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 комбинированной структуры данных

Вариант № 7.8

Выполнил:   
 студент группы 4210  
Султанов А.Д.

Проверил:   
 Доцент каф. ПМИ к.т.н.

Сотников С. В.

Казань 2021

**Оглавление**

[**ВВЕДЕНИЕ** 4](#_Toc73137746)

[**1.** **ПОСТАНОВКА ЗАДАЧИ** 5](#_Toc73137747)

[**2.** **ОПИСАНИЕ СТРУКТУРЫ ДАННЫХ СТАТИЧЕСКИЙ СТЕК УПОРЯДОЧЕННЫХ ДИНАМИЧЕСКИХ СПИСКОВ** 6](#_Toc73137748)

[**2.1** **. Структура данных типа «Статический стек»** 6](#_Toc73137749)

[**2.2** **.Структура данных типа «Упорядоченный динамический список»** 7](#_Toc73137750)

[**2.3** **.Комбинированная структура данных «Статический стек упорядоченных динамических списков»** 9](#_Toc73137751)

[**3** **РУКОВОДСТВО ПРОГРАММИСТА** 10](#_Toc73137752)

[**3.1** **. Описание структуры программы** 10](#_Toc73137753)

[**3.2** **. Описание структур** 11](#_Toc73137754)

[**3.3** **. Описание всех разработанных подпрограмм для работы со структурой** 11](#_Toc73137755)

[**3.3.1** **Класс Program.cs** 13](#_Toc73137756)

[**3.3.2** **Класс CommercialNetwork.cs** 13](#_Toc73137757)

[**3.3.3** **Класс Shop.cs** 15](#_Toc73137758)

[**3.3.4** **Класс Department.cs** 16](#_Toc73137759)

[**4** **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ** 17](#_Toc73137760)

[**4.1** **.Функции основного меню** 17](#_Toc73137761)

[**4.1.1** **Новая сеть магазинов** 17](#_Toc73137762)

[**4.1.2** **Добавить магазин** 18](#_Toc73137763)

[**4.1.3** **Добавить отделения** 18](#_Toc73137764)

[**4.1.4** **Вывод** 19](#_Toc73137765)

[**4.1.5** **Поиск** 19](#_Toc73137766)

[**4.1.6** **Удалить магазина** 20](#_Toc73137767)

[**4.1.7** **Удалить отделения** 20](#_Toc73137768)

[**4.1.8** **Очистить сеть магазинов** 20](#_Toc73137769)

[**4.1.9** **Работа с файлом** 20](#_Toc73137770)

[**4.1.10** **Выход** 21](#_Toc73137771)

[**4.2** **Загрузка данных из файла** 22](#_Toc73137772)

[**4.3** **Сохранение структуры во внешнем файле** 23](#_Toc73137773)

[**5** **РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММНОГО КОМПЛЕКСА** 24](#_Toc73137774)

[**5.1** **Организация тестирования ПО** 24](#_Toc73137775)

[**5.2** **Входные данные** 24](#_Toc73137776)

[**ЗАКЛЮЧЕНИЕ** 27](#_Toc73137777)

[**ЛИТЕРАТУРА** 28](#_Toc73137778)

[**ПРИЛОЖЕНИЯ** 29](#_Toc73137779)

[**1.** **Program.cs** 29](#_Toc73137780)

[**2.** **CommercialNetwork.cs** 36](#_Toc73137781)

[**3.** **Shop.cs** 39](#_Toc73137782)

[**4.** **Department.cs** 44](#_Toc73137783)

# **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: статический стек упорядоченных динамических списков, вариант (7.8), на основе объектного подхода.

Структура должна иметь следующее информационное наполнение: Торговая Сеть (наименование) – композиция магазинов (наименование); Магазин – композиция отделов (номер, профиль торговли)

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 файле (текстовом или XML) 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 СТАТИЧЕСКИЙ СТЕК УПОРЯДОЧЕННЫХ ДИНАМИЧЕСКИХ СПИСКОВ**

## **. 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).

Элементами стеков могут быть любые однотипные данные.

Программная реализация стеков возможна двумя способами:

* статически с помощью массива
* динамически с помощью механизма указателей

Основные операции со статическим стеком:

* добавление элемента на вершину стека
* удаление элемента из вершины стека

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. Выберем второй способ. Тогда для пустого стека переменная SP = 1 (если индексация элементов массива начинается с 1), и при каждом добавлении нового элемента переменная SP увеличивается на 1, а при удалении – уменьшается на 1.

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

## **.Структура данных типа «Упорядоченный динамический список»**

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 nil).

Опять же для удобства реализации будем считать, что список ВСЕГДА содержит хотя бы один элемент-заголовок с адресом первого реального элемента списка. Это позволяет унифицировать процедуры добавления и удаления крайних элементов и устранить некоторые проверки. Адрес элемента-заголовка задается переменной-указателем pHead. Эта переменная устанавливается при первоначальном создании списка и в дальнейшем НЕ изменяется.

Упорядоченный список говорит о том, что при добавление с писок нового элемента, для элемента автоматически подбирается нужно место в списке.

Создание пустого списка включает:

* выделение памяти под заголовок: new(pHead);
* установку пустой ссылочной части заголовка: pHead^.next := **nil**;

Проход по списку от первого элемента к последнему практически не отличается от прохода по очереди.

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

pCurrent := pHead^.next;

**while** (pCurrent <> **nil**) **and** (pCurrent^.inf <> ‘заданное значение’)

**do** pCurrent := pCurrent^.next;

**if** pCurrent = **nil** **then** ‘Элемента нет’ **else** ‘элемент найден’;

Удаление заданного элемента включает:

* поиск удаляемого элемента с определением адреса элемента-предшественника (для этого вводится еще одна вспомогательная ссылочная переменная pPrev, инициированная адресом заголовка и изменяющая свое значение внутри цикла)
* если удаляемый элемент найден, то изменяется ссылочная часть его предшественника:

pPrev^.next := pCurrent^.next

* удаляемый элемент обрабатывается необходимым образом, т.е. либо освобождается занимаемая им память, либо он включается во вспомогательный список

Довольно часто используется упорядоченная разновидность линейного списка, 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. Например, если порядок элементов определяется целыми числами по возрастанию, то при поиске подходящего места надо найти первый элемент, больший заданного и выполнить вставку ПЕРЕД этим элементом.

## **.Комбинированная структура данных «Статический стек упорядоченных динамических списков»**

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

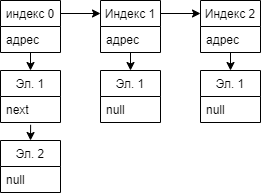


Рисунок 1 Схема комбинированной структуры данных

В первую очередь вводится тип данных, описывающий структуру элементов вспомогательного списка. После этого описывается структура элементов стека. Обработка таких структур включает больше операций, поскольку практически любая типовая операция (поиск, просмотр, добавление, удаление) может выполняться как со стеком, так и с любым вспомогательным списком. Например, полный **поиск** или **полный проход** реализуется двойным циклом: внешний цикл проходит по элементам стека, а внутренний обрабатывает отдельно каждый вспомогательный список.

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

# **РУКОВОДСТВО ПРОГРАММИСТА**

## **. Описание структуры программы**

Для разработки проекта была использована среда разработки программного обеспечения JetBrains Rider 2019.

Весь программный код, включая основную программу, хранится в файлах .cs. Разработанные отношения агрегации классов, представленная на рисунке 2, включают в себя коллекции и классы, отвечающие за информационное заполнение, а также функции и процедуры, предназначенные для работы с созданными структурами данных.

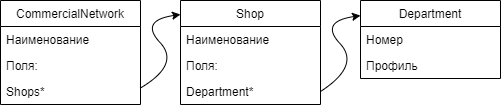


Рисунок 2 Отношения агрегации классов

## **. Описание структур**

Контейнер представляет собой комбинированную структуру. Структура Сеть магазинов – статический стек. Структура Магазин – упорядоченный динамический список.

Разработаны две основные структуры, представленные в следующих контейнерах.

## **. Описание всех разработанных подпрограмм для работы со структурой**

Контейнер представляет собой комбинированную структуру. Главная структура – статический стек. Дочерняя структура – упорядоченный динамический список.

Разработаны две основные структуры, представленные в следующих контейнерах:

1. Класс CommercialNetwork – представляет собой торговую сеть, реализованную через статический стек.

public string CommercialNetworkName { get; set; }  
public Shop[] Shops { get; set; }  
private int \_count;  
private const int \_maxLength = 10;  
  
public CommercialNetwork(string name, int length = \_maxLength)  
{  
 CommercialNetworkName = name;  
 Shops = new Shop[length];  
}

Shops – массив для хранения элементов стека.

CommercialNetworkName – наименование торговой сети

\_count - текущее количество элементов

\_maxLength – стандартное количество элементов

1. Класс Shop – Упорядоченный динамический список.

public string ShopName { get; set; }  
private Node \_head;  
private Node \_tail;  
private int \_count;

public Shop(string name)  
{  
 ShopName = name;  
}

ShopName – наименование магазина.  
\_head – Ссылка на начало списка.  
\_tail – Ссылка на конец списка.  
\_count - текущее количество элементов в списке

### **Класс Program.cs**

Описание класса, который реализует взаимодействие с пользователем через консоль.

Таблица 1 Описание методов класса Program.cs

|  |  |
| --- | --- |
| **Название метода** | **Описание** |
| void Main() | Метод реализует взаимодействие с пользователем посредством вывода на экран меню и отлова нажатий. Затем вызывает соответствующую функцию. |
| void Help() | Метод для вывода подсказки о названии команд на экран. |
| void CommandRecognizer() | Метод для обрабатывания ввода команд в командной строке. Тут же выполняет вызванные команды |
| void ReadFile() | Загрузка структуры из файла - метод, если файл не пустой, запрашивает подтверждение на очистку структур перед загрузкой. Затем производит чтение файла “input.xml” в корневой папке и сохраняет структуру в памяти. |

### **Класс CommercialNetwork.cs**

Таблица 2 Описание методов класса CommercialNetwork.cs

|  |  |
| --- | --- |
| **Название метода** | **Описание** |
| public CommercialNetwork(string name, int length = \_maxLength) | Конструктор класса. В нем происходит выделение памяти для имени и для размера. |
| public bool PushShop(string shopName) | Метод добавления магазина в стек. |
| public Shop PopShop() | Метод удаления магазина из стека |
| public void AddDepartment(string toShop, int number, string profile) | Метод добавления отделения в магазин |
| public void RemoveDepartment(string fromShop, int number) | Метод удаление отделения из магазина |
| public bool SearchShop(string name, out int index) | Метод Поиска магазина в стеке. |
| public override string ToString() | Метод преобразования магазина к читабельному виду в строку |
| Public string getShopInfo(string shop) | Метод получения читабельной информации омагазине |
| public Department GetDepartment(string shop, int number) | Метод Получения отеделния. |
| public bool IsEmpty() | Метод возвращает true, если стек магазинов пуст. Иначе – false. |
| public void Dispose() | Метод очищает память. |
| public bool SearchDepartmentFirst(string shop, int number) | Поиск отделения сначала списка |
| public bool SearchDepartmentEnd(string shop, int number) | Поиск отделения с конца списка |
| public XElement WriteXml() | Возвращает представление Xml Элемента главной структуры |

### **Класс Shop.cs**

Таблица 3 Описание методов класса Shop.cs

|  |  |
| --- | --- |
| **Название метода** | **Описание** |
| public string ShopName | Свойство доступа к переменной ShopName. |
| Public int Count | Свойство доступа к количеству элементов. |
| Public bool IsEmpty | Свойство доступа к значению пустоты списка. |
| public void AddDepartment(int number, string profile) | Метод добавления отделения |
| public bool RemoveDepartment(int number) | Метод удаления отделения |
| public bool ContainsFirst(int number) | Метод, который проверяет, есть ли отделение в списке, обходя список сначала |
| public bool ContainsEnd(int number) | Метод, который проверяет, есть ли отделение в списке, обходя список сначала |
| public Deraptment GetDepartment(int number) | Метод возвращающий информацию ою отделении |
| public override string ToString() | Метод возвращающий удобное представление отделения в виде строки |
| public void Dispose() | Метод очищающий память списка |
| public XElement WriteXml() | Метод возвращающий Магазин в виде Xml представления |
| public IEnumerator IEnumerator. GetEnumerator() | Метод для обхода списка магазинов в цикле |
| public IEnumerator<Department> IEnumerator<Department>  .GetEnumerator() | Реализация обхода списка с заголовочного элемента |
| public IEnumerator<Department> BackEnumerator() | Реализация обхода списка с последнего элемента |

### **Класс Department.cs**

Таблица 4 Описание методов класса Department.cs

|  |  |
| --- | --- |
| **Название метода** | **Описание** |
| public Department(int number, string profile) | Конструктор класса с параметрами. |
| public override string ToString() | Представление отделения в удобном строковом виде |
| Public XElement WriteXml() | Возвращает xml представление отделения |
| public void Dispose() | Освобождает память, занимаемую отделением. |
| public int Number | Свойство доступа к номеру отделения |
| public string Profile | Свойство доступа к профилю отделения |

# **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Взаимодействие с пользователем осуществляется через консольное меню. Запуск осуществляется посредством открытия файла Program.exe. После запуска программа выведет основное меню:

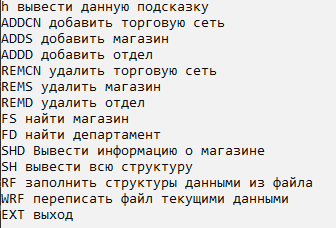


Рисунок 3 Основное меню

## **.Функции основного меню**

### **Новая сеть магазинов**

Подпрограмма проверяет созданы ли структуры данных. Если да, то выводит следующее сообщение.



Рисунок 4 Создание новой сети

Иначе подпрограмма создаст новый экземпляр класса CommercialNetwork.

**

Рисунок 5 Создание новой сети

### **Добавить магазин**

Если сеть создана, то при вводе команды ADDS программа запросит данные магазина (имя).

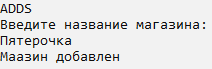


Рисунок 6 Добавление магазина

### **Добавить отделения**

Если сеть создана и стек магазинов не пуст, то программа запросит данные для добавления отделения к магазину. Затем она проверит существует ли магазин. Если проверки пройдены успешно, произойдет добавление отделения.

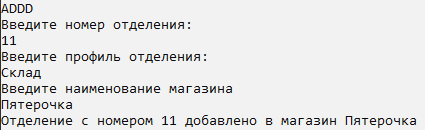


Рисунок 7 Добавление отделения

### **Вывод**

Программа, если сеть создана, выведет всю структуру в консоль.

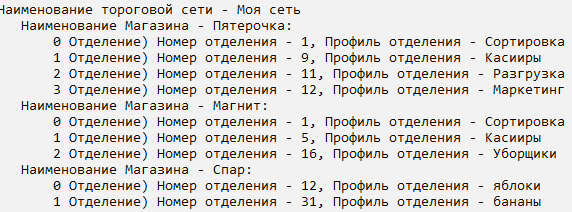


Рисунок 8 Вывод структуры

### **Поиск**

Если сеть создана и стек магазинов не пуст, то при вызове команды FS подпрограмма запросит имя магазина для его поиска в структуре.

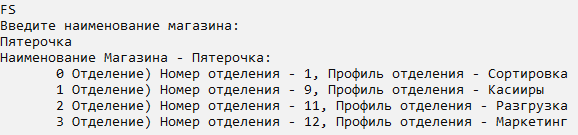


Рисунок 9 Поиск магазина

Если сеть магазинов создана и стек магазинов не пуст, то при вызове команды FD подпрограмма запросит номер отделения и магазин, в котором он находится.

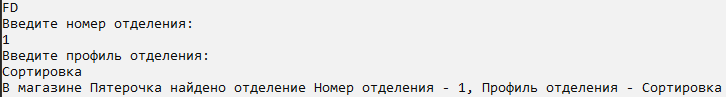


Рисунок 10 Поиск отделения

### **Удалить магазина**

Если сеть магазинов создана и стек магазинов не пуст, то при вызове команды REMS программа удалит последний магазин в стеке.

**

Рисунок 11 Удаление магазина

### **Удалить отделения**

Если сеть создана и стек магазинов не пуст, то подпрограмма запросит имя магазина, отделение которое нужно удалить.

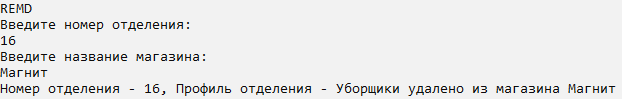


Рисунок 12 Удаление отделения

### **Очистить сеть магазинов**

Если структура создана, удаляет ее и освобождает память. Затем выводит сообщение об очистке структуры.



Рисунок 13 Очистка данных сети

### **Работа с файлом**

Для загрузки и выгрузки структуры использована встроенная библиотека System.Xml.Linq. При вызове команды WRF происходит запись в файл xml, при наличии созданной ранее структуры данных. При вызове команды RF происходит чтение данных из файла, при наличии созданной ранее структуры.

1. Загрузка из файла – подпрограмма, если структура не пустая, запрашивает подтверждение на удаление структуры перед загрузкой. Подпрограмма ищет файл с названием Input.xml в корневой папке программы и, если файл существует, произведется чтение. Если подпрограмма успешно загрузит структуру, выведется следующее сообщение.

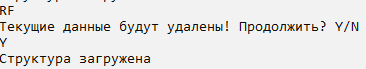


Рисунок 14 Загрузка из файла.



Рисунок 15 Загрузка из файла.

1. Выгрузка в файл – подпрограмма, если структура не пустая, записывает данные в файл network.xml в корневой папке программы.



Рисунок 16 Выгрузка в файл

### **Выход**

Подпрограмма, при вызове команды EXT, очистит базу данных и завершит работу программы. Предварительно пользователь будет уведомлен, что при закрытии программы его данные автоматически удалятся.



*Рисунок 17. Выход из программы.*

## **Загрузка данных из файла**

В файле “network.xml” должна быть записана структура с соответствующими тегами. Пример корректного заполнения файла показан на рис. 18.

-Открывается тег <CommercialNetwork> с атрибутов name

Записывается нужное количество магазинов:  
-Открываем тег поставщика < Shop> с атрибутами как показано на примере.

-Открываем тег товара <Department > с атрибутами как показано на примере. Записываем нужное количество трамваев.

Закрываем теги </ Department >

</ Shop >

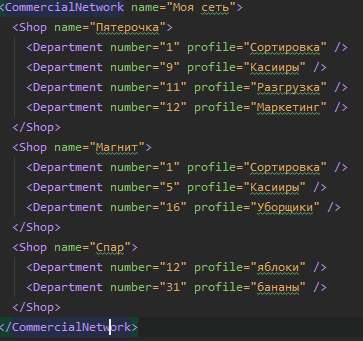
</ CommercialNetwork >  


Рисунок 18 “network.xml”

## **Сохранение структуры во внешнем файле**

Подпрограмма выгрузки создаст файл “network.xml” (при его отсутствии) и запишет в нее отсортированную структуру. Пример выгрузки показан на рис. 19.

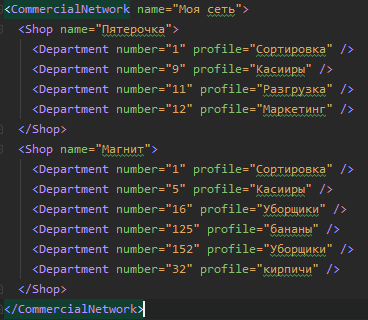


Рисунок 19 network.xml

# **РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММНОГО КОМПЛЕКСА**



## **Организация тестирования ПО**

Проводилось функциональное тестирование с целью установить соответствие разработанного программного обеспечения (ПО) исходным функциональным требованиям, 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 сети магазинов – любые символы.  
Имя магазина – любые символы;  
Количество магазинов – любые положительные числа (не больше 9-значных);  
Номер отделения – любые положительные числа (не больше 9-значных);  
 Профиль отделения – любые символы

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

1. Попытка добавить магазин, добавить отделение, удалить магазин, удалить отделение, вывести структуру, поиск, когда структура не создана – вывод предупреждения, представленного на риc. 20.



Рисунок 20 Обработка исключения

1. Ввод символа вместо числа – вывод сообщения об ошибке, представленный на рис. 21 и приостановление команды.

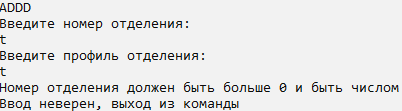


Рисунок 21 Обработка исключения

1. Ввод пустой строки – вывод предупреждения, представленного на рис. 22 и приостановление команды.

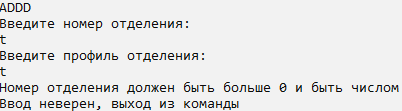


Рисунок 22 Обработка исключения

1. Если при добавлении отделения стек пуст – вывод предупреждения, представленного на рис. 23.



Рисунок 23 Обработка исключения

1. Если при добавлении отделения в стеке нет указанного магазина – вывод предупреждения, представленного на рис. 24.

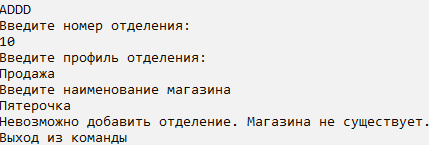


Рисунок 24 Обработка исключения

1. Поиск или удаление магазина, отделения при пустом стеке магазинов – вывод предупреждения, представленного на рис. 25.



Рисунок 25 Обработка исключения

1. Отсутствует файл «network.xml” - вывод предупреждения, представленного на рис. 26.

**

Рисунок 26 Обработка исключения

1. В файле для чтения не задан атрибут – вывод предупреждения, представленного на рис. 27.



Рисунок 27 Обработка исключения

# **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. Сделав вывод, можно сказать, что программа соответствует всем требованиям.

# **ЛИТЕРАТУРА**

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

# **ПРИЛОЖЕНИЯ**

Листинг программы:

## **Program.cs**

using System;  
using System.Xml.Linq;  
  
namespace Cursovik  
{  
 internal class Program  
 {  
 private static CommercialNetwork \_network = null;  
 private static XDocument xdoc;  
 public static void Main(string[] args)  
 {  
 Help();  
 var command = Console.ReadLine();  
 while (true)  
 {  
 if (command.Equals("EXT"))  
 {  
 Console.WriteLine("Данные автоматически не сохранятся. Вы уверены, что хотите выйти? Y/N");   
 var answer = Console.ReadLine();  
 if (string.Equals(answer, "Y"))  
 {  
 if (\_network != null)  
 {  
 \_network.Dispose();  
 \_network = null;   
 }  
 break;  
 }  
 else  
 {  
 command = Console.ReadLine();  
 }  
 }  
 CommandRecognizer(command);  
 command = Console.ReadLine();  
 }  
 }  
   
 private static void CommandRecognizer(string command) {   
 switch (command) {   
 case "h":   
 Help();  
 break;   
 case "ADDCN":  
 if (\_network != null)  
 {  
 Console.WriteLine("Текущие данные будут удалены! Продолжить? Y/N");  
 switch (Console.ReadLine())  
 {  
 case "Y":   
 \_network.Dispose();   
 \_network = null;   
 Console.WriteLine("Введите название сети:");  
 \_network = new CommercialNetwork(Console.ReadLine());   
 Console.WriteLine("Сеть магазинов создана");   
 break;   
 case "N":   
 Console.WriteLine("Выход из команды");   
 break;   
 default:   
 Console.WriteLine("Сеть магазинов НЕ создана");  
 Console.WriteLine("Ввод неверен, выход из команды");  
 break;  
 }   
 break;  
 }   
 Console.WriteLine("Введите название сети:");  
 var nameNetwork = Console.ReadLine();  
 Console.WriteLine("Введите количество магазинов в сети (длина стека):");  
 if (int.TryParse(Console.ReadLine(), out int n))  
 {  
 \_network = new CommercialNetwork(nameNetwork, n);   
 Console.WriteLine("Сеть магазинов создана");   
 }  
 else  
 {  
 Console.WriteLine("Ошибка ввода, выход из команды");  
 }  
 break;   
 case "ADDS":  
 if (\_network != null)  
 {  
 Console.WriteLine("Введите название магазина:");   
 var sName = Console.ReadLine();  
 try  
 {  
 if (\_network.PushShop(sName))  
 {  
 Console.WriteLine("Маазин добавлен");  
 }  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e.Message);  
 Console.WriteLine("Маазин НЕ добавлен");  
 Console.WriteLine("Выход из команды");  
 }  
 break;  
 }   
 Console.WriteLine("Сеть магазинов отсутствует");   
 break;   
 case "ADDD":   
 if (\_network != null)  
 {  
 if (!\_network.IsEmpty)  
 {  
 Console.WriteLine("Введите номер отделения:");   
 var dNumber = Console.ReadLine();   
 Console.WriteLine("Введите профиль отделения:");   
 var dProfile = Console.ReadLine();  
 if (int.TryParse(dNumber, out int number) && number >= 0)  
 {  
 try  
 {  
 Console.WriteLine("Введите наименование магазина");  
 var ss = Console.ReadLine();  
 \_network.AddDepartment(ss, number, dProfile);  
 Console.WriteLine($"Отделение с номером {dNumber} добавлено в магазин {ss}");  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e.Message);  
 Console.WriteLine("Выход из команды");  
 }  
 }  
 else  
 {  
 Console.WriteLine("Номер отделения должен быть больше 0 и быть числом");  
 Console.WriteLine("Ввод неверен, выход из команды");  
 }  
 }  
 else  
 {  
 Console.WriteLine("Отсутствуют магазины");  
 }  
 break;   
 }  
 Console.WriteLine("Сеть магазинов отсутсвует");   
 break;  
 case "REMCN":  
 if (\_network != null)  
 {  
 \_network.Dispose();   
 \_network = null;   
 Console.WriteLine("Данные удалены");   
 break;  
 }  
 Console.WriteLine("Сеть магазинов отсутсвует");  
 break;   
 case "REMS":  
 if (\_network != null)  
 {  
 try  
 {  
 var shop = \_network.PopShop();  
 Console.WriteLine($"Магазин {shop.ShopName} удален");   
 shop.Dispose();  
 shop = default;  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e.Message);  
 Console.WriteLine("Выход из команды");  
 }  
 break;  
 }   
 Console.WriteLine("Сеть магазинов отсутсвует");  
 break;  
 case "REMD":  
 if (\_network != null)  
 {   
 Console.WriteLine("Введите номер отделения:");   
 var dNumber = Console.ReadLine();   
   
 if (int.TryParse(dNumber, out int number))   
 {  
 try  
 {  
 Console.WriteLine("Введите название магазина:");  
 var s = Console.ReadLine();  
 Console.WriteLine($"{\_network.GetDepartament(s, number)} удалено из магазина {s}");  
 \_network.RemoveDepartment(s, number);  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e.Message);  
 Console.WriteLine("Выход из команды");  
 }  
 }  
 else  
 {  
 Console.WriteLine("Ввод неверен, выход из команды");  
 }   
 break;   
 }   
 Console.WriteLine("Сеть магазинов отсутсвует");   
 break;   
 case "FS":  
 if (\_network != null)  
 {  
 Console.WriteLine("Введите наименование магазина:");  
 var sName = Console.ReadLine();  
 try  
 {  
 Console.WriteLine(\_network.getShopInfo(sName));  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e.Message);  
 Console.WriteLine("Выход из команды");  
 throw;  
 }  
   
 break;  
 }   
 Console.WriteLine("Сеть магазинов отсутсвует");   
 break;   
 case "FD":  
 if (\_network != null)  
 {  
 Console.WriteLine("Введите номер отделения:");   
 var dNumber = Console.ReadLine();  
   
 Console.WriteLine("Введите профиль отделения:");   
 var dProfile = Console.ReadLine();  
   
 if (int.TryParse(dNumber, out int num))  
 {  
 try  
 {  
 Console.WriteLine(\_network.SearchDepartmentFirst(num, dProfile, out string shop) ? $"В магазине {shop} найдено отделение {\_network.GetDepartament(shop, num).ToString()}" : "Отделение не найдено");  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e.Message);  
 Console.WriteLine("Выход из команды");  
 }  
 }  
 else  
 {  
 Console.WriteLine("Ввод неверен, выход из команды");  
 }  
 break;  
 }   
 Console.WriteLine("Сеть магазинов отсутсвует");  
 break;   
 case "SHS":  
 if (\_network != null)  
 {  
 Console.WriteLine("Введите наименование магазина:");  
 var nnn = Console.ReadLine();  
 try  
 {  
 Console.WriteLine(\_network.getShopInfo(nnn));  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e.Message);  
 Console.WriteLine("Выход из команды");  
 }  
 break;  
 }   
 Console.WriteLine("Сеть магазинов отсутсвует");   
 break;   
 case "RF":  
 try  
 {  
 if (\_network != null)  
 {  
 if (!\_network.IsEmpty)  
 {  
 Console.WriteLine("Текущие данные будут удалены! Продолжить? Y/N");  
 var answer = Console.ReadLine();  
 if (string.Equals(answer, "Y"))  
 {  
 \_network.Dispose();  
 \_network = null;  
 ReadXml();  
 }  
 else  
 {  
 Console.WriteLine("Данные не прочтианы.");  
 }  
 }  
 else  
 {  
 ReadXml();  
 }   
 }  
 else  
 {  
 ReadXml();  
 }  
 }  
 catch (Exception e)  
 {  
 \_network.Dispose();  
 \_network = null;  
 Console.WriteLine(e.Message);  
 Console.WriteLine("Выход из команды");  
 }  
 break;   
 case "WRF":  
 if (\_network != null)  
 {  
 xdoc = new XDocument();  
 xdoc.Add(\_network.WriteXml());   
 xdoc.Save("network.xml");  
 Console.WriteLine("Структура выгружена в network.xml");  
 break;  
 }   
 Console.WriteLine("Сначала создайте Сеть магазинов");  
 break;   
 case "SH":  
 if (\_network != null)  
 {  
 Console.WriteLine(\_network.ToString());  
 break;  
 }   
 Console.WriteLine("Сначала создайте Сеть магазинов");   
 break;  
 default:   
 Console.WriteLine("Нет такой команды");   
 break;   
 }  
 }  
  
 private static void Help()  
 {  
 Console.WriteLine("h вывести данную подсказку");   
 Console.WriteLine("ADDCN добавить торговую сеть");   
 Console.WriteLine("ADDS добавить магазин");  
 Console.WriteLine("ADDD добавить отдел");   
 Console.WriteLine("REMCN удалить торговую сеть");   
 Console.WriteLine("REMS удалить магазин");  
 Console.WriteLine("REMD удалить отдел");   
 Console.WriteLine("FS найти магазин");   
 Console.WriteLine("FD найти департамент");   
 Console.WriteLine("SHD Вывести информацию о магазине");   
 Console.WriteLine("SH вывести всю структуру");   
 Console.WriteLine("RF заполнить структуры данными из файла");   
 Console.WriteLine("WRF переписать файл текущими данными");   
 Console.WriteLine("EXT выход");  
 }  
  
 private static void ReadXml()  
 {  
 XDocument xdoc;  
 try  
 {  
 xdoc = XDocument.Load("network.xml");  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e.Message);  
 return;  
 }  
   
 XElement networkElement = xdoc.Element("CommercialNetwork");  
 if(networkElement == null)   
 throw new Exception("Ошибка формата данных. В файле должен быть корневой элемент CommercialNetwork");  
   
 XAttribute networkNAme = networkElement.Attribute("name");  
 if(networkNAme == null || String.IsNullOrWhiteSpace(networkNAme.Value))   
 throw new Exception("Ошибка формата данных. У сети магазинов должен быть атрибут name и должен быть не пустым!");  
   
 \_network = new CommercialNetwork(networkNAme.Value);  
 foreach (XElement shop in networkElement.Elements("Shop"))  
 {  
 XAttribute shopName = shop.Attribute("name");  
 if (shopName == null || String.IsNullOrWhiteSpace(shopName.Value))  
 throw new Exception("Ошибка формата данных. У магазина должен быть атрибут name и должен быть не пустым!");  
 \_network.PushShop(shopName.Value);  
   
 foreach (var department in shop.Elements("Department"))  
 {  
 XAttribute dNumber = department.Attribute("number");  
 XAttribute dProfile = department.Attribute("profile");  
   
 if(dNumber == null || String.IsNullOrWhiteSpace(dNumber.Value))   
 throw new Exception("Ошибка формата данных. У отделения должен быть атрибут number и должен быть уникальным числом больше 0!");  
 if(dProfile == null || String.IsNullOrWhiteSpace(dProfile.Value))   
 throw new Exception("Ошибка формата данных. У отделения должен быть атрибут profile и должен быть не пустой строкой!");  
  
 if (int.TryParse(dNumber.Value, out int index))  
 {  
 \_network.AddDepartment(shopName.Value, index, dProfile.Value);  
 }  
 else  
 {  
 throw new Exception("Ошибка формата данных. У отделения атрибут number должен быть уникальным числом больше 0!");  
 }  
 }  
 }  
 Console.WriteLine("Структура загружена");  
 }  
 }  
}

## **CommercialNetwork.cs**

using System;  
using System.Text;  
using System.Xml.Linq;  
  
namespace Cursovik  
{  
 public class CommercialNetwork : IDisposable  
 {  
 public string CommercialNetworkName { get; set; }  
 public Shop[] Shops { get; set; }  
  
 private int \_count;  
 private const int \_maxLength = 10;  
   
 public CommercialNetwork(string name, int length = \_maxLength)  
 {  
 CommercialNetworkName = name;  
 Shops = new Shop[length];  
 }  
   
 public bool IsEmpty => \_count == 0;  
   
 public int Count => \_count;  
  
 public Shop Peek() => Shops[\_count - 1];  
  
 public bool PushShop(string shopName)  
 {  
 if(\_count == Shops.Length)  
 throw new InvalidOperationException("Стек заполнен, добавление невозможно!");  
 if(SearchShop(shopName, out int index))   
 throw new InvalidOperationException("Магазин с таким именем уже существует. Наименование магазина должно быть уникальным.");  
   
 Shops[\_count++] = new Shop(shopName);  
 return true;  
 }  
  
 public Shop PopShop()  
 {  
 if (IsEmpty)  
 throw new InvalidOperationException("Стек пуст!");  
   
 Shop item = Shops[--\_count];  
 Shops[\_count] = default(Shop);  
  
 return item;  
 }  
  
 public void AddDepartment(string toShop, int number, string profile)  
 {  
 if(!SearchShop(toShop, out int index)) throw new InvalidOperationException("Невозможно добавить отделение. Магазина не существует.");  
 Shops[index].AddDepartment(number, profile);  
 }  
  
 public void RemoveDepartment(string fromShop, int number)  
 {  
 if(!SearchShop(fromShop, out int index)) throw new InvalidOperationException("Невозможно удалить отделение. Магазина не существует.");  
 Shops[index].RemoveDepartment(number);  
 }  
  
 public bool SearchShop(string name, out int index)  
 {  
 for (int i = 0; i < \_count; i++)  
 {  
 if (Shops[i].ShopName == name)  
 {  
 index = i;  
 return true;  
 }  
 }  
 index = -1;  
 return false;  
 }  
  
 public override string ToString()  
 {  
 var b = new StringBuilder();  
 b.Append($"Наименование тороговой сети - {CommercialNetworkName}\n");  
 for (int i = 0; i < \_count; i++)  
 {  
 b.Append(" ");  
 b.Append(Shops[i].ToString());  
 }  
  
 return b.ToString();  
 }  
  
 public string getShopInfo(string shop)  
 {  
 if(!SearchShop(shop, out int index)) throw new InvalidOperationException("Магазин не существует.");  
 return Shops[index].ToString();  
 }  
  
 public Department GetDepartament(string shop, int number)  
 {  
 if(!SearchShop(shop, out int index)) throw new InvalidOperationException("Невозможно найти отделение. Магазина не существует.");  
 return Shops[index].GetDepartament(number);  
 }  
  
 public bool SearchDepartmentFirst(int number, string profile, out string shop)  
 {  
 foreach (var item in Shops)  
 {  
 if (item.ContainsFirst(number) && item.GetDepartament(number).Profile == profile)  
 {  
 shop = item.ShopName;  
 return true;  
 }  
 }  
 shop = "";  
 return false;  
 }  
   
 public bool SearchDepartmentEnd(int number, string profile, out string shop)  
 {  
 foreach (var item in Shops)  
 {  
 if (item.ContainsEnd(number) && item.GetDepartament(number).Profile == profile)  
 {  
 shop = item.ShopName;  
 return true;  
 }  
 }  
 shop = "";  
 return false;  
 }  
   
 public void Dispose()  
 {  
 CommercialNetworkName = default;  
 for (int i = 0; i < \_count; i++)  
 {  
 Shops[i].Dispose();  
 Shops[i] = default;  
 }  
  
 Shops = default;  
 \_count = default;  
 }  
  
 public XElement WriteXml()  
 {  
 XElement network = new XElement("CommercialNetwork");  
 network.Add(new XAttribute("name", CommercialNetworkName));  
 for (int i = 0; i < \_count; i++)  
 {  
 network.Add(Shops[i].WriteXml());  
 }  
 return network;  
 }  
 }  
}

## **Shop.cs**

using System;  
using System.Collections;  
using System.Collections.Generic;  
using System.Text;  
using System.Xml.Linq;  
  
namespace Cursovik  
{  
 public class Node  
 {  
 public Node(Department department)  
 {  
 Data = department;  
 }  
 public Department Data { get; set; }  
 public Node Previous { get; set; }  
 public Node Next { get; set; }  
 }  
 public class Shop : IDisposable, IEnumerable<Department>  
 {  
 public string ShopName { get; set; }  
  
 private Node \_head;  
 private Node \_tail;  
 private int \_count;  
  
 public Shop(string name)  
 {  
 ShopName = name;  
 }  
  
 public int Count => \_count;  
   
 public bool IsEmpty => \_count == 0;  
   
 public void AddDepartment(int number, string profile)  
 {  
 Node node = new Node(new Department(number, profile));  
  
 if (\_head == null)  
 {  
 \_head = node;  
 \_tail = node;  
 }  
 else  
 {  
 Node current = \_head;  
   
 while (current != null)  
 {  
 if (current.Data.Number.Equals(number)) throw new Exception("Отделение с таким номером уже существует, он должен быть уникальным для магазина.");  
 if (current.Data.Number > number) break;  
 current = current.Next;  
 }  
  
 if (current == null)  
 {  
 \_tail.Next = node;  
 node.Previous = \_tail;  
 \_tail = node;  
 }  
 else  
 {  
 if (current == \_head)  
 {  
 Node temp = \_head;  
 node.Next = temp;  
 \_head = node;  
 temp.Previous = node;  
 }  
 else  
 {  
 node.Previous = current.Previous;  
 node.Next = current;  
 current.Previous.Next = node;  
 current.Previous = node;  
 }  
 }  
 }  
   
 \_count++;  
 }  
   
 public bool RemoveDepartment(int number)  
 {  
 Node current = \_head;  
   
 while (current != null)  
 {  
 if (current.Data.Number.Equals(number)) break;  
 current = current.Next;  
 }  
  
 if (current == null) return false;  
   
 if(current.Next != null)  
 {  
 current.Next.Previous = current.Previous;  
 }  
 else  
 {  
 \_tail = current.Previous;  
 }  
   
 if(current.Previous != null)  
 {  
 current.Previous.Next = current.Next;  
 }  
 else  
 {  
 \_head = current.Next;  
 }  
 \_count--;  
   
 current = default;  
   
 return true;  
 }  
  
 public bool ContainsFirst(int number)  
 {  
 Node current = \_head;  
 while (current != null)  
 {  
 if (current.Data.Number.Equals(number)) return true;  
 current = current.Next;  
 }  
 return false;  
 }  
   
 public bool ContainsEnd(int number)  
 {  
 Node current = \_tail;  
 while (current != null)  
 {  
 if (current.Data.Number.Equals(number)) return true;  
 current = current.Previous;  
 }  
 return false;  
 }  
  
 public Department GetDepartament(int number)  
 {  
 Node current = \_head;  
 while (current != null)  
 {  
 if (current.Data.Number.Equals(number)) return current.Data;  
 current = current.Next;  
 }  
 throw new Exception("Отделения не существует");  
 }  
  
 public override string ToString()  
 {  
 var b = new StringBuilder();  
 b.Append($"Наименование Магазина - {ShopName}:\n");  
 int i = 0;  
 foreach (var item in this)  
 {  
 b.Append(" ");  
 b.Append($" {i} Отделение) {item.ToString()} \n");  
 i++;  
 }  
  
 return b.ToString();  
 }  
  
 public void Dispose()  
 {  
 ShopName = default;  
   
 Node current = \_head;  
 while (current != null)  
 {  
 RemoveDepartment(current.Data.Number);  
 current = current.Next;  
 }  
  
 \_head = null;  
 \_tail = null;  
 \_count = default;  
 }  
   
 public XElement WriteXml()  
 {  
 XElement shop = new XElement("Shop");  
 shop.Add(new XAttribute("name", ShopName));  
 foreach (var item in this)  
 {  
 shop.Add(item.WriteXml());  
 }  
 return shop;  
 }  
   
 IEnumerator IEnumerable.GetEnumerator()  
 {  
 return ((IEnumerable)this).GetEnumerator();  
 }  
   
 IEnumerator<Department> IEnumerable<Department>.GetEnumerator()  
 {  
 Node current = \_head;  
 while (current != null)  
 {  
 yield return current.Data;  
 current = current.Next;  
 }  
 }  
   
 public IEnumerable<Department> BackEnumerator()  
 {  
 Node current = \_tail;  
 while (current != null)  
 {  
 yield return current.Data;  
 current = current.Previous;  
 }  
 }  
 }  
}

## **Department.cs**

using System;  
using System.Xml.Linq;  
  
namespace Cursovik  
{  
 public class Department : IDisposable  
 {  
 public int Number { get; set; }  
 public string Profile { get; set; }  
  
 public Department(int number, string profile)  
 {  
 Number = number;  
 Profile = profile;  
 }  
  
 public override string ToString()  
 {  
 return $"Номер отделения - {Number}, Профиль отделения - {Profile}";  
 }  
  
 public void Dispose()  
 {  
 Number = default;  
 Profile = default;  
 }  
   
 public XElement WriteXml()  
 {  
 XElement department = new XElement("Department");  
 department.Add(new XAttribute("number", Number));  
 department.Add(new XAttribute("profile", Profile));  
 return department;  
 }  
 }  
}