**Lucrarea de laborator nr. 3**

**Tema: Supraincarcarea operatorilor**

**Scopul lucrării**: familiarizarea studenţilor cu noţiunile operatori supraîncărcaţi.

**Consideraţiile teoretice necesare**:

Atunci când creaţi o clasă, C++ vă permite să supraîncărcaţi operatorii pentru a lucra cu propriile clase.

Când supraîncărcaţi un operator, trebuie să continuaţi să utilizaţi operatorul în formatul său standard. De exemplu, dacă supraîncărcaţi operatorul plus (+), acesta trebuie să utilizeze operatorul sub forma ***operand+operand*** *.* Operatorul supraîncărcat creat se aplică numai instanţelor clasei specificate

Atunci când creaţi funcţii ***operator***membre pentru a supraîncărca funcţionarea unui operator, declaraţiile membrilor operatori vor avea forma generală prezentată mai jos:

**tip-return nume-clasa::operator #(lista-argumente)**

**{**

**// Operatii**

**}**

unde # este simbolul oricărui operator C++, exceptînd:

*.* – operator de membru al clase

\* – adresare la componenta prin pointer,

:: – operatorul de rezoluţie*,*

() ?: – operatorul condiţional*,*

operatorul *sizeof, etc.*.

Această definire se face în cadrul clasei, întocmai ca o funcţie membru.

Există două variante de definire a operatorilor:

* ca funcţie membru a clasei;
* ca funcţie prietenă a clasei.

O funcţie *operator* are aceleaşi componente pe care le are orice funcţie, include un nume, un tip returnat, argumente, corp şi, eventual, apartenenţa la o clasă.

Există trei elemente care trebuie stabilite la declararea operatorului, şi anume:

– este operator unar sau binar,

– este postfixat sau prefixat ca poziţie

– este funcţie membru sau nu – domeniu de acţiune.

***Funcţiile***operator ***membri******vor avea cu un argumen****t* ***mai puţin decît cele non-membri.***

Puteţi să supraîncărcaţi numai operatorii existenţi. C++ nu vă permite definirea unor operatori proprii.

Definirea operatorilor ca funcţii membri a unei clase prezintă o restricţie majoră: primul operand este obligatoriu să fie de tipul clasa respectiv.

În limbajul C++ supradefinirea operatorilor este supusă unui set de restricţii:

* nu este permis introducerea de noi simboluri de operatori;
* patru operatori nu pot fi redefiniţi (vezi mai sus);
* caracteristicile operatorilor nu pot fi schimbate:

1. pluralitatea (nu se poate supradefini un operator unar ca

operator binar sau invers),

2. precedenţa şi asociativitatea,

3. prioritatea lor;

* funcţia ***operator*** trebuie sa aibă ***cel******puţin***un parametru de tipul clasa căruia îi este asociat operatorul supradefinit.

Programatorul are libertatea de a alege natura operaţiei realizate de un operator, însă este recomandat ca noua operaţie să fie apropiată de semnificaţia iniţială.

Pot fi redefiniti operatorii din urmatoarea tabela:

**Tabelul 1.**

|  |  |  |
| --- | --- | --- |
| **Tipul operatorului** | **Simbolul operatorului** | **Observaţii** |
| **Binar** | **(); [] ;->;** | **Se definesc ca funcţii membri** |
| **Unar** | **+; - ;~; \*; & (tip)** |  |
| **Unar** | **++; - -** | **Nu se poate dis-tinge între pre- şi postfixare** |
| **Unar** | **new; delete** | **Poate fi supra-definit şi pentru o clasă** |
| **Binar** | **->; \*; /; %; +; -; &;|; &&; ||** |  |
| **Binar** | **<<; >> ;< ;<=; > ;>=; == ;!=** |  |
| **Binar** | **= ;+=; -= ;\*=; /=;%= ;&=; ^=;|= ;<<=; >>=** | **Se definesc ca funcţii membri** |
| **Binar** | **,** |  |

Pentru exemplificare, vom extinde clasa *Point* cu utilizarea unor operatori.

**class Point**

**{// ...**

**Point& operator += (Point p);**

**Point& operator -= (Point p);**

**Point operator + (Point p);**

**Point operator - (Point p);**

**Point& operator = (Point p);**

**int operator == (Point p);**

**int operator != (Point p);**

**int operator < (Point p);**

**int operator > (Point p);**

**int operator <= (Point p);**

**int operator >= (Point p);**

**};  
Point& Point::operator += (Point p)  
 {x += p.x; y += p.y; return \*this;}  
Point& Point::operator -= (Point p)  
 {x -= p.x; y -= p.y; return \*this;}  
Point Point::operator + (Point p)**

**{return Point(x + p.x, y + p.y);}  
Point Point::operator - (Point p)  
 {return Point(x -p.x, y -p.y);}  
int Point::operator == (Point p)  
 {return x == p.x && y == p.y;}  
int Point::operator != (Point p)  
 {return !(\*this == p);}  
int Point::operator < (Point p)  
 {return x < p.x && y < p.y;}  
int Point::operator > (Point p)  
 {return x > p.x && y > p.y;}  
int Point::operator <= (Point p)  
 {return x <= p.x && y <= p.y;}  
int Point::operator >= (Point p)**

**{return x >=p.x && y >= p.y;}**

Am utilizat mai sus varianta cu funcţii membri. Vom descrie implementarea operatorului *+* folosind cea de-a doua variantă.

**class Point {**

**// ...**

**friend Point operator + (Point p1, Point p2);  
};  
Point operator + (Point p1, Point p2)  
{return Point(p1.x + p2.x, p1.y + p2.y);}**

***1. Redefinirea operatorului =***

Operatorul **=** este deja predefinit în C++, pentru operanzi de tip clasă. Dacă nu este supradefinită, atribuirea se face membru cu membru în mod similar cu iniţializarea obiectului efectuată de către compilator. În caz de o atribuire specifică a clasei, operatorul **=** poate fi supradefinit:

**Point& Point::operator = (Point p)**

**{x = p.x; y =p.y; return \*this;}**

***2. Redefinirea operatorului +***

Următorul programcreează o clasă ***Sir*** şisupraîncarcă operatorul plus (+) astfel că acesta concatenează şirurile:

**#include <iostream.h>**

**#include <iomanip.h>**

**#include <string.h>**

**class Sir // Definirea clasei**

**{ public:**

**Sir \*operator +(char \*sir\_adaug); // operator**

**Sir (char \*in\_sir) // constructor**

**{ strcpy(buffer, in\_sir);**

**lungime = strlen(buffer); }**

**Sir(void) {lungime =0;};**

**void arata\_sir() { cout « buffer; };**

**private:**

**char buffer[256];**

**int lungime;**

**}; // Aici se termina clasa**

**Sir Sir::operator+(char \*sir\_adaug)**

**{**

**Sir temp;**

**int lungimeTemp;**

**lungimeTemp = strlen(buffer) + strlen(sir\_adaug) + 1;**

**if(lungimeTemp>256)**

**{**

**cout « "Sir prea lung!" « endl;**

**strcpy(temp.buffer, buffer);**

**return temp;**

**}**

**lungime = lungimeTemp;**

**strcpy(temp.buffer, buffer);**

**strcat(temp.buffer, sir\_adaug);**

**return temp;**

**}**

**void main(void)**

**{**

**Sir titlu("Autor ");**

**titlu = titlu +"Titlul carttii\n";**

**titlu.aratasir();**

**}**

Atunci când rulaţi programul *,* el va începe prin atribuirea membrului ***buffer***şirului ***„Autor".*** Programul va utiliza apoi operatorul ***plus*** supraîncărcat pentru a concatena caracterele ***„Titlul cartii".*** Observaţi că operatorul supraîncărcat este o funcţie simplă care primeşte un parametru. Funcţia primeşte numai un singur parametru. Parametrul este al doilea operand. Operaţia însăşi implică operandul instanţei.

Operatorul supraîncărcat ***plus***utilizează funcţiile ***strcpy***şi ***strcat***pentru a copia şirul de caractere dintre ghilimele în obiectul ***titlu****.* Observaţi că acest cod din cadrul funcţiei operator plus supraîncărcate se referă la datele membre ale obiectului ***titlu***în mod implicit, cu comenzi cum sunt următoarele, care plasează valoarea curentă a tidului în obiectul ***temp****:*

**strcpy (temp.buffer, buffer) ;**

Programul ar putea la fel de uşor să facă referire la obiect în mod explicit, utilizând pointerul ***this****,* ca mai jos:

**strcpy(temp.buffer, this.buffer);**

***3. Redefinirea operatorului de indexare[].***

C++ consideră operatorul de undexare ca pe un operator ***binar***pentru scopurile supraîncărcării. Prin urmare, forma generală a supraîncărcării unei funcţii membre operator este:

**tip numeClasa::operator[ ](int i)**

**{**

**// . . .**

**}**

Din punct de vedere tehnic, parametrul ***i*** nu trebuie neapărat să fie de tipul ***int****,* dar deoarece veţi defini de obicei matricele cu un parametru întreg, trebuie să evitaţi utilizarea unui parametru de tipul ***float***sau de alt tip. Atunci când apelaţi funcţia operator supraîncărcată, C++ va atribui pointerul ***this***la obiect şi va folosi parametrul pentru a controla dimensiunea. Pentru a înţelege mai bine prelucrările pe care le efectuează funcţia supraîncărcată [ ] pentru matrice, analizaţi următorul program:

**#include <iostream.h>**

**class TipOarecare**

**{**

**int a[3];**

**public:**

**TipOarecare(int i, int j, int k) // constructor**

**{**

**a[0] = i;**

**a[1] = j;**

**a[2] = k;**

**}**

**int operator[] (int i) {return a[i];}**

**};**

**void main(void)**

**{**

**TipOarecare ob(l, 2, 3);**

**cout << ob[l] ;**

**}**

Supraîncărcarea operatorului [ ] pentru matrice vă oferă posibilitatea de a controla mai bine crearea matricelor cu clase. Pe lângă faptul că vă permite să atribuiţi valori distincte pentru fiecare membru, puteţi utiliza funcţiile supraîncărcate pentru a crea un program care sî efectueze o indexare sigură a unei matrice. Indexarea sigură a unei matrice contribuie 1a prevenirea supradepăşirii sau subdepăşirii limitelor unei matrice în decursul execuţiei unu program. Următorul program extinde programul precedent pentru a realiza şi indexarea sigură a matricei:

**#include <iostream.h>**

**#include <stdlib.h>**

**class TipOarecare**

**{**

**int a[3];**

**public:**

**TipOarecare(int i, int j, int k)**

**{**

**a[0]=i;**

**a[1]=j;**

**a[2]=k;**

**}**

**int &operator[](int i);**

**};**

**int &tipoarecare::operator[] (int i)**

**{ if (i<0 || i>2 )**

**{**

**cout <<”eroare la margini. \n”;**

**exit (1);**

**}**

**return a[i];**

**}**

**};**

**void main (void)**

**{ TipOarecare ob(l, 2, 3) ;**

**cout <<ob[l]; cout <<endl;**

**ob[l] = 25;**

**cout <<ob[1]; cout <<endl;**

**ob[3] =44;**

**}**

Atunci când încercaţi să accesaţi un obiect de dincolo de marginile matricei, se va produce o eroare. În cazul programului*,* încercarea de a accesa elementul de la indicele 3 se face dincolo de margini şi prin urmare programul va returna o eroare. Atunci când executaţi programulel va genera următorul rezultat:

**2**

**25**

**Eroare la margini**

**C:\>**

***5. Exemplu numerelor complexe***

Să se scrie un program care să definească un nou tip de variabile - tipul complex şi să construiască funcţii adecvate pentru operaţiile de bază cu acest tip de date (adunare, scădere, înmulţire, calculul modulului numărului complex).

#include<iostream>

#include<math.h>

using namespace std;

class complex

{

public:

float x, y;

float modul(); //il folosim pentru a returna modulul

complex(); // vom initializa pe x si y cu 0

complex(float a, float b); //il folosim pentru a copia a in x si b in y

complex operator+(complex k);

complex operator-(complex k);

complex operator\*(complex k);

complex operator/(complex k);

};

complex::complex()

{

x=0;

y=0;

}

complex::complex(float a, float b) //constructor de copiere

{

x=a;

y=b;

}

//pentru adunarea a doua numere complexe folosim formula

// ... =x1+x2 +i(y1+y2)

complex complex::operator+(complex k)

{

complex h; //ne folosim de un obiect "auxiliar"

h.x=x+k.y;

h.y=y+k.y;

return h;

}

//pentru scaderea a doua numere complexe folosim formula

// ...=x1-x2+i(y1-y2)

complex complex::operator-(complex k)

{

complex h;

h.x=x-k.x;

h.y=x-k.y;

return h;

}

//pentru inmultirea a doua numere complexe folosim formula

// ...=x1x2-y1y2+i(x1y2+x2y1)

complex complex::operator\*(complex k)

{

complex h;

h.x=x\*k.x;

h.y=y\*k.y;

return h;

}

//pentru impartire folosim formula:

// ...=(x1x2+y1y1+i(x2y1-x1y2))/x2^2+x1^2

complex complex::operator/(complex k)

{

complex h;

float aux=k.x\*k.x+k.y\*k.y; //retine x^2+y^2

if(aux)

{

h.x=(x\*k.x+y\*k.y)/aux;

h.y=(k.x\*y-x\*k.y)/aux;

}

else

cout<<"impartire la 0!!";

return h;

}

float complex::modul()

{

return sqrt(x\*x+y\*y);

}

int main()

{

complex z1, z2, z3, z4(1, 0); //pentru z1, z2, z3 se apeleaza constructorul implicit care va initializa pe x si y pentru fiecare obiect cu 0.

//in cazul lui z4 se apeleaza cel cu 2 parametri

int n,i;

cout<<"n=";

cin>>n;

cout<<"z1="<<endl;

cin>>z1.x>>z1.y; //se citesc noi valori pentru x si y, atat in cazul lui z1,

cout<<"z2="<<endl;

cin>>z2.x>>z2.y; //cat si in cazul lui z2

z3=z1/z2; //z3 are x=0 si y=0, acestia se vor primi noi valori in urma operatiei de "/"

for(i=1; i<=n; i++)

z4=z4\*z3;

cout<<z4.x<<" "<<z4.y;

system("PAUSE");

return 0;

}

//obs: functiile obtinute prin supraincarcarea operatorilor au aceeasi prioritate si asociativitate ca operatorii respectivi

//functiile obtinute prin supaincarcarea operatorilor au aceeasi n-aritate ca si operatorii respecttivi, astfel:

//daca operatorul este unar si functia este unara, deci nu are parametri

//daca operatorul este binar si functia este binara, aceasta inseamna ca are un singur parametru.

**Sarcina pentru lucrările de laborator:**

**1.** Să se creeze clasa Vector – vector, utilizînd memoria dinamică. Să se definească operatorii "+" – adunarea element cu element a vectorilor, "-" – scăderea element cu element a vectorilor, "=" – atribuirea ca metode ale clasei. Să se definească operatorii de comparare: "==", "!=", "<", ">" ca funcţii prietene. Clasa trebuie să conţină toţi constructorii necesari şi destructorul cît și funcție de afișare.

**2.** Să se creeze clasa Matrix – matrice, utilizînd memoria dinamică. Să se definească operatorii "+" – adunarea matricilor, "-" – scăderea matricilor, "=" – atribuirea ca metode ale clasei. Să se definească operatorii de comparare: "==", "!=", "<", ">" ca funcţii prietene. Pentru realizarea ultimilor doi operatori să se definească funcţia de calcul a normei elementelor matricei. Să se definească operatorul "[]" astfel încît să fie cu sens adresarea [][] la elemente.

**3.** Să se creeze clasa Date – data, care conţine cîmpurile: ziua, luna, anul. Să se definească operatorii "+" şi "-" ca metode ale clasei, iar operatorii "++" şi "--" în ambele variante(prefixă şi postfixă) ca funcţii prietene. Operatorul "+" trebuie să permită realizarea operaţiilor numai cu variabilele de tip predefinit int (x=y+5). De prevăzut prelucrarea corectă a anilor bisecţi.

**4.**Să se creeze clasa Bool – variabile logice. Să se definească operatorii "+" – SAU logic, "\*" – ŞI logic, "^" – SAU EXCLUSIV, ca metode ale clasei, iar operatorii "==" şi "!=" – ca funcţii prietene. Operatorii trebuie să permită realizarea operaţiilor atît cu variabilele clasei date, cît şi cu variabilele de tip predefinit int. (Dacă numărul întreg este diferit de zero, se consideră că variabila este adevăr, altfel – fals.)

**5.**Să se creeze clasa Stack – stivă, utilizînd memoria dinamică. Să se definească următorii operatori ca metode ale clasei: "+" – de adunare a două stive, "=" – de atribuire, "()" – de returnare a unei stive noi care conţine ultimele n elemente ale stivei date. Să se definească operatorii de comparare: "==", "!=", "<", ">" ca funcţii prietene. Pentru realizarea ultimilor doi operatori să se definească funcţia de calcul a normei elementelor stivei. Să se definească operatorii "<<" şi ">>" pentru ieşiri/intrări de obiecte. Clasa trebuie să fie absolut funcţională, adică să conţină toţi constructorii necesari şi destructorul.

**6.** Scrieţi un program care efectuează următoarelor operaţii asupra unităţilor băneşti (de exemplu, lei, bani):

* adunarea,
* înmulţirea cu un coeficient,
* scăderea,
* împărţirea la un coeficient,
* valoarea numerică a unităţilor băneşti să se tipărească cu cuvinte.

Clasa trebuie să fie absolut funcţională, adică să conţină toţi constructorii necesari şi destructorul.

**7.** Scrieţi un program care efectuează următoarele operaţii asupra numerelor fracţionare:

* transformarea unui număr fracţionar compus într-o fracţie supraunitară,
* adunarea numerelor fracţionare,
* înmulţirea numerelor fracţionare,
* scăderea numerelor fracţionare,
* împărţirea numerelor fracţionare.

Clasa trebuie să fie absolut funcţională, adică să conţină toţi constructorii necesari şi destructorul.

**8.** Scrieţi un program care efectuează următoarele operaţii asupra unităţilor de lungime (de exemplu, metri, centimetri, milimetri):

* adunarea,
* înmulţirea cu un coeficient,
* scăderea,
* împărţirea la un coeficient,

Clasa trebuie să fie absolut funcţională, adică să conţină toţi constructorii necesari şi destructorul.

**9.** Scrieţi un program care efectuează următoarele operaţii asupra unităţilor de greutate (de exemplu, tone, kilograme, grame):

* adunarea,
* înmulţirea cu un coeficient,
* scăderea,
* împărţirea la un coeficient,

Clasa trebuie să fie absolut funcţională, adică să conţină toţi constructorii necesari şi destructorul.

**10.** Scrieţi un program care efectuează următoarele operaţii asupra unităţilor de timp (de exemplu, ceasul, minuta, secunda):

* adunarea,
* înmulţirea cu un coeficient,
* scăderea,
* împărţirea la un coeficient,
* valoarea numerică a unităţilor de timp să se tipărească cu cuvinte.

**11.** Scrieţi un program care ar defini clasa de bază string astfel ca ea să afişeze la display simbolurile prin codurile lor interioare. Să se efectueze următoarele operaţii asupra codurilor:

* să se determine spaţiul pentru stringul dat,
* să se compare două stringuri,
* să se extragă un substring din stringul dat,
* să se lichideze un substring din stringul dat din poziţia dată,
* să se inverseze un string dat.
* să se caute un substring din stringul dat.

**12.** Scrieţi un program care ar defini clasa mulţime astfel ca ea să afişeze la display valorile unei mulţimi. Să se efectueze următoarele operaţii asupra mulţimii:

* să se determine numărul de elemente a mulţimii
* să se determine spaţiul pentru mulţimea dată,
* să se compare două mulţimi,
* să se extragă o submulţime din mulţimea dată,
* să se adauge un nou element la mulţime,
* să se şteargă un element din mulţime,
* să se caute o submulţime din mulţimea dată.

Clasa trebuie să fie absolut funcţională, adică să conţină toţi constructorii necesari şi destructorul.

**13.** Să se creeze clasa 3-D de coordonate în spaţiu. Să se definească operatorii "+", "-", "=", ca metode ale clasei, iar operatorii de comparare – ca funcţii prietene. De prevăzut posibilitatea realizării de operaţii atît între coordonate cît şi între coordonate şi numere obişnuite. Clasa trebuie să fie absolut funcţională, adică să conţină toţi constructorii necesari şi destructorul.

**14.** Triplet de numere întregi

* adunarea,
* înmulţirea cu un coeficient,
* scăderea,
* împărţirea la un coeficient,

Clasa trebuie să fie absolut funcţională, adică să conţină toţi constructorii necesari şi destructorul.

**15.** Loc(Latitudine, longitudine)

* adunarea,
* înmulţirea cu un coeficient,
* scăderea,
* împărţirea la un coeficient,

Clasa trebuie să fie absolut funcţională, adică să conţină toţi constructorii necesari şi destructorul.