Семинар 8. Функции

Как ты думаешь, может ли в какой-нибудь огромной нефтяной компании работать только один человек? А что, сам бы и скважины бурил, сам бы нефть качал, и одновременно ездил бы на огромной дорогой машине к президенту страны на совещания. Очевидно, что это предположение нелепо. Почему? Да потому, что есть предел человеческим возможностям. Так же и в программировании – когда размер программы превышает некоторый максимум, разобраться в ней становится совершенно невозможно.

Программисты нуждаются в возможности разбить программу на несколько отдельных частей, чтобы она напоминала составной механизм, в котором одну деталь, можно заменить на другую, какие-то части временно вывести из строя и прикрутить новые элементы, добавляющие функциональность. В программировании эти составные части называются функциями.

8.1 Что такое функции

Функции, как уже было сказано ранее, можно сравнить с деталями автомобиля. Еще одна хорошая ассоциация – это функции в математике, у которых каждому значению аргумента соответствует одно единственное значение функции.

С помощью функций, можно разбить программу на несколько частей точно так же, как можно разделить автомобиль на двигатель, кузов, колеса, коврики и т.д. Каждая их этих частей, хотя и сложна, но все же несколько проще, чем автомобиль целиком.

Функции хороши еще и тем, что позволяют избежать ненужного копирования кусков кода, который может быть использован несколько раз в похожих ситуациях. Достаточно один раз придумать колесо и использовать его четыре раза, нежели заново придумывать каждое по отдельности. К тому же, это упрощает последующее редактирование программы.

8.1.1 Объявление функции

В языке C# каждая функция имеет свой тип, указывающий ей на то, какого рода значение она должна вернуть. Например, если функция целого типа, она обязана вернуть целочисленное значение и т.д.

```
В общем виде:
тип модификаторы название_функции (параметры функции)
{
      // тело функции
}

Пример:
int summ (int a, int b)
{
      int c;
      c = a + b;
      return c;
}
```

Бывают ситуации, когда не требуется, чтобы функция возвращала какие бы то ни было значения. В этом случае ее можно объявить с использованием специального нулевого типа void, который можно использовать только применительно к функциям.

8.1.2 Вызов функции и ее параметры

Набор действий, которые должна выполнить функция, называется ее телом. Эти действия выполняются каждый раз при вызове функции, а вызвать ее можно, обратившись к функции по имени.

```
//описание функции
static void MyFunction(string Stroka)
{
      Console.WriteLine(Stroka);
}
...
//вызов функции
MyFunction("Привет, Функция");
```

Для передачи в тело функции каких-либо данных извне предназначены параметры функции, которые так же называются ее аргументами. Все доступные параметры следует указать при объявлении функции в круглых скобках. Каждый параметр объявляется с указанием типа, если параметров несколько, их следует отделить друг от друга запятыми.

```
int mult (int a, int b)
{
      int c;
      c = a * b;
      return mult;
}

Console.Write(mult(7, 3));
```

В этом примере мы описали функцию mult с двумя параметрами «a» и «b» целочисленного типа, которая возвращает результат умножения этих чисел друг на друга. Все внутренние названия аргументов имеют смысл только во время работы функции, и они будут забыты программой сразу же после ее завершения. Возвращает функция только то значение, которое будет указано после оператора «return».

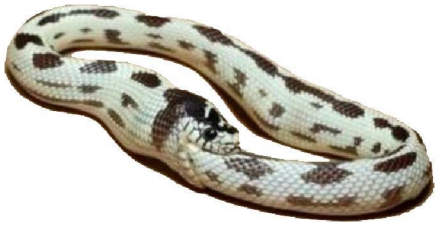
Для закрепления умений напишем функцию, которая будет возводить число в степень. Само число будет передаваться в первом параметре, а степень во втором. Для полноты картины приведем текст программы полностью, чтобы ты смог увидеть, что код функции следует вставлять перед определением функции main.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication2
{
    class Program
    {
        static Int32 MyPow(Int32 a, Int32 b)
        {
            Int32 r = a;
            for (Int32 i = 1; i < b; i++)
            {
                r = r * a;
            }
            return r;
        }
```

```
        static void Main(string[] args)
        {
            Console.WriteLine(MyPow(2, 3));
            Console.ReadKey();
        }
    }
}
```

8.1.3 Рекурсивный вызов функции

В программировании рекурсией называют вызов функции из нее же самой, непосредственно или через другие функции. Количество вложенных вызовов функции называется глубиной рекурсии.



Рекурсия используется довольно часто при работе с данными, имеющими древовидную структуру. Например, Интернет браузер рекурсивно обрабатывает Html-код, прежде чем отобразит страницу. Классическим же примером, хорошо иллюстрирующим полезность рекурсии, является функция для расчета факториала. С факториалом мы уже сталкивались ранее, но тогда решили эту задачу через цикл.

```
class Program
{
    static UInt64 fact(UInt64 n)
    {
        if (n <= 1)
            return 1;
        else
            return n * fact(n - 1);
    }

    public static void Main(string[] args)
    {
        Console.WriteLine(fact(5));
        Console.ReadLine();
    }
}
```

Рекурсия возможна благодаря тому, что каждый следующий рекурсивный вызов этой функции пользуется своим набором локальных переменных и за счет этого работает корректно. Оборотной стороной этого довольно простого по структуре механизма является то, что на каждый рекурсивный вызов требуется некоторое количество оперативной памяти компьютера, и при чрезмерно большой глубине рекурсии может наступить переполнение «стека» вызовов. Следующий код позволяет посмотреть максимально возможные уровень вложенности.

```
class Program
{
    static void a (UInt64 n)
    {
            Console.WriteLine(n);
            a(n + 1);
    }
    public static void Main(string[] args)
    {
            a(1);
    }
}
```

Когда-то давно, когда компьютеры были маломощными, а память стоила очень дорого, под стек выделялось не так много ресурсов, поэтому обычно рекомендовалось избегать рекурсивных алгоритмов, которые могут приводить к слишком большой глубине рекурсии. В настоящее время ситуация изменилась к лучшему. Если ты выполнишь несколько тестов, то увидишь, что глубина вложенности измеряется десятками тысяч, что довольно много.

В то же время, какой бы ни был у тебя компьютер, бесконечная рекурсия все равно приведет к фатальной ошибке, поэтому с ней нужно быть осторожнее.

8.2 Функция Main и параметры программы

Ранее мы довольно часто сталкивались с функциями, но не задавались особо вопросом, что это такое Вспомни такие команды, как «Console.ReadLine()» и «Console.WriteLine()», а так же «Convert.ToInt32()». У всех этих команд в конце были круглые скобки. Теперь ты понимаешь, что они означают. Ну и, наконец, тот самый Main, в котором мы пишем свои программки, тоже является функцией.

```
static void Main(string[] args)
{
      //тело функции
}
```

Слово «void» мы уже обсудили – это специальный тип функции, который позволяет ей не возвращать вообще никакого значения. Далее следует название функции и аргументы функции в скобках. И если с названием все понятно, то на аргументах необходимо остановиться подробнее.

У программ, как и у функций, могут быть параметры, которые в этом случае называются «ключами». Пользователь может определить значения параметров при запуске программы из командной строки, указав их через пробел после имени выполняемого файла. Например, ранее мы уже запускали команду с параметрами «ping mail.ru». Само по себе имя «ping» означает, что в системных папках есть соответствующая ей программа «ping.exe», название домена следует после пробела и является первым параметром вызова программы.

Число принимаемых программами параметров может быть довольно велико, поэтому разработчики всегда встраивают в них подсказки, получить которые можно, если запустить программу с ключом «/?», например «ping /?».

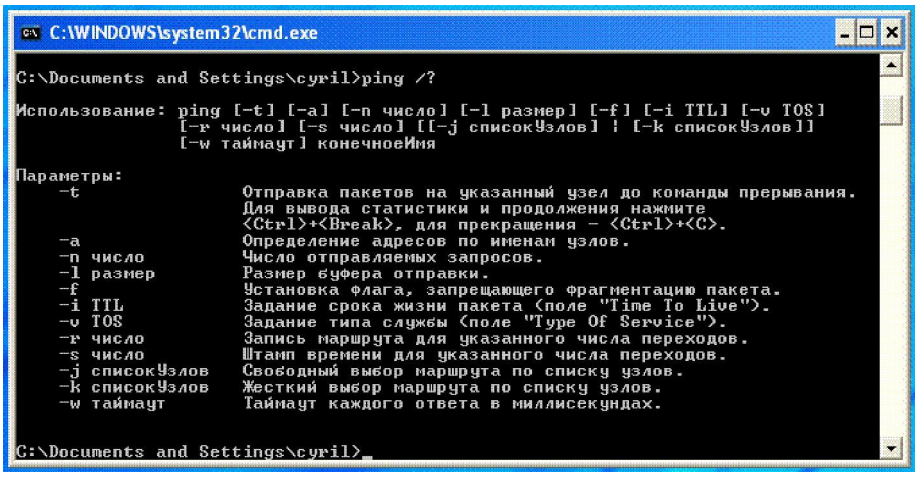


Рис. 8-5 Запуск программы ping с ключом /?

Чтобы прочитать параметры, которые пользователь ввел с клавиатуры при запуске программы, нам следует обратиться к массиву строк args, который является параметром функции Main. Первый параметр будет в элементе с нулевым индексом args[0], счет ведется, как обычно, с нуля. С помощью следующего кода можно получить полный список параметров, которые были переданы в программу

```
static void Main(string[] args)
{
      for (Int32 i=0; i<10; i++)
      {
            if (args.Length>0)
            {
                  Console.WriteLine("Параметр{0} = {1}", i, args[0]);
            }
      }
}
```

8.3 Использование функций на примере квеста

Область применения функций огромна, но для того чтобы нам было интересней, я вам скажу, что при помощи функций очень удобно программировать, так называемые, текстовые квесты – простейшие игры без графики, в которых сюжет развивается в зависимости от того, как игрок отвечает на вопросы. Скорее всего, вы хоть раз видели подобные игры, но даже если это не так, разобраться что к чему будет не так уж сложно.

Такие игры состоят из локаций (комнат), в которых можно совершать какие-либо действия и (или) общаться с персонажами игры. Кроме того, между локациями можно перемещаться. Пройти игру можно, только если правильно выполнить все задания игры и оказаться в нужной локации.

Думаю, что общие принципы понятны. Давай теперь для примера напишем маленький квест, состоящий из трех локаций: коридор, прихожая, комната. Целью игры будет попадание в из коридора в комнату. Естественно, что квест получится слишком очевидным, но это не страшно, так как мы пока еще только тренируемся.

```
static void Korridor()
{
      Console.WriteLine("вы в корридоре, видите дверь");
      Console.WriteLine("что делать?");
      Console.WriteLine("1 - войти в дверь");
      Console.WriteLine("2 - ковырять в носу");
```

```
      string str = Console.ReadLine();
      switch(str)
      {
            case "1": {
                        Prihozhaya();
                        break;
                  }
            case "2": break;
      }
}
static void Prihozhaya()
{
      Console.WriteLine("вы в прихожей, видите дверь,
                  зеркало и проход в комнату");
      Console.WriteLine("что делать?");
      Console.WriteLine("1 - войти в дверь");
      Console.WriteLine("2 - войти в проход");
      string str = Console.ReadLine();
```

```
      switch(str)
      {
            case "1": {
                        Korridor();
                        break;
                        }
            case "2": {
                        Komnata();
                        break;
                        }
      }
}
static void Komnata()
{
      Console.WriteLine("вы в своей комнате,
                  видите комнату, компьютер, кровать и кучу мусора");
      Console.WriteLine("Игра окончена");
      Console.ReadLine();
      Environment.Exit(1);
}

public static void Main(string[] args)
{
      while (true) Korridor();
}
```

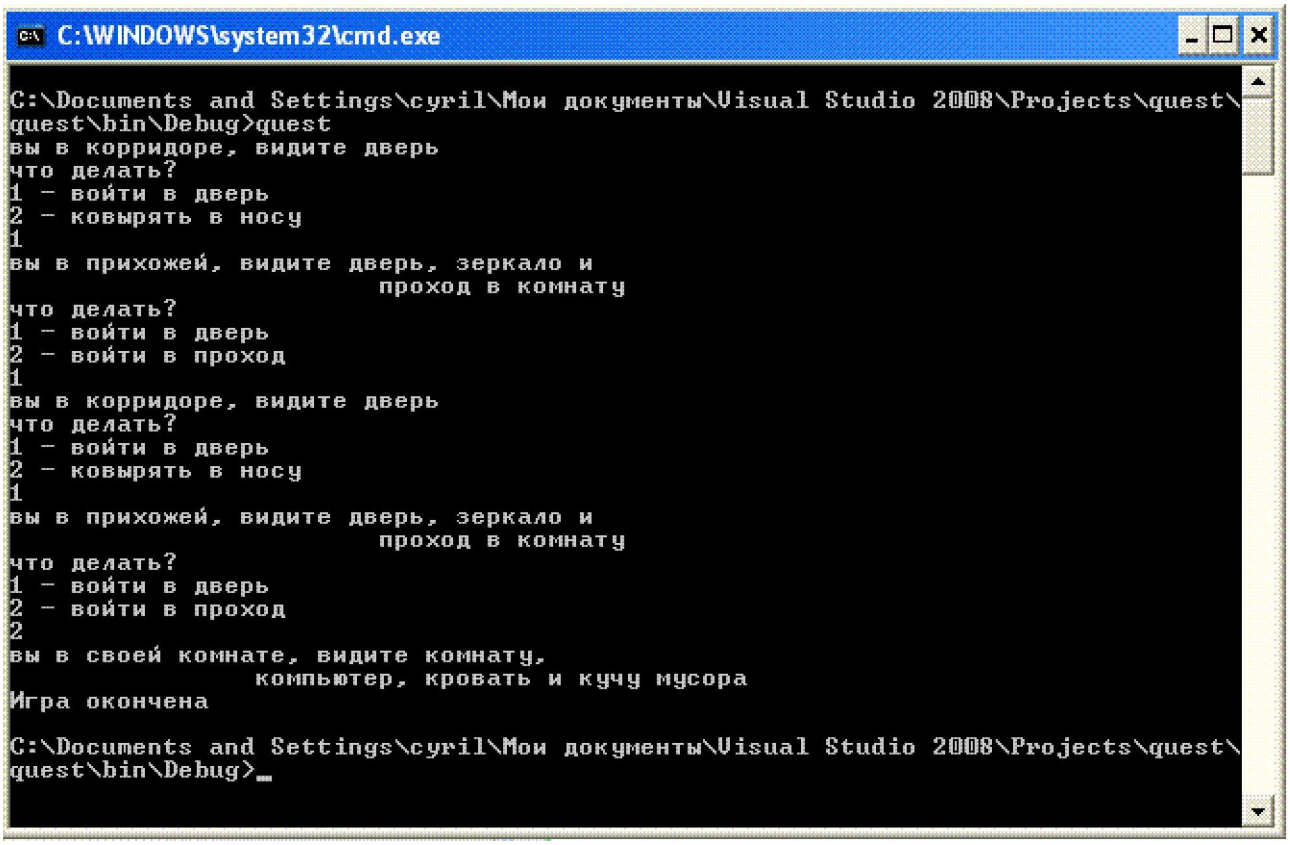


Рис. 8-6 Окно программы «Квест»

Как видите, даже для такого несложного квеста нам пришлось написать не так уж и мало строк. Естественно, что более-менее осмысленный квест был бы в несколько десятков раз больше! Так что без функций, ну совсем не обойтись.

Задание для работы в классе

#1 Напиши программу, в которой описана функция, возводящая в квадрат число, введенное с клавиатуры.

#2 Напиши программу, в которой описана функция, цель которой состоит в вычислении среднего арифметического двух чисел, введенных пользователем

#3 Напиши функцию, которая сравнивает два целых числа и возвращает один символ: >, < или =.

#4 Напиши функцию, которая сравнивает два целых числа и возвращает один символ: >, < или =.

#5 Напиши функцию, которая считает количество гласных букв во введенной строке.

#6 Напиши функцию, которая генерирует и выводит на экран всевозможные сочетания цифр и букв в пароле из 4 символов. Для того, чтобы превратить число в букву используй явное преобразование

```
Int32 n = 219;
Char c = (char)n;
Console.Write(c);
n = (int)c;
Console.Write(n);
Console.ReadLine();
```