**Dot Net Technologies (DA6220)**



DIT University 2019-2020

**Submitted To: Ms. Ajay Kumar Assistant Professor**

**Department of Computer Science**

**Submitted By: Lokesh Karki**

**160109049**

**CSE-BDA  
 3rd Year**

**PROGRAM 1: Program to demonstrate command line arguments.**

using System;

class Program

{

static void Main(string[] args)

{

Console.WriteLine("\nNumber of CommadLine Arguments :" + args.Length);

Console.Write("\nCommandline Arguments Are :\t");

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

{

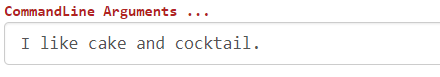
Console.Write(args[i] + "\t");

}

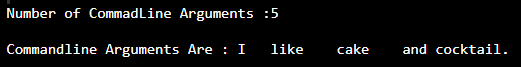
Console.ReadLine();

}

}



**OUTPUT:**



**PROGRAM 2: Program to demonstrate the static constructor, private constructor and copy constructor.**

**Static constructor:**

using System;

namespace ProgramCall

{

class Test3

{

public Test3()

{

Console.WriteLine("Instance Constructor");

}

static Test3()

{

Console.WriteLine("Static Constructor");

}

}

class StaticConstructor

{

static void Main()

{

//Static Constructor and instance constructor, both are invoked for first instance.

Test3 T1 = new Test3();

//Only instance constructor is invoked.

Test3 T2 = new Test3();

Console.ReadLine();

}

}

}

**OUTPUT:**



**Private Constructor :**

using System;

public class Counter

{

private Counter() { }

public static int currentCount;

public static int IncrementCount()

{

return ++currentCount;

}

}

class TestCounter

{

static void Main()

{

// If you uncomment the following statement, it will generate

// an error because the constructor is inaccessible:

// Counter aCounter = new Counter(); // Error

Counter.currentCount = 100;

Counter.IncrementCount();

Console.WriteLine("New count: {0}", Counter.currentCount);

Console.ReadKey();

}

}

**OUTPUT:**



**Copy Contructor :**

using System;

namespace ProgramCall

{

class Test2

{

int A, B;

public Test2(int X, int Y)

{

A = X;

B = Y;

}

//Copy Constructor

public Test2(Test2 T)

{

A = T.A;

B = T.B;

}

public void Print()

{

Console.WriteLine("A = {0}\tB = {1}", A, B);

}

}

class CopyConstructor

{

static void Main()

{

Test2 T2 = new Test2(80, 90);

//Invoking copy constructor

Test2 T3 = new Test2(T2);

T2.Print();

T3.Print();

Console.ReadLine();

}

}

}

**OUTPUT:**



**PROGRAM 3: Write a program of boxing and unboxing.**

**Boxing:**

class TestBoxing

{

static void Main()

{

int i = 123;

// Boxing copies the value of i into object o.

object o = i;

// Change the value of i.

i = 456;

// The change in i doesn't affect the value stored in o.

System.Console.WriteLine("The value-type value = {0}", i);

System.Console.WriteLine("The object-type value = {0}", o);

}

}

**OUTPUT:**



**Unboxing:**

using System;

class unbox {

// Main Method

static public void Main()

{

// assigned int value

// 23 to num

int num = 23;

// boxing

object obj = num;

// unboxing

int i = (int)obj;

// Display result

Console.WriteLine("Value of ob object is : " + obj);

Console.WriteLine("Value of i is : " + i);

}

}

**OUTPUT:**



**PROGRAM 4: Program to demonstrate “out” and “params” methods parameter modifier followed by pass by reference using “ref” keyword.**

**“out” parameter:**

using System;

namespace OutParameter

{

class Program

{

// User defined function

public void Show(out int a, out int b) // Out parameter

{

int square = 5;

a = square;

b = square;

// Manipulating value

a \*= a;

b \*= b;

}

// Main function, execution entry point of the program

static void Main(string[] args)

{

int val1 = 50, val2 = 100;

Program program = new Program(); // Creating Object

Console.WriteLine("Value before passing \n val1 = " + val1+" \n val2 = "+val2);

program.Show(out val1, out val2); // Passing out argument

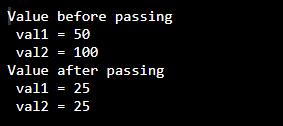
Console.WriteLine("Value after passing \n val1 = " + val1 + " \n val2 = " + val2);

}

}

}

**OUTPUT:**



**“params” parameter:**

using System;

namespace AccessSpecifiers

{

class Program

{

// User defined function

public void Show(params int[] val) // Params Paramater

{

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

{

Console.WriteLine(val[i]);

}

}

// Main function, execution entry point of the program

static void Main(string[] args)

{

Program program = new Program(); // Creating Object

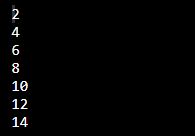
program.Show(2,4,6,8,10,12,14); // Passing arguments of variable length

}

}

}

**OUTPUT:**



**“ref” keyword:**

using System;

namespace CallByReference

{

class Program

{

// User defined function

public void Show(ref int val)

{

val \*= val; // Manipulating value

Console.WriteLine("Value inside the show function "+val);

// No return statement

}

// Main function, execution entry point of the program

static void Main(string[] args)

{

int val = 50;

Program program = new Program(); // Creating Object

Console.WriteLine("Value before calling the function "+val);

program.Show(ref val); // Calling Function by passing reference

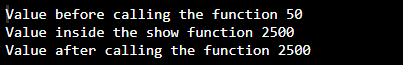
Console.WriteLine("Value after calling the function " + val);

}

}

}

**OUTPUT:**



**PROGRAM 5: Program to demonstrate System.Array class members like Clone(), Copy(), Clear(), Sort() and Reverse().**

using System;

namespace CSharpProgram

{

class Program

{

static void Main(string[] args)

{

// Creating an array

int[] arr = new int[6] { 5, 8, 9, 25, 0, 7 };

// Creating an empty array

int[] arr2 = new int[6];

int[] arr3 = new int[6];

// Displaying length of array

Console.WriteLine("length of first array: "+arr.Length);

// Sorting array

Array.Sort(arr);

Console.Write("Sorted array: ");

// Displaying sorted array

PrintArray(arr);

// Finding index of an array element

Console.WriteLine("\nIndex position of 25 is "+Array.IndexOf(arr,25));

// Coping first array to empty array

Array.Copy(arr, arr2, arr.Length);

Console.Write("Copied array elements: ");

// Displaying second array

PrintArray(arr2);

Array.Reverse(arr);

Console.Write("\nFirst Array elements in reverse order: ");

PrintArray(arr);

// Using clear() method

Array.Clear(arr, 1, 3);

Console.Write("\nClearing values: ");

PrintArray(arr);

// Using clone() method

object o = arr2.Clone();

arr3 = (int[]) o;

Console.Write("\nCloned Array: ");

PrintArray(arr3);

}

// User defined method for iterating array elements

static void PrintArray(int[] arr)

{

foreach (Object elem in arr)

{

Console.Write(elem+" ");

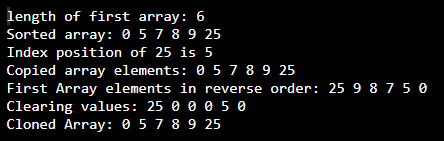
}

}

}

}

**OUTPUT:**



**PROGRAM 6: Program to demonstrate jagged array.**

class ArrayTest

{

static void Main()

{

// Declare the array of two elements:

int[][] arr = new int[2][];

// Initialize the elements:

arr[0] = new int[5] { 1, 3, 5, 7, 9 };

arr[1] = new int[4] { 2, 4, 6, 8 };

// Display the array elements:

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

{

System.Console.Write("Element({0}): ", i);

for (int j = 0; j < arr[i].Length; j++)

{

System.Console.Write("{0}{1}", arr[i][j], j == (arr[i].Length - 1) ? "" : " ");

}

System.Console.WriteLine();

}

System.Console.ReadKey();

}

}

**OUTPUT:**



**PROGRAM 7: Program to demonstrate System.string members like Contains(), Insert(), Remove(), Replace() and ToUpper().**

using System;

class Example

{

public static void Main()

{

string s1 = "The quick brown fox jumps over the lazy dog";

string s2 = "fox";

bool b = s1.Contains(s2);

Console.WriteLine("'{0}' is in the string '{1}': {2}",

s2, s1, b);

s1 = s1.Insert(s1.IndexOf("brown"), "dark");

Console.WriteLine(s1);

Console.WriteLine("{0}", s1.Remove(19));

Console.WriteLine("{0}", s1.Remove(3, 16));

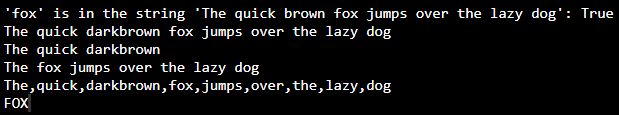
Console.WriteLine("{0}", s1.Replace(' ', ','));

Console.WriteLine("{0}",s2.ToUpper());

}

}

**OUTPUT:**



**PROGRAM 8: Write a program using enum and struct keyword.**

**“enum” keyword:**

using System;

public class EnumTest

{

enum Day { Sun, Mon, Tue, Wed, Thu, Fri, Sat };

static void Main()

{

int x = (int)Day.Sun;

int y = (int)Day.Fri;

Console.WriteLine("Sun = {0}", x);

Console.WriteLine("Fri = {0}", y);

}

}

**OUTPUT:**



**“struct” keyword:**

using System;

public struct Coords

{

public int x, y;

public Coords(int p1, int p2)

{

x = p1;

y = p2;

}

}

// Declare and initialize struct objects.

class TestCoords

{

static void Main()

{

// Initialize:

Coords coords1 = new Coords();

Coords coords2 = new Coords(10, 10);

// Display results:

Console.Write("Coords 1: ");

Console.WriteLine("x = {0}, y = {1}", coords1.x, coords1.y);

Console.Write("Coords 2: ");

Console.WriteLine("x = {0}, y = {1}", coords2.x, coords2.y);

Console.ReadKey();

}

}

**OUTPUT:**



**PROGRAM 9: Program to demonstrate Custom namespace.**

using System;

using red\_area;

using red\_area.blue\_area;

namespace red\_area {

class red {

public void area() {

Console.WriteLine("In red area");

}

}

namespace blue\_area {

class blue {

public void area() {

Console.WriteLine("In blue area");

}

}

}

}

class TestClass {

static void Main(string[] args) {

red rd = new red();

blue bl = new blue();

rd.area();

bl.area();

Console.ReadKey();

}

}

**OUTPUT:**



**PROGRAM 10: Program to demonstrate read only property.**

using System;

class SampleClass

{

public int x;

// Initialize a readonly field

public readonly int y = 25;

public readonly int z;

public SampleClass()

{

// Initialize a readonly instance field

z = 24;

}

public SampleClass(int p1, int p2, int p3)

{

x = p1;

y = p2;

z = p3;

}

public static void Main()

{

SampleClass p1 = new SampleClass(11, 21, 32); // OK

Console.WriteLine($"p1: x={p1.x}, y={p1.y}, z={p1.z}");

SampleClass p2 = new SampleClass();

p2.x = 55; // OK

Console.WriteLine($"p2: x={p2.x}, y={p2.y}, z={p2.z}");

}

}

**OUTPUT:**



**PROGRAM 11: Write a program for method overriding.**

using System;

class TestOverride

{

public class Employee

{

public string name;

// Basepay is defined as protected, so that it may be

// accessed only by this class and derived classes.

protected decimal basepay;

// Constructor to set the name and basepay values.

public Employee(string name, decimal basepay)

{

this.name = name;

this.basepay = basepay;

}

// Declared virtual so it can be overridden.

public virtual decimal CalculatePay()

{

return basepay;

}

}

// Derive a new class from Employee.

public class SalesEmployee : Employee

{

// New field that will affect the base pay.

private decimal salesbonus;

// The constructor calls the base-class version, and

// initializes the salesbonus field.

public SalesEmployee(string name, decimal basepay,

decimal salesbonus) : base(name, basepay)

{

this.salesbonus = salesbonus;

}

// Override the CalculatePay method

// to take bonus into account.

public override decimal CalculatePay()

{

return basepay + salesbonus;

}

}

static void Main()

{

// Create some new employees.

SalesEmployee employee1 = new SalesEmployee("Alice",

1000, 500);

Employee employee2 = new Employee("Bob", 1200);

Console.WriteLine("Employee4 " + employee1.name +

" earned: " + employee1.CalculatePay());

Console.WriteLine("Employee4 " + employee2.name +

" earned: " + employee2.CalculatePay());

}

}

**OUTPUT:**



**PROGRAM 12: Write a program of single cast delegate and multi cast delegate.**

**Singlecast delegate:**

using System;

//1. Declaration

public delegate int MyDelagate(int a, int b); //delegates having same signature as method

public class Example

{

// methods to be assigned and called by delegate

public int Sum(int a, int b)

{

return a + b;

}

public int Difference(int a, int b)

{

return a - b;

}

}

class Program

{

static void Main()

{

Example obj = new Example();

// 2. Instantiation : As a single cast delegate

MyDelagate sum = new MyDelagate(obj.Sum);

MyDelagate diff = new MyDelagate(obj.Difference);

// 3.Invocation

Console.WriteLine("Sum of two integer is = " + sum(10, 20));

Console.WriteLine("Difference of two integer is = " + diff(20, 10));

}

}

**OUTPUT:**



**Multicast delegate:**

using System;

//1. Declaration

public delegate void MyDelagate(int a, int b);

public class Example

{

// methods to be assigned and called by delegate

public void Sum(int a, int b)

{

Console.WriteLine("Sum of integers is = " + (a + b));

}

public void Difference(int a, int b)

{

Console.WriteLine("Difference of integer is = " + (a - b));

}

}

class Program

{

static void Main()

{

Example obj = new Example();

// 2. Instantiation

MyDelagate multicastdel = new MyDelagate(obj.Sum);

multicastdel += new MyDelagate(obj.Difference);

// 3. Invocation

multicastdel (50, 20);

}

}

**OUTPUT:**



**PROGRAM 13: Write a program of exception handling.**

using System;

class MyClient

{

public static void Main()

{

int x = 0;

int div = 0;

try

{

div = 100/x;

Console.WriteLine("Not executed line");

}

catch(DivideByZeroException)

{

Console.WriteLine("Exception occured");

}

finally

{

Console.WriteLine("Finally Block");

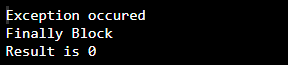
}

Console.WriteLine($"Result is {div}");

}

}

**OUTPUT:**

****

**PROGRAM 14: Write a program of indexers.**

using System;

class TempRecord

{

// Array of temperature values

private float[] temps = new float[10] { 56.2F, 56.7F, 56.5F, 56.9F, 58.8F,

61.3F, 65.9F, 62.1F, 59.2F, 57.5F };

// To enable client code to validate input

// when accessing your indexer.

public int Length

{

get { return temps.Length; }

}

// Indexer declaration.

// If index is out of range, the temps array will throw the exception.

public float this[int index]

{

get

{

return temps[index];

}

set

{

temps[index] = value;

}

}

}

class MainClass

{

static void Main()

{

TempRecord tempRecord = new TempRecord();

// Use the indexer's set accessor

tempRecord[3] = 58.3F;

tempRecord[5] = 60.1F;

// Use the indexer's get accessor

for (int i = 0; i < 10; i++)

{

System.Console.WriteLine("Element #{0} = {1}", i, tempRecord[i]);

}

// Keep the console window open in debug mode.

System.Console.WriteLine("Press any key to exit.");

System.Console.ReadKey();

}

}

**OUTPUT:**

