C# Programming With .NET
(06CS/IS761)

Chapter wise questions and answers appeared in previous year question papers:

	UNIT II: Building C# Applications	Markes & Year Appeared
1	Write a C# program to display the following information using the System environment class: i) Current directory of application. ii) Operating System Version iii) Host Name. iv) .NET Version.	June 12 (08m)

Ans	using System;
	class PlatformSpecifications {
	public static int Main(string[] args)
	{
	// Which OS version do we running on :
	Console.WriteLine("Operating System: {0}",Environment.OSVersion);
	
	// Which Directory……..?
	Console.WriteLine("App Directory: {0}",Environment.CurrentDirectory);
	
	// Nos of drivers on this system
	string[] drives = Environment.GetLogicalDrives();
	for(int i= 0; i< drives.Length; i++)
	Console.WriteLine("Drive {0} : {1}", i, drives[i]);
	
	//Which Version of .NET platform?
	Console.WriteLine("Current Version of .NET: {0}", Environment.Version);
	
	//Processor Count
	Console.WriteLine("Number of processors: {0}",Environment.ProcessorCount);
	return 0;
	}
	}	

2	Explain the building of a C# application using command line compiler CSE.exe. Or Explain how CSC.exe command is used to build C# applications on .NET. Explain any five flags with appropriate examples. Or Explain how CSC.exe compiler is used to build C# compiler is used to build C# applications. Explain any five flags with appropriate examples.	June 12 (04m) May-June 10 (06M) June-July 11 (06)

Ans	There are n- number of ways to compile C# source code.
	• Its possible to create .NET assemblies using the C# command line compiler "csc.exe (***csc : C Sharp Compiler).***
	• C-Sharp source code files are having the extension (filename.cs).
	• Steps used to create, compile and execute *.cs files:
	  • Create a source file with extension filename.cs
	  • Compile the code using the command: C:/csc filename.cs
	  • The filename.exe will be created and thus type ***filename*** in cmd prompt and hit enter key; Thus the output is displayed.
	Different flag types used with C# compiler output command csc.exe are:	

Table 2-1. *Output Options of the C# Compiler*

Option	Meaning in Life
/out	This option is used to specify the name of the assembly to be created. By default, the assembly name is the same as the name of the initial input *.cs file.
/target:exe	This option builds an executable console application. This is the default assembly output type, and thus may be omitted when building this type of application.
/target:library	This option builds a single-file *.dll assembly.
/target:module	This option builds a *module*. Modules are elements of multifile assemblies (fully described in Chapter 15).
/target:winexe	Although you are free to build graphical user interface–based applications using the /target:exe option, /target:winexe prevents a console window from appearing in the background.

3 Explain the C# preprocessor directives giving any three examples

Or

Explain C# preprocessor directives: i) #region, #endregion. ii) Conditional Code Compilation.

June 12 (08m) Dec 11 (05M)

Ans

- Pre- processing directives are processed as part of the lexical analysis phase of the compiler.
- The syntax of pre- processor directives is identical to that of the other members of the C family.

C# Preprocessor symbols.	Meaning in life
#define, #undef	Used to define and un- define conditional compilation symbols.
#if, #elif, #else, #endif	Used to conditionally skip sections of source code (based on specified compilation symbols).
#line	Used to control the line numbers emitted for errors and warnings.
#error, #warning	Used to issue errors and warnings for the current build.
#region, #endregion	Used to explicitly mark sections of source code. Under VS .NET, regions may be expanded and collapsed within the code window, other IDEs(including simple text editors) will ignore these symbols.

E. g: #region & #endregion:

- Using these tags, it is able to specify a block of code that may be hidden from view and identified by a friendly textual marker.
- This use of regions can help to keep lengthy *.cs files more manageable.
  E. g: It could help to create one region for a type's constructors, another for type properties and yet another for internal helper classes.

```
Class ProcessMe {
…
//Nested types will be examined later.
#region stuff I don't Care about.
Public class HelperClass
{ //stuff
}
```

```
Public interface MyHelperInterface
{ //stuff
}
#endregion
}
```

E.g: Conditional Code Compilation:

- Here checking for the DEBUG. If present, it will dump out a number of interesting statistics using System.Environment class.
- If DEBUG is not defined, The code placed between #if and & #endif will not be compiled.

```
Using System;
Class ProcessMe {
…
static void Main(string[] args)
     {
      //Are you in debug mode
     #if(DEBUG)
      Console.WriteLine("App Directory: {0}",Environment.CurrentDirectory);
      Console.WriteLine("Box : {0}",Environment.MachineName);
      Console.WriteLine("Operating System:  {0}",Environment.OSVersion);
      Console.WriteLine(".NET Version:  {0}",Environment.Version);
     #endregion
       }
}
```

4 Explain the following with respect to with respect to C# program in command prompt: Dec 11 (10m)

i) Referencing external assemblies. ii) Compiling Multiple Source files
ii) Response Files. iv) Generating Bug report.

Ans

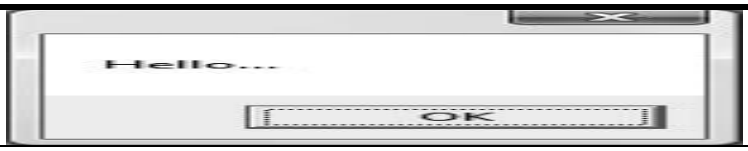
Referencing External Assemblies:

```
using System;
// Add this!
using System.Windows.Forms;
class TestApp
{
static void Main()
{
Console.WriteLine("Testing! 1, 2, 3");
// Add this!
MessageBox.Show("Hello...");
}
}
```

Consider the above code used to display a windows forms message box. Notice you are importin the System.Windows.Forms namespace via a C# ***using*** keyword.

- At the command line, you must inform csc.exe which assembly contains the namespaces you are using.
- Given that you have made use of the System.Windows.Forms.MessageBox class.
- you must specify the System.Windows.Forms.dll assembly using the /reference flag (which can be abbreviated to /r as shown below):

csc /r:System.Windows.Forms.dll TestApp.cs



Compiling Multiple Source files:

- The most of the basic applications are created using a single *.cs source code file.
- Most projects are composed of multiple *.cs files to keep your code base a bit more flexible.
- Consider the below set of codes working for one application written into two different files say TestApp.cs and HelloMessage.cs.
- Note: Here main function is present in TestApp.cs file which make use of the object of HelloMessage from HelloMessae.cs file for its execution.

```
using System;
class TestApp
{
static void Main()
{
Console.WriteLine("Testing! 1, 2, 3");
HelloMessage h = new HelloMessage();
h.Speak();
}
}
```

```
// The HelloMessage class
using System;
using System.Windows.Forms;
class HelloMessage
{ //BUILDING C# APPLICATIONS
public void Speak()
{
    MessageBox.Show("Hello...");
}
}
```

- The below commands can be used to execute such programs with multiple source files.

csc /r:System.Windows.Forms.dll TestApp.cs HelloMsg.cs

csc /r:System.Windows.Forms.dll *.cs

Response Files.

- If planned to build a complex C# application at the command prompt, It would be full of pain as to type in the flags that specify numerous referenced assemblies and *.cs i/p files.
- It has overcome with the help of C# response files, which contain all the instructions to be used during the compilation of current build.
- This type of file end in *.rsp (response) extension.

```
# External assembly references.
/r:System.Windows.Forms.dll
# output and files to compile (using wildcard syntax).
/target:exe /out:TestApp.exe *.cs
```

Note: Assuming this file is saved in the same directory as the C# source code files to be compiled, It is possible to build your entire application as follows (note the use of the @ symbol): csc @TestApp.rsp

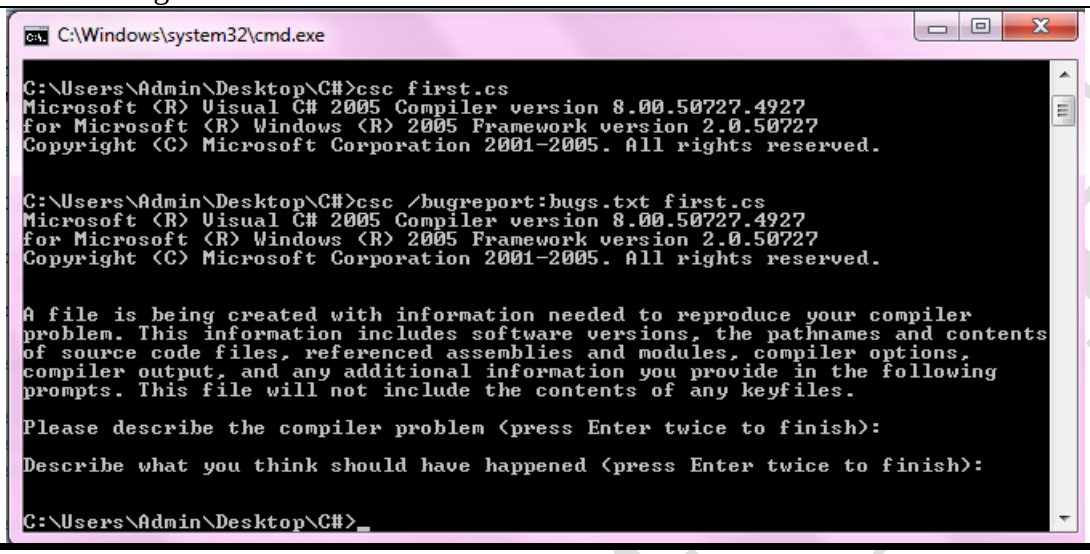
csc /out:MyCoolApp.exe @TestApp.rsp (Note: Flags listed explicitly on the command line before a response file will be overridden by the specified *.rsp file.)

Generating Bug report.

- C# compiler provides a flag named /bugreport.
- This flag allows you to specify a file that will be populated(by csc.exe) with various

statistics regarding the current build.

- It includes any errors encountered during the compilation process.
  C:/ csc /bugreport:bugs.txt *.cs
- ➢ When we specify /bugreport, it will be prompted to enter corrective information for the possible error(s) at hand.
- ➢ It will be saved into the file specified. E.g: in this current command line bugs.txt file will be generates as shown below.



5 Ans | Write a C# program to generate a Fibonacci series up to N. Value of N is read from Console. | Dec 11 (05M)

A C# program to generate a **Fibonacci series** up to N. Value of N is read from Console.

```
using System;
namespace SampleProgram
{
   class FibonacciSeries
   {
      public static void Main()
      {
         // Prompt the user to enter their target number
         Console.WriteLine("How many numbers do you want in the fibonacci series");

         // Read the user input from console and convert to integer
         int Target = int.Parse(Console.ReadLine());

         // Create integer variables to hold previous and next numbers
         int PreviousNumber = 0, PresentNumber = 0, NextNumber = 1;

         // This for loop controls the number of fibonacci series elements
         for (int i = 0; i < Target; i++)
         {
            // Logic to compute fibonacci series numbers
            Console.Write(PresentNumber + "  ");
            PreviousNumber = PresentNumber;
            PresentNumber = NextNumber;
            NextNumber = PreviousNumber + PresentNumber;
         }
         Console.ReadLine();
      }
   }
}
```

6 Write a program to count the number of object instances created inside or outside of an assembly. May-June 10 (08M)

Ans

A **C# program** to count the **number of object instances** created inside or outside of an assembly:

```
using System;
class object1
   {
   static int ob = 0;
   public object1()
     {
       ob = ob + 1;
     }
public static void Main(String[] args)
   {
     object1 ob1 = new object1();
     object1 ob2 = new object1();
     object1 ob3 = new object1();
     System.Console.WriteLine("Num of objected created are = {0}",ob);
   }
}
```

7 What is cordbg.exe? List and explain any five command line flags recognized by cordbg.exe while running .NET assemblies under debug mode. Dec- 10 (07M)

Or

What is command line debugger? Write source code in C# to compute the square root of a number passed as a command line argument. Dec- 09 (07M)

Ans

Cordbg.exe: is a tool that provides dozens of options that allow you to run .NET assemblies under debug mode.

- C:\ Cordbg -? : is the command used to view all possibles.
  Hand full use of cordbg.exe: command line.

CommLnFlgof	Meanng in life (cordbg.exe)
b[reak]	Set or display current breakpoints.
d[elete]	Remove one or more break points.
ex[it]	Exit the debugger
g[o]	Continue debugging the current process until hitting next breakpoint
si	Step into the next line.
o[ut]	Step out of the current function.
so	Step over the next line.
p[rint]	Print all the loaded variables (local, arguments, etc.)

8 What is CSC.rsp file? Where is it located? Dec- 10 (03M)

Ans

The Default Response File (csc.rsp):

- The C# compiler has an associated default response file (csc.rsp), which is located in the same directory as csc.exe itself.
- By default installed under:
  C:\Windows\Microsoft.NET\Framework\v3.5). (VS 2008) Or
  C:\Windows\Microsoft.NET\FrameWork\v2.5). (VS2005).
- If, wish to open this file using Notepad, you will find that numerous .NET assemblie

have already been specified using the /r: flag, including various libraries for web development, LINQ, data access, and other core libraries (beyond mscorlib.dll).
- While building the C# programs using csc.exe, this response file will be automatically referenced, even when you supply a custom *.rsp file.

9 How would you create object instance in C#? With examples, describe default assignment of .NET data types. Dec- 09 (08M)

Ans
- A class is a definition of a user- defined type (UDT).
- It is often regarded as a blueprint for variables of this type.
- Object is an instance of a particular class.
- "new" Keyword is the de- facto way of creating an object instance.
- The "new" keyword is in charge of allocating the correct number of bytes for the specified class and acquiring sufficient memory from the managed heap.
- C# object variables are actually a **reference** to the object in memory, not the actual memory itself.
- Note: Objects are stored in managed heap.

```
// Make HelloClass types correctly using the C# "new" keyword
Using System;
class HelloClass
{
    public static int Main(string[] args)
   {
     //You can declare and create a new object in a single line….
     HelloClass C1 = new HelloClass();
     //…. Or Break declaration and creation into two lines.
     HelloClass C2;
     C2 = new HelloClass();
     return 0;
   }
}
```

- Every C# class is automatically endowed with a default constructor, which you are free to define if need.
- Default constructors never take any parameters.

Note: Please add some more points for the question describe default assignment of .NET data types. As I'm not clear what all to be added for this Ans. Else the above Ans is ok for it

Program to perform simple arithmatic operations on two numbers and display the result in Decimal, Hexadecimal, Exponential and normal forms.

```
using System;

namespace ADD
{
  class Add
  {
     public static void Main()
     {
        int a = 30;
        int b = 20;
        int Addition = 0;
        int Substraction = 0;
        int Multiplication = 0;
        int Division = 0;

        Addition = a + b;
        Substraction = a - b;
```

```
            Multiplication = a * b;
            Division = a / b;
            Console.WriteLine("Result in normal form of A = {0} and B = {1}  gives\n
Addition = {2}\t,Substraction = {3}\t\n, Multiplication = {4}\t, Division = {5}\n",
a,b,Addition,Substraction,Multiplication,Division);

            Console.WriteLine("Result in Decimal Form A = {0:d} and B = {1:d}  gives\n
Addition = {2:d}\t,Substraction = {3:d}\t\n, Multiplication = {4:d}\t, Division = {5:d}\n",
a, b, Addition, Substraction, Multiplication, Division);

            Console.WriteLine("Result in Exponential form A = {0:e} and B = {1:e}  gives\n
Addition = {2:e}\t,Substraction = {3:e}\t\n, Multiplication = {4:e}\t, Division = {5:e}\n", a,
b, Addition, Substraction, Multiplication, Division);

            Console.WriteLine("Result in HexaDecimal form A = {0:x} and B = {1:x}  gives\n
Addition = {2:x}\t,Substraction = {3:x}\t\n, Multiplication = {4:x}\t, Division = {5:x}\n",
a, b, Addition, Substraction, Multiplication, Division);

            Console.Read();
        }
    }
}
```