

Practical-3

Console Application

Practical 3 – Array and User Define Functions

- 1. Write a program to remove duplicate elements of an array.**
- 2. Write a program for multiplication of two 2-dimentional matrices using 2-d array.**
- 3. Write a program to generate Pascal Triangle using jagged array.**
- 4. Write a user defined function to sort an array.**
- 5. Demonstrate the use of params keyword with the help of a program.**
- 6. Discuss out and ref parameters with the help of programs.**

Array

- It is used to store collection of data
- Store a fixed size sequential collection of element of same data type.
- Instead of declaring individual variable, you declare one array.
- Array elements are accessed by its index.
- Types of array:
 1. Single Dimensional Array
 2. Multi Dimensional Array
 3. Jagged Array

1-dim array

- `Int[] arr1= new int[5];`
`arr1[0]=10;`
`arr1[1]=20;`
`arr1[2]=30;`
- `Int[] arr2= new int[5] {1,2,3,4,5};`
- `Int[] arr3= new int[] {1,2,3,4,5};`

2-dim array

- `int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };`
- `int[,] array2Da = new int[4, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };`
- `string[,] array2Db = new string[3, 2] { { "one", "two" }, { "three", "four" }, { "five", "six" } };`

Multi Dimensional array

- It is also known as rectangular arrays in C#.
- It can be two dimensional or three dimensional.
- The data is stored in tabular form (row * column) which is also known as matrix.

Syntax to declare Multi Dimensional array:

- To create multidimensional array, use comma inside the square brackets.
- `int[,] arr=new int[3,3];` //declaration of 2D array
- `int[,,,] arr=new int[3,3,3];` //declaration of 3D array

Jagged Array

- It is an array whose elements are arrays.
- The elements of a jagged array can be of different dimensions and sizes.
- A jagged array is sometimes called an "array of arrays."
- **A special type of array is introduced in C#.**
- **A Jagged Array is an array of an array in which the length of each array index can differ.**

Jagged Array-Example

```
int[][] jaggedArray = new int[3][];  
jaggedArray[0] = new int[3];  
jaggedArray[1] = new int[5];  
jaggedArray[2] = new int[2];  
jaggedArray[0] = new int[] { 3, 5, 7 };  
jaggedArray[1] = new int[] { 1, 0, 2, 4, 6 };  
jaggedArray[2] = new int[] { 1, 6 };
```


Function

It is a group of statements to perform a task.

Syntax:

```
<Access Specifier> <Return Type> Function Name (Parameters)
{
    //function body with return statement
}
```

A function consists of the following components:

- **Function name:** It is unique name which is used to call function.
- **Return type:** It is used to specify the data type of function return value.
- **Body:** It is a block that contains executable statements.
- **Access specifier:** It is used to specify function accessibility in the application.
- **Parameters:** It is a list of arguments can pass to function during call.

Note: Access-specifier, parameters and return statement are optional. 9

Array As Function Argument

using System;

namespace HelloProgram {

class ArrayAsFunctionArgument {

static void Main(string[] args) {

int[] arr = { 1, 2, 3, 4, 5 };

PrintArray(arr);

Console.ReadKey();}

static void PrintArray(int[] array)

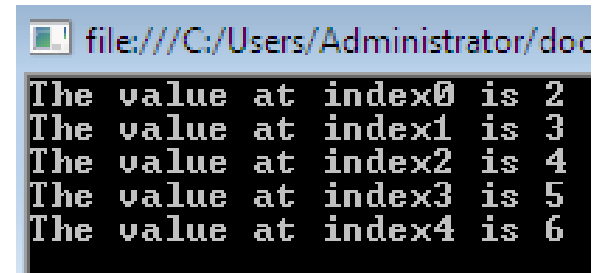
{

for (int i = 0; i < array.Length; i++)

{

array[i] = array[i] + 1;

Console.WriteLine("The value at index{0} is {1}", i, array[i]); } } }



```
file:///C:/Users/Administrator/doc
The value at index0 is 2
The value at index1 is 3
The value at index2 is 4
The value at index3 is 5
The value at index4 is 6
```

“Params” Keyword

- In simple function we can allow fixed number of function arguments

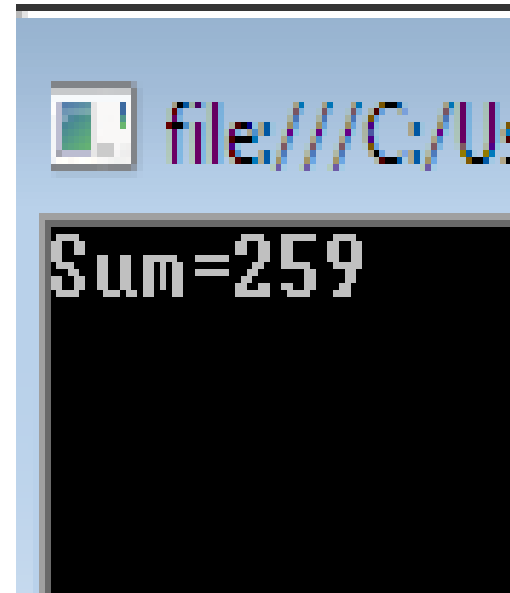
Program:

```
class Program
{
    static void Main(string[] args)
    {
        int y = Add (12,14,43);
    }
    public static int Add(int num1, int num2, int num3)
    {
        return num1+num2+num3;
    }
}
```

- But in this simple program1 we want to add more parameter run time the use “params keyword”
- Unknown parameter passing to array

“Params” Keyword-Program1

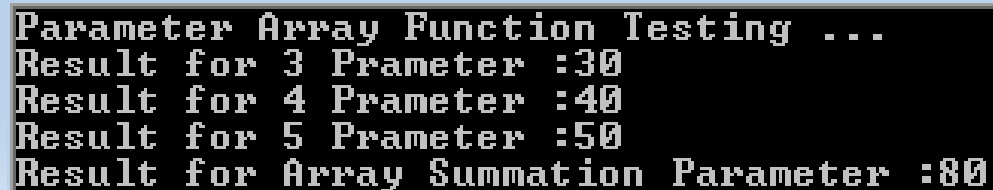
```
using System;
namespace HelloProgram {
    class ParamKeyword {
        /*******Usig Param keyword*****
        static void Main(string[] args)
        {
            int y = Add(12, 14, 43, 34, 56, 100);
            Console.WriteLine("Sum=" + y);
            Console.ReadKey();
        }
        public static int Add(params int[] ListNumbers)
        {
            int total = 0;
            foreach (int i in ListNumbers)
            {
                total = i + total;
            }
            return total;
        }
    }
}
```



"Params" Keyword- Program2

```
using System;
namespace HelloProgram
{
    class ParamKeyword
    {
        static int Add(params int[] nums)
        {
            int total=0;
            foreach(int i in nums)
            {
                total = total+i;
            }
            return total;
        }
    }
}
```

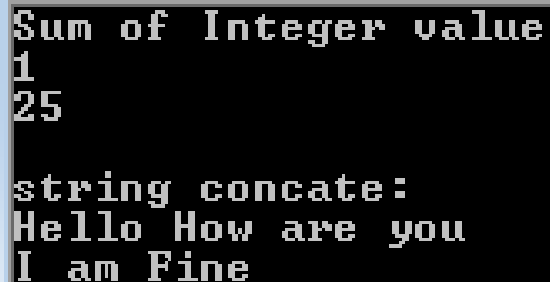
```
static void Main(string[] args)
{
    Console.WriteLine("Parameter Array Function Testing ...");
    int result = 0;
    result = Add(10, 10, 10);
    Console.WriteLine("Result for 3 Prameter :{0}", result);
    result = Add(10, 10, 10, 10);
    Console.WriteLine("Result for 4 Prameter :{0}", result);
    result = Add(10, 10, 10, 10, 10);
    Console.WriteLine("Result for 5 Prameter :{0}", result);
    int[] x = { 10, 10, 10, 10, 10, 10, 10, 10 };
    result = Add(x);
    Console.WriteLine("Result for Array Summation Parameter :{0}", result);
    Console.ReadKey(); } }
```



```
Parameter Array Function Testing ...
Result for 3 Prameter :30
Result for 4 Prameter :40
Result for 5 Prameter :50
Result for Array Summation Parameter :80
```

"Params" Keyword- Program2

```
using System;
namespace HelloProgram{
class Paramkeyword2 {
static void Main() {
    Console.WriteLine("integer value sum");
    ADDparameters(1);
    ADDparameters(1,3,5,7,9,0);
    Console.WriteLine("string concate:");
    ADDparameters1("Hello", " ", "How", " ",
        "are", " ", "you");
    ADDparameters1("I", " ", "am", " ", "Fine");
    Console.ReadKey();
}
```



```
Sum of Integer value
1
25

string concate:
Hello How are you
I am Fine
```

```
public static void ADDparameters(params int[]
arguments) {
    int add = 0;
    foreach (int arg in arguments)
    {
        add += arg;
    }
    Console.WriteLine(add);
}
```

```
public static void ADDparameters1(params string[]
arguments) {
    string add = "";
    foreach (string arg in arguments)
    {
        add += arg;
    }
    Console.WriteLine(add); } }
```

"Params" Keyword- Program3

```
using System;
namespace HelloProgram {
    class Paramkeyword3{
        static void Main() {
            Console.WriteLine("No parameter");
            Useparams();
            Console.WriteLine("\nlist of same type
arguments comma seperated:");
            Useparams(10, 20, 30, 40, 50);
            Console.WriteLine("\nDifferent typs of
arguments:");
            UseParams2(1, 'a', "Hello");
            Console.WriteLine("\nInteger array: ");
            int[] myIntArray = { 12, 13, 14, 15, 16 };
            Useparams(myIntArray);
            Console.WriteLine();
            Console.WriteLine("\narray with different
types of arguments");
        }
    }
}
```

```
object[] myObjArray={1,0.5,'a',"testing","again"};
UseParams2(myObjArray);
//It will give error because array can not be
converted into integer array
//Useparams(myObjArray);
Console.ReadKey(); }
```

```
public static void Useparams(params int[] list){
    for (int i = 0; i < list.Length; i++){
        Console.Write(list[i]+"\\t");
    }
    Console.WriteLine(); }
```

```
public static void UseParams2(params object[] list) {
    for (int i = 0; i < list.Length; i++) {
        Console.Write(list[i] + "\\t");
    }
    Console.WriteLine(); } }
```

“Params” Keyword- Program3

```
No parameter
```

```
list of same type arguments comma seperated:  
10      20      30      40      50
```

```
Different typs of arguments:  
1      a      Hello
```

```
Integer array:  
12      13      14      15      16
```

```
array with different types of arguments  
1      0.5      a      testing again
```


Output Parameter

- Out parameter can be used to return values in same variable declared as method parameter.
- Represents same storage location instead of creating new storage.
- **Declare with `out` modifier.**
- **Every output parameter of method must be assigned before the method return the output.**
- **It used in methods that produce multiple return value.**

Output Parameter-Program-1

using System;

namespace HelloProgram {

class OutParameter {

static void set_value(out int v)

{

v = 20;

}

static void Main()

{

int value;

set_value(out value); // method calling

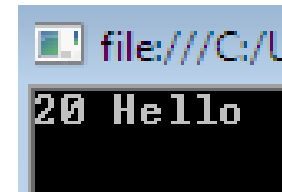
Console.WriteLine(value);

Console.ReadKey(); } } }



Output Parameter-Program-2

```
using System;
namespace HelloProgram{
    class OutParameter
    {
        static void value(out int i,out string s1, out string s2)
        {
            i =20;
            s1 ="Hello";
            s2 = null;
        }
        static void Main()
        {
            int v;
            string str1, str2;
            value(out v, out str1, out str2); // method calling
            Console.WriteLine(v + " " + str1 + " " + str2 );
            Console.ReadKey(); } } }
```



Reference Parameter-Program

- It is similar to **out** keyword but **ref** requires that the variable be initialized before being passed.
- Gives error when used unassigned local variable value.
- This method copies the reference to the memory location of an argument into the formal parameter. This means that changes made to the parameter affect the argument.

Reference Parameter-Program

```
using System;
namespace HelloProgram
{
    class ReferenceParameter
    {
        static void set_value(ref int v)
        {
            v = 20;
        }
        static void Main()
        {
            int value = 10;
            set_value(ref value); // method calling
            Console.WriteLine(value);
            Console.ReadKey(); } } }
```



REF KEYWORD	OUT KEYWORD
It is necessary the parameters should initialize before it pass to ref.	It is not necessary to initialize parameters before it pass to out.
It is not necessary to initialize the value of a parameter before returning to the calling method.	It is necessary to initialize the value of a parameter before returning to the calling method.
The passing of value through ref parameter is useful when the called method also need to change the value of passed parameter.	The declaring of parameter through out parameter is useful when a method return multiple values.
When ref keyword is used the data may pass in bi-directional.	When out keyword is used the data only passed in unidirectional.