СОДЕРЖАНИЕ

[Введение 5](#_Toc504649449)

[1. Aналитическая часть 6](#_Toc504649450)

[1.1. Обоснование актуальности задачи 6](#_Toc504649451)

[1.2. Обоснование выбора и описание методологии разработки программы решения задачи 8](#_Toc504649452)

[1.3. Обоснование выбора и описание инструментальных средств разработки программы решения задачи 9](#_Toc504649453)

[1.4. Содержательная постановка задачи 25](#_Toc504649454)

[2. Проектная часть 31](#_Toc504649455)

[2.1. Разработка структуры данных программы 31](#_Toc504649456)

[2.2. Логическая модель программы 38](#_Toc504649457)

[2.3. Разработка пользовательского интерфейса программы 41](#_Toc504649458)

[2.4. Листинг программы 55](#_Toc504649459)

[2.5. Разработка тестовых данных 55](#_Toc504649460)

[2.6. Результаты работы программы 62](#_Toc504649461)

[2.7. Руководство по использованию программы 69](#_Toc504649462)

[Заключение 85](#_Toc504649463)

[Приложение 1 87](#_Toc504649464)

[Приложение 2 88](#_Toc504649465)

[Приложение 3 89](#_Toc504649466)

[Приложение 4 90](#_Toc504649467)

[Приложение 5 92](#_Toc504649468)

[Приложение 6 93](#_Toc504649469)

[Приложение 7 94](#_Toc504649470)

[Приложение 8 95](#_Toc504649471)

[Приложение 9 96](#_Toc504649472)

[Приложение 10 98](#_Toc504649473)

[Приложение 11 99](#_Toc504649474)

[Приложение 12 100](#_Toc504649475)

ВВЕДЕНИЕ

Одим из важнейших процессов в любой организации является коммуникация между сотрудниками. В настоящее время для этого часто используют телефонию, СМС, факс и популярные мессанджеры, но при использовании не продконтрольных ей продуктов может произойти утечка информации или её полная потеря. Для защиты данных от внешних воздействий рекомендуется использовать собственные системы информационного собщения.

Объектом разработки является электронная коммуникация между сотрудниками организации.

Предметом разработки является система информационного сообщения организации.

Целью данной разработки является проектирование и реализация программного обеспечения для обмена сообщениями между сотрудниками.

# AНАЛИТИЧЕСКАЯ ЧАСТЬ

## Обоснование актуальности задачи

* + 1. Актуальность темы

///

Исходя из этого было разработано программное обеспечение «uMessager», которое учитывает необходимые условия для обмена сообщениями в организации.

* + 1. Анализ существующих разработок по теме

Мессенджер (англ. Messager) — это приложение для обмена мгновенными сообщениями. Приложение может считаться мессанджером, если оно соответствует следующим критериям:

* возможность мгновенной доставки сообщений;
* возможность общения сразу с несколькими собеседниками
  + - 1. Telegram

///

* + - 1. WhatsApp

///

* + - 1. ///

///

## Обоснование выбора и описание методологии разработки программы решения задачи

Важнейшим шагом на пути к совершенствованию языков программирования стало появление объектно-ориентированного подхода к программированию (или, сокращенно, ООП) и соответствующего класса языков. Наиболее известным примером объектно-ориентированного языка программирования является язык C++, развившейся из императивного языка С. Его прямым потомком и логическим продолжением является язык С# на основе которого и будет писаться клиентское приложение данной курсовой работы. При объектно-ориентированном подходе программа представляет собой описание объектов, их свойств (или атрибутов), совокупностей (или классов), отношений между ними, способов их взаимодействия и операций над объектами (или методов). Несомненным преимуществом данного подхода является концептуальная близость к предметной области произвольной структуры и назначения.

Механизм наследования атрибутов и методов позволяет строить производные понятия на основе базовых и таким образом создать модель сколь угодно сложной предметной области с заданными свойствами. В связи с тем, что объектно-ориентированный подход используется в основном для создания больших проектов и, когда важна управляемость проекта, а также скорость разработки и его модифицируемость, поэтому в данной работе он не используется в качестве основного.

Помимо этого, также будет использоваться и структурный. В его основе лежит представление программы в виде иерархической структуры блоков. В соответствии с данной методологией любая программа строится без использования оператора goto из трёх базовых управляющих структур: последовательность, ветвление, цикл; кроме того, используются подпрограммы. При этом разработка программы ведётся пошагово.

## Обоснование выбора и описание инструментальных средств разработки программы решения задачи

Для разработки данного проекта требуется программа для реализации клиентской части, которая будет предусматривать современный и удобный интерфейс для дальнейшего пользователя, в проекте для этой цели использована среда разработки MS Visual Studio 2017. Для реализации же серверной части, которая позволит хранить данные и управлять ими, выбрана база данных MS SQL Server 2014.

* + 1. Обоснование выбора средств разработки клиентской части

В качестве средства разработки клиентской части ПО автоматизации расчета заработной платы предприятия использована среда программирования Microsoft Visual Studio 2017. Среда разработана в соответствии с концепцией визуального программирования.

Microsoft Visual Studio 2017 — это набор инструментов для создания программного обеспечения: от планирования до разработки пользовательского интерфейса, написания кода, тестирования, отладки, анализа качества кода и производительности, развертывания в средах клиентов и сбора данных телеметрии по использованию. Эти инструменты предназначены для максимально эффективной совместной работы; все они доступны в интегрированной среде разработки (IDE) Visual Studio. Еще одним плюсом данной среды является, то, что она не нуждается в каких-либо настройках. По умолчанию Visual Studio обеспечивает поддержку C#, C и C++, JavaScript, F# и Visual Basic. В качестве языка программирования клиентской части данного проекта выбран язык C#.

Visual Studio можно использовать для создания различных типов приложений, от простых приложений для магазина и игр для мобильных телефонов до больших и сложных систем, обслуживающих предприятия и центры обработки данных. Visual Studio позволяет создавать:

* приложения и игры, которые выполняются не только на платформе Windows, но и на Android и iOS;
* веб-сайты и веб-службы на основе ASP.NET, JQuery, AngularJS и других популярных платформ;
* приложения для самых разных платформ и устройств, включая, но не ограничиваясь: Office, Sharepoint, Hololens, Kinect.

Для работы с Microsoft Visual Studio 2017 могут быть использованы следующие операционные системы: Windows7/8/8.1/10.

* + 1. Выбор языка программирования

В качестве языка программирования выбран язык C#.

C# является языком программирования, который разработан для создания множества приложений, работающих в среде .NET Framework. Язык C# прост, типобезопасен и объектно-ориентирован. Благодаря множеству нововведений C# обеспечивает возможность быстрой разработки приложений, но при этом сохраняет выразительность и элегантность, присущую С-подобным языков.

Visual C# — это реализация языка C# корпорацией Майкрософт. Поддержка Visual C# в Visual Studio обеспечивается с помощью полнофункционального редактора кода, компилятора, шаблонов проектов, конструкторов, мастеров кода, мощного и удобного отладчика и многих других средств. Библиотека классов .NET Framework предоставляет доступ ко многим службам операционной системы и к другим полезным, хорошо спроектированным классам, что существенно ускоряет цикл разработки.

C# упрощает разработку компонентов программного обеспечения благодаря нескольким инновационным конструкциям языка, в число которых входят следующие:

* инкапсулированные сигнатуры методов, называемые делегатами, которые поддерживают типобезопасные уведомления о событиях;
* свойства, выступающие в роли методов доступа для закрытых переменных-членов;
* атрибуты с декларативными метаданными о типах во время выполнения;
* встроенные комментарии XML-документации;
* LINQ, предлагающий встроенные возможности запросов в различных источниках данных.
  + 1. Общие сведения о языке программирования

Использованный язык C# является объектно-ориентированным языком программирования**.**

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Pascal, Модула, Smalltalk и, в особенности, Java — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

* + - 1. Элементы языка

1) Переменные

Синтаксис объявления переменных в C# представлен на рисунке 1.

ТипДанных Идентификатор;

Рис. 1 – Объявление переменной

Пример объявления переменной представлен на рисунке 2.

int i;

Рис. 2 – Пример объявления переменной

Объявить можно переменную любого действительного типа. Важно подчеркнуть, что возможности переменной определяются ее типом. Например, переменную типа bool нельзя использовать для хранения числовых значений с плавающей точкой. Кроме того, тип переменной нельзя изменять в течение срока ее существования. В частности, переменную типа int нельзя преобразовать в переменную типа char.

Все переменные в C# должны быть объявлены до их применения. Это нужно для того, чтобы уведомить, компилятор о типе данных, хранящихся в переменной, прежде чем он попытается правильно скомпилировать любой оператор, в котором используется переменная. Это позволяет также осуществлять строгий контроль типов в C#.

Начиная с версии C# 3.0, компилятору предоставляется возможность самому определить тип локальной переменной. Такая переменная называется неявно типизированной и объявляется с помощью ключевого слова var. Пример объявления переменной представлен на рисунке 3.

var i = 12;

Рис. 3 – Объявление переменной

2) Операция присваивания

Задать значение переменной можно, в частности, с помощью оператора присваивания. Кроме того, задать начальное значение переменной можно при ее объявлении. Для этого после имени переменной указывается знак равенства и присваиваемое значение. Пример присваивания переменной представлен на рисунке 4.

int i = 42;

Рис. 4 – Присваивание переменной

3) Комментарии

Комментарий - строка, которая не выполняется программой. Служит для записи пояснений к коду, а также исключения (комментирования) строк кода, которые не надо выполнять, но и удалять нельзя (например, чтобы были видны проведенные изменения). В C# используются традиционные комментарии в стиле С — однострочные и многострочные. Запись комментариев изображена на рисунке 5.

//Однострочный комментарий

/\*Многострочный

Комментарий\*/

Рис. 5 - Комментарии

4) Конструкция перехода по условию

Если условие выполняется, то выполняется блок кода 1, а блок кода 2 игнорируется. Если условие не выполняется, то выполняется блок кода 2, а блок кода 1 игнорируется:

Условные конструкции - один из базовых компонентов многих языков программирования, которые направляют работу программы по одному из путей в зависимости от определенных условий.

В языке C# используются следующие условные конструкции: if..else и switch..case.

Конструкция if/else проверяет истинность некоторого условия и в зависимости от результатов проверки выполняет определенный код. Пример конструкции if/else представлен на рисунках 6-7.

int num1 = 8;int num2 = 6;

if(num1 > num2)

{

    Console.WriteLine("Число {0} большечисла {1}", num1, num2);

Рис. 6 – Конструкция if/else

|  |
| --- |
| }  else  {      Console.WriteLine("Число {0} меньшечисла {1}", num1, num2);  } |

Рис. 7 – Конструкция if/else

После ключевого слова if ставится условие. И если это условие выполняется, то срабатывает код, который помещен в блоке if после фигурных скобок. Если условие не выполняется, срабатывает кусок кода после слова else.

Конструкция switch/case аналогична конструкции if/else, так как позволяет обработать сразу несколько условий. Пример конструкции switch/case представлен на рисунке 8.

Console.WriteLine("Нажмите Y или N");

string selection = Console.ReadLine();

switch (selection)

{

    case "Y":

        Console.WriteLine("Вынажалибукву Y");

        break;

    case "N":

        Console.WriteLine("Вынажалибукву N");

        break;

    default:

        Console.WriteLine("Вынажалинеизвестнуюбукву");

        break;

}

Рис. 8 - Конструкция switch/case

После ключевого слова switch в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями, помещенными после оператора case. И если совпадение будет найдено, то будет выполняться определенный блок case.

В конце каждого блока case должен ставиться один из операторов перехода: break,goto case,return или throw. Как правило, используется оператор break. При его применении другие блоки case выполняться не будут.

Однако если нужно, чтобы, после выполнения текущего блока case выполнялся другой блок case, то можно использовать вместо break оператор goto case.

Если мы необходимо также обработать ситуацию, когда совпадения не будет найдено, то можно добавить блок default, как в примере выше.

Применение оператора return позволит выйти не только из блока case, но и из вызывающего метода. То есть, если в методе Main после конструкции switch..case, в которой используется оператор return, идут какие-либо операторы и выражения, то они выполняться не будут, а метод Main завершит работу.

Оператор throw применяется для выброса ошибок.

5) Циклы

Циклы также являются управляющими конструкциями, позволяя в зависимости от определенных условий выполнять некоторое действие множество раз. В C# имеются следующие виды циклов:

* for;
* foreach;
* while;
* do...while.

Цикл for представлен на рисунке 9.

for ([инициализация счетчика]; [условие]; [изменение счетчика])

{

    // действия

}

Рис. 9 – Цикл for

Цикл foreach:

Цикл foreach предназначен для перебора элементов в контейнерах. Конструкция цикла foreach представлена на рисунке 10.

foreach (тип\_данныхназвание\_переменнойin контейнер)

{

    // действия

}

Рис. 10 – Цикл foreach

Цикл do:

В цикле do сначала выполняется код цикла, а потом происходит проверка условия в инструкции while. И пока это условие истинно, цикл повторяется. Конструкция цикла do представлена на рисунке 11.

do

{// действия }

while ([условие]);

Рис. 11 – Цикл do

Цикл while:

В отличие от цикла do цикл while сразу проверяет истинность некоторого условия, и если условие истинно, то код цикла выполняется. Конструкция цикла while представлена на рисунке 12.

while ([условие])

{

    // действия

}

Рис. 12 – Цикл while

Иногда возникает ситуация, когда требуется выйти из цикла, не дожидаясь его завершения. В этом случае используется оператор break. Если нужно, чтобы при проверке цикл не завершался, а просто переходил к следующему элементу, используется оператор continue.

6) Операторы передачи управления

В C# есть пять операторов, изменяющих естественный порядок выполнения вычислений:

* оператор безусловного перехода goto;
* оператор выхода из цикла break;
* оператор перехода к следующей итерации цикла continue;
* оператор возврата из функции return;
* оператор генерации исключения throw.

Оператор goto-метка передает управление на помеченный оператор. Метка — это обычный идентификатор, областью видимости которого является функция, в теле которой он задан. Метка должна находиться в той же области видимости, что и оператор перехода.

Оператор break используется внутри операторов цикла или выбора для перехода в точку программы, находящуюся непосредственно за оператором, внутри которого находится оператор break.

Оператор перехода к следующей итерации текущего цикла continue пропускает все операторы, оставшиеся до конца тела цикла, и передает управление на начало следующей итерации.

Оператор возврата из функции return завершает выполнение функции и передает управление в точку ее вызова.

* + - 1. Способы структурирования программы

1) Процедура

Процедура или метод - часть программного модуля, предназначенная для выполнения определенной задачи.

Процедура может иметь входящие параметры, а может их не иметь – все зависит от ее назначения.

В C# определение метода состоит из любых модификаторов (таких как спецификация доступности), типа возвращаемого значения, за которым следует имя метода, затем список аргументов в круглых скобках и далее - тело метода в фигурных скобках. Конструкция процедуры представлена на рисунке 13.

[модификаторы] тип\_возвратаИмяМетода([параметры])

{

// Тело метода

}

Рис. 13 – Конструкция процедуры

Каждый параметр состоит из имени типа параметра и имени, по которому к нему можно обратиться в теле метода. Вдобавок, если метод возвращает значение, то для указания точки выхода должен использоваться оператор возврата return вместе с возвращаемым значением.

Если метод не возвращает ничего, то в качестве типа возврата указывается void. Если же он не принимает аргументов, то все равно после имени метода должны присутствовать пустые круглые скобки. При этом включать в тело метода оператор возврата не обязательно — метод возвращает управление автоматически по достижении закрывающей фигурной скобки.

В целом, возврат из метода может произойти при двух условиях. Во-первых, когда встречается фигурная скобка, закрывающая тело метода. И во-вторых, когда выполняется оператор return. Имеются две формы оператора return: одна — для методов типа void (возврат из метода), а другая — для методов, возвращающих конкретные значения (возврат значения).

2) Функция

Тоже что и процедура, но есть одна особенность – функция возвращает некий результат, что оформляется с помощью ключевого слова «Return».

Синтаксис функции представлен на рисунке 14.

[модификаторы] тип\_возвратаИмяФункции([параметры])

{

Return [Возвращаемое значение];// Тело функции

}

Рис. 14 – Конструкция функции

Программа на языке C# может состоять из одного или нескольких файлов. Каждый файл может содержать нуль или несколько пространств имен. Пространство имен может содержать типы, такие как классы, структуры, интерфейсы, перечисления и делегаты, а также другие пространства имен. На рисунке 15 приведена структура программы на C#.

usingSystem;

namespaceYourNamespace

{

classYourClass

{ }

interfaceIYourInterface

{ }

delegate int YourDelegate();

enumYourEnum

{ }

namespaceYourNestedNamespace

{

structYourStruct

{ }

}

classYourMainClass

{

static void Main(string[] args) //Программаначинаетсяздесь

{ } } }

Рис. 15 – Конструкция программы

* + - 1. Средства обмена данными

Средства обмена данными позволяют создавать общий проект на несколько пользователей через подключение к локальной сети. С помощью этих механизмов можно осуществлять разработку раздельную проекта, т.е. есть возможность разработки отдельной части независимо от других частей проекта.

Для возможности совместной разработки используется расширение TeamFoundationServer.

TeamFoundationServer (сокр. TFS) - продукт корпорации Microsoft, представляющий собой комплексное решение, объединяющее в себе систему управления версиями, сбор данных, построение отчётов, отслеживание статусов и изменений по проекту и предназначенное для совместной работы над проектами по разработке программного обеспечения. Данный продукт доступен как в виде отдельного приложения, так и в виде серверной платформы для Visual Studio Team System (VSTS).

Team Foundation Server работает по трёхуровневой архитектуре: клиентский уровень, прикладной уровень и уровень данных. Клиентский уровень используется для создания и управления проектами, а также для доступа к хранимым и управляемым элементам проекта. На этом уровне TFS не содержит никаких пользовательских интерфейсов, но предоставляетвеб-сервисы, которые могут быть использованы клиентскими приложениями для самостоятельной интеграции в функциональность TFS. Эти веб-сервисы используются такими приложениями, как Visual Studio Team System для применения TFS в качестве серверной инфраструктуры хранилища информации или выделенного TFS управления приложениями, наподобие включенного приложения Team Foundation Client. Сами веб-сервисы находятся на прикладном уровне. Прикладной уровень также включает в себя веб-портал и репозиторий (хранилище) документации, поддерживаемые Windows SharePoint Services. Веб-портал, называемый Team Project Portal (портал командного проекта), выступает в роли центра взаимодействия для проектов, управляемых TFS. Репозиторий документов используется как для элементов проекта, так и для отслеживания ревизий (документирование изменений), а также для накопления и обработки данных и генерации отчётов. Уровень данных, основывающийся в первую очередь на установленномSQL Server 2005 Standard Edition, обеспечивает сервисы постоянного хранения данных для репозитория документов. Уровень данных и уровень приложений могут существовать на различных физических или виртуальных серверах при использовании Windows Server 2003 или более специализированных версий. Уровень данных не взаимодействует с клиентским уровнем напрямую, только через прикладной уровень.

* + - 1. Встроенные элементы

Встроенные процедуры и функции представлены в таблице 1.

Таблица 1 – Встроенные процедуры и функции языка C#.

|  |  |
| --- | --- |
| Процедура/Функция | Описание процедуры/функции |
| Res = Math.Round(Cost\*0.5, 2); | Округление |
| Res = Math.Floor(Cost); | Возвращает наибольшее целое число, которое не больше заданной переменной |
| var DbResult = db.ProcedureName(); | Передача данных из процедуры |
| db.TableName.ToList() | Передача данных из таблицы БД |
| db.TableName.Add(data); | Добавляет заданную сущность к контексту, поддерживающему набор, в добавленном состоянии, в результате чего она будет вставлена в базу данных при вызове метода SaveChanges |
| db.Entry<TableName>(data).State = System.Data.Entity.EntityState.Modified; | Изменяет выбранную сущность в контексте, поддерживающему набор, в изменённом состоянии, в результате чего она будет изменена в базе данных при вызове метода SaveChanges |
| db.TableName.Remove(data); | Помечает заданную сущность как удалённую, в результате чего она будет удалена из базы данных при вызове метода SaveChanges |
| await db.SaveChangesAsync(); | Асинхронно сохраняет все изменения основной базы данных, произведённые в контексте |
| base.Dispose(); | Устраняет неуправляемые и, если указано, управляемые ресурсы |

* + - 1. Средства отладки программы

В случае если на этапе написания какого-либо кода программист допускает ошибку, которая может быть проанализирована ещё ДО компиляции и запуска программы – Visual Studio об этом сообщит в окне «Список ошибок». В основном на этапе разработки фиксируются, если можно так выразиться, элементарные ошибки: пропуск запятой, несоответствие видимости классов, полей и т.д., то ошибки, которые могут быть исправлены быстро.

Довольно часто могут возникать ситуации, когда на этапе выполнения программы в какой-либо метод передается неверный параметр, скажем, в качестве пути к локальной базе данных, может случайно передаться пустая строка – в этом случае обязательно возникнет исключительная ситуация. Такие моменты студия также фиксирует, если приложение запускается из среды IDE. Visual Studio выводит окно, в котором дается небольшое пояснение об исключительной ситуации. Здесь можно:

* просмотреть сведения об ошибке в справочных материалах;
* поискать решение проблемы в сети;
* просмотреть сведения об ошибке;
* скопировать сведения исключения в буфер обмена.

При выборе опции «Просмотр сведений» откроется ещё одно окно, в котором сдержится подробная информация об исключительной ситуации.

Точка останова или breakpoint – это преднамеренная остановка выполнения программы, при котором выполняется вывод отладчика.

Чтобы установить breakpoint, необходимо навести курсор на нужную строку, и выбрать в меню «Отладка – Точка останова» или нажать [F9]. При этом строка с точкой останова будет помечена. Как только выполнение программы дойдет до точки останова – выполнение прервется и запустится отладчик. Во время выполнения отладчика можно проверить значения переменных.

## Содержательная постановка задачи