Microsoft®
.NET

UNIT 2:
C# LANGUAGE BASIC

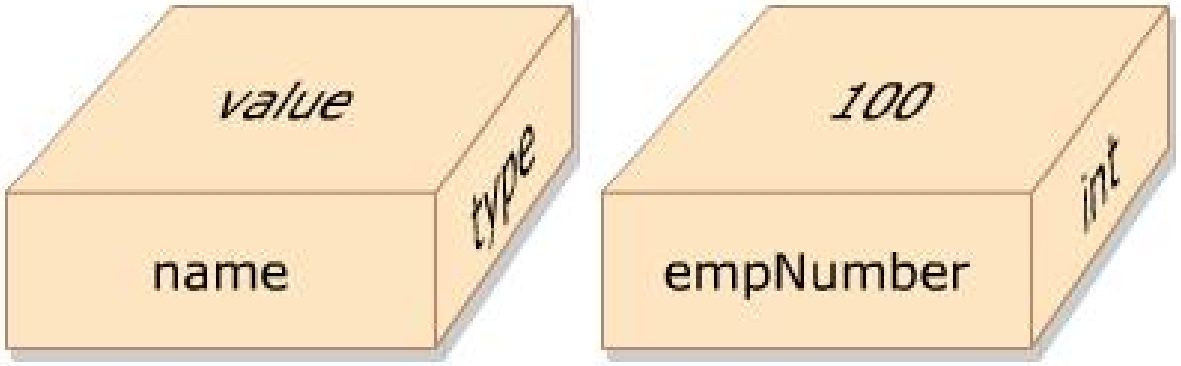
- Lesson 1. Variable and Data Types
- Lesson 2. Input and Output with Console class
- Lesson 3. Statements
- Lesson 4. Methods
- Lesson 5. Enum
- Lession 6. Array

LESSON 1. VARIABLE AND DATA TYPES

- Variables
- Data types

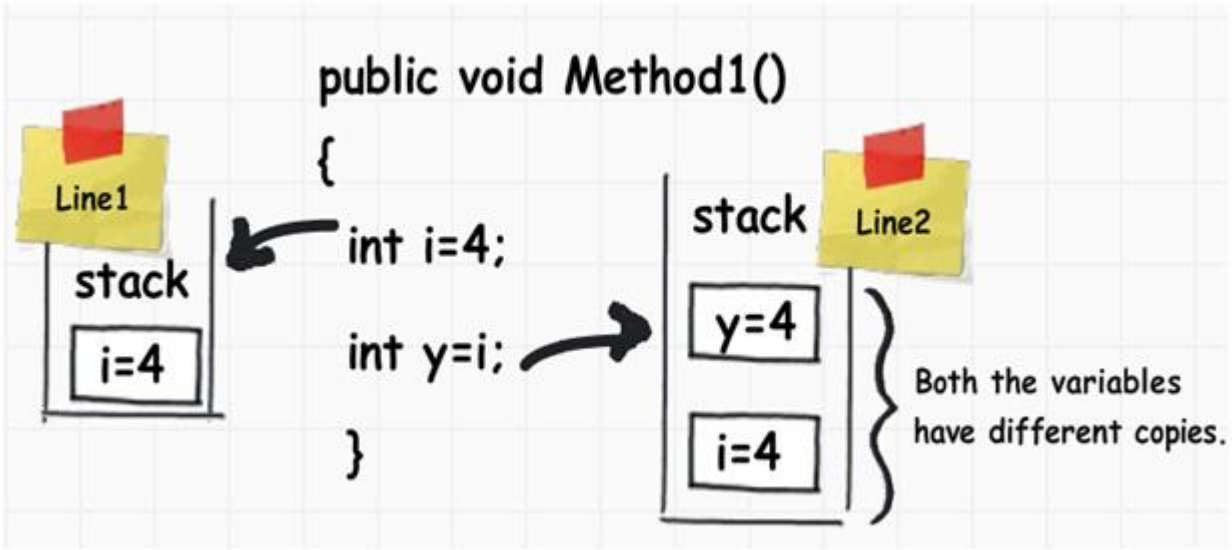
Variables

- **A variable is an entity whose value can keep changing.**

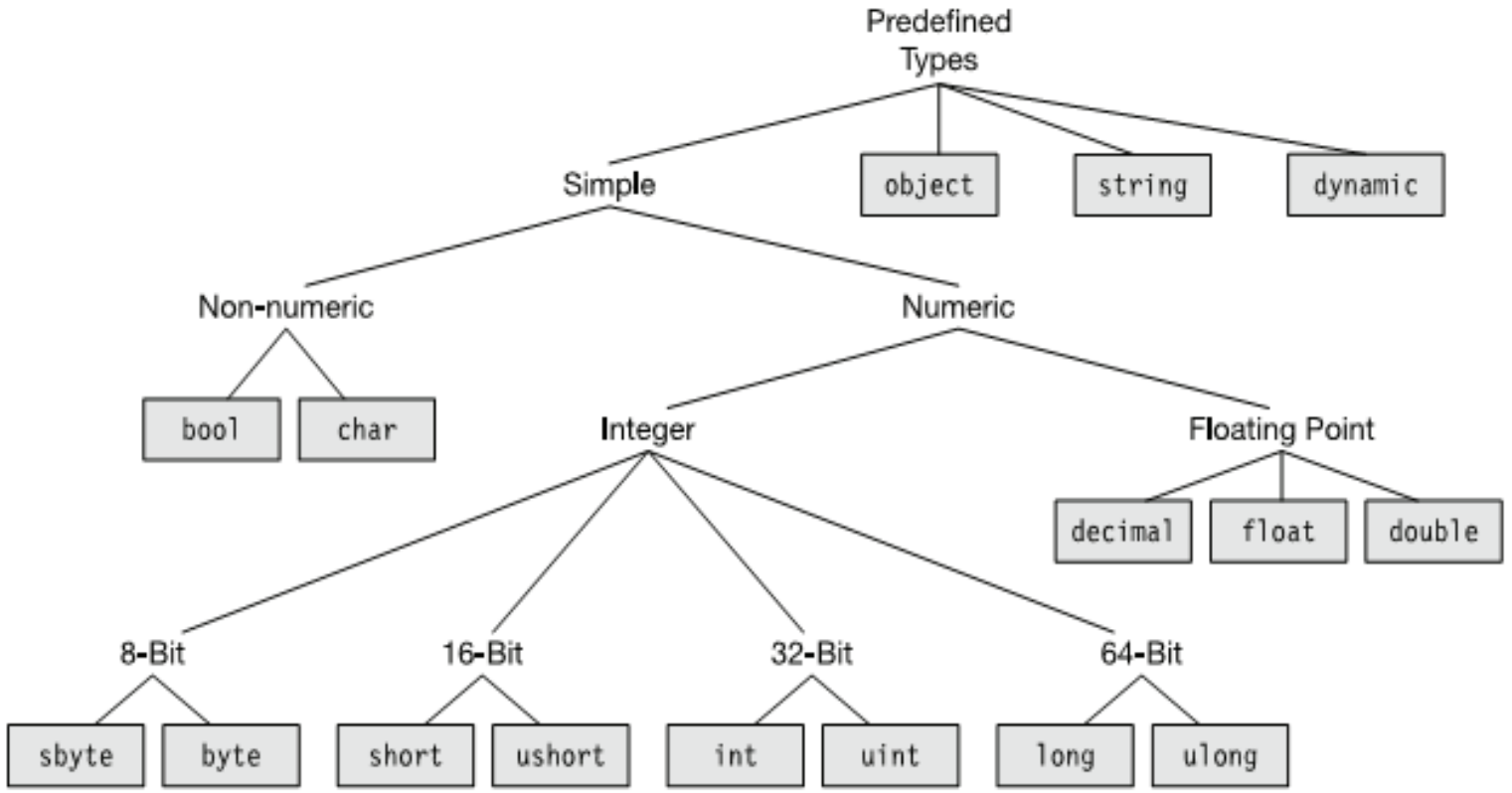


Data types

- **Definition:**
- **Divided two type:**
  - **Value types: Variable of value types store actual values. These values are store in a stack.**

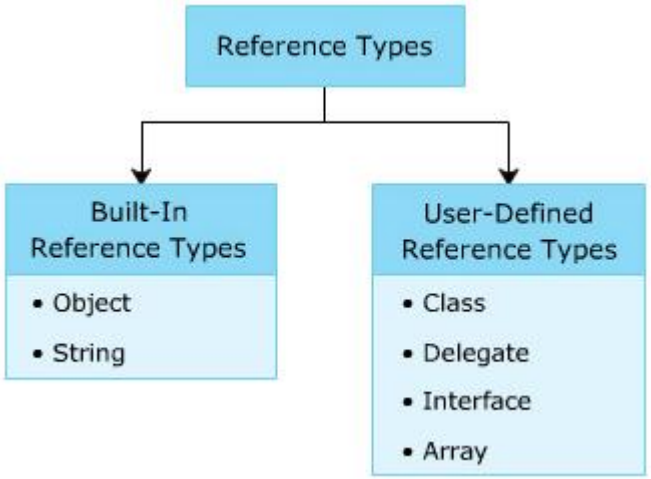


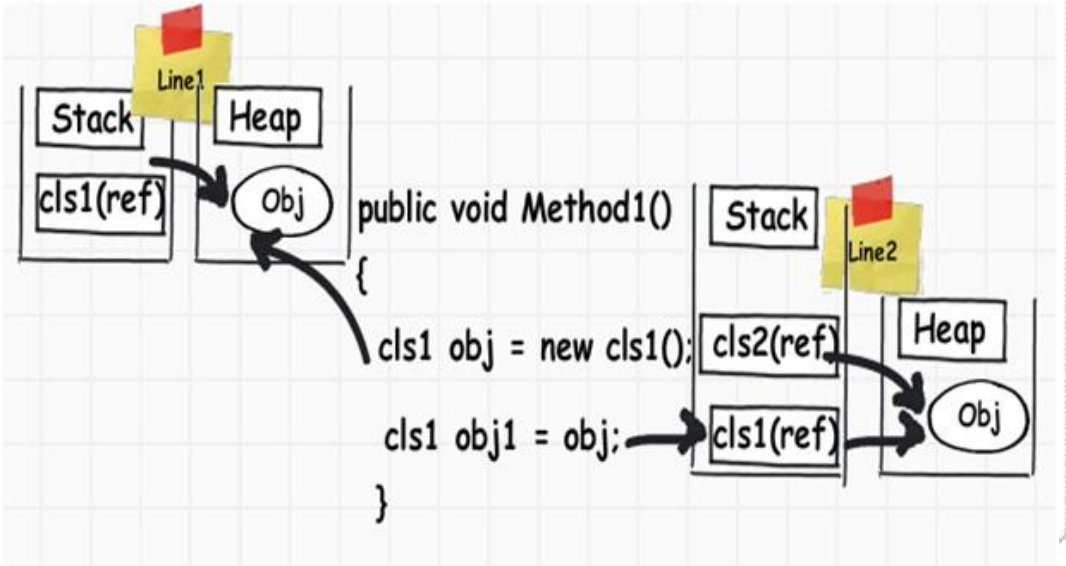
Data types


Predefined Types
Simple
object
string
dynamic
Non-numeric
Numeric
bool
char
Integer
Floating Point
decimal
float
double
8-Bit
16-Bit
32-Bit
64-Bit
sbyte
byte
short
ushort
int
uint
long
ulong

Data types

- **Divided two type:**
  - **Reference type: Variables of reference type store the memory address of other variables in a heap.**





Data types

- **Categorizing the C# Types**

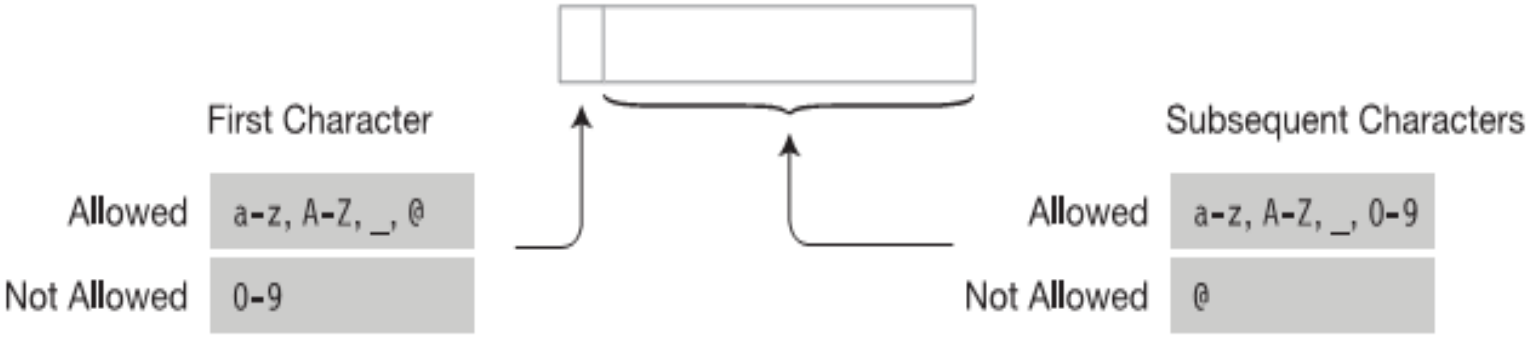
	Value Types			Reference Types
Predefined types	sbyte byte short int long bool	ushort uint ulong	float double char decimal	object string dynamic
User-defined types	struct enum			class interface delegate array

Declare and use variables

- **Syntax:**

  <data type> <variable name> = <value>;

- **Identifiers:**



- **Must assign an initial value to variable before use.**

Nullable type

- **Value types never be assigned the value of null**

```
// Compiler errors!
  // Value types cannot be set to null!
  bool myBool = null;
  int myInt = null;

  // OK! Strings are reference types.
  string myString = null;
```

- **To define a nullable variable type, the question mark symbol (?) is suffixed to the underlying data**

```
  int? nullableInt = 10;
  double? nullableDouble = 3.14;
  bool? nullableBool = null;
  char? nullableChar = 'a';
  int?[] arrayOfNullableInts = new int?[10];

// string? s = "oops"; // Error! Strings are reference types!
```

Nullable type

- **Nullable data types can be particularly useful when you are interacting with databases, given that columns in a data table may be intentionally empty.**
- **The Operator ??**

This operator allows you to assign a value to a nullable type if the retrieved value is in fact null

```
// If the value from GetIntFromDatabase() is null,
 // assign local variable to 100.
int myData = dr.GetIntFromDatabase() ?? 100;
```

LESSON 2. INPUT & OUTPUT

- Output
- Input

Output method

- **Syntax:**
  - **Console.Write(“<data>” + variables);**
  - **Console.WriteLine(“<data>” + variables);**
- **Format Console**
  - **Use place holders {0}, {1},…**
  - **Use Format characters: c, f,…**

Input method

- **Syntax:**
  - **Console.Read() - Reads a single character**
  - **Console.ReadLine() - Reads a line of strings**
- **Input a string**
- **Input a numeric value**
  - **Parse**
  - **Convert**
  - **TryParse**

LESSON 3. C# PROGRAMMING CONSTRUCTS

- Selection
- Loop

Selection construct

- **If/else**
  - **if-else statement controls the flow of program based on the evaluation of the boolean expression.**
  - **Use Relational Operators: ==, !=, <, >, <=, >=**
  - **Use Conditional Operators: &&, || , !**
- **If statement:**

```
if(boolean expression)
{
    // execute this code block if expression evalutes to true
}
```

- **If/else statement:**

```
if(boolean expression)
{
    // execute this code block if expression evalutes to true
}
else
{
    // always execute this code block when above if expression is false
}
```

Selection construct

- **Ternary operator ? :**

```
var result = Boolean conditional expression ? first statement : second statement
```

- **Ternary operator returns a value or expression included in the second or third part of it. It does not execute the statements.**
- **Ex:**

```
int x = 20, y = 10;

var result = x > y ? "x is greater than y" : "x is less than or equal to y";

Console.WriteLine(result);
```

Selection construct

- The switch statement executes the code block depending upon the resulted value of an expression.
- Can only switch on an expression of a type that can be statically evaluated: int, bool, string, char, enum types

```
switch(expression)
{
    case <value1>
        // code block
    break;
    case <value2>
        // code block
    break;
    case <valueN>
        // code block
    break;
    default
        // code block
    break;
}
```

Loop construct

- for loop
- foreach/in loop
- while loop
- do/while loop

For loop

- **Iterate over a block of code a fixed number of times**
- **Syntax:**

```
for (variable initialization; condition; steps)
{
    //execute this code block as long as condition is satisfied
}
```

Foreach Loop

- C# foreach keyword allows you to iterate over all items in a container without the need to test for an upper limit
- Syntax:

```
foreach (<datatype> <identifier> in <list>)
{
  // one or more statements;
}
```

While loop

- to execute a block of statements until some terminating condition has been reached
- Syntax

```
While(boolean expression)
{
    //execute code as long as condition returns true

}
```

Do / while loop

- **Is used when you need to perform some action an undetermined number of times.**
- **Syntax**

```
do
{
    //execute code block


} while(boolean expression)
```

LESSON 4. METHODS

- Parameter
- Optional method
- Named method
- Overloading method

Methods

- **Syntax**

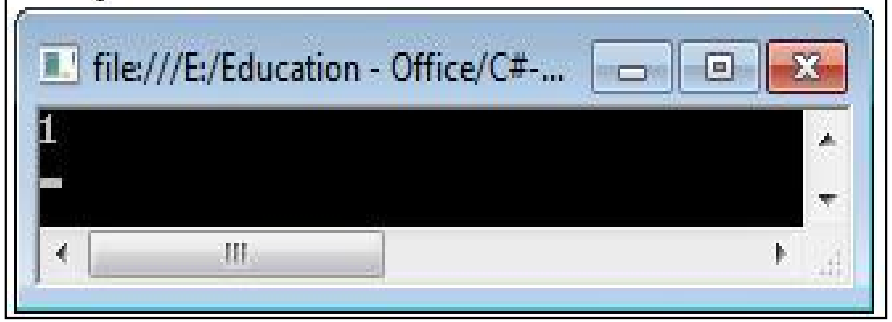
```
<Access Specifier> <Return Type> <Method Name>(Parameter List)
{
   Method Body
}
```

- **Parameters can be passed to a method by these following ways:**
  - **Value:**
  - **Out:**
  - **Ref:**
  - **Params**

Passing parameters

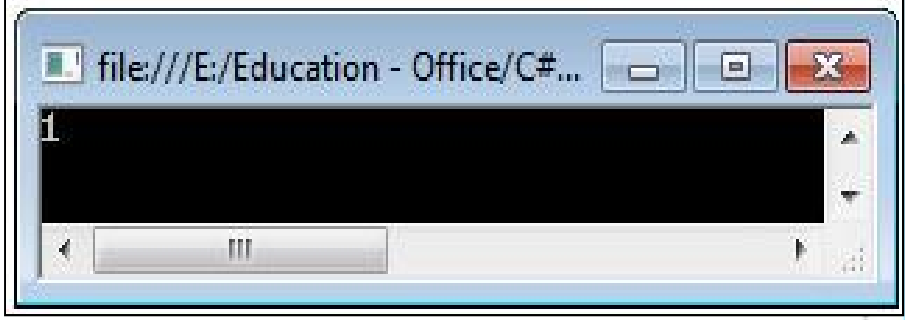
```
static private void Plus(ref int b)
{
    //b = 0;
    b++;
}

static void Main(string[] args)
{
    int a=0;
    Plus(ref a);
    Console.WriteLine(a);
    Console.ReadLine();
}
```




```
static private void Plus(out int b)
{
    b = 0;
    b++;
}

static void Main(string[] args)
{
    int a;
    Plus(out a);
    Console.WriteLine(a);
    Console.ReadLine();
}
```



Params modifier

- C# supports the use of parameter arrays using the params keyword.
- Is used when you not sure of the number of arguments passed as a parameter

```
class ParamArray
{
   public int AddElements(params int[] arr)
   {
      int sum = 0;
      foreach (int i in arr)
      {
         sum += i;
      }
      return sum;
   }
}

class TestClass
{
   static void Main(string[] args)
   {
      ParamArray app = new ParamArray();
      int sum = app.AddElements(512, 720, 250, 567, 889);
      Console.WriteLine("The sum is: {0}", sum);
      Console.ReadKey();
   }
}
```

Optional Parameters

- Allows the caller to invoke a method while omitting arguments deemed unnecessary.
- Optional Parameter if not passed will take default value
- Optional Parameter must define at the end of the any required parameter
- Ex:

```
static double VAT(double productCost, double currentRate = 20)
{
 double cR = (currentRate + 100) / 100;
 return productCost * cR;
}
```

Named Parameters

- **Named arguments allow you to invoke a method by specifying parameter values rather than passing parameters by position.**
- **Ex:**

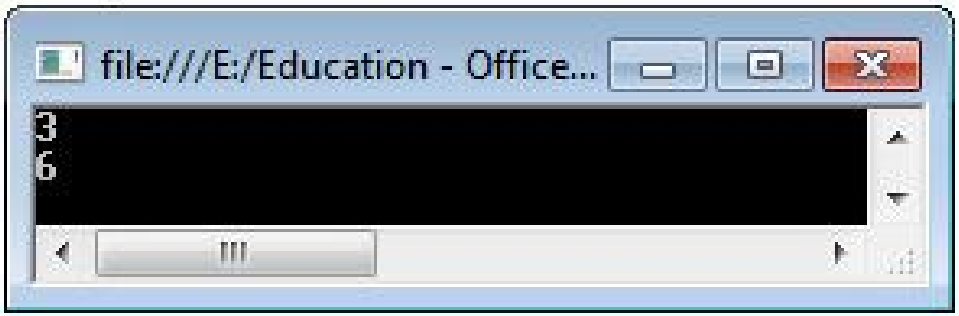
```
Person person = new Person("John", "Smith", new DateTime(1970, 1, 1));
Person person = new Person(firstName: "John", lastName: "Smith",
dateOfBirth: new DateTime(1970, 1, 1));
```

- **Invoke a method using positional parameters, they must be listed before any named parameters.**
- **if you have a method that defines optional arguments, this feature can actually be really helpful**

Method overloading

```
static int Add(int a, int b)
{
    return a + b;
}

static int Add(int a, int b, int c)
{
    return a + b + c;
}
static void Main(string[] args)
{
    Console.WriteLine(Add(1,2));
    Console.WriteLine(Add(1, 2, 3));
    Console.ReadLine();
}
```



- Define a set of identically named methods that differ by the number (or type) of parameters

LESSON 4. ENUMERATION TYPE

Enumeration type

- Enumeration is a value data type. The enum is used to declare a set of named integer constants.
- enum keyword allows you to define a custom set of name/value pairs.

- Syntax:

```
enum <enum_name>
{
   enumeration list
};
```

- Where:
- The *enum_name* specifies the enumeration type name.
- The *enumeration list* is a comma-separated list of identifiers.

Enumeration type

- Enum is an abstract class that includes static helper methods to work with enum
  - Format
  - GetName
  - GetNames
  - GetValues
  - Object Parse(type,string)
  - Bool TryParse(string,out Tenum)

```
enum WeekDays
{
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday,
    Sunday
}

Console.WriteLine(Enum.GetName(typeof(WeekDays), 4));

Console.WriteLine("WeekDays constant names:");

foreach (string str in Enum.GetNames(typeof(WeekDays)))
            Console.WriteLine(str);

Console.WriteLine("Enum.TryParse():");

WeekDays wdEnum;
Enum.TryParse<WeekDays>("1", out wdEnum);
Console.WriteLine(wdEnum);
```

LESSION 6. ARRAYS

- Introduction
- Declare and initialize an array
- Array Class

Introduction

- **An array always stores values of a single data type.**
- **Each value is referred to as an element.**

Jack	Kate	Francis	Glen	Frank	...

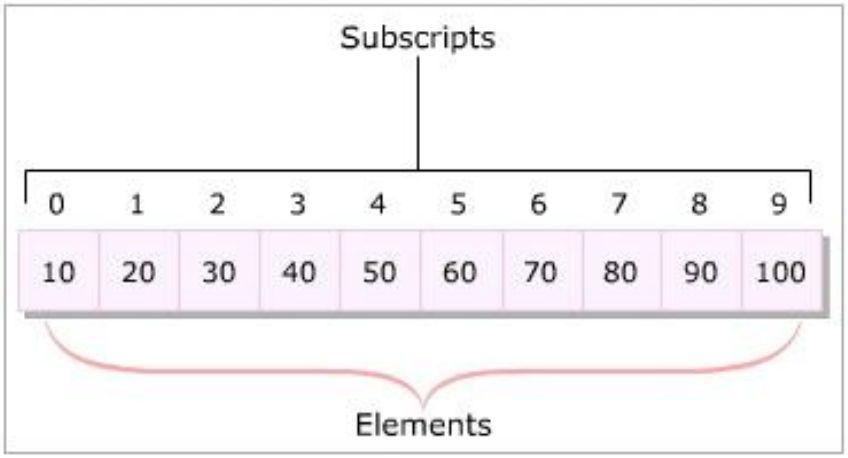
Array of 100 Names

Efficient Memory Utilization

```
//Program to store 100 names of students
string studentOne = "Jack Anderson";
string studentTwo = "Kate Jones";
string studentThree = "Francis Diaz";
string studentFour = "Glen Daniel";
string studentFive = "Frank James";
…
…
… Till 100 variables
```

100 Variables Storing Names

Inefficient Memory Utilization



Arrays

Declare arrays

1. type[] arrayName = new type[size-value];

Snippet:

```
int[] number = new int[5];
```

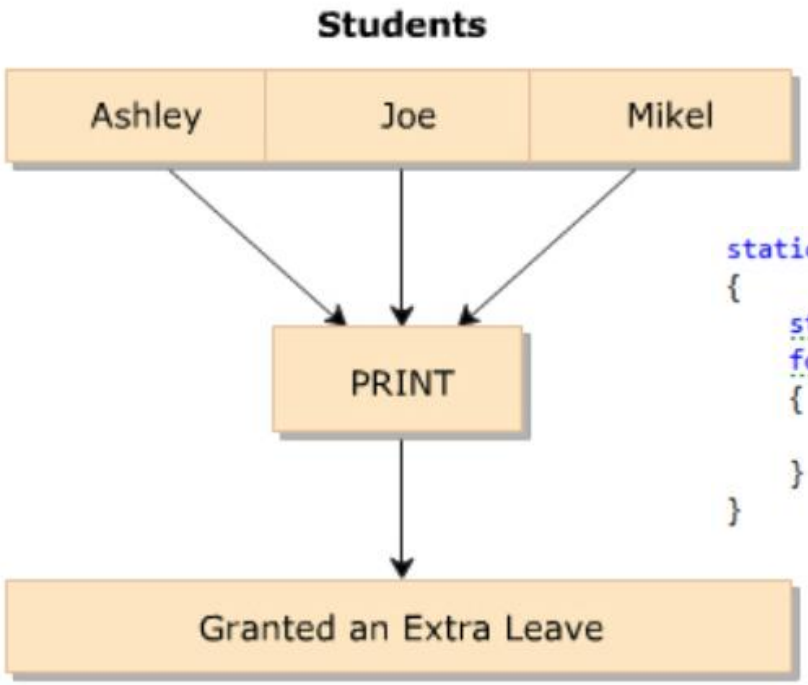
Data Types	Default Values
int	0
float	0.0
double	0.0
char	'\0'
string	null

2. type[ ] arrayIdentifier = {val1, val2, va13, ..., valN};

Snippet:

```
string[] studNames = {"Bob", "Mary", "Garry"};
```

Iterate through arrays with foreach




```
static void Main(string[] args)
{
    string[] students = new string[3] {"James", "Alex", "Fernando"};
    for (int i = 0; i < students.Length; i++)
    {
        Console.WriteLine(students[i]);
    }
}
```

Syntax:

```
foreach (type<identifier> in <list>)
    {
      // statements
    }
```

Array class

- The Array class is the base class for all the arrays in C#.
- The Array class provides various properties and methods to work with arrays.

```
class MyArray
{
   static void Main(string[] args)
   {
      int[] list = { 34, 72, 13, 44, 25, 30, 10 };
      int[] temp = list;
      Console.Write("Original Array: ");

      foreach (int i in list)
      {
         Console.Write(i + " ");
      }
      Console.WriteLine();

      // reverse the array
      Array.Reverse(temp);
      Console.Write("Reversed Array: ");

      foreach (int i in temp)
      {
         Console.Write(i + " ");
      }
      Console.WriteLine();

      //sort the array
      Array.Sort(list);
      Console.Write("Sorted Array: ");

      foreach (int i in list)
      {
         Console.Write(i + " ");
      }
```