Лабораторная работа №13

ХАРЬКОВСКИЙ НАЦИОНАЛЬНЫЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И МЕХАТРОНИКИ

Кафедра информационных технологий и мехатроники

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по проведению лабораторных работ по дисциплине «Алгоритмизация и программирование»
для студентов специальности 6.050101 “Компьютерные науки”

Разработчик - доцент кафедры информационных технологий и мехатроники
кандидат технических наук, старший научный сотрудник
Тимонин Владимир Алексеевич

Харків 2015

Лабораторная работа №13
Исследование возможностей интегрированной среды разработки Visual C# для создания приложений по обработке структур данных.

Цель работы – исследовать возможности интегрированной среды разработки Visual Studio 2010 и получить практические навыки по созданию приложений, обрабатывающие структуры данных.

1. Теоретические сведения

Структуры – это составные типы данных, построенные с использованием других типов. Они представляют собой объединенный общим именем набор данных различных типов и методов их обработки. Структуры — это типы значений, они обрабатываются напрямую, а не через ссылки, что означает, что при использовании структур расходуется меньший объем памяти Кроме того, благодаря прямому доступу к структурам, при работе с ними не снижается производительность, что имеет место при доступе к объектам классов.

1.1. Объявление структуры

В C# структуры объявляются с использованием ключевого слова struct и синтаксически подобны классам. Формат записи структуры имеет вид:

```
struct имя
    {
     // объявления членов
    }
```

где **имя** означает имя структуры. Членами (элементами) структур могут быть методы, поля, индексаторы, свойства, операторные методы и события. Структуры могут также определять конструкторы, но не деструкторы.

Структуры могут быть простыми, например:

```
struct Coordinate
    {
      public int x;
      public int y;
     }
```

или сложными (включают различные типы данных, методы и др.), например:

```
struct Point
    {
     public int X;
     public int Y;
     public void Increment()
            {
              X++; Y++;
            }
     public void Decrement ()
            {
              X--; Y--;
            }
     public void Display()
            {
              Console.WriteLine("X = {0}, Y= {1}", X, Y) ;
            }
```

}

Структура **Coordinate** имеет два целых члена с именами **x** и **y**, которые могут описывать точки на карте. Структура **Point** имеет два целых члена с именами **X** и Y, которые могут описывать позицию точки на экране монитора, три метода с именами **Increment()** – добавление 1 к позиции (X,Y), **Decrement ()** – вычитание 1 из позиции (X,Y), **Display()** – отображение текущей позиции.

В структурах определены элементы с использованием ключевого слова **public**, которое представляет собой один из модификаторов управления доступом. Объявление данных с использованием ключевого слова **public** гарантирует наличие у вызывающего кода возможности напрямую получать доступ к элементам структур (через операцию точки).

1.2. Создание и использование переменных типа структур

Объявление структуры представляет собой простую схему, которая описывает элементы структуры. Для создания переменной типа структуры доступно несколько вариантов:

● Что бы иметь возможность использовать структуру, необходимо сначала объявить переменную структурного типа

Point p1;

с последующим присваиванием значений каждому из ее общедоступных элементов данных типа полей перед вызовом ее членов. Для получения доступ к отдельным членам структуры, используется точечную нотацию, в которой **точка** называется оператором доступа к члену струк туры. Выражение **p1.X** ссылается на член типа int структуры **p1** и может, таким образом, использоваться в любом месте, где допускается пере менная типа int.

```
Point p1;
p1.X = 10;
p1.Y = 20;
p1.Display(); // выводит на экран монитора X=10, Y=20
```

Если значения общедоступным элементам структуры (в данном случае X и Y) не присвоены перед ее использованием, компилятор сообщит об ошибке.

```
Point p2;
p2.X = 10;
p2.Display(); // Ошибка! Элементу Y не было присвоено значение
```

● В качестве альтернативного варианта переменные типа структур можно создавать с применением ключевого слова **new**, поддерживаемого в C#, что предусматривает вызов для структуры конструктора по умолчанию. По определению используемый по умолчанию конструктор не принимает никаких аргументов. Преимущество подхода с вызовом для структуры конструктора по умолчанию состоит в том, что в таком случае каждому элементу данных полей автоматически присваивается соответствующее значение по умолчанию:

```
Point p1 = new Point ();
p1.Display(); // выводит на экран монитора X=0, Y=0
```

● Создавать структуру можно также с помощью специального конструктора, что позволяет указывать значения для полей данных при создании переменной, а не устанавливать их для каждого из них по отдельности. Для создания специального конструктора добавим в структуру **Point** следующий код:

```
public Point (int XPos, int YPos)
    {
      X = XPos;
      Y = YPos;
    }
```

После этого переменные типа **Point** можно создавать как:

```
Point p2 = new Point (50, 60);
p2.Display (); // выводит на экран монитора X=50, Y=60
```

Пример 1. Нижеприведенный пример демонстрирует использование структуры для хранения информации о книге. Результат выполнения приложения«Сведения о книгах» приведен на рис. 1.

```
// Определение структуры
struct Book
    {
     public string author; // автор
     public string title; // название
     public int year; // год издания
     public Book(string a, string t, int c)
         {
          author = a;
          title = t;
          year = c;
         }
    }
public static void Main()
    {
     Console.Title = "  Сведения о книгах";
     Console.BackgroundColor = ConsoleColor.White;
     Console.Clear();
     Console.ForegroundColor = ConsoleColor.Black;
     // Создание объекта без вызова конструктора
     Book book1;
     // Инициализация вручную
     book1.title = "Библия C#";
     book1.author = "Фленов М.Е.";
     book1.year = 2011;
     Console.WriteLine("  " + book1.author + ", " + book1.title + ", " +
                              book1.year + "\n");
     // Вызов конструктора по умолчанию. Инициализация по умолчанию
     Book book2 = new Book();
     if (book2.title == null)
       {
        Console.WriteLine("  При инициализации по умолчанию элемент
                              book2.title содержит null!");
        Console.WriteLine("  Содержимое структуры  book2: " + book2.author +
                              ", " + book2.title + ", " + book2.year + "\n");
       }
     // Запись в структуру book2 данных
     book2.title = "Язык программирования C# 2010 и платформа .NET 4.0";
     book2.author = "Троелсен Э.";
     book2.year = 2011;
     Console.Write("  После заполнения структура  book2 содержит:\n");
     Console.WriteLine("  " + book2.author + ", " + book2.title + ", " +
                             book2.year);
     Console.WriteLine();
     // Вызов явно заданного конструктора
     Book book3 = new Book("Голощапов А.Л.", "Microsoft Visual Studio 2010",
                              2011);
     Console.WriteLine("  " + book3.author + ", " + book3.title + ", " +
```

```
                book3.year + "\n");
        Console.Write("Для завершения работы приложения нажмите
                клавишу <Enter>");
        Console.Read();
}
```

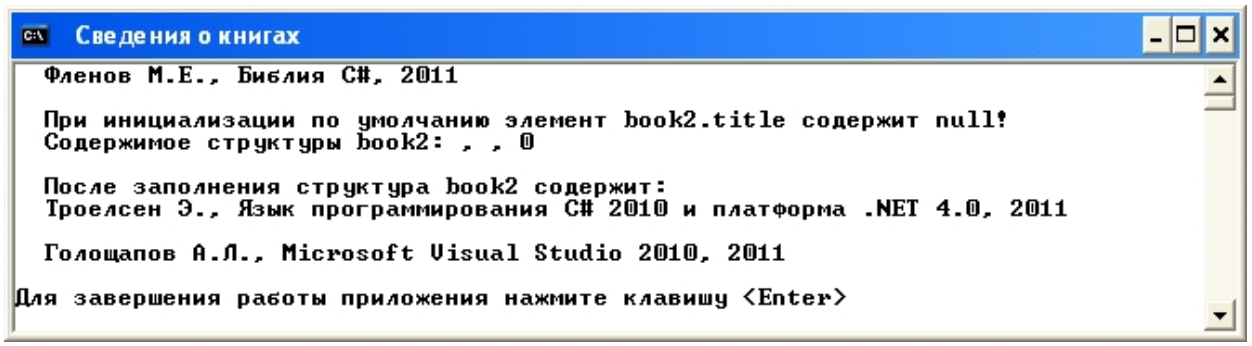

Рис. 1. Результат выполнения приложения«Сведения о книгах»

При присваивании одной структуры другой создается копия этого объекта, например:

```
struct MyStruct
    {
      public int x;
     }
public static void Main()
     {
      MyStruct a;
      MyStruct b;
      a.x = 10;
      b.x = 20;
      Console .WriteLine ("a.x {0}, b.x {1}", a.x, b.x); // a.x=10, b.x=20
      a = b;
      b.x = 30;
      Console.WriteLine("a.x {0}, b.x {1}", a.x, b.x); // a.x=20, b.x=30
     }
```

После операции присваивания (**a = b;**) структурные переменные **a** и **b** по-прежнему не зависят одна от другой, т.е. переменная **a** никак не связана с переменной **b**.

1.3. Массивы структур

Так как массив в C# — это коллекция переменных одинакового типа, обращение к которым происходит с использованием общего для всех имени, а структура является объектом, то можно организовать работу с массивами структур.

Для работы с массивами структур сначала объявляется структура, например, **Book** (см. пример1), после чего объявляется ссылочная переменная на массив, а затем для него выделяется память, и переменной массива присваивается ссылка на эту область памяти, например, с помощью операторов:

```
Book[ ] masBooks;
masBooks = new Book [10];
```

или

```
Book[ ] masBooks = new Book [10];
```

создается массив структур **masBooks** (состоящий из 10 структур типа **Book**), который связывается со ссылочной переменной массива **masBooks**. Переменная **masBooks** содержит ссылку на область памяти, выделенную оператором **new**. Таким образом, в C# массивы структур размещаются в памяти с

помощью оператора **new**.

Доступ к отдельному элементу массива осуществляется посредством индекса (индекс описывает позицию элемента внутри массива, в C# первый элемент массива имеет нулевой индекс), а элементу структуры – посредством имени элемента структуры, например, оператор:

Console.WriteLine(masBooks[1].title);

выдает на экран монитора название 2 книги, оператор

masBooks[7].author = "Фленов М.Е.";

записывает фамилию автора в структуру, находящуюся на 8-й позиции массива.

2. Рабочее задание

Задание 1. Руководствуясь теоретическим материалом раздела 1 изучить возможности языка C# по созданию приложений, обрабатывающие структуры данных, и выполнить практически все примеры, описанные в этом разделе.

Задание 2. Разработать приложение с заголовком «Расстояние между точками», с помощью которого можно рассчитать расстояние между точками $A_1(x_1, y_1, z_1)$ и $A_2(x_2, y_2, z_2)$. Для описания местоположения точек в пространстве A_1 и A_2 использовать структуры. Блок расчета расстояния оформить в виде метода. Исходные данные (координаты точек A_1 и A_2) вводятся с клавиатуры. Результат вычислений выводятся на экран монитора из метода **Main()**.

Задание 3. Разработать приложение с заголовком «Сведения о занятиях», с помощью которого можно получить сведения об общем количестве занятий по всем дисциплинам в семестре. Сведения о дисциплине (название, количество лекций, количество лабораторных работ, фамилия преподавателя, общее количество занятий) хранятся в структуре **Predmet**. Для расчета общего количества занятий по дисциплине использовать отдельный метод в структуре.
Исходные данные (сведения о 4-х дисциплинах) вводятся с клавиатуры.

Задание 4. Модифицировать приложение «Сведения о занятиях» таким образом, чтобы использовать массив структур типа **Predmet** (приложению дать название «Сведения о занятиях - модификация»). Количество дисциплин и сведения о дисциплинах вводятся с клавиатуры.

3. Контрольные вопросы

1. Что такое структура и ее назначение.
2. Как объявляется структура?
3. Перечислите способы создания переменных типа структуры.
4. Как осуществляется доступ к элементам структуры?
5. Как объявляется массив структур?
6. Как осуществляется доступ к элементам массива структур?

Литература

1. Голощапов А.Л. Microsoft Visual Studio 2010. – СПб.:БХВ-Петербург, 2011. – 544 с.: ил.
2. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 1. Пер. с англ. - М.: «Русская Редакция», 2002.- 576 с.: ил.
3. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 2. Пер. с англ. - М.: «Русская Редакция», 2002.- 624 с.: ил.
4. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4.0. Пер. с англ. - М.: Издательский дом "Вильямс", 2011. — 1392 с.: ил.
5. Фленов М.Е. Библия C#. - СПб.: БХВ-Петербург, 2011.– 560с.: ил.
6. Шилдт Г. C# Учебный курс. – СПб.: Питер, Издательская группа BHV, 2003. – 512 с.: ил.