========== Deriv1.cpp ==========

```
#include <iostream.h>

class A{
   int i;
 public:
   A(int i = 0){
     Cout << "Constructor A" << endl;
     this->i = i;
   }
};

class B{
   int j;
 public:
   B(int j = 0){
     cout << "Constructor B" << endl;
     this->j = j;
   }
};

class C:public A{
  B b;
};

void main(){
  C c;
}
```

========== Deriv2.cpp ==========

```
#include <iostream.h>

class Parent {
    int i;
  public:
    Parent(int I) : i(I) {
      cout << "Parent(int I)\n";
    }

    Parent(const Parent& b) : i(b.i) {
      cout << "Parent(Parent&)\n";
    }

    Parent() :i(0) { cout << "Parent()\n"; }
};

class Member {
    int i;
  public:
    Member(int I) : i(I) {
      cout << "Member(int I)\n";
    }

    Member(const Member& m) : i(m.i) {
      cout << "Member(Member&)\n";
    }
};

class Child : public Parent {
    int i;
    Member m;
```

```
  public:
    Child(int I) : Parent(I), i(I), m(I) {
      cout << "Child(int I)\n";
    }
};

int main() {
  Child d(2);
  cout << "copy-constructor: " << endl;
  Child d2 = d; // constructor de copiere
}
```

========== Biblioteca ==========

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

#define STR_LEN 30

class Cititor{
            char nume[STR_LEN];
            char prenume[STR_LEN];
            long codClient;
      public:
            Cititor(char[], char[], long);
            Cititor(Cititor&);

            char* getNume();
            char* getPrenume();
            long getCodClient();
            void setCodClient(long);
};

Cititor::Cititor(char _nume[], char _prenume[], long _cod){
  strcpy(nume, _nume);
  strcpy(prenume, _prenume);
  codClient = _cod;
}

Cititor::Cititor(Cititor& c){
  strcpy(nume, c.nume);
  strcpy(prenume, c.prenume);
  codClient = c.codClient;
}

char* Cititor::getNume(){
  return nume;
}

char* Cititor::getPrenume(){
  return prenume;
}

long Cititor::getCodClient(){
  return codClient;
}

void Cititor::setCodClient(long cod){
  codClient = cod;
}
/*
```

```
*   Clasa Carte implementeaza conceptul de carte.
*/

class Carte{
      char* autor; //autorul cartii
      char* title; //titlul cartii
      char* isbn;  //codul ISBN
      int   price; //pretul

    public:

      /*
      *   Constructor
      */
      Carte(char* autor, char* titlu, char* isbn, int price);
      /*
      *  Constructor de copiere
      */
      Carte(Carte&);

      /*
      *  Initializeaza autorul
      */
      void setAuthor(char*);
      /*
      *  Returneaza autorul cartii
      */
      char* getAuthor(void);

      /*
      *   Initializeaza titlul cartii.
      */
      void setTitle(char*);
      /*
      *  Returneaza titlul cartii
      */
      char* getTitle(void);

      /*
      *  Initialiezeaza cota ISBN
      */
      void setISBN(char*);
      /*
      *   Returneaza cota ISBN a unei carti
      */
      char* getISBN(void);

      /*
      *   Initializeaza pretul cartii.
      */
      void setPrice(int);
      /*
      *  Returneaza pretul unei carti
      */
      int  getPrice(void);
};

Carte::Carte(char* autor, char* titlu, char* isbn, int price){
  setAuthor(autor);
  setTitle(titlu);
  setISBN(isbn);
```

```
  setPrice(price);
}

Carte::Carte(Carte& b){
  setAuthor(b.autor);
  setTitle(b.title);
  setISBN(b.isbn);
  setPrice(b.price);
}

void Carte::setAuthor(char* newAuthor){
  autor = new char [strlen(newAuthor) + 1];
  strcpy(autor, newAuthor);
}

char* Carte::getAuthor(){
  return autor;
}

void Carte::setTitle(char* newTitle){
  title = new char [strlen(newTitle) + 1];
  strcpy(title, newTitle);
}

char* Carte::getTitle(){
  return title;
}

void Carte::setISBN(char* newISBN){
  isbn = new char [strlen(newISBN) + 1];
  strcpy(isbn, newISBN);
}

char* Carte::getISBN(){
  return isbn;
}

void Carte::setPrice(int newPrice){
  price = newPrice;
}

int Carte::getPrice(){
  return price;
}

typedef enum {liber, imprumutat} Stare;

class Item{
            char* dataImp;
            char* dataRet;
            Carte* carte;
      public:
            Item(){
              dataImp = NULL;
              dataRet = NULL;
              carte = NULL;
            }

            void setDataImprumut(char* data, Carte* _carte){
              dataImp = new char [strlen(data) + 1];
              strcpy(dataImp, data);
```

```
            carte = _carte;
          }

          char* getDataImprumut(){
            return dataImp;
          }

          void setDataReturnat(char* data){
            dataRet = new char [strlen(data) + 1];
            strcpy(dataRet, data);
          }

          char* getDataReturnat(){
            return dataRet;
          }

          Carte* getCarte(){
            return carte;
          }

          Item& operator = (Item& item){
            setDataImprumut(item.getDataImprumut(), item.getCarte());
            setDataReturnat(item.getDataReturnat());

            return *this;
          }

};

#define MAX_ITEMS 20

class Fisa{
          Item tab[MAX_ITEMS];
          int lastIndex;
    public:
          Fisa();
          Fisa(Fisa&);

          void imprumutaCarte(char*, Carte*);
          void returneazaCarte(char*, Carte*);
          Carte** getImprumut();
};

Fisa::Fisa(){
  lastIndex = -1;
}

Fisa::Fisa(Fisa& fisa){
  lastIndex = fisa.lastIndex;
  for (int i = 0; i <= lastIndex; i++){
    tab[i] = fisa.tab[i];
  }
}

void Fisa::imprumutaCarte(char* data, Carte* carte){
  if (lastIndex < MAX_ITEMS - 1){
    lastIndex++;
    tab[lastIndex].setDataImprumut(data, carte);
  }
}
```

```
void Fisa::returneazaCarte(char* data, Carte* carte){
  for (int i = 0; i <= lastIndex; i++)
    if (tab[i].getCarte()->getISBN() == carte->getISBN()
      && tab[i].getDataReturnat() != NULL)
      tab[lastIndex].setDataReturnat(data);
}

Carte** Fisa::getImprumut(){
  Carte** carti;
  int i;
  int counter = 0;

  for (i = 0; i <= lastIndex; i++){
    if (tab[i].getDataReturnat() == NULL)
      counter++;
  }
  carti = new Carte* [counter];

  for (i = 0; i <= lastIndex; i++)
    if (tab[i].getDataReturnat() == NULL)
      carti[i] = tab[i].getCarte();

  return carti;
}
```

==========

```
#include <iostream.h>

class Patrulater{
            float lat[4];
      public:
            Patrulater(float, float, float, float);
            Patrulater(Patrulater&);

            float getLatura(int);
            float getPerimetru();
            void print();
};

Patrulater::Patrulater(float l1, float l2, float l3, float l4){
  lat[0] = l1;
  lat[1] = l2;
  lat[2] = l3;
  lat[3] = l4;
}

Patrulater::Patrulater(Patrulater& p){
  for (int i = 0; i < 4; i++)
    lat[i] =p.lat[i];
}

float Patrulater::getLatura(int index){
  return lat[index];
}

float Patrulater::getPerimetru(){
  return lat[0] + lat[1] + lat[2] + lat[3];
}

void Patrulater::print(){
```

```
  cout << "Patrulater = [" << lat[0] << "," << lat[1] << ",";
  cout << lat[2] << "," << lat[3] << "] \n";
}

class Dreptunghi:public Patrulater{
      public:
            Dreptunghi(float, float);
            Dreptunghi(int, int, int, int);
            Dreptunghi(Dreptunghi&);

            void print();
};

Dreptunghi::Dreptunghi(float lungime, float latime) :
      Patrulater(lungime, latime, lungime, latime){}

Dreptunghi::Dreptunghi(int x1, int y1, int x2, int y2) :
      Patrulater( x2 - x1, y2 - y1, x2 - x1, y2 - y1){
}

Dreptunghi::Dreptunghi(Dreptunghi& d) : Patrulater(d){}

void Dreptunghi::print(){
  cout << "Dreptunghi = [" << getLatura(0) << "," << getLatura(1) << "] \n";
}

/*
1. Implementati clasa Patrat.

class Patrat:public Dreptunghi{
      public:
            Patrat(float);
            Patrat(float, float, float);
            Patrat(Patrat&);

            void print();
};
*/

void main(void){
  Patrulater p(5, 6, 8, 6);

  cout << "Perimetru: " << p.getPerimetru() << endl;
  p.print();

  Dreptunghi d(10, 17);

  cout << "Perimetru dreptunghi: " << d.getPerimetru() << endl;
  d.print();
}
```

Temă

La sfârşitul fiecărui semestru, secretariatul facultății trebuie să realizeze diferite rapoarte. Pentru fiecare student se cunoaşte numărul de credite obținute în cursul semestrului. Folosind clasele din diagrama de mai jos, implementați un program care să ofere următoarele funcționalități:

1) să citească o listă de studenți dintr-un fişier
2) să genereze o listă cu toți studenții ordonați alfabetic

3) să genereze o listă cu toți studenții ordonați în ordinea descrescătoare a numărului de credite
4) să genereze o listă cu studenții ordonați alfabetic pe grupe
5) să genereze o listă cu studenții ordonați în ordinea descrescătoare a numărului de credite pe grupe