Lab 9 – P2 – Polimorfism

Cuprins

1. Polimorfismul 1
2. Funcții virtuale 2
3. Tabela virtuală (virtual table) 4
4. Destructor virtual 4
5. Exerciții 5

1. Polimorfismul

Polimorfismul reprezintă capacitatea unui obiect de tip A de a exista și de a fi utilizat precum un obiect de tipul B. Considerând exemplul claselor `StudentAC` si `PersoanaAC`, unde clasa `StudentAC` este o clasa derivată din clasa `PersoanaAC`, polimorfismul reprezintă modul de utilizare a unui obiect `StudentAC` în locul unui obiect `PersoanaAC` pentru un obiect de tipul `PersoanaAC`. Altfel spus, un obiect de tipul unei clase derivate este compatibil ca pointer cu un obiect de tipul clasei de bază.

Considerând implementările precedente ale claselor `PersoanaAC` si `StudentAC`, un exemplu de utilizare a polimorfismului este:

```
StudentAC *s3 = new StudentAC ("1234567890122", "Ion",
          "Vaslui",2,10);
StudentAC s4("7734567890122","Vasile","Neamt",3,7);

PersoanaAC *ps2 = s3;
PersoanaAC *ps3 = &s4;
PersoanaAC &ps=s4;

ps2->afisareProfil();
ps3->afisareProfil();
delete s3;
```

Consideram codul urmator:

```
PersoanaAC *ps5 = new StudentAC ("9999567890122", "Pavel",
          "Iasi",1,10);
ps5->afisareProfil();
```

La rularea acestuia, se va afișa pe ecran

```
Nume: Pavel CNP: 9999567890122 Adresa: Iasi
```

Observam că pentru obiectul `ps5` nu se apelează funcția afișare profil din `StudentAC`, ci se apelează funcția `afisareProfil` din clasa `PersoanaAC`. Pentru a corecta acest lucru, se introduce conceptul de funcții virtuale.

2. Funcţii virtuale

O funcție virtuală este un tip special de funcție care permite apelarea funcției din clasa cea mai derivată cu același prototip. O funcție a unei clase ce poate fi redefinită într-o clasă derivată este o funcție virtuală. Pentru a declara o funcție virtuală, se folosește cuvântul cheie `virtual` înaintea declarării funcției.

Pentru a putea apela funcția `afisareProfil` din clasa `StudentAC` pentru obiectul `ps5`, se va declara funcția `afisareProfil` din clasa de baza virtuală:

```
class PersoanaAC
{
...
     virtual void afisareProfil();
...
};
...

PersoanaAC *ps5 = new StudentAC ("9999567890122","Pavel",
     "Iasi",1,10);
ps5->afisareProfil();
```

La rularea codului de mai sus, se va afișa:

```
Nume: Pavel CNP: 9999567890122 Adresa: Iasi
An studiu: 1 Nota P2: 10
```

Observații:

- `ps5` este o adresă către clasa de bază `PersoanaAC` a unui obiect derivat `StudentAC`
- când este evaluată funcția `afişareProfil`(), în mod normal ar fi apelată funcția din clasa de bază
- deoarece funcția `PersoanaAC::afisareProfil()` este virtuală, compilatorul caută să vadă dacă există o funcție cu același prototip în clasa derivată `StudentAC`
- daca aceasta funcție exista, va fi apelata in locul funcției din clasa de baza

Altfel spus, pentru că `ps5` este adresa clasei de bază a unui obiect `StudentAC`, compilatorul va verifica fiecare membru moștenit și va folosi varianta cea mai derivată pentru o funcție apelată. Funcția redefinită trebuie să aibă aceiași parametri și același tip de return ca și funcția originală din clasa de bază.

Observații:

- Pentru o clasă ce nu are nici o altă clasă derivată din ea, folosirea funcțiilor virtuale nu este necesară!
- Se recomandă a nu se face toate funcțiile virtuale pentru a spori claritatea codului scris!
- Folosirea funcțiilor virtuale presupune existența claselor derivate si a polimorfismului!

O funcție definită virtuală în clasa de bază, va rămâne virtuală în tot arborele ierarhic a acelei clase, indiferent câte niveluri are derivarea.

În acest sens, utilizam exemplul următor:

```
class A {
public:
   virtual void f() { cout << "Class A" << endl; }
};
class B: public A {
public:
   void f(int x) { cout << "Class B" << endl; }
};
class C: public B {
public:
   void f() { cout << "Class C" << endl; }
};
int main() {
   B b;
   C c;
   A *pa1 = &b;
   A &pa2 = c;

   pa1->f();
   pa2.f();
}
```

La rularea acestui exemplu, se va afișa:

```
Class A
Class C
```

In acest exemplu, Clasa `B` moștenește clasa `A`, iar clasa `C` moștenește clasa `B`. Funcția `B::f` nu este virtuală și are prototipul diferit față de funcția `A::f`. Deoarece C moștenește și clasa `A`, funcția `A::f()` este virtuală și pentru clasa `C`. Din acest motiv, la apelarea `pa2.f()` se apelează `C::f()` ca funcția cea mai derivată pentru un obiect de tipul `C`.

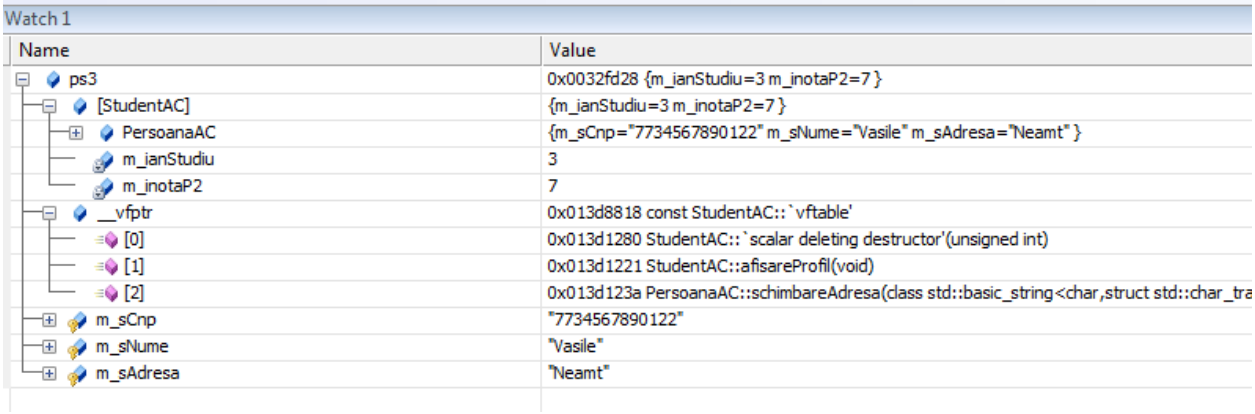
3. Tabela virtuală (virtual table)

Tabela virtuală conține pointeri către funcții virtuale. Ea este creată pentru fiecare clasă ce conține funcții membre virtuale sau suprascrie funcții virtuale. Pentru o anumită clasă există o singură tabelă virtuală. Există clase pentru care tabela virtuala nu este creată. Când un obiect este creat, se adaugă un membru ascuns, un pointer către această tabelă virtuală. Compilatorul generează automat codul în constructori pentru inițializarea pointerului către această tabelă. Tabela se accesează în momentul când se execută o funcție virtuală, pentru a asigura o eficiență sporită programului. Tabela poate fi consultată cu ajutorul debug-ului:

De exemplu, pentru un obiect ps3 declarat în modul următor:

```
StudentAC s4("7734567890122","Vasile","Neamt",3,7);
PersoanaAC *ps3 = &s4;
```

La efectuarea funcției watch pe acest obiect, se poate observa tabela sa virtuala ce conține pointeri spre funcțiile ce vor fi apelate pentru orice funcție accesibilă obiectului ps3:



4. Destructor virtual

Destructorii claselor de bază trebuie declarați virtuali pentru a asigura apelarea destructorilor din clasele derivate. În caz contrar există pericolul de a nu dealoca toată memoria utilizată. În cazul destructorilor, se apelează toți destructorii dinspre clasa derivată către clasa de bază (în ordinea inversă a apelării constructorilor).

5. Exerciţii

Scrieți un program ce conține o clasa de bază Figura si două clase derivate din aceasta Cerc și Dreptunghi.

Clasa Figura va avea

- un membru culoare de tip string
- metodele Perimetru() si Afisare()

Clasa Patrat va avea:

- ca membri:
  - coordonatele x, y ale vârfului din stânga-jos al pătratului (de tip întreg)
  - latura reprezentând lungimea laturii pătratului (de tip double)
- Metodele Perimetru() și Afisare()

Clasa Cerc va avea:

- Ca membri:
  - Coordonatele x, y ale centrului cercului (de tip întreg)
  - Raza r (de tip întreg)
- Metodele Perimetru() și Afisare()

Cerințe:

- Completați clasele descrise mai sus cu constructorii necesari pentru a putea citi de la tastatură toți parametrii obiectelor ce vor fi instanțiate din fiecare clasă.
- Completați clasele cu funcțiile Perimetru și Afisare.
- Creați un tablou de pointeri la clasa Figura care să conțină:
  - 2 obiecte de tip Cerc
  - 2 obiecte de tip Patrat
- Utilizând numai obiectele din acest tablou, testați metodele din clasele scrise.
- Determinați obiectul cu aria maxima dintre elementele tabloului.