Seminář C++

3. přednáška

Autor: doc. Ing. Jan Mikulka, Ph.D.

2023

Obsah přednášky

- Standardizace
- Rozdíly v C/C++
- Objektově orientované programování
  - Význam OOP
  - Základní pojmy
  - Vlastnosti OOP
- Třídy, členské funkce, členské proměnné

Standardizace

- I. Standard jazyka C
  - Verze autorů Brian W. Kernighan a Denis M. Ritchie (The C Programming Language)
  - Označován jako K&R, 1978
- II. Standard jazyka C
  - ANSI C, 1990, vychází z K&R
  - Přesně specifikuje množinu knihovních funkcí, které musí kompilátor obsahovat
- III. Standard jazyka C
  - Norma ANSI C byla převzata organizací ISO jako vzor pro evropskou normu
  - Současná norma ISO/IEC 9899:1999
- IV. Standard jazyka C++03
  - Autorem C++ je Bjarne Stroustrup, Vychází z ISO normy jazyka C a jazyka Algol 68 a Simula 67
  - ISO/IEC 14882:2003

Standardizace

- V. Standard jazyka C++11, C++14, C++17, C++20
  - https://en.cppreference.com/w/cpp/XX
- VI. Standard jazyka C++23
  - https://en.cppreference.com/w/cpp/23
  - Plán na 2023

Rozdíly v C/C++

- Při vývoji C++ byla zachována kompatibilita s jazykem C
- V C++ jsou některé konstrukce nepřípustné, i když v C jsou běžné
- Základním krokem od C k C++ bylo zavedení tříd/objektů
- C++ je vyšším programovacím jazykem oproti C

Rozdíly v C/C++

- Deklarace proměnných
  - C: začátek bloku
  - C++: kdekoliv
- Deklarace proměnné v cyklu **for**
  - C: před cyklem
  - C++: v definici cyklu
- Logické proměnné
  - C: pomocí **int**
  - C++: nový typ **bool**; klíčová slova **true** a **false**
- Předávání parametrů
  - C: odkazem a hodnotou
  - C++: odkazem, hodnotou a referencí
- Standardní knihovny
  - C++ má mnohem obsáhlejší standardní knihovny

Rozšíření C++

- Typ **bool**
- Typ reference
- Přetěžování funkcí a operátorů
- Operátory new, delete, new[] a delete[]
- Třídy (class), abstraktní třídy, řízená přístupu k prvkům třídy
- Obsluha výjimek
- Prostory jmen

Objektově orientované programování

- Význam OOP
  - Strukturované programování je v mnoha případech nepoužitelné.
  - Nutnost zvýšení přehlednosti kódu složitých aplikací.
  - Rozložení kódu na základní prvky; ty jsou používány jako samostatné a nezávislé objekty, které obsahují své vlastní funkce a data uvnitř objektu.
  - Kód je mnohem více znovu použitelný, bezpečný.
  - S pomocí generického programování lze použít jeden kód pro různé datové typy/objekty.

„Nevýhoda“ OOP

- Nutnost naučit se nový přistup k implementaci algoritmů
- Nutnost pochopení vztahu mezi objekty a reálným problémem
- Nutnost nového **stylu myšlení** při implementaci

Základní pojmy

- **Abstrakce** – pojem je převážně používán v souvislosti s implementací tříd. Označuje skutečnost, kdy třída je chápána jako vnějšími vlastnostmi popsaný objekt a není dále vysvětlena jeho vnitřní funkce.

- **Členská funkce** je taková, která patří k jedné třídě. Zahrnuje chování objektů této třídy. Někdy se mluví o metodě.

- **Dědění** je označení vztahů v hierarchii objektů a tříd. Třídy lze uspořádat do hierarchie a předávat (přebírat)-*dědit* tak vlastnosti (základních, rodičovských) tříd. Odvozená třída přebírá jak datové složky, tak metody. Odvozená třída představuje specializovanější objekt. Základní vlastností objektově orientovaného programování (OOP) je možnost zastoupení předka *dědicem*.

- **Instance** je vzor, příklad, případ typu třídy. Někdy se označuje jako objekt.

Základní pojmy

- **Objekt** je označení jak reálných typů, tak i objektových typů (instancí). Označení *objekt* se používá i pro oblast paměti, se kterou lze manipulovat (proměnná, konstanta, funkce, atd.)

- **Objektově orientovaný** je nazýván přístup, při kterém jsou problémy řešeny jako zavádění, určování a implementace objektů ve formě tříd a práce s nimi. Jádro práce programátora spočívá v *implementaci tříd* do řešení problému a volání *metod* z implementovaných tříd. Knihovny tříd se dají opakovaně používat, tím se práce zjednoduší a OOP se přiblíží k lidskému způsobu uvažování.

- **Polymorfismus** (mnohotvarost) je vlastnost, která umožňuje, aby jediný název/operátor byl použit jak pro více souvisejících a technicky odlišných účelů, tak pro rozdílné metody. Uvnitř třídy procesů je volba procesu dána typem dat, vně typem operandů.

- **Předefinování** je proces, při kterém se mění vlastnosti zděděné funkce v odvozené třídě. Důvodem je úprava funkce tak, aby reagovala správně na požadované volání. Předefinování je základním klíčem a krokem k realizaci *polymorfismu.*

Základní pojmy

- **Přetěžování funkcí** je pojem, kdy několik funkcí má stejné jméno, které překladač vzájemně rozlišuje podle počtu a složení a typů parametrů. Někdy se tento termín používá v souvislosti s hierarchií tříd a polymorfismem, ale je přesnější mluvíme-li o předefinování.

- **Přetěžování operátorů** je schopnost rozšířit operátory o uživatelem definované typy. Překladač určuje, kterou operaci chceme provést.

- **Třída** jako datový typ je představitelem skupiny objektů z řešeného problému se společnými vlastnostmi a způsobem chování.

- **Zapouzdření** je mechanismus, kdy objekty v programu lze popsat svými vlastnostmi (data) a chováním (metoda), je spojen dohromady kód a data v jednom objektu. To lze shrnout do třídy, která objekt reprezentuje. Tím se přistupuje k objektu z vnějšku jako k souhrnu vlastností a reakcí.

Vlastnosti OOP

- 1. zapouzdření (*encapsulation*)
  - public – veřejné
  - private – přístupné členským funkcím
  - protected – přístupné členským funkcím a public potomkům

  - rozdíl oproti struktuře

Vlastnosti OOP

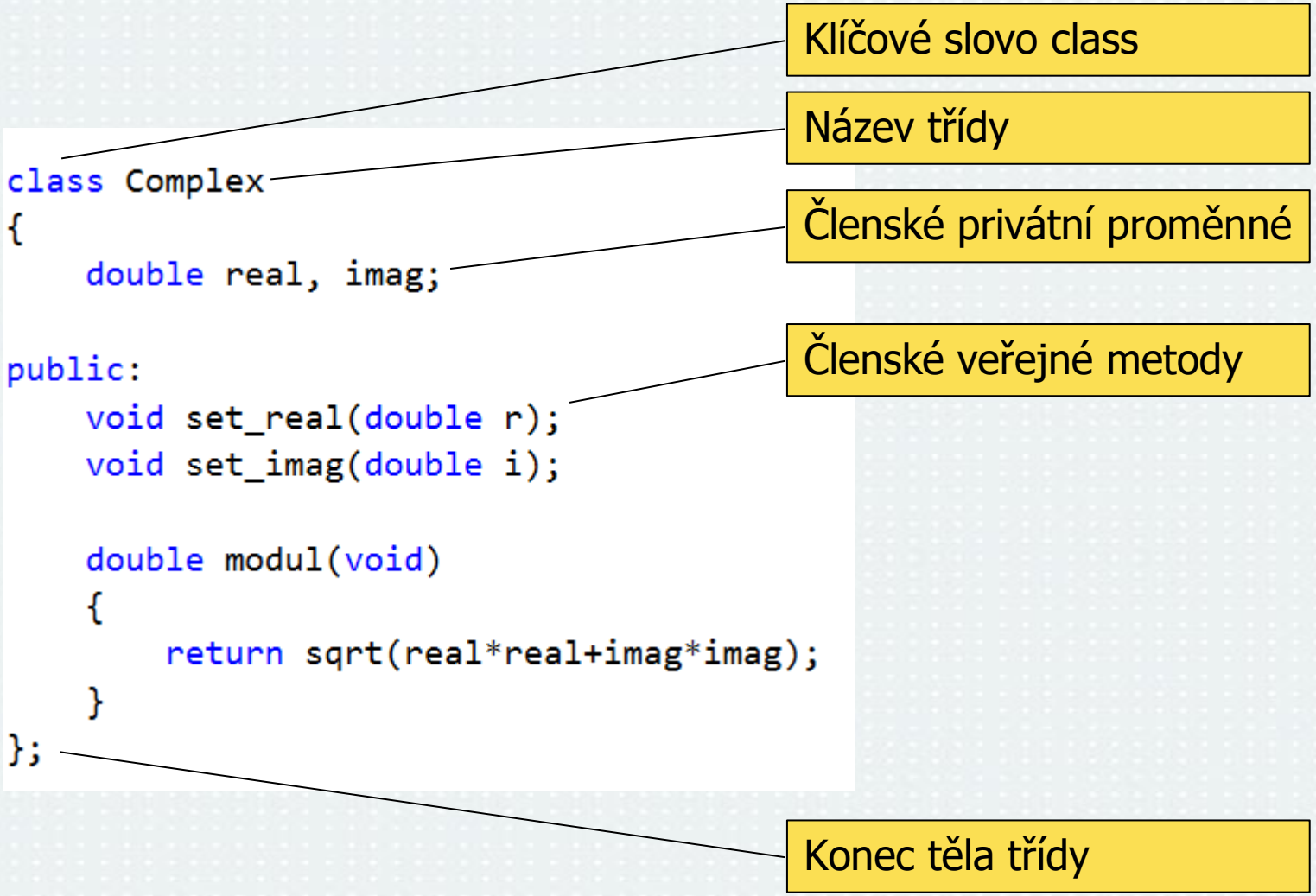
- 2. polymorfismus (*polymorphism*)
  - vlastnost jazyka, překladače, která umožňuje použití stejných názvů pro různě účely (objekty, funkce,…). Tam, kde polymorfismus nebyl povolen, hrozilo nebezpečí shody názvů interních i uživatelských a následná kolize programu. Polymorfismus dovoluje vytvářet standardní rozhraní k příslušným procesům. Někdy se polymorfismem označuje přetěžování funkcí a operátorů. Původní význam byl zaveden pro virtuální funkce.

Vlastnosti OOP

- 3. dědičnost (*inheritance*)
  - je proces, ve kterém jeden objekt může získat vlastnosti jiného procesu. Přesněji řečeno, může zdědit původní vlastnosti a doplnit je o vlastní, charakteristické pouze pro něj. Tato vlastnost OOP podporuje hierarchické uspořádání programu na místo plošného nebo lineárního uspořádání, což zkracuje kód a zpřehledňuje jej.

Třídy

- Třída slouží k **zapouzdření** dat a funkcí do jednoho (nového) typu
- Umožňují předepisovat různá **oprávnění** k přístupu k jednotlivým prvkům



Systém přidělování práv

- deklarace před první specifikací přístupových práv jsou automaticky považovány za
  - soukromé (**private**) ve třídách **class**.
  - veřejné (**public**) ve strukturách **struct** a unionech **union**.

Metody

- Nebo-li členské funkce, funkce součástí třídy
- Mohou být deklarovány jako **public**, **protected**, **private** a dále jako **virtual**, **inline**
- Mohou být přetěžovány pouze členskými funkcemi stejné nebo odvozené třídy
- Mají přístup ke všem prvkům své třídy bez ohledu na přístupová práva
- Jsou jediným prostředkem s funkcemi typu **friend**, jak získat přístup k **private** prvkům.
- Metody mohou být definovány jak uvnitř, tak vně deklarace třídy
- Metody definované uvnitř definice třídy jsou překládány jako **inline**

Definice metody vně třídy

```
class Complex
{
    double real, imag;

public:
    void set_real(double r);
    void set_imag(double i);

    double modul(void)
    {
        return sqrt(real*real+imag*imag);
    }
};

void Complex::set_real(double r)
{
    real = r;
}
```

deklarace

definice

Přístup k členům třídy

```
class Complex
{
    double real, imag;

public:
    void set_real(double r);
    void set_imag(double i);

    double modul(void)
    {
        return sqrt(real*real+imag*imag);
    }
};

void Complex::set_real(double r)
{
    real = r;
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    Complex C;

    C.set_real(5);
    cout << C.modul() << endl;

    return 0;
}
```

Přístup k členům třídy

```
class Complex
{
    double real, imag;

public:
    void set_real(double r);
    void set_imag(double i);

    double modul(void)
    {
        return sqrt(real*real+imag*imag);
    }
};

void Complex::set_real(double r)
{
    real = r;
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    Complex *C = new Complex;

    C->set_real(5);
    cout << C->modul() << endl;

    delete C;

    return 0;
}
```

Speciální metody - konstruktor

- Konstruktor slouží k vnitřní inicializaci objektu
- Konstruktor se volá při dynamické alokaci případu pomocí operátoru new
- Konstruktor má vždy stejné jméno jako jeho třída
- Konstruktor nemá žádnou návratovou hodnotu
- Pokud konstruktor nedeklarujeme, překladač v třídě sám vytvoří veřejně přístupný konstruktor bez parametrů
- Pokud explicitně deklarujeme jakýkoliv konstruktor, nebude překladač žádný další vytvářet
- Konstruktory mohou být přetěžovány
- Konstruktory se nedědí, konstruktor potomka automaticky volá konstruktory bezprostředních prvků
- Konstruktory, které mají právě jeden parametr, mohou provádět konverzi na typ své třídy

Speciální metody - destruktor

- Destruktor slouží k vnitřnímu „úklidu“ při rušení objektu
- Destruktory se volají automaticky při zániku případu, např. delete
- Destruktor nemá žádnou návratovou hodnotu
- Pokud v třídě nedeklarujeme explicitně destruktor, vytvoří si překladač veřejně přístupný standardní destruktor
- Destruktor se nedědí
- Destruktor nemá žádné parametry
- Destruktory můžeme explicitně volat

Konstruktor/destruktor

```
class Vektor
{
    double *vektor;

public:
    Vektor(int n)
    {
        vektor = new double[n];
    }
    ~Vektor()
    {
        delete [] vektor;
    }
};
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    Vektor v(10), *p;

    p = new Vektor(15);

    delete p;

    return 0;
}
```

Ukazatel this

- Ukazatel na aktuální instanci

```
class Complex
{
    double real, imag;

public:

    void set_real(double real)
    {
        this->real = real;
    }
};
```

real: členská proměnná

real: parametr funkce

Děkuji Vám
za pozornost.

mikulka@vut.cz
www.utee.fekt.vut.cz