Programare orientată obiect

Cursul 7

Sumar

- Moștenire/derivare (2)
  - Concepte de bază în limbajul C++
  - Polimorfism
  - Funcții virtuale
    - Destructori virtuali

Derivarea în C++

```
class Baza
{

};

class Derivata : [tip_derivare] Baza
{
};
```

Tipul derivării

- Stabilește modul de modificare a accesului la membrii clasei de bază din clasa derivată
- private
  - Implicit pentru clase
- protected
- public
  - Implicit pentru structuri

Tipul derivării

- Modificarea tipului de acces (Bază -> Derivată)
  - Membrii *private* sînt inaccesibili în clasa derivată
  - Regulă: este selectat cel mai restrictiv modificator (moștenire și acces)
- Posibilitatea păstrării tipului de acces public prin derivarea privată:
  - În clasa de bază, **membru** este în zona publică de acces
  - În clasa derivată, în zona de acces public:
    - Baza**::membru**;

Exemplul

```
class Produs
{
//…
};

class ProdusPerisabil : public Produs
{
protected:
        int valabilitate;
        umv umVal; //enum umv {zile, luni, ani};
public:
//…
}
```

Nu sînt moștenite (parțial sau integral)

- Constructorii și destructorii
  - Inclusiv constructorul de copiere
- Operatorul de atribuire
- Relațiile de tip *friend* (clase și funcții)

Constructorii

- Fiecare clasă are constructori proprii
  - Responsabili și de inițializarea membrilor
- Constructorii clasei derivate
  - Apelează constructorii clasei de bază
    - **Implicit** (trebuie să existe **constructorul fără parametri**)
    - **Explicit** (prin **listă de inițializare**)
- Constructorii se apelează în ordinea claselor în ierarhie
  - De sus în jos

Constructorii

```
class ProdusPerisabil : public Produs
{
protected:
        int valabilitate;
        umv umVal; //enum umv public:
public:
        ProdusPerisabil(int cod = 0, char * denumire="", char *um="",
                        float  pret = 0.0f, int valab = 0, durata dur = zile) :
                        Produs(cod, denumire, um, pret),
                                valabilitate(valab), valUM(dur) { }
};
```

Constructorul de copiere

- Contextul:
  - **Derivata d1;**
  - **Derivata d2 = d1;**
- Dacă **nu există** în nici o clasă este generat de compilator (copiere bit cu bit)
- Dacă **există doar în clasa de bază**: se apelează doar acesta pentru membrii din clasa de bază și constructorul de copiere implicit pentru ceilalți membri din clasa derivată
- Dacă **există doar în clasa derivată**: se apelează:
  - Implicit constructorul fără parametri al clasei de bază (dacă există)
  - Explicit alți constructori din clasa de bază
- și constructorul de copiere al acestei clase
- Dacă **există și în clasa de bază și în clasa derivată**: se apelează:
  - Implicit constructorul fără parametri al clasei de bază (dacă există)
  - Explicit constructori din clasa de bază sau constructorul de copiere al clasei de bază
- și constructorul de copiere pentru membrii din clasa derivată

Destructorii

- Fiecare destructor este responsabil de eliberarea propriilor resurse
  - Inițializate prin constructori
- Se apelează începînd de jos în sus în ierarhie
  - Derivată -> Bază

Operatorul de atribuire

- Contextul:
  - **Derivata d1, d2;**
  - **d1 = d2;**
- Dacă **nu există** în nici o clasă este generat de compilator (copiere bit cu bit)
- Dacă **există doar în clasa de bază**: se apelează acesta pentru membrii din clasa de bază și operatorul implicit pentru ceilalți membri din clasa derivată
- Dacă **există în clasa derivată**: se apelează acest operator

Conversii în ierarhii de clase

- Între obiecte
  - Derivată -> Bază, cu trunchiere
  - Bază -> Derivată : NU!
- Upcasting
  - Conversie pointer/referință clasă derivată -> clasă de bază
  - Se realizează implicit
- Downcasting
  - Conversie pointer/referință clasă de bază -> clasă derivată
  - Se realizează explicit (operatorul de conversie)

Conversii în ierarhii de clase

- Declarații:
  - Produs *prod;
  - ProdusPerisabil prodPerisabil;
- Pointeri inițializați la definire/prin atribuire
  - Produs *prod1 = new ProdusPerisabil();
  - prod = &prodPerisabil;
- Referințe inițializate la definire
  - Produs &rProd = prodPerisabil;
- Transfer de parametri prin pointeri/referințe
  - double calculeazaValoare(Produs *);
    - apel: **calculeazaValoare(&prodPerisabil);**
  - double calculeazaValoare(const Produs &);
    - apel: **calculeazaValoare(prodPerisabil);**
- Masive de pointeri la obiecte
  - Produs *produse[N];
    - Inițializare: **produse[0] = &prodPerisabil;**

Polimorfismul (2)

- Supraîncărcare (overloading)
  - Același nume, parametri diferiți, același context sau context diferit (clasă, global)
- **Redefinire (overriding)**
  - Același nume, aceeași parametri, clase diferite (în ierarhie)

Polimorfismul (2)

```
class Produs
{
//membri specifici
public:
    void afiseaza();
};
```

```
class ProdusPerisabil: public Produs
{
//membri specifici
public:
    void afiseaza();//si pentru membrii specifici
};
```

Polimorfismul (2)

Produs produs, *pProdus;

ProdusPerisabil prodPerisabil, * pProdPerisabil;

//obiecte

produs.afiseaza(); prodPerisabil.afiseaza();
prodPerisabil.Produs::afiseaza();

//pointeri la obiecte (același tip)

pProdus = &produs; pProdus->afiseaza();

pProdusPerisabil = &prodPerisabil; pProdPerisabil->afiseaza();

//pointeri la obiecte (derivat -> bază)

pProdus = &prodPerisabil; pProdus->afiseaza();

Funcții virtuale

- Mecanism care permite referirea corectă a funcțiilor dintr-o clasă derivată prin intermediul unui pointer sau referință la o clasă de bază, inițializat cu un adresa unui obiect din clasa derivată
- Early (static) binding
  - Tipul se determină la compilare
  - Funcții non-virtuale
- Late (dynamic) binding
  - Tipul se determină la execuție
  - Funcții virtuale
  - Pointeri/referințe clase de baze
- Funcția ramîne virtuală în cadrul ierarhiei
- Funcție include în prototip **virtual**

Funcții virtuale

- **Tabela de funcții virtuale (v-table, vtbl)**
  - Este creată global, la nivelul clasei
  - Conține pointeri către metodele virtuale
  - Pentru fiecare nouă clase în ierarhie, tabela este actualizată cu adresele funcțiilor redefinite
- **Pointer la tabela de funcții virtuale (v-pointer, vptr)**
  - Tabela de funcții virtuale este referită printr-un pointer
  - Există pentru fiecare obiect dintr-o clasă care are cel puțin o funcție virtuală
  - Este adăugat de compilator și inițializat în constructorul clasei

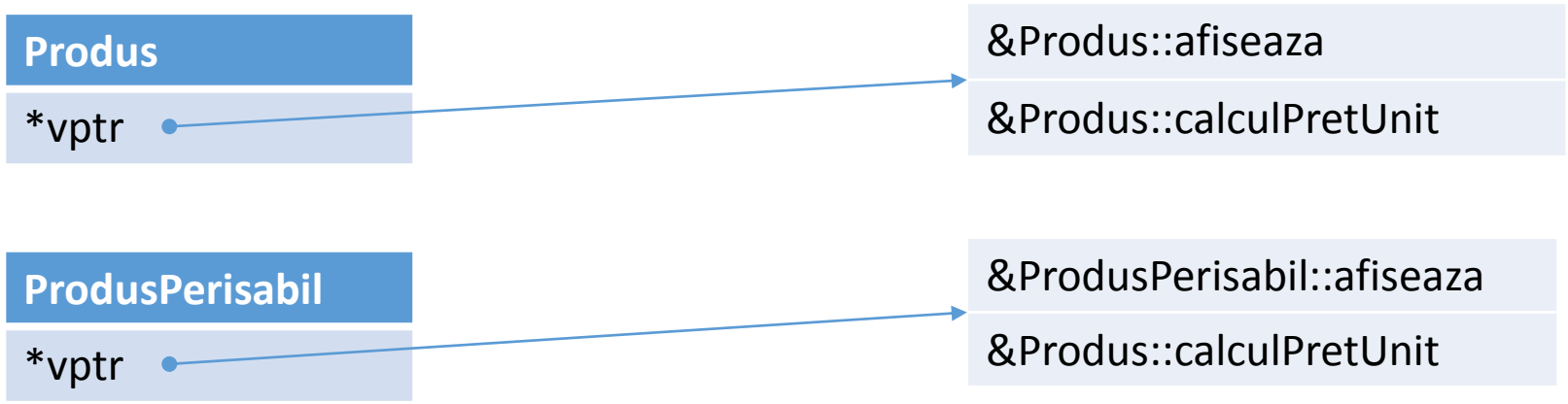
Funcții virtuale

```
class Produs
{
//membri specifici
public:
    virtual void afiseaza();
};
```

```
class ProdusPerisabil: public Produs
{
//membri specifici
public:
    void afiseaza();//si pentru membrii specifici
};
```

Funcții virtuale

pProdus = &prodPerisabil; pProdus->afiseaza();



pProdus->afiseaza(); => vptr[0](pProdus)

Destructori virtuali

Apel

```
Produs *prod1 = new ProdusPerisabil();
//…
delete prod1;
```

Iesire:

Apel constructor Produs

Apel constructor ProdusPerisabil

Apel destructor Produs

Destructori virtuali

Declarare și apel

```
class Produs
{
//membri specifici
public:
      virtual ~Produs();
};
//…
Produs *prod1 = new ProdusPerisabil();
//…
delete prod1;
```

Iesire:

Apel constructor Produs

Apel constructor ProdusPerisabil

Apel destructor ProdusPerisabil

Apel destructor Produs