# Chapter 1: Welcome to C#

1. **Features of Visual Studio 2015**
   - Provides integrated development environment (IDE) for editing, compiling (building), executing, debugging and profiling applications
   - Can be used to program in various languages under .NET framework like C#, VB, F#, J# and others
   - Supports wide variety of applications like Console, Graphical, Web, Mobile, IoT, Cloud and others
   - Console Applications allows interaction with terminals while Graphical application enables interactions with graphical widgets and forms (Graphical User Interface like TextBlock, TextBox, Radio Button, Button, etc)
   - Visual Studio 2015 allows graphical applications using drag and drop mechanism and automatic code generation behind the scene
   - Deployment of applications on various devices and simulators is integrated
   - The most fascinating feature of Visual Studio 2015 is the development of Universal Windows Platform (UWP) that can run on any Windows enabled device
   - Intellisense popmenus in Visual Studio eases programming and provides online help instantly

2. **Description of Visual Studio 2015**
   -Locate VS 2015 using search option in Windows

- A start screen appears which has following components :

| | |
|---|---|
| Menu Bar – gives the different menu options for editing, building, debgugging, deploying, profiling and analysing the code. | |
| Tool Bar – gives quick access to the frequently used options | |
| Actions:<br><br>New Project – to create a new project<br><br>Open Project – to open existing project<br><br>Recent – quick opening of recently created projects | News Section – information on latest news and development in .NET and its languages<br><br>Also news on Micosoft products is displayed<br><br>A few video links are also provided that helps understanding .NET featues |

- After a new project option is selected following window appears that enables creation of a new application using the language of your choice listed under .NET framework

| Language Selection<br>- VC#<br>- VB<br>- MC++<br>- J#<br>- F#<br>- Others<br>In each language application category:<br>- Web<br>- iOS<br>- Android<br>- Cloud<br>- Sliverlight<br>- Testing<br>- Others | **.NET framework version – currently supported 4.6**<br>- Application subtypes supported (specific applications supported)<br>- - Windows Forms – GUI but only on Windows Desktops<br>- Console Applications – Interaction with keyboard and monitor<br>- UWP Graphical Application – Interaction using GUI Forms and widgets and can run on any windows devices<br>- ASP.NET application – web development application<br>- Class Library – used not as an executable but for helping other applications by invoking the features in the library |
|---|---|
| - Giving Name of file – which is also the name of name space<br>- Folder Name – folder where the project and solution is saved<br>- Solution Name – Name of executable/dynamic link library to be created | |

Once the language and application is selected following window appears (Assuming Console Application)

| Menu bar - **Menu Bar – gives the different menu options for editing, building, debgugging, deploying, profiling and analysing the Console Application** | |
|---|---|
| Tool bar - **gives quick access to the frequently used options** | |
| Code and text editor window. Default code is generated that includes some predefined name spaces, user defined name space, class and a main method where execution starts | Solution Explorer files:<br>- Properties<br>- References<br>- App.config<br>- Program.cs |

**Properties :** Contains meta-data information regarding name of program, author name, date of creation, version number, organization , purpose and other details

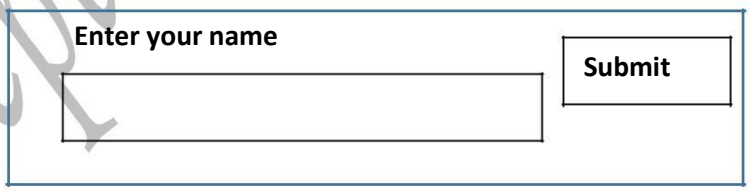
**References:** Links to the other external resources and assemblies used by this program

**App.config:** application level settings like .NET framework version to be used, compiler version etc

**-Program.cs –** Default name for program generated

**3. Graphical Application to display prompt Enter you name, collect the name and on pressing submit, should display a greeting message for the user with specified name**

Step 1: Start Visual Studio 2015
Step 2: Create a new project – language VC#- application Windows UWP
Step 3: Select minimum and target version to 10240
Step 4: Give file, folder and solution details
Step 5: An empty canvas is generated on which different widgets can be laid out. The canvas and widgets are created using XAML (eXtensible Application Markup Langauge)
Step 6: Drag and drop TextBlock, Text Box, Button widgets and layout should look as follows:



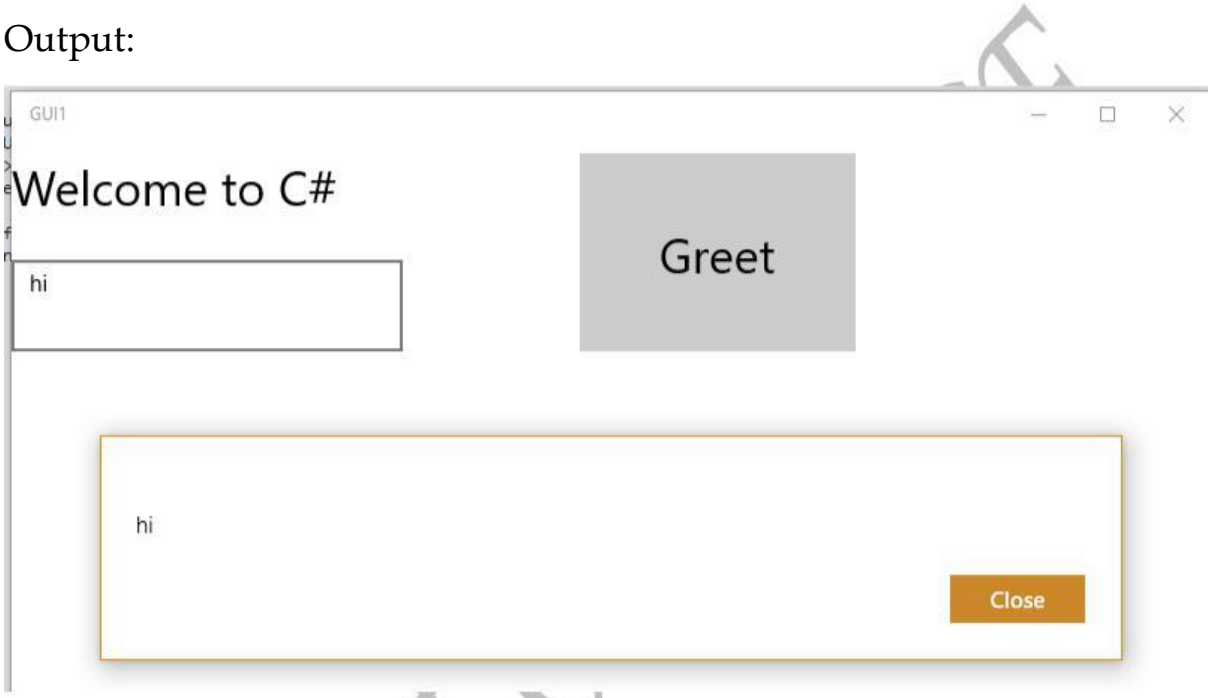

Step 7- Set the properties of each widget like : TextBlock text, size, TextBox: text, size Button: content, size. Name widgets: TextBox: t1, Button b1

Step 8: Click on the event icon of button properties and write following C# code:

```
private void b1Click(object sender, RoutedEventArgs e)
        {
          MessageDialog m=new MessageDialog($"Hello {t1.Text}");
            m.ShowAsync();
        }
```

Step 9: Execute on a local machine

Output:




**Chapter 2: Working with variables, operators and expressions**

**1. Graphical Application to display illustrative values of primitive data types**

Step 1: Start Visual Studio 2015

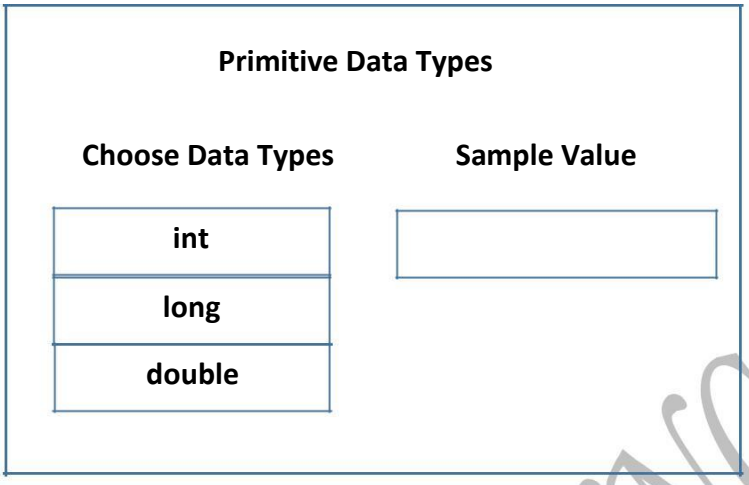
Step 2: Create a new project – language VC#- application Windows UWP

Step 3: Select minimum and target version to 10240

Step 4: Give file, folder and solution details

Step 5: An empty canvas is generated on which different widgets can be laid out. The canvas and widgets are created using XAML (eXtensible Application Markup Langauge)

Step 6: Drag and drop 3 TextBlocks, List Box, and TextBox as follows




Step 7- Set the properties of each widget like : TextBlock text, size, TextBox: text, size List Box – Add 3 items and set their content properties

Name widgets: TextBox: textBlock1, ListBox listbox

Step 8: Click on the event icon of ListBox properties and write following C# code:

```
private void show(object sender, SelectionChangedEventArgs e)
  {
    ListBoxItem l = listBox.SelectedItem as
    ListBoxItem; switch(l.Content.ToString()) {

      case "Int":showint();
        break;
      case "Long":showlong();
        break;
      case "Float":showfloat();
        break;

    }


  }
  private void showint()
  {
```

```
        int i = 100;
        textBlock1.Text = i.ToString();
    }
    private void showlong()
    {
        long l = 1000000L;
        textBlock1.Text = l.ToString();
    }
    private void showfloat()
    {
        float f = 3.5f;
        textBlock1.Text = f.ToString();
    }
```

Step 9: Execute on a local machine

**2. Code to demonstrate definite assignment rule:**

```
class Program
  {
    static void Main(string[] args)
    {
      int x;
      Console.WriteLine(x);
    }
  }
```

Compiler Error: x not assigned any value

**3. Code to demonstrating the NaN and Infinity**

```
class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine(5 / 0.0);
      Console.WriteLine(0.0 / 0.0);
      Console.WriteLine(7.0 %2.4);
      Console.WriteLine(5 /2);
      Console.WriteLine(5 / 2.0);
    }
  }
```

```
C:\WINDOWS\system32\cmd.exe
∞
NaN
2.2
2
2.5
Press any key to continue . . .
```

4. **Graphical Application to display working of arithmetic operators**

Step 1: Start Visual Studio 2015

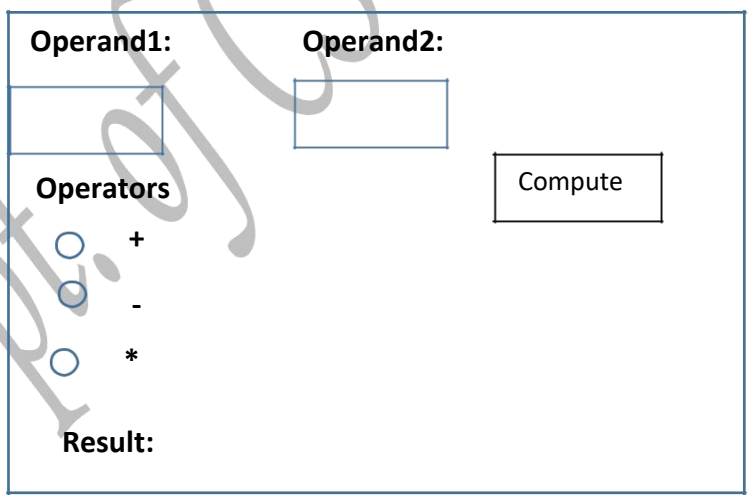
Step 2: Create a new project – language VC#- application Windows UWP

Step 3: Select minimum and target version to 10240

Step 4: Give file, folder and solution details

Step 5: An empty canvas is generated on which different widgets can be laid out. The canvas and widgets are created using XAML (eXtensible Application Markup Langauge)

Step 6: Drag and drop 5 TextBlocks, 2 TextBoxes, 3 Radio Buttons, and Button as follows



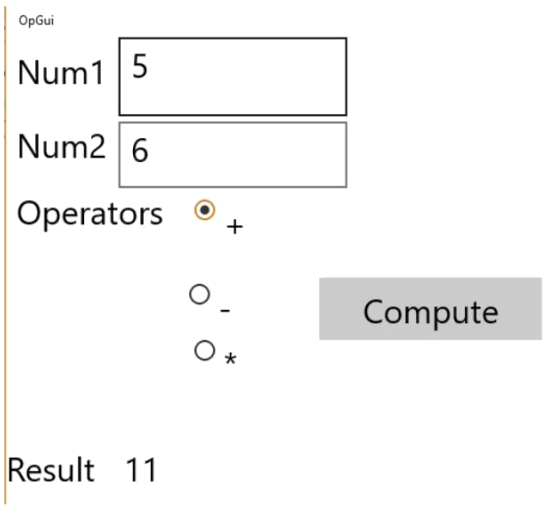

Step 7- Set the properties of each widget like : TextBlock text, size, TextBox: text, size Radio Buttons- content, Button – content, size

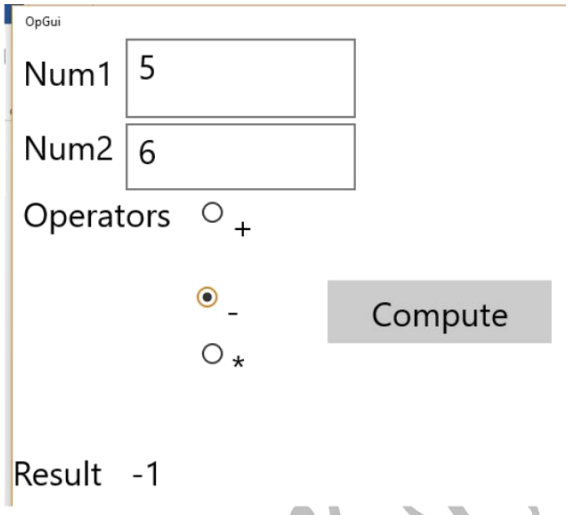
Name widgets: TextBoxes: t1 (operand 1) and t2 (operand2) , RadioButtons – plus and minus, TextBlocks –tb1 (for result)

Step 8: Click on the event icon of Button properties and write following C# code:

```
private void compute(object sender, RoutedEventArgs e)
    {
      if ((bool)plus.IsChecked)
         add();
      if ((bool)minus.IsChecked)
         sub();
     if ((bool)prod.IsChecked)
        prod();
    }
    private void add()
    {
      int res = int.Parse(t1.Text) + int.Parse(t2.Text);
      tb1.Text = $"Result:{res}";
    }
    private void sub()
    {
      int res = int.Parse(t1.Text) - int.Parse(t2.Text);
      tb1.Text = $"Result:{res}";
    }

    private void prod()
    {
      int res = int.Parse(t1.Text) * int.Parse(t2.Text);
      tb1.Text = $"Result:{res}";
    }
```

Step 9: Execute on a local machine




## Chapter 3: Writing methods and Applying scope:

## 1. Program demonstrating methods and expression bodied methods:

```
namespace Basic1
{
   class Program
   {
      int x;
      public int add(int x, int y) =>x+y;
      public void print(int ans) => Console.WriteLine($"ans is {ans}");
      static void Main(string[] args)
      {

         Program p = new Program();
         int res = p.add(2, 3);
         p.print(res);
         string usn = "xxx";
         string name = "yyy";
         float marks = 19.2F;
         int age = 15;
         Console.WriteLine($"Student usn={usn} and name={name} and marks={marks} and age={age}");
      }
   }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
ans is 5
Student usn=xxx and name=yyy and marks=19.2 and age=15
Press any key to continue . . .
```

## 2. C# code to demonstrate optional parameters and named arguments:

```
namespace Method1
{
  class Program
  {
    int add(int x,int y)
    {
      return x + y;

    }
    float add(float x,float y)
    {
      return x + y;
    }
    int add(int x, int y,int z)
    {
      return x + y+z;

    }
    void demo(int x=0,float y=0.0f,int A=0)
    {
      Console.WriteLine($"x={x},y={y},A={A}");
    }

     static void Main(string[] args)
    {
      Program p = new Program();
      p.demo(1, 2.2F,3);
      p.demo(1, 2.2F);
      p.demo(1);
      p.demo(A:3,x:4,y:0.5f);


    }
  }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
x=1,y=2.2,A=3
x=1,y=2.2,A=0
x=1,y=0,A=0
x=4,y=0.5,A=3
Press any key to continue . .
```

3. Program showing ambiguity with optional parameters and named arguments

```
namespace Method1
{
  class Program
  {

    void demo(int x=0,float y=0.0f,int A=0)
    {
      Console.WriteLine($"x={x},y={y},A={A}");
    }
    void demo(int x=0,float y=0.0f)
    {
      Console.WriteLine($"x={x},y={y}");
    }

     static void Main(string[] args)
    {
      Program p = new Program();
      p.demo(1, 2.2F,3);
      p.demo(1, 2.2F);
      p.demo(1); //Ambiguity
      p.demo(A:3,x:4,y:0.5f);

        }
  }
}
```

## Chapter 4,5: Decision Statements and Loops

## 1. C# code to check a number is prime or not:

```
class Program
  {
    static void Main(string[] args)
    {

      int N = int.Parse(Console.ReadLine());
      for (int i=2;i<=N/2;i++)
```

```
        {
            if(N%i==0)
            {
                Console.WriteLine("not prime");
                return;
            }

        }
        Console.WriteLine("prime");

    }
  }
```

Output:

```
C:\WINDOWS\system32\cmd.exe
23
prime
Press any key to continue .
```

## 2. Program to print numbers 1 to N without loops

```
class Program
  {
    static int x = 1;
    static void Main(string[] args)
    {
        int N = 5;

        if (x <=N)
        {
            Console.Write(x + " ");
            x++;
            Main(args);
        }
    }
  }
```

Output:

```
C:\WINDOWS\system32\cmd.exe
1 2 3 4 5 Press any key to continue .
```

## 3. C# code to convert decimal to octal

```
class Program
  {
    static void Main(string[] args)
    {
```

```
        Console.WriteLine("Enter a number");
        int N = int.Parse(Console.ReadLine());
        string acc = "";
        while(N>0)
        {
          int rem = N % 8;
          acc = rem+acc;
          N = N / 8;
        }
        Console.WriteLine(acc);

    }
  }
```

Output:

```
C:\WINDOWS\system32\cmd.exe
Enter a number
22
26
Press any key to continue . .
```

## 4. Adding 2 numbers without using arithmetic operators:

```
class Program
  {
    static void Main(string[] args)
    {
      int x = 15, y = 18;
      while(y!=0)
      {
        int carry = x & y;
        x = x ^ y;
        y= (carry & y)<< 1;
      }
      Console.WriteLine(x);

    }
  }
```

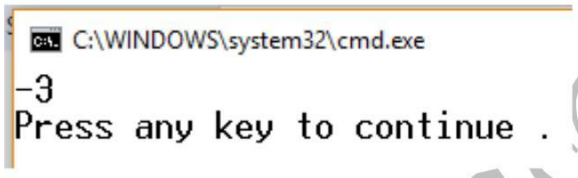
Output:

```
C:\WINDOWS\system32\cmd.exe
33
Press any key to continue .
```

### 5. Subtracting 2 numbers without arithmetic operators:

```
class Program
  {
    static void Main(string[] args)
    {
      int x = 15, y = 18;
      while(y!=0)
      {
        int carry = ~x & y;
        x = x ^ y;
        y= (carry & y)<< 1;
      }
      Console.WriteLine(x);

    }
  }
```

Output:




## Chapter 6: Managing Errors and Exceptions:

### 1. Demonstrating handling of exceptions:

```
class Program
  {
    static void Main(string[] args)
    {
      int n;
      Console.WriteLine("Enter n");
      try
      {
        n = int.Parse(Console.ReadLine());
      }

      catch (FormatException e)
      {
        Console.WriteLine("enter only integer");
      }
      catch (OverflowException e)
      {
        Console.WriteLine("overflow");
      }
```

```
        catch
        {
            Console.WriteLine("any");
        }


    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
Enter n
aa
enter only integer
Press any key to continue . . .
```

```
C:\WINDOWS\system32\cmd.exe
Enter n
9900923050
overflow
Press any key to continue .
```

## 2. Program demonstrating the use of keyword: checked

```
class Program
  {

    static void Main(string[] args)
    {

      checked
      {
          int num = int.MaxValue + 1; //Compiler error
          int x = int.MaxValue;
          Console.WriteLine(x);
      }



    }
  }
```

# Chapter 7: Creating and Managing class and objects:

## 1. Program to demonstrate writing classes and creating instances:

```
class Circle
  {
    float radius;
    int x, y;
    public Circle(float radius, int x, int y)
    {
      this.radius = radius;
      this.x = x;
      this.y = y;
    }
    public Circle()
    {
      radius = 1.0f;
      x = 1;
      y = 1;
    }
    public double area()
    {
      return Math.PI*radius*radius;
    }
   public float getRadius()
    {
      return radius;
    }


  }

  class Point
  {
    int x, y;
    public Point(int x,int y)
    {
      this.x = x;
      this.y = y;

    }
    public double distance(int x1,int y1)
    {
      return Math.Sqrt((x - x1) * (x - x1) + (y - y1) * (y - y1));
    }
  }
  class App //driver class
  {
    static void Main(string[] args)
    {
      Circle c = new Circle(2.1f, 1, 2);
```

```
        Console.WriteLine(c.area());
        Point p = new Point(1, 2);
         Console.WriteLine(p.distance(2, 3));
      }
   }
```

```
C:\WINDOWS\system32\cmd.exe
13.8544223439874
1.4142135623731
Press any key to continue . .
```

## 2. Program to use static variables for counting the number of objects:

```
namespace Class1
{
   class Point
   {
      int x, y;
      public static int count;
      public Point(int x,int y)
      {
         this.x = x;
         this.y = y;
         count++;
      }
      public double distance(int x1,int y1)
      {
         return Math.Sqrt((x - x1) * (x - x1) + (y - y1) * (y - y1));
      }
   }
   class PointApp //driver class
   {
      static void Main(string[] args)
      {
         Point p = new Point(1, 2);
         Point p1 = new Point(2, 2);
         Point p2 = new Point(2, 2);
         Console.WriteLine(Point.count);
      }
   }
}
```

```
C:\WINDOWS\system32\cmd.exe
3
Press any key to continue .
```

# Chapter 8: Understanding Values and References:

## 1. Program illustrating Shallow copy:

```
class Point
  {
    int x, y;
    public Point(int x,int y)
    {
      this.x = x;
      this.y = y;
    }
    public int getX()
    {
      return x;
    }
    public int getY()
    {
      return y;
    }
    public void setX(int x)
    {
      this.x = x;
    }
  }
  class Circle
  {
    public Point p;
    double radius;
    public void setRadius(double radius)
    {
      this.radius = radius;
    }
    public Circle(double radius,Point p)
    {
      this.radius = radius;
      this.p = p;
    }
    public void display()
    {
      Console.WriteLine($"{radius},{p.getX()},{p.getY()}");
    }
    public Circle copy()
    {
      return new Circle(this.radius, this.p);
    }

  }
  class CircleDriver
  {
    static void Main(string[] args)
```

```
    {
        Point p = new Point(1, 2);
        Circle c = new Circle(3.4, p);
        Circle c1 = c.copy();
        c1.p.setX(4);
        c.display();
        c1.display();
    }
}
```

```
C:\WINDOWS\system32\cmd.exe
3.4,4,2
3.4,4,2
Press any key to continue . . .
```

## 2. Program illustrating Deep copy:

```
class Point
  {
    int x, y;
    public Point(int x,int y)
    {
      this.x = x;
      this.y = y;
    }
    public int getX()
    {
      return x;
    }
    public int getY()
    {
      return y;
    }
    public void setX(int x)
    {
      this.x = x;
    }
  }
  class Circle
  {
    public Point p;
    double radius;
    public void setRadius(double radius)
    {
      this.radius = radius;
    }
    public Circle(double radius,Point p)
    {
```

```
        this.radius = radius;
        this.p = p;
    }
    public void display()
    {
        Console.WriteLine($"{radius},{p.getX()},{p.getY()}");
    }
    public Circle copy()
    {
        return new Circle(this.radius,new Point(this.p.getX(),this.p.getY()));
    }

}
class CircleDriver
{
    static void Main(string[] args)
    {
        Point p = new Point(1, 2);
        Circle c = new Circle(3.4, p);
        Circle c1 = c.copy();
        c1.p.setX(4);
        c.display();
        c1.display();
    }
}
```

```
C:\WINDOWS\system32\cmd.exe
3.4,1,2
3.4,4,2
Press any key to continue . .
```

## 3. Program illustrating Value Types and References:

```
class WrappedInt
{
    public int num;
    public WrappedInt(int num)
    {
        this.num = num;
    }
}


class Program
{
    public void valueinc(int num)
    {
        num++;
```

```
    }

    public void refinc(WrappedInt w)
    {
        w.num++;
    }

    static void Main(string[] args)
    {
        Program p = new Program();
        int a = 2;
        Console.WriteLine($"Before {a}");
        p.valueinc(a);
        Console.WriteLine($"After {a}");

        WrappedInt w = new WrappedInt(a);
        Console.WriteLine($"Before {w.num}");
        p.refinc(w);
        Console.WriteLine($"After {w.num}");

    }
}
```

C:\WINDOWS\system32\cmd.exe

```
efore 2
fter 2
efore 2
fter 3
ress any key to continue . . .
```

## 4. Program demonstrating null values:

```
class Circle
{
    int x, y;
    double radius;
    public Circle(double radius,int x,int y)
    {
        this.radius = radius;
        this.x = x;
        this.y = y;
    }
    public double area()
    {
        return Math.PI * radius * radius;
    }
}
class Program
{
```

```
    static void Main(string[] args)
    {

      Circle c = null; ;
      if(c==null)
      {
        Console.WriteLine("cant find area");
      }
      else
        Console.WriteLine($"{c.area()}");
    }
  }
```

```
C:\WINDOWS\system32\cmd.exe
cant find area
Press any key to continue . . .
```

## 5. Program illustrating Nullable types:

```
class Program
{
  static void Main(string[] args)
  {
    int? marks = 20;
    Console.WriteLine($"{marks.HasValue}, {marks.Value}");
  }
}
```

```
C:\WINDOWS\system32\cmd.exe
True, 20
Press any key to continue
```

## 6. Program illustrating ref and out keywords:

```
class Program
{
  public void valueinc(ref int num)
  {
    num++;
  }

  public void outinc(out int num)
  {
    num = 20;
  }

  static void Main(string[] args)
```

```
    {
        Program p = new Program();
        int a=2;
        Console.WriteLine($"Before {a}");
        p.valueinc(ref a);
        Console.WriteLine($"After {a}");

        a = 4;
        Console.WriteLine($"Before {a}");
        p.outinc(out a);
        Console.WriteLine($"After {a}");

    }
}
```

C:\WINDOWS\system32\cmd.exe

```
Before 2
After 3
Before 4
After 20
Press any key to continue . .
```

Note:

(i) Compiler error: forgetting to initialize ref parameter before sending it

```
Program p = new Program();
int a;
Console.WriteLine($"Before {a}");
p.valueinc(ref a);
```

(ii) Compiler error: forgetting to assign out parameter before leaving method

```
    public void outinc(out int num)
    {

    }
```

*C# Programming Assignment*

Date: 05/09/2018

**Rules:**

**(i) Assignment duration 7 – 10.30 pm**

**(ii) Can submit any number of times**

**(iii) Code your program in https://www.onlinegdb.com/ or Visual Studio software**

**(iv) Scan the code and output/error and send (WhatsApp) to 9900923050**

## Question:

**There are 72 students enrolled in 5th 'A' Section for C# and .NET. Assume they are numbered from 1 to 72. Each student should be assigned to a group and grouping is done based on number of 1's in number assigned to student. E.g if student number is 5, he/she is assigned to Group 2. If Student number is 17, he/she is also assigned to Group 2.**

**Write a C# code that displays Groups in descending order of the number of students enrolled. Also find group having maximum number of students enrolled.**

# Don't use Arrays!!!!!!!



## Solution:

```
using System;
class HelloWorld {
 static int count_ones(int num)
 {
    int count=0;
    while(num>0)
    {
```

```
        if(num%2==1)
         count++;
        num/=2;
    }
    return count;
}
 static void Main() {
     int g1=0,g2=0,g3=0,g4=0,g5=0,g6=0,g7=0;
     int max=0,gno=0;
     for(int i=1;i<=72;i++)
     {
        int n_o=count_ones(i);
        switch(n_o)
        {
           case 1:g1++;
                 if(g1>max)
                 {
                   max=g1;
                   gno=1;
                 }
                   break;
           case 2:g2++;
                 if(g2>max)
                 {
                   max=g2;
                   gno=2;
                 }
                   break;
           case 3:g3++;
                  if(g3>max)
                 {
```

```
        max=g3;
        gno=3;
      }
        break;
case 4:g4++;
      if(g4>max)
      {
        max=g4;
        gno=4;
      }
        break;
case 5:g5++;
      if(g5>max)
      {
        max=g5;
        gno=5;
      }
        break;
case 6:g6++;
      if(g6>max)
      {
        max=g6;
        gno=6;
      }
        break;
case 7:g7++;
      if(g7>max)
      {
        max=g7;
        gno=7;
      }
```

```
                break;
        }
    }
    Console.WriteLine("Group " + gno +" has maximum number of ones: "+ max);
    for(int i=72;i>=0;i--)
    {
        if(g1==i)
            Console.WriteLine(" Group 1 : "+g1 );
        if(g2==i)
            Console.WriteLine(" Group 2 : "+g2 );
        if(g3==i)
            Console.WriteLine(" Group 3 : "+g3 );
        if(g4==i)
            Console.WriteLine(" Group 4 : "+g4 );
        if(g5==i)
            Console.WriteLine(" Group 5 : "+g5 );
        if(g6==i)
            Console.WriteLine(" Group 6 : "+g6 );
        if(g7==i)
            Console.WriteLine(" Group 7 : "+g7 );
    }

  }
}
```



Write a C# program that has a method **"calci"** which takes 2 inputs and does arithmetic operations. The main method should call **"calci"** by taking inputs from user and return results of arithmetic operations to the user.

```
class Program
```

```
    {
        static void calci(int op1,int op2,out int sum,out int
diff,out int prod,out int quot)
        {
            sum= op1+op2;
            diff= op1 - op2;
            prod= op1* op2;
            quot= op1 / op2;

        }
        static void Main(string[] args)
        {
            int a = 6, b = 3;
             int sum,diff,prod,quot;
            calci(a, b,out sum, out diff, out prod, out quot);
            Console.WriteLine($"{sum},{diff},{prod},{quot}");
        }
    }
```

Output:

```
C:\WINDOWS\system32\cmd.exe
9,3,18,2
Press any key to continue .
```

**Write a C# program that simulates a biased weighing machine, by writing a method which takes a weight and subtract weight by 2.0 kg. Input weight should be taken from user and return biased weight back to user.**

```
class Program
    {

        static void reduce_weight(ref double weight)
        {
            weight = weight - 2;
        }
        static void Main(string[] args)
        {
            double wt = 52.3;
            reduce_weight(ref wt);
            Console.WriteLine(wt);

        }
    }
```

```
C:\WINDOWS\system32\cmd.exe
50.3
Press any key to continue .
```

**C# code to demonstrate boxing and unboxing:**

```
int x = 10;
object o = x;  //boxing
int j = (int)o;  //unboxing
```

**Write a C# program that has a method "calci" which takes 2 inputs and does arithmetic operations. The main method should call "calci" by taking inputs from user and return results of arithmetic operations to the user.**

```
class Program
    {
        static void calci(int x,int y,out int sum,out int diff, out int prod,out int
quot)
        {
            sum = x + y;
            diff = x - y;
            prod = x * y;
            quot = x / y;
        }
        static void Main(string[] args)
        {
            int a = 6, b = 2;
            int sum, diff, prod, quot;
            calci(a, b, out sum, out diff, out prod, out quot);

Console.WriteLine($"Sum={sum},Diff={diff},Product={prod},Quotient={quot}");

        }
    }
```

```
C:\WINDOWS\system32\cmd.exe
Sum=8,Diff=4,Product=12,Quotient=3
Press any key to continue . . .
```

**Write a C# program that simulates a biased weighing machine, by writing a method which takes a weight and subtract weight by 2.0**

## kg. Input weight should be taken from user and return biased weight back to user

```
class Program
  {
      static void baised_weight(ref double weight)
      {
          weight = weight - 1;
      }
      static void Main(string[] args)
      {

          double weight = 61.2;
          baised_weight(ref weight);
          Console.WriteLine($"Biased weight is {weight}");
      }
  }
```

```
Biased weight is 60.2
Press any key to continue .
```

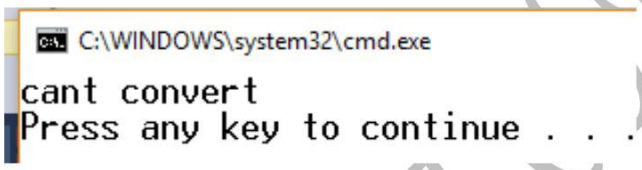
## C# code to demonstrate *is* keyword:

```
class WrappedInt
    {
        int num;
    }
    class Circle
    {
        int rad;
    }
    class Program
    {
        static void Main(string[] args)
        {
            WrappedInt w = new WrappedInt();
            Circle c = new Circle();
            object o = c;
            if(o is WrappedInt)
            {
                WrappedInt w1 = (WrappedInt)o;
            }
            else
            {
                Console.WriteLine("cant convert");
            }


        }
    }
```

```
C:\WINDOWS\system32\cmd.exe
cant convert
Press any key to continue .
```

## C# code to demonstrate *as* keyword:

```
class WrappedInt
    {
        int num;
    }
    class Circle
    {
        int rad;
    }
    class Program
    {
        static void Main(string[] args)
        {

            WrappedInt w = new WrappedInt();
            Circle c = new Circle();
            object o = c;

            WrappedInt w1 = o as WrappedInt;
            if (w1==null)
            {
                Console.WriteLine("cant convert");
            }


        }
    }
```




## C# program to perform swapping of 2 integers (also demonstrating unsafe pointers)

```
class WrappedInt
    {
        int num;
    }
    class Circle
    {
        int rad;
    }
    class Program
    {
        unsafe static void swap(int *x,int *y)
        {
            int temp = *x;
            *x = *y;
            *y = temp;
        }
        static void Main(string[] args)
        {
            int a = 12, b = 10;
```

```
            Console.WriteLine($"Before: {a},{b}");
            unsafe
            {
                swap(&a, &b);
            }
            Console.WriteLine($"After: {a},{b}");

        }
    }
```

C:\WINDOWS\system32\cmd.exe

```
Before: 12,10
After: 10,12
Press any key to continue .
```

# Chapter 9: Creating value types with enumerations and structures

## C# code to demonstrate enumerations in operation:

Changing underlying type of enumeration

Aliasing enumeration values

```
class Program
    {
        enum  Season :short { Summer=5, Rainy,Autumn=Rainy, Winter};
        static void Main(string[] args)
        {
            Season hot = Season.Summer;
            hot++;
            hot++;
            Console.WriteLine(hot); (Starts with Summer and 2 increments lead to
            Winter) hot++;
            Console.WriteLine(hot);(When no symbol mapped, u get the value itself)
        }
    }
```

```
C:\WINDOWS\system32\cmd.exe
Winter
8
Press any key to continue .
```

## C#code to demonstrate structures

```
class Program
    {
        enum Month { Jan=31,Feb=28,March=Jan};//Enumeration another
        example struct Time
        {
            int hours;
            int mins;
            int secs;
            public Time(int hours,int mins,int secs)
            {
                this.hours = hours;
                this.mins = mins;
                this.secs = secs;
            }
            //Overloaded Constructors
            public Time(int hours, int mins)
            {
                this.hours = hours;
                this.mins = mins;
                this.secs = 20;

            }
             //getter required as fields are private
            public int getHours()
            {
                return hours;
            }
```

```
        }
        static void Main(string[] args)
        {
            Month cur_month = Month.March;
            Console.WriteLine((int)cur_month);  //get value for enumeration
            Time t = new Time(20,30,40); //Parameterized constructor
            Time t1=new Time(); //Default constructor

            Console.WriteLine(t.getHours());
            Console.WriteLine(t1.getHours());

        }
    }
```

```
C:\WINDOWS\system32\cmd.exe
31
20
0
Press any key to continue . .
```

## Structure copy:

Time t1;

Time t2=t1; //Not possible – compiler error as RHS should be fully initialized

## Correct way:

Time t1=new Time(10,20,30);

Time t2=t1; //Creates a copy

# Chapter 10: Using arrays

## 1. C# program to demonstrate using and populating an array (both Deterministic and Random arrays)

```
class Program
    {
        static void display(int[] arr)
        {
            foreach(int a in arr)
            {
                Console.Write(a + " ");
            }
            Console.WriteLine();
        }
        static void Main(string[] args)
        {
            int[] room_nos = { 202, 203, 204 };
            Random r = new Random();
            int[] random_desk_nos = { r.Next() % 12, r.Next() % 12, r.Next() % 12 };
            display(room_nos);
            display(random_desk_nos);


        }
    }
```

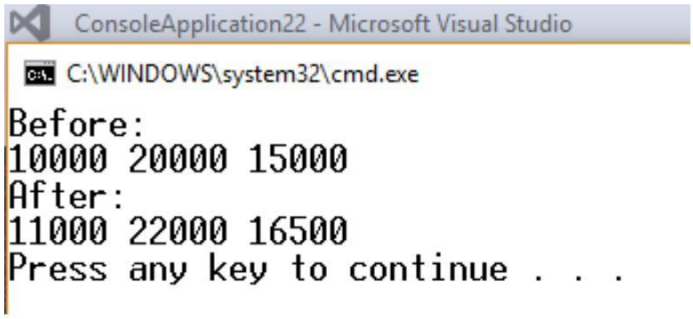
C:\WINDOWS\system32\cmd.exe

```
202 203 204
9 10 4
Press any key to continue . .
```

## 2. Passing Arrays as parameters: Example program: Write a C# program to update all employee salaries maintained in a 1D array by 10%

```
class Program
    {
        static void display(float[] arr)
        {
            foreach (float a in arr)
            {
                Console.Write(a + " ");
            }
            Console.WriteLine();
        }

        static void update_salaries(float[] sal)
        {
            for(int i=0;i<sal.Length;i++)
            {
                sal[i] = sal[i] * 1.1f;
            }
        }
        static void Main(string[] args)
        {
            float[] emp_sal = { 10000, 20000, 15000 };
            Console.WriteLine("Before:");
```

```
            display(emp_sal);
            update_salaries(emp_sal);
            Console.WriteLine("After:");
            display(emp_sal);

        }
    }
```




**3. Passing and Returning Arrays as parameters to methods: Example program: Write a C# program to remove 0's from array passed**

```
class Program
    {
        static void display(int[] arr)
        {
            foreach(int a in arr)
            {
                Console.Write(a + " ");
            }
            Console.WriteLine();
        }

        public static int[] remove_zeros(int[] arr)
        {

            int cnt = 0;
            for(int i=0;i<arr.Length;i++)
            {
                if (arr[i] != 0)
                    cnt++;
            }
            int[] temp = new int[cnt-1];
            int j = 0;
            for (int i = 0; i < cnt; i++)
            {
                if (arr[i] != 0)
                    temp[j++] = arr[i];
            }
            return temp;
        }
        static void Main(string[] args)
        {
            int[] a = { 1, 2, 0, 3, 0, 0, 4 };
            Console.WriteLine("Before:");
            display(a);
            int[] z = remove_zeros(a);
            Console.WriteLine("After:");
            display(z);
```

```
        }
    }
```

```
C:\WINDOWS\system32\cmd.exe
Before:
1 2 0 3 0 0 4
After:
1 2 3
Press any key to continue . . .
```

## 4. Demonstrating Array Copy:

```
class Program
    {
        static void display(string[] arr)
        {
            foreach (string a in arr)
            {
                Console.Write(a + " ");
            }
            Console.WriteLine();
        }


        static void Main(string[] args)
        {

            string[] subjects = { "MEC", "CN", "DBMS", "ATC"
            }; //Reference copy
            string[] subjects1 = subjects;
            Console.WriteLine("Original Array");
            display(subjects);
            Console.WriteLine("Reference copy results");
            display(subjects1);

            //Element by Element copy
            string[] subjects2 = new string[subjects.Length];
            for (int i = 0; i < subjects.Length; i++)
                subjects1[i] = subjects[i];
            Console.WriteLine("Element by Element copy results");
            display(subjects2);

            //Instance method CopyTo
            string[] subjects3 = new string[subjects.Length];
            subjects1.CopyTo(subjects3,0);//start copying from 0th
            index Console.WriteLine("Instance method CopyTo results");
            display(subjects3);

            //static method Copy
            string[] subjects4 = new string[subjects.Length];
            Console.WriteLine("Static method Copy results");
            Array.Copy(subjects,subjects4, subjects.Length);//number of elements to
copy -third parameter
            display(subjects4);

            //cloning
            string[] subjects5 = subjects.Clone() as
            string[]; Console.WriteLine("Clone() results");
```

```
            display(subjects5);

        }
    }
```

```
C:\WINDOWS\system32\cmd.exe
Original Array
MEC CN DBMS ATC
Reference copy results
MEC CN DBMS ATC
Element by Element copy results

Instance method CopyTo results
MEC CN DBMS ATC
Static method Copy results
MEC CN DBMS ATC
Clone() results
MEC CN DBMS ATC
Press any key to continue . . .
```

**Note: Reference copyonly references are copied. So making changes in copy affects original array and vice versa. Element by element copy creates a true copy and changes in one does not affect vother. The code of copying can be shortened using instance method CopyTo() or static method Copy() or Clone(). In CopyTo() , u can specify from where copying has to start. In static method, Copy(), u can specify how many elements u want to copy. The shortest one liner syntax for copying is in Clone() but it needs appropriate casting.**

### 5. Demonstrating a 2D Rectangular Array: Write a C# program to arrange students in a 2D Array of deskno's

```
class Program
    {
        static void Main(string[] args)
        {

            string[,] students = new string[3,2];
            students[0, 0] = "001";
            students[0, 1] = "002";
            students[1, 0] = "003";
            students[1, 1] = "004";
            students[2, 0] = "005";
            students[2, 1] = "006";

             for(int i=0;i<students.GetLength(0);i++)
             {
                 for(int j=0;j<students.GetLength(1);j++)
                 {
                     Console.Write(students[i,j] + " ");
                 }
                 Console.WriteLine();
             }
```

```
        }
    }
```

```
ConsoleApplication22 - Microsoft Visual Studio
C:\WINDOWS\system32\cmd.exe
001 002
003 004
005 006
Press any key to continue . . .
```

**6. Demonstrating a 2D Jagged Array: Write a C# program to create jagged array of students in different benches:**

```
class Program
    {

        static void Main(string[] args)
        {
             string[][] students = new string[3][];
            students[0] = new string[0];
             students[1] = new string[2] { "001", "002" };
             students[2]=new string[5] { "001", "002", "003","004","005" };


             for(int i=0;i<students.GetLength(0);i++)
             {
                 for(int j=0;j<students[i].Length;j++)
                 {
                     Console.Write(students[i][j] + " ");
                 }
                 Console.WriteLine();
             }

        }
    }
```

```
C:\WINDOWS\system32\cmd.exe

001 002
001 002 003 004 005
Press any key to continue . .
```

# Chapter 11: Understanding parameter Arrays

## 1. Passing variable number of parameters with same or variable type using param

```
class Program
    {
        static void m1(int x,int y)
        {
            Console.WriteLine("non param");
        }
        static void m1(params int[] x)
        {
            Console.WriteLine("param");
        }
        static void register(params object[] cust)
        {
            for (int i = 0; i < cust.Length; i += 2)
                Console.WriteLine($"{cust[i]}-{cust[i + 1]}");
        }
        static void register(params string[] cust)
        {
            for(int i=0;i<cust.Length;i++)
            Console.WriteLine($"{cust[i]}");
        }
        static void register(params int[] cust)
        {
            for (int i = 0; i < cust.Length; i ++)
                Console.WriteLine($"{cust[i]}");
        }

        static void Main(string[] args)
        {
            m1(2, 3);
            register("A",100);

            register("B",111, "C",222, "D",333);
            register("X",200, "Y",300);
            register("C1", "C2", "c3", "c4", "c5", "c6");
            register(100, 200);

        }
    }
```

```
C:\WINDOWS\system32\cmd.exe
non param
A-100
B-111
C-222
D-333
X-200
Y-300
C1
C2
c3
c4
c5
c6
100
200
Press any key to continue . . .
```

## 2. Adding variable number of data using param

```
class Program
    {
        static object sum(params object[] arr)
        {
            int isum = 0;
            float fsum = 0.0f;
            double dsum = 0.0;
            object o = null;
            for(int i=0;i<arr.Length;i++)
            {
                if (arr[i] is int)
                {
                    isum = isum + (int)arr[i];

                }

                if (arr[i] is float)
                {
                    fsum = fsum + (float)arr[i];

                }
                if (arr[i] is double)
                {
                    dsum = dsum + (double)arr[i];

                }
            }
            o = isum + fsum + dsum;
            return o;

        }
        static int min(params int[] arr)
        {

            int min = arr[0];
            for(int i=1;i<arr.Length;i++)
            {
                if(arr[i]<min)
                {
                    min = arr[i];
                }
            }
            return min;

        }
        static void Main(string[] args)
        {
            Console.WriteLine(sum(2, 3,3.1));
            Console.WriteLine(sum(4, 5.5, 6, 7.7, 8));
            Console.WriteLine(sum(1.4f));
        }
    }
```

```
ConsoleApplication27 - Microsoft Visual Studio
C:\WINDOWS\system32\cmd.exe
8.1
31.2
1.39999997615814
Press any key to continue . . .
```

## 3. Finding minimum of any number of integers using param:

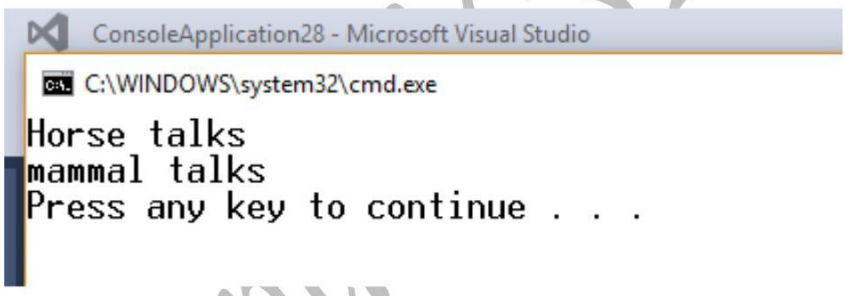
```
class Program
    {

        static int min(params int[] arr)
        {

            int min = arr[0];
            for(int i=1;i<arr.Length;i++)
            {
                if(arr[i]<min)
                {
                    min = arr[i];
                }
            }
            return min;

        }
        static void Main(string[] args)
        {
            Console.WriteLine(min(1));
            Console.WriteLine(min(3, 1, 2));
            Console.WriteLine(min(4, 0, 5, -1, 4));
        }
    }
```

```
ConsoleApplication27 - Microsoft Visual Studio
C:\WINDOWS\system32\cmd.exe
1
1
-1
Press any key to continue . . .
```

# Chapter 12: Working with Inheritance

## 1. C# program to demonstrate inheritance and also the concept of method hiding

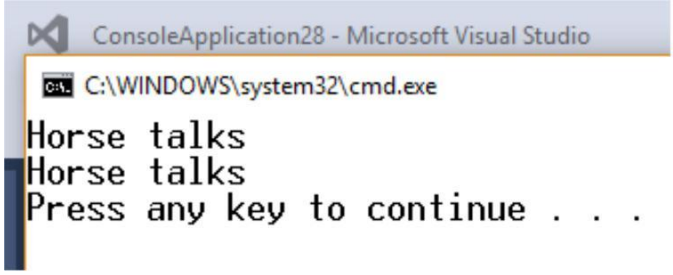
```
class Mammal
    {
        public void Talk()
        {
            Console.WriteLine("mammal talks");
        }
    }
    class Horse : Mammal
    {
        public new void Talk()   //Method hiding
        {
            Console.WriteLine("Horse talks");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
             Horse h = new Horse();
             h.Talk();
             Mammal m = h as Mammal;
             m.Talk();

        }
    }
```




## 2. C# program to demonstrate method overriding

```
class Mammal
    {
        public virtual void Talk()   //base class method virtual
        {
            Console.WriteLine("mammal talks");
        }
    }
    class Horse : Mammal
    {
        public override void Talk()   //derived class method override
        {
            Console.WriteLine("Horse talks");
        }
    }
```

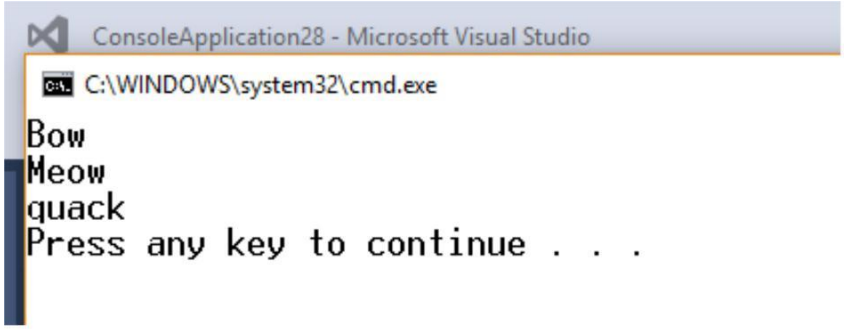
```
class Program
{
    static void Main(string[] args)
    {
         Horse h = new Horse();
         h.Talk();
         Mammal m = h as Mammal;

    }
}
```




## 3. C# program to demonstrate Dynamic method dispatch

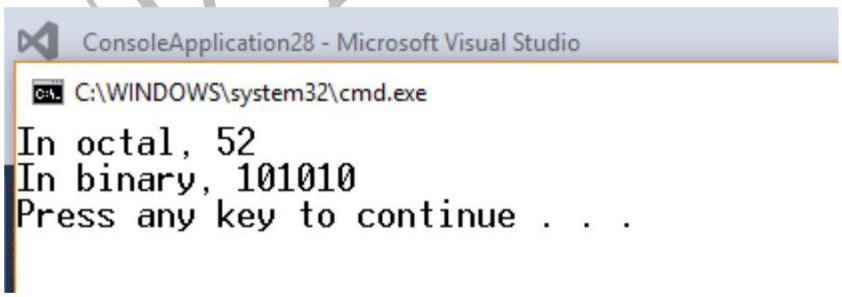
```
class Animal
    {
        public virtual void speak()
        {
            Console.WriteLine();
        }
    }
    class Dog:Animal
    {
        public override void speak()
        {
            Console.WriteLine("Bow");
        }
    }
    class Cat : Animal
    {
        public override void speak()
        {
            Console.WriteLine("Meow");
        }
    }
    class Duck : Animal
    {
        public override void speak()
        {
            Console.WriteLine("quack");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {

            Animal a = new Dog();
            a.speak();
            a = new Cat();
            a.speak();
            a = new Duck();
            a.speak();
```

```
        }
    }
```




## 4. Demonstrating extension methods – Write a C# extension method to convert a number from decimal to any base

```
static class Util
    {
        public static int ConvertToAny(this int dec,int b)
        {
            int rev = 0,i=0;
            while(dec!=0)
            {
                rev = rev + (dec % b)*(int)Math.Pow(10, i); ;
                dec = dec / b;
                i++;
            }
            return rev;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            int num = 42;
            Console.WriteLine($"In octal, {num.ConvertToAny(8)}");
            Console.WriteLine($"In binary, {num.ConvertToAny(2)}");

        }
    }
```



# Chapter 13: Creating Interfaces and defining Abstract classes

## 1. C# program to demonstrate interfaces:

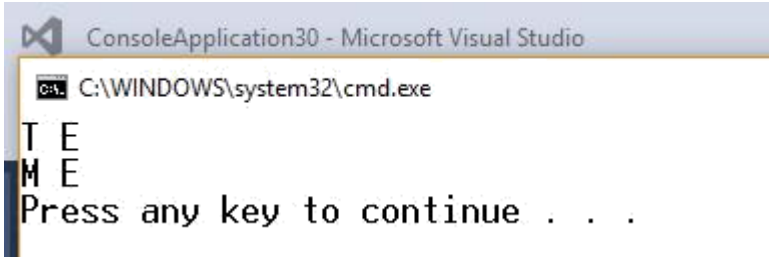
```
interface IAnimal
    {

       int number_of_legs();

    }
    public class Mammal
    {
        public void breathe()
        {
            Console.WriteLine("Mammal breathes");
        }
        public void speak()
        {
            Console.WriteLine("Animal speaks");
        }
    }
    class Horse : Mammal,IAnimal
    {
        int hooves;
        public new void speak()
        {
            Console.WriteLine("Horse speaks");
        }
        public int number_of_legs()
        {
            return 4;
        }

    }

    class Whale : Mammal,IAnimal
    {
        int flippers;
        int fluke;
        public int number_of_legs()
        {
            return 2;
        }
    }
    class Program:Mammal
    {

        static void Main(string[] args)
        {
            Horse h = new Horse();
            Mammal m = h;
            IAnimal i = h;
            Console.WriteLine(i.number_of_legs());
            i = new Whale();
            Console.WriteLine(i.number_of_legs());

            m.speak();
        }
```

```
ConsoleApplication29 - Microsoft Visual Studio
C:\WINDOWS\system32\cmd.exe
4
2
Animal speaks
Press any key to continue . . .
```

}

**2. C# program to demonstrate Explicit Interface Implementation:**

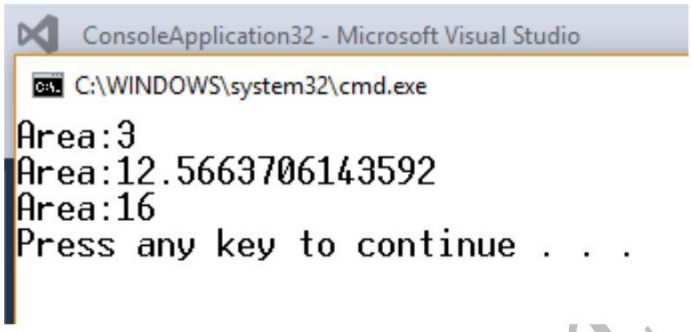
```
interface ITerrestrialAnimal
    {
        void eat();
        void swim();

    }
    interface IMarineAnimal
    {
        void eat();
        void swim();

    }
    class Crocodile:ITerrestrialAnimal,IMarineAnimal
    {
        void ITerrestrialAnimal.eat()
        {
            Console.WriteLine("T E");
        }
        void ITerrestrialAnimal.swim()
        {
            Console.WriteLine("T S");
        }
        void IMarineAnimal.eat()
        {
            Console.WriteLine("M E");
        }
        void IMarineAnimal.swim()
        {
            Console.WriteLine("M S");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Crocodile c = new Crocodile();
            ITerrestrialAnimal i = c;
            i.eat();
            IMarineAnimal m = c;
            m.eat();

        }
    }
```




## 3. C# program to illustrate Abstract classes

```
abstract class Shape    //Cant create objects
     {
         public abstract double area();  //abstract methods-enforce overriding
         public void disp()
         {
             Console.WriteLine("Shape");
         }
     }
     class Circle:Shape
     {
         int radius;
         public Circle(int radius)
         {
             this.radius = radius;
         }
         public override double area()
         {
             return Math.PI * radius * radius;
         }

     }

     class Square : Shape
     {
         int side;
         public Square(int side)
         {
             this.side = side;
         }
         public override double area()
         {
             return side*side;
         }

     }

     class Triangle : Shape
     {
         int length,breadth;
         public Triangle(int length, int breadth)
         {
             this.length = length;
             this.breadth = breadth;
         }
         public override double area()
         {
             return 0.5*length * breadth;
         }
```

```
    }


    class Program
    {

        static void Main(string[] args)
        {
            Triangle t = new Triangle(2, 3);
            Console.WriteLine($"Area:{t.area()}");
            Circle c = new Circle(2);;
            Console.WriteLine($"Area:{c.area()}");
            Square s = new Square(4);
            Console.WriteLine($"Area:{s.area()}");
        }
    }
```




## 4. C# program to illustrate sealed classes

```
class S
    {
        public virtual void m()
        {
            Console.WriteLine("S");
        }
    }
    class NS:S
    {
        public sealed override void m()
        {
            Console.WriteLine("NS");
        }
    }
    class NSS:NS
    {
        public override void m()   //Not possible as sealed methods
        {
            Console.WriteLine("S");
        }
    }
```

## Noe:

## 1. sealed methods end overriding, Sealed class end inheritance

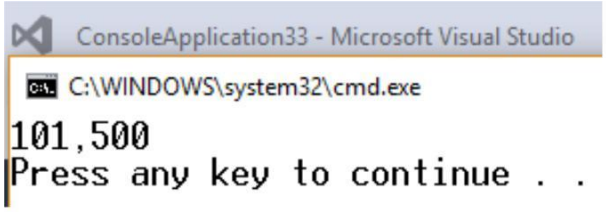
**2. interfaces give method prototype, virtual gives first implementation, override gives subsequent implementation and sealed ends the implementation**

## Chapter 15: Implementing properties to access fields:

## 1. C# code to implement encapsulation using get and set methods

```
struct Screen
    {
        int x, y;
        public Screen(int x, int y)
        {
            if (x < 0 || x > 800)
                throw new ArgumentOutOfRangeException("X");
            if (y < 0 || y > 800)
                throw new
            ArgumentOutOfRangeException("Y"); this.x = x;
            this.y = y;
        }
        public int getX()
        {
            return x;
        }
        public void setX(int x)
        {
            if (x < 0 || x > 800)
                throw new
            ArgumentOutOfRangeException("X"); this.x = x;
        }
        public int getY()
        {
            return y;
        }
        public void setY(int y)
        {
            if (y < 0 || y > 800)
                throw new
            ArgumentOutOfRangeException("Y"); this.y = y;
        }
        public void display()
        {
            Console.WriteLine($"{x},{y}");
        }

    }

    class Program
    {
        static void Main(string[] args)
        {
            Screen s = new Screen(100, 500);
            s.setX(s.getX() + 1);
            s.display();
```

```
        }
    }
```




## 2. C# code to implement previous example with properties

```
struct Screen
    {
        int _x, _y;
        public Screen(int x, int y)
        {
            if (x < 0 || x > 800)
                throw new ArgumentOutOfRangeException("X");
            if (y < 0 || y > 800)
                throw new
            ArgumentOutOfRangeException("Y"); this._x = x;
            this._y = y;
        }
        public int X
            {
                get { return _x; }
                set {
                if (value < 0 || value > 800)
                    throw new ArgumentOutOfRangeException("X");
                  _x = value;
            }
        }

        public int Y
        {
            get { return _y; }
            set
            {
                if (value < 0 || value > 800)
                    throw new
                ArgumentOutOfRangeException("Y"); _y = value;
            }
        }


    }

    class Program
    {
        static void Main(string[] args)
        {
            Screen s = new Screen(100, 500);
            s.X++;
            Console.WriteLine($"X={s.X}, Y={s.Y}");
        }
    }
```

```
ConsoleApplication34 - Microsoft Visual Studio
C:\WINDOWS\system32\cmd.exe
X=101, Y=500
Press any key to continue . . .
```

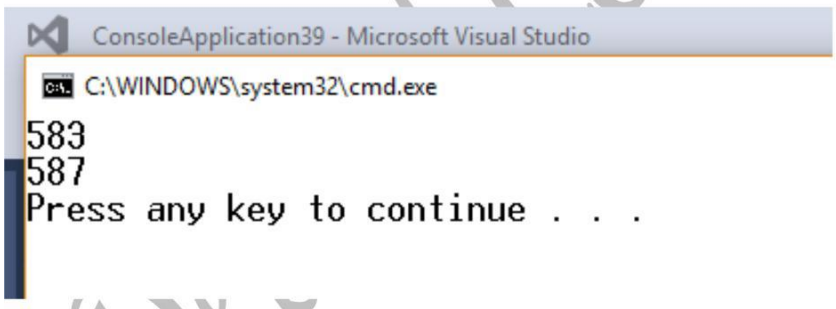
**3. C# program to demonstrate automatic property code generation and Object Initialization syntax**

```
class Triangle
    {
        int _s1=5;
        int _s2=5;
        int _s3=5;
        public int S1
        {
            get;
            set;
        }
        public int S2
        {
            get;
            set;
        }
        public int S3
        {
            get;
            set;
        }

    }
    class Program
    {
        static void Main(string[] args)
        {
            Triangle t = new Triangle() { S1 = 1 }; Triangle t1 = new
            Triangle() { S1 = 2,S3=3 };
            Console.WriteLine($"S1={t.S1},S2={t.S2},S3={t.S3}");
            Console.WriteLine($"S1={t1.S1},S2={t1.S2},S3={t1.S3}");
        }
    }
```

```
ConsoleApplication37 - Microsoft Visual Studio
C:\WINDOWS\system32\cmd.exe
S1=1,S2=0,S3=0
S1=2,S2=0,S3=3
Press any key to continue . . .
```

**4. C# program to demonstrate object ID's that are immutable (cant be changed further)**

```
class Triangle
    {
        int _s1=5;
        int _s2=5;
        int _s3=5;
        public int S1
        {
            get;
            set;
        }
        public int S2
        {
            get;
            set;
        }
        public int S3
        {
            get;
            set;
        }

    }
    class Program
    {
        static void Main(string[] args)
        {
            Triangle t = new Triangle() { S1 = 1 }; Triangle t1 = new
            Triangle() { S1 = 2,S3=3 };
            Console.WriteLine($"S1={t.S1},S2={t.S2},S3={t.S3}");
            Console.WriteLine($"S1={t1.S1},S2={t1.S2},S3={t1.S3}");
        }
    }
```



# Chapter 16: Using Indexers:

1. **C# program to demonstrate the use of indexers for accessing bits using array (ON/OFF and check status)**

```
struct Bits
    {
        int bits;
        public Bits(int bits)
        {
            this.bits = bits;
        }
        public bool this [ int index]
        {
            get
            {
                return ((bits & (1<<index))!=0);
            }
            set
            {
                if (value)
                    bits |= (1 << index);
                else
                    bits &= ~(1 << index);
            }
        }

    }
    class Program
    {
        static void Main(string[] args)
        {
            int num = 33;
            Bits b = new Bits(num);
            Console.WriteLine("Before, b[1]:" + b[1]);
            b[1] = true;
            Console.WriteLine("After, b[1]:" + b[1]);
        }
    }
```

C:\Windows\system32\cmd.exe

```
Before, b[1]:False
After, b[1]:True
Press any key to continue . . .
```

2. **C# program to compare arrays, properties and indexers**

**//When arrays are used as properties, they get updated when properties are updated. This is because arrays are passed by reference**

```
struct Wrapper
    {
        int[] data;

        public int[] Data
        {
            get;
            set;
```

```
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Wrapper w = new Wrapper();
            w.Data =new int[]{ 1,2,3};
            int[] md = w.Data;
            md[0]++;
            md[1]++;
            Console.WriteLine("property values");
            foreach(int d in w.Data)
            {
                Console.WriteLine(d);
            }
            Console.WriteLine("Array values");
            foreach (int e in md)
            {
                Console.WriteLine(e);
            }
        }
       }
```

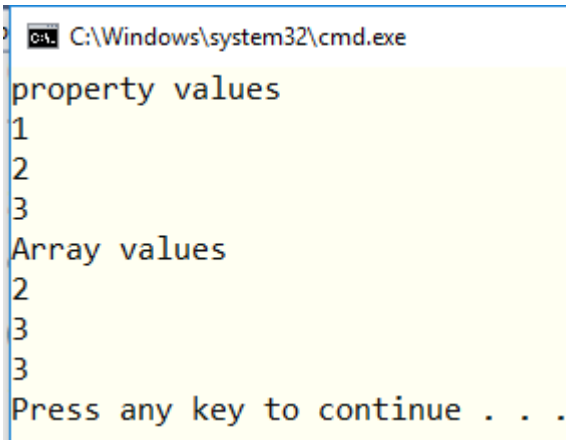
C:\Windows\system32\cmd.exe

```
property values
2
3
3
Array values
2
3
3
Press any key to continue . . .
```

## //Solution 1: Clone array and return to properties:

```
struct Wrapper
    {
        int[] data;

        public int[] Data
        {
            get
            {
                return this.data.Clone() as int[];

            }
            set
            {
                this.data = value.Clone() as int[];
            }

        }
    }
    class Program
```

```
    {
        static void Main(string[] args)
        {
            Wrapper w = new Wrapper();
            w.Data =new int[]{ 1,2,3};
            int[] md = w.Data;
            md[0]++;
            md[1]++;
            Console.WriteLine("property values");
            foreach(int d in w.Data)
            {
                Console.WriteLine(d);
            }
            Console.WriteLine("Array values");
            foreach (int e in md)
            {
                Console.WriteLine(e);
            }
        }
       }
```




## //Solution 2: Efficient using indexers no cloning required

```
struct Wrapper
    {
        int[] data;
        public Wrapper(int[] d)
        {
            data = d;
        }
        public int this[int i]
        {
            get
            {
                return this.data[i];
            }
            set
            {
                this.data[i] = value;
            }
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            int[] arr = new int[] { 1, 2, 3 };
            Wrapper w = new Wrapper(arr);
```

```
        int[] md = new int[2];
        md[0] = w[0];
        md[1] = w[1];
        md[0]++;
        md[1]++;
        Console.WriteLine("Before Array values");
        foreach (int d in arr)
        {
            Console.WriteLine(d);
        }
        Console.WriteLine("Afte Array values");
        foreach (int e in md)
        {
            Console.WriteLine(e);
        }
    }
   }
```

```
C:\Windows\system32\cmd.exe
Before Array values
1
2
3
Afte Array values
2
3
Press any key to continue . . .
```

# Chapter 17: Introducing Generics:

## 1. Write a C# program to create Queue of objects (Generalized class)

```
class Queue
 {
     Object[] data;
     int n;
     int rear = 0;
     int front = 0;
     public Queue(int size)
     {
         data = new Object[size];
     }
     public void Enqueue(Object ele)
     {
         if (n == data.Length)
             throw new Exception("queue overflow ");
         data[rear] = ele;
         rear = (rear + 1) % data.Length;
         n++;
     }
     public Object Dequeue()
     {
         if (n == 0)
```

```
                throw new Exception("queue underflow ");
            Object ele=data[front];
            front = (front + 1) % data.Length;
            n--;
            return ele;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Queue q = new Queue(5);
            try
            {
                for (int i = 0; i < 5; i++)
                    q.Enqueue(i);
                Console.WriteLine(q.Dequeue());
                Console.WriteLine(q.Dequeue());
            }
            catch(Exception e)
            {
                Console.WriteLine(e);
            }

           }
    }
```

C:\Windows\system32\cmd.exe

```
0
1
Press any key to continue . . .
```

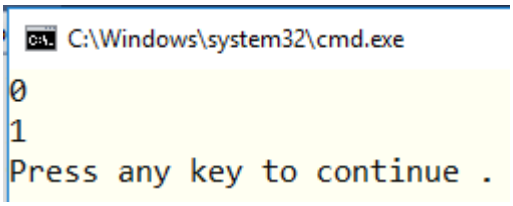
**Problems: (i) Explicit unboxing**

**(ii) It will create heterogeneous Queues as well**

## 2. C# program to create a Generic queue

```
class Queue<T>
    {
        T[] data;
        int n;
        int rear = 0;
        int front = 0;
        public Queue(int size)
        {
            data = new T[size];
        }
        public void Enqueue(T ele)
        {
            if (n == data.Length)
                throw new Exception("queue overflow ");
            data[rear] = ele;
            rear = (rear + 1) % data.Length;
            n++;
        }
        public T Dequeue()
        {
            if (n == 0)
                throw new Exception("queue underflow ");
            T ele=data[front];
```

```
            front = (front + 1) % data.Length;
            n--;
            return ele;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Queue<int> q = new Queue<int>(5);
            try
            {
                for (int i = 0; i < 5; i++)
                    q.Enqueue(i);
                Console.WriteLine(q.Dequeue());
                Console.WriteLine(q.Dequeue());
            }
            catch(Exception e)
            {
                Console.WriteLine(e);
            }

        }
       }
```




3. C# program to create a Generic Binary Search Tree with Inorder traversal

```
public class Tree<T> where T : IComparable<T>
    {
        public T data { get; set; }

        public Tree<T> left { get; set; }
        public Tree<T> right { get; set; }
        public Tree(T ele)
        {
            data = ele;
            left = null;
            right = null;
        }
        public void insert(T ele)
        {
            if (ele.CompareTo(data) < 0)
            {
                if (left != null)
                    left.insert(ele);
                else
                    left = new Tree<T>(ele);
            }
            else
            {
                if (right != null)
```

```
                right.insert(ele);
            else
                right = new Tree<T>(ele);
        }
    }
    public void inorder()
    {
        if (left != null)
        {
            left.inorder();
        }
        Console.WriteLine(data);
        if (right != null)
            right.inorder();
    }
}

    class Program
{
    static void Main(string[] args)
    {
            Tree<int> t = new Tree<int>(10);
            t.insert(5);
            t.insert(11);
            t.insert(12);
            t.insert(5);
            t.insert(-2);
            t.insert(0);
            t.inorder();
        }
       }
```

C:\Windows\system32\cmd.exe

```
-2
0
5
5
10
11
12
Press any key to continue . . .
```

4. C# program demonstrating generic swap method of any two things:

```
class Program
    {
        static void swap<T>(ref T a, ref T b)
        {
            T temp = a;
            a = b;
            b = temp;
        }
        static void Main(string[] args)
        {
            int p = 3, q = 2;
            Console.WriteLine($"Before:{p},{q} ");
            swap(ref p, ref q);
            Console.WriteLine($"After:{p},{q} ");
```

```
            string f = "hi";
            string g = "bye";
            Console.WriteLine($"Before:{f},{g} ");
            swap(ref f, ref g);
            Console.WriteLine($"After:{f},{g} ");

        }
    }
```

C:\Windows\system32\cmd.exe

```
Before:3,2
After:2,3
Before:hi,bye
After:bye,hi
Press any key to continue . . .
```

## 5. C# program that performs generic insert on Binary Search Generic Tree

```
public class Tree<T> where T : IComparable<T>
    {
        public T data { get; set; }

        public Tree<T> left { get; set; }
        public Tree<T> right { get; set; }
        public Tree(T ele)
        {
            data = ele;
            left = null;
            right = null;
        }
        public void insert(T ele)
        {
            if (ele.CompareTo(data) < 0)
            {
                if (left != null)
                    left.insert(ele);
                else
                    left = new Tree<T>(ele);
            }
            else
            {
                if (right != null)
                    right.insert(ele);
                else
                    right = new Tree<T>(ele);
            }
        }
        public void inorder()
        {
            if (left != null)
            {
                left.inorder();
            }
            Console.WriteLine(data);
            if (right != null)
                right.inorder();
        }
    }
```

```
    class Program
{
    static void insert<T>(T[] a,ref Tree<T> t) where T:IComparable<T>
    {
        foreach(T item in a)
        {
            if (t == null)
                t = new Tree<T>(item);
            else
                t.insert(item);
        }
    }
    static void Main(string[] args)
    {
        int[] a =new int[] { 1, 3, 0, -1, 2, 5 };

            Tree<int> t = null;
            insert(a, ref t);

            t.inorder();
        }
    }
```

C:\Windows\system32\cmd.exe

```
-1
0
1
2
3
5
Press any key to continue . .
```

# Chapter 18: Using Collections

## 1. C# program to illustrate Lists

```
struct Person
{
    public int ID { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
}
class Program
{
    static void Main(string[] args)
    {
        List<string> names = new List<string>();
        names.Add("Amar");
        names.Insert(0, "akbar");
        names.Add("Anthony");
        //names.RemoveAt(0);
        //names.Remove("akbar");
        Console.WriteLine("Before sorting");
        for (int i = 0; i < names.Count; i++)
            Console.WriteLine(names[i]);
```

```
            names.Sort();
            Console.WriteLine("After sorting");
            for (int i = 0; i < names.Count; i++)
                Console.WriteLine(names[i]);


        }
       }
```

C:\Windows\system32\cmd.exe

```
Before sorting
akbar
Amar
Anthony
After sorting
akbar
Amar
Anthony
Press any key to continue . . .
```

## 2. C# program to illustrate Linked List

```
struct Person
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
    }
    class Program
    {
        static void Main(string[] args)
        {

            LinkedList<int> marks = new LinkedList<int>();
            marks.AddFirst(24);
            marks.AddLast(29);
            LinkedListNode<int> node= marks.First;
            marks.AddAfter(node, 16);

 for (LinkedListNode<int> temp = marks.First; temp != null; temp = temp.Next)
                Console.Write(temp.Value + "->");

for (LinkedListNode<int> temp = marks.Last; temp != null; temp = temp.Previous)
                Console.Write(temp.Value + "->");

        }
  }
```

C:\Windows\system32\cmd.exe

```
24->16->29->29->16->24->Press any key to continue .
```

## 3. C# program to demonstrate Queues

```
struct Person
    {
```

```
        public int ID { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
    }
    class Program
    {
        static void Main(string[] args)
        {

            Queue<string> names = new Queue<string>();
            names.Enqueue("amar");
            names.Enqueue("akbar");
            names.Enqueue("birbal");

            for (int i = 0; i <=(names.Count+1); i++)
            {
                Console.WriteLine(names.Dequeue());

            }

        }
       }
```

C:\Windows\system32\cmd.exe

```
amar
akbar
birbal
Press any key to continue .
```

## 4. C# program to demonstrate Stacks

```
struct Person
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
    }
    class Program
    {
        static void Main(string[] args)
        {

             Stack<string> names = new Stack<string>();
            names.Push("amar");
            names.Push("akbar");
            names.Push("birbal");

            for (int i = 0; i <=(names.Count+1); i++)
            {
                Console.WriteLine(names.Pop());

            }
        }
       }
```

```
C:\Windows\system32\cmd.exe
birbal
akbar
amar
Press any key to continue . .
```

## 5. C# program to demonstrate Dictionary

```
struct Person
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
    }
    class Program
    {
        static void Main(string[] args)
            {
           Dictionary<string, int> student_marks = new Dictionary<string, int>();
            student_marks.Add("amar", 24);
            student_marks["akbar"] = 16;
            student_marks.Add("anthony", 29);

            foreach(KeyValuePair<string,int> k in student_marks)
            {
                Console.WriteLine(k.Key + "," + k.Value);
               }
      }
   }
```

```
C:\Windows\system32\cmd.exe
amar,24
akbar,16
anthony,29
Press any key to continue . . .
```

## 6. C# program to demonstrate SortedList

```
struct Person
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
    }
    class Program
    {
        static void Main(string[] args)
            {
           SortedList<string, int> student_marks = new SortedList<string, int>();
            student_marks.Add("amar", 24);
            student_marks["akbar"] = 16;
            student_marks.Add("anthony", 29);

            foreach(KeyValuePair<string,int> k in student_marks)
            {
                Console.WriteLine(k.Key + "," + k.Value);
               }
```

```
    }
  }
```

C:\Windows\system32\cmd.exe

```
akbar,16
amar,24
anthony,29
Press any key to continue .
```

## 7. C# program to demonstrate HashList

```
struct Person
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
    }
    class Program
    {
        static void Main(string[] args)
            {
            HashSet<string> aj_students =new HashSet<string>();
            aj_students.Add("ena");
            aj_students.Add("meena");
            aj_students.Add("sona");
            HashSet<string> dotnet_students = new HashSet<string>();
            dotnet_students.Add("ena");
           // dotnet_students.Add("meena");
            dotnet_students.Add("mona");
            // aj_students.IntersectWith(dotnet_students);
            if(dotnet_students.IsSubsetOf(aj_students))
            {
                Console.WriteLine("true");
            }
            else
            {
                Console.WriteLine("false");
            }
            Console.WriteLine("AJ students");
            foreach(string student in aj_students)
            {
                Console.WriteLine(student);

            }
            Console.WriteLine("Dotnet students");
            foreach (string student in dotnet_students)
            {
                Console.WriteLine(student);

               }
    }
    }
```

```
false
AJ students
ena
meena
sona
Dotnet students
ena
mona
Press any key to continue . . .
```

## 8. C# program to demonstrate Collection Initializers and Lambda expressions

```
struct Person
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
    }
    class Program
    {
        static void Main(string[] args)
           {
             List<Person> l = new List<Person>()
             {
                new Person(){ID = 1, Name = "Abc", Age = 25 },
                new Person(){ID = 1, Name = "Pqr", Age = 26 }
             };
           Person match=l.Find(p => { return p.ID == 1;});
           Console.WriteLine(match.ID+","+match.Name+","+match.Age);

        }
    }
```

```
C:\Windows\system32\cmd.exe
l,Abc,25
Press any key to continue .
```

# Chapter 19: Enumerating Collections

## 1. C# program to demonstrate Enumerating BST elements manually implementing an enumerator:

```
class Tree<T>:IEnumerable<T> where T : IComparable<T>
    {
        public T data { get; set; }
        public Tree<T> left { get; set; }
        public Tree<T> right { get; set; }
```

```
    public Tree(T data)
    {
        this.data = data;
        this.left = null;
        this.right = null;
    }

    public void insert(T newdata)
    {
        if (data.CompareTo(newdata) > 0)
        {
            if (this.left == null)
                this.left = new Tree<T>(newdata);
            else
                left.insert(newdata);
        }
        else
        {
            if (this.right == null)
                this.right = new Tree<T>(newdata);
            else
                right.insert(newdata);
        }
    }

    public void inorder()
    {
        if (left != null)
            left.inorder();
        Console.WriteLine(data);
        if (right != null)
            right.inorder();
    }

    public IEnumerator<T> GetEnumerator()
    {
         return new TreeEnum<T>(this);

    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        throw new NotImplementedException();
    }
}
class TreeEnum<T>:IEnumerator<T> where T:IComparable<T>
{
    Tree<T> c;
    Queue<T> q;
    T current;
    public TreeEnum(Tree<T> c)
    {
        this.c = c;
    }
    void populate(Queue<T> q,Tree<T> c)
    {
        if (c.left != null)
            populate(q, c.left);
        q.Enqueue(c.data);
        if (c.right != null)
            populate(q, c.right);
    }
```

```
    public void Dispose()
    {

    }

    public bool MoveNext()
    {
        if (q == null)
        {
            q = new Queue<T>();
            populate(q, c);
        }
        if (q.Count > 0)
        {
            current = q.Dequeue();
            return true;
        }
        return false;
    }

    public void Reset()
    {
        throw new NotImplementedException();
    }

    T IEnumerator<T>.Current
    {
        get
        {
            return current;
        }
    }

    public object Current => throw new NotImplementedException();
}

class Program
{
    static void Main(string[] args)
    {
        Tree<int> t = new Tree<int>(10);
        t.insert(4);
        t.insert(12);
        t.insert(-1);
        //t.inorder();
        foreach(int i in t)
        {
            Console.WriteLine(i);
        }
    }
   }
```

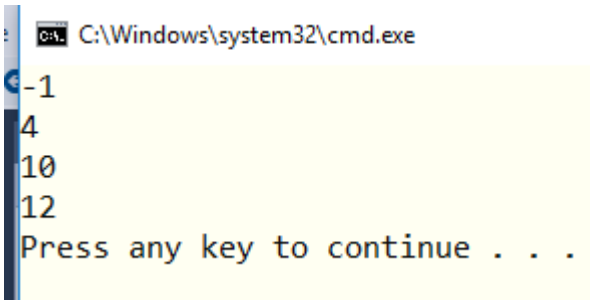
C:\Windows\system32\cmd.exe

```
-1
4
10
12
Press any key to continue . . .
```

## 2. C# program to implement an enumerator by using an iterator

```
class Tree<T>:IEnumerable<T> where T : IComparable<T>
    {
        public T data { get; set; }
        public Tree<T> left { get; set; }
        public Tree<T> right { get; set; }

        public Tree(T data)
        {
            this.data = data;
            this.left = null;
            this.right = null;
        }

        public void insert(T newdata)
        {
            if (data.CompareTo(newdata) > 0)
            {
                if (this.left == null)
                    this.left = new Tree<T>(newdata);
                else
                    left.insert(newdata);
            }
            else
            {
                if (this.right == null)
                    this.right = new Tree<T>(newdata);
                else
                    right.insert(newdata);
            }
        }

        public void inorder()
        {
            if (left != null)
                left.inorder();
            Console.WriteLine(data);
            if (right != null)
                right.inorder();
        }

        public IEnumerator<T> GetEnumerator()
        {
            if(left!=null)
            {
                foreach (T item in left)
                    yield return item;
            }
            yield return data;
            if (right != null)
            {
                foreach (T item in right)
                    yield return item;
            }

        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            throw new NotImplementedException();
```

```
    }
}


class Program
{
    static void Main(string[] args)
    {
        Tree<int> t = new Tree<int>(10);
        t.insert(4);
        t.insert(12);
        t.insert(-1);
        //t.inorder();
        foreach(int i in t)
        {
            Console.WriteLine(i);
        }
    }
   }
```




# Chapter 20: Decoupling application logic and handling events

## 1. C# program to demonstrate builtin delegates

```
struct Person
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }

    }
class Program
{
    static void Main(string[] args)
        {
           List<Person> l = new List<Person>()
           {
             {new Person(){ID=1,Name="Abc",Age=22 } },
             {new Person(){ID=2,Name="Bbc",Age=32 } },
             {new Person(){ID=3,Name="Cbc",Age=26 } }
         };
         double avg = l.Average(p => p.Age);
         Console.WriteLine(avg);
    }
   }
```

```
C:\Windows\system32\cmd.exe
26.6666666666667
Press any key to continue . . .
```

## 2. C# program to automate a factory scenario without Delegates

```
struct Person
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }

    }
    class ShapingMachine
    {
        public void StopShaping()
        {
            Console.WriteLine("Stop Shaping");
        }
    }
    class WeldingMachine
    {
        public void StopWelding()
        {
            Console.WriteLine("Stop Welding");
        }

    }
    class PaintingMachine
    {
        public void StopPainting()
        {
            Console.WriteLine("Stop Painting");
        }
    }
    class Controller
    {
        ShapingMachine s;
        WeldingMachine w;
        PaintingMachine p;
        public Controller(ShapingMachine s, WeldingMachine w,PaintingMachine p)
        {
            this.s = s;
            this.w = w;
            this.p = p;

        }
         public void ShutDown()
         {
            s.StopShaping();
            w.StopWelding();
            p.StopPainting();
         }
    }
    class Program
    {
        static void Main(string[] args)
        {
            ShapingMachine s = new ShapingMachine();
            WeldingMachine w = new WeldingMachine();
```

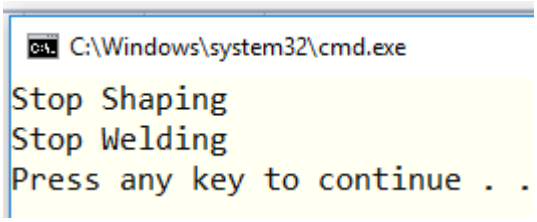
```
            PaintingMachine p = new PaintingMachine();

        }
    }
```

```
C:\Windows\system32\cmd.exe
Stop Shaping
Stop Welding
Stop Painting
Press any key to continue . . .
```

## 3. C# program to automate a factory scenario with Delegates

```
struct Person
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }

    }
    class ShapingMachine
    {
        public void StopShaping()
        {
            Console.WriteLine("Stop Shaping");
        }

    }
    class WeldingMachine
    {
        public void StopWelding()
        {
            Console.WriteLine("Stop Welding");
        }

    }
    class PaintingMachine
    {
        public void StopPainting()
        {
            Console.WriteLine("Stop Painting");
        }
    }
    class Controller
    {
        public delegate void ShutDownDelegate();
        public ShutDownDelegate s{get;set;}


       public void Add(ShutDownDelegate m)
        {
            s += m;
        }
        public void Remove(ShutDownDelegate m)
        {
            s -= m;
        }
         public void ShutDown()
         {
            s();
```

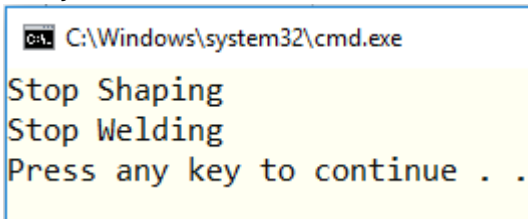
```
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            ShapingMachine s = new ShapingMachine();
            WeldingMachine w = new WeldingMachine();
            PaintingMachine p = new PaintingMachine();


            Controller c = new Controller();
            c.s+=(s.StopShaping);
            c.s+=(w.StopWelding);
            c.s+=(p.StopPainting);
            c.s-=(p.StopPainting);
            c.s();

        }
    }
```




## 4. C# program to automate a factory scenario with Delegates and Adapters

```
struct Person
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }

    }
    class ShapingMachine
    {
        public void StopShaping(int s)
        {
            Console.WriteLine("Stop Shaping");
        }
        public void FinishShaping()
        {
            StopShaping(0);
        }
    }
    class WeldingMachine
    {
        public void StopWelding()
        {
            Console.WriteLine("Stop Welding");
        }

    }
    class PaintingMachine
```

```
    {
        public void StopPainting()
        {
            Console.WriteLine("Stop Painting");
        }
    }
    class Controller
    {
        public delegate void ShutDownDelegate();
        public ShutDownDelegate s{get;set;}


       public void Add(ShutDownDelegate m)
        {
            s += m;
        }
        public void Remove(ShutDownDelegate m)
        {
            s -= m;
        }
         public void ShutDown()
         {
            s();

         }
    }

    class Program
    {
        static void Main(string[] args)
        {
            ShapingMachine s = new ShapingMachine();
            WeldingMachine w = new WeldingMachine();
            PaintingMachine p = new PaintingMachine();

            Controller c = new Controller();
            c.s += (() => s.StopShaping(0));
            c.s+=(w.StopWelding);
            c.s+=(p.StopPainting);
            c.s-=(p.StopPainting);
            c.s();
        }
           }
```




## 5. C# program to automate a factory scenario with Events

```
struct Person
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }

    }
    class ShapingMachine
    {
```

```
    public void StopShaping(int s)
    {
        Console.WriteLine("Stop Shaping");
    }
    public void FinishShaping()
    {
        StopShaping(0);
    }
}
class WeldingMachine
{
    public void StopWelding()
    {
        Console.WriteLine("Stop Welding");
    }

}
class PaintingMachine
{
    public void StopPainting()
    {
        Console.WriteLine("Stop Painting");
    }
}
class TemperatureMonitor
{
    public delegate void stopMachinery();
    public event stopMachinery OverHeat;
    public void notify()
    {
        if(OverHeat!=null)
             OverHeat();
    }

}

class Program
{
    static void Main(string[] args)
    {
        ShapingMachine s = new ShapingMachine();
        WeldingMachine w = new WeldingMachine();
        PaintingMachine p = new PaintingMachine();
        TemperatureMonitor t = new TemperatureMonitor();
        t.OverHeat+= (() => s.StopShaping(0));
        t.OverHeat += w.StopWelding;
        t.OverHeat += p.StopPainting;
        t.notify();

    }
   }
```

C:\Windows\system32\cmd.exe

```
Stop Shaping
Stop Welding
Stop Painting
Press any key to continue . . .
```

# Chapter 21: Querying in-memory data by using query expressions

## 1. C# code to demonstrate Select in LINQ

```
class Customer
    {
        public int ID { get; set; }
        public string firstName { get; set; }
        public string lastName { get; set; }
        public string companyName { get; set; }
    }
    class Location
    {
        public string companyName { get; set; }
        public string City { get; set; }
        public string Country { get; set; }

    }
    class Program
    {
        static void Main(string[] args)
        {
            var customers = new[]
            {
                   new {ID=1,firstName="John",lastName="A",companyName="TCS"} ,
                  new {ID=2,firstName="Johny",lastName="B",companyName="Semantec"}
,
                   new {ID=3,firstName="Janardhan",lastName="C",companyName="TCS"}
,
                   new
{ID=4,firstName="Jhansi",lastName="D",companyName="Semantec"}

            };
            var Location = new[]
            {
                new {companyName="TCS",City="Bangalore",Country="India"},
                new {companyName="Semantec",City="California",Country="US"}


            };
             var fnames = customers.Select(cust => new
             {
                 firstName = cust.firstName,
                 lastName = cust.lastName
             });
                 foreach(var f in fnames)
             {
                 Console.WriteLine(f.firstName + "," + f.lastName);
             }



        }
    }
```

```
C:\Windows\system32\cmd.exe
John,A
Johny,B
Janardhan,C
Jhansi,D
Press any key to continue . . .
```

## 2. C# program to illustrate Where, (filtering), Ordering, Grouping and Aggregating Data

```
class Customer
    {
        public int ID { get; set; }
        public string firstName { get; set; }
        public string lastName { get; set; }
        public string companyName { get; set; }
    }
    class Location
    {
        public string companyName { get; set; }
        public string City { get; set; }
        public string Country { get; set; }

    }

    class Program
    {
        static void Main(string[] args)
        {
            var customers = new[]
            {
                   new {ID=1,firstName="John",lastName="A",companyName="TCS"} ,
                  new {ID=2,firstName="Johny",lastName="B",companyName="Semantec"}
,
                   new {ID=3,firstName="Janardhan",lastName="C",companyName="TCS"}
,
                   new
{ID=4,firstName="Jhansi",lastName="D",companyName="Semantec"}

            };
            var Location = new[]
            {
                new {companyName="TCS",City="Bangalore",Country="India"},
                new {companyName="Semantec",City="California",Country="US"}


            };
             IEnumerable<string> fnames =
customers.Where(cust=>cust.companyName.Equals("Semantec")).Select(cust=>cust.first
Name);
             foreach(string f in fnames)
             {
                 Console.WriteLine(f);
             }
             IEnumerable<string> fnnames = customers.OrderByDescending(cust =>
cust.companyName).Select(cust => cust.firstName);
             foreach (string f in fnnames)
```

```
            {
                Console.WriteLine(f);
            }

            var res = customers.GroupBy(c => c.companyName);
            foreach(var r in res)
            {
                Console.WriteLine(r.Key + "," + r.Count());
            }




        }
    }
```

C:\Windows\system32\cmd.exe

```
Johny
Jhansi
John
Janardhan
Johny
Jhansi
TCS,2
Semantec,2
Press any key to continue . . .
```

## 3. C# program to demonstrate joining of 2 memory tables

```
class Customer
    {
        public int ID { get; set; }
        public string firstName { get; set; }
        public string lastName { get; set; }
        public string companyName { get; set; }
    }
    class Location
    {
        public string companyName { get; set; }
        public string City { get; set; }
        public string Country { get; set; }

    }
    class Program
    {
        static void Main(string[] args)
        {
            var customers = new[]
            {
                   new {ID=1,firstName="John",lastName="A",companyName="TCS"} ,
                  new {ID=2,firstName="Johny",lastName="B",companyName="Semantec"} ,
                   new {ID=3,firstName="Janardhan",lastName="C",companyName="TCS"} ,
                   new {ID=4,firstName="Jhansi",lastName="D",companyName="Semantec"}

            };
            var Location = new[]
```

```
            {
                new {companyName="TCS",City="Bangalore",Country="India"},
                new {companyName="Semantec",City="California",Country="US"}


            };

            var jr = customers.Join(Location, c => c.companyName, l => l.companyName,
(c, l) => new { c.firstName, l.City });
            foreach (var j in jr)
                Console.WriteLine(j.firstName + "," + j.City);

        }
    }
```

C:\Windows\system32\cmd.exe

```
John,Bangalore
Johny,California
Janardhan,Bangalore
Jhansi,California
Press any key to continue . .
```

## 4. C# program to demonstrate Query Operators

```
class Customer
    {
        public int ID { get; set; }
        public string firstName { get; set; }
        public string lastName { get; set; }
        public string companyName { get; set; }
    }
    class Location
    {
        public string companyName { get; set; }
        public string City { get; set; }
        public string Country { get; set; }

    }
    class Program
    {
        static void Main(string[] args)
        {
            var customers = new[]
            {
                   new {ID=1,firstName="John",lastName="A",companyName="TCS"} ,
                  new {ID=2,firstName="Johny",lastName="B",companyName="Semantec"}
,
                   new {ID=3,firstName="Janardhan",lastName="C",companyName="TCS"}
,
                   new
{ID=4,firstName="Jhansi",lastName="D",companyName="Semantec"}

            };
            var Location = new[]
            {
                new {companyName="TCS",City="Bangalore",Country="India"},
                new {companyName="Semantec",City="California",Country="US"}


            };
```

```
            //Select and Where operators
            var fnames = from c in customers
                         where c.companyName.Equals("TCS")
                         select c.firstName;

            foreach (var f in fnames)
            {
                Console.WriteLine(f);
            }

            //Grouping operator
            var fnnames = from c in customers
                          group c by c.companyName;

            foreach (var f in fnnames)
            {
                Console.WriteLine(f.Key + "," + f.Count());
            }

            //Join
            var jr = from c in customers
                     join j in Location
                     on c.companyName equals j.companyName
                     select new { c.firstName, j.City };
            foreach (var j in jr)
                Console.WriteLine(j.firstName + "," + j.City);



        }
    }
```

C:\Windows\system32\cmd.exe

```
John
Janardhan
TCS,2
Semantec,2
John,Bangalore
Johny,California
Janardhan,Bangalore
Jhansi,California
Press any key to continue . . .
```

# 5. C# program to create Employee Tree and querying using LINQ

```
public class Tree<T>:IEnumerable<T> where T : IComparable<T>
    {
        public T data { get; set; }
```

```
        public Tree<T> left { get; set; }
        public Tree<T> right { get; set; }
        public Tree(T ele)
        {
            data = ele;
            left = null;
            right = null;
        }
        public void insert(T ele)
        {
            if (ele.CompareTo(data) < 0)
            {
                if (left != null)
                    left.insert(ele);
                else
                    left = new Tree<T>(ele);
            }
            else
            {
                if (right != null)
                    right.insert(ele);
                else
                    right = new Tree<T>(ele);
            }
        }
        public void inorder()
        {
            if (left != null)
            {
                left.inorder();
            }
            Console.WriteLine(data);
            if (right != null)
                right.inorder();
        }

        public IEnumerator<T> GetEnumerator()
        {
            if (left != null)
            {
                foreach (T l in left)
                    yield return l;
            }
            yield return data;
            if (right != null)
            {
                foreach (T l in right)
                    yield return l;
            }
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            throw new NotImplementedException();
        }
    }
    class Emp:IComparable<Emp>
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public string Desg { get; set; }

        int IComparable<Emp>.CompareTo(Emp other)
```

```
        {
            if (ID > other.ID)
                return 1;
            else if (ID < other.ID)
                return -1;
            return 0;
        }
    }


    class Program
    {
        static void Main(string[] args)
        {
            Tree<Emp> t = new Tree<Emp>(
                new Emp() { ID=101,Name="E1",Desg="AP"}
                );
            t.insert(new Emp() { ID = 102, Name = "E2", Desg = "P" });
            t.insert(new Emp() { ID = 103, Name = "E3", Desg = "AP" });
            t.insert(new Emp() { ID = 104, Name = "E4", Desg = "P" });

            //var eid = t.Select(e => e.ID);
            var eid = from e in t.ToList()    //for Immediate evaluation otherwise
Deferred Evaluaiton
                      select e.ID;
            t.insert(new Emp() { ID = 105, Name = "E5", Desg = "P" });
            foreach (var e in eid)
                Console.WriteLine(e);

            int cnt = t.Select(e => e.ID).Count();
           int cnt1= (from e in t
                    select e.ID).Count();
            Console.WriteLine(cnt+","+cnt1);

            int dcnt = t.Select(e => e.Desg).Distinct().Count();
            Console.WriteLine(dcnt);

            var grp = t.GroupBy(e => e.Desg);
            foreach (var e in grp)
                Console.WriteLine(e.Key+","+e.Count());

        }
      }
```

C:\Windows\system32\cmd.exe

```
101
102
103
104
5,5
2
AP,2
P,3
Press any key to continue . . .
```

2

6.

# Chapter 22: Operator Overloading

## 1. C# program to demonstrate overloading of +, symmetric operators, operator pairs and implicit/explicit conversion operators

```
struct Hour
    {
        int val;
        public Hour(int val)
        {
            this.val = val;
        }
        public static Hour operator +(Hour lhs, Hour rhs)
        {
            return new Hour(lhs.val + rhs.val);
        }
        public static Hour operator +(Hour lhs, int v)
        {
            return new Hour(lhs.val + v);
        }
        public static Hour operator +(int v, Hour rhs)
        {
            return new Hour(rhs.val + v);
        }
        public static Hour operator ++(Hour rhs)
        {
            return new Hour(rhs.val + 1);
        }
        public static bool operator ==(Hour lhs,Hour rhs)
        {
            return (lhs.val==rhs.val);
        }
        public static bool operator !=(Hour lhs, Hour rhs)
        {
            return (lhs.val == rhs.val);
        }
        public static explicit operator int (Hour lhs)
        {
            return lhs.val;
        }
        public override string ToString()
        {
            return "Hours is:"+val;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Hour h1=new Hour(3);
            Hour h2 = new Hour(4);
            Hour res = h1 + h2;
            Hour res1=++h1;
            res1 += 3;
            int x = (int)res1;


            Console.WriteLine(res1);
```

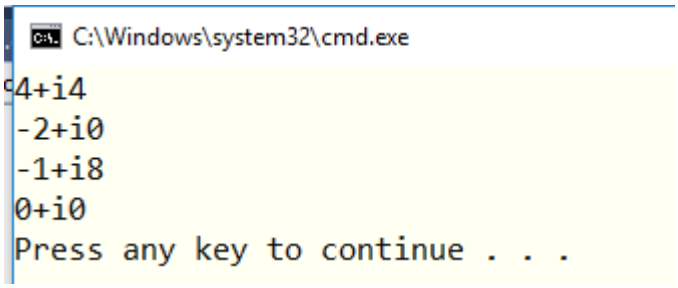
```
        }
    }
```

```
C:\Windows\system32\cmd.exe
Hours is:7
Press any key to continue . .
```

## 2. C# code to demonstrate overloading operators for Complex numbers

```
class Complex
    {
        int real;
        int imag;
        public Complex(int real,int imag)
        {
            this.real = real;
            this.imag = imag;

        }
        public static Complex operator +(Complex c1,Complex c2)
        {
            return new Complex(c1.real + c2.real, c1.imag + c2.imag);
        }
        public static Complex operator -(Complex c1, Complex c2)
        {
            return new Complex(c1.real - c2.real, c1.imag - c2.imag);
        }
        public static Complex operator *(Complex c1, Complex c2)
        {
            return new Complex(c1.real*c2.real-
c1.imag*c2.imag,c1.real*c2.imag+c1.imag*c2.real);
        }
        public static Complex operator /(Complex c1, Complex c2)
        {
            int r = (c1.real*c2.real+c1.imag*c2.imag) /
(c2.real*c2.real+c2.imag*c2.imag);
            int i = (c1.imag*c2.real-c1.real*c2.imag) / (c2.real * c2.real +
c2.imag * c2.imag);
            return new Complex(r,i);
        }
        public override string ToString()
        {
            return $"{real}+i{imag}";
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Complex c1 = new Complex(1, 2);
            Complex c2 = new Complex(3, 2);
            Complex c3 = c1 + c2;
            Complex c4 = c1 - c2;
            Complex c5 = c1 * c2;
            Complex c6= c1 / c2;
            Console.WriteLine(c3);
            Console.WriteLine(c4);
```

```
            Console.WriteLine(c5);
            Console.WriteLine(c6);


        }
    }
```




Links:

**For Module-1: prezi presentation (done in the class) at:**

http://prezi.com/5jl1eipajd4h/?utm_campaign=share&utm_medium=copy

**For Module-2: prezi presentation (done in the class) at:**

http://prezi.com/htt7_gmd_h18/?utm_campaign=share&utm_medium=copy

**For Module-3: prezi presentation (done in the class) at:**

http://prezi.com/r15jnr1cyh_v/?utm_campaign=share&utm_medium=copy

**For Module-4: prezi presentation (done in the class) at:**

http://prezi.com/n2xl6zuw6qer/?utm_campaign=share&utm_medium=copy

**For Module-5: prezi presentation (done in the class) at:**

http://prezi.com/gmk5lwcwey-b/?utm_campaign=share&utm_medium=copy

**********All the Best **********