МИНИСТЕРСТВО ОБРАЗОВАНИЯ МОСКОВСКОЙ ОБЛАСТИ
Государственное бюджетное профессиональное образовательное учреждение Московской области
«Воскресенский колледж»

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

ПМ 01. «Разработка модулей программного обеспечения для компьютерных систем»

Наименование специальности

09.02.07 «Информационные системы и программирование»

Квалификация выпускника

Программист

2020г.

Методические рекомендации по выполнению лабораторных работ по профессиональному модулю разработаны на основе Федерального государственного образовательного стандарта (далее – ФГОС) по специальности среднего профессионального образования (далее – СПО)

09.02.07 «Информационные системы и программирование»

Организация разработчик: Государственное бюджетное профессиональное образовательное учреждение Московской области «Воскресенский колледж»

Разработчики:

Вострякова А.В., преподаватель компьютерных дисциплин

Рабочая программа профессионального модуля рассмотрена на заседании предметной (цикловой) комиссией компьютерных дисциплин

«___»_____2020г.

Председатель цикловой комиссии _____/Рязанцева О.В./

Утверждена зам директора по УР _____/Куприна Н.Л./

«___»_____2020 г.

ВВЕДЕНИЕ

Данные методические указания для проведения лабораторных работ по МДК.01.01 «Разработка программных модулей» предназначены для реализации ФГОС СПО по специальности 09.02.07 Информационные системы и программирование с целью закрепления теоретических знаний и практических умений.

В сборнике содержатся методические указания по выполнению 8 лабораторных работ в интегрированной системе программирования C#.

При выполнении лабораторных работ студент должен

иметь практический опыт:

- в разработке кода программного продукта на основе готовой спецификации на уровне модуля;
- использовании инструментальных средств на этапе отладки программного продукта;
- проведении тестирования программного модуля по определенному сценарию;

уметь:

- осуществлять разработку кода программного модуля на языках низкого и высокого уровней;
- создавать программу по разработанному алгоритму как отдельный модуль;
- выполнять отладку и тестирование программы на уровне модуля;
- осуществлять разработку кода программного модуля на современных языках программирования;
- оформлять документацию на программные средства;

знать:

- основные этапы разработки программного обеспечения;
- основные принципы технологии структурного и объектно-ориентированного программирования;
- основные принципы отладки и тестирования программных продуктов;

Каждая лабораторная работа имеет следующую структуру: тема, цели, краткие теоретические сведения, порядок проведения работы, требования к составлению отчета.

После выполнения лабораторной работы студент должен представить отчет о проделанной работе. Оценку по практической работе студент получает, если студентом работа выполнена в полном объеме, студент может пояснить выполнение любого этапа работы, отчет выполнен в соответствии с требованиями к выполнению работы, студент отвечает на контрольные вопросы на удовлетворительную оценку и выше.

Зачет по выполнению лабораторных работ студент получает при условии выполнения всех предусмотренных программой лабораторных работ с отчетами по всем работам.

Отчет к лабораторной работе должен содержать:

- Тему работы
- Задание для выполнения, включая индивидуальное задание
- Описание алгоритма программы, (при необходимости - со схемой алгоритма)
- Описание переменных и структур данных, которые применяются в программе
- Описание ключевых программных решений, принятых при реализации алгоритма в тексте программы
- Текст программы
- Замечания к отладке программы
- Образец результатов программы
- Выводы

Перечень лабораторных работ

1. Работа с классами. Разработка методов класса. Перегрузка методов.
2. Определение операций в классе. Создание наследованных классов
3. Работа с объектами через интерфейсы.
4. Использование стандартных интерфейсов.
5. Работа с типом данных структура.
6. Коллекции. Параметризованные классы.
7. Использование регулярных выражений
8. Операции со списками

Лабораторные работы №1

Работа с классами. Разработка методов класса. Перегрузка методов.

Цель работы: Изучить возможности программирования классов на языке C#; Получить основные навыки программирования манипуляторов ввода/вывода.

Теория

Класс – это абстрактный тип данных. Другими словами, класс – это некоторый шаблон, на основе которого будут создаваться его экземпляры – объекты.

В Си-шарп классы объявляются с помощью ключевого слова class. Общая структура объявления выглядит следующим образом:

```
[модификатор доступа] class [имя_класса]
{
  //тело класса
}
```

Модификаторов доступа для классов есть два:

- public – доступ к классу возможен из любого места одной сборки либо из другой сборки, на которую есть ссылка;
- internal – доступ к классу возможен только из сборки, в которой он объявлен.

Поля класса

Поле – это переменная, объявленная внутри класса. Как правило, поля объявляются с модификаторами доступа private либо protected, чтобы запретить прямой доступ к ним. Для получения доступа к полям следует использовать свойства или методы.

Пример объявления полей в классе:

```
class Student
{
  private string firstName;
  private string lastName;
  private int age;
```

```
  public string group; // не рекомендуется использовать public для поля
}
```

Создание объектов

Объявив класс, мы теперь можем создавать объекты. Делается это при помощи ключевого слова new и имени класса:

Student student1 = new Student(); //создание объекта student1 класса Student

Доступ к членам объекта осуществляется при помощи оператора точка «.» :

```
  student1.group = "Group1";
```

конструктор класса

Конструктор – это метод класса, предназначенный для инициализации объекта при его создании. Инициализация – это задание начальных параметров объектов/переменных при их создании.

Особенностью конструктора, как метода, является то, что его имя всегда совпадает с именем класса, в котором он объявляется. При этом, при объявлении конструктора, не нужно указывать возвращаемый тип, даже ключевое слово void. Конструктор следует объявлять как public, иначе объект нельзя будет создать (хотя иногда в этом также есть смысл).

В классе, в котором не объявлен ни один конструктор, существует неявный конструктор по умолчанию, который вызывается при создании объекта с помощью оператора new.

Объявление конструктора имеет следующую структуру:

```
public [имя_класса] ([аргументы])
{
  // тело конструктора
}
```

Конструктор также может иметь параметры.

Пример с тем же автомобилем, только теперь при создании объекта мы можем задать любые начальные значения:

```
class Car
{
  private double mileage;
  private double fuel;

  public Car(double mileage, double fuel)
  {
```

```
    this.mileage = mileage;
    this.fuel = fuel;
  }
}
```

Создание полей класса массивного типа и заполнение через класс

```
class Cl1
   {
      int[] ff;
      public Cl1 (int[] g)
      {
         ff = g;
      }
      public  int ss()
      {
         return ff[0] + ff[1] + ff[2];
     } }
```

Запускная программа

```
 static void Main(string[] args)

 {int[] pp = { 1, 2, 3 };
int[] pp1 = { 1, 2, 3, 4, 5 }
         Cl1 aa = new Cl1(pp);

         int q = aa.ss();
         Console.WriteLine(Convert.ToInt32(q));
         Console.ReadKey();
      }
```

Методы класса

Метод – это небольшая подпрограмма, которая выполняет, в идеале, только одну функцию. Методы позволяют сократить объем кода.

Методы вместе с полями, являются основными членами класса.

Статический метод – это метод, который не имеет доступа к полям объекта, и для вызова такого метода не нужно создавать экземпляр (объект) класса, в котором он объявлен.

Простой метод – это метод, который имеет доступ к данным объекта, и его вызов выполняется через объект.

Простые методы служат для обработки внутренних данных объекта.

Приведу пример использования простого метода. Класс Телевизор, у него есть поле switchedOn, которое отображает состояние включен/выключен, и два метода – включение и выключение:

```
class TVSet
{
```

```
  private bool switchedOn;

  public void SwitchOn()
  {
   switchedOn = true;
  }
  public void SwitchOff()
  {
   switchedOn = false;
  }
}
class Program
{
  static void Main(string[] args)
  {
   TVSet myTV = new TVSet();
   myTV.SwitchOn(); // включаем телевизор, switchedOn = true;
   myTV.SwitchOff(); // выключаем телевизор, switchedOn = false;
  }
}
```

Чтобы вызвать простой метод, перед его именем, указывается имя объекта. Для вызова статического метода необходимо указывать имя класса.

Статические методы, обычно, выполняют какую-нибудь глобальную, общую функцию, обрабатывают «внешние данные». Например, сортировка массива, обработка строки, возведение числа в степень и другое.

Пример статического метода, который обрезает строку до указанной длины, и добавляет многоточие:

```
class StringHelper
{
  public static string TrimIt(string s, int max)
  {
   if (s == null)
    return string.Empty;
   if (s.Length <= max)
    return s;

   return s.Substring(0, max) + "...";
  }
}
class Program
{
  static void Main(string[] args)
  {
   string s = "Очень длинная строка, которую необходимо обрезать до указанной длины и добавить многоточие";
```

```
    Console.WriteLine(StringHelper.TrimIt(s, 20)); //"Очень длинная строка…"
    Console.ReadLine();
  }
}
```

Статический метод не имеет доступа к нестатическим полям класса:

```
class SomeClass
{
  private int a;
  private static int b;

  public static void SomeMethod()
  {
    a=5; // ошибка
    b=10; // допустимо
  }
```

Контрольные вопросы

1. Что представляет собой класс?
2. Какие спецификации доступа используются при описании класса?
3. Что является элементами класса?
4. Как осуществляется доступ к элементам класса?
5. Для чего используется указатель this?
6. Что такое конструктор?
7. Что такое деструктор?

Варианты заданий

1. Определить класс-строку. В класс включить два конструктора: для определения класса строки строкой символов и путем копирования другой строки (объекта класса строки). Предусмотреть функции поиска слова в строке и добавления другой строки, начиная с позиции N
2. Определить класс-строку. В класс включить два конструктора: для определения класса строки строкой символов и путем копирования другой строки (объекта класса строки). Предусмотреть функции слияния двух строк и функцию подсчёта предложений в строке.
3. Определить класс-строку. В класс включить два конструктора: для определения класса строки строкой символов и путем копирования другой строки (объекта класса строки). Предусмотреть функции сортировки слов в строке по алфавиту и подсчёта количества слов.
4. Определить класс список элементов. В определение класса включить два конструктора: для определения списка по его размеру и путем копирования другого списка. Предусмотреть функции подсчёта количества элементов списка и добавления одного списка в другой список, начиная с позиции N.
5. Определить класс список элементов. В определение класса включить два конструктора для определения списка по его размеру и путем копирования другого списка. Предусмотреть функции сортировки списка по возрастанию и вывода на экран в обратном порядке.

6. Определить класс список элементов. В определение класса включить два конструктора для определения списка по его размеру и путем копирования другого списка. Предусмотреть функции инверсии списка (123->321) и поиска подсписка в списке.

7. Определить класс сортированный список элементов. В определение класса включить два конструктора для определения списка по его размеру и путем копирования другого списка. Предусмотреть функции добавления элемента и слияния двух сортированных списков.

8. Определить класс список элементов. В определение класса включить два конструктора для определения списка по его размеру и путем копирования другого списка. Предусмотреть функции формирования нового списка из элементов, входящих только в один из двух других списков и вычисления суммы элементов списков.

9. Определить класс квадратная матрицу. В класс включить два конструктора для определения матрицы по количеству элементов и путем копирования другой матрицы. Предусмотреть функции вычисления детерминанта матрицы и умножения матрицы на число.

10. Определить класс стек. В класс включить два конструктора для определения стека по его размеру и путем копирования другого стека. Предусмотреть функции вычисления среднего арифметического из элементов стека и нахождения элемента по его номеру

11. Определить класс вектор. В класс включить два конструктора для определения вектора по его размеру и путем копирования другого вектора. При задании вектора по его размеру предусмотреть его заполнение случайными числами. Предусмотреть функции умножения векторов и подсчёта суммы элементов вектора.

12. Определить класс вектор. В класс включить два конструктора для определения вектора по его размеру и путем копирования другого вектора. При задании вектора по его размеру предусмотреть его заполнение случайными числами. Предусмотреть функции нахождения максимального и минимального из элементов вектора и умножения вектора на число.

13. Определить класс вектор. В класс включить два конструктора для определения вектора по его размеру и путем копирования другого вектора. При задании вектора по его размеру предусмотреть его заполнение случайными числами. Предусмотреть функции сортировки вектора по возрастанию и нахождения среднего арифметического из элементов вектора.

14. Определить класс прямоугольная матрицу. В класс включить два конструктора для определения матрицы по количеству элементов и путем копирования другой матрицы. Предусмотреть функции вычисления произведения матрицы на матрицу и сложения матриц.

15. Определить класс «вектор координат в пространстве». В класс включить два конструктора для определения вектора по умолчанию и путем копирования другого вектора. При задании вектора по умолчанию предусмотреть его заполнение случайными числами. Предусмотреть функции нахождения длины вектора, умножения вектора на вектор, скалярного произведения векторов.

Лабораторные работы №2

Определение операций в классе. Создание наследованных классов

Цель работы: Изучить возможности программирования классов на языке C#; Получить основные навыки программирования наследованных классов.

Теория

Наследование классов — очень мощная возможность в объектно-ориентированном программировании. Оно позволяет создавать производные классы (классы наследники), взяв за основу все методы и элементы базового класса (класса родителя

Класс может наследовать от другого класса таким образом расширяя или модифицируя его. Наследование от класса позволяет переиспользовать его функциональность, вместо того чтоб писать ее с нуля. Класс может наследовать только от одного класса, но сам может быть наследован любым количеством классов, формируя таким образом классовую иерархию.

Пример:

```
public class Stock : Asset       // Наследуется от Asset
{
   public long SharesOwned;
}

public class House : Asset          // Наследуется от Asset
{
   public decimal Mortgage;
}

Stock msft = new Stock { Name="MSFT",   SharesOwned=1000 };
Console.WriteLine (msft.Name); // MSFT
Console.WriteLine (msft.SharesOwned); // 1000
House mansion = new House { Name="Mansion",   Mortgage=250000 };
```

В примере подклассы Stock и House наследуют свойство Name от базового класса Asset. Подклассы также называют **производными классами (derived classes)**.

Полиморфизм (Polymorphism)

Ссылки полиморфны. Это означает, что переменная типа *X* может ссылаться не только на объекты типа *X*, но и на объекты всех производных от *X* классов.

```
1 public static void Display (Asset asset)
2 {
3     System.Console.WriteLine (asset.Name);
4 }
```

Методу из примера выше можно передавать не только объекты типа Asset, но и объекты типов Stock и House (т.к. оба относятся к типу Asset). Полиморфизм основывается на базисе, что производные классы (Stock и House) обладают всеми возможностями своего базового класса (Asset). Но не наоборот: базовый класс не обладает всеми возможностями производных. Если пример выше переделать, чтобы метод принимал только тип House, передать ему объект типа Asset будет нельзя.

Задания по вариантам

Задание 1

Составить консольное приложение, которое содержит описание класса Time (время), который должен содержать:

Класс должен включать:

Закрытые поля для хранения часов, минут и секунд
Свойства для доступа к полям
Конструктор или несколько конструкторов, для создания объектов
Метод - показать на экране время в формате (чч:мм:сс)
Метод - рассчитать разницу в секундах с заданным временем

```
public int Razn(Time t) { … }
```

Что сделать:

1. В функции Main() нужно объявить массив из 3 объектов класса
2. Задать им следующие значения (2ч 30м 10с, 5ч 15м 25с, 3ч 45м 11с)
3. Вывести на экран время, хранящееся во всех объектах.
4. Рассчитать разницу в днях между 1 и 2 объектами и вывести ее на экран.

Задание 2

Составить консольную программу, которая содержит описание класса Date - дата (год, месяц, день)

Класс должен включать:

Закрытые поля для хранения года, месяца, дня
Свойства для доступа к полям
Конструктор или несколько конструкторов, для создания объектов
Метод - показать на экране время в формате (дд/мм/гг)
Метод - рассчитать количество дней с начала года до даты
public int Days( )

Что сделать:

1. В функции Main() нужно объявит массив из 3 объектов класса

2. Задать им следующие значения (1.5.27.2001)
3. Вывести на экран даты, хранящиеся во всех объектах.
4. Рассчитать разницу в днях между 1 и 3 объектами и вывести ее на экран.

Задание 3

Составить консольную программу, которая содержит описание класса Box - ящик (высота, ширина, длина)

Класс должен включать:

Закрытые поля для хранения данных о высоте, ширине, длине
Конструктор или несколько конструкторов, для создания объектов
Метод - показать на экране данные о ящике
Метод - рассчитать объем коробки
public float Volume( )

Что сделать:
1. В функции Main() нужно объявит массив из 3 объектов класса
2. Задать им следующие значениями высоты, ширины и длины ((2,3,5), (6,7,2), (3,5,4))
3. Вывести на экран данные, хранящиеся во всех объектах.
4. Рассчитать в цикле суммарный объем коробок и вывести его на экран

Задание 4

Составить консольную программу, которая содержит описание класса Fraction – дробь (числитель, знаменатель)

Класс должен включать:

Закрытые поля для хранения данных о числителе и знаменателе
Конструктор или несколько конструкторов, для создания объектов
Метод - показать на экране данные о дроби в формате числитель/знаменатель
Метод – сложение с другим объектом данного класса
public float Sum(Fraction b) { … }

Что сделать:
1. В функции main() нужно объявит массив из 3 объектов класса
2. Задать им следующие значения - (1/2), (1/3), (2/3)
3. Вывести на экран данные, хранящиеся во всех объектах.
4. Рассчитать сумму всех дробей, используя метод Sum, вывести результат на экран

Задание 5

Составить программу, которая содержит описание класса Complex – комплексные числа (реальная часть, мнимая часть)

Класс должен включать:

Закрытые поля для хранения данных о реальной часть и мнимой части
Конструктор или несколько конструкторов, для создания объектов
Метод - показать на экране данные о комплексном числе в формате (r + i m) (r - реальная часть, m – мнимая часть)

Метод – сложение с другим объектом данного класса

public float Sum(Complex b) {…}

Что сделать:
1. В функции Main() нужно объявит массив из 3 объектов класса.
2. Задать им следующие значения (высота, ширина, длина+ i2), (2 + i3), (3 + i4)
3. Вывести на экран данные, хранящиеся во всех объектах.
4. Рассчитать в цикле сумму этих комплексных чисел и вывести ее на экран

Задание 6

Составить консольное приложение, содержащее описание класса Student (студент), который должен включать:

Закрытые поля для хранения имени и отметки
Свойства для доступа к закрытым полям
Конструктор или несколько конструкторов, для создания объектов
Метод - показать на экране данные о студенте

Что сделать:
1. В функции main() нужно объявит коллекцию
2. Создать 3 объекта класса и добавить их в коллекцию
3. Задать объектам следующие значения - ((Иванов, 3), (Сидоров, 4), (Петров, 5))
4. Вывести на экран данные, хранящиеся во всех объектах.
5. Рассчитать в цикле среднюю оценку студентов и вывести ее на экран

Задание 7

Составить консольное приложение, которое содержит описание класса Employee (служащий), который должен содержать:

Закрытые поля для хранения имени и зарплаты
Свойства для доступа к закрытым полям
Конструктор или несколько конструкторов, для создания объектов
Метод - показать на экране данные о служащем

Что сделать:
1. В функции Main() нужно объявит коллекцию
2. Создать 3 объекта класса и добавить их в коллекцию
3. Задать объектам следующие значения - ((Иванов, 6350), (Сидоров, 10350), (Петров, 56350))
4. Вывести на экран данные, хранящиеся во всех объектах.
5. Рассчитать в цикле среднюю зарплату служащих и вывести ее на экран

Задание 8

Составить консольное приложение, которое содержит описание класса Car (автомобиль), который должен содержать:

Закрытые поля для хранения марки, гос. номер и пробега
Свойства для доступа к закрытым полям
Конструктор или несколько конструкторов, для создания объектов
Метод - показать на экране данные об автомобиле

Что сделать:
1. В функции Main() нужно объявит коллекцию
2. Создать 3 объекта класса и добавить их в коллекцию
3. Задать объектам следующие значения - ((Волга, Н180ТО, 6350), (Жигули, Т144ТО, 10350), ((Нива, С234ТО, 56350))
4. Вывести на экран данные, хранящиеся во всех объектах.
5. Рассчитать в цикле суммарный пробег всех автомобилей и вывести его на экран

Задание 9

Составить консольное приложение, которое содержит описание класса Computer (вычислительная машина), который должен содержать:

Закрытые поля для хранения - инвентарного номера, частоты процессора и объема жесткого диска
Свойства для доступа к закрытым полям
Конструктор или несколько конструкторов, для создания объектов
Метод - показать на экране данные о компьютере

Что сделать:
1. В функции Main() нужно объявит коллекцию
2. Создать 3 объекта класса и добавить их в коллекцию
3. Задать объектам следующие значения - ((1 500 , 40 000 , (1 200 , 10 000 , ((850 , 1 500
4. Вывести на экран данные, хранящиеся во всех объектах.
5. Рассчитать в цикле суммарный объем жестких дисков всех компьютеров и вывести его на экран

Задание 10

Составить консольное приложение, которое содержит описание класса Rectangle (прямоугольник), который должен содержать:

Класс должен включать:

Закрытые поля для хранения высоты и ширины
Свойства для доступа к полям
Конструктор или несколько конструкторов, для создания объектов
Метод - показать на экране размер прямоугольника в формате “(высота : ширина)”
Метод - рассчитать площадь прямоугольника

public float Area ( ) { … }

Что сделать:
1. В функции Main() нужно объявит массив из 3 объектов класса
2. Создать 3 объекта класса и добавить их в коллекцию
3. Задать им следующие значения (5,4), (3.5, 4, 4.8)
4. Вывести на экран размеры всех прямоугольников.
5. Рассчитать суммарную площадь всех прямоугольников.

Задание 11

Составить программу, которая содержит описание класса Complex – комплексные числа (реальная часть, мнимая часть)

Класс должен включать:
Закрытые поля для хранения данных о реальной часть и мнимой части
Конструктор или несколько конструкторов, для создания объектов
Метод - показать на экране данные о комплексном числе в формате (r + i m) (r - реальная часть, m – мнимая часть)
Метод – получение модуля комплексного числа

public float Modul ( ) { … }

Что сделать:

1. В функции main() нужно объявит коллекцию
2. Создать 3 объекта класса и добавить их в коллекцию
3. Задать им следующие значения - (1 + i2), (2 + i3), (3 + i4)
4. Вывести на экран данные, хранящиеся во всех объектах.
5. Рассчитать в цикле сумму модулей этих комплексных чисел и вывести ее на экран

Примечание: модуль комплексного числа (a+ib) равен Math. sqrt (a*a + b*b)

Задание 12

Составить консольную программу, которая содержит описание класса Date - дата (год, месяц, день)

Класс должен включать:

Закрытые поля для хранения год, месяц, день
Свойства для доступа к полям
Конструктор или несколько конструкторов, для создания объектов
Метод - показать на экране время в формате (дд/мм/гг)
Метод - рассчитать количество дней оставшихся от даты до конца года

Public int Days( ) { … }

Что сделать:

1. В функции Main() нужно объявит коллекцию
2. Создать 3 объекта класса и добавить их в коллекцию
3. Задать им следующие значения (1.5.27.2002)
4. Вывести на экран даты, хранящиеся во всех объектах.
5. Рассчитать разницу в днях между 1 и 2 объектами и вывести ее на экран.

Лабораторные работы №3

Использование стандартных интерфейсов.

Цель работы: разработать приложение исходя из варианта задания, приобрести и развить навыки по работе с интерфейсами и коллекциями в C#

Теория

Интерфейсы

Синтаксис интерфейса

Интерфейс является «крайним случаем» абстрактного класса. В нем задается набор абстрактных методов, свойств и индексаторов, которые должны быть реализованы в производных классах. Иными словами, интерфейс определяет поведение, которое поддерживается реализующими этот интерфейс классами. Основная идея использования интерфейса состоит в том, чтобы к объектам таких классов можно было обращаться одинаковым образом.

Каждый класс может определять элементы интерфейса по-своему. Так достигается полиморфизм: объекты разных классов по-разному реагируют на вызовы одного и того же метода.

Синтаксис интерфейса аналогичен синтаксису класса:

[атрибуты] [спецификаторы] interface имя_интерфейса [: предки] тело_интерфейса [;]

Для интерфейса могут быть указаны спецификаторы new, public, protected, internal и private. Спецификатор new применяется для вложенных интерфейсов и имеет такой же смысл, как и соответствующий модификатор метода класса. Остальные спецификаторы управляют видимостью интерфейса. По умолчанию интерфейс доступен только из сборки, в которой он описан (internal).

Интерфейс может наследовать свойства нескольких интерфейсов, в этом случае *предки*перечисляются через запятую. *Тело интерфейса* составляют абстрактные методы, шаблоны свойств и индексаторов, а также события.

В качестве примера рассмотрим интерфейс IAction, определяющий базовое поведение персонажей компьютерной игры, встречавшихся в предыдущих работах. Допустим, что любой персонаж должен уметь выводить себя на экран, атаковать и красиво умирать:

```
interface IAction

{

void Draw();
int Attack(int a);
void Die();
int Power { get; }

}
```

В интерфейсе IAction заданы заголовки трех методов и шаблон свойства Power, доступного только для чтения. Если бы требовалось обеспечить возможность установки свойства, в шаблоне следовало указать ключевое слово set, например:

```
int Power { get; set; }
```

Отличия интерфейса от абстрактного класса:

· элементы интерфейса по умолчанию имеют спецификатор доступа public и не могут иметь спецификаторов, заданных явным образом;

· интерфейс не может содержать полей и обычных методов – все элементы интерфейса должны быть абстрактными;

· класс, в списке предков которого задается интерфейс, должен определять *все* его элементы, в то время как потомок абстрактного класса может не переопределять часть абстрактных методов предка (в этом случае производный класс также будет абстрактным);

· класс может иметь в списке предков несколько интерфейсов, при этом он должен определять все их методы.

Реализация интерфейса

В списке предков класса сначала указывается его базовый класс, если он есть, а затем через запятую интерфейсы, которые реализует этот класс. Например, реализация интерфейса IAction в классе Monster может выглядеть следующим образом:

```
using System;

namespace ConsoleApplication1

{

interface IAction

{

void Draw();

int Attack( int a );

void Die();

int Power { get; }

}


class Monster : IAction

{

public void Draw()

{

Console.WriteLine( "Здесь был " + name );
```

```
  }
  public int Attack( int ammo_ )
  {
  ammo -= ammo_;
  if ( ammo > 0 ) Console.WriteLine( "Ба-бах!" );
  else ammo = 0;
  return ammo;
  }

  public void Die()
  {
  Console.WriteLine( "Monster " + name + " RIP" );
  health = 0;
  }

  public int Power
  {
  get
  {
  return ammo * health;
  }}
  …
}
```

Сигнатуры методов в интерфейсе и реализации должны полностью совпадать. Для реализуемых элементов интерфейса в классе следует указывать спецификатор public. К этим элементам можно обращаться как через объект класса, так и через объект типа соответствующего интерфейса:

```
Monster Vasia = new Monster( 50, 50, "Вася" ); // объект класса Monster
Vasia.Draw(); // результат: Здесь был Вася
IAction Actor = new Monster( 10, 10, "Маша" ); // объект типа интерфейса
Actor.Draw(); // результат: Здесь был Маша
```

Существует второй способ реализации интерфейса в классе: *явное указание имени интерфейса*перед реализуемым элементом. Спецификаторы доступа при этом не указываются. К таким элементам можно обращаться в программе *только через объект типа интерфейса*, например:

```
class Monster : IAction
{
```

```
int IAction.Power
{
get
{
return ammo * health;
}}
void IAction.Draw()
{
Console.WriteLine( "Здесь был " + name );
}
...
}
...
IAction Actor = new Monster( 10, 10, "Маша" );
Actor.Draw(); // обращение через объект типа интерфейса

// Monster Vasia = new Monster( 50, 50, "Вася" );
// Vasia.Draw(); ошибка!
```

Таким образом, при явном задании имени реализуемого интерфейса соответствующий метод *не входит в интерфейс класса*. Это позволяет упростить его в том случае, если какие-то элементы интерфейса не требуются конечному пользователю класса.

Работа с объектами через интерфейсы. Операции is и as

При работе с объектом через объект типа интерфейса бывает необходимо убедиться, что объект поддерживает данный интерфейс. Проверка выполняется с помощью бинарной операции is. Эта операция определяет, совместим ли текущий тип объекта, находящегося слева от ключевого слова is, с типом, заданным справа.

Результат операции равен true, если объект можно преобразовать к заданному типу, и false в противном случае. Операция обычно используется в следующем контексте:

```
if ( объект is тип )
{
// выполнить преобразование "объекта" к "типу"
// выполнить действия с преобразованным объектом
}
```

Допустим, мы оформили какие-то действия с объектами в виде метода с параметром типа object. Прежде чем использовать этот параметр внутри метода для обращения к методам, описанным в производных классах, требуется выполнить преобразование к

производному классу. Для безопасного преобразования следует проверить, возможно ли оно, например так:

```
static void Act( object A )
{
if ( A is IAction )
{
IAction Actor = (IAction) A;
Actor.Draw();
}}
```

В метод Act можно передавать любые объекты, но на экран будут выведены только те, которые поддерживают интерфейс IAction.

Недостатком использования операции is является то, что преобразование фактически выполняется дважды: при проверке и при собственно преобразовании. Более эффективной является другая операція – as. Она выполняет преобразование к заданному типу, а если это невозможно, формирует результат null, например:

```
static void Act( object A )
{
IAction Actor = A as IAction;
if ( Actor != null ) Actor.Draw();
}
```

Обе рассмотренные операции применяются как к интерфейсам, так и к классам.

Задание:

1. Разработать приложение исходя их следующих требований:

1.1. Приложение должно состоять минимум из двух форм. Первая – для работы со списком объектов, вторая – для работы с содержимым самого объекта

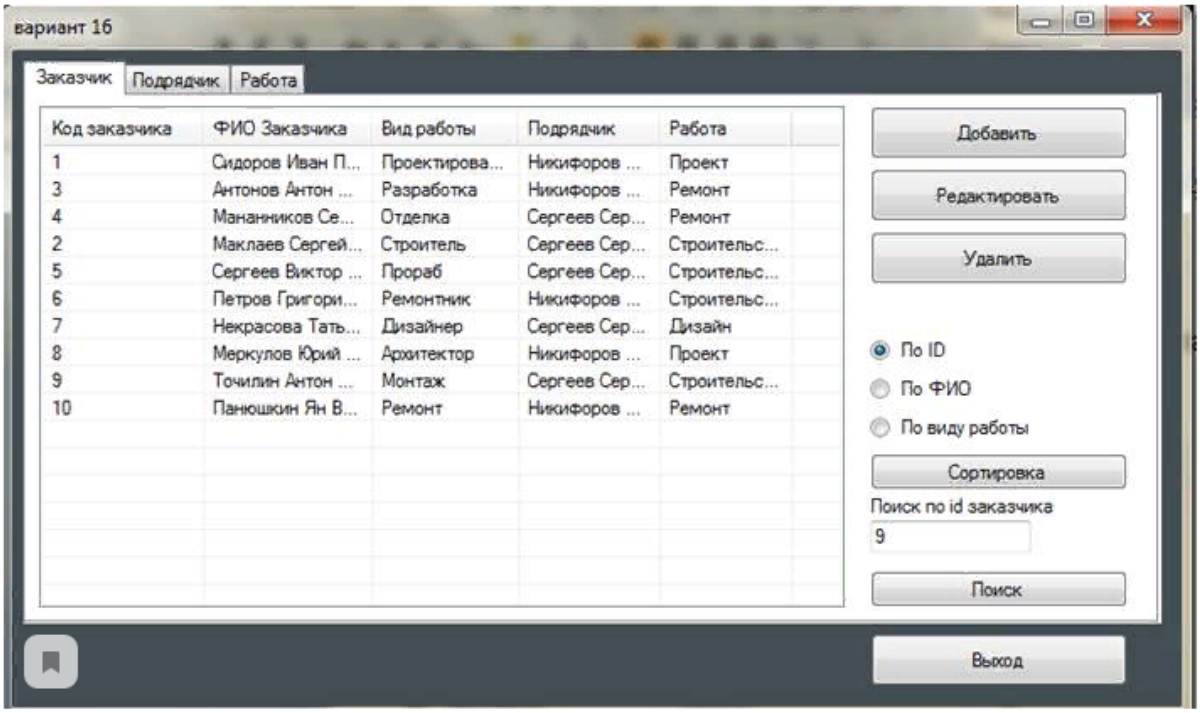
1.2. На форме должны быть предусмотрены операции “Добавления”, “Удаления”, “Изменения”, “Сортировки” и “Поиска” элементов списка.

1.3. Операции сортировки должны выполняться с использованием метода Sort() шаблонного класса List<T>. Количество вариантов сортировки – не менее 3.

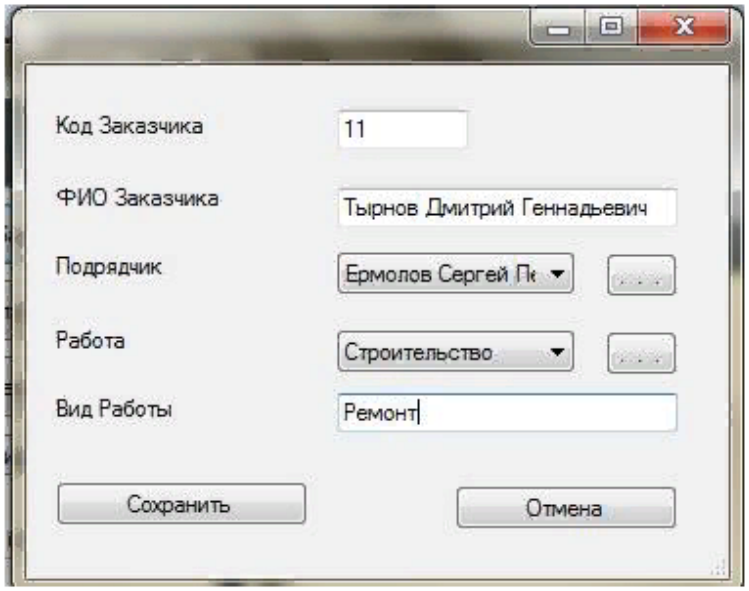
1.4. Операция поиска выполняется только по ключевому атрибуту, с использование возможностей класса Dictionary<T, T>.

Порядок выполнения работы

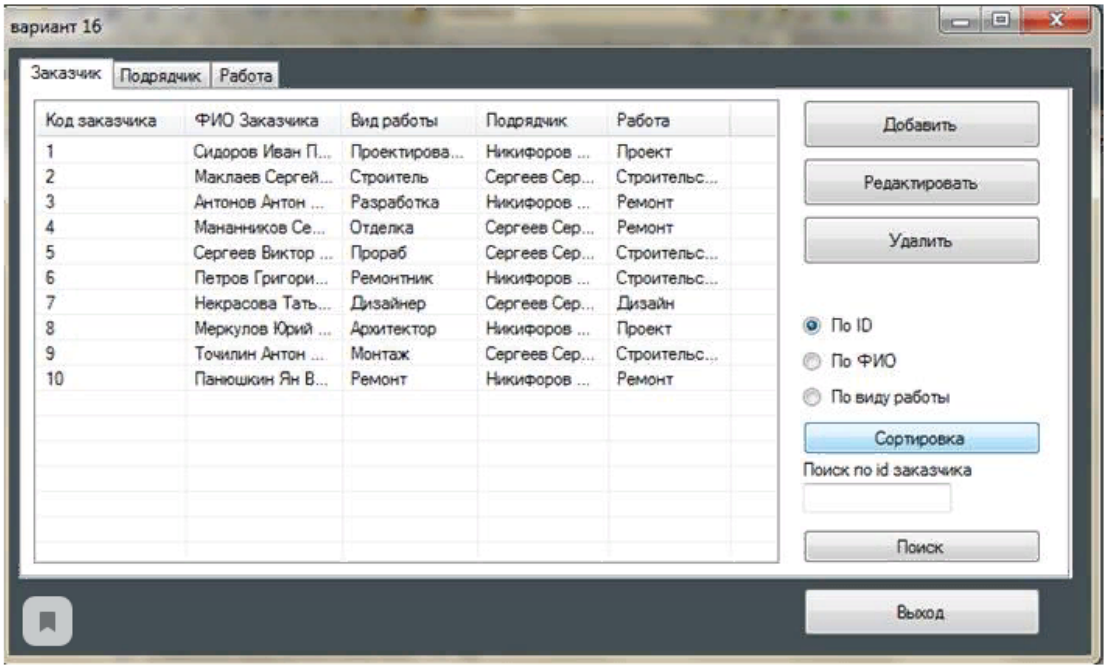
Форма для работы со списком объектов показана ниже:
На форме предусмотрены операции “Добавления”, “Удаления”, “Редактирования”, “Сортировки” и “Поиска” элементов списка.



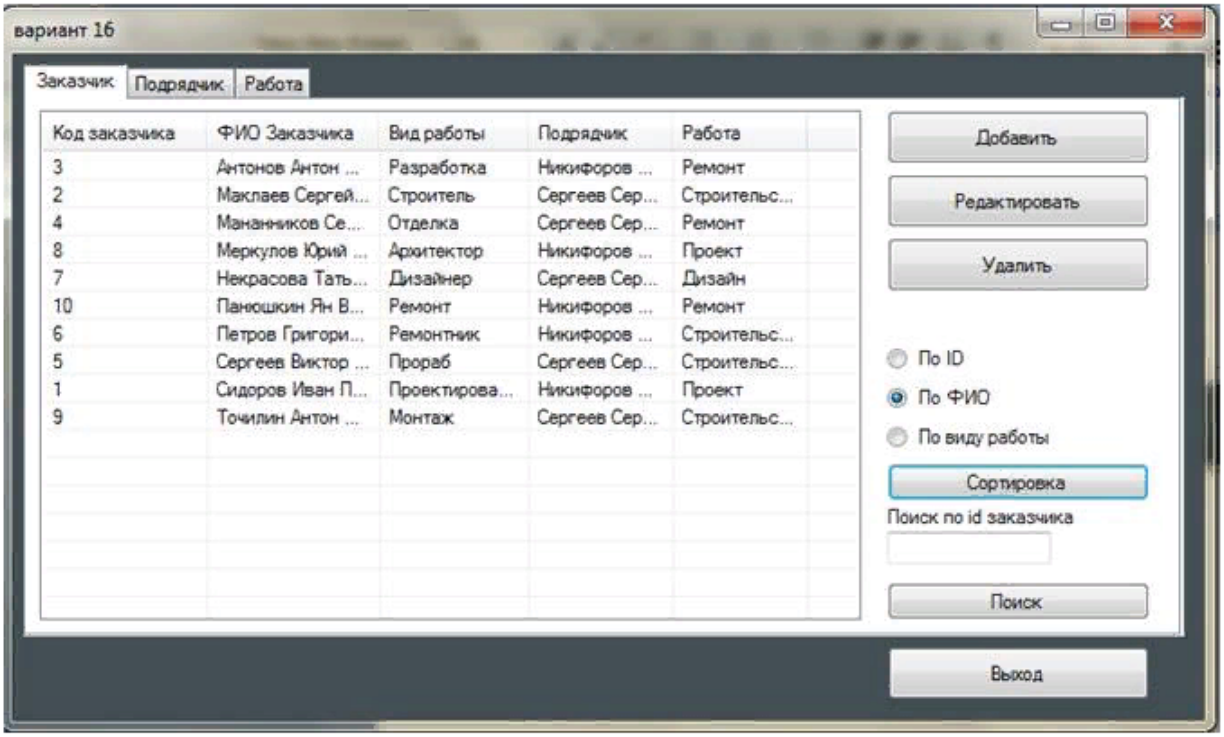
Форма для работы с содержимым самого объекта выглядит так:



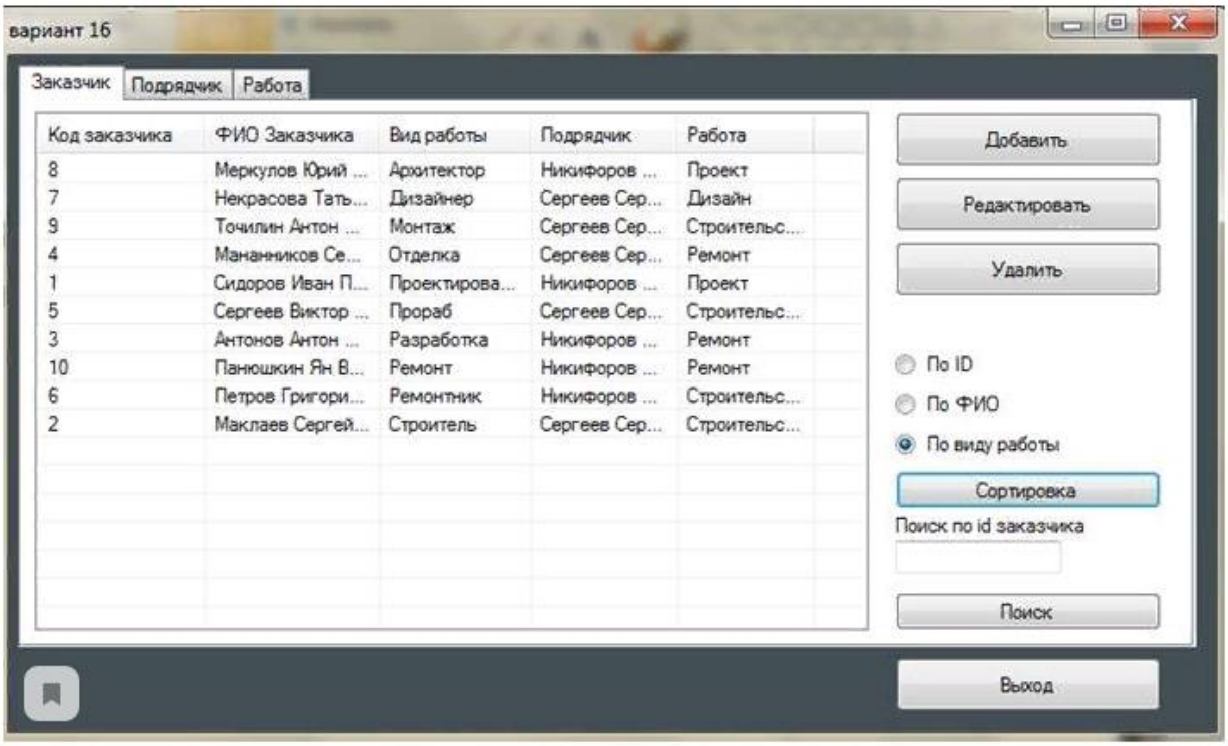
Сортировка может осуществляться 3 способами
По ID заказчика:



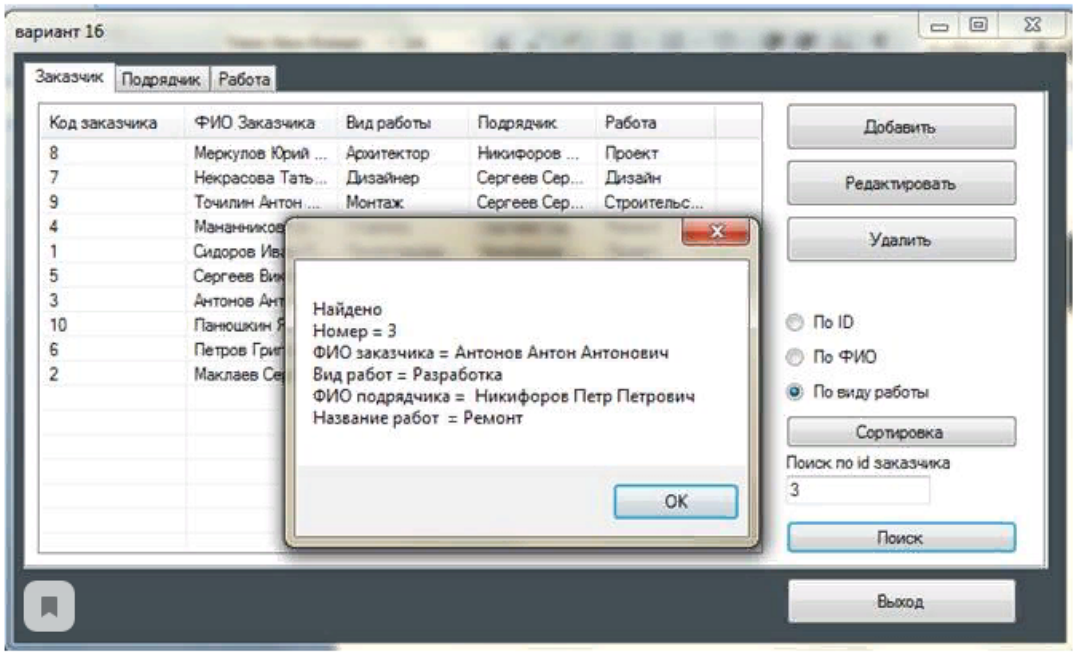
По ФИО заказчика (в алфавитном порядке):



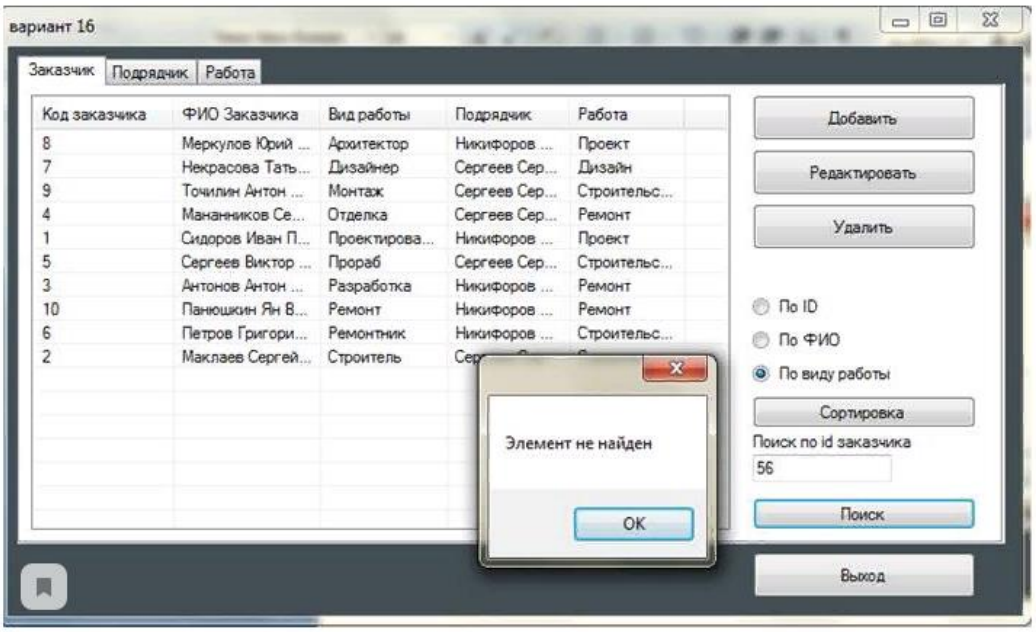
По виду работы (в алфавитном порядке):



Поиск происходит по ключевому полю:



Операция сортировки выполняется с помощью использования метода Sort() шаблонного класса List<T>.



Лабораторные работы №4

Использование стандартных интерфейсов.

Цель работы: Изучить возможности использование стандартных интерфейсов для реализации классов на языке C#. Изучить стандартные виды интерфейсов.

Теория

В библиотеке классов .NET определено множество стандартных интерфейсов, задающих желаемое поведение объектов. Например, интерфейс IComparable задает метод сравнения объектов на больше-меньше, что позволяет выполнять их сортировку. Реализация интерфейсов IEnumerable и IEnumerator дает возможность просматривать содержимое объекта с помощью конструкции foreach, а реализация интерфейса *ICloneable* — клонировать объекты.

Стандартные интерфейсы поддерживаются многими стандартными классами библиотеки. Например, работа с массивами с помощью цикла foreach возможна именно потому, что тип Array реализует интерфейсы IEnumerable и IEnumerator. Можно создавать и собственные классы, поддерживающие стандартные интерфейсы, что позволит использовать объекты этих классов стандартными способами.

Сравнение объектов

Интерфейс IComparable определен в пространстве имен System. Он содержит всего один метод CompareTo, возвращающий результат сравнения двух объектов — текущего и переданного ему в качестве параметра:

```
interface IComparable
{
   int CompareTo( object obj )
}
```

Метод должен возвращать:

- 0, если текущий объект и параметр равны;
- отрицательное число, если текущий объект меньше параметра;
- положительное число, если текущий объект больше параметра.

Реализуем интерфейс IComparable в знакомом нам классе Monster. В качестве критерия сравнения объектов выберем поле health. В листинге 9.1 приведена программа, сортирующая массив монстров по возрастанию величины, характеризующей их здоровье.

```
using System;
namespace ConsoleApplication1
{
   class Monster : IComparable
   {
      public Monster( int health, int ammo, string name )
      {
         this.health = health;
         this.ammo   = ammo;
         this.name   = name;
      }

      virtual public void Passport()
      {
         Console.WriteLine( "Monster {0} \t health = {1} ammo = {2}",
                        name, health, ammo );
      }

      public int CompareTo( object obj )          // реализация интерфейса
      {
         Monster temp = (Monster) obj;
         if ( this.health > temp.health ) return  1;
         if ( this.health < temp.health ) return -1;
         return 0;
      }

      string name;
      int health, ammo;
   }

   class Class1
   {   static void Main()
      {
         const int n = 3;
         Monster[] stado = new Monster[n];

         stado[0] = new Monster( 50, 50, "Вася" );
         stado[1] = new Monster( 80, 80, "Петя" );
         stado[2] = new Monster( 40, 10, "Маша" );

         Array.Sort( stado );              // сортировка стала возможной
         foreach ( Monster elem in stado ) elem.Passport();
      }
   }
}
```

Листинг 9.1. Пример реализации интерфейса IComparable

Результат работы программы:

Monster Маша health = 40 ammo = 10
Monster Вася health = 50 ammo = 50
Monster Петя health = 80 ammo = 80

Упражнение 1. Создание и реализация интерфейса

В этом упражнении Вы создадите интерфейс, определяющий поведение классов, которые будут его реализовывать.

Предполагается, что в библиотечной системе, разработанной в прошлой работе есть необходимость реализовать возможность оформления подписки на периодические издания. Включение этой функциональности в базовый класс **Item** не является правильным решением, так как к изданиям, оформляющим подписку не относятся, например книги. Поэтому необходимо создать интерфейс, объявляющий возможность оформления подписки и тогда классы, для которых предполагается данная функциональность должны будут реализовывать этот интерфейс.

Выполните подготовительные операции

Создайте папку Lab07 и скопируйте в нее решение MyClass, созданное в прошлом упражнении.

Создайте интерфейс IPubs с требуемой функциональностью

Откройте проект MyClass.sln в папке *install folder*\Labs\Lab07\.
Добавьте в проект новый интерфейс с именем **IPubs**: **Projects** (Проект) →**Add class** (Добавить класс). В окне **Добавление нового элемента** выберите **Интерфейс** и укажите его имя **IPubs**.
В интерфейсе **IPubs** объявите его функциональные члены – метод для проверки оформлена ли подписка на издание **Subs** и свойство **IfSubs** для
оформления подписки:interface IPubs

{
void Subs();
bool IfSubs { get; set;}
}

Реализуйте интерфейс в классе Magazine

Откройте класс **Magazine** и добавьте интерфейс в список наследования:class Magazine : Item,
IPubs
{
Реализуйте свойство и метод, объявленные в интерфейсе:public bool IfSubs { get; set; }

public void Subs()
{
Console.WriteLine("Подписка на журнал \"{0}\": {1}." , title, IfSubs);

}

Протестируйте новую функциональность

В методе **Main** класса **Program** добавьте для уже имеющегося журнала **mag1** установку свойству **IfSubs** значения, устанавливающую подписку и
вызовите метод **Subs** для отображения информации о подписке:

```
Magazine mag1 = new Magazine("О природе", 5,"Земля и мы", 2014, 1235, true);
mag1.TakeItem();
mag1.Show();
```

mag1.IfSubs = true; mag1.Subs();

Постройте проект и исправьте ошибки, если это необходимо. Запустите и протестируйте программу.

Упражнение 2. Использование стандартных интерфейсов

В библиотеке классов .Net определено множество стандартных интерфейсов, задающих желаемую функциональность объектов. В этом упражнении Вы примените интерфейс **IComparable,** который задает метод сравнения объектов по принципу больше и меньше, что позволяет переопределить соответствующие операции в рамках класса, наследующего интерфейс **IComparable**.
Сравнение и дальнейшая сортировка будет реализована по полю **invNumber** – *Инвентарный номер.*

Реализуйте наследование интерфейса IComparable

Добавьте в объявление абстрактного класса **Item** наследование интерфейса **IComparable:**

```
abstract class Item : IComparable
{
. . .
```

Интерфейс **IComparable** определен в пространстве имен **System** и содержит единственный метод **CompareTo**, возвращающий результат сравнения двух объектов – текущего и переданного ему в качестве параметра. Реализация данного метода должна возвращать: 0 – если текущий объект и параметр равны, отрицательное число, если текущий объект меньше параметра и положительное число, если текущий объект больше параметра.
Добавьте в класс **Item** реализацию этого метода, причем сравнение реализуйте по полю **invNumber:**

```
int IComparable.CompareTo(object obj)
{
Item it = (Item)obj;
if (this.invNumber == it.invNumber) return 0; else if (this.invNumber >
it.invNumber) return 1; else return -1;
}
```

Протестируйте использование новой функциональности

В методе **Main** класса **Program** создайте массив ссылок на абстрактный базовый класс **Item:**

Item[] itmas = new Item[4];

Заполните массив созданными ранее книгами и журналом:itmas[0] = b1;

itmas[1] = b2; itmas[2] = b3; itmas[3] = mag1;

Отсортируйте массив с помощью статического метода **Sort** класса **Array**:

Array.Sort(itmas);

Отобразите весь список книг и журналов, используя полиморфный вызов метода **Show**:

```
Console.WriteLine("\nСортировка по инвентарному номеру"); foreach (Item x in itmas)
{
x.Show();
}
```

Постройте проект и исправьте ошибки, если это необходимо. Запустите и протестируйте программу. Информация о каждом элементе хранения должна выводиться согласно возрастанию инвентарных номеров.

Упражнение 3. Создание иерархии классов «Фигуры»

В этом упражнении требуется создать иерархию классов – геометрических фигур: треугольник, окружность и квадрат, которые являются производными классами общего класса **Shape**.

Класс треугольник реализован в упражнении 3 лабораторной работы 5. Классы окружность и квадрат определяются соответственно радиусом и стороной.

Реализуйте конструкторы, создающие объекты с заданным радиусом и стороной и методы, позволяющие:

вывести длины радиуса окружности и стороны квадрата на экран;

рассчитать периметр и площадь фигур.

Общую функциональность фигур реализуйте в базовом классе **Shape**. Реализуйте дополнительную функциональность для треугольника и квадрата – вращение фигуры вокруг своего центра. Метод вращения представьте в интерфейсе. Классы треугольник и квадрат должны реализовывать этот интерфейс.

Лабораторные работы №5

Работа с типом данных структура.

Цель работы: Научиться создавать типы данных структуры и перечисление. Изучить методы работы со структурами и перечислениями в программе на C#.

Теория

Структура – это более простая версия классов. Все структуры наследуются от базового класса System. ValueType и являются типами значений, тогда как классы - ссылочные типы.

Структуры отличаются от классов следующими вещами:

- Структура не может иметь конструктора без параметров (конструктора по умолчанию);
- Поля структуры нельзя инициализировать, кроме случаев, когда поля статические.
private int x = 0; // в структуре недопустимо;
- Экземпляры структуры можно создавать без ключевого слова new;
- Структуры не могут наследоваться от других структур или классов. Классы не могут наследоваться от структур. Структуры могут реализовывать интерфейсы;
- Так как структуры это типы значений, они обладают всеми свойствами подобных типов (передача в метод по значению и т.д.), в отличии от ссылочных типов;
- Структура может быть nullable типом.

Структуры объявляются при помощи ключевого слова struct:

```
public struct Book
{
  public string Name;
  public string Year;
  public string Author;
}
```

Экземпляр структуры можно создавать без ключевого слова new:

```
static void Main(string[] args)

{
 Book b;
 b. Name = "BookName";
}
```

Структуры подходят для создания несложных типов, таких как точка, цвет, окружность. Если необходимо создать множество экземпляров подобного типа, используя структуры, мы экономим память, которая могла бы выделяться под ссылки в случае с классами.

Примерами структур в стандартной библиотеке классов .Net являются такие типы как int, float, double, bool и другие. Также DateTime, Point (точка), Color.

Задачи

1. Описать структуру с именем WORKED 1.фамилии и инициалы работника 2.название занимаемой должности 3.год поступление на работу написать программу выполняющие след.действие: 1.ввод с клавиатуры данных в массив,состоящий из 10 структур типа WORKED(записи должны быть упорядоченны по алфавиту) 2.вывод на экран фамилии работников,стаж работы которых превышает значение,введённое с клавиатуры(если нет,то вывести сообщение об этом) мне нужна помощь,я сделала вот это,не могу сообразить как вывести по алфавиту,и как 2.вывод на экран фамилии работников,стаж работы которых превышает значение,введённое с клавиатуры(если нет,то вывести сообщение об этом)

2. Создайте программу, которая будет находить окружность (структура) у которой радиус максимально близкий к среднему значению радиусов окружностей из списка.

3. Создать список из заданного количества элементов. Выполнить циклический сдвиг этого списка на N элементов вправо или влево.

4. Реализовать с помощью однонаправленного линейного списка операцию умножения длинного целого числа на N (целую цифру, то есть целое число от 0 до 9).

5. Исходный файл содержит два набора положительных значений, между наборами стоит отрицательное значение. Построить два списка C1 и C2, элементы которых содержат значение из наборов 1 и 2 соответственно. Элементы расположить по возрастанию. Выполнить соединение списков

6. Создать два стека и поменять информацию местами.

7. Исходный файл содержит наименование некоторых объектов. Построить список, элементы которого содержат наименование и шифры данных объектов, причем элементы списка должны быть упорядочены по возрастанию значений шифров. Выполнить «сжатие» списка, удалив продублированные наименования объектов.

8. Задано множество точек на двухмерной плоскости. Найти три разные точки, которые составляют треугольник наибольшего периметра. Структуру данных "множество" реализовать в виде класса.

9. Задано множество точек на двухмерной плоскости. Найти такую точку, от которой сумма расстояние к другим точкам наименьшая. Структуру данных "множество" реализовать в виде класса.

10. На плоскости заданы N множеств по M точек в каждой. Найти среди точек первого множества такую точку, которая принадлежит наибольшему количеству множеств. Структуру данных "множество" реализовать в виде класса.

11. Создать список из стихов одного автора. Выполнить сортировку списка по возрастанию размера стихов.

12. Ввести некоторое число и записать его цифры в стек. Вывести число, у которого цифры идут в обратном порядке. Предусмотреть возможность введения произвольного количества чисел.

13. Система состоит из трех процессоров P1, P2, P3, очереди F, стека S и распределителя задач R. В систему поступают запросы на выполнение задач трёх типов – T1, T2 и T3, каждая для своего процессора.Поступающие запросы ставятся в очередь. Если в начале очереди находится задача Ti и процессор Pi свободен, то распределитель R ставит задачу на выполнение в процессор Pi, а если процессор Pi занят, то распределитель R отправляет задачу в стек и из очереди извлекается следующая задача. Если в вершине стека находится задача, процессор которой в данный момент свободен, то эта задача извлекается и отправляется на выполнение. Осуществить моделирование работы данной системы.

14. Задано два списка, которые содержат результаты N-измерений тока (I) и напряжения (U) на неизвестном сопротивлении (R). Найти приближенное значение R методом наименьших квадратов.

15. Организовать хеш-таблицу с открытой адресацией, используя процедуру поиска и вставки по ключу. Для формирования начального хеш-адреса использовать метод деления, а затем при возникновении коллизии процедуру квадратичного исследования.

16. Написать программу, определяющую является ли двоичное дерево деревом поиска.

Лабораторные работы №6

Коллекции. Параметризованные классы.

Цель работы: Получить практические навыки работы со стандартными коллекциями пространства имен System.Collection.Generic и с нестандартными исключениями, создаваемыми программистами.

Теоретические сведения

Параметризированные коллекции.

Во вторую версию библиотеки .NET добавлены параметризованные коллекции для представления основных структур данных, применяющихся при создании программ, — стека, очереди, списка, словаря и т. д. Эти коллекции, расположенные в пространстве имен **System.Collections.Generic**, дублируют аналогичные коллекции пространства имен **System.Collections.**

Параметризованные коллекции библиотеки .NET

Класс- прототип	Обычный класс
Comparer <T>	Comparer
Dictionary <K,T>	HashTable
LinkedList <T>	-
List <T>	ArrayList
Queue<T>	Queue
SortedDictionary <K,T>	SortedList
Stack <T>	Stack

Параметром класса-прототипа является тип данных, с которым он работает (T – это тип, который является параметром коллекции, т. е. вместо него можно подставить любой другой тип данных).

В коллекциях, которые мы рассматривали ранее, хранятся ссылки на объекты типа object. Когда значение структурного типа приводится к типу object (или к типу того интерфейса, который реализован системой) выполняется операция упаковки – явного преобразования из типа значений в тип ссылок. Эта операция выполняется автоматически и не требует вмешательства программиста. Обратной операцией является распаковка – значение объекта присваивается переменной.

У таких коллекций есть два недостатка:

- в одной и той же коллекции можно хранить элементы любого типа, следовательно, ошибки при помещении в коллекцию невозможно проконтролировать на этапе компиляции, а при извлечении элемента требуется его явное преобразование;
- при хранении в коллекции элементов значимых типов выполняется большой объем действий по упаковке и распаковке элементов, что в значительной степени снижает эффективность работы.

Коллекция StudentGroup содержит объекты пользовательских классов иерархии классов Person->Student.

В коллекции, для которой объявлен тип элементов Person, благодаря полиморфизму можно хранить элементы любого производного класса, но не элементы других типов.

Казалось бы, по сравнению с обычными коллекциями это ограничение, а не универсальность, однако на практике коллекции, в которых действительно требуется хранить значения различных, не связанных межу собой типов, почти не используются.

Достоинством же такого ограничения является то, что компилятор может выполнить контроль типов во время компиляции, а не выполнения программы, что повышает ее надежность и упрощает поиск ошибок.

Коллекция Vector состоит из целых чисел, причем для работы с ними не требуются ни операции упаковки и распаковки, ни явные преобразования типа при получении элемента из коллекции, как это было в обычных коллекциях.

Классы-прототипы называют также родовыми или шаблонными, поскольку они представляют собой образцы, по которым во время выполнения программы строятся конкретные классы. Использование стандартных параметризованных коллекций для хранения и обработки данных является хорошим стилем программирования, поскольку позволяет сократить сроки разработки программ и повысить их надежность. Рекомендуется тщательно изучить по документации свойства и методы этих классов и выбирать наиболее подходящие в зависимости от решаемой задачи.

В C# имеется возможность обрабатывать исключения, создаваемые программистом. Для этого достаточно определить класс как производный от класса Exception. Как правило, определяемые программистом исключения, должны быть производными от класса ApplicationException, "родоначальника" иерархии, зарезервированной для исключений, связанных с прикладными программами. Созданные производные классы не должны ничего реализовывать, поскольку одно лишь их существование в системе типов уже позволит использовать их в качестве исключений.

Классы исключений, создаваемые программистом, будут автоматически иметь свойства и методы, определенные в классе Exception и доступные для них. Конечно, один или несколько элементов в новых классах нужно переопределить.

Задания

1 Создать иерархию классов (базовый – производный) в соответствии с вариантом 2. В производном классе определить свойство, которое возвращает ссылку на объект базового класса (это свойство должно возвращать ссылку на объект базового класса, а не ссылку на вызывающий объект производного класса). Например, для иерархии классов Person-Student в классе Student можно определить свойство

```
public Person BasePerson
{
get
{
return new Person(name, age);
}
}
```

3. Определить класс TestCollections, который содержит поля следующих типов

Коллекция_1<TKey> ;

Коллекция_1<string> ;

Коллекция_2<TKey, TValue> ;

Коллекция_2<string, TValue> .

где тип ключа TKey и тип значения TValue связаны отношением базовый-производный (см. задание 1), Коллекция_1 и Коллекция_2 – коллекции из пространства имен System.Collections.Generic.

4. Написать конструктор класса TestCollections, в котором создаются коллекции с заданным числом элементов.

5. Для автоматической генерации элементов коллекций в классе TestCollections надо определить статический метод, который принимает один целочисленный параметр типа int и возвращает ссылку на объект производного типа (Student). Каждый объект (Student) содержит подобъект базового класса (Person). Соответствие между значениями целочисленного параметра метода и подобъектами Person класса Student должно быть взаимно-однозначным.

6. Все четыре коллекции должны содержать одинаковое число элементов. Каждому элементу из коллекции Коллекция_1<TKey> должен отвечать элемент в коллекции Коллекция_2<TKey, TValue> с равным значением ключа. Список Коллекция_1<string> состоит из строк, которые получены в результате вызова метода ToString() для объектов TKey из списка Коллекция_1<TKey>. Каждому элементу списка Коллекция_1<string> отвечает элемент в Коллекция_2 <string, TValue> с равным значением ключа типа string.

7. Для четырех разных элементов – первого, центрального, последнего и элемента, не входящего в коллекцию – надо измерить время поиска элемента в коллекциях Коллекция_1<TKey> и Коллекция_1<string> с помощью метода Contains; элемента по ключу в коллекциях Коллекция_2< TKey, TValue> и Коллекция_2 <string, TValue > с помощью метода ContainsKey; значения элемента в коллекции Коллекция_2< TKey, TValue > с помощью метода ContainsValue.

8. Предусмотреть методы для работы с TestCollections (добавление и удаление элементов).

9. Предусмотреть собственные классы для обработки исключительных ситуаций и выполнить обработку исключений с помощью стандартных исключений, а также с помощью исключений, определенных программистом.

Варианты заданий:

	Коллекция_1	Коллекция_2
1.	List <T>	Dictionary <K,T>
2.	LinkedList <T>	SortedDictionary <K,T>
3.	List <T>	Dictionary <K,T>
4.	Queue<T>	SortedDictionary <K,T>
5.	Stack <T>	Dictionary <K,T>
6.	List <T>	SortedDictionary <K,T>
7.	LinkedList <T>	Dictionary <K,T>
8.	List <T>	SortedDictionary <K,T>
9.	Queue<T>	Dictionary <K,T>
10.	Stack <T>	SortedDictionary <K,T>
11.	List <T>	Dictionary <K,T>

12. LinkedList <T> SortedDictionary <K,T>
13. List <T> Dictionary <K,T>
14. Queue<T> SortedDictionary <K,T>
15. Stack <T> Dictionary <K,T>

Лабораторные работы №7

Использование регулярных выражений

Цель работы: освоить принципы поиска и сопоставления строковых данных с использованием регулярных выражений, написать программу с их использованием.

Теория

Регулярные выражения – это один из способов поиска подстрок (соответствий) в строках. Осуществляется с помощью просмотра строки в поисках некоторого шаблона (табл. 1.8). Очень эффективны библиотеки, интерпретирующие регулярные выражения, обычно пишутся на низкоуровневых высокопроизводительных языках (C, C++, Assembler). С помощью регулярных выражений выполняются три действия:

- проверка наличия соответствующей шаблону подстроки;
- поиск и выдача пользователю соответствующих шаблону подстрок;
- замена соответствующих шаблону подстрок.

Синтаксис регулярных выражений.

Регулярное выражение на C# задается строковой константой. Обычно используется @-константа. В C# работа с регулярными выражениями выглядит следующим образом:

Regex re = new Regex(«образец», «опции»);
MatchCollection me = re.Matches(—строка для поиска‖);
iCountMatchs = me.Count, где re – это объект типа Regex.
В конструкторе ему передается образец поиска и опции.

Символы описания шаблона

Категория: подмножества (классы) символов

. - Соответствует любому символу, за исключением символа конца **строки**

[aeiou]- Соответствует любому символу из множества, заданного в квадратных скобках

[^aeiou] - Отрицание. Соответствует любому символу, за исключением символов, заданных в квадратных скобках

[0-9a-fAF] - Задание диапазона символов, упорядоченных по коду. Так, 0-9 задает любую цифру

\w - Множество символов, используемых при задании идентификаторов – большие и малые символы латиницы, цифры и знак подчеркивания

\s - Соответствует символам белого пробела

\d - Соответствует любому символу из множества цифр

Категория: Операции (модификаторы)

*** -** Итерация. Задает ноль или более соответствий; например, \w* или (abc)*. Аналогично {0,}

+ - Положительная итерация. Задает одно или более соответствий; например, \w+ или (abc)+. Аналогично {1,}

?- Задает ноль или одно соответствие; например, \w? Или (abc)? Аналогично {0,1}

{n} - Задает в точности n соответствий; например, \w{2}

{n,} - Задает, по меньшей мере n соответствий; например, (abc){2,}

Категория: Группирование

(?<Name>) - При обнаружении соответствия выражению, заданному в круглых скобках, создается именованная группа, кото рой дается имя Name

() - Круглые скобки разбивают регулярное выражение на группы. Для каждого подвыражения, заключенного в круглые скобки, создается группа, автоматически получающая номер

Класс Regex.

Это основной класс, объекты которого определяют регулярные выражения.
В конструктор класса передается в качестве параметра строка, задающая регулярное выражение. Основные методы класса Regex:

- метод Match запускает поиск первого соответствия.

Параметром передается строка поиска. Метод возвращает объект класса Match, описывающий результат поиска.

- метод Matches позволяет разыскать все непересекающиеся вхождения подстрок, удовлетворяющие образцу.

В качестве результата возвращается объект MatchCollection, представляющий коллекцию объектов Match.

- метод NextMatch запускает новый поиск.
- метод Split является обобщением метода Split класса String.

Он позволяет, используя образец, разделить искомую строку на элементы.

- метод Replace – позволяет делать замену найденного образца.

Метод перегружен. При вызове метода передаются две строки: первая задает строку, в которой необходимо произвести замену, а вторая – на что нужно заменить найденную подстроку

Классы Match и MatchCollection.

Коллекция MatchCollection, позволяет получить доступ к каждому ее элементу – объекту Match. Для этого можно использовать цикл foreach. При работе с объектами класса Match наибольший интерес представляют свойства класса. Рассмотрим основные свойства:
 свойства Index, Length и Value наследованы от прародителя Capture.
Они описывают найденную подстроку – индекс начала подстроки в искомой строке, длину подстроки и ее значение;
 свойство Groups класса Match возвращает коллекцию групп – объект GroupCollection, который позволяет работать с группами, созданными в процессе поиска соответствия;

 свойство Captures, наследованное от объекта Group, возвращает коллекцию CaptureCollection.

Классы Group и GroupCollection.

Коллекция GroupCollection возвращается при вызове свойства Group объекта Match. Имея эту коллекцию, можно добраться до каждого объекта Group.

Свойства создаваемых групп:

- при обнаружении одной подстроки, удовлетворяющей условию поиска, создается не одна группа, а коллекция групп;
- группа с индексом 0 содержит информацию о найденном соответствии;
- число групп в коллекции зависит от числа круглых скобок в записи регулярного выражения. Каждая пара круглых скобок создает дополнительную группу;
- группы могут быть индексированы, но могут быть и именованными, в круглых скобках разрешается указывать имя группы.

Создание именованных групп крайне полезно при разборе строк, содержащих разнородную информацию.

Задания

Во всех заданиях исходные данные вводить с помощью ListBox. Вывод результата рганизовать в метку Label. Разработать метод класса Form, реализующий задание.

1. Дана строка, состоящая из групп нулей и единиц. Посчитать количество нулей и единиц.
2. Посчитать в строке количество слов.
3. Найти количество знаков препинания в исходной строке.
4. Дана строка символов. Вывести цифры, содержащиеся в строке.
5. Дана строка символов, состоящая из произвольных десятичных цифр, разделенных пробелами. Вывести количество четных чисел в этой строке.
6. Поменять местами символы с четными и нечетными номерами в заданной строке.
7. Дана строка символов. Вывести количество строчных русских букв, входящих в эту строку.
8. Дана строка символов. Вывести на экран только строчные русские буквы, входящие в эту строку.
9. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. В каждом слове заменить первую букву на прописную.
10. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Удалить первую букву в каждом слове.
11. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Поменять местами i- и j-ю буквы в каждом слове. Для ввода i и j на форме добавить свои поля ввода.
12. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Поменять местами первую и последнюю буквы каждого слова.

13. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Заменить все буквы латинского алфавита на знак ‘+’.
14. Дана строка символов, содержащая некоторый текст на русском языке. Заменить все большие буквы буквы ‘А’ на символ ‘*’.
15. Дана строка символов, содержащая некоторый текст. Проверить, является ли данный текст палиндромом, т.е. читается ли он слева направо так же, как и справа налево (например, «А роза упала на лапу азорА»).
16. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Сформировать новую строку, состоящую из чисел длин слов в исходной строке.
17. Дана строка символов. Сформировать из нее строку, в которой символы следуют в порядке возрастания их кодов.
18. Дана строка, содержащая слова, разделенные одиночными пробелами. Сформировать строку, в которой количество пробелов перед каждым словом будет равно его длине.
19. Дана строка, содержащая слова, разделенные пробелами. Вывести слова в порядке возрастания их длины.
20. Дана строка слов, разделенных группами пробелов. Вывести последовательность пар чисел «длина слова – длина группы пробелов».
21. Дана строка. Подсчитать и вывести частоту появления символов.
22. Дана строка, определить наибольшую длину последовательности, включающей заданные символы.
23. Найти и вывести все вещественные числа, содержащиеся в заданной строке.
24. Все под последовательности символов заданной строки, коды которых возрастают на единицу, заменить на тройки:
«первый символ» «–» «последний символ».
25. Дана строка, содержащая слова, разделенные пробелами. Вывести все слова, начинающиеся и заканчивающиеся одним и тем же символом без учета регистра.

Лабораторные работы №8

Операции со списками

Цель работы: научится работать со списками в C#.

Теория

C# имеет ряд классов для работы со списками. Они реализуют Интерфейс IList и наиболее популярной реализацией является общий список, часто называемый List<T>. T указывает Тип объектов, содержащихся в списке, который имеет дополнительное преимущество, что компилятор будет проверять и убедится, что вы добавите только объекты нужного типа в список - другими словами, List<T> является типобезопасным.

T обозначает тип и используется для указания типа объектов, которые должны одержаться в списке. В нашем первом примере, я покажу вам, как создать список, который должен содержать строки:

List<string> listOfStrings = new List<string>()

Это создает пустой список, но, легко добавляя в него впоследствии с помощью метода Add:

listOfStrings.Add("a string");

Однако, если вы попробуете добавить что-нибудь, что не является символьной строкой, компилятор немедленно пожалуется на это:

listOfStrings.Add(2);

Error CS1503 Argument 1: cannot convert from 'int' to 'string'

Инициализация списка с элементами

В приведенном выше примере мы просто создали список, а затем добавили в него элемент. Однако C# фактически позволяет создавать список и добавлять в него элементы в пределах одной инструкции, используя метод, называемый инициализаторами коллекций. Давайте посмотрим, как это делается:

```
List<string> listOfNames = new List<string>()
{
   "John Doe",
   "Jane Doe",
   "Joe Doe"
};
```

Синтаксис довольно прост: перед обычной конечной точкой с запятой у нас есть комплект фигурных скобок, который, в свою очередь, содержит список значений, которые мы с самого начала хотим, чтобы присутствовали в списке. Поскольку это список строк, начальные объекты, которые мы предоставляем, должны, конечно, иметь строковый тип. Однако то же самое можно сделать для списка других типов, даже если мы используем наши собственные классы, как я продемонстрирую в следующем примере.

Задания

Опишите классы-списки, необходимые для решения задачи, указанной в вашем варианте задания, и реализуйте его методы.

Составьте программу решения задачи с использованием динамических структур данных, реализованных в виде классов.

Варианты заданий

1. Описать односвязный линейный список, каждый элемент которого представляет собой запись, состоящую из двух полей: символ и количество его повторений в тексте.

Сформировать список из заданного текста. Для создания списка использовать метод вставки в список в отсортированном порядке.

2. Сложение длинных чисел. Используя двусвязные циклические списки сложить два числа: x =134557952499317879 и y = 79349864365110 . Числа разбить на группы по девять цифр, которые будут являться элементами списка. Результат сложения занести в циклический список.

3. Задача Джозефуса. По кругу стоит группа людей. С терминала вводятся число n и фамилия человека, с которого начинается счет. Начиная с введенной фамилии, по кругу отсчитывается n человек, /7-й выбывает из круга и т.д. Выводить фамилии исключаемых из круга людей. Задачу решить, используя односвязный циклический список.

4. Создать два односвязных циклических списка с целочисленными элементами. Переписать в односвязный линейный список элементы с совпадающими значениями.

5. Описать односвязный циклический список символов. Реализовать метод вставки, который выполняет вставку отсутствующего в списке символа в его конец и перемещение в начало элемента с присутствующим в списке значением. Продемонстрировать работу метода.

6. Проверить правильность расстановки скобок (круглых, квадратных, фигурных и угловых) в арифметическом выражении. Для слежения за скобками воспользоваться односвязным циклическим списком.

7. Описать двусвязный линейный список строк. Реализовать метод вставки, который выполняет вставку отсутствующего в списке символа в его конец и перемещение на один элемент к началу списка элемента с присутствующим в списке значением.

8. Из односвязного списка с фамилиями студентов двух групп сформировать два циклических по группам, причем вставка в циклические списки должна выполняться в отсортированном порядке.

9. С использованием односвязного циклического списка определить правильность вложения циклов в программе на языке Бейсик, состоящей из произвольного количества последовательно расположенных или вложенных инструкций:

10 FOR x=a TO b STEP c

30 NEXT c где x — переменная цикла. Для каждого цикла выводить сообщение вида: «Цикл по x открыт», «Цикл по x закрыт».

10. Создать односвязные списки студентов двух групп с полученными на экзаменах оценками. Сформировать в алфавитном порядке односвязный циклический список студентов, получивших отличные оценки.

11. Сформировать линейный список строк. Исключить из него последние три элемента. Скопировать в двусвязный циклический список вторую половину его элементов, начиная с конца.

12. С использованием односвязного циклического списка определить правильность вложения циклов в программе на языке Бейсик, состоящей из произвольного количества последовательно расположенных или вложенных инструкций:

10 FOR x=a TO b STEP c

30 NEXT c

где x — переменная цикла, один оператор NEXT может завершать одновременно несколько вложенных циклов (например: NEXT x,y,z). Для каждого цикла выводить сообщение вида: «Цикл по x открыт», «Цикл по x закрыт».

13. Описать односвязный циклический список, тип значения элемента которого — запись из двух полей: символ и количество его повторений в тексте. Сформировать список из заданного текста. Переписать в линейный список элементы с русскими буквами.

14. Сформировать два двусвязных циклических списка с элементами — фамилиями клиентов. Удалить из первого списка клиентов, находящихся в двух списках.

15. Сформировать циклический список целых чисел. Скопировать элементы исходного списка в линейный список таким образом, чтобы они были расположены в нем в порядке, обратном к исходному.