Министерство образования и науки Российской Федерации

ФГБОУ ВО «Пензенский государственный университет»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

Пенза 2025

**Отчет**

по лабораторной работе №3

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Динамические списки.»

Выполнил: студент группы 24ВВВ1

Марушкин Д.И.

Егорова К.А.

Винникова Е.С

Проверил: к.т.н., доцент

Юрова О.В.

Деев М.В.

## **Лабораторные задания:**

1. Реализовать приоритетную очередь: элементы вставляются в список в зависимости от приоритета (больший приоритет — выше в списке).
2. На основе приведённого кода реализовать структуру данных **Очередь**.
3. На основе приведённого кода реализовать структуру данных **Стек**.

## **Пояснительный текст к программе**

### **1. Назначение программы**

Цель программы — изучение динамических структур данных и исследование работы операций вставки, удаления и доступа к элементам.  
Программа позволяет:

* Создавать приоритетный список с автоматическим упорядочением элементов по приоритету.
* Реализовать классическую очередь (FIFO) и стек (LIFO) на базе связного списка.
* Демонстрировать работу всех структур через консольное меню с добавлением, удалением и выводом элементов.

### **2. Структура программы**

* **main.cpp** – основной модуль программы, отображает меню и управляет вводом пользователя.
* **List** – шаблонный односвязный список, реализующий методы:
  + push\_back(data, priority) — добавление элемента в конец списка;
  + push\_front(data, priority) — добавление элемента в начало списка;
  + pop\_back() — удаление последнего элемента;
  + pop\_front() — удаление первого элемента;
  + removeAt(index) — удаление элемента по индексу;
  + clear() — полная очистка списка;
  + push\_priority(data, priority) — вставка элемента в список с учётом приоритета;
  + оператор [] для доступа к элементу по индексу;
  + метод isEmpty() для проверки пустоты списка;
  + метод getSize() для получения текущего размера списка.
* **Queue** – очередь на основе List с методами:
  + enqueue(data) — добавить элемент в конец;
  + dequeue() — удалить элемент из начала;
  + front() — получить первый элемент;
  + print() — вывести все элементы;
  + empty() — проверить, пуста ли очередь;
  + size() — получить размер очереди.
* **Stack** – стек на основе List с методами:
  + push(data) — добавить элемент на вершину;
  + pop() — удалить верхний элемент;
  + top() — получить верхний элемент;
  + print() — вывести все элементы;
  + empty() — проверить, пуст ли стек;
  + size() — получить размер стека.

### **3. Используемые средства**

* Язык программирования **C++17**.
* Стандартная библиотека **<iostream>** для ввода/вывода.
* Шаблонные классы позволяют работать с любыми типами данных.

### **4. Описание работы программы**

#### **4.1 Приоритетный список**

Метод **push\_priority** вставляет элемент в список с учётом приоритета:

* Если список пуст или приоритет нового элемента выше, чем у головы списка, элемент становится первым.
* Иначе происходит последовательный обход списка до нахождения позиции, где следующий элемент имеет меньший приоритет, и вставка выполняется перед ним.

**Дополнительно реализованы:**

* Добавление в начало (push\_front) и конец (push\_back) списка.
* Удаление с начала (pop\_front) и конца (pop\_back) списка.
* Удаление по индексу (removeAt) с проверкой корректности индекса.
* Полная очистка списка (clear).

#### **4.2 Очередь**

Очередь реализована на основе списка и поддерживает:

* enqueue — вставку в конец списка;
* dequeue — удаление с начала списка;
* front — доступ к первому элементу;
* print — вывод всех элементов.

Очередь работает по принципу **FIFO** (первый пришёл — первый ушёл).

#### **4.3 Стек**

Стек реализован на основе списка и поддерживает:

* push — добавление элемента на вершину;
* pop — удаление верхнего элемента;
* top — получение верхнего элемента;
* print — вывод всех элементов.

Стек работает по принципу **LIFO** (последний пришёл — первый ушёл).

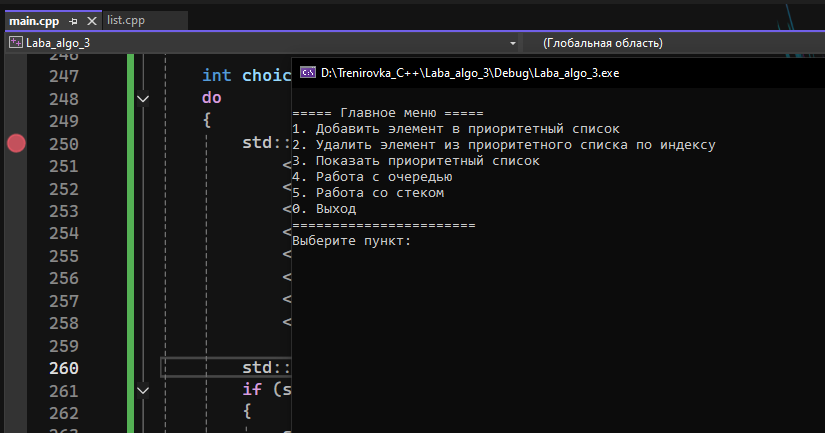
### **5. Консольное меню программы**

Программа имеет интуитивное меню, разделённое на следующие части:

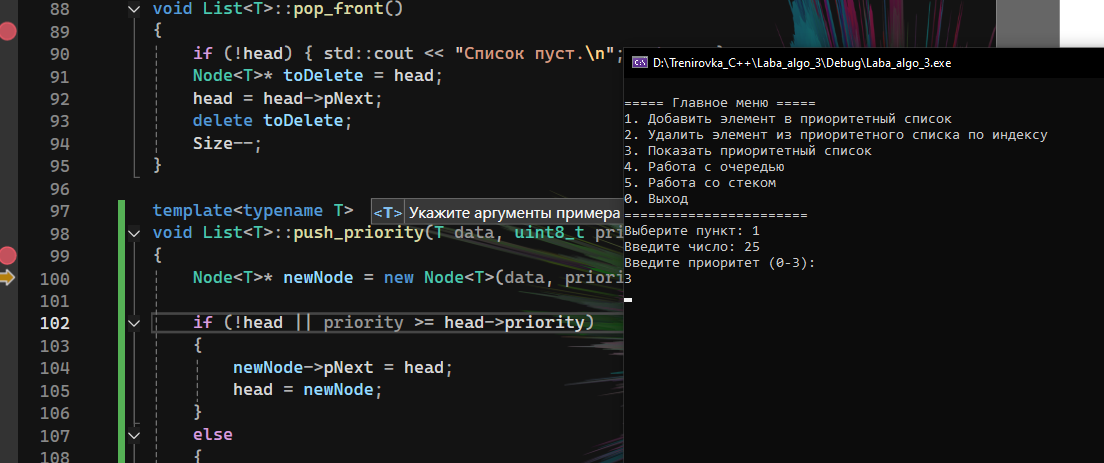
1. Добавление элемента в приоритетный список.
2. Показ текущего состояния приоритетного списка.
3. Работа с очередью (добавление, удаление, вывод).
4. Работа со стеком (добавление, удаление, вывод).

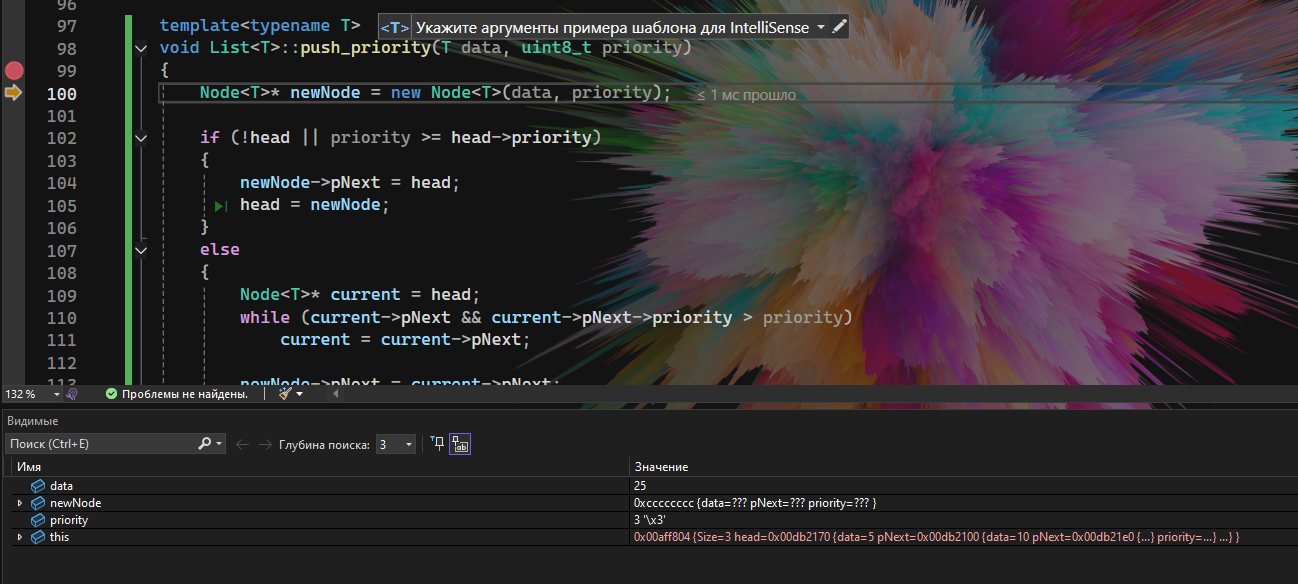
Меню позволяет пользователю в интерактивном режиме проверять работу всех структур данных и видеть изменения сразу после каждой операции.

**Трассировка программы:**

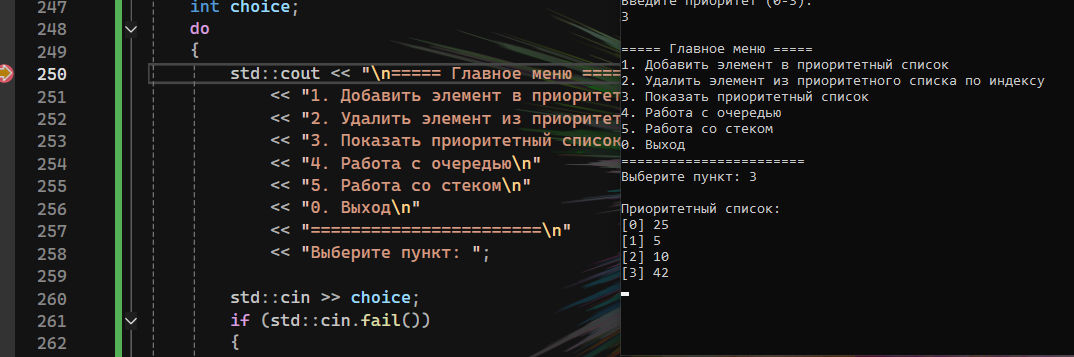
****

**Рисунок 1** – Первичный запуск и демонстрация главного меню программы.

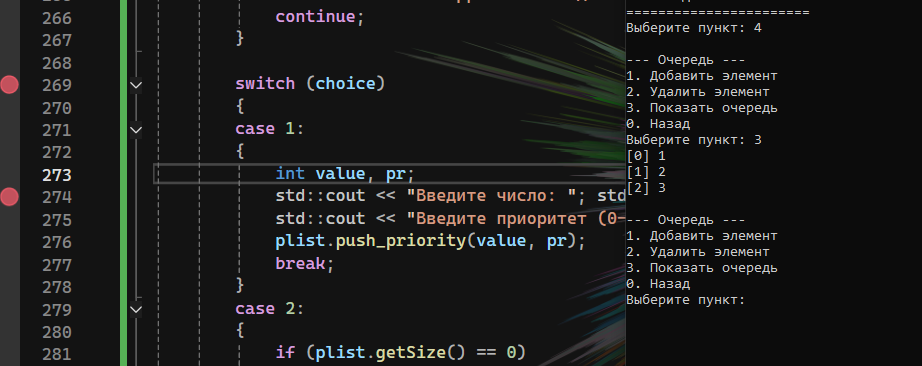




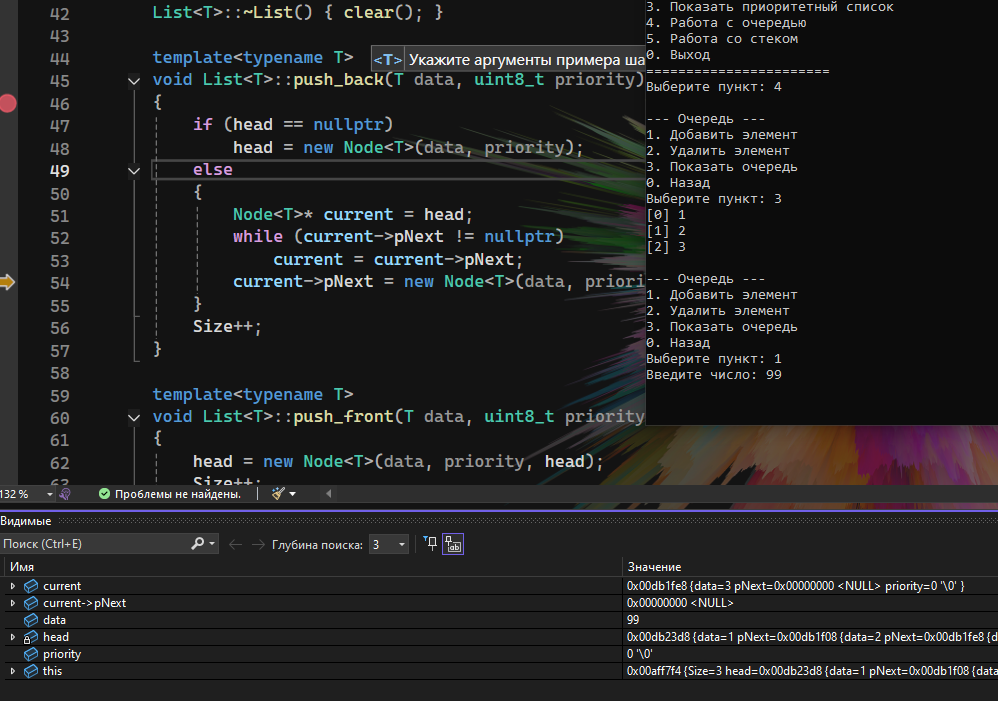
**Рисунок 2** – Добавление элементов в приоритетный список.



**Рисунок 3** – Вывод текущего состояния приоритетного списка.



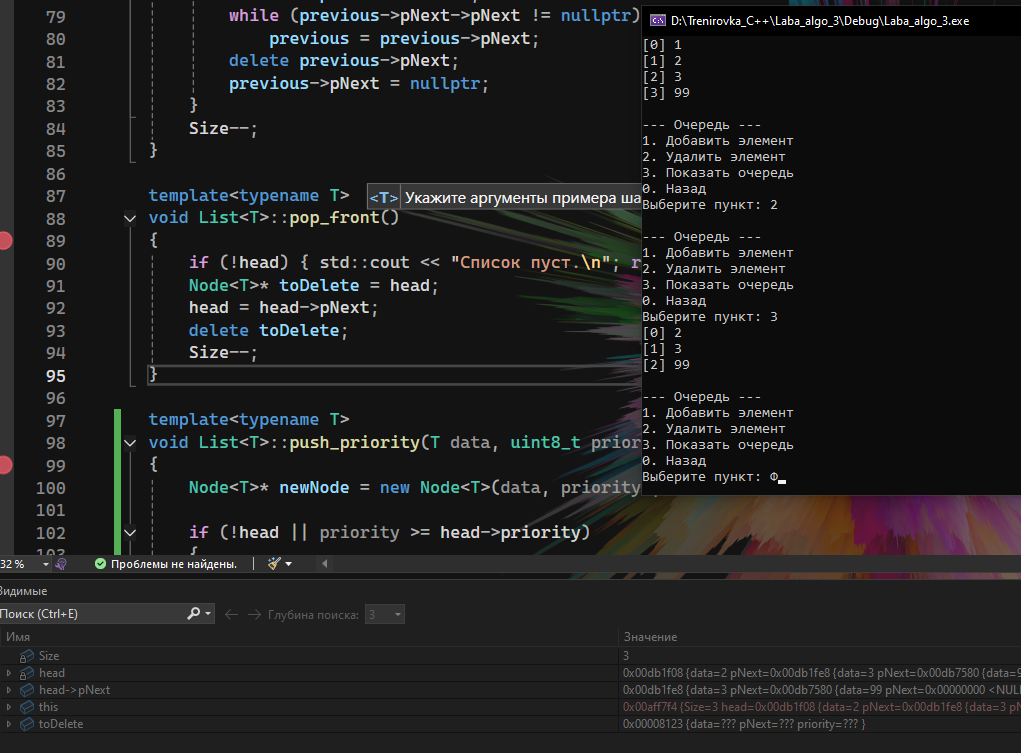
**Рисунок 4** – Работа с очередью: демонстрация подменю.



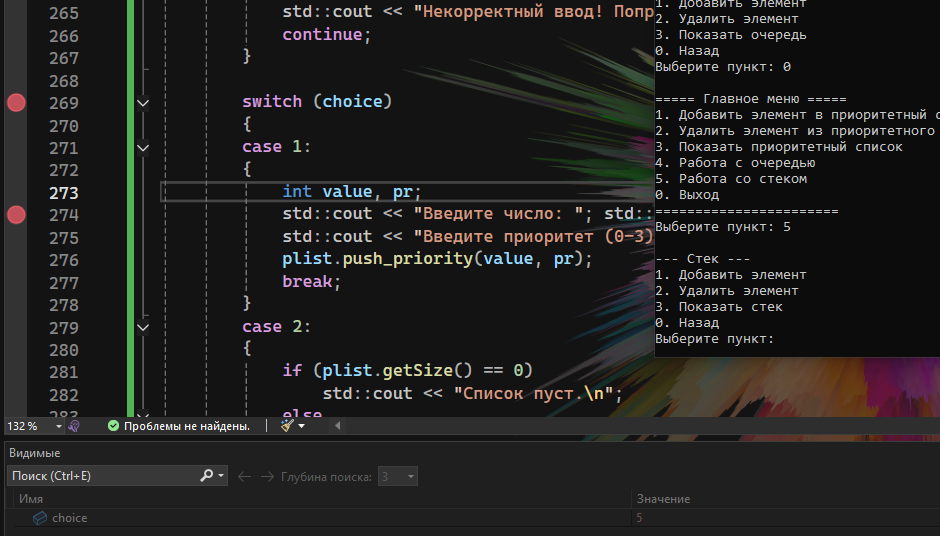
**Рисунок 5** – Работа с очередью: добавление элемента.



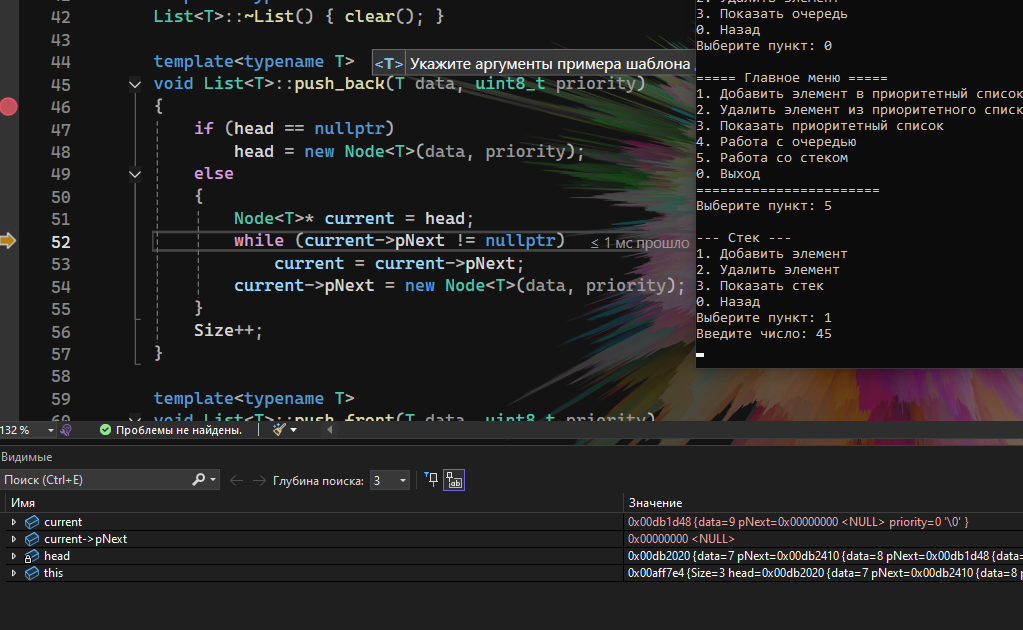
**Рисунок 6** – Работа с стеком: демонстрация очереди.



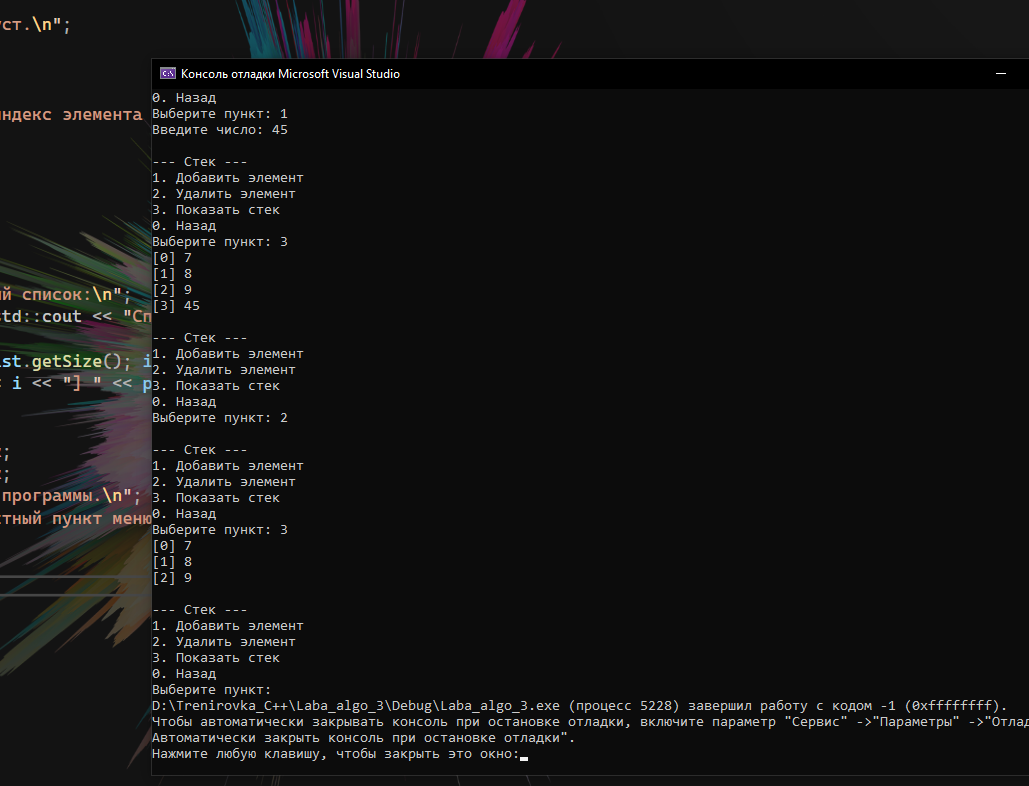
**Рисунок 7** – Работа с очередью: удаление элемента.



**Рисунок 8** – Работа со стеком: демонстрация подменю.



**Рисунок 8** – Работа со стеком: добавление элементов.



**Рисунок 9** – Работа со стеком: удаление элементов.

**6. Вывод**

В ходе лабораторной работы были:

* Изучены и реализованы динамические структуры данных на основе односвязного списка.
* Реализованы методы добавления, удаления и доступа к элементам для списка, очереди и стека.
* Реализована приоритетная вставка элементов и тестирование через консольное меню.
* Рассмотрены возможные улучшения по функциональности, визуализации и эффективности.

Программа демонстрирует корректную работу всех операций и позволяет экспериментировать с динамическими структурами, при этом её функционал легко расширять.

**Листинг программы:**

#include <iostream>

template<typename T>

class List

{

public:

List();

~List();

void push\_back(T data, uint8\_t priority = 0);

void push\_front(T data, uint8\_t priority = 3);

void pop\_back();

void pop\_front();

void removeAt(const int index);

void clear();

int getSize() { return Size; }

void push\_priority(T data, uint8\_t priority);

T& operator[](const int index);

bool isEmpty() { return Size == 0; }

private:

template<typename U>

class Node

{

public:

U data;

Node<U>\* pNext;

uint8\_t priority;

Node(U data = U(), uint8\_t priority = 0, Node<U>\* pNext = nullptr)

: data(data), pNext(pNext), priority(priority) {

}

};

int Size;

Node<T>\* head;

};

template<typename T>

List<T>::List() : Size(0), head(nullptr) {}

template<typename T>

List<T>::~List() { clear(); }

template<typename T>

void List<T>::push\_back(T data, uint8\_t priority)

{

if (head == nullptr)

head = new Node<T>(data, priority);

else

{

Node<T>\* current = head;

while (current->pNext != nullptr)

current = current->pNext;

current->pNext = new Node<T>(data, priority);

}

Size++;

}

template<typename T>

void List<T>::push\_front(T data, uint8\_t priority)

{

head = new Node<T>(data, priority, head);

Size++;

}

template<typename T>

void List<T>::pop\_back()

{

if (!head) { std::cout << "Список пуст.\n"; return; }

if (!head->pNext)

{

delete head;

head = nullptr;

}

else

{

Node<T>\* previous = head;

while (previous->pNext->pNext != nullptr)

previous = previous->pNext;

delete previous->pNext;

previous->pNext = nullptr;

}

Size--;

}

template<typename T>

void List<T>::pop\_front()

{

if (!head) { std::cout << "Список пуст.\n"; return; }

Node<T>\* toDelete = head;

head = head->pNext;

delete toDelete;

Size--;

}

template<typename T>

void List<T>::push\_priority(T data, uint8\_t priority)

{

Node<T>\* newNode = new Node<T>(data, priority);

if (!head || priority >= head->priority)

{

newNode->pNext = head;

head = newNode;

}

else

{

Node<T>\* current = head;

while (current->pNext && current->pNext->priority > priority)

current = current->pNext;

newNode->pNext = current->pNext;

current->pNext = newNode;

}

Size++;

}

template<typename T>

void List<T>::removeAt(const int index)

{

if (index < 0 || index >= Size) { std::cout << "Некорректный индекс.\n"; return; }

if (index == 0) { pop\_front(); return; }

Node<T>\* previous = head;

for (int i = 0; i < index - 1; i++)

previous = previous->pNext;

Node<T>\* toDelete = previous->pNext;

previous->pNext = toDelete->pNext;

delete toDelete;

Size--;

}

template<typename T>

void List<T>::clear()

{

while (Size > 0)

pop\_front();

}

template<typename T>

T& List<T>::operator[](const int index)

{

int count = 0;

Node<T>\* current = head;

while (current != nullptr)

{

if (count == index)

return current->data;

current = current->pNext;

count++;

}

throw std::out\_of\_range("Индекс вне диапазона");

}

template<typename T>

class Queue {

List<T> list;

public:

void enqueue(T data) { list.push\_back(data); }

void dequeue() { list.pop\_front(); }

T& front() { return list[0]; }

bool empty() { return list.isEmpty(); }

int size() { return list.getSize(); }

void print()

{

if (list.isEmpty()) { std::cout << "Очередь пуста.\n"; return; }

for (int i = 0; i < list.getSize(); i++)

std::cout << "[" << i << "] " << list[i] << "\n";

}

};

template<typename T>

class Stack {

List<T> list;

public:

void push(T data) { list.push\_back(data); }

void pop() { list.pop\_back(); }

T& top() { return list[list.getSize() - 1]; }

bool empty() { return list.isEmpty(); }

int size() { return list.getSize(); }

void print()

{

if (list.isEmpty()) { std::cout << "Стек пуст.\n"; return; }

for (int i = 0; i < list.getSize(); i++)

std::cout << "[" << i << "] " << list[i] << "\n";

}

};

void queueMenu(Queue<int>& queue)

{

int sub = -1;

while (sub != 0)

{

std::cout << "\n--- Очередь ---\n"

<< "1. Добавить элемент\n"

<< "2. Удалить элемент\n"

<< "3. Показать очередь\n"

<< "0. Назад\n"

<< "Выберите пункт: ";

std::cin >> sub;

if (sub == 1) { int v; std::cout << "Введите число: "; std::cin >> v; queue.enqueue(v); }

else if (sub == 2) queue.dequeue();

else if (sub == 3) queue.print();

}

}

void stackMenu(Stack<int>& stack)

{

int sub = -1;

while (sub != 0)

{

std::cout << "\n--- Стек ---\n"

<< "1. Добавить элемент\n"

<< "2. Удалить элемент\n"

<< "3. Показать стек\n"

<< "0. Назад\n"

<< "Выберите пункт: ";

std::cin >> sub;

if (sub == 1) { int v; std::cout << "Введите число: "; std::cin >> v; stack.push(v); }

else if (sub == 2) stack.pop();

else if (sub == 3) stack.print();

}

}

void menu()

{

List<int> plist;

Queue<int> queue;

Stack<int> stack;

plist.push\_priority(10, 2);

plist.push\_priority(5, 3);

plist.push\_priority(42, 1);

queue.enqueue(1);

queue.enqueue(2);

queue.enqueue(3);

stack.push(7);

stack.push(8);

stack.push(9);

int choice;

do

{

std::cout << "\n===== Главное меню =====\n"

<< "1. Добавить элемент в приоритетный список\n"

<< "2. Удалить элемент из приоритетного списка по индексу\n"

<< "3. Показать приоритетный список\n"

<< "4. Работа с очередью\n"

<< "5. Работа со стеком\n"

<< "0. Выход\n"

<< "=======================\n"

<< "Выберите пункт: ";

std::cin >> choice;

if (std::cin.fail())

{

std::cin.clear();

std::cin.ignore(1000, '\n');

std::cout << "Некорректный ввод! Попробуйте снова.\n";

continue;

}

switch (choice)

{

case 1:

{

int value, pr;

std::cout << "Введите число: "; std::cin >> value;

std::cout << "Введите приоритет (0-3): "; std::cin >> pr;

plist.push\_priority(value, pr);

break;

}

case 2:

{

if (plist.getSize() == 0)

std::cout << "Список пуст.\n";

else

{

int index;

std::cout << "Введите индекс элемента для удаления: ";

std::cin >> index;

plist.removeAt(index);

}

break;

}

case 3:

{

std::cout << "\nПриоритетный список:\n";

if (plist.getSize() == 0) std::cout << "Список пуст.\n";

else

for (int i = 0; i < plist.getSize(); i++)

std::cout << "[" << i << "] " << plist[i] << "\n";

break;

}

case 4: queueMenu(queue); break;

case 5: stackMenu(stack); break;

case 0: std::cout << "Выход из программы.\n"; break;

default: std::cout << "Некорректный пункт меню! Попробуйте снова.\n"; break;

}

} while (choice != 0);

}

int main()

{

setlocale(LC\_ALL, "Russian");

menu();

return 0;

}