Universitatea Tehnică „Gheorghe Asachi” din Iaşi
Facultatea de Automatică şi Calculatoare

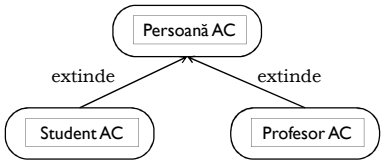
PROGRAMARE II

Curs 10

Polimorfism.
Funcţii virtuale

Polimorfismul

- Capacitatea unui obiect de tip A de a exista şi de a fi utilizat precum un obiect de tipul B
- Exemplu: utilizarea unui obiect StudentAC în locul unui obiect PersoanăAC



Pointeri la clasa de bază

- O clasă derivată este compatibilă ca tip cu un pointer către clasa de bază

```
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;
```

Operatorul de referențiere &

```
int a = 8;
int &b=a;
b=3;
//int &c;
cout<<a<<endl;
```

- Poate crea o referință (un sinonim) pentru o anumită variabilă
- Este obligatorie inițializarea referințelor (`error C2530: 'c' : references must be initialized`)
- Cand operatorul & este precedat de un tip de data (int, char, ProfesorAC) se creează o referință către acel tip
- Dacă operatorul & nu este precedat de un tip, atunci va returna adresa elementului pentru care este apelat

Tip declarat / tip real

- Fiecare variabilă are un tip declarat la compilare
- La rulare, variabila poate indica un obiect cu un tip real (fie tipul declarat, fie o subclasă a tipului declarat)

```
PersoanaAC *ps4 = new PersoanaAC ("1234567890123", "Ana",
                          "Roman");

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

- Tipul declarat pentru Ana și pentru Pavel este PersoanăAC
- Tipul real
  - Pentru Ana este PersoanăAC
  - Pentru Pavel este StudentAC

Apelarea unei funcții suprascrise

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

ps5->afisareProfil();
```

- Se afișează:

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

- De ce nu se afișează anul de studiu și nota la programare?

Funcții virtuale

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

Funcții virtuale

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

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

Funcții virtuale

- `ps5` este o adresă către clasa de bază `PersoanăAC` a unui obiect derivat `StudentAC`
- Când este evaluată funcția `afişareProfil()`, în mod normal ar fi apelată funcția din clasa de bază
- Deoarece funcția `PersoanăAC::afişareProfil()` este virtuală, compilatorul caută să vadă dacă există o funcție cu același prototip în clasa derivată `StudentAC`
- 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

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

Funcții virtuale

```
void afisareDate (PersoanaAC &pers)
{
       cout << "Afisare date persoana"<<endl;
       pers.afisareProfil();
}
```

In main:

```
afisareDate(*ps4);
afisareDate(ps);
```

```
Afisare date persoana
Nume: Ana CNP: 1234567890123 Adresa: Roman

Afisare date persoana
Nume: Vasile CNP: 7734567890122 Adresa: Neamt
An studiu: 3 Nota P2: 7
```

Funcții virtuale

- 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

Exemplu adaptat după http://publib.boulder.ibm.com

```
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();
}
```

```
Class A
Class C
```

- Funcția B::f nu este virtuală și are prototipul diferit față de funcția A::f
- Clasa B moștenește clasa A, iar clasa C moștenește clasa B
- Tipul real al obiectului `pa2` este C
- 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

Legături între apelul şi corpul funcției

- Conectarea unui apel al unei funcții cu corpul acesteia se numește legătura
- Când legătura se realizează înainte de rularea unui program(de către compilator și linkeditor), se numește **legare timpurie sau legare statică** (*early binding*)
- Legarea implicită în C++ este cea statică
- Polimorfismul este un mecanism prin care legarea dintre apelul unei metode și corpul acesteia se face la momentul rulării (**legarea dinamică**) (*late binding*)

Tabela virtuală (virtual table)

- Conține pointeri către funcțiile virtuale
- Este creată pentru fiecare clasă ce conține funcții membre virtuale sau suprascrie funcții virtuale
- Există numai una pentru o anumită clasă
- Există clase pentru care 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
- Se accesează în momentul când se execută o funcție virtuală

Tabela virtuală (virtual table)

```
class PersoanaAC
{
...
...
        virtual void afisareProfil();
        virtual void schimbareAdresa( char * adresaNoua);
};
...
        StudentAC s4("7734567890122","Vasile","Neamt",3,7);
        PersoanaAC *ps3 = &s4;
```

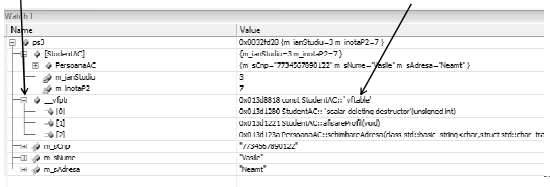
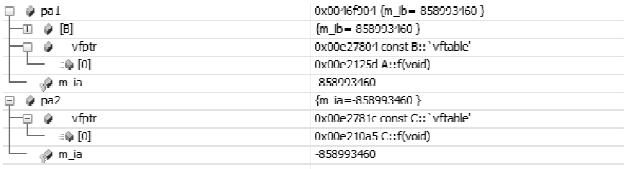


Tabela virtuală pentru ex. slide 12

```
    B b;
    C c;
    A *pa1 = &b;
    A &pa2 = c;
```



Apelare constructori şi destructori

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

delete ps5;
```

- Se va afişa pe ecran:

```
constr. cu arg. PersoanaAC
constr. cu arg. StudentAC
Nume: Pavel CNP: 9999567890122 Adresa: Iasi
An studiu: 1 Nota P2: 10
destructor PersoanaAC
```

Apelare constructori şi destructori

- Ce trebuie schimbat pentru a se apela și destructorul StudentAC?

```
class PersoanaAC
{
...
      virtual ~PersoanaAC();
};
```

- Se va afișa pe ecran:

```
constr. cu arg. PersoanaAC
constr. cu arg. StudentAC
Nume: Pavel CNP: 9999567890122 Adresa: Iasi
An studiu: 1 Nota P2: 10
destructor StudentAC
destructor PersoanaAC
```

Constructori virtuali

- Se pot declara constructorii virtuali?

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

```
error C2633: 'PersoanaAC' : 'inline' is the only legal
storage class for constructors
```

- În cazul constructorilor, pentru crearea unui obiect este necesară cunoașterea exactă a tipului acestuia
- În plus, tabela virtuală nu a fost inițializată și deci nu se poate crea dacă nu se cunoaște tipul obiectului

Destructori virtuali

- 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)

Părți din acest curs au fost traduse și adaptate după
MIT OpenCourseWare
http://ocw.mit.edu

6.088 Introduction to C Memory Management and C++ Object-Oriented Programming January IAP 2010