Министерство цифрового развития, связи и массовых коммуникаций

Российской Федерации

Ордена Трудового Красного Знамени

федеральное государственное бюджетное образовательное учреждение

высшего образования

Московский технический университет связи и информатики

(МТУСИ)

Кафедра «Информационная безопасность»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №11

по дисциплине

«Программирование в системах информационной безопасности»

на тему

«Шаблонные функции и классы»

Вариант №25

Выполнил:

студент группы БСУ1801

Убушуев Б.С

Проверил:

старший преподаватель кафедры ИБ

Барков В. В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва 2021

# **Цель**

Изучить механизм шаблонов в языке C++.

# **Задание**

Язык C++ поддерживает обобщённое программирование с помощью механизма шаблонов. Шаблонами могут быть функции и классы. В шаблонную функцию или класс вводится один или несколько параметров, которые разрешаются во время компиляции. Такими параметрами могут быть как целые числа, так и имена типов. Создание класса или функции из шаблона называется инстанцированием шаблона. Компилятор генерирует определение функции или класса, подставляя все необходимые параметры. Если такое определение ошибочно, компилятор выдаст ошибку. Применительно к контейнерам, параметром шаблона является тип данных элемента.

Преобразовать разработанный в практикуме №9 класс в шаблонный класс. Параметром шаблона является тип элементов в динамической структуре данных.

# **Индивидуальное задание**

|  |  |  |
| --- | --- | --- |
| Вариант №19 | Очередь на основе двунаправленного циклического списка | QueueBasedOnBidirectionalCyclicLinkedList |

# **4. Выполнение**

Выполнение задания:

1. **Класс с шаблонными значениями элементов**

Листинг 1 – Исходный код заголовочного файла, который содержит описание класса

#include <iostream>

using namespace std;

template <typename T>

struct Element

{

T number;

Element\* next;

Element\* prev;

};

template <typename T>

class Queue

{

Element<T>\* head;

Element<T>\* tail;

public:

Queue();

Queue(const Queue& other);

Queue(Queue&& other)noexcept;

~Queue();

void push(T element);

T pop();

T peek();

int get\_size();

Queue& operator=(const Queue& other);

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

friend ostream &operator<<(ostream& stream, const Queue& that)

{

if (that.head == NULL) return stream << "Очередь пуста";

Element<T>\* temp = that.head;

while (temp != that.tail)

{

stream << temp->number << " ";

temp = temp->prev;

}

return stream << temp->number;

}

};

Листинг 2 – Исходный код основного файла, который содержит описание методов класса и функцию main

#include "Header2.h"

template <typename T>

Queue<T>::Queue()

{

head = tail = NULL;

}

template <typename T>

Queue<T>::Queue(const Queue& other)

{

if (other.head == NULL)

{

head = tail = NULL;

return;

}

Element<T>\* temp = other.head;

while (temp != other.tail)

{

this->push(temp->number);

temp = temp->prev;

}

this->push(temp->number);

}

template <typename T>

Queue<T>::Queue(Queue&& other)noexcept

{

if (other.head == NULL)

{

head = tail = NULL;

return;

}

head = other.head;

tail = other.tail;

other.head = other.tail = NULL;

}

template <typename T>

Queue<T>::~Queue()

{

if (head == NULL) return;

while (this->get\_size() > 0)

{

this->pop();

}

head = tail = NULL;

}

template <typename T>

void Queue<T>::push(T element)

{

Element<T>\* temp = new Element<int>;

temp->number = element;

if (head != NULL)

{

temp->next = tail;

temp->prev = head;

tail->prev = temp;

head->next = temp;

tail = temp;

}

else temp->next = temp->prev = head = tail = temp;

}

template <typename T>

T Queue<T>::pop()

{

if (head == tail)

{

if (head == NULL) return 0;

int res = head->number;

delete head;

head = tail = NULL;

return res;

}

int res = head->number;

head = head->prev;

delete head->next;

head->next = tail;

tail->prev = head;

return res;

}

template <typename T>

T Queue<T>::peek()

{

if (head == NULL) return 0;

return head->number;

}

template <typename T>

int Queue<T>::get\_size()

{

int i = 0;

Element<T>\* temp = tail;

if (head == NULL) return 0;

while (temp != head)

{

i++;

temp = temp->next;

}

return i + 1;

}

template <typename T>

Queue<T>& Queue<T>::operator=(const Queue& other)

{

if (this == &other || other.head == NULL) return \*this;

Element<T>\* temp = other.head;

while (temp != other.tail)

{

this->push(temp->number);

temp = temp->prev;

}

this->push(temp->number);

return \*this;

}

template <typename T>

Queue<T>& Queue<T>::operator=(Queue&& other)noexcept

{

if (this == &other || other.head == NULL) return \*this;

head = other.head;

tail = other.tail;

other.head = other.tail = NULL;

return \*this;

}

int main()

{

setlocale(LC\_ALL, "ru");

cout << "Выберите действие с очередью:" << endl;

cout << "0. Выйти из программы" << endl;

cout << "1. Вставить элемент в конец" << endl;

cout << "2. Удалить и вывести головной элемент" << endl;

cout << "3. Вывести головной элемент" << endl;

cout << "4. Вывести длину очереди" << endl;

cout << "5. Вывести все элементы очереди" << endl;

cout << "6. Создать новую очередь с копированием и вывести ее элементы" << endl;

cout << "7. Создать новую очередь с перемещением и вывести ее элементы" << endl;

cout << "8. Копировать 1 очередь во 2 и вывести ее элементы" << endl;

cout << "9. Копировать 1 очередь во 2 с перемещением и вывести ее элементы" << endl;

Queue<int> queue;

int opt;

cin >> opt;

while (opt != 0)

{

switch (opt)

{

case 1:

{

int a;

cout << "Введите число для вставки: ";

cin >> a;

queue.push(a);

break;

}

case 2:

{

cout << "Значение удаленного головного элемента: " << queue.pop() << endl;

break;

}

case 3:

{

cout << "Значение головного элемента: " << queue.peek() << endl;

break;

}

case 4:

cout << "Длина очереди: " << queue.get\_size() << endl;

break;

case 5:

cout << queue << endl;

break;

case 6:

{

Queue<int> queue2 = queue;

cout << queue2 << endl;

break;

}

case 7:

{

Queue<int> queue3 = move(queue);

cout << queue3 << endl;

break;

}

case 8:

{

Queue<int> queue2;

queue2 = queue;

cout << queue2 << endl;

break;

}

case 9:

{

Queue<int> queue3;

queue3 = move(queue);

cout << queue3 << endl;

break;

}

default:

cout << "Введено недопустимое значение" << endl;

}

cin >> opt;

}

return 0;

}

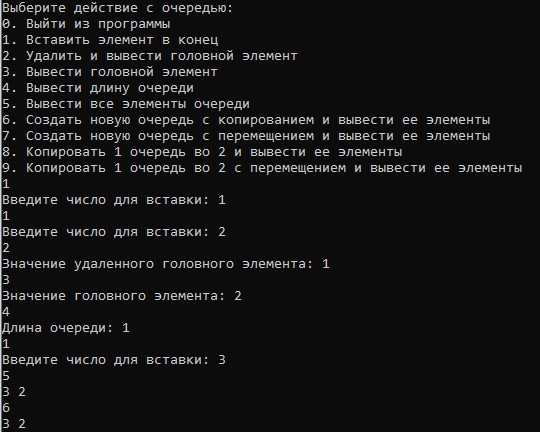


Рисунок 1 – Результат работы программы

# **Вывод**

В этой лабораторной работе можно научиться создавать шаблоны классов и работы с ними, используя динамическую структуру данных.