će provjeriti da li je broj djeljiv sa 2 (bez ostatka).

U sljedećem tutorijalu ćemo učiti o IF...ELSE naredbi koja vam pruža mogućnost da uz IF naredbu imate i naredbu koja će izvršiti neki kod ukoliko uvjet nije zadovoljen

If...Else naredba

If...Else naredba je proširenje If naredbe. Kao što smo već rekli u prošlim tutorijalu, If naredba omogućuje izvršavanje nekog koda ukoliko je uvjet zadovoljen, a ukoliko nije neće se izvršiti ništa.

Ako bi smo htjeli napraviti da ukoliko uvjet nije zadovoljen da se izvrši neki drugi kod onda koristimo If...Else naredbu.

Evo primjera kako koristiti If...Else naredbu (koristit ćemo primjere iz prošlog tutorijala)

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

main()
{
string password;
cout << "Unesite password: ";
cin >> password;

if(password=="G2105Z") {
cout << "Password tocan!"; }
else {
cout << "Password netcan!"; }

system("pause");
return 0;
}
```

Kao sto vidite kod je isti kao i za if naredbe, else pa vitičaste zagrade i unutra kod za izvršavanje.

Evo i onog drugog primjera:

```
#include <iostream>
using namespace std;
int main()
{
int godine;
cout << "Koliko imate godina? ";
cin >> godine;
if (godine < 18 ) {
cout << "Maloljetni ste!"; }
else {
cout << "Punoljetni ste!" }

system("pause");
return 0;
}
```

If...Else If...Else naredba

U prošlom tutorijalu smo govorili o IF...Else naredbi. Ovdje ćemo proširiti tu naredbu tako da možemo ispitati više uvjeta. Za to korisimo If...Else-If...Else narebu.

Evo primjera:

```
#include <iostream>
using namespace std;
int main()
{
int godine;
cout << "Koliko imate godina? ";
cin >> godine;
if (godine <= 17 ) {
cout << "Maloljetni ste!"; }

else if (godine>=18 and godine<=39) {
cout << "Punoljetni ste"; }

else if (godine>39 and godine<70) {
cout << "Najljepse godine"; }

else {
cout << "Stari ste"; }

system("pause");
return 0;
}
```

Dakle sa ovom narebom možemo ispitati više uvjeta. U ovom primjeru dosta koristimo odnosne operatore, pa ako niste naučite ih (4 tutorijala ispred ovoga).

Sve je isto kao i naredba if...else samo što ovdje možete više puta staviti naredbu *else if* kako biste ispitali više uvjeta.

else if možete postaviti koliko god hoćete puta, ali ako imate jako puno mogućih uvjeta onda vam preporučam da pogledate sljedeći tutorijal u kojem ćemo vam objasniti kako se koristi *switch* naredba za ispitivanje mogućih uvjeta.

Naredba while

Oba petlja nam sluzi za definiranje ciklusa sa nepoznatim brojem ponavljanja.
Format naredbe je:

Sintaxa:
while (uvjet)
{
naredbe;
}

npr:

```
broj=1;
while (broj<5)
{
broj=broj+1; // proizvoljna naredba
}
```

U ovom primjeru broj je manji od 5 i ova petlja ce se izvršiti, tako da će se povećati za 1. Slijedeći put će broj biti jednako 2 pa ce se petlja opet izvrsiti. Da bi se ,onda kada broj dobije vrijednost 5, petlja
preskočila tj. ne bi se izvršila i izvršio bi se ostatak koda, jer se nije ispunio uvjet koji glasi da broj mora da bude manji od 5.

```
#include<iostream.h>
int main ()
{
int broj=1;
while (broj<5)
{
broj=broj+1; //ovo se moze napisati i broj+=1;
}
cout<<"Ovo je broj: "<<broj<<endl;
return 0;
}
```

Rezultat ovog programa je:

Ovo je broj: 5

Naredba For

Naredba for

Kod while petlje smo imali da inicijlizaciju, uvjet i promjenu vrijednosti imamo na više razlicitih mjesta, dok je kod for petlje to sve sadržano u jednom redu koda tj. izmedu malih zagrada.

Format naredbe:
for (inicijalizacija; uvjet; promjena vrijednosti) naredba;.
ili korištenjem blokova:
for (inicijalizacija; uvjet; promjena vrijednosti) {
naredbe;
}

Uvjet mora biti logicki izraz, dok inicijalizacija i promjena vrijednosti mogu biti bilo kakvi izrazi. Petlja ce se izvršavati dok je uvjet tocan.

Npr:

```
//Brojac unatrag
#include <iostream.h>
int main ()
{
for (int i=10; i>0; i--) {
cout << i << ", ";
}
cout << "PALI!";
return 0;
}
```

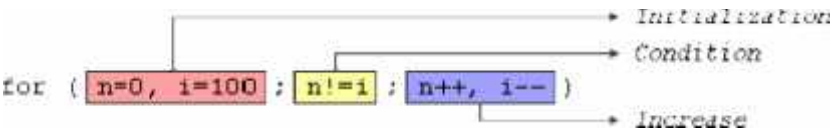
Rezultat ovog programa:
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, PALI!

Ukoliko dodamo na kraju ovog dijela koda “<<endl;”
cout << i << ", "<<endl;
Što ce znaciti da poslije svakog ispisa kursor šalje u novi red.

Nakon ovog rezultat ce biti:

10,
9,
8,
7,
6,
5,
4,
3,
2,
1,
PALI!

Evo na ovoj slici se jasno ilustrira format for petlje:



Naredba switch

Ova naredba sluzi za provjeravanje više uvjeta od jednom i

izvršavanja onoliko radnji koliko je uslova zadovoljeno odnosno tačno.
Ova naredba je korisna ukoliko trebamo provjeriti više "stvari" a da ne petljamo sa if i else if petljom :).

Format ove naredbe je:

switch (varijabla){

case mogucnost1:
naredba;
break;

case mogucnost2:
naredba;
break;

...

}

Pa da svaki red koda objasnimo pojedinacno:

```
switch (varijabla){
```

Ovaj dio prijavljuje varijablu (promjenljivu) čiju ćemo tačnost ispitivati.

```
case mogucnost1:
```

Ovaj dio provjerava da li je varijabla iz prvog reda jednaka nizu znakova (u ovom slucaju) mogucnost1.
Obavezno : (dvotočka) na kraju.

```
naredba;
```

Ovdje upisujemo naredbu ili više njih koje će se izvršavati ukoliko je zadovoljen uvjet.

```
break;
```

Ovom rijecju se zatvara niz naredbi i daje se mogucnost novom uslovu.

```
case mogucnost2:
```

```
naredba;
break;
```

Ovo je samo primjer kako treba postaviti drugi uvjet, tako ćemo dodavati i ostale.
Ukoliko želimo da postavimo neki defaultni uvjet tj. naredbe koje će se izvršiti ukoliko ni jedan uvjet nije zadovoljen onda ćemo dodati slijedeće:

```
default:
naredbe;
```

Ovdje na kraju ne ide break; zato što se podrazumjeva da će default kod biti na kraju tj. zadnji uvjet i da neće
biti više uvjeta.

Ovo sve malo zvuči zamršeno al na konkretnom primjeru će to drukčije (jednostavnije) izgledati.

```
#include<iostream.h>
int main()
{
int broj=0;
switch (broj){

//provjerava da li je broj=1
case 1:
//ako pšostoji ispisuje ’Broj je 1!!!’
cout<<"Broj je 1!!!"<<endl;
break;

case 2:
cout<<"Broj je 2!!!"<<endl;
break;

//Ukoliko broj nije jednak ni 1 niti 2 onda ispisuje
default:
cout<<"Broj je 0 (nula) !!!"<<endl;

}
system("pause");
return 0;
}
```

Evo sad za one koji misle da je lakse sa if petljama ispetljati vise uvjeta:

```
switch (x) {
case 1:
cout << "x je 1";
break;
case 2:
cout << "x je 2";
break;
default:
cout << "x je nepoznato";
}
```

```
if (x == 1) {
cout << "x je 1";
}
else if (x == 2) {
cout << "x je 2";
}
else {
cout << "x je nepoznato";
}
```

Vidimo jednu if i jednu else if petlju zamislite da moramo ispitati 50 vrijednosti tek onda bi bilo petljanja.
Zato toplo preporučujem naredbu switch kod ispitivanja više razlicitih vrijednosti. Dok kod jednostavnih tu je naravno
if i if else. ;)

Ukoliko uočite neku grešku ili vam pak nešto nije jasno javite se na forum.

Jednodimenzionalni nizovni tipovi

Dosadašnji primjeri koje smo radili mogli su da obrađuju samo jednu vrijednost, kombiniranje više vrijednosti jako komplicira rješenje.

Ako imamo 5 različitih varijabli:
a,b,c,d,e i vrijednost svake se unosi sa tipkovnice, to bi izgledalo ovako

```
cout<<"Unesi a: ";
cin>>a;
cout<<"Unesi b: ";
cin>>b;
cout<<"Unesi c: ";
cin>>c;
```

```
cout<<"Unesi d: ";
cin>>d;
cout<<"Unesi e: ";
cin>>e;
```

To predstavlja problem ako moramo unijeti 50 i više vrijednosti.
Zato koristimo jednodimenzionalni nizovni tip. Pa će ovaj gore primjer izgledati ovako:

```
int niz[20],i;

for (i=0;i<=4;i++)
{
cout<<"Unesi vrijednost "<<i+1<<". varijable: ";
cin>>niz[i];
}
```

Pa da objasnimo što radi koji dio koda:

```
int niz[20],i;
```

u ovom dijelu smo deklarirali varijablu niz i
u srednje zagrade smo stavili koliko maximalno mjesta može zauzeti u memoriji (u nasem primjeru 20)

```
for (i=0;i<=4;i++)
```

Pomoću for petlje povećavamo vrijednost varijable "i" svaki put za jedan više. Povećavanje će ići
sve dok vrijednost "i" ne dostigne 4. Zašto 4? Zato što smo htjeli da unosimo 5 varijabli pa ćemo
imati 0,1,2,3,4 to je ukupno 5 brojeva. Mogli bismo staviti i=1 pa bi onda postavili i<=5 ali se u praksi pogotovo kod c++ početna vrijednost brojača postavlja na 0 dok je kod Pascala od 1 itd.

```
cout<<"Unesi vrijednost "<<i+1<<". varijable: ";
```

Kako se varijabla "i" bude povećavala tako će i ispis ovog koda biti drugačiji.
Npr: za i=0 ispis će biti "Unesi vrijednost 1. varijable: ".
sad vidimo opet da piše 1., sigurno se pitate zašto nije 0.
Ovako...Postavili smo i+1 zato što će na početku "i" biti 0 i plus ono jedan=1 i zato je 1. ;)

```
cin>>niz[i];
```

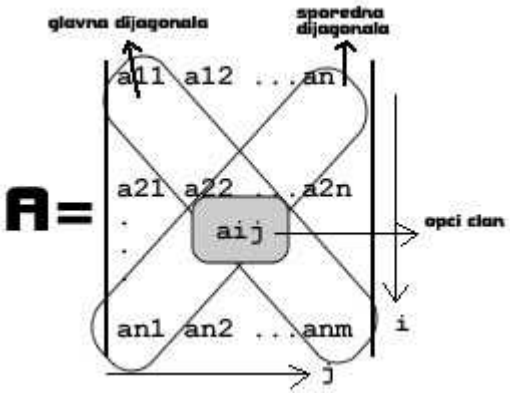
Jednostavna naredba kojom unosimo niz brojeva.

Dvodimenzionalni nizovni tip

DVODIMENZIONALNI NIZOVI - MATRICE

Najefikasniji način predstvljanja podataka je u obliku tabela ili matrica.
Ovi zadaci u programu se predstavljaju kao dvodimenzionalni niz.
Matricu tj. tabelu možemo predstaviti na slijedeći način:

SLIKA:



Sa indeksom j broje se elementi u redu, a sa i u koloni.

i=1,n - i ide od 1 do n
j=1,m - j ide od 1 do m

Elementi kod kojih su indeksi jednaki nazivamo glavnu dijagonalu,
nasuprot glavne dijagonale nalazi se sporedna dijagonala.

n=3
m=3
i=1
j=2

a11 - a12 - a13

a21 - a22 - a23

a31 - a32 - a33

Dvodimenzionalni niz ili matrice deklarisemo u vidu niza

npr:
int niz[20], i, j;

kod ispisa koristimo niz[i][j];