

# PB161 Programování v jazyce C++

## Přednáška 5

Jmenné prostory  
Vstupní/výstupní proudy

Nikola Beneš

19. října 2015

## Jmenné prostory

**Problém:** výskyt dvou entit se stejným jménem

```
// library1.h  
class Object { /* ... */ };
```

```
// library2.h  
class Object { /* ... */ };
```

```
// main.cpp  
#include "library1.h"  
#include "library2.h"
```

Při překladu main.cpp dojde k chybě:  
error: redefinition of 'class Object'

# Implicitní jmenné prostory

```
int x;  
// double x; // CHYBA!  
class Example {  
    float x;  
public:  
    void method() const {  
        double x;  
        for (int i = 0; i < 3; ++i) {  
            std::string x;  
        }  
        {  
            char x;  
        }  
    }  
};
```

# Implicitní jmenné prostory

globální jmenný prostor

```
int x;  
// double x; // CHYBA!  
class Example {  
    float x;  
public:  
    void method() const {  
        double x;  
        for (int i = 0; i < 3; ++i) {  
            std::string x;  
        }  
    }  
};
```

# Implicitní jmenné prostory

```
int x;
```

globální jmenný prostor

```
// double x; // CHYBA!
```

```
class Example {
```

jmenný prostor třídy Example

```
    float x;
```

```
public:
```

```
    void method() const {
```

```
        double x;
```

```
        for (int i = 0; i < 3; ++i) {
```

```
            std::string x;
```

```
        }
```

```
    {
```

```
        char x;
```

```
    }
```

```
    }
```

```
};
```

# Implicitní jmenné prostory

```
int x;
```

globální jmenný prostor

```
// double x; // CHYBA!
```

```
class Example {
```

jmenný prostor třídy Example

```
    float x;
```

```
public:
```

```
    void method() const { jmenný prostor metody method
```

```
        double x;
```

```
        for (int i = 0; i < 3; ++i) {
```

```
            std::string x;
```

```
        }
```

```
        {
```

```
            char x;
```

```
        }
```

```
    }
```

```
};
```

# Implicitní jmenné prostory

```
int x;
```

globální jmenný prostor

```
// double x; // CHYBA!
```

```
class Example {
```

jmenný prostor třídy Example

```
    float x;
```

```
public:
```

```
    void method() const {
```

jmenný prostor metody method

```
        double x;
```

```
        for (int i = 0; i < 3; ++i) {
```

```
            std::string x;
```

```
        }
```

jmenný prostor cyklu for

```
    {
```

```
        char x;
```

```
    }
```

jmenný prostor bloku

```
    }
```

```
};
```



# Jmenné prostory

## Přístup ke jmenným prostorům – operátor ::

```
int x = 17;
class Example {
    int x;
public:
    Example() : x(29) {}
    void print() const {
        int x = 3;
        {
            int x = 9;
            cout << x << endl; // 9
            cout << Example::x << endl; // 29
            cout << ::x << endl; // 17
        }
        cout << x << endl; // 3
    }
};
```

# Explicitní jmenné prostory

**Pojmenované jmenné prostory** – syntax: `namespace` `jmeno` { ... }

```
namespace MyLib {  
    void print();  
    namespace Experimental { // možno i vnořovat  
        void print();  
    }  
}  
  
namespace MyLib {  
    class Example {  
    public:  
        void print() const;  
    };  
}
```

[ukázka použití]

# Explicitní jmenné prostory – zpřístupnění

## Zpřístupnění jmenného prostoru

- plná kvalifikace – `std::string`
- direktiva `using namespace` `jmeno_prostoru`;

```
#include <string>
```

```
string s; // CHYBA!
```

```
void print() {  
    using namespace std;  
    string s; // OK  
}
```

```
int main() {  
    string s; // CHYBA!  
}
```

## Explicitní jmenné prostory – zpřístupnění (pokr.)

- deklarace `using` `jmeno_prostoru::jmeno_entity`
  - má prioritu před `using namespace`

```
#include "libAdam.h"
#include "libEve.h"
using namespace Adam; // obsahuje funkci getApple();
using namespace Eve;  // taky obsahuje getApple();
```

```
getApple(); // CHYBA!
```

```
using Eve::getApple;
```

```
getApple(); // OK, volá se Eve::getApple();
```

- alias jmenného prostoru

```
namespace SysWinWidget = System::Window::Widget;
```

## Používání `using` a `using namespace`

- v globálním prostoru
  - užívejte rozumně
  - **nikdy v hlavičkových souborech**
  - vždy až po všech `#include`
- lokálně
  - ve funkcích/metodách
  - ve vnořených blocích
  - není možno používat přímo uvnitř třídy (class scope)

# Explicitní jmenné prostory – použití

**Použití jmenných prostorů** ve vlastních knihovnách:

```
// cool_library.h
#ifndef COOL_LIBRARY_H
#define COOL_LIBRARY_H

namespace cool_library {

class Cool { /* ... */ };

}

#endif
```

# Explicitní jmenné prostory – použití (pokr.)

**Oddělení kolidujících jmen** při použití cizích knihoven:

```
namespace Lib1 {  
#include "library1.h" // obsahuje třídu System  
}  
  
namespace Lib2 {  
#include "library2.h" // obsahuje třídu System  
}  
  
int main() {  
    System s; // CHYBA!  
    Lib1::System s1; // OK  
    Lib2::System s2; // OK  
}
```

# Jmenné prostory – další informace

## Další čtení pro zvídání

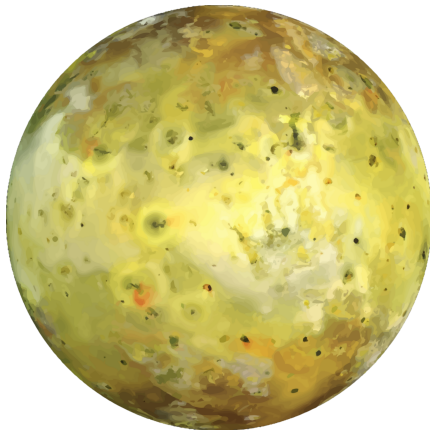
- <http://en.cppreference.com/w/cpp/language/namespace>
- <http://en.cppreference.com/w/cpp/language/lookup>
  - qualified name lookup
  - unqualified name lookup
  - argument-dependent lookup

```
namespace Test {  
    int x;  
    void print(int y) {}  
}
```

```
int main() {  
    print(x); // CHYBA!  
    Test::print(x); // CHYBA!  
    Test::print(Test::x); // OK  
    print(Test::x); // taky OK, argument-dependent lookup  
}
```



## Vstupní/výstupní proudy



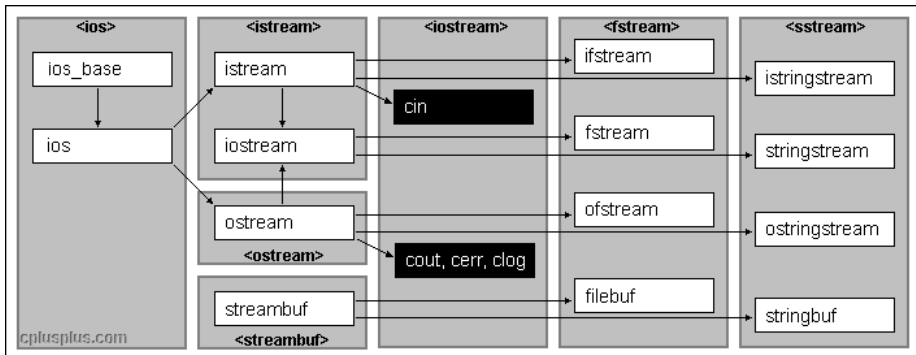
## **Vstup a výstup** z různých druhů zařízení

- soubory
- klávesnice, obrazovka (terminál)
- tiskárna, skener
- síťový socket

## **Abstrakce konkrétního zařízení**

- proudy (streams)
- data „plynou“ proudem od zdroje k cíli
- využívá principů OOP

# Hierarchie I/O proudů



- hierarchie umožňuje proudy s různými vlastnostmi
- proudy nelze kopírovat; proč?

## Standardní instance

- `cin` – standardní vstup, instance `istream` (`stdin` v C)
- `cout` – standardní výstup, instance `ostream` (`stdout` v C)
- `cerr` – standardní chybový výstup, instance `ostream` (`stderr` v C)
- `clog` – standardní logovací výstup, instance `ostream` (`stderr` v C)
- rozdíl mezi `cerr` a `clog`: `cerr` nepoužívá vyrovnávací paměť (buffer)

## Speciální entity pro manipulaci proudů (manipulátory)

- např. `endl` – vloží konec řádku a zároveň provede vyprázdnění bufferů

# Vstup/výstup – operátory

## Operátor výstupu <<

```
int x = 7;  
const double pi = 3.14;  
char cString [] = " is ";  
cout << x << " + " << pi << cString << x + pi << endl;  
// 7 + 3.14 is 10.14
```

- pozor na prioritu

# Vstup/výstup – operátory

## Operátor výstupu <<

```
int x = 7;
const double pi = 3.14;
char cString [] = " is ";
cout << x << " + " << pi << cString << x + pi << endl;
// 7 + 3.14 is 10.14
```

- pozor na prioritu
- přetížení << pro vlastní třídy

```
ostream & operator<<(ostream & out, const Object & object);
```

[ukázka: řetězení výstupů]

- přístup k **private** a **protected** atributům našeho objektu

[ukázka: příklad]

# Vstup/výstup – operátory (pokr.)

## Operátor vstupu >>

```
int x;  
double d;  
char cString [50];
```

```
std::cin >> x >> d;  
std::cin >> cString; // možná problém
```

```
std::string s;  
std::cin >> s; // přečte jedno slovo ze vstupu
```

# Vstup/výstup – operátory (pokr.)

## Operátor vstupu >>

```
int x;  
double d;  
char cString [50];
```

```
std::cin >> x >> d;  
std::cin >> cString; // možná problém
```

```
std::string s;  
std::cin >> s; // přečte jedno slovo ze vstupu
```

- přetížení >> pro vlastní třídy

```
istream & operator>>(istream & out, Object & object);
```

- přístup k **private** a **protected** atributům našeho objektu

[ukázka: příklad]



## Chybové stavy proudů

- proudy obsahují příznaky naznačující chybu
  - eofbit, failbit, badbit
- metody pro testování stavu
  - good(), fail(), eof()
  - přetížené chování proudů jako **bool**, přetížený operátor !

```
int x;
cin >> x;
if (cin) { /* načtení hodnoty do x se povedlo */ }
if (cin.good()) {
    /* načtení hodnoty se povedlo a není konec souboru */
}
```

- metoda pro vyčištění příznaků: clear()
- [http://en.cppreference.com/w/cpp/io/ios\\_base/iostate](http://en.cppreference.com/w/cpp/io/ios_base/iostate)

## Proudy pro vstup/výstup ze souborů

- třídy ifstream, ofstream, fstream
- konstruktor se jménem souboru, metody open() a close()

```
#include <fstream>
```

```
using namespace std;
```

```
int main() {  
    ofstream output("soubor.txt");  
    output << "Hello, world!\n";  
    output.close();  
    output.open("soubor2.txt");  
    output << 3.14 << endl;  
    // o zavření se postará destruktork  
}
```

### Mód otevření souboru (nepovinný parametr konstruktoru)

- binární příznaky (flags), kombinujeme pomocí |
- typ otevření
  - `ios::in` (vstup, implicitní pro `ifstream`)
  - `ios::out` (výstup, implicitní pro `ofstream`)
  - `ios::in | ios::out` (obojí, implicitní pro `fstream`)
- způsob otevření
  - `ios::binary` (binární data, implicitní jsou textová data)
  - `ios::app` (append, výstup na konec souboru)
  - `ios::trunc` (vymaže soubor)
- více viz <http://cppreference.com> nebo <http://cplusplus.com/reference>
  - hledejte `ios` a `openmode`

[ukázka: příklad práce se soubory]

# Třída `istream`

- operátor `>>` – už známe
- metoda `get()`
  - bez parametrů vrátí jeden znak
  - více znaků pomocí `get(buffer, length)` (čte do konce řádku)
- metoda `getline(buffer, length)`
  - podobně jako `get()`, ale zahodí znak konce řádku (`'\n'`)
- metoda `read(buffer, count)`
  - blokové čtení daného počtu bytů
  - hlavně pro binární soubory
- metoda `gcount()`
  - počet znaků načtených při posledním vstupu
- metoda `peek()`
  - náhled na další znak na vstup, bez jeho přečtení
- metoda `ignore(count)`
  - zahodí *count* znaků ze vstupu
- a další ...
  - viz reference na webu

# Třída istream (pokr.)

- načtení řádku do řetězce typu `std::string`
  - funkce `getline(istream &, string &)`

```
std::string s;  
getline(std::cin,s);
```

- součást knihovny `string`, ne knihovny `iostream`

- operátor << – už známe
- metoda `put()`
  - zapíše jeden znak
- metoda `write(output, count)`
  - protiklad `read()`
- zápis na konci souboru soubor zvětšuje
- zápis uvnitř souboru soubor přepisuje

# Pozice a posun v souboru

- odkud se čte, kam se zapisuje?
  - „get“ ukazatel (třída `istream` a její potomci)
  - „put“ ukazatel (třída `ostream` a její potomci)
- `tellg()`, `tellp()` – získání pozice ukazatelů
- `seekg()`, `seekp()` – nastavení pozice ukazatelů; dva parametry
  - odkud:
    - `ios::beg` začátek souboru
    - `ios::cur` aktuální pozice
    - `ios::end` (konec souboru)
  - `offset`: o kolik se posunout od pozice *odkud*
- počáteční pozice v souboru – závisí na módu otevření

# Vyrovnávací buffery

- data poslaná do proudu nemusí být ihned zapsána do cíle
- vyrovnávací paměť typu `streambuf` pro každý proud
- přenos z vyrovnávací paměti do cíle
  - při uzavření souboru (`close()`, destruktory)
  - při zaplnění bufferu
  - explicitně pomocí manipulátorů (`endl`, `flush`, `sync`, `unitbuf`)



- speciální objekty, které je možno předávat operátorům << a >>
- flush – vyprázdní buffer
- endl – zapíše konec řádku a vyprázdní buffer
- dec, hex, oct – změni způsob reprezentace čísel
- setw, setfill, setprecision – měni formát vstupu a výstupu
- left, right – měni zarovnání

```
const double pi = 3.1415;  
cout << setw(10) << setfill('^') << left << setprecision(3)  
    << pi << endl;  
// výstup: 3.14^^^^^
```

- <http://en.cppreference.com/w/cpp/io/manip>

## Proudy dat v paměti

- třídy stringstream, istream, ostream
- konstruktor může brát řetězec – počáteční stav proudu
- metoda str() nastaví obsah proudu
  - bez parametrů vrátí aktuální obsah proudu jako string

[ukázka: příklad použití]

## Jmenné prostory

- implicitní (lokální bloky, funkce, třídy, globální prostor)
- explicitní (**namespace**), mohou být vnořené
- zpřístupnění pomocí **using namespace** nebo **using**

## Vstupní/výstupní proudy

- abstrakce skutečných zařízení
- hierarchie
- standardní vstup/výstup
- souborový vstup/výstup
- vstup/výstup v paměti
- přetížené operátory << a >>
- další metody