Учреждение высшего образования

«Университет управления «ТИСБИ»

Факультет информационных технологий

Кафедра информационных технологий

КУРСОВАЯ РАБОТА

по дисциплине «Программная инженерия»

на тему: «Разработка объектной программы для задачи «Служба трудоустройства»

Выполнил: студент гр. ПИ-022

Фамилия И.О.

Проверил: ст. преподаватель

Якунина Е.А.

Казань 2022

Оглавление

[Постановка задачи 3](#_Toc105281351)

[1. Описание используемых структур данных с алгоритмами выполнения основных операций 5](#_Toc105281352)

[1.1 Адресный замкнутый (кольцевой) однонаправленный список без заголовка. 5](#_Toc105281353)

[1.2 Кольцевая очередь на основе обычного массива 8](#_Toc105281354)

[2. Краткие сведения об объектном подходе 10](#_Toc105281355)

[3. Формализованное описание разработанных классов 16](#_Toc105281356)

[4. Описание демонстрационного модуля с характеристикой использованных стандартных компонентов и списком реализованных обработчиков 22](#_Toc105281357)

[5. Описание структуры проекта в соответствии с использованным инструментом разработки 27](#_Toc105281358)

[Список литературы 29](#_Toc105281359)

[Листинг программы 31](#_Toc105281360)

Постановка задачи

**Цели работы:** отработка навыков курсов «программная инженерия» и «структуры и алгоритмы обработки данных».

**Постановка задачи:** разработать объектную программу для хранения и обработки данных о вакансии в службе трудоустройства. Программа должна поддерживать список организации-работодателей с указанием уникального наименования организации. Для каждой организации создаётся свой список вакансий с указанием должности и оклада.

Разработка программы включает в себя:

* определение необходимых объектов и способов их взаимодействия;
* формальное описание объектов в виде классов;
* программную реализацию всех необходимых методов, включая корректировку вкладов и подсчет текущего суммарного объема всех вкладов;
* всестороннее тестирование методов с помощью консольного (при разработке) и оконного (в окончательном варианте) приложения.

Для объединения организаций используется структура данных в виде адресного замкнутого (кольцевого) неупорядоченного однонаправленного списка с заголовком. Для объединения вакансий для каждой организации используется очередь на основе динамического массива со сдвигом элементов.

Разработка выполняется с учетом следующих требований:

1. имена классов, свойств и методов должны носить содержательный смысл и соответствовать информационной задаче
2. обязательное соблюдение принципа инкапсуляции – использование в классах только закрытых свойств и реализация необходимого набора методов доступа
3. наличие двух методов для сохранения всей объектной структуры во внешнем файле с обратной загрузкой, при этом стандартные механизмы сериализации разрешается использовать только как дополнение к самостоятельно реализованным методам
4. тестовое оконное приложение должно обладать удобным пользовательским интерфейсом с контролем вводимых данных и отображением текущего состояния объектной структуры с помощью списковых или табличных компонентов
5. стандартные контейнеры/коллекции (включая обобщенные классы) разрешается использовать только как дополнение к самостоятельно разработанным классам
6. в качестве языка разработки разрешается использовать Java, С, C++, Object/Free Pascal и соответствующие инструменты быстрой разработки приложений.
7. **Описание используемых структур данных с алгоритмами выполнения основных операций**

1.1 Адресный замкнутый (кольцевой) однонаправленный список без заголовка.

Каждый узел однонаправленного (односвязного) кольцевого списка содержит одно поле ссылочной переменной на следующий узел и поле с полезными няниными. Односвязный циклический список можно представить следующим образом:



Рис. 1: Блок схема добавления элемента в таблицу.

Узел однонаправленного кольцевого списка можно представить в виде структуры, аналогичной односвязному линейному списку.

class Node**{**

public**:**

Operation data**;**

Node **\*** pNext**;**

Node**(**Operation data**,** Node**\*** start**){**

**this->**data **=** data**;**

**this->**pNext **=** start**;**

**}**

**};**

Основные действия, производимые над элементами ОЦС:

* Инициализация списка
* Добавление узла в список
* Удаление узла из списка
* Вывод элементов списка
* Взаимообмен двух узлов списка

Поскольку список является циклическим, реализация отдельной функции для удаления корня списка не требуется. Инициализация списка

Рис. 2 Узел списка

Инициализация списка предназначена для создания корневого узла списка, у которого поле указателя на следующий элемент содержит адрес самого корневого элемента.

// конструктор списка

ListOfOperation**::**ListOfOperation**(**QString name**){**

ListOfOperation**::**set\_name**(**name**);**

operations\_counter **=** 0**;**

start **=** **nullptr;**

last **=** **nullptr;**

**}**

**// первое добавление**

start **=** **new** Node**(**data**,** start**);**

last **=** start**;**

last**->**pNext **=** start**;**

operations\_counter**++;**

Добавление узла в ОЦС

Функция добавления узла в список принимает два аргумента:

* Указатель на элемент, после которого происходит добавление
* Данные для добавляемого элемента.

Процедуру добавления элемента можно отобразить следующей

схемой:



Рис. 3: Добавление в список

Добавление элемента в ОЦС включает в себя следующие этапы:

* создание добавляемого узла и заполнение его поля данных;
* переустановка указателя узла, предшествующего добавляемому, на до бавляемый узел;
* установка указателя добавляемого узла на следующий узел (тот, на который указывал предшествующий узел).

Таким образом, функция добавления узла в ОЦС имеет вид, полностью аналогичный функции добавления узла в односвязный линейный список:

// добавляем в начало списка новые по дате транзакции

**}** **else** **if(**start**->**data**.**get\_date**()** **<** date**){**

temp **=** start**;**

start **=** **new** Node**(**data**,** temp**);**

last**->**pNext **=** start**;**

operations\_counter**++;**

// добавляем в конец списка старые по дате транзакции

**}** **else** **if** **(**last**->**data**.**get\_date**()** **>=** date**)** **{**

temp **=** last**;**

last **=** **new** Node**(**data**,** start**);**

temp**->**pNext **=** last**;**

operations\_counter**++;**

// добавление в середину перед датой растее

**}** **else{**

temp **=** start**;**

Node**\*** temp\_last**;** // [2] [8]

**while(**temp**->**data**.**get\_date**()** **>** date**){**

temp\_last **=** temp**;**

temp **=** temp**->**pNext**;**

**}**

Node**\*** create **=** **new** Node**(**data**,** temp**);**

temp\_last**->**pNext **=** create**;**

operations\_counter**++;**

**}**

Возвращаемым значением функции является адрес добавленного узла.

Удаление узла ОЦС

В качестве аргументов функции удаления узла ОЦС передается указатель на удаляемый узел. Поскольку список циклический, нет необходимости передавать указатель на корень списка. Функция возвращает указатель на узел, следующий за удаляемым элементом.

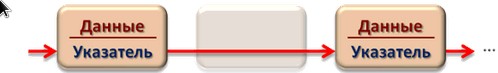
Удаление узла может быть представлено следующей схемой:

Рис. 4: Удаление узла

Удаление узла ОЦС включает в себя следующие этапы:

* установка указателя предыдущего узла на узел, следующий за удаляемым;
* освобождение памяти удаляемого узла.

**if(**operations\_counter **==** 1**){**

**delete** start**;**

operations\_counter **=** 0**;**

start **=** **nullptr;**

last **=** **nullptr;**

**return;**

**}**

temp **=** start**;**

last**->**pNext **=** start**->**pNext**;**

start **=** start**->**pNext**;**

**delete** temp**;**

operations\_counter**--;**

1.2 Кольцевая очередь на основе обычного массива

Кольцевая очередь работает по принципу FIFO (First – In – First – Out): «первым пришел - первым вышел». Отличие между кольцевой очередью и обычной очередью заключается в способе выхода из очереди первого элемента. В кольцевой очереди первый элемент перемещается в конец очереди. Примерами кольцевой очереди могут быть движение троллейбусов по кольцевому маршруту, круговорот воды в природе и т.д.

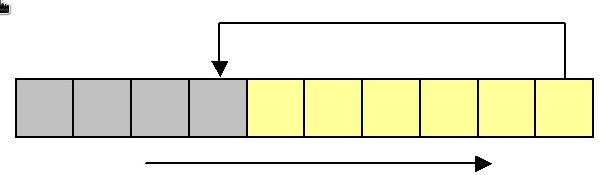


Рис. 5: Кольцевая очередь. Перемещение первого элемента в конец очереди

Кольцевая очередь работает по следующему принципу.

1. Сначала в очередь добавляются элементы до тех пор, пока не будет достигнут какой-то максимум. Элементы добавляются в конец очереди.
2. После достижения определенного (максимального) количества элементов происходит циклическое изменение первого элемента по следующему принципу: первый элемент очереди перемещается в конец очереди, а остальные элементы этой очереди сдвигаются на одну позицию впе ред. Таким образом второй элемент очереди становится первым, третий элемент становится вторым и т.д.

Базовый перечень выполняемых операций для кольцевой очереди следую щий:

* добавить новый элемент в очередь;
* проверка, есть ли очередь пустой;
* проверка, есть ли очередь полной;
* очистка очереди (удаление всех элементов очереди);
* вытянуть первый элемент из очереди и поместить в конец очереди.
* 
* Рис. 6 Добавление
* При добавление нового элемента данные добавляются в конец очереди, т.е. первую незанятую ячейку массива.
* 
* Рис 7. Удаление элемента.
* При удалении из памяти стирается первый элемента массива. Соседний элемент и все последующие смешаются на одну ячейку к началу массива.

1. Краткие сведения об объектном подходе

Класс (class) - элемент программы, который описывает какой-то тип данных. Класс описывает шаблон для создания объектов, как правило, указывает переменные этого объекта и действия, которые можно выполнять применимо к объекту.

Экземпляр класса (instance) - объект, который является представителем класса.

Метод (method) - функция, которая определена внутри класса и описывает какое-то действие, которое поддерживает класс

Переменная экземпляра (instance variable, а иногда и instance attribute) - данные, которые относятся к объекту

Переменная класса (class variable) - данные, которые относятся к классу и разделяются всеми экземплярами класса

Атрибут экземпляра (instance attribute) - переменные и методы, которые относятся к объектам (экземплярам) созданным на основании класса. У каждого объекта есть своя копия атрибутов.

Пример из реальной жизни в стиле ООП:

Проект дома - это класс

Конкретный дом, который был построен по проекту - экземпляр класса

Такие особенности как цвет дома, количество окон - переменные экземпляра, то есть конкретного дома

Дом можно продать, перекрасить, отремонтировать - это методы

Все основанные на объектах языки (C#, Java, С++, Smalltalk, Visual Basic и т.п.) должны отвечать трем основным принципам объектно-ориентированного программирования (ООП), которые перечислены ниже:

Инкапсуляция

Как данный язык скрывает детали внутренней реализации объектов и предохраняет целостность данных?

Наследование

Как данный язык стимулирует многократное использование кода?

Полиморфизм

Как данный язык позволяет трактовать связанные объекты сходным образом?

Прежде чем погрузиться в синтаксические детали реализации каждого принципа, важно понять базовую роль каждого из них.

Роль инкапсуляции

Инкапсуляция — это механизм программирования, объединяющий вместе код и данные, которыми он манипулирует, исключая как вмешательство извне, так и неправильное использование данных. В объектно-ориентированном языке данные и код могут быть объединены в совершенно автономный черный ящик. Внутри такого ящика находятся все необходимые данные и код. Когда код и данные связываются вместе подобным образом, создается объект. Иными словами, объект — это элемент, поддерживающий инкапсуляцию.

Т.е. инкапсуляция представляет собой способности языка скрывать излишние детали реализации от пользователя объекта. Например, предположим, что используется класс по имени DatabaseReader, который имеет два главных метода: Open() и Close().

Фиктивный класс DatabaseReader инкапсулирует внутренние детали нахождения, загрузки, манипуляций и закрытия файла данных. Программистам нравится инкапсуляция, поскольку этот принцип ООП упрощает кодирование. Нет необходимости беспокоиться о многочисленных строках кода, которые работают "за кулисами", чтобы реализовать функционирование класса DatabaseReader. Все, что потребуется — это создать экземпляр и отправлять ему соответствующие сообщения (например, "открыть файл по имени AutoLot.mdf, расположенный на диске С:").

С идеей инкапсуляции программной логики тесно связана идея защиты данных. В идеале данные состояния объекта должны быть специфицированы с использованием ключевого слова private (или, возможно, protected). Таким образом, внешний мир должен вежливо попросить, если захочет изменить или получить лежащее в основе значение. Это хороший принцип, поскольку общедоступные элементы данных можно легко повредить (даже нечаянно, а не преднамеренно).

Основной единицей инкапсуляции в C# является класс, который определяет форму объекта. Он описывает данные, а также код, который будет ими оперировать. В C# описание класса служит для построения объектов, которые являются экземплярами класса. Следовательно, класс, по существу, представляет собой ряд схематических описаний способа построения объекта.

Код и данные, составляющие вместе класс, называют членами. Данные, определяемые классом, называют полями, или переменными экземпляра. А код, оперирующий данными, содержится в функциях-членах, самым типичным представителем которых является метод. В C# метод служит в качестве аналога подпрограммы. (К числу других функций-членов относятся свойства, события и конструкторы.) Таким образом, методы класса содержат код, воздействующий на поля, определяемые этим классом.

Роль наследования

Следующий принцип ООП — наследование — касается способности языка позволять строить новые определения классов на основе определений существующих классов. По сути, наследование позволяет расширять поведение базового (или родительского) класса, наследуя основную функциональность в производном подклассе (также именуемом дочерним классом):

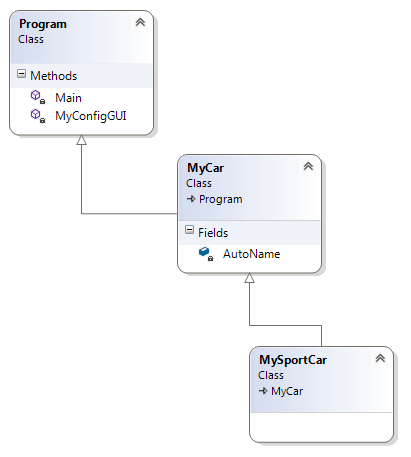


Рис 8 Принцип наследования

Т.е. наследование представляет собой процесс, в ходе которого один объект приобретает свойства другого объекта. Это очень важный процесс, поскольку он обеспечивает принцип иерархической классификации. Если вдуматься, то большая часть знаний поддается систематизации благодаря иерархической классификации по нисходящей.

Если не пользоваться иерархиями, то для каждого объекта пришлось бы явно определять все его свойства. А если воспользоваться наследованием, то достаточно определить лишь те свойства, которые делают объект особенным в его классе. Он может также наследовать общие свойства своего родителя. Следовательно, благодаря механизму наследования один объект становится отдельным экземпляром более общего класса.

Роль полиморфизма

Последний принцип ООП — полиморфизм. Он обозначает способность языка трактовать связанные объекты в сходной манере. В частности, этот принцип ООП позволяет базовому классу определять набор членов (формально называемый полиморфным интерфейсом), которые доступны всем наследникам. Полиморфный интерфейс класса конструируется с использованием любого количества виртуальных или абстрактных членов.

По сути, виртуальный член — это член базового класса, определяющий реализацию по умолчанию, которая может быть изменена (или, говоря более формально, переопределена) в производном классе. В отличие от него, абстрактный метод — это член базового класса, который не предусматривает реализации по умолчанию, а предлагает только сигнатуру. Когда класс наследуется от базового класса, определяющего абстрактный метод, этот метод обязательно должен быть переопределен в производном классе. В любом случае, когда производные классы переопределяют члены, определенные в базовом классе, они по существу переопределяют свою реакцию на один и тот же запрос.

Рассмотрим для примера стек, т.е. область памяти, функционирующую по принципу "последним пришел — первым обслужен". Допустим, что в программе требуются три разных типа стеков: один — для целых значений, другой — для значений с плавающей точкой, третий — для символьных значений. В данном примере алгоритм, реализующий все эти стеки, остается неизменным, несмотря на то что в них сохраняются разнотипные данные. В языке, не являющемся объектно-ориентированным, для этой цели пришлось бы создать три разных набора стековых подпрограмм с разными именами. Но благодаря полиморфизму для реализации всех трех типов стеков в C# достаточно создать лишь один общий набор подпрограмм. Зная, как пользоваться одним стеком, вы сумеете воспользоваться и остальными.

В более общем смысле понятие полиморфизма нередко выражается следующим образом: "один интерфейс — множество методов". Это означает, что для группы взаимосвязанных действий можно разработать общий интерфейс. Полиморфизм помогает упростить программу, позволяя использовать один и тот же интерфейс для описания общего класса действий. Выбрать конкретное действие (т.е. метод) в каждом отдельном случае — это задача компилятора. Программисту не нужно делать это самому. Ему достаточно запомнить и правильно использовать общий интерфейс.

3. Формализованное описание разработанных классов

Класс ListVacantion является самым главным в иеррархии классов. В нём содержаться данные о вакансии.

// ListVacantion

**using** System**;**

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

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

**namespace** \_3

**{**

class ListVacantion

**{**

**private** string work**;**

**private** int salary**;**

**public** ListVacantion**(**string \_work**,** int \_salary**)**

**{**

**this.**work **=** \_work**;**

**this.**salary **=** \_salary**;**

**}**

**public** string Work **{** **get** **{** **return** work**;** **}** **set** **{** **this.**work **=** **value;** **}** **}**

**public** int Salary **{** **get** **{** **return** salary**;** **}** **set** **{** **this.**salary **=** **value;** **}** **}**

**}**

**}**

Класс ListCompany создаёт внутри себя экземпляр класса ListVacantion и собирает вакансии в массив реализующий структуру данных очередь.

// ListCompany

**using** System**;**

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

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

**namespace** \_3

**{**

class ListCompany

**{**

**private** string nameCompany**;**

**private** ListVacantion**[]** Queue**;**

**private** int size**;**

**private** int first**;**

**private** int last**;**

**private** int count**;**

**public** ListCompany**(**string \_nameCompany**,** int \_arrSize**)**

**{**

**this.**size **=** \_arrSize**;**

**this.**nameCompany **=** \_nameCompany**;**

Queue **=** **new** ListVacantion**[**size**];**

first **=** 0**;**

last **=** 0**;**

count **=** 0**;**

**}**

**public** int Count **{** **get** **{** **return** count**;** **}** **}**

**public** ListVacantion**[]** QUeue **{** **get** **{** **return** Queue**;** **}** **}**

**public** string NameCompany**{** **get** **{** **return** nameCompany**;** **}** **set** **{** **this.**nameCompany **=** **value;** **}** **}**

**public** void AddVacantion**(**string \_work**,** int \_salary**)**

**{**

ListVacantion newList **=** **new** ListVacantion**(**\_work**,** \_salary**);**

**if** **(**count **>=** **(**0.75 **\*** size**))** **{** recizeArray**();** **}**

Queue**[**last**]** **=** newList**;**

last**++;**

count**++;**

**}**

**public** ListVacantion FindVacantion**(**string \_work**)**

**{**

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

**{**

**if** **(**Queue**[**i**]** **!=** **null)**

**{**

**if** **(**Queue**[**i**].**Work**.**Equals**(**\_work**))**

**{**

**return** Queue**[**i**];**

**}**

**}**

**}**

**return** **null;**

**}**

**public** void DeleteVacantion**()**

**{**

**if** **(**count **!=** 0**)**

**{**

Queue**[**first**]** **=** **null;**

count**--;**

**for** **(**int i **=** **(**first **+** 1**);** i **<** last**;** i**++)**

**{**

Queue**[**i **-** 1**]** **=** Queue**[**i**];**

**}**

last**--;**

Queue**[**last**]** **=** **null;**

**if** **(**count **>** 3 **&&** count **<=** **(**0.25 **\*** size**))** **{** recizeArray**();** **}**

**}**

**}**

**public** int SumVacantion**()**

**{**

**return** count**;**

**}**

**public** string chekFor**()**

**{**

string date **=** Convert**.**ToString**(**count**);**

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

**{**

date **+=** "\n" **+** Queue**[**i**].**Work**;**

date **+=** "\n" **+** Queue**[**i**].**Salary**;**

**}**

**return** date**;**

**}**

**public** string chek**()**

**{**

string date **=** "\n\n##### Organization " **+** nameCompany **+** "\n"**;**

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

**{**

date **+=** "\n\nvacantion: " **+** Queue**[**i**].**Work**;**

date **+=** "\nSalary: " **+** Queue**[**i**].**Salary**;**

**}**

date **+=**"\n\ncount vocation - " **+** SumVacantion**();**

date **+=**"\n#####################################"**;**

**return** date**;**

**}**

**public** void recizeArray**()**

**{**

**if** **(**count **>=** **(**0.75 **\*** size**))**

**{**

ListVacantion**[]** pTemp **=** **new** ListVacantion**[**size **\*** 2**];**

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

**{**

pTemp**[**i**]** **=** Queue**[**i**];**

**}**

size **\*=** 2**;**

Queue **=** **new** ListVacantion**[**size**];**

**for** **(**int i **=** 0**;** i **<** size**;** i**++)**

**{**

Queue**[**i**]** **=** pTemp**[**i**];**

**}**

**}**

**else** **if** **(**count **<=** **(**0.25 **\*** size**))**

**{**

ListVacantion**[]** pTemp **=** **new** ListVacantion**[**size **/** 2**];**

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

**{**

pTemp**[**i**]** **=** Queue**[**i**];**

**}**

size **/=** 2**;**

Queue **=** **new** ListVacantion**[**size**];**

**for** **(**int i **=** 0**;** i **<** size**;** i**++)**

**{**

Queue**[**i**]** **=** pTemp**[**i**];**

**}**

**}**

**}**

**}**

**}**

В классе Organization находятся данные о всех организациях с их ваквнмиями. Так же в данном классе создаётся экемпляр структуры списка котроый реализован в другом классе. Данный клас евляется вершиной всей иерархии.

// Organization

**using** System**;**

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

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

**namespace** \_3

**{**

class Organization

**{**

**private** string nameOrganization**;**//название организации

**private** StructList pHead**;**

**private** int count**;**

**public** Organization**(**string \_nameOrganization**)**

**{**

**this.**nameOrganization **=** \_nameOrganization**;**

pHead **=** **new** StructList**(null);**

pHead**.**First **=** **null;**

pHead**.**Last **=** **null;**

**}**

**public** string NameOrganization **{** **get;** **}**

**public** int Count **{** **get** **{** **return** count**;** **}** **}**

**public** StructList PHead **{** **get** **{** **return** pHead**;** **}** **}**

**public** void addComp**(**string \_nameCompany**,** int \_arrSize**)**

**{**

ListCompany newComp **=** **new** ListCompany**(**\_nameCompany**,** \_arrSize**);**

StructList vspom **=** **new** StructList**(**newComp**);**

count**++;**

**if** **(**pHead**.**First **==** **null** **&&** pHead**.**Last **==** **null)**

**{**

pHead**.**First **=** vspom**;**

pHead**.**Last **=** vspom**;**

pHead**.**First**.**Next **=** pHead**.**Last**;**

pHead**.**Last**.**Next **=** pHead**.**First**;**

**}**

**else**

**{**

//pHead.Last.Next = vspom;

pHead**.**Last **=** pHead**.**Last**.**Next **=** vspom**;**

pHead**.**Last**.**Next **=** pHead**.**First**;**

**}**

**}**

**public** ListCompany FindComp**(**string \_nameCompany**)**

**{**

StructList search **=** pHead**.**First**;**

**while** **(**search **!=** **null)**

**{**

**if** **(**search**.**Name **!=** **null)**

**{**

**if** **(**search**.**Name**.**NameCompany**.**Equals**(**\_nameCompany**))**

**{** **return** search**.**Name**;** **}**

**else** search **=** search**.**Next**;**

**}**

**}**

**return** **null;**

**}**

**public** void delAllComp**()**

**{**

pHead**.**First **=** **null;**

pHead**.**Last**=** **null;**

**}**

**public** bool delComp**(**String \_nameComp**)**

**{**

**if** **(**pHead **!=** **null)**

**{**

StructList vspom **=** pHead**.**First**;**

//StructList pTemp = pHead.First;

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

**{**

**if** **(**vspom **!=** **null)**

**{**

**if** **(**vspom **==** pHead**.**First**)**

**{**

**if** **(**vspom**.**Name**.**NameCompany**.**Equals**(**\_nameComp**))**

**{**

count**--;**

pHead**.**First **=** pHead**.**First**.**Next**;**

**return** **true;**

**}**

**else** **{** vspom **=** vspom**.**Next**;** **}**

**}**

**else**

**{**

**if** **(**vspom**.**Next**.**Name**.**NameCompany**.**Equals**(**\_nameComp**))**

**{**

count**--;**

vspom**.**Next **=** vspom**.**Next**.**Next**;**

**return** **true;**

**}**

**else** **{** vspom **=** vspom**.**Next**;** **}**

**}**

**}**

**}**

**}**

**return** **false;**

**}**

**public** string getQueueName**()**

**{**

string info **=** Convert**.**ToString**(**Count**);**

StructList pTemp **=** pHead**.**First**;**

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

**{**

info **+=** "\n" **+** pTemp**.**Name**.**NameCompany**;**

pTemp **=** pTemp**.**Next**;**

**}**

**return** info**;**

**}**

**public** string getQueueAll**()**

**{**

string info **=** ""**;**

StructList pTemp **=** pHead**.**First**;**

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

**{**

info **+=** "\n" **+** pTemp**.**Name**.**chekFor**();**

pTemp **=** pTemp**.**Next**;**

**}**

**return** info**;**

**}**

**public** string getQueue**()**

**{**

string date **=** "-=-=-=-=-=-Employment service-=-=-=-=-=-"**;**

StructList pTemp **=** pHead**.**First**;**

**do**

**{**

date **+=** pTemp**.**Name**.**chek**()** **+** "\n"**;**

pTemp **=** pTemp**.**Next**;**

**}** **while** **(**pTemp **!=** pHead**.**First**);**

date **+=** "\nTotal number of vacancies: " **+** getSumQueue**();**

**return** date**;**

**}**

**public** int getSumQueue**()**

**{**

StructList pTemp **=** pHead**.**First**;**

int summa **=** 0**;**

**do**

**{**

summa **+=** pTemp**.**Name**.**SumVacantion**();**

pTemp **=** pTemp**.**Next**;**

**}** **while** **(**pTemp **!=** pHead**.**First**);**

**return** summa**;**

**}**

**}**

**}**

Реализуется структура данных адресный замкнутый (кольцевой) однонаправленный список без заголовка.

// StructList

**using** System**;**

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

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

**namespace** \_3

**{**

class StructList

**{**

**public** ListCompany name**;** //данные о организации

**private** StructList next**;** //ссылка на потомка

**private** StructList last**;**

**private** StructList first**;**

**public** StructList**(**ListCompany \_date**)**

**{**

name **=** \_date**;**

next **=** **null;**

first **=** **null;**

last **=** **null;**

**}**

**public** ListCompany Name **{** **get** **{** **return** name**;** **}** **}**

**public** StructList Next

**{**

**get** **{** **return** next**;** **}**

**set** **{** **this.**next **=** **value;** **}**

**}**

**public** StructList Last

**{**

**get** **{** **return** last**;** **}**

**set** **{** **this.**last **=** **value;** **}**

**}**

**public** StructList First

**{**

**get** **{** **return** first**;** **}**

**set** **{** **this.**first **=** **value;** **}**

**}**

**}**

**}**

4. Описание демонстрационного модуля с характеристикой использованных стандартных компонентов и списком реализованных обработчиков

При первом запуске программы создаётся экземпляр класса Company с параметром HART необходимый для файлов сохранения и добавления. После создания экземпляра создаётся форма программы.

// Form1

**using** System**;**

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

**using** System**.**ComponentModel**;**

**using** System**.**Data**;**

**using** System**.**Drawing**;**

**using** System**.**IO**;**

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

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

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

**using** System**.**Windows**.**Forms**;**

**namespace** \_3

**{**

**public** **partial** class Form1 **:** Form

**{**

Organization comp **=** **new** Organization**(**"HART"**);**

**public** Form1**()**

**{**

InitializeComponent**();**

**}**

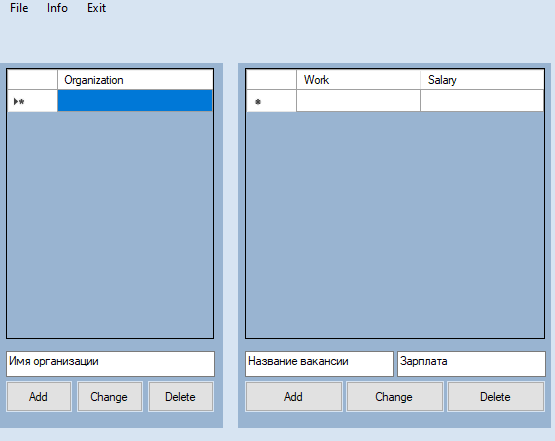


Рис. 8 Запуск программы.

Обрабатывается событие выбора элемента в таблице с компаниями. При нажатии на любую ячейку идёт обращение к соответствующему экземпляру организации и во вторую таблицу выводятся данные о вакансиях.

**private** void dataGridView1\_CellContentClick**(object** sender**,** DataGridViewCellEventArgs e**)**

**{**

dataGridView2**.**Rows**.**Clear**();**

string text **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

ListCompany vspom **=** comp**.**FindComp**(**text**);**

**if** **(**vspom **!=** **null)**

**{**

ListVacantion**[]** copy1 **=** vspom**.**QUeue**;**

int chet **=** comp**.**FindComp**(**text**).**Count**;**

**for** **(**int i **=** 0**;** i **<** chet**;** i**++)**

**{**

dataGridView2**.**Rows**.**Add**(**copy1**[**i**].**Work**,** copy1**[**i**].**Salary**);**

**}**

**}**

**}**

Валидаторы для TextBox.

**private** void textOrg\_Click**(object** sender**,** EventArgs e**)**

**{**

textOrg**.**Clear**();**

**}**

**private** void textWork\_Click**(object** sender**,** EventArgs e**)**

**{**

textWork**.**Clear**();**

**}**

**private** void textSal\_Click**(object** sender**,** EventArgs e**)**

**{**

textSal**.**Clear**();**

**}**

**private** void exitToolStripMenuItem\_Click**(object** sender**,** EventArgs e**)**

**{**

Close**();**

**}**

**private** void textSal\_KeyPress**(object** sender**,** KeyPressEventArgs e**)**

**{**

**if** **(!(**Char**.**IsDigit**(**e**.**KeyChar**)))**

**{**

**if** **(**e**.**KeyChar **!=** **(**char**)**Keys**.**Back**)**

**{**

e**.**Handled **=** **true;**

**}**

**}**

**}**

Обработка события добавления компании. Обновляется таблица компаний и создаётся новый экземпляр класса Company.

**private** void button1\_Click**(object** sender**,** EventArgs e**)**

**{**

comp**.**addComp**(**textOrg**.**Text**,**10**);**

dataGridView1**.**Rows**.**Add**(**textOrg**.**Text**);**

textOrg**.**Clear**();**

**}**

Кнопка изменения данных компании. Читаются данные с TextBox. Обновляется таблица с компаниями и изменятся данные в соответствующем экземпляре.

**private** void button8\_Click**(object** sender**,** EventArgs e**)**

**{**

string text **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

int index **=** dataGridView1**.**CurrentRow**.**Index**;**

comp**.**FindComp**(**text**).**NameCompany **=** textOrg**.**Text**;**

dataGridView1**.**Rows**[**index**].**Cells**[**0**].**Value **=** textOrg**.**Text**;**

**}**

Обработка изменения данных вакансий. Находиться советующая вакансия и изменяется значение зарплаты и вакансии.

**private** void button7\_Click**(object** sender**,** EventArgs e**)**

**{**

**if** **(**comp**.**Count **!=** 0**)**

**{**

string text **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

int index **=** dataGridView1**.**CurrentRow**.**Index**;**

StructList vspom **=** comp**.**PHead**.**First**;**

**for(**int i **=** 0**;** i**<**comp**.**Count**;** i**++)**

**{**

**if** **(**vspom**.**Name**.**NameCompany**.**Equals**(**text**))**

**{**

comp**.**delComp**(**text**);**

dataGridView1**.**Rows**.**RemoveAt**(**index**);**

**break;**

**}**

**else** vspom **=** vspom**.**Next**;**

**}**

**}**

**else** **{** MessageBox**.**Show**(**"Нечего удалять."**);** **}**

**}**

Отработка события добавления вакансии у определённой организации выбранной в первой таблице.

**private** void button2\_Click**(object** sender**,** EventArgs e**)**

**{**

string info **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

comp**.**FindComp**(**info**).**AddVacantion**(**textWork**.**Text**,** Convert**.**ToInt32**(**textSal**.**Text**));**

dataGridView2**.**Rows**.**Add**(**textWork**.**Text**,**textSal**.**Text**);**

textSal**.**Clear**();**textWork**.**Clear**();**

**}**

Появление диалогового окна с выводом всей информации о компаниях и вакансиях.

**private** void infoToolStripMenuItem\_Click**(object** sender**,** EventArgs e**)**

**{**

MessageBox**.**Show**(**comp**.**getQueue**());**

**}**

**private** void button3\_Click**(object** sender**,** EventArgs e**)**

**{**

string text **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

int index **=** dataGridView2**.**CurrentRow**.**Index**;**

**if** **(**textSal**.**Text **!=** "" **&&** textWork**.**Text **!=** ""**)**

**{**

comp**.**FindComp**(**text**).**FindVacantion**(**dataGridView2**.**CurrentCell**.**Value**.**ToString**()).**Work **=** textWork**.**Text**;**

comp**.**FindComp**(**text**).**FindVacantion**(**textWork**.**Text**).**Salary **=** Convert**.**ToInt32**(**textSal**.**Text**);**

dataGridView2**.**Rows**[**index**].**Cells**[**0**].**Value **=** textWork**.**Text**;**

dataGridView2**.**Rows**[**index**].**Cells**[**1**].**Value **=** textSal**.**Text**;**

**}**

**else** **{** MessageBox**.**Show**(**"Введите оба параметра для изменения"**);** **}**

**}**

**private** void button4\_Click**(object** sender**,** EventArgs e**)**

**{**

string text **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

comp**.**FindComp**(**text**).**DeleteVacantion**();**

dataGridView2**.**Rows**.**RemoveAt**(**0**);**

**}**

**private** void fileToolStripMenuItem\_Click**(object** sender**,** EventArgs e**)**

**{**

**}**

Метод выгрузки данных в файл формата txt. Для дальнейшего чтения в начале файла пишем "Файл", чтобы можно было идентифицировать файл программы от других txt.

**private** void saveToolStripMenuItem\_Click**(object** sender**,** EventArgs e**)**

**{**

string path**;**

string fileText **=** "Файл\n"**;**

//Создание экземпляра диалогового окна (Для того чтобы не занимать поток) + фильтр на создание текстовых файлов

**using** **(**SaveFileDialog saveFile **=** **new** SaveFileDialog**()** **{** Filter **=** "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*" **})**

**{**

**if** **(**saveFile**.**ShowDialog**()** **==** DialogResult**.**OK**)**

**{**

path **=** saveFile**.**FileName**;**

fileText **+=** comp**.**getQueueName**()** **+** "\n"**;**

fileText **+=** comp**.**getQueueAll**();**

//Та же работа с экземплярами файла и записи файла

**using** **(**FileStream file **=** **new** FileStream**(**path**,** FileMode**.**OpenOrCreate**))**

**using** **(**StreamWriter stream **=** **new** StreamWriter**(**file**))**

**{**

stream**.**Write**(**fileText**);**

stream**.**Close**();**

file**.**Close**();**

**}**

**}**

**}**

**}**

Метод загрузки из файла. Открывает диалоговое окно для отображения txt файлов. Если файл удалось открыть происходит проверка на корректные данные, т.е. первая сточка в файле должна быть "Файл"после этого происходит дальнейшее чтение файла, заполнение структуры и таблицы данными из файла.

**private** void uploadToolStripMenuItem\_Click**(object** sender**,** EventArgs e**)**

**{**

string path **=** ""**,** doIt **=** ""**;**

int Count**,** LCount**,** gName**;**

**using** **(**OpenFileDialog openFile **=** **new** OpenFileDialog**()** **{** Filter **=** "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*" **})**

**{**

**if** **(**openFile**.**ShowDialog**()** **==** DialogResult**.**OK**)**

**{**

path **=** openFile**.**FileName**;**

**}**

**}**

**using** **(**StreamReader stream3 **=** **new** StreamReader**(**path**))**

**{**

doIt **=** stream3**.**ReadLine**();**

//Проверка на пригодность файла для работы с программой

**if** **(**doIt **==** "Файл"**)**

**{**

comp**.**delAllComp**();**

dataGridView1**.**Rows**.**Clear**();**

dataGridView2**.**Rows**.**Clear**();**

Count **=** int**.**Parse**(**stream3**.**ReadLine**());**

**for** **(**int i **=** 1**;** i **<=** Count**;)**

**{**

string name **=** stream3**.**ReadLine**();**

comp**.**addComp**(**name**,**10**);**

dataGridView1**.**Rows**.**Add**();**

dataGridView1**.**Rows**[**i **-** 1**].**Cells**[**0**].**Value **=** name**;**

**if** **(**i **!=** Count**)**

i**++;**

**else**

**break;**

**}**

stream3**.**ReadLine**();** int hall **=** 0**;**

**while** **(**stream3**.**Peek**()** **!=** **-**1**)**

**{**

LCount **=** int**.**Parse**(**stream3**.**ReadLine**());**

**if** **(**LCount **!=** 0**)**

**for** **(**int i **=** 1**;** i **<=** LCount**;)**

**{**

string inf **=** stream3**.**ReadLine**();**

gName **=** int**.**Parse**(**stream3**.**ReadLine**());**

string name **=** Convert**.**ToString**(**dataGridView1**.**Rows**[**hall**].**Cells**[**0**].**Value**);**

comp**.**FindComp**(**name**).**AddVacantion**(**inf**,** gName**);**

**if** **(**i **!=** LCount**)**

i**++;**

**else**

**break;**

**}**

hall**++;**

**}**

**}**

**else** **{** MessageBox**.**Show**(**"Данный файл не подходит для работы с программой"**);** **}**

stream3**.**Close**();**

**}**

**}**

**}**

**}**

1. Описание структуры проекта в соответствии с использованным инструментом разработки

Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментов. Данные продукты позволяют разрабатывать как консольные приложения, так и игры и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, UWP а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, .NET Core, .NET, MAUI, Xbox, Windows Phone .NET Compact Framework и Silverlight. После покупки компании Xamarin корпорацией Microsoft появилась возможность разработки IOS и Android программ.

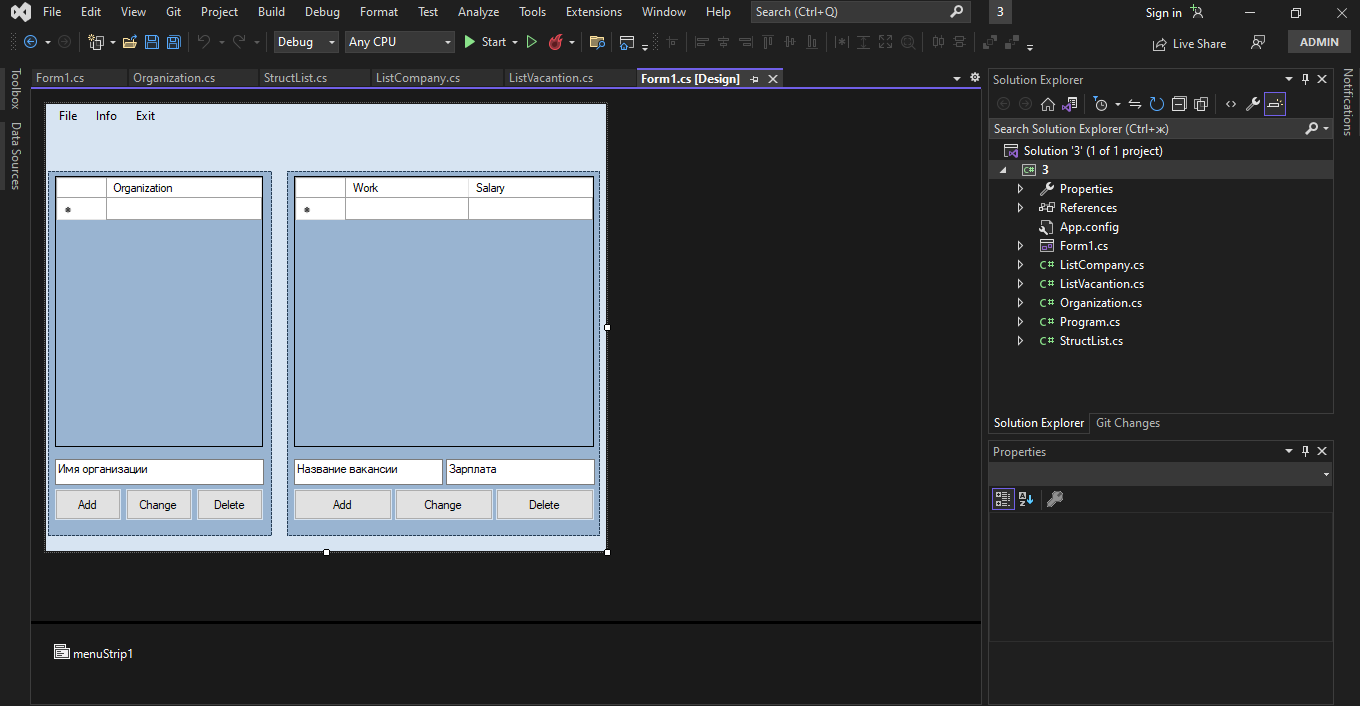


Рис. 9 Visual studio

В главной папке проекта находятся разработанные классы \*.cs , так же в App.config находятся настройки для среды разработки Visual Studio находятся настройки конфигурации для запуска приложения. Файл \*.resx содержит в себе дизайн оконного приложения, составленный на языке разметки XML.

В папке Propertis находятся все подключенные в разработанных классах стандартные библиотеки для установления зависимостей в проекте.

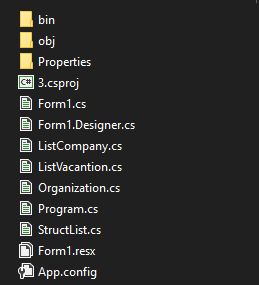


Рис. 10 Содержимое папки проекта.

Папка obj содержит в себе данные последнего Debug запуска проекта с соответствующими файлами конфигурации и логами.

В парке bin находиться главный исполняемый файл программы с последней сборки проекта, а также папка Relies с исполняемым файлом готовым к установке программы со всеми зависимостями на любой Windows операционной системе.

# **Список литературы**

1. Алгоритмы и структуры данных: учебное пособие / О.Б. Фофанов; Томский политехнический университет. – Томск: Изд-во Томского политех нического университета, 2019.
2. Васильев А. Н. Java. Объектно-ориентированное программирование. Учебное пособие. Стандарт третьего поколения. – "Издательский дом Питер 2021.
3. Кувшинов Д. Р., Осипов С. И. Основы программирования: язык C++: учебное пособие. – 2021.
4. Ляпин А. А., Быкова Ю. А. РАЗРАБОТКА БАЛЛИСТИЧЕСКОЙ ФУНКЦИИ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C/C++ //ПОЛИ- ХОТОМИЧЕСКИЕ КЛАССИФИКАЦИИ СПЕЦИАЛЬНЫХ ТЕРМИ- НОВ В НАУЧНОЙ И УЧЕБНОЙ ЛИТЕРАТУРЕ. – 2021. – С. 308-311.
5. Объектно-ориентированное программирование. Учебно-методическое пособие/ Козин.А.Н., Якунина Е.А. – Казань: УВО «Университет управления «ТИСБИ», 2020.
6. Структуры и алгоритмы обработки данных. Часть 2. Усложнённые структуры данных. Теория и методика обучения. Учебно-методическое пособие / А.Н. Козин, Л.Б. Таренко. - Казань: УВО "Университет управления "ТИСБИ 2018
7. Павловская Т. А., Щупак Ю. А. C/C++. Структурное и объектно- ориентированное программирование. Практикум. – "Издательский дом Питер 2021.
8. Структуры и алгоритмы обработки данных. Часть 3. Поиск и сортировка данных. Теория и методика обучения. Учебно-методическое пособие/ А.Н. Козин, Л.Б. Таренко. - Казань: УВО "Университет управления "ТИСБИ 2020
9. Puuronen M. Implementing Horizontal Layout for the Qt Design Studio’s Component Library. – 2021.
10. Sherriff N. Learn Qt 5: Build modern, responsive cross-platform desktop applications with Qt, C++, and QML. – Packt Publishing Ltd, 2018.

Листинг программы

// Form1

**using** System**;**

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

**using** System**.**ComponentModel**;**

**using** System**.**Data**;**

**using** System**.**Drawing**;**

**using** System**.**IO**;**

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

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

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

**using** System**.**Windows**.**Forms**;**

**namespace** \_3

**{**

**public** **partial** class Form1 **:** Form

**{**

Organization comp **=** **new** Organization**(**"HART"**);**

**public** Form1**()**

**{**

InitializeComponent**();**

**}**

**private** void dataGridView1\_CellContentClick**(object** sender**,** DataGridViewCellEventArgs e**)**

**{**

dataGridView2**.**Rows**.**Clear**();**

string text **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

ListCompany vspom **=** comp**.**FindComp**(**text**);**

**if** **(**vspom **!=** **null)**

**{**

ListVacantion**[]** copy1 **=** vspom**.**QUeue**;**

int chet **=** comp**.**FindComp**(**text**).**Count**;**

**for** **(**int i **=** 0**;** i **<** chet**;** i**++)**

**{**

dataGridView2**.**Rows**.**Add**(**copy1**[**i**].**Work**,** copy1**[**i**].**Salary**);**

**}**

**}**

**}**

**private** void textOrg\_Click**(object** sender**,** EventArgs e**)**

**{**

textOrg**.**Clear**();**

**}**

**private** void textWork\_Click**(object** sender**,** EventArgs e**)**

**{**

textWork**.**Clear**();**

**}**

**private** void textSal\_Click**(object** sender**,** EventArgs e**)**

**{**

textSal**.**Clear**();**

**}**

**private** void exitToolStripMenuItem\_Click**(object** sender**,** EventArgs e**)**

**{**

Close**();**

**}**

**private** void textSal\_KeyPress**(object** sender**,** KeyPressEventArgs e**)**

**{**

**if** **(!(**Char**.**IsDigit**(**e**.**KeyChar**)))**

**{**

**if** **(**e**.**KeyChar **!=** **(**char**)**Keys**.**Back**)**

**{**

e**.**Handled **=** **true;**

**}**

**}**

**}**

**private** void button1\_Click**(object** sender**,** EventArgs e**)**

**{**

comp**.**addComp**(**textOrg**.**Text**,**10**);**

dataGridView1**.**Rows**.**Add**(**textOrg**.**Text**);**

textOrg**.**Clear**();**

**}**

**private** void button8\_Click**(object** sender**,** EventArgs e**)**

**{**

string text **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

int index **=** dataGridView1**.**CurrentRow**.**Index**;**

comp**.**FindComp**(**text**).**NameCompany **=** textOrg**.**Text**;**

dataGridView1**.**Rows**[**index**].**Cells**[**0**].**Value **=** textOrg**.**Text**;**

**}**

**private** void button7\_Click**(object** sender**,** EventArgs e**)**

**{**

**if** **(**comp**.**Count **!=** 0**)**

**{**

string text **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

int index **=** dataGridView1**.**CurrentRow**.**Index**;**

StructList vspom **=** comp**.**PHead**.**First**;**

**for(**int i **=** 0**;** i**<**comp**.**Count**;** i**++)**

**{**

**if** **(**vspom**.**Name**.**NameCompany**.**Equals**(**text**))**

**{**

comp**.**delComp**(**text**);**

dataGridView1**.**Rows**.**RemoveAt**(**index**);**

**break;**

**}**

**else** vspom **=** vspom**.**Next**;**

**}**

**}**

**else** **{** MessageBox**.**Show**(**"Нечего удалять."**);** **}**

**}**

**private** void button2\_Click**(object** sender**,** EventArgs e**)**

**{**

string info **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

comp**.**FindComp**(**info**).**AddVacantion**(**textWork**.**Text**,** Convert**.**ToInt32**(**textSal**.**Text**));**

dataGridView2**.**Rows**.**Add**(**textWork**.**Text**,**textSal**.**Text**);**

textSal**.**Clear**();**textWork**.**Clear**();**

**}**

**private** void infoToolStripMenuItem\_Click**(object** sender**,** EventArgs e**)**

**{**

MessageBox**.**Show**(**comp**.**getQueue**());**

**}**

**private** void button3\_Click**(object** sender**,** EventArgs e**)**

**{**

string text **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

int index **=** dataGridView2**.**CurrentRow**.**Index**;**

**if** **(**textSal**.**Text **!=** "" **&&** textWork**.**Text **!=** ""**)**

**{**

comp**.**FindComp**(**text**).**FindVacantion**(**dataGridView2**.**CurrentCell**.**Value**.**ToString**()).**Work **=** textWork**.**Text**;**

comp**.**FindComp**(**text**).**FindVacantion**(**textWork**.**Text**).**Salary **=** Convert**.**ToInt32**(**textSal**.**Text**);**

dataGridView2**.**Rows**[**index**].**Cells**[**0**].**Value **=** textWork**.**Text**;**

dataGridView2**.**Rows**[**index**].**Cells**[**1**].**Value **=** textSal**.**Text**;**

**}**

**else** **{** MessageBox**.**Show**(**"Введите оба параметра для изменения"**);** **}**

**}**

**private** void button4\_Click**(object** sender**,** EventArgs e**)**

**{**

string text **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

comp**.**FindComp**(**text**).**DeleteVacantion**();**

dataGridView2**.**Rows**.**RemoveAt**(**0**);**

**}**

**private** void fileToolStripMenuItem\_Click**(object** sender**,** EventArgs e**)**

**{**

**}**

**private** void saveToolStripMenuItem\_Click**(object** sender**,** EventArgs e**)**

**{**

string path**;**

string fileText **=** "Файл\n"**;**

//Создание экземпляра диалогового окна (Для того чтобы не занимать поток) + фильтр на создание текстовых файлов

**using** **(**SaveFileDialog saveFile **=** **new** SaveFileDialog**()** **{** Filter **=** "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*" **})**

**{**

**if** **(**saveFile**.**ShowDialog**()** **==** DialogResult**.**OK**)**

**{**

path **=** saveFile**.**FileName**;**

fileText **+=** comp**.**getQueueName**()** **+** "\n"**;**

fileText **+=** comp**.**getQueueAll**();**

//Та же работа с экземплярами файла и записи файла

**using** **(**FileStream file **=** **new** FileStream**(**path**,** FileMode**.**OpenOrCreate**))**

**using** **(**StreamWriter stream **=** **new** StreamWriter**(**file**))**

**{**

stream**.**Write**(**fileText**);**

stream**.**Close**();**

file**.**Close**();**

**}**

**}**

**}**

**}**

**private** void uploadToolStripMenuItem\_Click**(object** sender**,** EventArgs e**)**

**{**

string path **=** ""**,** doIt **=** ""**;**

int Count**,** LCount**,** gName**;**

**using** **(**OpenFileDialog openFile **=** **new** OpenFileDialog**()** **{** Filter **=** "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*" **})**

**{**

**if** **(**openFile**.**ShowDialog**()** **==** DialogResult**.**OK**)**

**{**

path **=** openFile**.**FileName**;**

**}**

**}**

**using** **(**StreamReader stream3 **=** **new** StreamReader**(**path**))**

**{**

doIt **=** stream3**.**ReadLine**();**

//Проверка на пригодность файла для работы с программой

**if** **(**doIt **==** "Файл"**)**

**{**

comp**.**delAllComp**();**

dataGridView1**.**Rows**.**Clear**();**

dataGridView2**.**Rows**.**Clear**();**

Count **=** int**.**Parse**(**stream3**.**ReadLine**());**

**for** **(**int i **=** 1**;** i **<=** Count**;)**

**{**

string name **=** stream3**.**ReadLine**();**

comp**.**addComp**(**name**,**10**);**

dataGridView1**.**Rows**.**Add**();**

dataGridView1**.**Rows**[**i **-** 1**].**Cells**[**0**].**Value **=** name**;**

**if** **(**i **!=** Count**)**

i**++;**

**else**

**break;**

**}**

stream3**.**ReadLine**();** int hall **=** 0**;**

**while** **(**stream3**.**Peek**()** **!=** **-**1**)**

**{**

LCount **=** int**.**Parse**(**stream3**.**ReadLine**());**

**if** **(**LCount **!=** 0**)**

**for** **(**int i **=** 1**;** i **<=** LCount**;)**

**{**

string inf **=** stream3**.**ReadLine**();**

gName **=** int**.**Parse**(**stream3**.**ReadLine**());**

string name **=** Convert**.**ToString**(**dataGridView1**.**Rows**[**hall**].**Cells**[**0**].**Value**);**

comp**.**FindComp**(**name**).**AddVacantion**(**inf**,** gName**);**

**if** **(**i **!=** LCount**)**

i**++;**

**else**

**break;**

**}**

hall**++;**

**}**

**}**

**else** **{** MessageBox**.**Show**(**"Данный файл не подходит для работы с программой"**);** **}**

stream3**.**Close**();**

**}**

**}**

**}**

**}**

// ListCompany

**using** System**;**

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

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

**namespace** \_3

**{**

class ListCompany

**{**

**private** string nameCompany**;**

**private** ListVacantion**[]** Queue**;**

**private** int size**;**

**private** int first**;**

**private** int last**;**

**private** int count**;**

**public** ListCompany**(**string \_nameCompany**,** int \_arrSize**)**

**{**

**this.**size **=** \_arrSize**;**

**this.**nameCompany **=** \_nameCompany**;**

Queue **=** **new** ListVacantion**[**size**];**

first **=** 0**;**

last **=** 0**;**

count **=** 0**;**

**}**

**public** int Count **{** **get** **{** **return** count**;** **}** **}**

**public** ListVacantion**[]** QUeue **{** **get** **{** **return** Queue**;** **}** **}**

**public** string NameCompany**{** **get** **{** **return** nameCompany**;** **}** **set** **{** **this.**nameCompany **=** **value;** **}** **}**

**public** void AddVacantion**(**string \_work**,** int \_salary**)**

**{**

ListVacantion newList **=** **new** ListVacantion**(**\_work**,** \_salary**);**

**if** **(**count **>=** **(**0.75 **\*** size**))** **{** recizeArray**();** **}**

Queue**[**last**]** **=** newList**;**

last**++;**

count**++;**

**}**

**public** ListVacantion FindVacantion**(**string \_work**)**

**{**

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

**{**

**if** **(**Queue**[**i**]** **!=** **null)**

**{**

**if** **(**Queue**[**i**].**Work**.**Equals**(**\_work**))**

**{**

**return** Queue**[**i**];**

**}**

**}**

**}**

**return** **null;**

**}**

**public** void DeleteVacantion**()**

**{**

**if** **(**count **!=** 0**)**

**{**

Queue**[**first**]** **=** **null;**

count**--;**

**for** **(**int i **=** **(**first **+** 1**);** i **<** last**;** i**++)**

**{**

Queue**[**i **-** 1**]** **=** Queue**[**i**];**

**}**

last**--;**

Queue**[**last**]** **=** **null;**

**if** **(**count **>** 3 **&&** count **<=** **(**0.25 **\*** size**))** **{** recizeArray**();** **}**

**}**

**}**

**public** int SumVacantion**()**

**{**

**return** count**;**

**}**

**public** string chekFor**()**

**{**

string date **=** Convert**.**ToString**(**count**);**

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

**{**

date **+=** "\n" **+** Queue**[**i**].**Work**;**

date **+=** "\n" **+** Queue**[**i**].**Salary**;**

**}**

**return** date**;**

**}**

**public** string chek**()**

**{**

string date **=** "\n\n##### Organization " **+** nameCompany **+** "\n"**;**

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

**{**

date **+=** "\n\nvacantion: " **+** Queue**[**i**].**Work**;**

date **+=** "\nSalary: " **+** Queue**[**i**].**Salary**;**

**}**

date **+=**"\n\ncount vocation - " **+** SumVacantion**();**

date **+=**"\n#####################################"**;**

**return** date**;**

**}**

**public** void recizeArray**()**

**{**

**if** **(**count **>=** **(**0.75 **\*** size**))**

**{**

ListVacantion**[]** pTemp **=** **new** ListVacantion**[**size **\*** 2**];**

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

**{**

pTemp**[**i**]** **=** Queue**[**i**];**

**}**

size **\*=** 2**;**

Queue **=** **new** ListVacantion**[**size**];**

**for** **(**int i **=** 0**;** i **<** size**;** i**++)**

**{**

Queue**[**i**]** **=** pTemp**[**i**];**

**}**

**}**

**else** **if** **(**count **<=** **(**0.25 **\*** size**))**

**{**

ListVacantion**[]** pTemp **=** **new** ListVacantion**[**size **/** 2**];**

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

**{**

pTemp**[**i**]** **=** Queue**[**i**];**

**}**

size **/=** 2**;**

Queue **=** **new** ListVacantion**[**size**];**

**for** **(**int i **=** 0**;** i **<** size**;** i**++)**

**{**

Queue**[**i**]** **=** pTemp**[**i**];**

**}**

**}**

**}**

**}**

**}**

// StructList

**using** System**;**

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

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

**namespace** \_3

**{**

class StructList

**{**

**public** ListCompany name**;** //данные о организации

**private** StructList next**;** //ссылка на потомка

**private** StructList last**;**

**private** StructList first**;**

**public** StructList**(**ListCompany \_date**)**

**{**

name **=** \_date**;**

next **=** **null;**

first **=** **null;**

last **=** **null;**

**}**

**public** ListCompany Name **{** **get** **{** **return** name**;** **}** **}**

**public** StructList Next

**{**

**get** **{** **return** next**;** **}**

**set** **{** **this.**next **=** **value;** **}**

**}**

**public** StructList Last

**{**

**get** **{** **return** last**;** **}**

**set** **{** **this.**last **=** **value;** **}**

**}**

**public** StructList First

**{**

**get** **{** **return** first**;** **}**

**set** **{** **this.**first **=** **value;** **}**

**}**

**}**

**}**

// Organization

**using** System**;**

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

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

**namespace** \_3

**{**

class Organization

**{**

**private** string nameOrganization**;**//название организации

**private** StructList pHead**;**

**private** int count**;**

**public** Organization**(**string \_nameOrganization**)**

**{**

**this.**nameOrganization **=** \_nameOrganization**;**

pHead **=** **new** StructList**(null);**

pHead**.**First **=** **null;**

pHead**.**Last **=** **null;**

**}**

**public** string NameOrganization **{** **get;** **}**

**public** int Count **{** **get** **{** **return** count**;** **}** **}**

**public** StructList PHead **{** **get** **{** **return** pHead**;** **}** **}**

**public** void addComp**(**string \_nameCompany**,** int \_arrSize**)**

**{**

ListCompany newComp **=** **new** ListCompany**(**\_nameCompany**,** \_arrSize**);**

StructList vspom **=** **new** StructList**(**newComp**);**

count**++;**

**if** **(**pHead**.**First **==** **null** **&&** pHead**.**Last **==** **null)**

**{**

pHead**.**First **=** vspom**;**

pHead**.**Last **=** vspom**;**

pHead**.**First**.**Next **=** pHead**.**Last**;**

pHead**.**Last**.**Next **=** pHead**.**First**;**

**}**

**else**

**{**

//pHead.Last.Next = vspom;

pHead**.**Last **=** pHead**.**Last**.**Next **=** vspom**;**

pHead**.**Last**.**Next **=** pHead**.**First**;**

**}**

**}**

**public** ListCompany FindComp**(**string \_nameCompany**)**

**{**

StructList search **=** pHead**.**First**;**

**while** **(**search **!=** **null)**

**{**

**if** **(**search**.**Name **!=** **null)**

**{**

**if** **(**search**.**Name**.**NameCompany**.**Equals**(**\_nameCompany**))**

**{** **return** search**.**Name**;** **}**

**else** search **=** search**.**Next**;**

**}**

**}**

**return** **null;**

**}**

**public** void delAllComp**()**

**{**

pHead**.**First **=** **null;**

pHead**.**Last**=** **null;**

**}**

**public** bool delComp**(**String \_nameComp**)**

**{**

**if** **(**pHead **!=** **null)**

**{**

StructList vspom **=** pHead**.**First**;**

//StructList pTemp = pHead.First;

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

**{**

**if** **(**vspom **!=** **null)**

**{**

**if** **(**vspom **==** pHead**.**First**)**

**{**

**if** **(**vspom**.**Name**.**NameCompany**.**Equals**(**\_nameComp**))**

**{**

count**--;**

pHead**.**First **=** pHead**.**First**.**Next**;**

**return** **true;**

**}**

**else** **{** vspom **=** vspom**.**Next**;** **}**

**}**

**else**

**{**

**if** **(**vspom**.**Next**.**Name**.**NameCompany**.**Equals**(**\_nameComp**))**

**{**

count**--;**

vspom**.**Next **=** vspom**.**Next**.**Next**;**

**return** **true;**

**}**

**else** **{** vspom **=** vspom**.**Next**;** **}**

**}**

**}**

**}**

**}**

**return** **false;**

**}**

**public** string getQueueName**()**

**{**

string info **=** Convert**.**ToString**(**Count**);**

StructList pTemp **=** pHead**.**First**;**

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

**{**

info **+=** "\n" **+** pTemp**.**Name**.**NameCompany**;**

pTemp **=** pTemp**.**Next**;**

**}**

**return** info**;**

**}**

**public** string getQueueAll**()**

**{**

string info **=** ""**;**

StructList pTemp **=** pHead**.**First**;**

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

**{**

info **+=** "\n" **+** pTemp**.**Name**.**chekFor**();**

pTemp **=** pTemp**.**Next**;**

**}**

**return** info**;**

**}**

**public** string getQueue**()**

**{**

string date **=** "-=-=-=-=-=-Employment service-=-=-=-=-=-"**;**

StructList pTemp **=** pHead**.**First**;**

**do**

**{**

date **+=** pTemp**.**Name**.**chek**()** **+** "\n"**;**

pTemp **=** pTemp**.**Next**;**

**}** **while** **(**pTemp **!=** pHead**.**First**);**

date **+=** "\nTotal number of vacancies: " **+** getSumQueue**();**

**return** date**;**

**}**

**public** int getSumQueue**()**

**{**

StructList pTemp **=** pHead**.**First**;**

int summa **=** 0**;**

**do**

**{**

summa **+=** pTemp**.**Name**.**SumVacantion**();**

pTemp **=** pTemp**.**Next**;**

**}** **while** **(**pTemp **!=** pHead**.**First**);**

**return** summa**;**

**}**

**}**

**}**

// ListVacantion

**using** System**;**

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

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

**namespace** \_3

**{**

class ListVacantion

**{**

**private** string work**;**

**private** int salary**;**

**public** ListVacantion**(**string \_work**,** int \_salary**)**

**{**

**this.**work **=** \_work**;**

**this.**salary **=** \_salary**;**

**}**

**public** string Work **{** **get** **{** **return** work**;** **}** **set** **{** **this.**work **=** **value;** **}** **}**

**public** int Salary **{** **get** **{** **return** salary**;** **}** **set** **{** **this.**salary **=** **value;** **}** **}**

**}**

**}**