Programare Orientata pe Obiect
Lucrarea de laborator Nr. 3

Clase si obiecte.

1. Definitia clasei. Date si functii membre ale clasei.

În programarea problemelor complexe intervin concepte noi care nu pot fi exprimate simplu prin tipuri predefinite de date. Orice limbaj de programare pune la dispozitia programatorului un numar de tipuri predefinite, care însa, în mod frecvent, nu corespund tuturor conceptelor necesare programului. Astfel de concepte se implementeaza în limbajul C++ prin intermediul claselor. O *clasa* este un tip de data definit de utilizator. O declarare a unei clase defineste un tip nou care reuneste date si functii. Acest tip nou poate fi folosit pentru a declara obiecte de acest tip. Deci, un obiect este un exemplar (instanta) a unei clase. Forma generala de declaratie a unei clase este urmatoarea:

```
class nume_clasa{
        date si functii membre private
   specificatori_de_acces
        date si functii membre
   specificatori_de_acces
        date si functii membre
   ........................
   specificatori_de_acces
        date si functii membre
} lista_obiecte;
```

Corpul clasei contine definitii de date membre ale clasei si definitii sau declaratii (prototipuri) de functii membre ale clasei, despartite prin unul sau mai multi specificatori de acces. Un *specificator de acces* poate fi unul din cuvintele cheie din C++:

```
public:
private:
protected:
```

Specificatorii `private` si `protected` asigura o protectie de acces la datele sau functiile membre ale clasei respective, iar specificatorul `public` permite accesul la acestea si din afara clasei. Efectul unui specificator de acces dureaza pâna la urmatorul specificator de acces. Implicit, daca nu se declara nici un specificator de acces, datele sau functiile membre sunt de tip `private`.

O clasa are un domeniu de definitie (este cunoscuta în acest domeniu) care începe de la prima pozitie dupa încheierea corpului clasei si se intinde pâna la sfârsitul fisierului în care este introdusa definitia ei si al fisierelor care îl includ pe acesta. Datele si functiile membre ale clasei care nu sunt declarate `public` au în mod implicit, ca domeniu de definitie, domeniul clasei respective, adica sunt cunoscute si pot fi folosite numai din functiile membre ale clasei. Datele si functiile membre publice ale clasei au ca domeniu de definitie întreg domeniul de definitie al clasei, deci pot fi folosite în acest domeniu. Pentru definirea unei functii în afara clasei (dar, bineînteles în domeniul ei de definitie) numele functiei trebuie sa fie însotit de numele clasei respective prin intermediul operatorului de rezolutie (`::`). Sintaxa de definire a unei functii în afara clasei este urmatoarea:

```
tip_returnat nume_clasa::nume_functie(lista_argumente){
   //corpul functiei
}
```

În domeniul de definitie al unei clase se pot crea obiecte ale clasei. Fiecare obiect contine câte o copie individuala a fiecarei variabile a clasei respective si pentru fiecare obiect se poate apela orice fel de functie membra publica a clasei.

Accesul la datele membre publice sau apelul functiilor membre publice ale unui obiect se poate face folosind un operator de selectie membru: operatorul punct (`.`) daca se cunoaste obiectul, sau operatorul `->` daca se cunoaste pointerul la obiect.

Exercitiul 1:

Definiti o clasa `Complex` a numerelor complexe care contine doua date membre `private`, `re` si `im` de tip `double` si trei functii membre `public`, `init()`, `set()` si `display()`. Functia `init()` initializeaza datele membre ale clasei cu valoarea `0`. Functia `set()` modifica valorile datelor membre cu valorile transmise ca parametrii. Functia `display()` afiseaza partea reala respectiv partea imaginara a numarului complex. În functia `main()` a programului declarati un obiect cu numele `c1` de tipul `Complex`. Pentru acest obiect apelati fiecare dintre functiile membre ale clasei.

2. Constructori si destructori

Utilizarea unor functii membre ale unei clase, asa cum sunt functiile `init()` si `set()` din clasa `Complex`, pentru initializarea obiectelor este neeleganta si permite strecurarea unor erori de programare. Deoarece nu exista nici o constrângere din partea limbajului ca un obiect sa fie initializat (de exemplu, nu apare nici o eroare de compilare daca nu este apelata functia `init()` sau `set()` pentru un obiect din clasa `Complex`), programatorul poate sa uite sa apeleze functia de initializare sau sa o apeleze de mai multe ori ceea ce poate produce erori. Din aceasta cauza, limbajul C++ prevede o modalitate eleganta si unitara pentru initializarea obiectelor de tipuri definite de utilizatori, prin intermediul unor functii speciale numite functii constructor sau mai scurt, *constructori*.

Un constructor este o functie cu acelasi nume cu numele clasei, care nu returneaza nici o valoare (mai mult, nu are specificat tipul returnat) si care initializeaza datele membre ale clasei. Pentru aceeasi clasa pot fi definite mai multe functii constructor, ca functii supraîncarcate, care pot fi selectate de compilator în functie de numarul si tipul argumentelor de apel, la fel ca în orice supraîncarcare de functii. Un constructor implicit pentru o clasa `X` este un constructor care poate fi apelat fara nici un argument. Deci un constructor implicit este un constructor care are lista de argumente vida sau un constructor cu unul sau mai multe argumente, toate fiind prevazute cu valori implicite.

În general constructorii se declara de tip `public` pentru a putea fi apelati din orice punct al domeniului de definitie al clasei respective. La crearea unui obiect dintr-o clasa oarecare este apelat *implicit* acel constructor al clasei care prezinta cea mai buna potrivire a argumentelor. Daca nu este prevazuta *nici o* functie constructor, compilatorul genereaza un constructor implicit de tip `public`, ori de câte ori este necesar.

Un constructor este apelat ori de câte ori este creat un obiect dintr-o clasa care are un constructor (definit sau generat de compilator). Un obiect poate fi creat în urmatoarele moduri:

- ca variabila globala;
- ca variabila locala;
- prin utilizarea explicita a operatorului `new` ;
- ca obiect temporar.

Exercitiul 2:

Definiti mai multe functii constructor pentru clasa `Complex`: constructor fara argumente, cu un argument si cu doua argumente astfel încât urmatoarea secventa de program sa se execute corect.

```
void main(){
      Complex c1;
      Complex c2(5);
      Complex c3(1,2);
}
```

Multe din clasele definite într-un program necesita o operatie inversa celei efectuate de constructor, pentru stergerea completa a obiectelor atunci când sunt distruse (eliminate din memorie). O astfel de operatie este efectuata de o functie membra a clasei, numita functie *destructor*. Numele destructorului unei clase `X` este `~X()` si este o functie care nu primeste nici un argument si nu returneaza nici o valoare. Destructorii sunt apelati implicit în mai multe situatii:

1. atunci când un obiect local sau temporar iese din domeniul de definitie;
2. la sfârsitul programului pentru obiectele globale;
3. la apelul operatorului `delete` pentru obiectele alocate dinamic.

Daca o clasa nu are un destructor, compilatorul genereaza un destructor implicit.

3. Constructori de copiere

Functia principala a unui constructor este aceea de a initializa datele membre ale obiectului creat, folosind pentru aceasta operatie valorile primite ca argumente. O alta forma de initializare care se poate face la crearea unui obiect este prin copierea datelor unui alt obiect de acelasi tip. Aceasta operatie este posibila prin intermediul constructorului de copiere. Forma generala a constructorului de copiere al unei clase `X` este:

```
X::X(X& r){
   // initializare obiect folosind referinta r
}
```

Constructorul primeste ca argument o referinta `r` la un obiect din clasa `X` si initializeaza obiectul nou creat folosind datele continute în obiectul de referinta `r`.

Exercitiul 3:

Sa se implementeze constructorul de copiere pentru clasa `Complex` astfel încât sa se poata crea obiectele `c2` si `c3` ca în secventa de program de mai jos:

```
void main(){
   Complex c1(4,5);
   Complex c2(c1);
   Complex c3 = c2;
   c3.display();
}
```

Constructorul de copiere poate fi definit de programator. Daca nu este definit un constructor de copiere al clasei, compilatorul genereaza un constructor de copiere care copiaza datele membru cu membru din obiectul referinta în obiectul nou creat. Însa, în cazul în care un obiect contine date alocate

dinamic în memoria libera, constructorul de copiere generat implicit de compilator copiaza doar datele membre declarate în clasa (membru cu membru) si nu stie sa aloce date dinamice pentru obiectul nou creat. Folosind un astfel de constructor, se ajunge la situatia ca doua obiecte, cel nou creat si obiectul referinta, sa contina pointeri cu aceeasi valoare, deci care indica spre aceeasi zona de memorie. O astfel de situatie este o sursa de erori de executie subtile si greu de depistat. Solutia o reprezinta definirea unui constructor de copiere care sa previna astfel de situatii. Un constructor de copiere definit de programator trebuie sa aloce spatiu pentru datele dinamice create în memoria libera si dupa aceea sa copieze valorile din obiectul de referinta.

4. Stringuri în interiorul obiectelor

O situatie deseori întâlnita în programarea cu clase este aceea în care, ca si data membra a unei clase apar variabile tip sir (pointer catre un sir de caractere).

Exercitiul 4:

Creati o clasa `String` care sa contina o data membra de tip pointer catre un sir de caractere pentru alocarea unui mesaj. Implementati functiile necesare astfel încât codul urmator sa ruleze corect.

```
void main(){
   String s1(”Test 1”);
   String s2 = s1;
   s1.set(”Test 2”);
   s1.display();
   s2.display();
}
```

Un constructor este apelat la definirea obiectului iar destructorul este apelat atunci când obiectul este distrus. Daca exista mai multe declaratii de obiecte, atunci ele sunt construite în ordinea declaratiei si sunt distruse în ordinea inversa a declaratiei. Obiectele membre ale unei clase se construiesc înaintea obiectului respectiv. Destructorii sunt apelati în ordine inversa: destructorul obiectului si apoi destructorii membrilor. Functiile constructor ale obiectelor globale sunt executate înaintea executiei functiei `main()`. Destructorii obiectelor globale sunt apelati în ordine inversa, dupa încheierea functiei `main()`.

5. Obiecte încuibarite

Un obiect încuibarit poate fi ilustrat prin exemplul calculatorului, într-un mod foarte simplu. Computerul, în sine, este alcatuit din multe elemente care lucreaza împreuna, dar care lucreaza complet diferit, cum ar fi tastatura, hard-drive-ul sau sursa de alimentare.Computerul este astfel format din parti diferite si este de dorit „tratarea” tastaturii separat de hard-drive. O clasa computer poate fi compusa din mai multe obiecte diferite prin încuibarire.

Exercitiul 5:

Creati o clasa `Point` cu date membre coordonatele unui punct în plan si o clasa `Circle` cu date membre centrul cercului ca obiect de tip `Point` si raza cercului. Implementati constructorii si destructorii celor doua clase care sa afiseze tipul constructorului, respectiv al destructorului. Creati un obiect de tip `Circle` cu centrul în punctul de coordonate `(1,2)` si raza egala cu `3`. Observati ordinea mesajelor la construc tia si la distrugerea obiectelor.

O discutie despre disk-drive-uri ar putea începe prin examinarea caracteristicile disk-drive-urilor în general. Ar putea fi studiate detaliile unui hard-drive si diferentele pe care le are un floppy-disk-drive.

Aceasta ar implica mostenirea pentru ca multe dintre caracteristicile drive-urilor pot caracteriza drive-uri în general, dupa care apar diferente dependente de cazul particular (floppy, hard, etc.). Mostenirea va fi studiata în Lucrarea de laborator Nr. 5.

6. Alocarea dinamica a obiectelor

Obiectele (instante ale claselor) se pot aloca în memoria libera folosind operatorul `new`. La alocarea memoriei pentru un singur obiect se pot transmite argumente care sunt folosite pentru initializarea obiectului, prin apelul acelei functii constructor a clasei care prezinta cea mai buna potrivire cu argumentele de apel. Deasemenea constructorul nu este apelat atunci când se declara un pointer ci când se aloca obiectul. Eliberarea memoriei ocupata de un obiect se realizeaza prin operatorul `delete`, care apeleaza implicit destructorul clasei. Destructorul clasei este apelat în intructiunea `delete` înainte de stergerea efectiva a obiectului.

Exercitiul 6:

Alocati si eliberati obiecte de tip `Complex` care sa apeleze tipurile de constructori definiti în exercitiile anterioare.

Pentru alocarea dinamica a unui vector de obiecte de tipul `X`, trebuie sa existe un constructor implicit al clasei `X`, care este apelat pentru fiecare din elementele vectorului creat. Pentru stergerea unui vector de obiecte se foloseste operatorul `delete[]` care apeleaza implicit destructorul clasei pentru fiecare din elementele vectorului alocat. Daca, pentru un vector alocat dinamic, se apeleaza operatorul `delete` (în loc de `delete[]`) se apeleaza destructorul clasei o singura data dupa care rezultatul executiei este imprevizibil, cel mai adesea se produce eroare de executie si abandonarea programului.

7. Tablouri de obiecte

Pentru ca prin definirea unei clase se creaza de fapt un nou tip de date, se pot declara si defini vectori de obiecte, instante ale claselor, similar declararii de vectori de date de tipuri fundamentale. Pentru a declara un tablou de obiecte, constructorul clasei nu trebuie sa aiba parametrii. Acest lucru pare normal, pentru ca e putin probabil sa se doreasca creearea unor obiecte initializate cu aceleasi valori pentru datele membre.

Exercitiul 7:

Creati dinamic un vector de patru numere complexe. Implementati o functie care sa însumeze elementele vectorului. Afisati rezultatul apelând functia `display()` pentru obiectul `Complex` rezultat.

Adunarea numerelor complexe presupune, de fapt doua operatii de adunare, o adunare a partilor reale ale numerelor complexe si o adunare a partilor imaginare, asfel ca simpla folosire a operatorului de adunare (`+`) pentru obiecte ale clasei `Complex` va genera erori la compilare. Lasarea publica a datelor membre ale clasei si accesarea lor în afara clasei (într-o functie nemembra a clasei sau în functia `main()`) pentru a implementa o functie care sa adune obiecte de tipul `Complex` încalca principiul *încapsularii* datelor din programarea obiect-orientata.

8. Încapsularea obiectelor si modularizarea programelor

Programarea orientata pe obiecte permite programatorului sa îsi partitioneze programele în componente individuale (module) astfel încât sa ascunda informatia si sa reduca timpul de depanare. Astfel poate fi creat un *fisier header* (fisier cu extensia `.h`) care sa contina numai definitia de clasa, fara a

fi date detalii despre cum sunt implementate metodele. Fisierul header nu poate fi compilat sau executat. Sunt date definitiile si/sau declaratiile complete pentru folosirea clasei, dar nu sunt date detaliile despre implementare. Metodele clasei declarate în fisierul header sunt definite în *fisierul de implementare a clasei* (fisier sursa). Fisierul header este inclus în acest fisier. Fisierul de implementare poate fi compilat dar nu poate fi executat pentru ca nu contine functia `main()`. Daca implementarea unei metode este foarte simpla, atunci implementarea metodei respective poate sa apara în fisierul header ca parte a declaratiei.

Când implementarea este inclusa în declaratie, ea va fi asamblata *inline* oriunde aceasta functie este apelata, rezultând un cod mult mai rapid. Separarea definitiei si implementarii este un pas înainte în ingineria software. Fisierul de definire a clasei contine tot ceea ce are nevoie un utilizator pentru a folosi clasa efectiv într-un program. Nu este nevoie de o cunoastere a implementarii metodelor. Daca utilizatorul ar avea acces la codul implementarii, prin studierea lui ar putea gasi poate mici trucuri pentru a face programul putin mai eficient, dar s-ar ajunge la un software neportabil si posibile buguri, mai târziu, în cazul în care se schimba implementarea fara a schimba si interfata. Încapsularea separa comportarea (accesata prin interfata) de structura, definita prin implementare. Acest mod de ascundere a informatiei are un impact foarte mare în calitatea software-lui dezvoltat în cadrul unui proiect mare. Un alt motiv pentru ascunderea informatiei este si unul economic. Furnizorii de compilatoare au dat numeroase functii de librarie, dar nu furnizeaza si codul sursa, ci numai interfata la ele. Astfel în programe desi este cunoscut modul în care trebuie folosite functiile de biblioteca (de exemplu, `printf()` si `scanf()` din `stdio.h`), nu este cunoscut felul cum au fost scrise si nici nu este nevoie. În acelasi fel, o firma care produce librarii de înalta calitate dezvolta si testeaza clasele respective pentru o taxa de licenta. Utilizatorul va primi numai interfetele si codul obiect (librariile).

Având fisierul header si fisierul de implementare a clasei, poate fi creat un fisier sursa de folosire efectiva a clasei care contine functia `main()` si care poate fi compilat si executat.

Este prezentat în continuare un caz practic de folosire a unei clase. Clasa date este o clasa netriviala care poate fi folosita în orice program pentru a prelua datele curente si de a le afisa într-un string ASCII în unul din cele 4 formate predefinite. El poate fi folosit pentru a stoca orice data si a-l formata pentru afisare:

```
//DATE.H
// This date class is intended to illustrate how to write a non-
//  trivial class in C++.  Even though this class is non-trivial,
//  it is still simple enough for a new C++ programmer to follow
//  all of the details.

#ifndef DATE_H
#define DATE_H

class date {
protected:
   int month;                  // 1 through 12
   int day;                    // 1 through max_days
   int year;                   // 1500 through 2200
   static char out_string[25]; // Format output area
   static char format;         // Format to use for output
   int days_this_month(void); //Calculate how many days are in any given month
                              //Note - This is a private method which can be
                              //called only from within the class itself
public:
   date(void); // Constructor - Set the date to the current date
               // and set the format to 1
   int set_date(int in_month, int in_day, int in_year); // Set the date to
                                    //these input parameters
                                   // if return = 0 -> All data is valid
```

```
                                    // if return = 1 -> Something out of range
   int get_month(void) { return month; };// Get the month,
   int get_day(void)   { return day;   };// day, or
   int get_year(void)  { return year;  };// year of the stored date
   void set_date_format(int format_in) { format = format_in; };// Select the
                                   // desired string output format for use when
                                   // the get_date_string is called
   char *get_date_string(void); // Return an ASCII-Z string depending on the
                                // stored format
                              //format = 1    Aug 29, 1991
                               //format = 2    8/29/91
                               //format = 3    8/29/1991
                               //format = 4    29 Aug 1991    Military time
                               //format = ?    Anything else defaults to format 1
   char *get_month_string(void); // Return Jan Feb Mar Apr etc.
};
#endif

//DATE.CPP
// This file contains the implementation for the date class.

#include <stdio.h>       // Prototype for sprintf
#include <time.h>        // Prototypes for the current date
#include "date.h"

char date::format;         // This defines the static data member
char date::out_string[25]; // This defines the static string

date::date(void) // Constructor - Set date to current date, and set format to
{                // the default of 1
time_t time_date;
struct tm *current_date;

   time_date = time(NULL);                // DOS system call
   current_date = localtime(&time_date);  // DOS system call
   month = current_date->tm_mon + 1;
   day = current_date->tm_mday;
   year = current_date->tm_year + 1900;
   format = 1;
}
int date::set_date(int in_month, int in_day, int in_year)
{
int temp = 0;
int max_days;
   if (in_year < 1500) {   // The limits on the year are purely arbitrary
   year = 1500;           // Check that the year is between 1500 and 2200
      temp = 1;
   } else {
      if (in_year > 2200) {
         year = 2200;
         temp = 1;
      } else
         year = in_year;
   }
   if(in_month < 1) {             // Check that the month is between
      month = temp = 1;           //  1 and 12
   } else {
```

```
      if (in_month > 12) {
         month = 12;
         temp = 1;
      } else
         month = in_month;
   }
   max_days = days_this_month();
   if (in_day < 1) {               // Check that the day is between
      day = temp = 1;              //  1 and max_days
   } else {
      if (in_day > max_days) {
         day = max_days;
         temp = 1;
      } else
         day = in_day;
   }
   return temp;
}
static char *month_string[13] = {" ","Jan","Feb","Mar","Apr","May","Jun",
                                 "Jul", "Aug","Sep", "Oct", "Nov", "Dec"};
char *date::get_month_string(void) { return month_string[month];}
char *date::get_date_string(void)
{
    switch (format) {
    // This printout assumes that the year will be between 1900 and 1999
    case 2 : sprintf(out_string,"%02d/%02d/%02d", month , day, year - 1900);
             break;
    case 3 : sprintf(out_string, "%02d/%02d/%04d",month, day, year);
             break;
    case 4 : sprintf(out_string, "%d %s %04d",day, month_string[month], year);
             break;
    case 1  : // Fall through to the default case
    default : sprintf(out_string,"%s %d,%04d",month_string[month], day, year);
              break;
   }
   return out_string;
}
int days[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
int date::days_this_month(void) // Since this is declared in the private part
         // of the class header is is only available for use within the class.
         //  It is hidden from use outside of the class.
{
   if (month != 2)
      return days[month];
   if (year % 4)       // Not leap year
      return 28;
   if (year % 100)     // It is leap year
      return 29;
   if (year % 400)     // Not leap year
      return 28;
   return 29;          // It is leap year
}
```

```
//USEDATE.CPP
// This is a very limited test of the date class

#include <iostream.h>
#include "date.h"

void main(void)
{
date today, birthday;

   birthday.set_date(7, 21, 1960);
   cout <<"Limited test of the date class\n";
   cout <<"Today is "<<today.get_date_string()<<"\n";//Today is Jan 20, 1992
   cout <<"Birthday is "<<birthday.get_date_string()<<"\n";//Birthday is
                                                          //Jul 21, 1960
   today.set_date_format(4);
   cout << "Today is " << today.get_date_string() << "\n";// Today is
                                                        //20 Jan 1992
   cout << "Birthday is " << birthday.get_date_string() << "\n";// Birthday is
                                                              // 21 Jul 1960
}
```

Observatii:

1. Fisierul denumit `DATE.H` este fisierul header pentru clasa date. Ceea ce aduce nou acest fisier este cuvântul cheie `protected` care va fi explicat în Lucrarea de laborator Nr.5 . Pâna atunci considerati-l echivalent cu `private`.
2. Cuvântul cheie `static` specifica faptul ca datele membre declarate statice vor determina existenta câte unei singure copii a datelor respective, care nu apartine nici unuia dintre obiectele clasei dar este partajata de toate acestea. Definirea unei date membre statice se face în afara clasei prin redeclararea variabilei folosind operatorul de rezolutie si numele clasei careia îi apartine (similar cu definirea functiilor membre în afara clasei). O variabila membra de tip `static` a unei clase exista înainte de a fi creat un obiect din clasa respectiva, si daca nu este initializata explicit, este initializata implicit cu `0`. Cea mai frecventa utilizare a datelor membre statice este de a asigura accesul la o variabila comuna mai multor obiecte, deci pot înlocui variabilele globale.
3. Fisierul `DATE.CPP` este implementarea pentru clasa `date`.
4. Programul `USEDATE.CPP` este un program simplu care foloseste clasa `date` pentru a lista datele curente pe monitor.

Exercitiul 8:

Definitio clasa `nume` similara cu clasa `date` si care poate stoca orice nume în trei parti si care returneaza întregul nume în formatele urmatoare:

```
John Paul Doe
J.P. Doe
Doe, John Paul
```

În programarea orientata pe obiecte se folosesc deseori obiecte cu pointer catre un alt obiect din clasa proprie. Aceasta este structura standard pentru a creea o lista simplu înlantuita. Listele înlantuite de obiecte sunt studiate în cadrul temelor de proiect.