```
class Employee : IComparable<Employee>
    {
        int eid;
        string enm;
        int esalary;

        public Employee(int eid, string enm, int esalary)
        {
            this.Eid = eid;
            this.Enm = enm;
            this.Esalary = esalary;
        }

        public int Eid { get => eid; set => eid = value; }
        public string Enm { get => enm; set => enm = value; }
        public int Esalary { get => esalary; set => esalary = value; }

        public int CompareTo(Employee? other)
        {
            if (this.esalary == other.esalary)
                return this.enm.CompareTo(other.enm);
            else
                return this.esalary.CompareTo(other.esalary);
        }
        public override string ToString()
        {
            return $"Id: {eid} Name: {enm} Salary: {esalary}";
        }
    }

    class Demo
    {
        static void Main(string[] args)
        {
            SortedList<Employee, string> es = new
SortedList<Employee, string>();

            es.Add(new Employee(101, "Jayray", 90000), "Develper");
            es.Add(new Employee(102, "Abhishek", 70000), "Tester");
            es.Add(new Employee(103, "Onkar", 85000), "HR");
            es.Add(new Employee(104, "Siddhant", 123000), "Develper");
            es.Add(new Employee(105, "Vaibhav", 80000), "Accontant");

            foreach (KeyValuePair<Employee, string> e1 in es)
                Console.WriteLine(e1.Key + "," + e1.Value);
        }
    }
```

Linked list

```
class LinkedListDemo
    {
        static void Main(string[] args)
        {
            //Linked list is used if you want to insert in between and delete
in between is faster.
            //link = searching or retrival of data is faster.

            LinkedList<int> ll = new LinkedList<int>();
            ll.AddLast(90);
            ll.AddLast(45);
            ll.AddLast(67);
            ll.AddFirst(20);

            //Console.WriteLine(ll.Remove(45));

            foreach(int d in ll)
            {
                Console.WriteLine(d);
            }

            //20,90,45,67
            //Node = It is a variable which can store the data and address of
the next element in it.

            LinkedListNode<int> n1 = ll.Find(90);
            ll.AddAfter(n1, new LinkedListNode<int>(900));
            //ll.AddAfter(n1, 900);
            ll.Remove(45);//it will find the address of the data and then will
delete the data implicitly.

        }
    }
```

ListAnd Dictionary

```
class ListAndDictionaryDemo
    {
        static void Main(string[] args)
        {
            //underlying DS growable array
            //hashing
            //duplicate values are allowed

            List<int> al = new List<int>();
            al.Add(23);
            al.Add(90);
            Console.WriteLine(al[0]);

            //List<stud> al2 = new List<stud>();

            //array hashing
```

```
Dictionary<int, string> d1 = new Dictionary<int, string>();
d1.Add(23,"om");
d1.Add(90,"rohit");

Console.WriteLine(d1.ContainsKey(23));
Console.WriteLine(d1.ContainsValue("rohit"));
d1[23] = "Vaibhav";
d1.Remove(23);

foreach(KeyValuePair<int,string> k in d1)
{
    Console.WriteLine(k.Key + "==>" + k.Value);
}

foreach(int a in d1.Keys)
{
    Console.WriteLine(a + "==>" + d1[a]);
}


//LIFO
Stack<string> st = new Stack<string>();
st.Push("AAA");
st.Push("ABB");
st.Push("DDd");
Console.WriteLine(st.Pop());
Console.WriteLine(st.Peek());
```

```
            foreach(string d in st)
                Console.WriteLine(d);

            //FIFO
            Queue<double> q = new Queue<double>();
            q.Enqueue(9.4);
            q.Enqueue(7.4);
            Console.WriteLine(q.Dequeue());//9.4



        }
    }
```

SortedList Homework

```
class Book:IComparable<Book>
    {
        public int bid;
        public string bname;
        public int bprice;
         public Book(int bid, string bname, int bprice)
         {
            this.Bid = bid;
            this.Bname = bname;
            this.Bprice = bprice;

         }
    public int Bid { get => bid; set => bid = value; }
    public string Bname { get => bname; set => bname = value; }
    public int Bprice { get => bprice; set => bprice = value; }

    public int CompareTo(Book? other)
    {
        if (this.bprice == other.bprice)
            return this.bname.CompareTo(other.bname);
        else
            return this.bprice.CompareTo(other.bprice);
    }
    public override string ToString()
    {
        return $"Id : {bid} Name:{bname} Price:{bprice}";
    }
}

class SortedListBook
{
    static void Main(string[] args)
    {
        SortedList<Book, string> ss = new SortedList<Book, string>();
        ss.Add(new Book(1,"Harry Potter",200),"J.K.RowLing");
        ss.Add(new Book(2,"Insomnia",250),"Rachna Bisht");
        ss.Add(new Book(3,"Throne Of Glass",600),"Sarah J Maas");
        ss.Add(new Book(4,"Assassin's Blade",200),"Sarah J Maas");
```

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

HashTableDemo

```
class HashTableDemo
    {
        static void Main(string[] args)
        {
            //non generic
            //Key-Value Pair
            //Keys always should be unique.

            Hashtable ht = new Hashtable();
            ht.Add("Rohit", 90);
            ht.Add("Aadarsh", 91);
            ht.Add(23,"Jayraj");
           // ht.Add(new Student(1, "priya"),"priya@gmail.com");

            Console.WriteLine(ht[23]);
            ht["Aadarsh"] = 95;
            foreach(DictionaryEntry d in ht)
            {
                Console.WriteLine(d.Key + "=>" + d.Value);
            }
            Console.WriteLine("//////////////////////////////////////////");
            //ht.Clear();// it will clear entire hashtable.
            ht.Remove("Rohit");//it will remove the key-value pair if you
specify the key in remove method.
            foreach( var k in ht.Keys)
            {
                Console.WriteLine(k + "--->" + ht[k]);
            }

            Console.WriteLine(ht.ContainsKey(896));//it will check if the key
is present or not according to that it will return true or false
            Console.WriteLine(ht.ContainsValue("Jayraj"));

        }
    }


    class Hash
    {
        static void Main(string[] args)
        {
            Hashtable ht1 = new Hashtable();
            ht1.Add(new stud(1, "Rohit",85),90.4);
            ht1.Add(new stud(1, "Rohit",85),90.4);

            ht1.Add(new stud(1, "Rohit",85),90.4);

            ht1.Add(new stud(1, "Rohit",85),90.4);


            foreach(DictionaryEntry e in ht1)

            {

                Console.WriteLine(e.Key +"==>" + e.Value);

            }

        }

    }
```

HahTableExample

```
 class ItemPurchase

    {

        public static void Main(string[] args)

        {

            //frequency of items using hash table

            ArrayList al = new ArrayList()

        {

            "Laptop",

            "Mobile",

            "Headphones",

            "Tablet",

            "Laptop",

            "Mobile",
        };
            Hashtable ht = new Hashtable();
            foreach (dynamic data in al)
            {
                if (ht.ContainsKey(data))
                {
                    int value = (int)ht[data];
                    ht[data] = value + 1;
                }
                else
                {
                    ht.Add(data, 1);
                }
            }
            foreach (DictionaryEntry d in ht)
            {
                Console.WriteLine(d.Key + "==>" + d.Value);
            }
        }
    }
```

Custom Exception

```
 class Demo:ApplicationException
    {
        public Demo(string msg) : base(msg)
        {

        }

    class User
    {
        string nm;
        long mobile;
        string password;
        public void accept()
        {
            Console.WriteLine("Enter mobile number,name,Passs");
            nm = Console.ReadLine();
            mobile = long.Parse(Console.ReadLine());
            password = Console.ReadLine();
            validate();

        }
       public void validate()
        {
            if (password.Length < 8)
            {
                throw new Demo("Password size should br greater than
8");
            }
        }
    }

    class CustomExceptionDemo
    {
        static void Main(string[] args)
        {
            User u = new User();
            while (true)
            {
                try
                {
                    u.accept();
                    break;
                }
                catch (Demo e)
                {
                    Console.WriteLine(e.Message);
                }
            }
            Console.WriteLine("Main ends");
        }
    }
}
```

ExceptionHandlingDemo

```
 class ExceptionHandlingDemo
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter your name");
            string name = Console.ReadLine();
            Console.WriteLine("Enter your age");
            try
            {
                int age = int.Parse(Console.ReadLine());

                Console.WriteLine("Name= " + name + "Age=" + age);
                Console.WriteLine("Charcter at the 5th position is: "
+name[4]);
                try
                {
                    Console.WriteLine("Enter 2 numbers");
                    int a = int.Parse(Console.ReadLine());//12
                    int b = int.Parse(Console.ReadLine());//om
                    Console.WriteLine("Division=" + (a / b));
                }
                catch(DivideByZeroException e)
                {
                    Console.WriteLine(e.Message);
                }
                //as we do not have a format exception in inner try catch..it
will check if
                //its parent has it or not..if its parent class has it then it will
use that
                //catch block...so even if inner try do not have
FormatException the program will
                // not exit abruptly.

            }
            catch(FormatException e)
            {
                Console.WriteLine("Enter a numeric value");
            }
            catch(IndexOutOfRangeException e)
            {
                Console.WriteLine(e.Message);
            }
            catch(SystemException e)
            {
                Console.WriteLine(e.Message);
            }
            for(int i=0; i<=5; i++)
            {
                Console.WriteLine(i);
            }
        }
    }
    class A
    {
        static void divide(int a, int b)
        {
            Console.WriteLine("Division : " +(a /b));//either you can put a try
catch block here or in main method. the clr will
                                                     //see if you have used try catch in the
method if you havent,then it will check
                                                     //if you have written it in the main
method from where you are calling your divide
                                                     //method if you have put there a try
catch bloke then it will not abrubptly exit the code.
        }
        static void Main(string[] args)
        {
            Console.WriteLine("Main starts");
            try
            {
                divide(10, 0);
            }
            catch(DivideByZeroException e)
            {
                Console.WriteLine("in the main exception is handled");
                Console.WriteLine(e.Message);
            }
        }
```

```
}
class B
{
    static int division(int a, int b)
    {
        try
        {
            Console.WriteLine("in division");
            return a / b;
        }
        catch(DataMisalignedException e)
        {
            Console.WriteLine(e.Message);
        }
        finally
        {
            Console.WriteLine("in finally");
        }
        return 0;
    }
    static void Main(string[] args)
    {
        Console.WriteLine("Main starts");
        int result = division(10, 2);
        Console.WriteLine("Answer: " + result);
        Console.WriteLine("Main ends");
        }

    }

NullReferenceException

class NullReferenceExceptionDemo

    {

       public static string name;

        static void Main(string[] args)

        {

            Console.WriteLine("Enter the name which contains less than 4
elements");

            string str = Console.ReadLine();

            Console.WriteLine("Enter a number");

            int num1 = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine("Enter another number");

            int num2 = Convert.ToInt32(Console.ReadLine());

       using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Collections;
namespace Amazon
{
    class Harshad
    {
        static void Main(string[] args)
        {
```

```
        Console.WriteLine("Enter number");
        int n = int.Parse(Console.ReadLine());
        //n=45
        int sum = 0;
        int a = n;
        while (n > 0)
        {
            int last = n % 10;
            sum = sum + last;
            n = n / 10;
        }

        //sum=9

        if(a%sum==0)
            Console.WriteLine("Harshad niven number");
        else
            Console.WriteLine("Not");

    }
}

class Frequency
{
    static void Main(string[] args)
    {
        Console.WriteLine("enter mobile number");
        long mb = long.Parse(Console.ReadLine());
        Console.WriteLine("Enter digit to be searched");
        int search = int.Parse(Console.ReadLine());

        //mb=9822081081
        //0->2
        //1->2
        //8->3
        //9->1
        //2->2
        //0-9
        long temp = mb;
        for(int i=0;i<=9;i++)
        {
            int c = 0;
            //mb=0
            while(mb>0)
            {
                long last = mb % 10;
                if (last == i)
                    c++;

                mb = mb / 10;
            }
            mb = temp;
            if(c>0)
                Console.WriteLine("Freq of "+i+" = "+c);

        }


    }
}
class Demo1
{
    static void Main(string[] args)
    {
        ArrayList al = new ArrayList();
        al.Add("java");
        al.Add("java");


        Hashtable ht = new Hashtable();
        ht.Add("om","java");
        ht.Add("Beena", "C#");
        ht.Add("Reena", "java");
        ht.Add("Teena", "C#");
        ht.Add("raj", "React");
        Hashtable ht2 = new Hashtable();
        //key subject name
        //value --arraylist of student name
        //"java"==>[om,R]
        //"C#"=>[B,







    }
}

class OrderItem
{
    string itemname;
    int qty;
    int price;

}
class Bill
{


    static int GetPrice(string itemname,Hashtable ht)
    {
        int price = 0;
        foreach (DictionaryEntry d in ht)
        {
            Hashtable submenu = (Hashtable)d.Value;
            if(submenu.ContainsKey(itemname))
            {
                price = (int)submenu[itemname];
                break;
            }
        }
```

```
            return price;
        }
        static void Main(string[] args)
        {
            Hashtable beverages = new Hashtable();
            beverages.Add("coffee", 50);
            beverages.Add("tea", 50);
            beverages.Add("cold coffee", 100);
            beverages.Add("orange juice", 150);


            Hashtable snacks = new Hashtable();
            snacks.Add("pizza", 250);
            snacks.Add("burger", 100);
            snacks.Add("samosa", 40);
            snacks.Add("vada pav", 40);


            Hashtable menu = new Hashtable();
            menu.Add("Snacks", snacks);
            menu.Add("Beverages", beverages);


            foreach (DictionaryEntry menuitem in menu)
            {
                Console.WriteLine(menuitem.Key);

                Hashtable ht = (Hashtable)menuitem.Value;

                foreach(DictionaryEntry submenu in ht)
                {
                    Console.WriteLine(submenu.Key + "===> Rs" +
    submenu.Value);
                }
                Console.WriteLine("----------------------------------");
            }

            Hashtable orderlist = new Hashtable();
            do
            {
                Console.WriteLine("enter item from menu u want to order");
                string item = Console.ReadLine().ToLower();
                Console.WriteLine("enter qty");
                int qty = int.Parse(Console.ReadLine());
                orderlist.Add(item, qty);
                Console.WriteLine("do you want to add one more item to 
    your order(Y/N)");
                string choice = Console.ReadLine().ToLower();
                if (choice=="n")
                    break;

            }while (true);



            Console.WriteLine("========================================
    =================");
            Console.WriteLine("Order Details");
            int total = 0;
            Console.WriteLine("Menu \t Qty \t Price \t Amount");

            foreach(DictionaryEntry d in orderlist)
            {
                string itemname = (string)d.Key;
                int qty =(int) d.Value;
                int pr = GetPrice(itemname, menu);
                Console.WriteLine(itemname+"\t"+qty+"\t"+pr+"\t"+qty*pr);
                total = total + (qty * pr);



            }

            Console.WriteLine("Total Bill Amount "+total);
```

```
            }






        }
    }
     try

            {

                Console.WriteLine("The 6th element of string" + str[6]);

                Console.WriteLine("Division=" + (num1 / num2));



                //Console.WriteLine(name.ToCharArray());//it will give you a
NullReferenceException

                                                       //because if you do not put any value
inside a

                                                       //non Primitive so that means its
reference object will not be able to

                                                       //point the value where it is stored. so it
will give you a exception.

            }

            catch(NullReferenceException e)

            {

                Console.WriteLine(e.Message);

                Console.WriteLine("Null reference exception occured");

            }

            catch (IndexOutOfRangeException e)

            {
                Console.WriteLine(e.Message);
                Console.WriteLine("Index out of bound");
            }
            catch (DivideByZeroException e)
            {
                Console.WriteLine(e.Message);
            }
            catch (SystemException e)
            {
                Console.WriteLine(e.Message);
            }
        }



    }
```

Stack And Queue

```
 class StackDemo
    {
        static void Main(string[] args)
        {
            //LIFO
            Stack st = new Stack();
            st.Push("Aadarsh");
            st.Push("Jayraj");
            st.Push("Siddhant");

            //push = add
            //pop = remove and then returns the value

            foreach(object ob in st)
            {
                Console.WriteLine(ob);
            }

            string d = (string)st.Pop();
            Console.WriteLine("Removed ==>" + d);

            Console.WriteLine("Peek: "+ st.Peek()); // the difference
between pop and peek is that, pop removes the data from the stack
and then returns it.
                                        //whereas peek just shows what is
present at the top of the stack. peek does not remove the data that is
stored inside the stack.

            foreach(object ob in st)
            {
                Console.WriteLine(ob);
            }
        }
    }

    class QuequeDemo
    {
        static void Main(string[] args)
    {
        Queue q = new Queue();
        q.Enqueue(10);
        q.Enqueue("raj");
        q.Enqueue("shree");
        Console.WriteLine(q);

        int d = (int)q.Dequeue();
        Console.WriteLine(q.Peek());
        Console.WriteLine("remove" + d);

        foreach (var i in q)
        {
            Console.WriteLine(i);
        }

    }
}
```