ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»  
(РУТ (МИИТ))

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

ОТЧЁТ  
О ЛАБОРАТОРНОЙ РАБОТЕ №2

По дисциплине «Языки программирования»

ВАРИАНТ 7

Выполнил: ст. гр. ТКИ - 141

Пылаева Светлана Алексеевна

Проверил: к.т.н., доц. Васильева М. А.

(Проверил: к.т.н, доц. Балакина Е. П.)

Москва 2024

1. Формулировка задания

Создать решение (Solution), которое минимально содержит три проекта (Projects): исполняемый (Console Application), библиотеку классов (Library), и модульные тесты (Tests). Разработать библиотеку классов по заданному варианту. Важно! Библиотека классов не должна зависеть от потоков ввода-вывода. Каждый класс необходимо размещать в отдельных двух файлах, снабжённых «говорящим именем» и специальными расширениями: \*.h для заголовочных файлов (Header), содержащих API класса, и \*.cpp для компилируемых (Source), содержащих реализацию класса. В запускаемом проекте требуется создать файл main.cpp, содержащий точку входа в демонстрационную программу – главную функцию (main). В рамках данной функции показать работу с коллекцией.

Реализовать библиотеку заданной (по вариантам) структуры данных целых чисел. Предусмотреть конструктор со списком инициализации (<https://en.cppreference.com/w/cpp/utility/initializer_list>). Реализовать все конструкторы, создаваемые компилятором по умолчанию, реализовать деструктор. Предусмотреть методы вывода в строку содержимого коллекции. Переопределить операторы присваивания, сдвига влево и сдвига вправо. Предусмотреть метод определения наличия элементов коллекции (пустая ли коллекция?).

Очередь (Queue).

Коллекция, реализующая вставку и удаление элемента. Возможны

операции добавление элемента в конец очереди (enqueue), извлечение

элемента из начала очереди (dequeue) и чтение головного элемента (peek).

1. Диаграмма UML

Диаграмма UML для классов Queue и Node представлена ниже

(Рисунок 1).

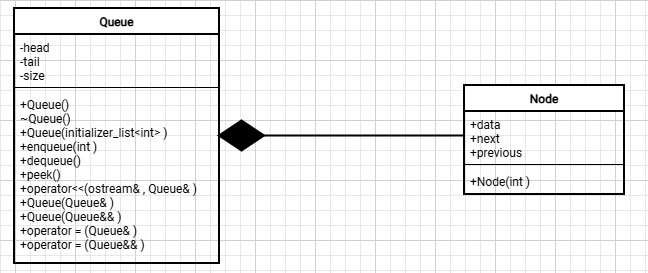


Рисунок 1 – Диаграмма UML

1. Текст программы на языке C++

main.cpp:

#include <iostream>

#include "..\Solver\Queue.h"

int main()

{

Queue::Queue list = { 1, 2, 3, 4, 5};

std::cout << list << std::endl;

list.enqueue(6);

std::cout << list << std::endl;

list.dequeue();

std::cout << list << std::endl;

std::cout << list.peek() << std::endl;

Queue::Queue list1 = { 1, 2, 3, 4, 5 };

Queue::Queue list2 = { 1, 2, 3, 4, 5 };

Queue::Queue queue1(std::move(list1));

Queue::Queue queue2 = std::move(list2);

std::cout << queue1 << std::endl << queue2 << std::endl;

std::cout << list1 << std::endl << list2 << std::endl;

return 0;

}

Node.h:

#pragma once

struct Node

{

int data;

Node\* next;

Node\* previous;

Node(int value);

};

Node.cpp:

#include "Node.h"

Node::Node(int value) :data{ value }, next{ nullptr }, previous{ nullptr } {}

Queue.h:

#pragma once

#include <iostream>

#include <initializer\_list>

#include "Node.h"

namespace Queue

{

class Queue

{

private:

/\*\*

\* @brief - Узел головы очереди.

\*/

Node\* head;

/\*\*

\* @brief - Узел хвоста очереди.

\*/

Node\* tail;

/\*\*

\* @brief - Размер очереди.

\*/

size\_t size;

public:

/\*\*

\* @brief - Конструктор класса очередь.

\*/

Queue();

/\*\*

\* @brief - Деструктор класса очередь.

\*/

~Queue();

/\*\*

\* @brief - Конструктор со списком инициализации.

\* @param list - Список

\*/

Queue(std::initializer\_list<int> list);

/\*\*

\* @brief - Функция,добавляющая элемент в конце очереди.

\* @param value - Значение нового элемента.

\*/

void enqueue(const int value);

/\*\*

\* @brief - Функция,извлекающая элемент из начала очереди.

\*/

void dequeue();

/\*\*

\* @brief - Функция,читающая головной элемент.

\* @return - Возвращает значение первого узла.

\*/

int peek();

/\*\*

\* @brief - Оператор сдвига влево.

\*/

friend std::ostream& operator << (std::ostream& output, const Queue& queue);

/\*\*

\* @brief - Конструктор копирования.

\* @param other - Другая очередь.

\*/

Queue(const Queue& other);

/\*\*

\* @brief - Конструктор перемещения.

\* @param other - Другая очередь.

\*/

Queue(Queue&& other) noexcept;

/\*\*

\* @brief - Оператор присваивания с помощью копирования.

\* @param other - Другая очередь.

\* @return - Возвращает очередь с другими параметрами.

\*/

Queue& operator = (const Queue& other);

/\*\*

\* @brief - Оператор присваивания с помощью перемещения.

\* @param other - Другая очередь.

\* @return - Возвращает очередь с другими параметрами.

\*/

Queue& operator = (Queue&& other) noexcept;

};

}

Queue.cpp:

#include <iostream>

#include "Queue.h"

Queue::Queue::Queue() :head{ nullptr }, tail{ nullptr }, size{ 0 } {}

Queue::Queue::~Queue()

{

while (head != nullptr)

{

Node\* temporary = head;

head = head->next;

delete temporary;

}

}

Queue::Queue::Queue(std::initializer\_list<int> list)

{

for (int value : list)

{

enqueue(value);

}

}

void Queue::Queue::enqueue(const int value)

{

Node\* lastNode = new Node(value);

if (size == 0)

{

head = tail = lastNode;

}

else

{

tail->next = lastNode;

lastNode->previous = tail;

tail = lastNode;

}

++size;

}

void Queue::Queue::dequeue()

{

if (size == 0)

{

throw std::out\_of\_range("");

}

if (tail == head)

{

tail = head = nullptr;

}

else

{

head = head->next;

head->previous = nullptr;

}

--size;

}

int Queue::Queue::peek()

{

if (size == 0)

{

throw std::out\_of\_range("");

}

return head->data;

}

std::ostream& Queue::operator<<(std::ostream& output, const Queue& queue)

{

Queue temporary = queue;

while (!(temporary.size == 0))

{

output << temporary.peek() << " ";

temporary.dequeue();

}

return output;

}

Queue::Queue::Queue(const Queue& other)

{

Node\* temporary = other.head;

while (temporary != nullptr)

{

enqueue(temporary->data);

temporary = temporary->next;

}

}

Queue::Queue::Queue(Queue&& other) noexcept

{

head = other.head;

tail = other.tail;

size = other.size;

other.head = other.tail = nullptr;

other.size = 0;

}

Queue::Queue& Queue::Queue::operator=(const Queue& other)

{

Queue temporary(other);

std::swap(this->head, temporary.head);

std::swap(this->tail, temporary.tail);

std::swap(this->size, temporary.size);

return \*this;

}

Queue::Queue& Queue::Queue::operator=(Queue&& other) noexcept

{

if (this != &other)

{

std::swap(other.head, this->head);

std::swap(other.tail, this->tail);

std::exchange(other.size, this->size);

}

return \*this;

}

1. Тесты классов

Tests.cpp:

#include "pch.h"

#include "CppUnitTest.h"

#include "..\Solver\Queue.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace Tests

{

TEST\_CLASS(QueueTest)

{

public:

TEST\_METHOD(Enqueue\_ValidData\_Success)

{

Queue::Queue queue{};

queue.enqueue(5);

Assert::AreEqual(queue.peek(), 5);

}

TEST\_METHOD(Enqueue\_ValidData\_Fail)

{

Queue::Queue queue{};

queue.enqueue(5);

Assert::AreNotEqual(queue.peek(), 3);

}

TEST\_METHOD(Dequeue\_ValidData\_Success)

{

Queue::Queue queue{1,2,3,4,5};

queue.dequeue();

Assert::AreEqual(queue.peek(), 2);

}

TEST\_METHOD(Dequeue\_ValidData\_Fail)

{

Queue::Queue queue{ 1,2,3,4,5 };

queue.dequeue();

Assert::AreNotEqual(queue.peek(), 3);

}

TEST\_METHOD(СopyСonstructor\_ValidData\_Success)

{

Queue::Queue queue1{ 1,2,3 }, queue2(queue1);

Assert::AreEqual(queue1.peek(), queue2.peek());

queue1.dequeue();

queue2.dequeue();

Assert::AreEqual(queue1.peek(), queue2.peek());

queue1.dequeue();

queue2.dequeue();

Assert::AreEqual(queue1.peek(), queue2.peek());

}

TEST\_METHOD(MoveConstructor\_ValidData\_Success)

{

Queue::Queue queue1{ 1,2,3 }, queue2 = queue1, queue3(std::move(queue2));

Assert::AreEqual(queue1.peek(), queue3.peek());

queue1.dequeue();

queue3.dequeue();

Assert::AreEqual(queue1.peek(), queue3.peek());

queue1.dequeue();

queue3.dequeue();

Assert::AreEqual(queue1.peek(), queue3.peek());

}

TEST\_METHOD(OperatorEqualCoping\_ValidData\_Success)

{

Queue::Queue queue1{ 1,2,3 }, queue2=queue1;

Assert::AreEqual(queue1.peek(), queue2.peek());

queue1.dequeue();

queue2.dequeue();

Assert::AreEqual(queue1.peek(), queue2.peek());

queue1.dequeue();

queue2.dequeue();

Assert::AreEqual(queue1.peek(), queue2.peek());

}

TEST\_METHOD(OperatorEqualMoving\_ValidData\_Success)

{

Queue::Queue queue1{ 1,2,3 }, queue2=queue1, queue3=std::move(queue2);

Assert::AreEqual(queue1.peek(), queue3.peek());

queue1.dequeue();

queue3.dequeue();

Assert::AreEqual(queue1.peek(), queue3.peek());

queue1.dequeue();

queue3.dequeue();

Assert::AreEqual(queue1.peek(), queue3.peek());

}

};

}

Результаты выполнения тестов классов представлены ниже

(Рисунок 2).

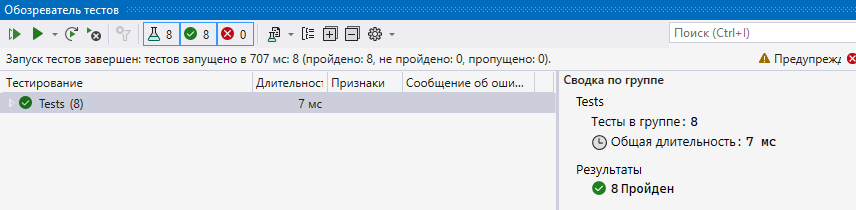


Рисунок 2 – Результат выполнения тестов

1. Отметка о выполнении задания в веб-хостинге системы контроля версий

Ниже представлено доказательство того, что задание было принято (Рисунок 3).

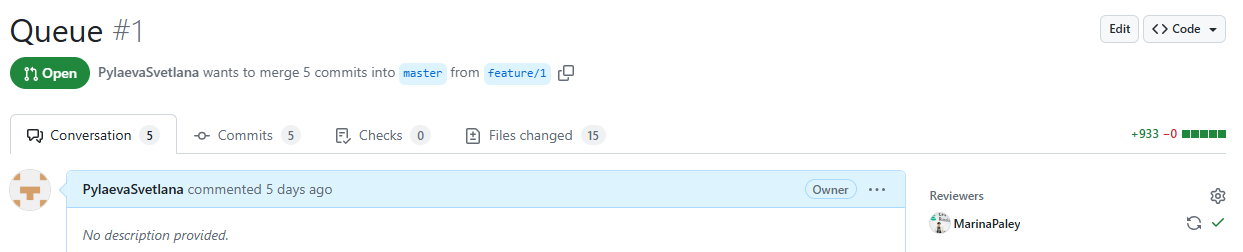


Рисунок 3 – Approve