Supraîncărcarea operatorilor

Curs 4

Curs anterior

- Cuvântul cheie this
- Funcţii/clase prietene
- Membri statici
- Modificatori de acces
- Relaţii între clase
  - Dependință
  - Asociere
  - Agregare
  - Compoziţie
  - Moştenire

Cuprins

- Supraîncărcarea operatorilor
  - Funcţii membre
  - Funcţii prietene
- Supraîncărcarea operatorilor
  - Asignare
  - Binari
  - Prescurtaţi
  - Relaţionali
  - Incrementare/decrementare
  - Indexare

CE ESTE SUPRAÎNCĂRCAREA OPERATORILOR?

- Supraîncărcarea operatoriilor
  - Permite definirea comportamentului operatorilor când sunt aplicaţi obiectelor unui nou tip de date (clase)
- Ce operatori trebui supraîncărcaţi?
  - Cei care au sens pentru noul tip de date definit
- Restricţii?

CE ESTE SUPRAÎNCĂRCAREA OPERATORILOR?

- Supraîncărcarea operatoriilor
  - Permite definirea comportamentului operatorilor când sunt aplicaţi obiectelor unui nou tip de date (clase)
- Ce operatori trebui supraîncărcaţi?
  - Cei care au sens pentru noul tip de date definit
- Restricţii?
  - Nu ne permit să schimbăm „înţelesul” operatorilor când sunt aplicaţi asupra tipurilor de bază
  - Nu ne permite definirea de noi simboluri pentru operatori
  - Unul dintre operanzi trebuie să fie de tipul clasei pentru care s-a supraîncărcat operatorul

SUPRAÎNCĂRCAREA OPERATORILOR

- Similară cu supraîncărcarea funcţiilor
  - Numele funcţiei este înlocuit de cuvântul cheie *operator* urmat de simbolul operatorului
  - Tipul de return reprezintă tipul valori care va fi rezultatul operaţiei
  - Argumentele sunt 1 sau 2 operanzii în funcţie de n-aritatea operatorului

SUPRAÎNCĂRCAREA OPERATORILOR

- Exemplu
  - Numerele complexe

```
class complex{
 public:
   friend complex& operator+ (const complex&, const complex &);
 ...
 private:
 double _real;
 double _imag;
};
int main(){
   complex c1, c2(4,5);
   complex c3 = c1+c2;
   cout << c3;
}
```

SUPRAÎNCĂRCAREA OPERATORILOR

- Nu se poate modifica
  - n-aritatea
  - Asociativitatea
  - Prioritatea
  - Nu se pot utiliza valori implicite (default)

- Observaţii
  - Nu este bine să modificăm sensul operatorilor
    - + sa nu însemne scădere
  - Oferim definiţi consistente
    - Dacă + este definit, atunci şi += ar trebui definit

SINTAXĂ

- Definire funcţie cu numele
  - tip_returnat operator simbol (lista_de_parametri);
  - simbol orice operator C++ cu excepţia
    - .
    - .*
    - ::
    - ?:
    - sizeof
- Invocare funcţii operator
  - Operatori unari
    - Utilizare: !a, ++b4
  - Operatori binari
    - Utilizare: a+b, c<=d

CARE OPERATORI POT FI SUPRAÎNCĂRCAȚI?

- Aproape toţi operatori
  - Aritmetici
    - +, -, *, /, %, ++, --
  - Operatori pe biţi
    - ^, &, >>, <<
  - Operatori logici
    - &&, ||, !
  - Operatori relaţionali
    - >, <, <=, >=, ==, !=
  - Operatorul de indexare, funcţie, virgulă
    - [] () ,
  - Operatorul de atribuire şi cei compuşi
    - =, ^=, |=, &=, +=, -=, *=, /=, %=, <<=, >>=,
  - Supraîncărcaţi global de către compilator
    - new, delete, =, &, ->*, ->

MODURI DE SUPRAÎNCĂRCARE

- Cum se mapează operatorii la funcţii?
  - Ex. utilizare a+b
- Soluţii
  - Ca funcţii prietene ale clasei
    - friend complex operator+ (const complex &a, const complex &b);
  - Ca funcţii membre ale clasei
    - complex operator+ (const complex &otherObj) const;

MODURI DE SUPRAÎNCĂRCARE

Ca funcţii prietene ale clasei

- Definire

```
complex operator + (const complex&
a, const complex& b)
{
    return complex(a._real+b._real,
                   a._imag+b._imag);
}
```

- Apel

```
complex x,y;
complex z = x+y;
```

Ca funcţii membre ale clasei

- Definire

```
complex complex::operator + (const
complex& altObj) const
{
    return complex(_real+ altObj._real,
                   _imag+ altObj._imag);
}
```

- Apel

```
complex x,y;
complex z = x+y;
```

MODURI DE SUPRAÎNCĂRCARE

- Operatori care trebuie supraîncărcați ca funcții membre sunt
  - = operatorul de atribuire
  - [] opratorul de indexare
  - () operatorul de apel al unei funcții
  - -> operatorul de acesare indirectă a unui membru
  - ->* operatorul de acesare indirectă a unui membr pointer

OPERATORI SUPRAÎNCĂRCAȚI ÎN MOD UZUAL

- Uni dintre operatori supraîncărcaţi în mod uzual sunt
  - <<, >>
  - =
  - <

SUPRAÎNCĂRCAREA OPERATORULUI =

- În lipsa supraîncărcări compilatorul generează o copiere membru cu membru a datelor clasei
- Comportament similar cu constructorul de copiere
- Operatorul de atribuire trebuie să fie supraîncărcat ca funcţie membră, nu modifică al doilea operand (ar trebuie să fie o referinţă constantă)
- Operatorul de asignare poate fi înlănţuit, deci ar trebui să întoarcă o referinţă
- Modifică obiectul curent, deci nu poate fi funcţie non-membră

SUPRAÎNCĂRCAREA OPERATORULUI =

```
class Student{
  char * nume;
   ...
  Student & operator = (const Student &);    //operator atribuire
  ...
};
```

```
Student & Student::operator = (const Student & s2) {
   if (this == &s2)
         return *this; //verificare autoreferinţă
   if (nume){
         delete []nume; //ştergere valoare existentă
         nume = new char[strlen(s2.nume)+1];
         strcpy(nume, s2.nume);
   }
 return *this;
}
```

SUPRAÎNCĂRCAREA OPERATORULUI =

- Observaţie
  - Dacă o clasă conţine variabile membru de tip pointer
  - următoarele funcţii trebuie implementate
    - Constructor
    - Constructor de copiere
    - Destructor
    - Supraîncărcarea operatorului =

SUPRAÎNCĂRCAREA OPERATORILOR << ŞI >>

- Operatorii << şi >> se supraîncarcă pentru a putea insera în fluxurile de ieşire şi extrage din fluxurile de intrare
- Biblioteca *iostream* suprascrie operaţiile de scriere/citire pentru tipurile implicite
- Pentru noile tipuri de date definite de utilizatori aceştia sunt responsabili cu suprascrierea lor
- Nu pot fi supraîncărcaţi ca funcţii membre, trebuie utilizate funcţii non-membere, deoarece primul operand este un obiect de tipul ostream/istream şi nu o referinţă la clasă

SUPRAÎNCĂRCAREA OPERATORILOR << ŞI >>

Prototip

ostream &operator<<(ostream &, const complex &);

istream &operator>> (istream &, complex &)

Valoare de return are acelaşi tip cu primul operand

Primul operand pot fi stremurile cin sau cout sau fişiere

Citirea modifică al doilea operand

Utilizare

```
complex c1(7,9);
cin >> c1;
cout << c1 << endl;
```

SUPRAÎNCĂRCAREA OPERATORILOR << ŞI >>

- Recomandare
  - Nu introduceţi mesaje şi formatare când supraîncărcaţi operatorii << şi >>

```
ostream &operator<<(ostream &out, const complex &c)
{
   out << " Complex[" << c._real << "+" << c._imag << "i]";
   return out;
}
```

```
istream &operator<<(istream &in, complex &c)
{
    cout << "Introduceti partea reala"; in >> c._real;
    cout << "Introduceti partea imaginara"; in >> c._imag;
    return in;
}
```

SUPRAÎNCĂRCAREA OPERATORILOR + ŞI +=

- Pentru consistenţă dacă supraîncărcăm operatorul + este bine să supraîncărcăm şi operatorul +=

```
class complex{
public:
      friend complex operator+ (double &, const complex &);
      friend complex operator+ (const complex &, double &);
      friend complex operator+ (const complex &, const complex &);
      complex & operator += (double &);
      complex & operator += (const complex &);
      …
};
complex operator+ (const complex &c1, const complex &c2) {
       return complex(c1._real+c2._real, c1._imag+c2._imag);
}
complex& complex ::operator+= (const complex &c) {
      _real += c._real;
      _imag += c._imag;
      return *this;
}
```

Cum am putea implementa altfel supraîncărcarea oprratorului +=?

SUPRAÎNCĂRCAREA OPERATORILOR + ŞI +=

- Pentru consistenţă dacă supraîncărcăm operatorul + este bine să supraîncărcăm şi operatorul +=

```
class complex{
public:
      friend complex operator+ (double &, const complex &);
      friend complex operator+ (const complex &, double &);
      friend complex operator+ (const complex &, const complex &);
      complex & operator += (double &);
      complex & operator += (const complex &);
      ...
};
complex operator+ (const complex &c1, const complex &c2) {
       return complex(c1._real+c2._real, c1._imag+c2._imag);
}
complex& complex ::operator+= (const complex &c) {
      _real += c._real;
      _imag += c._imag;
      return *this;
}
```

```
complex& operator+= (const complex &c)
{
      return *this = *this + c;
}
```

SUPRAÎNCĂRCAREA OPERATORILOR + ŞI +=

- Pentru consistenţă dacă supraîncărcăm operatorul + este bine să supraîncărcăm şi operatorul +=

```
class complex{
public:
      friend complex operator+ (double &, const complex &);
      friend complex operator+ (const complex &, double &);
      friend complex operator+ (const complex &, const complex &);
      complex & operator += (double &);
      complex & operator += (const complex &);
      …
};
complex operator+ (const complex &c1, const complex &c2) {
       return complex(c1._real+c2._real, c1._imag+c2._imag);
}
complex& complex ::operator+= (const complex &c) {
      _real += c._real;
      _imag += c._imag;
      return *this;
}
```

Cum arata supraîncărcarea operatorului + folosind funcţii membre?

SUPRAÎNCĂRCAREA OPERATORILOR + ŞI +=

- Pentru consistenţă dacă supraîncărcăm operatorul + este bine să supraîncărcăm şi operatorul +=

```
class complex{
public:
      friend complex operator+ (double &, const complex &);
      friend complex operator+ (const complex &, double &);
      friend complex operator+ (const complex &, const complex &);
      complex & operator += (double &);
      complex & operator += (const complex &);
      ...
};
complex operator+ (const complex &c1, const complex &c2) {
       return complex(c1._real+c2._real, c1._imag+c2._imag);
}
complex& complex ::operator+= (const complex &c) {
      _real += c._real;
      _imag += c._imag;
      return *this;
}
```

Funcţia este constanta pentru a nu modifica valoarea obiectului curent

```
complex complex::operator+ (const complex &c2) const {
      return complex(_real+c2._real, _imag+c2._imag);
}
```

SUPRAÎNCĂRCAREA OPERATORILOR RELAȚIONALI

- Recomandare
  - Supraîncărcați ca funcţii non-membre, deoarece primul operand ar putea fi o valoare de tip diferit faţă de clasă
  - Valorile operanzilor nu se modifică, deci ar trebui să fie referinţe constante
  - Valoarea de return ar trebui să fie de tip bool
  - Dacă sunt specificaţii ca funcţii membre, ele trebuie să fie constante

SUPRAÎNCĂRCAREA OPERATORILOR RELAȚIONALI

```
class complex{
public:
  friend bool operator< (double &, const complex &);
  friend bool operator< (const complex &, double &);
  friend bool operator< (const complex &, const complex &);
  friend bool operator== (double &, const complex &);
  friend bool operator== (const complex &, double &);
   friend bool operator== (const complex &, const complex &);
...
};
bool operator< (const complex &c1, const complex &c2) {
     return (modul(c1)<modul(c2));
}
```

SUPRAÎNCĂRCAREA OPERATORILOR ++ ŞI --

- Operatorii ++ şi -- pot fi utilizaţi în două moduri prefixaţi şi postfixaţi
- Recomandare
  - Ar trebui definiţi ca funcţii membre
- Prefixat

```
Contor & Contor ::operator ++ () { .... //corp}
Contor & Contor ::operator --() { .... //corp}
```

- Postfixat

```
Contor Contor ::operator ++ (int a) { .... //corp}
Contor  Contor ::operator --(int a) { .... //corp}
```

SUPRAÎNCĂRCAREA OPERATORILOR ++ ȘI --

- Exemplu

```
class X
{
   X& operator++()
   {
      // codul de incrementare
      return *this;
   }
   X operator++(int)
   {
      X tmp(*this); // copie
      operator++(); // pre-increment
      return tmp;   // returnarea vechi valori
   }
};
```

SUPRAÎNCĂRCAREA OPERATORULUI []

- Recomandări
  - Ar trebui supraîncărcată ca funcţie membră, primul operand ar trebui să fi un membru al clasei
  - Pentru consistentă al doilea operator ar trebui să fie de tip întreg (transmis prin valoare)
  - Deoarece starea obiectului nu se modifică funcţia ar trebui să fie constantă
  - Valoarea de return ar trebui să fie de tipul tabloului de elemente conţinut de clasă

SUPRAÎNCĂRCAREA OPERATORULUI []

```
class string {
    char*str;
    int len;
public:
    char & operator [] (int) const;
    •••
};

char & string::operator [] (int index) const {
    return str[index];
}
```

Test curs. Q1

- Supraîncărcați operatorul + pentru clasa string.

```
class string {
    char*str;
    int len;
public:
    …
};
```

Test curs. Q2

- Care dintre următori operatori este suprascris de compilator automat dacă nu e suprascris se utilizator
  1. Operatorul de comparare (==)
  2. Operatorul de asigmare (=)

A. doar varianta 1
B. doar varianta 2
C. ambele variante 1 și 2
D. nici o variantă

Test curs. Q3

- În cazul supraîncărcării operatorilor, funcția operator trebuie să fie
  1. funcție membră statică
  2. funcție membră
  3. funcție prietenă clasei

a) Doar varianta 2
b) Variantele 1 și 3
c) Variantele 2 și 3
d) Toate variantele 1, 2 și 3

CURS URMĂTOR

- Conversii de tip
- Excepţii