```
using System;
using System.Collections.Generic;
namespace Ordered_List
{
   class OrderedList
   {
      static void Main(string[] args)
      {
         List<string> orderdList = new List<string>();

         Console.Write("Please enter a term or 'END' to terminate: ");
         string term = Console.ReadLine();
         while (term.ToUpper() != "END")
         {
            term = term.ToLower();
            int insertPoint = 0;
            bool reject = false;
            for(int i = 0 ; i < orderdList.Count ; i++)
            {
               if (orderdList[i].IndexOf(term) != -1)
                  Console.WriteLine("Similar term found: {0}", orderdList[i]);
               if (orderdList[i] == term)      reject = true;
               if (orderdList[i].CompareTo(term) < 0)       insertPoint++;
            }
            if (!reject)
               orderdList.Insert(insertPoint, term);
            else
               Console.WriteLine("Term rejected");

            Console.WriteLine();
            Console.WriteLine("List:");
            for (int i = 0; i < orderdList.Count; i++)    Console.WriteLine("\t{0}", orderdList[i]);
            Console.WriteLine();

            Console.Write("Please enter a term or 'END' to terminate: ");
            term = Console.ReadLine();
         }
      }
   }
}
```

```
using System;
using System.Text.RegularExpressions;
namespace RegexTester
{
   class RegexTester
   {
      static void Main(string[] args)
      {
         Regex regexTelephone = new Regex(@"^\d{2}-\d{4}-\d{4}$");
         Regex regexCurrency = new Regex(@"^\$\d+\.\d{2}$");
         Regex regexAddress = new Regex(@"^\d+ \D+ \D+$");
         Regex regexPasswordUppers = new Regex(@"[A-Z]");
         // note that it is not possible to test any three from
         Regex regexPasswordLowers = new Regex(@"[a-z]");
         // four, in any order, without using multiple expressions
         Regex regexPasswordNumbers = new Regex(@"[0-9]");
         Regex regexPasswordSymbols = new Regex(@"[!@#$%^&*()]");

         Console.Write("Please enter text to test or 'END' to finish: ");
         string text = Console.ReadLine();
         while (text.ToUpper() != "END")
         {
            bool noMatch = true;
            if (regexTelephone.Match(text).Success)
            {
               noMatch = false;
               Console.WriteLine("The text appears to be a valid telephone number");
            }
            if (regexCurrency.Match(text).Success)
            {
               noMatch = false;
               Console.WriteLine("The text appears to be a valid currency value");
            }
            if (regexAddress.Match(text).Success)
            {
               noMatch = false;
               Console.WriteLine("The text appears to be a valid address");
            }
```

```
            int passwordCount = 0;

            if (regexPasswordUppers.Match(text).Success) passwordCount++;
            if (regexPasswordLowers.Match(text).Success) passwordCount++;
            if (regexPasswordNumbers.Match(text).Success) passwordCount++;
            if (regexPasswordSymbols.Match(text).Success) passwordCount++;
            if (passwordCount >= 3)
            {
                noMatch = false;
                Console.WriteLine("The text appears to be a valid password");
            }
            if (noMatch)
                Console.WriteLine("The text does not appear to be anything in particular!");

            Console.Write("Please enter text to test or 'END' to finish: ");
            text = Console.ReadLine();
        }
    }
  }
}
```

**Task 10.3**

**empty string**

A string object that contains no characters/text, i.e., "".

**null string**

A string variable that does not refer to any string object, i.e., is set to a null value

**regular expression**

A sequence of characters and escape sequences that defines a pattern that can be matched to any textual value.

**regular string literal**

A string literal (textual value appearing in code) whose contents is checked/interpreted for escape sequences such as \n, e.g., `"a regular string literal\n"`.

**verbatim string literal**

A string literal (textual value appearing in code) whose contents are not checked/interpreted for escape sequences, e.g., @"a backslash-n looks like \n".