Prof. Artan Boriçi

Leksione te Gjuhes se Programimit

Hyrje ne Gjuhen C++

1 Programet e Para dhe Kompilimi

Ne keto leksione do te paraqesim elementet baze te gjuhes C++. Keto elemente do te na lejojne te ndertojme programe te cfaredo lloj niveli komplikimi, por pa perdorur klasat dhe objektet. Pa perfshire keto te fundit, C++ eshte thuajse e njejte me gjuhen C. Klasat dhe objektet do t'i trajtojme si tema me zgjedhje ne nje cikel me te avancuar leksionesh.

1.1 Programi i pare ne C++

```
// my first program in C++
#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World!";
  return 0;
}
```

Verejme disa menjehere disa ndryshime prej C: libraria standarte e hyrje-daljeve ke mer tjeter, 'iostream' dhe funksioni perkates i daljes standarte 'cout' i merr ndryhsoret me ane te operatorit '¡¡'. Ndryshim tjeter eshte se komentet ne C++ mund te behen edhe duke vendosur '//'.

Gjithe elementet e librarise standarte te C++ deklarohen brenda asaj qe quhet 'namespace', pikerisht 'namespace std'. Me ane te kesaj shprehjeje ne deklarojme se duam te perdorim perberesit e kesaj librarie. Ky rresht vihet re shpesh ne programet C++ qe perdorin librine standarte.

Per te ekzekutuar nje program te nje gjuhe te kompiluar, sic eshte C++, duhet me pare te behet kompilimi i kodit burimor. Ka shume kompilatore te C++, por ne do perdorim g++. Ne Linux komanda e kompilimit te thjeshte do te

jete:

```
g++ -o emer1 emer2.cpp
```

ku `emer1` eshte emri ne dalje i komandes se ekzekutueshme qe rrjedh prej kompilimit dhe `emer2.cpp` eshte emri i kodit burimor. Verejme se shtrirja e emrit me `cpp` eshte e domosdoshme ne shume dialekte te Linux-it.

Pas prodhimit te kodit te ekzekutueshem do te donim te shuanim kureshtjen se cfare prodhon programi ne fjale. Kjo behet duke shkruar ne terminalin e Linux-it komanden:

```
./emer1
```

1.2 Programi i Dyte ne C++

Ne kete shembull te dyte do te paraqesim hyrje-dajle te thjeshta. Ne C++ hyrja dhe dalja prej terminalit default behet me ane te `cin` dhe `cout` duke perfshire skedarin koke `iostream`, p.sh.

```
// shembull hyrje-daljesh
#include <iostream>
using namespace std;

int main ()
{
  int i;
  cout << "Futni nje numer te plote: ";
  cin >> i;
  cout << "Vlera qe ju futet eshte" << i;
  cout << "dhe dyfishi i saj eshte" << i*2 << ".\n";
  return 0;
}
```

1.3 Detyrë

Te kopjohen programet e mesiperme ne nje redaktues teksti dhe te ruhen me emrin e zgjedhur prej jush (i cili sigurisht qe duhet te mbaroje me `cpp`, apo jo?!). Te kompilohen kodet burimore perkatese dhe te ekzekutohen kodet e ekzekutueshme perkatese.

2 Ndryshoret dhe tipet e te dhenave

Tani qe pame dy shembuj te thjeshte, duam te dime me shume per ndertimin e gjuhes ne menyre qe te kuptojme me shume per rreshtat e kodeve te mesiper, por edhe te mund te shkruajme programe te cilat zgjidhin probleme te thjeshta.

Nje gjuhe programimi, ashtu si edhe nje gjuhe tjeter cfaredo komunikimi, perbehet prej fjalesh. Fjalet ndahen ne fjale kyce ose te rezervuara vetem per gjuhen dhe fjale te tjera me ane te te cilave perdoruesei i gjuhes mund te identifikoje tipe te caktuara te dhenash, veprimesh apo objektesh me komplekse.

Ashtu si ne cdo gjuhe tjeter programimi, edhe ne C++ identifikuesit nuk duhet te jene fjale kyce te gjuhes ose te percaktuara si te tilla prej kompilatorit. Fjalet kyce te rezervuara jane:

```
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
```

Nuk lejohen qe operatoret te parqiten si identifikues meqe ato jane fjale te rezervuara:

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

3 Tipet themelore te te dhenave

Nje byte eshte sasia minimale e kujteses ne C++. Ne nje byte mund te vendoset nje sasi relativisht e vogel te dhenash: nje shenje ose nje numer te plote te vogel (zkonisht prej 0 deri ne 255). Nga ana tjeter, kompjuteri mund te manipuloje tipe te dhenash me komplekse qe formohen duke u grupuar ne disa byte, sic jane numrat e gjate ose numrat jo te plote. Me poshte jepet nje liste me tipet ekzisuese themelore te te dhenave ne C++ se bashku me kufinjte e vlerave qe ato mund te marrin:

Tipet themelore te te dhenave

```
Emri        Pershkrimi                      Madhesia Kufinjte
char        Karakter ose i plote i vogel    1byte    me shenje: -128 deri 127
```

			pa shenje: 0 deri 255
int	I plote	1fjale	me shenje:
			-2147483648 deri 2147483647
			pa shenje: 0 deri 4294967295
short int			
dhe short	I plote i shkurter	2bytes	me shenje: -32768 deri 32767
			pa shenje: 0 deri 65535
long int			
dhe long	I plote i gjate	4bytes	me shenje:
			-2147483648 deri 2147483647
			pa shenje: 0 deri 4294967295
bool	Vlere Booleane	1byte	true ose false
float	Numer me pike notuese	4bytes	3.4e +/- 38 (7 shifra)
double	Numer me pike notuese		
	me precizion te dyfishte	8bytes	1.7e +/- 308 (15 shifra)
long double	Numer i gjate me pike rrjedhese		
	me precizion te dyfishte	8bytes	1.7e +/- 308 (15 shifra)
wchar_t	Karakter i gjere	2bytes	1 karakter i gjere

Vlerat ne shtyllat Madhesia dhe Kufinjte varen prej arkitektures se sistemit ne te cilin kompilohet dhe ekzekutohet programi. Vlerat e dhena ketu jane ato qe ndeshen me shpesh ne sistemet me 32 bit. Po keshtu `short` dhe `long` jane praktikisht te njevlereshme me `short int` dhe `long int`.

3.1 Deklarimi i ndryshoreve

Deklarim i eshte i ngjashem si ne C: qe te perdorim nje ndryshore ne C++, do te duhet qe ta deklarojme me pare ate duke treguar se cfare tipi te dhenash duam qe ajo te jete.

Tipet ‘char, short, long, int’ mund te jene me shenje ose pa shenje. Default per shumicen e kompilatoreve te C++ eshte deklarimi me shenje. Perjashtim ben ‘char’ qe konsiderohet i ndryshem prej ‘signed char’ dhe ‘unsigned char’, te cilave u caktohen vlera numerike.

Po keshtu ‘signed’ dhe ‘unsigned’ mund te perdoren te vetem me nenkuptimin ‘int signed’ dhe ‘int unsigned’. Keshtu, deklarimet

```
unsigned VitiTjeter;
```

```
unsigned int VitiTjeter;
```

jane te njevlefshme.

3.2 Fusha e veprimit te ndryshoreve

Ndryshret mund te jene globale nese ato deklarohen ne fillim te programit perpara cdo funksioni dhe lokale nese ato deklarohen brenda trupit te nje funksioni:

```
#include <iostream>

// Ndryshore Globale
int numer_i_plote;
char nje_karakter;
char korde[20];
unsigned int numri_i_femijeve;

int main ()
{
  // Ndryshore Lokale
  unsigned short mosha;
  float nje_numer, nje_tjeter;

  // Urdhera
  cout << "Cila eshte mosha juaj?"
  cin >> mosha;
  ...
}
```

Ndryshoreve globale mund tu referohet kudo ne kod madje edhe brenda funksioneve, gjithnje pas deklarimit te tyre. Ndresa fusha e ndryshoreve lokale kufizohet brenda blloqeve te perfshira ne kllapa gjarperueshe.

3.3 Inicializimi i ndryshoreve

Nese duam te inicializojme nje ndryhsore te plote ne castin e deklarimit, atehere do te shkruajme:

```
int a = 0;
```

ose

```
int a (0);
```

3.4 Kordat

Kordat ne C++ nuk jane tip themelor por nje klase. Per deklaruar dhe perdorur kordat duhet te perfshijme nje skedar koke te quajtur "string" i cili i referohet `namespace std`, p.sh.

```
// shebull kordash
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string korde = "Kjo eshte nje korde";
  cout << korde;
  return 0;
}
```

3.5 Konstantet

Konstantet ne C++ u nenshtrohen te njejta rregullave si ne. Ne vecanti mund te percaktojme konstante me ane te direktives se preprocesorit `#define`. Po keshtu ato mund te deklarohen me ane te prefiksit `const` perpara tipit, si p.sh.

```
const float pi=3.1415926;
```

Ne rast se tipi nuk jepet, atehere ai nenkuptohet `int`.

4 Operatoret

Operatoret ne C++ jane thuajse ato te gjuhes C. Me poshte po japim tabelen permbledhese te tyre:

Perparesia	Operatori	Pershkrimi	Grupimi
1	::	fusha	Djathtas
2	() [] . -> ++ --		
	dynamic_cast		
	static_cast		
	reinterpret_cast		
	const_cast typeid	prapashtesa	Djathtas
3	++ -- ~ ! sizeof		
	new delete	unar (parashtese)	
	* &	ridrejtimi dhe referneca (shenjuesit)	
	+ -	operator unar te shenjes	Majtas
4	(tipi)	ndryshim tipi	Majtas
5	.* ->*	shenjues per tek anetar	Djathtas
6	* / %	shumezues	Djathtas
7	+ -	mbledhes	Djathtas
8	<< >>	zhvendoses	Djathtas
9	< > <= >=	krahasues	Djathtas
10	== !=	barazisese	Djathtas
11	&	bit AND	Djathtas
12	^	bit XOR	Djathtas
13	|	bit OR	Djathtas
14	&&	logjik AND	Djathtas
15	||	logjik OR	Djathtas
16	?:	kushtezues	Majtas
17	= *= /= %= += -=		
	>>= <<= &= ^= !=	i ckatimit	Majtas
18	,	presje	Djathtas

5 Aritmetika

Operatoret aritmetike jane '+', '-', '*', '/' (i cili, nese merr dy ndryshore "int", jep rezultat "int" duke hequr pjesen dhjetore) dhe operatori i mbetjes '%':

```
x = a % b;
```

Ne kete shembull, pasi pjestohet "a" nga "b", mbetja i kalohet "x"-it. Rezultati varet nga makina, veçse ne rastin kur "a" dhe "b" jane pozitive.
Ne aritmrtike, ndryshoret "char" trajtohen si "int" (merret kodi i tyre ASCII, qe eshte numer i plote). Rregulli i ndjekur ne çdo rast eshte qe karakteret konvertohen ne te plote dhe pastaj veprohet me to. Shembulli

```
c = c + 'A' - 'a';
```

konverton vleren e variables "c" nga shkronje e vogel, ne shkronje te madhe, duke shfrytezuar faktin qe distanca midis shkronjes se madhe dhe shkronjes se vogel ne ASCII eshte e njejte per çdo shkronje.

6 Libraria Matematike

Kurdohere qe duam te perdorim funksione matematike do duhet te perfshijme librarine matematike ne program. Shembulli i meposhtem ilustron perdorimin e librarise matematike ne C++:

```
// perdorimi i librarise matematike
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
int x = 0;
while(x < 10) {
double y = sqrt((double)x);
cout << "The square root of " << x << " e" << y << endl;
```

```
x++;
}
return 0;
}
```

7 Detyra Shtëpie

1. Te programohen shumat e meposhteme.

$$
\begin{aligned}
a) \quad S_n &= 1+\frac{1}{2}+\frac{1}{3} \\
b) \quad S_n &= \frac{1}{3}-\frac{1}{7}+\frac{1}{11}-\frac{1}{15} \\
c) \quad S_n &= \frac{1}{2}+\frac{1}{3}+\frac{2}{5}+\frac{3}{8}+\frac{5}{12} \\
d) \quad S_n &= \frac{1}{2}+\frac{1}{3}-\frac{3}{4}-\frac{3}{5}+\frac{5}{6}+\frac{5}{7}-\frac{7}{8}-\frac{7}{9} \\
e) \quad S_n &= \frac{1}{2}-\frac{1}{3}+\frac{3}{4}-\frac{3}{5}+\frac{5}{6}-\frac{5}{7} \\
f) \quad S_n &= \frac{1}{2}+\frac{3}{5}+\frac{6}{9}+\frac{10}{14} \\
g) \quad S_n &= 1+\frac{1}{3}-\frac{2}{5}-\frac{2}{7} \\
h) \quad S_n &= \frac{1}{2}+\frac{2}{3}+\frac{3}{4}
\end{aligned}
$$

2. Te programohen prodhimet e meposhteme.

$$
\begin{aligned}
a) \quad P_n &= \frac{1}{2}\cdot\frac{3}{4}\cdot\frac{5}{6} \\
b) \quad P_n &= \frac{1}{3}\cdot\frac{1}{3}\cdot\frac{2}{5}\frac{2}{5}\cdot\frac{4}{7}\cdot\frac{4}{7}
\end{aligned}
$$

8 Tabelat

Ne C, si ne Fortran mund te ndertohen tabela me elemente homogjene te tipave baze. Keshtu, per te deklaruar nje tabele me 10 elemente ”int” mund te shkruajme:

```
int x[10];
```

Te vihet re se ketu perdoren kllapat katrore. Indekset e elementeve do te fillojne nga 0, ne 9 dhe elementet e ”x”-it do te shkruhen:

```
x[0], x[1], x[2], ..., x[9]
```

Ne pergjithesi, nese nje tabele ka ”n” elemente, indekset do te shkojne nga 0, ne ”n-1”.
Mund te ndertohen edhe tabela shumepermasore. P.sh.:

```
int x[10] [15] [20];
```

Indekset mund te jene shprehje me vlera te plota. Keshtu mund te kete kuptim pohimi:

```
n = name[i+j] [1] + name[k] [2];
```

9 Strukturat e Kontrollit

C++ ka te njejtat struktura kontrolli si ato te gjuhes C:

```
- kushtezuese:  if (kushti) pohim1 else pohim2
- iterative:    while (shprehje) pohim
                do pohim while (kushti);
                for (vlera fillestare; kushti; rritja) pohim;
- perzgjedhese:
                switch (kushti)
                {
                  case konstante1:
                     grup pohimesh 1;
                     break;
                  case konstante2:
                     grup pohimesh 2;
                     break;
                  .
                  .
                  default:
                     grup pohimesh default;
```

```
                }
- pohimet break, continue dhe goto
- funksionin exit te percaktuar ne librarine cstdlib
```

9.1 Pohimi ”if”; operatoret e krahasimit; pohimet e perbera

Pohimi baze kushtezues ne C/C++ eshte pohimi if:

```
c = getchar ( );
if ( c == '?' )
        cout << "Perse shtype pikepyetje ?" << "\n";
```

Sintaksa e pergjithshme eshe:

```
if (shprehje) pohim
```

Shprehja ne kllapa te rrumbullaketa vleresohet dhe nese ajo eshte e ndryshme nga zero, pohimi ekzekutohet. Ekziston edhe klauzola fakultative ”else”, qe do pershkruhet shume shpejt.
Me poshte jepet lista e operatoreve te krahasimit.

```
==   e barabarte me;
!=   e ndryshme nga;
>    me e madhe se;
<    me e vogel se ;
>=   me e madhe e barabarte me;
<=   me e vogel e barabarte me;
```

Vlera e “shprehje *operator krahasimi* shprehje” eshte 1, nese eshte e vertete dhe 0, nese eshte e rreme. Te kihet parasysh se ”==” perdoret per test krahasimi, kurse ”=” eshte operatori i kalimit te vleres.
Shprehjet e krahasimit mund te jene te kombinuara me operatoret llogjike ”&&” (AND), ”||” (OR), ”!” (NOT). P.sh. nese duam te testojme nese nje karakter eshte hapesire boshe, apo tabulim, apo kalim ne rresht te ri mund te shkruajme:

```
if ( c ==' ' || c =='\t' || c =='\n' ) . . .
```

C garanton qe operatoret ”&&” dhe ”||” ne nje shprehje vleresohen nga e majta ne te djathte-se shprejti do te shohim raste ku radha eshte e rendesishme.
Nje nga gjerat e mira te C/C++ eshte se pjesa e pohimit tek ”if” mund te behet sa te duam e nderlikuar, mjafton te

vendoset brenda kllapave gjarperushe. Me poshte, duke perdorur pohimin ”if” krahasohen vlerat e dy ndryshoreve ”a” dhe ”b”, dhe nese a $<$ b, behet shkembimi i vlerave te tyre.

```
if  ( a < b ) {
            t = a;
            a = b;
            b = t;
 }
```

9.2 Shembuj

Shembujt e meposhtem ilustrojne edhe njehere perdorimin e operatoreve krahasues:

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

int main()
{
double x = 1;
double y = 2;
cout << "Rezultati: " << (x <= y) << endl;
return 0;
}
```

Ne shembullin e meposhtem vini re perdorimin e funksionit `main` me argumenta:

```
#include <iostream>
#include <string>

using namespace std;

void display(string a, string b, string comp)
{
cout << "Fjala \""
<< a << "\" eshte "
```

```
<< comp << " fjala \""
<< b << "\"" << endl;
}

int main(int argc, char **argv)
{
if(argc < 3) {
cout << "Ju lutem futni dy fjale pas emrit te programit te ekzekutueshem." << endl
}
else {
string a = argv[1];
string b = argv[2];
if(a < b) {
display(a,b,"me e vogel se");
}
else if(a > b) {
display(a,b,"me i madhe se");
}
else if(a == b) {
display(a,b,"i barabarte me");
}

}
return 0;
}
```

9.3 Pohimi ”while”; Kalimi i vleres brenda nje shprehjeje; pohimi zero

Mekanizmi baze i ciklimit ne C eshte pohimi ”while”. Ja nje shembull qe kopjon hyrjen e vet tek dalja e vet duke kojuar karakteret nje e nga nje. Mbani mend se ’$\backslash 0$’ eshte fundi i skedarit.

```
          main ( )  {
                      char c;
```

```
                while ( (c=getchar( )) != '\0' )
                        putchar(c);
        }
```

Sintaksa e pergjithshme eshte:

```
while ( shprehje ) pohim
```

Kuptimi i saj eshte:

(a) Vlereso shprehjen.

(b) Nese vlera e saj eshte e vertete, pra jo zero, ekzekuto pohimin dhe kthehu tek (a).

Meqenese shprehja kontrollohet perpara ekzekutimit te pohimit, pjesa e pohimit mud te ekzekutohet zero here, gje qe eshte shpesh e deshirueshme. Ashtu si tek "if", shprehja dhe pohimi mund te jene sa te duam te nderlikuara. Shembulli yne merr nje karakter, e kalon tek "c" dhe pyet nese ai eshte '$\backslash 0$'. Ne qofte se jo, ekzekutohet pjesa e pohimit, qe shtyp karkterin. Pastaj "while" perseritet. Kur me ne fund, karakteri hyres eshte '$\backslash 0$', "while" perfundon, keshtu ben edhe "main".

Verejme se eshte perdorur pohimi i kalimt te vleres,

```
c = getchar( )
```

brenda nje shprehjeje. Kjo praktike shkurton kodin dhe e ben ate me te qarte (mundohuni p.sh. te rishkruani kopjimin e skedarit pa perdorur kalimin e vleres brenda nje shprehjeje). Kjo menyre funksionon meqe pohimi i kalimit te vleres merr vlere njelloj si çdo shprehje tjeter. Vlera e saj eshte ajo anes se djathte. Kjo nenkupton se mund te bejme kalime te shumefishta vlere si me poshte:

```
x = y = z = 0;
```

Vleresimi behet nga e djathta ne te majte.

Verejme qe kllapat ne pohimin e kalimit te vleres brenda kushtezimit jane te nevojshme: ne qofte se do te shkruanim

```
  c = getchar( ) != '\0'
```

"c" do te jete 0 ose 1 ne varesi te kapjes ose jo te karakterit të fundit te skedarit. Kjo, pe arsye se, ne mungese te kllapave, operatori i kalimit te vleres, '=' vleresohet pas operatorit krahasues '!='. Ne rast dyshimi, madje edhe nese jo, vendos klappat.

Meqe `putchar(c)` kthen "c" si vlere te vet te funksionit, ne edhe mund te kopjojme hyrjen tek dalja duke nderfutur thirjet per tek `getchar` dhe `puchar`:

```
main( )  {
            while( putchar(getchar( )) != '\0' ) ;
 }
```

Çfare pohimi eshte duke u perseritur? Asnje, ose me sakte, pohimi zero, meqe e gjithe puna eshte bere brenda pjeses testuese te "while". Ky version eshte pak i ndryshem prej te meparshmit meqe '0" perfundimtare kpjohet tek dalja perpara se ne te vendosim te ndalojme.

9.4 Klauzola "else"; Shprehjet kushtezuese

Ne sapo perdorem nje "else" pas nje "if". Forma e pergjithshme e "if" eshte:

```
if (shprehje) pohim1 else pohim2
```

ku pjesa "else" eshte fakultative por shpesh e dobishme. Shembulli kanonik i jep "x"-it vleren e minimumit ndermjet "a" dhe "b":

```
if  ( a < b )
            x = a;
else
            x = b;
```

C jep nje forme kushtezuese alternative qe eshte shpesh me e permbledhur. Quhet "shprehje e kushtezuar" sepse eshte nje kushtezues qe ne te vertete merr nje vlere dhe mund te perdoret kudo ku mund te perdoret nje shprehje. Vlera e

```
a<b ? a : b;
```

eshte "a" ne qofte se "a" eshte me e vogel se "b"; perndryshe eshte "b". Ne pergjithesi forma,

```
shprehje1 ? shprehje2 : shprehje3
```

do te thote "vlereso `shprehje1`; nese nuk eshte zero, rezultati i se gjithes eshte `shprehje2`; perndryshe vlera eshte `shprehje3`.

Per t'i dhene "x"-it minimumin ndermjet "a" dhe "b", kemi:

```
x = (a<b ? a : b);
```

Klappat nuk jane te nevojshme sepse '?' vleresohet perpara '=', por siguria eshte paresore.

"if"-et dhe "else"-et mund te perdoren per ndertuar logjike qe degezohet ne nje ose disa udhe dhe qe rilidhet perseri, nje strukture e pergjithshme programimi kjo; pra,

```
if(...)
            {...}
 else if(...)
            {...}
 else if(...)
            {...}
 else
            {...}
```

Kushtet verifikohen me rradhe dhe vetem nje bllok pohimesh dhe vetem nje ekzekutohet: ose i pari per te te cilin kenaqet ”if”-i, ose ai i ”else”-it te fundit. Kur perfundon ky bllok, pohimi tjeter qe ekzekutohet eshte ai pas ”else”-it te fundit. Ne qofte se nuk duhet ndermarre asnje veprim per rastin *default*, atehere mos vendos ”else”-in e fundit.

9.5 Pohimi ”for”

Ne nje fare menyre pohimi ”for” eshte nje pohim i pergjithesuar ”while”, qe na lejon qe te vendosim pjesen inicializuese dhe inkrementuese te nje cikli ne nje pohim te vetem se bashku me testin. Forma e pergjithshme eshte:

```
for( inicializim; shprehja; inkrementim )
      pohim
```

Kuptimi eshte sikur te shkruanim:

```
inicializim;
 while( shprehje ) {
       pohim
       inkrementim;
 }
```

Keshtu, programi i meposhtem mbledh elementet e nje tabele:

```
sum = 0;
 for( i=0; i<n; i++)
        shuma = shuma + tabela[i];
```

Operatori ”++” tregon rritje ne vlere me nje, siç do te shohim me poshte.

Ne pohimin ”for” inicializimi mund te lihet jashte kllapes, por ne vendin e rezervuar per te ne kllape, duhet te jete pikepresja. Edhe inkrementimi eshte gjithashtu fakultativ dhe *nuk* ndiqet nga pikepresja. Testimi funksionon njelloj si tek ”while”: trupi i ciklit ekzekutohet per aq kohe qe shprehja eshte e ndryshme nga zero. Nese shprehja nuk shkruhet, merret sikur te ishte gjithmone e vertete. Keshtu:

```
for( ; ; ) ...
```

dhe

```
while( 1 ) ...
```

jane te dy cikle te pafundem.
Cikli ”for” eshte me kompakt se ”while” sepse e mban kodin aty ku perdoret dhe nganjehere eleminon nevojen e perdorimit te pohimeve te perbera, si ne kete shembull ku elementet e nje tabele 2-permasore behen zero:

```
for( i=0; i<n; i++ )
      for( j=0; j<m; j++ )
              tabela[i][j] = 0;
```

9.6 Pohimi `Switch`; `Break`; `Continue`

Pohimi ‘switch’ mund te perdoret per te zevendesuar testing e shumefishte te shembullit te fundit. Kur testet jane te tille

```
if( c == 'a' ) ...
else if( c == 'b' ) ...
else if( c == 'c' ) ...
else ...
```

pra tesimin e nje vlere perkundrejt nje serie konstantesh, pohimi ‘switch’ eshte shpesh me i qarte dhe zakonisht jep nje kod me te mire. Ai duhet te perdoret si me poshte:

```
switch( c ) {

case 'a':
        aflag++;
```

```
        break;
case 'b':
        bflag++;
        break;
case 'c':
        cflag++;
        break;
default:
        cout << c << "\n";
        break;
}
```

Pohimet 'case' etiketojne veprimet e ndryshme qe duam; default kryehet ne qofte se asnje prej rasteve te tjera nuk kenaqet. (Nje default eshte fakultativ; ne se nuk eshte aty dhe asnje rast nuk perputhet, atehere zbritet poshte.)

Pohimi 'break' ne kete shembull eshte i ri. Eshte aty meqenese rastet jane thjesht etiketa dhe pasi nje prej tyre eshte kryer, atehere kalohet tek rasti tjeter perveç se ne rastin kur vendoset ne menyre eksplicite per tu larguar. Kjo ka nje bekim te perzier. Ana pozitive eshte te paturit e shume rasteve ne nje pohim te vetem; ne mund te lejojme te kemi te medha dhe te vogla:

```
case 'a':  case 'A':    ...

case 'b':  case 'B':    ...
 etc.
```

Çfare ndodh nese duam te largohemi pasi te kermi kryer vetem rastin 'a'? Prej 'case' te pohimit 'switch' mund te dalim me ane te 'goto', por kjo eshte vertete e shumtuar. Pohimi 'break' na lejon te dalim pa perdorur 'goto' dhe etikete.

```
switch( c ) {

case 'a':
        aflag++;
        break;
case 'b':
        bflag++;
```

```
            break;
        ...
    }
    // pohimet break na sjellin direkt ketu
```

Pohimi ‘break’ funksionon edhe per pohimet ‘while’; ai shkakton daljen e menjehershme prej ciklit.

Pohimi ‘continue’ ecen vetem brenda ‘for’ dhe ‘while’; ai ben qe te filloje iteracioni i radhes ne cikel. Ne mund te kishim perdorur nje ‘continue’ ne shmebullin tone per te vazhduar iteracionin tjeter te ‘for’, por perdorimi i ‘break’ ne vend te tij duket me i qarte.

10 Operatoret e rritjes dhe zvogelimit

Gjuhet C/C++ permbajne operatoret ”++” (rritja me 1) dhe ”- -” (zvogelimi me 1). Supozojme se duam te numerojme rreshtat ne nje skedar.

```
        main( ) {
                int c,n;
                n = 0;
                while( (c=getchar( )) != '\0' )
                        if( c == '\n' )
                            ++n;
                cout << n << "\n";
    }
```

`++n` eshte ekuivalente me `n=n+1` por eshte me e qarte, veçanerisht kur `n` eshte shprehje e nderlikuar. Duhet thene veç se keto operatore vlejne vetem per tipat ”int” dhe ”char” (dhe per shenjuesit, qe do te shqyrtohen me tej).
Operatoret e ketij tipi mund te vihen si para, ashtu edhe prapa ndryshores qe do t’i ndryshohet vlera. Ndryshimi vihet re nese kalojme kete e vlere tek nje ndryshore tjeter. Keshtu,

```
    x = ++k;
```

do te thote qe ne fillim ”k”-se do t’i rritet vlera me 1, dhe pastaj kjo vlere do t’i jepet ”x”-it. Nese shkruajme:

```
    x = k++;
```

atehere, ne fillim ”x”-it i jepet vlera e ”k”-se dhe pastaj ”k”-se i rritet vlera me 1. Ne te dy rastet, per variablen ”k” nuk ka rendesi nese operatori i qendron para apo pas. Dallimi behet ne vleren qe merr ”x”-i.

11 Detyra Shtëpie

1. Per shumat e meposhteme te gjenden kufizat e tjera deri tek kufiza e n-te ku n eshte numer natyror i cfaredoshem. Te llogariten shumat me n kufiza me ane te programeve ne C++.

$$
\begin{aligned}
a) \quad S_n &= 1+\frac{1}{2}+\frac{1}{3}+\cdots+\cdots \\
b) \quad S_n &= \frac{1}{3}-\frac{1}{7}+\frac{1}{11}-\frac{1}{15}+\cdots \\
c) \quad S_n &= \frac{1}{2}+\frac{1}{3}+\frac{2}{5}+\frac{3}{8}+\frac{5}{12}+\cdots \\
d) \quad S_n &= \frac{1}{2}+\frac{1}{3}-\frac{3}{4}-\frac{3}{5}+\frac{5}{6}+\frac{5}{7}-\frac{7}{8}-\frac{7}{9}+\cdots \\
e) \quad S_n &= \frac{1}{2}-\frac{1}{3}+\frac{3}{4}-\frac{3}{5}+\frac{5}{6}-\frac{5}{7}+\cdots \\
f) \quad S_n &= \frac{1}{2}+\frac{3}{5}+\frac{6}{9}+\frac{10}{14}+\cdots \\
g) \quad S_n &= 1+\frac{1}{3}-\frac{2}{5}-\frac{2}{7}+\cdots \\
h) \quad S_n &= \frac{1}{2}+\frac{2}{3}+\frac{3}{4}+\cdots
\end{aligned}
$$

2. Per prodhimet e meposhteme te gjenden thyesat tjera deri tek thyesa e n-te ku n eshte numer natyror i cfaredoshem. Te llogariten prodhimet me n thyesa me ane te programeve ne C++.

$$
\begin{aligned}
a) \quad P_n &= \frac{1}{2}\cdot\frac{3}{4}\cdot\frac{5}{6}\cdots \\
b) \quad P_n &= \frac{1}{3}\cdot\frac{1}{3}\cdot\frac{2}{5}\frac{2}{5}\cdot\frac{4}{7}\cdot\frac{4}{7}\cdots
\end{aligned}
$$

3. Jepen dy vargje te renditura numra reale me n_1 dhe n_2 elemente. Te formohet vargu i renditur me n_1+n_2 elemente, i cili eshte bashkim i vargjeve fillestare. *Udhezim: te shfrytezohet fakti se vargjet fillestare jane te renditur.*

4. **Histograma**. Le te jete $x_1, x_2, \ldots, x_n$ varg numrash reale ne intervalet $[a, b)$ qe merren prej matjeve te nje eksperimenti. Te ndahet intervali ne m intevale te barabarta:

$$[a+(j-a)h, a+jh)\ , j=1,2,\ldots,m\ ,$$

me gjatesi

$$h=\frac{b-a}{m}\ .$$

Te gjenden numrat n_j qe tregojne sasine e pikave $x_i, i=1,2,\ldots,n$ qe ndodhen ne intervalin e j-te.

12 Funksionet

Funksionet bejne te mundur strukturimin e nje programi ne forme modulare. Ashtu si ne C argumentat pasohen me vlere ose me reference kur perdorim shenjuesit. Nje funksion duhet te jete percaktuar perpara se te mund t'i referohemi perndryshe ata duhet qe patjeter te deklarohen me para dhe te percaktohen pas `main`. Argumentave mund t'i caktojme edhe vlera default si ne shembullin e meposhtem:

```
// vlerat default ne funksione
#include <iostream>
using namespace std;

int pjestim (int a, int b=2)
{
  int r;
  r=a/b;
  return (r);
}

int main ()
{
  cout << pjestim (12);
  cout << endl;
  cout << pjestim (20,4);
  return 0;
}
```

Po keshtu funksionet mund te mbingarkohen:

```
// funksione te mbingarkuara
#include <iostream>
using namespace std;

int veprim (int a, int b)
{
```

```
  return (a*b);
}

float veprim (float a, float b)
{
  return (a/b);
}

int main ()
{
  int x=5,y=2;
  float n=5.0,m=2.0;
  cout << veprim (x,y);
  cout << "\n";
  cout << veprim (n,m);
  cout << "\n";
  return 0;
}
```

12.1 Funksionet `inline` dhe rekursiviteti

Sic dihet thirrja e nje funksioni ben qe te caktohet nje vend i perkoshem ne kujtese per ndryshoret e tij (stack) dhe pastaj te ekzekutohet kodi perkates. Kjo kerkon nje kohe te caktuar ekzekutimi e cila mund te jete e knosiderueshme kur funksioni ne fjale thirret shpesh. Per funksione te shkurtera eshte e mundur qe kompilatori te prodhoje nje kod qe i bashkangjitet kodit ne vendin ku thirret funksioni, proceduare qe quhet `inline`. Per kete perdoret formati:

```
inline tipi emri ( argumenta ... ) { udhezime ... }
```

C++ ofron rekursivitet ashtu si dhe C. P.sh. nese duam te llogarisim $n!$ mund te shkruajme:

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

```
long faktorial (long a)
{
  if (a > 1)
   return (a * faktorial (a-1));
  else
   return (1);
}

int main ()
{
  long number;
  cout << "Please type a number: ";
  cin >> number;
  cout << number << "! = " << faktorial (number);
  return 0;
}
```

Ne C++, ndryshoret e thjeshta (jo tabelat) pasohen me ane te “thirrjes me vlera”, qe do te thote se funksionit te thirrur i jepet nje kopje e argumentave te tij pa ditur adresen e tyre. Kjo ben te pamundur ndryshimin e vleres se agumentave hyres.

Ka dy menyra per te dale prej kesja dileme: njera eshte kalimi tek funksioni i adreses se ndryshores ne vend te vleres se saj dhe tjera eshte percaktimi i ndryshores si globale ose te jashteme, e cila eshte e njohur tek cdo funksion me ane te emrit te tij. Ne paragrafet e meposhteme do te diskutojme me tej keto mundesi.

12.2 Shembuj

Paneli i kontrollit te presidentit te SHBA-ve.

```
#include <iostream>

using namespace std;

// funksioni qe i tregon perdoruesi se cfare duhet te beje
```

```
void tregoni()
{
cout << "Paneli i Kontrolli i Presidentit te SHBA-ve\n\n";
cout << "Zgjidhni:\n\n";
cout << "\t1. Ndizni makinen e kafes.\n";
cout << "\t2. Filloni Luften e Trete Boterore.\n";
cout << "\t3. Ndisni mprehesen elektrike te lapsit.\n\n";
}

// pranoni zgjedhjen e perdoruesit dhe kthejeni ate

int zgjidhni()
{
// deklaroni ndryshoren e rezultatit
int zgjedhja;
// do while per sa kohe perdoruesi jep dicka te gabuar
do {
cout << "Enter (1,2,3):" << flush;
// prano hyrjen e perdoruesit, testo vlefshmerine
                // nese gabim, eshte gabim hyrjeje
if(!(cin >> zgjedhja)) {
// sheno gabimin
zgjedhja = -1;
// "riparo" rrjedhen hyrese
cin.clear();
cin.ignore(1000,'\n');
}
// nese eshte ne rregull, testo intervalin e hyrjeve te pranueshme
else if((zgjedhja < 1) || (zgjedhja > 3)) {
// sheno gabimin
```

```
zgjedhja = -1;
}
// ne rast se ka ndonje gabim
if(zgjedhja == -1) {
cout << "Gabim ne hyrje. Futni nje numer te plote ne intervalin 1-3.\n";
}
}
while(zgjedhja == -1);
return zgjedhja;
}

// veproni me te marre hyrjen prej perdoruesit: kjo eshte nje prej menyrave

void veproni0(int zgjedhja)
{
cout << "Ju zgjodhet " << zgjedhja << ", prandaj po ";
if(zgjedhja == 1) {
cout << "ndez makinen e kafes...\n";
}
else if(zgjedhja == 2) {
cout << "filloj luften e Trete Boterore...\n";
}
else if (zgjedhja == 3) {
cout << "ndez mprehesen elektrike te lapsit...\n";
}
}

// veproni me te marre hyrjen prej perdoruesit: kjo eshte nje menyre me efikase

void veproni(int zgjedhja)
{
```

```
cout << "Ju zgjodhet " << zgjedhja << ", prandaj po ";
switch(zgjedhja) {
case 1:
{
cout << "ndez makinen e kafes...\n";
break;
}
case 2:
{
cout << "filloj luften e Trete Boterore...\n";
break;
}
case 3:
{
cout << "ndez mprehesen elektrike te lapsit...\n";
break;
}
}
}

int main()
{
int zgjedhja;
tregoni();
zgjedhja = zgjidhni();
veproni(zgjedhja);
return 0;
}
```

Leximi i nje vektori prej nje skedari.

```
// Lexon nje vektor nga nje file
#include <iostream>
```

```
#include <fstream>
#define n 10
using namespace std;

double x[n];

void read(double x[])
{
  int k;
  ifstream fin("data");
  for (k=0; k<n; k++)
  {
   fin >> x[k];
  }
}

int main ()
{
  int k;
  read(x);
  for (k=0; k<n; k++)
  {
   cout << x[k] << endl;
  }
  return 0;
}
```

Shuma e elementeve te nje vektori.

```
// Shembulli i shumes
#include <iostream>
using namespace std;
```

```
double x[]={1,2,3,4,5};
int n=5;
double y;

double sum1(int n, double x[])
{
  double s1=0;

  for ( n=0 ; n<5 ; n++ )
  {
   s1 += x[n];
  }
  return s1;
}

int main ()
{
  y=sum1(n,x);
  cout << y << endl;
  return 0;
}
```

Vlera me e madhe e elementeve te nje vektori.

```
// Gjen maximumin e elementeve te nje vektori
#include <iostream>
#include <fstream>
#define n 10
using namespace std;

double x[n];

void read(double x[])
```

```
{
  int k;
  ifstream fin("data");
  for (k=0; k<n; k++)
  {
   fin >> x[k];
  }
}

double max(double x[])
{
  double max0=-1e-15;
  int k=0;
  for (k=0; k<n; k++)
  {
   if (x[k]>max0)
   {
    max0=x[k];
   }
  }
  return max0;
}
```

13 Shenjuesit

Nje shenjues ne C/C++ eshte adresa e diçkaje. Eshte me te vertete e rralle kur ne duam te dime adresen vete, nderkohe qe shenjuesit jene nje menyre krejt e zakonshme per te shkuar tek permbajtja e diçkaje. Operatori unar '&' perdoret per te prodhuar adresen e nje objekti, ne se ai ka te tille. Keshtu,

```
int a, b;
b = &a;
```

vendos adresen e ‘a’ tek ‘b’. Ne nuk mund te bejme shume me kaq pervec se ta printojme ate ose t’ia kalojme nje rutine tjeter. Kjo per aresye se ‘b’-se nuk i kemi dhene deklarimin e duhur. Por ne qofte se deklarojme ‘b’ si shenjues tek nje numer i plote, atehere jemi ne gjendje te mire:

```
int a, *b, c;
b = &a;
c = *b;
```

Tani ‘b’ permban adresen e ‘a’ dhe ‘c = *b’ do te thote te perdoret vlera e ‘b’-se si nje adrese, d.th. si nje shenjues. Efekti eshte qe ne te marrim perseri permbajtejen e ‘a’-se, ndonese ne menyre indirekte. (Eshte gjithnje e vertete se ‘*&x’ eshte e njejte me ‘x’ ne rast se ’x’ ka nje adrese.)

Perdorimi me i shpeshte i shenjuesve ne C eshte shetitja e shpejte pergjate tabelave. Ne fakt, emri i nje tabele ne C perfqason adresen e elementit zero te saj, prandaj ajo nuk mund te vendoset ne anen e majte te nje shprehjeje. (Adresa e diçkaje nuk mund te ndryshohet duke kaluar vlera tek ajo.) Ne se shkruajme:

```
char *y;
char x[100];
```

‘y’ eshte i tipit shenjues tek karakter (megjithese nuk tregon ende ne ndonje vend). Ne mund ta bejme ‘y’ te tregoje tek nje element i ‘x’-it me dy menyra:

```
y = &x[0];
y = x;
```

Meqe ‘x’ eshte adresa e ‘x[0]’ menyra e dyte eshte legale dhe konsistene. Nga ana tjeter ‘*y’ jep ‘x[0]’. Per me teper,

```
*(y+1)   /* jep x[1] */
*(y+i)   /* jep x[i] */
```

ndersa vargu i pohimeve,

```
y = &x[0];
y++;
```

ben qe ‘y’ te tregoje tek ‘x[1]’.

Le te perdorim shenjuesit ne nje funksion ‘length’ qe llogarit gjatesine e nje tabele karakteresh. Rikujtojme qe me mareveshje te gjithe tabelat e karaktereve mbarojne me ‘\0’. (Ne qofte se jo programi do te rrezohet patjeter). Menyra e vjeter eshte:

```
length(s)
   char s[ ]; {
        int n;
        for( n=0; s[n] != '\0'; )
                n++;
        return(n);
}
```

Duke e rishkruar me shenjues:

```
length(s)
   char *s; {
        int n;
        for( n=0; *s != '\0'; s++ )
                n++;
        return(n);
}
```

Tani mund te kuptohet perse duhet te percaktojme ne cfare gjeje ‘s’ tregon: nese do ta shtonim ate me ane te ‘s++’ atehere do ta shtonim me sasine e duhur.

Versioni me shenjues eshte me eficient (kjo eshte thuajse gjithnje e vertete), ndersa edhe me kompakt eshte:

```
        for( n=0; *s++ != '\0'; n++ );
```

‘*s’ kthen nje karakter; ‘++’ shton shenjuesin me nje, ne kete menyre marrim karakterin e radhes heren tjeter. Sic mund te shihet, duke i bere gjerat me eficiente i bejme ato njekohesiht me pak te qarta. Por ‘*s++’ eshte nje idom kaq e perdorur saqe duhet te mesohet patjeter.

Duke shkruar nje hap perpara, ketu eshte funksioni ‘strcopy’ qe kopjon tabelen e karaktereve ‘s’ tek nje tjeter ’t’:

```
strcopy(s,t)
   char *s, *t; {
        while(*t++ = *s++);
}
```

Testi perkundrejt ‘\0’ eshte lene menjeane, meqe ‘\0’ eshte identikisht zero; nje kodim te tille ndeshet shpesh. (Duhet vendosur hapsire prapa ‘=’: shih paragrafet qe vijojne).

Per argumentet e nje funksioni dhe vetem per to deklarimet

```
        char s[ ];
        char *s;
```

jane te njevlefshme: nje shenjues tek nje tip ose nje tabele me madhesi te papercaktuar jane e njejta gje.

Nese e gjitha kjo duket misterioze, eshte mire te mesohen keto forma permendesh deri sa te behen natyre e dyte. Shpesh nuk duhet dicka me e nderlikuar.

13.1 Shenjuesit tek Funksionet

Tregimi tek funksionet perdoret kryesisht per te pasuar nje funksion si argument te nje funksioni tjeter. Per te deklaruar nje shenjues tek nje funksion duhet te deklaruar prototipi i funksionit me emrin e funksionit te futur brenda kllapave dhe te filluar me nje yll:

```
// shenjues tek funksionet
#include <iostream>
using namespace std;

int mbledhje (int a, int b)
{ return (a+b); }

int zbritje (int a, int b)
{ return (a-b); }

int veprim (int x, int y, int (*funksioni_qe_thirret)(int,int))
{
  int g;
  g = (*funksioni_qe_thirret)(x,y);
  return (g);
}

int main ()
{
  int m,n;
```

```
  int (*minus)(int,int) = zbritje;

  m = veprim (7, 5, mbledhje);
  n = veprim (20, m, minus);
  cout << n << endl;
  return 0;
}
```

14 Kujtesa dinamike

Kujtesa dinamike sherben per te ndryshuar kerkesen per kujtese gjate ekzekutimit te programit. Per te bere te mundur kete perdoret operatori `new` pasuar prej tipit, p.sh. udhezimet:

```
int * ndim;
ndim = new int [5];
```

bejne qe te caktohet kujtese ne menyre dinamike per pese elemente te plote e te kthehet nje shenjues, `ndim`, qe tregon tek elementi i pare i plote ne kujtese. Ne qofte se sistemi nuk ka kujtese te lire atehere programi ndalon automaikisht. Ne rast se duam qe kjo te mos ndodhe shkruajme:

```
int * ndim;
ndim = new (nothrow) int [5];
if (ndim == 0) {
  // shenjues zero: programi nuk mund te caktoje kujtese; merr masa te tjera
  };
```

Pra, vendosja e `(nothrow)` pas `new` ben qe ne rast se nuk ka kujtese te lire, `ndim` i caktohet *shenjuesi zero, pra nje shenjues qe nuk tregon ne asnje vend ne kujtese*.

Per te kthyer mbrapsht veprimin e krijimit te kujteses perdoret operatori `delete`, p.sh.:

```
// kujtesa dinamike
#include <iostream>
using namespace std;

int main ()
```

```
{
  int i,n;
  int * p;
  cout << "Sa numra doni te futni? ";
  cin >> i;
  p= new (nothrow) int[i];
  if (p == 0)
    cout << "Gabimi: kujtesa nuk mund te caktohet";
  else
  {
    for (n=0; n<i; n++)
    {
      cout << "Futni numrin: ";
      cin >> p[n];
    }
    cout << "Ju keni futur: " << endl;
    for (n=0; n<i; n++)
      cout << p[n] << endl;
    delete[] p;
  }
  return 0;
}
```

15 Zbatime

Ne kete kapitull do te shohim disa zbatime te gjuhes C++ ne zgjidhjen e disa problemeve numerike.

15.1 Zgjidhja e Barazimeve Algjebrike Lineare

Sisteme Trekendore

Metoda e Eleminimit Gaussain

15.2 Perafrimi Linear i te Dhenave

```
// Perafrimi linear i pikave (x,y) i formes y=a+bx
#include <iostream>
#include <fstream>
#define n 4 // n > 2
using namespace std;

int k;
double x[n],y[n];
double s1=0,s2=0,s12=0,s22=0;

void read(double x[], double y[])
{
  ifstream fin("data");
  for (k=0; k<n; k++)
  {
   fin >> x[k];
   fin >> y[k];
  }
}

void sums(double x[],double y[],double& s1,double& s2,double& s12, double& s22)
{
  for (k=0 ; k<n ; k++)
  {
   s1 += x[k];
   s2 += y[k];
   s12 += x[k]*y[k];
   s22 += x[k]*x[k];
```

```
  }
}

void results(double x[],double y[],double s1,double s2,double s12, double s22,doub
{
  double det;
  det=n*s22-s1*s1;
  // trego ne qofte se det=0
  if (det==0)
  {
   cout << "rank deficient least squares" << endl;
  }
  // koeficientet
  b=(n*s12-s1*s2)/det;
  a=(s2-b*s1)/n;
  // hi-katrori
  chi2=0;
  for (k=0 ; k<n ; k++)
  {
   chi2 += (y[k]-a-b*x[k])*(y[k]-a-b*x[k]);
  }
  // gabimet e koeficienteve
  sa=sqrt(s22/det*chi2/(n-2));
  sb=sqrt(n/det*chi2/(n-2));
}

int main ()
{
  double a, b, sa, sb, chi2;
  read(x,y);
  sums(x,y,s1,s2,s12,s22);
```

```
  results(x,y,s1,s2,s12,s22,a,b,sa,sb,chi2);
  cout << "Koeficientet per perafrimin y=a+b*x:" << endl;
  cout << "a,sa=" << a << " " << sa << endl;
  cout << "b,sb=" << b << " " << sb << endl;
  cout << "chi2/dof=" << chi2/(n-2) << endl;
  return 0;
}
```

15.3 Metoda Newton per Zgjidhjen e Barazimeve Algjebrike Jolineare

```
// Rrenjet e barazimit algjebrik jolinear
// Metoda Newton

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

double  eps,x,y;
int     nmax,n;

// Shembull: f(x) = x^2 - 2
void func_and_deriv(double x, double *f, double *f_prime)
{
    // funksioni
    *f = x*x - 2;
    // derivati
    *f_prime = 2*x;
}

void  Newton()
{
  ofstream fout("rezult");
```

```
  double f,f_prime;
  n=0;
  do {
      fout << fabs(x-sqrt(2.)) << endl;
      // Llogarit f dhe f_prime
      func_and_deriv(x,&f,&f_prime);
      // Llogarit perafrimin e ri
      x=x-f/f_prime;
      n++;
  } while ( (fabs(f/f_prime) > eps) && (n<=nmax) );
}


int main()
{
 cout << "Jep perarfrimin fillestar: x = "; cin >> x;
 cout << "Jep saktesine e llogaritjes: eps = "; cin >> eps;
 cout << "Jep maksimumin e iteracioneve: nmax = "; cin >> nmax;

 // Therrit funksionin Newton
 Newton();

 cout << "\nRrenja: x = " << x << endl;
 cout << "Vlera e funksionit: y = " << y << endl;
 cout << "Numri i hapave iterative: n = " << n << endl;
 return 0;
}
```