



Programování v C++ cvičení (2020/21)

faltin@ksi.mff.cuni.cz

<https://fan1x.github.io/cpp20.html>



Programování v C++ cvičení 7 (18.11.2020)

faltin@ksi.mff.cuni.cz

<https://fan1x.github.io/cpp20.html>



Připomenutí

- Vymyslet & domluvit zápočtový program do konce listopadu
- 



1. Velký domácí úkol – Agregátor dat

- V Recodexu (viz web)
- Do 20.12. (neděle) 23:59
 - -5b za každý další započatý týden
- Body: 10+5b
 - 10b za funkcionalitu
 - 5b za kulturu kódu (čitelnost, konvence, ...)

Dědičnost

- Chcete podědit(=použít) vlastnosti (funkce, proměnné) svého předka

```
class container {
public:
    using size_type = size_t;

    size_t size() const { return size_; }

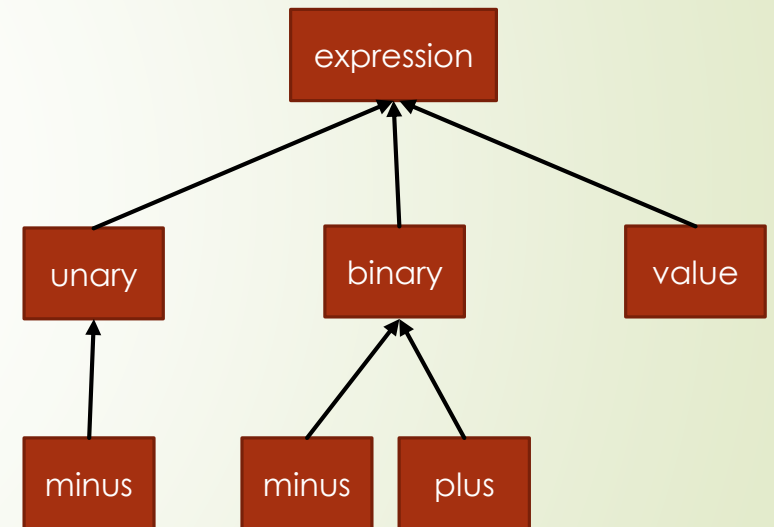
protected:
    size_t size_;
};

class vector_int : public container { ... };
class list_int : public container { ... };

int main() {
    vector_int vi;
    cout << vi.size();
}
```

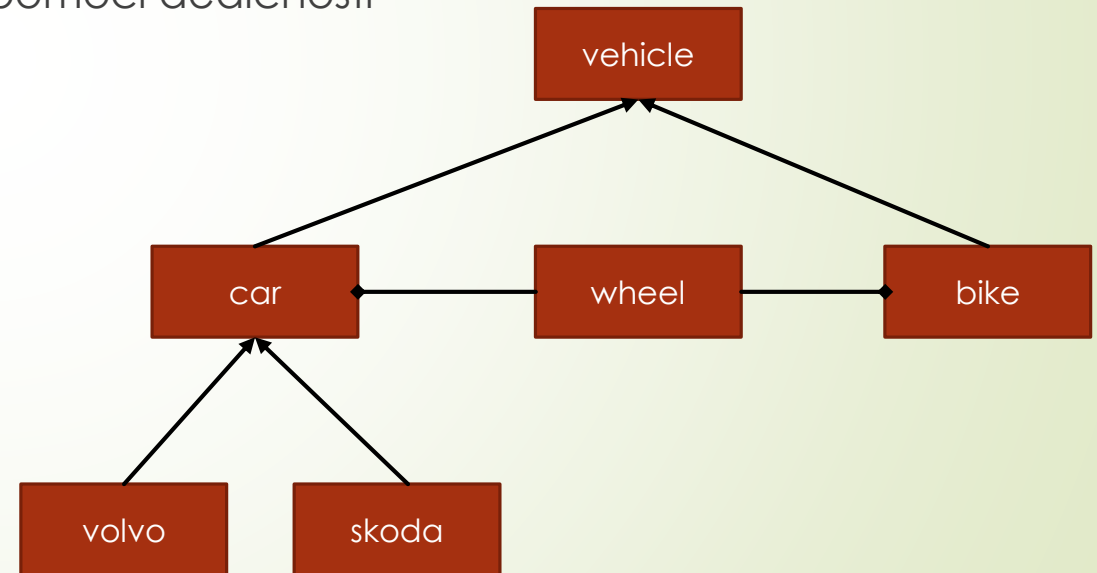
Příklady dědičnosti

```
class car {};  
class volvo : public car {};  
class skoda : public car {};  
  
class animal {};  
class dog : public animal {};  
class cat : public animal {};  
  
class employee {};  
class accountant : public employee {};  
  
class expression {};  
class binary : public expression {};  
class plus : public binary {};  
  
class object {} // JAVA  
class XYZ : public object {};
```



Dědičnost vs. skládání

- Dědičnost - existuje logická podřazenost (OOP)
- Skládání - jeden obsahuje druhý bez žádné logické návaznosti
 - Může být implementováno pomocí dědičnosti



Dynamický polymorfismus

- Funkce musí být `virtual` a voláme ji přes referenci/ukazatel!

```
class base {
protected:
    int value = 0;
public:
    virtual ~base() = default;
    virtual void print() const { std::cout << "base: " << value; };
};

class derived : public base {
public:
    void print() const override { std::cout << "derived: " << value; };
};


int main() {
    derived d;
    d.print(); // derived: 0
    base &b = d;
    b.print(); // derived: 0
    base *b_ptr = &d;
    b_ptr->print(); // derived: 0
    base b2 = d;
    b2.print(); // base: 0
}
```


Úkol: pole libovolných typů

- Podpora pro: int, double, string
- Použít dědičnost (NE union/variant/...)

```
// API:
push_back(value); // insert a value
print(size_type i); // print the value of i-th element
print_all(); // print all values inside the array

int main() {
    my_vec v;
    v.push_back(1); v.push_back(2.3); v.push_back("four");
    v.print(0); // 1
    v.print(2); // "four"
    v.print_all(); // [1, 2.3, "four"]
}
```



Programování v C++ cvičení 6 (11.11.2020)

faltin@ksi.mff.cuni.cz


<https://fan1x.github.io/cpp20.html>

C++ vs. ++C (1/2)

- C++ dovoluje overloading operatorů (*, ++, --, +, -, &&, *, ...)
 - Neměňte semantiku

```
// ex 1
for (size_t i = 0; i < N; ++i) { ... }
// vs.
for (size_t i = 0; i < N; i++) { ... }
```

```
// ex 2
for (auto it = my_class.begin(); it != my_class.end(); ++it) { ... }
// vs.
for (auto it = my_class.begin(); it != my_class.end(); it++) { ... }
```



C++ vs. ++C (2/2)

```
class C {  
    size_t idx;  
public:  
    C &operator++() { // ++i  
        ++idx;  
        return *this;  
    }  
  
    C operator++(int) { // i++  
        C copy(*this);  
        ++idx;  
        return copy;  
    }  
};
```



Připomenutí

- Téma zápočtového programu **do 30.11.**
- 1. velký DÚ **příště**
- Malé domácí úkoly:
 - **Warnings!**
 - Úkol uznán, pokud dostanete approval (od dú5 – vector)



Najdi chyby

▀ `ex06-matrix.cpp` (web, slack)



Úkol (volitelně): databáze lidí

- Data jsou ukládána ve formě **nevyváženého binárního stromu**
- Na vytváření jednotlivých uzlů se používá **dynamická alokace** (ne kontainery)
- Přemýšlejte nad návrhem API

```
// API:  
class person { name, age, address }, id=name  
insert(person) // vlož osobu pokud neexistuje  
find(person) // najdi osobu a vrať na ni nějaký odkaz nebo dej vědět, že  
neexistuje  
erase(person) // odstran osobu
```



Programování v C++ cvičení 5 (4.11.2020)

faltin@ksi.mff.cuni.cz

<https://fan1x.github.io/cpp20.html>

Dynamická alokace v moderním C++ (1/3)

```
struct Complex {
    int r, i;
    Complex(int r, int i) : r{r}, i{i} {}
};
void print(const Complex *c) {
    std::cout << "[" << c->r << "," << c->i << "]";
}
void clear(Complex *c) {
    c->r = 0;
    c->i = 0;
}

int main() {
    std::unique_ptr<Complex> ptr_c = std::make_unique<Complex>(1, 2);
    print(ptr_c.get());
    clear(ptr_c.get());
}
```

Dynamická alokace v moderním C++ (2/3)

```
struct Complex {
    int r, i;
    Complex(int r, int i) : r{r}, i{i} {}
};
void print(const Complex *c) {
    std::cout << "[" << c->r << "," << c->i << "]";
}
void clear(Complex *c) {
    c->r = 0;
    c->i = 0;
}

int main() {
    std::unique_ptr<Complex[]> ptr_c = std::make_unique<Complex[]>(10);
    print(&ptr_c[0]);
    clear(&ptr_c[0]);
}
```

Dynamická alokace v moderním C++ (3/3)

```
struct Complex {
    int r, i;
    Complex(int r, int i) : r{r}, i{i} {}
};
void print(const Complex *c) {
    std::cout << "[" << c->r << "," << c->i << "]";
}
void clear(Complex *c) {
    c->r = 0;
    c->i = 0;
}

int main() {
    std::shared_ptr<Complex[]> ptr_c = std::make_shared<Complex[]>(10);
    print(&ptr_c[0]);
    clear(&ptr_c[0]);
}
```

Dynamická alokace v moderním C++ (3/3)

- Moderní
 - `std::unique_ptr<T>, std::unique_ptr<T[]>`
 - `std::shared_ptr<T>, std::shared_ptr<T[]>`
 - `std::weak_ptr<T>`
- Low-level
 - `new T, new T[], delete, delete[]`



Úkoly

1. Vlastní implementace vektoru pro čísla

- `ctor(size_t, value_type)`, `size()`, `capacity()`, `reserve()`, `resize()`, `push_back()`, `operator[]()`




Programování v C++ cvičení 4 (21.10.2020)

faltin@ksi.mff.cuni.cz

<https://fan1x.github.io/cpp20.html>



Trigraphs (do C++17)



```
int main(int argc, char* argv[])
{
    // WTF, why is it not working ?????/
    std::cout << "Hello World";
}
```

Úkoly (zkušenosti)

- Zapnout varování překladače (warnings)
 - GCC: -Wall -Wextra, VS: /Wall
- Nezáporné typy: *unsigned int*, *size_t*, ...
 - **uintX_t**: *uint8_t*, *int8_t*, ..., *uint64_t*, *int64_t*
 - velikost v bytech **sizeof(X)** – *sizeof(size_t)*
- **const**-funkce: nemění data, pouze čte obsah třídy
- **static** funkce: funkce, která nepotřebuje přistupovat k vnitřním datům třídy
- `std::vector`
 - `resize(N, default_value)`,
 - `vector<vector<int>> vi(3, vector<int>(10, default_value))`

OOP

```
using row_t = vector<int>;
using matrix_t = vector<row_t>;

void print(const matrix_t &m) {
    for (auto &&row : m) {
        for (auto &&element : row) {
            std::cout << element;
        }
    }
}

const row_t &get_row(const matrix_t &m,
                    size_t row_id) {
    return m[row_id];
}

matrix_t data;
```

```
class matrix {
public:
    using row_t = vector<int>;
    using data_t = vector<row_t>;

    void print() const {
        for (auto &&row : data) {
            for (auto &&element : row) {
                std::cout << element;
            }
        }
    }

    const row_t &get_row(size_t row_id) const {
        return data[row_id];
    }
private:
    data_t data;
};
```

Deklarace/definice

```
// file: my_class.hpp
#ifndef MY_CLASS_HPP
#define MY_CLASS_HPP

void fn(int x);

class my_class {
public:
    my_class();
    int exec(int x);

private:
    double d;
    static size_t i;
};

#endif // MY_CLASS_HPP
```

```
// file: my_class.cpp
#include "my_class.hpp"
#include <iostream>

void fn(int x) {
    cout << "fn()";
}

my_class::my_class() : d(1.0) {
    cout << "ctor";
}

int my_class::exec(int x) {
    for(int i=0; i < x; ++i) { ... }
}


size_t my_class::i = 0;
```



Úkoly



- Piškvorky pro 2 hráče (5 v řadě)
 - Hráči se střídají v zadávání souřadnic
 - Kontroluje se správnost souřadnic
 - Kontroluje se, jestli některý hráč nevyhrál
 - `matice<vector<???`>>



Programování v C++ cvičení 3 (14.10.2020)

faltin@ksi.mff.cuni.cz

<https://fan1x.github.io/cpp20.html>



Operátor „down to“

```
// vypíše 9, 8, 7, ..., 0
void op_downto() {
    int x = 10;
    while (x --> 0) {
        cout << x;
    }
}
```

class/struct

- Speciální metody: constructor, copy-constructor, move-constructor, destructor, copy-operator, move-operator

```
class C {
    int x = 0;
public:
    C() { cout << "ctor\n"; }
    C(const C &c) : x(c.x) { cout << "copy-ctor\n"; }
    C(C &&c) : x(c.x) { cout << "move-ctor\n"; }
    ~C() { cout << "dtor\n"; }
    C &operator=(const C &c) {
        x = c.x;
        cout << "copy-op\n";
        return *this;
    }
    C &operator=(C &&c) {
        x = c.x;
        cout << "move-op\n";
    }
};
```

Úkol 1: implementace třídy C

- Implementovat třídu C, aby program vypsal čísla 1, 2, 3, ..., 20
 - NE **exit()** apod...

```
class C { ... }; // implement

// nesahat na věci níže!
void fn_copy(C) {}
void fn_cref(const C&) {}

int main(int argc, char* argv[])
{
    cout << "1";
    C c;
    cout << "5";
    fn_copy(c);
    cout << "10";
    fn_cref(c);
    fn_copy(std::move(c));
    cout << "15";
}
```



Úkoly



1. Implementovat třídu C
2. Třída pro komplexní čísla
 - naimplementovat všechny speciální metody



Programování v C++ cvičení 2 (7.10.2020)

faltin@ksi.mff.cuni.cz

<https://fan1x.github.io/cpp20.html>

Úkoly (zkušenosti)

- Nepoužívat copy&paste
- Rozdělovat do funkcí
- Místo komentářů používat funkce
- Využívat funkce z STL (např.: `std::stoi`, `std::list<T>`)
 - pozor na složitost funkcí (`std::vector::delete()`)
- Minimalizovat *continue*, *break*, *goto*



Úkoly (zkušenosti)

- Rozumné pojmenovávání
- Předávání parametrů





Úkoly (zkušenosti)

- Do GITu pouze zdrojáky, konf. soubory, ...
 - NE: *.obj*, *.log*, *.pdb*, ...
- Další úkoly odevzdat + vytvořit merge request (přidat mě)
 - Návod na stránkách

Předávání parametrů

- hodnotou (by value): **void fn(string str)**
 - Vytvoří se **kopie**, která se předá do funkce
- odkazem (by reference)
 - reference: **void fn(string &str)**
 - Funkce modifikuje parametr uvnitř
 - Výstupní parametry (pokud nelze návratovou hodnotou)
 - const-reference: **void fn(const string &str)**
 - Předává se parametr, ale nechci vytvářet kopii (pro velké třídy, kde je kopírování drahé)
 - r-value reference: **void fn(string &&str)**
 - Později
- *ukazatelem (by pointer)
 - **Není to způsob předávání parametrů** (ukazatel je předáván hodnotou)
 - V C-čku - nemá reference

class/struct (1/2)

- Chceme strukturovat data, funkce „k sobě“

```
class calculator {  
    void sum(); // private by default  
public:  
    void calc(const std::string &expression); // change internals  
    void print() const; // doesn't change internals  
private:  
    // ...  
};  
  
calculator c; // no need for `new`  
c.calc("1+2*3/4");
```

- **const** funkce může volat pouze **const** funkci

class/struct (2/2)

```
struct coordinate {  
    int x;  
    int y;  
    int z;  
    void set(int x, int y, int z);  
};
```

- class vs. struct
- **Use class if the class has an invariant; use struct if the data members can vary independently**

<http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#c-classes-and-class-hierarchies>

<http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#c2-use-class-if-the-class-has-an-invariant-use-struct-if-the-data-members-can-vary-independently>

std::vector<T>

```
#include <vector>
int main() {
    std::vector<int> vi{1, 2, 3, 4, 5, 6};
    std::vector<float> vf(5, 0.0f);
    std::cout << vi[3] << " " << vf.at(3) << std::endl;
    std::cout << vi.size();
    vi[3] = 100; vi.at(6) = 600;
    vf.push_back(100.0f); vf.emplace_back(200.0f);
    vf.insert(3, 300.0f); vf.emplace(3, 300.0f);
    vi.pop_back();
    vf.erase(3);
    vi.clear();
    vi.resize(10); vi.reserve(100);
}
```

- Pozor na časovou složitost operací
- vector<bool>

Úkoly

1. matice pro čísla

- `set(x, y, value), get(x, y), print()`
 - `set_width(), set_height(), get_width(), get_height()`
 - `get_row(x), get_column(x)` – vrať řádek/sloupec x
 - `get_rows(), get_columns()` – vrátí pole všech řádků/sloupců
 - `clear()` – nastav všechny hodnoty na 0
 - `fill_with_value(value)` – nastav všechny hodnoty na danou hodnotu
 - `reverse()` – prohodí hodnoty z `[x, y]` na `[y, x]`
 - `is_negative()` – jsou všechny čísla v matici záporná?
 - `get_negative(), get_positive()` – vrátí všechna negativní/pozitivní čísla v matici
 - `zero_count()` – počet 0 v matici
-
- POZOR: const metody, předávání parametrů
 - Odevzdat do Gitlabu + merge request



Programování v C++ cvičení 1 (30.9.2020)

faltin@ksi.mff.cuni.cz

<https://fan1x.github.io/cpp20.html>



Distanční výuka



- **Web:** <https://fan1x.github.io/cpp20.html>
- **Zoom:** online cvičení
 - Informace k přihlášení v SIS/Nástěnka
- **Slack:** rychlá komunikace se cvičícím/přednášejícím/kolegy
 - Informace k přihlášení v SIS/Nástěnka
- **Gitlab:** odevzdávání úkolů
 - <https://gitlab.mff.cuni.cz/>
- **Recodex:** odevzdávání větších úkolů + automatická oprava
 - <https://recodex.mff.cuni.cz/>



Požadavky na zápočet



- Dokončené + odladěné příklady ze cvičení **v Gitlabu do pondělí 23:59** před dalším cvičením
 - Ikdyž se neúčastníte cvičení
- 2 DÚ **v ReCodexu**
 - 1. menší úkol: listopad, 15b
 - 2. větší úkol: prosinec, 25b
 - Body se započítávají do zkoušky
- Zápočtový program
 - Téma do **30.11.**
 - 1. odevzdání do **30.4.**
 - Finální odevzdání do **28.5.**
- Invidividuální podmínky je možné domluvit na začátku semestru



Požadavky na úkoly

- Konzistence (alespoň v rámci jednoho úkolu)
- Čitelný kód
 - Čitelný kód >> komentáře
- Bezpečný kód
 - `std::vector<int> a(20);` >> `int *a = new int[20];`
- Moderní kód
 - `std::array<int, 20> a;` >> `int a[20];`
- Funkčnost



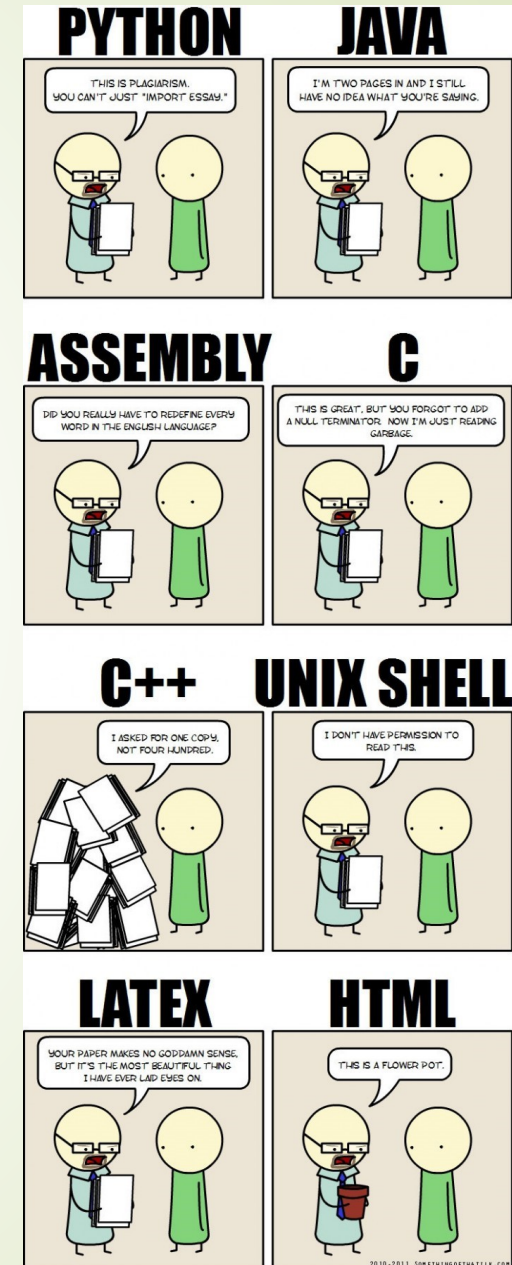
Bud'te aktivní 😊

- Nebojte se zeptat na Slacku/mailu/...
- Stáže
 - CppCon
 - Google, Microsoft, Oracle, ...
- BP, DP, SWP, PhD

Proč C++

- "C makes it easy to shoot yourself in the foot. C++ makes it harder, but when you do, it blows away your whole leg." - Bjarne Stroustrup
- C++ is like teenage sex:
 - It's on everyone's mind all the time.
 - Everyone talks about it all the time.
 - Everyone thinks everyone else is doing it.
 - Almost no one is really doing it.
 - The few who are doing it are
 - doing it poorly;
 - sure it will be better next time;
 - not practicing it safely.
- C++ != speed

Source: <http://devhumor.com/media/languages-as-essays>





Prostředí

➤ IDE

- Visual Studio (<https://portal.azure.com/...>)

- Clion

- Code::Blocks

- Eclipse

➤ Překladače

- MSVC, GCC, Clang+LLVM, ICC, ...



C++ (interesting) links

- Reddit, Slack, ...
- <http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines>
- <https://www.youtube.com/user/CppCon>
- <https://isocpp.org/>
- <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/>
- <https://gcc.godbolt.org/>
- <https://en.cppreference.com/w/>
- <http://www.cplusplus.com/>
- ...



Hello world

```
#include <iostream>
```

```
#include <string>
```

```
int main() {
```

```
    std::string name;
```

```
    std::cin >> name;
```

```
    std::cout << "Greetings from " << name << std::endl;
```

```
    return 0;
```

```
}
```



Užitečný kód

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int length(const string& s) { ... }

void pretty_print(const vector<string>& a) { ... a[i] ... }

int main(int argc, char** argv) {
    vector<string> arg(argv, argv+argc);
    if (arg.size() > 1 && arg[1] == "--help") {
        cout << "Usage: myprg [OPT]... [FILE]..." << endl;
        return 8;
    }
    pretty_print(arg);
    return 0;
}
```



Úkoly 30.9.2020

1. Hello world
2. Program pozdraví všechny lidi (jména zadaná jako argumenty programu)
 - ▀ ``Hello.exe Adam Bedrich Cecilie``
 - ▀ Pozor na první argument, tedy ``arg[0]``
3. Sčítání čísel zadaných jako argumenty
 - ▀ ``std::stoi()`, ...`
4. Jednoduchá kalkulačka nad zadanými argumenty
 - ▀ Jenom čísla a binární operace `+`, `-`, `*`, `/`
 - ▀ ``Calc.exe 1+2*3-4/5``