/*Plesa Diana lab06
1. Să se scrie o aplicaţie C++ care implementează o clasă numită PilotF1. Clasa definește variabilele private nume (șir de caractere), echipa (șir de caractere), varsta (int), record (int), nr_pole_position (int). Ca membri publici, clasa conține metode accesor/getter și mutator/setter distincte pentru fiecare din atributele clasei.
In funcția main( ), să se creeze 3 instanțe distincte ale clasei PilotF1 și să se folosească metodele mutator/setter pentru a inițializa datele din fiecare obiect cu informația corespunzătoare citită de la tastatură. Folosind metodele accesor/getter, să se afișeze toate datele pilotului cu cel mai bun record.*/

```
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
#include<string.h>
using namespace std;
#define DIM 30

class PilotF1 {
private:
 char nume[DIM];
 char echipa[DIM];
 int varsta;
 int record;
 int nr_pole_position;
public:
 void setNume(char a[])
 {
  strcpy(nume, a);
 }
 void setEchipa(char b[])
 {
  strcpy(echipa, b);
 }
 void setVarsta(int c)
 {
  varsta = c;
 }
 void setRecord(int d)
 {
  record = d;
 }
 void setPozitie(int e)
 {
  nr_pole_position = e;
 }
 char* getNume()
 {
  return nume;
 }
 char* getEchipa()
 {
  return echipa;
 }
 int getVarsta()
 {
```

```
 return varsta;
 }
 int getRecord()
 {
  return record;
 }
 int getPozitie()
 {
  return nr_pole_position;
 }
};

int main()
{
 PilotF1 p[3];
 int c, d, e;
 char a[DIM], b[DIM];
 for (int i = 0; i < 3; i++)
 {
  cout << "\n Citim datele pentru pilotul: " << i + 1;
  cout << "\n\nNumele pilotului: ";
  cin >> a;
  p[i].setNume(a);
  cout << "Echipa pilotului: ";
  cin >> b;
  p[i].setEchipa(b);
  cout << "Varsta: ";
  cin >> c;
  p[i].setVarsta(c);
  cout << "Recordul: ";
  cin >> d;
  p[i].setRecord(d);
  cout << "Pozitia pilotului: ";
  cin >> e;
  p[i].setPozitie(e);
 }
 cout << "\nDatele introduse sunt: ";
 for (int i = 0; i < 3; i++)
 {
  cout << "\n\nNumele pilotului este: " << p[i].getNume();
  cout << "\nDace parte din echipa: " << p[i].getEchipa();
  cout << "\nAvand varsta de " << p[i].getVarsta();
  cout << "\nRecordul personal este de: " << p[i].getRecord();
  cout << "\nClasandu-se pe pozitia: " << p[i].getPozitie();
 }
 int minim = 99999;
 for (int i = 0; i < 3; i++)
  if (p[i].getRecord() < minim)
   minim = p[i].getRecord();
 cout << "\n\n Afisarea candidatilor cu cel mai bun record:";
 for (int i = 0; i < 3; i++)
 {
  if (minim == p[i].getRecord()) {
   cout << "\n\nNumele pilotului este: " << p[i].getNume();
```

```
   cout << "\nDace parte din echipa: " << p[i].getEchipa();
   cout << "\nAvand varsta de " << p[i].getVarsta();
   cout << "\nRecordul personal este de: " << p[i].getRecord();
   cout << "\nClasandu-se pe pozitia: " << p[i].getPozitie();
  }
 }
}
```

```
/*Plesa Diana lab06
2. Să se modifice exemplul 2 astfel încât codul să poată fi lansat în execuție considerand
atributul clasei private.*/
#include <iostream>
using namespace std;

class Test2 {
int x;
public:
Test2() {
cout << "\nApel constructor explicit vid.";
}
void setX(int a) {
this->x = a;
}
};

int main() {
Test2 ob1; //instantiere imposibila, constructorul este privat!
int a;
cout << "\nIntroduceti valoarea variabilei de tip \"int\" din clasa: ";
cin >> a;
ob1.setX(a);
}
```

```
/*Plesa Diana lab06
3. Pornind de la exemplul care tratează lucrul cu matrice in varianta transformata cu alocare
dinamica, completați codul scris cu metodele specifice pentru:
- afișarea elementelor de pe diagonala secundara a matricei, dacă matricea este pătratică; în
caz contrar se afișează un mesaj corespunzător;
- afișarea elementelor de sub diagonala principala;
- afișarea unei matrice de dimensiunea celei inițiale ale cărei elemente pot avea valori de 0
(dacă elementul corespunzător este mai mare decât o valoare citita) sau 1 (în caz contrar);*/

#include<iostream>
using namespace std;

class Matrix {
 int matrix[10][10], dim1, dim2;
private:
 int returnElement(int row, int column)
 {
  return matrix[row][column];
 }
```

```
public:
 Matrix()
 {
  int i, j;
  cout << "\nIntroduceti dimensiunile matricii: ";
  cin >> dim1;
  cin >> dim2;
  cout << "\n Introduceti elementele matricii: ";
  for (int i = 0; i < dim1; i++)
   for (int j = 0; j < dim2; j++)
   {
    cout << "\n matrix[" << i << "][" << j << "]=";
    cin >> matrix[i][j];
   }
 }
 void displayMatrix()
 {
  int i, j;
  cout << "\n Elementele matricii: ";
  for (i = 0; i < dim1; i++)
  {
   cout << "\n";
   for (j = 0; j < dim2; j++)
    cout << returnElement(i, j) << " ";
  }
  cout << "\n";
 }
 void displayColumn(int col)
 {
  if (col < 0 || col > dim2)
   cout << "\nColoana cu nr: " << col << " nu exista in matricea clasa!\n";
  else
  {
   cout << "\n Elementele coloanei " << col << " sunt: ";
   for (int i = 0; i < dim1; i++)
    cout << returnElement(i, col) << " ";
  }
 }
 void elem_diagsec()
 {
  if (dim1 == dim2)
  {
   for (int i = 0; i < dim1; i++)
    cout << matrix[i][dim1 - 1 - i];
  }
  else cout << "\nMatricea nu este patratica!";
 }
 void elem_subdiagpr()
 {
  if (dim1 == dim2)
   for (int i = 0; i < dim1; i++)
   {
    for (int j = 0; j < i; j++)
     cout << matrix[i][j] << " ";
```

```
   cout << endl;
  }
 else cout << "\nMatricea nu este patratica!";
}
void init_matrix2(int val)
{
 int matrix2[10][10];
 for (int i = 0; i < dim1; i++)
  for (int j = 0; j < dim2; j++)
   if (matrix[i][j] > val)
    matrix2[i][j] = 0;
   else matrix2[i][j] = 1;
 cout << "\nNoua matrice este: \n";
 for (int i = 0; i < dim1; i++)
 {
  for (int j = 0; j < dim2; j++)
   cout << matrix2[i][j] << " ";
  cout << endl;
 }
}
};

int main()
{
 Matrix m1;
 int c;
 m1.displayMatrix();
 cout << "\n Elemtentele carei coloane vor fi afisate: ";
 cin >> c;
 m1.displayColumn(c - 1);
 cout << "\n Elementele de pe diag secundara sunt: ";
 m1.elem_diagsec();
 cout << "\n Elementele de sub diag princ sunt: ";
 m1.elem_subdiagpr();
 int val;
 cout << "\n Introduceti valoarea cu care comparam elem primei matrici pentru a initializa cu 1 si 0 noua matrice : ";
  cin >> val;
 m1.init_matrix2(val);
}


/*Plesa Diana lab06
4. Să se scrie o clasă care are ca variabilă privată un câmp de tip dată, definit într-o structură
externă clasei (zi – int, luna – int, an - int). Clasa conține metode mutator/setter și
accesor/getter (publice) pentru informația privată. In clasă se mai află doua metode publice
care:
- testează validitatea datei stocate;
- scrie într-un fișier toate datele din anul curent care preced (cronologic) data stocată în
clasă;
In functia main( ), după instanțierea clasei și citirea de la tastatură a componentelor unei date,
să se apeleze metodele membre și apoi să se verifice rezultatele obținute. */

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
```

```
using namespace std;

struct Data
{
 int day;
 int month;
 int year;
};

class Date
{
    struct Data x;
public:
 int getDay()
 {
  return x.day;
 }
 int getMonth()
 {
  return x.month;
 }
 int getYear()
 {
  return x.year;
 }
 void setDay(int day)
 {
  x.day = day;
 }
 void setMonth(int month)
 {
  x.month = month;
 }
 void setYear(int year)
 {
  x.year = year;
 }
 bool validity();
 void precede_dates();
};

bool Date::validity()
{
 static int pos_day[] = { 0,31,28,31,30,31,30,31,31,30,31,30,31 };
 if (x.year < 1500 || x.year>4000)
  return false;
 if (x.month <= 0 || x.month > 12)
  return false;
 if (x.day <= 0 || x.day > pos_day[x.month])
  if (x.month == 2 && x.day == 29 && x.year % 4 == 0)
   return true;
  else
   return false;
}
```

```
void Date::precede_dates()
{
 FILE* f;
 if ((f = fopen("Text.txt", "w")) == NULL)
 {
  cout << "Eroare!";
  exit(1);
 }
 static int pos_day[] = { 0,31,28,31,30,31,30,31,31,30,31,30,31 };
 for (int i = 1; i < x.month; i++)
  for (int j = 1; j <= pos_day[i]; j++)
  {
   fprintf(f, "\n%d.%d.%d", j, i, x.year);
   if (x.year % 4 == 0 && i == 2 && j == 28)
   {
    fprintf(f, "\n29.%d.%d", i, x.year);
   }
  }
 for (int i = 1; i < x.day; i++)
 {
  fprintf(f, "\n%d.%d.%d", i, x.month, x.year);
 }
 fclose(f);
}

int main()
{
 Date obj;
 int x;
 cout << "\nZi: ";
 cin >> x;
 obj.setDay(x);
 cout << "\nLuna: ";
 cin >> x;
 obj.setMonth(x);
 cout << "\nAn: ";
 cin >> x;
 obj.setYear(x);
 if (obj.validity() == true)
  cout << "\nData corecta";
 else {
  cout << "\nData incorecta";
  return 1;
 }
 cout << "\nDatele din anul curent care preced (cronologic) data stocata in clasa: ";
 obj.precede_dates();
 return 0;
}
```

/*Plesa Diana lab06
6. Să se scrie o aplicaţie C++ care implementează o clasă numită Triunghi. Clasa cuprinde atributele private pentru laturile a, b, c, un constructor cu parametrii, metode setter si getter

adecvate. Calculați aria și perimetrul prin metode specifice clasei. Scrieți o metodă care să indice dacă triunghiul este dreptunghic sau nu. Definiti o metoda private cu parametrii in clasa care permite verificarea condiției ca laturile să formeze un triunghi. Ea va fi folosita si de metodele setter.*/

```
//Triunghi.h
class Triunghi
{
 int a, b, c;
 int ok(int, int, int);
public:
 Triunghi(int a = 0, int b = 0, int c = 0)
 {
  this->a = a;
  this->b = b;
  this->c = c;
 }
 void setA(int x)
 {
  if (ok(x, b, c))
   a = x;
  else
  {
   cout << "\nNu se poate forma un triunghi";
   exit(1);
  }
 }
 void setB(int x)
 {
  if (ok(x, a, c))
   b = x;
  else
  {
   cout << "\nNu se poate forma un triunghi";
   exit(1);
  }
 }
 void setC(int x)
 {
  if (ok(x, b, a))
   c = x;
  else
  {
   cout << "\nNu se poate forma un triunghi";
   exit(1);
  }
 }
 int getA()
 {
  return a;
 }
 int getB()
 {
  return b;
```

```
 }
 int getC()
 {
  return c;
 }
 int perimetru();
 float aria();
 void verificare();
};

int Triunghi::ok(int x, int y, int z)
{
 if (x <= 0) return 0;
 if (y == 0 || z == 0) return 1;
 if (x + y > z && x + z > y && y + z > x) return 1;
 return 0;
}

int Triunghi::perimetru()
{
 return a + b + c;
}

float Triunghi::aria()
{
 float p = perimetru() / 2.0f;
 return sqrt(p * (p - a) * (p - b) * (p - c));
}

void Triunghi::verificare()
{
 if (a * a + b * b == c * c || a * a + c * c == b * b || b * b + c * c == a * a)
  cout << "\nTriunghiul este dreptunghic";
 else
  cout << "\nTriunghiul nu este dreptunghic";
}


//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
#include "Triunghi.h"

int main()
{
 Triunghi obj;
 int x;
 cout << "\na = ";
 cin >> x;
 obj.setA(x);
 cout << "\nb = ";
 cin >> x;
 obj.setB(x);
```

```
 cout << "\nc = ";
 cin >> x;
 obj.setC(x);
 cout << "\nAria este: " << obj.aria();
 cout << "\nPerimetrul este: " << obj.perimetru();
 obj.verificare();
 return 0;
}


/*Plesa Diana lab06
7. Să se scrie clasa Seif, cu atributele private cifru și suma. Descrieți metodele private getSuma()
și setSuma( ) și metodele publice puneInSeif( ) și scoateDinSeif( ) cu care să accesați suma de
bani care se află în seif. Metoda puneInSeif( ) poate apela getSuma() și setSuma( ), metoda
scoateDinSeif( ) poate apela getSuma( ) și setSuma( ). Instanțiați obiecte din clasa Seif, iar
metodele puneInSeif( ) și scoateDinSeif( ) vor putea accesa suma doar dacă parametrul de tip
cifru utilizat corespunde obiectului instanțiat. În caz de diferență de cifru, se va da un mesaj.*/

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;

class Seif
{
 int cifru;
 int suma;
 int getSuma()
 {
  return suma;
 }
 void setSuma(int suma)
 {
  this->suma = suma;
 }
public:
 Seif(int suma = 0, int cifru = 1234)
 {
  this->cifru = cifru;
  this->suma = suma;
 }
 void puneInSeif(int, int);
 void scoateDinSeif(int, int);
};

void Seif::puneInSeif(int cifru, int suma)
{
 if (cifru == this->cifru)
 {
  cout << "\nSuma intiala: " << getSuma();
  setSuma(suma + getSuma());
  cout << "\nNoua suma: " << getSuma();
 }
 else
 {
```

```
  cout << "\nCifru incorect";
  exit(1);
 }
}

void Seif::scoateDinSeif(int cifru, int x)
{
 if (cifru == this->cifru)
 {
  if (x <= getSuma())
  {
   cout << "\nSuma intiala: " << getSuma();
   setSuma(getSuma() - x);
   cout << "\nNoua suma: " << getSuma();
  }
  else
   cout << "\nSuma pentru retragere este mai mare decat suma din seif " << getSuma();
 }
 else
 {
  cout << "\nCifru incorect";
  exit(1);
 }
}

int main()
{
 Seif obj;
 int x;
 int cifru;
 cout << "\nIntroduceti cifrul: ";
 cin >> cifru;
 cout << "\nIntroduceti suma de adaugat: ";
 cin >> x;
 obj.puneInSeif(cifru, x);
 cout << "\nIntroduceti suma de scos din seif: ";
 cin >> x;
 obj.scoateDinSeif(cifru, x);
}
```

/*Plesa Diana lab07
1. Modificaţi exemplul 3 astfel încât să permită obtinerea unui nou punct, avand coordonatele obtinute prin adunarea coordonatelor a două astfel de puncte. Numele noului punct va fi rezultat prin concatenarea numelor celor două puncte. Adaugati si testati o metoda care calculeaza distanta de la un punct la origine. Modificaţi clasa astfel încât sa eliminaţi metoda afis() folosind in schimb metode accesor adecvate. Eliminaţi de asemenea atributul lungime_sir modificând adecvat metodele clasei. Testati utilizand si functii specifice sirurilor de caractere din VC++1y/2z (strcpy_s( ) si strcat_s( )).*/

```
//CPunctText.h
const int dim_sir = 20;

class CPunctText {
 int x;
 int y;
 char* sNume;
public:
 //constructor explicit vid
 CPunctText();
 //constructor cu parametri
 CPunctText(int ix, int iy, const char* sText = "Punct");
 //constructor de copiere
 CPunctText(const CPunctText& pct);
 //destructor:
 ~CPunctText();
 int getX()
 {
  return x;
 }
 int getY()
 {
  return y;
 }
 char* getSNume()
 {
  return sNume;
 }
 CPunctText sum(CPunctText);
 double distance();
};

CPunctText::CPunctText() {
 cout << "\n constructor explicit vid";
 sNume = new char[dim_sir];
}
CPunctText::CPunctText(int ix, int iy, const char* sText) {
 cout << "\n constructor cu parametri";
 sNume = new char[strlen(sText) + 1];
 x = ix;
 y = iy;
 cout << "\nLungime sir: " << sizeof(sNume);
 strcpy(sNume, sText);
}
CPunctText::CPunctText(const CPunctText& pct) {
```

```
 cout << "\n constructor de copiere";
 sNume = new char[strlen(pct.sNume) + 1];
 x = pct.x;
 y = pct.y;
 strcpy(sNume, pct.sNume);
}
CPunctText::~CPunctText() {
 cout << "\n destructor";
 delete[] sNume;
}
CPunctText CPunctText::sum(CPunctText pct)
{
 CPunctText c;
 c.x = x + pct.x;
 c.y = y + pct.y;
 strcpy(c.sNume, sNume);
 strcat(c.sNume, pct.sNume);
 return c;
}
double CPunctText::distance()
{
 return sqrt(x * x + y * y);
}

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
#include "CPunctText.h"

int main() {
 CPunctText cpt1(1, 2, "Punct1");//apel constructor cu parametri
 CPunctText cpt2(cpt1); //apel constructor de copiere
 CPunctText cpt3 = cpt2; //apel constructor de copiere
 CPunctText cpt4(4, 5); //apel constructor cu parametri
 CPunctText cpt5(3, 4, "Punct5");
 cout << "\nDistanta de la origine la " << cpt5.getSNume() << "este " << cpt5.distance();
 CPunctText cpt = cpt1.sum(cpt5); //copiere constructor
 cout << "\nSuma este " << cpt.getSNume() << " " << cpt.getX() << " " << cpt.getY();
}//main


/*Plesa Diana lab07
2. Să se scrie o aplicaţie C/C++ care să modeleze obiectual un tablou unidimensional de numere
reale. Creaţi două instanţe ale clasei şi afişaţi valorile unui al 3-lea tablou, obţinute prin
scăderea elementelelor corespunzătoare din primele 2 tablouri. Dacă tablourile au lungimi
diferite, tabloul rezultat va avea lungimea tabloului cel mai scurt*/

//Tablou.h
const int dim_tab = 50;

class Tablou {
 double* tab;
 int lungime;
```

```
public:
 Tablou()
 {
  lungime = dim_tab;
  tab = new double[lungime];
 }
 Tablou(int n)
 {
  this->lungime = n;
  tab = new double[n];
 }
 Tablou(const Tablou& s)
 {
  lungime = s.lungime;
  tab = new double[lungime];
  for (int i = 0; i < lungime; i++)
   *(tab + i) = s.tab[i];
 }
 int getLungime()
 {
  return lungime;
 }
 void setTab(double* tablou)
 {
  for (int i = 0; i < lungime; i++)
  {
   *(tab + i) = *(tablou + i);
  }
 }
 void afisTablou()
 {
  for (int i = 0; i < lungime; i++)
   cout << tab[i] << " ";
 }
 Tablou difTablou(Tablou, int);
};
Tablou Tablou::difTablou(Tablou ob, int l)
{
 Tablou dif_tablou(l);
 for (int i = 0; i < lungime; i++)
  dif_tablou.tab[i] = tab[i] - ob.tab[i];
 return dif_tablou;
}

//main
#include <iostream>
using namespace std;
#include Tablou.h

void citire(double*, int);

int main()
{
 int n1, n2;
```

```
 double* buf;
 cout << "\nIntroduceti dimensiunea primului tablou: ";
 cin >> n1;
 cout << "\nIntroduceti dimensiunea al doilea tablou: ";
 cin >> n2;
 Tablou obj1(n1), obj2(n2), rez;
 cout << "\nSirul 1: ";
 if (!(buf = new double[obj1.getLungime()]))
 {
  cout << "\nEroare!";
  exit(1);
 }
 citire(buf, obj1.getLungime());
 obj1.setTab(buf);
 delete[]buf;
 cout << "\nSirul 2: ";
 if (!(buf = new double[obj2.getLungime()]))
 {
  cout << "\nEroare!";
  exit(1);
 }
 citire(buf, obj2.getLungime());
 obj2.setTab(buf);
 delete[]buf;
 int l;
 if (obj1.getLungime() < obj2.getLungime())
  l = obj1.getLungime();
 else
  l = obj2.getLungime();
 rez = obj1.difTablou(obj2, l);
 cout << "\nDiferenta dintre cele 2 este: ";
 rez.afisTablou();
 return 0;
}

void citire(double* buf, int n)
{
 for (int i = 0; i < n; i++)
 {
  cout << "\n\t[" << i + 1 << "] = ";
  cin >> *(buf + i);
 }
}
```

/*Plesa Diana lab07
3. Modelați clasa Student care să conțină atributele private nume, prenume, note (tablou 7 valori int), grupa. Alocați dinamic memorie pentru n studenți. Calculați media cu o metoda din clasa și sortați studenții după medie, afisând datele fiecărui student (nume, prenume, grupa, medie). Implementati si destructorul clasei care să afișeze un mesaj.*/

```
//Student.h
const int dim_tab = 7;
const int dim_char = 20;
```

```
class Student
{
 char nume[dim_char];
 char prenume[dim_char];
 int note[dim_tab];
 int grupa;
public:
 void setNume(char nume[])
 {
  strcpy(this->nume, nume);
 }
 void setPrenume(char prenume[])
 {
  strcpy(this->prenume, prenume);
 }
 void setGrupa(int grupa)
 {
  this->grupa = grupa;
 }
 char* getNume()
 {
  return nume;
 }
 char* getPrenume()
 {
  return prenume;
 }
 int getGrupa()
 {
  return grupa;
 }
 void setNote(int* buf)
 {
  for (int i = 0; i < dim_tab; i++)
   note[i] = *(buf + i);
 }
 double med_note()
 {
  double sum = 0.;
  for (int i = 0; i < dim_tab; i++)
   sum += note[i];
  return (sum / dim_tab);
 }
 ~Student()
 {
  cout << "\nDestructor";
 }
};


//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
```

```
using namespace std;
#include Student.h

int cmp(Student*, Student*);

int main()
{
 int n, * buf, gr;
 char nume[dim_char];
 Student* obj;
 cout << "\nIntroduceti numarul de studenti: ";
 cin >> n;
 if (!(obj = new Student[n]))
 {
  cout << "\nEroare!";
  exit(1);
 }
 for (int i = 0; i < n; i++)
 {
  cout << "\nStudentul " << i + 1;
  cout << "\nNume: ";
  cin >> nume;
  (obj + i)->setNume(nume);
  cout << "\nPrenume: ";
  cin >> nume;
  (obj + i)->setPrenume(nume);
  cout << "\nGrupa: ";
  cin >> gr;
  (obj + i)->setGrupa(gr);
  cout << "\nIntroduceti notele: ";
  if (!(buf = new int[dim_tab]))
  {
   cout << "\nEroare!";
   exit(1);
  }
  for (int j = 0; j < dim_tab; j++)
  {
   cout << "\n\tNota " << j + 1 << ": ";
   cin >> buf[j];
  }
  (obj + i)->setNote(buf);
  delete[]buf;
 }
 qsort(obj, n, sizeof(Student), (int(*)(const void*, const void*))cmp);
 cout << "\nStudentii ordonati: ";
 for (int i = 0; i < n; i++)
 {
  cout << "\n" << (obj + i)->getNume() << " " << (obj + i)->getPrenume() << "grupa " << (obj + i)->getGrupa() << "
media notelor: " << (obj + i)->med_note();
 }
 delete[]obj;
 return 0;
}
```

```
int cmp(Student* obj1, Student* obj2)
{
 double a1, a2;
 a1 = obj1->med_note();
 a2 = obj2->med_note();
 if (a1 < a2) return 1;
 if (a1 == a2) return 0;
 return -1;
}


/*Plesa Diana lab07
4. Să se scrie o aplicaţie în care se modelează clasa Student cu nume, prenume, numar note si
notele din sesiunea din iarnă declarat printr-un pointer de tip int. Să se afişeze numele
studenţilor din grupă care au restanțe și apoi numele primilor 3 studenți din grupă în ordinea
mediilor, care se va afisa si ea*/

//Student.h
const int dim_tab = 5;


class Student
{
 char nume[dim_buf];
 char prenume[dim_buf];
 int* note;
 int nr_note;
public:
 void setNume(char[]);
 void setPrenume(char[]);
 void setNr_note(int);
 char* getNume();
 char* getPrenume();
 int getNr_note();
 void setNote(int*);
 void afisNote();
 int restanta();
 double media();
};

void Student::setNume(char nume[])
 {
  strcpy(this->nume, nume);
 }
void Student::setPrenume(char prenume[])
 {
  strcpy(this->prenume, prenume);
 }
void Student::setNr_note(int nr_note)
 {
  this->nr_note = nr_note;
 }
void Student::setNote(int* p)
{
```

```
 if (!(note = new int[nr_note]))
 {
  cout << "\nEroare!";
  exit(1);
 }
 for (int i = 0; i < nr_note; i++)
  *(note + i) = *(p + i);
}
char* Student::getNume()
 {
  return nume;
 }
char* Student::getPrenume()
 {
  return prenume;
 }
int Student::getNr_note()
 {
    return nr_note;;
 }
void Student::afisNote()
{
 for (int i = 0; i < nr_note; i++)
  cout << *(note + i) << " ";
}
int Student::restanta()
{
 for (int i = 0; i < nr_note; i++)
  if (*(note + i) < dim_tab) return 1;
 return 0;
}
double Student::media()
{
 double s = 0.;
 if (restanta() != 0) return 0;
 for (int i = 0; i < nr_note; i++)
  s += *(note + i);
 return s / nr_note;
}


//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
const int best = 3;
const int dim_buf = 20;
#include Student.h

int cmp(Student*, Student*);

int main()
{
 int n, * p, x;
```

```
 char buf[dim_buf];
 Student* obj;
 cout << "\nIntroduceti numarul de studenti: ";
 cin >> n;
 if (!(obj = new Student[n]))
 {
  cout << "\nEroare!";
  exit(1);
 }
 for (int i = 0; i < n; i++)
 {
  cout << "\nStudentul " << i + 1;
  cout << "\nNume: ";
  cin >> buf;
  obj[i].setNume(buf);
  cout << "\nPrenume: ";
  cin >> buf;
  obj[i].setPrenume(buf);
  cout << "\nNumarul de note: ";
  cin >> x;
  obj[i].setNr_note(x);
  if (!(p = new int[obj[i].getNr_note()]))
  {
   cout << "\nEroare!";
   exit(1);
  }
  for (int j = 0; j < obj[i].getNr_note(); j++)
  {
   cout << "\n\tSubiect " << j + 1 << ": ";
   cin >> *(p + j);
  }
  obj[i].setNote(p);
 }
 cout << "\nStudenti care au restante: ";
 for (int i = 0; i < n; i++)
  if (obj[i].restanta() != 0)
   cout << obj[i].getNume() << " " << obj[i].getPrenume();
 qsort(obj, n, sizeof(Student), (int(*)(const void*, const void*))cmp);
 cout << "\nPrimii 3 studenti sunt: ";
 for (int i = 0; i < best; i++)
 {
  cout << "\n" << obj[i].getNume() << " " << obj[i].getPrenume() << "Media =  " << obj[i].media();
 }
}

int cmp(Student* obj1, Student* obj2)
{
 double a1, a2;
 a1 = obj1->media();
 a2 = obj2->media();
 if (a1 < a2) return 1;
 if (a1 == a2) return 0;
 return -1;
}
```

/*Plesa Diana lab07
5. Să se scrie o aplicaţie C/C++ în care se citeşte de la tastatură un punct prin coordonatele x, y, z. Să se scrie o metodă prin care să se facă translaţia punctului cu o anumită distanţă pe fiecare dintre cele trei axe. Să se verifice dacă dreapta care uneşte primul punct şi cel rezultat în urma translaţiei trec printr-un al treilea punct dat de la consolă.*/

```
//Punct.h
class Punct
{
 double x, y, z;
public:
 Punct()
 {
  x = 0;
  y = 0;
  z = 0;
 }
 Punct(const Punct& pct)
 {
  x = pct.x;
  y = pct.y;
  z = pct.z;
 }
 void setX(double x)
 {
  this->x = x;
 }
 void setY(double y)
 {
  this->y = y;
 }
 void setZ(double z)
 {
  this->z = z;
 }
 double getX()
 {
  return x;
 }
 double getY()
 {
  return y;
 }
 double getZ()
 {
  return z;
 }
 void move(double, double, double);
 int cross(Punct, Punct);
};
void Punct::move(double a, double b, double c)
{
```

```
 x = a;
 y = b;
 z = c;
}
int Punct::cross(Punct pct1, Punct pct2)
{
 if (((x - pct1.x) / (pct2.x - pct1.x)) == ((y - pct1.y) / (pct2.y - pct1.y)) && ((x - pct1.x) / (pct2.x - pct1.x)) == ((z - pc
t1.z) / (pct2.z - pct1.z)))
  return 1;
 else
  return 0;
}


//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
#include Punct.h

int main()
{
 double var;
 Punct pct1, pct2, pct_cpy, * p;
 p = &pct1;
 cout << "\nPrimul punct: ";
 cout << "\n\tx = ";
 cin >> var;
 p->setX(var);
 cout << "\n\ty = ";
 cin >> var;
 p->setY(var);
 cout << "\n\tz = ";
 cin >> var;
 p->setZ(var);
 pct_cpy = pct1;
 double a, b, c;
 cout << "\nIntroduceti valorile pe care vreti sa le adaugati ficarei axe: ";
 cin >> a >> b >> c;
 pct1.move(a, b, c);
 cout << "\nPunctul mutat este (" << p->getX() << ", " << p->getY() << ", " << p->getZ() << ")";
 p = &pct2;
 cout << "\nPunctul doi: ";
 cout << "\n\tx = ";
 cin >> var;
 p->setX(var);
 cout << "\n\ty = ";
 cin >> var;
 p->setY(var);
 cout << "\n\tz = ";
 cin >> var;
 p->setZ(var);
 if (pct2.cross(pct1, pct_cpy))
  cout << "\nDreapta care uneste primul punct si cel rezultat in urma translatiei trece printr - un al treilea punct dat de
```

```
 la consola";
 else
  cout << "\nDreapta care uneste primul punct si cel rezultat in urma translatiei nu trece printr - un al treilea punct da
t de la consola";
 return 0;
}


/*Plesa Diana lab07
6. Definiți o clasă Complex modelată prin atributele de tip double real, imag și un pointer de tip
char către numele fiecărui număr complex. În cadrul clasei definiți un constructor explicit cu
doi parametri care au implicit valoarea 1.0 și care alocă spațiu pentru nume un șir de maxim
15 caractere, de exemplu "c1". De asemenea, definiți un constructor de copiere pentru clasa
Complex. Clasa va mai conține metode mutator/setter și accesor/getter pentru fiecare membru
al clasei, metode care permit operațiile de bază cu numere complexe și un destructor explicit.
Definiți cel mult 10 numere complexe într-un tablou. Calculați suma numerelor complexe din
tablou, valoare ce va fi folosită pentru a inițializa un nou număr complex, cu numele
"Suma_c". Realizați aceleași acțiuni făcând diferența și produsul numerelor complexe*/

//Complex.h
class Complex
{
private:
 double real, imag;
 char* name;
public:
 Complex(double re = 1.0, double im = 1.0)
 {
  real = re;
  imag = im;
  name = new char[7];
 }
 Complex(const Complex& nr)
 {
  name = new char[strlen(nr.name) + 1];
  real = nr.real;
  imag = nr.imag;
  strcpy(name, nr.name);
 }
 void setReal(double r)
 {
  real = r;
 }
 void setImag(double i)
 {
  imag = i;
 }
 void setNume(const char buf[])
 {
  strcpy(name, buf);
 }
 double getReal()
 {
  return real;
```

```
 }
 double getImag()
 {
  return imag;
 }
 char* getNume()
 {
  return name;
 }
 Complex sum(Complex c);
 Complex diff(Complex c);
 Complex inm(Complex c);
 Complex imp(Complex c);
 ~Complex()
 {
  delete[]name;
 }
};
Complex Complex::sum(Complex c)
{
 Complex rez;
 rez.imag = (c.imag + imag);
 rez.real = (c.real + real);
 return rez;
}
Complex Complex::diff(Complex c)
{
 Complex rez;
 rez.imag = (imag - c.imag);
 rez.real = (real - c.real);
 return rez;
}
Complex Complex::inm(Complex c)
{
 Complex rez;
 rez.imag = (c.imag * real) + (c.real * imag);
 rez.real = (c.real * real) - (c.imag * imag);
 return rez;
}
Complex Complex::imp(Complex c)
{
 Complex rez;
 rez.imag = (imag * c.real + real * c.imag) / (c.real * c.real + c.imag * c.imag);
 rez.real = (real * c.real + imag * c.imag) / (c.real * c.real + c.imag * c.imag);
 return rez;
}


//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
const int dim = 3;
const int dim_char = 7;
```

```
#include Complex.h

int main()
{
 Complex nr[dim], rez(0, 0);
 double x, s = 0;
 char* name;
 for (int i = 0; i < dim; i++)
 {
  cout << "\nNumraul " << i + 1 << " : ";
  cout << "\nNume: ";
  name = new char[dim_char];
  cin >> name;
  nr[i].setNume(name);
  delete[]name;
  cout << "\nReal = ";
  cin >> x;
  nr[i].setReal(x);
  cout << "\nImaginar = ";
  cin >> x;
  nr[i].setImag(x);
 }
 for (int i = 0; i < dim; i++)
 {
  rez.setReal(rez.sum(nr[i]).getReal());
  rez.setImag(rez.sum(nr[i]).getImag());
 }
 Complex complex_sum(rez);
 complex_sum.setNume("Complex_sum");
 cout << "\n" << complex_sum.getNume() << " = " << complex_sum.getReal() << " + (" << complex_sum.getImag(
) << ")i";
 rez.setReal(nr[0].getReal());
 rez.setImag(nr[0].getImag());
 for (int i = 0; i < dim; i++)
 {
  rez.setReal(rez.diff(nr[i]).getReal());
  rez.setImag(rez.diff(nr[i]).getImag());
 }
 Complex complex_diff(rez);
 complex_diff.setNume("Complex_diff");
 cout << "\n" << complex_diff.getNume() << " = " << complex_diff.getReal() << " + (" << complex_diff.getImag()
<< ")i";
 rez.setReal(nr[0].getReal());
 rez.setImag(nr[0].getImag());
 for (int i = 0; i < dim; i++)
 {
  rez.setReal(rez.inm(nr[i]).getReal());
  rez.setImag(rez.inm(nr[i]).getImag());
 }
 Complex complex_inm(rez);
 complex_inm.setNume("Complex_inm");
 cout << "\n" << complex_inm.getNume() << " = " << complex_inm.getReal() << " + (" << complex_inm.getImag()
 << ")i";
 rez.setReal(nr[0].getReal());
```

```
 rez.setImag(nr[0].getImag());
 for (int i = 0; i < dim; i++)
 {
  rez.setReal(rez.imp(nr[i]).getReal());
  rez.setImag(rez.imp(nr[i]).getImag());
 }
 Complex complex_imp(rez);
 complex_imp.setNume("Complex_imp");
 cout << "\n" << complex_imp.getNume() << " = " << complex_imp.getReal() << " + (" << complex_imp.getImag()
 << ")i";
 rez.setReal(nr[0].getReal());
 rez.setImag(nr[0].getImag());
}
```

/*Plesa Diana lab07
7. Consideram clasa Fractie care are doua atribute intregi private a si b pentru numarator si numitor, doua metode de tip set( ) respectiv get( ) pentru atributele clasei publice si o metoda simplifica( ) publica care simplifica obiectul curent Fractie de apel, returnând un alt obiect simplificat. Metoda simplifica( ) va apela o metoda private cmmdc( ) pentru simplificarea fracției. Definiți un constructor explicit fara parametri care initializeaza a cu 0 si b cu 1, si un constructor explicit cu doi parametri care va fi apelat dupa ce s-a verificat posibilitatea definirii unei fractii (b!=0). Definiți o metoda aduna_fractie( ) care are ca si parametru un obiect de tip Fractie si returneaza suma obiectului curent de apel cu cel dat ca si parametru, ca si un alt obiect de tip Fractie. Analog definiti metode pentru scadere, inmultire si impartire. Instantiati doua obiecte de tip Fractie cu date citite de la tastatura. Afisati atributele initiale si cele obtinute dupa apelul metodei simplifica( ). Efectuati operatiile implementate prin metodele clasei si afisati rezultatele.*/

```
//Fractie.h
class Fractie
{
 int a, b;
 int cmmdc();
public:
 Fractie()
 {
  a = 0;
  b = 1;
 }
 Fractie(int a, int b)
 {
  if (b != 0)
  {
   this->a = a;
   this->b = b;
  }
  else
  {
   cout << "\nNumaratorul este 0! Il transformam in 1.";
   b = 1;
  }
 }
 void setA(int x)
```

```
 {
  a=x;
 }
 void setB(int x)
 {
  if (x != 0)
   b = x;
  else {
   cout << "\nNumaratorul este 0! Il transformam in 1.";
   b = 1;
  }
 }
 int getA()
 {
  return a;
 }
 int getB()
 {
  return b;
 }
 Fractie simplif();
 Fractie adunare(Fractie);
 Fractie scadere(Fractie);
 Fractie inm(Fractie);
 Fractie imp(Fractie);
};

int Fractie::cmmdc()
{
 int aux, x, y;
 x = a;
 y = b;
 while (y)
 {
  aux = x % y;
  x = y;
  y = aux;
 }
 return x;
}
Fractie Fractie::simplif()
{
 Fractie frac;
 frac.a = a / cmmdc();
 frac.b = b / cmmdc();
 return frac;
}
Fractie Fractie::adunare(Fractie obj)
{
 Fractie rez;
 rez.a = a * obj.b * obj.a;
 rez.b = b * obj.b;
 return rez;
}
```

```
Fractie Fractie::inm(Fractie obj)
{
 Fractie rez;
 rez.a = a * obj.a;
 rez.b = b * obj.b;
 rez = rez.simplif();
 return rez;
}
Fractie Fractie::imp(Fractie obj)
{
 Fractie rez;
 rez.a = a * obj.b;
 rez.b = b * obj.a;
 rez = rez.simplif();
 return rez;
}

//main
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
const int dim = 3;
const int dim_char = 7;
#include Fractie.h

int main()
{
 Fractie obj1, obj2, * p, rez;
 int x;
 p = &obj1;
 cout << "\nFractie unu: ";
 cout << "\nNumitorul: ";
 cin >> x;
 p->setA(x);
 cout << "\nNumaratorul: ";
 cin >> x;
 p->setB(x);
 p = &obj2;
 cout << "\nFractie doi: ";
 cout << "\nNumitorul: ";
 cin >> x;
 p->setA(x);
 cout << "\nNumaratorul: ";
 cin >> x;
 p->setB(x);
 rez = obj1.simplif();
 cout << "\n" << obj1.getA() << " / " << obj1.getB() << " * " << rez.getA() << " / " << rez.getB();
 rez = obj2.simplif();
 cout << "\n" << obj2.getA() << " / " << obj2.getB() << " * " << rez.getA() << " / " << rez.getB();
 rez = obj1.adunare(obj2);
 cout << "\n" << "Suma: " << rez.getA() << " / " << rez.getB();
 rez = obj1.scadere(obj2);
 cout << "\n" << "Diferenta: " << rez.getA() << " / " << rez.getB();
 rez = obj1.inm(obj2);
```

```
 cout << "\n" << "Inmultire: " << rez.getA() << " / " << rez.getB();
 rez = obj1.adunare(obj2);
 cout << "\n" << "Impartire: " << rez.getA() << " / " << rez.getB();
}
```

/*Plesa Diana lab08
1. Construiţi o aplicaţie în care clasa OraCurenta are ca atribute private ora, minutele şi secundele şi metode publice de tip set/get pentru atributele clasei. Adaugaţi o functie friend clasei prin care să se poată copia conţinutul unui obiect OraCurenta dat ca si parametru, într-un alt obiect instanţă a aceleiaşi clase care va fi returnat de functie, ora fiind insa modificata la Greenwich Mean Time. Utilizati timpul curent al calculatorului.*/

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
class OraCurenta
{
 int ora, min, sec;
public:
 void setOra(int x)
 {
  ora = x;
 }
 int getOra()
 {
  return ora;
 }
 void setMin(int x)
 {
  min = x;
 }
 int getMin()
 {
  return min;
 }
 void setSec(int x)
 {
  sec = x;
 }
 int getSec()
 {
  return sec;
 }
 friend OraCurenta copy_ora(OraCurenta&);
};

OraCurenta copy_ora(OraCurenta& obj)
{
    OraCurenta obj1;
 if ((obj.ora - 3) >= 0)
  obj1.ora = obj.ora - 3;
 else
  obj1.ora = 24 + (obj.ora - 3);
 obj1.min = obj.min;
 obj1.sec = obj.sec;
 return obj1;
}
int main()
{
 OraCurenta ob1, ob2;
```

```
 time_t curr_timp;
 curr_timp = time(NULL);
 tm* local_tm = localtime(&curr_timp);
 ob1.setOra(local_tm->tm_hour);
 ob1.setMin(local_tm->tm_min);
 ob1.setSec(local_tm->tm_sec);
 cout << "Ora locala este: " << ob1.getOra() << ":" << ob1.getMin() << ":" << ob1.getSec();
 ob2 = copy_ora(ob1);
 cout<<"\nGMT time: "<<ob2.getOra()<< ":" << ob2.getMin() << ":" << ob2.getSec();
}
```

/*Plesa Diana lab08
2. Scrieţi o aplicaţie C/C++ în care clasa Calculator are un atribut privat memorie_RAM (int) şi o funcție prietenă tehnician_service( ) care permite modificarea valorii acestui atribut. Functia friend va fi membra intr-o alta clasa, Placa_de_baza care are o componentă denumire_procesor (sir de caractere). Scrieţi codul necesar care permite funcției prietene tehnician_service( ) să modifice (schimbe) valoarea variabilei denumire_procesor si memorie _RAM.*/

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
class Placa_de_baza {
 char denumire_procesor[10];
public:
 Placa_de_baza() {
  strcpy(denumire_procesor, "AMD");
 }
 void D_pr(char* t) {
  strcpy(denumire_procesor, t);
 }
 void g_den() {
  cout << "\nDenumire = " << denumire_procesor;
 }
};
class Calculator {
 int mram;
 Placa_de_baza o;
public:
 Calculator() { mram = 4; }
 void gram() {
  cout << "\nRam = " << mram;
 }
 friend Calculator tehnician_service(Calculator, Placa_de_baza);
};
Calculator tehnician_service(Calculator c, Placa_de_baza o) {
 Calculator n_r;
 c.mram = 16;
 char t[10];
 cout << "\nProcesor nume: ";
 cin >> t;
 o.D_pr(t);
 cout << "\nValoare ram = " << c.mram;
 cin.get();
```

```
 o.g_den();
 n_r.mram = c.mram;
 return n_r;
}
int main() {
 Calculator oc, n_oc;
 Placa_de_baza mb;
 oc.gram();
 mb.g_den();
 n_oc = tehnician_service(oc, mb);
 cout << "\nRam nou: ";
 n_oc.gram();
}
```

/*Plesa Diana lab08
3. Definţi o clasă numită Repository care are două variabile private de tip întreg. Clasa mai conţine un constructor explicit vid si unul cu 2 parametri şi o metodă accesor care afişează valorile variabilelor din clasă. Scrieţi o clasă numită Mathemathics, friend cu prima clasă, care implementează operaţiile aritmetice elementare (+, -, *, /) asupra variabilelor din prima clasa. Fiecare metoda primeşte ca parametru un obiect al clasei Repository.*/

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

class Mathemathics;
class Repository
{
 int x;
 int y;
public:
 Repository() {};
 Repository(int a, int b)
 {
  x = a;
  y = b;
 }
 int getX()
 {
  return x;
 }
 int getY()
 {
  return y;
 }
 void display();
 friend Mathemathics;
}; void Repository::display()
{
 cout << "\nNumerele sunt: " << getX() << " si " << getY();
}
class Mathemathics
{
public:
 Mathemathics() {};
```

```
 int add(Repository);
 int substract(Repository);
 int multiply(Repository);
 float divide(Repository);
};
int Mathemathics::add(Repository ob)
{
 return ob.x + ob.y;
}
int Mathemathics::substract(Repository ob)
{
 return ob.x - ob.y;
}
int Mathemathics::multiply(Repository ob)
{
 return ob.x * ob.y;
}
float Mathemathics::divide(Repository ob)
{
 if (ob.y != 0)
  return (float)ob.x / ob.y;
 else
 {
  cout << "\nNu poate fi impartit la 0!";
  return 0;
 }
}
int main() {
 int a, b;
 Mathemathics m;
 cout << "Primul numar este: ";
 cin >> a;
 cout << "Al doilea numar este: ";
 cin >> b;
 Repository ob(a, b);
 ob.display();
 cout << "\nSuma numerelor: " << m.add(ob);
 cout << "\nDiferenta numerelor este : " << m.substract(ob);
 cout << "\nProdusul numrelor este: " << m.multiply(ob);
 if (m.divide(ob) != 0)
  cout << "\nImpartirea numerelor este: " << m.divide(ob) << "\n";
}
```

/*Plesa Diana lab08
5. Rezolvaţi problema 4 în cazul în care variabila statică este de tip private. Definiţi o metodă accesor care returnează valoarea contorului.*/

```
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
using namespace std;
#define MAX 20
class Object
{
```

```
private:
 int val;
 static int var_static;
public:
 Object(int v)
 {
  val = v;
  var_static++;
 }
 int get_val()
 {
  return val;
 }
 int get_var_static()
 {
  return var_static;
 }
};
int Object::var_static;
int main()
{
 int valoare;
 cout << "valoare ="; cin >> valoare;
 Object ob1(valoare);
 cout << "\nvaloare ="; cin >> valoare;
 Object ob2(valoare);
 cout << "\nvaloare ="; cin >> valoare;
 cout << "\nNumarul de obiecte create: ";
 Object ob3(valoare);
 cout << ob3.get_var_static();
}
```

```
/*Plesa Diana lab08
6. Scrieţi o aplicaţie C/C++ în care să implementați clasa Punct cu atributele private x si y. Implementați o funcție
friend care să calculeze aria și perimetrul a doua forme geometrice definite de două puncte, considerând că
aceasta are doua puncte ca si parametrii P0(x0,y0) și P1 (x1, y1). Adăugați funcției un parametru întreg figura
prin care să indicați cele doua figuri geometrice ce sunt definite de punctele (x0, y0) si (x1, y1). Astfel, pentru
un cerc, figura=1, coordonatele (x0, y0) și (x1, y1) vor reprezenta două puncte complementare pe cerc
(diametrul). Dacă este triunghi dreptunghic, punctele definesc ipotenuza iar figura va fi =2; Celelalte laturi ale
triunghiului se vor determina pornind de la cele doua puncte. Coordonatele punctelor si apoi selecția figurii
geometrice se va realiza introducând de la tastatura parametrii.*/

#include <iostream>
using namespace std;
#define pi 3.14
class Point
{
private:
 int x, y;
public:
 friend float area(Point p0, Point p1, int s);
 void setPoints(int x1, int y1)
 {
```

```
  x = x1;
  y = y1;
 }
 int getX()
 {
  return x;
 }
 int getY()
 {
  return y;
 }
};
float area(Point p0, Point p1, int s)
{
 float nr = 0;
 if (s == 1) {
  float radius;
  radius = sqrt((p1.x - p0.x) * (p1.x - p0.x) + (p1.y - p0.y) * (p1.y -
   p0.y)); radius /= 2;
  nr = pi * pi * radius;
 }
 if (s == 2) {
  float hyp, cat1, cat2;
  cat1 = sqrt((p0.x) * (p0.x) + (p0.y) * (p0.y));
  cat2 = sqrt((p1.x) * (p1.x) + (p1.y) * (p1.y));
  hyp = sqrt((p1.x - p0.x) * (p1.x - p0.x) + (p1.y - p0.y) * (p1.y - p0.y));
  nr = (cat1 * cat2) / hyp;
 }
 return nr;
}
int main() {
 Point p0, p1;
 cout << "Introduceti coordonatele!\n";
 int p0x, p0y, p1x, p1y;
 cout << "p0x: "; cin >> p0x;
 cout << "p0y: "; cin >> p0y;
 cout << "p1x: "; cin >> p1x;
 cout << "p1y: "; cin >> p1y;
 p0.setPoints(p0x, p0y);
 p1.setPoints(p1x, p1y);
 cout << "\n\nCodurile pentru forme: \n1 este pentru cerc \n2 este pentru triunghi dreptunghic\n\nIntroduceti numaru
l: ";
 int s;
 cin >> s;
 if (s == 1) cout << "Este cerc!" << endl;
 else cout << "Este triunghi derptunghic!";
 area(p0, p1, s);
 cout << "\nAria este: " << area(p0, p1, s);
}
```

/*Plesa Diana lab08
7. La un chioșc se vând ziare, reviste și carți. Fiecare publicație are un nume, o editură, un număr de pagini, un număr de exemplare per publicație și un preț fără TVA. Scrieți clasa care modelează publicațiile. Adăugați un

membru static valoare_tva (procent) și o metodă statică pentru modificarea valorii TVA-ului. Să se calculeze suma totală cu TVA pe fiecare tip de publicație (ziare, reviste și carți) și prețul mediu pe pagina la fiecare publicație în parte. Modificați TVA-ul și refaceți calculele. Afișați editurile ordonate în funcție de încasări*/

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
static int VAT_value = 20;
class Publication
{
private:char* name;
    char* editorialHouse;
    int nrPages;
    int nrCopies;
    int price;
public:
 Publication()
 {
  name = new char[50];
  editorialHouse = new char[50];
 }
 void setName(char* nume)
 {
  strcpy(name, nume);
 }
 void setEditorialHouse(char* nume)
 {
  strcpy(editorialHouse, nume);
 }
 void setNrPages(int pagini)
 {
  nrPages = pagini;
 }
 void setNrCopies(int copii)
 {
  nrCopies = copii;
 }
 void setPrice(int pret)
 {
  price = pret;
 }
 int getPrice()
 {
  return price;
 }
 int getNrCopies()
 {
  return nrCopies;
 }
 int getNrPages()
 {
  return nrPages;
 }
 char* getName()
```

```
 {
  return name;
 }char* getEdit()
 {
  return editorialHouse;
 }
 ~Publication()
 {
  delete[]name;
  delete[]editorialHouse;
 }
};
void display(float* income, int n, Publication* kiosk);
void readNSet(float* income, int n, Publication* kiosk);
void calculateIncome(float* income, int n, Publication* kiosk);
int main() {
 int n;
 cout << "Introduceti numarul de publicatii: ";
 cin >> n;
 Publication* kiosk = new Publication[n];
 float* income = new float[n];
 readNSet(income, n, kiosk);
 calculateIncome(income, n, kiosk);
 display(income, n, kiosk);
 delete[]kiosk;
 delete[]income;
}
void display(float* income, int n, Publication* kiosk) {
 cout << "\nVeniturile pentru ficare publicatie: \n";
 for (int i = 0; i < n; i++) {
  cout << "\n\nNume: " << kiosk[i].getName() << endl;
  cout << "Editura: " << kiosk[i].getEdit() << endl;
  cout << "Venit total: " << income[i] << endl;
  cout << "Pretul mediu pe pagina: " << income[i] / kiosk[i].getNrPages();
 }
}
void readNSet(float* income, int n, Publication* kiosk) {
 char* name, * editorialHouse;
 int nrCopies, price, nrPages;
 for (int i = 0; i < n; i++) {
  name = new char[50];
  editorialHouse = new char[50];
  cout << "\nPublicatia " << i + 1 << endl;
  cout << "Nume: ";
  cin >> name;
  kiosk[i].setName(name);
  cout << "Numele editurii: ";
  cin >> editorialHouse;
  kiosk[i].setEditorialHouse(editorialHouse); cout << "Numarul de pagini: ";
  cin >> nrPages;
  kiosk[i].setNrPages(nrPages);
  cout << "Numarul de copii: ";
  cin >> nrCopies;
  kiosk[i].setNrCopies(nrCopies);
```

```
  cout << "Pret per copie: ";
  cin >> price;
  kiosk[i].setPrice(price);
 }
}
void calculateIncome(float* income, int n, Publication* kiosk) {
 for (int i = 0; i < n; i++) {
  income[i] = kiosk[i].getPrice() * kiosk[i].getNrCopies();
  income[i] += (VAT_value * income[i]) / 100;
 }
}
```

```
/*Plesa Diana lab09
2. Pornind de la exemplul 5 verificati/implementati următoarele cerințe:
a. citirea/scrierea unei matrici, unde dimensiunile sunt preluate de la tastatură
b. testați toți operatorii supraîncărcați. Implementati variante in care se folosesc metode membre la
supraincarcare.
c. afișați elementele de pe diagonala principală și secundară*/

#include<iostream>
using namespace std;
const int linii = 2;
const int coloane = 3;
class Matrix {
 int rows;
 int cols;
 int* elems;
public:
 Matrix();
 Matrix(int rows, int cols);
 Matrix(const Matrix&);
 ~Matrix(void) { delete elems; }
 int& operator () (int row, int col);
 Matrix& operator=(const Matrix&);
 friend Matrix operator+(Matrix&, Matrix&);
 friend Matrix operator-(Matrix&, Matrix&);
 friend Matrix operator*(Matrix&, Matrix&);
 int getRows(void) { return rows; }
 int getCols(void) { return cols; }
 void init(int r, int c);
 void citire();
 void afisare();
};//Matrix
Matrix::Matrix() : rows(linii), cols(coloane)
{
 elems = new int[rows * cols];
}
Matrix::Matrix(int r, int c) : rows(r), cols(c)
{
 elems = new int[rows * cols];
}
Matrix::Matrix(const Matrix& m) : rows(m.rows), cols(m.cols)
{
 int n = m.rows * m.cols;
 elems = new int[n];
 for (int i = 0; i < n; i++)
  elems[i] = m.elems[i];
}
void Matrix::init(int r, int c) {
 rows = r;
 cols = c;
 elems = new int[rows * cols];
}
int& Matrix:: operator()(int row, int col)
{
 return elems[row * cols + col];
```

```
}
Matrix& Matrix:: operator=(const Matrix& m) {
 if (this != &m)
 {
  int n = m.rows * m.cols;
  for (int i = 0; i < n; i++)
   elems[i] = m.elems[i];
 }
 return *this;
}
Matrix operator+(Matrix& p, Matrix& q) {
 Matrix m(p.rows, p.cols);
 for (int r = 0; r < p.rows; ++r)
  for (int c = 0; c < p.cols; ++c)
   m(r, c) = p(r, c) + q(r, c);
 return m;
}
Matrix operator-(Matrix& p, Matrix& q) {
 Matrix m(p.rows, p.cols);
 for (int r = 0; r < p.rows; ++r)
  for (int c = 0; c < p.cols; ++c)
   m(r, c) = p(r, c) - q(r, c);
 return m;
}
//op
Matrix operator*(Matrix& p, Matrix& q) {
Matrix m(p.rows, q.cols);
for (int r = 0; r < p.rows; ++r)
 for (int c = 0; c < q.cols; ++c) {
  m(r, c) = 0;
  for (int i = 0; i < p.cols; ++i)
   m(r, c) += p(r, i) * q(i, c);
 }
return m;
}//op*
void Matrix::citire() {
 for (int i = 0; i < rows; i++)
  for (int j = 0; j < cols; j++) {
   cout << "Dati elem. [" << i << "][" << j << "] ";
   cin >> elems[cols * i + j];
  }
}//citire
void Matrix::afisare() {
 for (int i = 0; i < rows; i++)
 {
  for (int j = 0; j < cols; j++)
   cout << elems[cols * i + j] << "\t";
  cout << endl;
 }
}//afisare
//main()
int main() {
 int i, j;
 Matrix m(linii, coloane);
```

```
cout << endl << "Supraincarcarea operatorului() pentru atribuirea unei valori pentru fiecare element din matrice : \n
";
 for (int i = 0; i < linii; i++)
  for (int j = 0; j < coloane; j++)
   m(i, j) = i + (j + 1) * 10;
for (i = 0; i < linii; i++)
{
 for (j = 0; j < coloane; j++)
  cout << m(i, j) << "\t";
 cout << endl;
}
int l, c;
cout << "Verificarea supraincarcarii operatorului () pentru un element de pe o pozitie citita de la tastatura" << endl;
 cout << "Dati numarului liniei (>=1): ";
cin >> l;
cout << "Dati numarului coloanei (>=1): ";
cin >> c;
if ((l >= 1 && l <= m.getRows()) && (c >= 1 && c <= m.getCols()))
 cout << "Elementul m[" << l << "," << c << "]=" << m(l - 1, c - 1) << endl;
else
 cout << "Indici eronati!" << endl;
cout << endl << "Utilizare constructor de copiere:" << endl;
if (m.getRows() > 0 && m.getCols() > 0) {
 Matrix mcopy = m;
 cout << "Matricea \"mcopy\" este:" << endl;
 mcopy.afisare();
}
else cout << "Dimensiuni invalide pentru matricea care se copiaza la instantiere!"
 << endl;
cout << endl << "Instantiem un nou obiect matrice \"n\" ";
Matrix n(linii, coloane);
cout << endl << "Dati matricea:" << endl;
n.citire();
cout << endl << "Matricea \"n\" este:" << endl;
n.afisare();
cout << endl << "Supraincarcarea operatorului =, copiere matrice \"m\" in matrice\"n\"" << endl;
if (m.getRows() == n.getRows() && m.getCols() == n.getCols()) {
 n = m;
 //n.afisare();
 for (i = 0; i < linii; i++) {
  for (j = 0; j < coloane; j++)
   cout << n(i, j) << "\t";//afisare prin Supraincarcarea operatorului()
   cout << endl;
 }///end for
}
else
 cout << "Matricile nu au aceleasi dimensiuni, deci nu pot fi copiate" <<
 endl;
cout << endl << "Instantiem un nou obiect matrice \"m1\" ";
Matrix m1(linii, coloane);
cout << endl << "Dati matricea:" << endl;
m1.citire();
cout << endl << "Matricea \"m1\" este:" << endl;
m1.afisare();
```

```
 Matrix m2(linii, coloane);
 cout << endl << "Supraincarcarea operatorului +" << endl;
 if (m.getRows() == m1.getRows() && m.getCols() == m1.getCols()) {
  m2 = m + m1;
  cout << endl << "Matricea rezultata din suma matricilor m+m1 este: " <<
   endl;
 }
 m2.afisare();
 cout << endl << "Supraincarcarea operatorului - " << endl;
 if (m.getRows() == m1.getRows() && m.getCols() == m1.getCols()) {
  m2 = m - m1;
  cout << endl << "Matricea rezultata din diferenta matricilor m-m1 este: "
   << endl;
 }
 m2.afisare();
 /*matricea m are 2 linii si 3 coloane deci pentru a fi posibil produsul m3 trebuie
 sa aiba 3 linii si 2 coloane*/
 cout << endl << "Dati matricea pentru produs \"m3\' (matricea trebuie sa aiba numarul de linii egal cu numarul de c
oloane al matricii \"m\")" << endl;
 cout << "Numar de linii: ";
 cin >> l;
 cout << "Numar coloane: ";
 cin >> c;
 Matrix m3;
 if (l > 0 && c > 0) m3.init(l, c);
 else cout << endl << "Dimensiuni negative (gresite)! Se vor folosi pentru instantiere valorile initiale implicite(2 linii
, 3 coloane)" << endl;
  m3.citire();
 cout << endl << "Matricea \"m3\" este:" << endl;
 m3.afisare();
 cout << endl << "Supraincarcarea operatorului * ";
 // pentru inmultire m*m3 nr. de coloane al matricii m trebuie sa fie egal cu numarul de randuri al matricii m3
  if (m.getCols() == m3.getRows())
  {
   Matrix m4(m.getRows(), m3.getCols());
   m4 = m * m3;
   cout << endl << "Matricea rezultata prin inmultirea matricilor m*m3 este: "
    << endl;
   m4.afisare();
  }
  else
   cout << endl << "Matricile nu pot fi inmultite - numarul de linii nu e egal cu numarul de coloane";
   return 0;
}
```

/*Plesa Diana lab09
3. Să se supraîncarce operatorul [ ] astfel încât, folosit fiind asupra unor obiecte din clasa Departament, ce contine un tablou de obiecte de tip Angajat (clasa Angajat contine variabilele nume (şir de caractere) şi salariu (double)), să returneze toată informaţia legată de angajatul al cărui număr de ordine este trimis ca parametru.*/

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

```
class Departament;
class Angajat
{
private:
 char nume[20];
 float salariu;
public:
 Angajat(float is = 0.0)
 {
  strcpy(nume, "unknown");
  salariu = is;
 }
 friend class Departament;
 void set_angajat(float is, char* inume)
 {
  strcpy(nume, inume);
  salariu = is;
 }
};
class Departament
{
 int nr_angajati;
 Angajat angajati[20];
public:
 Departament(int inr = 0)
 {
  nr_angajati = inr;
 }
 void set_departament()
 {
  char inume[20];
  float is;
  for (int i = 0; i < nr_angajati; i++)
  {
   cout << "\nIntroduceti datele pentru angajatul " << i + 1 << ":\n";
   cout << "\nNumele si salariul: ";
   cin >> inume;
   cin >> is;
   angajati[i].set_angajat(is, inume);
  }
 }
 void operator [](int i)
 {
  cout << "\nNumele: " << angajati[i - 1].nume;
  cout << "\nSalariul: " << angajati[i - 1].salariu << " lei" << endl;
 }
};


int main()
{
 int nr;
 cout << "Introduceti numarul de angajati: ";
```

```
 cin >> nr;
 Departament dep(nr);
 dep.set_departament();
 cout << "\nIntroduceti numarul de ordine pentru vizualizarea datelor: ";
 cin >> nr;
 dep[nr];
}
```

/*Plesa Diana lab09
4. Să se supraîncarce operatorii new şi delete într-una din clasele cu care s-a lucrat anterior, în vederea alocarii şi eliberarii de memorie pentru un obiect din clasa respectivă.*/

```
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
#include <conio.h>
#include <malloc.h>
using namespace std;
class Complex
{
 double re, im;
public:
 Complex(int, int);
 void Afisare(void);
 void* operator new(size_t dim);
 void operator delete(void* p);
};
Complex::Complex(int r = 1, int i = 1)
{
 re = r;
 im = i;
}
void Complex::Afisare(void)
{
 cout << "\nPartea reala: " << re;
 cout << "\nPartea imaginara: " << im << endl;
}
void* Complex::operator new(size_t dim)
{
 void* p = malloc(dim);
 if (p == 0)
  cout << "\nAlocare nereusita!!!";
 return p;
}
void Complex::operator delete(void* p)
{
 if (p)
  free(p);
}
int main()
{
 Complex* p;
 p = new Complex;
 p->Afisare();
```

```
 delete p;
 Complex* p1;
 p1 = new Complex(4, 6);
 p1->Afisare();
 delete p1;
}
```

/*Plesa Diana lab09
5. Să se scrie programul care considera o clasa MyClass cu trei atribute de tip int. Clasa considera pe baza mecanismului de supraincarcare metode publice int myFunction(…), care în funcţie de numărul de parametri primiţi, returnează fie valoarea primită (1 parametru), fie produsul variabilelor de intrare (0-toti, 2, 3 parametrii). Instantiati un obiect din clasa in main(), setati atributele cu metode setter adecvate din clasa si afisati valorile la apelurile metodelor.*/

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

class MyClass
{
 int x, y, z;
public:
 void setX(int a)
 {
  x = a;
 }
 void setY(int a)
 {
  y = a;
 }
 void setZ(int a)
 {
  z = a;
 }
 int getX(void)
 {
  return x;
 }
 int getY(void)
 {
  return y;
 }
 int getZ(void)
 {
  return z;
 }
 int MyFunction()
 {
  return x * y * z;
 }
 int MyFunction(int a)
 {
  return a;
```

```
 }
 int MyFunction(int a, int b)
 {
  return a * b;
 }
 int MyFunction(int a, int b, int c)
 {
  return a * b * c;
 }
};

int main()
{
 MyClass obj;
 int aux;
 cout << "\nIntroduceti x: ";
 cin >> aux;
 obj.setX(aux);
 cout << "\nIntroduceti y: ";
 cin >> aux;
 obj.setY(aux);
 cout << "\nIntroduceti z: ";
 cin >> aux;
 obj.setZ(aux);
 cout << "\nApel fara parametri: " << obj.MyFunction();
 cout << "\nApel cu un parametru: " << obj.MyFunction(obj.getX());
 cout << "\nApel 2 parametri: " << obj.MyFunction(obj.getX(), obj.getY());
 cout << "\nApel cu 3 parametri: " << obj.MyFunction(obj.getX(), obj.getY(), obj.getZ());
 return 0;
}


/*Plesa Diana lab09
6. Să se scrie programul care utilizează o clasă numită Calculator şi care are în componenţa sa metodele publice
supraincarcate:
- int calcul(int x) care returnează pătratul valorii primite;
- int calcul(int x, int y) care returnează produsul celor două valori primite;
- double calcul(int x, int y, int z) care returnează rezultatul înlocuirii în formula f(x,y,z) = (x-y)(x+z)/2. a
valorilor primite;*/

#include<iostream>
#include <conio.h>
using namespace std;
class Calculator
{
public:
 int calcul(int x)
 {
  return (x * x);
 }
 int calcul(int x, int y)
 {
  return (x * y);
 }
```

```
 float calcul(int x, int y, int z)
 {
  return ((x - y) * (x + z) / 2);
 }
};

int main(int argc, char* argv[])
{
 Calculator ob1;
 if (argc <= 1)
 {
  cout << "\nEroare! Nu exista date introduse!";
  exit(0);
 }
 switch (argc)
 {
 case 2:
  cout << "\nPatratul este:" << ob1.calcul(atoi(argv[1])) << endl;
  break;
 case 3:
  cout << "\nPatratul este:" << ob1.calcul(atoi(argv[1])) << endl;
  cout << "\nProdusul este:" << ob1.calcul(atoi(argv[1]), atoi(argv[2])) <<
   endl;
  break;
 case 4:
  cout << "\nPatratul este:" << ob1.calcul(atoi(argv[1])) << endl;
  cout << "\nProdusul este:" << ob1.calcul(atoi(argv[1]), atoi(argv[2])) <<
   endl;
  cout << "\nRezultatul functiei f(x,y,z) este:" << ob1.calcul(atoi(argv[1]),
   atoi(argv[2]), atoi(argv[3])) << endl;
  break;
 }
}
```

/*Plesa Diana lab10
1. Implementaţi programul prezentat în exemplul 3 şi examinaţi eventualele erori date la compilare daca exista prin eliminarea comentariilor. Modificaţi programul astfel încât să se poată accesa din funcţia main( ), prin intermediul obiectului obiect_derivat, şi metodele aduna( ) şi scade( ) din clasa de baza pastrand mostenirea de tip private.*/

```
//Baza_deriv.h
class Baza {
protected: int a, b;
public:
 Baza() { a = 1, b = 1; }
 void setA(int a) {
  this->a = a;
 }
 void setB(int b) {
  this->b = b;
 }
 int getA() {
  return a;
 }
 int getB() {
  return b;
 }
 int aduna() {
  return a + b;
 }
 int scade() {
  return a - b;
 }
};
class Derivata : private Baza
{
public:
 int inmulteste() {
  return a * b;
 }
 int scade()
 {
  return scade();
 }
 int aduna()
 {
  return aduna();
 }
};

#include <iostream>
using namespace std;
#include "Baza_deriv.h"

int main()
{
 Baza obiect_baza;
 cout << "\nAfis din baza (val. initiale): " << obiect_baza.getA() << " " << obiect_baza.getB() << '\n';
```

```
 cout << "\nSuma este (cu val. initiale, baza) = " << obiect_baza.aduna();
 cout << "\nDiferenta este (cu val. initiale, baza) = " << obiect_baza.scade() << "\n";

 obiect_baza.setA(2);
 obiect_baza.setB(3);
 cout << "\nAfis din baza (modificat): " << obiect_baza.getA() << " " << obiect_baza.getB() << '\n';
 cout << "\nSuma/Diferenta dupa setare= " << obiect_baza.aduna() << "/" << obiect_baza.scade() << '\n';

 Derivata obiect_derivat;
 cout << "\nProdusul este (din derivat cu val. initiale) = " << obiect_derivat.inmulteste() << '\n';
 cout << "\nSuma este (din derivat cu val. initiale, din baza) = " << obiect_derivat.aduna( ) << '\n';
 cout << "\nDiferenta este (din derivat cu val. initiale, din baza) = " << obiect_derivat.scade();
 return 0;
}
```

/*Plesa Diana lab10
2. Folosind modelul claselor de la mostenirea publica, implementaţi două clase, astfel:
- clasa de bază contine metode pentru:
-codarea unui şir de caractere (printr-un algoritm oarecare)
=> public;
-afişarea şirului original şi a celui rezultat din transformare
= > public;
- clasa derivata contine o metoda pentru:
-scrirea rezultatului codării într-un fişier, la sfârşitul acestuia.
Fiecare inregistrare are forma: nr_inregistrare: şir_codat;
Accesul la metodele ambelor clase se face prin intermediul unui obiect rezultat prin instanţierea clasei derivate. Programul care foloseşte clasele citeşte un şir de caractere de la tastatură şi apoi, în funcţie de opţiunea utilizatorului, afişează rezultatul codării sau îl scrie în fişier.*/

```
//Baza.h
class Baza {
 char p[dim_str];
public:
 void setP(char buf[])
 {
  strcpy(p, buf);
 }
 char* code()
 {
  char* c = new char[dim_str];
  for (int i = 0; i < strlen(p); i++)
   c[i] = p[i] xor 10;
  c[strlen(p)] = '\0';
  return c;
 }
 void display()
 {
  cout << "\nSirul intial: " << p;
  char c[dim_str];
  strcpy(c, code());
  cout << "\nSirul codat: " << c;
 }
```

```
};

//Derivata.h
class Derivata :public Baza
{
 static int n;
public:
 void fisier()
 {
  n++;
  FILE* f;
  if ((f = fopen("Text.txt", "a+")) == NULL)
  {
   cout << "\nEroare!";
   exit(1);
  }
  char c[dim_str];
  strcpy(c, code());
  fprintf(f, "\nNr_inregistrare: %d: ", n);
  fprintf(f, "%s", c);
  fclose(f);
 }
};

int Derivata::n = 0;

#include <iostream>
using namespace std;
#include Baza.h
#include Derivata.h

int main()
{
 char sir[dim_str], o;
 int op;
 do
 {
  Derivata obj;
  cout << "\nIntroduceti un sir de caractere fara spatiu: ";
  cin >> sir;
  obj.setP(sir);
  cout << "\nApasati tasta 1 pentru a afisa sirul codat \n Apasati tasta 2 pentru a scrie in fisier sirul codat";
  cin >> op;
  switch (op)
  {
  case 1:
  {
   obj.display();
   break;
  }
  case 2:
  {
   obj.fisier();
   break;
```

```
  }
  default: cout << "\nAti introduse o optiune invalida!";
  }
  cout << "\nApasati y/Y pentru a continua";
  cin >> o;
 } while (o == ('y') or o == ('Y'));
 return 0;
}
```

/*Plesa Diana lab10
3. Să se implementeze o clasă de bază cu două atribute protected de tip întreg care conţine o metoda mutator pentru fiecare atribut al clasei, parametri metodelor fiind preluati in main() de la tastatura şi metode accesor pentru fiecare atribut care returneaza atributul specific. Să se scrie o a doua clasă, derivată din aceasta, care implementează operaţiile matematice elementare: +, -, *, / asupra atributelor din clasa de bază, rezultatele fiind returnate de metode. Să se scrie o a III-a clasă, derivată din cea de-a doua, care implementează în plus o metoda pentru extragerea rădăcinii pătrate dintr-un număr (mul, rezultat al operatiei * din prima clasa derivata) şi de ridicare la putere (atât baza (plus, rezultat al operatiei + din prima clasa derivata) cât şi puterea (minus, rezultat al operatiei - din prima clasa derivata) sunt trimişi ca parametri). Verificati apelul metodelor considerand obiecte la diferite ierarhii.*/

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

//Baza.h
class Baza {
protected:
 int x, y;
public:
 void setX(int a)
 {
  x = a;
 }
 void setY(int a)
 {
  y = a;
 }
 int getX()
 {
  return x;
 }
 int getY()
 {
  return y;
 }
};

//Derivata1.h
class Derivata1 :public Baza
{
public:
 int plus()
 {
```

```
 return x + y;
}
int minus()
{
 return x - y;
}
double mul()
{
 return x * y;
}
double div()
{
 if (y != 0)
  return (double)x / y;
 else
 {
  cout << "\nImpartirea la 0 nu este posibila!";
  return 0;
 }
}
};

//Derivata2.h
class Derivata2 :public Derivata1
{
public:
 double sqr(double x)
 {
  return sqrt(x);
 }
 double power(int a, double b)
 {
  return pow(b, a);
 }
};

int main()
{
 int x;
 Derivata1 obj1;
 Derivata2 obj2;
 cout << "\nx = ";
 cin >> x;
 obj1.setX(x);
 cout << "\ny = ";
 cin >> x;
 obj1.setY(x);
 cout << "\nDerivata 1: \n";
 cout << obj1.getX() << "+" << obj1.getY() << "=" << obj1.plus() << endl;
 cout << obj1.getX() << "-" << obj1.getY() << "=" << obj1.minus() << endl;
 cout << "\nx = ";
 cin >> x;
 obj2.setX(x);
 cout << "\ny = ";
```

```
 cin >> x;
 obj2.setY(x);
 cout << "\nDerivata 2: ";
 cout << "\nSqrt: " << obj2.sqr(obj2.mul());
 cout << "\nPower: " << obj2.power(obj2.minus(), obj2.plus());
 return 0;
}
```

/*Plesa Diana lab10
4. Definiţi o clasă numita Triangle care are 3 atribute protected pentru laturi si o metoda care calculează perimetrul unui triunghi ale cărui laturi sunt citite de la tastatura (folosite de un constructor adecvat) şi apoi o clasă derivata in mod public din Triangle, Triangle_extended, care în plus, calculează şi aria triunghiului. Folosind obiecte din cele doua clase apelati metodele specifice. Verificati inainte de instantiere posibilitatea definirii unui triunghi.*/

```
#include <iostream>
using namespace std;
class Triangle
{
protected:
 int a, b, c;
public:
 Triangle(int x = 0, int y = 0, int z = 0)
 {
  c = x;
  a = y;
  b = z;
 }
 void setC(int z)
 {
  c = z;
 }
 void setA(int x)
 {
  a = x;
 }
 void setB(int y)
 {
  b = y;
 }
 int getC()
 {
  return c;
 }
 int getA()
 {
  return a;
 }
 int getB()
 {
  return b;
 }
 int perimeter()
```

```
 {
  return c + a + b;
 }
};
class Triangle_extended : public Triangle
{
public:
 Triangle_extended(int x = 0, int y = 0, int z = 0)
 {
  a = x;
  c = z;
  b = y;
 }
 float Area()
 {
  float semi_perimeter = (float)(c + a + b) / 2;
  return sqrt(semi_perimeter * (semi_perimeter - c) * (semi_perimeter - a) * (semi_perimeter - b));
 }
};
int main()
{
 cout << "Introduceti laturile triunghiului: ";
 int a, b, c;
 cin >> a >> b >> c;
 if ((a + b >= c || a + c >= b || c + b >= a) && (a > 0 && b > 0 && c > 0))
 {
  Triangle objective;
  objective.setC(c);
  objective.setA(a);
  objective.setB(b);
  cout << "\nPerimetrul este: " << objective.perimeter();
  Triangle_extended objective_extended(c, a, b);
  cout << "\nAria este: " << objective_extended.Area();
 }
 else
  cout << "\nNu se formeaza un triunghi. Introduceti alte laturi!";
}
```

/*Plesa Diana lab10
6. Definiţi o clasă numită Forme care defineste o figura geometrica cu un nume ca si atribut de tip pointer la un sir de caractere. Clasa va contine un constructor fara parametrii, unul cu parametrii, copy constructor si se va supraincarca operatorul de asignare. Clasa va avea si o metoda getter si un destructor. Derivati in mod public o clasa Cerc care adauga un atribut de tip int pentru raza si constructori adecvati considerand si atributele (nume, raza) si o metoda getter pentru raza si alte metode care calculeaza aria si perimetrul cercului de raza r, valoare introdusa in main( ) de la tastatura. Similar definiti o clasa Patrat si Dreptunghi care permit determinarea ariei si perimetrului obiectelor specifice. Instantiati obiecte din clasele derivate si afisati aria si perimetrul obiectelor. Datele specifice vor fi introduse de la tastatura. Definiti un obiect de tip Cerc cu parametrii care sa il copiati intr-un nou obiect la care sa ii afisati atributele. Definiti un obiect de tip Patrat cu parametrii si altul fara parametrii. Asignati celui fara parametrii obiectul instantiat cu parametrii si afisati atributele.*/

```
#define _CRT_SECURE_NO_WARNINGS
```

```
#include <iostream>
using namespace std;
#define DIM 50

class Forme
{
protected:
 char* name;
public:
 Forme()
 {
  name = new char[DIM];
  strcpy(name, "default");
 }
 void setName(char x[DIM])
 {
  strcpy(name, x);
 }
 char* getName()
 {
  return name;
 }
 ~Forme()
 {
  delete[] name;
 }
};
class Cerc: public Forme
{
protected:
 int radius;
public:
 Cerc(int r = 0)
 {
  name = new char[DIM];
  radius = r;
  strcpy(name, "default");
 }
 int getRadius()
 {
  return radius;
 }
 void setName(char x[DIM])
 {
  strcpy(name, x);
 }
 char* getName()
 {
  return name;
 }
 double area()
 {
  return 3.14 * radius * radius;
 }
```

```
 double perimeter()
 {
  return 2 * 3.14 * radius;
 }
 ~Cerc()
 {
  delete[] name;
 }
};
class Patrat : public Forme
{
protected:
 int lat;
public:
 Patrat(int x = 0)
 {
  name = new char[DIM];
  lat = x;
  strcpy(name, "default");
 }
 void setName(char x[DIM])
 {
  strcpy(name, x);
 }
 char* getName()
 {
  return name;
 }
 int getLat()
 {
  return lat;
 }
 int area()
 {
  return lat * lat;
 }
 int perimeter()
 {
  return 4 * lat;
 }
 ~Patrat()
 {
  delete[] name;
 }
};
class Dreptunghi : public Forme
{
protected:
 int lat1, lat2;
public:
 Dreptunghi(int x1 = 0, int x2 = 0)
 {
  name = new char[DIM];
  lat1 = x1;
```

```
 lat2 = x2;
 strcpy(name, "default");
}
void setName(char x[DIM])
{
 strcpy(name, x);
}
char* getName()
{
 return name;
}
int area()
{
 return lat1 * lat2;
}
int perimeter()
{
 return 2 * (lat1 + lat2);
}
int getLat1()
{
 return lat1;
}
int getLat2()
{
 return lat2;
}
~Dreptunghi()
{
 delete[] name;
}
};
int main()
{
 int radius, lat, lat1, lat2, i;
 char name[DIM];
 cout << "1- cerc";
 cout << "\n2- patrat";
 cout << "\n3- dreptunghi";
 cout << "\nIntroduceti forma dorita: "; cin >> i;
 if (i == 1)
 {
  cout << "\nIntroduceti raza: ";
  cin >> radius;
  cout << "Introduceti nume: ";
  cin >> name;
  Cerc C(radius);
  C.setName(name);
  cout << "Aria cercului: " << C.area();
  cout << "\nPerimetrul cercului: " << C.perimeter();
 }
 if (i == 2)
 {
  cout << "\n\nIntroduceti laturile patratului: ";
```

```
  cin >> lat;
  cout << "Introduceti nume: ";
  cin >> name;
  Patrat S(lat);
  S.setName(name);
  cout << "Aria patratului: " << S.area();
  cout << "\nPerimetrul patratului: " << S.perimeter();
 }
 if (i == 3)
 {
  cout << "\n\nIntroduceti laturile dreptunghiului: ";
  cin >> lat1 >> lat2;
  cout << "Introduceti nume: ";
  cin >> name;
  Dreptunghi R(lat1, lat2);
  R.setName(name);
  cout << "Aria dreotunghiului: " << R.area();
  cout << "\nPerimetrul dreptunghiului: " << R.perimeter();
 }
}
```

/*Plesa Diana lab10
7. Considerați o clasa de baza Cerc definita printr-un atribut protected raza, care are un constructor cu parametrii si o metoda care determina aria cercului. Considerați o alta clasa de baza Patrat cu un atribut protected latura similar clasei Cerc. Derivați un mod public clasa CercPatrat care are un constructor ce apelează constructorii claselor de baza si o metoda care verifica daca pătratul de latura l poate fi inclus in cercul de raza r. De asemenea clasa derivata determina si perimetrul celor doua figuri geometrice. Instanțiați un obiect din clasa derivata (datele introduse de la tastatura), determinați aria si perimetrul cercului si al pătratului. Afișați daca pătratul cu latura introdusa poate fi inclus in cercul de raza specificat.*/

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

class Cerc
{
protected:
 int radius;
public:
 Cerc(int r = 0)
 {
  radius = r;
 }
 void setRadius(int r)
 {
  radius = r;
 }
 int getRadius()
 {
  return radius;
 }
 float aria()
 {
```

```
  return 3.14 * radius * radius;
 }
};
class Patrat
{
protected:
 int lat;
public:
 Patrat(int l = 0)
 {
  lat = l;
 }
 float area()
 {
  return lat * lat;
 }
};
class CercPatrat : public Cerc, public Patrat
{
public:
 CercPatrat(int rad, int lat)
 {
  Cerc cerc(rad);
  Patrat patrat(lat);
 }
 int perimSquare(int lat)
 {
  return 4 * lat;
 }
 double perimCircle(int rad)
 {
  return 2 * 3.14 * rad;
 }
 int canFit(int lat, int radius)
 {
  int ok = 0;
  if (perimSquare(lat) < perimCircle(radius))
   ok = 1;
  return ok;
 }
};

int main()
{
 cout << "Introduceti laturile patratului: ";
 int lat, radius;
 cin >> lat;
 cout << "Introduceti raza cercului: ";
 cin >> radius;
 CercPatrat objective(radius, lat);
 cout << "\nPerimetrul patratului este: " << objective.perimSquare(lat);
 cout << "\nPerimetrul cercului este: " << objective.perimCircle(radius);
 if (objective.canFit(lat, radius) == 1) {
  cout << "\n\nPatratul cu latura introdusa poate fi inclus in cerc!";
```

```
 }
}
```

/*Plesa Diana lab10
8. Considerați clasa Fractie care are doua atribute întregi protected a si b pentru numărător si numitor, doua metode de tip set( ) respectiv get( ) pentru fiecare din atributele clasei. Declarați o metoda publica simplifica( ) care simplifica un obiect Fractie. Definiti un constructor explicit fara parametri care initializeaza a cu 0 si b cu 1, si un constructor explicit cu doi parametri care va putea fi apelat daca se verifica posibilitatea definirii unei fractii (b!=0). Supraîncărcați operatorii de adunare, scadere, inmultire si impartire (+,-,*,/) a fracțiilor folosind metode membre care si simplifica daca e cazul rezultatele obtinute, apeland metoda simplifica( ) din clasa. Definiți o clasa Fractie_ext derivata public din Fractie, care va avea un constructor cu parametrii (ce apelează constructorul din clasa de baza). Supraîncărcați operatorii de incrementare si decrementare prefixați care aduna/scade valoarea 1 la un obiect de tip Fractie_ext cu metode membre.
Instanțiați doua obiecte de tip Fractie fără parametrii. Setați atributele obiectelor cu date citite de la tastatura. Afișați atributele inițiale ale obiectelor si noile atribute definite. Efectuați operațiile implementate prin metodele membre, inițializând alte 4 obiecte cu rezultatele obținute. Simplificați si afișați rezultatele. Instanțiați doua obiecte de tip Fractie_ext cu date citite de la tastatura. Efectuați operațiile disponibile clasei, asignând rezultatele obținute la alte obiecte Fracti_ext. Simplificați si afișați rezultatele.*/

```
#include <iostream>
using namespace std;
class  Fractie
{
protected:
 int a, b;
public:
 Fractie()
 {
  a = 0;
  b = 1;
 }
 Fractie(int x, int y)
 {
  if (y != 0)
  {
   a = x;
   b = y;
  }
  else
   cout << "Operatia nu poate fi facuta. Introduceti alte numere!\n ";
 }
 void setA(int x)
 {
  a = x;
 }
 void setB(int y)
 {
  b = y;
 }
 int getA()
```

```
 {
  return a;
 }
 int getB()
 {
  return b;
 }
 void simplifica()
 {
  int i = 2;
  while (i <= a || i <= b)
  {
   if (a % i == 0 && b % i == 0)
   {
    a /= i;
    b /= i;
   }
   else i++;
  }
 }
 friend  Fractie operator+(Fractie, Fractie);
 friend  Fractie operator-(Fractie, Fractie);
 friend  Fractie operator*(Fractie, Fractie);
 friend  Fractie operator/(Fractie, Fractie);
};
Fractie operator+(Fractie object, Fractie object2)
{
 Fractie nr;
 if (object.b == object2.b)
 {
  nr.setA(object.a + object2.a);
  nr.setB(object.b);
 }
 else
 {
  nr.setA(object.a * object2.b + object.b * object2.a);
  nr.setB(object.b * object2.b);
 }
 return nr;
}
Fractie operator-(Fractie object, Fractie object2)
{
 Fractie nr;
 if (object.b == object2.b)
 {
  nr.setA(object.a - object2.a);
  nr.setB(object.b);
 }
 else
 {
  nr.setA(object.a * object2.b - object.b * object2.a);
  nr.setB(object.b * object2.b);
 }
 return nr;
```

```
}
Fractie operator*(Fractie object, Fractie object2)
{
 Fractie nr;
 nr.setA(object.a * object2.a);
 nr.setB(object.b * object2.b);
 return nr;
}
Fractie operator/(Fractie object, Fractie object2)
{
 Fractie nr;
 nr.setA(object.a * object2.b);
 nr.setB(object.b * object2.a);
 return nr;
}
class Fraction_ext : public  Fractie
{
public:
 Fraction_ext(int x, int y) : Fractie(x, y) {};
 Fraction_ext& operator++(int)
 {
  Fraction_ext object(a, b);
  object.a += object.b;
  return object;
 }
 Fraction_ext& operator--(int)
 {
  Fraction_ext object(a, b);
  object.a -= object.b;
  return object;
 }
};

int main()
{
 Fractie object, object2, object3;
 int x, y;
 cout << "Numerele initiale pentru prima fractie: " << object.getA() << " " << object.getB();
  cin >> x >> y;
 object.setA(x);
 object.setB(y);
 cout << "Numerele dupa set sunt: " << object.getA() << " " << object.getB();
 cout << "\n\nNumerele initiale pentru a doua fractie sunt: ";
 cin >> x >> y;
 object2.setA(x);
 object2.setB(y);
 cout << "\nNumerele dupa set sunt: " << object2.getA() << " " << object2.getB();
 object3 = object + object2;
 cout << "\n\nAdunare: " << object3.getA() << " " << object3.getB() << " ~ ";
 object3.simplifica(); cout << object3.getA() << " " << object3.getB() << endl;
 object3 = object - object2;
 cout << "Scadere: " << object3.getA() << " " << object3.getB() << " ~ ";
 object3.simplifica(); cout << object3.getA() << " " << object3.getB() << endl;
 object3 = object * object2;
```

```
 cout << "Inmultire: " << object3.getA() << " " << object3.getB() << " ~ ";
 object3.simplifica(); cout << object3.getA() << " " << object3.getB() << endl;
 object3 = object / object2;
 cout << "Impartire: " << object3.getA() << " " << object3.getB() << " ~ ";
 object3.simplifica(); cout << object3.getA() << " " << object3.getB() << endl;
 int a, b;
 cout << "\nAcum pentru clasa a doua! \n Va rugam sa introduceti 4 numere noi ca inainte";
 cout << "\n\nPentru prima fractie: ";
 cin >> a >> b;
 Fraction_ext f1(a, b);
 cout << "\nPentru a doua fractie: ";
 cin >> a >> b;
 Fraction_ext f2(a, b);
 cout << "\nInainte: " << f1.getA() << " " << f1.getB();
 cout << "\nDupa: " << f1.getA() << " " << f1.getB();
}
```

/*Plesa Diana lab13
1. Să se scrie un program care foloseşte metoda seekg( ) pentru poziţionare la mijlocul fişierului si apoi afişează conţinutul fişierului începând cu această poziţie. Numele fişierului se citeste din linia de comandă.*/

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

int main(int argc, char* argv[])
{
 if (argc != 2)
 {
  cout << "Introduceti numele fisierului!";
 }
 ifstream f;
 f.open(argv[1]);
 f.seekg(0, f.end);
 int length = f.tellg() / 2;
 f.seekg(length, f.beg);
 char* buffer = new char[length];
 f.read(buffer, length);
 cout.write(buffer, length);
 f.close();
 delete[]buffer;
}
```

/*Plesa Diana lab13
2. Scrieţi un program care utilizează metoda write( ) pentru a scrie într-un fişier şiruri de caractere. Afişaţi apoi conţinutul fişierului folosind metoda get( ). Numele fişierului se va citi de la tastatură.*/

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
 char name_file[100];
 cout << "\nIntroduceti numele fisierului: ";
 cin.getline(name_file, 100);
 strcat(name_file, ".txt");
 ofstream fout(name_file);
 char txt[100];
 cout << "\nIntroduceti un txt: ";
 cin.getline(txt, 100);
 fout.write(txt, strlen(txt));
 fout.close();
 ifstream fin(name_file);
 char aux;
 while (!fin.eof())
 {
  fin.get(aux);
```

```
  cout << aux;
 }
}
```

/*Plesa Diana lab13
3. Scrieţi o aplicaţie C++ care citeşte un fişier utilizând metoda read( ). Verificaţi starea sistemului după fiecare operaţie de citire. Numele fişierului se va citi din linia de comandă. Afişaţi pe ecran conţinutul fişierului. */

```
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc, char* argv[])
{
 if (argc != 2)
 {
  cout << "Introduceti numele fisierului!";
 }
 ifstream f;
 f.open(argv[1]);
 if (f)
 {
  f.seekg(0, f.end);
  int length = f.tellg();
  f.seekg(0, f.beg);
  char* buffer = new char[length];
  f.read(buffer, length);
  cout.write(buffer, length);
  delete[] buffer;
  f.close();
 }
}
```