NPRG041 – C++

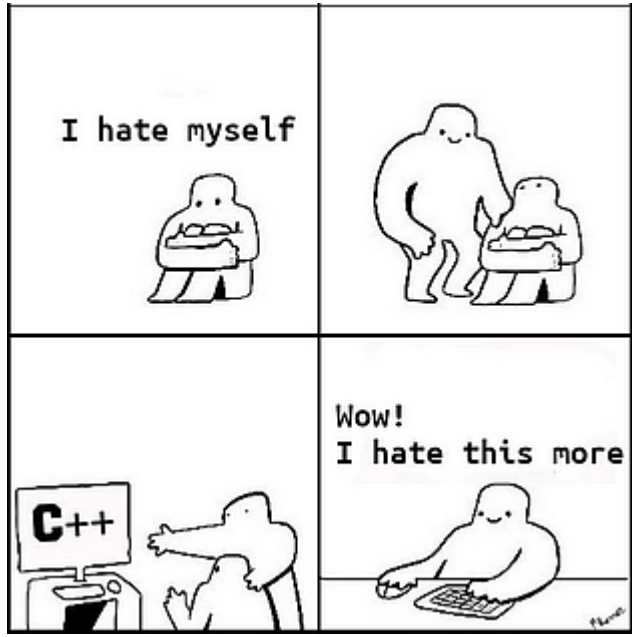
cvičení – Jiří Klepl

mattermost: ulita/2425ZS: nprg041-cpp-klepl (inv na SIS nástěnce)

Klepl@d3s.mff.cuni.cz

Agenda

- Předávání parametrů
- Referenční a hodnotová sémantika
- Pole a cykly
- Práce s textem
  - File streamy
  - char*, std::string a std::string_view
  - Práce se znaky



Nejběžnější způsoby předávání parametrů

- void foo(**Object object**) – **Používat pro triviální objekt (např. číslo)**
  - ⚠️ object je lokální (kopie) – zápisy do **object**u nevidíme venku
  - ☠️ neopatrné použití (např předávání **std::vector**) → drahé kopírování
- void foo(**const Object& object**) – **Pro objekt, který chceme číst**
  - ✅ nejběžnější
  - ✅ zdůrazňuje, že chceme cizí objekt pouze číst
- void foo(**Object& object**) – **Pro objekt, do kterého chceme zapisovat**
  - ⚠️ dovoluje nám omylem přepsat objekt
  - ⚠️ neumí pak přijmout temporary objekty: **foo(Object())**

Referenční vs hodnotová semantika

- Vše v C++ má samo o sobě hodnotovou semantiku
  - Neexistuje C#-ový kontrast mezi **class** a **struct**
  - Předávání hodnotou je default, Předávání referencí se značí znakem &
- Některé typy zastupují reference
  - Například pointery: předáním **char*** se text nekopíruje
    ⚠️ Předání **std::string** hodnotou (tj. bez &) znamená vytvoření nového objektu
  - Dnes si ukážeme
    - **std::span** – „pointer“ na pole
    - **std::string_view** – „pointer“ na string

C vs C++ pole

`#include <array>` `#include <vector>`

- C
  - Statický array na stacku: **T array[N]**
    - ⚠️ nemá metody, chová se jako tzv. „raw“ pointer **T***
  - Dynamický array na haldě: **T* array = malloc(sizeof(T) * N)**
    - ℹ️ v řeči C++ bychom psali: **T* array = new T[N]**
    - ☠️ musíme ručně dealokovat (funkce **free** pro **malloc**, **delete array** pro **new**)
    - ☠️ nevíme kdo je vlastník **T*** pointeru → někdy příště si ukážeme tzv. „chytré“ pointery
- C++
  - Statický array na stacku: **std::array<T, N> array**
    - ✅ Má metody: např. **.size()**
  - Dynamický array na haldě: **std::vector<T> array**
    - ✅ Sám uklidí paměť ve svém destruktoru
    - ✅ Umí metody jako **.push_back(new_last)**, **.back()**, **.pop_back()**

Range-based for loop

- Pracuje v podstatě se vším, co má začátek a konec (tzv. range)

```
int main() {
    std::string name = "Jirka Klepl";

    for (char c : name)
        std::cout << c << std::endl;
}
```

Co když nechci psát konkrétní typ/referenci?

```
for (auto&& whatever : fruit)
    std::cout << whatever << std::endl;
```

```
int main() {
    std::vector<std::string> fruit{
        "apples",
        "pears",
        "grapes",
        "melons"
    };

    // don't forget to add a reference
    for (const std::string& type : fruit)
        std::cout << type << std::endl;
}
```

File streamy

#include <fstream>

- Basic použití
  - Čtení ze souboru: `std::ifstream file("name.txt");`
  - Zápis do souboru: `std::ofstream file("name.txt");`
  - Podporují >> a << jako třeba `std::cin`/`std::cout`, `.eof()`, atd.

⚠️ `(file) streamy neházejí výjimky při chybách`

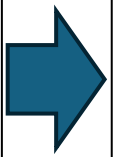
```
std::ifstream file("pairs.txt");
std::string word;
int number;

while (!file.eof()) {
    file >> word >> number;
    std::println("number: {1}, word: {0}", word, number);
}
```

```
dog 42
Jirka 5

C++ 10 C#
20

Tady error
```



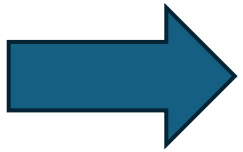
```
number: 42, word: dog
number: 5, word: Jirka
number: 10, word: C++
number: 20, word: C#
number: 0, word: Tady
number: 0, word: Tady
number: 0, word: Tady
number: 0, word: Tady
number: 0, word: Tady
number: 0, word: Tady
```

Je dobré kontrolovat správné čtení/zápis

```
std::string filename = "pairs.txt";
std::ifstream file(filename);
std::string word;
int number;

if (!file.is_open())
    std::println("failed to open {}", filename);

while (file.good()) {
    file >> word >> number;
    std::println("number: {1}, word: {0}", word, number);
}
```



```
number: 42, word: dog
number: 5, word: Jirka
number: 10, word: C++
number: 20, word: C#
number: 0, word: Tady
```

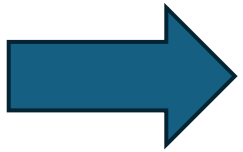
Stále zde máme chybu

Je dobré kontrolovat správné čtení/zápis

```
std::string filename = "pairs.txt";
std::ifstream file(filename);
std::string word;
int number;

if (!file.is_open())
    std::println("failed to open {}", filename);

while (file >> word >> number) {
    std::println("number: {1}, word: {0}", word, number);
}
```



```
number: 42, word: dog
number: 5, word: Jirka
number: 10, word: C++
number: 20, word: C#
```

Teď už to vypadá dobře

char* vs std::string vs std::string_view

char* = pointer na pole znaků (cstring)

✅ předáváním nehrozí kopírování, jednoduchý typ

☠️ není jasné, kdo má ten cstring dealokovat

☠️ srovnávání cstringů srovnává ty pointery, ne text

char* vs std::string vs std::string_view

char* = pointer na pole znaků (cstring)

✅ předáváním nehrozí kopírování, jednoduchý typ

☠️ není jasné, kdo má ten cstring dealokovat

☠️ srovnávání cstringů srovnává ty pointery, ne text

```
int main(int argc, char *argv[]) {
    if (argc > 1 && argv[1] == "--help")
        std::println("Where is the help?");
}
```

char* vs std::string vs std::string_view

#include <string>

std::string = trojice ("data", délka, kapacita)

✅ je bezpečné je předávat, porovnávat, sčítat, ...

✅ dealokují se automaticky

⚠️ předáváním bez zdůraznění reference (&) se kopírují

ℹ️ umí efektivně appendovat chary...

```
int main(int argc, char *argv[]) {
    std::vector<std::string> args(argv + 1, argv + argc);
    std::string help = "help"; here
    if (args.size() >= 1 && args[0] == "--help")
        std::println("Ahh, here is {}", "the " + help);
}
```

char* vs std::string vs std::string_view

```
#include <string_view>
```

std::string_view = „pointer“ na rozsah textu

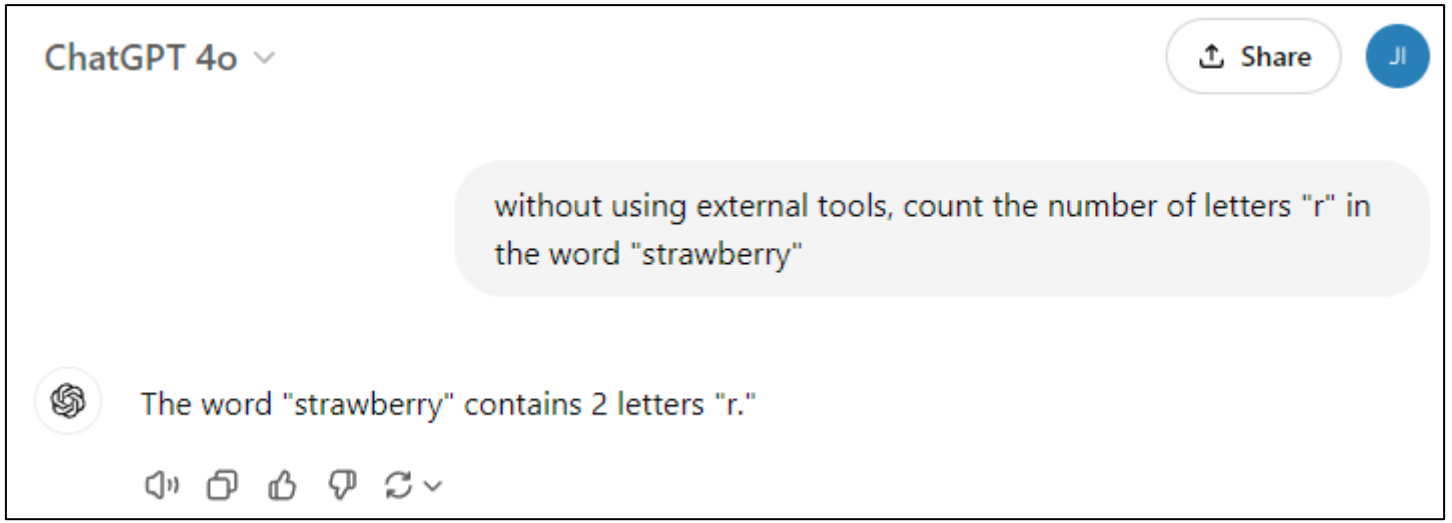
✅ je bezpečné je předávat i porovnávat

✅ předáváním nic nekopírujeme

⚠️ nejde ho sčítat a některé std funkce s ním neumí pracovat

např. std::stoi

ℹ️ umí „ukazovat“ jak na char*, tak na std::string




```
void count_r(std::string_view word) {
    size_t count = 0;
    for (char c : word) {
        if (c == 'r')
            ++count;
    }

    println("The word \"{}\" contains {} letters \"r.\"", word, count);
}
```

Takes char* and std::string as well
→ doesn't copy the string

Příklad: prep_for_ovecky

- Zadání: příprava na první ReCodexový úkol
  1. V cmd argumentech je seznam souborů (nevíme, jestli existují)
  2. Každý soubor obsahuje slova (některá jsou čísla)
     I. Připravíme si sum=0 a r_count=0
     II. Pokud je slovo číslo (slovo pouze z číslic), přičteme jeho součet číslic k sumě
     III. Pokud slovo není číslo, spočítáme kolik má písmen „r“ a přičteme k jejich počtu
        - A to slovo zahlásíme do std::cerr ve formátu "File {filename} contains \"{word}\"" v lower_casu
  3. Pro každý soubor vytiskneme "File {filename}: sum={sum}, r_count={r_count}"

#include <cctype>

std::isdigit, std::isalpha, std::isalnum
std::islower, std::isupper
std::tolower, std::toupper

```
char c = '5';
if (std::isdigit(c)) {
    // c >= '0' && c <= '9'
    int n = c - '0';

    // n == 5
}
```

prep_for_ovecky example.txt does_not_exist.txt