```
================== BinSTree.h ================
#ifndef _BINSTREE
#define _BINSTREE

#include <stdlib.h>
#include <string.h>
#include <iostream.h>

class Nod {
            char* m_key;
            void* m_reference;
            Nod* left;
            Nod* right;

      public:
            Nod(char* key, void* reference) {
              m_key = new char [sizeof(char)*(strlen(key) + 1)];
              strcpy(m_key, key);
              m_reference = reference;

              left = NULL;
              right = NULL;
            }

            friend class BinSTree;
};

class BinSTree {
            Nod* root;
            Nod* nodInsert(Nod* p, char* key, void* obj);
            Nod* LeftmostNod(Nod* parent, Nod* curent);
            Nod* deleteNod(Nod* p, char* key);
            void inordine(Nod* p, int level);

      public:
            BinSTree();

            void insert(char* key, void* obj);
            void* search(char* key);
            void remove(char* key);
            void inordine();
};

#endif

================== BinSTree.cpp ==============
#include "binstree.h"
#include "student.h"

/* ToDo:
   Sa se implementeze:
     - constructor de copiere
     - destructorul
     - functiile de vizitare in preordine si postordine
     - operatorul "="
     - operatorul ">>"  cu rol de stergere nod
     - "[]" intoarce al i-lea nod la parcurgerea in inordine
*/

/* Constructorul clasei*/
BinSTree::BinSTree() {
```

```
  root = NULL;
}

/* Insereaza un element pe pozitia corespunzatoare cheii.
   Parametrii:
     key - cheia de inserare (sir de caractere)
     obj - pointer catre un obiect
*/
void BinSTree::insert(char* key, void* obj) {
  root = nodInsert(root, key, obj);
}

/* Insereaza un element intr-un arbore de radacina p. Functie recursiva.
   Parametrii:
     p   - radacina arborelui
     key - cheia de inserare (sir de caractere)
     obj - pointer catre obiectul ce contine informatia primara
*/
Nod* BinSTree::nodInsert(Nod* p, char* key, void* obj) {
  if (p == NULL)
    return new Nod(key, obj);
  else {
    int c = strcmp(key, p->m_key);
    if (c < 0)
      p->left = nodInsert(p->left, key, obj);
    else
      if (c > 0)
      p->right = nodInsert(p->right, key, obj);
  }

  return p;
}

/* Cauta informatia atasata unei chei. Cautarea este nerecursiva.
   Parametrii:
     key - cheia
*/
void* BinSTree::search(char* key) {
  Nod* p = root;
  int c;

  while (p !=  NULL) {
    c = strcmp(key, p->m_key);
    if (c < 0)
      p = p->left;
    else
      if (c > 0)
      p = p->right;
      else
      break;
  }
  if (p != NULL)
    return p->m_reference;

  return NULL;
}

/* Intoarce cel mai din stinga nod al descendentului drept al unui nod.
   Parametrii:
     parent - nodul parinte
     curent - descendentul drept al nodului parent
```

```
*/
Nod* BinSTree::LeftmostNod(Nod* parent, Nod* curent) {
  while (curent->left != NULL) {
    parent = curent;
    curent = curent->left;
  }

  if (parent->right == curent)
    parent->right = curent->right;
  else
    parent->left = curent->right;

  return curent;
}

/* Sterge nodul de cheie key din arborele de radacina p.
   Parametrii:
     p   - radacina arborelui
     key - cheia
*/
Nod* BinSTree::deleteNod(Nod* p, char* key) {
  Nod* tmp;
  int ind;

  if (p != NULL)  {
    ind = strcmp(p->m_key, key);
    if (ind < 0)
      p->right = deleteNod(p->right, key);
    else
      if (ind > 0)
      p->left = deleteNod(p->left, key);
      else
      if (p->left == NULL || p->right == NULL) {
        if (p->left != NULL)
          tmp = p->left;
        else
          tmp = p->right;
        p = tmp;
      }
      else {
        tmp = p->right;
        tmp = LeftmostNod(p, tmp);
        tmp->left = p->left;
        tmp->right = p->right;
        p = tmp;
      }
  }

  return p;
}

/* Elimina un nod de cheie specificata dintr-un arbore.
   Parametrii:
     key - cheia
*/
void BinSTree::remove(char* key) {
  root = deleteNod(root, key);
}

/*  Viziteaza in inordine si afiseaza informatia asociata nodurilor arborelui.
```

```
    Aici este singurul loc din program in care arborele trebuie "sa stie" de
clasa Student.
    Parametrii:
       p  - nodul curent
      level - "adincimea" nodului curent
*/
void BinSTree::inordine(Nod* p, int level) {
  if (p != NULL) {
    inordine(p->left, level+1);
    for (int i = 0; i < level; i++)
      cout << "  ";
    ((Student*)p->m_reference)->afis();
    inordine(p->right, level + 1);
  }
}

/* Parcurge in inordine arborele.*/
void BinSTree::inordine() {
   cout<<"----------------------------\n";
   inordine(root, 1);
}
```

`================== Student.h =================`

```
#ifndef _STUDENT
#define _STUDENT

#include <stdlib.h>
#include <iostream.h>
#include <string.h>

class Student {
            int   matricol;
            char* nume;
            char* adresa;
            void makefree();

      public:
            Student();
            Student(char *nume, char* adresa, int matricol);
            Student(Student& s);
            ~Student();

            void afis();
            char* getName();
};

#endif
```

`================== Student.cpp ===============`

```
#include "student.h"

/* TODO:
    Sa se implementeze:
      - functiile de access la variabilele private
      - operatorul "="
      - operatorul "<<" pentru afisare
*/

/* Constructor fara parametrii pentru clasa student*/
Student::Student() {
  nume = "John Doe";
```

```
  adresa = "Unknown";
  matricol = 1;
}

/*Constructor obisnuit:
  parametrii:
    nume - numele studentului
    adresa - adresa (strada,nr,oras etc)
    matricol - numarul matricol
*/
Student::Student(char* _nume, char* _adresa, int _matricol) {
  if (_nume != NULL) {
    nume = new char [sizeof(char)*(strlen(_nume) + 1)];
    strcpy(nume, _nume);
  }
  if (_adresa != NULL) {
    adresa = new char [sizeof(char)*(strlen(_adresa) + 1)];
    strcpy(adresa, _adresa);
  }
  matricol = _matricol;
}

/* Constructor de copiere.
   Parametrii:
     s - referinta la un obiect de tip Student
*/
Student::Student(Student& s) {
  if (s.nume != NULL) {
    nume = new char [sizeof(char)*(strlen(s.nume) + 1)];
    strcpy(nume, s.nume);
  }
  if (s.adresa != NULL) {
    adresa = new char [sizeof(char)*(strlen(s.adresa) + 1)];
    strcpy(adresa, s.adresa);
  }
  matricol = s.matricol;
}

/* Elibereaza zona de memorie referita de variabilele m_nume si m_adresa.
Este apelata de destructor.
*/
void Student::makefree() {
  if (nume != NULL)
    delete[] nume;
  if (adresa != NULL)
    delete[] adresa;
}

/*Destructorul clasei Student*/
Student::~Student() {
  makefree();
}

/*Afiseaza intr-un mod cat mai placut elementele componente ale clasei.*/
void Student::afis() {
  cout<<"Nume: " << nume<<"\t";
  cout<<"Adresa: " << adresa<<"\t";
  cout<<"Matricol: " << matricol<<endl;
}

/* Functie de tip "getter": intoarce valoarea variabilei interne m_nume.*/
```

```
char* Student::getName() {
  return nume;
}

================== Testtree.cpp ====================
#include "binstree.h"
#include "student.h"

/* TODO:
   Sa se scrie o functie care sa:
     - introduca nodurile arborelui
     - stearga noduri pe baza unei chei
*/

Student* clasa[] = {new Student("Maria","A",1),new Student("Alin","B",2),
                new Student("Traian","C",3), new Student("Decebal","D",4),
                new Student("Cristina","E",5),new Student("Miruna","F",6),
                new Student("Maria","G",7),new Student("Dragos","H",8),
                new Student("Daniel","I",9),new Student("Costica","J",10),
                new Student("Daniela","K",11),new Student("Constantin","L",12),
                new Student("Dorin","M",13),new Student("Florica","N",14)
            };

void main(void) {
  BinSTree bst;
  char key[50];

  for (int i = 0; i < 14; i++)
    bst.insert(clasa[i]->getName(), clasa[i]);
  bst.inordine();
  while (1) {
    cout << "Cheia: ";
    cin >> key;
    if (strcmp(key, "Exit") == 0)
      break;
    bst.remove(key);
    bst.inordine();
  }
}
```

Probleme

1. **Gestiunea unei biblioteci.**
Informatiile aferente cartilor unei biblioteci sunt:
- nume carte (sir de 20 caractere)
- nume autor (sir de 20 caractere)
- greutate (numar real)
- numar cod (pot fi mai multe carti de acelasi fel, dar cu numere de cod diferite); (longint)
- numele clientului care a împrumutat-o (sir de 20 caractere)
Operatii:
- actualizare carti (modificare nume client care a imprumutat-o si/sau numar cod)
- adaugare carte noua;
- afisarea cartilor împrumutate de un client;
- afisarea tuturor cartilor;
Cerinte:
- cel putin 2 clase: clasa Carte, clasa Biblioteca (ce va fi asociata cu un fisier de tip text)
- operatiile se vor desfasura direct in fisier

2. Sa se scrie o aplicatie care sa permita efectuarea unor operatii cu matrici rare (care au dimensiuni foarte mari, dar putine elemente nenule), care au elemente reale.
O matrice rara este memorata ca un vector de pointeri la o lista circulara. Vectorul este asociat liniilor matricei, fiecare element al sau indica spre lista elementelor nenule corespunzatoare liniei curente. Un element dintr-o lista circulara contine valoarea elementului nenul, precum si indicele coloanei la care apartine.
Operatii:
- Citire matrice;
- Afisare matrice;
- Evaluarea unei expresii cu matrici rare de forma: a+b.
Cerinte:
- Cel putin 3 clase: clasa MatriceRara, clasa Lista, clasa Element.
- Posibilitatea salvarii unui obiect matrice într-un fisier, precum si a restaurarii dintr-un fisier (supraîncarcarea operatorilor << si >> pentru clasa MatriceRara).

3. Concurs de admitere la facultate (se specifica numele facultatii si data de sustinere a concursului).
Pentru fiecare concurent se cunosc urmatoarele informatii de intrare:
- nume si prenume;
- data nasterii;
- numarul si seria buletinului de identitate;
- pentru fiecare din cele doua probe, trei note ale celor trei corectori;
Operatii:
- sortare concurenti dupa nume;
- calcul nota la fiecare proba;
- calcul medie;
- sortare concurenti dupa medie, iar pentru medii egale dupa nume;
- separarea în 2 liste (admisi, respinsi);
Cerinte:
- cel putin 2 clase: clasa Candidat, clasa Admitere (ce va fi asociata cu un fisier binar)
- operatiile se vor desfasura direct in fisier

4. Să se realizeze o clasa *Queue* implementată sub forma unei liste liniare simplu înlănțuită utilizând template.

5. Să se realizeze clasa *Queue* ce implementează o coadă circulară sub forma unui tablou şi utilizează template.

6. Să se implementeze clasa următoare ce reprezintă numere mari şi operații cu acestea:

```
class BigDecimal{
            byte *cifre;
            int len;
      public:
            BigDecimal(byte* a, int lungime);
            BigDecimal(BigDecimal&);
            BigDecimal(long number);

            ~BigDecimal();

            void  readNumber (istream&);
            void  writeNumber (ostream&);
            BigDecimal add (BigDecimal&);
            BigDecimal multiplz (BigDecimal& );
            int compareTo (BigDecimal&);
};
```

7. Definiți clasa *Year* ce gestionează zilele unui an sub froma de zile lucrătoare şi zile nelucrătoare. Clasa memorează zilele din an cu ajutorul unui vector de biți (un caracter fără semn reprezintă 8 biți):

```
enum Month {
  Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
};

class Year {
      public:
            // numarul de octeti din vector = 46
            enum Bytes { YearBytes = 46 };
      private:
            short year; // anul curent
            unsigned char vec[YearBytes]; // vectorul de 8*bytes biti
      public:
            Year(const short year);
            void WorkDay (const short day); // seteaza o zi lucratoare
            void OffDay (const short day); // seteaza o zi nelucratoare
            Bool Working (const short day); // determina tipul zilei
            // converteste o data la o zi din anul curent
            short Day (const short day, const Month month, const short year);
};
```

8. Să se realizeze un program C++ care să gestioneze comenzile de la o fabrică de componente electronice. O comandă este compusă din:

- un cod numeric format din 11 cifre
- descrierea clientului care a realizat comanda
- lista componentelor.

O componentă electronică se identifică prin: un cod numeric format din maxim 6 cifre, denumire, cantitate, costul pe unitatea de produs.

Informațiile corespunzătoare comenzilor vor fi pastrate pe disc într-un fişier. Aplicația va putea să creeze, modifice şi să şteargă o comandă.

9. Realizați un container eterogen de figuri. Containerul poate gestiona în acelaşi timp cercuri, triunghuri, pătrate. Fără a modifica în nici un fel codul său, acesta va putea gestiona şi alte tipuri de figuri ce pot fi adăugate ulterior.

```
class Point {
            int x, y;

      public:
            Point(int initX, int initY);
            int getX();
            void setX(int);

            int getY();
            void setY(int);
};

class Rectangle {
            int lungime, latime;
            Point p;
            int culoare;

      public:
            Rectangle(Punct _p, int _lungime, int _latime);

            void setCuloare(int);
            void translate(int dx, int dy);
};

class Circle {
            int raza;
            Point p;
            int culoare;

      public:
            Circle(Punct _p, int _raza);

            void setCuloare(int);
            void translate(int dx, int dy);
};
```

a) Introduceti o clasa container *Grup* ce va contine un numar de forme geometrice. Aceasta clasa va avea o metoda

```
void translate(int dx, int dy)
```

a carei aplicare va conduce la translatarea intregii multimi de forme geometrice.

Abstractizati conceptul de forma geometrica (*Rectangle, Circle*) intr-o clasa abstracta *Shape*.

b) Clasa *Grup* va trebui sa fie derivata din clasa

```
class Colorable{
      set getColorate(int);
};
```

Metoda `getColorate()` va intoarce o multime de elemente (forme geometrice) ce sunt colorate cu acceasi culoare.

c) Implementati in clasa *Grup* o metoda

```
map getFormeGroupbyColor()
```

astfel incat va intoarce o clasa *map* (din STL) unde cheia este culoarea, iar valorile sunt o lista de forme geometrice (*Rectangle* sau *Circle*) ce au aceeasi culoare.

10.

```
class User {
            string name;

      public:
            User(string _name);
            string getName();
            void setName(string);
};

class File {
            int size;
            User owner;
            String fileName;

      public:
            File(string _nameFile, int _size, User _owner);

            string getFileName();
            int getSize();
            void setOwner(User);
};
```

a) Introduceti o clasa container *Folder* ce poate contine fisiere si alte directoare. Aceasta clasa va avea o metoda

```
void setOwner(User _owner)
```

a carei aplicare va conduce la modificarea proprietarului intregii multimi de fisiere si subdirectoare.

Abstractizati conceptul de fisier (*File*, *Director*) intr-o clasa abstracta *Resource*.

Un director va contine o colectie de resurse.

b) Clasa *Folder* va trebui sa fie derivata din clasa

```
class  Component{
            set getOwners();
};
```

Metoda `getOwners()` va intoarce multimea tuturor proprietarilor de fisiere asociate unei resurse.

c) Implementati in clasa *Folder* o metoda

```
map getResourcesGroupbyOwner()
```

astfel incat va intoarce o clasa *map* ce contine perechi de forma <utilizator, lista tuturor fisierelor ce il au ca proprietar >.