Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная кафедра»

**ОТЧЕТ**

по лабораторной работе №3

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Динамические списки»

Выполнили:

студент группы 23ВВВ2

Кокарев Д.С.

Приняли:

Юрова О.В.

Митрохин М.А.

Пенза 2024

**Название**

Динамические списки.

**Цель работы**

Изучить работу динамических списков.

**Лабораторное задание**

1. Реализовать приоритетную очередь, путём добавления элемента в список в соответствии с приоритетом объекта (т.е. объект с большим приоритетом становится перед объектом с меньшим приоритетом).

2. \*На основе приведенного кода реализуйте структуру данных Очередь.

3. \*На основе приведенного кода реализуйте структуру данных Стек.

**Описание метода решения задачи**

Задание 1:

*Приоритетная очередь.* Запись (добавление элемента) в любое место списка по приоритету - функция void enqueue.

Извлечение (удаление элемента) из конца списка - функция void dequeue. При извлечении удаляется элемент с самым низким приоритетом.

Задание 2:

*Очередь.* Запись (добавление элемента) в начало списка - функция void push.

Извлечение (удаление элемента) из конца списка - функция void popSpecificPriority. При извлечении удаляется элемент, который добавили самым первым.

Задание 3:

*Стек.* Запись (добавление элемента) в конец списка - функция void push.

Извлечение (удаление элемента) из конца списка - функция void pop. При извлечении удаляется элемент, который добавили самым последним.

**Листинг**

**lab3.cpp**

//Задание 1

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <cstring>

#include <locale>

#include <memory>

struct Node {

char inf[256]; // Массив для хранения данных

std::shared\_ptr<Node> next; // Указатель на следующий узел стека

};

std::shared\_ptr<Node> head = nullptr; // Указатель на вершину стека

// Функция для создания нового узла стека

std::shared\_ptr<Node> createNode() {

char s[256];

std::cout << "Введите название элемента: ";

std::cin.getline(s, sizeof(s));

// Проверка на пустую строку

if (strlen(s) == 0) {

std::cout << "Элемент не добавлен\n";

return nullptr;

}

// Создание нового узла и копирование данных

auto newNode = std::make\_shared<Node>();

strcpy(newNode->inf, s);

newNode->next = nullptr;

return newNode;

}

// Функция для добавления узла на вершину стека

void push() {

auto newNode = createNode();

if (newNode == nullptr) return;

newNode->next = head;

head = newNode;

}

// Функция для удаления узла с вершины стека

void pop() {

if (head == nullptr) {

std::cout << "Стек пуст\n";

return;

}

head = head->next; // Удаляем узел

}

// Функция для просмотра данных на вершине стека

void peek() {

if (head == nullptr) {

std::cout << "Стек пуст\n";

return;

}

std::cout << "Верхний элемент стека: " << head->inf << std::endl;

}

// Функция для просмотра всех узлов стека

void review() {

auto temp = head;

if (head == nullptr) {

std::cout << "Стек пуст\n";

return;

}

while (temp) {

std::cout << "Элемент стека: " << temp->inf << std::endl;

temp = temp->next;

}

}

int main() {

setlocale(LC\_ALL, "RUS");

int choice;

// Цикл для работы с меню

do {

std::cout << "-------------------------------------------------\n";

std::cout << "1. Добавить элемент в стек \n"

<< "2. Удалить элемент из стека\n"

<< "3. Посмотреть верхний элемент стека\n"

<< "4. Просмотреть весь стек\n"

<< "5. Выйти\n";

std::cout << "Ваш выбор: ";

std::cin >> choice;

std::cin.ignore();

switch (choice) {

case 1:

push();

break;

case 2:

pop();

break;

case 3:

peek();

break;

case 4:

review();

break;

case 5:

std::cout << "Выход\n";

break;

default:

std::cout << "Неправильный выбор. Попробуйте еще раз.\n";

break;

}

} while (choice != 5);

return 0;

}

//Задание 2

#include <iostream>

#include <locale>

#include <memory>

struct Node {

int data;

int priority;

Node\* next;

};

class PriorityQueue {

private:

Node\* head;

public:

PriorityQueue() : head(nullptr) {}

void push(int data, int priority) {

Node\* temp = new Node{ data, priority, nullptr };

if (head == nullptr || head->priority < priority) {

temp->next = head;

head = temp;

}

else {

Node\* current = head;

while (current->next != nullptr && current->next->priority >= priority) {

current = current->next;

}

temp->next = current->next;

current->next = temp;

}

}

int popHighestPriority() {

if (head == nullptr) {

std::cout << "Очередь пуста.\n";

return -1;

}

Node\* temp = head;

head = head->next;

int data = temp->data;

delete temp;

return data;

}

int popLowestPriority() {

if (head == nullptr) {

std::cout << "Очередь пуста.\n";

return -1;

}

Node\* current = head;

Node\* prev = nullptr;

while (current->next != nullptr) {

prev = current;

current = current->next;

}

int data = current->data;

delete current;

if (prev == nullptr) {

head = nullptr;

}

else {

prev->next = nullptr;

}

return data;

}

void display() {

Node\* current = head;

while (current != nullptr) {

std::cout << "Данные: " << current->data << ", Приоритет: " << current->priority << std::endl;

current = current->next;

}

}

int popSpecificPriority(int targetPriority) {

Node\* current = head;

Node\* prev = nullptr;

while (current != nullptr) {

if (current->priority == targetPriority) {

int data = current->data;

if (current == head) {

head = current->next;

}

else {

prev->next = current->next;

}

delete current;

std::cout << "Удаленный элемент с заданным приоритетом: " << data << std::endl;

return 1;

}

else {

prev = current;

current = current->next;

}

}

std::cout << "Элемент с приоритетом " << targetPriority << " не найден.\n";

return -1;

}

};

int main() {

setlocale(LC\_ALL, "RUS");

PriorityQueue pq;

int choice;

int data;

int priority;

while (true) {

std::cout << "-------------------------------------------------\n";

std::cout << "1. Добавить элемент\n"

<< "2. Удалить элемент с наивысшим приоритетом\n"

<< "3. Удалить элемент с наименьшим приоритетом\n"

<< "4. Удалить элемент с выбранным приоритетом\n"

<< "5. Показать очередь\n"

<< "6. Выход\n";

std::cout << "Выберите действие: ";

std::cin >> choice;

switch (choice) {

case 1:

std::cout << "Введите число: ";

std::cin >> data;

std::cout << "Введите приоритет: ";

std::cin >> priority;

pq.push(data, priority);

break;

case 2:

data = pq.popHighestPriority();

if (data != -1) {

std::cout << "Удаленный элемент с наивысшим приоритетом: " << data << std::endl;

}

break;

case 3:

data = pq.popLowestPriority();

if (data != -1) {

std::cout << "Удаленный элемент с наименьшим приоритетом: " << data << std::endl;

}

break;

case 4:

std::cout << "Введите приоритет: ";

std::cin >> priority;

pq.popSpecificPriority(priority);

break;

case 5:

pq.display();

break;

case 6:

return 0;

default:

std::cout << "Неверный выбор. Пожалуйста, попробуйте снова.\n";

}

}

return 0;

}

//Задание 3

#include <iostream>

#include <string>

#include <vector>

#include <locale>

class Node {

public:

std::string inf;

Node\* next;

Node(const std::string& info) : inf(info), next(nullptr) {}

};

class Queue {

private:

Node\* head;

Node\* last;

public:

Queue() : head(nullptr), last(nullptr) {}

void enqueue(const std::string& info) {

Node\* newNode = new Node(info);

if (head == nullptr) {

head = last = newNode; // Очередь пуста

}

else {

last->next = newNode; // Добавляем элемент в конец

last = newNode;

}

}

void dequeue() {

if (head == nullptr) {

std::cout << "Очередь пуста.\n";

return;

}

Node\* temp = head;

head = head->next;

delete temp;

}

void display() const {

Node\* current = head;

if (current == nullptr) {

std::cout << "Очередь пуста.\n";

return;

}

while (current) {

std::cout << "Имя - " << current->inf << "\n";

current = current->next;

}

}

~Queue() {

while (head) {

dequeue();

}

}

};

int main() {

setlocale(LC\_ALL, "RUS");

int coloch;

std::cout << "Введите количество очередей: ";

std::cin >> coloch;

std::vector<Queue> queues(coloch);

while (true) {

int choice, queueNum;

std::cout << "-------------------------------------------------\n";

std::cout << "1. Добавить элемент в очередь\n2. Удалить элемент из очереди\n3. Просмотреть очередь\n4. Выйти\n";

std::cout << "Выберите действие: ";

std::cin >> choice;

if (choice == 4) break;

std::cout << "Выберите номер очереди (от 1 до " << coloch << "): ";

std::cin >> queueNum;

queueNum--;

if (queueNum < 0 || queueNum >= coloch) {

std::cout << "Неверный номер очереди.\n";

continue;

}

if (choice == 1) {

std::string name;

std::cout << "Введите название объекта: ";

std::cin >> name; // Считывание информации

queues[queueNum].enqueue(name); // Добавление элемента в очередь

}

else if (choice == 2) {

queues[queueNum].dequeue(); // Удаление элемента из очереди

}

else if (choice == 3) {

queues[queueNum].display(); // Просмотр содержимого очереди

}

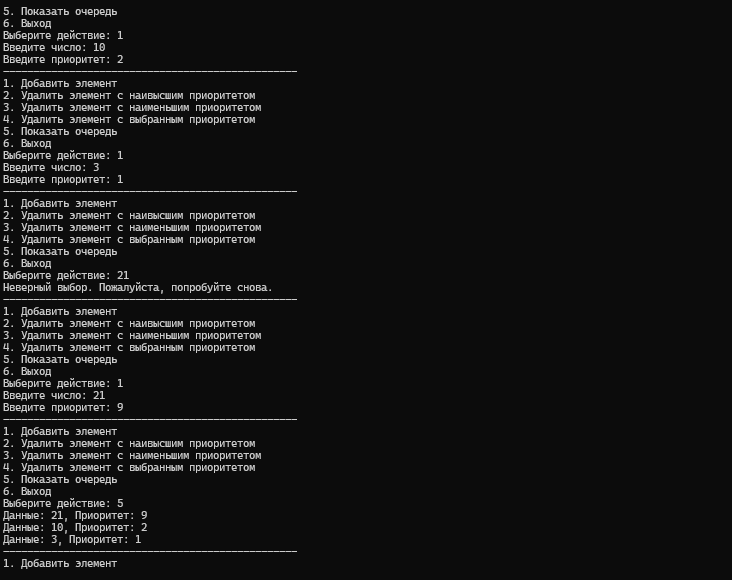
}

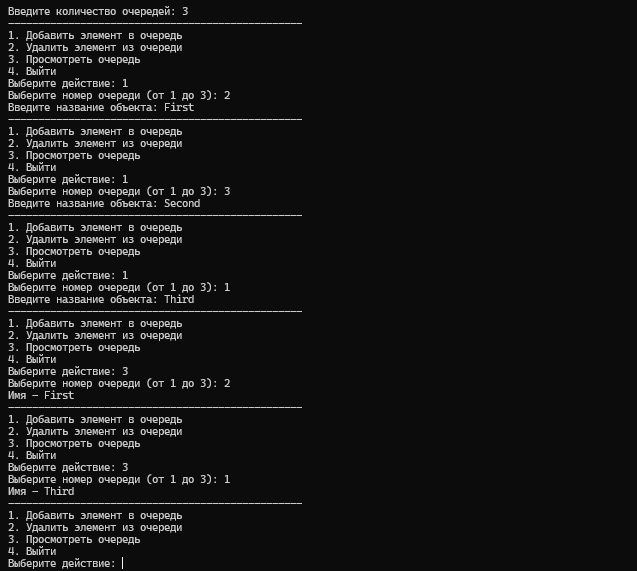
return 0;

}

**Результаты работы программы**

Рисунок 1 - Стек

Рисунок 2 - Приоритетная очередь

Рисунок 3 - Очередь

**Вывод**

В ходе выполнения лабораторной работы была разработана программа с использованием динамических структур данных. Результаты работы программы совпали с ожидаемыми результатами, следовательно, программа работает без ошибок. Также был получен опыт в создании проектов в среде Microsoft Visual Studio и приобретен навык программирования алгоритмов.