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::list`
6. `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
- znajduje się w jednym, ciągłym obszarze pamięci (tak jak tablica)
- 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()`

PODSTAWY C++

std::list<T>

CODERS SCHOOL

LISTA

PYTANIE: JAKIE CECHY MIAŁ `std::vector<T>`?

Lista w przeciwieństwie do wektora jest porozrzucana po pamięci. Co czasami jest wygodne, gdyż możemy wykorzystać fragmenty pamięci, które mogłyby, by być niedostępne dla wektora.

PYTANIE: SKĄD ELEMENTY LISTY WIEDZĄ O SWOIM WZAJEMNYM ISTNIENIU?

Każdy element listy przechowuje wskaźnik na element następny (lista jedno kierunkowa) lub następny i poprzedni (lista dwukierunkowa).

OPERACJE NA `std::list`

- pobranie pierwszego i ostatniego elementu listy
  - `front()`
  - `back()`
- początek i koniec mapy
  - `begin()`
  - `end()`
- informacja o liczbie elementów w liście
  - `size()`
- informacja czy lista jest pusta
  - `empty()`
- dodanie elementu na koniec listy
  - `push_back()`
- **NOWOŚĆ** dodanie elementu na początek listy
  - `push_front()`
- **NOWOŚĆ** sortowanie elementów listy (nie możemy korzystać z `std::sort` dla listy)
  - `sort()`

Dokumentacja na cppreference.org

PYTANIE: JAK DOSTAĆ SIĘ DO 10 ELEMENTU LISTY?

Ponieważ każdy element listy wie tylko o poprzednim i następnym elemencie, nie możemy tak łatwo dostać się do 10 elementu listy.

Dostęp do pierwszego elementu możemy otrzymać przez `front()` lub `*begin()`

```
int main() {
    std::list<int> list {1, 2, 3, 4, 5};
    std::cout << *list.begin();
    std::cout << list.front();
}
```

Dostęp do 10 elementu możemy uzyskać przechodząc od 1 do 10.

```
int main() {
    std::list<int> list {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    auto it = list.begin();
    for (size_t i = 0 ; i < 10 ; ++i) {
        ++it; // jump to next element
    }
    std::cout << *it;
}
```

Zajmuje to więcej czasu, niż dostanie się do 10 elementu w `std::vector`.

ZADANIE

Napisz funkcję, która przyjmuje wektor i zwraca listę, która zawiera posortowane wartości z wektora. Pobierz zadanie

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

// Implement createSortedList
// It should take a vector and return a list of sorted elements
// add proper include :)

int main() {
    std::vector<int> vec{2, 3, 4, 1, 6, 5, 8, 7, 9, 0};
    auto list = createSortedList(vec);

    for (const auto& el : list)
        std::cout << el << " ";

    return 0;
}
```

PYTANIE: KIEDY OPŁACA SIĘ UŻYWAĆ `std::list`, A KIEDY `std::vector`?

PODSTAWY C++

std::map<K, V>

CODERS SCHOOL

MAPA, SŁOWNIK

- mapa to zbiór par (klucz - Key, wartość - Value)
- `std::map` w C++ to odpowiednik `dict` z Pythona

Przykładowo tworzymy kolekcję ulubionych płyt i układamy je w szafce.

Oczywiście płyt tych mamy ogromną liczbę i chcielibyśmy móc łatwo odnaleźć płytę, gdy będziemy jej poszukiwać.

W tym celu numerujemy sobie wszystkie płyty i zapisujemy sobie na kartce informacje, pod jakim numerem znajduje się określony tytuł. W ten sposób tworzymy właśnie mapę.

```
std::map<size_t, std::string> discs {
    {1, "The Lord of the Rings: The Fellowship of the Ring"},
    {2, "The Lord of the Rings: The Two Towers"},
    {3, "The Lord of the Rings: The Return of the King"}
};
```

Kluczem jest tutaj numer, natomiast wartością jest tytuł filmu.

OPERACJE NA `std::map`

- początek i koniec zakresu
  - `begin()`
  - `end()`
- informacje o liczbie elementów w mapie
  - `size()`
- informacja czy mapa jest pusta
  - `empty()`
- dostęp do elementu dla określonego klucza
  - `operator[key]`
- dodanie parę (klucz, wartość) do mapy o ile taka para jeszcze w niej nie występuje
  - `insert({key, value})`

Dokumentacja na cppreference.org

PYTANIE

Co się wydarzy, gdy zawołamy na wspomnianej mapie:

```
discs[4] = "Harry Potter";
```

Przypisanie czegoś do elementu mapy poprzez `operator[ ]` sprawia, że:

- jeżeli istnieje już wartość dla danego klucza to ją podmienimy.
- gdy nie istnieje wartość dla danego klucza, to utworzymy nową parę (klucz, wartość)

WYKONAJMY TEN KOD

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

void Print(const std::map<size_t, std::string>& map) {
    for (const auto& pair : map) {
        std::cout << pair.first << " | " << pair.second << '\n';
    }
}

int main() {
    std::map<size_t, std::string> discs {
        {1, "The Lord of the Rings: The Fellowship of the Ring"},
        {2, "The Lord of the Rings: The Two Towers"},
        {3, "The Lord of the Rings: The Return of the King"}
    };

    Print(discs);
    std::cout << "\nAfter adding a new element\n";
    discs[4] = "Harry Potter";
    Print(discs);
    std::cout << "\nAfter modification of an element\n";
    discs[4] = "Harry Potter and the Philosopher's Stone";
    Print(discs);
}
```

WYNIK

```
1 | The Lord of the Rings: The Fellowship of the Ring
2 | The Lord of the Rings: The Two Towers
3 | The Lord of the Rings: The Return of the King

After adding a new element
1 | The Lord of the Rings: The Fellowship of the Ring
2 | The Lord of the Rings: The Two Towers
3 | The Lord of the Rings: The Return of the King
4 | Harry Potter

After modification of an element
1 | The Lord of the Rings: The Fellowship of the Ring
2 | The Lord of the Rings: The Two Towers
3 | The Lord of the Rings: The Return of the King
4 | Harry Potter and the Philosopher's Stone
```

ZADANIE

Napisz funkcję, która przyjmuje `std::vector<int>` oraz `std::list<std::string>` i zwraca mapę `std::map<int, std::string>`. Pobierz zadanie

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

// Implement createMap. It should take a vector and list and
// return a map of merge them as keys from the vector and values from the list


int main() {
    std::vector<int> vec{1, 2, 3, 4, 5};
    std::list<std::string> list{"One", "Two", "Three", "Four", "Five"};
    auto map = createMap(vec, list);

    for (const auto& pair : map)
        std::cout << pair.first << " | " << pair.second << '\n';

    return 0;
}
```

PODSTAWY C++

PODSUMOWANIE

CODERS SCHOOL

CO PAMIĘTASZ Z DZISIAJ?

NAPISZ NA CZACIE JAK NAJWIĘCEJ HASEŁ

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

PRACA DOMOWA

POST-WORK

- Jeśli nie wiesz czym jest `operator%` to się dowiedz. Przyda się do pracy domowej :)
- Zadanie 1 - AddEven (4 punkty)
- Zadanie 2 - NWD (LCM) i NWW (GCD) (6 punktów)
- Zadanie 3 - MaxOfVector (4 punkty)
- Zadanie 4 - GenerateSequence (4 punkty)

BONUS ZA PUNKTUALNOŚĆ

Za dostarczenie każdego zadania przed 31.05.2020 (niedziela) do 23:59 dostaniesz 2 bonusowe punkty (razem 8 punkty za 4 zadania).

ZADANIA W REPO

PRE-WORK

- Przypomnij sobie informacje o wskaźnikach np z wideo pana Zelenta
- Poczytaj o `enumach`
- Zainteresuj się tematem smart pointerów i poszukać informacji czym jest `std::shared_ptr` i `std::unique_ptr`
- Możesz przyjrzeć się plikom z testami w zadaniach domowych i spróbować dopisać własne przypadki testowe

CODERS SCHOOL