PB161 Programování v jazyce C++
Přednáška 1

Organizace
Úvod do C++

Nikola Beneš

20. září 2016

Cíle předmětu

1. ukázat možnosti jazyka C++
   - moderní C++ podle standardu C++14
2. vysvětlit základy OOP
   - jak se realizují v C++
3. podpořit praktické programátorské schopnosti
   - intenzivním programováním

Organizace předmětu

http://www.fi.muni.cz/pb161

Organizace předmětu

Přednášky

- nepovinné, ale doporučované
- v 7. týdnu vnitrosemestrální test (20 bodů)
- zvaná přednáška ke konci semestru

Cvičení

- nepovinná, možnost kdykoli přijít a kdykoli odejít
- týdenní miniúkoly (celkem 10 po 3 bodech)
- programování s možností okamžité pomoci
- poslední týden zápočtový příklad

Úkoly

Týdenní miniúkoly (10 po 3 bodech)

- zadání před přednáškou, odevzdání další týden v pátek
- smysl: procvičit látku probranou na přednášce
- odevzdávání pomocí fakultního GitLabu, 3 pokusy (jen naostro)
- část testů k dispozici předem

Domácí úkoly (4 + 1 navíc)

- brzký deadline, deadline pro odevzdání
- 15 bodů za úkol
  - 2 body za odevzdání nanečisto před brzkým deadline
  - 8 bodů za projití testy v kontru
  - 2 body za pěkně napsaný kód
  - 3 body za počet odevzdání
- odevzdávání použitím fakultního GitLabu, max 3 ostré pokusy, odevzdávání nanečisto

detaily na `http://www.fi.muni.cz/pb161`

Varování

Neopisujte!

- podvádění je nemorální
- ubližujete sami sobě (nic se nenaučíte)
- provádíme automatickou kontrolu opisování
- opisování = 0b za úkol (pro oba)

Nezveřejňujte svá řešení!

- stejný postih jako za opisování
- zákaz zveřejňovat řešení i po termínu domácího úkolu
- dobře si zkontrolujte práva repozitářů

Bodování

- tvrdé a měkké body
  - měkké se nepočítají do limitu pro zápočet a zkoušku

domácí úkoly	$5 \times 15 = 75$
miniúkoly	$10 \times 3 = 30$
vnitrosemestrální test	20
celkem (tvrdé body)	125

- zkouškový test: 80 bodů
- měkké body:
  - bonusové části domácích úkolů
  - jiné (upozornění na chyby, aktivita)

Hodnocení

Zápočet

- >= 65 tvrdých bodů, 4 nenulové domácí úkoly, zápočtový příklad
  - stačí 3 nenulové domácí úkoly, pokud máte alespoň 70 % bodů z miniúkolů

Zkouška

- >= 95 tvrdých bodů, zápočet
- známka podle součtu tvrdých a měkkých bodů:

>= 175 bodů	A
>= 155 bodů	B
>= 135 bodů	C
>= 115 bodů	D
>= 95 bodů	E

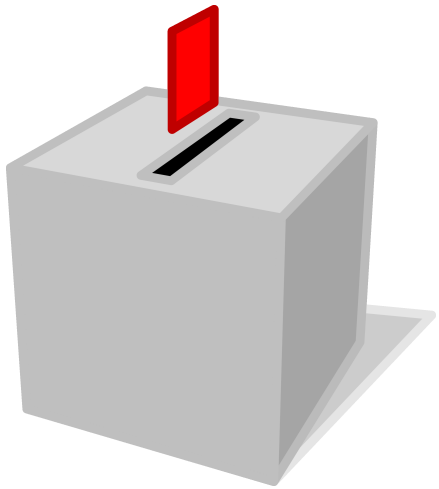
Učební materiály

`http://cppreference.com`

ukázka: online dokumentace

Zpětná vazba

- předmětová anketa v ISu (až na konci semestru)
- občasné dotazníky (obtížnost úloh apod.)
- osobně, e-mailem
- krabice pro anonymní vzkazy



Kahoot!

https://kahoot.it

Doporučované vývojové nástroje

CLion

- komerční, studentská licence zdarma
- k dispozici v unixových učebnách
- k dispozici v B116/B117

cmake

- v současnosti asi nejpopulárnější nástroj pro automatizaci překladu programů
- multiplatformní

clang

- populární multiplatformní překladač
- součást projektu LLVM

googletest

- oblíbená knihovna pro psaní *unit testů*

Plán semestru

Blok 1: Základní programování v C++

- základy vstupu a výstupu, řetězce, dynamická pole
- základy tříd, reference, `const`
- standardní knihovna, kontejnery, algoritmy

Blok 2: Správa paměti a zdrojů, I/O

- konstruktory a destruktory, kopírování
- práce s pamětí a zdroji, princip RAII
- přetěžování operátorů, vstup a výstup podrobněji

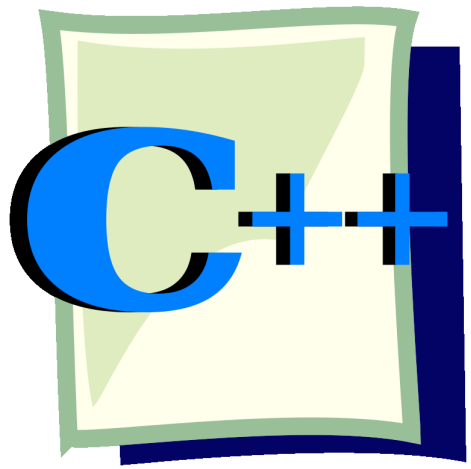
Blok 3: OOP

- principy OOP, virtuální metody
- návrhové vzory, jmenné prostory, `static`
- výjimky

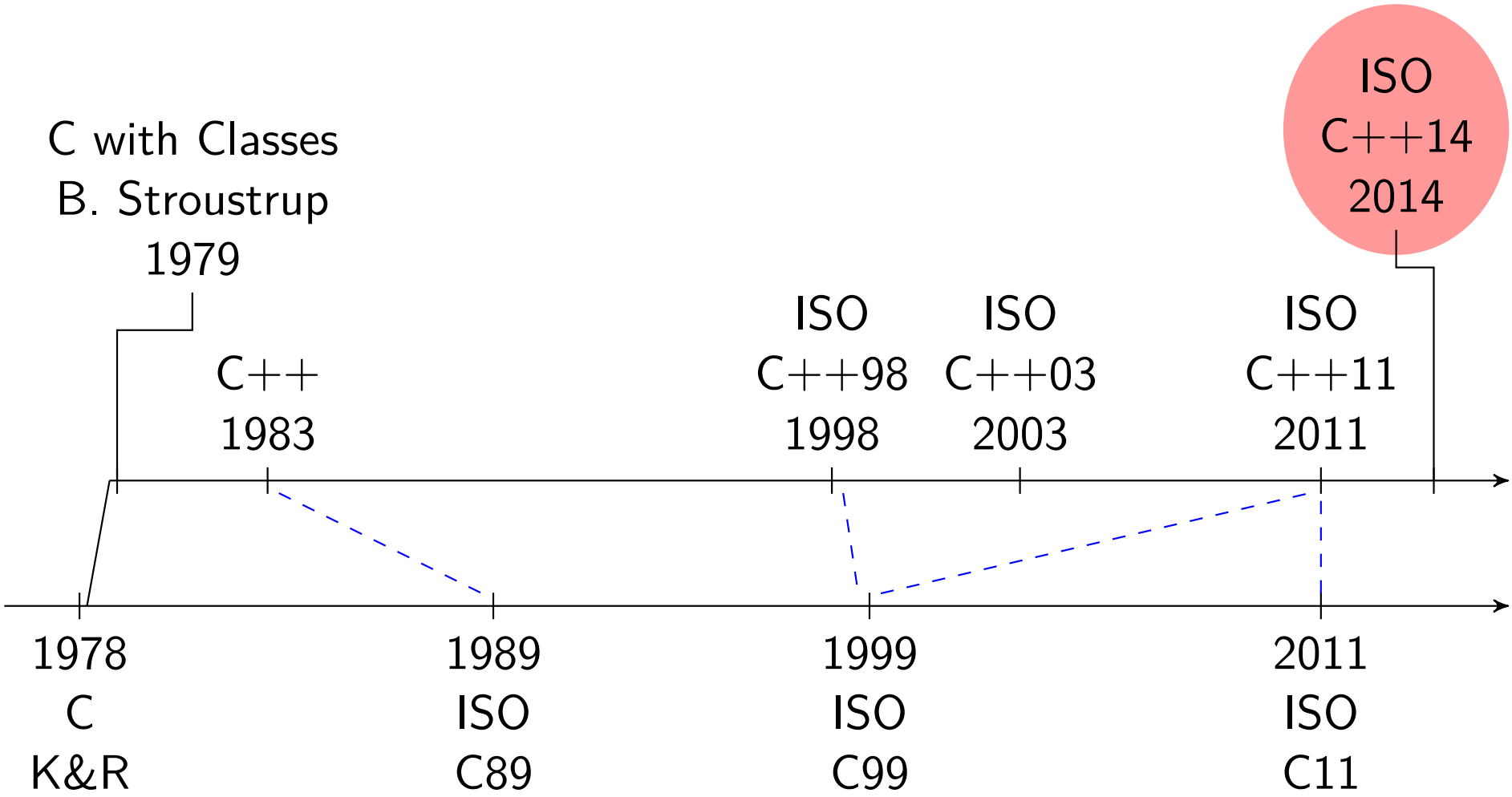
Blok 4: Šablony a vůbec

- šablony, úvod do generického programování
- ukázka pokročilého C++, `move` sémantika

Programovací jazyk C++



Historie a vývoj



Budoucnost: C++17, ...

Nestandardizovaná rozšíření

Charakteristika C++

- imperativní, staticky typovaný jazyk
- objektově-orientovaný; *hodnotová sémantika* objektů
- s funkcionálními prvky (zejména od C++11)
- podporuje generické programování a metaprogramování (šablony)
- částečně zpětně kompatibilní s C
- rozsáhlá standardní knihovna
- C++11 přineslo významné změny k lepšímu

Proč používat C++?

- široké rozšíření
- vysoká rychlost kódu
- univerzálnost
- RAII (deterministická správa zdrojů)
- vhodné pro
  - větší projekty
  - systémové aplikace
  - rychlou grafiku
  - embedded zařízení
- spíše nevhodné pro
  - webové aplikace
  - rychlé prototypy

`http://benchmarksgame.alioth.debian.org/`

Hello, world!

```cpp
// C++
#include <iostream>

int main() {
    std::cout << "Hello, world!\n";
}
```

```c
// C
#include <stdio.h>

int main() {
    printf("Hello, world!\n");
}
```

Hello, world! 2.0

```cpp
// C++
#include <iostream>
#include <string>

int main() {
    std::cout << "What is your name? ";
    std::string name;
    std::cin >> name;
    std::cout << "Hello, " << name << "!\n";
}
```

- Jak bychom podobný program napsali v C?

Řetězce v C++

`std::string`

- hlavičkový soubor `<string>`
- sám si alokuje a dealokuje paměť
- indexování funguje jako v C

```
std::string s = "Hello!";
s[0] = 'J';
std::cout << s; // Jello!
```

- navíc se umí přiřazovat

```
std::string s = "Hello!";
std::string t;
t = s;
```

- co se stane, když teď provedeme `t[0] = 'J'`?

Řetězce v C++ (pokr.)

`std::string`

- umí se řetězit pomocí operátoru + (a operátoru +=)

```
std::string h = "Hello";
std::string w = "world";
std::string s = h + ", " + "!";
```

- **Pozor!** tohle nebude fungovat: (proč?)

```
std::string s = "James" + " " + "Bond";
```

- umí toho ještě mnohem více:
  `http://en.cppreference.com/w/cpp/string`

Dynamická pole v C++

std::vector

- hlavičkový soubor <vector>
- dynamické (rozšiřitelné) pole

```
std::vector<int> v;
v.push_back(1); // vložení prvku za konec pole
v.push_back(2);
v.push_back(3);

for (int i = 0; i < v.size(); ++i) {
    std::cout << v[i];
}

for (int x : v) { // od C++11: range-for
    std::cout << x;
}
```

Dynamická pole v C++ (pokr.)

`std::vector`

- inicializace

```
std::vector<int> v = {1, 2, 7, 17, 42}; // od C++11
std::vector<int> v{1, 2, 7, 17, 42}; // totéž
std::vector<int> v(10); // 10 nul
std::vector<int> v(10, 17); // 10 sedmnáctek
```

- pozor na rozdíl mezi `()` a `{}`
- více na `http://en.cppreference.com/w/cpp/container/vector`
- pozor na rozdíl mezi `resize` a `reserve`

Hodnotová sémantika

Objekty se chovají stejně jako primitivní typy

- přiřazení je vytvoření kopie
- předání do funkce je vytvoření kopie
- konec bloku znamená zánik objektu

```
void f(int z) {
    z = 7;
}


int x = 3;
int y = x;
f(y);
x = 5;
// jaké jsou hodnoty x, y?
```

- `std::string` i `std::vector` se chovají stejně [ukázka]

Volání funkcí

```
void f1(std::string s) {
    // s je kopie skutečného argumentu
    // změny s se navenek nijak neprojeví
}

void f2(std::string& s) {
    // s je reference na skutečný argument
    // změny s se navenek projeví
}

void f3(const std::string& s) {
    // s je reference na skutečný argument
    // je zakázáno s měnit
}
```

- více si o referencích a hodnotové sémantice řekneme příště

Range-for

- funguje podobně jako volání funkcí na předchozím slajdu

```
std::vector<std::string> names;
// ...
for (std::string s : names) {
    // s je kopie položky vectoru
}

for (std::string& s : names) {
    // s se odkazuje na položku vectoru
    // můžeme jej měnit
}

for (const std::string& s : names) {
    // s se odkazuje na položku vectoru
    // nesmíme jej měnit
}
```

Základy vstupu a výstupu <iostream>

`std::cout`

- standardní výstup, objekt typu `std::ostream`
- výpis probíhá pomocí operátoru <<
  - přetížený operátor, automatická detekce typu

```
std::cout << "Jmenuji se " << name
          << " a je mi " << age << " let.\n";
```

- manipulátor `std::endl` – konec řádku a vyprázdnění bufferu

```
std::cout << "In a galaxy far, far away ..."
          << std::endl;
```

Základy vstupu a výstupu (pokr.)

`std::cin`

- standardní vstup, objekt typu `std::istream`
- vstup probíhá pomocí operátoru >>
  - vstup je ukončen bílým znakem

```
std::string s;
std::cin >> s; // načte jen jedno slovo
```

- vstup celého řádku pomocí `std::getline(std::cin, s)`

- více o vstupu a výstupu později

Co znamená `std::`?

Jmenné prostory

- umožňují lepší koexistenci různých knihoven
- více si o nich řekneme později

Jmenný prostor standardní knihovny `std`

- všechny typy, objekty, funkce standardní knihovny začínají `std::`

Musíme všude psát `std::`?

Direktiva `using namespace` a deklarace `using`

```
using std::cout;     // odteď můžeme psát jen cout
// ...
using namespace std;
// odteď můžeme psát všechno bez std::
```

- platnost končí koncem bloku
  - pokud není uvnitř bloku, nekončí nikdy

Doporučení a pravidla

- preferujte `using std::něco;` před `using namespace std;` (proč?)
- **nikdy** nepište `using namespace std;` nebo `using std::něco` do hlavičkového souboru, pokud to není uvnitř bloku (funkce)
  - Proč? Co strašného se stane?

Závěrečný kvíz

https://kahoot.it

Závěrečný kvíz

```
#include <iostream>
#include <string>
#include <vector>
void f(std::vector<std::string> z) {
    z[0] = "Omikron";
}
int main() {
    std::vector<std::string> x = {"Alpha", "Beta", "Gamma"};
    std::vector<std::string> y = x;
    x[1] = "Delta";
    f(y);
    for (std::string s : y) {
        std::cout << s;
    }
    std::cout << '\n';
}
```