Introduction to Object Oriented Programming

ADVANCED DIPLOMA

MCAST Technical College

INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY

Strings and Enumerated Types

CLASS NOTES

C#.NET Language Fundamentals

CLASS NOTES

Table of Contents

`

The String Data Type

The `string` data type allows text to be stored as a sequential collection of Unicode characters.

To declare a variable of type `string` you may use the following syntax:

```
string <variable name>;
```

For example:

```
string name;
```

To initialise a variable of type `string` remember to include the value within " " as follows:

```
string name;
name = "Tommy";
```

To initialise a variable of type string to empty or zero-length string use "" or `String.Empty`

For example:

```
string bookTitle = "";
```

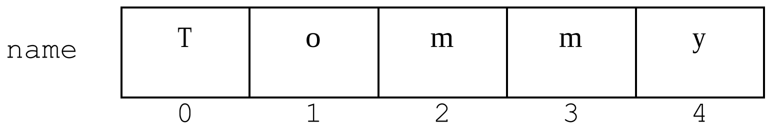
or

```
string bookTitle = String.Empty;
```

`

Accessing characters within a string

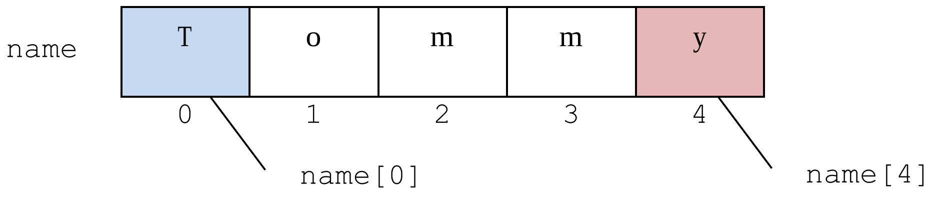
A string variable stores a sequence of characters each of which has a number (called index) starting from 0. For example:



Each character within a string can be accessed individually by using [] as follows:

```
<variable name> [<index>]
```

For example:



The following program shows how to display characters within a string:

```
string name = "Tommy";

Console.WriteLine("The first letter is : " + name[0]);
Console.WriteLine("The last letter is : " + name[4]);
```

Note that it is not allowed to change the contents of a string by using []:

```
name[0] = 't'; //invalid code
```

Note also that if an invalid index is used the following `IndexOutOfRangeException` will be thrown:

```
Index was outside the bounds of the array.
```

String Operators

There are several useful operators which allow you to handle strings easily:

Operator	Purpose	Example

+	Concatenate two or more string variables in one	`string name = "Tom";`
		`string surname = "Grech";`
		`string fullname = name + " " + surname;`

[ ]	Access individual characters within a string, by using the index	`char letter = name[0];`

==	Returns true if two strings have the same value	`string str1 = Console.ReadLine();`
		`string str2 = Console.ReadLine();`
		`bool check = str1 == str2;`

!=	Returns true if two strings do not have the same value	`string str1 = Console.ReadLine();`
		`string str2 = Console.ReadLine();`
		`bool check = str1 != str2;`

Escape characters

An escape character is a special sequence of characters in a string literal. An escape character always starts with a \ and is followed by another character. Each escape character has a particular purpose defined by the following table:

Escape Character	Purpose
\\	Backslash (\). As the backward slash character defines as escape character sequence, the double-backslash is required to indicate the insertion of a single backslash.
\n	New Line. Inserts a new line into the string literal. When outputted, the output starts a new line of text when the control character is reached.
\t	Horizontal Tab. Inserts a horizontal tab into a text string, moving the current position to the next tab stop in a similar manner to pressing the tab key in a word processor.
\a	Alert. In some scenarios this character sounds an alert through the computer's speaker.
\r	Carriage Return. This is similar to the new line character when used for screen output. Some printers use this as an indicator to return to the start of the current line. In this case, to begin a new line both a carriage return and new line character (\r\n) are required.
\v	Vertical Tab. Inserts a vertical tab.
\uxxxx	Inserts the character with the Unicode character number xxxx. The xxxx portion is a four digit hexadecimal number.

`

Useful methods and properties to handle strings

Unlike other simple data types such as `int`, `char` or `byte`, the `string` data type is a reference type. We say a string variable is an object, an instance of the class `string`. Since `string` is a class, we can use several properties and methods which are readily available within the class, to handle our string variables.

The following table illustrates some of the most useful properties and methods to handle string variables:

Property	Purpose	**Example**
Length	Gets the number of characters in the string	`string str = "abc";` `int num = str.Length;`

Method	Purpose	**Example**
Contains()	Returns true if this string contains a sequence of characters or false otherwise	`string sub = "car";` `string str = "One car";` `bool check = str.Contains(sub);`
IndexOf()	Returns the index of a character within this string or -1 if it does not exists	`string str = "Hello";` `int pos = str.Index('o');`
CompareTo()	Compares this string with another.	`string first = "Jane";` `string second = "Mary";` `int check = first.CompareTo(second);`
ToLower()	Returns a copy of this string to lowercase	`string str = "ABC";` `string strLower = str.ToLower();`
ToUpper()	Returns a copy of this string to uppercase	`string str = "abc";` `string strUpper = str.ToUpper();`
Remove()	Deletes the part of this string staring from the specified position	`string str = "Good Morning";` `str = str.Remove(0, 5);`
Substring()	Returns a substring of this string	`string str = "Good Morning";` `string copy = str.Substring(0, 5);`
Trim()	Removes all occurrences of a set of specified characters from beginning and end of this string	`string str = "    Hello ";` `str = str.Trim();`

`

Enumerated Types

An enumerated type is a type which consists of a set of named constants called an enumerator list. An enumerated type should be used when a group of related constants are required. Each constant in the enumerated list has a value, which by default, starts from 0. An enumerated type should be defined within a namespace so that all classes have access to it. Enumerated types make programs more readable and easier to maintain.

Defining and using an enumerated type

The following program shows how an enumerated type, `Days` may be defined and used.

```
namespace Enums
{
        //defining an enumerated type
    enum Days
    {
        Sun, Mon, Tue, Wed, Thu, Fri, Sat
    }

    class Program
    {
        static void Main(string[] args)
        {
             //declaring a variable of type Days
            Days oneDay;

             //assigning an enumerator value to oneDay
            oneDay = Days.Mon;

        }
    }
}
```

The first part of the program defines the enumerated type `Days` which represents the days of the week:

```
namespace Enums
{
    enum Days
    {
        Sun, Mon, Tue, Wed, Thu, Fri, Sat
    }
```

The first enumerator has a value 0, and the value of each successive enumerator is increased by 1.

```
enum Days
    {
        Sun, Mon, Tue, Wed, Thu, Fri, Sat
    }
```

Enumerator value 0

Enumerator value 6

The second part of the program declares a variable, `oneDay` of type days.

```
  //declaring a variable of type Days
        Days oneDay;
```

The variable `oneDay` can have any value within the enumerated list.

To set the value of the variable use the dot operator as follows:

```
  //assigning an enumerator value to oneDay
           oneDay = Days.Mon;
```

Sometimes it is useful to know the enumerator value of a variable such as `oneDay`.

To get the enumerator value of `oneDay` we need to use typecasting as follows:

```
  //returning the enumerator value of a variable
           int enumValue = (int) oneDay;
```