МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего

профессионального 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**: комбинированная структура

на основе динамического упорядоченного списка статических стеков

Выполнила:   
 студент группы 4211  
Дубакина Д.И.

Проверил:   
 Доцент каф. ПМИ к.т.н.

Сотников С. В.

Казань 2018

**ОГЛАВЛЕНИЕ**

[1. ПОСТАНОВКА ЗАДАЧИ 4](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308544)

[2. ОПИСАНИЕ СТРУКТУРЫ ДАННЫХ ДИНАМИЧЕСКИЙ УПОРЯДОЧЕННЫЙ СПИСОК СТАТИЧЕСКИХ СТЕКОВ 5](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308545)

[2.1. Структура данных типа «ДИНАМИЧЕСКИЙ УПОРЯДОЧЕНЫЙ СПИСОК» 5](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308546)

[2.2. Структура данных типа «СТАТИЧЕСКИЙ СТЕК» 7](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308547)

[2.3. Комбинированная структура данных «Динамический упорядоченный список статических стеков» 8](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308548)

[3. РУКОВОДСТВО ДЛЯ ПРОГРАММИСТА 1](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308549)0

[3.1. Описание структуры проекта 10](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308550)

[3.2. Описание всех разработанных структур 1](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308551)1

[3.3. Описание всех разработанных подпрограмм для работы со структурами 12](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308552)

[4. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 17](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308553)

[4.1. Функции основного меню 18](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308554)

[4.1.1. Добавить автозаправку 18](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308555)

[4.1.2. Добавить бензоколонку 18](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308556)

[4.1.3. Удалить автозаправку 19](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308557)

[4.1.4. Удалить бензоколонку 20](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308558)

[4.1.5. Поиск автозаправки 20](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308559)

[4.1.6. Вывод данных на консоль 21](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308561)

[4.1.7. Сохранение базы 21](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308562)

[4.1.8. Загрузка базы 22](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308563)

[4.1.9. Удалить базу 23](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308563)

[4.2. Сохранение структуры во внешнем txt файле 24](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308564)

[5. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММНОГО КОМПЛЕКСА](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308565) 26

[5.1. Организация тестирования ПО 26](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308566)

[5.2. Входные данные 26](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308567)

[ЗАКЛЮЧЕНИЕ 30](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308568)

[ЛИТЕРАТУРА 31](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308569)

[ПРИЛОЖЕНИЕ 32](file:///C:\Users\1\Desktop\KURSACh_SAOD_OBRAZETs.docx#_Toc483308570)

**ВВЕДЕНИЕ**

Типизация данных является одним из фундаментальных понятий современного программирования. Отнесение переменных к тому или иному типу позволяет установить внутренний формат хранения значений этой переменной и набор допустимых операций. Все распространенные языки программирования имеют набор базовых простейших типов данных (целочисленные, вещественные, символьные, логические) и возможность объединения их в составные наборы – массивы, записи, файлы. Понятие структуры данных определяется двумя моментами:

* способом объединения отдельных компонент в единую структуру
* способами обработки как отдельных компонент структуры, так и всей структуры в целом.

Большинство дополнительных структур данных можно реализовать двумя способами:

* статически на основе массива
* динамически с помощью специальных переменных-указателей

Каждый из этих способов имеет свои преимущества и недостатки.

## **ПОСТАНОВКА ЗАДАЧИ**

Необходимо реализовать комбинированную структуру вида: динамический упорядоченный список статических стеков (вариант 26,12) на основе объектного подхода.

Структура должна иметь следующее информационное наполнение: Бензиновая компания (название) – Композиция автозаправок (номер) - Автозаправка – композиция бензоколонок (номер, марка бензина).

Требования к программному комплексу:

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

**Ожидаемый результат**

Ожидаемым результатом выполнения курсовой работы является разработанная иерархия классов, которая включает в себя классы, отвечающие за информационное заполнение, функции и процедуры для работы с этими структурами, реализующие весь намеченный функционал.

А также разработка демонстрационного модуля, показывающего всю специфику работы разработанных классов.

## **ОПИСАНИЕ СТРУКТУРЫ ДАННЫХ ДИНАМИЧЕСКИЙ УПОРЯДОЧЕННЫЙ СПИСОК СТАТИЧЕСКИХ СТЕКОВ**

## **2.1. Структура данных типа «Динамический упорядоченный список»**

Упорядоченный линейный список – это набор связанных однотипных элементов, в котором элементы выстраиваются в соответствии с заданным порядком, например – целые числа по возрастанию, текстовые строки по алфавиту. Для таких списков изменяется процедура добавления – новый элемент должен вставляться в соответствующее место для сохранения порядка элементов. В отличие от стека и очереди, добавление нового элемента возможно в любом месте списка, также можно удалить любой элемент списка. Ясно, что списковые структуры являются более гибкими, но и немного более сложными в реализации. Фактически, стеки и очереди можно считать частными случаями списков, в которых добавление и удаление элементов может выполняться только на концах.

Стандартный набор операций со списком включает:

* добавление нового элемента
* удаление заданного элемента
* проход по списку от первого элемента к последнему с выполнением заданных действий
* поиск в списке заданного элемента

Динамическая реализация линейного списка, также как стека и очереди, основана на динамическом выделении и освобождении памяти для элементов списка. Логическая последовательность элементов списка создается ссылочными переменными с адресами последующих элементов (последний элемент имеет пустую ссылку NULL).

**Создание пустого списка включает:**

* выделение памяти под заголовок
* установку пустой ссылочной части заголовка

**Поиск заданного элемента включает:**

* установку вспомогательного указателя в адрес первого элемента списка
* организацию цикла прохода по списку с завершением либо по совпадению информационной части элемента с заданным значением, либо по достижению конца списка
* после завершения цикла проверить значение вспомогательного указателя и сделать вывод об успешности поиска

**Удаление заданного элемента включает:**

* определение каким-то образом удаляемого элемента
* поиск удаляемого элемента с определением адреса элемента-предшественника (для этого вводится еще одна вспомогательная ссылочная переменная, инициированная адресом заголовка и изменяющая свое значение внутри цикла)
* если удаляемый элемент найден, то изменяется ссылочная часть его предшественника
* удаляемый элемент обрабатывается необходимым образом, т.е. либо освобождается занимаемая им память, либо он включается во вспомогательный список

**Добавление нового элемента включает:**

* поиск элемента, после или до которого будет добавлен элемент с одновременным определением элемента-предшественника
* выделение памяти для нового элемента с помощью еще одного указателя
* формирование полей нового элемента, в частности – настройка ссылочной части
* изменение ссылочной части элемента-предшественника на адрес нового элемента

## **2.2. Структура данных типа «Статический стек»**

Стек – это линейная структура данных, в которую элементы добавляются и удаляются только с одного конца, называемого вершиной стека. Стек работает по принципу “элемент, помещенный в стек последним, извлечен будет первым”. Иногда этот принцип обозначается сокращением LIFO (Last In – First Out, т.е. последним зашел – первым вышел).

Пусть в стеке требуется сохранять целые числа, причем заранее известно их максимальное количество. Для реализации стека надо объявить массив и одну переменную – указатель вершины стека (SP – Stack Pointer). Будем считать, что стек-массив заполняется (растет) от первых элементов массива к последним. Тогда указатель SP может определять либо последнюю занятую ячейку массива, либо первую свободную ячейку.

Со стеком связываются две основные операции: добавление (вталкивание, PUSH) элемента в стек и удаление (выталкивание, POP) элемента из стека.

**Добавление включает в себя:**

* проверку возможности добавления (стек-массив заполнен полностью?)
* размещение нового элемента в ячейке, определяемой значением переменной SP
* увеличение SP на 1

**Удаление включает в себя:**

* проверку возможности удаления (стек пустой?)
* уменьшение SP на 1
* извлечение элемента из ячейки, определяемой значением переменной SP

## **2.3. Комбинированная структура данных «Динамический упорядоченный список статических стеков»**

Более сложным случаем является использование списка стеков. Здесь каждый элемент основного списка имеет ссылку на начало вспомогательного стека. Логическая схема реализации комбинированной структуры данных представлена на рис.1.

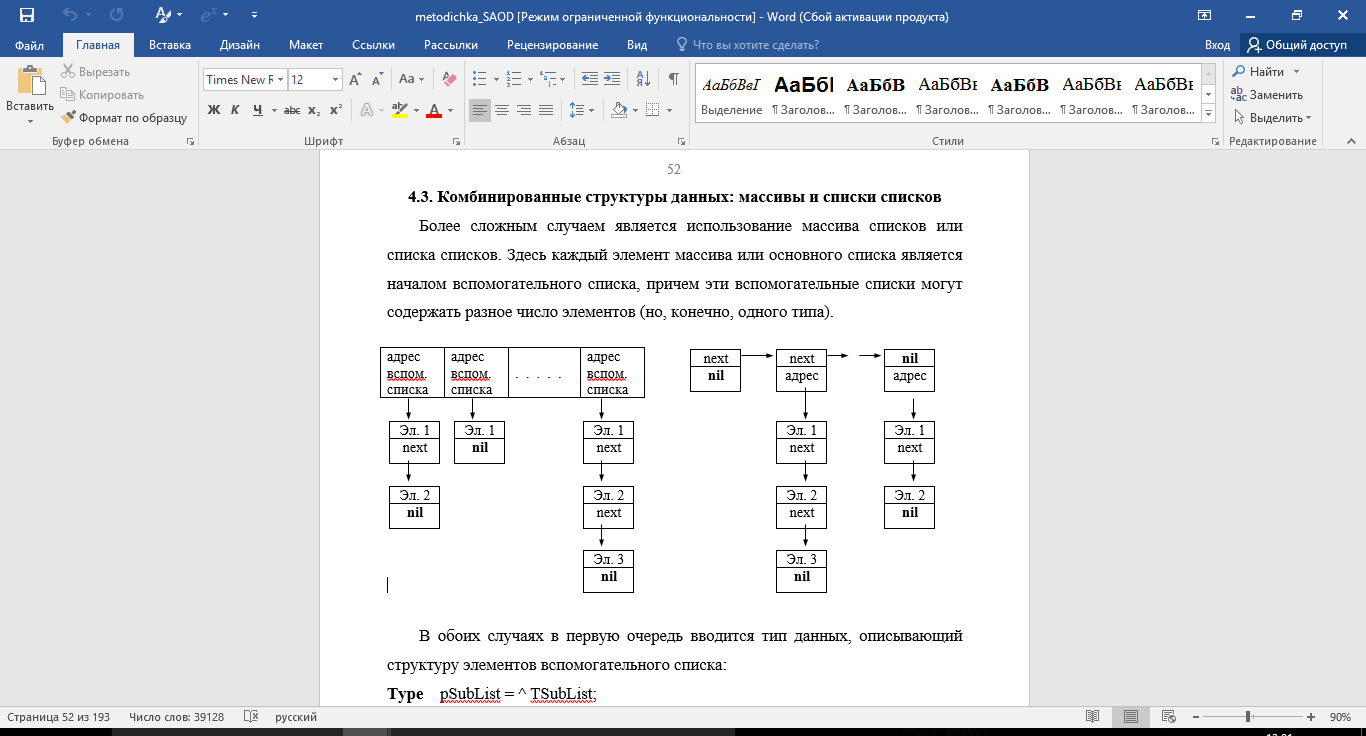


Рисунок 1 «Схема комбинированной структуры данных»

В первую очередь вводится тип данных, описывающий структуру элементов вспомогательного стека. После этого описывается структура элементов основного списка. Обработка таких структур включает больше операций, поскольку практически любая типовая операция (поиск, просмотр, добавление, удаление) может выполняться как с основным списком, так и с любым вспомогательным стеком. Например, полный проход реализуется двойным циклом: внешний цикл проходит по элементам основного списка, а внутренний обрабатывает отдельно каждый вспомогательный стек. Для этого необходимы две ссылочные переменные: для прохода по основному списку и для прохода по вспомогательному стеку.

Добавление и удаление элементов во вспомогательных стеках выполняется обычным образом. Небольшие особенности имеет удаление элемента из основного списка, поскольку в этом случае как правило надо удалить и весь вспомогательный стек. Поэтому прежде всего организуется проход по вспомогательному стеку с удалением каждого элемента, а потом уже производится удаление элемента основного списка.

## **РУКОВОДСТВО ДЛЯ ПРОГРАММИСТА**

## **Описание структуры проекта**

Для разработки проекта была использована среда разработки программного обеспечения Microsoft Visual Studio 2017.

Весь программный код, включая основную программу, хранится в файлах .cpp и .h. Инструментарий Microsoft Visual Studio 2017 использует расширение .sln и .vcxproj чтобы отличать модуль основной программы от прочих модулей, имеющих традиционное расширение .cpp.

Разработанная иерархия классов, представленная на рисунке 2, включает в себя классы, отвечающие за информационное заполнение, а также функции и процедуры, предназначенные для работы с созданными структурами данных.

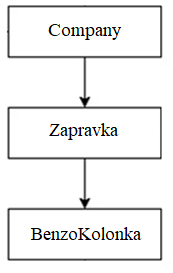


Рис.2. Иерархия классов

## **Описание всех разработанных структур**

Контейнер представляет собой комбинированную структуру данных. Главной структурой является динамический упорядоченный список. Дочерняя структура (элементы главной структуры) – статический стек.

Разработаны две основные структуры, представленные в следующих контейнерных классах:

* Zapravka (динамический упорядоченный список)

class Zapravka

{

private:

Zapravka \*next;

string number;

BenzoKolonka \*kolonka;

int count;

}

number – информационное поле – строка, содержащая номер автозаправки

\*kolonka – ссылочное поле – массив элементов вспомогательного стека

\*next – ссылочное поле – ссылка на следующий элемент списка

count – вершина стека, определяющая первую свободную ячейку

* BenzoKolonka (статический стек)

class BenzoKolonka

{

private:

string nomer, oil;

}

nomer - информационное поле – строка, содержащая номер бензоколонки

oil – информационное поле – строка, содержащая марку бензина

## **Описание всех разработанных подпрограмм для работы со структурами**

**Класс Company**

Класс, реализующий работу с главным списком (отделы).

Табл.1. Класс Company

|  |  |
| --- | --- |
| **Название метода** | **Описание** |
| public:  Company(string name1) | Конструктор данного класса, инициализирует информационное поле (название) бензиновой компании. Параметр name1 – название компании. |
| public:  Content() | Конструктор данного класса. |
| public:  void AddZapravka() | Метод добавления автозаправки. |
| public:  void AddBenzokolonka() | Метод добавления бензоколонки.  Перед добавлением выполнятся проверка:   * Создана ли автозаправка * существует ли данная автозаправка. |
| public:  void RemoveZapravka() | Метод удаления автозаправки.  Перед удалением выполняется проверка на наличие автозаправки, бензоколонок в данной автозаправке. Если бензоколонки имеются в автозаправке, то сначала удаляются все бензоколонки, только потом удаляется сама автозаправка. |
| public:  void RemoveBenzoKolonka() | Метод удаления бензоколонки.  Перед удалением выполняется проверка на наличие автозаправки. |
| public:  void AllRemove() | Метод удаления всех автозаправок. |
| public:  void SaveToFile(string www) | Метод сохранения базы данных в файл “file.txt”. Если файл успешно открывается, то выполняется запись базы данных в файл, если не открылась, то создаётся новый файл. «www» указывает на перезапись файла или сохранение базы данных в конец файла. |
| public:  void DownloadFromFile() | Метод загрузки данных из текстового файла. После того, как вызвана функция выполняется проверка на существование этого файла, соответствия его расширения (расширение должно соответствовать .txt), и при успешном исходе считывается массив строк, из которого состоит файл. Далее происходит проверка синтаксиса загруженного текста. В случае ошибки, срабатывает исключение и выходит сообщение о соответствующей ошибке в текстовом файле. |
| public:  Zapravka\* AddDownload(string number) | Метод добавления автозаправок при загрузке данных из файла, возвращает указатель на только что добавленный элемент главного списка. Параметр number – номер автозаправки. |
| public:  void SearchZapravka() | Данный метод осуществляет поиск автозаправки по информационному полю (номеру автозаправки). |
| public:  void Show() | Метод отображения базы данных на консоли. |
| public:  string ClearStr(string str) | Метод удаления пробелов и табуляций. “str” – параметр (строка) из которого удаляются пробелы и табуляции. Метод возвращает строку, из которой удалены пробелы и табуляции. |
| public:  string Proverka(string str) | Метод проверки строки на наличие символов, не соответствующих цифрам. Параметр str – строка, которая проверяется. Метод возвращает пустую строку в случае наличия символов, не цифр, в строке. |
| public:  ~Content() | Деструктор данного класса. |

**Класс Zapravka**

Контейнерный класс, где содержится описание главной структуры «Динамический упорядоченный список» и её методов.

Табл.2. Класс Zapravka

|  |  |
| --- | --- |
| **Название метода** | **Описание метода** |
| public:  Zapravka(string nomer) | Конструктор данного класса, инициализирует информационное поле (номер автозаправки). Параметр nomer – номер автозаправки. |
| public  Section() | Конструктор данного класса. |
| public  void AddKolonka() | Метод добавления бензоколонки. |
| public  void RemoveKollonka() | Метод удаления бензоколонки.  Перед удалением выполняется проверка на наличие бензоколонок. |
| public  void Show() | Метод выводит на консоль информационные поля бензоколонок. |
| public  void SaveToFile () | Метод сохранения в файл “file.txt” всех бензоколонок в автозаправке. |
| public:  bool DonwloadAddKolonka(string nomer, string oil) | Метод добавления бензоколонок при загрузке базы данных из файла. Параметр nomer – номер автозаправки, а параметр oil – марка бензина. Возвращает true, в случае добавления автозаправки и false в противном случае. |
| public  void RemoveAllKollonka() | Метод удаления всех бензоколонок из автозаправки. |
| public  Zapravka\* GetNext() | Метод возвращает указатель на следующий элемент основного списка. |
| public  void SetNext(Zapravka \*temp) | Метод устанавливает указатель на следующий элемент основного списка. |
| public  void SetNumber(string num) | Метод устанавливает информационное поле автозаправки (номер автозаправки). |
| public  string GetNumber() | Метод возвращает номер автозаправки. |
| public:  string ClearStr(string str,int w) | Метод удаления пробелов и табуляций. “str” – параметр (строка) из которого удаляются пробелы и табуляции. Метод возвращает строку, из которой удалены пробелы и табуляции. Параметр “w” указывает на слияние слов, удалив все пробелы или оставив пробел между словами. |
| public:  string Proverka(string str) | Метод проверки строки на наличие символов, не соответствующих цифрам. Параметр str – строка, которая проверяется. Метод возвращает пустую строку в случае наличия символов, не цифр, в строке. |
| public:  ~Section() | Деструктор данного класса. |

**Класс BenzoKolonka**

Контейнерный класс, где содержится описание вспомогательной структуры «Статический стек» и её методов.

Табл.3. Класс BenzoKolonka

|  |  |
| --- | --- |
| **Название метода** | **Описание метода** |
| public  BenzoKolonka() | Конструктор данного класса. |
| public  void SetNomer(string nom) | Метод устанавливает информационное поле (номер) бензоколонки. Параметр nom – номер бензоколонки. |
| public  string GetNomer() | Метод возвращает строку – информационное поле (номер) бензоколонки. |
| public  void SetOil(string oill) | Метод устанавливает информационное поле (марка бензина) бензоколонки. Параметр oil – марка бензина. |
| public  string GetOil() | Метод возвращает строку – информационное поле (марка бензина) бензоколонки. |
| public  ~BenzoKolonka() | Деструктор данного класса. |

## **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Приложение использует консольное приложение для взаимодействия с пользователем. Запуск осуществляется через файл kursovaya\_diana.exe. Запустив программу, вы увидите оконное приложение следующего вида (рис. 3):

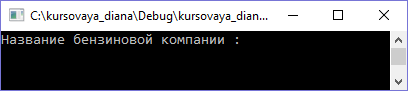


Рис 3.

После запуска программы, у пользователя запрашивается название бензиновой компании.

После этого на консоль выводится основное меню (Рис 4.):

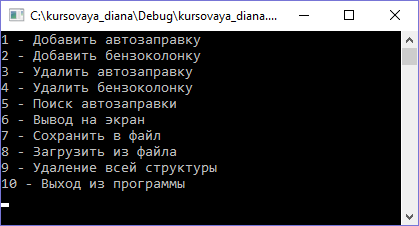


Рис 4.

## **Функции основного меню**

## Добавить автозаправку

Для добавления автозаправки необходимо ввести цифру 1 в главном меню. После этого нужно добавить отдел (Рис 5.) – необходимо ввести номер автозаправки. После того, как добавлен отдел, выводится на экран сообщение о добавлении (Рис. 6).

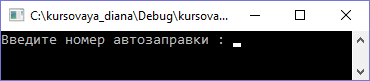


Рис. 5. Добавление автозаправки

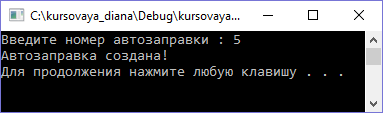


Рис. 6.

## Добавить бензоколонку

Для добавления бензоколонки надо ввести цифру 2 в главном меню и ввести номер автозаправки, в которую надо добавить бензоколонку (Рис. 7). В случае нахождения автозаправки с данным номером запрашивается информационные поля новой бензоколонки (Рис. 8), (Рис. 9) и выводится на консоль сообщение о добавлении автозаправки (Рис. 10). В противном случае выйдет сообщение об отсутсвии автозаправки с данным номером (Рис.11).

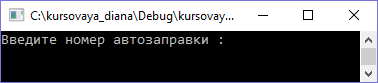


Рис.7.

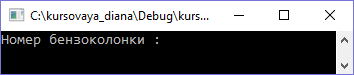


Рис.8.

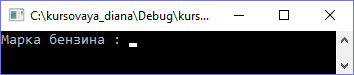


Рис.9.

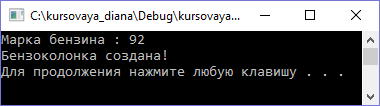


Рис. 10.

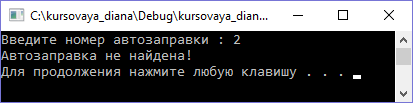


Рис. 11.

## Удалить автозаправку

Для удаления автозаправки, необходимо ввести цифру 3 в главном меню и ввести номер автозаправки (Рис. 7). В случае нахождения автозаправки, автозаправка удалится и выйдет на консоль сообщение об удалении (Рис. 12).

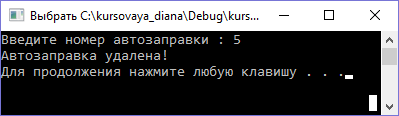


Рис.12.

## Удалить бензоколонку

Для удаления бензоколонки необходимо ввести цифру 4 в главном меню и ввести номер автозаправки, из которой будет удаляться бензоколонка (Рис. 7). В случае нахождения автозаправки с введённым номером и наличия бензоколонок, бензоколонка удалится и выведется сообщение об удалении (Рис. 13).

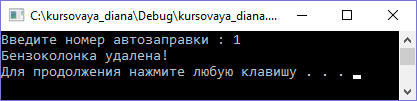


Рис.13.

## Поиск автозаправки

Для поиска автозаправки необходимо ввести цифру 5 в главном меню и ввести номер автозаправки (Рис. 7), которое надо найти. В результате поиска на консоль выйдет сообщение о нахождении (Рис. 14) или отсутствии данной автозаправки (Рис. 15).

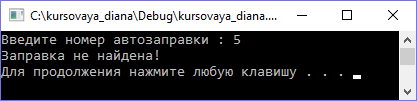


Рис.14.

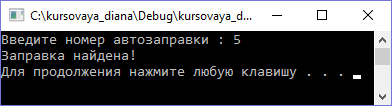


Рис.15.

## Вывод базы данных на консоль

Для вывода данных на консоль необходимо ввести цифру 6 в главном меню. После этого на консоль будет выведены информация о бензиновой компании, автозаправках и бензоколонках (Рис. 16).

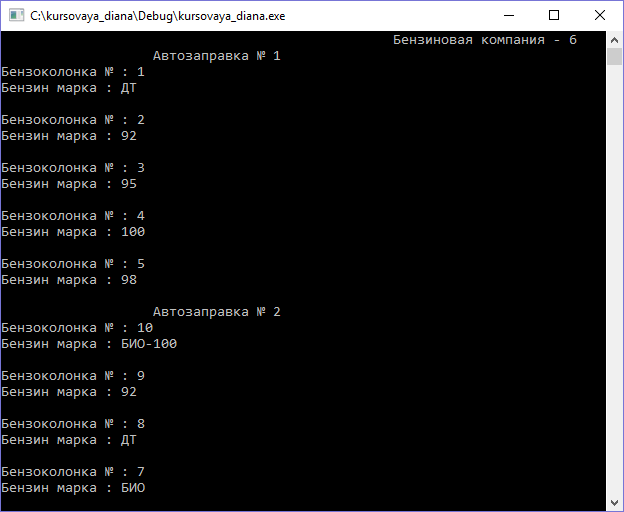


Рис. 16.

## Сохранение базы данных

Для сохранения созданной базы данных необходимо ввести цифру 7 в главном меню. После этого надо выбрать сохранение в конец файла или перезапись. В первом случае, созданная база данных сохранится в конец файла “file.txt”. В случае перезаписи, все данные из файла удалятся, и в файл “file.txt” сохранится созданная база данных. База данных – данные о автозаправках и бензоколонках. (Рис. 17.).

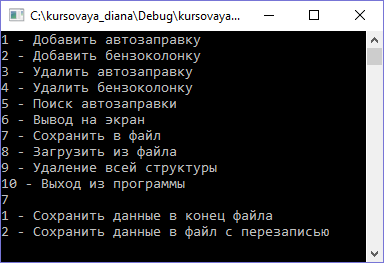


Рис.17.

## Загрузить базу

Для загрузки сохраненной базы данных необходимо ввести цифру 8 в главном меню и выбрать одну из следующих загрузок – загрузка данных из файла с сохранением созданной базы данных в файл или перезаписью и загрузка данных без сохранения. Для выбора надо ввести цифру 1 или 2 и 3 соответственно (Рис. 18). При выборе первого и второго пункта – база данных сохраняется в конец файла или осуществляется перезапись файла соответственно, а затем происходит загрузка данных из файла, а для пункта 3 – считываются данные из файла, при этом данные, которые имеются до начала загрузки, удаляются.

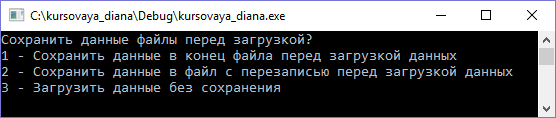


Рис.18.

Если файл повреждён или не существует, то выйдет соответствующая ошибка (например, представленная на рис. 19).

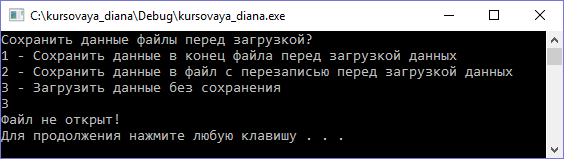


Рис.19.

## Удалить базу

Для того, чтобы удалить базу данных, необходимо в главном меню ввести цифру 9. База удалится, и выведется об этом сообщение (Рис. 20.).

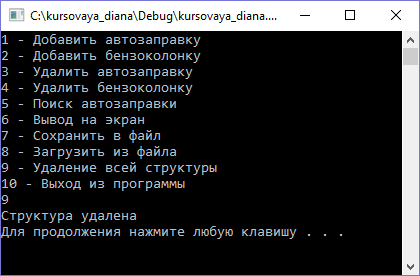


Рис. 20.

## **Сохранение структуры во внешнем txt файле**

Для добавления автозаправки пишем метку «Автозаправка:» и после неё в следующей строке пишем номер автозаправки. После метки должна стоять только одна автозаправка (номер автозаправки)! Для добавления нескольких автозаправок, необходимо для каждой автозаправки записать свою метку.

Автозаправка:

1

Для добавления бензоколонки пишем метку «Бензоколонка:» и записываем номер бензоколонки в одну строку, а марку бензина в другую. Для добавления нескольких бензоколонок необязательно писать каждый раз метку, можно после одной метки записать несколько бензоколонок. У каждой бензоколонки должны быть заданы оба информационный поля, в противном случае выйдет сообщение об этом (Рис. 27).

Бензоколонка:

5

98

4

100

3

95

Txt файл представлен на рисунке 21.

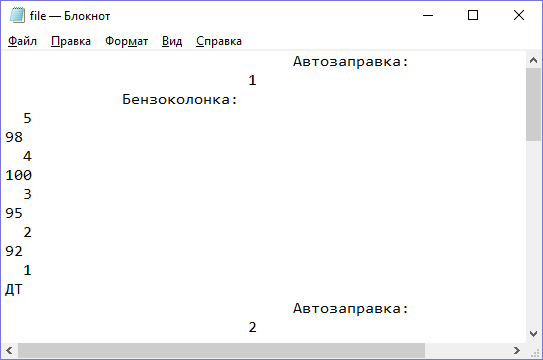


Рисунок 21.

## **РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММНОГО КОМПЛЕКСА**

## **Организация тестирования ПО**

Проводилось функциональное тестирование. Данное тестирование проводилось в целях проверки реализуемости функциональных требований, то есть способности ПО в определённых условиях решать задачи, нужные пользователям. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает.

## **Входные данные**

Номер автозаправки – цифры.

Номер бензоколонки – цифры.

Марка бензина бензоколонки – любые символы (кроме меток из пункта 4.2 и пустой строки).

При тестировании программы рассмотрены следующие ситуации (ошибки), возникающие во время использования программного комплекса, в других случаях программа выдает об успешности выполненной операции:

1. Число бензоколонок заполнена - выходит предупреждение, представленное на рис. 22.

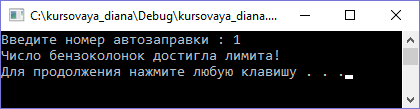


Рис.22.

1. Добавляется бензоколонка без создания автозаправки – выходит предупреждение, представленное на рис. 23.

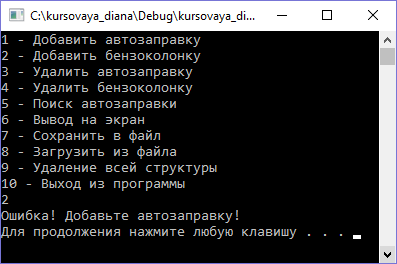


Рис.23.

1. При удалении автозаправки, если автозаправок не существует – выходит предупреждение, представленное на рис. 24.

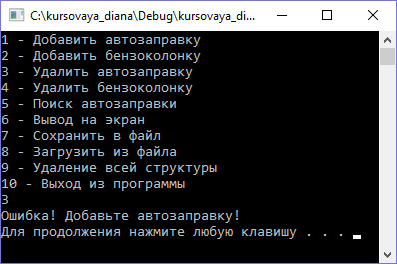


Рис.24.

1. При загрузке данных из файла данные находятся вне меток – выходит предупреждение, представленное на рис. 25.

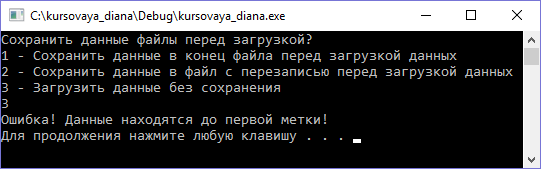


Рис. 25

1. При загрузке новой базы данных из текстового файла автозаправка не имеет своей метки – выходит предупреждение представленный на рис. 26.

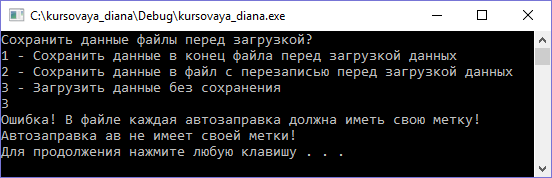


Рис. 26.

1. При загрузке новой базы данных из текстового файла бензоколонка не имеет марку бензина – выходит предупреждение представленное на рис. 27.

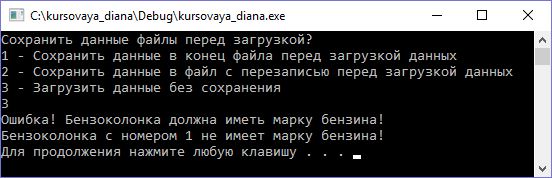


Рис. 27.

1. При загрузке новой базы данных из текстового файла в номере автозаправки находятся символы или буквы – выходит предупреждение представленный на рис. 28.

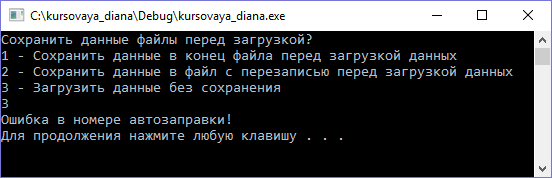


Рис. 28.

## **ЗАКЛЮЧЕНИЕ**

Полученная структура данных обладает всем требуемым функционалом, который был определен в постановке задачи. Оконное 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.
2. Вирт Н. Алгоритмы и структуры данных. – Изд. Невский Диалект, 2001 г.

## 

## **ПРИЛОЖЕНИЕ**

Листинг программы:

**kursovaya\_diana.cpp**

#include "stdafx.h"

#include <iostream>

#include <cstring>

#include <cstdlib>

#include <conio.h>

#include <string>

#include <windows.h>

#include "Company.h"

using namespace std;

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

string name;// название организации

string WMI = "";

int stop = 0;

do {

system("cls");

do//проверка на пустоту

{

system("cls");

cout << "Название бензиновой компании : ";

getline(cin, name);

bool flag = false;

for (int j = 0; j < name.length(); j++)

{

if (name.at(j) != ' ') { flag = true; }

}

if (flag == true) {

stop = 1;//1 - символ есть

}

else { stop = -1; }//-1 - символа нету

} while (name == "" || stop == -1);

Company comp = Company(name);//число отделов и название организации

bool stopp = true;//пока не выйдет

while (stopp != false)

{

system("cls");

cout << "1 - Добавить автозаправку \n2 - Добавить бензоколонку \n3 - Удалить автозаправку \n4 - Удалить бензоколонку \n5 - Поиск автозаправки \n6 - Вывод на экран \n7 - Сохранить в файл \n8 - Загрузить из файла\n9 - Удаление всей структуры \n10 - Выход из программы" << endl;

string ask;

getline(cin, ask);

WMI = ask;

if (ask == "1") {

comp.AddZapravka();

}

else if (ask == "2") {

comp.AddBenzokolonka();

}

else if (ask == "3") {

comp.RemoveZapravka();

}

else if (ask == "4") {

comp.RemoveBenzoKolonka();

}

else if (ask == "5") {

comp.SearchZapravka();

}

else if (ask == "6") {

system("cls");

comp.Show();

}

else if (ask == "7") {

string r = "0";

cout << "1 - Сохранить данные в конец файла" << endl << "2 - Сохранить данные в файл с перезаписью" << endl;

getline(cin, r);

comp.SaveToFile(r);

cout << "Данные сохранены в файл" << endl;

system("pause");

}

else if (ask == "8") {

string a;

system("cls");

cout << "Сохранить данные файлы перед загрузкой?" << endl;

cout << "1 - Сохранить данные в конец файла перед загрузкой данных " << endl << "2 - Сохранить данные в файл с перезаписью перед загрузкой данных " << endl << "3 - Загрузить данные без сохранения" << endl;

getline(cin, a);

if (a == "1" || a == "2")

{

comp.SaveToFile(a);

comp.DownloadFromFile();

}

if (a == "3")

{

comp.DownloadFromFile();

}

}

else if (ask == "9") {

comp.~Company();

cout << "Структура удалена" << endl;

system("pause");

stopp = false;

}

else if (ask == "10") {

comp.~Company();

stopp = false;

}

}

} while (WMI != "10");

return 0;

}

**Company.h**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "stdafx.h"

#include <fstream>

#include <iostream>

#include <cstdlib>

#include <cstring>

#include <conio.h>

#include <string>

#include <windows.h>

#include "Zapravka.h"

using namespace std;

#pragma once

class Company

{

private:

string name;

Zapravka \*zapravka;

public:

void DownloadFromFile() //загрузка из файла

{

AllRemove();

Zapravka \*Set = NULL;

int k = 0;

int flag = 0;

int number = -1;

int key = 0;

string h = "";

ifstream txt("file.txt");

if (txt.is\_open()) // вызов метода is\_open()

{

getline(txt, h);

h = ClearStr(h);

while (key != 5)

{

if (h == "Автозаправка:") {

flag = 0;

if (!txt.eof()) {

getline(txt, h);

h = ClearStr(h);

}

if (!txt.eof()) {

int g1 = 1;

if (h.empty() || h == "") { g1 = 0; }

while (g1 != 1)

{

if (h.empty() || h == "") {

if (!txt.eof()) {

getline(txt, h); h = ClearStr(h);

}

else { g1 = 1; }

}

else { g1 = 1; }

}

if (txt.eof()) { key = 5; }

if (h != "Автозаправка:"&&h != "Бензоколонка:" && !h.empty())

{

h = Proverka(h);

if (h == "") { cout << "Ошибка в номере автозаправки!" << endl;

txt.close(); system("pause"); return; }

flag = 1;

Set = AddDownload(h);

if (Set == NULL) {

txt.close();

return; }

if (!txt.eof()) {

getline(txt, h);

h = ClearStr(h);

}

else { key = 5; }

int g1 = 1;

if (h.empty() || h == "") { g1 = 0; }

while (g1 != 1)

{

if (h.empty() || h == "") {

if (!txt.eof()) {

getline(txt, h);

h = ClearStr(h);

}

else { g1 = 1; }

}

else { g1 = 1; }

}

if (txt.eof()) { key = 5; }

if (h != "Автозаправка:" && h != "Бензоколонка:" && !h.empty())

{

cout << "Ошибка! В файле каждая автозаправка должна иметь свою метку!" << endl << "Автозаправка " << h << " не имеет своей метки!" << endl;

system("pause");

txt.close();

return;

}

}

}

else { key = 5; }

}

else if (h == "Бензоколонка:") {

string h1;

int a = 0;

if (flag != 1) { a = 1; key = 5; cout << "Не указана автозаправка!" << endl; system("pause"); }

while (a != 1) {

if (!txt.eof()) {

getline(txt, h);

h = ClearStr(h);

}

if (!txt.eof()) {

int g = 1;

if (h.empty() || h == "") { g = 0; }

while (g != 1)

{

if (h.empty() || h == "") {

if (!txt.eof()) {

getline(txt, h);

h = ClearStr(h);

}

else g = 1;

}

else { g = 1; }

if (txt.eof()) { g = 1; }

}

if ((txt.eof()) || h == "Автозаправка:" || h == "Бензоколонка:") { a = 1; }

else {

if (!txt.eof()) {

getline(txt, h1);

h1 = ClearStr(h1);

}

int g1 = 1;

if (h1.empty() || h == "") { g1 = 0; }

while (g1 != 1)

{

if (txt.eof()) {

g1 = 1;

a = 1;

key = 5;

}

if (h1.empty() || h == "") {

if (!txt.eof()) {

getline(txt, h1);

h1 = ClearStr(h1);

}

else g1 = 1;

}

else { g1 = 1; }

}

if (txt.eof()) { key = 5; a = 1; }

if (h1 != "Автозаправка:" && h1 != "Бензоколонка:" && !h1.empty() && !(h == ""))

{

string ggg = h;

h = Proverka(h);

if (h == "") { cout << "Ошибка в номере колонки! ---- " <<ggg<< endl;

txt.close(); system("pause"); return; }

if (Set->DonwloadAddKolonka(h, h1) == false) { txt.close(); return; }

}

else {

a = 1;

cout << "Ошибка! Бензоколонка должна иметь марку бензина!" << endl << "Бензоколонка с номером " << h << " не имеет марку бензина!" << endl;

system("pause");

txt.close();

return;

}

}

}

else { a = 1; key = 5; }

}

if (txt.eof()) { key = 5; a = 1; }

}

if (txt.eof()) {

key = 5;

txt.close();

return;

}

if (h.empty()) {

if (!txt.eof()) {

getline(txt, h);

h = ClearStr(h);

}

}

if (h != "Автозаправка:" && h != "Бензоколонка:" && !h.empty())

{

bool ad = false;

for (int j = 0; j < h.length(); j++)

{

if (h.at(j) != ' ') { ad = true; }

}

if (ad == true) {

cout << "Ошибка! Данные находятся до первой метки!" << endl;

system("pause");

txt.close();

return;

}

else {

if (!txt.eof()) {

getline(txt, h);

h = ClearStr(h);

}

}

}

}

txt.close();

}

else

{

cout << "Файл не открыт!" << endl;

system("pause");

return;

}

}

void AllRemove() {

while (zapravka->GetNext() != NULL) {

Zapravka \*temp = zapravka->GetNext();

zapravka = zapravka->GetNext();

temp->~Zapravka();

}

}

void SaveToFile(string www) { //сохранить в файл

if (www == "2")

{

remove("file.txt");

}

fstream txt("file.txt", ios::app); if (txt.is\_open()) // вызов метода is\_open()

{

Zapravka \*temp = zapravka->GetNext();

while (temp != NULL) {

txt << " Автозаправка:" << endl;

txt << " " << temp->GetNumber() << endl;

temp->SaveToFile();

temp = temp->GetNext();

}

txt.close();

return;

}

else

{

cout << "Файл не открыт!\n\n" << endl;

system("pause");

return;

}

}

Company(string name1) {

name = name1;

zapravka = new Zapravka();

}

void SearchZapravka() {

string ask = "";

if (zapravka->GetNext() != NULL) {

do {

system("cls");

cout << "Введите номер автозаправки : ";

getline(cin, ask);

ask = Proverka(ClearStr(ask));

} while (ask == "");

Zapravka \*temp = zapravka->GetNext();

while (temp != NULL) {

if (temp->GetNumber() == ask) {

cout << "Заправка найдена!" << endl;

system("pause");

return;

}

temp = temp->GetNext();

}

cout << "Заправка не найдена!" << endl;

system("pause");

}

}

void RemoveZapravka() {

string ask = "";

if (zapravka->GetNext() != NULL) {

do {

system("cls");

cout << "Введите номер автозаправки : ";

getline(cin, ask);

ask = Proverka(ClearStr(ask));

} while (ask == "");

Zapravka \*temp = zapravka->GetNext();

Zapravka \*temp1 = zapravka;

while (temp != NULL) {

if (temp->GetNumber() == ask) {

temp->RemoveAllKollonka();

temp1->SetNext(temp->GetNext());

delete temp;

cout << "Автозаправка удалена!" << endl;

system("pause");

return;

}

temp = temp->GetNext();

temp1 = temp1->GetNext();

}

cout << "Автозаправка с данным номером не найдена!" << endl;

system("pause");

return;

}

else {

cout << "Ошибка! Добавьте автозаправку!" << endl;

system("pause");

}

}

void RemoveBenzoKolonka() {

string ask = "";

if (zapravka->GetNext() != NULL) {

do {

system("cls");

cout << "Введите номер автозаправки : ";

getline(cin, ask);

ask = Proverka(ClearStr(ask));

} while (ask == "");

Zapravka \*temp = zapravka->GetNext();

while (temp != NULL) {

if (temp->GetNumber() == ask) {

temp->RemoveKollonka();

return;

}

temp = temp->GetNext();

}

cout << "Автозаправка не найдена!" << endl;

system("pause");

}

else {

cout << "Ошибка! Добавьте автозаправку!" << endl;

system("pause");

}

}

string ClearStr(string str) { //преобразовать строку

if (str.size() > 5000) {

cout << "Ошибка! Вводимые символы не должны быть больше 1000!" << endl;

system("pause");

return "Error";

}

string endword;

char chars[5000];

strcpy\_s(chars, str.c\_str());

char \*nextword = strtok(chars, " \t");

while (nextword != NULL)

{

endword += nextword;

nextword = strtok(NULL, " \t");

}

return endword;

}

string Proverka(string str) {

if (str.empty() || str == "") { return ""; }

for (int i = 0; i < str.size(); i++) {

if (str.at(i) != '1' && str.at(i) != '0' && str.at(i) != '2' && str.at(i) != '3' && str.at(i) != '4' && str.at(i) != '5' && str.at(i) != '6' && str.at(i) != '7' && str.at(i) != '8' && str.at(i) != '9')

{

return "";

}

}

if (str.size() > 1) {

if (str.at(0) == '0'&&(str.at(0) == '0'|| str.at(0) == '1'|| str.at(0) == '2'|| str.at(0) == '3'|| str.at(0) == '4'|| str.at(0) == '5'|| str.at(0) == '6'|| str.at(0) == '7'|| str.at(0) == '8'|| str.at(0) == '9')) { return ""; }

}

return str;

}

void Show() {

cout <<" Бензиновая компания - "<< name << endl;

if (zapravka->GetNext() != NULL) {

Zapravka \*temp = zapravka;

while (temp->GetNext() != NULL) {

cout << " Автозаправка № " << temp->GetNext()->GetNumber() << endl;

temp->GetNext()->Show();

temp = temp->GetNext();

}

}

system("pause");

}

void AddBenzokolonka() {

string ask = "";

if (zapravka->GetNext() != NULL) {

do {

system("cls");

cout << "Введите номер автозаправки : ";

getline(cin, ask);

ask = Proverka(ClearStr(ask));

} while (ask == "");

Zapravka \*temp = zapravka->GetNext();

while (temp != NULL) {

if (temp->GetNumber() == ask) {

temp->AddKolonka();

return;

}

temp = temp->GetNext();

}

cout << "Автозаправка не найдена!" << endl;

system("pause");

}

else {

cout << "Ошибка! Добавьте автозаправку!" << endl;

system("pause");

}

}

void AddZapravka() {

string number = "";

do {

system("cls");

cout << "Введите номер автозаправки : ";

getline(cin, number);

number = Proverka(ClearStr(number));

} while (number == "");

Zapravka \*temp = new Zapravka(number);

if (zapravka->GetNext() == NULL) {

zapravka->SetNext(temp);

cout << "Автозаправка создана!" << endl;

system("pause");

return;

}

else {

Zapravka \*add = zapravka->GetNext();

Zapravka \*pred = zapravka;

string poisk = add->GetNumber();

while (add != NULL&& number.length()>=poisk.length()) {

string poisk = add->GetNumber();

if (poisk.length() > number.length()) {

temp->SetNext(add);

pred->SetNext(temp);

cout << "Автозаправка создана!" << endl;

system("pause");

return;

}

if (poisk.length() < number.length()) {

if (add->GetNext() == NULL) {

add->SetNext(temp);

cout << "Автозаправка создана!" << endl;

system("pause");

return;

}

}

if (number == poisk) {

cout << "Ошибка! Такая автозаправка уже существует!" << endl;

system("pause");

return;

}

if (poisk.length() == number.length() && poisk > number) {

temp->SetNext(add);

pred->SetNext(temp);

cout << "Автозаправка создана!" << endl;

system("pause");

return;

}

add = add->GetNext();

pred = pred->GetNext();

}

pred->SetNext(temp);

temp->SetNext(add);

cout << "Автозаправка создана!" << endl;

system("pause");

}

}

Zapravka\* AddDownload(string number) {

Zapravka \*temp = new Zapravka(number);

if (zapravka->GetNext() == NULL) {

zapravka->SetNext(temp);

return temp;

}

else {

Zapravka \*add = zapravka->GetNext();

Zapravka \*pred = zapravka;

string poisk = add->GetNumber();

while (add != NULL&& number.length() >= poisk.length()) {

string poisk = add->GetNumber();

if (poisk.length() > number.length()) {

temp->SetNext(add);

pred->SetNext(temp);

return temp;

}

if (poisk.length() < number.length()) {

if (add->GetNext() == NULL) {

add->SetNext(temp);

return temp;

}

}

if (number == poisk) {

cout << "Ошибка! Такая автозаправка уже существует!" << endl;

system("pause");

return NULL;

}

if (poisk.length() == number.length() && poisk > number) {

temp->SetNext(add);

pred->SetNext(temp);

return temp;

}

add = add->GetNext();

pred = pred->GetNext();

}

pred->SetNext(temp);

temp->SetNext(add);

return temp;

}

}

Company() {}

~Company() {

while (zapravka->GetNext() != NULL) {

Zapravka \*temp = zapravka->GetNext();

zapravka = zapravka->GetNext();

temp->SetNext(NULL);

temp->~Zapravka();

}

zapravka->~Zapravka();

}

};

**Zapravka.h**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "stdafx.h"

#include <fstream>

#include <iostream>

#include <cstdlib>

#include <cstring>

#include <istream>

#include <conio.h>

#include <string>

#include <windows.h>

#include "BenzoKolonka.h"

#pragma once

using namespace std;

class Zapravka

{

private:

Zapravka \*next;

string number;

BenzoKolonka \*kolonka;

int count;

public:

~Zapravka() {

number = "";

for (int i = 0; i < count; i++) {

kolonka[i].~BenzoKolonka();

}

}

Zapravka(string nomer) {

number = nomer;

kolonka = new BenzoKolonka[5];

count = 0;

next = NULL;

}

string ClearStr(string str,int w) { //преобразовать строку

if (str.size() > 5000) {

cout << "Ошибка! Вводимые символы не должны быть больше 5000!" << endl;

system("pause");

return "Error";

}

string endword;

char chars[5000];

strcpy\_s(chars, str.c\_str());

char \*nextword = strtok(chars, " \t");

while (nextword != NULL)

{

endword += nextword;

nextword = strtok(NULL, " \t");

if (nextword != NULL && w == 1)

endword += ' ';

}

return endword;

}

string Proverka(string str) {

if (str.empty() || str == "") { return ""; }

for (int i = 0; i < str.size(); i++) {

if (str.at(i) != '1' && str.at(i) != '0' && str.at(i) != '2' && str.at(i) != '3' && str.at(i) != '4' && str.at(i) != '5' && str.at(i) != '6' && str.at(i) != '7' && str.at(i) != '8' && str.at(i) != '9')

{

return "";

}

}

if (str.size() > 1) {

if (str.at(0) == '0' && (str.at(0) == '0' || str.at(0) == '1' || str.at(0) == '2' || str.at(0) == '3' || str.at(0) == '4' || str.at(0) == '5' || str.at(0) == '6' || str.at(0) == '7' || str.at(0) == '8' || str.at(0) == '9')) { return ""; }

}

return str;

}

void RemoveAllKollonka() {

if (count > 0) {

for(int i=count;i>0;i--) {

kolonka[count-1].SetNomer("");

kolonka[count-1].SetOil("");

}

count = 0;

}

delete[] kolonka;

}

void SaveToFile() {

{

fstream txt("file.txt", ios::app); if (txt.is\_open()) // вызов метода is\_open()

{

if (count > 0)

{

txt << " Бензоколонка:" << endl;

for (int i = 0; i < count; i++)

{

txt << " " << kolonka[i].GetNomer() << endl;

txt << "" << kolonka[i].GetOil() << endl;

}

}

txt.close();

}

else

{

cout << "Файл не открыт!\n\n" << endl;

system("pause");

return;

}

}

}

bool DonwloadAddKolonka(string nomer, string oil) {

if (count < 5) {

kolonka[count].SetNomer(nomer);

kolonka[count].SetOil(oil);

count++;

return true;

}

else {

cout << "Число бензоколонок достигла лимита!" << endl;

system("pause");

return false;

}

}

void RemoveKollonka() {

if (count > 0) {

kolonka[count-1].SetNomer("");

kolonka[count-1].SetOil("");

count--;

cout << "Бензоколонка удалена!" << endl;

system("pause");

}

else {

cout << "Ошибка! Бензоколонок нет!" << endl;

system("pause");

}

}

void AddKolonka() {

if (count < 5) {

string nomer, oil;

do {

system("cls");

cout << "Номер бензоколонки : ";

getline(cin, nomer);

nomer = Proverka(ClearStr(nomer,0));

} while (nomer == "");

do {

system("cls");

cout << "Марка бензина : ";

getline(cin, oil);

oil = ClearStr(oil,1);

} while (oil == "" || oil == "Автозаправка:" || oil == "Бензоколонка:");

kolonka[count].SetNomer(nomer);

kolonka[count].SetOil(oil);

count++;

cout << "Бензоколонка создана!" << endl;

system("pause");

}

else {

cout << "Число бензоколонок достигла лимита!" << endl;

system("pause");

}

}

Zapravka() { next = NULL; kolonka = NULL; }

void Show() {

for (int i = count - 1; i >= 0; i--) {

cout << "Бензоколонка № : " <<kolonka[i].GetNomer()<< endl;

cout << "Бензин марка : " << kolonka[i].GetOil() << endl << endl;

}

}

Zapravka\* GetNext() {

return next;

}

void SetNext(Zapravka \*temp) {

next = temp;

}

string GetNumber() {

return number;

}

void SetNumber(string num) {

number = num;

}

};

**BenzoKolonka.h**

#include "stdafx.h"

#include <iostream>

#include <string>

#pragma once

using namespace std;

class BenzoKolonka

{

private:

string nomer, oil;

public:

BenzoKolonka() {}

~BenzoKolonka() {

nomer = "";

oil = "";

}

void SetNomer(string nom) {

nomer = nom;

}

string GetNomer() {

return nomer;

}

void SetOil(string oill) {

oil = oill;

}

string GetOil() {

return oil;

}

};

Worker\* GetRight() { return right; }

void SetRight(Worker \*ptr) { right = ptr; }

string GetMayor() { return mayor; }

void SetMayor(string may) { mayor = may; }

Worker();

~Worker();

};