МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего

образования

«Казанский национальный исследовательский технический университет

им. А.Н. Туполева-КАИ»

Институт компьютерных технологий и защиты информации

Кафедра Прикладной математики и Информатики имени Ю. В. Кожевникова

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
К КУРСОВОЙ РАБОТЕ**

**по дисциплине** «Структуры и алгоритмы обработки данных»

**Тема работы**: Реализация комбинированных структур данных

Вариант № 66.12

Выполнил:   
 студент группы 4211  
Ишгулов Р.Р.

Проверил:   
 Доцент каф. ПМИ к.т.н.

Сотников С. В.

Казань 2025

**Оглавление**

[Введение 3](#_Toc199293598)

[1. Постановка задачи 4](#_Toc199293599)

[2. Описание структуры данных «Динамически приоритетная очередь динамически приоритетных очередей» 5](#_Toc199293600)

[2.1. Динамическая приоритетная очередь 5](#_Toc199293601)

[2.2. Комбинированная структура данных «Динамически приоритетная очередь динамически приоритетных очередей» 8](#_Toc199293602)

[3. Руководство программиста 9](#_Toc199293603)

[3.1. Описание структуры программы 9](#_Toc199293604)

[3.2. Описание структур 10](#_Toc199293605)

[3.3. Описание разработанных классов и подпрограмм 14](#_Toc199293606)

[3.4. Описание структуры внешнего файла 19](#_Toc199293607)

[Формат файла (построчно): 19](#_Toc199293608)

[4. Руководство пользователя 20](#_Toc199293609)

[4.1. Функции основного меню и их тестирование 20](#_Toc199293610)

[Заключение 23](#_Toc199293611)

[Литература 24](#_Toc199293612)

[Приложения 25](#_Toc199293613)

## Введение

**Динамически приоритетная очередь динамически приоритетных очередей** — это структура данных, в которой каждая очередь имеет свой приоритет, и каждая из этих очередей тоже может содержать элементы с приоритетами. То есть:

* Наружная очередь сортируется по приоритету самих очередей.
* Внутри каждой очереди — тоже элементы с приоритетами.

Она реализуется **динамически** — с помощью указателей и списков, что позволяет:

* добавлять новые очереди и элементы во время работы программы,
* гибко управлять памятью.

Такую структуру удобно использовать там, где задачи приходят в группах и каждая группа имеет свою важность.

## 1. Постановка задачи

**Цель работы:** реализовать комбинированную структуру вида: динамически приоритетная очередь динамически приоритетных очередей. Структура должна иметь следующее информационное наполнение: «Бензиновая компания (название) — композиция автозаправок (номер), автозаправка — композиция бензоколонок (номер, марка бензина))».

Требования к программному комплексу:

1. Реализация всех необходимых операций (Добавление и удаление в основной и присоединенной структурах, поиск в списке).
2. Возможность сохранения всей структуры во внешнем .txt файле с обратной загрузкой.
3. Реализация структуры для хранения и обработки данных конкретной информационной задачи.
4. Именования типов, структур и их полей, классов и их свойств, и методов в соответствии с конкретной информационной задачей.
5. Наличие демо-модуля с удобным оконным пользовательским интерфейсом.

Требования к программной документации:

1. Описание всех функций каждого модуля (для чего они предназначены, их входные и выходные параметры, тип и назначение).
2. Описание возможностей, предоставляемых данным программным комплексом (интерфейсная часть).

Ожидаемым результатом выполнения курсовой работы является разработанная структура данных в виде классов и функций для работы с ними, реализующие весь необходимый функционал.

## 2. Описание структуры данных «Динамически приоритетная очередь динамически приоритетных очередей»

### 2.1. Динамическая приоритетная очередь

Динамически приоритетная очередь представляет собой линейную структуру данных, в которой каждый элемент имеет значение и приоритет. Элементы упорядочиваются **по возрастанию приоритета**, то есть элемент с меньшим значением приоритета находится ближе к началу очереди.

Каждый элемент реализуется в виде записи (структуры), содержащей три поля: информационное поле data, числовой приоритет priority и указатель next на следующий элемент. Очередь управляется через указатель head, указывающий на первый (наиболее приоритетный) элемент.



Рисунок 1. Схема логического порядка элементов приоритетной очереди

#### Основные операции приоритетной очереди:

* проверка пустоты очереди (isEmpty);
* добавление нового элемента с приоритетом (enqueue);
* удаление элемента с наивысшим приоритетом (dequeue);
* получение доступа к первому элементу (front);
* удаление элемента с заданным приоритетом (remove);
* поиск элемента по приоритету (find);
* просмотр всех элементов в очереди (display, toString);
* очистка всей очереди (clear).

Добавление нового элемента немного по-разному реализуется для пустой и непустой приоритетной очереди. Аналогично, по-разному выполняется удаление из очереди, содержащей один или более одного элемента. Чтобы эти операции выполнялись единообразно, в базовой реализации может использоваться фиктивный элемент-заголовок. Однако в данной реализации PriorityQueue фиктивный заголовок **не используется**, и управление очередью ведется непосредственно через указатель head, указывающий на первый по приоритету элемент.

В этом случае создание пустой приоритетной очереди включает в себя:

* инициализацию указателя head значением nullptr;
* указатель tail в данной реализации отсутствует, так как элементы вставляются не только в конец.

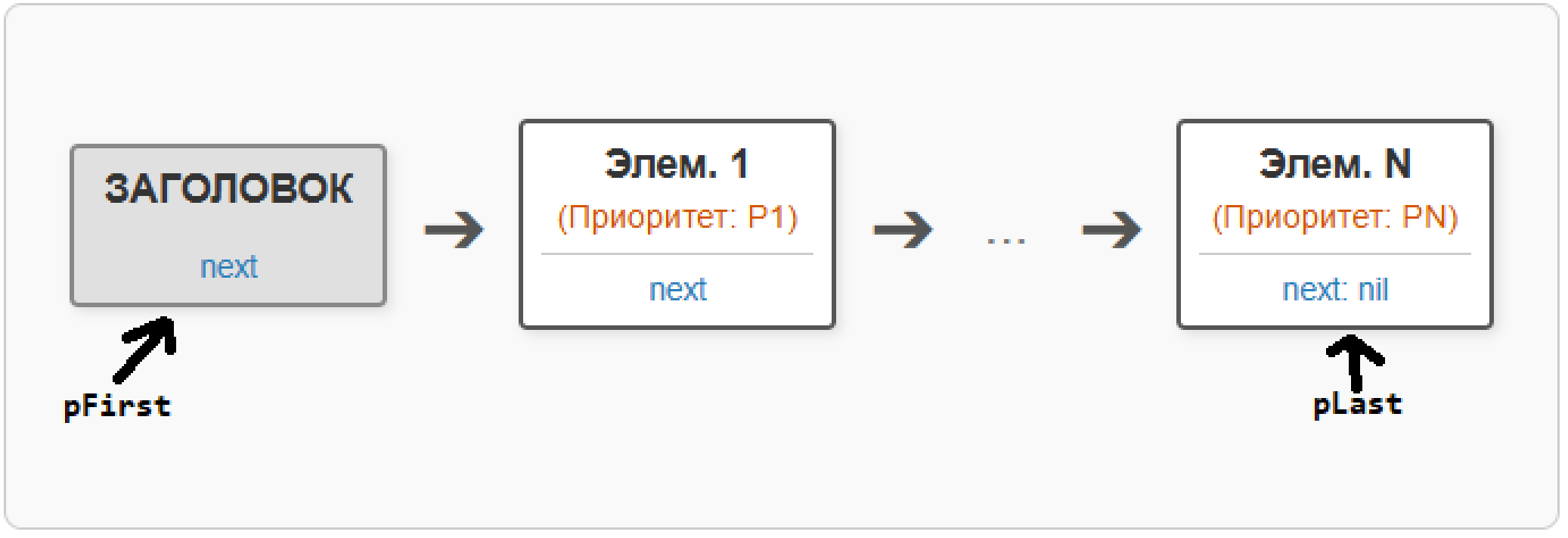


Рисунок 2. Схемы пустой и непустой приоритетных очередей:й

Для прохода по приоритетной очереди от первого к последнему элементу необходимо:

* ввести вспомогательную ссылочную переменную temp;
* установить temp в адрес первого элемента head;
* организовать цикл до достижения конца очереди (temp != nullptr);
* в теле цикла обработать очередной элемент и перейти к следующему temp = temp->next.

Добавление нового элемента в приоритетную очередь (enqueue) выполняется следующим образом:

* выделить память для нового элемента newNode с помощью стандартной функции new;
* установить его поля data, priority, next;
* если очередь пуста (head == nullptr) или новый элемент имеет меньший приоритет, чем head, он вставляется первым;
* иначе выполняется линейный проход по очереди до нужной позиции с учётом приоритета;
* вставка осуществляется в отсортированном порядке: новый элемент становится между подходящими элементами, сохраняя упорядоченность по приоритету.

Удаление элемента с наивысшим приоритетом (dequeue) выполняется следующим образом:

* вспомогательная переменная temp указывает на head;
* указатель head сдвигается на следующий элемент;
* память, занимаемая temp, освобождается.

Если после удаления очередь становится пустой (head == nullptr), дополнительная инициализация не требуется.

### 2.2. Комбинированная структура данных «Динамически приоритетная очередь динамически приоритетных очередей»

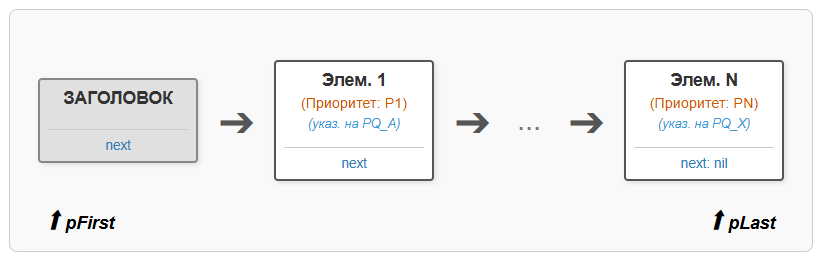
Рассмотренные выше способы объединения элементов могут комбинироваться друг с другом, образуя достаточно сложные структуры. Один из возможных случаев такого объединения – список указателей на приоритетные очереди. Каждый элемент такого списка содержит два поля: указатель на соседний элемент списка и указатель на базовую приоритетную очередь. Поскольку структура базовой приоритетной очереди отличается от структуры элемента внешнего списка, их необходимо описывать отдельно и вводить два ссылочных типа данных. 

Рисунок 3 Схема динамической очереди, содержащей указатели на другие приоритетные очереди.

## 3. Руководство программиста

### 3.1. Описание структуры программы

Для разработки проекта была использована среда разработки программного обеспечения **Visual Studio** на языке **C++**.

Разработанная **структура данных "Динамически приоритетная очередь динамически приоритетных очередей"** реализована в виде набора классов, обеспечивающих как хранение приоритетных данных, так и их управление.

Агрегационное взаимодействие построено следующим образом:

* один внешний класс содержит динамически расширяемый список, каждый элемент которого ссылается на отдельную приоритетную очередь;
* каждая приоритетная очередь реализована на основе связного списка, упорядоченного по значению приоритета;
* функциональные методы обеспечивают добавление, удаление, поиск и отображение данных как внутри отдельных очередей, так и в совокупности всей структуры.

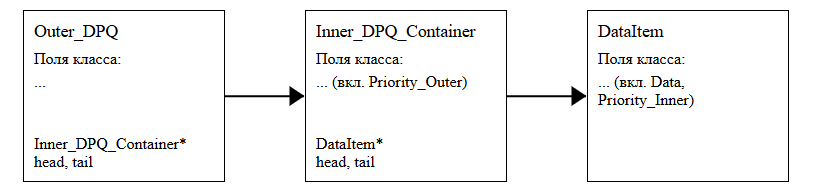


Рисунок 4. Схема агрегационного взаимодействия классов

### 3.2. Описание структур

Контейнер представляет собой комбинированную структуру.

Основными элементами контейнера являются динамически приоритетные очереди, каждая из которых реализована как динамическая очередь.

1. Класс GasCompany:

class GasCompany {

private:

wstring companyName;

PriorityQueue<GasStation> stations;

public:

GasCompany() : companyName(L"") {}

GasCompany(const wstring& name) : companyName(name) {}

wstring getName() const { return companyName; }

void setName(const wstring& name) { companyName = name; }

bool addStation(const GasStation& station);

bool removeStation(int stationNumber);

GasStation\* findStation(int stationNumber);

void display();

wstring toString();

bool isEmpty();

};

1. Класс GasStation:

class GasStation {

private:

int stationNumber;

PriorityQueue<GasPump> pumps;

public:

GasStation() : stationNumber(0) {}

GasStation(int num) : stationNumber(num) {}

int getNumber() const { return stationNumber; }

void setNumber(int num) { stationNumber = num; }

bool addPump(const GasPump& pump);

bool removePump(int pumpNumber);

GasPump\* findPump(int pumpNumber);

void display();

wstring toString();

bool isEmpty();

bool operator==(const GasStation& other);

}

};

1. Класс GasPump:

class GasPump {

private:

int pumpNumber;

wstring fuelType;

public:

GasPump() : pumpNumber(0), fuelType(L"") {}

GasPump(int num, const wstring& type) : pumpNumber(num), fuelType(type) {}

int getNumber();

wstring getFuelType();

void setNumber(int num);

void setFuelType(const wstring& type);

void display() const {

wcout << L"Колонка #" << pumpNumber << L" (" << fuelType << L")";

}

wstring toString();

bool operator==(const GasPump& other);

};

1. Класс PriorityQueue:

template <typename T>

class PriorityQueue {

private:

struct Node {

T data;

int priority;

Node\* next;

Node(const T& d, int p);

};

Node\* head;

public:

PriorityQueue();

PriorityQueue(const PriorityQueue& other);

PriorityQueue& operator=(const PriorityQueue& other);

PriorityQueue(PriorityQueue&& other) noexcept;

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

~PriorityQueue();

void clear();

bool enqueue(const T& data, int priority);

bool dequeue();

T\* front() const;

bool isEmpty() const;

bool remove(int priority);

T\* find(int priority);

void display() const;

std::wstring toString() const;

};

### 3.3. Описание разработанных классов и подпрограмм

**Класс** GasCompany

|  |  |
| --- | --- |
| **Название поля** | **Описание** |
| wstring companyName | Название газовой компании. |
| PriorityQueue<GasStation> stations | Очередь автозаправок с приоритетом по номеру станции. |
| **Название метода** | **Описание** |
| GasCompany() | Конструктор по умолчанию. |
| GasCompany(const wstring& name) | Конструктор с заданием имени компании. |
| wstring getName() const | Возвращает название компании. |
| void setName(const wstring& name) | Устанавливает название компании. |
| bool addStation(const GasStation& station) | Добавляет АЗС, если её номер уникален. |
| bool removeStation(int stationNumber) | Удаляет АЗС по номеру. |
| GasStation\* findStation(int stationNumber) | Ищет АЗС по номеру. |
| void display() const | Выводит информацию о компании и АЗС. |
| wstring toString() const | Представляет компанию в виде строки. |
| bool isEmpty() const | Проверка на наличие АЗС. |

**Класс** GasStation

|  |  |
| --- | --- |
| **Название поля** | **Описание** |
| int stationNumber | Номер автозаправочной станции. |
| PriorityQueue<GasPump> pumps | Очередь колонок на станции. |
| **Название метода** | **Описание** |
| GasStation() | Конструктор по умолчанию. |
| GasStation(int num) | Конструктор с заданием номера станции. |
| int getNumber() const | Возвращает номер станции. |
| void setNumber(int num) | Устанавливает номер станции. |
| bool addPump(const GasPump& pump) | Добавляет колонку, если номер уникален. |
| bool removePump(int pumpNumber) | Удаляет колонку по номеру. |
| GasPump\* findPump(int pumpNumber) | Ищет колонку по номеру. |
| void display() const | Выводит информацию о станции и колонках. |
| wstring toString() const | Представляет станцию в виде строки. |
| bool isEmpty() const | Проверка на наличие колонок. |
| bool operator==(const GasStation& other) const | Сравнение по номеру станции. |

**Класс** GasPump

|  |  |
| --- | --- |
| **Название поля** | **Описание** |
| int pumpNumber | Номер топливной колонки. |
| wstring fuelType | Тип топлива. |
| **Название метода** | **Описание** |
| GasPump() | Конструктор по умолчанию. |
| GasPump(int num, const wstring& type) | Конструктор с параметрами. |
| int getNumber() const | Возвращает номер колонки. |
| wstring getFuelType() const | Возвращает тип топлива. |
| void setNumber(int num) | Устанавливает номер колонки. |
| void setFuelType(const wstring& type) | Устанавливает тип топлива. |
| void display() const | Выводит информацию о колонке. |
| wstring toString() const | Представляет колонку в виде строки. |
| bool operator==(const GasPump& other) const | Сравнение по номеру. |

**Класс** PriorityQueue

|  |  |
| --- | --- |
| **Название поля** | **Описание** |
| Node\* head | Указатель на начало очереди с приоритетом. |
| **Название метода** | **Описание** |
| PriorityQueue() | Конструктор по умолчанию. |
| ~PriorityQueue() | Деструктор, очищает очередь. |
| PriorityQueue(const PriorityQueue& other) | Конструктор копирования. |
| PriorityQueue(PriorityQueue&& other) | Конструктор перемещения. |
| PriorityQueue& operator=(const PriorityQueue& other) | Оператор копирующего присваивания. |
| PriorityQueue& operator=(PriorityQueue&& other) | Оператор перемещающего присваивания. |
| bool enqueue(const T& data, int priority) | Добавление элемента с приоритетом. |
| bool dequeue() | Удаление элемента с наивысшим приоритетом. |
| T\* front() const | Получение элемента с наивысшим приоритетом. |
| bool isEmpty() const | Проверка на пустоту. |
| bool remove(int priority) | Удаление элемента по приоритету. |
| T\* find(int priority) | Поиск элемента по приоритету. |
| void display() const | Вывод всех элементов очереди. |
| wstring toString() const | Представляет очередь в виде строки. |
| void clear() | Очистка очереди. |

**Класс** Storage

|  |  |
| --- | --- |
| **Название поля** | **Описание** |
| wstring filePath | Путь к файлу хранения данных. |
| **Название метода** | **Описание** |
| Storage(const wstring& path) | Конструктор с указанием пути к файлу. |
| wstring getFilePath() const | Возвращает путь к файлу. |
| bool saveToFile(const GasCompany& company) | Сохраняет компанию в файл. |
| bool loadFromFile(GasCompany& company) | Загружает компанию из файла. |

**Класс** Main

|  |  |
| --- | --- |
| **Название метода** | **Описание** |
| displayMenu() | Выводит текстовое меню на экран. |
| getIntInput() | Считывает и валидирует целое число. |
| main() | |  | | --- | |  |  |  | | --- | | Главная функция программы. | |

## 3.4. Описание структуры внешнего файла

Файл предназначен для хранения и восстановления данных о бензиновой компании. Структура файла отражает иерархию объектов:  
**Бензиновая компания → Автозаправки → Бензоколонки**.

#### Информационная модель:

* **Бензиновая компания** — содержит название и список автозаправок.
* **Автозаправка** — содержит номер и список бензоколонок.
* **Бензоколонка** — содержит номер и марку бензина.

### Формат файла (построчно):

<Название компании>

<Номер автозаправки>

<Номер бензоколонки 1> <Марка бензина 1>

<Номер бензоколонки 2> <Марка бензина 2>

## 4. Руководство пользователя

Взаимодействие с пользователем осуществляется через консоль. После запуска программы в консоль выводится следующее меню:

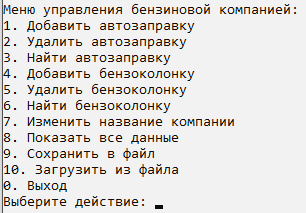


Рисунок 5. Основное меню

### 4.1. Функции основного меню и их тестирование

#### 4.1.1. Создание / Редактирование компании

Для успешного завершения операции нужно ввести название компании.



*Рисунок 6. Создание / Редактирование компании*

#### 4.1.2. Создание автозаправки

Для успешного завершения операции нужно ввести уникальный номер автозаправки.

**

*Рисунок 7. Создание автозаправки*

#### 4.1.3. Удаление автозаправки

Для успешного завершения операции нужно ввести уникальный номер автозаправки.

**

*Рисунок 8. Удаление автозаправки*

#### 4.1.4. Поиск Автозаправки

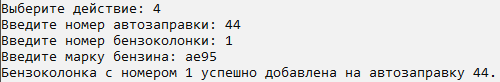
Для успешного завершения операции нужно ввести уникальный номер автозаправки.

**

*Рисунок 9. Поиск автозаправки*

#### 4.1.5. Создание бензоколонки

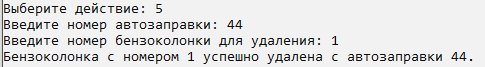
Для успешного завершения операции нужно ввести уникальный номер автозаправки, уникальный номер бензоколонки из привязанных к автозаправке и марку бензина.

**

*Рисунок 10. Создание бензоколонки*

#### 4.1.6. Удаление бензоколонки

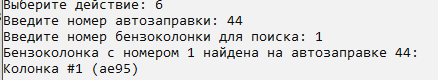
Для успешного завершения операции нужно ввести уникальный номер автозаправки, уникальный номер бензоколонки из привязанных к автозаправке.

**

*Рисунок 11. Удаление бензоколонки*

#### 4.1.7. Поиск бензоколонки

Для успешного завершения операции нужно ввести уникальный номер автозаправки.

**

*Рисунок 12 . Поиск бензоколонки*

#### 4.1.8. Вывод информации о компании, автозаправках и бензоколонках.

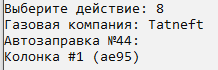


Рисунок 13. Вывод информации о компании, автозаправках и бензоколонка

#### 4.1.11. Выгрузка в файл

Выгрузка производится в соответствии с шаблоном.

#### 4.1.12. Загрузка из файла

Загрузка из файла производится в соответствии с шаблоном.

#### 4.1.13. Завершение работы с программой

Удаление всей базы данных и выход из программы.

## Заключение

Полученная структура данных обладает всем требуемым функционалом, который был определен в постановке задачи. Разработанная структура данных реализует иерархическую модель: **бензиновая компания (название) — композиция автозаправок (номер), автозаправка — композиция бензоколонок (номер, марка бензина)**. Консольное приложение позволяет наглядно продемонстрировать функциональность разработанных структур и реализованных методов взаимодействия с пользователем.

Результаты работы предоставлены в виде исходного кода программного комплекса, включающего реализацию основных классов и методов управления данными. Также был разработан внешний текстовый файл, структура которого соответствует требованиям задания. Было обеспечено сохранение и восстановление данных из файла, а также отображение информации в удобной форме.

Была написана пояснительная записка по курсовой работе, содержащая описание структуры программы, алгоритмов и логики работы приложения.

Таким образом, поставленная задача может считаться полностью выполненной.

## Литература

1. Козин А. Н. Учебно-методическое пособие «Структуры и алгоритмы обработки данных». – Казань.: КГТУ им. А.Н. Туполева, 2007.

## Приложения

Листинг программы:

GasCompany.h

#pragma once

#include "GasStation.h"

#include "PriorityQueue.h"

#include <string>

#include <iostream>

using namespace std;

class GasCompany {

private:

wstring companyName;

PriorityQueue<GasStation> stations;

public:

GasCompany() : companyName(L"") {}

GasCompany(const wstring& name) : companyName(name) {}

wstring getName() const { return companyName; }

void setName(const wstring& name) { companyName = name; }

bool addStation(const GasStation& station) {

if (stations.find(station.getNumber()) != nullptr)

{

wcout << L"Ошибка: автозаправка с номером " << station.getNumber() << L" уже существует\n";

return false;

}

return stations.enqueue(station, station.getNumber());

}

bool removeStation(int stationNumber) {

return stations.remove(stationNumber);

}

GasStation\* findStation(int stationNumber) {

return stations.find(stationNumber);

}

void display() const {

wcout << L"Газовая компания: " << companyName << L"\n";

if (!stations.isEmpty()) {

stations.display();

}

else {

wcout << L"Список автозаправок пуст.\n";

}

}

wstring toString() const {

return companyName + L"\n" + stations.toString();

}

bool isEmpty() const {

return stations.isEmpty();

}

};

GasPump.h

#pragma once

#include <string>

#include <iostream>

using namespace std;

class GasPump {

private:

int pumpNumber;

wstring fuelType;

public:

GasPump() : pumpNumber(0), fuelType(L"") {}

GasPump(int num, const wstring& type) : pumpNumber(num), fuelType(type) {}

int getNumber() const { return pumpNumber; }

wstring getFuelType() const { return fuelType; }

void setNumber(int num) { pumpNumber = num; }

void setFuelType(const wstring& type) { fuelType = type; }

void display() const {

wcout << L"Колонка #" << pumpNumber << L" (" << fuelType << L")";

}

wstring toString() const {

return to\_wstring(pumpNumber) + L" " + fuelType;

}

bool operator==(const GasPump& other) const {

return pumpNumber == other.pumpNumber;

}

};

GasStation.h

#pragma once

#include "GasPump.h"

#include "PriorityQueue.h"

#include <iostream>

using namespace std;

class GasStation {

private:

int stationNumber;

PriorityQueue<GasPump> pumps;

public:

GasStation() : stationNumber(0) {}

GasStation(int num) : stationNumber(num) {}

int getNumber() const { return stationNumber; }

void setNumber(int num) { stationNumber = num; }

bool addPump(const GasPump& pump) {

if (pumps.find(pump.getNumber()) != nullptr) {

wcout << L"Ошибка: колонка с номером " << pump.getNumber() << L" уже существует\n";

return false;

}

return pumps.enqueue(pump, pump.getNumber());

}

bool removePump(int pumpNumber) {

return pumps.remove(pumpNumber);

}

GasPump\* findPump(int pumpNumber) {

return pumps.find(pumpNumber);

}

void display() const {

wcout << L"Автозаправка №" << stationNumber << L":\n";

pumps.display();

}

wstring toString() const {

wstring result = to\_wstring(stationNumber) + L"\n";

result += pumps.toString();

return result;

}

bool isEmpty() const {

return pumps.isEmpty();

}

bool operator==(const GasStation& other) const {

return stationNumber == other.stationNumber;

}

};

PriorityQueue.h

#pragma once

#include <iostream>

using namespace std;

template <typename T>

class PriorityQueue {

private:

struct Node {

T data;

int priority;

Node\* next;

Node(const T& d, int p) : data(d), priority(p), next(nullptr) {}

};

Node\* head;

public:

PriorityQueue() : head(nullptr) {}

PriorityQueue(const PriorityQueue& other) : head(nullptr) {

Node\* currentOther = other.head;

Node\* lastNewNode = nullptr;

while (currentOther) {

Node\* newNode = new Node(currentOther->data, currentOther->priority);

if (!head) {

head = newNode;

}

else {

lastNewNode->next = newNode;

}

lastNewNode = newNode;

currentOther = currentOther->next;

}

}

PriorityQueue& operator=(const PriorityQueue& other) {

if (this == &other) {

return \*this;

}

clear();

Node\* currentOther = other.head;

Node\* lastNewNode = nullptr;

while (currentOther) {

Node\* newNode = new Node(currentOther->data, currentOther->priority);

if (!head) {

head = newNode;

}

else {

lastNewNode->next = newNode;

}

lastNewNode = newNode;

currentOther = currentOther->next;

}

return \*this;

}

PriorityQueue(PriorityQueue&& other) noexcept : head(other.head) {

other.head = nullptr;

}

PriorityQueue& operator=(PriorityQueue&& other) noexcept {

if (this == &other) {

return \*this;

}

clear();

head = other.head;

other.head = nullptr;

return \*this;

}

~PriorityQueue() {

if (head) {

clear();

}

}

void clear() {

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

bool enqueue(const T& data, int priority) {

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

if (!head || priority < head->priority) {

newNode->next = head;

head = newNode;

return true;

}

Node\* current = head;

while (current->next && current->next->priority <= priority) {

current = current->next;

}

newNode->next = current->next;

current->next = newNode;

return true;

}

bool dequeue() {

if (!head) return false;

Node\* temp = head;

head = head->next;

delete temp;

return true;

}

T\* front() const {

if (!head) return nullptr;

return &head->data;

}

bool isEmpty() const {

return head == nullptr;

}

bool remove(int priority) {

if (!head) return false;

if (head->priority == priority) {

Node\* temp = head;

head = head->next;

delete temp;

return true;

}

Node\* current = head;

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

current = current->next;

}

if (!current->next) return false;

Node\* temp = current->next;

current->next = temp->next;

delete temp;

return true;

}

T\* find(int priority) {

Node\* current = head;

while (current) {

if (current->priority == priority) {

return &current->data;

}

current = current->next;

}

return nullptr;

}

void display() const {

Node\* current = head;

while (current) {

current->data.display();

wcout << L"\n";

current = current->next;

}

}

wstring toString() const {

wstring result;

Node\* current = head;

while (current) {

result += current->data.toString() + L"\n";

current = current->next;

}

return result;

}

};

Storage.h

#pragma once

#include "GasCompany.h"

#include <fstream>

#include <locale>

#include <codecvt>

using namespace std;

class Storage {

private:

wstring filePath;

public:

Storage(const wstring& path) : filePath(path) {}

wstring getFilePath() const { return filePath; }

bool saveToFile(const GasCompany& company) {

wofstream file(filePath);

if (!file.is\_open()) return false;

file.imbue(locale(file.getloc(), new codecvt\_utf8<wchar\_t>));

file << company.toString();

file.close();

return true;

}

bool loadFromFile(GasCompany& company) {

wifstream file(filePath);

if (!file.is\_open()) {

return false;

}

file.imbue(locale(file.getloc(), new codecvt\_utf8<wchar\_t>));

GasCompany tempCompany;

wstring line;

if (!getline(file, line)) {

file.close();

return false;

}

tempCompany.setName(line);

int stationCount = 0;

while (true) {

streampos lineStartPos = file.tellg();

if (!getline(file, line)) {

if (file.eof()) {

}

else {

}

break;

}

if (line.empty()) {

continue;

}

int stationNumber;

try {

stationNumber = stoi(line);

}

catch (const invalid\_argument& ia) {

continue;

}

catch (const out\_of\_range& oor) {

continue;

}

GasStation station(stationNumber);

stationCount++;

int pumpCount = 0;

while (true) {

streampos pumpLineStartPos = file.tellg();

if (!getline(file, line)) {

if (file.eof()) {

}

else {

}

break;

}

if (line.empty()) {

file.clear();

file.seekg(pumpLineStartPos);

break;

}

size\_t spacePos = line.find(L' ');

bool isPumpLine = false;

int pumpNumber = 0;

wstring fuelType;

if (spacePos != wstring::npos && spacePos > 0) {

wstring pumpNumStr = line.substr(0, spacePos);

try {

pumpNumber = stoi(pumpNumStr);

fuelType = line.substr(spacePos + 1);

isPumpLine = true;

}

catch (const invalid\_argument& ia) {

isPumpLine = false;

}

catch (const out\_of\_range& oor) {

isPumpLine = false;

}

}

if (isPumpLine) {

GasPump pump(pumpNumber, fuelType);

if (!station.addPump(pump)) {

}

else {

pumpCount++;

}

}

else {

file.clear();

file.seekg(pumpLineStartPos);

break;

}

}

tempCompany.addStation(station);

}

file.close();

company = tempCompany;

return true;

}

};

Main.cpp

#include <iostream>

#include <limits>

#include <locale>

#include <codecvt>

#include "GasCompany.h"

#include "Storage.h"

using namespace std;

void displayMenu() {

wcout << L"\nМеню управления бензиновой компанией:\n";

wcout << L"1. Добавить автозаправку\n";

wcout << L"2. Удалить автозаправку\n";

wcout << L"3. Найти автозаправку\n";

wcout << L"4. Добавить бензоколонку\n";

wcout << L"5. Удалить бензоколонку\n";

wcout << L"6. Найти бензоколонку\n";

wcout << L"7. Изменить название компании\n";

wcout << L"8. Показать все данные\n";

wcout << L"9. Сохранить в файл\n";

wcout << L"10. Загрузить из файла\n";

wcout << L"0. Выход\n";

wcout << L"Выберите действие: ";

}

int getIntInput(const wstring& prompt) {

int value;

while (true) {

wcout << prompt;

wcin >> value;

if (wcin.fail()) {

wcin.clear();

wcin.ignore(numeric\_limits<streamsize>::max(), L'\n');

wcout << L"Ошибка ввода. Пожалуйста, введите целое число.\n";

}

else {

wcin.ignore(numeric\_limits<streamsize>::max(), L'\n');

return value;

}

}

}

int main() {

locale ru("ru\_RU.UTF-8");

locale::global(ru);

wcout.imbue(ru);

wcin.imbue(ru);

GasCompany company(L"Моя Бензиновая Компания");

Storage storage(L"data.txt");

int choice;

do {

displayMenu();

choice = getIntInput(L"");

switch (choice) {

case 1: {

int number = getIntInput(L"Введите номер автозаправки: ");

GasStation station(number);

if (company.addStation(station)) {

wcout << L"Автозаправка с номером " << number << L" успешно добавлена.\n";

}

break;

}

case 2: {

int number = getIntInput(L"Введите номер автозаправки для удаления: ");

if (company.removeStation(number)) {

wcout << L"Автозаправка с номером " << number << L" успешно удалена.\n";

}

else {

wcout << L"Ошибка: не удалось удалить автозаправку с номером " << number << L". Возможно, она не существует.\n";

}

break;

}

case 3: {

int number = getIntInput(L"Введите номер автозаправки для поиска: ");

GasStation\* station = company.findStation(number);

if (station) {

wcout << L"Автозаправка с номером " << number << L" найдена:\n";

station->display();

}

else {

wcout << L"Автозаправка с номером " << number << L" не найдена.\n";

}

break;

}

case 4: {

int stationNumber = getIntInput(L"Введите номер автозаправки: ");

GasStation\* station = company.findStation(stationNumber);

if (!station) {

wcout << L"Автозаправка с номером " << stationNumber << L" не найдена.\n";

break;

}

int pumpNumber = getIntInput(L"Введите номер бензоколонки: ");

wcout << L"Введите марку бензина: ";

wstring fuelType;

getline(wcin, fuelType);

GasPump pump(pumpNumber, fuelType);

if (station->addPump(pump)) {

wcout << L"Бензоколонка с номером " << pumpNumber << L" успешно добавлена на автозаправку " << stationNumber << L".\n";

}

break;

}

case 5: {

int stationNumber = getIntInput(L"Введите номер автозаправки: ");

GasStation\* station = company.findStation(stationNumber);

if (!station) {

wcout << L"Автозаправка с номером " << stationNumber << L" не найдена.\n";

break;

}

int pumpNumber = getIntInput(L"Введите номер бензоколонки для удаления: ");

if (station->removePump(pumpNumber)) {

wcout << L"Бензоколонка с номером " << pumpNumber << L" успешно удалена с автозаправки " << stationNumber << L".\n";

}

else {

wcout << L"Ошибка: не удалось удалить бензоколонку с номером " << pumpNumber << L" с автозаправки " << stationNumber << L". Возможно, она не существует.\n";

}

break;

}

case 6: {

int stationNumber = getIntInput(L"Введите номер автозаправки: ");

GasStation\* station = company.findStation(stationNumber);

if (!station) {

wcout << L"Автозаправка с номером " << stationNumber << L" не найдена.\n";

break;

}

int pumpNumber = getIntInput(L"Введите номер бензоколонки для поиска: ");

GasPump\* pump = station->findPump(pumpNumber);

if (pump) {

wcout << L"Бензоколонка с номером " << pumpNumber << L" найдена на автозаправке " << stationNumber << L":\n";

pump->display();

wcout << endl;

}

else {

wcout << L"Бензоколонка с номером " << pumpNumber << L" не найдена на автозаправке " << stationNumber << L".\n";

}

break;

}

case 7: {

wcout << L"Введите новое название компании: ";

wstring newName;

getline(wcin, newName);

company.setName(newName);

wcout << L"Название компании изменено на: " << newName << L"\n";

break;

}

case 8: {

company.display();

break;

}

case 9: {

if (storage.saveToFile(company)) {

wcout << L"Данные успешно сохранены в файл " << storage.getFilePath() << L".\n";

}

else {

wcout << L"Ошибка: не удалось сохранить данные в файл.\n";

}

break;

}

case 10: {

if (storage.loadFromFile(company)) {

wcout << L"Данные успешно загружены из файла " << storage.getFilePath() << L".\n";

wcout << L"Загруженная компания: " << company.getName() << endl;

company.display();

}

else {

wcout << L"Ошибка: не удалось загрузить данные из файла. Файл может не существовать или быть поврежден.\n";

}

break;

}

case 0: {

wcout << L"Выход из программы.\n";

break;

}

default: {

wcout << L"Неверный выбор. Попробуйте снова.\n";

}

}

} while (choice != 0);

return 0;

}