

NPRG041 – C++

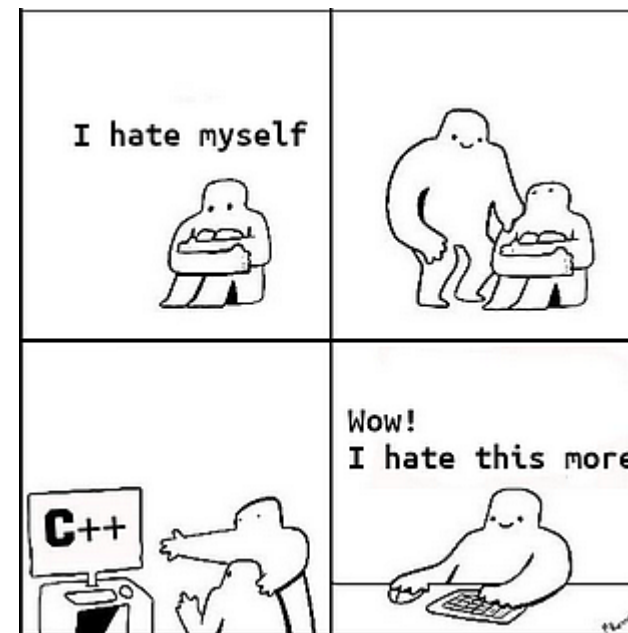
cvičení – Jiří Klepl

mattermost: ulita/2526ZS: nprg041-cpp-klepl (inv na SIS nástěnce)

klepl@d3s.mff.cuni.cz

Agenda

- Osmá úloha: **link_checker**
- Práce s file systémem přes **std::filesystem**
- Regulární výrazy (**std::regex**)
- Formátování výstupu přes **std::format**
- Měření času s **std::chrono**
- Náhoda



Filesystem

```
#include <filesystem>
```

- Knihovna pro platformě nezávislou práci s filesystemem
- Základní třídy a funkce (**pozor, většina hází 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, ...**

Path

```
namespace fs = std::filesystem;
fs::path p = fs::current_path();
std::cout << "The current path " << p << " decomposes into:\n"
           << "root-path " << p.root_path() << '\n'
           << "relative path " << p.relative_path() << '\n';
```

The current path `"/app"` decomposes into:
root-path `"/"`
relative path `"app"`

The current path `"C:\\Windows\\TEMP\\compiler-explorer-compiler20241029-4456-12p4p9r.bbvem"` decomposes into:
root-path `"C:\\"`
relative path `"Windows\\TEMP\\compiler-explorer-compiler20241029-4456-12p4p9r.bbvem"`

Ukázka funkcí na práci se složkami

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

fs::create_directory("example_directory"); // mkdir example_directory
fs::create_directories(fs::path("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 (detaily 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
const 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
const 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 2024 (podle cppreference):
 - **MSVC**: alias ke steady_clocku
 - **libstdc++ (GCC)**: alias k system_clocku
 - **libc++ (Clang)**: alias ke steady_clocku (pokud to lze), jinak system_clock

Ukázka měření času

```
namespace { int local_function(int i = 0) {  
    for (; i < 100'000'000; ++i) {  
        i += i % 3;  
    }  
    return i;  
}  
  
}  
  
// int main():  
namespace chrono = std::chrono;  
auto start = chrono::high_resolution_clock::now(); // Start the timer  
int number = local_function(0); // Code to be benchmarked  
auto stop = chrono::high_resolution_clock::now(); // Stop the timer  
  
auto duration = chrono::duration_cast<chrono::microseconds>(stop - start);  
std::cout << "Time taken to compute the number: " << duration << std::endl;  
  
// print the result so that the compiler doesn't throw away the computation  
std::cout << number << std::endl;
```

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
```

Kontrola linků v markdown dokumentaci

- Zadání:
 - <https://www.ksi.mff.cuni.cz/teaching/nprg041-klepl-web/data/09/doc.zip>
 - <https://www.ksi.mff.cuni.cz/teaching/nprg041-klepl-web/data/09/doc.tar.gz>
- V doc je soubor **link_checker.md** s návodem