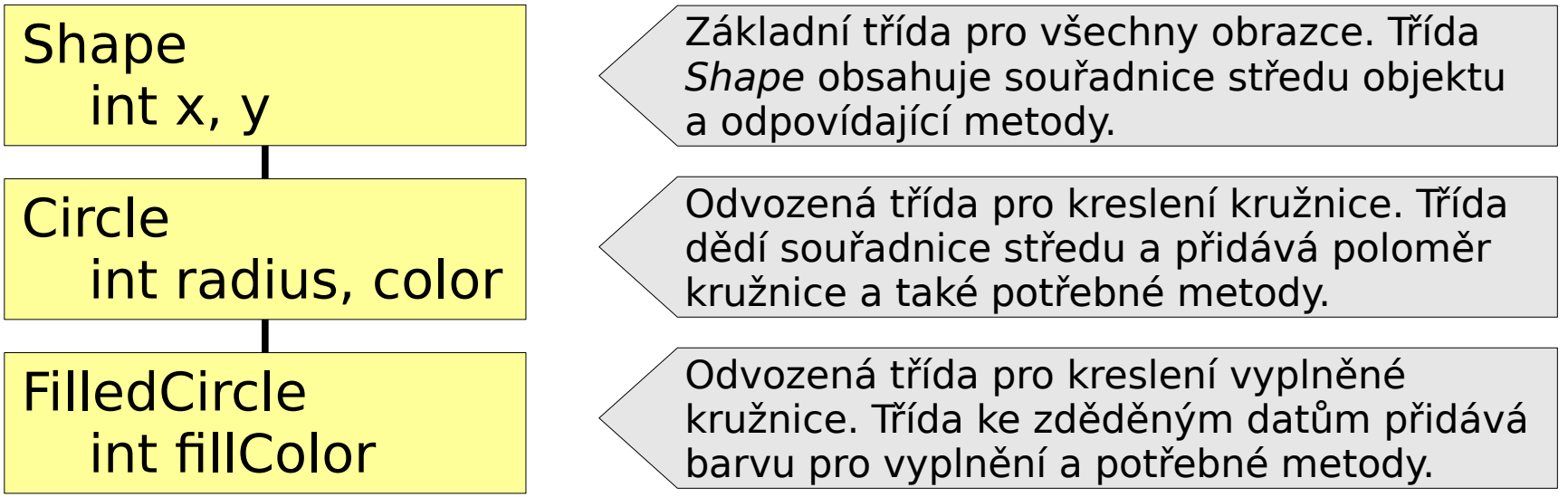
Pokročilé programování v jazyce C pro chemiky

(C3220)

Dědičnost tříd v C++

Dědičnost tříd

- Dědičnost umožňuje vytvářet nové třídy z tříd existujících tak, že odvozené třídy (tzv. potomci, *derived class*) dědí vlastnosti základních tříd (tzv. předek nebo rodič, *base class*)
- Potomci mají přístup k datovým členům a metodám předků, navíc k ním přidávají vlastní členy třídy
- Děděním je vytvářena hierarchie tříd
- Dědičnost přináší úsporu programátorské práce, protože při vytváření potomka můžeme využívat již existujících metod předka



Definice třídy potomka

- Základní třídu definujeme standardním způsobem
- Odvozená třída obsahuje následující záhlaví:

```
class Odvozena : public Zakladni
```

```
// Zakladni trida
class Shape
{
  public:
    void setCentre(int ax, int ay);
  private:
    int x, y;    // Souradnice stredu grafickeho objektu
};

// Odvozena trida dedi data a metody zakladni tridy
class Circle : public Shape
{
  public:
    void setRadius(int r);
  private:
    int radius;    // Polomer kruznice
    int color;     // Barva kruznice
};
```

Přístup ke členům předka

- Členy třídy (data a metody) mohou být specifikovány v sekcích **`public`**, **`protected`** nebo **`private`**, přístup ke členům v jednotlivých sekcích je následující:

Přístup	`public`	`protected`	`private`
metody téže třídy	ANO	ANO	ANO
metody potomků	ANO	ANO	NE
metody ostatních tříd a globální funkce	ANO	NE	NE

- Členy deklarované jako **`protected`** jsou přístupné v metodách vlastní třídy a metodách potomka, nikoliv však v metodách jiných tříd nebo v nečlenských funkcích (pro ně jsou přístupné pouze členy deklarované jako **`public`**)
- Potomek dědí data i metody nejen od přímých předků, ale též od prapředků, prapraprapředků atd.

Přístup ke členům předka - příklad

```
class Shape   // Zakladni trida
{
  public:
    void setCentre(int ax, int ay);
  protected:
    void printCentre();
  private:
    int x, y;    // Souradnice stredu grafickeho objektu
};

class Circle : public Shape    // Circle je odvozena trida
{
  public:
    void printValues()
      { printCentre(); /* Vola se zdedena metoda */ };
};

int main()
{
  Circle circ;
  circ.setCentre(10, 30); // Metoda setCentre() je zdedena
  circ.printValues();
  // Nasledujici by nefungovalo, protoze se jedna o protected metodu:
  circ.printCentre();
}
```

Překrytí metod předka

- V odvozené třídě můžeme deklarovat metodu se stejným jménem (a parametry a návratovým typem) jako v předkovi
- Pokud takovou metodu zavoláme pro objekt odvozené třídy, zavolá se metoda odvozené třídy (tato metoda tedy překryje metodu v předkovi)
- Pokud chceme zavolat překrytou metodu předka, musíme před jejím názvem uvést název třídy předka oddělený pomocí dvou dvojteček (např. `Predek::metoda_predka()`).

Překrytí metod předka – příklad 1

```
class Shape
{
  public:
    void printValues()
      { cout << "Centre: " << x << ", " << y << endl; };
  private:
    int x, y;
};

class Circle : public Shape
{
  public:
      // Tato metoda prekryva metodu predka printValues()
    void printValues()
      { cout << "Radius, color: " << radius << "," << color << endl;};
  private:
    int radius, color;
};

int main()
{
  Shape s;
  Circle circ;
  s.printValues();             // Vola se metoda tridy Shape
  circ.printValues();          // Vola se metoda tridy Circle
  circ.Shape::printValues();   // Vola se metoda tridy Shape
}
```

Překrytí metod předka – příklad 2

```
class Shape
{
  public:
    void printValues()
      { cout << "Center: " << x << ", " << y << endl; };
  private:
    int x, y;
};

class Circle : public Shape
{
  public:
    // Nasledujici metoda prekryva metodu predka se stejnym jmenem
    void printValues()
    {
      Shape::printValues(); // Volame metodu predka
      // pokud bychom volali pouze printValues(), zavolala
      // by se rekurzivne zase metoda Circle::printValues()
      cout << "Radius: " << radius << endl;
      cout << "Color: " << color << endl;
    };

  private:
    int radius, color;
};
```

Konstruktory a dědičnost

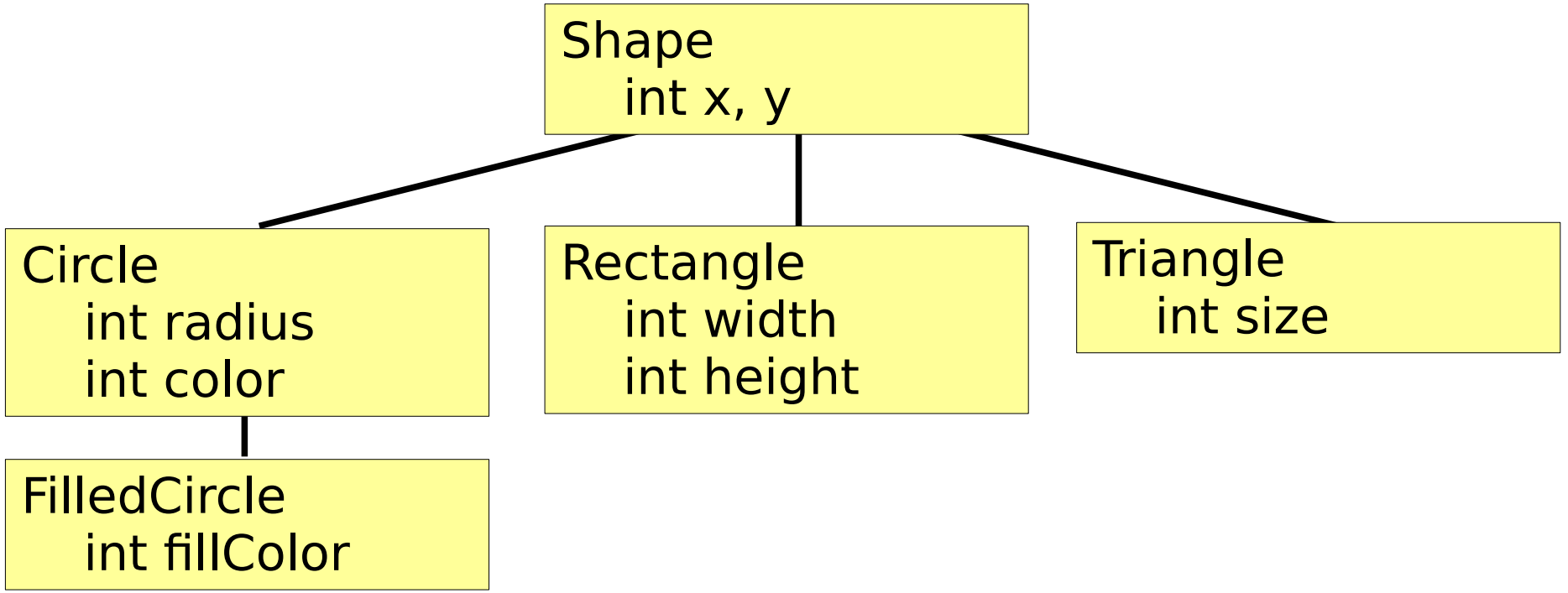
- Při vytváření instance odvozené třídy je nejdříve zavolán konstruktor základní třídy, teprve potom konstruktor odvozené třídy – tímto je zajištěno, že v okamžiku kdy se začne vykonávat kód konstruktoru potomka, jsou již datové členy předka plně inicializovány
- Konstruktor základní třídy lze explicitně zavolat v inicializačním seznamu konstruktoru odvozené třídy (tím můžeme předat libovolné argumenty a ovlivnit tak konstrukci základní třídy)
- Konstruktor základní třídy uvádíme v inicializačním seznamu na prvním místě (odpovídá skutečnému pořadí inicializace členů)
- Pokud konstruktor základní třídy nezavoláme, překladač sám zavolá výchozí (bezparametrový) konstruktor
- Analogicky se automaticky volají i destruktory (v opačném pořadí konstrukce, tedy napřed odvozený a po něm základní)

Konstruktory a dědičnost – příklad

```
class Shape
{
  public:
    Shape(int ax, int ay) : x(ax), y(ay) {};
  private:
    int x, y;
};

class Circle : public Shape
{
  public:
    Circle(int ax, int ay, int r, int c);
  private:
    int radius, color;
};

Circle::Circle(int ax, int ay, int r, int c) : Shape(ax, ay), radius(r), color(c)
{
}

int main()
{
  Circle circ(20, 30, 5, 1); //Argumenty se predaji konstruktoru ve tride
        // Circle, ten ale nejdrive preda nektere z nich konstruktoru v
        // tride Shape, pote se vykona kod konstruktoru Shape()
        // a pak teprve kod konstruktoru Circle()
}
```

Více potomků odvozených z jednoho předka

- Z jednoho předka lze odvozovat libovolný počet potomků

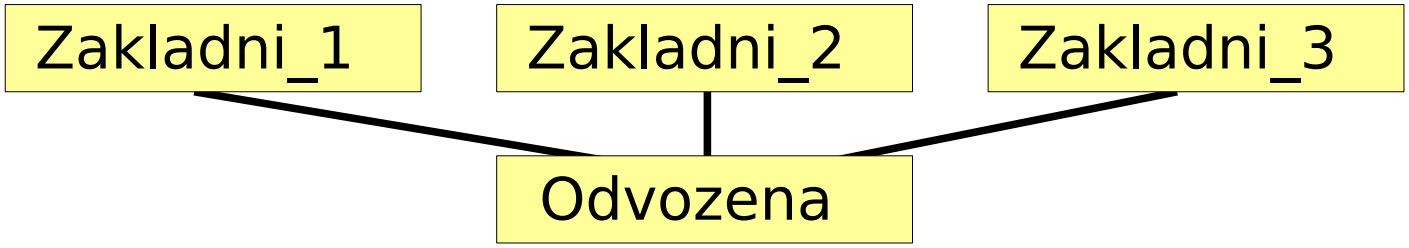


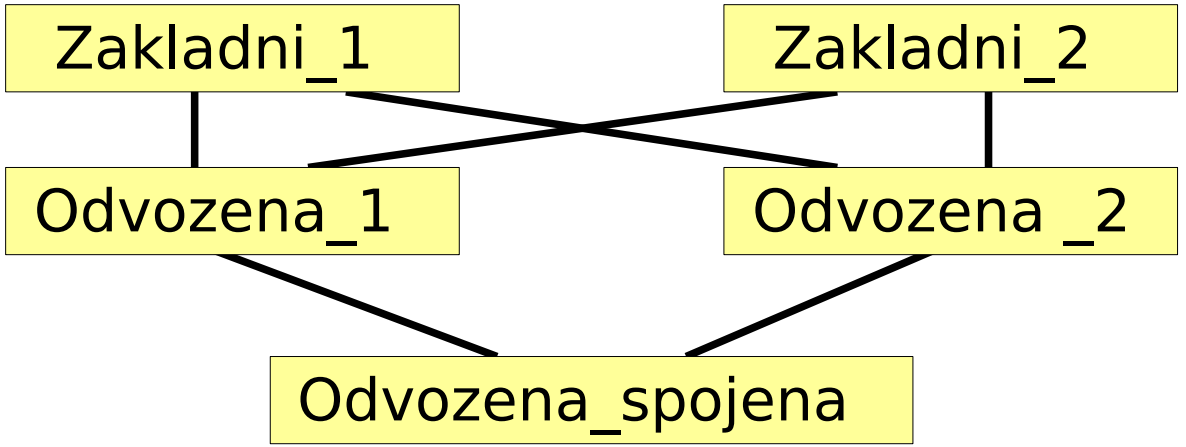
Vícenásobná dědičnost

- Potomky lze odvodit z více než jedné základní třídy – mluvíme o vícenásobné dědičnosti
- Deklarace třídy s vícenásobnou dědičností:

```
class Odvozena : public Zakladni_1, public Zakladni_2
```

- Vícenásobná dědičnost se používá při vývoji knihoven, v běžných programech se s ní téměř nesetkáme (její používání se nedoporučuje, není-li pro ni dobrý důvod)

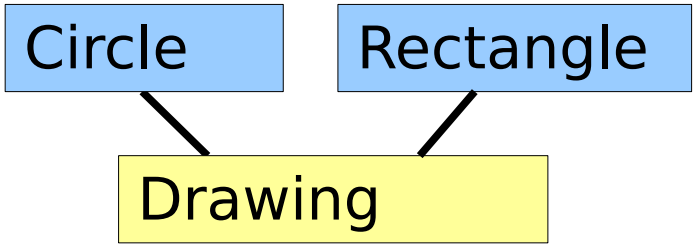




Dědičnost versus skládání

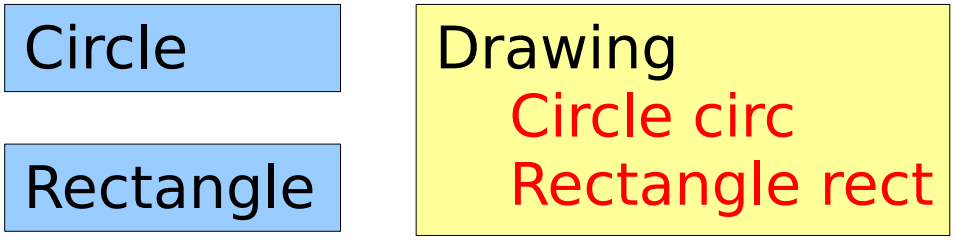
- Při tvorbě programů je někdy potřeba rozhodnout, zdali použijeme k propojení tříd dědičnost, nebo skládání (do třídy vložíme objekt jiné třídy jako její člen)
- Pokud si nejsme jisti, který přístup je v daném případě lepší, upřednostňujeme většinou skládání
- Objekt odvozujeme pomocí dědičnosti pouze v případě, že jejich vztah odpovídá větě: objekt_odvozeny **je** objekt_zakladni („Pes je Savec“ (dědičnost), ne však „Auto je Motor“ (skládání))

Chybné použití dědičnosti:




```
class Drawing : public Circle,
                public Rectangle
{
};
```

Správné použití skládání:




```
class Drawing
{
  private:
    Circle circ;
    Rectangle rect;
};
```

Dodržujte následující pravidla

- Při tvorbě programu postupujte pomalu. Nejdříve vytvořte základní třídu, teprve později přidejte odvozenou třídu. Vytvořte vždy nejdříve základní kostru třídy (proměnné s výchozími hodnotami, případně konstruktor bez parametrů) a funkci `main()`, program přeložte a opravte chyby. Potom postupně přidávejte jednotlivé metody, pokaždé přeložte (překladač nesmí hlásit chyby). Pak přidejte odvozenou třídu.
- Všechny členské proměnné základních typů inicializujte vhodnou hodnotou (ideálně přímo v definici, jinak v inicializačním seznamu v konstruktoru, výjimečně v těle konstruktoru). **Členské proměnné objektového typu není třeba zvlášť inicializovat, protože jejich konstruktor se zavolá automaticky a inicializaci zajistí.**
- Při načítání dat od uživatele a při výpisu hodnot vždy vypisujte vhodný informační text, aby uživatel věděl, jaká data má zadat a jaké informace program vypsal.
- Metody neměnící data třídy (`get*()`) označujte jako konstantní.

Cvičení – 1. část

1. Vytvořte program obsahující základní třídu Shape a z ní odvozenou `Circle` s následujícími vlastnostmi:
   - Třída Shape bude obsahovat:
     - souřadnice středu grafického objektu (`int x, y;`)
     - konstruktor s parametry pro inicializaci souřadnic středu:
       `Shape(int ax, int ay)`
     - metodu `setValues(int ax, int ay)` pro nastavení souřadnic středu
     - metody `getCentreX()` a `getCentreY()` pro získání souřadnic středu
     - metodu `printValues()` pro výpis hodnot datových položek třídy (tj. hodnot `x` a `y`).
   - Třída `Circle` bude obsahovat:
     - hodnotu poloměru kružnice (`int radius;`) a číslo barvy kružnice (`int color;`)
     - konstruktor s parametry pro inicializaci souřadnic středu, poloměru a čísla barvy (souřadnice středu bude předávat konstruktoru předka):
       `Circle(int ax, int ay, int r, int c)`
     - metodu `setValues(int ax, int ay, int r, int c)` pro nastavení souřadnic středu, poloměru a čísla barvy (pro nastavení souřadnic středu bude tato metoda volat metodu předka).
     - metodu `getRadius()` pro získání hodnoty poloměru kružnice a `getColor()` pro získání čísla barvy
     - metodu `printValues()` pro výpis hodnot datových položek třídy (pro výpis souřadnic středu bude tato metoda volat metodu předka)
     - metodu `draw()` pro vykreslení kružnice pomocí knihovny g2
   - Ve funkci `main()` vytvořte objekt typu `Circle` a inicializujte jeho souřadnice a poloměr vhodnými hodnotami. Vypište na výstup hodnoty objektu zavoláním jeho metody `printValues()`, pak ještě zavolejte přímo jeho metodu `printValues()` zděděnou z předka Shape (vypíše jen souřadnice středu). Potom zavolejte metodu `setValues()`, která nastaví nové hodnoty (souřadnic středu a poloměru) a ty se následně opět vypíší pomocí `printValues()` na výstup. Nakonec zavolá metodu `draw()`.

2 body

Cvičení – 2. část

2. Vytvořte program, který bude vycházet z předchozí úlohy, ale s následujícími úpravami:
   - Vytvořte třídu `Rectangle` (odvozenou z Shape) pro kreslení obdélníku. Třída bude mít členy `width` a `height` (typu `int`) a související metody podobné jako ve třídě `Circle` (tj. `setValues()`, `getWidth()`, `getHeight()`, `printValues()` a `draw()`). Obdélník se bude vždy vykreslovat černou barvou. Pro vykreslení obdélníku použijte funkci `g2_rectangle (int dev, double x1, double y1, double x2, double y2)`.
   - Vytvořte třídu `FilledCircle` (odvozenou z `Circle`) pro kreslení kružnice vyplněné barvou. Třída bude mít člen `fillColor` pro číslo barvy (typu `int`) a související metody podobné jako ve třídě `Circle` (tj. `setValues()`, `getFillColor()`, `printValues()` a `draw()`) (Pozn.: při vykreslování vyplněné kružnice se musí nejdříve nastavit barva výplně pomocí `g2_pen()` a vykreslit vyplněná kružnice pomocí `g2_filled_circle()`, pak nastavit barva obrysu kružnice také pomocí `g2_pen()` a kreslit obrys kružnice pomocí `g2_circle()`).
   - V každé třídě dále implementujte metodu `readValues()`, která načte ty právě ty hodnoty, které daná třída potřebuje (a předtím volá obdobnou metodu předka pro načtení hodnot vyžadovaných předkem).

   Program nabídne uživateli možnost kreslit kružnici, čtverec nebo vyplněnou kružnici. Potom si od něj vyžádá příslušné hodnoty souřadnic, poloměru, barvy atd. (voláním metody `readValues()`). Na konec na obrazovku terminálu vypíše načtené hodnoty (voláním `printValues()`) a vykreslí příslušný objekt. **2 body**

Cvičení – 3. část

3. Vytvořte program který vykreslí obrazec se třemi vyplněnými kružnicemi ohraničenými obdélníkem (viz. obrázek níže), barva výplně kružnic bude specifikována uživatelem. Program bude vycházet z předchozí úlohy. V programu implementujte novou třídu `Drawing`, která bude obsahovat 3 objekty vyplněné kružnice (typu `FilledCircle`) a jeden objekt obdélníku (typu `Rectangle`). Dále bude obsahovat:
   - konstruktor `Drawing()`, který vytvoří všechny obrazce ve vhodných pozicích
   - metodu `readValues()` pro načtení barvy výplně
   - metodu `setFillColor(int fc)` pro nastavení barvy výplně kružnic
   - metodu `draw()`, která otevře okno a vykreslí do něj tři vyplněné kružnice a ohraničující obdélník.

   Podle potřeby upravte existující třídy (např. do třídy `FilledCircle` doplňte metodu `setFillColor(int fc)`, okno pro kreslení bude otevřeno v `Drawing::draw()`, a metody `draw()` grafických tříd budou přijímat parametr s číslem okna a budou provádět pouze operace kreslení do tohoto okna). Funkce `main()` pouze vytvoří instanci `Drawing` a zavolá `readValues()` a `draw()`. **1 bod**