Казанский Национальный Исследовательский Технический Университет им. А. Н. Туполева

Институт Клиентных Технологий и Защиты Информации

Кафедра Прикладной Математики и Информатики

Пояснительная записка к курсовой работе

по дисциплине «Структуры и Алгоритмы Обработки данных»

на тему

Статическая очередь неупорядоченных двунаправленных динамических списков

Выполнил:

студент группы 4212

Георгиев Д.С.

Проверил:

доц. кафедры ПМИ Сотников С.В.

Казань, 2022

**Содержание**

[Постановка задачи](#_heading=h.gjdgxs) 3

[Теоретическое описание всех разработанных классов с алгоритмами реализации основных операций](#_heading=h.2et92p0) 4

[Описание структуры проекта](#_heading=h.tyjcwt) 8

[Описание всех разработанных подпрограмм для работы со структурами](#_heading=h.w2nx3o6x4zhx) 15

[Схема взаимодействия классов](#_heading=h.1t3h5sf) 19

[Демонстрационный модуль](#_heading=h.wvzrp5qmxujh) 21

[Список используемой литературы](#_heading=h.5h9nbzeaofrs) 29

[Приложение. Листинг](#_heading=h.2s8eyo1) 30

# **Постановка задачи**

Цель работы:

Реализовать комбинированную структуру данных на основе объектного

подхода (возможные языки: Object Pascal, C#, С++)

Информационное наполнение:

Организация (название) – композиция отделов (название)

Отдел – композиция сотрудников (фамилия, должность)

Общие требования:

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. Именование типов, структур и их полей, классов и их свойств и методов в соответствии с конкретной информационной задачей
5. Наличие демо-модуля с удобным оконным пользовательским интерфейсом

# **Теоретическое описание всех разработанных классов с алгоритмами реализации основных операций**

**Статическая реализация очереди**

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. Пустая очередь: First = Last =1

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

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

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

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

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

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

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

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

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

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

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

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

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

Рассмотренная выше простейшая реализация очереди-массива имеет один существенный недостаток: освобождающиеся при удалении ячейки в начале массива НЕ используются при последующих добавлениях, и поэтому при интенсивном использовании очереди быстро может возникнуть ситуация, когда указатель Last выходит за пределы массива, тогда как в начале массива есть свободные ячейки.

Для устранения этого недостатка можно использовать два подхода:

* при очередном удалении элемента из начала очереди сдвигать все элементы влево на одну ячейку, что при большом числе элементов в очереди может привести к большим вычислительным затратам
* более эффективно использовать так называемую **кольцевую** очередь, в которой при достижении указателем Last конца массива добавление производится в начало массива:

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

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

1. Новое число 22 добавляется в первую ячейку:

First = 3 , Last = 2

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

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

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

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.

Само **добавление элемента** в очередь выполняется следующим образом:

* 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:

* 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

**Динамическая реализация неупорядоченного двунаправленного списка**

То, что список неупорядоченный, говорит нам о том, что добавление нового элемента осуществляется 2 способами:

* Перед заданным элементом (по информационной части)
* После заданного элемента (по информационной части)

Во всем остальном динамический неупорядоченный список – обычный динамический список.

Динамическая реализация линейного списка 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).

Для удобства реализации будем считать, что список содержит хотя бы один элемент-заголовок с адресом первого реального элемента списка. Эта переменная устанавливается при первоначальном создании списка и в дальнейшем не изменяется.

Проход по списку:

* ввести вспомогательную переменную Current для отслеживания текущего элемента списка и установить Current = (\*DinamList).Next (где DinamList – заголовочный элемент списка)
* организовать цикл по условию Current != DinamList, внутри которого обработать текущий элемент (\*DinamList).Inf и изменить указатель Current на следующий элемент: Current =(\*Current).Next

Добавление нового элемента включает:

* поиск места для нового элемента в списке с помощью вспомогательного указателя Current
* выделение памяти для нового элемента с помощью еще одного указателя Temp
* формирование полей нового элемента, в частности – настройка ссылочной части : (\*Temp).next = Current
* изменение ссылочных частей соседних элементов нового элемента

Поиск заданного элемента включает:

* установку вспомогательного указателя в адрес первого элемента списка Current = (\*DinamList).Next
* организацию цикла прохода по списку с завершением либо по совпадению информационной части элемента с заданным значением (\*Current ).Inf == Inf, либо по достижению заголовочного элемента списка Current = DinamList
* после завершения цикла проверить значение вспомогательного указателя и сделать вывод об успешности поиска

Удаление заданного элемента включает:

* поиск удаляемого элемента
* 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

# **Описание структуры проекта**

Для разработки была использована среда разработки ПО Microsoft Visual Studio 2019 с версией .NETFramework 4.7.2.

Описание всех разработанных структур

1. Класс **Employee**

/// <summary>

/// Класс, описывающий сотрудника какого-либо отдела

/// </summary>

**public** class Employee

**{**

#region Описание свойств класса

/// <summary>

/// Поле - фамилия сотрудника отдела

/// </summary>

**private** string \_serName **{** **get;** **set;** **}**

/// <summary>

/// Поле - должность сотрудника

/// </summary>

**private** string \_position **{** **get;** **set;** **}**

#endregion

#region Описание конструктора и деструктора класса

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="serName">Фамилия сотрудника</param>

/// <param name="position">Должность сотрудника</param>

**public** Employee**(**string serName**,** string position**)**

**{**

\_serName **=** serName**;**

\_position **=** position**;**

**}**

/// <summary>

/// Деструктор класса

/// </summary>

**public** void EmployeeDestructor**()**

**{**

\_serName **=** **null;**

\_position **=** **null;**

**}**

#endregion

**}**

2. Класс **Department**

/// <summary>

/// Класс, описывающий отдел какой-либо компании

/// </summary>

**public** class Department

**{**

#region Описание свойств класса

/// <summary>

/// Название отдела

/// </summary>

**private** string \_nameOfDepartment **{** **get;** **set;** **}**

/// <summary>

/// Список всех сотрудников отдела

/// </summary>

**private** List**<**Employee**>** \_departmentStaff **{** **get;** **set;** **}**

/// <summary>

/// Кол-во сотрудников отдела

/// </summary>

**private** int \_countOfDepartmentStaff **{** **get;** **set;** **}**

#endregion

#region Описание конструктора и деструктора класса

/// <summary>

/// Конструктор класса

/// </summary>

**public** Department**(**string nameOfDepartment**)**

**{**

\_nameOfDepartment **=** nameOfDepartment**;**

\_departmentStaff **=** **new** List**<**Employee**>();**

\_countOfDepartmentStaff **=** 0**;**

Console**.**Write**(**"\nВведите кол-во сотрудников: "**);**

var count **=** Convert**.**ToInt32**(**Console**.**ReadLine**());**

**for** **(**var i **=** 0**;** i **<** count**;** i**++)**

**{**

Console**.**Write**(**$"Введите фамилию сотрудника №{i}: "**);**

var serName **=** Console**.**ReadLine**();**

Console**.**Write**(**$"Введите должность сотрудника №{i}: "**);**

var position **=** Console**.**ReadLine**();**

AddNewEmployee**(**serName**,** position**);**

**}**

**}**

/// <summary>

/// Второй конструктор класса, используется для работы с файлом

/// </summary>

/// <param name="nameOfDepartment">Название отдела</param>

/// <param name="serName">Фамилия сотрудника</param>

/// <param name="position">Должность сотрудника</param>

**public** Department**(**string nameOfDepartment**,** string serName**,** string position**)**

**{**

\_nameOfDepartment **=** nameOfDepartment**;**

\_departmentStaff **=** **new** List**<**Employee**>();**

\_countOfDepartmentStaff **=** 0**;**

**if** **(**serName **!=** "" **&&** position **!=** ""**)**

**{**

AddNewEmployee**(**serName**,** position**);**

**}**

**}**

/// <summary>

/// Деструктор класса

/// </summary>

**public** void DepartmentDestructor**()**

**{**

\_nameOfDepartment **=** **null;**

\_departmentStaff **=** **null;**

\_countOfDepartmentStaff **=** 0**;**

**}**

#endregion

**}**

3. Класс **Company**

/// <summary>

/// Класс, описывающий какую-либо компанию

/// </summary>

**public** class Company

**{**

#region Описание свойств класса

/// <summary>

/// Название компании

/// </summary>

**private** string \_nameOfCompany **{** **get;** **set;** **}**

/// <summary>

/// Список всех отделов компании

/// </summary>

**private** List**<**Department**>** \_companyDepartments **{** **get;** **set;** **}**

/// <summary>

/// Кол-во отделов компании

/// </summary>

**private** int \_countOfCompanyDepartments **{** **get;** **set;** **}**

#endregion

#region Описание конструктора и деструктора класса

/// <summary>

/// Конструктор класса

/// </summary>

**public** Company**(**string nameOfCompany**)**

**{**

\_nameOfCompany **=** nameOfCompany**;**

\_companyDepartments **=** **new** List**<**Department**>();**

\_countOfCompanyDepartments **=** 0**;**

Console**.**Write**(**"\nВведите кол-во отделов: "**);**

var count **=** Convert**.**ToInt32**(**Console**.**ReadLine**());**

**for** **(** var i **=** 0**;** i **<** count**;** i**++)**

**{**

Console**.**Write**(**$"\nВведите название отдела №{i}: "**);**

var name **=** Console**.**ReadLine**();**

AddNewDepartment**(**name**);**

**}**

**}**

/// <summary>

/// Второй конструктор класса, используется для работы с файлом

/// </summary>

/// <param name="nameOfCompany">Название компании</param>

/// <param name="nameOfDepartment">Название отдела</param>

/// <param name="serName">Фамилия сотрудника</param>

/// <param name="position">Должность сотрудника</param>

**public** Company**(**string nameOfCompany**,** string nameOfDepartment**,** string serName**,** string position**)**

**{**

\_nameOfCompany **=** nameOfCompany**;**

\_companyDepartments **=** **new** List**<**Department**>();**

\_countOfCompanyDepartments **=** 0**;**

**if** **(**nameOfDepartment **!=** ""**)**

**{**

AddNewDepartment**(**nameOfDepartment**,** serName**,** position**);**

**}**

**}**

/// <summary>

/// Деструктор класса

/// </summary>

**public** void CompanyDestructor**()**

**{**

\_nameOfCompany **=** **null;**

\_companyDepartments **=** **null;**

\_countOfCompanyDepartments **=** 0**;**

**}**

#endregion

**}**

4. Класс **ItemOfUnorderedBidirectionalDynamicList**

/// <summary>

/// Класс, описывающий элемент неупорядоченного двунапрявленного динамического списка

/// </summary>

**public** class ItemOfUnorderedBidirectionalDynamicList

**{**

#region Описание свойств класса

/// <summary>

/// Информационная часть элемента

/// </summary>

**private** Company \_data **{** **get;** **set;** **}**

/// <summary>

/// Левый сосед

/// </summary>

**private** ItemOfUnorderedBidirectionalDynamicList \_left **{** **get;** **set;** **}**

/// <summary>

/// Правый сосед

/// </summary>

**private** ItemOfUnorderedBidirectionalDynamicList \_right **{** **get;** **set;** **}**

#endregion

#region Описание конструктора и деструктора класса

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="nameOfCompany">Название компании</param>

/// <param name="numbOfCompany">Номер компании</param>

**public** ItemOfUnorderedBidirectionalDynamicList**(**string nameOfCompany**)**

**{**

\_data **=** **new** Company**(**nameOfCompany**);**

\_left **=** **null;** // Пока что обнуляем левого соседа

\_right **=** **null;** // Пока что обнуляем правого соседа

**}**

/// <summary>

/// Второй конструктор класса, используется при работе с файлами

/// </summary>

/// <param name="nameOfCompany">Название компании</param>

/// <param name="nameOfDepartment">Название отдела</param>

/// <param name="serName">Фамилия сотрудника</param>

/// <param name="position">Должность сотрудника</param>

**public** ItemOfUnorderedBidirectionalDynamicList**(**string nameOfCompany**,** string nameOfDepartment**,** string serName**,** string position**)**

**{**

\_data **=** **new** Company**(**nameOfCompany**,** nameOfDepartment**,** serName**,** position**);**

\_left **=** **null;** // Пока что обнуляем левого соседа

\_right **=** **null;** // Пока что обнуляем правого соседа

**}**

/// <summary>

/// Деструктор класса

/// </summary>

**public** void ItemOfUnorderedBidirectionalDynamicListDestructor**()**

**{**

\_data **=** **null;**

\_left **=** **null;**

\_right **=** **null;**

**}**

#endregion

**}**

5. Класс **UnorderedBidirectionalDynamicList**

/// <summary>

/// Класс, описывающий неупорядоченный двунаправленный динамический список

/// </summary>

**public** class UnorderedBidirectionalDynamicList

**{**

#region Описание свойств класса

/// <summary>

/// Заголовок списка

/// </summary>

**private** string \_title **{** **get;** **set;** **}**

/// <summary>

/// Головной элемент списка

/// </summary>

**private** ItemOfUnorderedBidirectionalDynamicList \_head **{** **get;** **set;** **}**

/// <summary>

/// Хвостовой элемент списка

/// </summary>

**private** ItemOfUnorderedBidirectionalDynamicList \_tail **{** **get;** **set;** **}**

/// <summary>

/// Кол-во элементов списка

/// </summary>

**private** int \_count **{** **get;** **set;** **}**

#endregion

#region Описание конструктора и деструктора класса

/// <summary>

/// Конструктор класса

/// </summary>

**public** UnorderedBidirectionalDynamicList**(**int numb**)**

**{**

\_count **=** 0**;**

\_title **=** $"Список №{numb}"**;**

\_head **=** **null;**

\_tail **=** **null;**

Dialog**();**

**}**

/// <summary>

/// Второй конструктор класса, используется при работе с файлами

/// </summary>

/// <param name="numb">Номер списка</param>

/// <param name="index"></param>

**public** UnorderedBidirectionalDynamicList**(**int numb**,** int index**)**

**{**

\_count **=** 0**;**

\_title **=** $"Список №{numb}"**;**

\_head **=** **null;**

\_tail **=** **null;**

**}**

/// <summary>

/// Деструктор класса

/// </summary>

**public** void UnorderedBidirectionalDynamicListDestructor**()**

**{**

\_count **=** 0**;**

\_title **=** **null;**

\_head **=** **null;**

\_tail **=** **null;**

**}**

#endregion

**}**

6. Класс **StaticQueueOfUnorderedBidirectionalDynamicLists**

/// <summary>

/// Класс, описывающий статическую очередь неупорядоченных двунаправленных динамических списков

/// </summary>

**public** class StaticQueueOfUnorderedBidirectionalDynamicLists

**{**

#region Описание свойств класса

/// <summary>

/// Массив с неупорядоченными двунаправленными динамическими списками

/// </summary>

**private** UnorderedBidirectionalDynamicList**[]** \_array **{** **get;** **set;** **}**

/// <summary>

/// Начало очереди

/// </summary>

**private** int \_first **{** **get;** **set;** **}**

/// <summary>

/// Конец очереди

/// </summary>

**private** int \_last **{** **get;** **set;** **}**

/// <summary>

/// Кол-во элементов в очереди

/// </summary>

**private** int \_count **{** **get;** **set;}**

/// <summary>

/// Размер очереди

/// </summary>

**private** int \_size **{** **get;** **set;** **}**

#endregion

#region Описание конструктора и деструктора класса

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="size">Размер очереди</param>

**public** StaticQueueOfUnorderedBidirectionalDynamicLists**()**

**{**

Console**.**Write**(**"Введите кол-во элементов очереди: "**);**

var size **=** Convert**.**ToInt32**(**Console**.**ReadLine**());**

\_size **=** size**;**

\_count **=** 0**;**

\_first **=** 0**;**

\_last **=** 0**;**

\_array **=** **new** UnorderedBidirectionalDynamicList**[**size**];**

**}**

/// <summary>

/// Деструктор класса

/// </summary>

**public** void StaticQueueOfUnorderedBidirectionalDynamicListsDestructor**()**

**{**

\_size **=** 0**;**

\_count **=** 0**;**

\_first **=** 0**;**

\_last **=** 0**;**

\_array **=** **null;**

**}**

#endregion

**}**

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

Класс **Employee**

| Название метода | Описание |
| --- | --- |
| void GetSerNameAndPosition2() | Вывод строку с информацией о фамилии и должности сотрудника |
| string GetSerName() | Гет-метод, возвращающий фамилию сотрудника |
| string GetPosition() | Гет-метод, возвращающий должность сотрудника |

Класс **Company**

| Название метода | Описание |
| --- | --- |
| void AddNewDepartment(string nameOfDepartment) | Метод добавления нового отдела |
| void AddNewDepartment(string nameOfDepartment, string serName, string position) | Метод добавления нового отдела, используется для работы с файлом |
| void GetNameOfCompany2() | Метод выводит название компании и выполняет вывод отделов |
| string GetNameOfCompany() | Гет-метод возвращает название компании |
| List<Department> GetAllDepartment() | Гет-метод возвращает список с отделами |
| Department SearchDepartment(string nameOfDepartment) | Метод поиска какого-то конкретного отдела |

Класс **Department**

| Название метода | Описание |
| --- | --- |
| void AddNewEmployee(string serName, string position) | Метод добавления нового сотрудника |
| void GetNameOfDepartment2() | Метод выводит название отделов и выполняет вывод сотрудников |
| string GetNameOfDepartment() | Гет-метод возвращает название отдела |
| List<Employee> GetAllEmployee() | Гет-метод возвращает список с сотрудниками отдела |

Класс **ItemOfUnorderedBidirectionalDynamicList**

| Название метода | Описание |
| --- | --- |
| Company GetCompany() | Гед-метод возвращает объект класса Company |
| ItemOfUnorderedBidirectionalDynamicList GetLeftNeighbor() | Гед-метод возвращает левого соседа |
| ItemOfUnorderedBidirectionalDynamicList GetRightNeighbor() | Гед-метод возвращает правого соседа |
| void SetLeftNeighbor(ItemOfUnorderedBidirectionalDynamicList leftNeighbor) | Сет-метод устанавливает левого соседа для элемента |
| void SetRightNeighbor(ItemOfUnorderedBidirectionalDynamicList rightNeighbor) | Сет-метод устанавливает правого соседа для элемента |

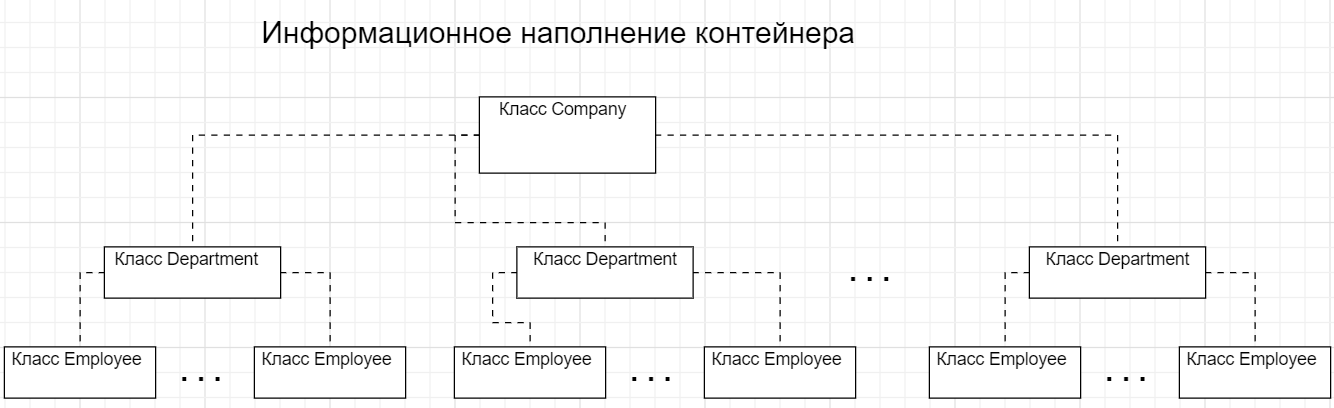
Класс **UnorderedBidirectionalDynamicList**

| Название метода | Описание |
| --- | --- |
| void Dialog() | Диалоговый метод общения с пользователем |
| void AddNewItemBefore(string nameOfCompany) | Метод добавления нового элемента перед заданным по информационной части |
| void AddNewItemAfter(string nameOfCompany) | Метод добавления нового элемента после заданного по информационной части |
| AddNewItem(ItemOfUnorderedBidirectionalDynamicList newItem, string nameOfCompany) | Метод добавления нового элемента в конец |
| List<Department> GetAllDepartment() | Гет-метод возвращает список с отделами |
| Department SearchDepartment(string nameOfDepartment) | Метод поиска какого-то конкретного отдела |
| ItemOfUnorderedBidirectionalDynamicList SearchNext(string nameOfCompany) | Метод поиска элемента по информационной части (название компании) в прямом направлении |
| ItemOfUnorderedBidirectionalDynamicList SearchPrevious(string nameOfCompany) | Метод поиска элемента по информационной части (название компании) в обратном направлении |
| void SearchElementInForwardDirection() | Метод поиска элемента по информационной части в прямом направлении |
| void SearchElementBackwards() | Метод поиска элемента по информационной части в обратном направлении |
| void ShowForward() | Метод вывода списка в прямом направлении |
| void ShowBackwards() | Метод вывода списка в обратном направлении |
| void RemoveItem() | метод удаления элемента по заданной информационной части |
| bool Empty() | Метод проверки на пустоту списка |
| string GetTitle() | Гет-метод возвращает заголовок списка |
| ItemOfUnorderedBidirectionalDynamicList GetHeadItem() | Гет-метод возвращает головной элемент |

Класс **StaticQueueOfUnorderedBidirectionalDynamicLists**

| Название метода | Описание |
| --- | --- |
| bool Occupancy() | Метод проверки на заполненность очереди |
| bool Empty() | Метод проверки на пустоту очереди |
| void ShowQueue() | Метод вывода очереди |
| void Add() | Метод добавления элемента очереди |
| void Add(int indexOfList) | Метод добавления элемента очереди, используется при работе с файлами |
| void Remove() | Метод удаления элемента очереди |
| void SelectList() | Метод выбора списка для дальнейшей работы с ним |
| void InputToFile() | Метод записи в файл |
| void OutputFromFile() | Метод берет информацию из файла |
| void Menu(UnorderedBidirectionalDynamicList element) | Метод работы со списком |

# **Схема взаимодействия классов**



Каждый объект класса Company - композиция классов Department

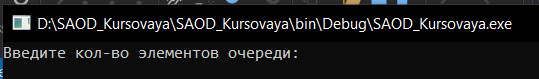
Каждый объект класса Department - композиция классов Employee



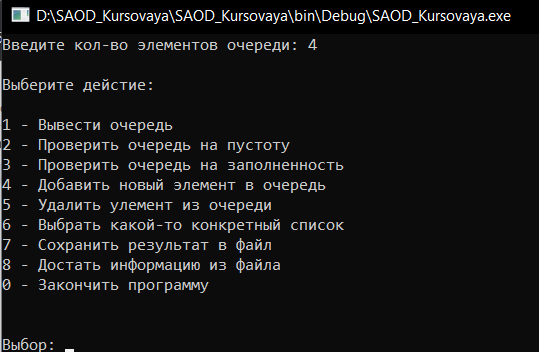
Объект класса StaticQueueOfUnorderedBidirectionalDynamicLists описывает статическую очередь, элементами которой являются неупорядоченные двунаправленные динамические списки. Объекты класса UnorderedBidirectionalDynamicList хранят в себе ссылку на головной элемент списка \_head и на хвостовой элемент \_tail. Класс ItemOfUnorderedBidirectionalDynamicList представляет собой элемент списка, которые связаны между собой указателями prev (предыдущий элемент) и next (следующий элемент).

# **Демонстрационный модуль**

В начале работы программы пользователь должен ввести кол-во элементов очереди

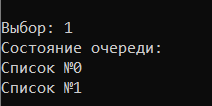


Далее перед пользователем выводится меню работы со статической очередью:



При выборе:

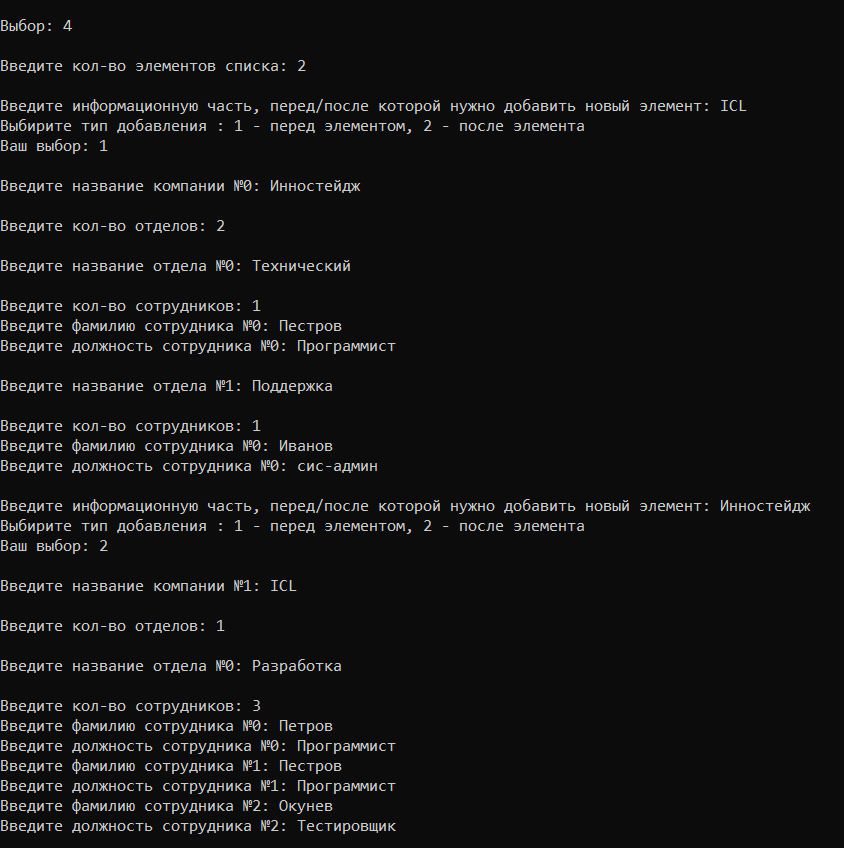
1. В консоль выводятся состояние очереди

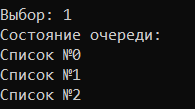


1. В консоль выведется ответ, пуста ли очередь
2. В консоль выведется ответ, заполнена ли очередь

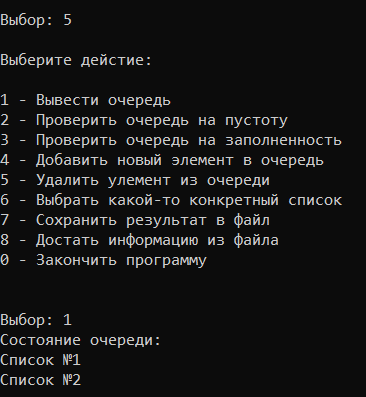


1. Пользователю предлагается ввести данные для добавления нового списка в очередь

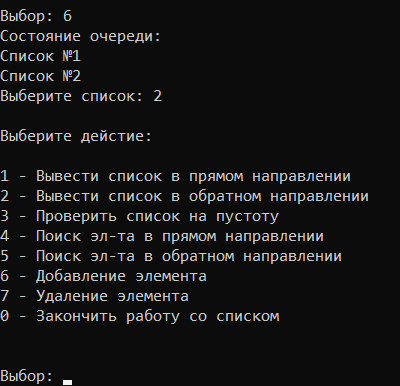




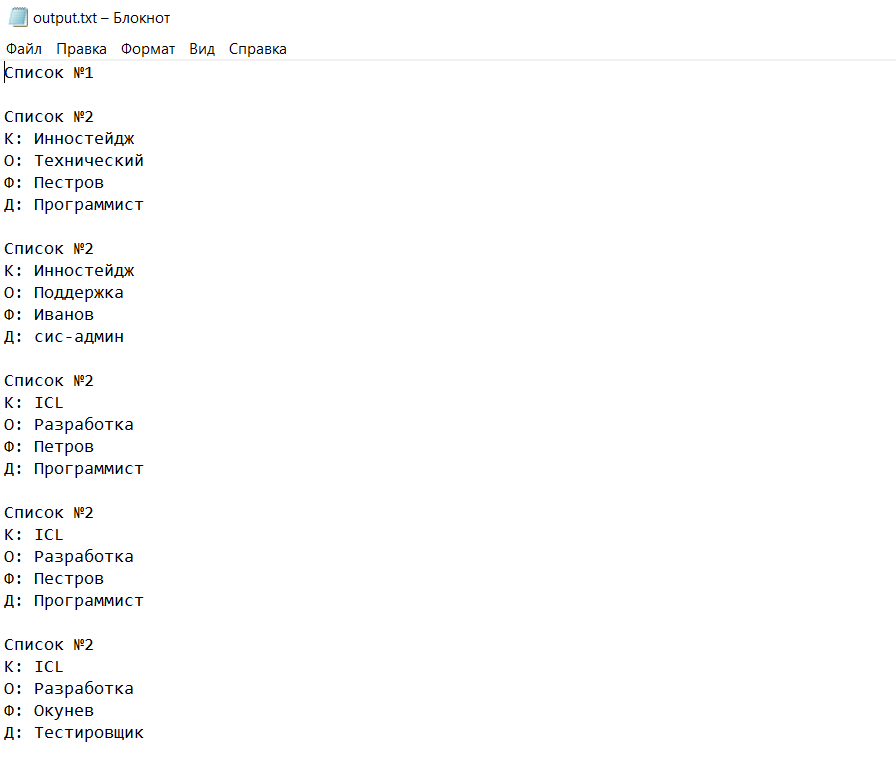
1. Удаляется элемент (список) из очереди



1. Пользователю предлагается выбрать какой-то конкретный список, чтобы выполнить с ним какие-то операции

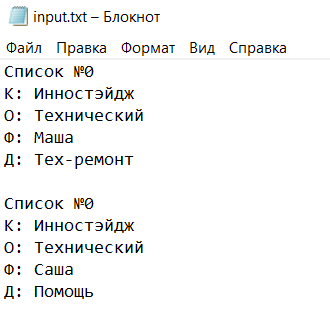


1. Выполняется сохранение результатов в файл, путь к которому указан внутри программы



1. Выполняется считывание данных из файла и заполнение очереди (и списков) этими данными.

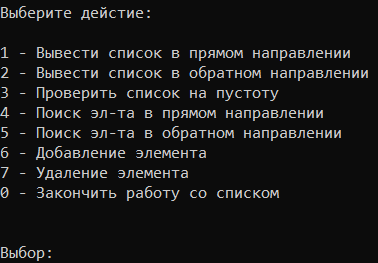
Чтобы добавление было корректным, файл должен иметь следующую структуру:



Вначале указывается из какого номера списка данные. После указывается название компании, затем название отдела этой компании, а после фамилия и должность сотрудника. После каждой такой записи должна стоять пустая строка.

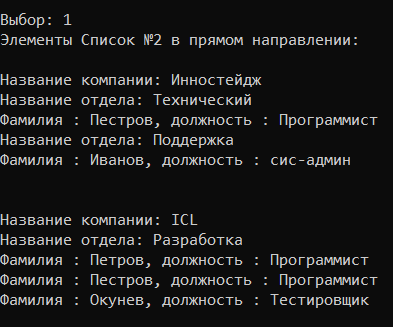
При выборе 0 программа завершается

**Работа со списком**

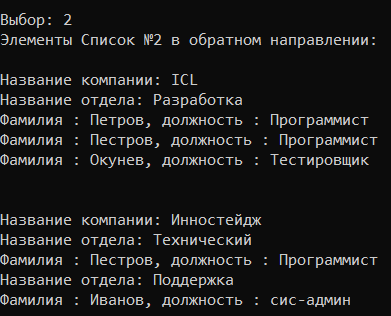
Пользователю предлагается выбрать действие

Выбрав:

1. Выводится список в прямом направлении



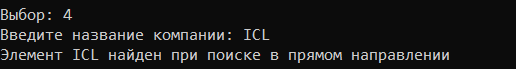
1. Выводится список в обратном направлении



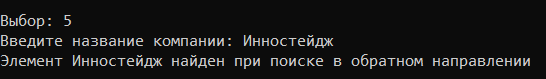
1. Программа даст ответ, пуст ли список



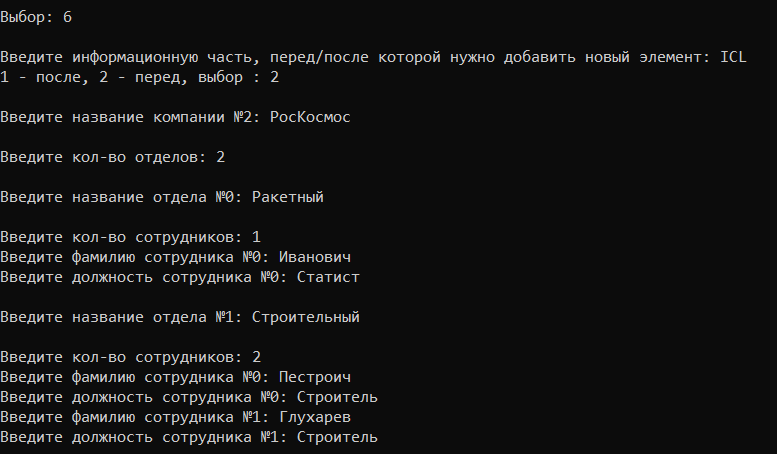
1. Пользователю предлагается ввести данные, для нахождения в прямом направлении

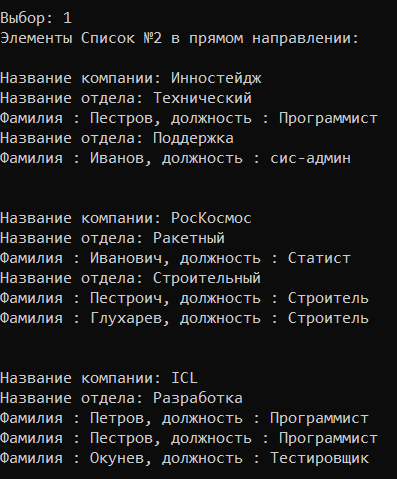


1. Пользователю предлагается ввести данные, для нахождения в обратном направлении

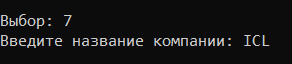


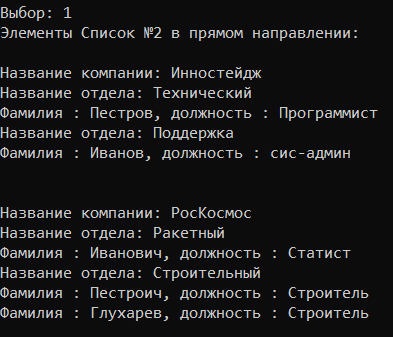
1. Пользователю предлагается ввести данные для добавления нового элемента в список



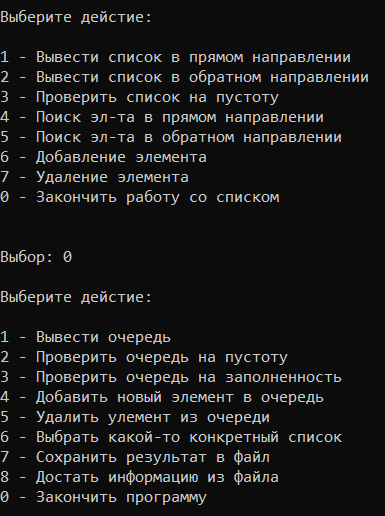


1. Пользователю предлагается ввести данные для удаления элемента





При выборе 0 программа выйдет в меню работы с очередью



# **Список используемой литературы**

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

# **Приложение. Листинг**

Класс **Employee**

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Linq**;**

**using** System**.**Text**;**

**using** System**.**Threading**.**Tasks**;**

**namespace** SAOD\_Kursovaya

**{**

/// <summary>

/// Класс, описывающий сотрудника какого-либо отдела

/// </summary>

**public** class Employee

**{**

#region Описание свойств класса

/// <summary>

/// Поле - фамилия сотрудника отдела

/// </summary>

**private** string \_serName **{** **get;** **set;** **}**

/// <summary>

/// Поле - должность сотрудника

/// </summary>

**private** string \_position **{** **get;** **set;** **}**

#endregion

#region Описание конструктора и деструктора класса

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="serName">Фамилия сотрудника</param>

/// <param name="position">Должность сотрудника</param>

**public** Employee**(**string serName**,** string position**)**

**{**

\_serName **=** serName**;**

\_position **=** position**;**

**}**

/// <summary>

/// Деструктор класса

/// </summary>

**public** void EmployeeDestructor**()**

**{**

\_serName **=** **null;**

\_position **=** **null;**

**}**

#endregion

#region Описание методов класса

/// <summary>

/// Функция выполняет вывод сотрудников

/// </summary>

**public** void GetSerNameAndPosition2**()**

**{**

Console**.**WriteLine**(**$"Фамилия : {\_serName}, должность : {\_position}"**);**

**}**

/// <summary>

/// Функция вызвращает фамилию сотрудника

/// </summary>

/// <returns>Фамилия сотрудника</returns>

**public** string GetSerName**()**

**{**

**return** $"Ф: {\_serName}"**;**

**}**

/// <summary>

/// Функция вызвращает должность сотрудника

/// </summary>

/// <returns>Должность сотрудника</returns>

**public** string GetPosition**()**

**{**

**return** $"Д: {\_position}"**;**

**}**

#endregion

**}**

**}**

Класс **Department**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Text;**

**using System.Threading.Tasks;**

**namespace SAOD\_Kursovaya**

**{**

**/// <summary>**

**/// Класс, описывающий отдел какой-либо компании**

**/// </summary>**

**public class Department**

**{**

**#region Описание свойств класса**

**/// <summary>**

**/// Название отдела**

**/// </summary>**

**private string \_nameOfDepartment { get; set; }**

**/// <summary>**

**/// Список всех сотрудников отдела**

**/// </summary>**

**private List<Employee> \_departmentStaff { get; set; }**

**/// <summary>**

**/// Кол-во сотрудников отдела**

**/// </summary>**

**private int \_countOfDepartmentStaff { get; set; }**

**#endregion**

**#region Описание конструктора и деструктора класса**

**/// <summary>**

**/// Конструктор класса**

**/// </summary>**

**public Department(string nameOfDepartment)**

**{**

**\_nameOfDepartment = nameOfDepartment;**

**\_departmentStaff = new List<Employee>();**

**\_countOfDepartmentStaff = 0;**

**Console.Write("\nВведите кол-во сотрудников: ");**

**var count = Convert.ToInt32(Console.ReadLine());**

**for (var i = 0; i < count; i++)**

**{**

**Console.Write($"Введите фамилию сотрудника №{i}: ");**

**var serName = Console.ReadLine();**

**Console.Write($"Введите должность сотрудника №{i}: ");**

**var position = Console.ReadLine();**

**AddNewEmployee(serName, position);**

**}**

**}**

**/// <summary>**

**/// Второй конструктор класса, используется для работы с файлом**

**/// </summary>**

**/// <param name="nameOfDepartment">Название отдела</param>**

**/// <param name="serName">Фамилия сотрудника</param>**

**/// <param name="position">Должность сотрудника</param>**

**public Department(string nameOfDepartment, string serName, string position)**

**{**

**\_nameOfDepartment = nameOfDepartment;**

**\_departmentStaff = new List<Employee>();**

**\_countOfDepartmentStaff = 0;**

**if (serName != "" && position != "")**

**{**

**AddNewEmployee(serName, position);**

**}**

**}**

**/// <summary>**

**/// Деструктор класса**

**/// </summary>**

**public void DepartmentDestructor()**

**{**

**\_nameOfDepartment = null;**

**\_departmentStaff = null;**

**\_countOfDepartmentStaff = 0;**

**}**

**#endregion**

**#region Описание методов класса**

**/// <summary>**

**/// Функция добавления нового сотрудника**

**/// </summary>**

**/// <param name="serName">Фамилия сотрудника</param>**

**/// <param name="position">Должность сотрудника</param>**

**public void AddNewEmployee(string serName, string position)**

**{**

**var newEmployee = new Employee(serName, position); // Композиция сотрудников**

**\_departmentStaff.Add(newEmployee);**

**\_countOfDepartmentStaff++;**

**}**

**/// <summary>**

**/// Функция выводит название отделов и выполняет вывод сотрудников**

**/// </summary>**

**public void GetNameOfDepartment2()**

**{**

**Console.WriteLine($"Название отдела: {\_nameOfDepartment}");**

**foreach (var el in \_departmentStaff)**

**{**

**el.GetSerNameAndPosition2();**

**}**

**}**

**/// <summary>**

**/// Функция возвращает название отдела**

**/// </summary>**

**/// <returns>Название отдела</returns>**

**public string GetNameOfDepartment()**

**{**

**return \_nameOfDepartment;**

**}**

**/// <summary>**

**/// Функция возвращает список с сотрудниками отдела**

**/// </summary>**

**/// <returns>Список с сотрудниками отдела</returns>**

**public List<Employee> GetAllEmployee()**

**{**

**return \_departmentStaff;**

**}**

**#endregion**

**}**

**}**

Класс **Company**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Text;**

**using System.Threading.Tasks;**

**namespace SAOD\_Kursovaya**

**{**

**/// <summary>**

**/// Класс, описывающий какую-либо компанию**

**/// </summary>**

**public class Company**

**{**

**#region Описание свойств класса**

**/// <summary>**

**/// Название компании**

**/// </summary>**

**private string \_nameOfCompany { get; set; }**

**/// <summary>**

**/// Список всех отделов компании**

**/// </summary>**

**private List<Department> \_companyDepartments { get; set; }**

**/// <summary>**

**/// Кол-во отделов компании**

**/// </summary>**

**private int \_countOfCompanyDepartments { get; set; }**

**#endregion**

**#region Описание конструктора и деструктора класса**

**/// <summary>**

**/// Конструктор класса**

**/// </summary>**

**public Company(string nameOfCompany)**

**{**

**\_nameOfCompany = nameOfCompany;**

**\_companyDepartments = new List<Department>();**

**\_countOfCompanyDepartments = 0;**

**Console.Write("\nВведите кол-во отделов: ");**

**var count = Convert.ToInt32(Console.ReadLine());**

**for ( var i = 0; i < count; i++)**

**{**

**Console.Write($"\nВведите название отдела №{i}: ");**

**var name = Console.ReadLine();**

**AddNewDepartment(name);**

**}**

**}**

**/// <summary>**

**/// Второй конструктор класса, используется для работы с файлом**

**/// </summary>**

**/// <param name="nameOfCompany">Название компании</param>**

**/// <param name="nameOfDepartment">Название отдела</param>**

**/// <param name="serName">Фамилия сотрудника</param>**

**/// <param name="position">Должность сотрудника</param>**

**public Company(string nameOfCompany, string nameOfDepartment, string serName, string position)**

**{**

**\_nameOfCompany = nameOfCompany;**

**\_companyDepartments = new List<Department>();**

**\_countOfCompanyDepartments = 0;**

**if (nameOfDepartment != "")**

**{**

**AddNewDepartment(nameOfDepartment, serName, position);**

**}**

**}**

**/// <summary>**

**/// Деструктор класса**

**/// </summary>**

**public void CompanyDestructor()**

**{**

**\_nameOfCompany = null;**

**\_companyDepartments = null;**

**\_countOfCompanyDepartments = 0;**

**}**

**#endregion**

**#region Описание методов класса**

**/// <summary>**

**/// Функция добавления нового отдела**

**/// </summary>**

**/// <param name="nameOfDepartment">Название отдела</param>**

**public void AddNewDepartment(string nameOfDepartment)**

**{**

**var newDepartment = new Department(nameOfDepartment); // Композиция отделов**

**\_companyDepartments.Add(newDepartment);**

**\_countOfCompanyDepartments++;**

**}**

**/// <summary>**

**/// Функция добавления нового отдела, используется для работы с файлом**

**/// </summary>**

**/// <param name="nameOfDepartment">Название отдела</param>**

**/// <param name="serName">Фамилия сотрудника</param>**

**/// <param name="position">Должность сотрудника</param>**

**public void AddNewDepartment(string nameOfDepartment, string serName, string position)**

**{**

**var newDepartment = new Department(nameOfDepartment, serName, position); // Композиция отделов**

**\_companyDepartments.Add(newDepartment);**

**\_countOfCompanyDepartments++;**

**}**

**/// <summary>**

**/// Функция выводит название компании и выполняет вывод отделов**

**/// </summary>**

**public void GetNameOfCompany2()**

**{**

**Console.WriteLine($"\n\nНазвание компании: {\_nameOfCompany}");**

**foreach (var el in \_companyDepartments)**

**{**

**el.GetNameOfDepartment2();**

**}**

**}**

**/// <summary>**

**/// Функция возвращает название компании**

**/// </summary>**

**/// <returns>Строка - название компании</returns>**

**public string GetNameOfCompany()**

**{**

**return \_nameOfCompany;**

**}**

**/// <summary>**

**/// Функция возвращает список с отделами**

**/// </summary>**

**/// <returns>Список отделов</returns>**

**public List<Department> GetAllDepartment()**

**{**

**return \_companyDepartments;**

**}**

**/// <summary>**

**/// Функция поиска какого-то конкретного отдела**

**/// </summary>**

**/// <param name="nameOfDepartment">Название отдела</param>**

**/// <returns>Объект класса Department</returns>**

**public Department SearchDepartment(string nameOfDepartment)**

**{**

**if (\_companyDepartments != null)**

**{**

**foreach (var el in \_companyDepartments)**

**{**

**if (el.GetNameOfDepartment() == nameOfDepartment)**

**{**

**return el;**

**}**

**}**

**return null;**

**}**

**else**

**{**

**return null;**

**}**

**}**

**#endregion**

**}**

**}**

Класс **ItemOfUnorderedBidirectionalDynamicList**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Text;**

**using System.Threading.Tasks;**

**namespace SAOD\_Kursovaya**

**{**

**/// <summary>**

**/// Класс, описывающий элемент неупорядоченного двунапрявленного динамического списка**

**/// </summary>**

**public class ItemOfUnorderedBidirectionalDynamicList**

**{**

**#region Описание свойств класса**

**/// <summary>**

**/// Информационная часть элемента**

**/// </summary>**

**private Company \_data { get; set; }**

**/// <summary>**

**/// Левый сосед**

**/// </summary>**

**private ItemOfUnorderedBidirectionalDynamicList \_left { get; set; }**

**/// <summary>**

**/// Правый сосед**

**/// </summary>**

**private ItemOfUnorderedBidirectionalDynamicList \_right { get; set; }**

**#endregion**

**#region Описание конструктора и деструктора класса**

**/// <summary>**

**/// Конструктор класса**

**/// </summary>**

**/// <param name="nameOfCompany">Название компании</param>**

**/// <param name="numbOfCompany">Номер компании</param>**

**public ItemOfUnorderedBidirectionalDynamicList(string nameOfCompany)**

**{**

**\_data = new Company(nameOfCompany);**

**\_left = null; // Пока что обнуляем левого соседа**

**\_right = null; // Пока что обнуляем правого соседа**

**}**

**/// <summary>**

**/// Второй конструктор класса, используется при работе с файлами**

**/// </summary>**

**/// <param name="nameOfCompany">Название компании</param>**

**/// <param name="nameOfDepartment">Название отдела</param>**

**/// <param name="serName">Фамилия сотрудника</param>**

**/// <param name="position">Должность сотрудника</param>**

**public ItemOfUnorderedBidirectionalDynamicList(string nameOfCompany, string nameOfDepartment, string serName, string position)**

**{**

**\_data = new Company(nameOfCompany, nameOfDepartment, serName, position);**

**\_left = null; // Пока что обнуляем левого соседа**

**\_right = null; // Пока что обнуляем правого соседа**

**}**

**/// <summary>**

**/// Деструктор класса**

**/// </summary>**

**public void ItemOfUnorderedBidirectionalDynamicListDestructor()**

**{**

**\_data = null;**

**\_left = null;**

**\_right = null;**

**}**

**#endregion**

**#region Описание методов класса**

**/// <summary>**

**/// Фцнкция возвращает объект класса Company**

**/// </summary>**

**/// <returns>Объект класса Company</returns>**

**public Company GetCompany()**

**{**

**return \_data;**

**}**

**/// <summary>**

**/// Функция возвращает левого соседа**

**/// </summary>**

**/// <returns>Объект класса ItemOfUnorderedBidirectionalDynamicList</returns>**

**public ItemOfUnorderedBidirectionalDynamicList GetLeftNeighbor()**

**{**

**return \_left;**

**}**

**/// <summary>**

**/// Функция возвращает правого соседа**

**/// </summary>**

**/// <returns>Объект класса ItemOfUnorderedBidirectionalDynamicList</returns>**

**public ItemOfUnorderedBidirectionalDynamicList GetRightNeighbor()**

**{**

**return \_right;**

**}**

**/// <summary>**

**/// Функция устанавливает левого соседа для элемента**

**/// </summary>**

**/// <param name="leftNeighbor">Левый сосед - объект класса Company</param>**

**public void SetLeftNeighbor(ItemOfUnorderedBidirectionalDynamicList leftNeighbor)**

**{**

**\_left = leftNeighbor;**

**}**

**/// <summary>**

**/// Функция устанавливает правого соседа для элемента**

**/// </summary>**

**/// <param name="rightNeighbor">Правый сосед - объект класса Company</param>**

**public void SetRightNeighbor(ItemOfUnorderedBidirectionalDynamicList rightNeighbor)**

**{**

**\_right = rightNeighbor;**

**}**

**#endregion**

**}**

**}**

Класс **UnorderedBidirectionalDynamicList**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Text;**

**using System.Threading.Tasks;**

**namespace SAOD\_Kursovaya**

**{**

**/// <summary>**

**/// Класс, описывающий неупорядоченный двунаправленный динамический список**

**/// </summary>**

**public class UnorderedBidirectionalDynamicList**

**{**

**#region Описание свойств класса**

**/// <summary>**

**/// Заголовок списка**

**/// </summary>**

**private string \_title { get; set; }**

**/// <summary>**

**/// Головной элемент списка**

**/// </summary>**

**private ItemOfUnorderedBidirectionalDynamicList \_head { get; set; }**

**/// <summary>**

**/// Хвостовой элемент списка**

**/// </summary>**

**private ItemOfUnorderedBidirectionalDynamicList \_tail { get; set; }**

**/// <summary>**

**/// Кол-во элементов списка**

**/// </summary>**

**private int \_count { get; set; }**

**#endregion**

**#region Описание коструктора и деструктора класса**

**/// <summary>**

**/// Конструктор класса**

**/// </summary>**

**public UnorderedBidirectionalDynamicList(int numb)**

**{**

**\_count = 0;**

**\_title = $"Список №{numb}";**

**\_head = null;**

**\_tail = null;**

**Dialog();**

**}**

**/// <summary>**

**/// Второй конструктор класса, используется при работе с файлами**

**/// </summary>**

**/// <param name="numb">Номер списка</param>**

**/// <param name="index"></param>**

**public UnorderedBidirectionalDynamicList(int numb, int index)**

**{**

**\_count = 0;**

**\_title = $"Список №{numb}";**

**\_head = null;**

**\_tail = null;**

**}**

**/// <summary>**

**/// Деструктор класса**

**/// </summary>**

**public void UnorderedBidirectionalDynamicListDestructor()**

**{**

**\_count = 0;**

**\_title = null;**

**\_head = null;**

**\_tail = null;**

**}**

**#endregion**

**#region Описание методов класса**

**/// <summary>**

**/// Диалоговя функции общения с пользователем**

**/// </summary>**

**public void Dialog()**

**{**

**Console.Write("\nВведите кол-во элементов списка: ");**

**var count = Convert.ToInt32(Console.ReadLine()); // Вводится кол-во создаваемых элементов в списке**

**for (var i = 0; i < count; i++)**

**{**

**Console.Write("\nВведите информационную часть, перед/после которой нужно добавить новый элемент: ");**

**var nameOfCompany = Console.ReadLine();**

**Console.Write("Выбирите тип добавления : 1 - перед элементом, 2 - после элемента\nВаш выбор: ");**

**var inputOption = Console.ReadLine();**

**while (inputOption != "1" && inputOption != "2")**

**{**

**Console.Write("\nНеизвестная команда, повторите ввод: ");**

**inputOption = Console.ReadLine();**

**}**

**if (inputOption == "1")**

**{**

**AddNewItemBefore(nameOfCompany);**

**}**

**else if (inputOption == "2")**

**{**

**AddNewItemAfter(nameOfCompany);**

**}**

**}**

**}**

**/// <summary>**

**/// Функция добавления нового элемента перед заданным по информационной части**

**/// </summary>**

**public void AddNewItemBefore(string nameOfCompany)**

**{**

**Console.Write($"\nВведите название компании №{\_count}: "); // Запрашивает у пользователя название компании**

**var nameOfNewCompany = Console.ReadLine();**

**var newItem = new ItemOfUnorderedBidirectionalDynamicList(nameOfNewCompany); // Новый, добавляемый, элемент списка**

**if (\_head == null) // Если список пуст**

**{**

**\_head = newItem;**

**\_tail = \_head;**

**}**

**else // Если в списке есть хотя бы 1 элемент**

**{**

**var searchingItem = SearchNext(nameOfCompany);**

**if (searchingItem != null) // Элемент найден**

**{**

**if (Equals(searchingItem, \_head) == true) // Если найденный элемент - головной элемент списка**

**{**

**newItem.SetRightNeighbor(\_head);**

**\_head.SetLeftNeighbor(newItem);**

**\_head = newItem; // Новый элемент становится головным**

**}**

**else**

**{**

**var beforeItem = searchingItem.GetLeftNeighbor(); // Получаем соседа слева у найденного элемента**

**beforeItem.SetRightNeighbor(newItem);**

**searchingItem.SetLeftNeighbor(newItem);**

**newItem.SetRightNeighbor(searchingItem);**

**newItem.SetLeftNeighbor(beforeItem);**

**}**

**}**

**else // Элемент не найден**

**{**

**AddNewItem(newItem, nameOfNewCompany);**

**}**

**}**

**\_count++; // Увеличиваем число элементов списка**

**}**

**/// <summary>**

**/// Функция добавления нового элемента после заданного по информационной части**

**/// </summary>**

**public void AddNewItemAfter(string nameOfCompany)**

**{**

**Console.Write($"\nВведите название компании №{\_count}: "); // Запрашивает у пользователя название компании**

**var nameOfNewCompany = Console.ReadLine();**

**var newItem = new ItemOfUnorderedBidirectionalDynamicList(nameOfNewCompany); // Новый, добавляемый, элемент списка**

**if (\_head == null) // Если список пуст**

**{**

**\_head = newItem;**

**\_tail = \_head;**

**}**

**else // Если в списке есть хотя бы 1 элемент**

**{**

**var searchingItem = SearchNext(nameOfCompany);**

**if (searchingItem != null) // Элемент найден**

**{**

**if (Equals(searchingItem, \_tail) == true) // Если найденный элемент - хвостовой элемент списка**

**{**

**newItem.SetLeftNeighbor(\_tail);**

**\_tail.SetRightNeighbor(newItem);**

**\_tail = newItem; // Новый элемент становится хвостовым**

**}**

**else**

**{**

**var afterItem = searchingItem.GetRightNeighbor(); // Получаем соседа справа у найденного элемента**

**afterItem.SetLeftNeighbor(newItem);**

**searchingItem.SetRightNeighbor(newItem);**

**newItem.SetLeftNeighbor(searchingItem);**

**newItem.SetRightNeighbor(afterItem);**

**}**

**}**

**else // Элемент не найден**

**{**

**AddNewItem(newItem, nameOfNewCompany);**

**}**

**}**

**\_count++; // Увеличиваем число элементов списка**

**}**

**/// <summary>**

**/// Функция добавления нового элемента в конец**

**/// </summary>**

**/// <param name="newItem">Добавляемый элемент</param>**

**/// <param name="nameOfCompany">Название компании</param>**

**public void AddNewItem(ItemOfUnorderedBidirectionalDynamicList newItem, string nameOfCompany)**

**{**

**if (\_head == null) // Если список пуст**

**{**

**\_head = newItem;**

**\_tail = \_head;**

**}**

**else**

**{**

**\_tail.SetRightNeighbor(newItem);**

**newItem.SetLeftNeighbor(\_tail);**

**\_tail = newItem; // Новый элемент становится хвостовым**

**}**

**\_count++; // Увеличиваем число элементов списка**

**}**

**/// <summary>**

**/// Функция поиска элемента по информационной части (название компании) в прямом направлении**

**/// </summary>**

**/// <param name="nameOfCompany">Название компании, которое пользователь хочет найти</param>**

**/// <returns>Элемент списка</returns>**

**public ItemOfUnorderedBidirectionalDynamicList SearchNext(string nameOfCompany)**

**{**

**var current = \_head;**

**if (current == null)**

**{**

**return null;**

**}**

**while (current.GetRightNeighbor() != null)**

**{**

**if (current.GetCompany().GetNameOfCompany() == nameOfCompany)**

**{**

**return current;**

**}**

**current = current.GetRightNeighbor();**

**}**

**if (current.GetCompany().GetNameOfCompany() == nameOfCompany)**

**{**

**return current;**

**}**

**return null; // если компания не найдена**

**}**

**/// <summary>**

**/// Функция поиска элемента по информационной части (название компании) в обратном направлении**

**/// </summary>**

**/// <param name="nameOfCompany">Название компании, которое пользователь хочет найти</param>**

**/// <returns>Элемент списка</returns>**

**private ItemOfUnorderedBidirectionalDynamicList SearchPrevious(string nameOfCompany)**

**{**

**var current = \_tail;**

**while (current.GetLeftNeighbor() != null)**

**{**

**if (current.GetCompany().GetNameOfCompany() == nameOfCompany)**

**{**

**return current;**

**}**

**current = current.GetLeftNeighbor();**

**}**

**if (current.GetCompany().GetNameOfCompany() == nameOfCompany)**

**{**

**return current;**

**}**

**return null; // если компания не найдена**

**}**

**/// <summary>**

**/// Функция поиска элемента по информационной части в прямом направлении**

**/// </summary>**

**public void SearchElementInForwardDirection()**

**{**

**if (Empty() == true)**

**{**

**Console.WriteLine($"Список пуст, поиск невозможен");**

**return;**

**}**

**Console.Write("Введите название компании: "); // Запрашивает у пользователя название компании**

**var nameOfCompany = Console.ReadLine();**

**var current = SearchNext(nameOfCompany);**

**if (current == null) // Элемент не найден**

**{**

**Console.WriteLine($"Элемент {nameOfCompany} не найден при поиске в прямом направлении");**

**}**

**else**

**{**

**Console.WriteLine($"Элемент {nameOfCompany} найден при поиске в прямом направлении");**

**}**

**}**

**/// <summary>**

**/// Функция поиска элемента по информационной части в обратном направлении**

**/// </summary>**

**public void SearchElementBackwards()**

**{**

**if (Empty() == true)**

**{**

**Console.WriteLine($"Список пуст, поиск невозможен");**

**return;**

**}**

**Console.Write("Введите название компании: "); // Запрашивает у пользователя название компании**

**var nameOfCompany = Console.ReadLine();**

**var current = SearchPrevious(nameOfCompany);**

**if (current == null) // Элемент не найден**

**{**

**Console.WriteLine($"Элемент {nameOfCompany} не найден при поиске в обратном направлении");**

**}**

**else**

**{**

**Console.WriteLine($"Элемент {nameOfCompany} найден при поиске в обратном направлении");**

**}**

**}**

**/// <summary>**

**/// Функция вывода списка в прямом направдении**

**/// </summary>**

**public void ShowForward()**

**{**

**if (Empty() == true)**

**{**

**Console.WriteLine($"Список пуст");**

**return;**

**}**

**var current = \_head;**

**Console.Write($"Элементы {\_title} в прямом направлении: ");**

**while (current.GetRightNeighbor() != null)**

**{**

**current.GetCompany().GetNameOfCompany2();**

**current = current.GetRightNeighbor();**

**}**

**current.GetCompany().GetNameOfCompany2();**

**Console.WriteLine("\n");**

**}**

**/// <summary>**

**/// Функция вывода списка в обратном направлении**

**/// </summary>**

**public void ShowBackwards()**

**{**

**if (Empty() == true)**

**{**

**Console.WriteLine($"Список пуст");**

**return;**

**}**

**var current = \_tail;**

**Console.Write($"Элементы {\_title} в обратном направлении: ");**

**while (current.GetLeftNeighbor() != null)**

**{**

**current.GetCompany().GetNameOfCompany2();**

**current = current.GetLeftNeighbor();**

**}**

**current.GetCompany().GetNameOfCompany2();**

**Console.WriteLine("\n");**

**}**

**/// <summary>**

**/// Функция удаления элемента по заданной информационной части**

**/// </summary>**

**public void RemoveItem()**

**{**

**if (Empty() == true)**

**{**

**Console.WriteLine($"Список пуст, удаление невозможно");**

**return;**

**}**

**Console.Write("Введите название компании: "); // Запрашивает у пользователя название компании**

**var nameOfCompany = Console.ReadLine();**

**var searchingItem = SearchNext(nameOfCompany);**

**if (searchingItem != null) // Элемент найден**

**{**

**if (\_count == 1)**

**{**

**\_head = null;**

**\_tail = null;**

**searchingItem = null; // Удаляем элемент**

**}**

**else if (Equals(searchingItem, \_head) == true) // Если удаляемый элемент - головной элемент**

**{**

**var removeItem = \_head;**

**\_head = \_head.GetRightNeighbor(); // Головной элемент - следующий элемент после \_head**

**\_head.SetLeftNeighbor(null);**

**removeItem = null; // удаляем элемент**

**}**

**else if (Equals(searchingItem, \_tail) == true) // ЕСли удаляемый элемент - хвостовой элемент**

**{**

**var removeItem = \_tail;**

**\_tail = \_tail.GetLeftNeighbor(); // Хвостовой элемент - предыдущий элемент перед \_tail**

**\_tail.SetRightNeighbor(null);**

**removeItem = null; // удаляем элемент**

**}**

**else**

**{**

**var nextItem = searchingItem.GetRightNeighbor();**

**var previousItem = searchingItem.GetLeftNeighbor();**

**nextItem.SetLeftNeighbor(previousItem);**

**previousItem.SetRightNeighbor(nextItem);**

**searchingItem = null; // Удаление элемента**

**}**

**\_count--; // Уменьшаем число элементов списка**

**}**

**else**

**{**

**Console.WriteLine($"Элемент {nameOfCompany} в списке не найден, удаление элемента не произошло");**

**}**

**}**

**/// <summary>**

**/// Проверка на пустоту списка**

**/// </summary>**

**/// <returns>true - если список пуст, иначе - false</returns>**

**public bool Empty()**

**{**

**if (\_count == 0)**

**{**

**return true;**

**}**

**else**

**{**

**return false;**

**}**

**}**

**/// <summary>**

**/// Функция возвращает заголовок списка**

**/// </summary>**

**/// <returns></returns>**

**public string GetTitle()**

**{**

**return \_title;**

**}**

**/// <summary>**

**/// Функция возвращает головной элемент**

**/// </summary>**

**/// <returns>Лбъкт класса ItemOfUnorderedBidirectionalDynamicList</returns>**

**public ItemOfUnorderedBidirectionalDynamicList GetHeadItem()**

**{**

**return \_head;**

**}**

**#endregion**

**}**

**}**

Класс **StaticQueueOfUnorderedBidirectionalDynamicLists**

**using System;**

**using System.Collections.Generic;**

**using System.IO;**

**using System.Linq;**

**using System.Text;**

**using System.Threading.Tasks;**

**namespace SAOD\_Kursovaya**

**{**

**/// <summary>**

**/// Класс, описывающий статическую очередь неупорядоченных двунаправленных динамических списков**

**/// </summary>**

**public class StaticQueueOfUnorderedBidirectionalDynamicLists**

**{**

**#region Описание свойств класса**

**/// <summary>**

**/// Массив с неупорядоченными двунаправленными динамическими списками**

**/// </summary>**

**private UnorderedBidirectionalDynamicList[] \_array { get; set; }**

**/// <summary>**

**/// Начало очереди**

**/// </summary>**

**private int \_first { get; set; }**

**/// <summary>**

**/// Конец очереди**

**/// </summary>**

**private int \_last { get; set; }**

**/// <summary>**

**/// Кол-во элементов в очереди**

**/// </summary>**

**private int \_count { get; set;}**

**/// <summary>**

**/// Размер очереди**

**/// </summary>**

**private int \_size { get; set; }**

**#endregion**

**#region Описание конструктора и деструктора класса**

**/// <summary>**

**/// Конструктор класса**

**/// </summary>**

**/// <param name="size">Размер очереди</param>**

**public StaticQueueOfUnorderedBidirectionalDynamicLists()**

**{**

**Console.Write("Введите кол-во элементов очереди: ");**

**var size = Convert.ToInt32(Console.ReadLine());**

**\_size = size;**

**\_count = 0;**

**\_first = 0;**

**\_last = 0;**

**\_array = new UnorderedBidirectionalDynamicList[size];**

**}**

**/// <summary>**

**/// Деструктор класса**

**/// </summary>**

**public void StaticQueueOfUnorderedBidirectionalDynamicListsDestructor()**

**{**

**\_size = 0;**

**\_count = 0;**

**\_first = 0;**

**\_last = 0;**

**\_array = null;**

**}**

**#endregion**

**#region Описание методов класса**

**/// <summary>**

**/// Функция проверки на заполненность очереди**

**/// </summary>**

**/// <returns>true - если заполнена, иначе - false</returns>**

**public bool Occupancy()**

**{**

**if (\_size == \_count) // Очередь заполнена**

**{**

**return true;**

**}**

**else**

**{**

**return false;**

**}**

**}**

**/// <summary>**

**/// Функция проверки на пустоту очереди**

**/// </summary>**

**/// <returns>true - если пуста, иначе - false</returns>**

**public bool Empty()**

**{**

**if (\_count == 0) // Очередь пуста**

**{**

**return true;**

**}**

**else**

**{**

**return false;**

**}**

**}**

**/// <summary>**

**/// Функция вывода очереди**

**/// </summary>**

**public void ShowQueue()**

**{**

**if (Empty() == true)**

**{**

**Console.WriteLine("Очередь пуста");**

**return;**

**}**

**if (\_last > \_first)**

**{**

**Console.WriteLine("Состояние очереди:");**

**for (var i = \_first; i < \_last; i++)**

**{**

**Console.WriteLine($"{\_array[i].GetTitle()}");**

**}**

**}**

**else**

**{**

**Console.WriteLine("Состояние очереди:");**

**for (var i = \_first; i < \_size; i++)**

**{**

**Console.WriteLine($"{\_array[i].GetTitle()}");**

**}**

**for (var i = 0; i < \_last; i++)**

**{**

**Console.WriteLine($"{\_array[i].GetTitle()}");**

**}**

**}**

**}**

**/// <summary>**

**/// Функция добавления элемента очереди**

**/// </summary>**

**public void Add()**

**{**

**if (Occupancy() == true)**

**{**

**Console.WriteLine("В очереди нет места, добавление невозможно");**

**return;**

**}**

**\_array[\_last] = new UnorderedBidirectionalDynamicList(\_last); // Добавление элемента**

**\_last++;**

**if (\_last >= \_size)**

**{**

**\_last = 0;**

**}**

**\_count++;**

**}**

**/// <summary>**

**/// Функция добавления элемента очереди, используется при работе с файлами**

**/// </summary>**

**/// <param name="indexOfList"></param>**

**public void Add(int indexOfList)**

**{**

**if (Occupancy() == true)**

**{**

**Console.WriteLine("В очереди нет места, добавление невозможно");**

**return;**

**}**

**\_array[\_last] = new UnorderedBidirectionalDynamicList(\_last, indexOfList); // Добавление элемента**

**\_last++;**

**if (\_last >= \_size)**

**{**

**\_last = 0;**

**}**

**\_count++;**

**}**

**/// <summary>**

**/// Функция удаления элемента очереди**

**/// </summary>**

**public void Remove()**

**{**

**if (Empty() == true)**

**{**

**Console.WriteLine("Очередь пуста, удаление невозможно");**

**return;**

**}**

**\_array[\_first].UnorderedBidirectionalDynamicListDestructor(); // Вызов деструктора списка**

**\_array[\_first] = null; // Удаление элемента**

**\_first++;**

**if (\_first >= \_size)**

**{**

**\_first = 0;**

**}**

**\_count--;**

**}**

**/// <summary>**

**/// Функция выбора списка для дальнейшей работы с ним**

**/// </summary>**

**public void SelectList()**

**{**

**ShowQueue();**

**if (Empty() == false)**

**{**

**Console.Write("Выберите список: ");**

**var choose = Convert.ToInt32(Console.ReadLine());**

**Menu(\_array[choose]);**

**}**

**}**

**/// <summary>**

**/// Функция записи в файл**

**/// </summary>**

**public void InputToFile()**

**{**

**if (Empty() == true)**

**{**

**Console.WriteLine("Очередь пуста, нет смысла записывать её в файл");**

**return;**

**}**

**var str = "";**

**foreach (var el in \_array)**

**{**

**if (el != null)**

**{**

**var nameOfList = el.GetTitle(); // Название списка**

**var head = el.GetHeadItem();**

**var k = 0;**

**while (head != null)**

**{**

**k++;**

**var company = head.GetCompany();**

**var nameOfCompany = company.GetNameOfCompany(); // Название компании**

**var j = 0;**

**foreach (var item in company.GetAllDepartment()) // Получение списка отделов**

**{**

**j++;**

**var nameOfDepartmen = item.GetNameOfDepartment(); // Название отдела**

**var i = 0;**

**foreach (var element in item.GetAllEmployee())**

**{**

**i++;**

**var nameOfEmployee = element.GetSerName(); // Фамилия сотрудника**

**var positionOfEmployee = element.GetPosition(); // Должность сотрудника**

**var newStr = $"{nameOfList}\nК: {nameOfCompany}\nО: {nameOfDepartmen}\n{nameOfEmployee}\n{positionOfEmployee}\n\n";**

**str += newStr;**

**}**

**if (i == 0)**

**{**

**var newStr = $"{nameOfList}\nК: {nameOfCompany}\nО: {nameOfDepartmen}\n\n";**

**str += newStr;**

**}**

**}**

**if (j == 0)**

**{**

**var newStr = $"{nameOfList}\nК: {nameOfCompany}\n\n";**

**str += newStr;**

**}**

**head = head.GetRightNeighbor();**

**}**

**if (k == 0)**

**{**

**var newStr = $"{nameOfList}\n\n";**

**str += newStr;**

**}**

**}**

**}**

**File.WriteAllText(@"D:\SAOD\_Kursovaya\SAOD\_Kursovaya\output.txt", str); // Сохранение в файл информации**

**}**

**/// <summary>**

**/// Функция берет информацию из файла**

**/// </summary>**

**public void OutputFromFile()**

**{**

**string[] lines = File.ReadAllLines(@"D:\SAOD\_Kursovaya\SAOD\_Kursovaya\input.txt"); // Считываем с файла информацию**

**int indexOfList = 0; // Индекс списка**

**string nameOfCompany = ""; // Название компании**

**string nameOfDepartment = ""; // Название отдела**

**string serName = ""; // Фамилия сотрудника**

**string position = ""; // Должность сотрудника**

**foreach (string el in lines)**

**{**

**if (el != "") // Проверяем на разбивку (пустую строку)**

**{**

**if (el[0] == 'С') // Получаем(считываем) индекс списка**

**{**

**var str = "";**

**for (var i = 8; i < el.Length; i++)**

**{**

**str += el[i];**

**}**

**indexOfList = Convert.ToInt32(str);**

**}**

**else if (el[0] == 'К') // Получаем(считываем) название компании**

**{**

**for (var i = 3; i < el.Length; i++)**

**{**

**nameOfCompany += el[i];**

**}**

**}**

**else if (el[0] == 'О') // Получаем(считываем) название отдела**

**{**

**for (var i = 3; i < el.Length; i++)**

**{**

**nameOfDepartment += el[i];**

**}**

**}**

**else if (el[0] == 'Ф') // Получаем(считываем) фамилию сотрудника**

**{**

**for (var i = 3; i < el.Length; i++)**

**{**

**serName += el[i];**

**}**

**}**

**else if (el[0] == 'Д') // Получаем(считываем) должность сотрудника**

**{**

**for (var i = 3; i < el.Length; i++)**

**{**

**position += el[i];**

**}**

**}**

**}**

**else**

**{**

**if (\_array[indexOfList] == null) // Если списка нет, то добавляем новый список**

**{**

**Add(indexOfList);**

**}**

**if (\_array[indexOfList].SearchNext(nameOfCompany) == null && nameOfCompany != "") // Если компании нет, то добавляем новую компанию**

**{**

**ItemOfUnorderedBidirectionalDynamicList newItem = new ItemOfUnorderedBidirectionalDynamicList(nameOfCompany, nameOfDepartment, serName, position);**

**\_array[indexOfList].AddNewItem(newItem, nameOfCompany);**

**}**

**else // Если компания есть, то проверяем дальше**

**{**

**if (\_array[indexOfList].SearchNext(nameOfCompany).GetCompany().SearchDepartment(nameOfDepartment) == null && nameOfDepartment != "") // Если отдела нет, то добавляем новый отдел**

**{**

**\_array[indexOfList].SearchNext(nameOfCompany).GetCompany().AddNewDepartment(nameOfDepartment, serName, position);**

**}**

**else // Если отдел есть, то дабовляем в него человека**

**{**

**\_array[indexOfList].SearchNext(nameOfCompany).GetCompany().SearchDepartment(nameOfDepartment).AddNewEmployee(serName, position);**

**}**

**}**

**indexOfList = 0;**

**nameOfCompany = "";**

**nameOfDepartment = "";**

**serName = "";**

**position = "";**

**}**

**}**

**}**

**#endregion**

**#region Меню роботы со списком**

**/// <summary>**

**/// Функция работы со списком**

**/// </summary>**

**/// <param name="element">Список</param>**

**public void Menu(UnorderedBidirectionalDynamicList element)**

**{**

**var run = true;**

**while (run)**

**{**

**Console.WriteLine("\nВыберите дейстие:\n\n" +**

**"1 - Вывести список в прямом направлении\n" +**

**"2 - Вывести список в обратном направлении\n" +**

**"3 - Проверить список на пустоту\n" +**

**"4 - Поиск эл-та в прямом направлении\n" +**

**"5 - Поиск эл-та в обратном направлении\n" +**

**"6 - Добавление элемента\n" +**

**"7 - Удаление элемента\n" +**

**"0 - Закончить работу со списком\n\n");**

**Console.Write("Выбор: ");**

**var x = Console.ReadLine();**

**if (x == "0")**

**{**

**run = false;**

**}**

**else if (x == "1")**

**{**

**element.ShowForward();**

**}**

**else if (x == "2")**

**{**

**element.ShowBackwards();**

**}**

**else if (x == "3")**

**{**

**Console.WriteLine($"Список пуст? - {element.Empty()}");**

**}**

**else if (x == "4")**

**{**

**element.SearchElementInForwardDirection();**

**}**

**else if (x == "5")**

**{**

**element.SearchElementBackwards();**

**}**

**else if (x == "6")**

**{**

**Console.Write("\nВведите информационную часть, перед/после которой нужно добавить новый элемент: ");**

**var nameOfCompany = Console.ReadLine();**

**Console.Write("1 - после, 2 - перед, выбор : ");**

**var y = Console.ReadLine();**

**if (y == "1")**

**{**

**element.AddNewItemAfter(nameOfCompany);**

**}**

**else if (y == "2")**

**{**

**element.AddNewItemBefore(nameOfCompany);**

**}**

**}**

**else if (x == "7")**

**{**

**element.RemoveItem();**

**}**

**else**

**{**

**Console.WriteLine("Неизвестная команда..");**

**}**

**}**

**}**

**#endregion**

**}**

**}**