

NPRG 041 – cvičení 9

Programování v C++

Jiří Klepl

mail:

klepl@d3s.mff.cuni.cz

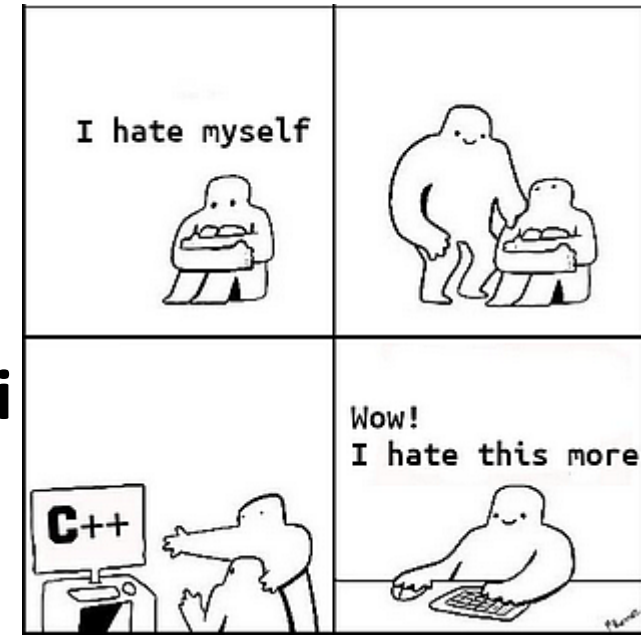
mattermost:

<https://ulita.ms.mff.cuni.cz/mattermost/ar2324zs/channels/nprg041-cpp-klepl>

Agenda

- **Osmá úloha: Kontrola linků v markdown dokumentaci**

- Práce s file systémem přes `std::filesystem`
- Regulární výrazy
- Formátování výstupu přes `std::format`
- Měření času s `chrono`
- Náhoda



Filesystem

```
#include <filesystem>
```

- Knihovna pro platformě nezávislou práci s filesystemem
- Základní třídy a funkce (**pozor, většina háže výjimky**):
 - **path**
 - Dekompozice: root, parent direktory, filename (stem + extension)
 - Konkatenace přes operator/: full_path = first_part / second_part
 - Překlad mezi relative a absolute, vyhazování useless .. a ., překlad z linuxových / do \
 - **(recursive_)directory_iterator**
 - Dovoluje procházet soubory (rekurzivně), optional argumenty (např. symlink follow)
 - **directory_entry**
 - Získávání metadat o souborech: cesta, typ (reg./dir./fifo/...), práva, velikost, ...
 - **Vytváření složek/symlinků/hardlinků, mazání, přejmenovávání, práva, ...**

Ukázka funkcí na práci se složkami

```
namespace fs = std::filesystem;           // namespace alias

fs::create_directory("example_directory"); // mkdir example_directory
fs::create_directories("example/dir");     // mkdir -p example/dir

std::ofstream("somefile.txt") << "Hello, World!";

fs::remove_all("example_directory");      // rm -rf example_directory
fs::remove("somefile.txt");               // rm somefile.txt
```

Ukázka procházení složek

```
namespace fs = std::filesystem;

// Recursive iterator that DOES NOT follow symlinks
for (const auto& entry : fs::recursive_directory_iterator("./some_dir/")) {
    std::cout << entry.path() << std::endl;
}

using fs::directory_options::follow_directory_symlink; // to shorten the names
// alternatively:
//     using dir_opts = fs::directory_options; ... dir_opts::follow_directory_symlink

// Recursive iterator that follows symlinks
for (const auto& entry : fs::recursive_directory_iterator("./some_dir",
                                                         follow_directory_symlink)) {
    std::cout << entry.path() << std::endl;
}
```

Ukázka čtení metadat

```
namespace fs = std::filesystem;

// Function to check execute permissions for owner, group, and others
bool has_execute_permission(const fs::path& path) {
    auto perms = fs::status(path).permissions();
    return (perms & fs::perms::owner_exec) != fs::perms::none ||
           (perms & fs::perms::group_exec) != fs::perms::none ||
           (perms & fs::perms::others_exec) != fs::perms::none;
}

// Usage within the directory iteration
for (const auto& entry : fs::directory_iterator("path/to/directory")) {
    if (entry.is_regular_file() && has_execute_permission(entry.path())) {
        std::cout << entry << ": " << entry.file_size() << " bytes\n";
    }
}
```

Ukázka čtení metadat

```
namespace fs = std::filesystem;

// Function to check execute permissions for owner, group, and others
bool has_execute_permission(const fs::path& path) {
    auto perms = fs::status(path).permissions();
    return (perms & fs::perms::owner_exec) != fs::perms::none ||
           (perms & fs::perms::group_exec) != fs::perms::none ||
           (perms & fs::perms::others_exec) != fs::perms::none;
}

// Usage within the directory iteration
for (const auto& entry : fs::directory_iterator("path/to/directory")) {
    if (entry.is_regular_file() && has_execute_permission(entry.path())) {
        std::cout << entry << ": " << entry.file_size() << " bytes\n";
    }
}
}
```

Čtení jednotlivých práv jako u bitů, to samé kombinování přes |

Regular Expressions

- Objekty:
 - **std::regex** – reprezentuje regulární výraz
 - **std::smatch** – reprezentuje nalezený výskyt výrazu v zadaném stringu
 - Pomocí operátoru [] lze přistoupit k podvýrazu (nultý odpovídá celému výrazu)
 - Pomocí .str(index) lze získat stringovou reprezentaci podvýrazu (zkratka za [index].str())
 - **std::regex_iterator** – slouží pro iterování více výskytů výrazu
- Funkce (detailedly podle flagů):
 - **std::regex_match** – zjistí, zda zadaný string odpovídá výrazu
 - **std::regex_search** – najde substring odpovídající výrazu
 - **std::regex_replace** – provádí nahrazování

Ukázka procházení nálezů regexu

```
const std::string s = "Quick brown fox.";

// raw string so that we don't have to escape backslashes;
// parentheses are not a part of the regular expression
std::regex words_regex(R"([\w]+)"); // read as: R"delim(<raw string>delim"
auto words_begin = std::sregex_iterator(s.begin(), s.end(), words_regex);
auto words_end = std::sregex_iterator();

std::cout << "Found " << std::distance(words_begin, words_end) << " words: ";

for (std::sregex_iterator i = words_begin; i != words_end; ++i)
{
    std::smatch match = *i;
    std::cout << match.str() << (std::next(i) != words_end ? ", " : "\n");
}

// Output: Found 3 words: Quick, brown, fox
```

Ukázka procházení nálezů regexu

```
const std::string s = "Quick brown fox.";

// raw string so that we don't have to escape backslashes;
// parentheses are not a part of the regular expression
std::regex words_regex(R"([\w]+)"); // read as: R"delim(<raw string>delim"
auto words_begin = std::sregex_iterator(s.begin(), s.end(), words_regex);
auto words_end = std::sregex_iterator();

std::cout << "Found " << std::distance(words_begin, words_end) << " words: ";

for (std::sregex_iterator i = words_begin; i != words_end; ++i)
{
    std::smatch match = *i;
    std::cout << match.str() << (std::next(i) != words_end ? ", " : "\n");
}

// Output: Found 3 words: Quick, brown, fox
```

delim je optional součást
raw stringu – vyznačuje
začátek a konec

Format

```
// implicit argument ordering; // writes "Hello world!"
std::cout << std::format("{} {}!", "Hello", "world") << std::endl;

// explicit argument ordering; writes "Hello world!"
std::cout << std::format("{1} {0}!", "world", "Hello") << std::endl;

// type-safe argument access; writes "Hello 42 3.14!"
std::cout << std::format("{} {} {}!", "Hello", 42, 3.14) << std::endl;

// Formatting of a table
std::cout << std::format(":{^33}", "Table name") << std::endl; // center alignment
std::cout << std::format(":{<10} {<10} {<10}", "col1", "col2", "col3") << std::endl;
std::cout << std::format(":{<10} {<10} {<10}", 1, 2, 3) << std::endl;
std::cout << std::format(":{<10} {<10} {<10}", 123456, 1234567, 12345678) << std::endl;
```

Chrono

```
#include <chrono>
```

- Knihovna na měření času; od C++20 i formáty + timezones + kalendář
 - **system_clock** – měří unixový čas podle systému
 - Systém může kdykoliv přetočit hodiny, takže není vhodný pro měření intervalů
 - **steady_clock** – monotonický čas na měření intervalů
 - Není vztažený k reálnému bodu v čase, není vhodný na získávání aktuálního času
 - **high_resolution_clock** – čas s nejnižším dostupným tikem
 - Nemusí být monotonický ani se vztahovat k reálnému času
 - Implementace v roce 2023 (podle cppreference):
 - MSVC: alias ke steady_clocku
 - libstdc++ (GNU – GCC): alias k system_clocku
 - libc++ (llvm – clang): alias ke steady_clocku (pokud to lze), jinak system_clock

Ukázka měření času

```
namespace { int local_function(int i = 0) {
    for (; i < 500'000; ++i) {
        i += i % 3;
    }

    return i;
}

// int main():
auto start = std::chrono::high_resolution_clock::now(); // Start the timer
int number = local_function(0); // Code to be benchmarked
auto stop = std::chrono::high_resolution_clock::now(); // Stop the timer

auto duration = std::chrono::duration_cast<std::chrono::microseconds>(stop - start);
std::cout << "Time taken to compute the number: " << duration << std::endl;

return number & 1; // so that the compiler doesn't throw away the computation
```

Random ukázka

```
#include <random>
```

```
std::random_device rd{};
std::mt19937 gen{rd()};

// define normal distribution with mean=5.0 and stddev=2.0
std::normal_distribution d{5.0, 2.0};

// draw a sample from the distribution and round it to an integer
auto random_int = [&d, &gen]{ return std::round(d(gen)); };

std::map<int, int> hist{}; // define a histogram
for (int n = 0; n != 10000; ++n)
    ++hist[random_int()]; // increment a bin of the histogram

for (auto [x, y] : hist) // print the histogram
    std::cout << std::setw(2) << x << ' ' << std::string(y / 200, '*') << '\n';
```

```
-3
-2
-1
0
1 *
2 ***
3 *****
4 *****
5 *****
6 *****
7 *****
8 ***
9 *
10
11
12
```

Dnešní úloha:

Kontrola linků v markdown dokumentaci

- Zadání na <https://gitlab.mff.cuni.cz/teaching/nprg041/klepl/web/-/tree/master/data/sources/doc>
 - Konkrétní úkoly: https://gitlab.mff.cuni.cz/teaching/nprg041/klepl/web/-/blob/master/data/sources/doc/link_checker.md
- Celá dokumentace (součást zadání – je to dokumentace k neexistujícímu projektu) i se zdrojovým kódem link-checkeru by měla být v labs/doc

