КАФЕДРА КОМП’ЮТЕРНИХ ТЕХНОЛОГІЙ

# ЗВІТ

# З ЛАБОРАТОРНОЇ РОБОТИ №1

# за курсом «Програмування»

# Варіант № 5

Виконав

студент групи МА-21-1

Гудзь Вячеслав

Дніпро, 2023

1. Постановка задачі

2. Опис розв’язку

3. Вихідний код програми

5. Аналіз завдання

# Постановка задачі

Постановка задачі

Розробити програму – консольний застосунок для ОС Windows, що демонструє можливості мови C++, а також його стандартної бібліотеки по роботі з потоками, контейнерами та ітераторами.

Використання всіх вищезгаданих можливостей є обов’язковим в кожному з варіантів індивідуальних завдань.

В кожному з нижченаведених варіантів завдань Вами повинні бути реалізовані основні можливості бібліотеки стандартних шаблонів: ітератор(и), пошук, упорядкування, сума або щось аналогічне зі згоди викладача.

При написанні програми слід враховувати необхідність забезпечення її зборки у двох режимах:

1. Створення виконуваного файлу із підключенням/використанням типів і алгоритмів бібліотеки стандартних шаблонів (STL).

2. Створення виконуваного файлу без застосування STL.

Наприклад: Ваше завдання – втілити алгоритм швидкого сортування для контейнеру «вектор»; в такому разі Ви мусите продемонструвати, як Ваша програма працює із Вашою реалізацією вектора, а також як вона працює із стандартною реалізацією вектора з бібліотеки STL. Увага: Ви пишете одну програму, а не дві. Необхідні зміни вихідного тексту для переналаштування з/на STL бажано робити за допомогою директив умовної компіляції.

Індивідуальний варіант завдання 3 ([hudz.v21@fpm.dnu.edu.ua](mailto:hudz.v21@fpm.dnu.edu.ua))

(16.14) Напишіть шаблон, що реалізує контейнер з тими ж функціями-членами та типами членів, як і стандартний vector (деякого нестандартного, не такого, як був на лекціях) ко-нтейнерного типу. Не модифікуйте вже існуючий контейнерний тип. Як би ви стали пра-цювати з функціональністю, яка запропонована нестандартним vector, але не реалізована стандартним?

# Опис розв’язку

За допомогою методів Visual Studio 2022 мною був реалізований нестандартний та не STL контейнер CustomVector, який може виконувати роботу з будь-якими шаблонними функціями бібліотеки algorithm та типами даних.  
За допомогою препроцесорів #ifdef, #ifndef, в моїй програмі можна використовувати стандартний вектор та навпаки мій «кастомний».  
Методи мого вектору правильно працюють з пам’яттю та звільнюють її після спрацювання деструктору.

# Вихідний код програми

#pragma once

#include <iostream>

#ifndef CUSTOMVECTOR\_H

#define CUSTOMVECTOR\_H

template <typename T>

class CustomVector {

private:

T\* data;

int size\_;

int capacity\_;

void moveData(T\* source, T\* destination, int count);

public:

CustomVector();

~CustomVector();

explicit CustomVector(int initialCapacity);

CustomVector(const CustomVector& other);

CustomVector& operator=(const CustomVector& other);

CustomVector(std::initializer\_list<T> values) : size\_(values.size()), capacity\_(values.size()) {

data = new T[capacity\_];

int i = 0;

for (const T& value : values) {

data[i++] = value;

}

}

void push\_back(const T& value);

void pop\_back();

void clear();

int size() const;

int capacity() const;

T& operator[](int index);

const T& operator[](int index) const;

void reserve(int newCapacity);

void resize(int newSize, const T& value = T());

bool empty() const;

T\* begin();

const T\* begin() const;

T\* end();

const T\* end() const;

};

template <typename T>

CustomVector<T>::CustomVector() : data(nullptr), size\_(0), capacity\_(0) {}

template <typename T>

CustomVector<T>::CustomVector(int initialCapacity) : size\_(0), capacity\_(initialCapacity) {

data = new T[capacity];

}

template <typename T>

CustomVector<T>::CustomVector(const CustomVector& other) : size\_(other.size), capacity\_(other.capacity) {

data = new T[capacity\_];

moveData(other.data, data, size\_);

}

template <typename T>

CustomVector<T>& CustomVector<T>::operator=(const CustomVector& other) {

if (this != &other) {

delete[] data;

size = other.size;

capacity = other.capacity;

data = new T[capacity];

moveData(other.data, data, size);

}

return \*this;

}

template <typename T>

CustomVector<T>::~CustomVector() {

delete[] data;

}

template <typename T>

void CustomVector<T>::moveData(T\* source, T\* destination, int count) {

for (size\_t i = 0; i < count; ++i) {

destination[i] = source[i];

}

}

template <typename T>

void CustomVector<T>::push\_back(const T& value) {

if (size\_ == capacity\_) {

reserve(capacity\_ \* 2 + 1);

}

data[size\_++] = value;

}

template <typename T>

void CustomVector<T>::pop\_back() {

if (size\_ > 0) {

--size\_;

}

}

template <typename T>

void CustomVector<T>::clear() {

size = 0;

}

template <typename T>

int CustomVector<T>::size() const {

return size\_;

}

template <typename T>

int CustomVector<T>::capacity() const {

return capacity\_;

}

template <typename T>

T& CustomVector<T>::operator[](int index) {

if (index < size\_) {

return data[index];

}

else {

throw std::out\_of\_range("Index out of range");

}

}

template <typename T>

const T& CustomVector<T>::operator[](int index) const {

if (index < size\_) {

return data[index];

}

else {

throw std::out\_of\_range("Index out of range");

}

}

template <typename T>

void CustomVector<T>::reserve(int newCapacity) {

if (newCapacity > capacity\_) {

T\* newData = new T[newCapacity];

moveData(data, newData, size\_);

delete[] data;

data = newData;

capacity\_ = newCapacity;

}

}

template <typename T>

void CustomVector<T>::resize(int newSize, const T& value) {

if (newSize > size) {

reserve(newSize);

for (size\_t i = size; i < newSize; ++i) {

data[i] = value;

}

}

size = newSize;

}

template <typename T>

bool CustomVector<T>::empty() const {

return size == 0;

}

template <typename T>

std::ostream& operator<<(std::ostream& os, const CustomVector<T>& vec) {

for (size\_t i = 0; i < vec.size(); ++i) {

os << vec[i];

}

return os;

}

template <typename T>

std::istream& operator>>(std::istream& is, CustomVector<T>& vec) {

T value;

vec.clear();

while (is >> value) {

vec.push\_back(value);

}

return is;

}

template <typename T>

T\* CustomVector<T>::begin() {

return data;

}

template <typename T>

const T\* CustomVector<T>::begin() const {

return data;

}

template <typename T>

T\* CustomVector<T>::end() {

return data + size\_;

}

template <typename T>

const T\* CustomVector<T>::end() const {

return data + size\_;

}

#endif

Main метод

#include <iostream>

#include <functional>

#include <algorithm>

#define USE\_STL\_VECTOR 1

#ifdef USE\_STL\_VECTOR

#include "CustomVector.h"

template<typename T>

using CV = CustomVector<T>;

#else

#include <vector>

template<typename T>

using CV = std::vector<T>;

#endif

template <typename Iterator, typename Function>

void my\_for\_each(Iterator begin, Iterator end, Function func) {

for (auto it = begin; it != end; ++it) {

func(\*it);

}

}

int main() {

CV<int> v = { 3,4,6,1,3,8,2,3,7,3,4,9,1,5,89,1,2 };

my\_for\_each(v.begin(), v.end(), [](int n) {

std::cout << n << " ";

});

std::cout << std::endl;

//std::sort(v.begin(), v.end());

int i = 0;

std::for\_each(v.begin(), v.end(), [&v, &i](int n)

{

if (n % 2 == 0)

{

v[i] \*= 2;

}

i++;

});

std::for\_each(v.begin(), v.end(), [](const int& n) { std::cout << n << " "; });

std::cout << std::endl;

auto max = std::find(v.begin(), v.end(), 89);

std::cout << max - &v[0] << std::endl;

auto result = std::find\_if(v.begin(), v.end(), [](int n) { return n == 8; });

std::for\_each(v.begin(), v.end(), [result, &v](const int& n)

{

if (n == \*result)

{

std::cout << result - &v[0] << std::endl;

}});

return 0;

}

# Аналіз роботи

Я навчився створювати шаблонні функції схожі на функції бібліотеки algorithm та створювати шаблонні класи для роботи з різними типами даних.  
Повторив спосіб роботи з препроцесорами та використав їх у програмі.