ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №6

за курсом «Об’єктно Орієнтованого Програмування»

студента групи ПЗ-21-2

Гудимова Дениса Сергійовича

кафедра математичного забезпечення ЕОМ, ДНУ

2022/2023 навч.р.

1. Постановка задачі

**Постановка задачі**

Розробити програму – 32-разрядний додаток 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.

**Індивідуальний варіант завдання**

(18.9) Напишіть набір об’єктів-функцій, які виконують бітові логічні операції над своїми операндами. Перевірте ці об’єкти на векторах з елементами char, int.

2. Опис розв’язку

Для того, щоб реалізувати поставлену задачу, було використано стандартні бібліотеки по роботі з контейнерами та ітераторами.

Для реалізації бітових логічних операцій було створено inline клас BitwiseLogicalOperations у якому реалізовано 3 логічні методи із загальними типами даних OR, AND та XOR, які виконують відповідні дії. Також було створено загальний клас Container<T> з приватними полями first\_vec та second\_vec з типом std::vector<T>. Для полів класу було відповідно створено гетери та сетери та конструктор за замовченням. Для зручного використання контейнеру були створені методи додавання, видалення елементів, метод для виведення векторів у консоль та внутрішні методи валідації для уникнення помилок а також метод console\_interface для зручної взаємодії з кодом.

3. Вихідний текст програми розв’язку задачі

**BitwiseLogicalOperations.h**

class BitwiseLogicalOperations

{

public:

// Функція для виконання бітового AND над двома числами

template <typename T>

static T bitwise\_AND(T a, T b) {

return a & b;

}

// Функція для виконання бітового OR над двома числами

template <typename T>

static T bitwise\_OR(T a, T b) {

return a | b;

}

// Функція для виконання бітового XOR над двома числами

template <typename T>

static T bitwise\_XOR(T a, T b) {

return a ^ b;

}

};

**Container.h**

#include <iostream>

#include <vector>

#include <algorithm>

#include "BitwiseLogicalOperations.h"

template <typename T>

class Container

{

private:

std::vector<T> first\_vec;

std::vector<T> second\_vec;

public:

Container() {} // default constructor

std::vector<T> get\_first\_vec() const;

std::vector<T> get\_second\_vec() const;

void set\_first\_vec(const std::vector<T> &);

void set\_second\_vec(const std::vector<T> &);

void print\_vectors();

void add\_element();

void remove\_elemnt(int i, int index);

bool validate\_size();

bool validate\_vectors\_size();

bool console\_interface();

};

**Container.cpp**

#include "Container.h"

template <typename T>

std::vector<T> Container<T>::get\_first\_vec() const {

return first\_vec;

}

template <typename T>

std::vector<T> Container<T>::get\_second\_vec() const {

return second\_vec;

}

template <typename T>

void Container<T>::set\_first\_vec(const std::vector<T>& vector) {

first\_vec = vector;

}

template <typename T>

void Container<T>::set\_second\_vec(const std::vector<T>& vector) {

second\_vec = vector;

}

template <typename T>

void Container<T>::print\_vectors() {

std::cout << "Vector 1: ";

if (first\_vec.empty()) {

std::cout << "is empty";

}

for (const T& element : first\_vec) {

std::cout << element << " ";

}

std::cout << std::endl;

std::cout << "Vector 2: ";

if (second\_vec.empty()) {

std::cout << "is empty";

}

for (const T& element : second\_vec) {

std::cout << element << " ";

}

std::cout << std::endl;

}

template <typename T>

void Container<T>::add\_element() {

T first\_element, second\_element;

std::cout << "\nEnter first element: ";

std::cin >> first\_element;

std::cout << "\nEnter second element: ";

std::cin >> second\_element;

first\_vec.push\_back(first\_element);

second\_vec.push\_back(second\_element);

}

template <typename T>

void Container<T>::remove\_elemnt(int i, int index) {

if (i == 1) {

if (first\_vec.empty()) {

std::cout << "Vector alredy empty";

return;

}

first\_vec.erase(first\_vec.begin() + index);

}

else if (i == 2) {

if (second\_vec.empty()) {

std::cout << "Vector alredy empty";

return;

}

second\_vec.erase(second\_vec.begin() + index);

}

else {

std::cout << "You enter incorect index of vector, it's only 1 or 2!!!";

}

}

template <typename T>

bool Container<T>::validate\_size() {

return first\_vec.size() == second\_vec.size();

}

template <typename T>

bool Container<T>::validate\_vectors\_size() {

if (!(validate\_size())) {

std::cout << "vector size not the same" << std::endl;

print\_vectors();

return false;

}

return true;

}

template <typename T>

bool Container<T>::console\_interface() {

int action, vector\_number, index;

T element;

std::cout << "Chose action:\n";

std::cout << "1) Add element\n";

std::cout << "2) Remove element by index\n";

std::cout << "3) Print all vectors\n";

std::cout << "4) Invoke bitwise AND\n";

std::cout << "5) Invoke bitwise OR\n";

std::cout << "6) Invoke bitwise XOR\n";

std::cout << "0) Exit out\n";

std::cin >> action;

if (action == 1) {

add\_element();

}

else if (action == 2) {

std::cout << "\nEnter vector(1 or 2) form wich you want to delete: ";

std::cin >> vector\_number;

std::cout << "\nEnter index: ";

std::cin >> index;

remove\_elemnt(vector\_number, index);

}

else if (action == 3) {

print\_vectors();

}

else if (action == 4) {

if (!validate\_vectors\_size()) {

return true;

}

std::vector<T> first\_vec = get\_first\_vec();

std::vector<T> second\_vec = get\_second\_vec();

// AND vector

std::vector<T> AND\_vector(get\_first\_vec().size());

std::transform(first\_vec.begin(), first\_vec.end(),

second\_vec.begin(), AND\_vector.begin(), BitwiseLogicalOperations::bitwise\_AND<T>);

std::cout << "Vector AND result: ";

for (T element : AND\_vector) {

std::cout << element << " ";

}

std::cout << std::endl;

}

else if (action == 5) {

if (!validate\_vectors\_size()) {

return true;

}

std::vector<T> first\_vec = get\_first\_vec();

std::vector<T> second\_vec = get\_second\_vec();

// OR vector

std::vector<T> OR\_vector(get\_first\_vec().size());

std::transform(first\_vec.begin(), first\_vec.end(),

second\_vec.begin(), OR\_vector.begin(), BitwiseLogicalOperations::bitwise\_OR<T>);

std::cout << "Vector OR result: ";

for (T element : OR\_vector) {

std::cout << element << " ";

}

std::cout << std::endl;

}

else if (action == 6) {

if (!validate\_vectors\_size()) {

return true;

}

std::vector<T> first\_vec = get\_first\_vec();

std::vector<T> second\_vec = get\_second\_vec();

// XOR vector

std::vector<T> XOR\_vector(get\_first\_vec().size());

std::transform(first\_vec.begin(), first\_vec.end(),

second\_vec.begin(), XOR\_vector.begin(), BitwiseLogicalOperations::bitwise\_XOR<T>);

std::cout << "Char vector XOR result: ";

for (T element : XOR\_vector) {

std::cout << element << " ";

}

std::cout << std::endl;

}

else if (action == 0) {

return false;

}

else {

std::cout << "Wrong action!\n";

}

return true;

}

**6Lab4SemC++(OOP).cpp**

#include <iostream>

#include "Container.cpp"

using namespace std;

int main() {

int conteiner\_type;

cout << "Select type of Container 1 is char, 2 is int" << endl;

cin >> conteiner\_type;

if (conteiner\_type == 1) {

Container<char> container;

bool res = true;

while (res) {

res = container.console\_interface();

}

}

else if (conteiner\_type == 2) {

Container<int> container;

bool res = true;

while (res) {

res = container.console\_interface();

}

}

return 0;

}

4. Опис інтерфейсу (керівництво користувача)

При запуску програми користувач повинен вибрати з яким типом даних він хоче працювати int або char

Після вибору Інструкції для користувача виводяться у консоль з усіма можливими діями, а необхідну дію користувач вибирає відповідною цифрою (рис. 4.1):

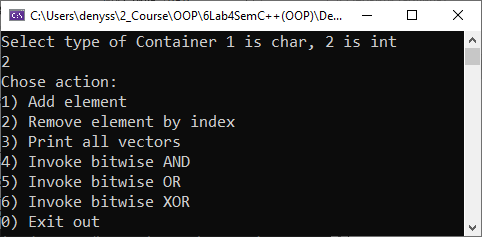


Рисунок 4.1 – Користувацький інтерфейс

1) Додати елементи у контейнери

2) Прибрати елемент із контейнери по індексу

3) Вивід у консоль векторів

4) Виклик логічного І

5) Виклик логічного АБО

6) Виклик логічного НЕ АБО

0) Вихід із застосунку

5. Опис тестових прикладів

Додавання елементів (5.1):

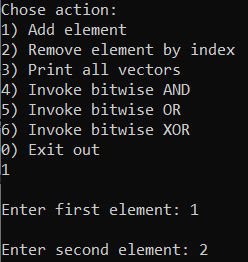


Рисунок 5.1 – Додавння елементів у обидва вектори

Приклад видалення елемента 2 з другого вектору після видалення вектори більше не рівні, а отже логічні операції не будуть працювати і необхідно видалити якийсь елемент з іншого вектору також (5.2):

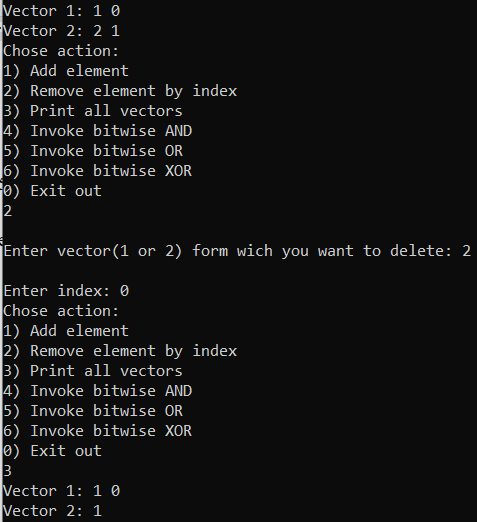


Рисунок 5.2 – Видалення елемента із вектора

Вивід елементів векторів у консоль (5.3):

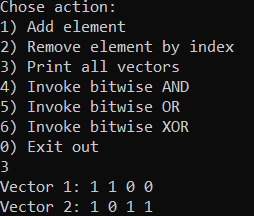


Рисунок 5.3 – Вивід елементів векторів у консоль

Спроба викликати логічну операцію коли кількість елементів не однакова (5.4):

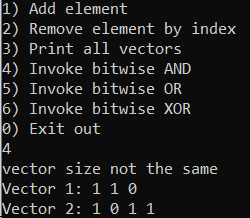


Рисунок 5.4 – Логічна операція з помилкою

Спроба викликати логічну операцію І коли все гаразд (5.5):

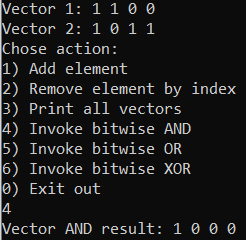


Рисунок 5.5 – Логічна операція І

Спроба викликати логічну операцію АБО коли все гаразд (5.6):

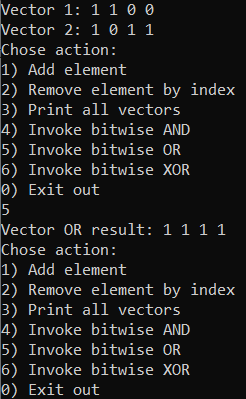


Рисунок 5.6 – Логічна операція АБО

Спроба викликати логічну операцію НЕ АБО коли все гаразд (5.7):

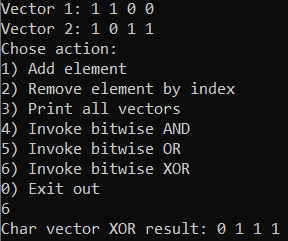


Рисунок 5.7 – Логічна операція НЕ АБО

Спроба викликати логічну операцію НЕ АБО для CHAR коли все гаразд (5.8):

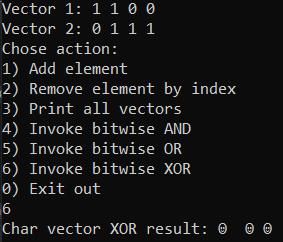


Рисунок 5.7 – Логічна операція НЕ АБО для CHAR

6. Аналіз помилок (опис усунення зауважень)

…