PROGRAMAREA ORIENTATĂ OBIECT C++

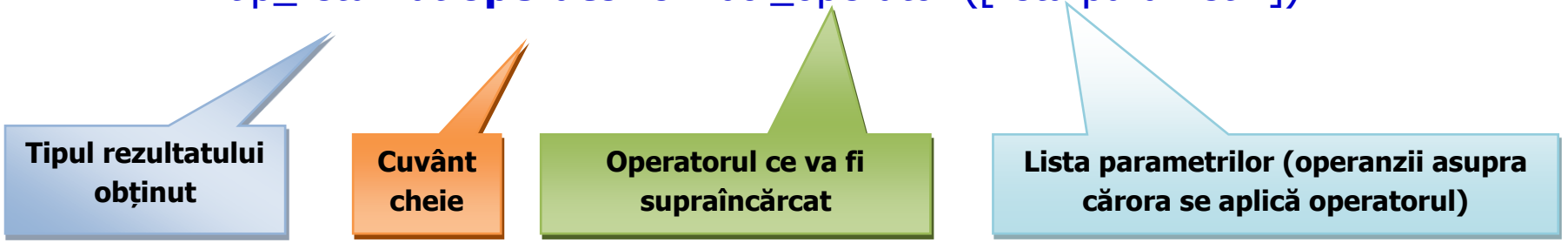
Conf.univ.dr. Ana Cristina DĂSCĂLESCU

Universitatea Titu Maiorescu

- Supraîncărcarea operatorilor de incrementare/decrementare
- Supraîncărcarea operatorului de asignare
- Supraîncărcarea operatorului de indexare
- Supraîncărcarea operatorului de comparație
- Supraîncărcarea operatorilor de inserție și extragere

- Forma generală a funcţiilor operator membre ale clasei este următoarea:

tip_returnat **operator** simbol_operator ([lista parametrii])



- O funcție membra de tip operator primește ca argument pointerul this.

- Numărul de parametri este cu 1 mai mic decât aritatea operatorului.

- **Argumentul transmis funcţiei operator este operandul din dreapta operaţiei**, iar operandul din stânga este obiectul pentru care se apelează funcţia operator, adresat prin pointer *this*.

SUPRAÎNCĂRCAREA OPERATORILOR FOLOSIND FUNCȚII FRIEND

- Sintaxa unei funcții operator de tip friend:

```
class IdClasa
{
  ...
  friend tip_rez operator simbol_operator (lista_parametri);
};

tip_rez operator simbol_operator (lista_parametri){ ... }
```

- Numărul de parametri este egal cu aritatea operatorului, fiind necesară transmiterea tuturor operanzilor, deoarece nu mai există un obiect al cărui pointer să fie transferat implicit funcţiei.

- Primul operand este obiectul curent pentru care se apelează operatorul.

- **Observație:** În multe cazuri, utilizarea fie a funcțiilor friend, fie a funcțiilor membre pentru supraîncărcarea unui operator nu provoacă diferențe funcționale programului. Totuși, în unele situații anumiți operatori se pot supraîncărca doar cu funcții membre, în timp ce alți operatori doar prin funcții friend.

- Pentru clasa **Complex**, cu datele membre `re` (partea reală) și `im` (partea imaginară), vom defini mai multe operații:
  - suma a două numere complexe, folosind o metodă membră;
  - conjugatul unui număr complex, folosind o metodă membră;
  - înmulțirea unui număr complex cu un scalar, folosind o funcție independentă de tip friend;
  - opusul unui număr complex, folosind o funcție independentă de tip friend.

SUPRAÎNCĂRCAREA OPERATORILOR - EXEMPLU

```
class  Complex {
 float  re,im;
 public:
   Complex(float re=0,float im=0);
   void afisare();
  //Supraîncărcare cu metode membru
  Complex operator+(Complex z);

  Complex operator~();
  //Supraîncărcare cu funcții friend
  friend Complex operator*(double v, Complex z2);

  friend Complex operator-(Complex z);
};
Complex::Complex(float re, float  im){
  this->re = re;
  this->im = im;
}
void Complex::afisare(){
   cout<<re<<" "<<im;
}
```

Funcţia supraîncarcă un operator binar. Argumentul transmis funcţiei este **operandul din dreapta operaţiei**. Adresa operandului din stânga este obiectul pentru care se apelează funcţia operator, accesat prin pointer-ul this.

Funcţia supraîncarcă un operator unar.

Funcţia supraîncarcă un operator binar. Deoarece funcția nu este membră a clasei, se transmit toți operanzii necesari.

SUPRAÎNCĂRCAREA OPERATORILOR - EXEMPLU

```
Complex Complex::operator+(Complex  z){
 Complex rez;
 rez.re =this->re + z.re;
 rez.im = this->im + z.im;
 return rez;
}

Complex& Complex::operator ~(){
     this->im = -im;
     return *this;
}

Complex operator*(double v, Complex z2){
 return Complex(z1.re*v,z1.im*v);
}

Complex operator -(Complex z){
 return Complex(-z.re,-z.im);
}

int main(){
 Complex z1(4,5), z2(3,1), z;
 z=z1+z2;
 z=3.5*z1;
 ~z1;
 z=-z1;
 return 0;
}
```

OUTPUT

```
7.0 6.0
24.5 21.0
24.5 -21.0
-24.5 21.0
```

Echivalentă cu:

```
z = z1.operator+(z2);
```

Echivalentă cu:

```
z = operator(3.5,z1);
```

SUPRAÎNCĂRCAREA OPERATORILOR DE INCREMENTARE/DECREMENTARE

- Operatorii de incrementare ++ și decrementare -- sunt operatori unari care modifică operanzii.
- La supraîncărcarea operatorilor de incrementare sau decrementare (++, --) se poate diferenţia un operator prefixat de un operator postfixat folosind două versiuni ale funcţiei operator. Pentru varianta postfixată funcția operator++ sau operator -- are un argument suplimentar.

Operator de incrementare prefixat (++z)	Operator de incrementare postfixat (z++)

• **prin funcție membru:** se modifică obiectul current, deci se va returna adresa acestuia (pointer-ul *this*)
`class IdClasa{`
`  IdClasa&  operator ++ ();`
`};`
	• **prin funcție membru:** se modifică obiectul curent, deci se va returna adresa acestuia (pointer-ul this)
	`class IdClasa{`
	`  IdClasa& operator ++ (int n) ;`
	`};`

• **prin funcție de tip friend:** se modifică obiectul transmis ca parametru și se returnează
`class IdClasa {`
`  IdClasa operator ++ (IdClasa &ob) ;`
`};`
	• **prin funcție de tip friend:** se modifică obiectul transmis ca parametru și se returnează
	`class IdClasa {`
	`  IdClasa operator ++ (IdClasa &ob,`
	`  int n) ;`
	`}`

SUPRAÎNCĂRCAREA OPERATORILOR DE INCREMENTARE/DECREMENTARE

```
class  Complex {
  float  re,im;
   public:
      Complex(float re=0,float im=0);

       //supraîncărcare prin metode membre
       Complex& operator++();
       Complex& operator++(int a);

      //supraîncărcare prin metode friend
       friend Complex operator--(Complex &ob);
       friend Complex operator--(Complex &ob, int a);
};

Complex::Complex(float re, float  im){
       this->re = re;
       this->im = im;
}

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

Operator de incrementare prefixat

Operator de incrementare postfixat

SUPRAÎNCĂRCAREA OPERATORILOR DE INCREMENTARE/DECREMENTARE

```
Complex&::Complex operator++(int a){
 Complex cap = *this;
 this->re++;
 this->im++;
 return cap;
}


int main()
{
    Complex z1(1.2,3.2), z2;
    ++z1;//preincrementare
    z2++;//postincrementare
  return 0;
}
```

Observație:
Pentru funcţia `operator++(int a)` parametrul `a` are valoarea `0`.

SUPRAÎNCĂRCAREA OPERATORULUI DE ASIGNARE

- Pentru asignarea a două obiecte de același tip, în limbajul C++ se poate utiliza definiţia implicită de asignare, prin care se realizează copierea la nivel de bit a datelor membre ale obiectului sursă.

```
Complex z1(1,2), z2;
z2=z1; //copiere la nivel de bit
```

- **Problemă:** Dacă datele membre ale unei clase sunt alocate dinamic, copia bitwise, care se execută implicit la asignare, conduce la existența a doi pointeri care vor indica către aceeaşi zonă de memorie.

```
class X{
 public:
   char *p;

    X(char *p){
        this->p=new char[10];
        strcpy(this->p,p);
    }

    void afisare(){
        cout<<p<<" ";
    }
};
```

```
int main(){
    X ob1("abc"),ob2("def");
    ob1=ob2;
    ob1.afisare();
    strcpy(ob2.p,"aaaa");
    ob1.afisare();
    ob2.afisare();
    return 0;
}
```

Modificarea `ob2` conduce la modificarea `ob1`

```
Output: def  aaa aaa
```

➢Operatorul de asignare = se supraîncarcă numai prin funcție membru!

```
class IdClasa
{
  IdClasa& operator = (const IdClasa &ob);
}
```

➢Pentru clasele cu date alocate dinamic, funcția **operator=** eliberează spațiul ocupat de obiectul pentru care se realizează asignarea, alocă un nou spațiu pentru obiectul care va fi copiat și realizează copia datelor membre ale obiectului sursă.

➢**Exemplu:** Definim clasa **String** care conţine un pointer `pstr` la un şir de caractere şi o variabilă de tip întreg `dim` care memorează dimensiunea vectorului de caractere corespunzător.

SUPRAÎNCĂRCAREA OPERATORULUI DE ASIGNARE

```
class String{
      char *pstr;
      int dim;
      public:
       String(const char *p);
       String(const String& r);
      ~String();
       String& operator=(const String &op2);
};

String::String(const char *p){
    dim = strlen(p) + 1;
    pstr = new char[dim];
    strcpy(pstr, p);
}

String::String(const String& r){
    dim = r.dim;
    pstr = new char[dim];
    strcpy(pstr, r.pstr);
}
```

Constructor de copiere

Supraîncărcarea operatorului de asignare

```
String::String(const String& r){
    dim = r.dim;
    pstr = new char[dim];
    strcpy(pstr, r.pstr);
}

String::~String(){
    if (pstr) delete []pstr;
    dim = 0;
}

String& String::operator=(const String &op2){
    if (pstr) delete []str;
    dim = op2.dim;
    str = new char[dim];
    strcpy(str, op2.pstr);
    return *this;
}

int main(){
    String sir1("abc"), sir2("def");
    String sir3 = sir1;
    sir2 = sir3;
}
```

Apel constructor de copiere

Apel operator de asignare

➢Supraîncărcare operatorului de indexare

- Operatorul de indexare poate fi supraîncărcat pentru tipuri de date definite de programator

- Funcția operator de indexare nu poate fi decât o metodă membră nesatică a clasei
- Sintaxa

```
tip_returnat operator[](int index)
{
    return val;
}
```

- Exemplu pentru clasa `String`

```
char String::operator[](int i){
    return str[i];
}
```

➢Supraîncărcarea operatorilor relaționali

- Operatorii relaționali (==, <, <=, >, >=, !=) pot fi supraîncărcați pentru tipuri de date definite de programator prin funcții operator

- O funcția operator de relație specifică criteriul de comparație

- Exemple:
  - Două obiecte de tip `Persoana` se pot compara din punct de vedere lexicografic după nume
  - Două obiecte de tip `Produs` se pot compara în raport cu prețul lor
  - Două obiecte de tip `Firma` se pot compara în raport cu cifra de afaceri

➢Supraîncărcarea operatorului de extragere <<

- Se consideră o clasă definită `class C {…}`

```
C ob;
cout<<ob; //eroare
```

- Operatorul de extragere nu este supraîncărcat implicit pentru tipuri de date definite de utilizator

- Operatorul de extragere este binar, operandul din dreapta este de tipul `ostream`, iar operandul din stânga este de tipul `C`

- Operatorul de extragere se poate supraîncărca doar printr-o funcție de tip friend!!!

```
friend ostream& operator<<(ostream &out, C ob);
```

➢Supraîncărcarea operatorului de inserție >>

- Se consideră o clasă definită `class C {…}`

```
C ob;
cin>>ob; //eroare
```

- Operatorul de inserție nu este supraîncărcat implicit pentru tipuri de date definite de utilizator

- Operatorul de inserție este binar, operandul din dreapta este de tipul `istream`, iar operandul din stânga este de tipul `C`

- Operatorul de inserție se poate supraîncărca doar printr-o funcție de tip friend!!!

```
friend istream& operator>>(istream &out, C &ob);
```

➢ Exemplu

```
class String
  { .....

    friend ostream& operator<<(ostream &out, String ob);
    friend istream& operator>>(istream &in, String &ob);

  }
  ostream& operator<<(ostream &out, String ob)
  {
      out<<ob.pstr;
      return out;
  }
  istream& operator>>(istream &in, String &ob)
  {
      char sir[50];
      in.get(sir, 50);
      strcpy(ob.pstr, sir);
      return in;
  }
```

- Supraîncărcarea funcţiilor şi a operatorilor (*overloading*) sunt mecanisme importante în C++ care oferă flexibilitate şi extensibilitate limbajului.

- Se pot supraîncărca doar operatorii existenți în limbaj.

- Nu se pot supraîncărca opeartorii: `.  .* :: ?: sizeof`

- Operatorii se pot supraîncărca prin metode membre sau prin funcții de tip friend.

- Sunt operatori care se pot supraîncărca doar prin metode membre (=, [] etc.) și operatori care se pot supraîncărca doar prin funcții friend (<<, >>).