# Informatika 3

# 5

# Dedičnosť a kompozícia

KST

# Alokácia pamäte a inicializácia

```
class X
{
      private:
          long l;
      public:  X() {};
};
void f(int i)
{
     char c;
     if(i < 10) {
        int ii;
     }
     X x1;  // Konštruktor sa volá až tu

     for(i=0;i<5;i++)  {      }
}
```

```
void f(int i)
{
 // Tu sa alokuje objekt x1
      if(i < 10)  {
         goto Nav1;
// Chyba: príkaz goto obchádza inicializáciu
      objektu x1, t.j. volanie jeho konštruktora
      (ak je i<10)            }
     X x1;  // Až tu sa volá konštruktor
Nav1:
      switch(i)  { // Tu sa alokujú x2 a x3
        case 1 :
           X x2;  // Tu sa volá konštruktor
          break;
       case 2 :
            // Chyba: príkaz case obchádza
               inicializáciu objektu x2
           X x3;  // Tu sa volá konštruktor
              break;
       }
}
```

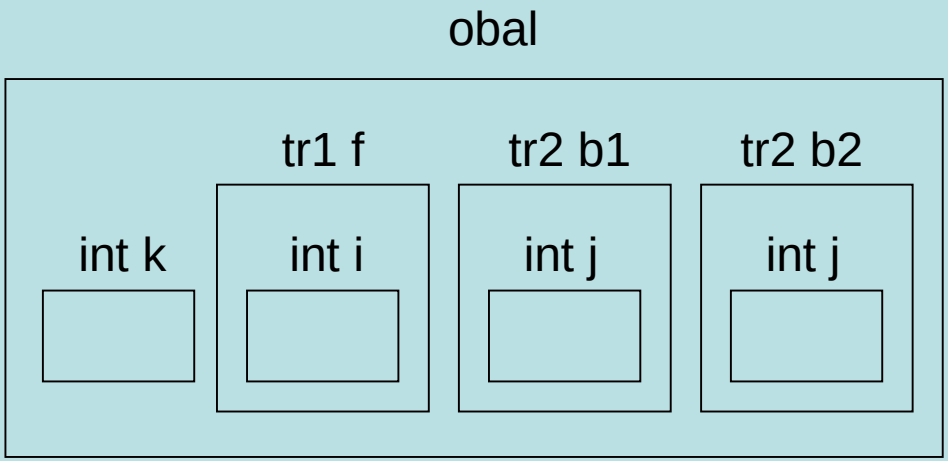
# Konštruktory a objektové členy

- keď zostavujeme triedu s objektov inej triedy – nazývame  kompozícia
- triedu, ktorá obsahuje objekty inej triedy, nazývame obalová trieda
- objektové členy sa vytvárajú zadaním inicializačného zoznamu členov v konštruktore obalovej triedy
- v prípade, že chceme volať konštruktor bez argumentov, nemusíme ho do zoznamu písať
- poradie volania inicializácií nezávisí od poradia uvedeného v konštruktore obalovej triedy ale od poradia definície objektových členov v definícií obalovej triedy
- Je vhodné inicializovať v inicializačnom zozname konštruktora aj interné typy(int,char,...)

# Konštruktory a objektové členy

```
class tr1 {
public:
    int i;
    tr1() { i=0; }
};
class tr2 {
public:
    int j;
    tr2(int x) { j=x; }
};
class obal {
public:
    int k;
    tr1 f;
    tr2 b1;
    tr2 b2;
    obal(int x): b2(x), b1(x++), k(x++) { }
};

obal obj(6);          // čo bude v jednotlivých položkách???
obj.k = 5;
obj.b1.j = 10;
obj.b2.j = 20;
```



# Dedičnosť - jednoduchá

- Pojmy:
  - **odvodená trieda** - trieda, ktorá dedí vlastnosti inej triedy
  - **základná trieda** - trieda, z ktorej sa vlastnosti dedia
- odvodená trieda dedí všetky členy so základnej triedy (táto môže byť tiež zdedená)
- odvodená trieda nemá prístup k privátnym členom základnej triedy
- deštruktor odvodenej triedy po skončení automaticky volá deštruktor základnej triedy
- ak prekryjeme člen zákl. triedy, odkážeme sa opäť naň **zakladna_trieda::člen**

# Prístupový kvalifikátor dedenia

- **public** - všetky public členy základnej triedy ostávajú public členy odvodenej triedy
- **private** - všetky public členy zákl. triedy sú private členy odvodenej triedy

```
class  A {
    int ii;              // private
public:
    int kk;
};
```

```
class A1 : public A {
      void f() {
       ii = 3;   // ii je nedostupná
       kk = 4; // ok
      }
};
A1 a1;
a1.ii = 30; // ii – nedostupná
a1.kk = 40; // ok
```

```
class A2 : private A {
      void f() {
       ii = 3;   // ii je nedostupná
       kk = 4; // ok
      }
};
A2 a2;
a2.ii = 30; // ii – nedostupná
a2.kk = 40; // kk – nedostupná
```

# Prístupový kvalifikátor dedenia

- konštruktory a deštruktory sa nededia
- konštruktory odvodených tried musia obsahovať parametrické informácie konštruktorov základnej triedy

```
enum Farba { Red, Blue, Yellow, Black };

class Hmyz {
   int Nohy;
   Farba farb;
public:
   Hmyz(int numNohy, Farba c) : Nohy(numNohy), farb(c) { }
   void Kresli();
};

class LietajuciHmyz : public Hmyz {
   int Frekvencia;
public:
   LietajuciHmyz(int numNohy, Farba c, int frekv)
            : Hmyz(numNohy,c), Frekvencia(freq) { }
   void Lietaj();
};
```

KST

# Smerník na základnú triedu

- objekt odvodenej triedy možno automaticky používať ako keby to bol objekt jeho základnej triedy (odvodená trieda je špecializáciou základnej triedy), opačne to neplatí

```
LietajuciHmyz  vcela(6, Yellow, 200);
Hmyz *zviera;
LietajuciHmyz *lieta;
lieta = &vcela;
zviera = &vcela;
lieta -> Kresli(); // ok
lieta -> Lietaj(); // ok
zviera -> Kresli(); // ok
zviera -> Lietaj(); // chyba
```

# Prístupový kvalifikátor protected

- **protected** členy základnej triedy sú ako **private** ale sú prístupné aj v odvodenej triede.
- ak je základná trieda dedená **private**, **protected** členy základnej triedy sú **private** členy odvodenej triedy
- ak je základná trieda dedená **public**, **protected** členy základnej triedy sú **protected** členy odvodenej triedy

```
class trieda {
protected:
   // protected členy
public:
   // public členy
private:
   // private členy
}
```

```
class  A {
   int ii;              // private
protected:
   int pp;
public:
   int kk;
};
```

```
class A1 : public A {
   void f() {
      ii = 3;   // ii je nedostupná
      pp = 5; // ok
      kk = 4; // ok
   }
};
A1 a1;
a1.ii = 30; // nedostupná
a1.pp = 50; // nedostupná
a1.kk = 40; // ok
```

```
class A2 : private A {
   void f() {
      ii = 3;   // ii je nedostupná
      pp = 5; // ok
      kk = 4; // ok
   }
};
A2 a2;
a2.ii = 30; // nedostupná
a2.pp = 50; // nedostupná
a2.kk = 40; // nedostupná
```

```
class B1 : public A1 {
   void f() {
      ii = 3;   // ii je nedostupná
      pp = 5; // ok
      kk = 4; // ok
   }
};
B1 b1;
b1.ii = 30; // nedostupná
b1.pp = 50; // nedostupná
b1.kk = 40; // ok
```

```
class B2 : private A2 {
   void f() {
      ii = 3;   // nedostupná
      pp = 5; // nedostupná
      kk = 4; // nedostupná
   }
};
B2 b2;
b2.ii = 30; // nedostupná
b2.pp = 50; // nedostupná
b2.kk = 40; // nedostupná
```

# Viacnásobná dedičnosť

- trieda môže mať viac ako jednu základnú triedu

```
class Zaklad1 {
protected:
   int b1;
public:
   void Set(int i) { b1 = i; }
};
class Zaklad2 {
private:
   int b2;
public:
   void Set(int i) { b2 = i; }
   int Get() { return b2; }
};
class Odvodena : public Zaklad1, private Zaklad2 {
public:
   void Print() { printf("b1 = %d a b2 = %d", b1, Get()); }
};
```

# Nejednoznačnosť

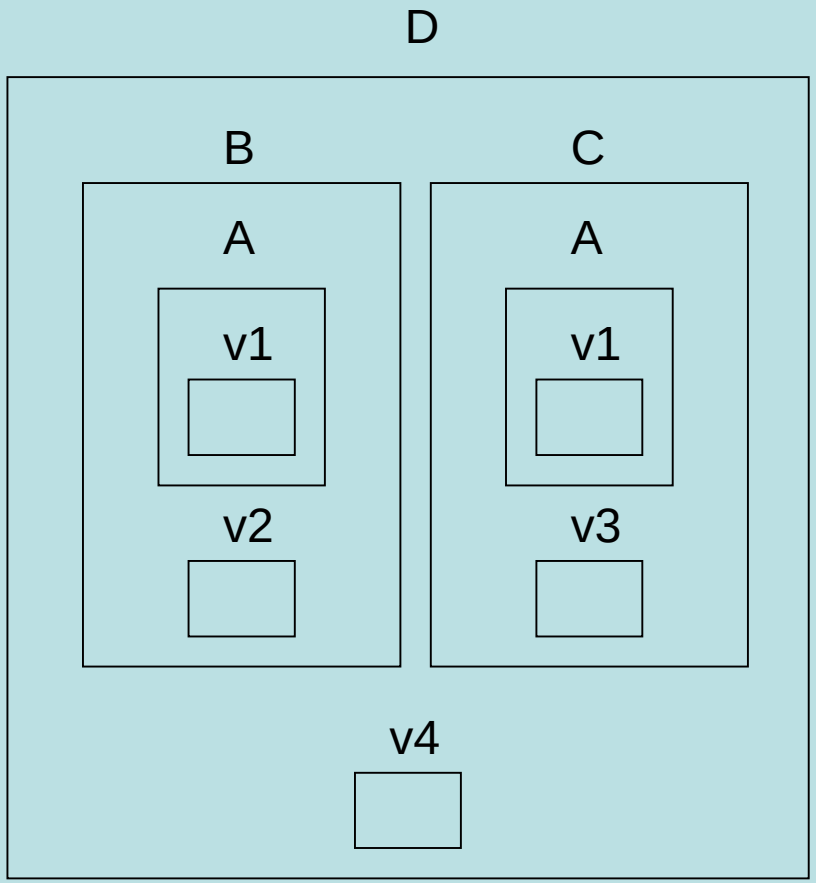
- v prípade nejednoznačnosti je nutné použiť klasifikátor triedy:

```
Odvodena d;
d.Set(10);              // chyba nejednoznačnosti
d.Zaklad1::Set(10);
```

- Konštruktor odvodenej triedy musí volať konštruktory všetkých základných tried. Ak tieto majú bezparametrický konštruktor, netreba ho volať explicitne.

# Virtuálne základné triedy

```
class A {
public:
    int v1;
};
class B : public A {
public:
    double v2;
};
class C : public A {
public:
    float v3;
};
class D : public B, public C {
public:
    char v4;
};
int main()
{
    D obj;
    obj.v4 = ´a´;
    obj.v3 = 3.14;
    obj.v2 = 1.8;
    obj.v1 = 0; // nejednoznačné
}
```




KST

# Virtuálne základné triedy

- ak chceme zamedziť, aby sa pri viacnásobnej dedičnosti v rodovom strome tá istá trieda vyskytla viackrát, deklarujeme ju pri vytváraní odvodenej triedy ako virtuálnu

```
class B : virtual public A {
public:
  double v2;
};
class C : virtual public A {
public:
  float v3;
};
```

KST