

Lab 5 – P2

Cuprins

1. Supraîncărcarea operatorilor.....	1
2. Exemplu	2
3. Operatori implicați.....	3
4. Supraîncărcarea operatorului ++.....	4
5. Exerciții	5

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;` $\Leftrightarrow$ `var+=1;` $\Leftrightarrow$ `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
<pre>class Data { unsigned char zi, luna; int an; public: void afisare(); Data(unsigned char z, unsigned char l, int a); };</pre>
Data.cpp
<pre>void Data::afisare() { cout << "Data calendaristica este " << (int)zi << "-" << (int)luna << "-" << an << endl; } Data::Data(unsigned char z, unsigned char l, int a) { // cout << "S-a apelat:Data(...)" << endl; zi = z; luna = l; an = a; }</pre>
Test.cpp
<pre>int main() { Data d1(3,4,2000); Data d2(1,2,2008); Data d3; d3 = d2 + d1; // eroare return 0; }</pre>

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
<pre>class Data { unsigned char zi, luna; int an; public: void afisare(); Data(unsigned char z, unsigned char l, int a); Data operator+(Data); };</pre>

Data.cpp (de adăugat)
<pre>... 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; }</pre>

2.2. Supraîncărcarea operatorilor prin funcții globale

În fișierul .cpp principal se adaugă definiția operatorului de adunare `Data operator+ (Data a, Data b)`, iar în clasă se modifică membrii ca fiind publici pentru a putea fi accesați:

Data.h
<pre>class Data { public: unsigned char zi, luna; int an; public: void afisare(); Data(unsigned char z, unsigned char l, int a); Data(); };</pre>
Test.cpp (de adăugat codul)
<pre>... Data operator+ (Data a, Data b) { Data temp; temp.zi = a.zi + b.zi; temp.luna = a.luna + b.luna; temp.an = a.an + b.an; //de completat cu conditii pentru depistarea eventualelor transporturi return temp; }</pre>

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 impliciți

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

Test.cpp (de adăugat codul în funcția main)
<pre>... Data *d4 = &d2; d4->afisare(); ...</pre>
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
<pre>class Data { unsigned char zi, luna; int an; public: void afisare(); Data(unsigned char z, unsigned char l, int a); Data operator++(); //forma pre-fixata Data operator++(int); //forma post-fixata };</pre>
Data.cpp (de adăugat codul)
<pre>... //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; }</pre>
Test.cpp (de adăugat codul în funcția main)
<pre>... cout<<"supraincercarea operatorului ++ pre-fixat"<<endl; Data d5 (3,4,2000); ++d5; d5.afisare(); // cout<<"supraincercarea operatorului ++ post-fixat"<<endl; Data d6 = ++d1; d6.afisare(); cout<<"d7:"<<endl; Data d7 = d2++; // Data d2 (1,2,2008); d7.afisare(); d2.afisare(); ...</pre>
Se va afișa: supraincercarea operatorului ++ pre-fixat Data calendaristica este 4-4-2000 supraincercarea operatorului ++ post-fixat Data calendaristica este 4-4-2000 d7: Data calendaristica este 1-2-2008 Data calendaristica este 2-2-2008

5. Exerciții

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**`, 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 dealocarea zonelor de memorie alocate dinamic.
- Supraîncărcarea operatorilor (a și b sunt două obiecte de tipul `Matrice`, x este de tipul `int`):
 - o `a + b` – adunarea a două matrice;
 - o `a - b` – scăderea a două matrice;
 - o `a * b` – înmulțirea a două matrice;
 - o `a == b` – testarea egalității a două matrice;
 - o `a + x` – adunarea fiecărui element din matrice cu valoarea x;
 - o `x + a` – adunarea lui x la fiecare element din matrice;
 - o `~a` – inversarea elementelor din matrice față de diagonala principală;
 - o `++a` și `a++` – incrementarea elementelor de pe diagonala principală;
 - o `a[x]` – returnarea elementului din matrice de pe linia `x/nrC` și coloana `x%nrC`.

Testați toate funcțiile implementate.