Programare Orientata pe Obiect
Lucrarea de laborator Nr. 4

Functii si clase friend. Supraîncarcarea operatorilor. Sistemul de intrare/iesire din C++.

Asa cum am vazut în lucrarea precedenta datele si functiile membre ale unei clase sunt grupate în sectiuni *private*, *protected* sau *publice*. Datele private continute într-o clasa pot fi folosite si modificate numai de catre functiile membre, ele neputând fi accesate din afara clasei. Datele publice pot fi accesate din afara clasei.

1. Functii si clase friend

O functie nemembra a unei clase poate accesa datele `private` sau `protected` ale acesteia daca este declarata functie de tip `friend` a clasei. Pentru declararea unei functii `f()` de tip `friend` a clasei `X` se include prototipul functiei `f()`, precedat de specificatorul `friend` în definitia clasei `X`, iar functia însasi se defineste în alta parte în program astfel:

```
class X{
      ..................
      friend tip_returnat f(lista_argumente);
      ..................
};
..................
tip_returnat f(lista_argumente){
      //corpul functiei
}
```

Exercitiul 1:

Se considera problema de înmultire a unei matrice cu un vector. Cele doua clase care descriu o matrice 4x4 si un vector de dimensiune 4 sunt definite astfel:

```
class Matrix {
      double m[4][4];
public: Matrix();
      Matrix(double pm[][4]);
};
class Vector{
      double v[4];
public: Vector();
      Vector(double *pv);
};
```

Implementati constructorii celor doua clase si definiti functia `multiply()` care înmulteste o matrice cu un vector ca functie `friend` în cele doua clase:

```
friend Vector multiply (const Matrix &mat, const Vector &vect );
```

Apelul acestei functii de înmultire în functia main a programului este urmatorul:

```
void main(){
      double v[] = {1,2,3,4};
      double m[][4] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
```

```
            Matrix m1(m);
            Vector v1(v);
            Vector v2 = multiply(m1,v1);
}
```

Este posibil ca o functie membra a unei clase sa fie declarata `friend` în alta clasa. De asemenea, este posibil ca o întreaga clasa `Y` sa fie declarata `friend` a unei clase `X` si, în aceasta situatie, toate functiile membre ale clasei `Y` sunt functii `friend` ale clasei `X` si pot accesa toate datele si functiile membre ale acesteia (inclusiv cele `private` sau `protected`).

Prin accesul liber la membrii privati ai unei clase al unei functii `friend` se deschide o bresa în scutul protector al clasei, astfel ca acest mecanism trebuie folosit rar si cu atentie. Nici constructorii si nici destructorii nu pot fi functii `friend`.

Exercitiul 2 :

Reluati exercitiul 7 din lucrarea precedenta, implementând functia care aduna obiectele vectorului ca functie `friend` a clasei `Complex`.

2. Supraîncarcarea operatorilor

Ati folosit în Lucrarea de laborator Nr.2 mecanismul de supraîncarcare a numelor de functii. Selectarea functiilor se realiza în functie de tipul si numarul parametrilor formali, iar tipul returnat nu era semnificativ. Ati folosit, de asemenea, în Lucrarea de laborator Nr.3 supraîncarcarea numelor de functii într-o clasa, atunci când defineati mai multi constructori, iar cel apelat era selectat prin numarul si tipul parametrilor din definitie. Deasemenea, ati folosit pâna acum un operator redefinit, fara sa va fi dat seama. Este operatorul `cout` care afiseaza datele în functie de variabila de intrare. Multe limbaje de programare folosesc supraîncarcarea astfel ca este posibila afisarea datelor de formate diferite prin acelasi nume de functie.

În limbajul C++ sunt definite putine tipuri de date ca date fundamentale, iar pentru reprezentarea altor tipuri de date necesare în diferite domenii (cum ar fi aritmetica numerelor complexe, algebra matricilor, etc.), se definesc clase care contin functii ce pot opera asupra acestor tipuri. Definirea operatorilor care sa opereze asupra obiectelor unei clase permite un mod mult mai convenabil de a manipula obiectele decât prin folosirea unor functii ale clasei. O functie care defineste pentru o clasa o operatie echivalenta operatiei efectuate de un operator asupra unui tip predefinit este numita *functie operator*. Majoritatea operatorilor limbajului C++ pot fi supraîncarcati (*overloading*), si anume:

```
new   delete      ()    []
+     -     *     /     %     ^     &     |     ~
!     =     <     >     +=    -=    *=    /=    %=
^=    &=    |=    <<    >>    >>=   <<=   ==    !=
<=    >=    &&    ||    ++    --    ,     ->*   ->
```

Operatorul `()` este apelul unei functii, iar operatorul `[]` este operatorul de indexare. Urmatorii operatori nu se pot supraincarca:

```
.   .* :: ?: sizeof
```

Nu pot fi supraîncarcati operatorii pentru tipurile predefinite ale limbajului. Functiile operator nu pot avea argumente implicite.

Functiile operator pentru o anumita clasa pot sa fie sau nu functii membre ale clasei. Daca nu sunt functii membre ele sunt, totusi, functii friend ale clasei ti trebuie sa aiba ca argument cel putin un obiect din clasa respectiva sau o referinta la aceasta. Exceptie fac operatorii `=, (), [], ->`, care nu pot fi supraîncarcati folosind functii friend ale clasei.

Forma generala a *functiilor operator membre ale clasei* este urmatoarea:

```
tip_returnat operator#(lista_argumente){
      // operatii
}
```

În aceasta forma generala semnul # reprezinta oricare dintre operatorii care pot fi supraîncarcati.

Exercitiu 3:

Pentru clasa `Complex` definita în lucrarea precedenta se pot defini mai multe operatii cu numere complexe: suma, diferenta, produsul a doua numere complexe. Aceste operatii se pot implementa prin supraîncarcarea corespunzatoare a operatorilor. Prima operatie este oferita ca exemplu:

```
Complex Complex::operator+( Complex op2){
      Complex temp;
      temp.re = re + op2.re;
      temp.im = im + op2.im;
      return temp;
}
void main(){
      Complex op1(10,20);
      Complex op2(30,40);
      Complex op3;
      op1.display();
      op2.display();
      op3 = op1 + op2;        //echivalent cu op3 = op1.operator+(op2);
      op3.display();    }
```

Supraîncarcati operatorii `-` si `*` pentru implementarea operatiilor diferenta si produs de numere complexe si reluati exercitiul 7 din lucrarea precedenta adunând obiectele din vector folosind operatorul supraîncarcat.

Pentru acelasi operator se pot defini mai multe functii supraîncarcate, cu conditia ca selectia uneia dintre ele în functie de numarul si tipul argumentelor sa nu fie ambigua. În implementarea prezentata, functia `operator+()` creaza un obiect temporar, care este distrus dupa returnare. În acest fel, ea nu modifica nici unul dintre operanzi, asa cum nici operatorul `+` pentru tipurile predefinite nu modifica operanzii. În expresia scrisa mai sus, `op3 = op1 + op2`, se executa, pe lânga operatia de adunare, `si` o operatie de asignare pentru o variabila de tip `Complex`. Pentru clasa `Complex`, aceasta operatie se executa corect, chiar daca nu a fost supraîncarcata functia `operator=()`, deoarece asignarea implicita se executa printr-o copiere membru cu membru si în acest caz nu produce erori.

La supraîncarcarea unui operator folosind o functie care nu este membra a clasei ca *functii operator friend* este necesar sa fie transmisi toti operanzii necesari, deoarece nu mai exista un obiect al carui pointer (`this`) sa fie transferat implicit functiei. Din aceasta cauza, functiile operator binar necesita doua argumente de tip clasa sau referinta la clasa, primul argument transmis este operandul stânga, iar al doilea argument este operandul dreapta.

Exercitiul 4:

Reluati exercitiul precedent implementând functiile operator ca functii friend ale clasei.

Referitor la supraîncarcarea operatorilor folosind functii nemembre ale clasei se mai pot face câteva observatii. Daca functia operator nu ar fi declarata functie friend a clasei, ea nu ar avea acces la variabilele protejate ale clasei. Problema s-ar putea rezolva prin adaugarea unor functii publice de citire si scriere a datelor membre ale clasei respective, care sa fie apelate în functia operator. Dar o astfel de solutie este incomoda, ineficienta si nu aduce nici un avantaj. O alta observatie referitoare la supraîncarcarea operatorilor folosind functii friend este aceea ca pentru operatorii care trebuie sa modifice operandul (cum sunt operatorii de incrementare, decrementare, complement, etc) este necesar transmiterea operandului ca parametru prin referinta, ceea ce permite modificarea lui în functia operatordupa cum se arata în exemplul urmator:

```
Complex Complex::operator++(){      //ca functie membra a clasei Complex
      re++;
      im++;
      return *this;

Complex operator++(Complex &p){     //ca functie friend a clasei Complex
      p.re++;
      p.re++;
      return p;
}
```

O functie `operator[]()` poate fi folosita pentru a defini o operatie de indexare pentru obiecte de tipuri definite de utilizator (clase). Argumentul functiei reprezinta al doilea operand al operatiei de indexare si este un indice. Acest argument poate fi orice tip de date, spre deosebire de indicii în tablouri care nu pot avea decat valori întregi. Primul argument al functiei este obiectul pentru care se executa operatia de indexare si pointerul la acesta (pointerul `this`) este transmis implicit functiei operator de asignare.

Exercitiul 5:

Sa se defineasca o clasa `Dreptunghi` care sa descrie un dreptunghi prin latime si înaltime, ca date membre private. Definiti constructorii de initializare si de copiere, destructorii, functii de acces la date, o functie de afisare a datelor dreptunghiului si functia operator de indexare care sa returneze referinta la latime pentru indice egal cu `0` si referinta la înaltime pentru indice egal cu `1`. Fie functia `void f(Dreptunghi d){Dreptunghi dr1(1,2);}`. Completati functia `f()` astfel încât sa creati în memoria libera (heap) un obiect prin copierea obiectului `d`, apoi dublati dimensiunile acestui dreptunghi folosind functia operator de indexare si afisati datele acestuia. Înainte de iesirea din functia `f()` eliberati zona de memorie ocupata. Apelati functia `f()` din functia `main()` cu argument un dreptunghi de dimensiuni `7,8`.

În cazul asignarii simple (=), valoarea expresiei care reprezinta operandul dreapta înlocuieste valoarea operandului stânga. Daca ambii operanzi sunt de tip aritmetic, operandul dreapta este convertit la tipul operandului stânga, dupa care are loc atribuirea valorii. În lipsa unei functii `operator=()` definita de utilizator pentru o clasa `X`, este utilizata definitia implicita de asignare prin copierea membru cu membru. În cazul claselor de obiecte care nu contin date alocate dinamic la initializare sau prin intermediul altor functii membre, asignarea prin copiere membru cu membru functioneaza corect si în general nu mai este necesar sa fie supraîncarcat operatorul de asignare. Situatia care apare pentru clasele care contin date alocate dinamic este asemanatoare celei întâlnita la constructorii de copiere: daca o clasa contine date alocate dinamic, copierea membru cu membru care se executa implicit la asignare sau la constructia prin copiere, are ca efect copierea pointerilor la datele alocate dinamic, deci doi pointeri din doua obiecte vor indica catre aceeasi zona din memoria heap. Acesta situatie poate conduce la numeroase erori.

Exercitiul 6:

Definiti o clasa `Stiva` ce contine un vector de numere întregi care se aloca dinamic în memoria libera la constructia unui obiect, dimensiunea stivei si un index de parcurgere a stivei, ca date membre private. Definiti functiile necesare, constructori, destructor, functii `push()` si `pop()` pentru introducerea, respectiv extragerea unui numar întreg din obiectul de tip Stiva pentru care sunt apelate astfel încât codul urmator sa ruleze corect:

```
void main()
{
Stiva s1(100);
Stiva s2(s1);
s1.push(11);
s1.push(12);
s2.push(21);
s2.push(22);
cout<<s1.pop()<<endl<<s1.pop()<<endl<<s1.pop()<<endl;
Stiva s3=s1;
s3=s2;
s3.display();
}
```

Observatie: La declararea unui obiect de clasa Stiva se transmite ca argument dimensiunea stivei, iar constructorul aloca spatiul necesar în memoria libera.

3. Sistemul de intrare/iesire din C++

În exercitiul de mai sus pentru a afisa elementele stivei a fost necesara crearea unei functii `display()` care sa afiseze continutul stivei element cu element. Deasemenea si pentru clasa `Complex` ati lucrat cu o functie `display()` care presupunea afisarea partilor reala si imaginara. Mult mai naturala ar fi parut o instructiune de forma: `cout<<c1;` sau `cout<<s1;` pentru afisarea la consola a numarului complex `c1` sau a continutului stivei `s1`.

Operatiile de I/O din C++ se efectueaza folosind functiile operator de insertie `<<` si operator de extragere `>>` din streamuri. Streamurile pot fi considerate ca recipienti de caractere aranjate într-o anumita ordine în care se introduce si din care se extrag date. Functiile de operare asupra streamurilor specifica modul în care se executa conversia între un sir de caractere din stream si o variabila de un anumit tip. Aceste functii operator sunt definite în clasa `ostream`, respective `istream`, pentru toate tipurile predefinite ale limbajului, iar pentru tipurile definite de utilizator ele pot fi supraîncarcate. Functiile de I/O pentru tipuri definite de utilizator se obtin prin supraîncarcarea operatorilor de insertie si de extragere, care au urmatoarea forma generala:

```
ostream& operator<<(ostream& os,tip_clasa nume){
            // corpul functiei
            return os;
      }
      istream& operator<<(istream& is,tip_clasa& nume){
            // corpul functiei
            return is;
      }
```

Primul argument al functiei este o referinta la streamul de iesire, respectiv de intrare. Pentru functia operator de extragere << al doilea argument este dat printr-o referinta la obiectul care trebuie sa fie extras din stream; în aceasta referinta sunt înscrise datele extrase din streamul de intrare. Pentru functia operator de insertie >> al doilea argument este dat prin tipul si numele obiectului care trebuie sa fie inserat, sau printr-o referinta la acesta. Functiile operator de insertie si extractie returneaza o referinta la streamul pentru care au fost apelate, astfel încât o alta operatie de I/O poate fi adaugata acestuia. Functiile operator << sau >> nu sunt membre ale clasei pentru care au fost definite, dar pot (si este recomandabil) sa fie declarate functii friend în clasa respectiva. Motivul pentru care o functie operator << sau >> nu poate fi membra al unei clase este simplu de observat: atunci când o functie operator de orice fel este membru al unei clase, operandul din stânga, care este un obiect din clasa respectiva, este cel care genereaza apelul si transmite implicit functiei pointerul this. Ori, în cazul functiilor operator << sau >> operandul din stânga trebuie sa fie un stream, nu un obict din clasa respectiva, deci aceste functii nu pot fi functii membre ale claselor.

Exercitiul 7:

Pentru clasa `Complex` definita în lucrarile precedente, supraîncarcati functiile operator >> si <<.

Se poate vorbi si despre dezavantaje ale redefinirii operatorilor. Supraîncarcarea poate pune probleme de folosire incorecta. Redefinirea operatorilor este disponibila numai pentru clase si nu se pot redefini operatori pentru tipuri standard. Acest lucru ar fi oricum dezavantajos pentru ca ar rezulta un cod foarte greu de înteles si citit. SI-ul logic (&&) si SAU-ul logic (||) pot fi redefiniti pentru noi clase dar nu vor opera în aceeasi maniera. Toti membrii constructiilor logice vor opera fara a tine cont de ce rezulta. Desigur, operatorii logici predefiniti vor continua sa opereze în mod normal spre deosebire de cei redefiniti. Daca operatorii de incrementare (++) sau decrementare (--) sunt redefiniti, sistemul nu are cum sa stie daca ei sunt folositi ca preincrement sau postincrement (respectiv predecrement sau postdecrement). Care din ele este folosita depinde strict de implementare.

Exercitiul 8:

Definiti o clasa `Dreptunghi` care sa descrie un dreptunghi în spatiul bidimensional, definit prin dimensiunile celor doua laturi, care sunt date membre de tip `private`. Definiti constructorii necesari, destructorul, functia operator de adunare care sa returneze un dreptunghi cu laturile egale cu suma laturilor celor doi operanzi, fara sa modifice operanzii initiali si functia operator de scriere în stream. În toti constructorii si destructorii definiti, adaugati o instructiune care sa afiseze la consola un mesaj care sa precizeze ce operatie se executa. Într-o functie oarecare creati un obiect dreptunghi cu laturile 3, 4, un alt obiect dreptunghi cu laturile 5,6 si calculati dreptunghiul care are laturile egale cu suma laturilor celor doua dreptunghiuri date.