ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Кафедра «Вычислительные системы и информационная безопасность»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к курсовой и лабораторным работам
по дисциплине

«Технологии и методы программирования»

Автор
Айдинян А.Р.

Ростов-на-Дону, 2014

Аннотация

В компактной форме приводятся краткие теоретические сведения к выполнению лабораторных работ, порядок выполнения лабораторных работ, индивидуальные задания и контрольные вопросы.

Автор

д.т.н., доцент Айдинян А.Р.

Оглавление

Лабораторная работа 1 Организация хранения объектов в списке на языке C# с возможностью редактирования 5

Краткие теоретические сведения..... 5
Задание: 8
Варианты индивидуальных заданий. 8

Лабораторная работа 2 Интерфейсы, абстрактные классы, наследование и исключения 10

Краткие теоретические сведения.....10
Задание для выполнения работы.....18
Варианты заданий 20
Контрольные вопросы 24

Лабораторная работа 3 Использование технологии LINQ для работы с классами.....25

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 25
ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ.....26
ВАРИАНТЫ ИНДИВИДУАЛЬНЫХ ЗАДАНИЙ.....27

Лабораторная работа 4 Использование технологии LINQ для работы с классами.....31

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 31
ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ.....32
КОНТРОЛЬНЫЕ ВОПРОСЫ.....37

Лабораторная работа 5 Использование COM-технологии на языке C# 38

Краткие теоретические сведения.....38
Порядок использования технологии COM на примере Word. 39
Задания для выполнения работы.....41

Лабораторная работа 6 Рекурсивные алгоритмы 42

Краткие теоретические сведения.....42
Индивидуальные задания 45
Контрольные вопросы 47

Лабораторная работа 7 Реализация рекурсивных алгоритмов на примере головоломки «Ханойские башни»

.....49

Краткие теоретические сведения.....49
Порядок выполнения работы.....52
Индивидуальные задания.....53
Контрольные вопросы.....53
Лабораторная работа 8 Реализация графического вывода на языке C# 54

Краткие теоретические сведения.....54
Порядок выполнения работы.....58
Контрольные вопросы.....58
Лабораторная работа 9 Алгоритмы поиска.....60

Теоретические сведения.....60
Порядок выполнения работы и рекомендации по реализации программы.....64
Контрольные вопросы.....65
Лабораторная работа 10 Алгоритмы сортировки.....66

Алгоритмы сортировки.....66
Алгоритм сортировки выбором.....66
Алгоритм быстрой сортировки.....70
Вычислительная сложность алгоритмов.....73
Порядок выполнения работы.....75
Контрольные вопросы.....76
Методические указания к курсовой работе.....77

1. ЦЕЛЬ И ЗАДАЧИ КУРСОВОГО ПРОЕКТИРОВАНИЯ.....77
2. ОБЩИЕ РЕКОМЕНДАЦИИ ПО РАЗРАБОТКЕ ПО.....77
3. СОДЕРЖАНИЕ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ.....80
4. ПРИМЕРНЫЕ ВАРИАНТЫ ЗАДАНИЙ.....84
Варианты заданий на курсовой проект.....85
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....89
ПРИЛОЖЕНИЕ 1.....90

ПРИЛОЖЕНИЕ 2.....92

ПРАВИЛА ОФОРМЛЕНИЯ ДИАГРАММЫ КЛАССОВ.....92

ЛАБОРАТОРНАЯ РАБОТА 1
ОРГАНИЗАЦИЯ ХРАНЕНИЯ ОБЪЕКТОВ В СПИСКЕ НА ЯЗЫКЕ C# С ВОЗМОЖНОСТЬЮ РЕДАКТИРОВАНИЯ

Цель работы: изучить способ создания, отображения и редактирования списков на платформе .NET.

Краткие теоретические сведения

Распространенным средством хранения набора данных является объект класса List. Он позволяет добавлять в него новые элементы, удалять, сортировать и так далее.

List является шаблонным классом. Поэтому при описании объекта необходимо указать тип данных, которые он будет содержать.

Используем объект *list* класса **List** для хранения граждан – объектов класса *Grajdanin*.

Класс гражданин содержит 3 свойства типа string.

```
class Grajdanin
{
    public string Fam { get; set; }  //фамилия
    public string Nam { get; set; }  //имя
    public string Country { get; set; }  //страна
}
```

Запись

```
public string Fam { get; set; }  //фамилия
```

вводит свойство *Fam* и автоматически создаваемое поле и является более компактной и удобной формой ранее изучаемой записи

```
private string m_Fam;
public string Fam
{
  get { return m_Fam; }
  set { m_Fam = value; }
}
```

В классе автоматически создается конструктор по умолчанию. Это правило выполняется для любого класса, не имеющего явно описанного конструктора.

Для создания списка *list* для хранения граждан необходимо

записать

List<*Grajdanin*> *list* = **new List**<*Grajdanin*>();

При таком описании создается пустой список, в который можно добавлять объекты класса *Grajdanin*, используя метод Add класса **List**, и конструктор по умолчанию класса *Grajdanin*.

Добавление объекта класса *Grajdanin* в список записывается следующим образом

list.Add(**new** *Grajdanin*());

К сожалению, конструктор по умолчанию инициализирует поля пустыми строками, что потребует им явное присваивание значений.

list[0].Fam = "Иванов";
list[0].Nam = "Иван";
list[0].Country = "Россия";
или инициализировать поля до добавления в список так:
Grajdanin g = new *Grajdanin*();
g.Fam = "Иванов";
g.Nam = "Иван";
g.Country = "Россия";
list.Add(*g*);

В языке C#, начиная с Visual Studio 2008, имеется возможность добавлять объекты в список и инициализировать поля с помощью более компактной и удобной записи, которая выполняет то же, что и предыдущий программный код.

List<*Grajdanin*> *list* = **new List**<*Grajdanin*>()
{
new *Grajdanin* { *Fam* = "Иванов", *Nam* = "Иван", *Country* ="Russia"},
new *Grajdanin* { *Fam* = "Петров", *Nam* = "Петр", *Country* ="Russia"},
new *Grajdanin* { *Fam* = "Сидоров", *Nam* = "Сема", *Country* ="Germany"},
new *Grajdanin* { *Fam* = "Кизин", *Nam* = "Киз", *Country* ="Russia"}
};

Обратите внимание на то, что новая коллекция заполняется непосредственно внутри фигурных скобок. Даже, несмотря на то, что тип коллекции — **List<T>**, а не массив, мы все равно имеем возможность добавлять элементы без непосредственного вызова

метода Add**()**. Даже при отсутствии конструктора с тремя параметрами для класса *Grajdanin* мы все равно имеем возможность создавать объекты этого типа как выражения, в которых внутри фигурных скобок задаются значения полей этого объекта.

Для отображения полученного списка можно использовать **DataGridView** и **BindingNavigator**, которые привяжем к **BindingSource**, а его в свою очередь к списку *list*.

```
BindingSource bindingSource = new BindingSource();
bindingSource.DataSource = list;
dataGridView1.AutoGenerateColumns = true;
dataGridView1.DataSource = bindingSource;
bindingNavigator1.BindingSource = bindingSource;
```

Рекомендуется к классу *Grajdanin* добавить необходимые конструкторы и метод ToString(), например, так

```
class Grajdanin
 {
    public string Fam { get; set; }
    public string Nam { get; set; }
    public string Country { get; set; }
    public override string ToString()
    {
       return Fam + " " + Nam + " " + Country;
    }

    public Grajdanin()// конструктор по умолчанию
    {
       Fam = "Иванов"; Nam = "Иван"; Country = "Russia";
    }
    public Grajdanin(Grajdanin g) // конструктор кописрования
    {
       Fam = g.Fam; Nam = g.Nam; Country = g.Country;
    }
   // конструкторы с параметрами  - 2 штуки
    public Grajdanin(string fam, string nam)
    {
       Fam = fam; Nam = nam; Country = "Russia";
    }
```

```
        public Grajdanin(string fam, string nam, string country)
        {
            Fam = fam; Nam = nam; Country = country;
        }
    }
```

Задание:

1. Создать приложение типа **Window Forms**.
2. Создать класс в соответствии с вариантом. Добавить в него указанные поля (свойства) и дополнительно *Id* (уникальный номер). Также добавить конструкторы и метод ToString.
3. Создать объект класса **List** для хранения списка объектов созданного класса. Заполнить его 3 – 4 объектами программно.
4. Отобразить список на форме в элементе управления **DataGridView** с использованием **BindingNavigator**.
5. Проверить возможность добавления, редактирования и удаления элементов в **DataGridView** с использованием клавиш клавиатуры и через **BindingNavigator**.

Варианты индивидуальных заданий.

1. Разработать класс **CWorker** (Сотрудник), содержащий свойства PersonID (Табельный номер сотрудника), Family (Фамилия сотрудника), Birth (Дата рождения сотрудника).
2. Разработать класс **C**Jobless (Безработный), содержащий свойства JoblessID (регистрационный номер безработного), LastName (Фамилия безработного), FirstName (Имя безработного), Age (возраст безработного).
3. Разработать класс CDocument (Документ), содержащий свойства: Serial (серия документа), Number (номер документа), Date (Дата составления), Who (кем составлен)
4. Разработать класс **CZakazchik** (Заказчик), содержащий свойства: IdZakazchika (Идентификатор заказчика), Phone (телефон заказчика), Address (адрес заказчика).
5. Разработать класс **CHoz** (Владелец автомобиля), содержащий свойства: Family (фамилия владельца), Number (номер паспорта владельца).
6. Разработать класс **CRoute** (Маршрут), содержащий свойства: RouteID (идентификатор маршрута), RouteName (название маршрута), Period (срок пребывания), Cost (стоимость путевки).

7. Разработать класс **CDoctor** (Врач), содержащий свойства: RouteID (идентификатор врача), Family (фамилия врача), Specialnost (специальность врача), Room (номер кабинета).

8. Разработать класс **CCust** (Арендатор), содержащий свойства: CustomerID (ИНН арендатора), Customer (название арендатора), AddressCust (адрес арендатора), Room (номер кабинета), Chief (фамилия руководителя).

9. Разработать класс **CVlad** (Владелец), содержащий свойства: FIO (ФИО владельца), Phone (телефон владельца), Registr (Регистрационный номер клиента).

10. Разработать класс **CTelephonNumber** (Телефонный номер), содержащий свойства: ID (идентификатор клиента), Family (фамилия клиента), PhoneAddress (адрес клиента), PhoneNumber (номер телефона), Value (Ежемесячная плата за телефон).

11. Разработать класс **CCount** (Счет-фактура), содержащий свойства: CountNumber (Номер счет-фактуры), Date (Дата выписки счет-фактуры), Value (Сумма к уплате), Cost (Стоимость приобретения).

12. Разработать класс **CRabotnik** (Работник автовокзала), содержащий свойства: FIO (ФИО работника), Dolg (Должность), Zarplata (оклад).

13. Разработать класс **CAgreement** (Договор), содержащий свойства: Number (Номер договора), Date (Дата договора).

14. Разработать класс **CClient** (Клиент), содержащий свойства: Family (Фамилия), BeginDate (дата въезда), EndDate (Дата отъезда), Summa (Сумма оплаты), NKvit (номер квитанции).

15. Разработать класс **CBuy** (покупка), содержащий свойства: Number (Номер покупателя), Sum (Сумма покупки), Otdel (название отдела магазина, где совершена покупка).

16. Разработать класс **CSpec** (специальность), содержащий свойства: Spec (название специальности), FIO (ФИО работника), Balls (количество баллов, необходимых для поступления).

17. Разработать класс **CBilet (**Билет**)**, содержащий свойства: Time1 (время вылета), Time2 (Время прилета), Cost (цена билета), Name (владелец билета), Nomer (номер места).

ЛАБОРАТОРНАЯ РАБОТА 2
ИНТЕРФЕЙСЫ, АБСТРАКТНЫЕ КЛАССЫ, НАСЛЕДОВАНИЕ И ИСКЛЮЧЕНИЯ

Цель работы: изучить объектно-ориентированное программирование с использованием наследования, абстрактных классов и исключений.

Краткие теоретические сведения

Наследование, вместе с инкапсуляцией и полиморфизмом, является одним из базовых понятий объектно-ориентированного программирования. Наследование позволяет создавать новые классы, которые повторно используют, расширяют и изменяют поведение, определенное в других классах. Класс, члены которого наследуются, называется базовым классом, а класс, который наследует эти члены, называется производным классом. Производный класс может иметь только один непосредственный базовый класс.

Введем абстрактный класс Shape. Объекты абстрактного класса созданы быть не могут.

```
abstract class Shape
{
public string Name { get; set; }
public int X { get; set; }
public int Y { get; set; }
public Shape()
{
        Name="Figura";
        X=0;
        Y=0;
}

public Shape(string name, int x, int y)
{
        Name=name;
        X=x;
        Y=y;
}
}
```

На основе класса *Shape* будем разрабатывать конкретные

классы, например, класс *Kvadrat*.

```
class Kvadrat : Shape
{
public int Dlina { get; set; }
public Kvadrat(): base()
{
        Dlina = 1;
}
public Kvadrat(string name, int x, int y, int dlina)
        : base(name, x, y)
{
        Dlina = dlina;
}

public override void Draw(Graphics g)
{
        g.DrawPolygon(new                                Pen(new
  SolidBrush(Color.Black)),
        new Point[4]
        {
                new Point(X, Y),
                new Point(X+Dlina, Y),
                new Point(X+Dlina, Y+Dlina),
                new Point(X, Y+Dlina)
        }
        );
}
public override float P ( )
{
        return Dlina * 4;
}

public override float  S ( )
{
        return Dlina * Dlina;
}
}
```

Для того, чтобы класс *Kvadrat* и остальные производные классы имели возможность быть прорисованными на форме, необходимо добавить виртуальный метод (virtual-метод) Draw в

класс *Shape*. Однако, метод Draw лучше реализовать в интерфейсе **IDraw**:

```
interface IDraw
{
        void Draw(Graphics g);
}
```

Параметр *g* класса **Graphics** предназначен для определения графического контекста, на котором будет осуществляться прорисовка.

Для прорисовки на форме необходимо получить графический контекст формы, создав его с помощью программного кода, например в конструкторе формы:

```
Graphics g = this.CreateGraphics(); .
```

Тогда класс *Shape* имеет вид:

```
abstract class Shape : IDraw
{
        public string Name { get; set; }
        public int X { get; set; }
        public int Y { get; set; }
        public Shape()
        {
                Name="Figura";
                X=0;
                Y=0;
        }

        public Shape(string name, int x, int y)
        {
                Name = name;
                X = x;
                Y = y;
        }

        public abstract void Draw(Graphics g);
}
```

Метод Draw в классе *Shape* является абстрактным, поскольку, неизвестно, как рисовать абстрактную фигуру. Однако в классе *Kvadrat* необходимо переопределить метод Draw

```
public override void Draw(Graphics g)
{
```

```
            g.DrawPolygon(new                    Pen(new
    SolidBrush(Color.Black)),
                    new    Point[4]{new    Point(X,    Y),    new
                Point(X+Dlina, Y),
                    new Point(X+Dlina, Y+Dlina), new Point(X,
                Y+Dlina) });
        }
```

Для создания объекта класса *Kvadrat* можно использовать следующий код

```
        Kvadrat k = new Kvadrat("1", -63, 4, 122);
```

На базе класса *Kvadrat* можно создать класс «Цветной квадрат»

```
        class ColorKvadrat : Kvadrat
        {
        public Color Color { get; set; }
        public ColorKvadrat ( )  : base( )
        {
                Color = Color.Blue;
        }

        public ColorKvadrat(string name, int x, int y, int dlina, Color
color)
                : base(name, x, y, dlina)
        {
                Color = color;
        }

        public override void Draw(Graphics g)
        {
                g.DrawPolygon(  new Pen(new SolidBrush(Color)),
                                        new Point[4] {
                                                new   Point(X,
                                        Y),
                                                new Point(X +
                                        Dlina, Y),
                                                new Point(X +
                                        Dlina, Y + Dlina),
                                                new   Point(X,
```

```
                                                    Y + Dlina)
                                                }
            );
    }
    }
```

Также необходимо добавить интерфейс с методами для вычисления площади и периметра фигуры.

```
interface ICalcPS
{
/// <summary>
/// Вычисление периметра
/// </summary>
/// <returns>Периметр</returns>
float P( );

/// <summary>
/// Вычисление площади
/// </summary>
/// <returns>Площадь</returns>
float S( );
}
```

В классе *Shape* необходимо реализовать интерфейс *ICalcPS* и реализовать эти методы в классе *Kvadrat*. Класс *ColorKvadrat* наследует методы *P* и *S* без изменений, а метод *Draw* необходимо переопределить в связи с необходимостью учета цвета прорисовки.

Для хранения списка геометрических фигур *Kvadrat* и *ColorKvadrat* необходимо создать список

```
List<Shape> list = new List<Shape> ( )
{
new Kvadrat(),
new ColorKvadrat("Цв. квадрат 1", 5, 88, 34, Color.Yellow),
new ColorKvadrat("Цв. квадрат 2", 45, 58, 134, Color.Green),
new Kvadrat("Квадрат 1", 145, 158, 64)
};
```

Благодаря тому, что список **List** содержит элементы класса *Shape*, в него можно помещать элементы любого производного от него класса: в данном случае *Kvadrat* и *ColorKvadrat*.

Для вызова правильных методов необходимо описать мето-

ды *Draw*, *P* и *S* как **override**.

Исключения. Далеко не всегда ошибки случаются по вине того, кто кодирует приложение. Иногда приложение генерирует ошибку из-за действий конечного пользователя, или же ошибка вызвана контекстом среды, в которой выполняется код. В любом необходимо ожидать возникновения ошибок в своих приложениях и проводить кодирование в соответствии с этими ожиданиями.

В .NET Framework предусмотрена развитая система обработки ошибок. Механизм обработки ошибок C# позволяет закодировать пользовательскую обработку для каждого типа ошибочных условий, а также отделить код, потенциально порождающий ошибки, от кода, обрабатывающего их.

Для того чтобы справиться с возможными ошибочными ситуациями в коде C#, программа обычно делится на блоки трех разных типов:

Блоки **try** инкапсулируют код, формирующий часть нормальных действий программы, которые потенциально могут столкнуться с серьезными ошибочными ситуациями.

Блоки **catch** инкапсулируют код, который обрабатывает ошибочные ситуации, происходящие в коде блока **try**. Это также удобное место для протоколирования ошибок.

Блоки **finally** инкапсулируют код, очищающий любые ресурсы или выполняющий другие действия, которые обычно нужно выполнить в конце блоков **try** или **catch**. Важно понимать, что этот блок выполняется независимо от того, сгенерировано исключение или нет.

Основу обработки исключительных ситуаций в C# составляет пара ключевых слов **try** и **catch**. Эти ключевые слова действуют совместно и не могут быть использованы порознь. Ниже приведена общая форма определения блоков **try**/**catch** для обработки исключительных ситуаций:

```
try {
// Блок кода, проверяемый на наличие ошибок.
}

catch (ExcepType1 exOb) {
// Обработчик исключения типа ExcepType1.
}
catch (ExcepType2 exOb) {
// Обработчик исключения типа ExcepType2.
}
```

где ExcepType — это тип возникающей исключительной ситуации. Когда исключение генерируется оператором **try**, оно перехватывается составляющим ему пару оператором **catch**, который затем обрабатывает это исключение. В зависимости от типа исключения выполняется и соответствующий оператор **catch**. Так, если типы генерируемого исключения и того, что указывается в операторе **catch**, совпадают, то выполняется именно этот оператор, а все остальные пропускаются. Когда исключение перехватывается, переменная исключения exOb получает свое значение. На самом деле указывать переменную exOb необязательно. Так, ее необязательно указывать, если обработчику исключений не требуется доступ к объекту исключения, что бывает довольно часто. Для обработки исключения достаточно и его типа.

Следует, однако, иметь в виду, что если исключение не генерируется, то блок оператора **try** завершается как обычно, и все его операторы **catch** пропускаются. Выполнение программы возобновляется с первого оператора, следующего после завершающего оператора **catch**. Таким образом, оператор **catch** выполняется лишь в том случае, если генерируется исключение.

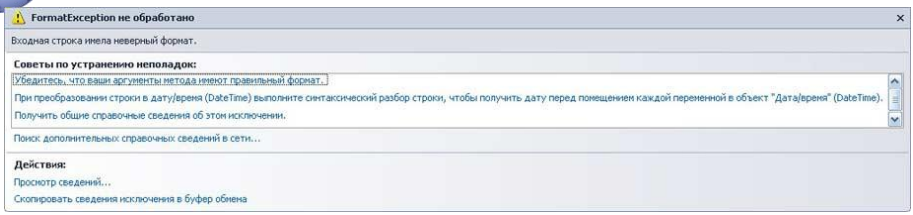
Давайте рассмотрим пример, в котором будем обрабатывать исключение, возникающее при делении числа на 0:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static int MyDel(int x, int y)
      {
         return x / y;
      }

      static void Main()
      {
      link1:
         try
         {
            Console.Write("Введите x: ");
```

```
            int x = int.Parse(Console.ReadLine());
            Console.Write("Введите y: ");
            int y = int.Parse(Console.ReadLine());
            int result = MyDel(x, y);
            Console.WriteLine("Результат: " + result);
        }
        // Обрабатываем исключение возникающее при делении на ноль
        catch (DivideByZeroException)
        {
            Console.WriteLine("Деление на 0 detected!!!\n");
            goto link1;
        }
        // Обрабатываем исключение при неккоректном вводе числа в консоль
        catch (FormatException)
        {
            Console.WriteLine("Это НЕ число!!!\n");
            goto link1;
        }

    Console.ReadLine();
    }
  }
}
```

Последствия неперехвата исключений. Перехват исключения исключает аварийное завершение программы. Как только исключение будет сгенерировано, оно должно быть перехвачено каким-то фрагментом кода в определенном месте программы. Вообще говоря, если исключение не перехватывается в программе, то оно будет перехвачено исполняющей системой. Но дело в том, что исполняющая система выдаст сообщение об ошибке и прервет выполнение программы.

Например, если убрать из предыдущего примера исключение FormatException, то при вводе неккоректной строки, IDE-среда VisualStudio выдаст предупреждающее сообщение:



Такие сообщения об ошибках полезны для отладки программы, но, по меньше мере, нежелательны при ее использовании на практике! Именно поэтому так важно организовать обработку исключительных ситуаций в самой программе.

Задание для выполнения работы.

1. Реализовать абстрактный класс *Shape* (геометричекая фигура) со свойствами, предназначенными для описания координат фигуры *X* и *Y*, и имени фигуры). Также определить конструктор по умолчанию и конструктор с параметрами.
2. Определить производный класс "Конкретная геометрическая фигура" в соответствии с вариантом на базе класса *Shape*. Реализовать в полученном классе свойства для описания конкретной фигуры и необходимые конструкторы. Конструкторы производного класса должны обращаться к базовому классу.
3. Сделать возможным вывод фигуры на форму. Для этого определить интерфейс *IDraw* с методом, предназначенным для рисования геометрической фигуры на форму. Учесть, что метод рисования должен получить информацию том, куда нарисовать фигуру. Разработать класс *Shape* и реализовать в нем интерфейс *IDraw*. Добавить метод рисования как абстрактный метод в класс *Shape*.
4. Реализовать методы интерфейса *IDraw* в производном классе "Конкретная геометрическая фигура".
5. В классе *Form1* создать объект класса "Конкретная геометрическая фигура" и нарисовать на экране.
6. В классе *Form1* создать объект класса «Конкретная геометрическая фигура» и сохранить ссылку на него в объекте класса *Shape* и нарисовать на экране.
7. Описать интерфейс *ICalcPS* с методами для вычисления периметра и площади геометрической фигуры. Наследовать базовый класс *Shape* от интерфейса *ICalcPS*.
8. Реализовать методы интерфейса *ICalcPS* в классе «Конкретная геометрическая фигура».
9. Реализовать класс «Конкретная геометрическая

фигура2» в соответствии с вариантом.

10.Создать список объектов класса *Shape* и записать в него объекты классов «Конкретная геометрическая фигура» и «Конкретная геометрическая фигура2» в произвольном порядке через элементы управления формы. Для этого на форме создать кнопки «Добавить Конкретную геометрическую фигуру», «Добавить Конкретную геометрическую фигуру2».

11.Нарисовать на экране фигуры из списка с помощью оператора **foreach** следующим образом

list.**ForEach**(*a* => *a*.Draw(*g*));

где *g* — графический контекст формы.

12.Поместить на форму элемент управления **DataGridView**, добавить в него 2 столбца и вывести в них периметры и площади всех фигур из списка.

Пример для вывода информации о площади фигур в первый столбец **DataGridView** приведен ниже:

```
int j = 0;
foreach (var i in list)
{
        s =  i.S().ToString() ;
        dataGridView1.Rows.Add(new DataGridViewRow());
        dataGridView1.Rows[j].Cells[0].Value = s;
        j++;
}
```

13.Реализовать для классов «Конкретная геометрическая фигура» и «Конкретная геометрическая фигура2» метод ToString для получения строки с описанием фигуры. Вывести информацию о фигуре в третий столбец **DataGridView**.

14.Реализовать исключения при задании параметров фигур некорректным образом в соответствии с вариантом. Для этого заменить краткий формат определения свойств на полный и осуществить проверку. (**ArgumentExeption**).

Для этого переопределить свойство, требующее проверки и в случае ошибки сгенерировать исключение:

```
private int x;
public int X
{
        get { return x; }
        set
        {
                if (value < 0)
                        throw                    new
```

```
                    ArgumentException("x<0" + value.ToString() );
                //Этот оператор не выполнится в случае исключения.
                    x = value;
                }
        }
```

а затем отловить это исключение в форме приложения при создании объекта

```
Kvadrat k = null;
try
{
        Kvadrat k = new ColorKvadrat("1", -63, 4, 122, Color.Red);
}
catch(ArgumentException ex)
{
        MessageBox.Show(ex.Message);
}
```

или изменении значений свойства

```
try
{
        k.X=-200;
}
catch(ArgumentException ex)
{
        MessageBox.Show(ex.Message);
}
```

Варианты заданий

№	Конкретная геометрическая фигура	Конкретная геометрическая фигура2	Способ задания фигуры	Исключение

1	Квадрат с произвольной ориентацией	Квадрат с заданным цветом заливки	Квадрат задается координатами всех четырех вершин	1) Длины сторон не равны между собой. 2) Координаты 2 вершин совпадают
2	Квадрат со сторонами параллельными осям координат	Квадрат с заданным цветом заливки	Квадрат задается длиной стороны	1) Длина стороны =0. 2) Длина стороны < 0.
3	Треугольник	Цветной треугольник	Задается координатами 3 вершин	3) Длина стороны = 0. 4) Сумма 2 сторон меньше длины третьей стороны. 5) Сумма 2 сторон равна длине третьей стороны.

4	Треугольник	Цветной треугольник	Задается длинами сторон	1) Длина стороны <= 0. 2) Сумма 2 сторон меньше длины третьей стороны. 3) Сумма 2 сторон равна длине третьей стороны.
5	Треугольник	Цветной треугольник	Задается длиной 1 стороны и координатой вершины, не принадлежащей стороне	1) Вершина 3 принадлежит первой стороне. 2) Вершина 3 принадлежит первой стороне. 3)

6	Круг	Цветной круг со сплошной заливкой	Задается радиусом	1) Радиус = 0. 2) Радиус < 0.
7	Круг	Цветной круг с несплошной заливкой	Задается площадью	1) Площадь = 0. 2) Площадь < 0.
8	Окружность	Цветная окружность	Задается радиусом	1) Радиус <= 0. 2) Радиус > 200.
9	Овал	Цветной овал	Задается двумя радиусами	1) Радиус 1 <= 0 2) Радиус 2 <= 0.
10	Овал	Цветной овал	Задается габаритами	1) Габарит1 <= 0 или Габарит2 <= 0. 2) Габарит1 < Габарит2.

11	Четырех-угольник произ-вольный	Цветной четырех-угольник произ-вольный	Задается коор-динатами 4 вершин	1) Дл ина од-ной из сторон равна 0. 2) Су мма длин 3 сторон меньше длины четвер-той
12	Цилиндр	Цветной цилиндр	Задается радиу-сом и высотой	1) Радиус <=0. 2) Высота <=0.
13	Цилиндр	Цветной цилиндр	Задается одним числом, который равен радиусу и высоте	1) Число <=0, 2) Высота >130.

Контрольные вопросы

1. Назначение наследования.
2. Реализация исключений на языке C#.
3. Способ реализация наследования на языке C#.
4. Правила реализации конструкторов в классе-наследнике.
5. Чем отличается абстрактный класс от неабстрактного класса.
6. Чем отличается интерфейс от класса.

ЛАБОРАТОРНАЯ РАБОТА 3
ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ LINQ ДЛЯ РАБОТЫ С КЛАССАМИ

Цель работы: исследовать возможности технологии LINQ при работе с коллекцией чисел.

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

LINQ (Language Integrated Query — Язык Интегрированных Запросов) — это набор расширений языка, поддерживающий формирование запросов данных с контролем типов.

Базовые единицы данных в LINQ — последовательности. Последовательность — любая коллекция, реализующая интерфейс IEnumerable<> или IQueryable<>.

Запрос — выражение, которое преобразует последовательность с помощью операторов запроса. Оператор запроса — метод, преобразующий последовательность. Оператор запроса никогда не изменяет входную последовательность, вместо этого он возвращает новую.

Для использования LINQ необходимо подключить пространство имен System.Linq и наличие Framework 3.5 или выше.

LINQ поддерживает два вида синтаксиса: синтаксис запросов и синтаксис методов.

Синтаксис запросов:

```
int[] msv = { -1, 2, 3, -4 };
var res = from n in msv
where n > 0
select n; // 2, 3
```

Такой синтаксис всегда должен начинаться с from in и заканчиваться select или group.

Синтаксис методов:

```
int[] msv = { -1, 2, 3, -4 };
var res = msv.Where(n => n > 0); // 2, 3
```

Методы можно применять один за другим

```
int[] msv = { -1, 2, 3, -4, 0, 2, 8, 9, 3 };
var res = msv.Where(n => n > 0 && n % 2 == 0).Select(n => n + 1).Take(3).Sum(); // 15
```

Лямбда-выражения — это специальный синтаксис для сокращённого объявления анонимных методов. Пример синтаксиса: n => n * n. Это выражение принимает один параметр n и возвращает n * n.

```
int[] msv = { 1, 2, 3 };
var res = msv.Select(n => n * n); // 1, 4, 9
```

Оператор => — лямбда оператор, который читается как "переходит в".

Слева от лямбда оператора стоят параметры ввода, справа — выражение.

Это лямбда-выражение эквивалентно такому: n => { return n * n; } и такому delegate(int n) { return n * n; }.

Стандартные операторы запроса используют обобщенные делегаты Func, например Func<int, bool> соответствует лямбда-выражению int => bool, а Func<string, string, int> — (string, string) => int. Возвращаемое значение Func всегда указывается в качестве последнего аргумента.

Методы для работы с последовательностями в LINQ: Aggregate; All, Any, Average, Contains, Count, First, Max, Min, OrderBy, OrderByDescending, Reverse, Select, SequenceEqual, Skip, SkipWhile, Sum, Take, TakeWhile, ToArray, ToList, ToString, Union, Where.

Рассмотрим примеры использования технологии LINQ.

Определить отношение суммы элементов массива, расположенных до первого из минимальных элементов в массиве, к произведению элементов, расположенных после минимального

```
var min = V.Min(a=>a);
var s = V.TakeWhile(a => a != min).Sum() * 1.0 /
V.SkipWhile(a => a != min).Skip(1).Sum();
```

или одним оператором

```
var s2 = V.TakeWhile(a => a != V.Min()).Sum() * 1.0 /
V.SkipWhile(a => a != V.Min()).Skip(1).Sum();
```

Определить сумму элементов после последнего максимального

```
var s4 = V.Reverse().TakeWhile(a => a != V.Max()).Sum();
```

Найти произведение элементов после последнего максимального

```
var s5 = V.Reverse().TakeWhile(a => a != V.Max()).Aggregate((x, y) => x * y);
```

Найти минимальный по модулю элемент массива

```
var min = V.Min(a => Math.Abs(a));
```

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Напишите программу для выполнения действий над массивом в соответствии с индивидуальным заданием 1 и 2. Элементы массива можно задать программно.

ВАРИАНТЫ ИНДИВИДУАЛЬНЫХ ЗАДАНИЙ

Задание 1. Обработка элементов вектора

Дан одномерный числовой массив.

1) Определить отношение суммы элементов массива, расположенных до первого из минимальных элементов в массиве, к произведению элементов, расположенных после минимального. Если по какой-либо причине вычислить отношение не удается, выдать об этом сообщение с указанием причины.

2) Определить отношение произведения элементов массива, расположенных до последнего из максимальных элементов в массиве, если он не единственный, к сумме элементов, расположенных после максимального. Если по какой-либо причине вычислить отношение не удается, выдать об этом сообщение с указанием причины.

3) Найти среднее арифметическое элементов массива, расположенных между максимальным элементом и минимальным элементом (первыми по порядку, если их несколько). Если по какой-либо причине вычислить среднее арифметическое не удается, выдать об этом сообщение с указанием причины.

4) Определить количество элементов массива, расположенных между максимальным элементом и минимальным элементом (первыми по порядку, если их несколько). Если по какой-либо причине количество элементов определить не удается, выдать об этом сообщение с указанием причины.

5) Найти произведение всех элементов массива, предшествующих первому нулевому элементу. Если по какой-либо причине вычислить произведение не удается, выдать об этом сообщение с указанием причины.

6) Найти произведение положительных элементов массива, предшествующих первому отрицательному элементу. Если по какой-либо причине вычислить произведение не удается, выдать об этом сообщение с указанием причины.

7) Найти среднее арифметическое отрицательных элементов массива, предшествующих первому положительному элементу. Если по какой-либо причине вычислить среднее арифметическое не удается, выдать об этом сообщение с указанием причины.

8) Найти произведение положительных элементов массива, следующих после первого нулевого элемента. Если по какой-либо причине вычислить произведение не удается, выдать об этом сообщение с указанием причины.

9) Найти среднее арифметическое элементов, располо-

женных между первым и вторым нулевыми элементами. Если по какой-либо причине вычислить среднее арифметическое элементов не удается, выдать об этом сообщение с указанием причины.

10) Найти наименьший по абсолютной величине элемент среди элементов, расположенных между первым и вторым нулевыми элементами. Если по какой- либо причине найти такой элемент не удается, выдать об этом сообщение с указанием причины.

11) Определить количество элементов массива, расположенных между максимальным и «центральным» элементами массива (предполагается, что число элементов - нечётное и максимальный - единственный). Если по какой-либо причине количество элементов определить не удается, выдать об этом сообщение с указанием причины.

12) Определить произведение элементов массива, расположенных между максимальным и «центральным» элементами массива (предполагается, что число элементов - нечетное и максимальный единственный). Если по какой-либо причине вычислить произведение не удается, выдать об этом сообщение с указанием причины.

13) Найти отношение минимального элемента массива к максимальному среди элементов, предшествующих первому нулевому элементу. Если по какой- либо причине вычислить отношение не удается, выдать об этом сообщение с указанием причины.

14) Найти отношение суммы отрицательных элементов массива к минимальному элементу массива. Если по какой-либо причине вычислить отношение не удается, выдать об этом сообщение с указанием причины.

15) Найти отношение максимального элемента массива к произведению положительных элементов. Если по какой-либо причине вычислить отношение не удается, выдать об этом сообщение с указанием причины.

16) Определить отношение произведения элементов массива, расположенных до первого из минимальных элементов в массиве, к сумме элементов, расположенных после минимального. Если по какой-либо причине вычислить отношение не удается, выдать об этом сообщение с указанием причины.

17) Определить отношение суммы элементов массива, расположенных до последнего из максимальных элементов в массиве, если он не единственный, к произведению элементов, расположенных после максимального. Если по какой-либо причине вычислить отношение не удается, выдать об этом сообщение с указанием причины.

18) Найти сумму элементов массива, расположенных между максимальным элементом и минимальным элементом (первыми по порядку, если их несколько). Если по какой-либо причине вычислить сумму не удается, выдать об этом сообщение с указанием причины.

19) Найти произведение элементов массива, расположенных между максимальным элементом и минимальным элементом (первыми по порядку, если их несколько). Если по какой-либо причине вычислить произведение не удается, выдать об этом сообщение с указанием причины.

20) Найти произведение отрицательных элементов массива, предшествующих первому положительному элементу. Если по какой-либо причине вычислить произведение не удается, выдать об этом сообщение с указанием причины.

21) Найти среднее арифметическое элементов массива, предшествующих первому нулевому элементу. Если по какой-либо причине вычислить среднее арифметическое не удается, выдать об этом сообщение с указанием причины.

22) Найти среднее арифметическое положительных элементов массива, предшествующих первому отрицательному элементу. Если по какой-либо причине вычислить среднее арифметическое не удается, выдать об этом сообщение с указанием причины.

23) Найти произведение отрицательных элементов массива, следующих после первого положительного элемента. Если по какой-либо причине вычислить произведение не удается, выдать об этом сообщение с указанием причины.

24) Найти произведение элементов, расположенных между первым и вторым нулевым элементами. Если по какой-либо причине вычислить произведение не удается, выдать об этом сообщение с указанием причины.

25) Найти наибольший по абсолютной величине элемент среди элементов, расположенных между первым и вторым нулевыми элементами массива. Если по какой-либо причине найти такой элемент не удается, выдать об этом сообщение с указанием причины.

Задание 2. Проверка состояния вектора

1) Проверить, есть ли в массиве нулевые элементы.
2) Проверить, есть ли в массиве положительные элементы.
3) Проверить, есть ли в массиве отрицательные элементы.
4) Проверить, есть ли в массиве элементы, равные задан-

ному значению.

5) Проверить, есть ли в массиве элементы больше 12.

6) Проверить, содержится ли в массиве чётное количество нулевых элементов.

7) Проверить, упорядочены ли элементы по возрастанию.

8) Проверить, упорядочены ли элементы по неубыванию.

9) Проверить, упорядочены ли элементы по невозрастанию их абсолютных величин.

10) Проверить, есть ли в массиве несколько элементов с максимальным значением.

11) Проверить, содержатся ли в массиве положительные элементы.

12) Проверить, есть ли в массиве одинаковые элементы.

13) Проверить, есть ли в массиве элементы, равные заданному значению.

14) Проверить, все ли элементы массива положительны.

15) Проверить, все ли элементы массива отрицательны.

16) Проверить, все ли элементы массива нулевые.

17) Проверить, все ли элементы массива имеют одинаковые значения.

18) Проверить, все ли отрицательные элементы вектора расположены перед элементом равным 0.

19) Проверить, содержится ли в массиве нечётное количество нулевых элементов.

20) Проверить, упорядочены ли элементы по убыванию.

21) Проверить, упорядочены ли элементы по невозрастанию.

22) Проверить, упорядочены ли элементы по неубыванию их абсолютных величин.

23) Проверить, есть ли в массиве несколько элементов с минимальным значением.

24) Проверить, все ли элементы вектора равны 0 или 1.

25) Проверить, все ли элементы в массиве равны нулю.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Поясните назначение методов Aggregate, All, Any, Where, Select.
2. В чем разница между записями var n = m.First() и var n2 = m.First(a=>a>0)?
3. Что такое лямбда-выражение?
4. Что означает запись v.Where(a=>a<0).Select(a=>a*a)?

ЛАБОРАТОРНАЯ РАБОТА 4
ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ LINQ ДЛЯ РАБОТЫ С КЛАССАМИ

Цель работы: исследовать особенности использования технологии LINQ для работы с классами.

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Пусть имеется список граждан

```
List<Grajdanin> list = new List<Grajdanin>()
{
new Grajdanin {Fam = "Иванов", Nam = "Иван", Country ="Russia"},
new Grajdanin {Fam = "Петров", Nam = "Петр", Country ="Russia"},
new Grajdanin {Fam = "Braid", Nam = "Aux", Country ="Germany"}
};
```

Рассмотрим примеры использования технологии LINQ на этом списке.

Получить список граждан России можно одним из двух способов

(синтаксис запросов):

```
var results = from g in list where g.Country == “Russia” select g;
```

(синтаксис методов):

```
var results = list.Where(g.Country == “Russia”);
```

Выше определенная запись означает: для каждого элемента, условно названного g, списка list, для которого country == “Russia” выбираем целиком элемент g. Иначе говоря, выбираем объекты g класса Grajdanin из списка list, для которых Country == “Russia”.

Отсортируем список по фамилии

(синтаксис запросов):

```
var results = from g in list where g.Country == “Russia” orderby g.Fam  select g;
```

(синтаксис методов):

```
var results = list.Where(g.Country == “Russia”).OrderBy(g=>g.Fam);
```

Данный список можно вывести в Label

```
label1.Text = "";
```

```
foreach (var g in results)
        label1.Text += "\n" + g.ToString();
```

Для отображения списка results в dataGridView необходимо записать следующий код

```
BindingSource bindingSource = new BindingSource();
bindingSource.DataSource = results;
dataGridView1.AutoGenerateColumns = true;
dataGridView1.DataSource = bindingSource;
bindingNavigator1.BindingSource = bindingSource;
```

Список не будет иметь возможности добавлять элементы. Для возможности изменения количества элементов в коллекции необходимо преобразовать коллекцию в список с помощью метода ToList():

```
var results = (from g in list where g.Country == "Russia" orderby g.Fam  select g).ToList();
```

или в два этапа:

```
var r = from g in list where g.Country == "Russia" orderby g.Fam  select g;
var results = r.ToList();
```

Возможность добавления элементов в коллекцию имеется благодаря не только преобразованию коллекции в список, но и наличию конструктора по умолчанию, поскольку при добавлении элемента в список создается объект через вызов конструктора по умолчанию.

В выше описанном примере фильтруются элементы, удовлетворяющие указанным требованиям, но извлекаются все поля. В результате находятся элементы типа Grajdanin

Выбор только части полей:

(синтаксис запросов):

```
var results = (from g in list where g.Country == "Russia" orderby g.Fam  select new {g.Fam, g.Nam}.ToList();
```

(синтаксис методов):

```
var results = list.Where(g.Country == "Russia"). OrderBy(g=>g.Fam). Select (g=>new {g.Fam, g.Nam}).ToList();
```

Выбор всех фамилий по одному разу:

```
var results = list..OrderBy(g=>g.Fam).Select (g=>new {g.Fam }).Distinct().ToList();
```

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Ответьте на вопросы приложения WDATA.
2. Создайте список объектов с возможностью добавления, редактирования и удаления в элементе управления DataGridView.

3. Осуществите выборку объектов по требуемому свойству и сортировку элементов списка. Форма может иметь вид, представленный на рис. 1.

ВАРИАНТЫ ИНДИВИДУАЛЬНЫХ ЗАДАНИЙ

Вариант 1. Разработать класс CWorker (Сотрудник), содержащий свойства PersonID (Табельный номер сотрудника), Family (Фамилия сотрудника), Birth (Дата рождения сотрудника).

Для списка объектов этого класса:

- отсортировать по табельному номеру сотрудника;
- вычислить количество сотрудников с указанной фамилией;
- отфильтровать сотрудников с возрастом в указанном диапазоне;
- вычислить средний возраст этих сотрудников.

Вариант 2. Разработать класс CJobless (Безработный), содержащий поля JoblessID (регистрационный номер безработного), LastName (Фамилия безработного), FirstName (Имя безработного), Age (возраст безработного).

Для списка объектов этого класса:

- отсортировать по регистрационному номеру безработного;
- вычислить возраст и определить фамилию самого молодого безработного;
- отфильтровать безработных по указанным фамилии и имени;
- вычислить средний возраст безработных.

Вариант 3. Разработать класс CDocument (Документ), содержащий поля: Serial (серия документа), Number (номер документа), Date (Дата составления), Who (кем составлен)

Для списка объектов этого класса:

- отсортировать документы по дате составления;
- вывести список документов в указанном диапазоне дат составления;
- отфильтровать документы по указанной серии;
- определить количество документов, составленных указанным исполнителем.

Вариант 4. Разработать класс CZakazchik (Заказчик), содержащий поля: IdZakazchika (Идентификатор заказчика), Phone (телефон заказчика), Address (адрес заказчика).

Для списка объектов этого класса:

- отсортировать заказчиков по идентификатору;
- вывести список заказчиков с указанным адресом;
- отфильтровать заказчиков по указанным первым трем

цифрам номера телефона.

Вариант 5. Разработать класс CHoz (Владелец автомобиля), содержащий поля: Family (фамилия владельца), Number (номер паспорта владельца).

Для списка объектов этого класса:

- отсортировать владельцев по фамилии;
- вывести список владельцев с указанной фамилией;
- отфильтровать владельцев по указанным первым двум цифрам номера паспорта;
- вывести список номеров всех паспортов.

Вариант 6. Разработать класс CRoute (Маршрут), содержащий поля: RouteID (идентификатор маршрута), RouteName (название маршрута), Period (срок пребывания), Cost (стоимость путевки).

Для списка объектов этого класса:

- отсортировать маршруты по названию;
- вывести список маршрутов с указанным интервалом периода пребывания;
- отфильтровать маршруты с ценой меньше указанной.
- вывести количество и среднюю цену этих маршрутов.

Вариант 7. Разработать класс CDoctor (Врач), содержащий поля: RouteID (идентификатор врача), Family (фамилия врача), Specialnost (специальность врача), Room (номер кабинета).

Для списка объектов этого класса:

- отсортировать врачей по фамилии;
- вывести список врачей с указанной специальностью;
- определить количество врачей, принимающих в кабинетах с указанным интервалом номеров.
- вывести количество и список кабинетов, в которых принимают врачи.

Вариант 8. Разработать класс CCust (Арендатор), содержащий поля: CustomerID (ИНН арендатора), Customer (название арендатора), AddressCust (адрес арендатора), Room (номер кабинета), Chief (фамилия руководителя).

Для списка объектов этого класса:

- отсортировать арендаторов по фамилии;
- вывести список арендаторов с указанным адресом;
- определить количество арендаторов в кабинетах с указанным интервалом номеров.
- вывести количество и список кабинетов арендаторов.

Вариант 9. Разработать класс CVlad (Владелец), содержащий поля: FIO (ФИО владельца), Phone (телефон владельца), Re-

gistr (Регистрационный номер владельца).

Для списка объектов этого класса:

- отсортировать владельцев по фамилии;
- вывести список владельцев с указанным номером телефона;
- вывести список владельцев по указанным первым двум цифрам номера телефона;
- отфильтровать владельцев с регистрационным номером в указанном диапазоне.
- вывести количество этих владельцев.

Вариант 10. Разработать класс CTelephonNumber (Телефонный номер), содержащий поля: ID (идентификатор клиента), Family (фамилия клиента), PhoneAddress (адрес клиента), PhoneNumber (номер телефона), Value (Ежемесячная плата за телефон).

Для списка объектов этого класса:

- отсортировать телефонные номера по фамилии;
- вывести список телефонных номеров с указанным диапазоном ежемесячной платы;
- вывести список телефонных номеров по указанным первым двум цифрам номера телефона;
- вывести среднюю ежемесячную плату.

Вариант 11. Разработать класс CCount (Счет-фактура), содержащий поля: CountNumber (Номер счет-фактуры), Date (Дата выписки счет-фактуры), Value (Сумма к уплате), Cost (Стоимость приобретения).

Для списка объектов этого класса:

- отсортировать счет-фактуры по номеру;
- вывести список счет-фактур с указанным диапазоном номеров;
- вывести среднюю сумму к уплате и количество счет-фактур с указанным диапазоном стоимости приобретения;
- вывести счет-фактуры с указанной датой выписки и среднюю сумму к уплате по ним.

Вариант 12. Разработать класс **CRabotnik** (Работник автовокзала), содержащий поля: FIO (ФИО работника), Dolg (Должность), Zarplata (оклад).

Для списка объектов этого класса:

- отсортировать работников по фамилии;
- вывести список работников с указанным диапазоном зарплат;

- вывести среднюю зарплату работников с указанной должностью и количество этих работников;
- вывести работников с указанной начальной буквой фамилии;
- вывести список всех должностей.

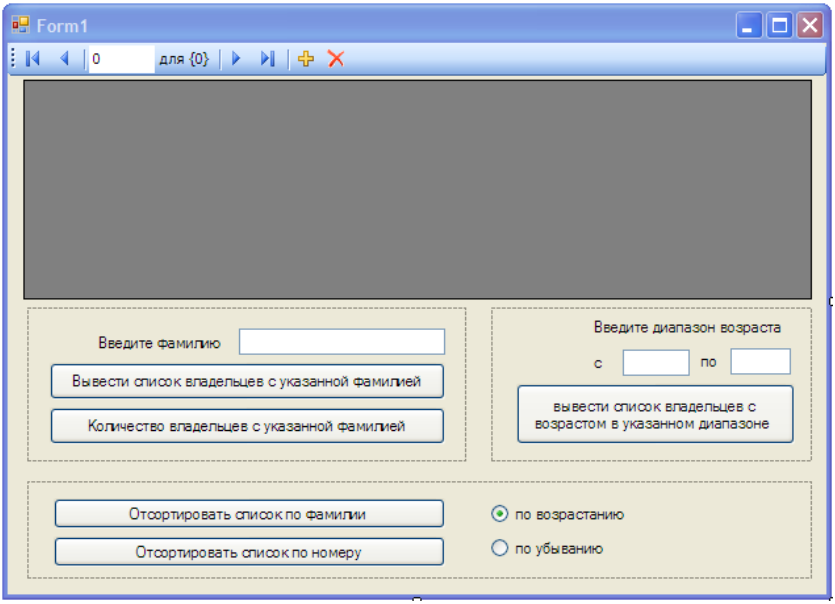


Рис. 1. Образец интерфейса программы

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как отсортировать список по указанному полю?
2. Какой метод служит для фильтрации списка?
3. Какие методы предназначены для определения количества, среднего и суммы?
4. Что означает запись

var results = list.Where(g.Country <= “Russia”)?

ЛАБОРАТОРНАЯ РАБОТА 5
ИСПОЛЬЗОВАНИЕ COM-ТЕХНОЛОГИИ НА ЯЗЫКЕ C#

Цель работы: Исследовать COM-технологию.

Краткие теоретические сведения

Работа с приложениями в операционной системе Windows через программное обеспечение называется автоматизацией. Для автоматизации используется технология COM.

В данной работе рассмотрим пример использования COM-технологии на примере автоматизации работы с приложением Microsoft Word, поддерживающей COM-технологию.

Основными элементами приложения Word являются:

Documents — набор всех документов (коллекция). Каждый документ в коллекции имеет номер;

ActiveDocument — активный документ приложения;

Selection — выделенная область документа или место, в котором установлен курсор.

Элементы документа. Основными элементами документа являются:

PageSetup — свойство документа (составной объект). Позволяет задавать ширину и высоту страницы, отступы и т.д.;

Characters — набор символов документа (коллекция);

Paragraphs — набор абзацев документа (коллекция);

Words — набор слов документа (коллекция). Словом в документе Word является алфавитно-буквенные символы, разделенные знаками препинания или пробелами и знаки препинания;

Tables — набор таблиц документа (коллекция).

Элементы абзаца Paragraph. Абзац имеет следующие элементы:

Space1 — метод устанавливает одинарный интервал;

Space15 — метод устанавливает полуторный интервал;

Space2 — метод устанавливает двойной интервал;

Alignment — выравнивание абзаца (свойство);

Borders — рамка вокруг абзаца (составной объект).

Например, для изменения интервала 1-го абзаца активного документа на двойной достаточно написать

W.ACTIVEDOCUMENT.PARAGRAPHS[1].SPACE2();

СВОЙСТВО RANGE. **СВОЙСТВО** RANGE **ЗАНИМАЕТ ОСОБОЕ МЕСТО. СВОЙСТВО** RANGE **ИМЕЕТСЯ У ДОКУМЕНТА,**

АБЗАЦА И ДРУГИХ СОСТАВНЫХ ЧАСТЕЙ ДОКУМЕНТА, КРОМЕ WORD **И** CHARACTER**. ОНО ВОЗВРАЩАЕТ ФРАГМЕНТ ДОКУМЕНТА, ДЛЯ КОТОРОГО ВЫЗЫВАЕТСЯ, И ПОЗВОЛЯЕТ ОПЕРИРОВАТЬ ЭТИМ ФРАГМЕНТОМ.**

СОДЕРЖИТ СЛЕДУЮЩИЕ ПОЛЯ:

BOLD **— ПОЛУЖИРНОЕ НАЧЕРТАНИЕ (СВОЙСТВО);**

ITALIC **— КУРСИВНОЕ НАЧЕРТАНИЕ (СВОЙСТВО);**

FONT **— ШРИФТ (СОСТАВНОЙ ОБЪЕКТ);**

COPY **— КОПИРОВАТЬ (МЕТОД);**

CUT **— ВЫРЕЗАТЬ (МЕТОД);**

DELETE **— УДАЛИТЬ (МЕТОД);**

CHARACTERS — **НАБОР СИМВОЛОВ (КОЛЛЕКЦИЯ);**

WORDS **— НАБОР СЛОВ (КОЛЛЕКЦИЯ);**

PARAGRAPHS **— НАБОР АБЗАЦЕВ (КОЛЛЕКЦИЯ);**

TABLES **— НАБОР ТАБЛИЦ (КОЛЛЕКЦИЯ);**

INSERTPARAGRAPH **— ЗАМЕНИТЬ АБЗАЦЕМ;**

INSERTPARAGRAPHAFTER **— ДОБАВИТЬ АБЗАЦ ПОСЛЕ ФРАГМЕНТА;**

INSERTPARAGRAPHBEFORE **— ДОБАВИТЬ АБЗАЦ ПЕРЕД ФРАГМЕНТОМ;**

INSERTAFTER **— ДОБАВИТЬ ТЕКСТ ПОСЛЕ ФРАГМЕНТА;**

INSERTBEFORE **— ДОБАВИТЬ ТЕКСТ ПЕРЕД ФРАГМЕНТОМ;**

CHECKGRAMMER **— ПРОВЕРИТЬ ГРАММАТИКУ;**

START **— НАЧАЛО ФРАГМЕНТА;**

End — конец фрагмента.

НАПРИМЕР,

W.ACTIVEDOCUMENT.PARAGRAPHS[2].RANGE.DELETE();

УДАЛЯЕТ ВТОРОЙ ПАРАГРАФ АКТИВНОГО ДОКУМЕНТА.

Порядок использования технологии COM на примере Word.

Для работы с Word из программы, написанной на языке C# необходимо добавить ссылку на COM-объект Microsot Word xx Object Library в окне обозреватель решений проекта в ветку «Ссылки».

Далее необходимо добавить пространство имен Word

using Word;

Для работы с Word используется объект типа

Word.Application. Для этого добавляется в класс формы поле

Word.Application w = null;

Поскольку многие методы для работы с Word требуют большое количество параметров, которые должны браться по умолчанию, будем их значения задавать объектом missing, котовый опишем в виде поля формы:

Object missing = System.Reflection.Missing.Value;

Напишем программу, которая преобразует строку к верхнему регистру.

Форма будет иметь строку ввода для ввода исходной строки и 2 кнопки. Первая кнопка предназначена для выполнения преобразований со строкой и вывода результатов в Word. Вторая кнопка предназначена для сохранения сгенерированного в Word документа на диске.

```
private void button1_Click(object sender, EventArgs e)
{
if (w==null)
w = new Word.Application();
string s = textBox1.Text;
string s2 = s.ToUpper(); //Преобюразуем к верхнему регистру
w.Documents.Add(ref missing, ref missing, ref missing, ref missing);
w.Visible = true;
w.Options.Overtype = false;
w.Selection.TypeText("Исходная строка  - " + s);
w.Selection.TypeText("\n");
w.Selection.TypeText("Строка в верхнем регистре  - " + s2);
}

private void button2_Click(object sender, EventArgs e)
{
try
{
saveFileDialog1 = new SaveFileDialog();
saveFileDialog1.Filter = "Документ doc|*.doc";
if (saveFileDialog1.ShowDialog(this) == DialogResult.OK)
{
Object fn = saveFileDialog1.FileName;
w.ActiveDocument.SaveAs(ref fn, ref missing, ref missing,
ref missing, ref missing, ref missing, ref missing,
ref missing, ref missing, ref missing, ref missing,
```

```
ref missing, ref missing, ref missing, ref missing,
ref missing);
w.Quit(ref missing, ref missing, ref missing);
w = null;
}
}
catch
{
MessageBox.Show(this, "Word открыт не был. Сохранение выполнено быть не может.");
}
}
```

Конструкция **try** … **catch** предназначена для отлавливания исключений. Например, при нажатии на кнопку сохранить при отсутствующем документе или не запущенном Word произойдет ошибка выполнения программы. В случае ошибки приложение не будет аварийно завершено, а появится диалоговое окно с сообщением.

Задания для выполнения работы

1. Найдите ситуации, при которых может произойти ошибка выполнения программы, кроме тех, которые уже учтены в программе, и добавьте необходимые обработчики.
2. Доработайте программу из лабораторной работы 1 с целью вывода в Word информации об объектах в списке в виде таблицы.

ЛАБОРАТОРНАЯ РАБОТА 6
РЕКУРСИВНЫЕ АЛГОРИТМЫ

Цель работы: научиться составлять рекурсивные алгоритмы.

Краткие теоретические сведения

Рекурсивный алгоритм — алгоритм, в описании которого прямо или косвенно содержится обращение к самому себе. В технике процедурного программирования данное понятие распространяется на функцию, которая реализует решение отдельного блока задачи посредством вызова из своего тела других функций, в том числе и себя самой. Если при этом на очередном этапе работы функция организует обращение к самой себе, то такая функция является рекурсивной.

Рекурсивный вызов процедуры мало чем отличается от вызова другой функции. Выполняется тот же код, но с другими значениями параметров и локальных переменных.

Важным для понимания идеи рекурсии является то, что в подобных подпрограммах можно выделить две серии шагов. Первая серия – это шаги рекурсивного погружения подпрограммы в себя до тех пор, пока выбранный параметр не достигнет граничного значения. Это важное требование должно выполняться, чтобы функция не создала бесконечную последовательность вызовов самой себя. Вторая серия – это шаги рекурсивного выхода до тех пор, пока выбранный параметр не достигнет конечного значения.

Рассмотрим примеры реализации рекурсий.

Пример реализации прямой рекурсии для вычисления факториала может выглядеть так:

```
Int64 Factorial(Int64 num)
        {
          return num * Factorial (num – 1);
}
```

Здесь в теле функции *Factorial* осуществляется вывозов самой функции *Factorial*. Такая рекурсия будет осуществляться бесконечно, поскольку отсутствует условие выхода из рекурсии.

Из этого примера видно, что очевидная опасность рекурсии заключается в бесконечной рекурсии. Если неправильно построить алгоритм, то функция может пропустить условие остановки рекурсии и выполняться бесконечно.

Для нахождения условия остановки необходимо выделить базу рекурсии. Выделение базы рекурсии предполагает нахож-

дение в решаемой задаче тривиальных случаев, результат для которых очевиден и не требует проведения расчетов. Верно найденная база рекурсии обеспечивает завершенность рекурсивных обращений, которые в конечном итоге сводятся к базовому случаю.

Функция также может вызывать себя бесконечно, если условие остановки не прекращает все возможные пути рекурсии. В следующей ошибочной версии функции Factorial2, функция будет бесконечно вызывать себя, если входное значение — не целое число, или если оно меньше 0.

```
Int64 Factorial2 (Int64 num)
{
  if (num == 0)
          Factorial2 = 1
  else
          Factorial2 = num * Factorial2 (num-1)
  return Factorial2;
}
```

Для устранения бесконечной рекурсии необходимо сделать следующее. Вначале функция *Factorial* должна проверять, что число меньше или равно 0. Факториал для чисел меньше нуля не определен, но это условие проверяется для подстраховки. Если бы функция проверяла только условие равенства числа нулю, то для отрицательных чисел рекурсия была бы бесконечной. Если входное значение меньше или равно 0, функция возвращает значение 1. В остальных случаях, значение функции равно произведению входного значения на факториал от входного значения, уменьшенного на единицу. То, что эта рекурсивная функция, в конце концов, остановится, гарантируется двумя фактами. Во-первых, при каждом последующем вызове, значение параметра *num* уменьшается на единицу. Во-вторых, когда *num* становится меньше либо равным 0, функция останавливает рекурсию.

Хотя рекурсия и может упростить понимание некоторых проблем, люди обычно не мыслят рекурсивно. Они обычно стремятся разбить сложные задачи на задачи меньшего объема, которые могут быть выполнены последовательно одна за другой до полного завершения. Для того чтобы думать рекурсивно, нужно разбить задачу на подзадачи, которые затем можно разбить на подзадачи меньшего размера. В какой-то момент подзадачи становятся настолько простыми, что могут быть выполнены непосредственно. Когда завершится выполнение подзадач, большие

подзадачи, которые из них составлены, также будут выполнены. Исходная задача окажется выполнена, когда будут все выполнены образующие ее подзадачи.

Рассмотрим примеры решения задач с использованием рекурсивных функций.

Пример 1. Определить является ли строка палиндромом с использованием рекурсии.

Строка является палиндромом, если первый символ строки равен последнему символу строки и палиндромом является строка без первого и последнего символа. Такое определение палиндрома является рекурсивным.

Окончание рекурсии гарантируется уменьшением длины строки на 2 символа на каждой рекурсии и выходом из рекурсии при уменьшении строки до 0 символов.

Причем при очередном обращении к рекурсивной функции строка будет короче на 2 символа.

Когда получен результат по подстроке, можно определить является ли вся строка палиндромом: для этого необходимо, чтобы подстрока была палиндромом и первый и последний символы строки были равны друг другу.

Равенство первого и последнего символов строки определяется выражением *s*[0] == *s*[*s*.Length - 1].

Подстрока без первой и последней букв определяется выражением

s.Substring(1, *s*.Length – 2);

Рекурсивная функция возвращает результат в виде логической величины *f* равной **true** – если подстрока является палиндромом и **false** – в противном случае.

```
        private void button1_Click(object sender, EventArgs
  e)
{
        string s = "aceeca";
        bool b = palindrom(s);
        Text = b.ToString();
}

private bool palindrom(string s)
{
        if (s.Length == 0) return true;
        bool f = palindrom(s.Substring(1, s.Length - 2));
        return f & (s[0] == s[s.Length - 1]);
```

```
}
```

В выше приведенном примере имеется ошибка. Пример не учитывает тот факт, что строка может содержать нечетное количество символов и ее длина не станет равной 0.

Пример 2. Возвести число *a* в степень *n* с использованием рекурсии.

На каждом шаге рекурсии будем уменьшать степень в 2 раза, то есть *a* в степени *n* запишем в виде $a^n = a^{n/2} \cdot a^{n/2}$. Однако для нечетной степени $a^n = a \cdot a^{(n-1)/2} \cdot a^{(n-1)/2}$. Таким образом, степень будет уменьшаться на каждом шаге в 2 раза и при уменьшении степени до 1 задача становится тривиальной.

Программа будет иметь следующий вид

```
private void button1_Click(object sender, EventArgs e)
{
        int a = 3;
        int n = 33;
        Int64 b = V_Stepen(a, n);
        Text = b.ToString();
}

private Int64 V_Stepen(int a, int n)
{
        if (n==1) return a;
        Int64 f;
        if (n%2==0)
                f = V_Stepen (a, n / 2) * V_Stepen (a, n / 2);
        else
                f = a * V_Stepen (a, n / 2) * V_Stepen (a, n /
          2);
        return f;
}
```

Данный алгоритм является неоптимальным по быстродействию.

Индивидуальные задания

Вариант 1. Запрограммировать рекурсивный алгоритм вычисления числа сочетаний, используя рекуррентную формулу $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$, и учитывая, что $C_n^1 = n$ и $C_n^n = 1$.

Вариант 2. Дано целое неотрицательное число n. Записать

цифры данного числа в элементы массива, используя рекурсию.

Вариант 3. Дано целое неотрицательное число n. Преобразовать его в двоичное число.

Вариант 4. Запрограммировать рекурсивный алгоритм вычисления квадрата натурального числа, используя рекуррентное соотношение

$$n^2 = \begin{cases} 1, & \textit{если } n = 1, \\ (n-1)^2 + n + n - 1, & \textit{если } n > 1. \end{cases}$$

Вариант 5. Запрограммировать рекурсивный алгоритм перемножения двух натуральных чисел, используя рекуррентное соотношение

$$a \cdot b = \begin{cases} b, & \textit{если } a = 1, \\ (a-1) \cdot b + b, & \textit{если } a > 1. \end{cases}$$

Вариант 6. Числа Фибоначчи u_0, u_1, u_2, … определяются следующим образом: $u_0 = 0$, $u_1 = 1$, $u_n = u_{b-1} + u_{n-2}$, при $n = 2,3,4,...$ Написать программу вычисления u_n для данного целого неотрицательного числа n, включающую рекурсивную функцию, которая основана на непосредственном использовании соотношения $u_n = u_{b-1} + u_{n-2}$.

Вариант 7. Даны неотрицательные целые числа n, m; вычислить $A(n,m)$, где

$$A(n,m) = \begin{cases} m+1, & \text{если } n = 0, \\ A(n-1,1), & \text{если } n \neq 0, m = 0, \\ A(n\text{-}1,A(n,m\text{-}1)), & \text{если } n > 0, m > 0 \end{cases}$$

Написать программу, включающую рекурсивную функцию.

Вариант 8. Вычислить рекурсивно сумму чисел от 1 до *n* по формуле sum(*n*)=sum(*n*-1)+*n*, где *n* – целое положительное число.

Вариант 9. Вычислить рекурсивно сумму чисел от 1 до *n*, по формуле sum(*n*) = sum(*n*,2)+sum(*n*-1,2), где *n* – целое положительное число, sum(*n*,*a*)=*n*+sum(*n*-*a*, *a*), sum(1,2)=1, sum(2,2)=2.

Вариант 10. Создайте функцию, подсчитывающую сумму элементов массива по следующему алгоритму: массив делится пополам, подсчитываются и складываются суммы элементов в каждой половине. Сумма элементов в половине массива

подсчитывается по тому же алгоритму, то есть снова путем деления пополам. Деления происходят, пока в получившихся кусках массива не окажется по одному элементу и вычисление суммы, соответственно, не станет тривиальным.

Вариант 11. Дано натуральное число *N*. Выведите все его цифры по одной, в обратном порядке, разделяя их пробелами или новыми строками. При решении этой задачи нельзя использовать строки, списки, массивы, циклы. Разрешена только рекурсия и целочисленная арифметика.

Вариант 12. Дано натуральное число *N*. Выведите все его цифры по одной, в обычном порядке, разделяя их пробелами или новыми строками. При решении этой задачи нельзя использовать строки, списки, массивы, циклы. Разрешена только рекурсия и целочисленная арифметика.

Вариант 13. Дано натуральное число $n>1$. Проверьте, является ли оно простым. Программа должна вывести слово **YES**, если число простое и **NO**, если число составное. Алгоритм должен иметь сложность O($log\ n$). Понятно, что задача сама по себе нерекурсивна, т.к. проверка числа *n* на простоту никак не сводится к проверке на простоту меньших чисел. Поэтому нужно добавить еще один параметр рекурсии: делитель числа, и именно по этому параметру делать рекурсию.

Вариант 14. Дано число *n*, десятичная запись которого не содержит нулей. Получите число, записанное теми же цифрами, но в противоположном порядке. При решении этой задачи нельзя использовать циклы, строки, списки, массивы, разрешается только рекурсия и целочисленная арифметика. Фунция должна возвращать целое число, являющееся результатом работы программы, выводить число по одной цифре нельзя.

Вариант 15. Дано натуральное число *N*. Выведите слово **YES**, если число N является точной степенью двойки, или слово **NO** в противном случае. Операцией возведения в степень пользоваться нельзя.

Контрольные вопросы

1. Исправьте алгоритм из примера 1 таким образом, чтобы он учитывал возможность определения палиндрома для строки с нечетным количеством символов.

2. Какие изменения необходимо сделать в алгоритме из примера 2, чтобы уменьшить время вычислений?

3. Как парограммно определить количество вызовов

функции V_Stepen для неоптимального и оптимального варианта?

4. Почему в формуле

f = *a* * *V_Stepen* (*a*, *n* / 2) * *V_Stepen* (*a*, *n* / 2);

вместо (*n*-1)/2 записано *n*/2? Является ли это ошибкой?

ЛАБОРАТОРНАЯ РАБОТА 7
РЕАЛИЗАЦИЯ РЕКУРСИВНЫХ АЛГОРИТМОВ НА ПРИМЕРЕ ГОЛОВОЛОМКИ «ХАНОЙСКИЕ БАШНИ»

Цель работы: изучение реализации рекурсивных алгоритмов на примере головоломки «Ханойские башни».

Краткие теоретические сведения

Ханойские Башни — это головоломка, которую в 1883 г. придумал французский математик Эдуард Люка. Суть головоломки в следующем: есть три стержня и восемь дисков разных диаметров, вначале все диски собраны на одном стержне так, что меньшие диски лежат на больших. Люка предлагал переложить все диски с первого стержня на третий, используя второй. При этом следует соблюдать следующее правило: диски можно перекладывать с одного стержня на другой, при этом нельзя класть диск поверх диска меньшего радиуса; за один ход можно переносить лишь один диск,

Если бы в пирамиде был только 1 диск, то решение очевидно.

Если дисков 2, то переложим сначала меньший диск на второй стержень, затем перенесем второй диск на третий стержень, а за ним первый диск.

При большем количестве дисков для того чтобы перенести самый большой диск, нужно сначала перенести все диски кроме последнего на второй стержень, потом перенести самый большой на третий, после чего останется перенести все остальные диски со второго на третий. Задачу о переносе N-1 диска мы решаем аналогично, только поменяем стержни местами (N-2 диска перенесем со второго на первый, оставшийся диск перенесем со второго на третий и затем все диски с первого на третий). Как видите, задачка разрешима. И правда, ведь задачу о N-1 дисков мы сведем к задаче о N-2 дисков, ту в свою очередь к N-3 дискам, и так вплоть до 1 диска. Этот метод легко программируется с помощью рекурсии (рис. 1).

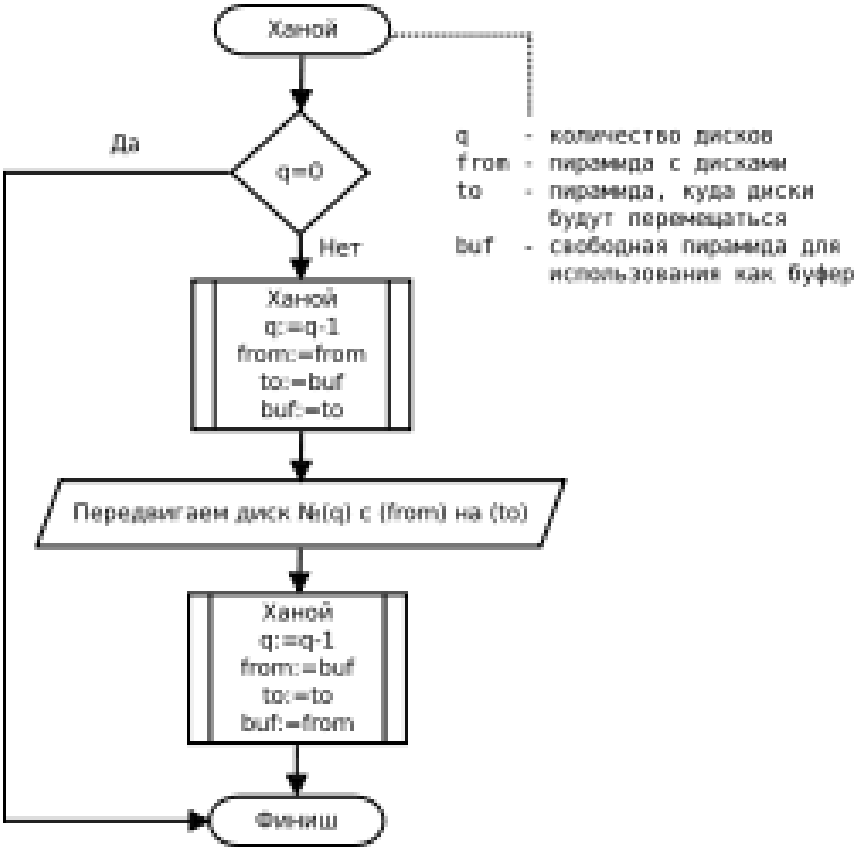


Рис. 1. Схема алгоритма головоломки «Ханойские башни»

Присмотримся повнимательнее к процессу переноса дисков. Для этого перенумеруем их в порядке возрастания: первый — самый маленький и т. д. Теперь начнем переносить диски в соответствии с алгоритмом и будем записывать номера переносимых дисков:

12131214121312151213121 41…

Каждый нечетный перенос — это перенос первого диска. Еще одно наблюдение: расположим стержни в вершинах равностороннего треугольника и понаблюдаем за движением первого диска. Оказывается, что он движется по кругу в одну и ту же сторону; при этом, если начальное количество дисков четно, то по часовой стрелке, а если нечетно — против. Эти наблюдения позволили в 1961 году сформулировать следующий алгоритм пере-

носа дисков: сначала переносим первый диск, потом не первый, затем снова первый, потом не первый и т. д., причем первый диск переносится в одном направлении: по часовой стрелке в случае четного количества дисков и против часовой стрелки, в случае их нечетного количества.

Заметим, что указание «переносится не первый диск» полностью определяет, какой диск и куда следует переносить в данный момент, поскольку меньший диск всегда кладется на больший.

Для хранения списка дисков на каждом стержне будем использовать стек.

Поскольку диски помещаются и извлекаются со стержней по правилу «первый вошел — последний вышел», для хранения дисков на каждом стержне предлагается использовать стек. Стеков должно быть фиксированное количество (3 — по количеству стержней).

Stack<int>[] h;

Для запуска процесса перестановки дисков на форму установим кнопку Button button1.

При нажатии на кнопку button1 инициализируем диски на стержнях и запустим процесс их перестановки.

```
private void button1_Click(object sender, EventArgs e)
{
        h = new Stack<int>[3];
        h[0] = new Stack<int>();
        h[1] = new Stack<int>();
        h[2] = new Stack<int>();
        h[0].Push(8);
        h[0].Push(7);
        h[0].Push(6);
        h[0].Push(5);
        h[0].Push(4);
        h[0].Push(3);
        h[0].Push(2);
        h[0].Push(1);
        Hanoi(1, 3, 8);
}
```

Функция Hanoi имеет следующий заголовок

private void Hanoi(int from, int to, int cou)

и служит для перестановки дисков со стержня номер from на стержень номер to в количестве cou штук.

Данная задача является сложной, и поэтому будем ее упрощать путем уменьшения количества переставляемых дисков на 1 на каждом шаге и вызывая эту же функцию рекурсивно.

Определяем номер свободного стержня to2 и если необходимо переставить более 1 диска, то задачу упрощаем – переставляем на 1 диск меньше на свободную позиция to2, последний диск устанавливаем на нужную позицию to и затем переставляем все диски с позиции to2 на позицию to. Перестановка одного диска является элементарной операцией, которая выполняется командой

```
h[to - 1].Push(h[from - 1].Pop());
```

А перестановка с позиции to2 на позицию to является аналогичной задаче перестановки с позиции from на to2, но меньше ее на один шаг. Таким образом, наша задача завершится за конечное число шагов.

```
private void Hanoi(int from, int to, int c)
{
        int to2 = 6 - from - to; //to2 – номер промежуточной
  башни
        if (c > 1)
        {
                Hanoi(from, to2, c – 1);
                Hanoi (from, to, 1);
                Hanoi (to2, to, c – 1);
        }
        else
        {
                h[to - 1].Push(h[from - 1].Pop());
        }
}
```

Порядок выполнения работы

1. Создайте приложение Windows Forms в среде C#.
2. Напишите программу для решения головоломки «Ханойские башни» с учетом выше приведенных пояснений.
3. Запустите программу.
4. Поскольку программа не осуществляет вывод на экран, то проверьте ее путем выполнения программы в отладчике, используя средства отладки (точки останова, панель инструментов «отладка», «локальные переменные» и «контрольные значения»).
5. Запустите программу еще раз и проанализируйте «Стек

вызовов» при установке прерывания в строке:

int to2 = 6 – from – to;

Обратите внимание как исходная задача разбивается на более простые путем вызова рекурсивных функций.

Индивидуальные задания

1. Перенести диски с первого стержня на третий, так, чтобы первый диск ни разу не оказывался бы на втором стержне.
2. Перенести диски с первого стержня на третий, так, чтобы первый диск ни разу не оказывался бы на втором стержне.
3. Перенести диски с первого стержня на третий, так, чтобы n-й диск ни разу не оказывался бы на втором стержне.
4. Перенести диски с первого стержня на третий, если первоначально они лежали на первом стержне в произвольном порядке.
5. Перенести диски на третий стержень, если они расположены произвольно на всех трех стержнях.

Контрольные вопросы

1. Почему для хранения дисков используется стек, а не массив?
2. Почему для хранения дисков используется стек, а не список?
3. Поясните, почему to2 вычисляется по формуле to2 = 6 - from – to.
4. Поясните наличие «-1» в выражении h[to - 1].Push(h[from - 1].Pop());
5. Поясните код

```
h = new Stack<int>[3];
h[0] = new Stack<int>();
h[1] = new Stack<int>();
h[2] = new Stack<int>();
```

6. Поясните программный код

```
Hanoi(1, 3, 6);
```

7. По каким признакам можно определить, что метод Hanoi является рекурсивным?
8. Каким образом проводилась отладка программы?

ЛАБОРАТОРНАЯ РАБОТА 8
РЕАЛИЗАЦИЯ ГРАФИЧЕСКОГО ВЫВОДА НА ЯЗЫКЕ C#

Цель работы: исследовать возможность графического вывода на языке C#.

Краткие теоретические сведения

Для рисования на языке C# используется класс Graphics, который предоставляет методы для вывода объектов в устройстве отображения. Объект Graphics связан с конкретным контекстом устройства.

Объект Graphics можно получить путем вызова метода Control.CreateGraphics для объекта, который наследует из объекта System.Windows.Forms.Control, или путем обработки события Control.Paint элемента управления и обращения к свойству Graphics класса System.Windows.Forms.PaintEventArgs.

Используя объект Graphics, можно нарисовать различные фигуры и линии. К этим методам относятся DrawLine, DrawArc, DrawClosedCurve, DrawPolygon и DrawRectangle. Рисунки и значки можно также рисовать с помощью методов DrawImage и DrawIcon, соответственно.

Приведем несколько методов класса Graphics:

— Clear — Очищает всю поверхность рисования и выполняет заливку поверхности указанным цветом фона.

— DrawClosedCurve — Строит замкнутую кривую.

— DrawCurve — Строит кривую.

— DrawEllipse — Рисует эллипс, определяемый ограничивающей структурой Rectangle.

— DrawIcon — Формирует изображение, представленное указанным объектом Icon.

— DrawImage — Рисует заданный объект Image в заданном месте, используя его исходный фактический размер.

— DrawLine — Проводит линию.

— DrawLines — Рисует набор сегментов линий.

— DrawPolygon — Рисует многоугольник.

— DrawRectangle — Рисует прямоугольник.

— DrawString — Рисует указанную текстовую строку в заданном месте.

— FillClosedCurve — Рисует закращенную замкнутую кривую.

— FillEllipse — Рисует закрашенный эллипс.

— FillRectangle — Рисует закрашенный прямоугольник.

Реализуем программу Ханойские башни с графическим выводом и использованием ООП. Для этого создадим класс стержень (Tower), который будет хранить информацию об одном стержне и может себя прорисовать на графическом устройстве.

Добавим в проект класс ***Tower***.

На каждом стержне находятся диски, помещенные в стек.

Стержень будет иметь следующие поля:

public Stack<**int**> h; — стек для хранения дисков.

Каждый диск имеет номер. В стеке будут храниться номера дисков. Диаметр диска будет пропорционален номеру. Причем, диск, помещенный на стержень первым, будет извлечен с него последним.

Каждый стержень будет иметь поле для хранения его номера:

public int nomer { **get**; **set**; } – свойство для чтения и записи номера стержня.

Для добавления дисков на стержень используется метод Add

```
public void Add(int i)
{
        h.Push(i);
}
```

где i – номер диска.

Для рисования добавим пространство имен System.Drawing

using System.Drawing;

Для реализации рисования добавим поле

Graphics g; — графическое устройство, на котором стержень будет себя прорисовывать.

Также класс будет иметь конструктор

```
public Tower(Graphics g, int nomer)
{
        this.g = g;
                this.nomer = nomer;
        h = new Stack<int>();
}
```

и свойства X и Y для получения, соответственно, координаты X и Y левого верхнего угла, стержня на графическом устройстве

```
public int X
{
        get { return 30 + (nomer – 1) * 200; }
```

} – свойство для хранения координаты X стержня,

public int Y

{

get { return 70; }

} – свойство для хранения координаты Y стержня.

Координата X определяется номером башни – чем больше номер, тем правее она расположена.

Для очистки холста, на котором рисуется стержень с дисками используется метод Clear()

```
private void Clear()
{
        g.FillRectangle(new SolidBrush(Color.Gray),
    new Rectangle(X, Y, 200, 200));
}
```

Для отрисовки башни используется метод Draw(), в котором очищается холст, а затем прорисовываются все диски в виде прямоугольников (вид сбоку).

```
public void Draw()
{
    Clear();
    for (int i=0; i<h.Count(); i++)
    {
        int a = h.ElementAt(i);
            Rectangle r = new Rectangle(X + 100 –
      a * 10, Y + 20 * (10 – h.Count() +  i), a * 20,
      20);
        g.FillRectangle(new SolidBrush(Color.Red), r);
    }
}
```

Для работы с классом Tower необходимо, создать объекты для хранения стержней в классе формы.

В форме опишем поле *c* для хранения стержней

Tower [] *c*;

И при нажатии на кнопку создадим графическое устройство *g*, создадим 3 стержня, заполним первый стержень 8 дисками, вызовем рекурсивную функцию *Hanoi* для запуска процесса перестановки.

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g;  //Для рисования стержня на форме
    g = this.CreateGraphics(); //Создаем область для
  рисования на форме
```

```
c = new Tower[3]  //создаем стержни …
{
        new Tower(this.g, 1), // … с номерами 1,
        new Tower(this.g, 2), // 2
        new Tower(this.g, 3)  // и 3
};
c[0].Add(8); //Добавляем диски на стержни
                        //Число означает номер диска
c[0].Add(7);
c[0].Add(6);
c[0].Add(5);
        c[0].Add(4);
        c[0].Add(3);
c[0].Add(2);
c[0].Add(1);
Hanoi(1, 3, 7);  //Запускаем алгоритм перестановки
                        //     со стержня 1 на 3 всех
                    восьми дисков
}
```

Функция *Hanoi* претерпела незначительные изменения с учетом графического вывода процесса перестановки и введения класса Tower.

```
private void Hanoi(int from, int to, int cou)
{
        for (Int64 i = 0; i < 4E6; i++);    //задержка
  для уменьшения скорости
                                        //
                                    отображения кадров
                                    анимации
    int to2 = 6 – from – to;
    if (cou > 1)  //если переставить надо более 1 диска
    {
        Hanoi(from, to2, cou – 1); //то задачу делим
   на 3 более простые
        Hanoi(from, to, 1);
        Hanoi(to2, to, cou – 1);
    }
    else  //если диск 1, то непосредственно перставляем
    {
        c[to – 1].h.Push(c[from – 1].h.Pop());
    }
    if (c != null)  //перед рисованием проверяем, есть ли
```

```
стержни
        {
                //рисуем 3 пирамиды
                c[0].Draw();
                c[1].Draw();
                c[2].Draw();
                }
}
```

Порядок выполнения работы

1. Открыть проект, созданный в лабораторной работе «Реализация рекурсивных алгоритмов на примере головоломки «Ханойские башни»».

2. Добавить класс Tower и переделать класс Form1 в соответствии с настоящими методическими указаниями.

3. Переделать метод Hanoi таким образом, чтобы программа была выполнена в более строгом объектно-ориентированном стиле. Для этого заголовок функции должен иметь следующий формат

private void Hanoi(Tower from, Tower to, int col).

4. Модификатор поля *h* в классе Tower изменить на private. Изменить программу, чтобы она компилировалась.

5. Для добавления на стержень (в стек *h*) дисков определить метод Push в классе Tower.

6. Для извлечения со стержня (из стека *h*) дисков определить метод Pop в классе Tower.

7. Поскольку Add и Push повторяют друг друга — убрать метод Add из класса Tower.

8. Нарисовать у каждой пирамиды из дисков столбик на который нанизаны диски.

9. Сделать обводку вокруг дисков.

10. Сделать самый маленький диск каждого стержня желтого цвета.

11. Сделать самый большой диск из всех красного цвета.

12. Добавьте в программу вывод информации о количестве перестановок, за которое было выполнена задача.

13. Подумайте, можно ли уменьшить количество перестановок? Если да, то измените программу.

Контрольные вопросы

1. Поясните строку из метода Hanoi

```
for (Int64 i = 0; i < 1E7; i++) ;
```

2. Поясните фрагмент программы

```
    if (c != null)
{
        c[0].Draw();
        c[1].Draw();
        c[2].Draw();
}
```

3. Что означает фрагмент

```
        c = new Tower[3]
{
        new Tower(this.g, 1),
        new Tower(this.g, 2),
        new Tower(this.g, 3)
        }?
```

4. Как по другому можно записать фрагмент

```
        c = new Tower[3]
{
        new Tower(this.g, 1),
        new Tower(this.g, 2),
        new Tower(this.g, 3)
        }?
```

5. Для чего предназначен объект *g*?

ЛАБОРАТОРНАЯ РАБОТА 9
АЛГОРИТМЫ ПОИСКА

Цель работы: Изучение алгоритмов поиска в не отсортированном и отсортированном массивах.

Теоретические сведения

Поиск элемента массива на основе линейного просмотра. Результатом работы алгоритма линейного поиска значения *Val* в массиве *A размерности N* (0 **to** *N*-1) являются индекс *Pos* и логическая переменная *ResultOK*, которая принимает значение *True*, если такой элемент содержится в массиве *A*, и *False* — в противном случае. Индекс *Pos* принимает значение, равное номеру искомого элемента, если такой найден, и значение, равное «–1» в противном случае.

Алгоритм линейного поиска имеет вид.

Шаг 1. Полагается *Pos*=–1 и *ResultOK*=*False*, и значение переменной цикла *j*=0.

Шаг 2. Если *A*(*j*)=*Val*, то переменным *Pos* и *ResultOK* присваиваются соответственно значения *Pos*=*j*, *ResultOK*=*True* и алгоритм завершает работу. В противном случае значение переменной цикла увеличивается на единицу *j*=*j*+1.

Шаг 3. Если *j*<*N*, то выполняется *Шаг* 2, в противном случае — работа алгоритма завершена.

Конец алгоритма.

На рис. 1 показана схема алгоритма линейного поиска.

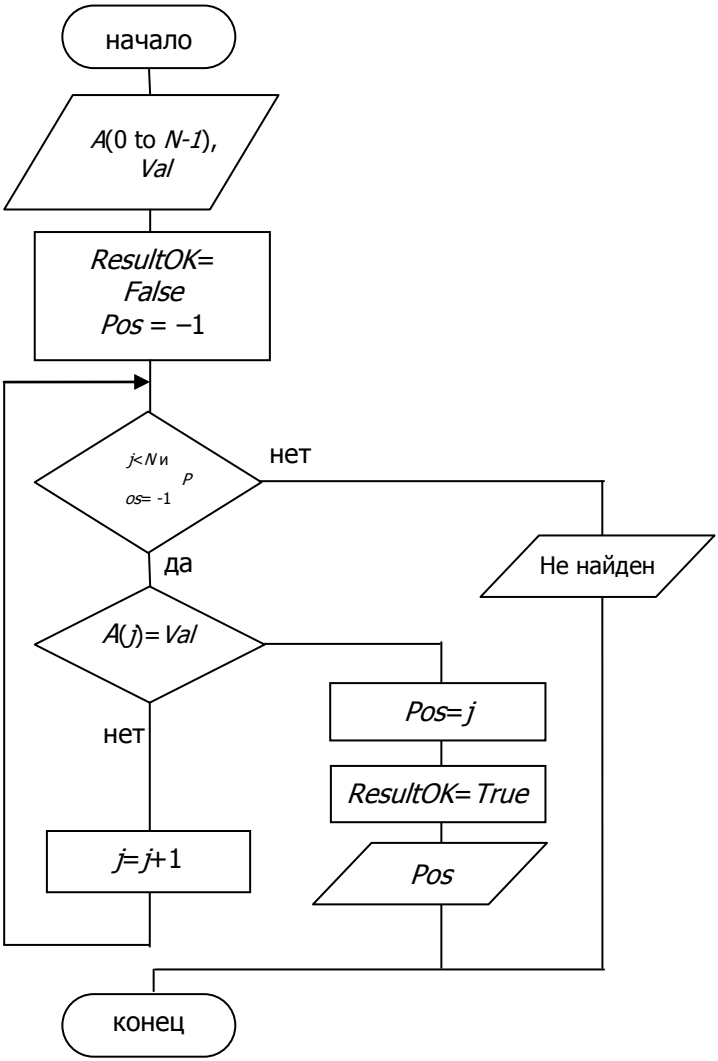


Рис. 1. Схема алгоритма линейного поиска

Метод двоичного поиска. Результатом работы алгоритма является индекс *Pos*, показывающий на место в упорядоченном массиве *A* с номерами элементов от *First* до *Last*, на которое необходимо поставить значение *Val* так, чтобы вновь образованный массив остался упорядоченным. Формируется в качестве резуль-

тата и логическая переменная *ResultOK*, которая принимает значение *True*, если искомый элемент содержится в массиве *A*, и — *False* — в противном случае.

Алгоритм двоичного поиска.

Шаг 1. Пока справедливо условие *First*<*Last*, выполняется *Шаг* 2.

Шаг 2. Вычисляется середина массива *Middle*:=(*Last*+*First*) **div** 2. Если *Val* = *A*(*Middle*), то полагается *First*:=*Middle; Last*:=1. Если *Val* > *A*(*Middle*), то полагается *First*:=*Middle*+1, в противном случае полагается *Last*:=*Middle*. После чего управление передается на *Шаг 1.*

Шаг 3. Полагатся *Pos*=*First*.

Шаг 4. Проверка успеха поиска элемента *Val* в массиве. Полагается *ResultOk*=*FALSE*. После чего проверяется, содержится ли элемент со значением *Val* в массиве, и при положительном ответе на этот вопрос переменной *ResultOk* присваивается значение *True*.

Шаг 5. В случае если, найденная позиция находится в конце массива, а искомый элемент больше элемента в найденной позиции, то к *pos* прибавляется 1, поскольку искомое число необходимо добавить после последнего.

Конец алгоритма.

На рис. 2 приведена схема алгоритма *BinarySource*. Входными аргументами алгоритма являются массив *A*, в котором осуществляется поиск, индекс *i1* первого элемента массива, индекс *i2* последнего элемента массива и число *Val*, подлежащее поиску. Выходным значением функции являются позиция элемента, в котором находится искомое число или позиция, в которое можно поместить искомое число без нарушения упорядоченности массива.

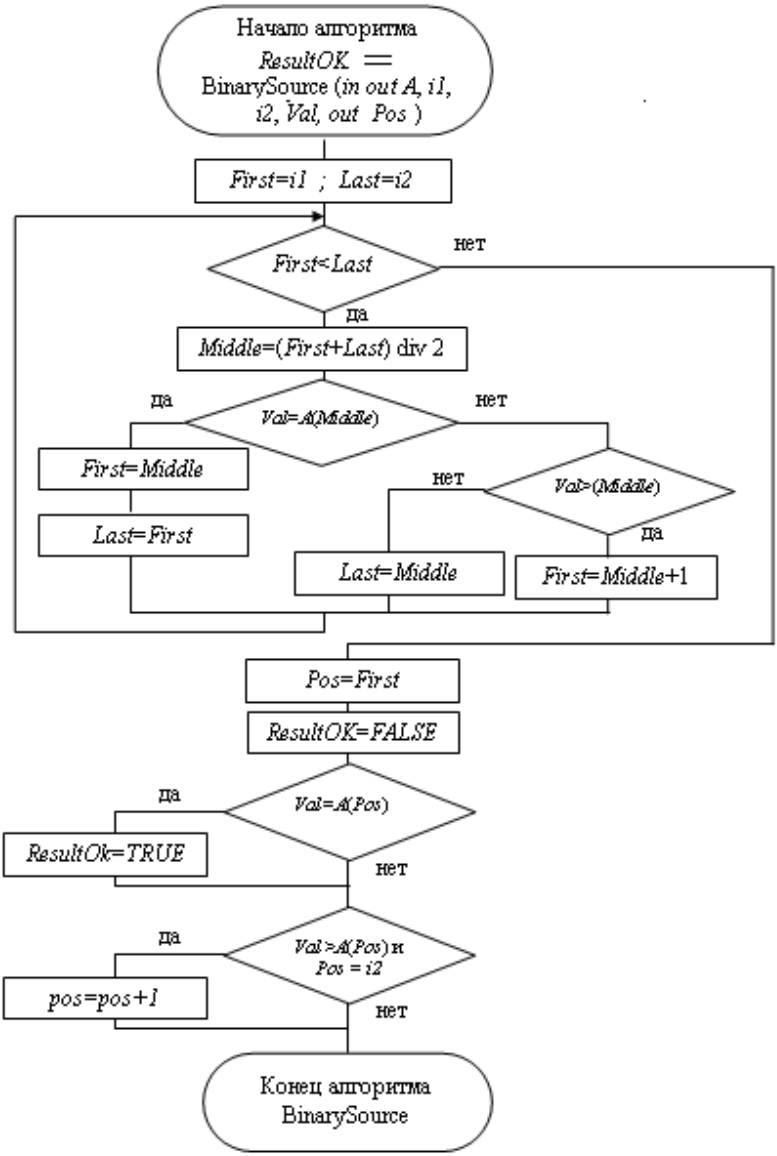


Рис. 2. Схема алгоритма двоичного поиска

Задание. Осуществить поиск указанного элемента в не отсортированном и отсортированном массивах, используя, соответственно, алгоритмы линейного поиска и двоичного поиска.

Порядок выполнения работы и рекомендации по реализации программы

При составлении программы необходимо разработать две функции:

– метод SearchLin для реализации линейного поиска в не отсортированном массиве. Заголовок метода

bool SearchLin (int [] m, int Val, out Pos).

Входными аргументами метода должны быть: массив, в котором осуществляется поиск и число, подлежащее поиску. Выходным аргументом функции должна быть позиция в массиве, в которой находится искомый элемент, или число –1, если элемент не найден. Результатом функции должна быть логическая величина **True**, если искомый элемент находится в массиве, и **False** – в противном случае;

– функцию *SearchBin* для реализации бинарного поиска в отсортированном массиве. Входными аргументами функции должны быть: массив, в котором осуществляется поиск и число, подлежащее поиску. Выходным аргументом и результат функции должны быть такими же, как в функции *SearchLin*. Выходным аргументом функции должна быть позиция в массиве, в которой находится искомый элемент, или позиция, в которую можно добавить искомый элемент без нарушения упорядоченности массива, если элемент не найден. Результатом функции должна быть логическая величина **True**, если искомый элемент находится в массиве, и **False** – в противном случае;

Разработанная программа должна генерировать случайную последовательность чисел. Затем пользователь должен иметь возможность указать элемент, подлежащий поиску, вызвать требуемую функцию поиска. В результате работы программы на экран необходимо вывести позицию искомого числа, если оно содержится в массиве. Если искомое число не содержится в массиве – выдать соответствующее сообщение.

Для формирования последовательности неупорядоченных чисел рекомендуется использовать класс **Random**. Метод NextDouble этого класса генерирует случайные числа в диапазоне [0, 1). Для получения чисел в диапазоне [A, B) необходимо число в диапазоне [0, 1) умножить на (B-A) и затем прибавить A. Таким образом, для генерации 200 чисел в диапазоне [0, 100) и записи этих чисел в массив *m* необходимо использовать поля или локальные переменные

```
int N;
int [ ] m;
Random r;
```

и следующий программный код:

```
N=200;
m = new int [N];
r = new Random( );
for (int i=0; i<N; i++)
  m[i] = (int) (r.NextDouble( ) * 100-200);
```

Для формирования отсортированной последовательности рекомендуется сформировать первое число случайным образом, а затем, получение очередного числа свести в прибавлению небольшого случайного положительного числа к текущему.

```
m[0] = (int)(r.NextDouble( ) * 5);
for (int i=1; i<N; i++)
  m[i] = m[i -1] + (int)(r.NextDouble() * 4 + 1);
```

Для реализации алгоритмов поиска необходимо познакомиться с лекцией **«Алгоритмы поиска»**.

Контрольные вопросы

1. В чем заключается идея алгоритма бинарного поиска?
2. Какова вычислительная сложность линейного поиска?
3. Какова вычислительная сложность бинарного поиска?

ЛАБОРАТОРНАЯ РАБОТА 10
АЛГОРИТМЫ СОРТИРОВКИ

Цель работы: Исследование алгоритма сортировки и определение его вычислительной сложности.

Краткие теоретические сведения

Алгоритмы сортировки

В рассматриваемых алгоритмах сортировки будем осуществлять сортировку элементов массива *A*(0 **to** *N*-1) длиной *N* по возрастанию.

Метод сортировки выбором. Исходный массив длиной *N* разбивается на две части: итог и остаток. Участок массива, называемый итогом, располагается с начала массива и должен быть упорядоченным, а участок массива, называемый остатком, располагается вплотную за итогом и содержит исходные числа не отсортированной части исходного массива. Пусть первый элемент остатка является *j*-ый элементом массива.

Алгоритм сортировки выбором.

Шаг 1. Полагается *j*=0, т.е. считается, что итоговый участок пуст.

Шаг 2. В остатке массива ищется минимальный и меняется с первым элементом остатка (*j*-ым элементом массива).

Шаг 3. Если *j*<*N*-1, то повторяется *Шаг* 2. В противном случае — конец алгоритма, т.к. итог становится равный всему массиву.

Конец алгоритма.

На рис. 2 приведена схема алгоритма сортировки выбором.

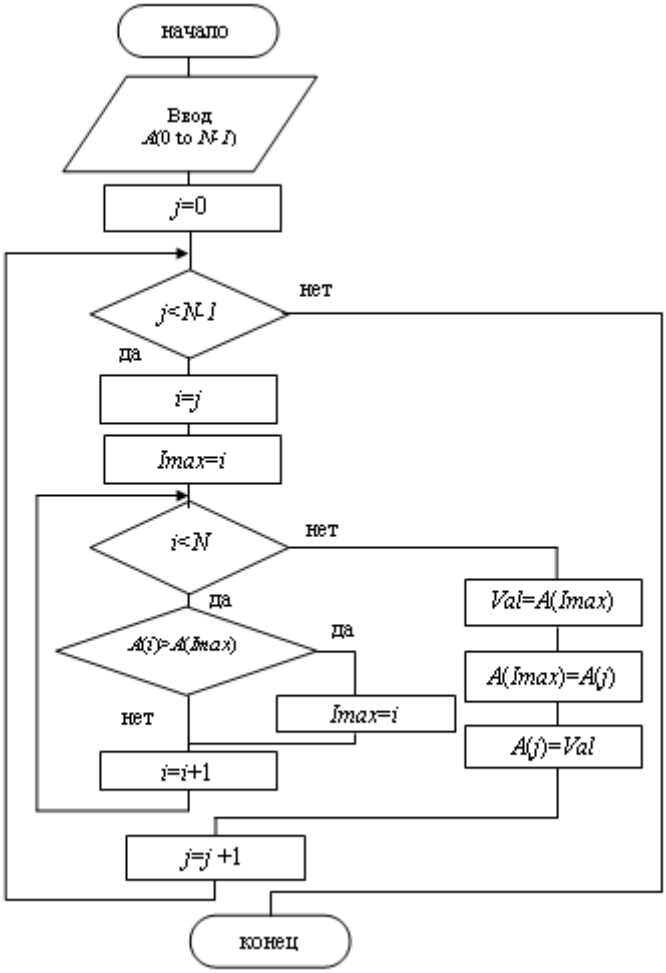


Рис. 2. Схема алгоритма сортировки выбором

Метод сортировки пузырьком. Аналогично, как и в методе выбора, исходный массив длиной *N* разбивается на две части: отсортированную (итог) и не отсортированную (остаток). Пусть первый элемент остатка будет *j*-ым элементом массива.

Алгоритм сортировки пузырьком.

Шаг 1. Пусть *j*=1, т.е. итоговый участок состоит из одного элемента.

Шаг 2. Берется первый элемент остатка и перемещается на место в итоговый участок мас- сива так, чтобы итог остался

упорядоченным. Первый элемент остатка назовем перемещаемым. Перемещение выполняется путем сравнения перемещаемого элемента с предшествующим ему элементом. Если предшествующий элемент меньше сравниваемого элемента, то процесс перемещения закончен. В противном случае сравниваемые элементы переставляются и, если элемент не достиг начала массива, то повторяется *Шаг* 2.

Шаг 3. После того, как первый элемент остатка переместился в итоговый участок, увеличивается на единицу значение переменной *j*, тем самым увеличивая отсортированную часть массива. Если $j<N$, то управление передается на *Шаг* 2, в противном случае — работа алгоритма завершена.

Конец алгоритма.

На рис. 3 приведена схема алгоритма сортировки пузырьком.

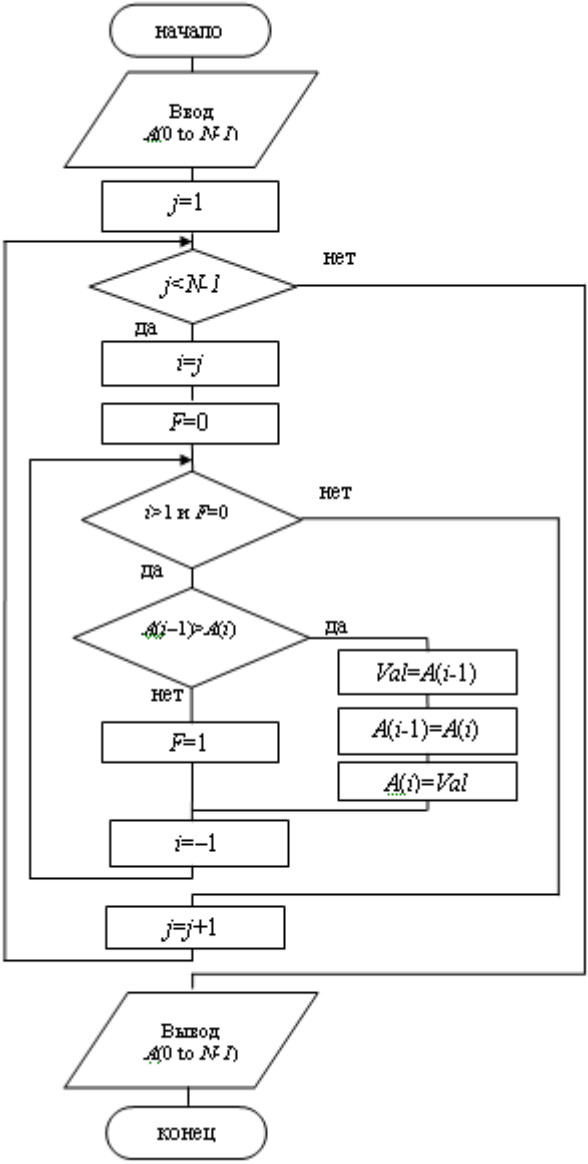


Рис. 3. Схема алгоритма сортировки пузырьком

Метод быстрой сортировки. Исходным является массив *A* с номерами элементов от *First* до *Last*. В алгоритме используют-

ся еще два индекса массива, обозначенные как *Index* и *ContrIndex*. Первый из них всегда указывает на переставляемый элемент, а второй — на элемент, который сравнивается по значению с переставляемым. В процессе вычислений применяются переменная *h* (равная либо «1», либо «–1») — шаг движения индексов навстречу друг другу, используемая для обозначения направления движения индекса *ContrIndex*, и логическая переменная *Condition* (равная либо *TRUE*, либо *FALSE*), используемая для изменения условия сравнения на противоположное при обратном движении индекса *ContrIndex*.

Алгоритм быстрой сортировки.

Шаг 1. Если *First*>=*Last*, то происходит выход из алгоритма. В противном случае полагается *h*=1, *Condition*=*TRUE*, *Index*=*First*, *ContrIndex*=*Last* и делаются шаги: *Шаг* 2 — *Шаг* 3.

Шаг 2. Пока *Index* не равно *ContrIndex*, делаются шаги: *Шаг* 2a — *Шаг* 2b.

Шаг 2a. Если справедливо ((*A*(*Index*)>*A*(*ContrIndex*))=*Condition*), то переставляются как сами элементы, на которые указывают *Index* и *ContrIndex*, (*Val*=*A*(*Index*), *A*(*Index*)=*A*(*ContrIndex*), *A*(*ContrIndex*)=*Val*), так и сами вспомогательные индексы массивов (*Val*=*Index*, *Index*=*ContrIndex*, *ConterIndex*=*Val*). Затем меняется направление движения (*h*=–*h*) и условие сравнения (*Condition*=not *Condition*). В процессе таких перестановок слева от переставляемого элемента всегда будут находиться меньшие значения, а справа — большие значения.

Шаг 2b. Сдвигается индекс массива *Index* навстречу индексу *ContrIndex*, т.е. *Index*=*Index*+*h*.

Шаг 3. Перед выполнением этого шага индексы *Index*=*ContrIndex* и элемент *A*(*Index*) находится на нужном месте, т.е. исходный массив разбит на три части: часть массива до этого элемента, значения в котором меньше величины *A*(*Index*); сам элемент *A*(*Index*); часть массива после этого элемента со значениями большими значения *A*(*Index*). Поэтому для дальнейшего упорядочивания массива достаточно рекурсивно обратиться к алгоритму быстрой сортировки два раза: для первой и второй частей массива. т.к. длина сортируемых участков массива уменьшается, то в итоге алгоритм конечен и после применения алгоритма массив будет полностью отсортирован.

Конец алгоритма.

На рис. 4 приведена схема алгоритма быстрого поиска

QuickSort с тремя входными параметрами: *A* — сортируемый массив, *First* — индекс первого элемента сортируемого массива, *Last* — индекс последнего элемента сортируемого массива. Поскольку массив в функцию передается по ссылке, то параметр *A* также является выходным.

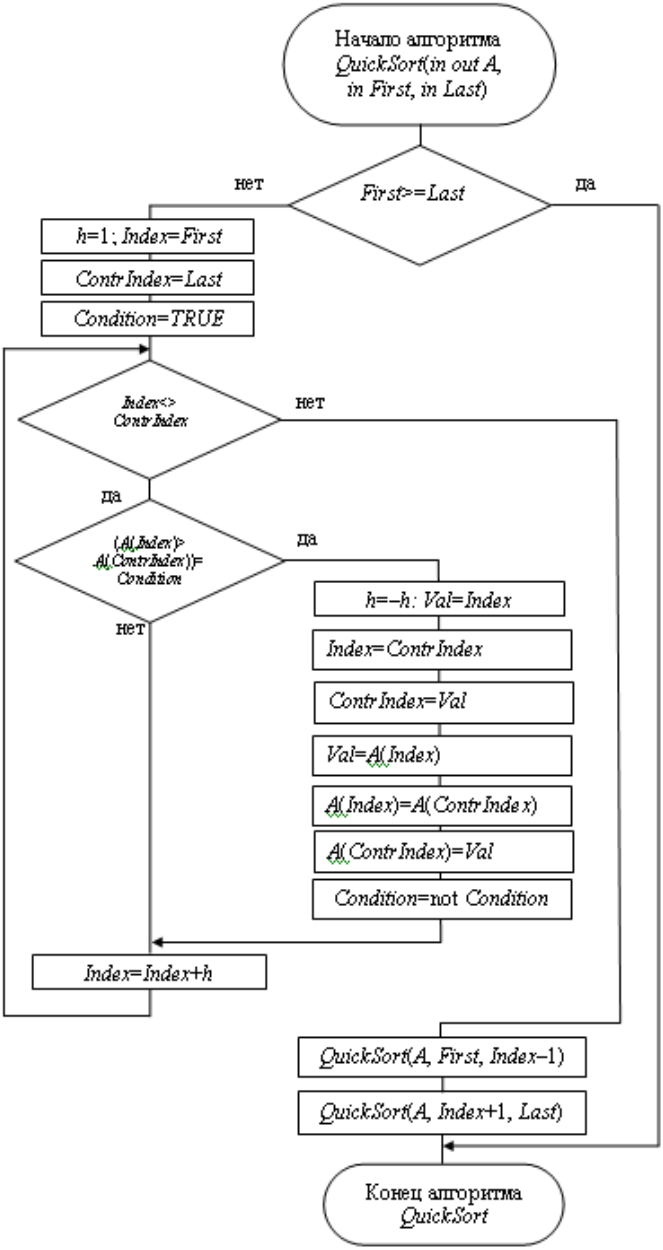


Рис. 4. Схема алгоритма быстрой сортировки

Вычислительная сложность алгоритмов

С развитием вычислительной техники было создано большое количество алгоритмов и потребовалось обратить серьезное внимание на вопросы их эффективности. Поскольку ресурсы памяти и времени работы компьютера ограничены, то недостаточно знать, что существует алгоритм решения данной задачи. Нужно представлять, какие ресурсы понадобятся ЭВМ для реализации алгоритма, сможет ли соответствующая программа поместиться в памяти и даст ли она результаты за приемлемое время. Исследования этих вопросов создали новый раздел теории алгоритмов — теорию сложности алгоритмов.

Выделяют три формы сложности алгоритма:

— временная сложность — количественная характеристика, которая определяет время необходимое для выполнения алгоритма;

— емкостная (пространственная) сложность измеряется объемом памяти, требуемой для выполнения алгоритма. Если две программы реализуют идентичные функции, то та, которая использует меньший объем памяти, характеризуется большей пространственной эффективностью;

— интеллектуальная сложность — позволяет количественно оценить понятность алгоритма и сложность его разработки.

Все три формы сложности обычно взаимосвязаны. Как правило, при разработке алгоритма с хорошей временной оценкой сложности приходится жертвовать его пространственной и/или интеллектуальной сложностью. Часто бывает так, что существует два разных алгоритма решения одной задачи, причем первый алгоритм существенно быстрее, чем второй, однако первый требует большего объема памяти. Кроме того, первый алгоритм характеризуется также и большей интеллектуальной сложностью по сравнению со вторым.

Развитие технологии привело к тому, что память стала дешевой и компактной. Как правило, она не является критическим ресурсом. Поэтому чаще всего под анализом сложности ал

. Далее под сложностью будем понимать именно временную сложность, ее еще называют трудоемкостью алгоритма. Обычные единицы измерения времени (секунды и т.д.) для измерения временной сложности здесь не подходят — одна и та же программа при одних и тех же входных данных на разных компьютерах будет выполняться разное время.

Физическое время выполнения алгоритма — это величина τt, где t — число действий (элементарных шагов, команд), а τ — среднее время выполнения одного элементарного действия.

Число t определяется описанием алгоритма и не зависит от физической реализации модели, а τ зависит от скорости обработки сигналов в элементах и узлах ЭВМ. Поэтому объективной математической характеристикой временной сложности алгоритма является число элементарных действий, выполняемых в ходе работы алгоритма. Однако далеко не всегда ясно, какие операции следует считать элементарными. Кроме того, разные операции требуют для своего выполнения разного количества времени, да и перевод операций, используемых в описании алгоритма, в операции, используемые в компьютере, является неоднозначным, зависящим от таких, например, факторов, как свойства компилятора и квалификация программиста. Таким образом, задача анализа сложности алгоритма состоит в исследовании того, как меняется время работы при увеличении объема входных данных.

Временную сложность алгоритма выражают функцией $T(n)$ зависимости времени работы (числа элементарных действий) от размера входа. Размер входа определяется для каждой задачи индивидуально. Обычно у задачи есть какой-нибудь естественный параметр, характеризующий объем входных данных, и сложность оценивается по отношению к этому параметру. В задачах обработки одномерных массивов под размером входа принято считать количество элементов в массиве.

К элементарным действиям, определяющим временную сложность алгоритма, следует отнести, прежде всего, операции сравнения и присваивания.

Функции, часто встречающиеся при анализе алгоритмов:

— A — константная сложность. Затраты времени при выполнении алгоритма не зависят от размера задачи;

— n — линейная сложность. Время выполнения алгоритма линейно зависит от размерности задачи;

— $\log n$ — логарифмическим сложность. Когда время работы программы логарифмическое, программа начинает работать намного медленнее с увеличением n. Такое время работы встречается обычно в программах, которые делят большую проблему на маленькие и решают их по отдельности;

— $n \log n$ — такое время работы имеют те алгоритмы, которые делят большую проблему в маленькие, а затем, решив их,

соединяют их решения;

— n^{α} — полиномиальная сложность, причем, n^2 — квадратичное время, n^3 — кубическое. Квадратические и кубические алгоритмы не рекомендуется использовать при больших n, т.к. их вычислительная сложность резко возрастает;

— 2^n — экспоненциальная сложность. Такие алгоритмы чаще всего возникают в результате подхода именуемого методом грубой силы.

Задание. Осуществить сортировку сгенерированных случайным образом чисел с помощью различных алгоритмов сортировки.

Порядок выполнения работы

1. Сгенерировать одномерный массив с помощью генератора случайных чисел размера N (например, 10000* - количество чисел можно взять другим в зависимости от быстродействия компьютера).

```
int N=10000;
int [] m = new int [N];
for (int i=0; i<N; i++)
  m[i] = (int)( r.NextDouble() * 15000-30000);
```

2. Отсортировать массив методом в соответствии с индивидуальным заданием.

Для реализации модуля сортировки необходимо ознакомиться с лекцией **«Алгоритмы сортировки»**. Обратите внимание, что в алгоритмах предполагается, что элементы массива нумеруются с нуля.

Для реализации алгоритма сортировки включением можно использовать функцию *SeachBin* из лабораторной работы **«Алгоритмы поиска»**.

3. Определить время сортировки массива.

Для этого:

```
// Сгенерировать массив
// Запустить программный секундомер с помощью класса Stopwatch
Stopwatch sw = Stopwatch.StartNew();
// Отсортировать массив
sw.Stop();  //Остановить таймер
// Вывод затраченного на сортировку времени
```

Console.WriteLine("Время выполнения: " + sw.ElapsedMilliseconds + " мс");

Для перезапуска секундомера используется метод Restart() в виде sw.Restart();

4. Увеличить количество чисел в массиве в 2 раза.
5. Определить время сортировки массива.
6. Увеличить количество чисел в массиве в 2 раза.
7. Определить время сортировки массива.
8. Определить вычислительную сложность алгоритма O(??) на основе полученных экспериментальных данных, занесенных в таблицу 1.

Таблица 1.

Кол-во чисел / Метод сортировки	10000	20000	40000	80000
	Время сортировки			
Вариант 1. Сортировка выбором				
Вариант 2. Метод пузырька				
Вариант 3. Сортировка включением				
Вариант 4. Метод быстрой сортировки				

Контрольные вопросы

1. В чем заключается алгоритм сортировки пузырьком?
2. В каких случаях лучше использовать алгоритм сортировки пузырьком?
3. В чем заключается алгоритм быстрого поиска?

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К КУРСОВОЙ РАБОТЕ

1. ЦЕЛЬ И ЗАДАЧИ КУРСОВОГО ПРОЕКТИРОВАНИЯ

Выполнение курсовой работы является завершающим этапом изучения дисциплины "Технологии и методы программирования ". Целью выполнения курсовой работы является закрепление и углубление теоретических знаний, приобретение практических навыков при разработке сложных программных систем для заданной проблемной области на языке высокого уровня с использованием объектно-ориентированной технологии и технологии визуального программирования с использованием стандартных библиотек классов языка программирования C#.

Курсовое проектирование связано с разработкой программного обеспечения, необходимого для функционирования предприятия.

В каждом варианте описывается деятельность предприятия, требующая информатизации.

Этапами выполнения курсовой работы являются:

- изучение особенностей конкретной предметной области, относящейся к теме курсовой работы;
- анализ системных и программных требований;
- анализ возможных подходов и методов решения поставленной задачи с обоснованием выбранного подхода;
- проектирование классов, структур данных, алгоритмов и программных структур;
- кодирование;
- тестирование разработанного программного продукта;
- анализ результатов, полученных в результате работы разработанного программного обеспечения (ПО);
- разработка документации к ПО.

В результате выполнения курсовой работы должны быть получены оттестированное программное обеспечение и пояснительная записка.

2. ОБЩИЕ РЕКОМЕНДАЦИИ ПО РАЗРАБОТКЕ ПО

Разработка ПО является определяющим элементом выполнения курсовой работы.

Разработка ПО должна начинаться с тщательного изучения задания на курсовую работу. Первоочередной задачей разработчика является формирование полного технического задания в со-

ответствии с ГОСТ 19.201-78 [1].

Для разработки ПО одним или малым количеством программистов рекомендуется использовать инкрементную стратегию, которая заключается в следующем. В начале процесса разработки определяются все пользовательские и системные требования, оставшаяся часть конструирования выполняется в виде последовательности итераций.

На рис. 1 приведена схема инкрементной модели разработки ПО.

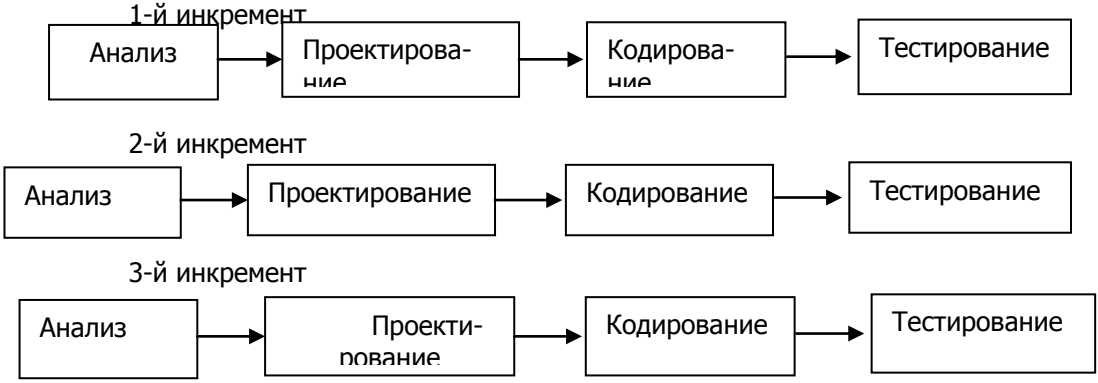


Рис. 1. Схема инкрементной модели разработки ПО

Первая версия разработанного программного продукта (1-й инкремент) приводит к получению базового продукта, реализующего базовые требования. Многие вспомогательные требования остаются нереализованными. План следующего инкремента предусматривает модификацию базового продукта, обеспечивающего дополнительные характеристики и функциональность.

Итеративная модель обеспечивает на каждом инкременте работающий продукт.

При разработке ПО необходимо использовать объектно-ориентированную технологию разработки, определяющими этапами которой являются объектно-ориентированный анализ и объектно-ориентированное проектирование.

Объектно-ориентированный анализ и проектирование принципиально отличаются от методов, используемых при структурном подходе к программированию. Методы структурного проектирования помогают упростить процесс разработки сложных систем за счет использования алгоритмов как готовых блоков. Аналогично, методы объектно-ориентированного проектирования позволяют использовать в качестве блоков классы и объекты.

В процессе объектно-ориентированного анализа осуществляется моделирование проблемы с целью обнаружения объектов и классов, которые составляют словарь проблемной области.

Итогом выполнения этапа объектно-ориентированного анализа является идентификация и моделирование основных объектов и классов и логических отношений и взаимодействий между ними.

На этапе объектно-ориентированного проектирования вводятся абстракции и механизмы, обеспечивающие поведение уже идентифицированных классов и объектов. Признак завершения процесса объектно-ориентированного проектирования состоит в том, что полученные ключевые абстракции становятся достаточно простыми и не требуют дальнейшей декомпозиции.

При разработке методов идентифицированных классов используются структурные методы проектирования.

Целью этапа структурного проектирования является иерархическое разбиение сложной задачи создания метода класса на подзадачи меньшей сложности.

На этапе проектирования решаются следующие задачи:

- формирование структуры ПО и разработка алгоритмов, задаваемых спецификациями;
- определение состава модулей с разделением их на иерархические уровни;
- выбор структуры информации в базе данных;
- фиксация межмодульных интерфейсов.

Цель этапа проектирования — иерархическое разбиение сложной задачи создания ПО на подзадачи меньшей сложности.

Эволюция в жизненном цикле разработки объектно-ориентированного ПО совмещает традиционные этапы, включающие составление программ, их тестирование и интеграцию. Таким образом, процесс разработки превращается в постепенное составление ряда прототипов, которые затем входят в конечную реализацию. Составление развивающихся прототипов в свою очередь стимулирует разработку и оценку альтернативных решений, что позволяет получить разумный компромисс, удовлетворяющий ряду ограничений, которые соответствуют функциональным требованиям, требованиям на сроки разработки и занимаемый объем.

В процессе эволюции ПО в нем могут произойти следующие изменения:

- добавление нового класса;
- изменение реализации класса;

- изменение представления класса;
- реорганизация структуры класса;
- изменение интерфейса класса.

Результатом работы на этом этапе являются спецификации на отдельные модули, дальнейшая декомпозиция которых нецелесообразна.

Целью этапа тестирования и отладки является выявление в ПО ошибок, его соответствие предъявляемым требованиям, демонстрация соответствия функций программы ее назначению.

В ходе этого этапа решаются следующие задачи:
- подготовка данных для отладки;
- планирование отладки;
- разработка тестирующих программ и модулей;
- испытание ПО.

Результатом этого этапа должно быть оттестированное ПО.

3. СОДЕРЖАНИЕ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

3.1. Рекомендуется следующая структура пояснительной записки к курсовой работе:

Титульный лист
Аннотация
Оглавление
Введение
1. Описание программы
1.1. Общие сведения
1.2. Функциональное назначение
1.3. Описание алгоритма функционирования программы
1.4. Диаграммы классов
1.5. Разработка интерфейса
1.6. Разработка запросов
1.5. Используемые технические и программные средства
1.6. Вызов и загрузка
1.7. Входные данные
1.8. Выходные данные
2. Программа и методика испытаний
2.1. Цель испытаний
2.2. Требования к программе
2.3. Программа и методика испытаний
3. Руководство системного программиста

3.1. Назначение и условия применения программы
3.2. Характеристика программы
3.3. Структура программы
3.4. Настройка программы
3.5. Проверка работоспособности программы
3.6. Дополнительные возможности
3.7. Сообщения
4. Руководство оператора
Заключение
Список литературы
Приложение 1. Техническое задание
Приложение 2 и т.д.

3.2. В зависимости от особенностей разрабатываемой программы отдельные пункты (подпункты) допускается объединять, а также вводить новые пункты (подпункты). Рекомендуемый объем пояснительной записки (без приложений) — 28-34 страницы машинописного текста.

3.3. Оформление пояснительной записки выполняется в соответствии с ГОСТ 19.106-78.

3.4. Техническое задание выполняется в соответствии с ГОСТ 19.201-78. Требования к структуре технического задания приведены в приложении 1. Образец оформления технического задания приведен в приложении 2.

3.5. В аннотации приводится краткое содержание работы.

3.6. Во введении дается описание предметной области темы курсовой работы, кратко формулируются цель разработки и актуальность разрабатываемого ПО.

3.7. Раздел «Описание программы» выполняется в соответствии с ГОСТ 19.402-78.

3.8. В подразделе «Общие сведения» должны быть указаны:

- обозначение (наименование) и назначение программы;
- программное и аппаратное обеспечение, необходимое для функционирования разработанного ПО;
- языки программирования, на которых реализовано ПО.

3.9. В подразделе "Функциональное назначение" должны быть указаны классы решаемых задач и (или) назначение программы и сведения о функциональных ограничениях на применение.

3.10. В подразделе "Описание алгоритма функционирования программы" приводится укрупненный алгоритм функционирования ПО с обоснованием выбора схемы алгоритма решения задачи,

возможные взаимодействия ПО с другими программами. Также приводится описание алгоритмов, используемых при разработке программ.

3.11. Логическая модель описывает перечень и смысл ключевых абстракций и механизмов, которые формируют предметную область или определяют архитектуру системы. При анализе необходимо определить требуемое поведение системы, роли и обязанности объектов по поддержанию этого поведения.

При проектировании необходимо определить, какие необходимы классы, каковы связи между ними, какие механизмы регулируют взаимодействие классов, как должен быть объявлен каждый класс.

Для выражения решений на этапах анализа и синтеза используются диаграмма классов. Пример описания диаграммы классов приведен в приложении 4.

3.12. В подразделе «Разработка интерфейса» необходимо описать формы, входящие в программное обеспечение, их назначение и состав визуальных элементов.

3.13. В подразделе «Разработка запросов» необходимо описать запросы, написанные с использованием LINQ для работы с наборами данных предметной области. К таким запросам относятся фильтрация, подсчет статистических данных и так далее.

3.14. В подразделе «Используемые технические и программные средства» должны быть указаны типы электронно-вычислительных машин и устройств, которые могут использоваться для работы программы. В этом разделе необходимо обосновать выбор состава технических и программных средств на основании расчетов и/или анализов; распределение носителей данных, которые использует программа.

3.15. В подразделе «Вызов и загрузка» должны быть указаны способ вызова программы с соответствующего носителя данных. Допускается указывать адреса загрузки, сведения об использовании оперативной памяти, объем программы.

3.16. В подразделе «Входные данные» должны быть указаны характер, организация и предварительная подготовка входных данных; формат, описание и способ их кодирования.

3.17. В подразделе "Выходные данные" должны быть указаны характер и организация выходных данных; формат, описание и способ их кодирования.

3.18. Раздел «Программа и методика испытаний» выполняется по ГОСТ 19.301-79.

3.19. В подразделе «Цель испытаний» указывается цель

проведения испытаний.

3.20. В подразделе "Требования к программе" указываются требования к программе, подлежащие проверке во время испытаний и заданные в техническом задании на программу.

3.21. В подразделе «Средства испытаний@ должны быть указаны технические и программные средства, используемые во время испытаний.

3.22. В подразделе «Программа и методика испытаний» должна быть приведена последовательность действий по испытанию каждого требования, предъявляемого к программе. Перечень методик испытаний рекомендуется по отдельным показателям располагать в той последовательности, в которой эти показатели расположены в разделах «Требования к программе».

В подразделе должны быть приведены способы проведения проверок с указанием ожидаемых результатов.

В приложении к пояснительной записке могут быть включены исходные модули программы, тестовые примеры, результаты работы программы и т.п.

3.23. Раздел «Руководство системного программиста» выполняется в соответствии с ГОСТ 19.503-79 и ГОСТ 19.504-79

3.24. В подразделе «Назначение и условия применения программы» должны быть указаны назначение и функции, выполняемые программой, условия, необходимые для ее работы (объем оперативной памяти, требования к составу и параметрам периферийных устройств, требования к программному обеспечению и т.п.).

3.25. В подразделе «Характеристика программы» должно быть приведено описание основных характеристик и особенностей программы (временные характеристики, режим работы, средства контроля правильности выполнения и самовосстанавливаемости программы и т.п.).

3.26. В подразделе «Структура программы» должны быть приведены сведения о структуре программы, ее составных частях, о связях между составными частями и о связях с другими программами. В данном разделе необходимо привести структуру, как исполняемых, так и исходных модулей.

3.27. В подразделе «Настройка программы» должно быть приведено описание действий по настройке программы на условия конкретного применения (настройка на состав технических средств, выбор функций и др.).

При необходимости привести поясняющие примеры.

3.28. В подразделе «Проверка работоспособности програм-

мы» должно быть приведено описание способов проверки, позволяющих дать общее заключение о работоспособности программы после установки (описание правильной реакции на различные действия пользователя).

3.29. В подразделе «Дополнительные возможности» должно быть приведено описание дополнительных разделов функциональных возможностей ПО и способов их выбора (возможности программы, не описанные в инструкции пользователя, например, настройка путем изменения параметров реестра или ini-файлов).

3.30. В подразделе «Сообщения» должны быть указаны тексты сообщений ошибок, выдаваемых программисту или оператору в ходе выполнения программы, описание их содержания и действий, которые необходимо предпринять системному программисту по этим сообщениям для устранения неисправности.

3.31. В заключении необходимо привести анализ результатов, полученных в ходе выполнения ПО, и выводы, сделанные на их основе. Также необходимо привести рекомендации по дальнейшему улучшению разработанного ПО. Заключение должно содержать только те выводы, которые согласуются с целью исследования, сформулированной в разделе «ВВЕДЕНИЕ» и должны быть изложены таким образом, чтобы их содержание было понятно без чтения текста работы.

3.32. Список литературы должен содержать перечень библиографических описаний документов (книги, статьи, нормативно-технические документы и т.п.), использованных при выполнении работы. Описание документов в списке следует располагать в порядке появления ссылок на них в тексте пояснительной записки.

4. ПРИМЕРНЫЕ ВАРИАНТЫ ЗАДАНИЙ

Большинство заданий относится к имитационному моделированию дискретных систем, каковыми могут считаться обслуживающие организации при представлении их в виде системы массового обслуживания. Разработанное ПО должно обеспечивать выявление важнейших характеристик функционирования системы с целью выработки рекомендаций по улучшению их характеристик на основе принятия своевременных управленческих решений.

Так, признаком улучшения функционирования магазина является увеличение прибыли, что, например, возможно благодаря своевременному заказу более рентабельных товаров в оптимальных количествах у более выгодных поставщиков. Таким образом,

разработанное ПО должно помогать менеджеру принимать своевременные решения по снабжению магазина.

Варианты заданий на курсовой проект

Вариант ***1***. Разработать прикладное программное обеспечение деятельности отдела кадров института. В отделе кадров находятся данные всех сотрудников: рабочих различных специальностей и преподавательского состава, и их трудовой деятельности. Для рабочих учитывается специальность, а для преподавателей и ректора занимаемая должность, сведения об ученой степени сотрудника (кандидат наук, доктор) и ученом звании (доцент, профессор). Также в отделе кадров хранится информация о трудовой деятельности сотрудника: о предыдущих сроках и местах работы.

Вариант ***2***. Разработать прикладное программное обеспечение деятельности биржи труда. На биржу труда обращаются люди, не сумевшие самостоятельно устроиться на работу, но все ещё желающие найти работу по специальности. Организации предоставляют бирже список свободных вакансий. Каждый обратившийся ставится на учет. В день обращения ему предлагается список вакансий. Если свободных вакансий нет или они не устраивают ищущего работу, то ему будет предложено подождать пока подходящее свободное место работы не появится. Зарегистрированный на бирже получает пособие по безработице до тех пор, пока не будет трудоустроен. После этого его данные переносятся в архив, и выплата ему пособия прекращается.

Вариант ***3***. Разработать прикладное программное обеспечение деятельности отдела учета налогообложения физических лиц городской налоговой инспекции. Гражданин, имеющий доходы помимо основного места работы, должен представить в налоговую инспекцию декларацию о полученных доходах. Шкала налогообложения – линейная (13 процентов со всей заработанной суммы за год), но лицам, затратившим средства на обучение, покупку лекарств и т.д., из бюджета должна быть возвращена некоторая сумма, рассчитываемая по специальной методике.

Вариант ***4***. Разработать прикладное программное обеспечение деятельности телеателье. Эта организация занимается послегарантийным ремонтом теле-, радиоаппаратуры. Клиенты

этого телеателье – жители и организации нашего города и близлежащих сел. Расчет с физическими лицами ведется наличными, а с организациями – через банк. Выдача отремонтированной техники производится после полной оплаты выполненного ремонта.

Вариант ***5***. Разработать прикладное программное обеспечение деятельности ГИБДД города. База данных ГИБДД содержит сведения обо всех транспортных средствах города и их владельцах. В нее заносятся сведения о технических осмотрах транспортных средств и об угонах. Описание угнанного автомобиля не удаляется из базы данных. Истории переходов транспортных средств от одних владельцев к другим не накапливаются. Сведения об автомобилях, снятых с учета, навсегда удаляются из базы данных.

Вариант ***6***. Разработать прикладное программное обеспечение деятельности туристической компании. Эта компания формирует туристические группы для заграничных поездок и обеспечивает им полную поддержку на маршруте. Количество туристов в группе заранее известно и ограничено. Маршрут группы может пролегать через несколько городов страны назначения. Экскурсии в несколько стран одновременно не проводятся.

Вариант ***7***. Разработать прикладное программное обеспечение деятельности регистратуры ведомственной поликлиники. Работники регистратуры организуют запись пациентов на прием к врачам поликлиники. Так как поликлиника ведомственная, медицинское обслуживание работников предприятия — бесплатное (за счет средств предприятия). Врач ведет прием всегда в одном кабинете.

Вариант ***8***. Разработать прикладное программное обеспечение деятельности рекламного агентства. В собственности этого агентства находятся рекламные щиты, расположенные по городу. На этих щитах может быть размещена реклама по заказу любой организации города. Срок размещения, стоимость аренды щита — договорные. Одна организация может арендовать несколько рекламных щитов. Один щит не сдается в аренду нескольким арендаторам, так как является неделимой рекламной единицей.

Вариант ***9***. Разработать прикладное программное обеспе-

чение деятельности ООО «Центр оценки и продажи недвижимости». Одним из источников прибыли этой организации является покупка и продажа квартир. Владелец квартиры, желающий ее продать, заключает договор с Центром. Обмен квартир специалисты центра непосредственно не производят. Для этих целей используется вариант купли-продажи.

Вариант ***10.*** Разработать прикладное программное обеспечение деятельности телефонной компании. Основное назначение программного средства — отслеживание абонентской платы за телефоны. Клиентами компании могут быть как физические лица, так и организации. Расчет с организациями ведется в безналичной форме через банк. Физические лица вносят плату через кассу компании. Клиент телефонной компании может иметь несколько телефонных номеров. Дополнительная плата за подключенный параллельно аппарат не взимается. Междугородние и международные звонки оплачиваются отдельно по заранее установленным расценкам.

Вариант ***11.*** Разработать прикладное программное обеспечение деятельности мелкооптового книжного магазина. Менеджер магазина, изучив спрос на книжную продукцию в городе, принимает решение о закупке партии книг в том или ином издательстве. Часть продукции заказывается через Internet. Покупателем в мелкооптовом магазине может быть любой человек. Покупателю выписывается счет-фактура, который имеет уникальный номер и содержит список книг с указанием их стоимости.

Вариант ***12.*** Разработать прикладное программное обеспечение деятельности ОАО «Автовокзал». Это открытое акционерное общество занимается междугородними пассажирскими перевозками. В его собственности находится автобусы различной вместимости. Штат водителей превышает количество автобусов. Билеты на рейсы продаются только в здании автовокзала. Возможна предварительная продажа. Маршрут автобуса может пролегать через несколько населенных пунктов. В этом случае пассажир может купить билет до любого промежуточного пункта.

Вариант ***13***. Разработать прикладное программное обеспечение деятельности ломбарда. Человек обращается в ломбард в том случае, если ему срочно нужны деньги. В ломбарде с клиентом заключается договор. В нем оговариваются следующие усло-

вия: до какого срока выкуп вещи возможен без процентов, с какого времени будет взиматься процент, по истечении какого срока выкуп вещи невозможен, и она поступает в собственность ломбарда.

Вариант ***14***. Разработать прикладное программное обеспечение деятельности гостиницы. Все номера различаются по категориям (суперлюкс, люкс и т.д.), по количеству комнат в номере, количеству мест в каждом номере, а также по обустройству комнат: наличие телевизора, холодильника, телефона. В обязанности администратора гостиницы входит подбор наиболее подходящего для гостя варианта проживания, регистрация гостей, прием платы за проживание, оформление квитанций, выписка отъезжающих.

Вариант ***15***. Разработать прикладное программное обеспечение института селекции растений. Данный институт занимается сбором, выведением и продажей различных сортов семян. В его ассортименте можно найти семена практически всех возможных видов растений: от помидоров до редких цветов. Только что выведенные сорта заносятся в отдельный список для дальнейшего тестирования. Каждый сорт семян имеет свои характеристики. Покупатель может выбрать сорт, отвечающий тем или иным характеристикам. Компания занимается как оптовыми, так и розничными продажами.

Вариант ***16***. Разработать прикладное программное обеспечение деятельности приемной комиссии университета. Каждый год университет зачисляет новых абитуриентов для возможного их поступления в университет после сдачи вступительных экзаменов. На бюджетную основу могут быть зачислены: абитуриенты, получившие необходимый для бесплатного поступления балл ЕГЭ. Все остальные могут поступить в университет на платной основе, набрав необходимое установленное университетом число баллов на вступительных экзаменах.

Вариант ***17***. Разработать прикладное программное обеспечение деятельности кассы авиакомпании. Касса авиакомпании занимается продажей билетов на предстоящие рейсы. В билете указывается номер и название рейса, время вылета, прибытия, номер места, цена билета.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Единая система программной документации. — М.: Изд-во стандартов, 1985.

2. Гагарина Л.Г., Кокорева Е.В., Виснадул Б.Д. Технология разработки программного обеспечения. — М.: ИД «Форум» — Инфра-М, 2009.

3. Смирнов А.А. Технологии программирования. — М.: Евразийский открытый институт, 2011.

4. Буч Г., Коналлен Д., Максимучук Р., Хьюстон К., Энгл М., Янг Б. Объектно-ориентированный анализ и пректирование с примерами приложений, 2-е изд. — М.: Бином, 1999.

5. Страуструп Б. Язык программирования C++, 3-е изд. — М.: Изд-ий дом «Вильямс», 2008.

ПРИЛОЖЕНИЕ 1

ТРЕБОВАНИЯ К СОДЕРЖАНИЮ И ОФОРМЛЕНИЮ ТЕХНИЧЕСКОГО ЗАДАНИЯ

В зависимости от особенностей программы или программного изделия допускается уточнять содержание разделов, вводить новые разделы или объединять отдельные из них.

1. ВВЕДЕНИЕ

В разделе указывают наименование, краткую характеристику области применения программы или программного изделия и объекта, в котором используют программу или программное изделие.

2. ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ

В разделе должны быть указаны:

- документ (документы), на основании которых ведется разработка;
- организация, утвердившая этот документ, и дата его утверждения;
- наименование и/или условное обозначение темы разработки.

3. НАЗНАЧЕНИЕ РАЗРАБОТКИ

В разделе должно быть указано функциональное и эксплуатационное
назначение программы или программного изделия.

4. ТРЕБОВАНИЯ К ПРОГРАММЕ ИЛИ ПРОГРАММНОМУ ИЗДЕЛИЮ

Раздел должен содержать следующие подразделы:

4.1. Требования к функциональным характеристикам

В подразделе должны быть указаны требования к составу выполняемых функций, организации входных и выходных данных, временным характеристикам и т.п.

4.2. Требования к надежности

В подразделе должны быть указаны требования к обеспечению надежного функционирования (обеспечение устойчивости, контроль входной и выходной информации, время восстановления после отказа и т.п.).

4.3. Условия эксплуатации

В подразделе должны быть указаны условия эксплуатации (температура окружающего воздуха, относительная влажность и т.п. для выбранных типов носителей данных), при которых долж-

ны обеспечиваться заданные характеристики, а также вид обслуживания, необходимое количество и квалификация персонала.

4.4. Требования к составу и параметрам технических средств

В подразделе указывают необходимый состав технических средств с указанием их основных технических характеристик.

4.5. Требования к информационной и программной совместимости

В подразделе должны быть указаны требования к информационным структурам на входе и выходе и методам решения, исходным кодам, языкам программирования и программным средствам, используемым программой.

.

5. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

В разделе должен быть указан предварительный состав программной документации и, при необходимости, специальные требования к ней.

6. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

В разделе должны быть указаны: ориентировочная экономическая эффективность, предполагаемая годовая потребность, экономические преимущества разработки по сравнению с лучшими отечественными и зарубежными образцами или аналогами.

7. СТАДИИ И ЭТАПЫ РАЗРАБОТКИ

В разделе устанавливают необходимые стадии разработки, этапы и содержание работ (перечень программных документов, которые должны быть разработаны, согласованы и утверждены), а также как правило, сроки разработки и определяют исполнителей.

8.ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ

В разделе должны быть указаны виды испытаний и общие требования к приемке работы.

ПРИЛОЖЕНИЕ 2

ПРАВИЛА ОФОРМЛЕНИЯ ДИАГРАММЫ КЛАССОВ

Диаграммы классов представляют собой статические модели объектно-ориентированных систем и показывают классы и их отношения. На стадии анализа диаграммы классов используются, чтобы выделить общие роли и обязанности сущностей, обеспечивающих требуемое поведение системы. На стадии проектирования диаграмма классов позволяет передать структуру классов, формирующих архитектуру системы.

Класс обозначается прямоугольником, представленным на рис. 4. Имя класса указывается всегда, свойства и операции — выборочно.

Рис. 4. Обозначение класса

Общий синтаксис представления свойства имеет вид:

Видимость Имя [Множественность]: Тип {Характеристики свойств}.

Видимость обозначается одним из следующих символов: + (для public), – (для private), # (для protected).

Характеристики свойств могут принимать следующие значения:

changeable — нет ограничений на модификацию значения;

frozen — после инициализации объекта значение не изменяется;

addOnly — для свойств с множественностью большей единицы; дополнительные значения могут быть добавлены, но после создания значение не может удаляться или изменяться.

Примеры объявления свойств:

#начало

point_of_begin: CPoint

Family[0..10]: String

Point: CPoint {frozen}

Операция — это услуга, предоставляемая классом. Об-

щий синтаксис представления операции имеет вид

Видимость Имя(Список параметров): ВозврТип {Характеристики}.

В сигнатуре операции можно указать 0 или более параметров, форма представления параметра имеет следующий синтаксис:

Направление Имя: Тип = ЗначениеПоУмолчанию.

Элемент направление может принимать одно из значений: in — параметр не может модифицироваться, out — параметр может модифицироваться для передачи значения в вызывающий объект; inout — входной параметр может модифицироваться.

Примеры объявления операций:

- записать
- зарегестрировать(in и:Имя, in ф:Фамилия): Boolean

На рис. 5 приведены значки отношений между классами.

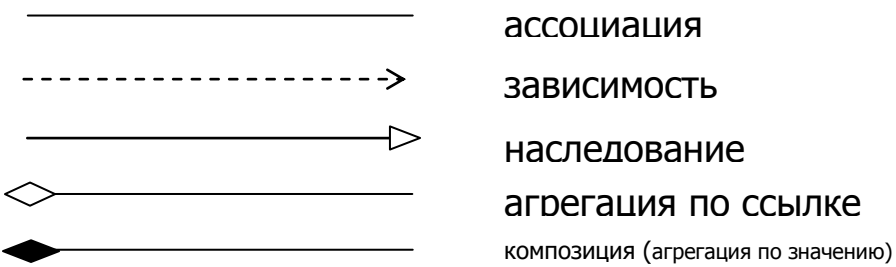


Рис. 5. Условные обозначения отношений между классами

Ассоциация означает семантическое соединение классов и отражает наиболее общее и неопределеннее отношение между ними. Ассоциация не указывает направление и точную реализацию отношения. Она пригодна для анализа проблемы, когда требуется лишь идентифицировать связи. С помощью создания ассоциации происходит понимание участников семантических связей, их ролей и мощности.

Запись мощности на одном конце ассоциации определяет количество объектов, соединяемых с каждым объектом на противоположном конце ассоциации. Можно задать следующие варианты мощности:

1 — в точности одна связь; * — неограниченное число;

0..* — ноль или более; 1..* — один или более;

0..1 — ноль или один; 1..3,7 — указанный интервал или точное число.

На рис.6. приведен пример отображения ассоциации.

РИС. 6. ПРИМЕР ОТОБРАЖЕНИЯ АССОЦИАЦИИ

Зависимость является отношением использования между клиентом (зависимым элементом) и поставщиком (независимым элементом). Обычно операции клиента вызывают операции поставщика или имеют сигнатуры, в которых возвращаемое значение или аргументы принадлежат классу поставщика.

Наследование — отношение, при котором один класс (подкласс) разделяет структуру и поведение, определенные в одном другом (простое наследование) или во многих других (множественное наследование) суперклассах. Значок наследования выглядит как значок ассоциации с треугольником на конце, который указывает от подкласса к суперклассу.

Агрегация обозначает отношения "целое/часть". Стрелка с ромбом на конце указывает от части к целому.

Конкретизация представляет собой наполнение шаблона (параметризированного класса). Целью является получение класса, от которого возможно создание экземпляров. Параметризованные классы достаточно сильно отличаются от обычных, что отмечается специальным украшением на их значках. Параметризованный класс обозначается значком обычного класса с пунктирным прямоугольником в правом верхнем углу, в котором указаны параметры. Инстанцированный класс изображается обычным значком класса с украшением в виде прямоугольника со сплошной границей с перечисленными в нем шаблонными параметрами. Связь между параметризованным классом и его инстанцированием изображается пунктирной линией со стрелкой, указывающей на параметризованный класс.

На рис. 7 приведен пример отображения инстанцированного класса. Инстанцированием параметризированного класса

List с шаблонным параметром Item получены класс StudentBand, в котором параметром является класс Student, и класс VectorOfInteger с параметром int.

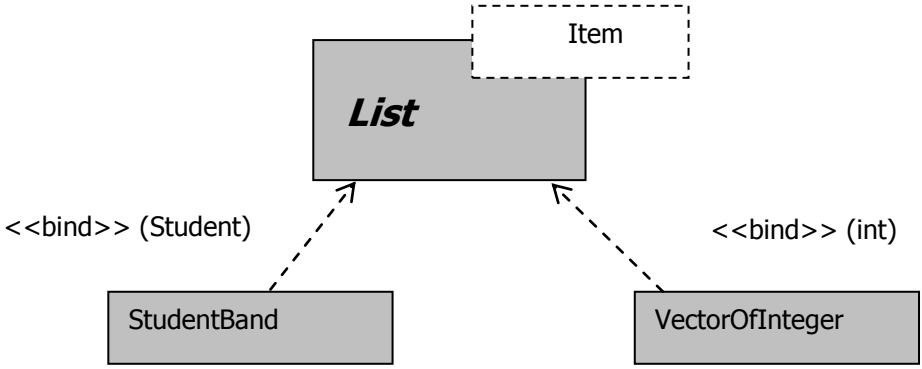


Рис. 7. Пример отображения инстанцированных классов

Рекомендации и этапы разработки программного обеспечения

Первый этап разработки программного обеспечения к курсовой работе

1. Разработать главную форму программного обеспечения с кнопкой, предназначенной для открытия второй формы.
2. Разработать класс, указанный ниже в соответствии с вариантом. Добавить в класс автоинкрементное поле Id (уникальный номер).
3. Добавить поле для хранения коллекции объектов этого класса.
4. Разработать вторую форму для ввода, редактирования объектов класса.
5. Реализовать фильтрацию и сортировку объектов класса во второй форме.
6. Реализовать сохранение и считывание объектов из XML-файла.

Вариант 1. Разработать класс **CWork (Место предыдущей работы)**, содержащее поля: WorkBegin (год начала работы на предыдущем месте), WorkEnd (год окончания работы на предыдущем месте), Work (должность предыдущей работы), WorkPlace (название предприятия), Reason (причина увольнения).

Вариант 2. Разработать класс **CJob (Вакансия)**, содержащий поля: JobID (номер вакансии), JobName (Название вакансии), Mobile (телефон работодателя), Money (примерный размер зарплаты).

Вариант 3. Разработать класс **CDohod (Строка дохода)**, содержащий поля: Name (название организации), InnEnterprise (ИНН организации), Address (адрес организации), SumAll (полученная в организации сумма).

Вариант 4. Разработать класс CZakaz (Заказ), содержащий поля: IdZakaz (Идентификатор заказа), DataZakaza (дата заказа), NameIzd (название изделия), DataEndZakaza (дата выдачи заказа), Summa (сумма ремонта), Period (срок гарантии).

Вариант 5. Разработать класс **CAuto (Автомобиль)**, содержащий поля: BodyID (номер кузова), EngineID (номер двигателя), Model (модель автомобиля), Volume (объем двигателя), Power (мощность двигателя), Drive (привод на все колеса?), Year (год выпуска автомобиля).

Вариант 6. Разработать класс **CClient (Клиент)**, содержащий поля: ClientID (идентификатор клиента), LastName (фамилия клиента), FirstName (имя клиента).

Вариант 7. Разработать класс **CPacient (Пациент)**, содер-

жащий поля: PacientID (Номер карточки пациента), Fio (ФИО пациента), Address (Адрес пациента), PolicyNumber (номер страхового полиса).

Вариант 8. Разработать класс **CSheet (Щит)**, содержащий поля: RegNumber (Регистрационный номер щита), Address (Адрес расположения щита), Square (площадь щита).

Вариант 9. Разработать класс **CFlat (Квартира)**, содержащий поля: Address (Адрес квартиры), Floor (Этаж расположения квартиры), SqAll (площадь кварптиры), SqLife (Жилая площадь).

Вариант 10. Разработать класс **CZvonok (Звонок)**, содержащий поля: ID (Уникальный номер звонка), PhoneNumber (вызываемый номер), Len (Длительность звонка).

Вариант 11. Разработать класс **CBook (Книга)**, содержащий поля: ID (Уникальный номер книги), Name (название книги), Author (Автор книги), Year (Год издания).

Вариант 12. Разработать класс **CAutobus (Автобус)**, содержащий поля: BusNumber (номер автобуса), Model (модель автобуса), Year (год выпуска автобуса), Capacity (количество мест в автобусе).

Вариант 13. Разработать класс **CIzdelie (Изделие)**, содержащий поля: ThingID (Уникальный номер изделия), Name (название изделия), Defects (Наличие дефектов (да/нет)), Cost (Оценочная цена изделия), Sum (Сумма, полученная клиентом).

Вариант 14. Разработать класс **CRoom (Гостиничный номер)**, содержащий поля: Number (уникальный номер), TV (наличие телевизора), Type (категория номера), Cost (цена аренды номера).

Вариант 15. Разработать класс **CRastenie (Растение)**, содержащий поля: Name (Название), Tip (тип семян), Tehnologi (Технология посадки).

Вариант 16. Разработать класс **CAbitur (Абитуриент)**, содержащий поля: PersonID (номер абитуриента), Name (Фамилия абитуриента), Date (дата регистрации), DateBirth (Дата рождения), School (оконченная школа).

Вариант 17. Разработать класс **CReis (Рейс)**, содержащий поля: **ReisID** (номер рейса), **Place1** (место вылета), **Place2** (место прилета), **Seats** (наличие промежуточных посадок).

Второй этап разработки программного обеспечения к курсовой работе

1. Добавить на главную форму кнопку, предназначенную для открытия третьей формы.

2. Разработать класс, указанный ниже в соответствии с вариантом. Добавить в класс автоинкрементное поле Id (уникальный номер).

3. Добавить поле для хранения коллекции объектов этого класса.

4. Разработать третью форму для ввода, редактирования объектов класса (в соответствии с вариантом ниже).

5. Реализовать фильтрацию и сортировку объектов класса во второй форме.

6. Реализовать сохранение и считывание объектов из XML-файла.

Вариант 1. Разработать класс **CWorker** (Сотрудник), содержащий поля PersonID (Табельный номер сотрудника), Family (Фамилия сотрудника), Birth (Дата рождения сотрудника).

Вариант 2. Разработать класс **C**Jobless (Безработный), содержащий поля JoblessID (регистрационный номер безработного), LastName (Фамилия безработного), FirstName (Имя безработного), Age (возраст безработного).

Вариант 3. Разработать класс CDocument (Документ), содержащий поля: Serial (серия документа), Number (номер документа), Date (Дата составления), Who (кем составлен)

Вариант 4. Разработать класс **CZakazchik** (Заказчик), содержащий поля: IdZakazchika (Идентификатор заказчика), Phone (телефон заказчика), Address (адрес заказчика).

Вариант 5. Разработать класс **CHoz** (Владелец автомобиля), содержащий поля: Family (фамилия владельца), Number (номер паспорта владельца).

Вариант 6. Разработать класс **CRoute** (Маршрут), содержащий поля: RouteID (идентификатор маршрута), RouteName (название маршрута), Period (срок пребывания), Cost (стоимость путевки).

Вариант 7. Разработать класс **CDoctor** (Врач), содержащий поля: RouteID (идентификатор врача), Family (фамилия врача), Specialnost (специальность врача), Room (номер кабинета).

Вариант 8. Разработать класс **CCust** (Арендатор), содержащий поля: CustomerID (ИНН арендатора), Customer (название арендатора), AddressCust (адрес арендатора), Room (номер кабинета), Chief (фамилия руководителя).

Вариант 9. Разработать класс **CVlad** (Владелец), содержащий поля: FIO (ФИО владельца), Phone (телефон владельца), Registr (Регистрационный номер клиента).

Вариант 10. Разработать класс **CTelephonNumber** (Телефонный номер), содержащий поля: ID (идентификатор клиента), Family (фамилия клиента), PhoneAddress (адрес клиента), PhoneNumber (номер телефона), Value (Ежемесячная плата за телефон).

Вариант 11. Разработать класс **CCount** (Счет-фактура), содержащий поля: CountNumber (Номер счет-фактуры), Date (Дата выписки счет-фактуры), Value (Сумма к уплате), Cost (Стоимость приобретения).

Вариант 12. Разработать класс **CRabotnik** (Работник автовокзала), содержащий поля: FIO (ФИО работника), Dolg (Должность), Zarplata (оклад).

Вариант 13. Разработать класс **CAgreement** (Договор), содержащий поля: Number (Номер договора), Date (Дата договора).

Вариант 14. Разработать класс **CClient** (Клиент), содержащий поля: Family (Фамилия), BeginDate (дата въезда), EndDate (Дата отъезда), Summa (Сумма оплаты), NKvit (номер квитанции).

Вариант 15. Разработать класс **CBuy** (покупка), содержащий поля: Number (Номер покупателя), Sum (Сумма покупки), Otdel (название отдела магазина, где совершена покупка).

Вариант 16. Разработать класс **CSpec** (специальность), содержащий поля: Spec (название специальности), FIO (ФИО работника), Balls (количество баллов, необходимых для поступления).

Вариант 17. Разработать класс **CBilet (**Билет**)**, содержащий поля: Time1 (время вылета), Time2 (Время прилета), Cost (цена билета), Name (владелец билета), Nomer (номер места).

Третий этап разработки программного обеспечения к курсовой работы

1. Реализовать агрегированную связь между объектами путем добавления поля, указывающего на объект другого класса (связь один-к-одному) в соответствии с вариантом.

Вариант 1. Добавить в класс **CWorker** поле, хранящее список предыдущих мест работы (**CWork)**.

Вариант 2. Добавить в класс **CJobless** поле, хранящее список вакансий, которые ему были предложены (**CJob**).

Вариант 3. Добавить в класс **CTax** поле, хранящее список объектов **CDohod**.

Вариант 4. Добавить в класс **CZakazchik** поле, хранящее список его заказов (CZakaz).

Вариант 5. Добавить в класс **CAuto** поле, хранящее список

всех владельцев автомобиля (**CHoz**).

Вариант 6. Добавить в класс **CRoute** поле, хранящее список всех клиентов (**CClient**).

Вариант 7. Добавить в класс **CDoctor** поле, хранящее список всех его пациентов (**CPacient**).

Вариант 8. Добавить в класс **CCust** поле, хранящее список всех щитов, арендованных им (**CSheet**).

Вариант 9. Добавить в класс **CFlat** поле, хранящее список историй владения этой квартирой (**CVladDogovor**).

Вариант 10. Добавить в класс **CTelephonNumber** поле, хранящее список всех телефонных номеров, сделанных им (**CZvonok**).

Вариант 11. Добавить в класс **CCount** поле, хранящее список всех купленных по этой счет-фактуре книг (**CBook**).

Вариант 12. Добавить в класс **CAutobus** поле, хранящее список всех водителей, обслуживающих этот автобус (**CVoditel**).

Вариант 13. Добавить в класс **CAgreementVlad** поле, хранящее список всех изделий, включенных в договор (**CIzdelie**).

Вариант 14. Добавить в класс **CRoom** поле, хранящее список всех клиентов, живших в нем (**CClient**).

Вариант 15. Добавить в класс **CBuy** поле, хранящее список всех купленных за эту покупку растений (**CRastenie**).

Вариант 16. Добавить в класс **CSpec** поле, хранящее список всех абитуриентов, на этой специальность (**CAbitur**).

Вариант 17. Добавить в класс **CReis** поле, хранящее список всех билетов, проданных на этот рейс (**CBilet**).

Четвертый этап разработки программного обеспечения к курсовой работы

1. Добавить в проект форму для статистических расчетов.
2. Реализовать 8-10 запросов для получения статистических данных.
3. Расположить элементы управления на форме для статистических расчетов таким образом, чтобы можно было найти необходимые команды удобно. Для этого их можно расположить группами, в рамках и так далее.

Пятый этап разработки программного обеспечения к курсовой работы (повышенной сложности)

1. Реализовать возможность вывода информации в документы Winword с использованием COM-технологии.

2. Улучшить оформление DataGridView (программно изменить ширину столбцов, отключить возможность редактирования столбца Id, изменить надписи в заголовках на русскоязычные, информацию об агрегированном объекте сделать более наглядной).

3. Расширить возможности меню.

4. Реализовать более сложные фильтры по нескольким полям одновременно.

5. Добавить команды в меню.

6. Реализовать обработку исключительных ситуаций при неправильной работе пользователя (например, при статистических расчетах при отсутствии данных).

7. Реализовать связь «один-ко-многим» между объектами.