PODSTAWY C++ #2

CODERS SCHOOL

MATEUSZ ADAMSKI

ŁUKASZ ZIOBROŃ

AGENDA

1. STL - co to?
2. `std::vector`
3. Pętla `for` po kolekcji
4. `std::string`
5. `std::map`

ZADANIA

Repo GH `coders-school/kurs_cpp_podstawowy`

https://github.com/coders-school/kurs_cpp_podstawowy/tree/master/module2

KRÓTKIE PRZYPOMNIENIE

CO JUŻ WIEMY

- Co zapamiętaliście z poprzednich zajęć?
- Co sprawiło największą trudność?
- Co najłatwiej było wam zrozumieć?

PODSTAWY C++

STL

CODERS SCHOOL

STANDARD TEMPLATE LIBRARY

- standardowa biblioteka szablonów (Standard Template Library) dostępna w standardzie języka C++
- często używane rzeczy z STLa:
  - `std::vector<T>`
  - `std::string`
  - `std::map<K, V>`
  - `std::cout` i `std::cin`
  - iteratory

PODSTAWY C++

std::vector<T>

CODERS SCHOOL

CECHY `std::vector<T>`

- bardzo powszechnie używany
- dynamiczna tablica
- nie musimy z góry precyzować ile ma być elementów
- sam zarządza pamięcią
  - zadba o alokację nowej pamięci, gdy będzie to potrzebne
  - zadba o dealokację pamięci, gdy już jej nie będziemy potrzebować

UTWORZENIE WEKTORA

```
std::vector<int> numbers;
```

- wektor zawsze musi wiedzieć jakiego typu przechowuje dane
- typ danych podajemy w nawiasach trójkątnych <>

INICJALIZACJA WEKTORA WARTOŚCIAMI

```
std::vector<int> numbers = {1, 2, 3, 4, 5};
std::vector<int> numbers {1, 2, 3, 4, 5};
```

- oba typy inicjalizacji (z = i bez) są równoważne w przypadku wektora

OPERACJE NA WEKTORZE

- dodanie elementu do wektora
  - `numbers.push_back(5)`
- odczytanie elementu z wektora
  - `numbers[1]`
- przypisanie wielu elementów do wektora
  - `numbers = {1,2,3,4,5}`
- pobieranie pierwszego elementu z wektora
  - `numbers.front()`
- pobieranie ostatniego elementu z wektora
  - `numbers.back()`

Dokumentacja na cppreference.org

PODSTAWY C++

PĘTLA `for` PO KOLEKCJI

CODERS SCHOOL

ZAKRESY

- Każdy kontener (w tym również tablica, czy wektor) posiada swój koniec i początek
  - funkcja `begin()` zwraca początek kontenera
  - funkcja `end()` zwraca koniec kontenera
  - (w dużym uproszczeniu, temat rozszerzymy przy iteratorach)

RANGE BASED `for` LOOP

Dzięki informacji o początku i końcu zakresu, możemy napisać pętlę iterującą po całym zakresie kontenera.

```
for (auto i = vec.begin(); i != vec.end(); ++i) {
    auto element = *i;
    // do sth on element
}
```

Taki zapis jest jednak niepotrzebnie złożony i mało czytelny. Dlatego powstały `range loop` które umożliwiają łatwy zapis `for (typ nazwa : kontener)`.

Kompilator może sam go wygenerować powyższy kod, jeśli użyjemy poniższego zapisu.

```
for (auto element : vec) {
    // do sth on element
}
```

ZADANIE

Napisz funkcję `printVector`, która przyjmie jako argument `std::vector<std::string>` i wypisze jego zawartość przy użyciu pętli for przy kolekcji. Każdy element w nowej linii. Pobierz zadanie

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

// Implement printVector

int main() {
    std::vector<std::string> vec {
        "Hello Coders School!",
        "Welcome to the best C++ course ever",
        "Man, this is crazy :)"
    };
    printVector(vec);
    return 0;
}
```

ZADANIE

Napisz funkcję `concatenateVector`, która przyjmie jako argumenty 2 wektory a następnie zwróci jeden, który będzie zawierał naprzemiennie elementy z pierwszego i drugiego wektora. Np. dla poniższych vec1 i vec2 powinna zwrócić: `{1, 11, 2, 12, 3, 13, 4, 14, 5, 15}` Pobierz zadanie

```cpp
#include <iostream>
#include <vector>

// Implement concatenateVector

int main() {
    std::vector<int> vec1 {1, 2, 3, 4, 5};
    std::vector<int> vec2 {11, 12, 13, 14, 15};

    auto vec = concatenateVector(vec1, vec2);
    for (const auto& el : vec) {
        std::cout << el << " ";
    }
    return 0;
}
```

PODSTAWY C++

std::string

CODERS SCHOOL

KONTENER ZNAKÓW - `std::string`

- specjalny kontener, który przechowuje znaki
- `std::string` ma również swój początek i koniec, jak każdy kontener
- podobne funkcje jak `std::vector`

OPERACJE NA `std::string`

- dodanie znaku na koniec
  - `str.push_back('a')` (nikt tak nie robi :))
  - polecamy `str += 'a';`
- odczytanie pojedynczego znaku
  - `str[1]`
- inicjalizacja
  - `std::string str("Witam")`
  - `std::string str = "Witam"`
- przypisanie całego napisu
  - `str = "Witam"`
- pobieranie pierwszego znaku
  - `str.front()`
- pobieranie ostatniego znaku
  - `str.back()`