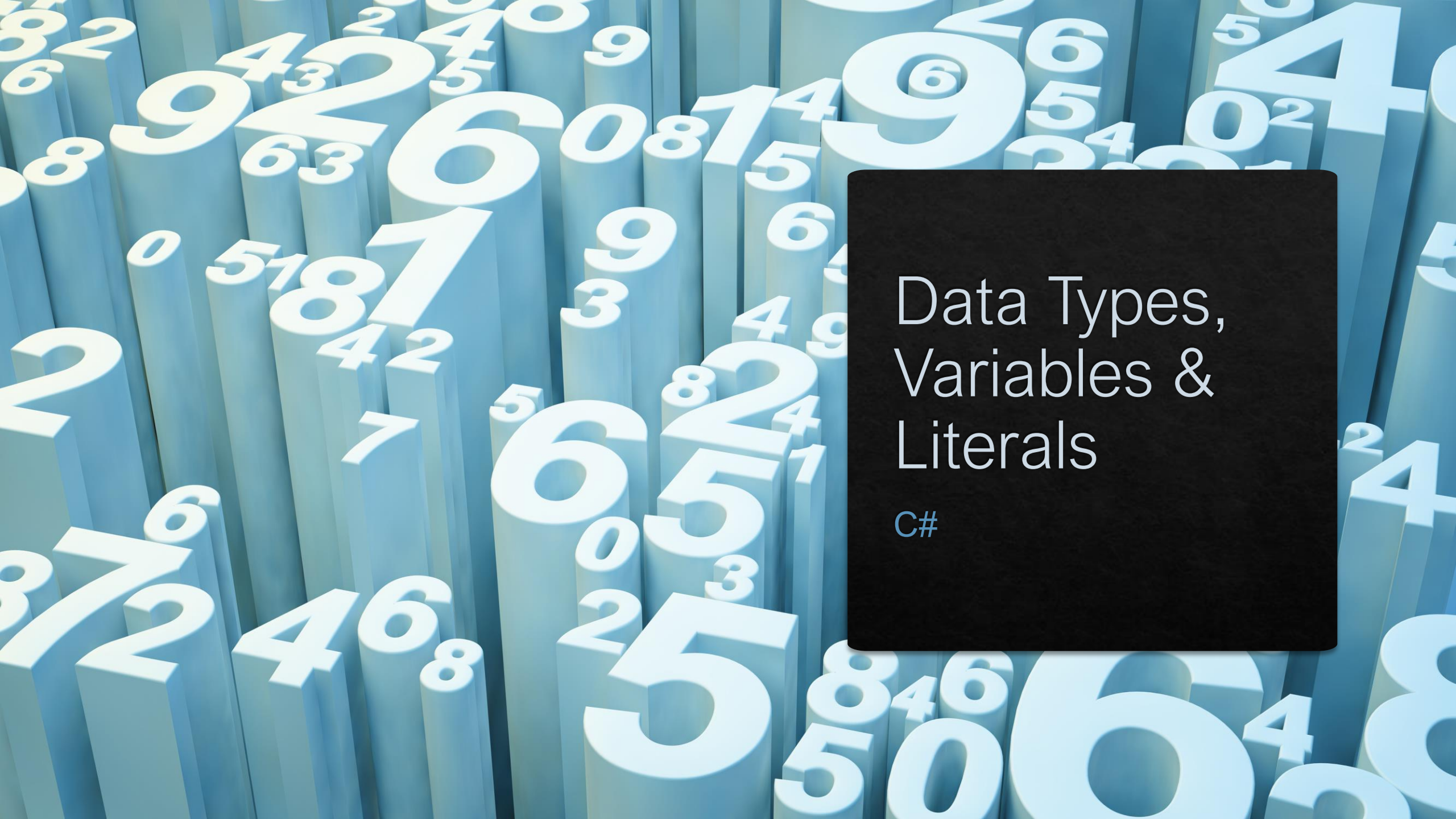


Data Types, Variables & Literals

C#

C# Basics

Data Types

Types of data that a valid C# variable can hold

Variables

Placeholder of information which can be changed at runtime

Literals

a fixed value that is used by variables

Data Types

- Specify the type of data that a valid C# variable can hold.
- C# is a strongly typed programming language
- Each type of data is predefined as part of the programming language. E.g.
  - Integer
  - Character
  - float …
- All constants or variables defined for a given program must be described with one of the data types.

Three categories

- Value Data Types
  - directly store the variable value in memory
  - Includes both signed and unsigned
  - System.ValueType is the implied parent/base class for all value types
- Reference Data Types
  - contain a memory address of variable value
  - System.Object is the implied parent/base class for all reference types
- Pointer Data Type
  - contain a memory address of the variable value
  - Associated Operators ampersand (&) and asterisk (*)
  - In C# can only be used with the unsafe keyword

1. Value Data Types

- Signed & Unsigned Integral Types

Alias	Type Name	Type	Size(bits)	Range	Default Value
sbyte	System.Sbyte	signed integer	8	-128 to 127	0
short	System.Int16	signed integer	16	-32768 to 32767	0
Int	System.Int32	signed integer	32	-2^{31} to $2^{31}-1$	0
byte	System.byte	unsigned integer	8	0 to 255	0
uint	System.UInt32	unsigned integer	32	0 to 2^{32}	0
long	System.Int64	signed integer	64	-2^{63} to $2^{63}-1$	0L
ushort	System.UInt16	unsigned integer	16	0 to 65535	0
ulong	System.UInt64	unsigned integer	64	0 to 2^{63}	0

1. Value Data Types

- Floating Point Types

Alias	Type name	Size(bits)	Range (aprox)	Default Value
float	System.Single	32	±1.5 × 10-45 to ±3.4 × 1038	0.0F
double	System.Double	64	±5.0 × 10-324 to ±1.7 × 10308	0.0D

- Decimal Type

Alias	Type name	Size(bits)	Range (aprox)	Default value
decimal	System.Decimal	128	±1.0 × 10-28 to ±7.9228 × 1028	0.0M

1. Value Data Types

- Character Types

Alias	Type name	Size In(Bits)	Range	Default value
char	System.Char	16	U +0000 to U +ffff	'\0'

- Boolean Types

Alias	Type name	Values
bool	System.Boolean	True / False

2. Reference Data Types

- Contain a memory address of variable value
- Built-in reference types
  - String
    - represents a sequence of Unicode characters and its type name is System.String. So, string and String are equivalent.
  - Object
    - All types, predefined and user-defined, reference types and value types, inherit directly or indirectly from Object
    - is the base class for all the data types in C#

More on Object class

- Since System.Object is the base class for all types (directly or indirectly)
  - All variables can be converted to System.Object or vice versa
  - Value Types inherit from System.ValueType which in turn inherits from System.Object
  - Boxing
    - When variable of a value types is converted to System.Object
  - Unboxing
    - When variable of type object is converted to a value type
  - class, interface, delegate, array are all reference data types

Stack & Heap

Stack

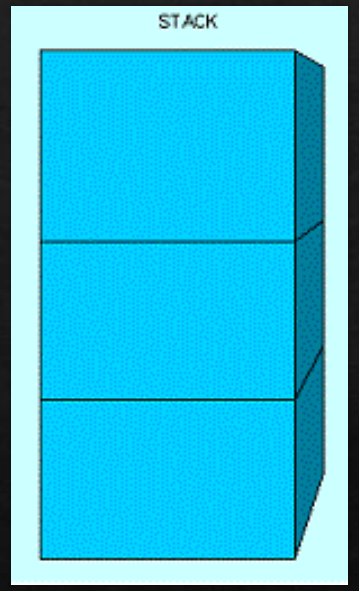
- It is an array of memory.
- It is a LIFO (Last In First Out) data structure.
- Data can be added to and deleted only from the top of it.
- Used to store entire value types as well as reference variables

Heap

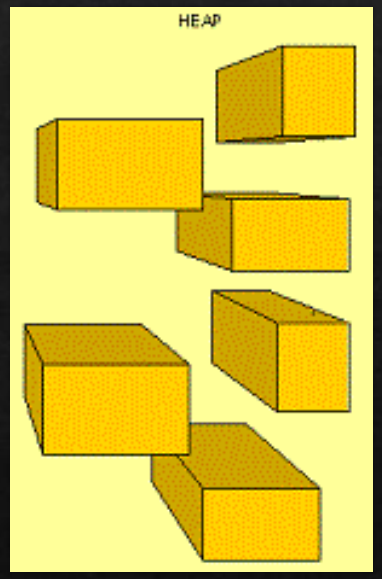
- It is an area of memory where chunks are allocated to store certain kinds of data objects.
- In it data can be stored and removed in any order.

Stack & Heap : How memory is managed

Stack

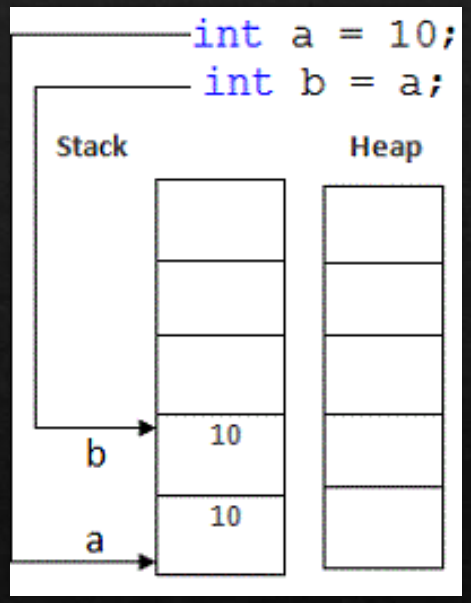


Heap

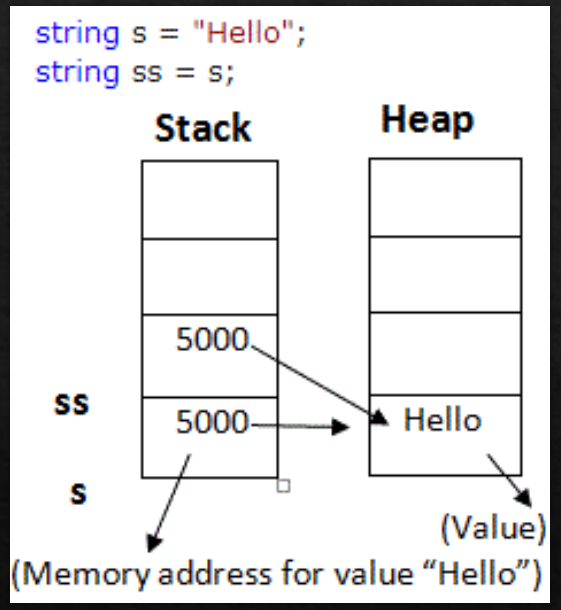


Stack & Heap: Practical Scenario

Stack



Heap



Stack & Heap: What is stored?

Stack

- "Things" declared with the following list of type declarations are Value Types
- bool, byte, char, decimal, double, enum, float, int, long, sbyte, short, struct, uint, ulong, ushort

Heap

- "Things" declared with following list of type declarations are Reference Types
- class, interface, delegate, object, string

Stack & Heap: Some more differences

Stack

- Memory allocation is Static
- It is stored Directly
- Local variables get wiped off once they lose the scope

Heap

- Memory allocation is Dynamic
- It is stored indirectly
- Only cleaned by Garbage Collector

3. Pointer Data Types

- Pointer Data Types will contain a memory address of the variable value
- To get the pointer details we have a two operators
  - ampersand (&): It is Known as Address Operator. It is used to determine the address of a variable.
  - asterisk (*): It also known as Indirection Operator. It is used to access the value of an address.
- Syntax :
  - type* identifier;
- Example :
  - int* addr1;

Variables

- A typical program uses various values that may change during its execution
- A variable stores data in RAM (Stack or Heap)
- Syntax
  - type variable_name = value;
  - or
  - type variable_names;
- Example
  - char varText = 'h'; // Declaring and Initializing character variable
  - int a, b, c; // Declaring variables a, b and c of int type

Characteristics of Variables:

- Example: char varText = 'h';
- Name
  - It must be a valid identifier. In above example, varText is a valid identifier.
- DataType
  - It defines the types of information which is to be stored into the variable. In above example char is a datatype.
- Value
  - It is the actual data which is to be stored in the variable. In above example ‘h’ is the value

Rules for Naming Variables

- Variable names can contain the letters
  - ‘a-z’
  - ‘A-Z’
  - digits 0-9
  - ‘_’ (underscore)
- The name of the variables cannot start with a digit.
- The name of the variable cannot be any C# keyword, say int, float, null, String, etc.

- Valid Variables Names
  - int age;
  - float _studentname;
  - string AgeAbove18;
- Invalid Variables Names
  - int if; // "if" is a C#keyword
  - float 12studentname; // Cannot start with digit

Defining or Declaring a Variable

- There are some rules that must be followed while declaring variables :
  - specify its type (such as int)
  - specify its name (such as interest)
  - Can provide initial value(such as 17)
- Example :
  - int geeks;
  - float interest;
  - short minutes = 17

Initializing Variables

- Initializing means to assign some value to the variable
- Variables can either be initialized
  - in the same statement as the declaration
  - or later in the code.
- Example :
  - int y = 7; // Declaring and initializing the variable at same time
  - int x; // Declaring variable x
  - x = 5; // initializing x with value 5
- Each data type has some default value which is used for uninitialized variables

Literals

- A literal is a fixed value that is used by variables.
  - // Here 100 is a constant / literal.
  - int x = 100;
- Literals can be of the following types:
  - Integer Literals
  - Floating-point Literals
  - Character Literals
  - String Literals
  - Boolean Literals

Integer Literal

- An integer literal is a number.
- The literal of integer type can be decimal, hexadecimal or binary
  - int num1 = 100; // 100 is a decimal literal, (this is the default)
  - int num3 = 0xff; // 0xff is a hexadecimal literal (0x prefix)
  - int num4 = 0b1000001; // Same as writing num4 = 65;

string literal

- string literals are enclosed in “” and stored in a string variable.
- Maximum size of a string object in memory is 2GB or about 1 billion characters
- String literals cannot contain some special characters
  - e.g.: double quotes
  - string s = "This is a "string" in C#."; // Compile Error

Escape Sequence

- Double quotes cannot be used in a c# string literal.
- An escape sequence character \ (backslash) prefix allows you to include double quotes
  - E.g.: string s = "This is a \"string\" in C#.";;
- There are other special characters which can be used in a string literal with the escape sequence.

Escape sequence	Character name
\'	Single quote
\"	Double quote
\\	Backslash
\0	Null
\a	Alert
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab

Escape Sequence

- You can use the @ prefix before a string literal to ignore escape sequences.
  - string path1 = “c:\\logs\\log1.txt”; // \\ Escape sequence
  - string path2 = @“c:\logs\log1.txt”; // @ without escape sequence
- Note: @ is only for special characters. For double quotes you cannot use @ and must use \
  - string text1 = @"This is a "string." in C#."; // error
  - string text2 = @"This is a \"string\" in C#."; // error
  - string text3 = "This is a \"string\" in C#."; // valid
- Multi line string.
  - string str = @"this is a
    multi line
    string";

String Concatenation

- Multiple strings can be concatenated with + operator.
  - string firstName = "James“, lastName = "Bond“, code = "007";
  - string agent = "Mr. " + firstName + " " + lastName + ", Code: " + code;
- A String is immutable in C#, so above code is not very efficient.
  - string firstName = "James“, lastName = "Bond“, code = "007";
  - StringBuilder sb = new StringBuilder();
  - sb.Append("Mr. ").Append(firstName).Append(" ").Append(lastName) .Append(", Code: ").Append(code);
  - Console.WriteLine(sb.ToString());

String interpolation

- With C# 6 and later
  - string firstName = "James“, lastName = "Bond“, code = "007";
  - string agent = $"Mr. {firstName} {lastName}, Code: {code}";
  - Console.WriteLine(agent);
- Just like escape sequences, if you want to include '{' or '}' inside a string literal, use two braces, "{{" or "}}“
  - string firstName = "James“
  - string str = $“{firstName} changed his behaviour at pivotal ages {{5, 11, 14, 20}}."

string class in c#

- Initialize
  - char[] chars = { 'w', 'o', 'r', 'd' };
  - string string1 = new string(chars); // create a string form char array
  - string string2 = new string('c', 20); // repeat a character ‘c’ 20 times
- Substring
  - string sentence = "This lion is scary.";
  - string word2 = sentence.Substring(5, 4);
  - Console.WriteLine(word2);
- Format
  - string.Format("At {0:t} on {0:D}, the temperature was {1:F1} degrees Fahrenheit.", DateTime.Now, 0.85)

Copy a string

- You can call the following String methods to make a copy of a string:
  - Copy creates a copy of an existing string.
    - string oldString = “Hello World”;
    - string newString = string.Copy(oldString);
  - CopyTo copies a portion of a string to a character array.
    - string strSource = “Hello";
    - char [] destination = { 'T', 'h', 'e', ' ', 'i', 'n', 'i', 't', 'i', 'a', 'l', ' ‘, 'a', 'r', 'r', 'a', 'y' };
    - strSource.CopyTo (0, destination, 4, strSource.Length);