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**: структурная реализация контейнера

на основе динамического упорядоченного двунаправленного списка

неупорядоченных двунаправленных динамических списков

Выполнил:   
 студент группы 4212  
Фенагеев С.С.

Проверил:   
 Доцент каф. ПМИ к.т.н.

Сотников С. В.

Казань 2020

**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 [1]:

* 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 двунаправленный список неупорядоченных двунаправленных динамических списков (вариант 90, 9) на основе объектного подхода.

Структура должна иметь следующее информационное наполнение: Магазин (название) – Композиция отделов (наименование) – Отдел – Композиция менеджеров по продаже (Фамилия, Зарплата)

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.

**Ожидаемый результат**

Ожидаемым результатом выполнения курсовой работы является разработанная иерархия классов, которая включает в себя коллекции и классы, функции и процедуры для работы с этими структурами, реализующие весь необходимый функционал.

## **ОПИСАНИЕ СТРУКТУРЫ ДАННЫХ ДИНАМИЧЕСКИЙ УПОРЯДОЧЕННЫЙ ДВУНАПРАВЛЕННЫЙ СПИСОК ДИНАМИЧЕСКИХ НЕУПОРЯДОЧННЫХ ДВУНАПРАВЛЕННЫХ СПИСКОВ**

### **2.1 ДИНАМИЧЕСКИЙ УПОРЯДОЧЕННЫЙ ДВУНАПРАВЛЕННЫЙ СПИСОК**

Недостатком однонаправленных списковых структур является их однонаправленность от первого элемента к последнему. Если при обработке списков часто бывает необходимо переходить от текущего элемента к его предшественнику, то такая односторонняя организация становится неудобной. Выходом является построение **двунаправленных** списков, в которых каждый элемент “знает” **обоих** своих соседей, как левого, так и правого [1].

Для этого каждый элемент должен иметь не одно, а два связующих поля: указатель на элемент слева и указатель на элемент справа.

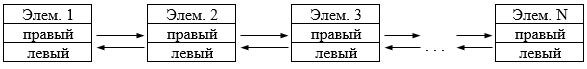


Рисунок 1 Схема двунаправленного списка

Аналогично обычным спискам, удобно ввести фиктивный заголовочный элемент, а поскольку двунаправленные списки являются симметричными, удобно сделать их замкнутыми, кольцевыми: правая ссылка последнего элемента указывает на заголовок, а левая ссылка заголовка – на последний элемент. Адрес заголовка определяется указателем pHead [1]..

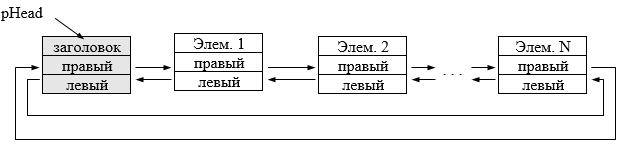


Рисунок 2Схема двунаправленного кольцевого списка

Отметим достоинства и недостатки двунаправленных списков. Достоинство – простота перехода от текущего элемента к любому его соседу. Недостатки – увеличиваются затраты памяти и число операций на поддержку дополнительного указателя [1]..

Двунаправленный список можно реализовать как на основе массива (причем – обеими способами), так и динамически.

**Создание пустого списка** [1].**:**

* выделяется память под заголовок, адресуемая указателем pHead;
* оба ссылочных поля заголовка устанавливаются в адрес самого;

**Проход по списку** [1].**:**

• ввести вспомогательную переменную указатель для отслеживания текущего элемента списка и установить указатель на следующий элемент заголовка

• организовать цикл по условию, внутри которого обработать текущий элемент и изменить указатель на следующий элемент

**Добавление после** **заданного** [1].**:**

* выделение памяти для нового добавляемого
* устанавливаем правый и левый указатель добавляемого элемента
* устанавливаем левый указатель у правого соседа добавляемого и правый указатель заданного элемента

**Добавление перед заданным** [1].**:**

* Аналогично, только вместо правого соседа обрабатывается левый сосед текущего элемента.

**Удаление** **заданного элемента** [1].**:**

* изменить правый указатель у левого соседа удаляемого элемента
* изменить указатель у правого соседа



Рисунок 3Схема удаления элемента

**Поиск заданного элемента** [1].**:**

* установка вспомогательного указателя в адрес первого элемента списка
* организация 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

### **2.2 ДИНАМИЧЕСКИЙ НЕУПОРЯДОЧЕННЫЙ ДВУНАПРАВЛЕННЫЙ СПИСОК**

Пусть в очереди требуется сохранять целые числа, причем заранее известно их максимальное количество. Тогда для реализации очереди надо объявить массив и две переменные – указатель начала очереди First и указатель конца очереди Last. Будем считать, что очередь-массив заполняется (растет) от первых элементов массива к последним. Тогда указатель First будет определять первую **занятую** ячейку массива, а указатель Last - первую **свободную** ячейку. Тогда пустую очередь определим как First = Last = 1 (если индексация элементов массива начинается с 1), и при каждом добавлении нового элемента переменная Last увеличивается на 1, а при удалении на 1 увеличивается указатель First. Последовательность операций для массива из пяти элементов показана на следующей схеме [1].:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

1. Пустая очередь: First = Last =1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **15** |  |  |  |  |

2. Добавлено первое число 15, First = 1, Last =2 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 15 | **33** |  |  |  |

3. Добавлено второе число 33, First = 1, Last = 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 15 | 33 | **07** |  |  |

4. Добавлено третье число 07, First = 1, Last = 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 33 | **07** |  |  |

5. Удалено число 15, First = 2, Last = 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **07** |  |  |

6. Удалено число 33, First = 3 , Last = 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 07 | **44** |  |

7. Добавлено число 44, First = 3 , Last = 5

Удобно использовать так называемую кольцевую очередь, для того, чтобы после удаления ячейка использовалась в дальнейшей работе программы. при достижении указателем Last конца массива добавление производится в начало массива:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 07 | 44 | **11** |

8. Добавлено число 11, First = 3 , Last = 6 (= 1)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **22** |  | 07 | 44 | 11 |

1. Новое число 22 добавляется в первую ячейку:

First = 3 , Last = 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 22 | **99** | 07 | 44 | 11 |

10. Добавлено число 99, First = 3 , Last = 3

В этом случае добавление становится невозможным только если в массиве нет ни одной свободной ячейки, как в рассмотренном примере.

Для программной реализации удобно ввести переменную-счетчик числа элементов в очереди, с помощью которой легко отслеживаются состояния пустой и заполненной очереди.

**Проход по очереди** [1].**:**

* Проверка очереди на наличие элементов
* Ввести вспомогательную переменную с адресом First
* Организовать цикл с выходом по достижении переменной указателя Last

**Добавление элемента** [1].

* проверить возможность добавления (в массиве есть свободные ячейки?)
* добавить элемент в массив по индексу Last
* изменить указатель Last на 1
* если Last выходит за пределы массива, то установить Last в 1
* увеличить счетчик числа элементов в очереди

**Удаление элемента** из очереди [1].:

* проверить возможность удаления (в очереди есть элементы?)
* извлечь элемент из массива по индексу First и выполнить с ним необходимые действия
* увеличить указатель First на 1
* если First выходит за пределы массива, то установить First в 1
* уменьшить счетчик числа элементов в очереди

### **2.3 КОМБИНИРОВАННАЯ СТРУКТУРА ДАННЫХ «ДИНАМИЧЕСКИЙ НЕУПОРЯДОЧЕННЫЙ ДВУНАПРАВЛЕННЫЙ СПИСОК СТАТИЧЕСКИХ ОЧЕРЕДЕЙ»**

Рассмотренные выше способы объединения элементов могут комбинироваться друг с другом, образуя достаточно сложные структуры. Один из возможных случаев такого объединения – список указателей на базовый тип. Каждый элемент такого списка содержит два поля: указатель на соседний элемент и указатель на базовый объект. Поскольку структура базового объекта отличается от структуры элемента списка, их надо описывать отдельно и вводить два ссылочных типа данных [1]..

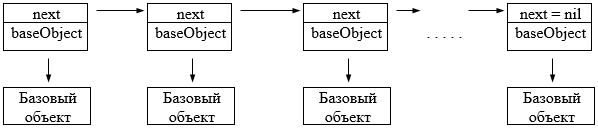


Рисунок 4 Схема списка указателей на базовый объект

В нашем случае список двунаправленный.

## **3 РУКОВОДСТВО ПРОГРАММИСТА**

3.1 ОПИСАНИЕ СТРУКТУРЫ ПРОГРАММЫ

Для разработки проекта была использована среда разработки программного обеспечения Visual Studio 2020.

Весь программный код, включая основную программу, хранится в файлах .cpp. Разработанная иерархия классов, представленная на рисунке 3, включает в себя коллекции и классы, отвечающие за информационное заполнение, а также функции и процедуры, предназначенные для работы с созданными структурами данных.

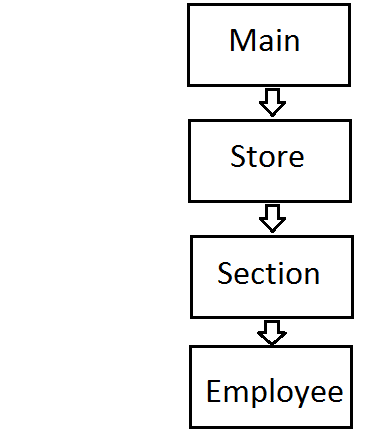


Рисунок 5 Иерархия классов

## **3.2 ОПИСАНИЕ СТРУКТУР**

Контейнер представляет собой комбинированную структуру. Главная структура – динамический двунаправленный список. Дочерняя структура – статическая очередь.

Разработаны две основные структуры, представленные в следующих контейнерах.

class Store {

private:

string \*storeName;

public:

Section\* Head;

storeName – информационное поле содержащее название магазина

Head – заголовочный элемент списка

class Section {

private:

string\* name;

Employee\* Mass;

Employee\* First;

Employee\* Last;

int Counter = 0;

public:

Section\* Next;

Section\* Previous;

name – название отдела  
mass – массив, каждый элемент которого содержит фамилию и зарплату сотрудника  
First – указатель на первый элемент очереди  
Last – указатель на первую пустую ячейку очереди  
Counter – счетчик числа элементов  
Next – указатель на следующий элемент связного списка  
Previous – указатель на предыдущий элемент связного списка

## **3.3 ОПИСАНИЕ ВСЕХ РАЗРАБОТАННЫХ ПОДПРОГРАММ ДЛЯ РАБОТЫ СО СТРУКТУРАМИ**

**Класс Main.cpp**

Описание класса, который реализует взаимодействие с пользователем через консоль.

|  |  |
| --- | --- |
| **Название метода** | **Описание** |
| void NewStore(); | Пункт «Новый магазин». Если магазин был ранее объявлен, удаляет всю базу данных.  Делает запрос названия магазина у пользователя и создает новую базу данных. |
| void AddSection(); | Пункт «Добавить отдел». Перед выполнением всех действий осуществляет проверки:   * создана ли база данных * пустой ли список, если да, то запрашивает первый элемент списка. Иначе, запрашивает добавить до/после заданного   Если все проверки пройдены осуществляется добавление. |
| void AddEmployee(); | Пункт «Добавить сотрудника». Перед выполнением всех действий осуществляет проверки:   * создана ли база данных * существует ли указанный отдел в списке   Если все проверки пройдены осуществляется добавление. |
| void Show(); | Пункт «Вывод». Перед выполнением всех действий осуществляет проверки:   * создана ли база данных * не пустой ли список   Если все проверки пройдены осуществляется вывод всей базы данных. |
| void DeleteSection(); | Пункт «Удалить отдел». Перед выполнением всех действий осуществляет проверки:   * создана ли база данных * пустой ли список, если нет, то запрашивает название удаляемого отдела   Если отдел существует в списке, производится удаление. |
| void DeleteEmployee(); | Пункт «Удалить отдел». Перед выполнением всех действий осуществляет проверки:   * создана ли база данных * пустой ли список, если нет, то запрашивает фамилию и зарплату удаляемого сотрудника   Если все сотрудник существует в списке, производится удаление. |
| void Upload(); | Пункт «Загрузка из файла». Перед выполнением всех действий осуществляет проверки:   * создана ли база данных   Производит загрузку данных из файла “In.txt” |
| void DownLoad(); | Пункт «Выгрузка в файл». Перед выполнением всех действий осуществляет проверки:   * создана ли база данных   Производит выгрузку данных из файла “Out.txt” |
| void SearchSection(); | Пункт «Поиск отдела». Перед выполнением всех действий осуществляет проверки:   * создана ли база данных   Запрашивает назание отдела для поиска и выводит результат поиска по списку данного отдела. |
| void SearchEmployee(); | Пункт «Поиск сотрудника». Перед выполнением всех действий осуществляет проверки:   * создана ли база данных   Запрашивает фамилию сотрудника и возвращает название отдела, в котором сотрудник найден. |

**Store.cpp**

|  |  |
| --- | --- |
| **Название метода** | **Описание** |
| void AddSectionAfter(Section\* ElementAfterWhAdd, string SectionName) | Производит добавление нового отдела после указанного, если список инициализирован. |
| void SetFirst(string SectionName) | Добавляет первый элемент списка, если тот инициализирован. |
| void AddSub(string SectionToAddName, string Surname, int Pay) | Добавляет нового сотрудника в указанный отдел, если этот отдел есть в списке. |
| void Delete(string SectionToDeleteName) | Удаляет заданный отдел, если список инициализирован. |
| void DeleteSub(string SectionToDeleteEmp) | Удаляет сотрудника в указанном отделе. |
| void Show() | Вывод всей базы данных, если она создана. |
| Section\* SearchForID | Производит поиск указанного отдела и возвращает указатель на него, если он есть в списке. |
| Section\* GlobalSearch(string SearchSurName) | Поиск сотрудника, возвращает указатель на отдел, если в этом отделе найден сотрудник. |
| void DownLoad() | Загрузка базы данных из файла. |
| void UpLoad() | Выгрузка базы данных в файл. |
| Store(string StoreName) | Конструктор класса. |
| ~Store() | Деструктор класса, удаляет всю выделенную ранее под объект память. |
| void SetStoreName(string StoreName) | Задает название магазина. |
| string GetStoreName() | Возвращает название магазина. |

**Section.cpp**

|  |  |
| --- | --- |
| **Название метода** | **Описание** |
| void Add(string Surname, int Pay) | Добавляет сотрудника в отдел. Если очередь не наполнена. |
| void Delete() | Удаляет сотрудника, если очередь не пустая. |
| void Show() | Выводит очередь, если она не пустая. |
| bool Search(string SurName) | Поиск сотрудника. |
| bool IsFull() | Проверка заполненности очереди. |
| Section(string Name) | Конструктор класса. |
| ~Section() | Деструктора класса. |
| string GetName() | Возвращает название отдела. |
| Employee\* GetMass() | Возвращает указатель на массив. |
| Employee\* GetFirst() | Возвращает указатель на первый элемент очереди. |
| Employee\* GetLast() | Возвращает указатель на первую пустую ячейку очереди. |
| void SetName(string Name) | Задает название отдела. |

**Emplyee.cpp**

|  |  |
| --- | --- |
| **Название метода** | **Описание** |
| void SetSurname(string Surname) | Возвращает фамилию сотрудника. |
| string GetSurname() | Задает фамилию сотрудника. |
| void SetPay(int Pay) | Задает размер зарплаты сотрудника. |
| int GetPay() | Возвращает размер зарплаты сотрудника. |
| Employee(string Surname, int Pay) | Конструктор класса. |
| ~Employee() | Деструктор класса. |

## **4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Взаимодействие с пользователем осуществляется через консольное меню. Запуск осуществляется в среде разработки Visual Studio 2020. После запуска пользователь увидит следующее меню:

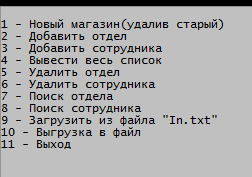


Рисунок 6 Основное меню

### **Функции основного меню**

#### Новый магазин

Для успешного выполнения действия требуется ввести корректно название магазина, состоящее из букв алфавита. В случае, если пользовательский ввод не проходит валидацию, программа выдаст сообщение с новым запросом

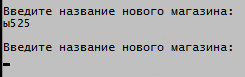


Рисунок 7 Зарос нового ввода

* + 1. Добавить отдел

Запрос ввода названия отдела с проверкой на валидность ввода. Если добавляемый отдел первый, запрашивается сразу название отдела, если нет, программа пользователю предлагается добавить отдел ДО/ПОСЛЕ заданного.



Рисунок 8 Добавление отдела



Рисунок 9 Проверка ввода на валидность

* + 1. Добавить сотрудника

Запрос у пользователя название отдела, в который нужно добавить сотрудника, в случае отсутствия такого в списке, в консоли выведется сообщение. Если отдел есть в списке, подпрограмма добавит сотрудника в очередь.

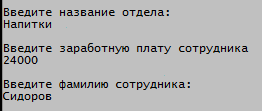


Рисунок 10 Добавление сотрудника

* + 1. Вывести весь список

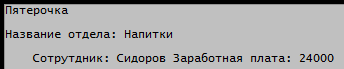


Рисунок 11 Вывод базы данных

* + 1. Удалить отдел

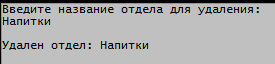


Рисунок 12 Удаление отдела

* + 1. Удалить сотрудника

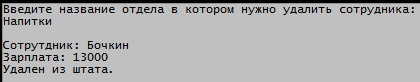


Рисунок 13 Удаление сотрудника

* + 1. Поиск отдела



Рисунок 14 Поиск отдела

* + 1. Поиск сотрудника



Рисунок 15 Поиск сотрудника

* + 1. Загрузка из файла

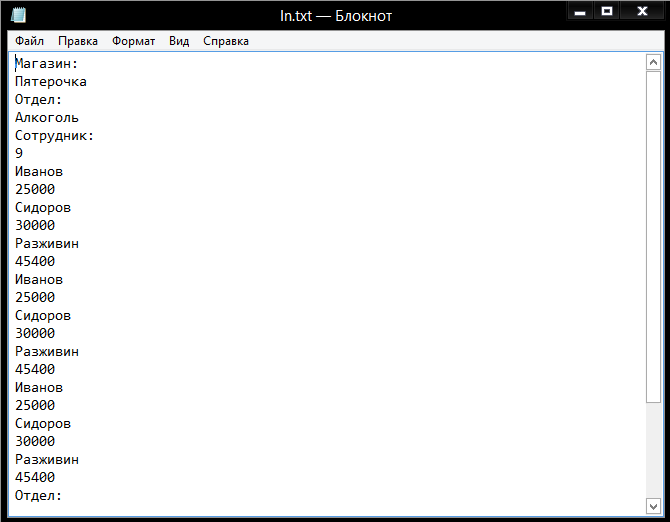
Подпрограмма производит загрузку базы данных из файла In.txt расположенного в корневом каталоге проекта. Структура данных имеет следующий вид: 

Рисунок 16 Файл In.txt

После тега «Магазин:» читается название магазина, после тега «Отдел:» - название отдела, после тега «Сотрудник:» - количество сотрудников в отделе и далее фамилия и зарплата сотрудника.

* + 1. Выгрузка из файла

Выгрузка производится аналогичным образом.

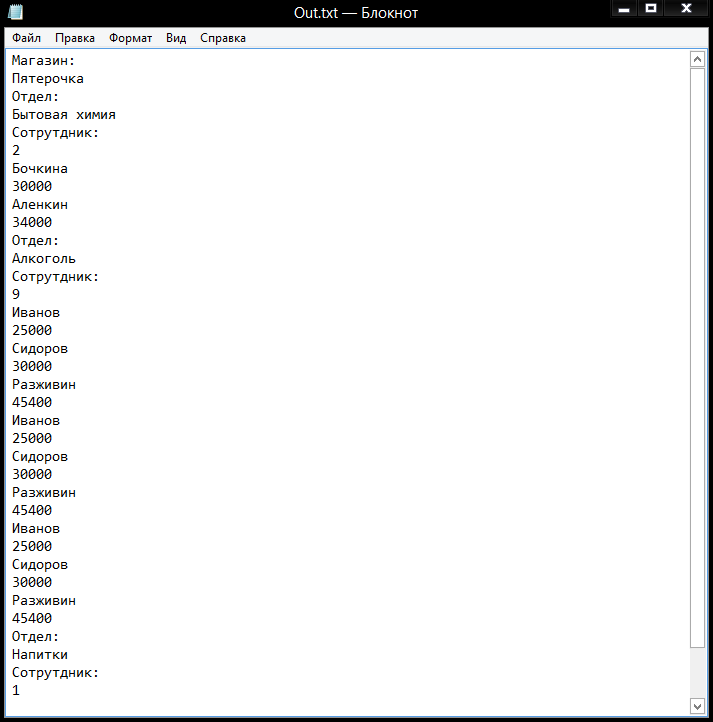


Рисунок 17 Файл Out.txt

* + 1. Выход

Удаление всей базы данных и выход из программы.

## **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**

Название магазина – любые символы, конкретно латинские и кириллицы;

Название отдела – любые символы, конкретно латинские и кириллицы;

Количество сотрудников в отделе – любое, в данном случае 10;

Фамилия сотрудника – любые символы;

Заработная плата сотрудника – любые числа, до 10-значных;

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

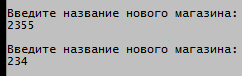


Рисунок 18 Повтор запроса строкового ввода

В дальнейшем любой строковый ввод будет запрашивать у пользователя ввод только символов. Если введена цифра, запрос повторится.

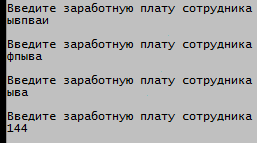


Рисунок 19 Повтор запроса числа

Аналогично с запросом числа, запрос будет повторятся пока пользователь не введет только цифры.

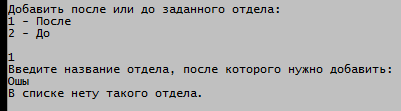


Рисунок 20 Ошибка при вводе отсутствующего отдела

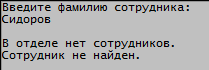


Рисунок 21 Поиск сотрудника

При запуске программы, если мы попытаемся совершить какое-либо действие, и база данных не будет создана, программа выведет следующее сообщение:



Рисунок 22 Не создана база данных

## **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.

## **ПРИЛОЖЕНИЕ**

Листинг программы: