C10: Biblioteca standard C++

Cuprins:

Standard C++ Library

- String-uri
- Stream-uri

C10: Biblioteca standard C++

Biblioteca Standard C++ - Standard C++ Library

Standard C++ Library- cuprinde toate bibliotecile standard C precum si biblioteci dedicate C++.
Acest material (si C11) face o introducere (nu o sa descrie in amananunt) a claselor si functiilor din aceasta biblioteca.

Standard C++ Library

- clasa **string** – operatii de procesare text
- clase pentru procesarea fisierelor si operatii consola: **iostream, fstream, etc**
- **STL (Standard Template Library)** - clase template – containere, iteratori si algoritmi (C11)
- si alte capabilitati

Bibliografie:

- Bruce Eckel, *Thinking in C++ - Volume 2*
- Bjarne Stroustrup's *The C++ Programming Language*
- http://www.cplusplus.com/reference/iolibrary/
- http://www.cplusplus.com/reference/string/

C10: Biblioteca standard C++

Clasa string - #include <string>

- un obiect de tip string reprezinta un sir de caractere + informatii despre localizarea lui in memorie si dimensiunea sa.
- crearea si initializarea de obiecte de tip string se face folosind constructorii pusi la dispozitie de aceasta clasa (mai jos sunt listati o parte dintre ei):

string()	**Creaza un obiect string default, cu lungimea 0**
string(const char *s)	Creeaza si initializeaza un string folosind un sir de caractere
string(size_type n, char c)	Creeaza un string de lungime n si initializeaza fiecare element cu c
string(string s, int poz_init, int dim)	Creeaza un alt string din s, incepand cu caracterul de pe pozitia poz_init, de lungime dim (sau pana la finalul lui s)
string(const string& s)	Constructor de copiere

```
#include <string>
#include <iostream>
using namespace std;
int main() {
string s1;
cout<<"s1: "<<s1<<endl; //s1:
//operator<< e supradefinit pentru tipul de date string

string s2("al doilea string");
cout<<"s2: "<<s2<<endl;//s2: al doilea string

string s3(s2);
cout<<"s3, copia celui de-a doilea string: "<<s3<<endl; // s3, copia celui de-a doilea string:
                                                        // al doilea string
string s4(5,'x');
cout<<"s4: "<<s4;  // s4: xxxxx

string s5 = "al 5-lea"; //operator= e supradefinit pentru tipul de date string cu al doilea
                        //operand string, char* sau char
cout<<"s5: "<<s5<<endl;  //s5: al 5-lea

string s6(s5, 3, 5);
cout<<“s6: “ << s6;  // s6: 5-lea

string s7(s5, 3, 10);   //se opreste la ultimul caracter din s5
cout<<“s7: “ << s7;  // s7: 5-lea
return 0;}
```

C10: Biblioteca standard C++

Operatorii supradefiniti pentru clasa string:

bool operator==(const string & str1, **const string & str2)**	Verifica daca doua stringuri contin acelasi text; implementat si pentru cazul cand un parametru e de tip char*
bool operator<(const string & str1, const string & str2)	-”-
bool operator>=(const string & str1, const string & str2)	-”-
bool operator>(const string & str1, const string & str2)	-”-
bool operator!=(const string & str1, const string & str2)	Testeaza daca doua stringuri contin texte diferite
string& string::**operator**=(const String &str2)	Operator de atribuire (poate sa functioneze si cu un parametru de tip char* sau char)

string operator+(const string & str1, const string &str2)	Supradefinirea operatorului + pentru clasa string; Concateneaza doua stringuri – iar stringul rezultat are dimensiunea egala cu suma dimensiunilor celor doi parametri Supradefinit si pentru cazul cand unul dintre parametri e de tip char*
string& string::**operator**+=(const string & str2)	Adauga la finalul primului string, pe cel de-al doilea, cu redimensionare. Functioneza si cu al doilea parametru de tip char*
ostream& operator<<(ostream& dev, const string & s)	Supradefinirea operatorului << pentru clasa string
istream& operator>>(istream& dev, string & s)	Supradefinirea operatorului >> pentru clasa string

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

int main() {

string s1 = "ABC"; // initializare cu un sir de caractere
string s2=s1; // initializare cu un alt string

cout<<"s1 concatenat cu s2: "<<s1+s2<<endl;   // s1 concatenat cu s2:  ABCABC

cout<<"este "<<s1<<" identic cu "<<s2<<"? "<<(s1==s2)<<endl;  //1

string s3=s2+'D';//concatenare cu un caracter
cout<<s3<<endl; // ABCD

s2+="EFG"; //concatenare cu un sir de caractere
cout<<s2; // ABCEFG

string s4;
cin>>s4; //stie sa aloce spatiu

return 0;
}
```

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

int main() {

string s1 = "A"; // initializare cu un sir de caractere
string s2="B"; // initializare cu un sir de caractere
cout<<(s1<s2);//1

string s3("AB");  //constructor cu parametru char*
cout<<(s1<s3);//1

string s4(s3); //constructor de copiere
cout<<(s4==s3);//1

cout<<(s4!=s3);//0

return 0;
}
```

C10: Biblioteca standard C++

Functii membre ale clasei string

string::substr	Functie membra a clasei **string** **Primeste ca argumente:** **- pozitia de inceput a subsirului** **- lungimea noului string** Amandoua argumentele au valori default – inceputul si, respectiv, finalul sirului.

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

int main() {
string s1 = "ABCEFG"; // initializare cu un sir de caractere

string s2=s1.substr(3,2); //substring de 2 caractere, extras incepand cu pozitia 3 din s1
cout<<s2<<endl;   // EF

string s3=s1.substr(); //substring extras incepand cu pozitia 0, de lungimea stringului s1
cout<<s3<<endl;  //ABCEFG

string s4=s1.substr(4,5);
cout<<s4<<endl;   //FG
 //daca lungimea subsirului depaseste sirul, se extrag caracterele pana la finalul sirului;
//fara eroare sau aruncare de exceptie

return 0;
}
```

C10: Biblioteca standard C++

Aflarea dimeniunii. Adaugare la final, inserare si concatenare de stringuri

Clasa string din C++ - permite adaugare / eliminare de caractere fara a fi necesar sa redimensionam manual sirul rezultat.
De asemenea, pune la dispozitie metode pentru aflarea dimensiunii/lungimii si capacitatii obiectelor string si rezervare de spatiu.

size length	Lungimea sirului ; cate caractere avem stocate
capacity	Capacitatea sirului; cate caractere putem stoca fara realocare (automata)
insert	Adauga incepand cu pozitia specificata o secventa de caractere sau un string, cu redimensionarea sirului initial
append	Adauga la finalul sirului o secventa de caractere sau un string, cu redimensionarea sirului initial
reserve	Creste capacitatea sirului la o dimensiune dorita

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

int main() {

string s="ABC";
cout<<s<<endl;

// Cate caractere avem stocate?
cout << "dimensiune = " << s.size() << endl;//3

// Cate caractere se pot stoca fara realocare?
cout << "capcitate = " << s.capacity() << endl;//3

// inserare string dupa primul element din s:
s.insert(1, "123");
cout << s << endl;//A123BC

cout << "dimensiune = " << s.size() << endl;//6
cout << "capacitate = " << s.capacity() << endl;//6

// rezervare spatiu = 20
s.reserve(20);

// adaugare de string la finalul lui s:
s.append("DEF");
cout << s << endl;  //A123BCDEF

cout << "dimensiune = " << s.size() << endl;//9
cout << "capacitate = " << s.capacity() << endl;
                                                //20
return 0;
}
```

C10: Biblioteca standard C++

Inlocuire si cautare de elemente din/in string

replace	Exista mai multe versiuni supradefinite ale acestei functii **Cea mai simpla versiune are 3 argumente**: - Inceputul zonei pe care o modificam in cadrul stringului - Cate caractere se elimina din string - Sirul care se insereaza pe pozitia dorita
find	Gaseste si returneaza pozitia pe care incepe o secventa cautata. Primeste ca parametru secventa respectiva. Daca secventa nu e gasita este returnat : **string::npos** (constanta a clasei string)

```
#include <iostream>
#include <string>
using namespace std;
int main() {
string s("ABC ABC ABC");

int loc = s.find('A');//cauta prima pozitie a caracterului A
//inlocuieste incepand cu locatia loc, urmatoarele string("A").size() caractere cu X
s.replace(loc, 1, "X");
cout<<s<<endl; //XBC ABC ABC

string secv="ABC";
loc = s.find(secv); //cauta sirul ABC
s.replace(loc, secv.size(), "Y"); //inlocuieste secv cu Y
cout<<s<<endl; //XBC Y ABC
cout << "dimensiune = " << s.size() << endl; //9
cout << "capacitate = " << s.capacity() << endl; //11

cout<<(s.find("M")==string::npos);//1    //verific ca M nu se gaseste in s

loc = s.find(secv); //cauta sirul ABC
s.replace(loc, secv.size(), "1234567"); //inlocuieste-l cu 1234567
cout<<s<<endl; // XBC y 1234567
cout << "dimensiune = " << s.size() << endl; //13
cout << "capacitate = " << s.capacity() << endl; //13


return 0;}
```

OBS! Ar trebui mereu testat ca s-a gasit secventa s.find!=string::npos

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

//functie care inlocuieste in sirul s toate secventele secv cu cuv; returneaza cate modificari au
//fost facute   (OBS cuv nu contine secv)
int replaceAll(string& s, const string& secv, const string& cuv)
{int cate = 0;
 int loc;
 while ((loc=s.find(secv))!=string::npos)//cat timp se gaseste o secventa nemodificata
 {                                       //retin pozitia de inceput a secventei in loc
       s.replace(loc, secv.size(), cuv); //inlocuieste secventa cu cuv
       cate++; //incrementeaza numarul de modificari
 }
 return cate;
 }

int main()
{

   string s("ABC ABC AMN AMB ABI ABC");
   cout<< "am facut " <<replaceAll(s,"AB","X") <<" inlocuiri si avem sirul: "<<s;
            // am facut 4 inlocuiri si avem sirul: XC XC AMN AMB XI XC
   return 0;
   }
```

Alte functii de cautare in string

Familia de functii **find (de cautare)** din clasa string contine functii care localizeaza un caracter sau secventa de caractere intr-un string.

find	Cauta intr-un string un caracter sau o secventa de caractere si returneaza prima pozitie unde este intalnita secventa in string sau **npos** daca nu se gaseste nicio secventa identica.
find_first_of	Cauta intr-un string si returneaza pozitia primului element gasit care corespunde oricarui caracter specificat in grup; daca nu gaseste niciunul dintre caractere returneaza **npos**.
find_last_of	Cauta intr-un string si returneaza pozitia ultimului element gasit care corespunde oricarui caracter specificat in grup; daca nu gaseste niciunul dintre caractere returneaza **npos**.

find_first_not_of	Cauta intr-un string si returneaza pozitia primului element gasit care NU corespunde unui caracter specificat in grup; daca nu gaseste niciunul dintre caractere returneaza npos.
find_last_not_of	Cauta intr-un string si returneaza pozitia ultimului element gasit care NU corespunde unui caracter specificat in grup; daca nu gaseste niciunul dintre caractere returneaza **npos**.
rfind	Cauta intr-un string de la final la inceput un caracter sau o secventa de caractere si returneaza prima pozitia unde este intalnita secventa in string sau **npos** daca nu se gaseste nicio secventa identica.

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

int main() {
string s = "ABCEFCG"; // initializare cu un sir de caractere

int loc=s.find_first_of( "\B\G");
cout<<loc;//1

loc=s.find_last_of("\C");
cout<<loc;//5

loc=s.find_first_not_of("\A\C");
cout<<loc;//1

loc=s.rfind("C");
cout<<loc;//5

return 0;
}
```

OBS: Nu exista functii in clasa **string** cu care putem transforma caracterele din litere mici in litere mari sau invers, dar se pot folosi functiile din biblioteca Standard C: **toupper( )** and **tolower( )**, cu care putem implementa niste functii care transforma fiecare caracter al unui string in litera mare/mica:

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

// Transformare string s– in string cu litere mari
inline string literaMare(const string& s) {
string var(s);
for(size_t i = 0; i < s.length(); ++i)
var[i] = toupper(var[i]);
return var;
}
// Transformare string s– in string cu litere mici
inline string literaMica(const string& s) {
string var (s);
for(int i = 0; i < s.length(); ++i)
var[i] = tolower(var[i]);
return var;
}
```

```
int main()
{

   string s("ABC ABC AMN AMB ABI ABC");
   cout<<literaMica (s);
   cout<<literaMare (s)<<endl;
   cout<<literaMica (s);

   return 0;
   }
```

Observatie: operator[] este supradefinit pentru string

C10: Biblioteca standard C++

Stergere de caractere dintr-un string

- **erase( )** - functie membra a clasei string
  - doua argumente: de unde incepem stergerea (default 0) si cate caractere stergem (default **npos**)
  - daca sunt specificate mai multe caractere decat se pot sterge, nu o sa avem eroare

Interschimbare intre doua obiecte de tip string

- **swap( )** - functie membra a clasei string care interschimba continutul a doua obiecte string

Comparare de substringuri din doua obiecte de tip string

Pentru a compara un substring dintr-un obiect de tip string cu alt substring din alt string se poate folosi una din versiunile supraincarcate ale functiei **compare( )**:
s1.compare(s1StartPos, s1NumberChars, s2, s2StartPos,s2NumberChars);

Functia at()

- returneaza caracterul de pe pozitia precizata ca argument

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

int main() {
string sir1(" ABC");
string sir2("ABC");
string sir3(sir1);
cout<<sir1.compare(sir3)<<endl;    //0

cout<<sir1.compare(sir2)<<endl;     //-1

cout<<sir2.compare(0, 1, sir1, 1, 1) <<endl;   //0

sir1.swap(sir2);
cout<<sir1<<" are dimensiunea"<<sir1.size()<<"si capacitatea"<<sir1.capacity()<<endl;
//ABC are dimensiunea 3 si capacitatea 3
cout<<sir2<<" are dimensiunea"<<sir2.size()<<"si capacitatea"<<sir2.capacity()<<endl;
// ABC are dimensiunea 4 si capacitatea 4

cout<<sir1.at(1)<<endl; //B

sir1.erase(0,2);   //sterge 2 caractere din s1 incepand cu pozitia 0
cout<<sir1;   //C
return 0;    }
```

Observatie:

Desi se poate deriva din clasa string, nu este indicat sa se faca astfel de derivari. Clasa nu a fost realizata pentru acest scop. (Unul dintre motive pentru care aceasta clasa nu este una polimorfica este faptul ca destructorul clasei nu e virtual.)

Utilizarea clasei string in agregarea este, in schimb, un procedeu extrem de larg folosit. **Ex?**

Tema:

Ce fac urmatoarele functii?

const char* data() const;
bool empty() const;
void push_back(char c);
void pop_back();
- stoi(), stol(), stod(),…

C10: Biblioteca standard C++

Fluxuri de intrare si iesire

In C++ exista o ierarhie de clase predefinite, care asigura un suport eficient pentru operatiile de intrare/iesire, atat cu consola cat si cu discul.

Conceptul de baza este cel de dispozitiv logic de intrare/iesire (flux de intrare/iesire).

Aceste fluxuri sunt numite "**stream"-uri**.

Fisierele antet care asigura interfata cu clasele predefinite sunt <iostream> (operatii consola) si <fstream> (operatii fisiere).

Exista doua clase de baza pentru astfel de operatii: streambuf si ios.

Clasa **streambuf** ofera operatii minimale de acces la un dispozitiv de intrare/iesire.

Clasa **ios** contine adresa unui obiect de tip streambuf, o variabila de stare (pentru memorarea erorilor), variabile de control al formatului si o colectie de functii.

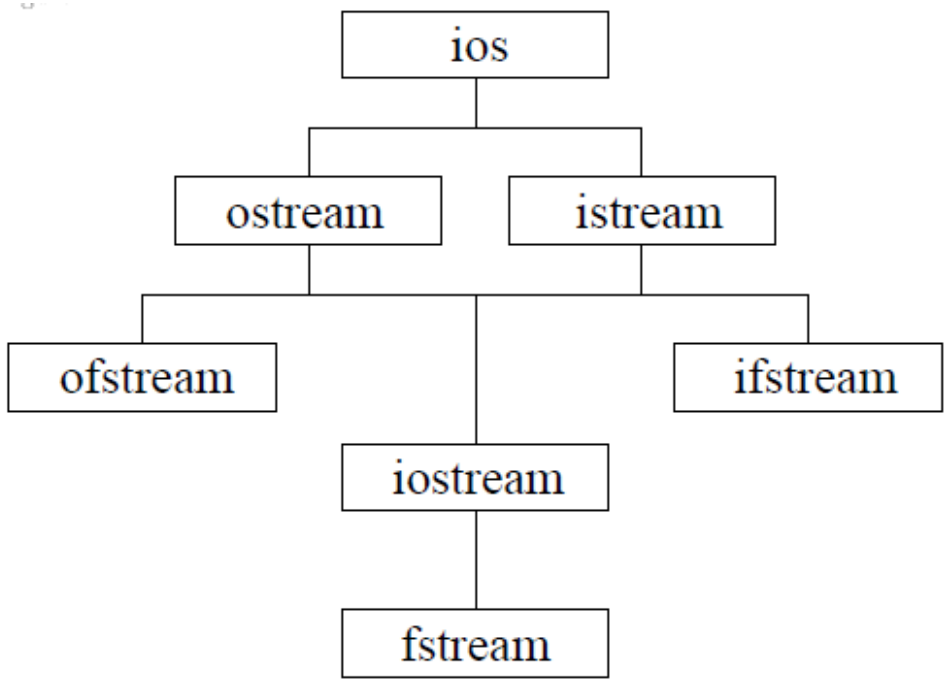
Din ios este dezvoltata o ierarhie de clase precizate in ierarhia din figura.

Operatii generale cu streamuri

-clasa istream (transfer de la un obiect de tip streambuf); obiectele de acest tip reprezinta dispozitive de intrare;

-clasa ostream (transfer catre un obiect de tip streambuf); obiectele de acest tip reprezinta dispozitive de iesire;

-clasa iostream, derivata din istream si ostream, pentru operatii bidirectionale (dispozitive de intrare si iesire).



Operatii cu fisiere

-clasa ifstream (pentru citire din fisiere);

-clasa ofstream (pentru scriere in fisiere);

-clasa fstream (pentru operatii de citire si scriere in fisiere).

Operatii generale

Exista urmatoarele dispozitive logice (obiecte) predefinite (in namespace-ul std) :

cin: intrare consola
cout: iesire consola
cerr: eroare standard consola

In clasele **istream** si **ostream** sunt supradefiniti operatorii de **insertie** si **extractie:** **<<** si **>>** pentru tipurile de date de baza(int, float, char *, char, double, etc.).

Acesti operatori sunt definiti dupa modelele:

```
ostream& operator<< (ostream&, const tip_de_baza&);
istream& operator>> (istream& ,tip_de_baza&);
```

Acesti operatori pot fi supradefiniti pentru tipuri de date definite de utilizator dupa modelele:

```
friend ostream & operator<< (ostream &, const  TIP &);
friend istream & operator>> (istream &, TIP &);
```

Pe langa operatorii << si >>, clasele istream si ostream au, respectiv, implementate functiile elementare **get** si **put** (la nivel de octet).

Acestea intorc referinte la stream-ul respectiv, avand prototipurile:

```
ostream& put (char c);   // Scrie c
istream& get (char& c);  // Citeste c
```

Aceste functii se apeleaza, dupa modelul de mai jos (presupunand ca dispozitivul de lucru este consola):

```
char c;
cin.get(c); //citesc c
cout.put(c); //afisez c
```

Functii de citire/modificare a starii

bool eof();	specifica daca este sfarsit de fisier
bool bad();	specifica daca a existat o eroare la ultima operatie
bool good();	specifica daca operatia anterioara s-a incheiat cu succes
void clear();	sterge indicatorii de eroare

Un stream poate fi testat in raport cu eventualele erori folosind aceste functii (sau prin testarea numelui sau). O valoare diferita de zero inseamna fara eroare.

Un program de copiere cin --> cout poate fi scris:

```
char c;
while (cin.get(c))
    cout.put(c);
```

Aici se testeaza de fapt starea streamului cin dupa apelul functiei membre get().

Controlul formatului

Operatorii << si >> presupun niste formate implicite.
Formatul de intrare/iesire al datelor poate fi controlat prin manipulatori.

Manipulatori fara parametri

Se folosesc in operatii de I/O, dupa sintaxa:

```
out << manipulator<<…;
in >> manipulator>>…;
```

Ei reprezinta operatii care pot fi inserate in secvente de citire/scriere.
Ei seteaza o serie de parametri interni ai obiectului de tip stream, care fac ca urmatoarele operatii de citire sau scriere sa aiba loc cu un alt tip de conversie.

De exemplu, la citirea/scrierea de variabile de tip int, se presupune implicit ca reprezentarea externa a numerelor este in baza 10. O scriere de forma:

cout << hex << n;

va face ca variabila intreaga n sa fie afisata in hexazecimal.

Similar, o citire de forma:

cin >> oct >> i;

va considera ca i este citit de la consola in baza 8.

Au efect pana la schimbarea monipulatorului:

```
cout<<hex<<10<<endl<<11; //a  b
cout<<dec<<12;  //12
```

Manipulatorii fara parametri sunt:

-**dec** (specifica baza 10)
-**oct** (specifica baza 8)
-**hex** (specifica baza 16)
-**endl** (end of line, adica insereaza '\n')
-**ws** (ignora spatiile goale: space, tab)
-**ends** (end of string, adica insereaza '\0')
-**flush** (goleste bufferul de iesire)

//Ex. utilizare:

```
int x;
cin>>hex>>x;//F
cout<<dec<<x;//15

string s,s1;
cin>>ws>>s; // introduc de la tastatura textul : ”xx yy zz”
cout<<s;      //xx

cin>>s1;     //fara sa scriu ceva de la tastatura in s1 se citeste in continuare yy
cout<<s1;   //yy

cout<<flush;
```

Manipulatori cu parametri (definiti in biblioteca iomanip.h)

setbase (int n)	Baza de numeratie n (8,10,16)
setprecision (int n)	Precizia pentru numere reale este setata la n cifre (. nu e luat in considerare)
setw (int n)	Seteaza latimea campului pentru afisare la n spatii
setfill (char c)	Caracter de umplere c (pozitiile vide din camp vor fi completate cu c)

Exemplu:

```
 float x = 12.345678911111;
//vreau sa afisez 7 cifre din x, care sa ocupe 12 pozitii
// iar pozitiile goale sa fie completate cu caracterul .
 cout <<setfill('.')<< setw(12) << setprecision(7) << x << endl;   //....12.34567
```

Operatii cu fisiere

Se folosesc clasele **ifstream, ofstream** si **fstream** si functiile lor membre.

Sunt pusi la dispozitie constructori fara/cu parametri pentru toate tipurile de fisiere (intrare, iesire si intrare-iesire).

ofstream::ofstream(const char*nume, int mod=ios::out, int acces=0)

- nume - numele fisierului
- mod – modul de acces
- acces – tipul de protectie/acces
  acces = 0 (normal), 1 (read-only), 2 (ascuns), 3 (sistem).

//fisierul cu numele(calea) precizat(a) este deschis in modul si cu protectia dorita

Exista si o functie speciala de deschidere, similara ca efect cu constructorul:

void open (const char *nume_fisier, int mod, int acces)

Modurile sunt predefinite in clasa ios si pot fi:
ios::in (intrare)
ios::out (iesire)
ios::ate (deschidere cu pozitionare la sfarsitul fisierului)
ios::app (append, adaugare la sfarsitul fisierului)
ios::binary (deschidere in mod binar; implicit este in mod text)

Sunt implicite modurile **ios::in** pentru ifstream, **ios::out** pentru ofstream si acces = 0.

Modurile pot fi combinate cu sau logic la nivel de bit (|).

In urma deschiderii, variabila stream pentru care s-a apelat constructorul sau functia open trebuie testata pentru erori la deschidere.
Este indicat sa se testeze acest lucru imediat dupa apelul constructorului sau functiei open:

```
if (!numefis) cout<<“eroare”;
```

Functia pentru inchiderea unui fisier este:

```
void close();
```

Ea este apelata de destructorul pus la dispozitie de clasele care lucreaza cu fisiere, ceea ce inseamna ca fisierul va fi inchis automat la finalul duratei de viata a variabilei de tip fisier, chiar daca nu apelam functia close().

//Exemplu: program de copiere dintr-un fisier in altul

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

int main() {
    ofstream ofis("date1.txt",ios::app); //apel constructor cu parametri; acces implicit-normal
    ifstream ifis; //apel constructor fara parametri

    if(!ofis)   cout << "Eroare la deschidere date1.dat";

    ifis.open("date2.txt");  //deschidere cu functia open
    if(!ifis)    cout<< "Eroare la deschidere date2.dat";

    char c;
    while (ofis && ifis.get(c))    //cat timp starea lui ofis este buna si se poate citi din ifis
                                   //se citeste un octet din ifis in variabila c
    ofis.put(c);                   //si se scrie in ofis; la final – ofis deschis in mod ios::app

    ofis.close();
    ifis.close(); //pot sa lipseasca

    return 0;}
```

Putem citi o linie dintr-un fisier text (cau cin) cu functia:

getline(unsigned char *buf, int l, char delim);

- care citeste in buf cel mult l-1 caractere sau pana la intalnirea caracterului delim;
- citirea se face de un dispozitiv de intrare (cin sau fisier);
- al treilea parametru este implicit caracterul ‘\n’(linie noua);
- functia pune '\0' la sfarsitul liniei citite.

O bucla de citire la nivel de linii dintr-un fisier text si afisare la consola s-ar putea scrie astfel:

```
ifstream ifis("date2.dat");
char buf [256];
 while (!ifis.eof()) {   //cat timp nu am ajuns la finalul fisierului
            ifis.getline(buf, 256, '\n');
            cout << buf << endl;
 }

//cum as putea sa fac stocarea liniilor in alt fisier?
```

```
// citire la nivel de linii dintr-un fisier text si scriere in alt fisier
#include <fstream>
#include <iostream>
using namespace std;
int main(void) {
    ofstream ofis("date1.txt",ios::app);
    ifstream ifis("date2.txt");

    if(!ofis)  cout << "Eroare la deschidere date1.dat";
    if(!ifis)   cout << "Eroare la deschidere date2.dat";

       int d=100;
       char buf [d];

       while (!ifis.eof()) {
              ifis.getline(buf, d, '\n');
              ofis << buf << endl;  //in ofis se insereaza, la final, liniile citite din ifis
       }

   // ofis.close();
   // ifis.close();
   //se inchid oricum la iesirea din program, cand se apeleaza destructorii

    return 0;
}
```

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

class Persoana
{    string nume=“doe”;
     int varsta=0;
   public :
     Persoana(){}
     Persoana(string s,int v):nume(s),varsta(v){}
     friend ostream& operator<<(ostream &d, const Persoana & x){
          d <<x.nume<<" "<<x.varsta<<endl;
          return d;
     }
}; //tema implementati operator>> si cititi din date.txt

int main(void) {
fstream fis("date.txt“, ios::app|ios::out); //fisier intrare-iesire; adaugare la final
if(!fis) cerr << "Eroare la deschidere date1.dat";

Persoana p(“ana",13),p1;
fis<<p<<p1;  // fiecare obiect de tip Persoana este inserat in fisierul fis (pe linii separate)

return 0;
}
```

Diferenta intre modul binar si text

Fisierele text presupun cunoasterea formatului in care sunt scrise/citite datele. Operatiile de citire, scriere se pot face usor cu operatorii << si >>, si functii ca getline. Dar ca sa ajung la linia x-> trebuie sa parcurg primele x-1 linii.

In cazul fisierelor binare nu se pot folosi aceste functii, deoarece datele nu sunt formatate in vreun fel anume.

Clasele specializate pentru fisiere binare includ doua functii pentru citirea si scrierea secventiala de date binare: **write** si **read**.

Aceste functiile nu fac niciun fel de prelucrari - sunt functii de scriere si citire "oarbe" . Au prototipurile:

```
istream& read(unsigned char *buf, int n);
ostream& write(const unsigned char *buf, int n);
```

care citesc/scriu din/in fisierul respectiv n octeti in/din buf.

Dar, pot sa ma duc direct la o anumita pozitie intr-un fisier binar.

Functii de control al pozitiei in fisier

Functiile de pana acum presupuneau tacit ca citirea si scrierea se face secvential.
Ca sa citim al 100-lea octet dintr-un fisier trebuia obligatoriu sa fi citit cei 99 de octeti anteriori (analog la scriere).
Uneori dorim sa citim/scriem d/intr-un fisier existent, d/intr-o anumita pozitie.
Metodele seekg (la citire din ifstream) si seekp (la scriere in ofstream) realizeaza pozitionarea absoluta in fisier binar. Prototipul lor este:

```
ifstream& seekg(long poz, int reper);
ofstream& seekp(long poz, int reper);
```

unde poz este pozitia (in octeti), iar reper poate fi una din constantele:

ios::beg ios:: cur ios::end

(pozitionare fata de inceput, fata de pozitia curenta, fata de sfarsit).

De exemplu, daca dorim sa ne plasam in fisierului f, deschis in mod binar pentru scriere, dupa al 10-lea octet fata de inceput si sa inseram sirul de octeti c vom utiliza urmatoarea secventa de cod:

```
ofstream f("datex.dat“,ios::binary);  //implicit pozitionarea e la inceputul fisierului
f.seekp (10, ios::beg);
f.write(c,sizeof(c));
```

Tema

Implementati clasa Matrice de fractii care are supradefiniti operatorii: = si << si constructorul de copiere, destructorul, etc.

Stocati obiectele de tip matrice create - intr-un fisier.

Realizati o metoda care permite citirea de matrici dintr-un fisier si returneaza un string.

Matricile apar in fisier sub forma:

lin col el[0][0] el[0][1] … el[0][col-1] …. el[lin][0] el[lin][1] … el[lin-1][col-1]

Ex: 2 3 0 1 3 4 5 8 <=> 0 1 3
4 5 8

Realizati o metoda care parseaza stringul si il transforma in obiect de tip matrice si afisati numarul de linii, coloanele si elementele.