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](#_Toc105498204)

[1. Постановка задачи 4](#_Toc105498205)

[2. Описание структуры данных «Динамически приоритетная очередь динамически приоритетных очередей 5](#_Toc105498206)

[2.1. Динамически приоритетная очередь 5](#_Toc105498207)

[2.2. Комбинированная структура данных «Динамически приоритетная очередь динамически приоритетных очередей» 5](#_Toc105498208)

[3. Руководство программиста 9](#_Toc105498210)

[3.1. Описание структуры программы 9](#_Toc105498211)

[3.2. Описание структур 10](#_Toc105498212)

[3.3. Описание разработанных классов для работы со структурами 13](#_Toc105498213)

[3.4. Схема взаимодействия классов. 18](#_Toc105498214)

[3.5. Описание структуры внешнего файла 19](#_Toc105498215)

[4. Руководство пользователя 20](#_Toc105498216)

[4.1. Функции основного меню и их тестирование 20](#_Toc105498217)

[Заключение 25](#_Toc105498218)

[Использованная литература 26](#_Toc105498219)

[Приложение 27](#_Toc105498220)

## 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.

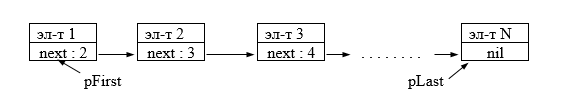


Рисунок 1. Схема логического порядка элементов приоритетной очереди

#### 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.

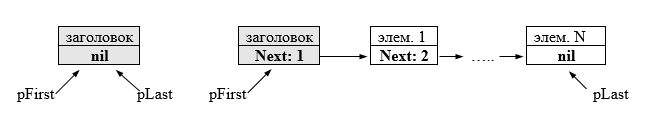


Рисунок 2. Схемы пустой и непустой приоритетных очередей:й

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»

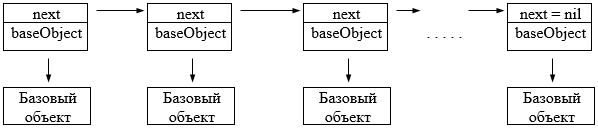
Рассмотренные выше способы объединения элементов могут комбинироваться друг с другом, образуя достаточно сложные структуры. Один из возможных случаев такого объединения – список указателей на приоритетные очереди. Каждый элемент такого списка содержит два поля: указатель на соседний элемент списка и 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.

Factory

Поля класса:

…

Object\*

head, tail

Company

Поля класса:

…

Factory\* head, tail

Object

Поля класса:

…

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:

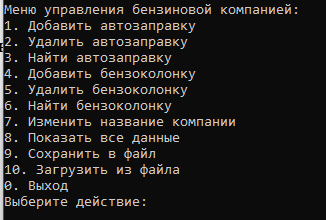
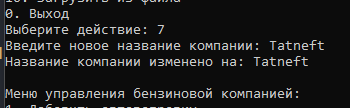


Рисунок 5. Основное меню

### 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. Удаление меню

Для успешного завершения операции необходимо ранее создать компанию с помощью предыдущего пункта.



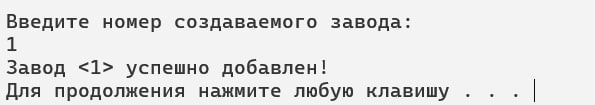
*Рисунок 7. Удаление компании*



*Рисунок 8. Ошибка удаления компании*

#### 4.1.3. Добавление подменю

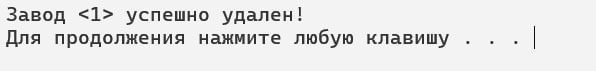
Для успешного завершения операции следует ввести непустую строку.



*Рисунок 9. Добавление филиала*

#### 4.1.4. Удаление подменю

Удаление завершится успешно при наличии в очереди хотя бы одного элемента. В противном случае выведется соответствующее сообщение.



*Рисунок 10. Успешное удаление подменю*

Изображение выглядит как текст, Шрифт, чек, белый

Автоматически созданное описание

*Рисунок 11. Вывод сообщения об ошибке*

#### 4.1.5. Добавление объекта

Для успешного завершения операции необходимо выбрать существующее подменю и корректно ввести строки текста и комбинации клавиш команды.

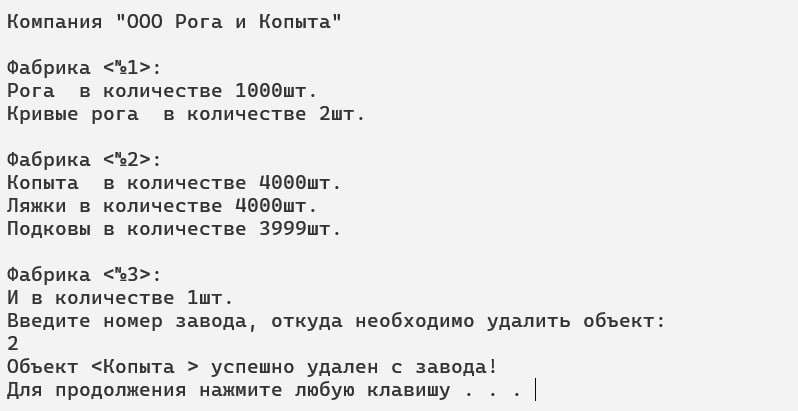
Изображение выглядит как текст, снимок экрана, Шрифт, алгебра

Автоматически созданное описание

*Рисунок 12. Добавление команды*

#### 4.1.6. Удаление объекта

Для успешного завершения операции необходимо, чтобы в очереди находился как минимум один элемент. В противном случае выведется сообщение об ошибке.



*Рисунок 13. Удаление объекта*

#### 4.1.7. Поиск команды в фабрике

Для успешного завершения операции необходимо ввести текст существующего подменю и ввести текст находящейся там команды. В случае ввода на одном из этапов несуществующего элемента, выведется соответствующее сообщение.

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

*Рисунок 14. Поиск объекта в фабрике*

#### 4.1.8. Поиск фабрики в меню

Для завершения операции пользователю необходимо ввести название искомой команды.

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

*Рисунок 15. Поиск фабрики по всей компании*

#### 4.1.9. Вывод состояния компании в консоли.

Изображение выглядит как текст, чек, снимок экрана, Шрифт

Автоматически созданное описание

*Рисунок 16. Вывод состояния меню в консоль*

#### 4.1.10. Загрузка из файла

Производится загрузка данных из файла «save.txt», расположенного в корневом каталоге проекта. Файл имеет следующий вид:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

*Рисунок 17. Файл «save.txt»*

На первой строке файла расположено название компании. На каждой из последующих строк располагается информация о входящих в неё подменю(фабрик).

Первым элементом строки является номер завода. Каждые последующие два элемента являются информационной частью одной из объектов, хранимых в фабриках:

В случае, если файл пуст или его не удалось найти, в консоль выведется соответствующее сообщение. Каждая строка файла должна содержать не более тысячи символов.

Изображение выглядит как текст, чек, снимок экрана, Шрифт

Автоматически созданное описание

*Рисунок 18. Результат загрузки файла «save.txt»*

#### 4.1.11. Выгрузка в файл

Выгрузка производится в соответствии с шаблоном.

#### 4.1.12. 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. Консольное приложение позволяет наглядно продемонстрировать функциональность разработанных структур. Результаты работы предоставлены в виде исходного кода программного комплекса. Была написана пояснительная записка по курсовой работе. Таким образом, задачу можно считать 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:

Company.cpp:

#pragma once

#include "Factory.cpp"

class Company

{

private:

struct CompanyInfo {

string name;

}company;

Factory\* head;

Factory\* tail;

public:

Company() {

company.name = "";

head = new Factory();

tail = head;

}

Company(string name) {

company.name = name;

head = new Factory();

tail = head;

}

~Company() {

while (!IsEmpty()) {

Factory\* temp = head->GetNext();

head->SetNext(temp->GetNext());

delete temp;

}

delete head;

}

bool IsEmpty() {

return head->GetNext() == nullptr;

}

void Add(Factory\* manufacture) {

tail->SetNext(manufacture);

tail = manufacture;

}

string Delete() {

if (IsEmpty())

return nullptr;

Factory\* temp = head->GetNext();

head->SetNext(temp->GetNext());

if (IsEmpty())

tail = head;

string \_temp = temp->GetText();

delete temp;

return \_temp;

}

Factory\* Search(string num){

if (IsEmpty())

return nullptr;

Factory\* temp = head->GetNext();

while (temp != nullptr) {

if (temp->GetText() == num)

return temp;

temp = temp->GetNext();

}

return nullptr;

}

Factory\* SearchElement(string name) {

if (IsEmpty())

return nullptr;

Factory\* temp = head->GetNext();

while (temp != nullptr) {

Object\* \_temp = temp->Search(name);

if (\_temp != nullptr)

return temp;

temp = temp->GetNext();

}

return nullptr;

}

void Show() {

cout << "Компания \"ООО " << company.name << "\"\n";

if (IsEmpty())

cout << "Производства отсутствуют\n";

else {

Factory\* temp = head->GetNext();

while (temp != nullptr) {

temp->Show();

temp = temp->GetNext();

}

}

}

void SetName(string \_name) {

company.name = \_name;

}

void SetHead(Factory\* \_head) {

head = \_head;

}

void SetTail(Factory\* \_tail) {

tail = \_tail;

}

string GetName() {

return company.name;

}

Factory\* GetHead() {

return head;

}

Factory\* GetTail() {

return tail;

}

};

Factory.cpp:

#pragma once

#include "Object.cpp"

#include <iostream>

using namespace std;

class Factory

{

private:

struct FactoryInfo {

string num;

}factory;

Factory\* next;

Object\* head;

Object\* tail;

public:

Factory() {

factory.num = -1;

next = nullptr;

head = new Object();

tail = head;

}

Factory(string \_num){

factory.num = \_num;

next = nullptr;

head = new Object();

tail = head;

}

~Factory() {

while (!IsEmpty()) {

Object\* temp = head->GetNext();

head->SetNext(temp->GetNext());

delete temp;

}

delete head;

head = nullptr;

tail = nullptr;

next = nullptr;

}

bool IsEmpty() {

return(head->GetNext() == nullptr);

}

void Add(Object\* obj) {

tail->SetNext(obj);

tail = obj;

}

string Delete() {

if (IsEmpty())

return "";

Object\* temp = head->GetNext();

head->SetNext(temp->GetNext());

if (IsEmpty()) //Если после удаления очередь стала пустой

head = tail;

string \_temp = temp->GetName();

delete temp;

return \_temp;

}

Object\* Search(string name){

if (IsEmpty())

return nullptr;

Object\* temp = head->GetNext();

while (temp != nullptr)

{

if (temp->GetName() == name)

return temp;

temp = temp->GetNext();

}

return nullptr;

}

void Show() {

cout << "\nФабрика <№" << factory.num << ">:\n";

if (IsEmpty())

cout << "Оборудование отсутствует!\n";

else{

Object\* temp = head->GetNext();

while (temp != nullptr) {

cout << temp->GetName() << " в количестве " << temp->GetNum() << "шт.\n";

temp = temp->GetNext();

}

}

}

void SetNext(Factory\* \_next) {

next = \_next;

}

void SetTNum(int \_num) {

factory.num= \_num;

}

void SetHead(Object\* \_head) {

head = \_head;

}

void SetTail(Object\* \_tail) {

tail = \_tail;

}

string GetText() {

return factory.num;

}

Factory\* GetNext() {

return next;

}

Object\* GetHead() {

return head;

}

Object\* GetTail() {

return tail;

}

};

Object.cpp:

#pragma once

#include <string>

using namespace std;

class Object {

private:

struct ObjectInfo {

string name;

string num;

}object;

Object\* next;

public:

Object(){

object.name = "";

object.num = -1;

next = nullptr;

}

Object(string name, string num){

object.name = name;

object.num = num;

next = nullptr;

}

~Object(){

next = nullptr;

}

void SetName(string name){

object.name = name;

}

void SetNum(int num){

object.num = num;

}

void SetNext(Object\* \_next) {

next = \_next;

}

string GetName(){

return object.name;

}

string GetNum(){

return object.num;

}

Object\* GetNext() {

return next;

}

};

CW\_40.4.cpp:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <iostream>

#include <cstdlib>

#include <string>

#include <fstream>

#include "Company.cpp"

using namespace std;

Company\* company = nullptr;

int GetNaturalValue() {

int answer;

while (!(cin >> answer) || (cin.peek() != '\n') || (answer < 0))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка: введено некорректное число. " << endl <<

"Повторите ввод: " << endl;

}

return answer;

}

string GetCorrectString() {

string str;

getline(cin, str);

while (str.empty())

{

cout << "Ошибка: введена некорректная строка." << endl <<

"Повторите ввод: " << endl;

getline(cin, str);

}

return str;

}

void Dialog() {

int answer = -1;

while (answer != 0)

{

cout <<

"Выберите пункт: " << endl <<

"1 - Создать компанию." << endl <<

"2 - Удалить компанию." << endl <<

"3 - Добавить завод." << endl <<

"4 - Удалить завод." << endl <<

"5 - Добавить объект." << endl <<

"6 - Удалить объект." << endl <<

"7 - Поиск объекта на заводе." << endl <<

"8 - Поиск объекта в компании." << endl <<

"9 - Вывести состояние компании в консоль." << endl <<

"10 - Загрузить." << endl <<

"11 - Сохранить." << endl <<

"0 - Завершить работу с программой." << endl;

answer = GetNaturalValue();

system("cls");

switch (answer)

{

case 1:

{

delete company;

cout << "Введите название компании: " << endl;

cin.ignore();

string name = GetCorrectString();

company = new Company(name);

cout << "Компания <" << company->GetName() << "> успешно создана!" << endl;

break;

}

case 2:

if (company != nullptr)

{

delete company;

company = nullptr;

cout << "Компания успешно удалена!" << endl;

}

else

cout << "Для работы с компанией необходимо её создание." << endl;

break;

case 3:

if (company != nullptr)

{

cout << "Введите номер создаваемого завода: " << endl;

cin.ignore();

string number = GetCorrectString();

company->Add(new Factory(number));

cout << "Завод <" << number << "> успешно добавлен!" << endl;

}

else

cout << "Для работы с компанией необходимо её создание." << endl;

break;

case 4:

if (company != nullptr)

{

string number = company->Delete();

if (!number.empty())

cout << "Завод <" << number << "> успешно удален!" << endl;

else

cout << "Удаление невозможно: компания пуста." << endl;

}

else

cout << "Для работы с компанией необходимо её создание." << endl;

break;

case 5:

if (company != nullptr)

{

company->Show();

cout << "Введите номер завода, куда планируется добавить объект: " << endl;

cin.ignore();

string number = GetCorrectString();

Factory\* factory = company->Search(number);

if (factory != nullptr)

{

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

string name = GetCorrectString();

cout << "Укажите количество объектов: " << endl;

string num = GetCorrectString();

factory->Add(new Object(name, num));

cout << "Объект <" << name << "> добавлен на завод в количестве " << num << "!" << endl;

}

else

cout << "К сожалению, искомый завод не найден." << endl;

}

else

cout << "Для работы с компанией необходимо её создание." << endl;

break;

case 6:

if (company != nullptr)

{

company->Show();

cout << "Введите номер завода, откуда необходимо удалить объект: " << endl;

cin.ignore();

string number = GetCorrectString();

Factory\* factory = company->Search(number);

if (factory != nullptr)

{

string name = factory->Delete();

if (!name.empty())

cout << "Объект <" << name << "> успешно удален с завода!" << endl;

else

cout << "Ошибка удаления: завод пуст." << endl;

}

else

cout << "К сожалению, искомый завод не найден." << endl;

}

else

cout << "Для работы с компанией необходимо её создание." << endl;

break;

case 7:

if (company != nullptr)

{

company->Show();

cout << "Введите номер завода, где необходимо найти объект: " << endl;

cin.ignore();

string number = GetCorrectString();

Factory\* factory = company->Search(number);

if (factory != nullptr)

{

cout << "Введите название искомого объекта:" << endl;

string name = GetCorrectString();

Object\* object = factory->Search(name);

if (object)

cout << "Объект <" << name << "> обнаружен!" << endl <<

"Количество объектов - " << object->GetNum() << endl;

else

cout << "Объект не найден." << endl;

}

else

cout << "К сожалению, искомый завод не найден." << endl;

}

else

cout << "Для работы с компанией необходимо её создание." << endl;

break;

case 8:

if (company != nullptr)

{

company->Show();

cout << "Введите название искомого объекта:" << endl;

cin.ignore();

string name = GetCorrectString();

Factory\* factory = company->SearchElement(name);

if (factory)

cout << "Объект <" << name << "> обнаружен " <<

"на заводе <" << factory->GetText() << ">!" << endl;

else

cout << "Объект не найден." << endl;

}

else

cout << "Для работы с компанией необходимо её создание." << endl;

break;

case 9:

if (company != nullptr)

{

company->Show();

}

else

cout << "Для работы с компанией необходимо её создание." << endl;

break;

case 10:

{

ifstream in;

in.open("save.txt");

if (!in.is\_open())

{

cout << "Ошибка чтения файла." << endl;

}

else

{

delete company;

company = nullptr;

char line[1000];

in.getline(line, 1000);

if (strlen(line) != 0)

company = new Company(line);

else

{

cout << "Ошибка чтения: файл пуст!" << endl;

break;

}

while (!in.eof()) {

in.getline(line, 1000);

char\* parts = strtok(line, "><");

if (parts == NULL)

continue;

Factory\* factory = new Factory(parts);

company->Add(factory);

parts = strtok(NULL, "><");

while (parts) {

string text = parts;

parts = strtok(NULL, "><");

string num = parts;

parts = strtok(NULL, "><");

Object\* object = new Object(text, num);

factory->Add(object);

}

}

}

in.close();

cout << "Данные успешно загружены!" << endl;

break;

}

case 11:

{

if (company != nullptr)

{

ofstream out;

out.open("save.txt");

if (!out.is\_open())

{

cout << "Ошибка записи файла." << endl;

}

else

{

out << company->GetName() << endl;

Factory\* factory = company->GetHead()->GetNext();

while (factory != nullptr) {

out << factory->GetText();

Object\* object = factory->GetHead()->GetNext();

while (object != nullptr)

{

out << "><" << object->GetName() << "><" << object->GetNum();

object = object->GetNext();

}

factory = factory->GetNext();

out << endl;

}

out.close();

cout << "Данные успешно сохранены!" << endl;

}

}

else

cout << "Для работы с компанией необходимо её создание." << endl;

break;

}

case 0:

cout << "Всего доброго!" << endl;

break;

default:

cout << "Ошибка: выбранного вами пункта не существует." << endl;

break;

}

system("pause");

system("cls");

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Dialog();

return 0;

}