Laborator 7 – Stream-uri

Tema 7.1

Analizaţi programul din fişierele `EX7.CPP, EX7.H, CLASE.CPP, CLASE.H` din anexa 7.

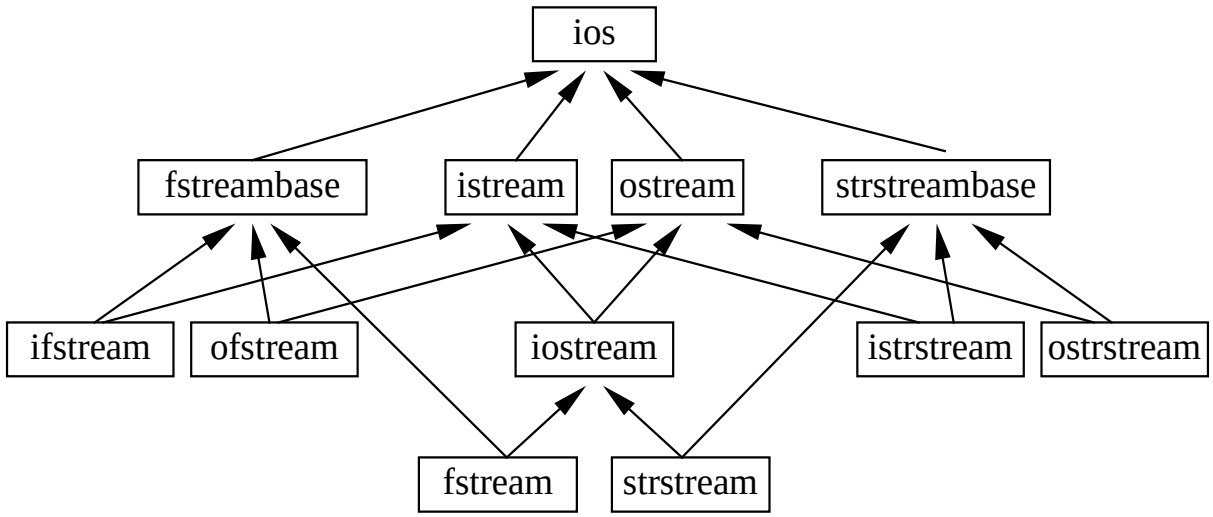
Tema 7.2

Să se implementeze operatorii de inserţie "<<" care să permită, la apăsarea tastei 5 (având codul '5'), salvarea într-un stream a obiectului de tip `MULŢIME` (şi a celor de tip `FIGURA` incluse). Se va folosi ca stream un obiect de tip ofstream mapat pe fisierul „FISIER.CFG".

Consideraţii teoretice 7.2

La baza bibliotecii de streamuri stă **redefinirea operatorilor** << **şi** >> care, în acest context, se numesc operatori de **inserţie** (<<) şi respectiv de **extracţie** (>>). Această denumire sugerează, împreună cu simbolul, semnificaţia operatorului: variabila respectivă se inserează în respectiv se extrage din streamul respectiv.

Deşi nu vom putea prezenta o descriere completă a ierarhiilor de clase ale bibliotecii I/O prezentăm în figura următoare o parte a acestor ierarhii, cu notaţia consacrată în UML în care sensul săgeţilor indică spre clasa părinte.



Inserţia formatată a tipurilor standard

După cum se cunoaşte, pentru operaţiile de ieşire cu format se folosesc **operatorii de inserţie** <<, definiţi pentru fiecare din **tipurile de date standard** existente în limbaj. Declaraţiile acestora se găsesc în `<iostream.h>` şi le prezentăm în continuare:

```
class ostream : virtual public ios
{
//  . . . . . .
public:
       // inserează un caracter
```

```
    ostream& operator<< (  signed char);
    ostream& operator<< (unsigned char);

       // inserează reprezentarea ascii a valorii numerice
    ostream& operator<< (short);
    ostream& operator<< (unsigned short);
    ostream& operator<< (int);
    ostream& operator<< (unsigned int);
    ostream& operator<< (long);
    ostream& operator<< (unsigned long);
    ostream& operator<< (float);
    ostream& operator<< (double);
    ostream& operator<< (long double);

       // inserează un şir
    ostream& operator<< (const   signed char *);
    ostream& operator<< (const unsigned char *);

       // inserează reprezentarea ascii a valorii pointerului
    ostream& operator<< (void *);
//  . . . . . .
};
```

Cele de mai sus justifică posibilitatea inserării tipurilor standard pe stream ca în exemple binecunoscute:

```
    cout<<"x="<<x<<"c="<<c;   //evident x si c sunt variabile definite anterior
```

Inserţie pe stream pentru tipuri definite de utilizator

O altă facilitate a bibliotecii de streamuri este că permite (pe lângă inserţia tipurilor standard) în mod unitar inserţia tipurilor definite de utilizator.

Deşi mecanismul nu este prea complicat, având la bază simpla redefinire a operatorilor de inserţie pentru tipul definit de utilizator, eleganţa acestui mecanism face ca această facilitate să fie una foarte răspândită.

Exemplificăm în cele ce urmează acest mecanism pentru o clasă proprie `PUNCT` (simplificată):

```
#include <iostream.h>

class PUNCT
   {
   int x,y;
   friend ostream& operator<<(ostream& o, PUNCT& ob)
      {
          o<<ob.x<<" "<<ob.y; return o; // inserare membrii PUNCT
      }
   };
```

situaţie în care este posibil următorul cod:

```
PUNCT un_punct;
         ...
         cout<<un_punct;   //insertia tipului propriu pe stream
         ...
```

Facem următoarele observaţii:

- operatorul de inserţie s-a redefinit ca şi funcţie friend pentru clasa `PUNCT`. Redefinirea prin funcţie friend este esenţială pentru a permite utilizarea operatorului cu stream-ul în partea stângă şi `PUNCT`-ul în dreapta, fapt care nu ar fi putut fi posibil dacă redefinirea s-ar fi făcut prin metodă (când `PUNCT`-ul era obligatoriu operandul stâng). **Necesitatea posibilităţii redefinirii operatorilor şi prin funcţii friend este încă o dată justificată** (programatorul nu are acces la codul stream-urilor pentru a defini acolo operatorul prin metodă).
- inserţia obiectului `un_punct` s-a realizat simplu şi elegant. Avantajul este şi mai vizibil pentru clase mai complexe (cum ar fi tablouri, liste, arbori, etc.) şi pentru streamuri de tip `fstream`.
- deşi nu apare vizibil în acest exemplu, inserţia tipurilor predefinite şi a celor definite de utilizator poate fi făcută "amestecat", în orice succesiune.

Streamuri şi fişiere

Pentru lucrul cu streamuri reprezentate pe fişiere există, în cadrul ierarhiilor din bibliotecă, clase speciale. Pentru utilizarea acestora este necesară includerea antetului `<fstream.h>`**.** Acesta, la rândul său, include implicit `<iostream.h>` care nu mai trebuie inclus explicit (dar clasele descrise în el sunt vizibile).

Prezentăm în continuare un exemplu minimal de cod care instanţiază un stream de tip `ofstream` (mapat pe fişier) şi inserează un obiect de tip definit de programator (de exemplu `PUNCT`) în acest stream. Pentru alte detalii privind aceste stream-uri (alte metode, de exemplu pentru tratarea erorilor) se poate consulta help-ul.

```
// insereaza un punct in stream (fisier)
ofstream ofs("fisier.txt"); // instantiere un ofstream mapat pe fisier.txt
         ofs<<un_punct;     // inserare a PUNCT-ului un_punct pe stream
         ofs.close();       // inchidere stream
```

Indicaţii 7.2

- ⇨ Trebuie redefiniţi operatorii de inserţie pentru clasele `FIGURA` şi `MULTIME`
- ⇨ Clasa `MULTIME` îşi va salva complet starea (numărul de figuri, figura curentă şi fiecare figură din mulţime). Pentru a înţelege mai uşor conţinutul fişierului recomandăm scrierea pe prima linie a caracteristicilor `MULTIMII` iar apoi, pe liniile următoare, caracteristicile fiecărei figuri. Pentru trecerea la linia următoare se poate folosi manipulatorul `endl`.
- ⇨ Pentru a verifica funcţionarea se poate urmării de fiecare dată conţinutul fişierului rezultat
- ⇨ Atenţie: pentru a putea ulterior separa câmpurile salvate e important să se insereze câte un separator de tip spaţiu între valorile salvate.

Tema 7.3

Să se implementeze operatorii de extracţie `">>"` care să permită la apăsarea tastei 6 (având codul '6') restaurarea dintr-un stream a obiectului de tip `MULTIME` (şi a celor de tip `FIGURA` incluse)**.** Se va folosi ca stream un obiect de tip ifstream mapat pe fişierul „FISIER.CFG".

Consideraţii teoretice 7.3

Extracţia formatată a tipurilor standard

După cum se cunoaşte, pentru operaţiile de intrare cu format se folosesc **operatorii de extracţie** `>>` definiţi pentru fiecare din **tipurile de date standard** existente în limbaj. Declaraţiile acestora se găsesc în `<iostream.h>` şi le prezentăm în continuare:

```
class istream : virtual public ios
{
//  . . . . . .
public:
       // extrage un caracter
    istream& operator>> (  signed char &);
    istream& operator>> (unsigned char &);

       // extrage valori numerice din reprezentarea lor ascii
    istream& operator>> (short &);
    istream& operator>> (unsigned short &);
    istream& operator>> (int &);
    istream& operator>> (unsigned int &);
    istream& operator>> (long &);
    istream& operator>> (unsigned long &);
    istream& operator>> (float &);
    istream& operator>> (double &);
    istream& operator>> (long double &);

       // extrage un şir
    istream& operator>> (  signed char *);
    istream& operator>> (unsigned char *);
//  . . . . . .
};
```

Facem următoarea remarcă referitoare la operaţiile de intrare cu format:

- la extragerea unui singur caracter se extrage primul caracter din istream diferit de ***caracter de tip spaţiu*** - deci se sar caracterele de tip spaţiu. Faptul este explicabil deoarece acum avem de a face cu o extragere formatată. Amintim că, conform funcţiei isspace, caracterele de tip staţiu sunt: *horizontal tab* (‘\t’,9), *newline* (‘\n’,10), *vertical tab* (‘\v’,11), *formfeed* (‘\f’,12), *carriage return* (‘\r’,13), space (‘ ‘,32).

Cele de mai sus justifică posibilitatea extracţiei tipurilor standard de pe stream ca în exemple binecunoscute:

```
       cin>>x>>c;   //evident x si c sunt variabile definite anterior
```

Extracţie de pe stream pentru tipuri definite de utilizator

Reluăm exemplul clasei `PUNCT` exemplificând de această dată extracţia sa:

```
#include <iostream.h>

class PUNCT
   {
   int x,y;
   friend istream& operator>>(istream& i, PUNCT& ob)
      {
      i>>ob.x; i>>ob.y; return i; // extractie membrii PUNCT
      }
   };
```

situaţie în care este posibil următorul cod:

```
PUNCT un_punct;
         ...
         cin>>un_punct;   //extractia tipului propriu pe stream
         ...
```

Asemănător cu exemplul prezentat în consideraţiile teoretice 7.2 prezentăm în continuare un exemplu minimal de cod care instanţiază un stream de tip `ifstream` (mapat pe fişier) şi extrage un obiect de tip definit de programator (de exemplu `PUNCT`) din acest stream.

```
     // extrage un punct din stream (fisier)
   ifstream ifs("fisier.txt"); // instantiere un ifstream mapat pe fisier.txt
            ifs>>un_punct;     // extragere a PUNCT-ului un_punct de pe stream
            ifs.close();       // inchidere stream
}
```

Atragem atenţia că, în mod normal (deşi nu obligatoriu), inserţia şi extracţia sunt gândite în pereche, pentru a putea ulterior extrage de pe stream obiectul inserat.

Indicaţii 7.3

- Se va goli mulţimea înainte de restaurare.
- La restaurare nu cunoaşte tipul clasei obiectului care a fost salvat. Toate obiectele se vor considera ca fiind de tip `CERC`.
- Atenţie ca prin inserţie şi extracţie a mulţimii să nu schimbăm ordinea figurilor din mulţime.

Tema 7.4

Să se modifice inserţia-extracţia astfel încât să se corecteze problema anterioară.

Indicaţii 7.4

- Se va implementa la clasa `FIGURA` o metodă virtuală care să întoarcă (sub forma unui caracter) tipul obiectului respectiv. La extracţia `MULTIMII` se vor instanţia corespunzător `CERC-uri` şi `PATRAT-e`.

Tema 7.5

Să se modifice programul astfel încât fişierul folosit să fie un fişier binar.

Consideraţii teoretice 7.5

Pe lângă inserţia şi extracţia formatată bazată pe operatorii << şi >> există şi un mod de lucru fără format. Pentru aceste operaţii clasele istream respectiv ostream pun la dispoziţie un set de metode publice de citire respectiv scriere descrise în continuare (conform `<iostream.h>`):

```
class istream : virtual public ios
{
//  . . . . . .
public:
       // extrage caractere într-un vector
    istream& read(  signed char *, int);    // extrage vector
    istream& read(unsigned char *, int);    //  - '' -
     // extragere un singur caracter
    istream& get(unsigned char &);          // extrage caracter
    istream& get(  signed char &);          //  - '' -
    int      get();                         //  - '' -
//  . . . . . .
};

class ostream : virtual public ios
{
```

```
//  . . . . . .
public:
   ostream& put(char);                          // inserează caracter
   ostream& write(const signed char *, int);    // inserează vector
   ostream& write(const unsigned char *, int); //  - '' -
//  . . . . . .
};
```

După cum se constată transferul datelor se face fără a ţine cont de semnificaţia acestora, metodele disponibile fiind echivalente cu cele existente în C (`getc, putc, read, write`), numite acolo de nivel coborât. Aceste operaţii sunt specifice situaţiilor în care datele sunt binare (şi, de cele mai multe ori, fluxul implicat este un fişier).

În legătură cu metodele prezentate anterior mai facem următoarele remarci:

- metodele `read` şi `write` au ca şi parametrii adresa vectorului care reţine datele ce se transfera şi numărul de octeţi care se transferă.
- metodele get de extragere a unui singur caracter extrag primul caracter din istream indiferent care este acesta, chiar dacă este ***caracter de tip spaţiu.***
- ca şi în C, operaţiile de intrare-ieşire fără format se folosesc mai ales când avem de a face cu fişiere binare.

Indicaţii 7.5

- ⇨ Verificaţi după salvare lungimea fişierului.
- ⇨ Pentru a simplifica extracţia recomandăm ca scrierea şirurilor (numelui figurilor) să se facă într-un format „de tip PASCAL" adică să precedăm şirul de lungimea sa.

Tema 7.6

Să se execute aplicaţia pas cu pas pentru inserţia şi extracţia mulţimii m.

Consideraţii teoretice 7.6

Se vor revedea consideraţiile teoretice de la 3.8

Anexa 7

ex7.h

```
#define TAB    9
#define ESC   27

#define LEFT  75
#define RIGHT 77
#define UP    72
#define DOWN  80

#define UNU      '1'
#define DOI      '2'
#define TREI     '3'
#define PATRU    '4'
#define CINCI    '5'
#define SASE     '6'
#define SAPTE    '7'
#define OPT      '8'
// prototipuri de functii

void OurInitGraph(void);
```

ex7.cpp

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#include "ex7.h"
#include "clase.h"

MULTIME m;

void main()
//*********
{
int gata=0;

   OurInitGraph();

   m += new CERC;
   m = m + new PATRAT(500,300,75,RED,"red");
   m += new CERC(100,100,25,BLUE,"blue");
   m = m + new PATRAT;
   m = m + new PATRAT(400,200,100,YELLOW,"yellow");
   m += new CERC;

   m.Afiseaza();

   while(!gata)
     switch(getch())
       {
       case ESC:
         gata=1;
         break;
```

```
       case TAB:
         ++m;
         break;
         case UNU:    m.Get()->Creste( +10     ); m.Afiseaza();break;
         case DOI:    m.Get()->Creste( -10     ); m.Afiseaza();break;
         case TREI:   if(m.NrElem()<MAX_FIGURI)
                       if(m.NrElem()%2)
                           m += new CERC;
                       else
                           m += new PATRAT;
                    m.Afiseaza();
                              break;
           case PATRU: m.Elimina(); m.Afiseaza(); break;
       case 0:
         switch(getch())
           {
           case LEFT:  m.Get()->Muta(   -10,  0 ); break;
           case RIGHT: m.Get()->Muta(    10,  0 ); break;
           case UP:    m.Get()->Muta(     0,-10 ); break;
           case DOWN:  m.Get()->Muta(     0, 10 ); break;
           }
         m.Afiseaza();
       }
   closegraph();
}

void OurInitGraph()
//*****************
{
int gdriver = DETECT, gmode, errorcode;

   initgraph(&gdriver,&gmode,"");

   errorcode = graphresult();
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }
   settextjustify(CENTER_TEXT,CENTER_TEXT);
}
```

clase.h

```
class LOCATION
      {
      public:
             LOCATION();
             LOCATION(int x0,int y0);
        void Muta(int dx,int dy);
      protected:
        int  x;
        int  y;
      };

class FIGURA : public LOCATION
      {
      protected:
         int  r;
```

```
         int  c;
         char *Nume;
      public:
         char Tip;
            FIGURA();
            FIGURA(int x0,int y0,int r0,int c0,char *n0);
           ~FIGURA();
              void Muta(int dx,int dy);
              void Creste(int dr);
      virtual void Afiseaza()=0;
      virtual void Sterge()=0;
      };

class CERC : public FIGURA
      {
      public:
              CERC();
              CERC(int x0,int y0,int r0,int c0,char *n0);
         void Afiseaza();
         void Sterge();
      };

class PATRAT : public FIGURA
      {
      public:
              PATRAT();
              PATRAT(int x0,int y0,int r0,int c0,char *n0);
         void Afiseaza();
         void Sterge();
      };

#define MAX_FIGURI 20

class MULTIME
      {
        int      NrFiguri;
        int      FiguraCurenta;
        FIGURA  *pe[MAX_FIGURI];
      public:
                 MULTIME();
        void     Goleste();
        void     Afiseaza();
        void     Elimina();
        FIGURA*  Get();
        int      NrElem();
        void     operator ++();
        void     operator += ( FIGURA* f );
        MULTIME& operator +  ( FIGURA* f );
      };
```

clase.cpp

```
#include <graphics.h>
#include <string.h>
#include <alloc.h>

#include "clase.h"

//~~~~~~~~~~~~~~~~~~
//  clasa LOCATION
//~~~~~~~~~~~~~~~~~~
```

```
LOCATION::LOCATION()
//******************
{
      x =   320;
      y =   240;
}

LOCATION::LOCATION(int x0,int y0)
//*******************************
{
      x = x0;
      y = y0;
}

void LOCATION::Muta(int dx,int dy)
//********************************
{
      x+=dx;
      y+=dy;
}

//~~~~~~~~~~~~~~~~
//  clasa FIGURA
//~~~~~~~~~~~~~~~~

FIGURA::FIGURA():LOCATION()
//*************************
{
      r =    50;
      c = WHITE;
      Nume = NULL;
}

FIGURA::FIGURA(int x0,int y0,int r0,int c0, char *n0):LOCATION(x0,y0)
//*********************************************************************
{
      r = r0;
      c = c0;
      Nume = (char*)malloc(strlen(n0)+1); // aloca memorie suplimentara
      strcpy(Nume,n0);
}

FIGURA::~FIGURA()
//***************
{
      free(Nume);   // elibereaza memoria suplimentara alocata
}

void FIGURA::Muta(int dx,int dy)
//******************************
{
      Sterge();               //stergere
      LOCATION::Muta(dx,dy);  //mutare
      Afiseaza();             //afisare in noua pozitie
}

void FIGURA::Creste(int dr)
//*************************
{
      Sterge();               //stergere
      r+=dr;                  //redimensionare
      Afiseaza();             //afisare cu noua dimensiune
```

```
}

//~~~~~~~~~~~~~~
//  clasa CERC
//~~~~~~~~~~~~~~

CERC::CERC():FIGURA()
//*******************
{
}

CERC::CERC(int x0,int y0,int r0,int c0, char *n0):FIGURA(x0,y0,r0,c0,n0)
//*************************************************************************
{
}

void CERC::Sterge()
//*****************
{
      setcolor(BLACK);
      circle(x,y,r);
      if(Nume!=NULL)
         outtextxy(x,y,Nume);
}

void CERC::Afiseaza()
//*******************
{
      setcolor(c);
      circle(x,y,r);
      if(Nume!=NULL)
         outtextxy(x,y,Nume);
}

//~~~~~~~~~~~~~~~~
//  clasa PATRAT
//~~~~~~~~~~~~~~~~

PATRAT::PATRAT():FIGURA()
//***********************
{
}

PATRAT::PATRAT(int x0,int y0,int r0,int c0, char *n0):FIGURA(x0,y0,r0,c0,n0)
//*************************************************************************
{
}

void PATRAT::Sterge()
//*******************
{
      setcolor(BLACK);
      rectangle(x-r,y-r,x+r,y+r);
      if(Nume!=NULL)
         outtextxy(x,y,Nume);
}

void PATRAT::Afiseaza()
//*********************
{
      setcolor(c);
      rectangle(x-r,y-r,x+r,y+r);
      if(Nume!=NULL)
```

```
        outtextxy(x,y,Nume);
}

//~~~~~~~~~~~~~~~~~
//  clasa MULTIME
//~~~~~~~~~~~~~~~~~

MULTIME::MULTIME()
//****************
{
      NrFiguri=0;
      FiguraCurenta=0;
}

void MULTIME::Goleste()
//**********************
{
int k;
      for(k=0;k<NrFiguri;k++)
         {
         pe[k]->Sterge();  // stergere de pe ecran
         delete pe[k];     // stergere din memorie
         }
      NrFiguri=0;
      FiguraCurenta=0;
}

void MULTIME::Afiseaza()
//***********************
{
int k;
      for(k=0;k<NrFiguri;k++)
         pe[k]->Afiseaza();
}

FIGURA* MULTIME::Get()
//*********************
{
      return(pe[FiguraCurenta]);
}

int MULTIME::NrElem()
//********************
{
      return(NrFiguri);
}

void MULTIME::operator++()
//*************************
{
      FiguraCurenta++;
      FiguraCurenta%=NrFiguri;
}

void MULTIME::operator += ( FIGURA* f)
//*************************************
{
      if(NrFiguri==MAX_FIGURI)
         return;                                    // nu mai putem insera
      memmove(&pe[FiguraCurenta]+1,&pe[FiguraCurenta],
         (NrFiguri-FiguraCurenta)*sizeof(pe[0]) );  // facem loc
      pe[FiguraCurenta]=f;                          // inserare figura noua
      NrFiguri++;
```

```
}

MULTIME& MULTIME::operator + ( FIGURA* f)
//***************************************
{
      if(NrFiguri==MAX_FIGURI)
         return(*this);                                  // nu mai putem insera
      memmove(&pe[FiguraCurenta]+1,&pe[FiguraCurenta],
         (NrFiguri-FiguraCurenta)*sizeof(pe[0]) );       // facem loc
      pe[FiguraCurenta]=f;                               // inserare figura noua
      NrFiguri++;
      return(*this);                                     // returnam multimea
}

void MULTIME::Elimina()
//*********************
{
      if(NrFiguri==1)
         return;                                         // lista sa nu fie vida
      pe[FiguraCurenta]->Sterge();                       // stergere de pe ecran
      delete pe[FiguraCurenta];                          // distrugere element
      memmove(&pe[FiguraCurenta],&pe[FiguraCurenta]+1,
         (NrFiguri-FiguraCurenta)*sizeof(pe[0]) );       // mutare in multime
      NrFiguri--;
      if(FiguraCurenta==NrFiguri)
         FiguraCurenta--;
}
```