УВО «Университет Управления «ТИСБИ»

Факультет информационных технологий

Кафедра информационных технологий

**Курсовая работа**

по дисциплине «Объектно-ориентированное программирование»

на тему: «Разработка объектной программы для обработки данных о пациентах платной медицинской клиники»

Выполнил: студент гр. П-011 Сучёв Н.Е.

Проверил: ст. преп. Якунина Е.А.

Казань 2022

Оглавление

[Постановка задачи 3](#_Toc105286639)

[1. Описание используемых структур данных с алгоритмами выполнения основных операций 5](#_Toc105286640)

[1.1 Адресный однонаправленный разомкнутый список. 5](#_Toc105286641)

[1.2 Стек на основе массива 6](#_Toc105286642)

[2. Краткие сведения об объектном подходе 8](#_Toc105286643)

[3. Формализованное описание разработанных классов 14](#_Toc105286644)

[4. Описание демонстрационного модуля с характеристикой использованных стандартных компонентов и списком реализованных обработчиков 20](#_Toc105286645)

[5. Описание структуры проекта в соответствии с использованным инструментом разработки 25](#_Toc105286646)

[Список литературы 27](#_Toc105286647)

[Листинг программы 29](#_Toc105286648)

Постановка задачи

**Цели работы:** отработка навыков курсов «объектно-ориентированное программирование» и «структуры и алгоритмы обработки данных».

**Постановка задачи:** Медицинский центр ведет учет оказываемых услуг с помощью списка своих пациентов с указанием уникальной фамилии пациента, числа посещений и суммарной стоимости оказанных услуг. Для каждого пациента создается список его посещений с указанием уникальной даты посещения, вида услуги и ее стоимости.

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 пациентов используется структура данных в виде стека на основе обычного массива. Для объединения посещений каждого пациента используется структура данных в виде адресного разомкнутого упорядоченного однонаправленного списка с заголовком.

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 однонаправленный разомкнутый список.

Реальное многообразие структур данных базируется всего на двух основных способах получения адреса хранимого элемента: вычисление (массив) и хранение (указатель). До сих пор основной компонентой структуры данных являлся массив (обычный массив, массив указателей). Если же попытаться построить структуру данных, исходя только из указателей, то получается цепочка (последовательность) элементов, содержащих указатели друг на друга. В простейшем случае она может быть линейной (список), в более сложных случаях – ветвящейся (деревья, графы). Итак, список – линейная последовательность элементов, каждый из которых содержит указатели (ссылается) на своих соседей.

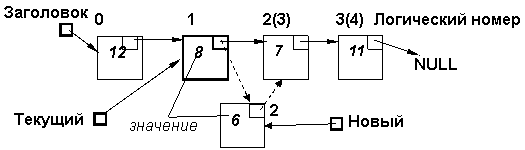


рис. 1. Списковая структура

Сразу же отметим основную особенность: физическое размещение в памяти элементов списка не имеет никакого значения, все определяется наличием ссылок на него в других элементах и извне. У массива всегда есть «начало». У списка по определению отсутствует фиксированная привязка к памяти. Перечислим основные релятивистские свойства списка:

элемент списка доступен в программе через указатель. «Смысл» этого указателя отражает функциональное назначение элемента списка в программе: первый, последний, текущий, предыдущий, новый и т.п.. Между указателем и элементом списка имеется такая же взаимосвязь, как между индексом в массиве и элементом массива;

в программе список задается посредством заголовка – указателя на первый элемент списка;

порядок следования элементов определяется последовательностью связей между элементами. Изменение порядка следования элементов (вставка, удаление) осуществляются изменением переустановкой указателей на соседние элементы.

логический (порядковый) номер элемента списка также задается его естественной нумерацией в цепочке элементов;

список является структурой данных с последовательным доступом. Для получения n-го по счету элемента необходимо последовательно пройти по цепочке от элемента, на который имеется указатель (например, от заголовка);

список удобен для использования именно как динамическая структура данных: элементы списка обычно создаются как динамические переменные, а связи между ними устанавливаются программно (динамически);

список обладает свойством локальности изменений: при вставке/удалении элемента изменения касаются только текущего и его соседей. Вспомним массив: при вставке/удалении его элементов происходит физическое перемещение (сдвиг) всех элементов от текущего до конца.

Отсюда следует, что преимущества списков проявляются в таких структурах данных, где операции изменения порядка превалируют над операциями доступа и поиска.

**1.2 Стек на основе массива**

Стек представляет собой структуру данных, которая работает по принципу LIFO (Last In First Out - "последний пришел - первый вышел"). Графически стек можно представить в виде столбика или стопки объектов:

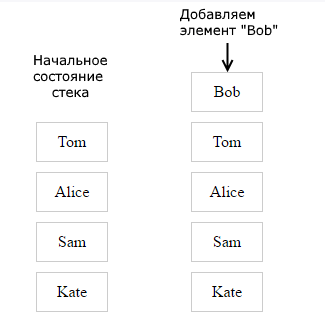


Рис. 2 Стек

Стек имеет вершину, который образует последний добавленный элемент. При добавлении новый элемент помещается поверх вершины стека и образует новую вершину. При удалении удаляется элемент из вершины стека, а предыдущий элемент образует новую вершину.

Так, на приведенном рисунке вначале вершиной стека является "Tom". После добавления нового элемента "Bob" этот элемент располагается поверх элемента "Tom" и становится новой вершиной.

В библиотеке классов .NET в принципе уже есть свой класс, который выполняет роль стека. Это класс - System.Collections.Generic.Stack. Но рассмотрим, как мы сами можем реализовать структуру в виде стека.

Структура стек вне зависимости от языка программирования обладает неким общим функционалом, который составляют метод добавления элемента (как правило, называется push()) и метод извлечения элемента из вершины стека (обычно называется pop()). Кроме того, нередко реализации стеков содержат метод получения элемента из вершины без его извлечения, метод определения размера стека и ряд других.

1. Краткие сведения об объектном подходе

Объе́ктно-ориенти́рованное программи́рование (сокр. ООП) — методология программирования, основанная на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

Идеологически ООП — подход к программированию как к моделированию информационных объектов, решающий на новом уровне основную задачу структурного программирования: структурирование информации с точки зрения управляемости, что существенно улучшает управляемость самим процессом моделирования, что, в свою очередь, особенно важно при реализации крупных проектов.

Управляемость для иерархических систем предполагает минимизацию избыточности данных (аналогичную нормализации) и их целостность, поэтому созданное удобно управляемым — будет и удобно пониматься. Таким образом, через тактическую задачу управляемости решается стратегическая задача — транслировать понимание задачи программистом в наиболее удобную для дальнейшего использования форму.

Основные принципы структурирования в случае ООП связаны с различными аспектами базового понимания предметной задачи, которое требуется для оптимального управления соответствующей моделью:

абстракция для выделения в моделируемом предмете важного для решения конкретной задачи по предмету, в конечном счёте — контекстное понимание предмета, формализуемое в виде класса;

инкапсуляция для быстрой и безопасной организации собственно иерархической управляемости: чтобы было достаточно простой команды «что делать», без одновременного уточнения как именно делать, так как это уже другой уровень управления;

наследование для быстрой и безопасной организации родственных понятий: чтобы было достаточно на каждом иерархическом шаге учитывать только изменения, не дублируя всё остальное, учтённое на предыдущих шагах;

полиморфизм для определения точки, в которой единое управление лучше распараллелить или наоборот — собрать воедино.

То есть фактически речь идёт о прогрессирующей организации информации согласно первичным семантическим критериям: «важное/неважное», «ключевое/подробности», «родительское/дочернее», «единое/множественное». Прогрессирование, в частности, на последнем этапе даёт возможность перехода на следующий уровень детализации, что замыкает общий процесс.

Обычный человеческий язык в целом отражает идеологию ООП, начиная с инкапсуляции представления о предмете в виде его имени и заканчивая полиморфизмом использования слова в переносном смысле, что в итоге развивает выражение представления через имя предмета до полноценного понятия-класса.

Абстракция данных

Абстрагирование означает выделение значимой информации и исключение из рассмотрения незначимой. В ООП рассматривают лишь абстракцию данных (нередко называя её просто «абстракцией»), подразумевая набор наиболее значимых характеристик объекта, доступных остальной программе.

Инкапсуляция

Инкапсуляция — свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе. Одни языки (например, C++, Java или Ruby) отождествляют инкапсуляцию с сокрытием, но другие (Smalltalk, Eiffel, OCaml) различают эти понятия.

Наследование

Наследование — свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствованной функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником, дочерним или производным классом.

Полиморфизм подтипов

Полиморфизм подтипов (в ООП называемый просто «полиморфизмом») — свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. Другой вид полиморфизма — параметрический — в ООП называют обобщённым программированием.

Класс

Класс — универсальный, комплексный тип данных, состоящий из тематически единого набора «полей» (переменных более элементарных типов) и «методов» (функций для работы с этими полями), то есть он является моделью информационной сущности с внутренним и внешним интерфейсами для оперирования своим содержимым (значениями полей). В частности, в классах широко используются специальные блоки из одного или чаще двух спаренных методов, отвечающих за элементарные операции с определённым полем (интерфейс присваивания и считывания значения, геттер-сеттер), которые имитируют непосредственный доступ к полю. Эти блоки называются «свойствами» и почти совпадают по конкретному имени со своим полем (например, имя поля может начинаться со строчной, а имя свойства — с заглавной буквы). Другим проявлением интерфейсной природы класса является то, что при копировании соответствующей переменной через присваивание копируется только интерфейс, но не сами данные, то есть класс — ссылочный тип данных. Переменная-объект, относящаяся к заданному классом типу, называется экземпляром этого класса. При этом в некоторых исполняющих системах класс также может представляться некоторым объектом при выполнении программы посредством динамической идентификации типа данных. Обычно классы разрабатывают таким образом, чтобы обеспечить отвечающие природе объекта и решаемой задаче целостность данных объекта, а также удобный и простой интерфейс. В свою очередь, целостность предметной области объектов и их интерфейсов, а также удобство их проектирования, обеспечивается наследованием.

Объект

Сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса (например, после запуска результатов компиляции и связывания исходного кода на выполнение).

Лука Карделли[en] и Мартин Абади построили теоретическое обоснование ООП и классификацию на основе этого обоснования. Они отмечают, что выделенные ими понятия и категории вместе встречаются далеко не во всех ОО-языках, большинство языков поддерживают лишь подмножества теории, а порой и своеобразные отклонения от неё.

Ключевые понятия:

квантификация переменных типа (универсальная, экзистенциальная, ограниченная);

подтипизация[en] (англ. subtyping — отношения «супертип-подтип»);

включение (англ. subsumption — см. принцип подстановки Барбары Лисков) — частный случай подтипизации — см. подтипизация на записях;

объект;

объектный тип (фактически, специальная форма «записи», в число полей которой входят функции первого класса);

класс.

Класс описывает абстрактное поведение. Объектные типы строятся на основе класса посредством добавления различных приватных полей и методов. Объект (то есть значение объектного типа, традиционно называемое «экземпляром класса») порождается конструктором на основе начальных параметров.

Традиционно перечисляемые основные принципыПерейти к разделу «#Принципы» ООП не апеллируют к данному теоретическому обоснованию, а являются устоявшимися в сообществе догмами (что приводит к сильному разбросу вариантов изложения в разных источниках). По большей части они относятся к языкам — потомкам Алгола и Симулы; в меньшей степени — к потомкам Smalltalk (в частности, часто упоминаемый принцип сокрытия в потомках Smalltalk семантически недоступен и идеологически считается несущественным). В значительно большей степени теоретически обоснованные концепции ООП поддерживают ОО-языки, развившиеся на поприще функционального программирования: OCaml, диалекты Haskell (O’Haskell, Mondrian), successor ML. Более того, основные идеи объектного моделирования в этом случае не требуют прямой поддержки со стороны языка, а могут быть сравнительно легко эмулированы.

Наиболее заметные отличия в проявлении показателей качества между языками разных видов:

В мейнстримных языках декларируемые принципы нацелены на повышение изначально низкого для императивного программирования коэффициента повторного использования кода. В полиморфно типизированных применение концепций ООП, напротив, означает очевидное его снижение из-за перехода от параметрического полиморфизма к ad-hoc-полиморфизму. В динамически типизированных языках (Smalltalk, Python, Ruby) эти принципы используются для логической организации программы, и их влияние на коэффициент повторного использования трудно спрогнозировать — он сильно зависит от дисциплины программиста. Например, в CLOS мультиметоды одновременно являются функциями первого класса, что позволяет рассматривать их одновременно и как связанно квантифицированные, и как обобщённые (истинно полиморфные).

Традиционные ОО-языки используют номинативную типизацию[en], то есть допустимость соиспользования объектов разных классов только при условии явного указания родственных отношений между классами. Для полиморфно типизированных языков характерна структурная типизация, то есть согласование классов между собой тем же механизмом, что и согласование числа 5 с типом int. Динамически типизированные языки также занимают здесь промежуточную позицию.

Обобщённое обоснование динамической диспетчеризации (включая множественную) в середине 1990-х годов построил Джузеппе Кастанья.

3. Формализованное описание разработанных классов

Основной класс в программе — это посещения. Ге храниться дата посещения стоимость, услуга предоставленнаая.

**using** System**;**

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

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

**namespace** \_3ЭтапКлиника

**{**

class Visiting

**{**

**private** int date**;**

**private** string service**;**//создание необходим переменных

**private** int cost**;**

**public** Visiting**(**int \_date**,** string \_service**,** int \_cost**)**//конструктор класса

**{**

**this.**date **=** \_date**;**

**this.**service **=** \_service**;**

**this.**cost **=** \_cost**;**

**}**

**public** int getDate**()**//получение даты

**{**

**return** date**;**

**}**

**public** string getService**()**//получение оказанной услуги

**{**

**return** service**;**

**}**

**public** int getCost**()**//получение стоимости услуги

**{**

**return** cost**;**

**}**

**public** void setDate**(**int \_date**)**//изменение даты

**{** **this.**date **=** \_date**;** **}**

**public** void setService**(**string \_service**)**//изменение оказанной услуги

**{** **this.**service **=** \_service**;** **}**

**public** void setCost**(**int \_cost**)**//изменение стоимости

**{** **this.**cost **=** \_cost**;** **}**

**}**

**}**

Красс пациент содержит в себе посещения пациента, а также его фамилию. Посещения храниться в структуре данных список.

**using** System**;**

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

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

**namespace** \_3ЭтапКлиника

**{**

class Patient

**{**

**private** string name**;**//фамилия пациента

**private** int visitCount**;**//количество посещений

**private** int sum**;**//сумма потраченных денег

**private** List pHead**,**pTemp**;**//ссылка на первый элемент

**public** Patient**(**string \_name**)**//создание конструктора

**{**

name **=** \_name**;**//присваиваем имя пациенту

pHead **=** **new** List**(null);**//создаем заголовок в списке с пустыми полезными данными

pHead**.**First **=** **null;**//ссылка на первый элемент равну нулл

pHead**.**Last **=** **null;**//ссылка на последний элемент равна нулл

pTemp **=** **null;**//вспомогательная переменная так-же равна нулл

visitCount **=** 0**;**//количество посещений клиники данным пациентом равна 0

**}**

**public** string getName**()** **{** **return** name**;** **}**//метод получения имени пациента

**public** string setName **{** **set** **{** **this.**name **=** **value;** **}** **get** **{** **return** name**;** **}** **}**

**public** List GetpHead**()**

**{**

**return** pHead**.**First**;**

**}**

**public** int VisitCount **{** **get** **{** **return** visitCount**;** **}** **}**

**public** void addVisit**(**string \_service**,** int \_date**,** int \_cost**)** //добавление посещения

**{**

Visiting newVisit **=** **new** Visiting**(**\_date**,** \_service**,** \_cost**);**//создаем объект-посещение

List adds **=** **new** List**(**newVisit**);**//создаем ссылку в списке на посещение

visitCount**++;**//увеличиваем количество посещений

**if** **(**pHead**.**First **==** **null** **&&** pHead**.**Last **==** **null)**//1-е добавление в пустой список

**{**

pHead**.**First **=** adds**;**

pHead**.**Last **=** adds**;**

**}**

**else** **if** **(**\_service**[**0**]** **<** pHead**.**First**.**info**.**getService**()[**0**])**//упорядочиваем элемент, если он должен встать перед первым элементом

**{**

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

pHead**.**First **=** adds**;**

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

**}**

**else** **if** **(**\_service**[**0**]** **>=** pHead**.**Last**.**info**.**getService**()[**0**])**//упорядочиваем элемент, если он должен встать после последнего элемента

**{**

pTemp **=** pHead**.**Last**;**

pHead**.**Last **=** adds**;**

pTemp**.**Next **=** pHead**.**Last**;**

**}**

**else**//упорядочиваем элемент, если он должен встать куда то в середину списка

**{**

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

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

**{**

**if** **((**\_service**[**0**]** **>** pTemp**.**info**.**getService**()[**0**])** **&&** **(**\_service**[**0**]<** pTemp**.**Next**.**info**.**getService**()[**0**]))**

**{**

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

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

**break;**

**}**

**else**

**{**

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

**}**

**}**

/\*while ((pTemp.info.getService()[0] > \_service[0]) && (pTemp.Next.info.getService()[0] <= \_service[0]))

{

pTemp = pTemp.Next;

}\*/

**}**

**}**

**public** int SearchInt**(**string \_service**)**

**{**

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

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

**{**

**if(**vspom**.**info**.**getService**().**Equals**(**\_service**))**

**{**

**return** i**;**

**}**

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

**}**

**return** **-**1**;**

**}**

**public** void delVisit**(**int \_date**,** string \_service**)** //удаление посещения

**{**

pTemp **=** pHead**.**First**;**//вспомогательная переменная с ссылкой на первый элемент в списке

**while** **(**pTemp**.**Next **!=** **null)**

**{**

**if** **(**pTemp **==** pHead**.**First**)**//удаление первого элемента списка

**{**

**if** **(**pTemp**.**info**.**getDate**()** **==** \_date **&&** pTemp**.**info**.**getService**().**Equals**(**\_service**))**

**{**

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

visitCount**--;**

**}**

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

**}**

**else**//удаление всех остальных элементов

**{**

**if** **(**pTemp**.**Next**.**info**.**getDate**()** **==** \_date **&&** pTemp**.**Next**.**info**.**getService**().**Equals**(**\_service**))**

**{**

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

visitCount**--;**

**}**

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

**}**

**}**

**}**

**public** int summa**()** //сумма потраченных денег

**{**

sum **=** 0**;**

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

**while** **(**pTemp **!=** **null)**

**{**

sum **+=** pTemp**.**info**.**getCost**();**

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

**}**

**return** sum**;**

**}**

**public** string print**()** //вывод информации о пациенте

**{**

string prints **=** "\n\nName patient - " **+** name**;**

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

**while** **(**pTemp **!=** **null)**

**{**

prints **+=** "\nService - " **+** pTemp**.**info**.**getService**();**

prints **+=** "\nDate visiting - " **+** pTemp**.**info**.**getDate**();**

prints **+=** " Cost service - " **+** pTemp**.**info**.**getCost**();**

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

**}**

prints **+=** "\nAmount of funds spent - " **+** summa**();**

prints **+=** "\nNumber of visits - " **+** visitCount**;**

**return** prints**;**

**}**

**public** string printLoad**()** //вывод информации о пациенте

**{**

string prints **=**""**;**

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

prints **+=** visitCount**;**

**while** **(**pTemp **!=** **null)**

**{**

prints **+=** "\n" **+** pTemp**.**info**.**getService**();**

prints **+=** "\n"**+**pTemp**.**info**.**getDate**();**

prints **+=** "\n" **+** pTemp**.**info**.**getCost**();**

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

**}**

**return** prints**;**

**}**

**}**

**}**

Структура данных список необходима в классе клиент для хранения посещений .

**using** System**;**

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

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

**namespace** \_3ЭтапКлиника

**{**

class List

**{**

**public** Visiting info**;** //данные посещения

**private** List tempNext**;** //вспомогательная переменная-ссылка на потомка

**private** List last**,**first**;**//заголовок списка

**public** List**(**Visiting ainfo**)** //конструктор

**{**

info **=** ainfo**;**

tempNext **=** **null;**

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

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

**}**

**public** List Next//get и set для переменной next

**{**

**get** **{** **return** tempNext**;** **}**

**set** **{** tempNext **=** **value;** **}**

**}**

**public** List Last//get и set для переменной last

**{**

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

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

**}**

**public** List First//get и set для переменной first

**{**

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

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

**}**

**}**

**}**

Класс клиника реализует структуру данных стек на основе массива. А также в иерархии данный класс находиться на самой вершине позволяя работать с методами всех реализованных классов.

**using** System**;**

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

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

**namespace** \_3ЭтапКлиника

**{**

class Clinic

**{**

**private** string nameComp**;**//название клиники

**private** int size**;**//длина массива

**private** Patient**[]** stack**;**//массив типа Patient

**private** int SP**,** summ**,** visits**;**//вспомогательная переменная для добавления/удаления

**public** Clinic**(**string \_nameComp**,** int \_size**)**//конструктор класса

**{**

**this.**nameComp **=** \_nameComp**;**

**this.**size **=** \_size**;**

stack **=** **new** Patient**[**size**];**

SP **=** 0**;**

**}**

**public** string Getname**()**

**{**

**return** nameComp**;**

**}**

**public** int GetSp**()**

**{**

**return** SP**;**

**}**

**public** void addPatient**(**string \_name**)** //добавление пациента

**{**

Patient newPatient **=** **new** Patient**(**\_name**);**

**if(**SP **<=** size**)**

**{**

stack**[**SP**]** **=** newPatient**;**

SP**++;**

**}**

**}**

**public** Patient search**(**string \_name**)**//поиск пациента

**{**

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

**{**

**if** **(**stack**[**i**].**getName**().**Equals**(**\_name**))**

**{**

**return** stack**[**i**];**

**}**

**}**

**return** **null;**

**}**

**public** void delPatient**()** //удаление пациента

**{**

**if(**SP **>** 0**)**

**{**

stack**[**SP**-**1**]** **=** **null;**

SP**--;**

**}**

**}**

**public** string summPatient**()** //количество пациентов

**{**

**return** "Summ patient - " **+** SP**;** **;**

**}**

**public** int sumCost**()** //сумма потраченных денег на услуги

**{**

summ **=** 0**;**

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

**{**

summ **+=** stack**[**i**].**summa**();**

**}**

**return** summ**;**

**}**

**public** string info**()** //получение информации о всех клиентах

**{**

string resultAll **=** "name company - " **+** nameComp **+** "\n"**;**

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

**{**

resultAll **+=** "\n" **+** stack**[**i**].**print**();**

**}**

resultAll **+=** "\n\n\nAmount of funds spent all clients - " **+** sumCost**();**

resultAll **+=** "\n" **+** summPatient**();**

**return** resultAll**;**

**}**

**public** string infoLoad**()**

**{**

string resultAllLoad **=** ""**;**

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

**{**

resultAllLoad **+=** "\n" **+** stack**[**i**].**setName**;**

**if** **(**i **>** GetSp**())**

**{**

**break;**

**}**

**}**

resultAllLoad **+=** "\n"**;**

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

**{**

resultAllLoad **+=** "\n" **+** stack**[**i**].**printLoad**();**

**}**

**return** resultAllLoad**;**

**}**

**}**

**}**

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

**using** System**;**

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

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

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

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

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

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

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

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

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

**namespace** \_3ЭтапКлиника

**{**

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

**{**

Clinic Ptn **=** **new** Clinic**(**"insaf"**,** 10**);**

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

**{**

InitializeComponent**();**

**}**

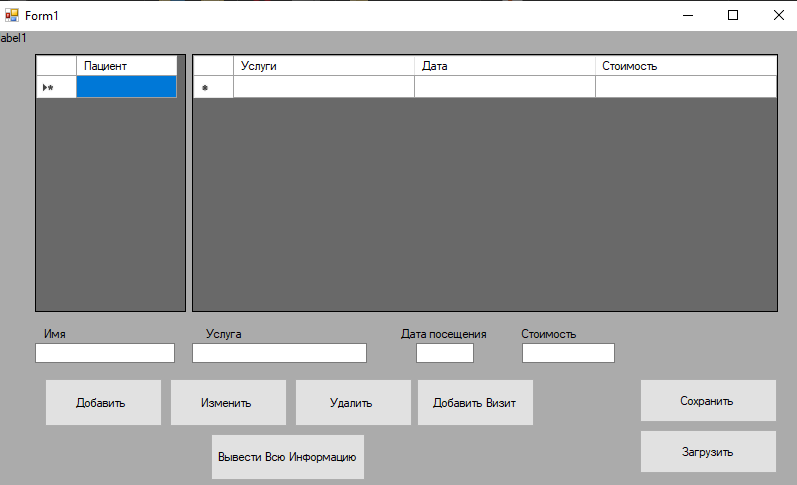


Рис. 3 Работа программы.

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

**{**

**}**

**private** void label3\_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 LoaDinFile\_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"**)**

**{**

**for** **(**int i **=** 0**;** i **<=** Ptn**.**GetSp**();** i**++)**

**{**

Ptn**.**delPatient**();**

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

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

**}**

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

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

**{**

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

Ptn**.**addPatient**(**name**);**

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**)**

**{**

dataGridView2**.**Rows**.**Add**();**

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

**{**

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

int gdata **=** int**.**Parse**(**stream3**.**ReadLine**());** //дата

int gsalaty **=** int**.**Parse**(**stream3**.**ReadLine**());**// цена

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

Ptn**.**search**(**name**).**addVisit**(**inf**,** gdata**,**gsalaty**);**

dataGridView2**.**Rows**[**hall**].**Cells**[**0**].**Value **=** inf**;**

dataGridView2**.**Rows**[**hall**].**Cells**[**1**].**Value**=** gdata**;**

dataGridView2**.**Rows**[**hall**].**Cells**[**2**].**Value **=** gsalaty**;**

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

i**++;**

**else**

**break;**

**}**

hall**++;**

**}**

**}**

**}**

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

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

**}**

**}**

Метод добавления пациента. Обновляется таблица новыми данными и данные передаются в экземпляр класса.

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

**{**

string name **=** textBox1**.**Text**;**

Ptn**.**addPatient**(**name**);**

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

**}**

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

**{**

MessageBox**.**Show**(**Ptn**.**info**());**

**}**

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

**{**

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

Ptn**.**delPatient**();**

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

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

**}**

**private** void Form1\_Load**(object** sender**,** EventArgs e**)**

**{**

**}**

Метод добавления в таблицу и в список посещения. Данные берутся и TextBox и передаются в таблицу и экземпляр класса клиники.

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

**{**

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

Ptn**.**search**(**text**).**addVisit**(**textBox2**.**Text**,**Convert**.**ToInt32**(**textBox3**.**Text**),**Convert**.**ToInt32**(**textBox4**.**Text**));**

int i **=** Ptn**.**search**(**text**).**SearchInt**(**textBox2**.**Text**);**

**if** **(**i **>=** 0**)**

**{**

dataGridView2**.**Rows**.**Add**();**

**if** **(**dataGridView2**.**Rows**[**i**].**Cells**[**0**].**Value **==** **null)**

**{**

dataGridView2**.**Rows**[**i**].**Cells**[**0**].**Value **=** textBox2**.**Text**;**

dataGridView2**.**Rows**[**i**].**Cells**[**1**].**Value **=** textBox3**.**Text**;**

dataGridView2**.**Rows**[**i**].**Cells**[**2**].**Value **=** textBox4**.**Text**;**

textBox2**.**Clear**();**

textBox3**.**Clear**();**

textBox4**.**Clear**();**

**}**

**else**

**{**

**for** **(**int j **=** Ptn**.**search**(**text**).**VisitCount**;** j **>** i**;** j**--)**

**{**

dataGridView2**.**Rows**[**j**].**Cells**[**0**].**Value **=** dataGridView2**.**Rows**[**j **-** 1**].**Cells**[**0**].**Value**;**

dataGridView2**.**Rows**[**j**].**Cells**[**1**].**Value **=** dataGridView2**.**Rows**[**j **-** 1**].**Cells**[**1**].**Value**;**

dataGridView2**.**Rows**[**j**].**Cells**[**2**].**Value **=** dataGridView2**.**Rows**[**j **-** 1**].**Cells**[**2**].**Value**;**

dataGridView2**.**Rows**[**j **-** 1**].**Cells**[**0**].**Value **=** **null;**

dataGridView2**.**Rows**[**j **-** 1**].**Cells**[**1**].**Value **=** **null;**

dataGridView2**.**Rows**[**j **-** 1**].**Cells**[**2**].**Value **=** **null;**

**}**

dataGridView2**.**Rows**[**i**].**Cells**[**0**].**Value **=** textBox2**.**Text**;**

dataGridView2**.**Rows**[**i**].**Cells**[**1**].**Value **=** textBox3**.**Text**;**

dataGridView2**.**Rows**[**i**].**Cells**[**2**].**Value **=** textBox4**.**Text**;**

**}**

**}**

**else** MessageBox**.**Show**(**"Добавить не получилось"**);** **;**

**}**

Метод отображающий посещения когда пользователь выбирает пациента в таблице с пациентами.

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

**{**

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

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

Patient vspom **=** Ptn**.**search**(**text**);**

List help **=** vspom**.**GetpHead**();**

**while(**help **!=** **null)**

**{**

dataGridView2**.**Rows**.**Add**(**help**.**info**.**getService**(),**help**.**info**.**getDate**(),**help**.**info**.**getCost**());**

help **=** help**.**Next**;**

**}**

**}**

Метод изменения данных о пациенте. Читаем с TextBox и изменяем таблицу и экземпляр класса.

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

**{**

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

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

Ptn**.**search**(**choose**).**setName **=** textBox1**.**Text**;**

dataGridView1**.**Rows**[**index**].**Cells**[**0**].**Value **=** textBox1**.**Text**;**

**}**

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 SaVeinFile\_Click**(object** sender**,** EventArgs e**)**

**{**

string path**;**

string fileText **=** "Файл"**;**

//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 += Ptn.Getname() + "\n";

fileText**+=** "\n"**+** Ptn**.**GetSp**();**

fileText **+=** Ptn**.**infoLoad**();**

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

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

**{**

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

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

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

**}**

**}**

**}**

**}**

**}**

**}**

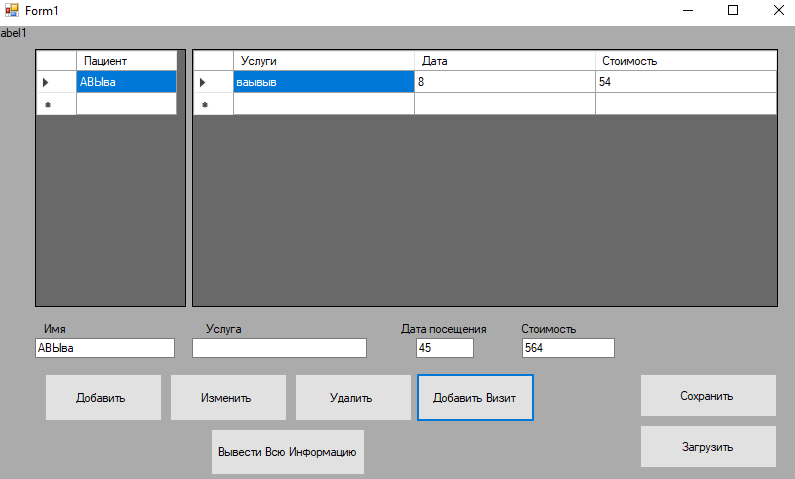


Рис 4 работа программы

1. Описание структуры проекта в соответствии с использованным инструментом разработки

Программа была разработана в среде разработки Visual Studio 2022. Это универсальная среда разработке позволяющая разрабатывать программы на языке программирования C# , а так же С++ и Python. IDE позволяет созывать комплексные программы, а интуитивно понятный интерфейс позволяет легко разобраться в программе и концентрироваться только на разработке.

Помимо удобного интерфейса программа позволяет тестировать программу на ресурсоёмкость, показывая программисту потребление программой ресурсов оперативной памяти и процессора.

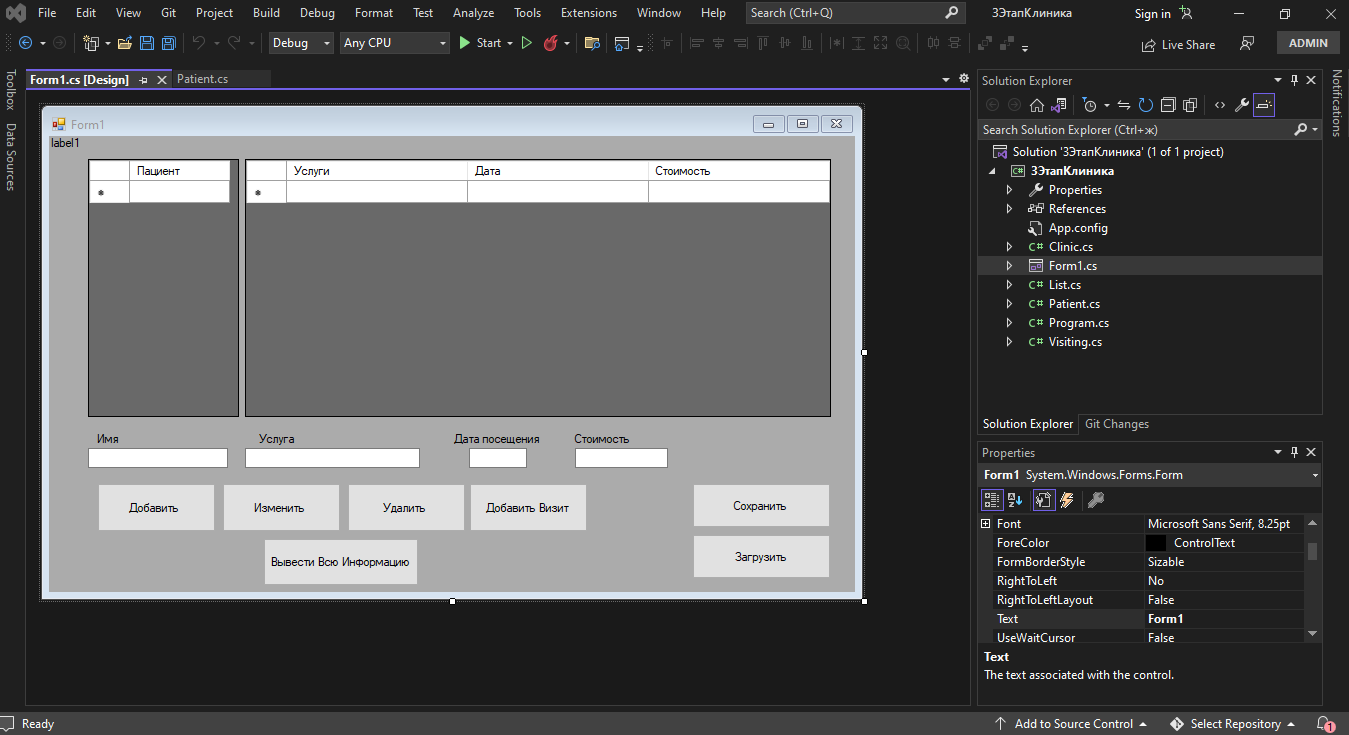


Рис. 5 Работа Visual Studio 2022

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.

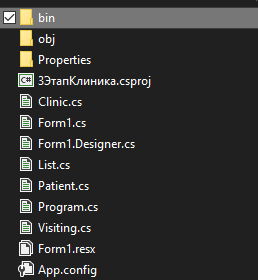


Рис. 6 Содержимое папки проекта.

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

**using** System**;**

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

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

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

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

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

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

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

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

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

**namespace** \_3ЭтапКлиника

**{**

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

**{**

Clinic Ptn **=** **new** Clinic**(**"insaf"**,** 10**);**

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

**{**

InitializeComponent**();**

**}**

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

**{**

**}**

**private** void label3\_Click**(object** sender**,** EventArgs e**)**

**{**

**}**

**private** void LoaDinFile\_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"**)**

**{**

**for** **(**int i **=** 0**;** i **<=** Ptn**.**GetSp**();** i**++)**

**{**

Ptn**.**delPatient**();**

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

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

**}**

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

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

**{**

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

Ptn**.**addPatient**(**name**);**

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**)**

**{**

dataGridView2**.**Rows**.**Add**();**

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

**{**

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

int gdata **=** int**.**Parse**(**stream3**.**ReadLine**());** //дата

int gsalaty **=** int**.**Parse**(**stream3**.**ReadLine**());**// цена

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

Ptn**.**search**(**name**).**addVisit**(**inf**,** gdata**,**gsalaty**);**

dataGridView2**.**Rows**[**hall**].**Cells**[**0**].**Value **=** inf**;**

dataGridView2**.**Rows**[**hall**].**Cells**[**1**].**Value**=** gdata**;**

dataGridView2**.**Rows**[**hall**].**Cells**[**2**].**Value **=** gsalaty**;**

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

i**++;**

**else**

**break;**

**}**

hall**++;**

**}**

**}**

**}**

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

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

**}**

**}**

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

**{**

string name **=** textBox1**.**Text**;**

Ptn**.**addPatient**(**name**);**

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

**}**

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

**{**

MessageBox**.**Show**(**Ptn**.**info**());**

**}**

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

**{**

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

Ptn**.**delPatient**();**

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

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

**}**

**private** void Form1\_Load**(object** sender**,** EventArgs e**)**

**{**

**}**

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

**{**

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

Ptn**.**search**(**text**).**addVisit**(**textBox2**.**Text**,**Convert**.**ToInt32**(**textBox3**.**Text**),**Convert**.**ToInt32**(**textBox4**.**Text**));**

int i **=** Ptn**.**search**(**text**).**SearchInt**(**textBox2**.**Text**);**

**if** **(**i **>=** 0**)**

**{**

dataGridView2**.**Rows**.**Add**();**

**if** **(**dataGridView2**.**Rows**[**i**].**Cells**[**0**].**Value **==** **null)**

**{**

dataGridView2**.**Rows**[**i**].**Cells**[**0**].**Value **=** textBox2**.**Text**;**

dataGridView2**.**Rows**[**i**].**Cells**[**1**].**Value **=** textBox3**.**Text**;**

dataGridView2**.**Rows**[**i**].**Cells**[**2**].**Value **=** textBox4**.**Text**;**

textBox2**.**Clear**();**

textBox3**.**Clear**();**

textBox4**.**Clear**();**

**}**

**else**

**{**

**for** **(**int j **=** Ptn**.**search**(**text**).**VisitCount**;** j **>** i**;** j**--)**

**{**

dataGridView2**.**Rows**[**j**].**Cells**[**0**].**Value **=** dataGridView2**.**Rows**[**j **-** 1**].**Cells**[**0**].**Value**;**

dataGridView2**.**Rows**[**j**].**Cells**[**1**].**Value **=** dataGridView2**.**Rows**[**j **-** 1**].**Cells**[**1**].**Value**;**

dataGridView2**.**Rows**[**j**].**Cells**[**2**].**Value **=** dataGridView2**.**Rows**[**j **-** 1**].**Cells**[**2**].**Value**;**

dataGridView2**.**Rows**[**j **-** 1**].**Cells**[**0**].**Value **=** **null;**

dataGridView2**.**Rows**[**j **-** 1**].**Cells**[**1**].**Value **=** **null;**

dataGridView2**.**Rows**[**j **-** 1**].**Cells**[**2**].**Value **=** **null;**

**}**

dataGridView2**.**Rows**[**i**].**Cells**[**0**].**Value **=** textBox2**.**Text**;**

dataGridView2**.**Rows**[**i**].**Cells**[**1**].**Value **=** textBox3**.**Text**;**

dataGridView2**.**Rows**[**i**].**Cells**[**2**].**Value **=** textBox4**.**Text**;**

**}**

**}**

**else** MessageBox**.**Show**(**"Добавить не получилось"**);** **;**

**}**

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

**{**

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

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

Patient vspom **=** Ptn**.**search**(**text**);**

List help **=** vspom**.**GetpHead**();**

**while(**help **!=** **null)**

**{**

dataGridView2**.**Rows**.**Add**(**help**.**info**.**getService**(),**help**.**info**.**getDate**(),**help**.**info**.**getCost**());**

help **=** help**.**Next**;**

**}**

**}**

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

**{**

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

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

Ptn**.**search**(**choose**).**setName **=** textBox1**.**Text**;**

dataGridView1**.**Rows**[**index**].**Cells**[**0**].**Value **=** textBox1**.**Text**;**

**}**

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

**{**

string path**;**

string fileText **=** "Файл"**;**

//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 += Ptn.Getname() + "\n";

fileText**+=** "\n"**+** Ptn**.**GetSp**();**

fileText **+=** Ptn**.**infoLoad**();**

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

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

**{**

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

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

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

**}**

**}**

**}**

**}**

**}**

**}**

**using** System**;**

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

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

**namespace** \_3ЭтапКлиника

**{**

class List

**{**

**public** Visiting info**;** //данные посещения

**private** List tempNext**;** //вспомогательная переменная-ссылка на потомка

**private** List last**,**first**;**//заголовок списка

**public** List**(**Visiting ainfo**)** //конструктор

**{**

info **=** ainfo**;**

tempNext **=** **null;**

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

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

**}**

**public** List Next//get и set для переменной next

**{**

**get** **{** **return** tempNext**;** **}**

**set** **{** tempNext **=** **value;** **}**

**}**

**public** List Last//get и set для переменной last

**{**

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

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

**}**

**public** List First//get и set для переменной first

**{**

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

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

**}**

**}**

**}**

**using** System**;**

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

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

**namespace** \_3ЭтапКлиника

**{**

class Patient

**{**

**private** string name**;**//фамилия пациента

**private** int visitCount**;**//количество посещений

**private** int sum**;**//сумма потраченных денег

**private** List pHead**,**pTemp**;**//ссылка на первый элемент

**public** Patient**(**string \_name**)**//создание конструктора

**{**

name **=** \_name**;**//присваиваем имя пациенту

pHead **=** **new** List**(null);**//создаем заголовок в списке с пустыми полезными данными

pHead**.**First **=** **null;**//ссылка на первый элемент равну нулл

pHead**.**Last **=** **null;**//ссылка на последний элемент равна нулл

pTemp **=** **null;**//вспомогательная переменная так-же равна нулл

visitCount **=** 0**;**//количество посещений клиники данным пациентом равна 0

**}**

**public** string getName**()** **{** **return** name**;** **}**//метод получения имени пациента

**public** string setName **{** **set** **{** **this.**name **=** **value;** **}** **get** **{** **return** name**;** **}** **}**

**public** List GetpHead**()**

**{**

**return** pHead**.**First**;**

**}**

**public** int VisitCount **{** **get** **{** **return** visitCount**;** **}** **}**

**public** void addVisit**(**string \_service**,** int \_date**,** int \_cost**)** //добавление посещения

**{**

Visiting newVisit **=** **new** Visiting**(**\_date**,** \_service**,** \_cost**);**//создаем объект-посещение

List adds **=** **new** List**(**newVisit**);**//создаем ссылку в списке на посещение

visitCount**++;**//увеличиваем количество посещений

**if** **(**pHead**.**First **==** **null** **&&** pHead**.**Last **==** **null)**//1-е добавление в пустой список

**{**

pHead**.**First **=** adds**;**

pHead**.**Last **=** adds**;**

**}**

**else** **if** **(**\_service**[**0**]** **<** pHead**.**First**.**info**.**getService**()[**0**])**//упорядочиваем элемент, если он должен встать перед первым элементом

**{**

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

pHead**.**First **=** adds**;**

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

**}**

**else** **if** **(**\_service**[**0**]** **>=** pHead**.**Last**.**info**.**getService**()[**0**])**//упорядочиваем элемент, если он должен встать после последнего элемента

**{**

pTemp **=** pHead**.**Last**;**

pHead**.**Last **=** adds**;**

pTemp**.**Next **=** pHead**.**Last**;**

**}**

**else**//упорядочиваем элемент, если он должен встать куда то в середину списка

**{**

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

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

**{**

**if** **((**\_service**[**0**]** **>** pTemp**.**info**.**getService**()[**0**])** **&&** **(**\_service**[**0**]<** pTemp**.**Next**.**info**.**getService**()[**0**]))**

**{**

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

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

**break;**

**}**

**else**

**{**

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

**}**

**}**

/\*while ((pTemp.info.getService()[0] > \_service[0]) && (pTemp.Next.info.getService()[0] <= \_service[0]))

{

pTemp = pTemp.Next;

}\*/

**}**

**}**

**public** int SearchInt**(**string \_service**)**

**{**

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

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

**{**

**if(**vspom**.**info**.**getService**().**Equals**(**\_service**))**

**{**

**return** i**;**

**}**

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

**}**

**return** **-**1**;**

**}**

**public** void delVisit**(**int \_date**,** string \_service**)** //удаление посещения

**{**

pTemp **=** pHead**.**First**;**//вспомогательная переменная с ссылкой на первый элемент в списке

**while** **(**pTemp**.**Next **!=** **null)**

**{**

**if** **(**pTemp **==** pHead**.**First**)**//удаление первого элемента списка

**{**

**if** **(**pTemp**.**info**.**getDate**()** **==** \_date **&&** pTemp**.**info**.**getService**().**Equals**(**\_service**))**

**{**

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

visitCount**--;**

**}**

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

**}**

**else**//удаление всех остальных элементов

**{**

**if** **(**pTemp**.**Next**.**info**.**getDate**()** **==** \_date **&&** pTemp**.**Next**.**info**.**getService**().**Equals**(**\_service**))**

**{**

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

visitCount**--;**

**}**

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

**}**

**}**

**}**

**public** int summa**()** //сумма потраченных денег

**{**

sum **=** 0**;**

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

**while** **(**pTemp **!=** **null)**

**{**

sum **+=** pTemp**.**info**.**getCost**();**

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

**}**

**return** sum**;**

**}**

**public** string print**()** //вывод информации о пациенте

**{**

string prints **=** "\n\nName patient - " **+** name**;**

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

**while** **(**pTemp **!=** **null)**

**{**

prints **+=** "\nService - " **+** pTemp**.**info**.**getService**();**

prints **+=** "\nDate visiting - " **+** pTemp**.**info**.**getDate**();**

prints **+=** " Cost service - " **+** pTemp**.**info**.**getCost**();**

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

**}**

prints **+=** "\nAmount of funds spent - " **+** summa**();**

prints **+=** "\nNumber of visits - " **+** visitCount**;**

**return** prints**;**

**}**

**public** string printLoad**()** //вывод информации о пациенте

**{**

string prints **=**""**;**

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

prints **+=** visitCount**;**

**while** **(**pTemp **!=** **null)**

**{**

prints **+=** "\n" **+** pTemp**.**info**.**getService**();**

prints **+=** "\n"**+**pTemp**.**info**.**getDate**();**

prints **+=** "\n" **+** pTemp**.**info**.**getCost**();**

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

**}**

**return** prints**;**

**}**

**}**

**}**

**using** System**;**

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

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

**namespace** \_3ЭтапКлиника

**{**

class Visiting

**{**

**private** int date**;**

**private** string service**;**//создание необходим переменных

**private** int cost**;**

**public** Visiting**(**int \_date**,** string \_service**,** int \_cost**)**//конструктор класса

**{**

**this.**date **=** \_date**;**

**this.**service **=** \_service**;**

**this.**cost **=** \_cost**;**

**}**

**public** int getDate**()**//получение даты

**{**

**return** date**;**

**}**

**public** string getService**()**//получение оказанной услуги

**{**

**return** service**;**

**}**

**public** int getCost**()**//получение стоимости услуги

**{**

**return** cost**;**

**}**

**public** void setDate**(**int \_date**)**//изменение даты

**{** **this.**date **=** \_date**;** **}**

**public** void setService**(**string \_service**)**//изменение оказанной услуги

**{** **this.**service **=** \_service**;** **}**

**public** void setCost**(**int \_cost**)**//изменение стоимости

**{** **this.**cost **=** \_cost**;** **}**

**}**

**}**

**using** System**;**

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

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

**namespace** \_3ЭтапКлиника

**{**

class Clinic

**{**

**private** string nameComp**;**//название клиники

**private** int size**;**//длина массива

**private** Patient**[]** stack**;**//массив типа Patient

**private** int SP**,** summ**,** visits**;**//вспомогательная переменная для добавления/удаления

**public** Clinic**(**string \_nameComp**,** int \_size**)**//конструктор класса

**{**

**this.**nameComp **=** \_nameComp**;**

**this.**size **=** \_size**;**

stack **=** **new** Patient**[**size**];**

SP **=** 0**;**

**}**

**public** string Getname**()**

**{**

**return** nameComp**;**

**}**

**public** int GetSp**()**

**{**

**return** SP**;**

**}**

**public** void addPatient**(**string \_name**)** //добавление пациента

**{**

Patient newPatient **=** **new** Patient**(**\_name**);**

**if(**SP **<=** size**)**

**{**

stack**[**SP**]** **=** newPatient**;**

SP**++;**

**}**

**}**

**public** Patient search**(**string \_name**)**//поиск пациента

**{**

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

**{**

**if** **(**stack**[**i**].**getName**().**Equals**(**\_name**))**

**{**

**return** stack**[**i**];**

**}**

**}**

**return** **null;**

**}**

**public** void delPatient**()** //удаление пациента

**{**

**if(**SP **>** 0**)**

**{**

stack**[**SP**-**1**]** **=** **null;**

SP**--;**

**}**

**}**

**public** string summPatient**()** //количество пациентов

**{**

**return** "Summ patient - " **+** SP**;** **;**

**}**

**public** int sumCost**()** //сумма потраченных денег на услуги

**{**

summ **=** 0**;**

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

**{**

summ **+=** stack**[**i**].**summa**();**

**}**

**return** summ**;**

**}**

**public** string info**()** //получение информации о всех клиентах

**{**

string resultAll **=** "name company - " **+** nameComp **+** "\n"**;**

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

**{**

resultAll **+=** "\n" **+** stack**[**i**].**print**();**

**}**

resultAll **+=** "\n\n\nAmount of funds spent all clients - " **+** sumCost**();**

resultAll **+=** "\n" **+** summPatient**();**

**return** resultAll**;**

**}**

**public** string infoLoad**()**

**{**

string resultAllLoad **=** ""**;**

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

**{**

resultAllLoad **+=** "\n" **+** stack**[**i**].**setName**;**

**if** **(**i **>** GetSp**())**

**{**

**break;**

**}**

**}**

resultAllLoad **+=** "\n"**;**

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

**{**

resultAllLoad **+=** "\n" **+** stack**[**i**].**printLoad**();**

**}**

**return** resultAllLoad**;**

**}**

**}**

**}**