МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра вычислительной техники

Отчет по лабораторной работе №7
по дисциплине: «Программирование»

Факультет: АВТФ

Группа: АВТ-143

Студент: Васютин А. М.

Преподаватель: Новицкая Ю. В.

Вариант: 4

Новосибирск, 2022 г.

ОГЛАВЛЕНИЕ

1. ЛАБОРАТОРНАЯ РАБОТА №7. УНИВЕРСАЛЬНОСТЬ. ПРИМЕНЕНИ ШАБЛОНОВ ФУНКЦИЙ И КЛАССОВ

1.1. Цель работы

Ознакомиться с созданием шаблонов функций и классов. Изучить написание контейнерных шаблонных классов и их применение для построения различных структур данных.

1.2. Задание на лабораторную работу

Задание представляет собой типовую задачу по разработке шаблонов стандартных структур данных. Протестировать структуру данных. В качестве хранимых объектов использовать встроенные типы C++ (int и char).

– Структура данных: циклическая очередь (динамический массив).
– Способ хранения объектов: ссылки на объекты.
– Размерность структуры данных: параметр шаблона.
– Операция: включение в конец очереди.
– Операция: исключение из очереди.

1.3. Решение

Шаблоны классов позволяют создавать параметризованные классы.Параметриз класс создает семейство родственных классов, которые можно применять к любому типу данных, передаваемому в качестве параметра. Наиболее широкое применение шаблоны находят при создании контейнерных классов. Контейнерным называется класс, который предназначен для хранения каким-либо образом организованных данных и работы с ними.

Преимущество использования шаблонов состоит в том, что как только алгоритм работы с данными определен и отлажен, он может применяться к любым типам данных без переписывания кода.

В лабораторной работе требуется реализовать шаблонный класс циклической очереди на динамическом массиве. Так как язык не предполагает создание массива ссылок, так как ссылка на объект должна быть обязательно инициализированная (ссылаться на объект), а массив создается неинициализированным, следовательно, элементы хранятся по значению.

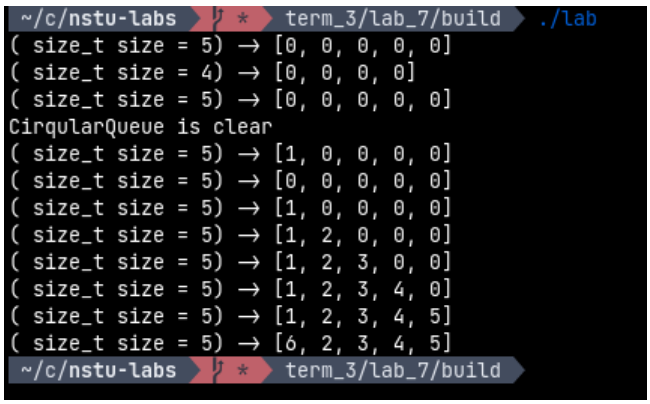

Рисунок 1 – Вывод работы программы.

1.4. Исходный код

Листинг 1 – main.cpp

```cpp
#include "CircularQueue.hpp"

int main() {
  CircularQueue<int> cqueue(5);
  CircularQueue<double, 4> cq1;
  CircularQueue<float, 4> cq2(5);

  cqueue.print();
  cq1.print();
  cq2.print();

  try {
    cqueue.remove();
  } catch (CircularQueue<int>::CircularQueueIsClear &e) {
    std::cout << e.what() << std::endl;
  }

  cqueue.add(1);
  cqueue.print();
  int a = cqueue.remove();
  cqueue.print();
  cqueue.add(1);
  cqueue.print();
  cqueue.add(2);
  cqueue.print();
  cqueue.add(3);
  cqueue.print();
  cqueue.add(4);
  cqueue.print();
  cqueue.add(5);
  cqueue.print();
  cqueue.add(6);
  cqueue.print();

  return 0;
```

```
}
```

Листинг 2 – matrix.hpp

```
#include <cstddef>
#include <iostream>
#include <ostream>

template <typename T, size_t static_initial_size = 0> class CircularQueue {
private:
  size_t _size;
  ptrdiff_t _last;
  ptrdiff_t _first;
  T *_data;

public:
  // CircularQueue() : CircularQueue<T,
  // static_initial_size>(static_initial_size) {}
  CircularQueue(size_t size = -1) : _last(-1), _first(-1) {
    if (size == -1)
      _size = static_initial_size;
    else
      _size = size;

    if (_size == 0) {
      _data = nullptr;
    } else {
      _data = new T[_size];
    }
  }
  ~CircularQueue() {
    if (_size != 0) {
      delete[] _data;
    }
  }

  T remove() {
    if (_first == -1)
      throw CircularQueueIsClear();
    T result = _data[_first];
    if (_first == _last) {
      _data[_last] = 0;
      _first = -1;
      _last = -1;
    }
    return result;
  }
  void add(T element) {
    _last = (_last + 1) % _size;
    if (_first == -1)
      _first = _last;
    _data[_last] = element;
```

```
  }
  void print() const {
    std::cout << "( size_t size = " << _size << ") -> ";
    std::cout << "[";
    for (size_t i = 0; i < _size; ++i) {
      std::cout << _data[i];
      if (i + 1 < _size) {
        std::cout << ", ";
      }
    }
    std::cout << "]\n";
  }
  class CircularQueueIsClear : public std::exception {
  public:
    const char *what() const noexcept { return "CirqularQueue is clear"; }
  };
};
```

1.5. Выводы

В данной лабораторной работе были изучены возможности использования языка C++ в обобщённом программировании при помощи механизма шаблонов. Был разработан динамический шаблонный контейнер (Циклическая очередь на динамическом массиве), протестированный на базовых типах.