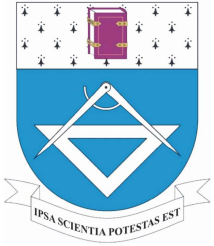


Universitatea Tehnică „Gheorghe Asachi” din Iași
Facultatea de Automatică și Calculatoare

Programarea Calculatoarelor II

Curs 4

Pointerul this.

Supraîncărcarea operatorilor. Cuvântul cheie static.

Pointerul `this`

- Pointerul `this` este o variabilă predefinită în C++ accesibilă în corpul oricărei metode non-statice din cadrul unei clase
- Valoarea pointerului este dată de adresa obiectului pentru care s-a apelat o anume metodă non-statică din clasă
- Este folosit:
  - Pentru a înlătura ambiguitățile dintre un parametru al unei funcției și o variabilă membră
  - În cazurile când este necesar un pointer către obiectul pentru care s-a apelat o anumită metodă

Pointerul `this`

- Compilatorul C++ convertește apelul funcției non-statice apelate și pune ca prim parametru pointerul `this`:
- Funcția:

```
void Data::afisare()
{
  std::cout << "Data este" << zi << "-" << luna << "-
      " << an << std::endl;
}
```

- devine:

```
void Data::afisare(Data * this)
{
   std::cout << "Data este" << this->zi << "-"
      << this->luna << "-" << this->an << std::endl;
}
```

Pointerul `this` (evitarea ambiguitatilor)

```
class A
{
        int nr;
public:
        int getNr();
        A(int nr);
};
A::A(int nr)
{
        nr=nr;
}
int A::getNr()
{
        return nr;
}
void main ()
{
        A var1 (5);
        cout << var1.getNr();
}
```

-858993460

Pointerul `this` (evitarea ambiguitatilor)

```
class A
{
        int nr;
public:
        int getNr();
        A(int nr);
};
A::A(int nr)
{
        this->nr=nr;
}
int A::getNr()
{
        return nr;
}
void main ()
{
        A var1 (5);
        cout << var1.getNr();
}
```

Supraîncărcarea operatorilor complex.h

```
class Complex
{
    int re, im;
public:
    Complex();
    Complex(int, int);
    void Citeste();
    void Afiseaza();
};
```

Supraîncărcarea operatorilor

```
#include <iostream>
#include "complex.h"

Complex::Complex()
{
    re = 0;
    im = 0;
}

Complex::Complex(int r, int i)
{
    re = r;
    im = i;
}
```

```
void Complex::Citeste()
{
    std::cout << "re = ";
    std::cin >> re;
    std::cout << "im = ";
    std::cin >> im;
}

void Complex::Afiseaza()
{
        std::cout << re << " +
" << im << "i" << std::endl;
}
```

Supraîncărcarea operatorilor

```
#include "complex.h"
int main()
{
    Complex c1(1, 2), c2(3, 4), c3;
    c1.Afiseaza();
    c2.Afiseaza();

    c3 = c2 + c1;
    c3.Afiseaza();

    return 0;
}
```

```
Error C2676 binary '+': 'Complex' does not define this
operator or a conversion to a type acceptable to the
predefined operator
```

Supraîncărcarea operatorilor

- Folosirea *funcțiilor operator* având ca scop extinderea folosirii operatorilor care se pot aplica tipurilor fundamentale și către obiecte
- *Funcțiile operator:*
  - nu sunt funcții obișnuite
  - ele apar în expresii, iar compilatorul se supune unui set de reguli privind interpretarea și apelul acestora
- **Tipul operatorilor** supraîncărcați **(unari, binari)**, **precedența lor** și **asociativitatea lor** se va păstra și pentru variantele supraîncărcate de către utilizator.

Supraîncărcarea operatorilor

- În expresia **ob1 == ob2 – ob3,** *operator-()* se va evalua înaintea lui *operator==()*

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

- **++var**; ⇔ **var+=1**; ⇔ **var=var+1**; nu mai au loc și pentru tipurile abstracte.

- *Funcțiile operator* se pot supraîncărca global sau ca metode în cadrul unei clase.

Supraîncărcarea operatorilor

- Următorii operatori nu pot fi supraîncărcați:
  - .
  - ::
  - ?:
  - sizeof
- Următorii operatori pot fi supraîncărcați:
  - + - * / = < > += -= *= /= << >> <<= >>= == != <= >= ++ -- % & ^ ! | ~ &= ^= |= && || %= [] () , ->* -> new delete new[] delete[]

Supraîncărcarea operatorilor

- Supraîncărcarea operatorilor se poate realiza prin:
  - Utilizarea funcțiilor membre
  - Utilizarea funcțiilor globale

S.o. prin funcții membre

- In clasă se adaugă

```
Complex operator+ (Complex);
```

- Se adaugă funcția

```
Complex Complex::operator+(Complex c)
{
    Complex rez;
    rez.re = re + c.re;
    rez.im = im + c.im;
    return rez;
}
```

- La rularea programului, se afișează: `4 + 6i`

S.o. prin funcții globale

```
Complex operator+(Complex c1, Complex c2)
{
    Complex rez (c1.GetRe() + c2.GetRe(),
                 c1.GetIm() + c2.GetIm() );
    return rez;
}
```

Supraîncărcarea operatorilor

- 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 'Complex Complex::operator +(Complex)'
or       'Complex operator +(Complex,Complex)'
while trying to match the argument list '(Complex, Complex)'
```

Supraîncărcarea operatorilor

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->()	-

Where a is an object of class A, b is an object of class B and c is an object of class C.

Operatori impliciți

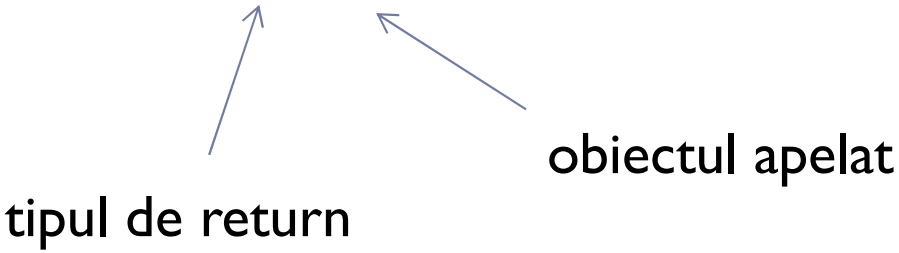
- Compilatorul are definite variante implicite ale lui *operator*= (atribuire) și *operator*& (adresa lui)
- Pentru folosirea acestor operatori în scopul lor clasic nu trebuie supraîncărcați operatorii = și &

```
Complex *c4;
c4 = &c2;
c4->Afisare();
```

- Se pot supraîncărca acești operatori dar se recomandă să nu se schimbe semnificația lor

Supraîncărcarea operatorului ++

- Forma pre-fixată
  - ++a
- Forma 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 functionarea apelurilor de genul
    - b = a++



Operatorii ++ pentru numere intregi

```
int a = 1, x = 1;
int b =  a++;
int y = ++x;

cout << b << endl;
cout << x << endl;

int c = ++b + a++;

cout << c << endl;
```

- 1
- 2
- 4

Forma pre-fixată

- În clasă se adaugă:

```
Complex operator++();
```

- Funcția de supraîncărcare:

```
Complex Complex::operator++()
{
    re++;
    im++;
    return *this;
}
```

Forma post-fixată a operatorului ++

- În clasă se adaugă:

```
Complex operator++(int);
```

- Funcția de supraîncărcare:

```
Complex Complex::operator++(int)
{
    Complex temp = *this;
    re++;
    im++;
    return temp;
}
```

Forma pre si post-fixată a operatorului ++

Exemplu VS

Supraîncărcarea operatorului =

- Operatorul = (pentru o atribuire de forma a=b) trebuie să asigure
  - dealocarea zonelor de memorie in cazul in care obiectul a are zone de memorie alocate dinamic anterior
  - funcționalitatea pentru cazul unei auto-atribuiri (a=a)
  - că obiectul a nu se modifică dacă atribuirea nu poate avea loc
  - funcționarea corectă a unei atribuiri înlănțuite (a=b=c)

- Operatorul = poate să returneze o referință pentru a evita apelarea constructorului de copiere la return

Supraîncărcarea operatorului =

- În clasă se adaugă

```
Complex operator= (Complex);
```

- Definiția funcției fiind:

```
Complex Complex::operator= (Complex c)
{
    std::swap(re, c.re);
    std::swap(im, c.im);

    return *this;
}
```

- Această metodă se numește “copy-and-swap idiom”

Cuvantul cheie static - C

- Variabilele declarate static
  - Nu-și pierd valoarea între apelurile funcției în care au fost declarate
  - Variabilele statice globale nu sunt vizibile în afara fișierului în care au fost declarate
- Funcțiile declarate static
  - Nu sunt vizibile decât din fișierul de unde au fost declarate

Cuvantul cheie static – C++

- Variabile statice membre
- Funcții statice membre

Cuvantul cheie static – C++

- Variabile statice membre
  - Toate obiectele de un anumit tip împart accesul la variabilele statice ale clasei
  - Pot fi declarate în clasă, nu și definite
  - Trebuie definite (inițializate) în afara clasei în scop global
  - Pentru că este o variabilă unică pentru toate obiectele de tipul clasei, **poate fi accesată direct ca un membru al clasei**, fără a avea un obiect instanțiat de tipul clasei din care fac parte

Variabile membre statice

```
class A
{
public:
      static int x;
};

int A::x = 10;

int main ()
{
      A a1,a2;
      cout << "a1.x=" << a1.x << endl;
      a1.x = 99;
      cout << "a2.x=" << a2.x << endl;
      cout << "A::x=" << A::x << endl;
}
```

```
a1.x=10
a2.x=99
A::x=99
```

Funcții statice membre

- Pot fi apelate fără a folosi un obiect instanțiat de tipul clasei
- Nu au acces la pointerul `this`
- Se comportă precum o funcție globală, dar au acces la membrii privați ai clasei din are fac parte
- Nu pot accesa membrii non-statici ai clasei decât prin intermediul unui obiect

Funcții statice membre

- In clasă se adaugă

```
public:
        static int x;
        static void f1 ();
```

- Se adaugă funcția

```
void A::f1()
{
        x++;
}
```

```
A::x=100
A::x=101
```

- In main, se adaugă:

```
a1.f1();
cout << "A::x=" << A::x << endl;
A::f1();
cout << "A::x=" << A::x << endl;
```

Funcții statice membre

```
class A
{
private:
        int y;
public:
        static int x;
        static void f1 ();
};

void A::f1()
{
        x++;
        y=0;
}
```

```
void A::f1()
{
        x++;
        A tempClass;
        tempClass.y=0;
}
```

- error C2597: illegal reference to non-static member 'A::y'