

**Pokročilé programování  
v jazyce C pro chemiky  
(C3220)**

**Třídy v C++**

# Třídy v C++

- Třídy jsou uživatelsky definované typy podobné strukturám v C, kromě datových položek (proměnných) však mohou obsahovat i funkce, sloužící k manipulaci s datovými položkami dané třídy
- Proměnné definované uvnitř třídy se nazývají **členská data**, funkce se nazývají **metody**, společně se označují jako členy třídy
- Třídy definujeme pomocí klíčového slova **class**, za kterým uvádíme název třídy, potom následují složené závorky, mezi kterými jsou uvedeny členy třídy
- Definice třídy je **ukončena středníkem**

```
class Circle          // Trida reprezentujici kruznici
{
    public:
        int x, y;      // Souradnice stredu kruznice
        int radius;   // Polomer kruznice

        void printValues();    // Deklarace metody
        void draw();          // Deklarace dalsi metody
};                          // Na konci musi byt strednik
```

# Definice metod třídy

- Metody se definují podobně jako funkce, jejich příslušnost ke třídě se vyjádří uvedením jména třídy před jménem funkce oddělené dvěma dvojtečkami ::

```
class Circle      // Třída reprezentující kružnici
{
    public:
        int x, y;      // Souradnice stredu kruznice
        int radius;    // Polomer kruznice

        void printValues();    // Deklarace metody
        void draw();           // Deklarace dalsi metody
};    // Na konci musi byt strednik

void Circle::printValues()
{
    // Zde bude kod pro vypsani souradnic stredu a polomeru
}

void Circle::draw()
{
    // Zde bude kod pro vykresleni kruznice
}
```

# Instance třídy

- Třída je datový typ a proto se jménem třídy pracujeme jako se jmény základních datových typů (int, char, ...)
- Jméno třídy lze použít k definici proměnné, taková proměnná se nazývá **instance třídy**, **objekt** nebo **objektová proměnná**
- K datovým položkám a k metodám přistupujeme pomocí **operátoru tečka**

```
// Na začátku programu je definována třída Circle

int main()
{
    Circle circ; // Definice proměnné, instance třídy (tj. objektu)

    circ.x = 120;
    circ.y = 150;
    circ.radius = 80;

    circ.draw(); // Voláme metodu draw()

    cout << "Souradnice stredu: " << circ.x << ", " << circ.y << endl;

    return 0;
}
```

# Přístup ke členům třídy z metod

- Uvnitř metod lze ke členům dané třídy přistupovat přímo

```
class Circle
{
    public:
        int x, y;
        int radius;

        void printValues();
        void draw();
};

void Circle::draw()
{
    printValues(); // Volame metodu dane tridy
    // Zde by byl dalsi kod pro vykresleni kruznice
}

void Circle::printValues()
{
    cout << "Stred kruznice: " << x << ", " << y << endl;
    cout << "Polomer kruznice: " << radius << endl;
}
```

# Metody s parametry

- Metody mohou přijímat parametry stejně jako ostatní funkce
- Názvy parametrů volíme odlišné od členů třídy

```
class Circle
{
    public:
        int x, y;
        int radius;
        void setCentre(int ax, int ay);
        void printValues();
        void draw();
};

void Circle::setCentre(int ax, int ay)
{
    x = ax;
    y = ay;
}

int main()
{
    Circle circ; // Definice promenne, instance tridy (tj. objektu)
    circ.setCentre(120, 150);
    circ.printValues();
    return 0;
}
```

# Více instancí třídy

- Pro každý objekt (tj. instanci třídy) je v paměti alokován prostor pro datové členy; jednotlivé metody pracují nad daty objektu, na němž jsou volány
- Metody proto nikdy nemůžeme volat samostatně, ale voláme je vždy ve spojení s objektem, nad jehož daty mají operovat

```
// Na začátku je definována třída Circle a její metody

int main()
{
    Circle circ1, circ2; // Pro každou ze dvou objektových
                          // promenných se v paměti alokuje místo
                          // pro datové členy x, y, a radius

    // Následující dvě metody budou operovat nad daty objektu circ1
    circ1.setCentre(100, 90);
    circ1.draw();

    // Následující dvě metody budou operovat nad daty objektu circ2
    circ2.setCentre(200, 120);
    circ2.draw();

    return 0;
}
```

# Veřejné a soukromé členy třídy

- Členy třídy rozdělujeme na veřejné a soukromé; v definici třídy je rozlišujeme pomocí klíčových slov **public** a **private**
- K soukromým členům lze přistupovat pouze z metod dané třídy
- K veřejným členům lze přistupovat i zvenčí, tedy z libovolných jiných funkcí nebo metod

```
class Circle
{
    public:           // Verejne cleny tridy
    void setCentre(int ax, int ay);
    void draw();

    private:        // Soukrome cleny tridy
    int x, y;
    int radius;
    void printValues();
};
```



# Veřejné a soukromé členy třídy

```
class Circle
{
    public:           // Verejne clen tridy
    void setCentre(int ax, int ay);
    void draw();

    private:        // Soukrome clen tridy
    int x, y;
    int radius;
    void printValues();
};

int main()
{
    Circle circ;
    circ.setCentre(120, 150);    // setCentre() je verejna metoda
    circ.draw();                // draw() je verejna metoda

    // Nasledujici by nefungovalo!!! Prekladac by ohlasil chybu!
    circ.radius = 80;         // radius je soukromy clen tridy
    circ.printValues();       // printValues() je soukromy clen tridy

    return 0;
}
```

# Veřejné vs. soukromé členy třídy

- Jako veřejné ponecháme pouze ty metody, ke kterým potřebujeme přistupovat z funkcí nebo z metod jiných tříd, ostatní ponecháme soukromé
- **Datové položky ponecháváme téměř vždy jako soukromé**, pro nastavení nebo získání jejich hodnoty použijeme veřejné metody

```
class Circle
{
    public:
    void setCentre(int ax, int ay);
    void setRadius(int r);
    int getCentreX();
    int getCentreY();
    int getRadius();

    private:
    int x, y;
    int radius;
};

void Circle::setCentre(int ax, int ay) { x = ax; y = ay; }
void Circle::setRadius(int r) { radius = r; }
int Circle::getCentreX() { return x; }
int Circle::getCentreY() { return y; }
int Circle::getRadius() { return radius; }
```

# Inicializace členů třídy

- Jednotlivým datovým členům třídy můžeme přiřadit výchozí hodnoty při jejich deklaraci v těle třídy. Každá vytvořená instance bude pak inicializována danými konstantami.

```
class Circle
{
    public:
        void draw();

    private:
        int x = 50, y = 50, radius = 10;
};

int main()
{
    Circle circ; // vytvoří kružnici s výchozími parametry

    circ.draw();
}
```

# Konstruktor

- Konstruktor je metoda, která je zavolána automaticky při definici instance třídy (nejdříve se přidělí paměť pro datové členy třídy, pak se volá konstruktor)
- **Konstruktor má vždy stejné jméno jako třída** (podle toho překladač pozná, že to je konstruktor)
- Konstruktory se využívají zejména pro **složitější inicializaci třídy** (nastavení hodnot datových členů třídy, u nichž to nelze provést přímo při deklaraci, například pokud výchozí hodnoty nejsou konstanty)
- Konstruktory nemají žádnou návratovou hodnotu

```
class Circle
{
    public:
        Circle(); // Deklarace konstrukturu
    private:
        int x, y, radius;
};

Circle::Circle() { cout << "Kruznice vytvorena!" << endl; }

int main()
{
    Circle circ; // V tomto okamziku se alokuje pamet pro promennou
                  // a pote se automaticky zavola konstruktor Circle()
}
```

# Konstruktor s parametry

- Konstruktor může přijímat parametry, které mu lze předat při definici instance třídy (předávané hodnoty uvádíme v závorkách za jménem definované proměnné)

```
class Circle
{
    public:
        Circle(int r); // Konstruktor s jedním parametrem
    private:
        int x = 50, y = 50, radius = 10;
};

Circle::Circle(int r)
{
    if (r > 0) {
        radius = r;
    } else {
        cout << "Neplatny polomer, pouzivam vychozi" << endl;
    }
}

int main()
{
    Circle circ(80); // Definice promenne s inicializaci.
                    // Parametr je predan konstrukturu.
}
```

# Inicializace datových členů konstruktoru

- Jednoduchou inicializaci datových členů (například parametry konstruktoru) provádíme v **inicializačním seznamu** (*initializer list*): za dvojtečkou za seznamem parametrů
- Datové členy uvádíme v tom pořadí, jak jdou za sebou v definici třídy
- Inicializační seznam se provádí před vstupem do těla konstruktoru. V těle konstruktoru pak můžeme provádět složitější operace.

```
class Circle
{
public:
    Circle(int ax, int ay, int r); // Konstruktor s parametry
private:
    int x, y, radius;
};

// Inicializacni seznam
Circle::Circle(int ax, int ay, int r) : x(ax), y(ay), radius(r)
{
    if (radius <= 0) {
        cout << "Neplatny polomer!" << endl;
    }
}

int main()
{
    Circle circ(120, 150, 80);
}
```

# Destruktor

- Destruktor je metoda, která je automaticky zavolána při rušení instance třídy (lokální proměnné jsou rušeny po opuštění bloku/funkce, v němž jsou definovány, globální proměnné při ukončení programu)
- Jméno destrukturu je tvořeno znakem ~ a **jménem třídy** (podle toho překladač pozná že to je destruktore)
- Destruktory se využívají zejména pro uvolnění zdrojů spravovaných objektem (zavření souborů a pod.)
- Destruktory nemají žádnou návratovou hodnotu a nemohou přijímat žádné parametry

```
class Circle
{
public:
    Circle();           // Konstruktor
    ~Circle();         // Destruktor
private:
    int x, y, radius;
};

Circle::~Circle()
{
    // Tady muze byt kod napr. pro uvolneni dynamicky alokovane pameti
}
```

# Správa zdrojů, koncept RAII

- Při správě externích zdrojů (souborů, síťových spojení, atd.) v C++ se využívá koncept **RAII** (*Resource Acquisition Is Initialization*): alokaci zdroje svážeme s existencí nějakého objektu (alokujeme v konstruktoru, uvolníme v destruktoru)

```
class G2Device
{
public:
    G2Device(int width, int height);          // Konstruktor
    ~G2Device();                             // Destruktor
    int dev;  // Pro jednoduchost public, lepsi by bylo pridat metodu getDev()
};

G2Device::G2Device(int width, int height)
{
    dev = g2_open_x11(width, height);
}

G2Device::~~G2Device()
{
    g2_close(dev);
}

int main()
{
    G2Device okno(500, 500); // Tady nam konstruktor okno vytvori
    g2_circle(okno.dev, 100, 100, 10);
    // atd.
    // Pri skonceni funkce bude okno automaticky uzavreno
}
```



# Předávání instancí třídy jako parametry funkcí a metod

- S instancemi třídy se pracuje podobně jako s běžnými proměnnými, lze je předávat jako parametry do funkcí a metod

```
// Někde na začátku je definována třída Circle a její metody

// Funkce vykreslí dvě kružnice, které jsou jí předány jako
// parametr
void drawCircles(Circle circ1, Circle circ2)
{
    circ1.draw();
    circ2.draw();
}

int main()
{
    Circle circ1, circ2;

    drawCircles(circ1, circ2);

    return 0;
}
```

# Instance třídy jako návratová hodnota funkce nebo metody

- Instance třídy lze taktéž vrátit z funkce příkazem **return**

```
// Tady někde na začátku je definována třída Circle

// Funkce vrátí kružnici s větším poloměrem
Circle getLargerCircle(Circle circ1, Circle circ2)
{
    if (circ1.getRadius() > circ2.getRadius())
        return circ1;
    else
        return circ2;
}

int main()
{
    Circle circ1, circ2;
    Circle largerCirc;

    largerCirc = getLargerCircle(circ1, circ2);

    return 0;
}
```

# Funkce vs. metody

- Metody upřednostňujeme před funkcemi, zejména pokud funkce/metoda pracuje s datovými položkami dané třídy

```
class Circle
{
public:
    void printValues();
};

void Circle::printValues()    // Metoda tridy Circle
{
    cout << "Stred kruznice: " << x << ", " << y << endl;
    cout << "Polomer kruznice: " << radius << endl;
}

void printCircleValues(Circle circ)    // Funkce
{
    cout << "Stred kruznice: " << circ.getCentreX()
        << ", " << circ.getCentreY() << endl;
    cout << "Polomer kruznice: " << circ.getRadius() << endl;
}

int main()
{
    Circle circ;
    circ.printValues();    // Optimalni reseni - volani metody
    printCircleValues(circ);    // Mene vhodne reseni - volani funkce
    return 0;
}
```

# Konvence ve jménech

- Standard jazyka C++ nepředepisuje žádná pravidla pro používání velkých a malých písmen v názvech
- Existuje několik široce rozšířených neoficiálních konvencí
- Pro naše účely budeme používat následující konvenci:
  - Jména tříd začínáme velkým písmenem
  - Jména datových položek a metod začínáme malým písmenem (s výjimkou konstruktorů a destruktorů)
  - Pokud se název proměnné nebo metody skládá z více slov, začínáme každé nové slovo velkým písmenem (bez mezery)

```
class Circle
{
public:
    Circle(int ax, int ay, int r);
    ~Complex();
    void setCentre(int newX, int newY);
    int getRadius();
    int getColorNumberFromString(string str);
private:
    int x, y, radius;
    void printValues();
};
```

# Dodržujte následující pravidla

- Na konci definice třídy nezapomeňte uvádět středník.
- Vždy inicializujte všechny datové členy třídy: pokud je vhodnou výchozí hodnotou nějaká konstanta (např. nula), inicializujte přímo v definici třídy. Má-li třída konstruktor s parametry, pro jejich uložení do členů třídy použijte přednostně inicializační seznam. V těle konstrukturu provádějte jen operace, které nelze provést v inicializačním seznamu.
- Není-li to nezbytné, nepoužívejte konstruktory bez parametrů či destruktory.
- Datové členy třídy uvádějte vždy jako soukromé (tj. v sekci `private`).

# Cvičení - 1. část

1. Vytvořte program pro kreslení kružnice. V programu definujte třídu `Circle`, která bude mít následující veřejné metody:
  - `setCentre()` nastavující souřadnice kružnice
  - `setRadius()` nastavující poloměr kružnice
  - `setColor()` nastavující číslo barvy kružnice
  - `printValues()` vypisující hodnoty datových členů (souřadnice středu, poloměr, číslo barvy)
  - `draw()` vykreslující kružnici na obrazovku (pomocí knihovny `g2`)

Ve funkci `main()` definujte objektovou proměnnou pro kružnici. Potom si program si vyžádá od uživatele souřadnice středu kružnice (v pixelech), poloměr kružnice (v pixelech) a číslo barvy (1, 3, 7, 19 nebo 25). Hodnoty se načtou do lokálních proměnných a potom se nastaví příslušné hodnoty v objektové proměnné kružnice. Nakonec se zavolá metoda `printValues()`, která vypíše hodnoty a pak metoda `draw()`, která vykreslí kružnici.

**2 body**

## Cvičení - 2. část

2. Program modifikujte tak, že třída `Circle` bude mít konstruktor přijímající čtyři parametry (souřadnice středu, poloměr, číslo barvy). Při definici proměnné kružnice předejte do konstruktoru vhodné hodnoty. Dále implementujte ve třídě `Circle` metody `readCentre()`, `readRadius()` a `readColor()`, které od uživatele vyžádají a načtou příslušné hodnoty. Dále implementujte metodu `readValues()`, která postupně zavolá tři výše zmíněné metody. Tuto metodu zavolejte pro načtení dat od uživatele. **1 bod**
3. Modifikujte program z úlohy 2 tak, aby barva nebyla zadávána jako číslo ale formou textu (*black, blue, green, red, yellow*). Podobně při výpisu hodnot se barva vypíše jako text. Ve třídě `Circle` však bude hodnota barvy uchovávána i nadále jako číslo. Pro tento účel implementujte ve třídě `Circle` dvě soukromé metody `getColorNumberFromString()` a `getColorStringFromNumber()`. **nepovinná, 1 bod**

## Cvičení - 3. část

4. Vytvořte program, který si od uživatele postupně vyžádá souřadnice a poloměr pro dvě kružnice. Potom v jednom okně vykreslí tyto dvě kružnice a navíc třetí kružnici, jejíž střed bude ležet uprostřed spojnice středů těchto dvou kružnic a velikost poloměru bude průměrem z poloměrů dvou zadaných kružnic. Použijte stejnou třídu `Circle` jako v předchozí úloze. Navíc v ní implementujte metody `getCentreX()`, `getCentreY()` a `getRadius()` pro získání hodnot příslušných členských proměnných. Dále implementujte metodu `setAverageCircle()`, která přijme jako parametr dvě kružnice a z nich spočítá a nastaví hodnoty svého středu a poloměru, jak je uvedeno výše. První kružnice bude vždy zelená, druhá modrá a třetí zprůměrovaná kružnice bude červená. **1 bod**