Программирование

Тема 7.3 Строки в C#

Символы и массивы символов в C#

Пример 1: Использование методов структуры System.Char

```
B c 2
b C 2
Введите символ: A
Введен символ A, его код - 65
Буква
Введите символ: 5
Введен символ 5, его код - 53
Число
```

```
using System;
namespace ConsoleApplication
{ class Program
  { static void Main(string[] args)
      { try
          { char b = 'B', c = '\x63', d = '\u0032';
              Console.WriteLine("{0} {1} {2}", b, c, d);
              Console.WriteLine("{0} {1} {2}", char.ToLower(b), char.ToUpper(c),
                  char.GetNumericValue(d));
              char a;
              do
              { Console.Write("Введите символ: "); a = char.Parse(Console.ReadLine());
                Console.WriteLine("Введен символ {0}, его код - {1}", a, (int)a);
                if (char.IsLetter(a)) Console.WriteLine("Буква");
                if (char.IsNumber(a)) Console.WriteLine("Число");
                if (char.IsWhiteSpace(a)) Console.WriteLine("Пробельный символ");
                if (char.IsPunctuation(a)) Console.WriteLine("Разделитель");
              } while(a != 'q');

             // Console.ReadLine();
          }
          catch { Console.WriteLine("Возникло исключение"); }
          Console.ReadLine();
      }
  }
}
```

Символы и массивы символов в C#

Пример 2: Работа с массивом символов

```
using System;
namespace ConsoleApplication
{ class Program
  { static void Main(string[] args)
      { char[] a = {'m','a','s','s','i','v'};
        char[] b = "abcdba".ToCharArray();
        PrintArray("Исходный массив a:", a);
        int pos = Array.IndexOf(a, 'm');
        a[pos] = 'M';
        PrintArray("Измененный массив a:", a);
        PrintArray("Исходный массив b:", b);
        Array.Reverse(b);
        PrintArray("Измененный массив b:", b);
        Console.ReadLine();
      }
    static void PrintArray(string h, Array a)
      { Console.WriteLine(h);
        foreach (object el in a)
              Console.Write(el);
        Console.WriteLine("\n");
      }
    }
}
```

```
Исходный массив a:
massiv

Измененный массив a:
Massiv

Исходный массив b:
abcdba

Измененный массив b:
abdcba
```

Символы и массивы символов в C#

- В C# для обработки текстовой информации применяются отдельные символы, массивы символов, изменяемые и неизменяемые строки и регулярные выражения.
- Символьный тип char предназначен для хранения одиночного символа в кодировке Unicode. Символ в памяти занимает 2 байта. Тип char является псевдонимом типа System.Char библиотеки .NET.
- В структуре System.Char определены статические методы, позволяющие задавать вид и категорию символа, а также преобразовывать символ в верхний или нижний регистр и в число.
- В примере 1 продемонстрировано использование некоторых методов типа System.Char.
- В примере 1 символьные переменные b, c и d инициализируются символьными константами в различных формах представления (обычной, шестнадцатеричной и в виде escape-последовательности Unicode). В цикле анализируется вводимый с клавиатуры символ. Можно вводить и управляющие символы, используя сочетание клавиши Ctrl с латинскими буквами.
- Массив символов, как и массив любого иного типа, построен на основе класса Array.
- Пример 2 демонстрирует работу с массивом символов. Символьный массив можно инициализировать, либо непосредственно задавая его элементы, либо применяя метод ToCharArray класса System.String, который разбивает исходную строку на отдельные символы.

Пример 3: Способы создания строк, операции со строками

```
using System;
namespace ConsoleApplication
{ class Program
  { static void Main(string[] args)
    { string s1 = "hello";
      string s2 = null;
      string s3 = new String('a', 6);
      string s4 = new String(new char[] { 'w', 'o', 'r', 'l', 'd' });
      s2 = s1 + " world";
      Console.WriteLine(s1);
      Console.WriteLine(s2);
      if (s1 == s2) Console.WriteLine("Строки равны");
          else Console.WriteLine("Строки не равны");
      Console.WriteLine(s3);
      Console.WriteLine(s4);
      Console.WriteLine(s4[0]);
      Console.ReadLine();
    }
  }
}
```

```
hello
hello world
Строки не равны
aaaaaa
world
w
```

Строки типа string в C#

- В языке C# строки описываются типом **string**, который является псевдонимом для класса **System.String**. Объект этого класса представляет текст как последовательность символов Unicode.
- Создавать сроки можно, как используя переменную типа string и присваивая ей значение, так и применяя один из конструкторов класса String (пример 3).
- Для строк определены следующие операции:
  - присваивание (=);
  - проверка на равенство (==);
  - проверка на неравенство (!=);
  - обращение по индексу ([ ]);
  - сцепление (конкатенация) строк (+).
- Несмотря на то что строки являются ссылочным типом данных, на равенство и неравенство проверяются не ссылки, а значения строк. Строки равны, если имеют одинаковое количество символов и совпадают посимвольно.
- Обращаться к отдельному элементу строки по индексу можно только для получения значения, но не для его изменения. Это связано с тем, что ***строки типа string являются неизменяемыми***. Методы, изменяющие содержимое строки, на самом деле создают новую копию строки. Неиспользуемые «старые» копии автоматически удаляются сборщиком мусора.

Строки типа string в C#

Пример 4: Объединение и сравнение строк

```
using System;
namespace ConsoleApplication
{ class Program
  { static void Main(string[] args)
    { string s1 = "Привет"; string s2 = "мир";
      string s3 = s1 + " " + s2;
      string s4 = String.Concat(s3, "!!!");
      Console.WriteLine(s4);
      string s5 = "apple"; string s6 = "a day";
      string s7 = "keeps"; string s8 = "a doctor";
      string s9 = "away";
      string[] values = new string[] { s5, s6, s7, s8, s9 };
      String s10 = String.Join(" ", values);
      Console.WriteLine(s10);
      if (s1.Equals(s2)) Console.WriteLine("Строки равны");
          else Console.WriteLine("Строки не равны");
      if (String.Compare(s1, s2) == 0) Console.WriteLine("Строки равны");
          else Console.WriteLine("Строки не равны");
      if (s1.CompareTo(s2) == 0) Console.WriteLine("Строки равны");
          else Console.WriteLine("Строки не равны");
      Array.Sort(values);
      String s11 = String.Join(" ", values);
      Console.WriteLine(s11);
      Console.ReadLine();
    }
  }
}
```

```
Привет мир!!!
apple a day keeps a doctor away
Строки не равны
Строки не равны
Строки не равны
a day a doctor apple away keeps
```

Пример 5: Поиск в строке, разделение строк

```
4
6
И
поэтому
все
так
произошло
```

```
using System;
namespace ConsoleApplication
{ class Program
    { static void Main(string[] args)
        {  string s1 = "hello world";
            char ch = 'o';
            int indexOfChar = s1.IndexOf(ch); // равно 4
            Console.WriteLine(indexOfChar);
            string subString = "wor";
            int indexOfSubstring = s1.IndexOf(subString); // равно 6
            Console.WriteLine(indexOfSubstring);

            string text = "И поэтому все   так произошло";
            // string[] words = text.Split(new char[] { ' ' });
            string[] words = text.Split(new char[] { ' ' },
                                StringSplitOptions.RemoveEmptyEntries);
            foreach (string s in words)
                Console.WriteLine(s);
            Console.ReadLine();
        }
    }
}
```

Строки типа string в C#

- В классе System.String предусмотрено множество методов, полей и свойств для работы со строками.
- В примере 4 показано использование методов класса System.String, которые используются для объединения (конкатенации) и сравнения строк.
- Статический метод Concat выполняет сцепление (конкатенацию) строк.
- Статический метод Join выполняет слияние массива строк в единую строку. Между элементами массива вставляются разделители.
- Статический метод Compare и метод CompareTo принимают две строки и возвращают число. Если первая строка по алфавиту стоит выше второй, то возвращается число больше нуля. В противном случае возвращается число меньше нуля. И третий случай - если строки равны, то возвращается число 0.
- Статический метод Compare и метод CompareTo предназначены в основном для использования при сортировке строк.
- Для сравнения строк удобно использовать метод Equals или операцию сравнения (==). Метод Equals возвращает true, если строки равны и false – в противном случае.
- С помощью метода IndexOf мы можем определить индекс первого вхождения отдельного символа или подстроки в строке (пример 5).
- Подобным образом действует метод LastIndexOf, только находит индекс последнего вхождения символа или подстроки в строку.
- С помощью функции Split можно разделить строку на массив подстрок. В качестве параметра функция Split принимает массив символов или строк, которые и будут служить разделителями (пример 5). string[] words = text.Split(new char[] { ' ' }) - это не лучший способ разделения по пробелам, так как во входной строке могло бы быть несколько подряд идущих пробелов и в итоговый массив также бы попадали пробелы, поэтому лучше использовать другую версию метода: string[] words = text.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries). Второй параметр StringSplitOptions.RemoveEmptyEntries задает удаление всех пустых подстрок.

Пример 6: Обрезка строки

```
using System;
namespace ConsoleApplication
{ class Program
   { static void Main(string[] args)
      { string text = " hello world ";
        text = text.Trim();
        Console.WriteLine(text);
        text = text.Trim(new char[] { 'd', 'h' });
        Console.WriteLine(text);
        text = "Хороший день";
        // обрезаем начиная с третьего символа
        text = text.Substring(2);
        Console.WriteLine(text);
        // обрезаем сначала до последних двух символов
        text = text.Substring(0, text.Length - 2);
        Console.WriteLine(text);
        Console.ReadLine();
      }
   }
}
```

```
hello world
ello worl
роший день
роший де
```

Строки типа string в C#

Пример 7: Вставка строк, удаление и замена в строке

```
using System;
namespace ConsoleApplication
{ class Program
  { static void Main(string[] args)
    { string text = "Хороший день";
      string subString = "замечательный ";
      text = text.Insert(8, subString);
      Console.WriteLine(text);
      text = "Хороший день";
      // индекс последнего символа
      int ind = text.Length - 1;
      // вырезаем последний символ
      text = text.Remove(ind);
      Console.WriteLine(text);
      // вырезаем первые два символа
      text = text.Remove(0, 2);
      Console.WriteLine(text);
      text = "хороший день";
      text = text.Replace("хороший", "плохой");
      Console.WriteLine(text);
      text = text.Replace("о", "");
      Console.WriteLine(text);
      Console.ReadLine();
    }
  }
}
```

```
Хороший замечательный день
Хороший ден
роший ден
плохой день
плхй день
```

Строки типа string в C#

- Для обрезки начальных или концевых символов используется функция Trim (пример 6). Функция Trim без параметров обрезает начальные и конечные пробелы и возвращает обрезанную строку. Чтобы явным образом указать, какие начальные и конечные символы следует обрезать, можно передать в функцию массив этих символов.
- Функция Trim имеет частичные аналоги: функция TrimStart обрезает начальные символы, а функция TrimEnd обрезает конечные символы.
- Обрезать определенную часть строки позволяет функция Substring (пример 6). Функция Substring также возвращает обрезанную строку. В качестве параметра первая использованная версия применяет индекс, начиная с которого надо обрезать строку. Вторая версия применяет два параметра - индекс начала обрезки и длину вырезаемой части строки.
- Для вставки одной строки в другую применяется функция Insert (пример 7). Первым параметром в функции Insert является индекс, по которому надо вставлять подстроку, а второй параметр - собственно подстрока.
- Удалить часть строки помогает метод Remove (пример 7). Первая версия метода Remove принимает индекс в строке, начиная с которого надо удалить все символы. Вторая версия принимает еще один параметр - сколько символов надо удалить.
- Чтобы заменить один символ или подстроку на другую, применяется метод Replace (пример 7). Во втором случае применения функции Replace строка из одного символа "o" заменяется на пустую строку, то есть фактически удаляется из текста. Подобным способом легко удалять какой-то определенный текст в строках.

Пример 8: Создание строк StringBuilder

```
using System;
using System.Text;
namespace ConsoleApplication
{ class Program
  { static void Print(StringBuilder sb)
    { Console.WriteLine("Длина строки: {0}", sb.Length);
      Console.WriteLine("Емкость строки: {0}", sb.Capacity);
    }
    static void Main(string[] args)
    { StringBuilder sb1 = new StringBuilder("Название: ", 40);
      Print(sb1);
      sb1.Append(" Руководство по C++");
      Print(sb1); Console.WriteLine();
      StringBuilder sb2 = new StringBuilder("Название: ");
      Print(sb2);
      sb2.Append(" Руководство");
      Print(sb2);
      sb2.Append(" по C++");
      Print(sb2);
      sb2.Append(" и C#");
      Print(sb2); Console.WriteLine();
      StringBuilder sb3 = new StringBuilder("Название: Руководство");
      Print(sb3);
      sb3.Append(" по Java");
      Print(sb3); Console.ReadLine();
    }
  }
}
```

```
Длина строки: 10
Емкость строки: 40
Длина строки: 29
Емкость строки: 40

Длина строки: 10
Емкость строки: 16
Длина строки: 22
Емкость строки: 32
Длина строки: 29
Емкость строки: 32
Длина строки: 34
Емкость строки: 64

Длина строки: 21
Емкость строки: 21
Длина строки: 29
Емкость строки: 42
```

Класс StringBuilder в C#

- Объект класса String представляет собой неизменяемую строку. Когда выполняется какой-нибудь метод класса String, система создает новый объект в памяти с выделением ему достаточного места.
- Под объект класса String выделяется ровно столько памяти, сколько необходимо для хранения данной строки.
- Класс StringBuilder из пространстве имен System.Text представляет динамическую строку.
- Под объект класса StringBuilder обычно выделяет больше памяти, чем нужно в данный момент. Класс StringBuilder имеет два главных свойства:
  - Length, показывающее длину строки, содержащуюся в объекте в данный момент;
  - Capacity, указывающее максимальную длину строки, которая может поместиться в выделенную для объекта память.
- В классе StringBuilder имеется конструкторы, которые позволяют задавать начальное значение свойству Capacity, то есть указать, сколько именно памяти должен быть выделено под строку.
- Если не использовать эти конструкторы , то будет выбран объем выделяемой памяти по умолчанию, который зависит от размера начального текста, инициализирующего экземпляр StringBuilder. Если этот размер меньше или равен 16, то память будет выделяться под 16 символов, если этот размер больше 16, то память будет выделена под этот размер (пример 8).
- Метод Append класса StringBuilder выполняет добавление в конец строки. Разные варианты метода позволяют добавлять в строку величины любых встроенных типов, массивы символов, строки и подстроки типа string.

Пример 9: Использование методов класса StringBuilder

```
using System;
using System.Text;
namespace ConsoleApplication
{ class Program
  { static void Main(string[] args)
    { StringBuilder sb = new StringBuilder("Привет мир!");
      sb.Insert(7, "компьютерный ");
      Console.WriteLine(sb);
      sb.Replace("мир", "world");
      Console.WriteLine(sb);
      sb.Remove(7, 13);
      string s = sb.ToString();
      Console.WriteLine(s);
      Console.Write("Введите зарплату: ");
      double salary = double.Parse(Console.ReadLine());
      StringBuilder a = new StringBuilder();
      a.Append("зарплата ");
      a.AppendFormat("{0 ,15:C}", salary);
      Console.WriteLine(a);
      Console.ReadLine();
    }
  }
}
```

```
Привет компьютерный мир!
Привет компьютерный world!
Привет world!
Введите зарплату: 5000
зарплата       5 000,00р.
```

Класс StringBuilder в C#

- Кроме метода Append класс StringBuilder предлагает еще ряд методов для операций над строками (пример 9):
  - Insert: вставляет подстроку в объект StringBuilder, начиная с определенного индекса
  - Remove: удаляет определенное количество символов, начиная с определенного индекса
  - Replace: заменяет все вхождения определенного символа или подстроки на другой символ или подстроку
  - AppendFormat: добавляет форматированную строку в конец строки.
  - ToString: преобразует строку типа StringBuilder в строку string.

- Любые модификации строки происходят внутри блока памяти, выделенного экземпляру StringBuilder. Это делает добавление подстрок и замену индивидуальных символов строки очень эффективными. Удаление или вставка подстрок остаются менее эффективными, потому что при этих операциях приходится перемещать в памяти части строки. Выделять новую память и, возможно, полностью перемещать ее содержимое приходится только при выполнении ряда действий, которые приводят к превышению выделенной емкости строки.

Контрольные вопросы

1. Строки типа string в C#: понятие, способы создания, допустимые операции, основные методы класса System.String. Примеры.
2. Строки класса StringBuider в C#: понятие, способы создания, методы класса StringBuider. Примеры.