Символы и строки постоянной длины

Лекция №9

Строковый тип

Строковый тип – это:

- **отдельные символы, чаще всего, его называют типом char;**
- **строки постоянной длины, часто они представляются массивом символов;**
- строки переменной длины – это, как правило, тип string, соответствующий современному представлению о строковом типе.

Класс char

- В C# есть **символьный класс char**, основанный на классе System.Char и использующий двухбайтную кодировку Unicode представления символов.

Класс char

Константу можно задавать:

- символом, заключенным в одинарные кавычки;
- escape-последовательностью;
- Unicode-последовательностью, задающей Unicode код символа.

Класс char. Пример работы

- char ch1=’A', ch2 =’\x5A’, ch3=’\u0058′;
- char ch = new Char();
- int code; string s;
- ch = ch1;
- *//преобразование символьного типа в тип int*
- code = ch; ch1=(char) (code +1);
- *//преобразование символьного типа в строку*
- *//s = ch;*
- s = ch1.ToString()+ch2.ToString()+ch3.ToString();
- Console.WriteLine(«s= {0}, ch= {1}, code = {2}»,
- s, ch, code);
- Результат: BZX

Escape – последовательности

Escape-последовательность	Имя символа	Кодировка Юникода
\'	Одинарная кавычка	0x0027
\"	Двойная кавычка	0x0022
\\	Обратная косая черта	0x005C
\0	Null	0x0000
\a	Звуковой сигнал (звонок)	0x0007
\b	Удаление предыдущего символа	0x0008
\f	Подача страницы (для перехода к началу следующей страницы)	0x000C

Escape – последовательности

Escape-последовательность	Имя символа	Кодировка Юникода
\n	Новая строка	0x000A
\r	Возврат каретки	0x000D
\t	Горизонтальная табуляция	0x0009
\u	Escape-последовательность Юникода	\u0041 = "A"
\v	Вертикальная табуляция	0x000B
\x	Escape-последовательность Юникода аналогична "\u", за исключением строк с переменной длиной.	\x0041 = "A"

Статические методы и свойства класса char

Метод	Описание
GetNumericValue	Возвращает численное значение символа, если он является цифрой, и (-1) в противном случае.
GetUnicodeCategory	Все символы разделены на категории. Метод возвращает Unicode категорию символа. Ниже приведен пример.
IsControl	Возвращает true, если символ является управляющим.
IsDigit	Возвращает true, если символ является десятичной цифрой.
IsLetter	Возвращает true, если символ является буквой.
IsLetterOrDigit	Возвращает true, если символ является буквой или цифрой.

Статические методы и свойства класса char

IsLower	Возвращает true, если символ задан в нижнем регистре.
IsNumber	Возвращает true, если символ является числом (десятичной или шестнадцатеричной цифрой).
IsPunctuation	Возвращает true, если символ является знаком препинания.
IsSeparator	Возвращает true, если символ является разделителем.
IsSurrogate	Некоторые символы Unicode с кодом в интервале [0x1000, 0x10FFF] представляются двумя 16-битными «суррогатными» символами. Метод возвращает true, если символ является суррогатным.
IsUpper	Возвращает true, если символ задан в верхнем регистре.
IsWhiteSpace	Возвращает true, если символ является «белым пробелом». К белым пробелам, помимо пробела, относятся и другие символы, например, символ конца строки и символ перевода каретки.

Статические методы и свойства класса char

Parse	Преобразует строку в символ. Естественно, строка должна состоять из одного символа, иначе возникнет ошибка.
ToLower	Приводит символ к нижнему регистру.
ToUpper	Приводит символ к верхнему регистру.
MaxValue, MinValue	Свойства, возвращающие символы с максимальным и минимальным кодом. Возвращаемые символы не имеют видимого образа.

Примеры работы

```
Console.WriteLine("Статические методы класса char:");
char ch = 'a', ch1 = '1', lim = ';', chc = '\xA';
double d1, d2;
d1 = char.GetNumericValue(ch);
d2 = char.GetNumericValue(ch1);
Console.WriteLine("Метод GetNumericValue:");
Console.WriteLine("sym 'a' - value {0}", d1);
Console.WriteLine("sym '1' - value {0}", d2);
```

```
Статические методы класса char:
Метод GetNumericValue:
sym 'a' - value -1
sym '1' - value 1
```

Примеры работы

```
System.Globalization.UnicodeCategory cat1, cat2;
cat1 = char.GetUnicodeCategory(ch1);
cat2 = char.GetUnicodeCategory(lim);
Console.WriteLine("Метод GetUnicodeCategory:");
Console.WriteLine("sym '1' - category {0}", cat1);
Console.WriteLine("sym ';' - category {0}", cat2);
Console.WriteLine("Метод IsControl:");
Console.WriteLine("sym '\xA' - IsControl - {0}",
   char.IsControl(chc));
Console.WriteLine("sym ';' - IsControl - {0}",
   char.IsControl(lim));
```

```
Метод GetUnicodeCategory:
sym '1' - category DecimalDigitNumber
sym ';' - category OtherPunctuation
Метод IsControl:
sym '
' - IsControl - True
sym ';' - IsControl - False
```

Примеры работы

```
Console.WriteLine("Метод IsSeparator:");
Console.WriteLine("sym ' ' - IsSeparator - {0}",
   char.IsSeparator(' '));
Console.WriteLine("sym ';' - IsSeparator - {0}",
   char.IsSeparator(lim));
Console.WriteLine("Метод IsSurrogate:");
Console.WriteLine("sym '\u10FF' - IsSurrogate - {0}",
    char.IsSurrogate('\u10FF'));
Console.WriteLine("sym '\\' - IsSurrogate - {0}",
    char.IsSurrogate('\\'));
string str = "\U00010F00";
// Символы Unicode в интервале [0x10000,0x10FFF]
// представляются двумя 16-
//суррогатными символами
```

```
Метод IsSeparator:
sym ' ' - IsSeparator - True
sym ';' - IsSeparator - False
Метод IsSurrogate:
sym '?' - IsSurrogate - False
sym '\' - IsSurrogate - False
```

Примеры работы

```
Console.WriteLine("str = {0}, str[0] = {1}",
    str, str[0]);
Console.WriteLine("str[0] IsSurrogate - {0}",
    char.IsSurrogate(str, 0));
Console.WriteLine("Метод IsWhiteSpace:");
str = "пробелы, пробелы!" + "\xD" + "\xA" +
    "Всюду пробелы!";
Console.WriteLine("sym '\xD ' - IsWhiteSpace - {0}",
    char.IsWhiteSpace('\xD'));
Console.WriteLine("str: {0}", str);
```

```
str = ??, str[0] = ?
str[0] IsSurrogate - True
Метод IsWhiteSpace:
 ' - IsWhiteSpace - True
str: пробелы, пробелы!
Всюду пробелы!
```

Примеры работы

```
Console.WriteLine("и ее пробелы - символ 8 {0}," +
   "символ 17 {1}",
   char.IsWhiteSpace(str, 8),
   char.IsWhiteSpace(str, 17));
Console.WriteLine("Метод Parse:");
str = "A";
ch = char.Parse(str);
Console.WriteLine("str:{0}  char: {1}", str, ch);
Console.WriteLine("Минимальное и максимальное " +
   "значение: {0}, {1}",
   char.MinValue.ToString(),
   char.MaxValue.ToString());
Console.WriteLine("Их коды: {0}, {1}",
    SayCode(char.MinValue), SayCode(char.MaxValue));
```

```
и ее пробелы - символ 8 True,символ 17 True
Метод Parse:
str:A  char: A
Минимальное и максимальное значение:  , ?
Их коды: 0, 65535
```

Класс char[]

- В языке C# определен класс Char[], и его можно использовать для представления строк постоянной длины.

Класс char[]

В C# не определены взаимные преобразования между классами String и Char[], даже явные. Однако:

У класса String есть динамический метод ToCharArray, задающий подобное преобразование.

Возможно также посимвольно передать содержимое переменной string в массив символов.

```
// ошибка: нет преобразования класса string в класс char[]
// char[] strM1 = "Здравствуйте!";
// а надо так:
string hello = "Здравствуйте!";
char[] strM1 = hello.ToCharArray();
// вывод на экран посимвольно
for(int i = 0; i < strM1.Length; i++)
        Console.Write(strM1[i]);
Console.WriteLine();
// копирование подстроки методом класса Array
char[] World = new char[3];
Array.Copy(strM1,12,World,0,3);
// вывод массива с преобразованием в строку
Console.WriteLine(CharArrayToString(World));
```

Обратный перевод(Char[]->string)

- К сожалению, обратная операция не определена, поскольку метод ToString, которым, конечно же, обладают все объекты класса Char[], печатает информацию о классе, а не содержимое массива. Ситуацию легко исправить, написав соответствующий цикл. Например:
- **string** result = "";
- **for**(**int** i = 0; i < strM1.Length; i++)
- result = result + strM1[i];
-
- **return**(result);

Существует ли в C# строки типа char*

- В языке C# указатели допускаются в блоках, отмеченных как небезопасные. Теоретически в таких блоках можно объявить переменную типа **char*,** рассматривая ее как строку. В C# строки типа char* использовать не рекомендуется.