Lucrarea de laborator nr. 5.

Tema: Fluxurile Input şi Output standard şi definite de utilizatori. Formatarea fluxurilor numerice şi textuale. Fluxurile stringuri şi de memorie.

Scopul lucrării: familiarizarea studenţilor cu fluxurile input şi output standard şi definite de utilizatori, cu formatarea fluxurilor numerice şi textuale, cu fluxurile stringuri şi de memorie.

Consideraţiile teoretice necesare:

Noţiune de Fluxuri

Limbajul C++ foloseşte conceptul abstract de *stream* (*flux*) pentru realizarea operaţiilor input/output. Din punctul de vedere al limbajului C++, fluxul este o clasă în sens obişnuit. El oferă metode de scriere şi citire a datelor independente de dispozitivul I/O. Operaţiile de input/output se fac prin intermediul obiectelor de tip *flux*. Acestea pot fi clasificate în funcţie de dispozitivul fizic asociat respectivei operaţii. Din acest punct de vedere, limbajul C++ distinge trei categorii de fluxuri:

– input/output standard (de exemplu, tastatura şi ecranul);
– input/output prin intermediul fişierelor;
– input/output în memorie.

Fluxurile incapsulează (ascund) problemele specifice dispozitivului cu care se lucrează, sub biblioteca standard *iostream.h*.. Biblioteca *iostream.h*. este scrisă însăşi în limbajul C++ şi este o bibliotecă a claselor.

Alt avantaj al utilizării fluxurilor se datorează implementării bibliotecii *iostream.h*, care utilizează un sistem de zone tampon. Operaţiile de intrare/ieşire cu dispozitivele periferice sînt consumatoare de timp. Informaţiile trimise către un flux nu sînt scrise imediat în dispozitivul în cauză, ci sînt transferate într-o zonă de memorie tampon, din care sînt descărcate către dispozitiv în momentul umplerii acestei zone de memorie.

In limbajul C++ fluxurile au fost implementate utilizînd clase, după cum urmează:

clasa *streambuf*	gestionează zonele tampon,
clasa *ios*	este clasa de bază pentru clasele de fluxuri de intrare şi de ieşire. Clasa *ios* are ca variabilă membru un obiect de tip *streambuf,*
clasele *istream* şi *ostream*	sînt derivate din *ios*,
clasa *iostream*	este derivată din *istream* şi *ostream* şi oferă metode pentru lucrul cu terminalul,
clasa *fstream*	oferă metode pentru operaţii cu fişiere.

.***Obiecte standard.*** La lansarea în execuţie a unui program C++, care include *iostream.h*, in mod automat compilatorul limbajului C++ creează şi iniţializează patru obiecte:

cin	gestionează intrarea de la intrarea standard (tastatura),
cout	gestionează ieşirea către ieşirea standard (ecranul),
cerr	gestionează ieşirea către dispozitivul standard de eroare (ecranul), neutilizînd zone tampon,
clog	gestionează ieşirea către dispozitivul standard de eroare (ecranul), utilizînd zone tampon

Expresia "*cout «*" se utilizează pentru afişarea diverselor valori. De fapt, *cout* este un obiect de tip *flux*, şi anume obiectul "*flux standard output*". Operaţia *output* (afişare) se realizează prin intermediul operatorului supraîncărcat «.

Obiectul de tip *flux standard input* este *cin*, iar operaţia *input* (citirea) se realizează prin intermediul operatorului supraîncărcat ». Obiectele *cout* şi *cin* sînt declarate în biblioteca *iostream.h*.

Redirectări. Dispozitivele standard de intrare, ieşire şi eroare pot fi *redirectate* către alte dispozitive. Erorile sînt, de obicei, redirectate către fişiere, iar intrarea şi ieşirea pot fi *conduse* ("*piped*") către fişiere utilizînd comenzi ale sistemului de operare (utilizarea ieşirii unui program ca intrare pentru altul). Sintaxa pentru operaţii de ieşire, *cout*:

cout << InformatieDeTrimisLaIesire;

Respectiv pentru intrare, *cin*:

cin >> NumeVariabilă;

De fapt, *cin* şi *cout* sînt nişte obiecte definite global, care au supraîncărcat operatorul >> respectiv << de mai multe ori, pentru fiecare tip de parametru în parte (*int*, *char* *, etc.):

istream &operator >> (TipParametru &)

De exemplu:

```
#include <iostream.h>
void main()
{int IntegerNumber=50;
cout << "IntegerNumber = "; cin >> IntegerNumber;
cout<<"\nWhat you entered = "<<IntegerNumber<<endl;
}
```

Rezultatul îndeplinirii programului:

```
IntegerNumber = 200
What you entered = 200
```

Acest scurt program citeşte de la intrarea standard o valoare întreagă, pe care o trimite apoi către ieşirea standard. Se observă posibilitatea de a utiliza simbolurile '*n*', '*t*', s.a.m.d (ca la *printf*, *scanf*, etc.). Utilizarea simbolului *endl* va forţa golirea zonei tampon, adică trimiterea datelor imediat către ieşire.

Atît operatorul >>, cît şi << returnează o referentă către un obiect al clasei *istream*. Deoarece *cin*, respectiv *cout*, este şi el un obiect *istream*, valoarea returnată de o operaţie de citire/scriere din/în *stream* poate fi utilizată ca intrare/ieşire pentru următoarea operaţie de acelaşi fel.

Operaţia de intrare cin. Funcţia cin.get() poate fi utilizată pentru a obţine un singur caracter din intrare, apelînd-o fără nici un parametru, caz în care returnează valoarea utilizată, sau ca referinţă la un caracter.

```
get();          //fără parametri
```

În această formă, funcţia întoarce valoarea caracterului găsit. Spre deosebire de operatorul >>, funcţia nu poate fi utilizată pentru a citi mai multe intrări, deoarece valoarea returnată este de tip întreg, nu un obiect *istream*. Un exemplu de utilizare:

```
#include <iostream.h>
void main()
{char c;
while((c = cin.get()) != '*')
```

{cout << "c = " << c << endl;}
}
Rezultatul îndeplinirii programului:
*asdfgh**
c = a
c = s
c = d
c = f
c = g
c = h

Citirea de şiruri de caractere utilizînd get(). Operatorul >> nu poate fi utilizat pentru a citi corect şiruri de caractere de la intrare, deoarece spaţiile sînt interpretate ca separator între diverse valori de intrare. În astfel de cazuri trebuie folosită funcţia *get()*. Sintaxa de utilizare a funcţiei *get* în acest caz este următoarea:

*cin.get(char *PointerLaSirulDeCaractere, int Lungime Maximă, char Sfîrşit);*

Primul parametru este un pointer la zona de memorie în care va fi depus şirul de caractere. Al doilea parametru reprezintă numărul maxim de caractere ce poate fi citit plus unu. Cel de-al treilea parametru este caracterul de încheiere a citirii, care este opţional (implicit considerat '*n*').

În cazul în care caracterul de încheiere este întîlnit înainte de a fi citit numărul maxim de caractere, acest caracter nu va fi extras din flux. Există o funcţie similară funcţiei *get()*, cu aceeaşi sintaxă, numită *getline()*. Funcţionarea sa este identică cu *get()*, cu excepţia faptului că acel ultim caracter menţionat mai sus este şi el extras din flux.

Funcţia cin.ignore() se utilizează pentru a trece peste un număr de caractere pînă la întîlnirea unui anume caracter. Sintaxa sa este:

cin.ignore(int NumărMaximDeCaractere, char Sfîrşit);

Primul parametru reprezintă numărul maxim de caractere ce vor fi ignorate, iar al doilea parametru – caracterul care trebuie găsit.

Funcţia cin.peek() returnează următorul caracter din flux, fără însă a-l extrage.

Funcţia cin.putback() inserează în flux un caracter.

cout . Funcţii membri ale cout

***Funcţia cout.flush()*)** determină trimiterea către ieşire a tuturor informaţiilor aflate în zona de memorie tampon. Această funcţie poate fi apelată şi în forma *cout << flush*.

***Funcţia cout.put()*)** scrie un caracter către ieşire. Sintaxa sa este următoarea:

cout.put(char Caracter);

Deoarece această funcţie returnează o referinţă de tip *ostream*, pot fi utilizate apeluri succesive ale acesteia, ca în exemplul de mai jos:

#include <iostream.h>
void main()
{cout.put('H').put('i').put('!').put('\n');}

Funcţia cout.write() are acelaşi rol ca şi operatorul <<, cu excepţia faptului că se poate specifica numărul maxim de caractere ce se doresc scrise. Sintaxa funcţiei *cout.write()* este:

*cout.write(char *SirDeCaractere, int CaractereDeScris);*

Formatarea ieşirii

Funcţia cout.width() permite modificarea dimensiunii valorii trimise spre ieşire, care implicit este considerată exact mărimea cîmpului în cauză. Ea modifică dimensiunea numai pentru următoarea operaţie de ieşire. Sintaxa este:

cout.width(int Dimensiune);

Funcţie cout.fill() permite modificarea caracterului utilizat pentru umplerea eventualului spaţiu liber creat prin utilizarea unei dimensiuni mai mari decît cea necesară ieşirii, cu funcţia *cout.width()*. Sintaxa acesteia este:

cout.fill(char Caracter);

Opţiuni de formatare a ieşirii. Pentru formatarea ieşirii sînt definite două funcţii membri ale *cout*, şi anume:

Funcţia cout.setf() activează o opţiune de formatare a ieşirii, primită ca parametru:

cout.setf(ios::Opţiune);

unde *Opţiune* poate fi:

Showpos	determină adăugarea semnului plus (+) în faţa valorilor numerice pozitive;
left, right, internal	schimbă alinierea ieşirii (la stînga. La dreapta, centrează);
dec, oct, hex	schimbă baza de numeraţie pentru valori numerice;
showbase	determină adăugarea identificatorului bazei de numeraţie în faţa valorilor numerice.

Funcţia cout.setw() modifică dimensiunea ieşirii, fiind similară funcţiei *cout.width()*. Sintaxa sa este:

cout.setw(int Dimensiune);

În continuare vom exemplifica utilizarea funcţiilor pentru formatarea ieşirii:

```
#include <iostream.h>
#include <iomanip.h>
void main()
{int number = 783;
cout << "Număr = " << number<<endl;
cout.setf(ios::showbase);
cout<<"Număr în sistem hexazecimal = "<<hex << number<<endl;
cout.setf(ios::left);
cout << "Număr în sistemul octal, aliniat la stînga = " << oct << number<<endl;
}
```

Rezultatul îndeplinirii programului:

Număr = 783
Număr în sistem hexazecimal = 0x30f
Număr în sistemul octal, aliniat la sîinga = 01417

Redefinirea operatorilor de intrare şi ieşire pentru fluxul standard. Operatorii de intrare şi ieşire pentru fluxul standard pot fi redefiniţi pentru citirea şi înregistrarea obiectului de tipul clasei definite de utilizator.

Operatorul de intrare a fluxului standard pentru un obiect din clasa *obj* se redefineşte în modul următor:

```
istream & operator >> (istream &s, class obj)
{ //citirea componentelor din clasa obj
return s; }
```

Operatorul de ieşire a fluxului standard pentru un obiect din clasa *obj* se redefineşte în modul următor:

```
ostream & operator<< (ostream &s, class obj)
{ // tipărirea componentelor din clasa obj
return s;  }
```

De exemplu, vom redefini operatorii menţionaţi pentru clasa *persoana.*

```
# include <stdio.h>
# include <iostream.h>
# include <string.h>
class persoana
{ private:
    char nume[40];
    char prenume[40];
    long int telefon;
    int virsta;
public:
 persoana(char *Nume=NULL, char *n=NULL, int v=0, long int t=0)
          {strcpy(nume,Nume);  strcpy(prenume,n);
           virsta=v; telefon= t;  }
 friend istream & operator >> (istream &s, persoana &P);
 friend ostream & operator<< (ostream &s, persoana &P);
 void set(char* Nume, char *n, int v, long int t)
     {strcpy(nume,Nume);  strcpy(prenume,n);
      virsta=v; telefon= t; };
};
istream & operator >> (istream &s, persoana &P)
{ if((s >>P.nume) && (s >>P.prenume) &&
   (s >>P.virsta) && (s >>P.telefon))
   P.set(P.nume,P.prenume,P.virsta, P.telefon);
   return s;  }
ostream & operator << (ostream &s, persoana &P)
 {return(s<<P.nume<<'  '<<P.prenume<<'  '<<P.virsta<<
          '  '<<P.telefon);}
void main()
{   persoana p;
  cout<<"Introdu datele despre persoana:"<<endl;
  cin >>p;    cout<< p;
}
```

Rezultatele îndeplinirii programului:

```
Introdu datele despre persoana:
Bradu
Maria
20
123456
Bradu Maria 20 123456
```

Operaţii de intrare/ieşire cu fişiere. În afară de fluxurile standard input/output se pot defini fluxuri ale utilizatorului definite prin:

ifstream nume pentru un flux input (citire),

ofstream nume pentru un flux output (scriere),
fstream nume flux utilizat şi pentru citire şi pentru scriere în oricare din cazurile de mai sus.

Numele va fi un obiect de tipul clasei corespunzătoare. Lucrul cu fişierele se face prin intermediul clasei *ifstream* pentru citire, respectiv *ofstream* pentru scriere. Pentru a utiliza fişiere, aplicaţiile trebuie să includă *fstream.h*. Clasele *ofstream* şi *ifstream* sînt derivate din clasa *iostream*, ca urmare toţi operatorii şi toate funcţiile descrise mai sus sînt moştenite şi de această clasă.

Definirea fluxurilor se poate face numai prin includerea în program a fişierului *fstream.h*. *ifstream*, *ofstream* şi *fstream* sînt clase. Declararea unui flux al utilizatorului este o declarare a unui obiect din clasa respectivă. Clasele pot conţine atît date, cît şi metode (funcţii). Operaţiile input/output se fac prin intermediul unor funcţii membri ale claselor respective. Sintaxele pentru constructorii acestor două clase sînt:

*ofstream Variabila(char *NumeFişier, ios::Mod);*
*ifstream Variabila(char *NumeFişier);*

Funcţia de deschidere a unui *stream* este funcţia *open*. Fiind o funcţie membru, ea va fi apelată numai prin intermediul unui obiect declarat de tipul uneia din clasele de mai sus. De exemplu:

```
#include<fstream.h>
void main()
{ ofstream scriere;
scriere.open(“disk.dat");
scriere.close();
}
```

Programul de mai sus exemplifică faptul, că nu există nici o diferenţă între lucrul obişnuit cu clase şi definirea unui flux output. Astfel, variabila *scriere* a fost declarată ca fiind de tipul (clasa) *ofstream*. Definiţia clasei *ofstream* poate fi găsită în fişierul *fstream.h.* Funcţia membru *open*, apelată prin intermediul obiectului *scriere*, creează un fişier cu numele *disk.dat*. În continuare acesta este închis prin apelul funcţiei membru *close()*, prin intermediul obiectului *scriere*. Execuţia programului va determina crearea unui fişier, care, însă nu va conţine nimic. Scrierea propriu-zisă se face cu operatorul (supraîncărcat) «, iar citirea cu operatorul », – la fel ca în cazul fluxurilor standard input/output.

Ideea de a utiliza noţiunea abstractă de flux s-a impus din dorinţa de a asigura independenţa operaţiilor input/output faţă de calculator sau de sistemul de operare. Producătorii de compilatoare livrează (împreună cu compilatorul) biblioteci, care conţin clase, funcţii, variabile, care permit ca, în mare măsură, această independenţă să fie asigurată.

Funcţia *open* are primul parametru de tip şir de caractere modificată prin modificatorul *const.* Acesta reprezintă numele fişierului, care va fi asociat fluxului. Al doilea parametru specifică modul de acces la fişierul asociat fluxului şi poate avea următoarele valori:

ios:: app; accesul se face prin adăugare la sfîrşitul fişierului,
ios:: ate; poziţionarea se face la sfîrşitul fişierului,
ios:: binary; fişierul este interpretat ca fişier binar,
ios::in; se asociază unui flux de intrare,
ios::out; se asociază unui flux de ieşire,
ios::nocreate; fişierul trebuie să existe deja,

ios::noreplace;	fişierul trebuie să nu existe,
ios::trunc;	distruge un fişier preexistent, avînd acelaşi nume.

Aceşti constructori au rolul de a deschide fişierul specificat ca parametru.

Valorile de mai sus sînt constante definite în clasa *ios*, definite în fişierul *ifstream.h.* Al treilea parametru este ignorat în cazul în care parametrul al doilea este *ios :: nocreate*.

Caracteristicile de tip *text* şi *binary* (binar) trebuie privite în legătură cu operatorii folosiţi în operaţiile input/output. Operatorii, pe care i-am folosit pînă acum (*« şi* »), utilizează codificarea ASCII a datelor. De exemplu:

```
#include<fstream.h>
void main()
{ofstream s1;
s1.open("dl.dat",ios::binary);
s1<<"text \n"<<12<<"\n"<<4.2;
s1.close();
ofstream s2("d2.dat");
s2<<"text \n"<<12<<"\n"<<4.2;
s2.close ();
}
```

Rezultatul îndeplinirii programului:

Conţinutul fişierului “d1.dat”:

```
text
12
4.2
```

Conţinutul fişierului “d2.dat”:

```
text
12
4.2
```

Funcţiile specializate pentru operaţii input/output, care realizează copii ale memoriei, sînt *read* şi *write*.

Funcţiile read şi write.

Funcţia *write* scrie într-un fişier conţinutul unei zone de memorie, care trebuie să fie adresată printr-un pointer de tip caracter. 0 zonă de memorie poate fi privită ca un şir de octeţi, indicat prin adresă de început şi lungime. De exemplu:

```
#include<fstream.h>
void main ()
{ofstream s1;
int i;
s1. open ("d1. dat", ios:: binary) ;
s1.write((char *) &i, sizeof(i)) ;
s1.write("\n",4) ;
s1.close();
ofstream s2;
s2.open("d2.dat");
s2.write((char *) &i, sizeof(i));
s2. write (" \n", 4);
```

```
s2. close ();
}
```

Funcţia de conversie *(char *)* se utilizează pentru a converti adresele oricăror obiecte la tipul adresă de caracter. Astfel, se poate de interpretat orice adresă din memorie ca fiind adresa unui şir de caractere. Operatorul *sizeof* se utilizează pentru determinarea lungimii acestei zone de memorie.

Citirea unui fişier, utilizînd funcţia *read*, se face respectînd aceleaşi principii ca în cazul funcţiei *write*.

Diferenţa între fişierele de tip *text* şi *binary* se păstrează şi în cazul utilizării funcţiilor *read* şi *write* şi constă în reprezentarea diferită a caracterelor de control. De exemplu:

```
#include<fstream.h>
void main()
{char *p="\n";
ofstream s1;
s1.open("d1.dat",ios::binary);
s1.write(p,sizeof(p));
s1.close();
ofstream s2;
s2.open("d2.dat") ;
s2,write(p,sizeof(p));
s2.close() ;
}
```

Reprezentarea datelor într-un fişier depinde exclusiv de funcţiile care se utilizează pentru înregistrarea (<< sau *write*) respectiv, citirea acestora (>> sau *read*).

De cîte ori citim un fişier, despre care ştim cum a fost creat, este bine să-l citim cu aceleaşi caracteristici şi cu aceeaşi operatori. Cînd vrem să citim din fişier, despre care nu ştim cum a fost creat, este bine să-l citim în mod binar, caracter cu caracter. De exemplu:

```
#include<fstream.h>
void main()
{int i,j;  float k;
char p[10];
ofstream s1;
s1.open("d1.dat") ;
s1« 122 « "\n" « 147;
s1« "\n abcd\n" « 9.3;
s1.close();
ofstream s2;
s2.open("d.dat",ios::binary);
s2 « i « 3 « p « k;
s2.close();
}
```

Rezultatul îndeplinirii programului:

În fişierul textual “d1.dat” s-a înscris următoarele

```
122
147
 abcd
```

9.3

În fişierul binar s-a înscris următoarele:

102273BO39¤ _3.138909e-42

În alt exemplu:

```
#include<fstream>.h>
void main()
{ char p[20];
ifstream s;
s.open("d.dat",ios::binary);
s.read(p,19);
s.close();
p[19]=NULL;     cout « p;
}
```

fişierul creat poate fi citit în mod binar, utilizînd funcţia *read*. În acest mod conţinutul fişierului este considerat copia unei zone de memorie. Deoarece fişierul a fost creat printr-o codificare ASCII, conţinutul acestuia se ia ca un şir de caractere. Instrucţiunea *read* copie conţinutul fişierului în şirul de caractere *p*. Ultima instrucţiune din programul de mai sus afişează la ieşirea standard conţinutul vectorului *p*. Atribuirea explicită *p[19]=NULL* este obligatorie pentru a marca sfîrşitul şirului de caractere.

Pentru a închide aceste fişiere, trebuie apelată funcţia membru *close()*.

Fluxuri în memorie constituie o interfaţă între program şi un dispozitiv fizic. Am utilizat fluxurile standard şi operatorii «, » şi, de asemenea, fluxurile asociate cu fişiere. Există posibilitatea de definire a unor fluxuri în memorie. Acestea sînt fişiere, care fizic sînt localizate în memorie. Un flux în memorie este un şir de caractere, care are exact aceeaşi structură ca un fişier obişnuit. Clasele, care definesc aceste fluxuri, sînt:

istrstream(input string stream);	flux input,
ostrstream(output string stream);	flux output,
strstream (string stream);	flux input/output.

***Funcţii de formatare*.** Clasele bazate pe fluxuri conţin o serie de funcţii, care oferă o mare flexibilitate a operaţiilor de scriere cu formatare. Pentru datele numerice cele mai importante sînt funcţiile, care permit tipărirea într-un cîmp de lungime fixă, alinierea (la stînga sau la dreapta) şi precizia, cu care se face afişarea.

Funcţia *width* stabileşte dimensiunea cîmpului, calculată în caractere, pe care se va face scrierea. Funcţia are un singur parametru de tip întreg şi trebuie apelată prin intermediul unui obiect de tip flux. Astfel, obiectul de tip flux în memorie, declarat prin simbolul *a*, va avea funcţia asociată *a. width*, prin care am cerut ca fiecare dată, să fie înregistrată într-un cîmp format din 8 caractere.

Funcţia *setf* are, de asemenea, un singur parametru în cazul dat *ios::right*, avînd semnificaţia de aliniere la dreapta.

Funcţia *precision* are un singur parametru de tip întreg, prin care se specifică numărul de poziţii, scrise după punctul zecimal. De exemplu:

```
#include<fstream.h>
#include<strstream.h>
void main ()
 { float a[4][5];
   int i, j;
```

```
for ( i=0;  i<4; i++)
  for ( j=0; j<5; j++)
    a[i][j]=(float) (i+2)/(j+1);
char s[200];
ostrstream scrie(s,sizeof(s));
for ( i =0; i<4; i++)
{for ( j=0; j<5; j++)
  {scrie.width(8);
   scrie.setf(ios::right);
   scrie.precision(2);
   scrie << a[i][j];
  }
   scrie << "\n";};
   s[164]=NULL;
   cout << s;
 }
```

Rezultatul îndeplinirii programului:

```
   2     1    0.67   0.5  0.4
   3   1.5       1  0.75  0.6
   4     2    1.33     1  0.8
   5   2.5    1.67  1.25    1
```

În programul de mai sus parametrul funcţiei *precision* este 2, deci numerele vor fi scrise cu două semne zecimale. Programul reprezintă un bun exemplu, prin care putem scoate în evidenţă utilitatea fluxurilor de memorie.

Rezultatul operaţiilor de intrare/ieşire poate fi testat prin intermediul a patru funcţii membri:

- *eof()* verifică dacă s-a ajuns la sfîrşitul fişierului;
- *bad()* verifică dacă s-a executat o operaţie invalidă;
- *fail()* verifică dacă ultima operaţie a eşuat;
- *good()* verifică dacă toate cele trei rezultate precedente sînt false.

Funcţiile de formatare pot fi aplicate oricărui obiect de tip *flux*, fie că este un flux standard, un flux de tip fişier sau un flux în memorie.

Întrebările pentru verificarea cunoştinţelor:

1. Care sînt tipurile de fluxuri standard definite de utilizator?
2. Scrieţi un exemplu de utilizare a funcţiilor *read* şi write a conţinutului unei zone de memorie.
3. Definiţi parametrii funcţiei membru *open* ale unei clase flux.
4. Cum se definesc fluxurile de memorie?
5. Care sînt funcţiile de formatare pentru fluxuri?

Temele pentru acasă:

1. Corectaţi greşelile şi lansaţi următoarele exemple la execuţie. Ce rezultate vor apărea pe ecran?

Exemplul 1:

```
#include <iostream.h>
```

```
define N 80
void main(0
{ inc c;
  int cnt=0; charcnt=0;
  While(1)
        { c=cin.get();
        if (c==EOF) break;
        if (c='/' && cin.peek() =='/')
              { cnt++;
              cin.ignore(n,"\n");
              charcnt++=cin.gcount();
              charcnt++;
              }
        }
cout<<"In" << cnt << " comentarii avem " << charcnt << "caractere";
}
```

Exemplul 2 .

```
#include <iostream.h>
void main()
{While(1)
        { char c; cin.get©;
        if(cin.eof()) break;
        cout.put( c);
}
```

Exemplul 3.

```
#include <iostream.h>
void main()
{cout <<setfill('%'}<<setw(4)<< 17;}
```

Exemplul 4.

```
#include <iostream.h>
#include <fstream.h>
void main()
{ int I=42;
fstream f ("dat.txt",ios::out|ios::binary);
f.seekp(17,ios::beg);
f.write((const char*)&I,2) ;
f.close();
f.open("dat.txt", ios::in|ios::binary);
f.seekg(17,ios::beg);
f,read((char*)&I,2);
f.close();
cout<<I;
}
```

Exemplul 5.

```
#include <iostream.h>
#include <strstream.h>
```

```
void main()
{char buffer [80];
strstream s (buffer,80,ios::out);
float pret=123.45;
s<<"Preţul stocului este de";
s;;serw(6)<<setprecision(2) ;
}
```

Sarcina pentru lucrări de laborator:

1. Scrieţi un program care compară două fişiere date.

2. Scrieţi un program care calculează numărul de elemente ale unui fişier care sînt mai mici ca valoarea medie aritmetică a tuturor elementelor acestui fişier.

3. Scrieţi un program care tipăreşte toate cuvintele diferite de ultimul cuvînt dintr-un fişier. Cuvintele din fişier sînt separate prin virgulă, iar.după ultimul cuvînt se pune punct.

4. Scrieţi un program care determină numărul maximal şi cel minimal dintr-un şir de 100 de numere aleatoare dintr-un fişier. Subconsecutivitatea de elemente dintre numărul maximal şi cel minimal determinat să se înregistreze într-un nou fişier.

5. Scrieţi un program care formează un fişier nou după următoarea legitate: din trei fişiere date mai întîi se selectează numerele divizibile la 3, la 5 şi la 7, apoi numerele pozitive pare de pe locuri impare.

6. Scrieţi un program care formează un fişier nou după următoarea legitate: din trei fişiere date mai întîi se selectează numerele negative, zerourile, apoi numerele pozitive.

7. Scrieţi un program care ordonează lexicografic o consecutivitate de înregistrări (dintr-un fişier) de tipul

```
struct { char nume [30];
int ani}  înregistrare;
```

Rezultatul ordonării să se înscrie într-un nou fişier.

8. Scrieţi un program care din două fişiere ordonate descrescător se va forma unul în care se va păstra ordinea descrescătoare de sortare.

9. Scrieţi un program care sortează lexicografic cuvintele dintr-un text. Pentru fiecare element sortat să se indice numărul de repetări ale cuvîntului în textul dat.

10. Scrieţi un program care calculează suma înmulţirii numerelor vecine pe pe locuri pare dintr-un fişier dat.

11. Scrieţi un program care va tipări în ordine inversă subconsecutivitatea de numere dintre valoarea minimă şi maximă ale unei consecutivităţi de numere citite dintr-un fişier.

12. Scrieţi un program care formează un fişier nou din cel dat după următoarea legitate: elementele fişierului nou se obţine din inversul numerelor din fişierul dat.

13. Scrieţi un program care determină frecvenţa cu care a fost generat fiecare element în fişierului creat. Valorile elementelor sînt cuprinse între 1 şi 100.

14. Scrieţi un program care determină numărul maximal şi cel minimal din numerele unui fişier dat. Să se determine elementele mai mari ca cel minimal şi mai mici ca numărul maximal.

15. Scrieţi un program care efectuează reformatarea unui fişier textual în felul următor. Lungimea rîndului de caractere în fişierul nou are lungimea de 60 de caractere. Dacă în rîndul dat se depistează punctul, restul rîndului din fişierul dat se scrie din rînd nou.

16. Scrieţi un program care din două fişiere date ordonate crescător se va forma unul nou, în care se va păstra ordinea crescătoare de sortare.