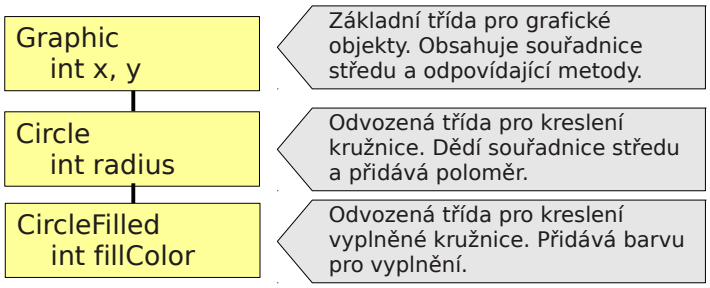
# 3. Dědičnost tříd v C++

## Dědičnost tříd

Dědičnost umožňuje vytvářet nové třídy z tříd existujících tak, že odvozené třídy (tzv. potomci) dědí vlastnosti základních tříd (tzv. předek nebo rodič).

Potomci mají přístup k datovým členům a metodám předků, navíc k ním přidávají vlastní členy třídy.




### Definice třídy potomka

Základní třídu definujeme standardním způsobem.

Odvozená třída obsahuje následující záhlaví:

```
class Odvozena : public Zakladni
```

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

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

## Přístup ke členům předka

Členy třídy (data a metody) mohou bý 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 globálních funkcí | ANO | NE | NE |

Členy deklarované jako `protected` jsou přístupné v metodách vlastní třídy a v metodách potomků, nikoliv však v metodách jiných tříd nebo nečlenských funkcích (pro ně jsou přístupné pouze členy deklarované jako public).

Potomek dědí data i metody nejen od přímých předků ale též od prapředků (a jejich předků atd...).

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

// Trida Circle je odvozena z Graphics
class Circle : public Graphic
{
  public:
    void printValues();
};


void Circle::printValues()
{
  // Vola se zdedena metoda
  printCentre();
}

int main()
{
  Circle circ;

  // Metodu setCentre() zdedila trida
  // Circle z predka Graphic
  circ.setCentre(10, 30);

  // Nasledujici by nefungovalo, protoze
  // se jedna o protected metodu.
  // Prekladac by ohlasil chybu
  circ.printCentre();

  circ.printValues(); // Vola se verejna
                      // metoda
}
```

## Překrytí metod předka

V odvozené třídě můžeme deklarovat metodu se stejným jménem 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()`).

```
// Ukazka prekryti metody predka,
// bude prekryta metoda printValues()
// a ve funkci main() budou volany jeji
// jednotlive verze

class Graphic
{
  public:
    void printValues();
  private:
    int x, y;
};

class Circle : public Graphic
{
  public:
    // Nasledujici metoda prekryva metodu
    // predka se stejnym jmenem
    void printValues();
  private:
    int radius;
    int color;
};

void Graphic::printValues()
{
  cout << "Centre: " << x << ", "
       << y << endl;
}

void Circle::printValues()
{
  cout << "Radius: " << radius << endl;
  cout << "Color: " << color << endl;
}

int main()
{
  Graphic gra;
  Circle circ;

  // Bude se volat metoda ve tride Graphic
  gra.printValues();

  // Bude se volat metoda ve tride Circle
  circ.printValues();

  // Bude se volat metoda ve tride Graphic
  circ.Graphic::printValues();
}
```

```
// Ukazka prekryti metody predka,
// bude prekryta metoda printValues()
// a ve tride Circle bude z metody
// printValues() volana metoda
// printValues() predka Graphic

class Graphic
{
  public:
    void printValues();
  private:
    int x, y;
};

class Circle : public Graphic
{
  public:
    // Nasledujici metoda prekryva metodu
    // predka se stejnym jmenen
    void printValues();
  private:
    int radius;
    int color;
};

void Graphic::printValues()
{
  cout << "Center: " << x << ", "
       << y << endl;
}

void Circle::printValues()
{
  // Bude se volat metoda v tride Graphic
  Graphic::printValues();

  cout << "Radius: " << radius << endl;
  cout << "Color: " << color << endl;
}
```

## Konstruktory a dědičnost

Při vytvoření objektu 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 inicializovány.

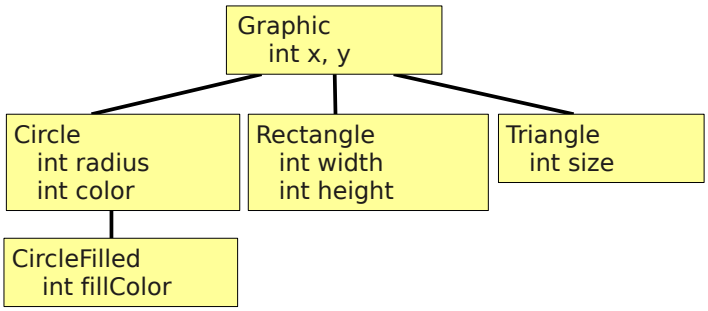
Do konstruktrou základní třídy lze předat argumenty které byly předány konstruktoru odvozené třídy (viz. následující příklad).

```
// Priklad demonstruje, ze konstruktor
// zakladni dridy je volan nejdrive a az
// pak konstruktor odvozene tridy

class Graphic
{
  public:
    Graphic() { x = 0; y = 0; };
  private:
    int x, y;
};

class Circle : public Graphic
{
  public:
    Circle() { radius = 0; color = 0;};
  private:
    int radius, color;
};

int main()
{
  Circle circ;   // Nejdrive se zavola
  // konstruktor zakladni tridy Graphic,
  // teprve potom konstruktor tridy Circle
}
```

```
//Ukazka predavani agrumentu konstruktorum

class Graphic
{
  public:
    Graphic(int ax, int ay)
          { x = ax; y = ay; };
  private:
    double x, y;
};

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

Circle::Circle(int ax, int ay,
      int r, int c) : Graphic(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 Graphic, pote se
  // vykona kod v konstruktoru Graphic()
  // a nakonec kod v konstruktoru Circle()
}
```

## Více potomků odvozených z jednoho předka

Z jednoho předka lze odvozovat libovolný počet potomků.




## Dědičnost vs. člen třídy

Při tvorbě programů je někdy potřeba rozhodnout, zda-li použijeme k propojení tříd dědičnost, nebo do třídy vložíme objekt jako její člen

Pokud si nejsme jisti který přístup je v daném případě lepší, upřednostňujeme v praxi druhou možnost

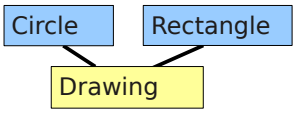
Dědičnost používáme pouze v dobře odůvodněných případech

Objekt odvozujeme pomocí dědičnosti pouze v případě, že jejich vztah odpovídá větě:

objekt_odvozeny **je** objekt_zakladni

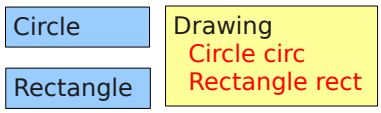
např.: Kružnice (`Circle`) **je** grafický objekt (`Graphic`).
Vyplněná kružnice (`FilledCircle`) **je** kružnicí (`Circle`).
Kresba (`Drawing`) **není** kružnicí ani obdélníkem.

**Nevhodný způsob použití dědičnosi:**




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

**Vhodný přístup bez použití dědičnosi:**




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

## Dodržujte následující pravidla

- Při tvorbě programu postupujte pomalu. Nejdříve vytvořte základní třídou teprve později přidejte odvozenou třídu. Vytvořte vždy nejdříve základ třídy (proměnné a konstruktor bez parametrů) a funkci `main()` a program přeložte, opravte chyby. Potom postupně přidávejte jednotlivé metody, pokaždé přeložte (překladač nesmí hlásit chyby). Pak přidejte odvozenou třídu.
- V každé třídě vytvořte konstruktor a v něm inicializujte členská data (na základě předaných parametrů nebo vhodnými implicitními hodnotami). Pokud by třída neměla konstruktor, členská data zůstala neinicializovaná. **Členská data objektového typu není třeba inicializovat, protože jejich konstruktor se zavolá automaticky a v něm budou inicializována.**
- 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.

## Úloha 1 — 2 body

Vytvořte program obsahující základní třídu `Graphic` a z ní odvozenou `Circle` s následujícími vlastnostmi:

- Třída `Graphic` bude obsahovat:
  - souřadnice středu grafického objektu (`int x, y;`)
  - konstruktor s parametry pro inicializaci souřadnic středu: `Graphic(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 `Graphic` (vypíše jen souřadnice středu). Potom zavolejte metodu `setValues()` která nastaví nové hodnoty (souřadnic středu a poloměru) a opět se vypíší pomocí `printValue()` na výstup. Nakonec zavolá metodu `draw()`.

## Úloha 2 — 2 body

Vytvořte program který bude vycházet z předchocí úlohy, ale implementujte následující modifikace:

- Vytvořte třídu `Rectangle` (odvozenou z `Graphic`) 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 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 mačtení 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.

**Nápověda:**

Pro kreslení obdélníku použijte funkci knihovny g2:

```
void g2_rectangle(int dev, double x1,
double y1, double x2, double y2);
```

kde `x1` a `y1` jsou souradnice leveho horniho rohu a `x2`, `y2` pravého dolniho rohu.

Pro kreslení vyplněné kružnice použijte funkci:

```
void g2_filled_circle(int dev, double x,
double y, double r);
```

Aby byla vyplněná kružnice ohraničená, je potřeba přes ni ještě nakreslit nevyplněnou kružnici (a před tím nastavit barvu okraje).

## Úloha 3 — 1 bod

Vytvořte program který vykreslí obrazec se třemi vyplněnými kružnicemi ohraničenými obdélníkem (viz. obrázek níže), barva výplně kružnic bude specifikována uživatelem. Program bude vycházet z předchocí 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()`
- metodu `readValues()` pro načtení barvy výplně
- metodu `setFillColor(int fc)` pro nastavení barvy výplně všech tří kružnic
- metodu `draw()`, která otevře okno a vykreslí do něj tři vyplněné kružnice a ohraničující obdélník.

Podle potřeby upravte existující třídy (např. budou potřebovat konstruktory bez parametrů, 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).