Programare Orientată Obiect

Laborator 6

Despre ce vorbim azi

- Supraîncărcarea operatorilor și a funcțiilor

Ce este supraîncărcarea?

- Supraîncărcarea (overloading) permite, general vorbind, atribuirea a mai multor semnificații aceluiași simbol
- În practică înseamnă cu un operator sau o funcție este definită de mai multe ori, diferența între definiri fiind dată de tipul parametrilor sau numărul lor (niciodată de tipul returnat) adică de signatura lui/ei
- Supraîncărcarea este o primă formă de polimorfism (efectuează operații diferite în funcție de context)
- Ex: operatorul + care funcționează atât pe întregi cât și pe numere reale

Selectarea funcției potrivite

- În cazul în care funcția potrivită nu se poate selecta după numărul de parametri, atunci selecția se va face după tipul lor în următoarea ordine:

- Se încearcă identificarea funcției fără efectuarea vreunei conversii
- Se încearcă conversii nedegradante (fără pierdere de informații)
- Se încearcă conversii degradante (cu pierdere de informații)
- Se apelează eventuale conversii definite de programator prin supraîncărcarea operatorului de cast

Erori posibile

- Căutarea se oprește când o singură funcție coincide apelului
- Dacă mai multe funcții corespund apelului vom primi eroare de ambiguitate
- Dacă niciuna nu corespunde apelului atunci vom avea eroare de linkeditare
- Atenție! În cadrul claselor putem vorbi de supraîncărcarea funcțiilor membre doar atunci când acestea fac parte din aceeași clasă (distincția între clase fiind făcută prin operatorul de rezoluție)

Supraîncărcarea operatorilor

- Operatorii sunt de fapt funcții ce au denumirea *operator<simbol grafic>*
- Ex: a + b poate fi privit ca apel de a.operator+(b)
- Cu toate că pot fi supraîncărcați, anumite caracteristici ale operatorilor nu pot fi schimbate:

- precedența și direcția de evaluare
- asociativitatea
- pluralitatea trebuie conservată
- pot fi supraîncărcați doar operatori existenți

Excepții ale supraîncărcării operatorilor

- operatorii nu se compun automat;
- operatorii . .* :: ?: sizeof() nu pot fi supraîncărcați
- nu se garantează implicit comutativitatea
- predefinirea și postdefinirea trebuie implementate explicit
- Supraîncărcarea se face numai prin funcții membre nestatice sau funcții friend

Supraîncărcarea operatorilor unari ++ și --

- Atenție le prefixare/postfixare precum și la apelul în cascadă

```
friend const Student& operator++(Student&);
friend const Student operator++(Student&, int);
```

```
const Student& operator++(Student& s)
{
    s.medie++;
    return s;
}

const Student operator++(Student &s, int i)
{
    Student aux = s;
    s.medie++;
    return aux;
}
```

```
cout<<"Medie pentru s1 inainte de preincrementare: "<<s1.get_medie()<<endl;
++s1;
cout<<"Medie pentru s1 inainte dupa preincrementare: "<<s1.get_medie()<<endl;
Student s5 = ++s1;
Student s6 = s1++;
cout<<"Diferenta intre preincrementare -> "<< s5.get_medie() <<" si postincrementare -> "<< s6.get_medie() <<endl;
```

Supraîncărcarea operatorilor binari + și -

- Tipul returnat de operatorii binari e stabilit de programator

```
friend double operator+(Student &, float);
friend double operator+(float, Student&);
double operator+(Student& s)
{
    return this->medie + s.medie;
}
```

```
double operator+(Student &s, float m)
{
    return s.medie + m;
}

double operator+(float m, Student &s)
{
    return s.medie + m;
}
```

```
cout<<s5+2<<endl;
cout<<3+s5<<endl;
cout<<s5+s6<<endl;
```

Supraîncărcarea operatorilor << și >>

- Se poate face doar prin funcții friend

```
friend istream& operator>>(istream&, Student &);
friend ostream& operator<<(ostream&, Student);
```

```
istream& operator>>(istream& intrare, Student &s)
{
    cout<<"Nume: ";
    intrare>>s.nume;
    cout<<endl<<"Matricol: ";
    intrare>>s.matricol;
    cout<<endl<<"Medie: ";
    intrare>>s.medie;
    return intrare;
}

ostream& operator<<(ostream& iesire, Student s)
{
    iesire<<"Studentul cu matricolul "<<s.matricol<<" si media "<<s.medie<<" are numele "<<s.nume<<endl;
    return iesire;
}
```

```
Student s7;
cin>>s7;
cout<<s7;
```

Alți operatori ce pot fi supraîncărcați

- Operatorul **[]** – pentru încadrarea indicelui în domeniu, pentru a adresa un element sau pentru a ascunde prelucrări complexe
- Operatorii **new** și **delete** pentru a nu mai aloca/dezaloca spațiu de memorie explicit în constructor/destructor
- Operatorul **cast**
- Operatorul **,**
- Operatorul **funcție**
- Operatorul **->**

Conversia dintr-un obiect în altul

- Poate fi făcută prin doua metode:

1. Constructor în prima clasă ce primește drept parametru obiect de tipul celei de-a doua clase
2. Supraîncărcarea operatorului cast la prima clasă în cea de-a doua clasă