МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

ЭЛЕКТРОТЕХНИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА «ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И АВТОМАТИЗИРОВАННЫХ СИСТЕМ»

# ОТЧЁТ
# «ЛАБОРАТОРНАЯ №11.1: ОЧЕРЕДЬ»

Дисциплина: «Программирование»

Выполнил:
Студент группы ИВТ-21-2б
Безух Владимир Сергеевич

Проверил:
Доцент кафедры ИТАС
Полякова Ольга Андреевна

Пермь, 2022

# Содержание

# 1. Постановка задачи

Продемонстрировать работу очереди как структуры данных.

## 2. Анализ задачи

Очередь можно реализовать на основе двусвязного списка с ограниченным функционалом. Необходимо реализовать методы добавления узла в конец очереди и удаления узла из начала очереди для организации принципа FIFO (first in, first out).

# 3. Исходный код

```
#include<string>
#include<iostream>

template <typename T>
class Queue {
public:
    Queue();
    ~Queue();

    size_t size() const;

    T back() const;
    void push(const T& data);

    T front() const;
    void pop();

    void clear();

private:
    struct Node {
            Node(T data = T(), Node* pointer_to_prev_node = nullptr, Node*
pointer_to_next_node = nullptr)
                    : data(data), pointer_to_prev_node(pointer_to_prev_node),
pointer_to_next_node(pointer_to_next_node) {}

            Node(const Node& copy)
                    : data(copy.data), pointer_to_prev_node(copy.pointer_to_prev_node),
pointer_to_next_node(copy.pointer_to_next_node) {}

            Node& operator=(const Node& right) {
                    if (this != &right) {
                            data = right.data;
                            pointer_to_prev_node = right.pointer_to_prev_node;
                            pointer_to_next_node = right.pointer_to_next_node;
                    }

                    return *this;
            }

            T data;
            Node* pointer_to_prev_node;
            Node* pointer_to_next_node;
    };

    void pushFirstNode(Node* node);
    void pushBackNode(Node* node);

    void popFirstNode();
    void popFrontNode();

    size_t queue_size;
    Node* head_node;
    Node* tail_node;
};

template<typename T>
Queue<T>::Queue()
    : queue_size(size_t{0}), head_node(nullptr), tail_node(nullptr) {}
```

```cpp
template<typename T>
Queue<T>::~Queue()
{
    clear();
}

template<typename T>
size_t Queue<T>::size() const
{
    return queue_size;
}

template<typename T>
T Queue<T>::back() const
{
    return tail_node->data;
}

template<typename T>
void Queue<T>::push(const T& data)
{
    Node* new_node = new Node(data);
    queue_size ? pushBackNode(new_node) : pushFirstNode(new_node);
    ++queue_size;
}

template<typename T>
T Queue<T>::front() const
{
    return head_node->data;
}

template<typename T>
void Queue<T>::pop()
{
    if (queue_size == size_t{0}) return;

    Node* remove_node = head_node;
    (queue_size == size_t{1}) ? popFirstNode() : popFrontNode();
    delete remove_node;
    --queue_size;
}

template<typename T>
void Queue<T>::clear()
{
    if (queue_size == size_t{0}) return;

    Node* remove;
    Node* next_node = head_node;

    while (queue_size) {
           remove = next_node;
           next_node = next_node->pointer_to_next_node;
           delete remove;
           --queue_size;
    }

    head_node = nullptr;
    tail_node = nullptr;
}
```

```
template<typename T>
void Queue<T>::pushFirstNode(Node* node)
{
    head_node = node;
    tail_node = node;
}

template<typename T>
void Queue<T>::pushBackNode(Node* node)
{
    tail_node->pointer_to_next_node = node;
    node->pointer_to_prev_node = tail_node;
    tail_node = node;
}

template<typename T>
void Queue<T>::popFirstNode()
{
    head_node = nullptr;
    tail_node = nullptr;
}

template<typename T>
void Queue<T>::popFrontNode()
{
    head_node->pointer_to_next_node->pointer_to_prev_node = nullptr;
    head_node = head_node->pointer_to_next_node;
}

template <typename T>
void print(const Queue<T>& queue)
{
    std::cout << queue.front() << " <- начальный элемент | конечный элемент -> " <<
queue.back() << '\n';
}

int main()
{
    Queue<std::string> queue;
    std::setlocale(LC_ALL, "Russian");

    queue.push("str1"); print(queue);
    queue.push("str3");  print(queue);
    queue.push("str5");  print(queue);
    queue.push("str7");  print(queue);

    queue.pop(); print(queue);
    queue.pop(); print(queue);
    queue.pop(); print(queue);
    queue.pop();

}
```

# 4. Анализ результатов

Результаты работы программы (рис. 1).

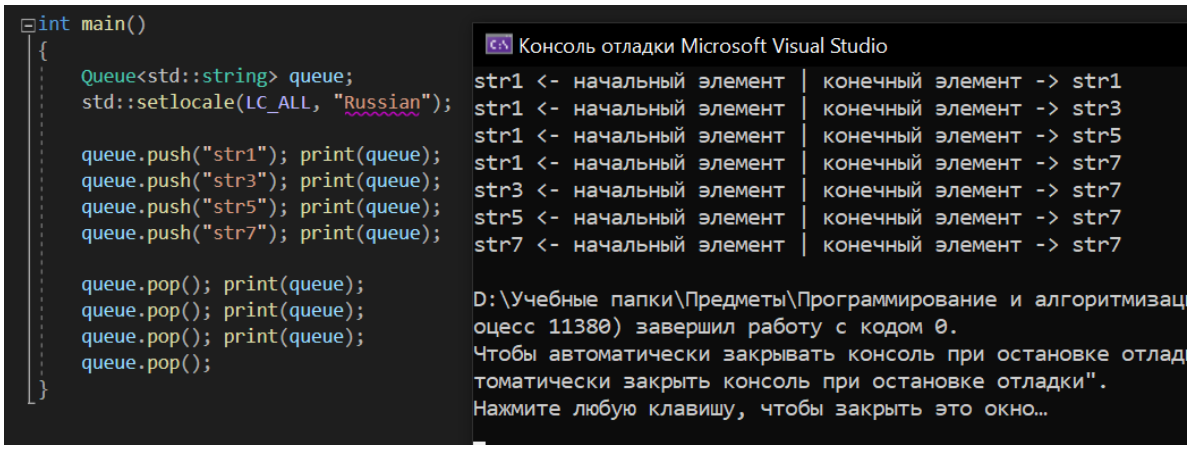


*Рисунок 1 — Результаты*