Com[a Ciprian
Grupa 5404

Programare Orientat\ Obiect

Tema 1

Alocarea dinamicã a memoriei: matrici şi vectori

Paradigma de programare prin abstractizarea datelor

vectc2.h

```
#include <malloc.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#ifndef _MATRICIC1_INCLUDED_
#define _MATRICIC1_INCLUDED_
enum Boolean { false = 0,  true = 1};
enum Error { error = 0, success = 1 };
typedef struct VectorInt2   {  int *Vector;   int NrElem;};
                                        /*Testeaz\ dac\ valoarea pointerului este NULL.
                                            Dac\ da returneaz\ True, altfel returneaz\ False.*/
int NULLMemTestVec(void *);
                                            /*Testeaz\ dac\ valoarea pointerului este NULL.
                                Dac\ da returneaz\ True [i afi[eaz\ un mesaj, altfel returneaz\ False.*/
int AllocErrorVec(void *);
                                /*Aloc\ dinamic memorie pentru un vector cu NrElem elemente.
                                                                    Returneaz\ error sau success.*/
int AllocVectorInt2(struct VectorInt2 *);
                                        /*Elibereaz\ memoria alocat\ dinamic pentru un vector.
                                         Memoria poate fi alocat\ folosind malloc sau calloc !!!
                                                                    Returneaz\ error sau success.*/
int FreeVectorInt2(struct VectorInt2 *);
                                             /*Tip\re[te la consol\ vectorul de `ntregi specificat.*/
void PrintVectorInt2(struct VectorInt2 *);
                /**********************Matrice************************/
typedef struct MatrixInt2
{  int **Matrix;
   int NrLin;
   int NrCol;
};                                          /*Testeaz\ dac\ valoarea pointerului este NULL.
                                                Dac\ da returneaz\ True, altfel returneaz\ False.*/
int NULLMemTest(void *);
                                            /*Testeaz\ dac\ valoarea pointerului este NULL.
                                Dac\ da returneaz\ True [i afi[eaza un mesaj, altfel returneaz\ False.*/
int AllocError(void *);
                        /*Aloc\ dinamic memorie pentu o matrice cu NrLin linii [i NrCol coloane.
                                                                    Returneaz\ error sau success.*/
int AllocMatrixInt2(struct MatrixInt2 *);
                                        /*Elibereaz\ memoria alocat\ dinamic pentru o matrice.
                                         Memoria poate fi alocat\ folosind malloc sau calloc !!!
                                                                    Returneaz\ error sau success.*/
int FreeMatrixInt2(struct MatrixInt2 *);
                                             /*Tip\re[te la consol\ matricea de `ntregi specificat\.*/
void PrintMatrixInt2(struct MatrixInt2 *);
#endif
```

matrvec2.c

```
#include "vectc2.h"
void main(void)
{   struct VectorInt2 Vector;
    struct VectorInt2 VectorResult;
    int contelem,cont,ContVec,sumcol;
    struct MatrixInt2 Matrix;
    int contlin, contcol;
    printf("\n\n\n");
    puts("Introduceti numarul de linii: ");
    scanf("%d",&Matrix.NrLin);
    puts("Introduceti numarul de coloane: ");
    scanf("%d",&Matrix.NrCol);
    if(!AllocMatrixInt2(&Matrix))
       return;
    PrintMatrixInt2(&Matrix);                    /*Matricea nu a fost initializat\ ...*/
    for(contlin = 0; contlin < Matrix.NrLin; contlin++)
       for(contcol = 0; contcol < Matrix.NrCol; contcol++)
            Matrix.Matrix[contlin][contcol] = contlin+ contcol;
    PrintMatrixInt2(&Matrix);
         /*****************************************************************/
    puts("Introduceti numarul de elemente: ");
    scanf("%d",&Vector.NrElem);
    VectorResult.NrElem=Matrix.NrLin;
    if(!AllocVectorInt2(&Vector))           return;
    if(!AllocVectorInt2(&VectorResult))              return;
    PrintVectorInt2(&Vector);               /*Vectorul nu a fost initializat ...*/
    for(contelem = 0; contelem < Vector.NrElem; contelem++)
                    Vector.Vector[contelem] = contelem+2;
    PrintVectorInt2(&Vector);
         /******************* Produsul Matrice * Vector **********************/
    if (Matrix.NrCol!=Vector.NrElem)
          { printf("Eroare produs: NrColoaneMatrice<>NrElementeVector!\n"); }
    else
    { for ( cont = 0; cont < Matrix.NrLin; cont++)
       { sumcol=0;
            for ( ContVec = 0; ContVec < Matrix.NrCol; ContVec++)
              sumcol+=(Matrix.Matrix[cont][ContVec]*Vector.Vector[ContVec]);
            VectorResult.Vector[cont]=sumcol;
         }
       puts("Produsul Matrice*Vector:\n");
       PrintVectorInt2(&VectorResult);
     }     /*****************************************************************/
    FreeVectorInt2(&Vector);
    FreeVectorInt2(&VectorResult);
         /*****************************************************************/
    FreeMatrixInt2(&Matrix);                                    /*Apel aproape incorect:
                              Era r\u tare dac\ func]ia FreeMatrixInt2 nu [tia s\ rezolve situa]ia: */
    FreeMatrixInt2(&Matrix);
    /*free(Matrix.Matrix);*/                    /*Apel total eroant.*/
    getch();
}
```

memalloc.c

```
#include "vectc2.h"
                        /*Aloc\ dinamic memorie pentu o matrice cu NrLin linii [i NrCol coloane.
                                                                    Returneaz\ error sau success.*/
int AllocMatrixInt2(struct MatrixInt2 *PtrMatrix)
{
   int contorlin;
   int **MatrixTemp;
                                                        /*Se aloc\ memorie pentru pointeri la linii ...*/
   PtrMatrix->Matrix = (int **)malloc(PtrMatrix->NrLin * sizeof(int *));
   if(AllocError(PtrMatrix->Matrix))
      return error;

   MatrixTemp = PtrMatrix->Matrix;
   for(contorlin = 0; contorlin < PtrMatrix->NrLin; (contorlin++, MatrixTemp++))
   {                                                    /*Se aloc\ memorie pentru o linie ...*/
      *MatrixTemp = (int *)malloc(PtrMatrix->NrCol * sizeof(int));
      if(AllocError(*MatrixTemp))
      {                              /*~n caz de eroare la alocare se [terge memoria deja alocat\ ...*/
         PtrMatrix->NrLin = contorlin;
         FreeMatrixInt2(PtrMatrix);
         free(PtrMatrix->Matrix);
         PtrMatrix->Matrix = NULL;
         PtrMatrix->NrLin = 0;
         PtrMatrix->NrCol = 0;
         return error;
      }
   }
   return success;
}
       /*Elibereaz\ memoria alocat\ dinamic pentru o matrice. Memoria poate fi alocat\ folosind
                                                  malloc sau calloc !!! Returneaz\ error sau success.*/
int FreeMatrixInt2(struct MatrixInt2 *PtrMatrix)
{
   int contlin;
   int **MatrixTemp = PtrMatrix->Matrix;
                                       /*Se testeaz\ dac\ memoria pentru matrice a fost alocat\ ...*/
   if(NULLMemTest(PtrMatrix->Matrix))
      return error;
                                                                    /*Se [terge memoria pe linii ...*/
   for(contlin = 0; contlin < PtrMatrix->NrLin; (contlin++, MatrixTemp++))
   {                                    /*Se testeaz\ dac\ a fost alocat\ memorie pentru linia curent\ ...*/
      if(NULLMemTest(MatrixTemp))
            return error;
      free(MatrixTemp);              /*Se [terge memoria pentru linia curenta ...*/
   }
   free(PtrMatrix->Matrix);
   PtrMatrix->Matrix = NULL;
   PtrMatrix->NrLin = 0;
   PtrMatrix->NrCol = 0;
   return success;
}
```

memerror.c

```
#include "vectc2.h"                    /*Testeaz\ dac\ valoarea pointerului este NULL.
                                         Dac\ da returneaz\ True, altfel returneaz\ False.*/
int NULLMemTest(void *Ptr)
{  if(!Ptr)      return true;
   return false;
}                                      /*Testeaz\ dac\ valoarea pointerului este NULL.
                                 Dac\ da returneaz\ True [i afi[eaz\ un mesaj, altfel returneaz\ False.*/
int AllocError(void * Ptr)
{  if(NULLMemTest(Ptr))
   {  puts("Eroare la alocarea memoriei ...\n");
      return true;
   }
   return false;
 }
```

vecaloc2.c

```
#include "vectc2.h"
                                /*Aloc\ dinamic memorie pentu un vector cu NrElem elemente.
                                                          Returneaz\ error sau success.*/
int AllocVectorInt2(struct VectorInt2 *PtrVector)
{
   int *VectorTemp;
   VectorTemp = PtrVector->Vector;
                                                          /*Se aloc\ memorie pentru vector ...*/
   VectorTemp = (int *)malloc(PtrVector->NrElem * sizeof(int));
   if(AllocErrorVec(VectorTemp))
            {                     /*~n caz de eroare la alocare se [terge memoria deja alocat\ ...*/
             FreeVectorInt2(PtrVector);
             free(PtrVector->Vector);
             PtrVector->Vector = NULL;
             PtrVector->NrElem = 0;
             return error;
            }
   return success;
}
      /*Elibereaz\ memoria alocat\ dinamic pentru un vector. Memoria poate fi alocat\ folosind
                                          malloc sau calloc !!! Returneaz\ error sau success.*/
int FreeVectorInt2(struct VectorInt2 *PtrVector)
{
   int *VectorTemp = PtrVector->Vector;
                                     /*Se testeaz\ dac\ memoria pentru vector a fost alocat\ ...*/
   if(NULLMemTestVec(PtrVector->Vector))
         return error;
                                                          /*Se [terge memoria  ...*/
   free(VectorTemp);             /*Se [terge memoria pentru vector ...*/
   free(PtrVector->Vector);
   PtrVector->Vector = NULL;
   PtrVector->NrElem = 0;
   return success;
}
```

vecmerr2.c

```
#include "vectc2.h"

                                                    /*Testeaz\ dac\ valoarea pointerului este NULL.
                                                   Dac\ da returneaz\ True, altfel returneaz\ False.*/
int NULLMemTestVec(void *Ptr)
{
   if(!Ptr)
      return true;
   return false;
}
                                                    /*Testeaz\ dac\ valoarea pointerului este NULL.
                                  Dac\ da, returneaz\ True [i afi[eaz\ un mesaj, altfel returneaz\ False.*/
int AllocErrorVec(void * Ptr)
{
   if(NULLMemTestVec(Ptr))
   {
      puts("Eroare la alocarea memoriei ...\n");
      return true;
   }
   return false;
}
```

printmat.c

```
#include "vectc2.h"

                                                /*Tip\re[te la consol\ matricea de `ntregi specificat\.*/
void PrintMatrixInt2(struct MatrixInt2 *PtrMatrix)
{
   char Test;
   int contlin, contcol;

   puts("Sa afisez matricea ? (y/n)");
   do
   {
      Test = toupper(getch());
   }
   while(Test != 'Y' && Test != 'N');

   if(Test == 'Y')
   {
      puts("Matricea este :");
      for(contlin = 0; contlin < PtrMatrix->NrLin; contlin++)
      {
         for(contcol = 0; contcol < PtrMatrix->NrCol; contcol++)
               printf("%3d",PtrMatrix->Matrix[contlin][contcol]);
         printf("\n");
      }
   }
}
```

prntvec2.c

```
#include "vectc2.h"

                                        /*Tip\re[te la consol\ vectorul de `ntregi specificat.*/
void PrintVectorInt2(struct VectorInt2 *PtrVector)
{
   char Test;
   int cont;

   puts("Sa afisez vectorul ? (y/n)");
   do
   {
      Test = toupper(getch());
   }
   while(Test != 'Y' && Test != 'N');

   if(Test == 'Y')
   {
         puts("Vectorul este :");
         for(cont = 0; cont < PtrVector->NrElem; cont++)
                  printf("%4d",PtrVector->Vector[cont]);
         printf("\n");
   }
}
```

Observaþii:

1.Aceste fiºiere au fost realizate prin modificarea sursei “Matrici2”, pusã la dispoziþie la laboratorul de “Programare Orientatã Obiect”, incluzând ºi comentariile.

2.Proiectul realizat cu aceste fiºiere a fost testat ºi funcþionează conform cerinþelor temei.

Com[a Ciprian
Grupa 5404

Programare Orientat\ Obiect

Tema 2

Lucrul cu membrii "private", "protected" şi "public"

Programul 1

clase1.hpp

```
#include <iostream.h>
#include <conio.h>
#include <malloc.h>
                                                        //Compilare condi]ionat\ ...
//#define _APEL_METODA_
                                                        //Declararea tipului de date Clasa1
class Clasa1                                    //Declararea membrilor clasei Clasa1:
{ private:
      int           IntPrivate;
      double        DoublePrivate;
                  const char    *Nume;                    //{ir constant de caractere
                        //reprezint\ numele obiectului ...  Sau pointer la char constant ...
                              //Declararea func]iilor membru private ale clasei Clasa1
       void Mesaj(char *);
  protected: int          IntProtected;
             double       DoubleProtected;
                              //Declararea func]iilor membru protected ale clasei Clasa1
             void PrintNumeCon(void);                   //Afi[eaz\ numele la consol\ ...
  public:       int         IntPublic;
                double      DoublePublic;
                    //Declararea func]iilor membru publice ale clasei Clasa1 : metodele clasei
                Clasa1(void);                           //Constructor implicit ...
                Clasa1(Clasa1 &);                       //Constructor de copiere ...
                Clasa1(int, double, const char *, int, double, int, double);    //Constructor ...
                Clasa1(const char *);                                   //Constructor ... .
                    //Modificatorul const asigur\ faptul c\ valoarea de la adresa transferat\
                              //nu va fi modificat\.          Testul se face la compilare.
                ~Clasa1();                              //Destructor ...
                              //Atribuie membrilor valorile precizate ca parametri ...
     void Set(int, double, const char *, int, double, int, double);
     void SetNume(const char *);
     void SetPrivate(int, double);
     void SetProtected(int, double);
     void SetPublic(int, double);
                                                        //Func]ii membru "inline"
                          //Returneaz\ referin]a -> un pseudonim pentru obiectul returnat
                                                        //-> este lvalue [i rvalue
     int & GetPrivateInt(void) {return IntPrivate;}
     double & GetPrivateDouble(void)    {return DoublePrivate;}
     int & GetProtectedInt(void)    {return IntProtected;}
     double & GetProtectedDouble(void)    {return DoubleProtected;}
     void PrintCon(void);           //Afi[eaz\ valorile variabilelor membru la consol\ ...
     void ScanCon(void);            //Cite[te valorile variabilelor membru de la consol\ ...
};
```

conscls1.cpp

```
#include "clase1.hpp"
                                                    //Defini]ia func]iilor membru ale clasei Clasa1.
          //Se folose[te operatorul de rezolu]ie pentru a specifica clasa c\reia `i apar]ine func]ia.
Clasa1::Clasa1(void)
{  Nume = NULL;
   IntPrivate = IntProtected = IntPublic = 0;
   DoublePrivate = DoubleProtected = DoublePublic = 0.0;
   Mesaj("S-a apelat constructorul implicit al clasei \"Clasa1\"" );
}

Clasa1::Clasa1(Clasa1 &RefObiectClasa1)
{  Nume = RefObiectClasa1.Nume;
   IntPrivate = RefObiectClasa1.IntPrivate;
   DoublePrivate = RefObiectClasa1.DoublePrivate;
   IntProtected = RefObiectClasa1.IntProtected;
   DoubleProtected = RefObiectClasa1.DoubleProtected;
   IntPublic = RefObiectClasa1.IntPublic;
   DoublePublic = RefObiectClasa1.DoublePublic;
   Mesaj("S-a apelat constructorul de copiere al clasei \"Clasa1\"");
}

Clasa1::Clasa1(int IntPrivate, double DoublePrivate, const char *NumeObiect,
                  int IntProtected, double DoublePotected,
                  int IntPublic, double DoublePublic)
{
 #ifdef _APEL_METODA_
   Nume = NULL;
         Set(IntPrivate, DoublePrivate, NumeObiect, IntProtected, DoublePotected,
              IntPublic,   DoublePublic);
 #else
         Clasa1::IntPrivate = IntPrivate;                    //Se utilizeaz\ operatorul de rezolu]ie
                                                                        //`n cadrul clasei...
         Clasa1::DoublePrivate = DoublePrivate;
         Nume = NumeObiect;
         Clasa1::IntProtected = IntProtected;
         Clasa1::DoubleProtected = DoublePotected;
         Clasa1::IntPublic = IntPublic;
         Clasa1::DoublePublic = DoublePublic;
 #endif
 Mesaj("S-a apelat constructorul cu parametri al clasei \"Clasa1\"");
}

Clasa1::Clasa1(const char *NumeObiect)
{        Nume = NumeObiect;
         IntPrivate = IntProtected = IntPublic = 0;
         DoublePrivate = DoubleProtected = DoublePublic = 0.0;
         Mesaj("S-a apelat constructorul cu parametri al clasei \"Clasa1\"");
}

Clasa1::~Clasa1()
{ Mesaj("S-a apelat destructorul clasei \"Clasa1\""); }

void Clasa1::Mesaj(char *Mesaj)
```

```
{
   PrintNumeCon();
   cout << " : " << Mesaj << endl;
   getch();
}
```

setcls1.cpp

```
#include "clase1.hpp"

void Clasa1::Set(int IntPrivateParam, double DoublePrivateParam,
                                                    const char * NumeParam,
                 int IntProtectedParam, double DoubleProtectedParam,
                 int IntPublicParam, double DoublePublicParam)
{
#ifdef _APEL_METODA_
   SetNume(NumeParam);
   SetPrivate(IntPrivateParam, DoublePrivateParam);
   SetProtected(IntProtectedParam, DoubleProtectedParam);
   SetPublic(IntPublicParam, DoublePublicParam);
#else
   IntPrivate = IntPrivateParam;
   DoublePrivate = DoublePrivateParam;
   if(!Nume)                                        //Dac\ obiectul nu are nume ...
                Nume = NumeParam;
   IntProtected = IntProtectedParam;
   DoubleProtected = DoubleProtectedParam;
   IntPublic = IntPublicParam;
   DoublePublic = DoublePublicParam;
#endif
}

void Clasa1::SetNume(const char *NumeParam)
{   if(!Nume)
                Nume = NumeParam;
}

void Clasa1::SetPrivate(int Int, double Double)
{  IntPrivate = Int;
   DoublePrivate = Double;
}

void Clasa1::SetProtected(int Int, double Double)
{  IntProtected = Int;
   DoubleProtected = Double;
}

void Clasa1::SetPublic(int Int, double Double)
{  IntPublic = Int;
   DoublePublic = Double;
}
```

getcls1.cpp

```
#include "clase1.hpp"

void Clasa1::ScanCon(void)
{  cout << endl << "Pentru ";
   PrintNumeCon();
   cout << endl << "Introduceti:" << endl;
   cout << "Valoarea \"int\" private:" << endl;
   cin >> IntPrivate;                                  //Acces indirect la membru private ...
   cout << "Valoarea \"double\" private:" << endl;
   cin >> DoublePrivate;                               //Acces indirect la membru private ...
   cout << "Valoarea \"int\" protected:" << endl;
   cin >> IntProtected;                                //Acces indirect la membru protected ...
   cout << "Valoarea \"double\" protected:" << endl;
   cin >> DoubleProtected;                             //Acces indirect la membru protected ...
   cout << "Valoarea \"int\" public:" << endl;
   cin >> IntPublic;                                   //Acces direct la membru public ...
   cout << "Valoarea \"double\" public:" << endl;
   cin >> DoublePublic;                                //Acces direct la membru public ...
   cout << endl;
}
```

prntcls.cpp

```
#include "clase1.hpp"

void Clasa1::PrintNumeCon(void)
{  cout << "obiectul ";
   if(Nume) //Daca Nume != NULL
      cout << Nume;
   else //Daca Nume == NULL
      cout << "Anonim ";
}

void Clasa1::PrintCon(void)
{  cout << endl;
   PrintNumeCon();
   cout << " : \n" << "Valoarea \"int\" private:\t\t\t" << IntPrivate << endl;
                                                       //Acces indirect la membru private ...
   cout << "Valoarea \"double\" private:\t\t" << DoublePrivate << endl;
                                                       //Acces indirect la membru private ...
   cout << "Valoarea \"int\" protected:\t\t" << IntProtected << endl;
                                                       //Acces indirect la membru protected ...
   cout << "Valoarea \"double\" protected:\t\t" << DoubleProtected << endl;
                                                       //Acces indirect la membru protected ...
   cout << "Valoarea \"int\" public:\t\t\t" << IntPublic << endl;
                                                       //Acces direct la membru public ...
   cout << "Valoarea \"double\" public:\t\t" << DoublePublic << endl;
                                                       //Acces direct la membru public ...
   cout << endl;
}
```

maincls1.cpp

```
#include "clase1.hpp"

                                                        //Obiect global de tipul Clasa1
Clasa1 Obiect1;                                 //Constructor implicit

void main(void)
{
   cout <<endl<<endl<< "     ****************************************\n";
   cout <<                   "     * TEMA2  OOP  AN IV 5404B Comsa Ciprian *"<<endl;
   cout <<                   "     ****************************************"<<endl;
   cout <<endl;
                                  //Citirea de la consol\ a valorilor pentru membrii unui obiect ... .
   cout << "Introduceti pentru Obiect1:\nValoarea \"int\" private" << endl;
   cin >> Obiect1.GetPrivateInt();                      //Acces indirect la membru private ...
   cout << "Valoarea \"double\" private" << endl;
   cin >> Obiect1.GetPrivateDouble();                   //Acces indirect la membru private ...
   cout << "Valoarea \"int\" protected" << endl;
   cin >> Obiect1.GetProtectedInt();                    //Acces indirect la membru protected ...
   cout << "Valoarea \"double\" protected" << endl;
   cin >> Obiect1.GetProtectedDouble();                 //Acces indirect la membru protected ...
   cout << "Valoarea \"int\" public" << endl;
   cin >> Obiect1.IntPublic;                            //Acces direct la membru public ...
   cout << "Valoarea \"double\" public" << endl;
   cin >> Obiect1.DoublePublic;                         //Acces direct la membru public ...
                        //Scrierea la consol\ a valorilor membrilor unui obiect ... .
   Obiect1.PrintCon();
                                                        //  cout << Obiect1.IntPrivate; Inaccesibil !!

   Clasa1 Obiect2(Obiect1);                     //Se apeleaz\ constructorul de copiere ...
   cout << endl << "Valorile unei copii:" << endl;
   Obiect2.PrintCon();
   int              Int1, Int2, Int3;
   double           Double1, Double2, Double3;
   cout << "Introduceti trei valori \"int\"" << endl;
   cin >> Int1 >> Int2 >> Int3;
   cout << "Introduceti trei valori \"double\"" << endl;
   cin >> Double1 >> Double2 >> Double3;
   Clasa1 Obiect3(Int1, Double1, "Obiect 3", Int2, Double2, Int3, Double3);
   Obiect3.PrintCon();
                                                        //   Instruc]iunea compus\ : bloc
   {
      Clasa1 Obiect4("\"Obiect local unui bloc\"");
      Obiect4.ScanCon();
      Obiect4.PrintCon();
   }
                                  //Alocare dinamic\ de memorie pentru un obiect din clasa Clasa1
        Clasa1 *PtrClasa1;
        PtrClasa1 = new Clasa1("\"Obiect alocat dinamic\"");
        if(PtrClasa1)
        {
                PtrClasa1->ScanCon();
                PtrClasa1->PrintCon();
        }
                                        //Eliberarea memoriei
```

```
        if(PtrClasa1)
        {       delete PtrClasa1;
                PtrClasa1 = NULL;                       //Obicei bun ...
        }
        int Dim;
        cout << "Introduceti dimensiunea vectorului: ";
        cin >> Dim;
                  //Alocare dinamic\ de memorie pentru un vector de obiecte din clasa Clasa1
        PtrClasa1 = new Clasa1[Dim];
                                                //Se apeleaz\ constructorul implicit de Dim ori ...
        if(PtrClasa1)
        {       Clasa1 *PtrClasa1Temp = PtrClasa1;
                for(int Contor = 0; Contor < Dim; Contor++)
                {       cout << endl << "Obiectul " << Contor << " alocat dinamic:"<< endl;
                        (PtrClasa1Temp++)->PrintCon();
                        cout << "Press any key when ready ...";
                        getch();
                 }
                 cout << endl;
         }
                                                                                //Eliberarea memoriei
   if(PtrClasa1)
   {            delete [Dim]PtrClasa1;
                //delete PtrClasa1;
                                        //Nu este apelat destructorul dec=t pentru primul obiect ...
                PtrClasa1 = NULL;                       //Obicei bun ...
        }
                                                        //Alocarea memoriei folosind malloc() ... .
                                                //Nu se apeleaz\ nici constructor nici destructor !!! .
        PtrClasa1 = (Clasa1 *)malloc(Dim * sizeof(class Clasa1));
        if(!PtrClasa1)
        {       cout << "Eroare la alocarea dinamica de memorie ..." << endl;
                return;
        }
        for(int Contor = 0; Contor < Dim; Contor++)
                PtrClasa1[Contor].ScanCon();
        for(Contor = 0; Contor < Dim; Contor++)
                PtrClasa1[Contor].PrintCon();
        if(PtrClasa1)
        {
                free(PtrClasa1);
        }

                                                        //Alocarea memoriei folosind calloc() ... .
                                                //Nu se apeleaz\ nici constructor nici destructor !!! .
        PtrClasa1 = (Clasa1 *)calloc(Dim, sizeof(class Clasa1));
        if(PtrClasa1)
        {
                free(PtrClasa1);
        }

        getch();
}
```

Programul 2

clase2.hpp

```
#include <iostream.h>
#include <conio.h>
                                                    //Starea `n care se poate afla un obiect ... .
enum State
{  Distrus = 0,
   Construit = 1,
   SetUpCompleted = 3
};                              //Alte st\ri pe care le poate avea obiectul ... .
enum Error
{  success = 0,   error = 1
 };
class Clasa2
{  private:
       int     StareObiect;                 //Starea obiectului ... .
       int    *PtrIntPrivate;               //Vector de `ntregi ... .
       int     Dim;                         //Dimensiunea [irului de `ntregi ... .
    public:
       Clasa2(void);                       //Constructor implicit ... .
       Clasa2(Clasa2 const &);        //Constructor explicit ... .
                    Clasa2(int);          //Constructor cu parametru dimensiunea
                                                  //vectorului de `ntregi ... .
                    ~Clasa2();           //Destructor ... .
                    void ClearAndDestroy(void);       //Func]ie pentru distrugerea explicit\
                                                            //a unui obiect ... .
                    int SetUp(int);         //Func]ie pentru reconstruirea explicit\ a unui obiect ... .
                    int Copy(Clasa2 const &);          //Copierea obiectului parametru ... .
                    void ScanCon(void);                //Citirea valorilor de la cosol\ ... .
                    void PrintCon(void);               //Afi[area valorilor la consol\ ... .
 };
```

conscls2.cpp

```
#include "clase2.hpp"

Clasa2::Clasa2(void)
{          StareObiect = Construit;
           PtrIntPrivate = NULL;
           Dim = 0;
}
Clasa2::Clasa2(Clasa2 const &C)
{          StareObiect = Distrus;
           PtrIntPrivate = NULL;
           Copy(C);
}
Clasa2::Clasa2(int Dim)
{          StareObiect = Distrus;
           SetUp(Dim);
}
Clasa2::~Clasa2() {  ClearAndDestroy();  }
```

intncls2.cpp

```
#include "clase2.hpp"
                        /*Are grij\ s\ nu distrug\ efectiv de dou\ ori consecutiv acela[i obiect.
                Aceasta previne printre altele eliberarea unei zone de memorie deja eliberat\ . */
void Clasa2::ClearAndDestroy(void)
{       if(!StareObiect)            //Dac\ obiectul este deja distrus ... .
                return;
        StareObiect = Distrus;
        if(!PtrIntPrivate)
                return;
        delete [Dim]PtrIntPrivate;                  //Este eliberat\ zona de memorie
                                                            //corespunz\toare vectorului ... .
        PtrIntPrivate = NULL;                       //Obicei bun ...
}
int Clasa2::Copy(Clasa2 const &C)
{       if(StareObiect)
                ClearAndDestroy();              //Este distrus explicit obiectul curent ... .
        if(!C.StareObiect)
                return success;
        StareObiect = C.StareObiect;
        Dim = C.Dim;
        if(!Dim)
          {     PtrIntPrivate = NULL;
                return success;
          }
        PtrIntPrivate = new int[Dim];               //Alocare dinamic\ a memoriei
                                                      //necesare pentru Dim int ... .
        if(!PtrIntPrivate)
                return error;
                                                      //Copierea valorilor din obiectul surs\ ... .
        for(int Contor = 0; Contor < Dim; Contor++)
                PtrIntPrivate[Contor] = C.PtrIntPrivate[Contor];
    /*                                  //Sau cu acces direct prin pointer ... .
        int *PtrIntDest = PtrIntPrivate;
        int *PtrIntSursa = C.PtrIntPrivate;
        for(int Contor = 0; Contor < Dim; Contor++)
                *PtrIntDest++ = *PtrIntSursa++;
    */
        StareObiect = SetUpCompleted;
        return success;
}
int Clasa2::SetUp(int Dim)
{  if(StareObiect) ClearAndDestroy();          //Este distrus explicit obiectul curent …
                                    //Se reconstruieste obiectul ...
  Clasa2::Dim = Dim;
  PtrIntPrivate = new int[Dim];
  if(!PtrIntPrivate)                        //Nu s-a putut aloca memorie pentru Dim int ... .
    {       StareObiect = Distrus;
            return error;
    }                       //S-a reu[it alocarea memoriei pentru Dim int ... .
  StareObiect = SetUpCompleted;
  return success;
}
```

maincls2.cpp

```
#include "clase2.hpp"

void main(void)
{
	Clasa2 Obiect1;

	Obiect1.PrintCon();
	Obiect1.ScanCon();
	Obiect1.PrintCon();

	Clasa2 Obiect2(Obiect1);
	Obiect2.PrintCon();
	Obiect1.SetUp(20);

	Obiect1.PrintCon();
	Obiect1.ScanCon();
	Obiect1.ClearAndDestroy();
	Obiect1.PrintCon();
	Obiect1.Copy(Obiect2);
	Obiect1.PrintCon();
	Obiect2.PrintCon();

	getch();
}
```

i&o_cls2.cpp

```
#include "clase2.hpp"

void Clasa2::ScanCon(void)
{
	if(!StareObiect || !PtrIntPrivate)
	   {
		cout << "Care este dimensiunea vectorului de intregi ? ";
		cin >> Dim;
		if(SetUp(Dim))
		  {
			cout << "Completati mesajul adecvat !!!" << endl;
			return;
		  }
	   }
	cout << "Introduceti valorile intregi ale elementelor vectorului de "
		<< Dim << " intregi:" << endl << "Indice\t\tvaloare" << endl;
	for(int Contor = 0; Contor < Dim; Contor++)
	{
		cout << Contor << " :\t\t";
		cin >> PtrIntPrivate[Contor];
	}
	return;
}
```

```
void Clasa2::PrintCon(void)
{
	if(!StareObiect)
	{
		cout << "Obiect neconstruit ..." << endl;
		return;
	}
	if(!PtrIntPrivate)
	{
		cout << "Vector vid ..." << endl;
		return;
	}
	cout << "Elementele vectorului de "
		<< Dim << " intregi:" << endl << "Indice\t\tvaloare" << endl;
	for(int Contor = 0; Contor < Dim; Contor++)
	{
		cout << Contor << " :\t\t" << PtrIntPrivate[Contor] << endl;
	}
	cout << "Press any key when ready ..." << endl;
	getch();
}
```

Observaþii:

1.Aceste fiºiere au fost realizate prin modificarea surselor “Clase1” ºi “Clase2”, puse la dispoziþie în cadrul laboratorului de “Programare Orientatã Obiect”.

2.Proiectele “Clase1” ºi “Clase2”, realizate cu aceste fiºiere au fost testate ºi funcþioneazã conform cerinþelor temei.

Com[a Ciprian
Grupa 5404

Programare Orientat\ Obiect

Tema 3

Clasã de vectori şi programul de test

vector.hpp

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
#include <ctype.h>

#ifndef _MATRIXCPP3_INCLUDED_
#define _MATRIXCPP3_INCLUDED_

enum Boolean
  {false = 0, true = 1};
enum Error
  {error = 0, success = 1};
enum ObjState
  {Destroyed = 0, Constructed = 1};

                              //Declara]ia de clas\ ...
class VectorInt
{
private:
        int State;                                  //Starea curent\ a obiectului ...
        int *Vector;
        int NrElem;
protected:
        void Copy(VectorInt const &);               //Realizeaz\ copierea ...
public:
        VectorInt(void);                            //Constructor implicit ...
        VectorInt(VectorInt const &);               //Constructor de copiere ...
        VectorInt(int);                             //Constructor ...
        ~VectorInt(void);                           //Destructor ...
        void ClearAndDestroy(void);
        int SetUp(int);                             //Returneaz\ starea de eroare ...
        void Print(ostream & = cout , int = 3) const;          //Tip\re[te vectorul la consol\ ...
        friend ostream & operator <<(ostream &, VectorInt const &);
        int & operator()(int);                      //Operator de acces aleatoriu ...
        VectorInt & operator =(VectorInt const &);          //Operator de atribuire ...
        VectorInt & operator +=(VectorInt const &);         //Operator de adunare ...
        VectorInt & operator *=(int);               //Operator de `nmul]ire cu scalar `ntreg ...
        VectorInt operator +(VectorInt const &);            //Operator de adunare ...
        VectorInt operator *(int);                  //Operator de `nmul]ire ...
        int operator ==(VectorInt const &);         //Operator rela]ional ...
        int GetNrElem(void) {return NrElem;}
        int * GetVector(void) {return Vector;}
};

#endif
```

consvec.cpp

```
#include "vector.hpp"

VectorInt::VectorInt(void)                    //Constructor implicit ...
{
   State = Constructed;
   Vector = NULL;
   NrElem = 0;
}

VectorInt::VectorInt(VectorInt const &V)      //Constructor de copiere ...
{
   State = Constructed;
   Vector = NULL;
   NrElem = 0;
   *this = V;                                 //Se folose[te operatorul de atribuire ...
}

VectorInt::VectorInt(int Nr)
{
   State = Constructed;
   Vector = NULL;
   NrElem = 0;
   SetUp(Nr);
}
```

destrvec.cpp

```
#include "vector.hpp"

//Elibereaz\ memoria alocat\ dinamic pentru un vector.
void VectorInt::ClearAndDestroy(void)
{
      if ( !((State)||(Vector)) )
        {
           NrElem=0;
           State=Destroyed;
           return;
        }
      delete [NrElem]Vector;
      NrElem=0;
      State=Destroyed;
      Vector=NULL;
}

VectorInt::~VectorInt(void)
{
          ClearAndDestroy();
}
```

alocavec.cpp

```
#include "vector.hpp"

void VectorInt::Copy(VectorInt const & V)            //Relizeaz\ o copie a matricii ...
{
   if (State != Destroyed)  ClearAndDestroy();
   if (!V.State)  return;
   State=V.State;
   NrElem=V.NrElem;
   if (!NrElem)
     {   Vector=NULL;
         return;
     }
   Vector = new int[NrElem];
   if (!Vector)
     {
      ClearAndDestroy();
      cout<<"Eroare la alocarea memeoriei...\n";
      return;
     }
/*  int *VectorTemp(V.Vector);
   for (int contor=0; contor<NrElem; contor++)
      Vector[contor]=VectorTemp[contor];

   SAU Direct, dar mai lent:  */
   for (int contor=0; contor<NrElem; contor++)
      Vector[contor]=V.Vector[contor];

/*    DE CE NU? :
   int *Vector(V.Vector);  //Ce efect are? Constructor doar?
*/

   State=Constructed;
   return;
}

//Aloc\ dinamic memorie pentu un vector cu NrElem elemente.
//Returneaz\ error sau success.
Int VectorInt::SetUp(int NrElem)
{  if (State != Destroyed)
      ClearAndDestroy();
   State = Constructed;
   VectorInt::NrElem = NrElem;
   Vector= new int[NrElem];
   if (!Vector)
     {
      ClearAndDestroy();
      cout<<”Eroare la alocarea memoriei…\n”;
      return error;
     }
   return success;
}
```

operavec.cpp

```
#include "vector.hpp"

int & VectorInt::operator()(int i)
{
   if(!State || i >= NrElem || i < 0 )
   {  cout << "Eroare la apelarea operatorului ()\n";
      return State;
   }
   return Vector[i];
}

VectorInt & VectorInt::operator =(VectorInt const &V)
{
   if(!V.State)
   {
      ClearAndDestroy();
      return *this;
   }
   ClearAndDestroy();                  //Este distrus vechiul vector ...
   Copy(V);                            //Vechiul vector este `nlocuit cu o copie a argumentului ...
   return *this;
}

VectorInt & VectorInt::operator +=(VectorInt const &V)
{
   if(!V.State)                        //Vectorul V nu exist\ ...
      return *this;
   int MustChange = 0;
   int NrElemTemp, NrElemTempPrim;
   NrElemTemp = NrElemTempPrim = V.NrElem;
   if(NrElem > NrElemTemp)
   {
      NrElemTemp = NrElem;
      MustChange = 1;
   }
if(MustChange)                         //Dac\ dimensiunea vectorului original  nu este cea adecvat\ ..
         {
                  VectorInt VectorIntTemp(*this);
                                 //Se realizeaz\ o copie temporar\ a obiectului curent ...
                  ClearAndDestroy();                  //Se distruge obiectul curent ...
                  if(!SetUp(NrElemTemp))              //Se redimensioneaz\ obiectul curent ...
                                 return *this;
                           //Se refac valorile vectorului original `n noul vector (mai mare) ...
                  int *VectorTemp=VectorIntTemp.Vector;
                  for(int contor = 0; contor < NrElem; contor++)
                     Vector[contor] = VectorTemp[contor];
                           //Se initializeaz\ cu 0 valorile din zona bordat\ ...
                  for(contor = 0; contor < NrElem; contor++)
                     Vector[contor] = 0;
                  for(contor = NrElem; contor < NrElemTemp; contor++)
                     Vector[contor] = 0;
         }
         int *VTemp=V.Vector;
```

```
	for(int contor = 0; contor < NrElemTempPrim; contor++)
		Vector[contor] += VTemp[contor];
	return *this;
}

VectorInt & VectorInt::operator *=(int Scalar)
{
	for(int contor = 0; contor < NrElem; contor++)
		Vector[contor] *= Scalar;
	return *this;
}

VectorInt VectorInt::operator +(VectorInt const &V)
{
	if(!V.State)
		return *this;
	VectorInt  VectorResult(*this);		//Constructor de copiere!
				//Vector rezultat initializat cu valoarea vectorului curent ...
	VectorResult += V;
	return VectorResult;			//Se `ntoarce rezultatul adun\rii ...
}

VectorInt VectorInt::operator *(int Scalar)
{
	VectorInt VectorIntTemp(NrElem);
	int *VectorTemp=VectorIntTemp.Vector;
	for(int contor = 0; contor < NrElem; contor++)
		VectorTemp[contor] = Vector[contor] * Scalar;
	return VectorIntTemp;
}

int VectorInt::operator ==(VectorInt const &V)
{
	if(NrElem != V.NrElem)
		return false;

	int *VectorTemp=V.Vector;
	for(int contor = 0; contor < NrElem; contor++)
		if(Vector[contor] != VectorTemp[contor])
			return false;
	return true;
}
```

mainvec.cpp

```
#include "vector.hpp"

void main(void)
{
   VectorInt Vector1, Vector2;
   int NrElem;
   cout <<endl<<endl<< "      ****************************************\n";
   cout <<                    "      * TEMA3  OOP  AN IV 5404B Comsa Ciprian *"<<endl;
   cout <<                    "      *          Test pentru clasa de vectori *"<<endl;
   cout <<                    "      ****************************************"<<endl;
   cout <<endl<< "Introduceti numarul de elemente ale vectorului Vector1 : ";
   cin >> NrElem;
   Vector1.SetUp(NrElem);
   cout <<endl<<"Vectorul Vector1 neinitializat:\n";
   Vector1.Print(cout,9);                              //Vectorul nu a fost initializat ...
   for(int cont = 0; cont < Vector1.GetNrElem(); cont++)
         Vector1(cont) = cont;
   cout <<endl<<"Vector1 initializat:\n";
   Vector1.Print();
   Vector2 = Vector1;
   cout <<endl<< "Vector2 = Vector1 !"<<endl;
   cout << "Relatia intre cei doi vectori este: " << (Vector2 == Vector1) << endl;
   cout <<endl<< "Vector2:"<<endl;
   Vector2.Print();
   VectorInt Vector3(Vector2);
   Vector1.ClearAndDestroy();
   cout <<endl<< "Am distrus Vector1! Am initializat Vector3: VectorInt
Vector3(Vector2)!\n";
   cout << "Vector3:"<<endl;
   Vector3.Print();
   Vector2 += Vector3;
   cout <<endl<< "Vector2+=Vector3; Vector2:"<<endl;
   Vector2.Print();
   Vector3 *= 100;
   cout <<endl<< "Vector3*=100; Vector3:"<<endl;
   Vector3.Print(cout, 5);
   VectorInt Vector4;                                  //Constructor implicit...
   Vector4 = Vector2 + Vector3;
   cout <<endl<< "Vector4=Vector2+Vector3; Vector4:"<<endl;
   Vector4.Print(cout, 5);
   VectorInt Vector5;
   Vector5 = Vector4 * 100;
   cout <<endl<< "Vector5=Vector4*100; Vector5:"<<endl;
   Vector5.Print(cout, 8);
   cout <<endl<< "Vector5 afisat cu operatorul supradefinit ~<<~ :" <<endl;
   cout << Vector5 <<endl<< "Gata ... pentru moment ..." << endl;
   Vector3 = Vector4 = Vector5 + Vector2;
   cout <<endl<< "Vector3=Vector4=Vector5+Vector2" <<endl;
   cout << "Vector3, Vector4, Vector5+Vector2, afisati cu operatorul supradefinit ~<<~ :";
   cout <<endl<< Vector3 << Vector4 << Vector5 + Vector2 << "Gata ...!" << endl;
   getch();
}
```

prntvec.cpp

```
#include "vector.hpp"

            /*Tip\re[te la consol\ vectorul de `ntregi specificat.*/
void VectorInt::Print(ostream &StrOut, int Width) const
{
   StrOut << "Sa afisez vectorul ? (y/n)";
   int Test;
   do
   {
      Test = toupper(cin.get());
   }
   while(Test != 'Y' && Test != 'N');

   if(Test == 'Y')
   {
      StrOut << "Vectorul este :\n";
      for(int contor = 0; contor < NrElem; contor++)
          StrOut << setw(Width) << Vector[contor];
      StrOut << endl;
   }
}

ostream & operator <<(ostream &StrOut, VectorInt const &V)
{
   V.Print(StrOut,9);
   return StrOut;
}
```

Observaþii:

1.Aceste fiºiere au fost realizate prin modificarea sursei “Lab4”, pusã la dispoziþie la laboratorul de “Programare Orientatã Obiect”.

2.Proiectul realizat cu aceste fiºiere a fost testat ºi funcþioneazã conform cerinþelor temei.