c++	Značenje
Uvod	
// komentar /* komentar */	komentar (može biti u jednom redu ili između zagrada)
#include <iostream> using namespace std;	početak programa

int main()
{
naredbe
}
	glavni (izvršni) dio programa

cout<<varijabla;	ispis varijable sa položajem kursora iza nje
cout<<"tekst"<<endl;	ispis teksta sa položajem kursora u novom redu
<<	znakovi koji nam omogućavaju da u naredbi pisanja povezujemo kombinacije različitih varijabli i drugačijeg teksta
+ - *	sabiranje, oduzimanje i množenje
/* u c++ treba voditi računa da bar jedan od dva djeljena broja bude realan broj. U protivnom, kao rezultat djeljenja dva cijela broja, ćemo dobiti cijeli broj (biće izvršeno cjelobrojno djeljenje). Dva cijela broja je moguće podjeliti da kao rezultat dobijemo realan broj i tu je potrebno samo iza jednog broja staviti tačku tj.: *brojA. / brojB* */	
brojA / brojB brojA**.** / brojB	djeljenje dva broja
Promjenjive	
	u c++ jedino je ograničenje da se promjenjiva mora deklarisati prije nego se upotrijebi
tip imeVarijable;	deklaracija varijable
tip imeV1, imeV2;	deklaracija dvije varijable
tip1 imeV1; tip2 imeV2, imeV3;	deklaracija tri varijable koje mogu biti različitog tipa
int float long double char	tipovi varijabli
cin>>varijabla;	naredba čitanja (u jednom ili dva oblika)
>>	znakovi koji nam omogućavaju da u naredbi čitanja povezujemo više varijabli
imeVarijable = izraz;	znak dodjeljivanja

i+=j; k++; i=j++;
a-=b; ++k;
 	
c*=d; k--; i=++j;
e/=f; --k;
	ekvivalenti i upotreba znaka dodjele
	
	povečavanje i smanjivanje vrijednosti varijable k za jedan
	
	ekvivalenti i slične upotrebe znakova dodjele

Složenije računske operacije	
/* za korištenje matematičkih funkcija u c++ treba, na samom početku programa, uključiti math.h biblioteku, tj. treba napisat: #include <math.h> */	
pow(x,y)	x^y (x na y)
exp(n)	e^n (e na n) e=2.718281828459045240
pow(x,2)	x^2 (x na kvadrat)
sqrt(m)	drugi korijen broja m
log(a)	logaritam broja a po bazi e
log10(x)	logaritam broja x po bazi 10
abs(a)	apsolutna vrijednost broja a
int(x)	odsjeca decimalni dio broja x

sin(ugla) cos(ugla) tan(ugla)	asin(ugla) acos(ugla) atan(ugla)	trigonometriske funkcije (sinus, kosinus, tangens, arkussinus, arkuskosinus i arkustangens) koje daju ispravan rezultate za ugao u radijanima

atan(1)*4
radijan=(atan(1)*4) / 180;
ugao*=radijan;
		vrijednost broja pi (pi=3.141592653589793240)
		vrijednost jednog radijana (1rad = pi / 180)
		pretvaranje ugla koji je u stepenima u ugao radijana (da bi se mogle ispravno primjeniti trigonometrijske funkcije)

/ %		računske operacije koje se upotrebljavaju samo na cijelim brojevima: cijeli dio djeljenja dva broja (cjelobrojno djeljenje) ostatak pri djeljenju (modul)
<u>Ciklične strukture (petlje)</u>		
Naredbe koje su skoro izašle iz upotrebe: naredba goto i naredba case (naredba goto se treba izbjegavati pod svaku cijenu)		
... goto imeLabele; ... imeLabele: naredba ...	/*ili*/	... imeLabele: naredba ... goto imeLabele; ...	dva načina upotrebe naredbe goto

switch(varijabla)
{
case konstanta:
naredba
...
}
		opšti oblici case naredbe

Ostale naredbe ponavljanja		
== <	!= >=	> <=	relacioni operatori (znakovi jednakosti, različito, veće od, manje od, veće ili jednako i manje ili jednako)

{
...
}
		početak i kraj neke cjeline ili bloka (može i tijela petlje)

do
{
...
}
while (uvjet);
		naredba do za c++

break;		naredba za izlaženje iz petlji ponavljanja

while(uvjet)
{
...
}
		naredba while

for (inicijalizacija; uvjet; izmjena)
{
tijeloPetlje
}
 	
for (i=10; i>=1; i--)
cout<<i<<endl;
		naredbe for
		
		/* primjer ispisuje brojeve od 10 do 1 unatraške */

<u>Strukture grananja</u>		

if (uvjet)
naredba
else
{
naredbe
}
		naredba grananja if

randomize();
a=random(1000);
/* za korištenje funkcije random() potrebno je uključiti stdlib.h biblioteku, tj. treba napisati:
#include <stdlib.h> */
	uključivanje generatora slučajnih brojeva
	broj a će dobiti vrijednost iz intervala {0,1,...,999}

&& || !(izraz)	konjukcija, disjunkcija i negacija
Nizovi kao složene strukture podataka	

tip *imeNiza;
...
//kad unesemo broj elemenata
imeNiza = new tip[brojElemenata+1];
 	
tip imeNiza[brojElemenata+1];
 	
naredba imeNiza[brojElementa]
	dinamički način definiranja niza
	
	statički način definiranja niza
	
	pristup nizu

tip imeMatrice[brojRedova+1][brojKolona+1]; naredba imeMatrice[red][kolona]	definiranje matrice pristup matrici
Stringovi (alfanumeričke promjenjive)	
char imeS[maksimalnaDužinaStringa];	definiranje alfanumeričke promjenjive pod imenom: imeS
cin.getline(imeS,maksimalnaDužinaStringa); cin>>imeS;	učitavanje stringova u varijablu imeS učitavanje stringova u varijablu imeS do prvog praznog mjesta
strlen(imeS) /* za korištenje funkcije strlen() potrebno je uključiti string.h biblioteku, tj. treba napisati: #include <string.h> */	funkcija koja za rezultat vraća broj znakova alfa. promj. imeS
imeS[i]	pristup i-tom znaku u stringu imeS
Potprogrami	
/*u c++ potprogrami se mogu pisati bilo prije bilo poslije glavnog programa. Ja ću, radi nešto lakše varijante, potprograme pisati prije početka glavnog dijela programa*/	

1° void imePotprograma()
{
naredbe
}
	definiranje potprograma

imePotprograma();	dva načina pozivanja potprograma 1° za Basic i poziv potprograma 1° za Pascal i c++

2° void imePotprograma()
{
tip imeVarijable;
...
}
	definiranje potprograma koji će koristiti svoju lokalnu promjenjivu

imePotprograma();	dva načina pozivanja potprograma 2° za Basic i poziv potprograma 2° za Pascal i c++
sasvim je legalno da postoje lokalne i globalne promjenjive istog imena. Tada unutar potprograma vrijedi lokalna promjenjiva a ne globalna promjenjiva istog imena, dok u ostatku programa vrijedi globalna promjenjiva	

3° void imePotprograma(tip formalniParametar)
{
naredbe
}
	definiranje potprograma koji će koristiti prenos parametara po vrijednostima (vrijednosni parametri) u c++-u

imePotprograma(stvarniParametar);	poziv potprograma 3° za c++

4° void imePotpr(tip forPar1, tip forPar2, tip forPar3)
{
naredbe
}
	definiranje potprograma koji koristi tri formalna parametra i koji će koristiti prenos parametara po vrijednostima (vrijednosni parametri) u c++

imePotprograma(stvPar1, stvPar2, stvPar3);	poziv potprograma 4° za c++
formalni parametri su lokalne promjenjive, dok stvarni parametar može biti neka vrijednost, varijabla(kojoj je dodjeljena vrijednost) ili neki izraz	

5° tip imePotprograma(tip formalniParametar)
{
...
return traženaVrijednost;
}
	definiranje potprograma kao funkciju (funkcija je potprogram koja za rezultat vraća neku vrijednost koja nam je potrebna za dalji rad)

naredba imePotprograma(stvarniParametar)	poziv potprograma 5°

6° tip imePotprograma(tip forPar1, tip forPar2)
{
...
return traženaVrijednost;
}
	definiranje potprograma kao funkciju koja koristi dva formalna paramerra

naredba imePotprograma(stvPar1, stvPar2)	poziv potprograma 6°

7° tip imePotpr(tip1 forPar1, tip2 forPar2, tip3 forPar3)
{
...
return traženaVrijednost;
}
	generalno potprogrami mogu imati više parametara koji su različitog tipa. Ovo je jedan primjer za definiranje potprograma kao funkcije koja koristi tri formalna parametra, koja mogu biti različitog tipa

naredba imePotprograma(stvPar1, stvPar2, stvPar3)	poziv potprograma 7°

8° void imePotprograma(tip &formalniParametar)
{
naredbe
}
	definiranje potprograma koji će koristiti prenos parametara po referenci (promjenjivi parametri) u c++-u

imePotprograma(stvarniParametar);	poziv potprograma 8°

9° void imePotpr(tip &forPar1, tip &forPar2, tip &forPar3)
{
naredbe
}
	definiranje potprograma koji koristi tri formalna parametra i koji će koristiti prenos parametara po referenci (promjenjivi parametri) u Pascal-u i c++-u

imePotprograma(stvPar1, stvPar2, stvPar3);	poziv potprograma 9°
prenos parametara po vrijednosti podrazumjeva da se stvarni parametar ne može promjeniti. Parametri koji se prenose po vrijednosti nazivaju se vrijednosnim parametrima prenos parametara po referenci podrazumjeva da će stvarni parametar dobiti vrijednost koju želimo i čiju izmjenu ćemo definisati unutar nekog potprograma. U ovom slučaju stvarni parametar mora biti neka varijabla. Parametri koji se prenose po referenci nazivamo promjenjivi parametrima	
potprograme možemo učiniti praktički neovisnim od glavnog programa. Ovo je upravo jedno od osnovnih zadataka metodologije programiranja poznate pod nazivom struktuirano ili modularno programiranje. Pod tim podrazumjevamo razbijanje programa na više manjih cjelina - modula koji se mogu razvijati neovisno jedan od drugoga	
programske jezike možemo podjeliti na: jezike niskog nivoa 1.mašinski jezik(kombinacija 0 i 1), 2.asembler (ili simbolički jezik) i jezike visokog nivoa 3.proceduralni jezici 4.neproceduralni jezici. Za proceduralne jezike je neophodno znati definisati algoritam tj. jasan i nedvosmislen niz koraka koja u konačnom nizu dolazi do rešenja ili do zaključka da zadatak nema rješenja. Proceduralni jezici su: FORTRAN, COBOL, od novijih: BASIC, Pascal, C, c++ i tipični predstavnici Objektno orjentisanih jezika (posebna vrsta procedural. jezika): Visual Basic, Object Pascal (Delphi), C++ i Java. Kod neproceduralnih jezika nemora se sastavljati algoritam već moramo jasno definisati od čega se problem sastoji. Koriste se za baze podataka. Predstavnici su: SQL, Prolog, Lisp.	