Лабораторная работа №2.

Объекты и классы

Цель: четко разграничить понятия класса, объекта.

Ход работы:

1. Создать новое консольное приложение под именем: NUmericApple, для вывода на экран номера создаваемого объекта (экземпляра) класса.
2. В приложении создать класс «Яблоко». Для этого в Solution Explorer на имени проекта нажать правую кнопку мыши → последовательно выбрать пункты Add → Class → дать имя новому классу Apple.
3. В файле Apple.cs объявить класс, описывающий яблоко:

```
class Apple
{
    // номер первого создаваемого объекта класса
    static int number = 1;

    // номер по порядку создаваемого объекта класса
    private int muNumber = 0;

    // конструктор класса
    public Apple()
    {
        this.muNumber = number;
        number++;

    }

    //метод вывода на экран номера объекта класса
    public string GetNumber()
    {
        return String.Format("Яблоко номер {0}", this.muNumber);
    }
}
```

4. В основной программе дописать:

```
static void Main(string[] args)
    {
        Apple a = new Apple();
        Console.WriteLine(a.GetNumber());
        Apple a1 = new Apple();
        Console.WriteLine(a1.GetNumber());
        Apple a2 = new Apple();
        Console.WriteLine(a2.GetNumber());
    }
```

5. Запустить проект.

Теоретические сведения.

Любая программа на C# содержит как минимум один класс и использует классы из стандартных пространств имен.

Во всех объектно-ориентированных языках четко различаются понятия «класс» и «объект».

Класс — это определяемый пользователем тип данных (user-defined type, UDT). Класс (в программировании) – это способ реализации описания чего-либо, что может и не существовать. Описание можно разбить на две группы – свойства описываемого объекта и его функции.

К примеру, возьмем велосипед. Все велосипеды можно как-то описать, не привязываясь при этом к какой-либо конкретной единице (например, у него есть руль, колеса и т.д.). Кроме того, велосипед способен еще выполнять действия, например, ехать, можно поворачивать его рули и вместе с этим поворачивается колесо, и т.д. Описание класса может определить свойства и поведение «абстракции» (в программировании – описание характеристик и функций).

В отличие от класса **объектом** (сущностью) называется конкретный экземпляр определенного класса («абстракции»), построенный на основании этого описания и с помощью которого обычно и производятся определенные действия в программе.

Единственны и способ создания нового объекта в C# — использовать ключевое слово **new**.

Даже если создать два объекта одного и того же класса, то это будут разные объекты (так же как можно купить два велосипеда одной модели, которые будут иметь одинаковые характеристики, но это будут два разных велосипеда).

Класс.

С точки зрения синтаксиса, между объявлениями классов (class) и структур (struct) существуют незначительные различия:

- в структуре не допускается объявлений членов класса со спецификаторами доступа protected и protected internal,
- при объявлении структуры не допускается объявление конструктора без параметров.

Объявление **класса** состоит из нескольких элементов:

- объявление атрибутов[1] – необязательный элемент объявления,
- модификаторы (в том числе модификаторы прав доступа) – необязательный элемент объявления,
- partial (спецификатор разделения объявления класса) – необязательный элемент объявления,
- class,
- имя класса,
- имена предков (класса и интерфейсов) – необязательный элемент объявления,
- тело класса.

Модификаторы прав доступа (см. табл.1.) обеспечивают реализацию принципа инкапсуляции, используются при объявлении классов и их составляющих компонентов.

[1] Атрибуты – средство добавления ДЕКЛАРАТИВНОЙ (вспомогательной) информации к элементам программного кода. Назначение атрибутов: организация взаимодействия между программными модулями, дополнительная информация об условиях выполнения кода, управление сериализацией (правила сохранения информации), отладка, многое другое.

Табл. 1. Модификаторы прав доступа членов класса.

public	обозначение для общедоступных членов класса. К ним можно обратиться из любого метода любого класса программы.
protected	обозначение для членов класса, доступных в рамках объявляемого класса и из методов производных классов.
internal	обозначение для членов класса, доступных из методов классов, объявляемых в рамках сборки, содержащей объявление данного класса.
protected internal	обозначение для членов класса, доступных в рамках объявляемого класса и из методов производных классов, а также доступных из методов классов, объявляемых в рамках сборки, содержащей объявление данного класса.
private	обозначение для членов класса, доступных в рамках объявляемого класса.

Спецификатор разделения объявления класса **partial** позволяет разбивать код объявления класса на несколько частей, каждая из которых размещается в собственном файле. Если объявление класса занимает большое количество строк, его размещение по нескольким файлам может существенно облегчить работу над программным кодом, его документирование и модификацию. Транслятор способен восстановить полное объявление класса. Спецификатор partial может быть использован при объявлении классов, структур и интерфейсов.

В **теле класса** могут быть объявлены:

- константы;
- поля;
- конструкторы и деструкторы;
- методы;
- свойства;
- делегаты;
- классы.

О свойствах и делегатах предстоит подробный разговор в последующих лабораторных. Из синтаксиса следует, что классы могут быть вложенными. Такая ситуация довольно редкая. Ее стоит использовать, когда некоторый класс носит вспомогательный характер, разрабатывается в интересах другого класса и есть полная уверенность, что внутренний класс никому не понадобится кроме класса, в который он вложен. Как уже упоминалось, внутренние классы обычно имеют модификатор доступа, отличный от public.

Основу любого класса составляют его конструкторы, поля и методы.

Поля класса

Поля класса синтаксически являются обычными переменными (объектами) языка Их описание удовлетворяет обычным правилам объявления переменных.

Поля задают представление той самой абстракции данных, которую реализует класс. Поля характеризуют **свойства объектов класса**. Два объекта одного класса имеют один и тот же набор полей, но разнятся значениями, хранимыми в этих полях.

Методы класса

Методы класса синтаксически являются обычными процедурами и функциями языка. Их описание удовлетворяет обычным правилам объявления процедур и функций. Содержательно методы определяют ту самую абстракцию данных, которую реализует класс. Методы описывают операции, доступные над объектами класса. Два объекта одного класса имеют один и тот же набор методов.

Пример

В качестве примера определим простой класс «Оценки» (см. рис. 1.):

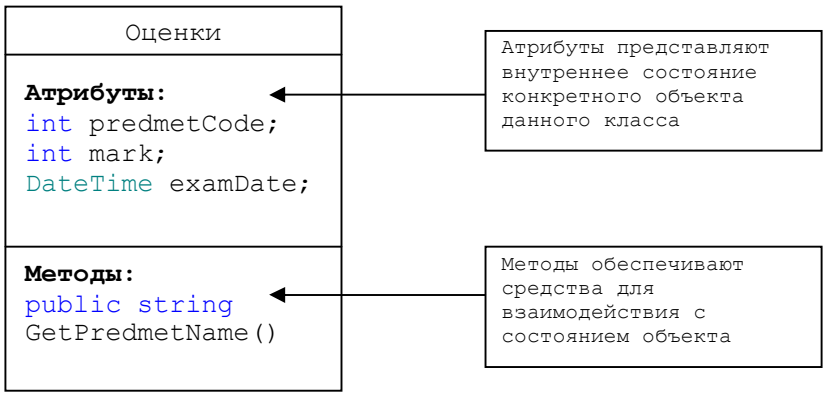

Рис. 1. Определение простого класса

Объявление этого класса будет иметь вид:

```
public class XMark
  {
    //  код предмета
    int predmetCode;
        //  оценка по предмету
    int mark;
        //  дата сдачи
    DateTime examDate;
    // метод получения названия предмета
    public string GetPredmetName()
    {
      return XPredmetList.GetNames()[this[2].predmetCode];
    }
  }
```

Однако, т.к. список предметов храниться в коллекции, то для его заполнения целесообразно использовать объявление еще одного класса «Предмет»:

```
public class XPredmetList
  {
    public static string[] GetNames()
    {
      return items.ToArray();
```

[2] Ключевое слово **this** обозначает ссылку на тот объект, функцию которого мы вызвали. Эта ссылка доступна в любом методе любого объекта.

```
    }
    static List<string> items;
    // конструктор класса, кот. устанавливает список предметов
    static XPredmetList()
    {
       items.AddRange(new string[] { "p1", "p2", "p3" });
    }
}
```

Тогда описание класса «Студент», характеризующегося только атрибутами, примет следующий вид:

```
public class Student
{
    int studentID;
    // Имя студента
    string firstName;
    // Фамилия студента
    string surName;
// Оценки данного студента
    List<XMark> exams = new List<XMark>();
}
```

Объект.

Объекты в C# создаются на основе класса.

У каждого объекта имеется четко определенный жизненный цикл, который длится с момента использования определения класса до уничтожения объекта. Кроме обычного состояния "используется", в жизненном цикле объекта присутствуют еще два важных этапа:

• **Создание объекта** — состояние, когда происходит первоначальное создание экземпляра объекта. Такая инициализация известна под названием создания объекта и осуществляется **конструктором** объекта.

• **Уничтожение объекта** — состояние, когда происходит уничтожение объекта, при котором очень часто возникает необходимость проведения различных восстановительных мероприятий, например освобождения памяти. Эта работа входит в функции **деструктора** объекта.

Конструктор.

Инициализация объекта происходит автоматически транслятором. Нам не приходится беспокоиться, допустим, о поиске свободной памяти, в которой будет размещен новый объект. Однако иногда возникают ситуации, когда на этапе инициализации может потребоваться выполнение каких-либо дополнительных действий. Например, очень часто бывает необходимо инициализировать данные, хранящиеся в объекте. Как раз это и входит в функции конструктора.

У каждого объекта имеется **конструктор, используемый по умолчанию,** который **представляет собой метод без параметров (**обеспечивает инициализацию переменной

предопределённым значением[3]**)**. Его имя совпадает с именем самого класса, вызывается ключевым словом new. Например, мы можем создать объект типа string следующим образом:

```
string myString = new string () ;
```

или:

```
Class X
{

}
::::::::::
X x = new X();                  // Работает конструктор умолчания.
```

Определение класса может включать в себя несколько конструкторов. Они могут иметь отличающиеся сигнатуры, которые используются для создания экземпляра объекта. Для присваивания начальных значений данным, хранящимся внутри объекта, часто применяются конструкторы с параметрами.

Попытка объявления ЛЮБОГО варианта конструктора (с параметрами или без) приводит к тому, что транслятор перестаёт заниматься построением собственных версий конструкторов. Отныне в классе нет больше конструктора умолчания. Теперь всё зависит от соответствия оператора определения объекта построенному нами конструктору. Объявим в производном классе оба варианта конструкторов.

```
Class X
{
public X(int key){}
}
::::::::::
X x0 = new X(125);          //Работает конструктор с параметрами.
X x1 = new X();                     //Ошибка! Конструктора умолчания уже нет!
```

Однажды взявшись за дело объявления конструкторов, разработчик класса должен брать на себя ответственность за создание ВСЕХ без исключения вариантов конструкторов, которые могут потребоваться для решения поставленной задачи.

```
Class X
{
public X(int key){}
public X(){}
}
::::::::::
X x0 = new X(125);          // Работает конструктор с параметрами.
X x1 = new X();                     // Работает новый конструктор без параметров.
```

[3] Со списком предопределённых значений, которыми инициализируются объекты предопределённых типов, можно ознакомиться в Default Values Table.

Особенности конструктора заключаются в том, что:

- конструктор НЕ ИМЕЕТ НИКАКОГО спецификатора возвращаемого даже void,
- имя конструктора полностью совпадает с именем класса или структуры,
- в классе и в структуре можно объявлять множество вариантов конструкторов. Они должны отличаться списками параметров. В структуре невозможно объявить конструктор с пустым списком параметров,
- не существует выражения вызова для конструктора, управление в конструктор передаётся посредством выполнения специальной операции new.

В некоторых классах вообще отсутствуют общедоступные конструкторы, что означает невозможность для внешнего кода создавать экземпляры данного класса. Однако, как вы вскоре сможете убедиться, такие классы вовсе не являются совершенно бесполезными.

Деструктор.

Деструкторы используются в .NET Framework для того, чтобы выполнять уборку за объектами. В общем случае нам не требуется писать какой бы то ни было код для метода деструктора; напротив, за нас будет работать операция, выполняющаяся по умолчанию. Однако если необходимо выполнить какие-либо важные действия перед уничтожением экземпляра объекта, то можно задать определенные инструкции.

Когда, например, мы выходим за область действия переменной, она становится недоступной для нашего кода, но при этом она может по-прежнему существовать где-нибудь в памяти компьютера. Только после того как сборщик мусора среды исполнения .NET выполнит свою работу, она будет окончательно уничтожена.

Это означает, что не следует полагаться на деструктор в плане освобождения ресурсов, которые использовались экземпляром объекта, поскольку объект может оставаться неиспользуемым на протяжении длительного времени.

Если используемые ресурсы критичны, то это может привести к возникновению проблем.

Операция new

Ключевое слово **new** означает, что среде выполнения следует выделить необходимое количество оперативной памяти под экземпляр создаваемого объекта. Выделение памяти производится из «кучи», находящейся в распоряжении среды выполнения .NET (managed heap — управляемой кучи).

new операция используется для создания объектов и передачи управления конструкторам, например:

```
Class1 myVal = new Class1();          //Объект ссылочного типа. Создаётся в куче.
```

new также используется для обращения к конструкторам типа–значений, например:

```
int myInt = new int();                //Объект типа int размещается в стеке!
```

При определении объекта myInt ему было присвоено начальное значение 0, которое является значением по умолчанию для типа int. Следующий оператор имеет тот же самый эффект:

```
int myInt = 0;                    // Для размерного типа аналогично.
```

А вот для типов-значений конструкторов с параметрами в принципе нет!

```
int q = new int();
//int q = new int(125); // Такого нет.
```

Вложенные классы.

Классы могут быть вложенными. Предположим, необходимо описать структуру фирмы, которая состоит из нескольких отделов. Таким образом, можно описать структуру отделов внутри класса фирмы, причем имеется возможность ограничить доступ к этим классам:

```
class Company
   {
                  // описание счетного отдела
      public class Account
      {

      }
                  // описание бухгалтерии
      private class AccountBalance
      {

      }
 }
. . .
class Program
   {
      static void Main(string[] args)
      {
         Company.Account var1;

         // ошибка компиляции, т.к. класс недоступен
  Company.AccountBalance var2;
      }
   }
```

Ссылки как параметры.

Напомним, что любые переменные можно разделить на два типа: значения (или величины) и ссылки.

Переменные-ссылки не содержат данные непосредственно, они лишь указывают на ту ячейку в памяти, где хранятся реальные данные, то есть фактически содержат адрес памяти. Память под такие переменные выделяется отдельно от той памяти, в которой хранятся данные.

Напомним, что ссылки можно использовать для передачи параметров в методы и возврата значений из методов. Чтобы создать ссылку для переменных встроенных типов, необходимо использовать ключевое слово **ref.**

Рассмотрим три примера:

Текст кода:	Результат:

class Point
{
public int x;
public int y;
}
class Program
{
static void PointSet(Point p)
{
p.x=10;
p.y=20;
}
static void Main()
{
Point val1 = new Point();
PointSet(val1);
Console.WriteLine(val1.x);
Console.WriteLine(val1.y);
}
}
	10
	20
	
	создается копия, но не класса, а ссылки. Новая ссылка будет указывать на ту же память, что и предыдущая.

class Point
{
public int x;
public int y;
}
class Program
{
static void PointSet(Point p)
{
p = new Point();
p.x=10;
p.y=20;
}
static void Main()
{
Point val1 = new Point();
PointSet(val1);
Console.WriteLine(val1.x);
Console.WriteLine(val1.y);
	0
	0
	
	выделив память под другой объект класса, мы тем самым разорвали связь со старым объектом. На новый объект показывает только локальная ссылка p, которая уничтожается после завершения работы метода и подлежит удалению сборщиком мусора вместе с созданным объектом.

} }	

class Point
{
public int x;
public int y;
}
class Program
{
static void PointSet(ref Point p)
{
p = new Point();
p.x=10;
p.y=20;
}
static void Main()
{
Point val1 = new Point();
PointSet(ref val1);
Console.WriteLine(val1.x);
Console.WriteLine(val1.y);
}
}
	10
	20
	
	не создается вторая копия ссылки, а передаётся переменная-ссылка в метод, используя механизм передачи по ссылке. И создавая новый объект, используется не копия ссылки на класс, а все та же старая ссылка, которая была объявлена в методе Main(). Поэтому объект, созданный в методе Main() подлежит удалению сборщиком мусора, а ссылка будет указывать на новый объект, созданный в методе класса.

Оператор is

Позволяет проверить соответствие типов, не прибегая к конвертированию (см. пример UsingInher).

```
class Point
{
  public int x;
  public int y;
}
class Program
{
  static void Main()
  {
    object refToPoint = new object();
    if (refToPoint is Point)
    {
      Point val = (Point)refToPoint;
      Console.WriteLine(val.ToString());
    }
    else
```

```
            {
                Console.WriteLine("Invalid cast");
            }

        }
    }
```

Оператор as

Операция **as** позволяет выполнить безопасное преобразование типа за один шаг, без необходимости первоначальной проверки его на совместимость, но если этого сделать нельзя, то результатом проверки его выполнения будет NULL (см. пример UsingInher).

```
class Point
    {
        public int x;
        public int y;
    }
    class Program
    {
        static void Main()
        {
            object refToPoint = new object();
            Point val = refToPoint as Point;
            if (val!=null)
            {
                Console.WriteLine(val.ToString());
            }
            else
            {
                Console.WriteLine("Invalid cast");
            }

        }
    }
```

Переменные только для чтения и константы.

Среди полей класса можно выделить такие два типа, как:

- константы,
- «только для чтения» (readonly).

Константные переменные определяются исключительно при объявлении. Значение константной переменной изменить невозможно:

```
class Point
    {
        public const double pi=3.14;

        static void Main()
        {
```

```
            pi++;   //ошибка компиляции
        }
    }
```

В отличие от констант, **readonly** – переменные инициализируются уже на этапе выполнения, и компилятору не обязательно знать их значение на этапе компиляции.

Инициализация осуществляется тремя способами:

1) используется значение переменной по умолчанию:

```
class ReadOnlyTest
   {
      public readonly int test;
   }
```

после компиляции test=0;

2) устанавливается заданное значение, используя конструктор:

```
class Program
   {

      class ReadOnlyTest
      {
         public readonly int test;
         public ReadOnlyTest(int i)
         {
            test = i;
         }
      }

      static void Main()
      {
         ReadOnlyTest t = new ReadOnlyTest(5);
         Console.WriteLine(t.test);
      }
}
```

3) инициализируется непосредственно при объявлении:

```
class ReadOnlyTest
      {
         public readonly int test=10;
         public readonly ArrayList test2 = new ArrayList();

      }
      static void Main()
      {
         ReadOnlyTest t = new ReadOnlyTest();
         Console.WriteLine(t.test);
      }
```

Ключевое слово static.

1) Если ключевое слово применяется к переменной класса, то речь идет о статической переменной.

Статическая переменная содержится в классе, но выделение памяти под нее происходит один раз, причем не зависимо от того, существует ли объект данного класса. Эта переменная ассоциируется не с объектом, а с классом и создается на этапе загрузки класса. Каждый объект класса, осуществляющий доступ к этой переменной, будет использовать один и тот же участок памяти.

2) Если ключевое слово применяется к классу, то речь идет о **статическом классе**.

Это классы, которые способны содержать только статические данные и методы. Основное их назначение – создание утилитных классов, т.е. таких классов, методы которых вызываются без создания объекта класса.

```
static class TestStr
    {
      public static void Mystr()
      {
        Console.WriteLine("My string");
      }
    }

  class Program
  {
    static void Main()
    {
      TestStr.Mystr();
    }
  }
```

3) **Статические конструкторы** – это те методы класса, которые будут гарантированно вызваны до создания хоть одного объекта класса, причем только один раз. Чаще всего применяется для инициализации статических переменных.

```
 class StaticTest
    {
      public static int i;
      static StaticTest()
      {
        i = 10;
        Console.WriteLine("Static Constructor");
      }
    }

  class Program
  {
    static void Main()
    {
```

```
        StaticTest t = new StaticTest();
        StaticTest t2 = new StaticTest();
        StaticTest t3 = new StaticTest();
    }
}
```

В результате на экране отобразиться сообщение "Static Constructor" только один раз.

Задания для лабораторной работы

Вариант 1

Описать класс, реализующий бинарное дерево, обладающее возможностью добавления новых элементов, удаления существующих, поиска элемента по ключу, а также последовательного доступа ко всем элементам.

Написать программу, использующую этот класс для представления англо-русского словаря. Программа должна содержать меню, позволяющее выполнить проверку всех методов класса. Предусмотреть возможность создания словаря из файла и с клавиатуры.

Вариант 2

Построить систему классов для описания плоских геометрических фигур: круг, квадрат, прямоугольник. Предусмотреть методы для создания объектов, перемещения на плоскости, изменения размеров и вращения на заданный угол. Написать программу, демонстрирующую работу с этими классами. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 3

Построить описание класса, содержащего информацию о почтовом адресе организации. Предусмотреть возможность раздельного изменения составных частей адреса, создания и уничтожения объектов этого класса.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 4

Составить описание класса для определения одномерных массивов целых чисел (векторов). Предусмотреть возможность обращения к отдельному элементу массива с контролем выхода за пределы массива, возможность задания произвольных границ индексов при создании объекта и выполнения операций поэлементного сложения и вычитания массивов с одинаковыми границами индексов, умножения и деления всех элементов массива на скаляр, вывода на экран элемента массива по заданному индексу и всего массива.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 5

Составить описание класса для определения одномерных массивов строк фиксированной длины. Предусмотреть возможность обращения к отдельным строкам массива по индексам, контроль выхода за пределы массива, выполнения операций поэлементного сцепления двух массивов с образованием нового массива, слияния двух массивов с исключением повторяющихся элементов, вывода на экран элемента массива по заданному индексу и всего массива.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 6

Составить описание класса одномерных массивов строк, каждая строка задается длиной и указателем на выделенную для нее память. Предусмотреть возможность обращения к отдельным строкам массива по индексам, контроль выхода за пределы массивов, выполнения операций поэлементного сцепления двух массивов с образованием нового массива, слияния двух массивов с исключением повторяющихся элементов, вывода на экран элемента массива и всего массива Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 7

Составить описание класса, обеспечивающего представление матрицы произвольного размера с возможностью изменения числа строк и столбцов, вывод на экран подматрицы любого размера и всей матрицы.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 8

Описать класс «домашняя библиотека». Предусмотреть возможность работы с произвольным числом книг, поиска книги по какому-либо признаку (например, по автору или по

году издания), добавления книг в библиотеку, удаления книг из нее, сортировки книг по разным полям.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 9

Описать класс «записная книжка». Предусмотреть возможность работы с произвольным числом записей, поиска записи по какому-либо признаку (например, по фамилии, дате рождения или номеру телефона), добавления и удаления записей, сортировки по разным полям.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 10

Описать класс «студенческая группа». Предусмотреть возможность работы с переменным числом студентов, поиска студента по какому-либо признаку (например, по фамилии, дате рождения или номеру телефона), добавления и удаления записей, сортировки по разным полям.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 11

Описать класс, реализующий стек. Написать программу, использующую этот класс для отыскания прохода по лабиринту.

Лабиринт представляется в виде матрицы, состоящей из квадратов. Каждый квадрат либо открыт, либо закрыт. Вход в закрытый квадрат запрещен. Если квадрат открыт, то вход в него возможен со стороны, но не с угла. Каждый квадрат определяется его координатами в матрице. После отыскания прохода программа печатает найденный путь в виде координат квадратов.

Вариант 12

Описать класс «предметный указатель». Каждый компонент указателя содержит слово и номера страниц, на которых это слово встречается. Количество номеров страниц, относящихся к одному слову, от одного до десяти. Предусмотреть возможность формирования указателя с клавиатуры и из файла, вывода указателя, вывода номеров страниц для заданного слова, удаления элемента из указателя. Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.