Lab 5 – P2

Cuprins

1. Supraîncărcarea operatorilor

Limbajul C++ permite ca acțiunea operatorilor să fie redefinită pentru noi tipuri de date.

Pentru a supraîncărca un operator care să poată fi aplicat obiectelor, trebuie să definim funcțiile `operator`, care pot avea domeniu local (declarate în cadrul clasei) sau global (declarate și definite în afara clasei). Funcțiile `operator` sunt funcții ale căror nume este format din cuvântul cheie **`operator`** urmat de simbolul operatorului. De exemplu: `tip operator+(...).`

La fel ca în cazul operatorilor aplicați tipurilor fundamentale, tipul operatorilor supraîncărcați (unari, binari), precedența lor și asociativitatea lor se vor păstra și pentru variantele supraîncărcate de către utilizator.

Operatorii +, -, *, & au două forme, una unară și una binară, ambele putând fi supraîncărcate.

Echivalențele `++var;` ⇔ `var+=1;` ⇔ `var=var+1;` nu mai au loc și pentru tipurile abstracte.

Supraîncărcarea operatorilor se poate realiza prin:

- Utilizarea funcțiilor membre
- Utilizarea funcțiilor globale

Expression	Operator	Member function	Global function
`@a`	`+ - * & ! ~ ++ --`	`A::operator@()`	`operator@(A)`
`a@`	`++ --`	`A::operator@(int)`	`operator@(A, int)`
`a@b`	`+ - * / % ^ & | < > == != <= >= << >> && || ,`	`A::operator@(B)`	`operator@(A, B)`
`a@b`	`= += -= *= /= %= ^= &= |= <<= >>= []`	`A::operator@(B)`	-
`a(b, c...)`	`()`	`A::operator()(B, C...)`	-
`a->x`	`->`	`A::operator->()`	-

unde a este un obiect de tipul A, b este un obiect de tipul B și c este un obiect de tipul C.

- `a@b` poate fi apelat și ca `a.operator @(b)`
- Operatorii care **NU** pot fi supraîncărcați sunt:

`. :: ?: sizeof`

2. Exemplu

Fie următorul exemplu:

Data.h

```
class Data
{
	int zi, luna, an;
public:
	void afisare();
	Data(int z, int l, int a);
};
```

Data.cpp

```
void Data::afisare()
{
	std::cout << "Data calendaristica este " << zi << "-" << luna << "-"
<< an << std::endl;
}
Data::Data(int z, int l, int a)
{
	// std::cout << "S-a apelat:Data(...)" << std::endl;
	zi = z;
	luna = l;
	an = a;
}
```

Test.cpp

```
int main()
{
	Data d1(3,4,2000);
	Data d2(1,2,2008);
	Data d3;

	d3 = d2 + d1; // eroare

	return 0;
}
```

În urma adunării celor două obiecte `d2` și `d1` va rezulta o eroare deoarece nu a fost definit operatorul + pentru clasa `Data`.

```
error C2784: 'std::_String_iterator<_Elem,_Traits,_Alloc> std::operator +
(_String_iterator<_Elem,_Traits,_Alloc>::difference_type,
std::_String_iterator<_Elem,_Traits,_Alloc>)' : could not deduce template argument
for 'std::_String_iterator<_Elem,_Traits,_Alloc>' from 'Data'
```

2.1. Supraîncărcarea operatorilor prin funcții membre

În clasa Data (fișierul .h) se adaugă declarația operatorului de adunare `Data operator+(Data);`, iar înafara clasei (în fișierul .cpp) se adaugă definiția:

Data.h

```
class Data
{
	int zi, luna, an;
public:
	void afisare();
	Data(int z, int l, int a);
	Data operator+(Data);
};
```

Data.cpp (de adăugat)

```
…
```

```
Data Data::operator+(Data ad)
{
	Data temp;
	temp.zi = zi + ad.zi;
	temp.luna = luna + ad.luna;
	temp.an = an + ad.an;
	//de completat cu conditii pentru depistarea eventualelor transporturi
	return temp;
}
```

2.2. Supraîncărcarea operatorilor prin funcţii globale

În fișierul .cpp principal se adaugă definiția operatorului de adunare `Data operator+ (Data a, Data b)`, iar în clasă se adaugă trei funcții get pentru accesarea membrilor privați din clasă (vezi exemplul de la curs):

Data.h

```
class Data
{
	int zi, luna, an;
public:
	void afisare();
	Data(int z, int l, int a);
	Data();
};
```

Test.cpp (de adăugat codul)

```
…
Data operator+ (Data a, Data b)
{
	Data temp(a.getZi() + b.getZi(),a.getLuna() + b.a.getLuna(),a.getAn() +
b.getAn());
	return temp;
}
```

Observație: Dacă se declară atât o funcție membră cât și o funcție globală pentru supraîncărcarea aceluiași operator, compilatorul va semnala eroare:

```
error C2593: 'operator +' is ambiguous could be 'Data Data::operator +(Data)'
or 'Data operator +(Data,Data)'
```

3. Operatori impliciţi

Compilatorul are definite variante implicite ale lui `operator=` (atribuire) și `operator&` (adresa lui). Pentru folosirea acestor operatori în scopul lor clasic nu trebuiesc supraîncărcaţi operatorii = și &. Se pot supraîncărca acești operatori dar se recomandă să nu se schimbe semnificaţia lor

Test.cpp (de adăugat codul în funcția main)

```
…
Data *d4 = &d2;
d4->afisare();
…
```

Se va afișa:

```
Data calendaristica este 1-2-2008
```

4. Supraîncărcarea operatorului ++

Operatorul ++ are două forme: pre-fixată (++a) și post-fixată (a++).

Funcțiile trebuie să asigure:

- Modificarea obiectului pentru care s-a făcut apelarea.
- Returnarea unui obiect cu membrii modificați corespunzător, pentru a asigura funcționarea apelurilor de genul: `b = a++`, unde `b` este tipul de `return` și `a` este obiectul apelat.

Data.h

```
class Data
{
...
public:
...
	Data operator++(); //forma pre-fixata
	Data operator++(int); //forma post-fixata
};
```

Data.cpp (de adăugat codul)

```
…
//forma pre-fixata
Data Data::operator++()
{
	zi++;
	//conditii pentru transporturi
	return *this;
}
//forma post-fixata
Data Data::operator++(int)
{
	Data temp = *this;
	//conditii pentru transporturi
	zi++;
	return temp;
}
```

Test.cpp (de adăugat codul în funcția main)

```
…
std::cout << "supraincarcarea operatorului ++ pre-fixat" << std::endl;
Data d5 (3,4,2000);
++d5;
d5.afisare();
//
std::cout << "supraincarcarea operatorului ++ post-fixat" << std::endl;
Data d6 = ++d1;
d6.afisare();

std::cout << "d7:" << std::endl;
Data d7 = d2++; // Data d2 (1,2,2008);
d7.afisare();
d2.afisare();
…
```

Se va afișa:

```
supraincarcarea operatorului ++ pre-fixat
Data calendaristica este 4-4-2000
supraincarcarea operatorului ++ post-fixat
Data calendaristica este 4-4-2000
d7:
Data calendaristica este 1-2-2008
Data calendaristica este 2-2-2008
```

5. Exerciţii

1. Testați exemplele din laborator și adăugați condițiile pentru depistarea eventualelor transporturi de la operatorii + și ++ (formele pre-fixată și post-fixată).

2. Să se implementeze o clasă Matrice cu următorii membri:
   a. `nrL` și `nrC` de tipul `int`, care să reprezinte numărul de linii, respectiv coloane, din matrice;
   b. `elemente` de tipul `int[10][10]` care să conțină elementele matricei.

   Să se completeze clasa cu următoarele metode:
   - O metodă de citire a unei matrice de la tastatură (se citesc inclusiv numărul de linii și de coloane).
   - O metodă de `afisare` a elementelor matricei.
   - Un destructor pentru afișarea unui mesaj.
   - Supraîncărcarea operatorilor (a și b sunt două obiecte de tipul Matrice, x este de tipul int):
     - a + b – adunarea a două matrice;
     - a – b – scăderea a două matrice;
     - a * b – înmulțirea a două matrice;
     - a == b – testarea egalității a două matrice;
     - a + x – adunarea fiecărui element din matrice cu valoarea x;
     - x + a – adunarea lui x la fiecare element din matrice;
     - ~a – inversarea elementelor din matrice față de diagonala principală;
     - ++a și a++ – incrementarea elementelor de pe diagonala principală;
     - a[x] – returnarea elementului din matrice de pe linia x/nrC și coloana x%nrC.

   Testați toate funcțiile implementate.