ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ

за курсом «Програмування»

студента групи ПЗ-24у-1

Кондрачука Олександра Вадимовича

кафедра математичного забезпечення ЕОМ, ДНУ

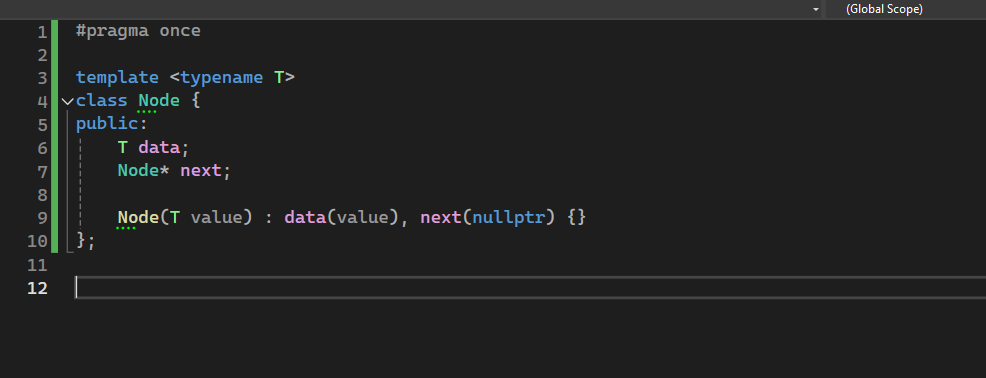
2024/2025 навч.р.

1. **Постановка завдання**

Варіант 8: Визначте чергу використовуючи (тільки) два стека.

1. **Рішення**

Було створено клас Node, який є класом для передачі даних. Він зберігаю у собі значення ноди стеку та посилання на наступну ноду.



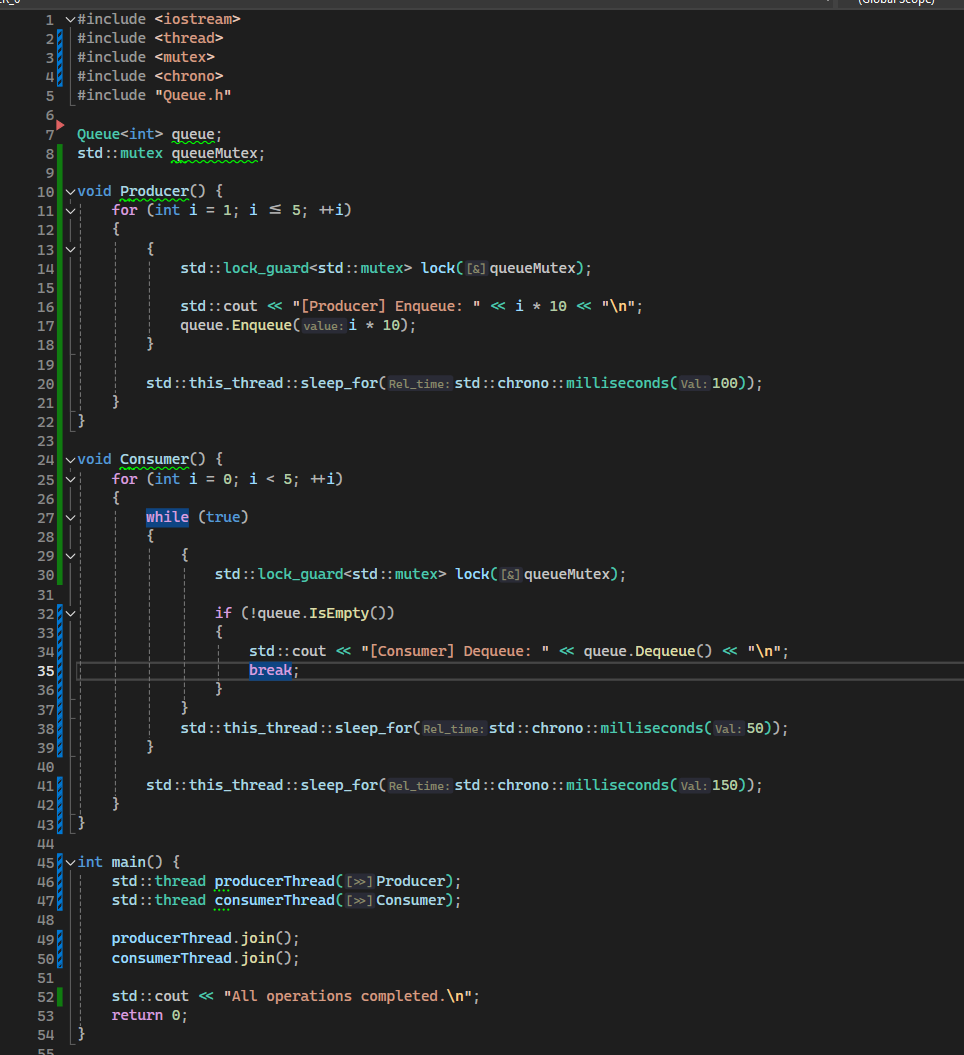
Також було клас Stack з методами: Push (для додавання даних), Pop (для взяття останньої доданої ноди та видалення її зі стеку), Peek (для взяття останньої доданої ноди без видалення), IsEmpty (для перевірки, чи стек пустий).



Та було створено клас Queue, який представляє чергу, що працює через 2 стеки. У ній є методи: Enqueue (для додавання у чергу), Dequeue (для взяття та видалення першого елементу черги), Peek (для взяття першого елементу черги без видалення), IsEmpty (для перевірки, чи черга не пуста). Для всіх цих операції використовуються стеки. У першому зберігаються додані елементи через метод Enqueue, у другому – елементи, що будуть використані через методи Dequeue та Peek.



У функції main() створюється черга типу int, потім до неї додаються та видаляються елементи через методи Producer та Consumer використовуючи std::thread. Метод Producer додає елементи у чергу, а метод Consumer – видаляє.



1. **Лістинги:**

#pragma once

template <typename T>

class Node {

public:

T data;

Node\* next;

Node(T value) : data(value), next(nullptr) {}

};

#pragma once

#include <cstddef>

#include "Node.h"

template <typename T>

class Stack {

private:

Node<T>\* top;

public:

Stack() : top(nullptr) {}

void Push(T value) {

Node<T>\* node = new Node<T>(value);

node->next = top;

top = node;

}

T Pop() {

if (IsEmpty())

{

return NULL;

}

Node<T>\* temp = top;

T value = temp->data;

top = top->next;

delete temp;

return value;

}

T Peek() const {

if (IsEmpty())

{

return NULL;

}

return top->data;

}

bool IsEmpty() const {

return top == nullptr;

}

~Stack() {

while (!IsEmpty()) Pop();

}

};

#pragma once

#include "Stack.h"

template <typename T>

class Queue {

private:

Stack<T> inStack;

Stack<T> outStack;

void TransferInToOut() {

while (!inStack.IsEmpty()) {

outStack.Push(inStack.Pop());

}

}

public:

void Enqueue(T value) {

inStack.Push(value);

}

T Dequeue() {

if (IsEmpty())

{

return NULL;

}

if (outStack.IsEmpty())

{

TransferInToOut();

}

return outStack.Pop();

}

T Peek() {

if (IsEmpty())

{

return NULL;

}

if (outStack.IsEmpty())

{

TransferInToOut();

}

return outStack.Peek();

}

bool IsEmpty() {

return inStack.IsEmpty() && outStack.IsEmpty();

}

};

#include <iostream>S

#include <thread>

#include <mutex>

#include <chrono>

#include "Queue.h"

Queue<int> queue;

std::mutex queueMutex;

void Producer() {

for (int i = 1; i <= 5; ++i)

{

{

std::lock\_guard<std::mutex> lock(queueMutex);

std::cout << "[Producer] Enqueue: " << i \* 10 << "\n";

queue.Enqueue(i \* 10);

}

std::this\_thread::sleep\_for(std::chrono::milliseconds(100));

}

}

void Consumer() {

for (int i = 0; i < 5; ++i)

{

while (true)

{

{

std::lock\_guard<std::mutex> lock(queueMutex);

if (!queue.IsEmpty())

{

std::cout << "[Consumer] Dequeue: " << queue.Dequeue() << "\n";

break;

}

}

std::this\_thread::sleep\_for(std::chrono::milliseconds(50));

}

std::this\_thread::sleep\_for(std::chrono::milliseconds(150));

}

}

int main() {

std::thread producerThread(Producer);

std::thread consumerThread(Consumer);

producerThread.join();

consumerThread.join();

std::cout << "All operations completed.\n";

return 0;

}