1. 将字符串s1中所有出现在字符串s2中的字符删去

[复制代码](javascript:void(0);)

using System;

namespace ConsoleApplication56

{

class Program

{

static void Main(string[] args)

{

string s1 = "aaabbccabfcabcc";

string s2 = "abcd";

char[] results = new char[s1.Length];

int m=0;

int j, i;

for (i = 0; i < s1.Length; i++)

{

for (j=0; j < s2.Length; j++)

{

if (s1[i] != s2[j])

{

continue;

}

else

{

j = 0;

break;

}

}

if (j == s2.Length)

{

results[m++] = s1[i];

}

}

foreach (char a in results)

{

Console.WriteLine(a.ToString());

}

Console.ReadLine();

}

}

}

[复制代码](javascript:void(0);)

2. 计算数字出现次数(using LinkedList)

[复制代码](javascript:void(0);)

using System;

using System.Collections.Generic;

namespace ConsoleApplication57

{

class Program

{

static void Main(string[] args)

{

int[] nums = { 1, 2, 3, 4, 2, 1 };

int count = 0;

if (nums != null)

{

LinkedList<int> numbers = new LinkedList<int>(nums);

int givenInt = 1;

LinkedListNode<int> currentNode = numbers.First;

while (currentNode != null)

{

if (currentNode.Value == givenInt)

{

count++;

}

currentNode = currentNode.Next;

}

}

Console.WriteLine(count);

Console.ReadLine();

}

}

}

[复制代码](javascript:void(0);)

3.I have two strings the ending substring of first is the starting substring of the second,ex

string left : ONESTRING

string right : STRINGTWO

I have to merge them so that the resulting string is

result string : ONESTRINGTWO

[复制代码](javascript:void(0);)

using System;

namespace ConsoleApplication58

{

class Program

{

static void Main(string[] args)

{

string left = "OneStart";

string right = "StartTwo";

string result = FindAndMerge(left, right);

Console.WriteLine(result);

Console.ReadLine();

}

public static string FindAndMerge(string left,string right)

{

int i=0;

string result=string.Empty;

for (i = right.Length; i >= 0; i--)

{

if (left.Contains(right.Substring(0, i)))

{

break;

}

}

if (i < right.Length)

{

result = right.Substring(i);

}

result = left + result;

return result;

}

}

}

[复制代码](javascript:void(0);)

4. 删除重复元素

[复制代码](javascript:void(0);)

using System;

namespace ConsoleApplication41

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Please input the size of the int Array:");

int size = Int32.Parse(Console.ReadLine());

int[] array = new int[size];

Console.WriteLine("Please input {0} numbers:", size);

if (size > 0)

{

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

{

array[i] = GetInputNum();

}

int[] result = DelDupElements(array);

foreach (int i in result)

{

Console.WriteLine(i);

}

}

else

{

Console.WriteLine("Please input new array with more than zero numbers");

}

Console.ReadLine();

}

/// <summary>

///

/// </summary>

/// <returns></returns>

public static int GetInputNum()

{

try

{

return Int32.Parse(Console.ReadLine());

}

catch

{

Console.WriteLine("Please input a number");

return GetInputNum();

}

}

/// <summary>

///

/// </summary>

/// <param name="a"></param>

/// <returns></returns>

public static int[] DelDupElements(int[] a)

{

// Special judgement

if (a.Length == 0)

{

Console.WriteLine("Please input new array with more than zero numbers");

return null;

}

else

{

// Sort the array

Sort(a);

Console.WriteLine("The sorted array:");

foreach (int i in a)

{

Console.WriteLine(i);

}

Console.ReadLine();

int[] resultArrary;

int j = 1;

int size = 1;

for (int i = 1; i < a.Length; i++)

{

if (a[i] != a[i - 1])

{

size++;

}

}

resultArrary = new int[size];

resultArrary[0] = a[0];

for (int i = 1; i < a.Length; i++)

{

if (a[i] != a[i - 1])

{

resultArrary[j++] = a[i];

}

}

return resultArrary;

}

}

/// <summary>

///

/// </summary>

/// <param name="a"></param>

public static void Sort(int[] a)

{

int temp;

for (int i = 1; i < a.Length; i++)

{

for (int j = 0; j < a.Length - i; j++)

{

if (a[j] > a[j + 1])

{

temp = a[j];

a[j] = a[j + 1];

a[j + 1] = temp;

}

}

}

}

}

}

[复制代码](javascript:void(0);)

另外几种种写法：

[复制代码](javascript:void(0);)

public static void DeleteDuplicateNumbers(int[] array)

{

if (array != null)

{

List<int> result = new List<int>();

if (array.Length > 1)

{

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

{

if (!result.Contains(array[i]))

{

result.Add(array[i]);

}

}

}

else

{

result.Add(array[0]);

}

foreach (int i in result)

{

Console.WriteLine(i);

}

Console.ReadLine();

}

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

public static void DeleteDuplicateNumbersUsingIndexOf(int[] array)

{

if (array != null)

{

List<int> result = new List<int>();

if (array.Length > 1)

{

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

{

if (result.IndexOf(array[i]) < 0)

{

result.Add(array[i]);

}

}

}

else

{

result.Add(array[0]);

}

foreach (int i in result)

{

Console.WriteLine(i);

}

Console.ReadLine();

}

else

{

Console.WriteLine("The array is null");

}

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

public static void DeleteDuplicateNumbersUsingInsert(int[] array)

{

if (array != null)

{

List<int> result = new List<int>();

result.Add(array[0]);

if (array.Length > 1)

{

for (int i = 1; i < array.Length; i++)

{

if (array[i] < result[0])

{

result.Insert(0, array[i]);

}

else if (array[i] > result[result.Count - 1])

{

result.Add(array[i]);

}

else

{

for (int j = 1; j < result.Count-1; j++)

{

if (array[i] > result[j-1] && array[i] < result[j])

{

result.Insert(j, array[i]);

}

}

}

}

}

foreach (int i in result)

{

Console.WriteLine(i);

}

Console.ReadLine();

}

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

public static int[] DeleteDuplicateNumbersUsingArrayList(int[] list)

{

ArrayList arr = new ArrayList();

for (int i = 0; i < list.Length - 1; i++)

{

if (list[i] != list[i + 1])

{

arr.Add(list[i]);

}

}

arr.Add(list[list.Length - 1]);

int[] getarr = new int[arr.Count];

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

{

getarr[i] = Convert.ToInt32(arr[i]);

}

return getarr;

}

[复制代码](javascript:void(0);)

5. 合并有序数组

[复制代码](javascript:void(0);)

using System;

namespace ConsoleApplication62

{

class Program

{

static void Main(string[] args)

{

int[] a = { 1, 2, 2, 3, 5, 6, 7, 7 };

int[] b = { 1, 2, 3, 4, 8, 8, 9, 10, 11, 12, 12, 13, 14 };

int[] c = MergeList(a, b);

foreach (int i in c)

{

Console.WriteLine(i);

}

Console.ReadLine();

}

public static int[] MergeList(int[] a, int[] b)

{

if (checkSort(a) && checkSort(b))

{

int len = a.Length + b.Length;

int[] result = new int[len];

int i = 0, j = 0, k = 0;

while (i < a.Length && j < b.Length)

{

if (a[i] < b[j])

{

result[k++] = a[i++];

}

else

{

result[k++] = b[j++];

}

}

while (i < a.Length)

{

result[k++] = a[i++];

}

while (j < b.Length)

{

result[k++] = b[j++];

}

return result;

}

else

{

return null;

}

}

public static bool checkSort(int[] a)

{

for (int i = 0; i < a.Length - 1; i++)

{

for (int j = i + 1; j < a.Length; j++)

{

if (a[j - 1] > a[j])

{

return false;

}

}

}

return true;

}

}

}

[复制代码](javascript:void(0);)

 6. 有一列数1，1，2，3，5，........求第30个数.

[复制代码](javascript:void(0);)

public class MainClass

{

public static void Main()

{

Console.WriteLine(Foo(12));

Console.ReadLine();

}

public static int Foo(int i)

{

if (i <= 0)

return 0;

else if (i > 0 && i <= 2)

return 1;

else return Foo(i - 1) + Foo(i - 2);

}

}

[复制代码](javascript:void(0);)

7. 冒泡排序、选择排序、插入排序

[复制代码](javascript:void(0);)

// 冒泡排序 bubble sort

public static int[] BubbleSort(int[] array)

{

int temp;

bool isDone = false;

for (int i = 0; i < array.Length && !isDone; i++)

{

isDone = true;

for (int j = 0; j < array.Length - i - 1; j++)

{

if (array[j] > array[j + 1])

{

isDone = false;

temp = array[j];

array[j] = array[j + 1];

array[j + 1] = temp;

}

}

}

return array;

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

// 插入排序 Insertion sort

public static int[] InsertionSort(int[] array)

{

for (int i = 1; i < array.Length; i++)

{

int current = array[i];

int j = i;

while (j > 0 && current < array[j - 1])

{

array[j] = array[j - 1];

j--;

}

array[j] = current;

}

return array;

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

// 选择排序 Selection sort

public static int[] SelectionSort(int[] array)

{

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

{

int min = i;

for (int j = i + 1; j < array.Length; j++)

{

if (array[j] < array[min])

{

min = j;

}

}

int temp = array[i];

array[i] = array[min];

array[min] = temp;

}

return array;

}

[复制代码](javascript:void(0);)

8. 找第二大的数

[复制代码](javascript:void(0);)

// 获得数组中的第二大数

public static int SecondMax(int[] array)

{

int max = array[0];

int secondMax = array[0];

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

{

if (array[i] > max)

{

secondMax = max;

max = array[i];

}

else if (array[i] > secondMax && array[i] != max)

{

secondMax = array[i];

}

}

if (secondMax == max)

{

throw new Exception("Have no second max!");

}

return secondMax;

}

[复制代码](javascript:void(0);)

9. 有一个字符串 "I am a good man",设计一个函数,返回 "man good a am I"

[复制代码](javascript:void(0);)

// 反转单词顺序

public static string WordReverse(string str)

{

char[] array = str.ToArray();

CharArrayReverse(array, 0, array.Length - 1);

int start = -1;

int end = -1;

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

{

if (!((array[i] >= 'a' && array[i] <= 'z') || (array[i] >= 'A' && array[i] <= 'Z')))

{

if (start < end)

{

CharArrayReverse(array, start + 1, end);

}

start = i;

}

else

{

end = i;

}

}

return new string(array);

}

public static void CharArrayReverse(char[] array, int start, int end)

{

if (array != null && start < array.Length && end < array.Length)

while (start < end)

{

char temp = array[start];

array[start] = array[end];

array[end] = temp;

start++;

end--;

}

}

[复制代码](javascript:void(0);)

10.  Ａ、Ｂ、Ｃ、Ｄ、Ｅ五名学生有可能参加计算机竞赛，根据下列条件判断哪些   人参加了竞赛：

   （１）Ａ参加时，Ｂ也参加；

   （２）Ｂ和Ｃ只有一个人参加；

   （３）Ｃ和Ｄ或者都参加，或者都不参加；

   （４）Ｄ和Ｅ中至少有一个人参加；

   （５）如果Ｅ参加，那么Ａ和Ｄ也都参加。

[复制代码](javascript:void(0);)

public static void Main()

{

char[] name = { 'A', 'B', 'C', 'D', 'E' };

int[] value = new int[5];

for (value[0] = 0; value[0] < 2; value[0]++)

{

for (value[1] = 0; value[1] < 2; value[1]++)

{

for (value[2] = 0; value[2] < 2; value[2]++)

{

for (value[3] = 0; value[3] < 2; value[3]++)

{

for (value[4] = 0; value[4] < 2; value[4]++)

{

if ((value[1] >= value[0]) && (value[1] + value[2] == 1) && (value[2] == value[3]) && (value[3] + value[4] == 1) && (value[4] == 0 || value[4] == 1 && value[0] == 1 && value[3] == 1))

{

for (int i = 0; i < 5; i++)

{

if (value[i] == 1)

{

Console.WriteLine("{0}Join", name[i]);

}

else

{

Console.WriteLine("{0}Not Join", name[i]);

}

}

}

}

}

}

}

}

Console.ReadLine();

}

[复制代码](javascript:void(0);)

11. a user entered an integer value into a text box. Without using a buit-in library, convert the numeric string to its integer representation.

[复制代码](javascript:void(0);)

public static int StringToInt(string s)

{

int sum = 0;

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

{

sum = sum \* 10 + (s[i] - '0');

}

return sum;

}

[复制代码](javascript:void(0);)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApplication10

{

public class MainClass

{

public static void Main()

{

//char[] name = { 'A', 'B', 'C', 'D', 'E' };

//int[] value = new int[5];

//for (value[0] = 0; value[0] < 2; value[0]++)

//{

// for (value[1] = 0; value[1] < 2; value[1]++)

// {

// for (value[2] = 0; value[2] < 2; value[2]++)

// {

// for (value[3] = 0; value[3] < 2; value[3]++)

// {

// for (value[4] = 0; value[4] < 2; value[4]++)

// {

// if ((value[1] >= value[0]) && (value[1] + value[2] == 1) && (value[2] == value[3]) && (value[3] + value[4] == 1) && (value[4] == 0 || value[4] == 1 && value[0] == 1 && value[3] == 1))

// {

// for (int i = 0; i < 5; i++)

// {

// if (value[i] == 1)

// {

// Console.WriteLine("{0}Join", name[i]);

// }

// else

// {

// Console.WriteLine("{0}Not Join", name[i]);

// }

// }

// }

// }

// }

// }

// }

//}

int a = 8;

int b = 5;

DivisorAndMultiple(a, b);

Console.ReadLine();

}

public static int StringToInt(string s)

{

int sum = 0;

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

{

sum = sum \* 10 + (s[i] - '0');

}

return sum;

}

// 递归

public static int Foo(int i)

{

if (i <= 0)

return 0;

else if (i > 0 && i <= 2)

return 1;

else return Foo(i - 1) + Foo(i - 2);

}

// 冒泡排序 bubble sort

public static int[] BubbleSort(int[] array)

{

int temp;

bool isDone = false;

for (int i = 0; i < array.Length && !isDone; i++)

{

isDone = true;

for (int j = 0; j < array.Length - i - 1; j++)

{

if (array[j] > array[j + 1])

{

isDone = false;

temp = array[j];

array[j] = array[j + 1];

array[j + 1] = temp;

}

}

}

return array;

}

// 插入排序 Insertion sort

public static int[] InsertionSort(int[] array)

{

for (int i = 1; i < array.Length; i++)

{

int current = array[i];

int j = i;

while (j > 0 && current < array[j - 1])

{

array[j] = array[j - 1];

j--;

}

array[j] = current;

}

return array;

}

// 获得数组中的第二大数

public static int SecondMax(int[] array)

{

int max = array[0];

int secondMax = array[0];

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

{

if (array[i] > max)

{

secondMax = max;

max = array[i];

}

else if (array[i] > secondMax && array[i] != max)

{

secondMax = array[i];

}

}

if (secondMax == max)

{

throw new Exception("Have no second max!");

}

return secondMax;

}

// 反转单词顺序

public static string WordReverse(string str)

{

char[] array = str.ToArray();

CharArrayReverse(array, 0, array.Length - 1);

int start = -1;

int end = -1;

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

{

if (!((array[i] >= 'a' && array[i] <= 'z') || (array[i] >= 'A' && array[i] <= 'Z')))

{

if (start < end)

{

CharArrayReverse(array, start + 1, end);

}

start = i;

}

else

{

end = i;

}

}

return new string(array);

}

public static void CharArrayReverse(char[] array, int start, int end)

{

if (array != null && start < array.Length && end < array.Length)

while (start < end)

{

char temp = array[start];

array[start] = array[end];

array[end] = temp;

start++;

end--;

}

}

// 判断素数

public static bool Prime(int a)

{

if (a == 2)

{

return true;

}

else if (a < 2)

{

return false;

}

for (int i = 2; i < a; i++)

{

if (a % i == 0)

{

return false;

}

}

return true;

}

// 合并有序数组

public static int[] MergeList(int[] a, int[] b)

{

if (checkSort(a) && checkSort(b))

{

int len = a.Length + b.Length;

int[] result = new int[len];

int i = 0, j = 0, k = 0;

while (i < a.Length && j < b.Length)

{

if (a[i] < b[j])

{

result[k++] = a[i++];

}

else

{

result[k++] = b[j++];

}

}

while (i < a.Length)

{

result[k++] = a[i++];

}

while (j < b.Length)

{

result[k++] = b[j++];

}

return result;

}

else

{

return null;

}

}

public static bool checkSort(int[] a)

{

for (int i = 0; i < a.Length - 1; i++)

{

for (int j = i + 1; j < a.Length; j++)

{

if (a[j - 1] > a[j])

{

return false;

}

}

}

return true;

}

// 最大公约数和最小公倍数

public static void DivisorAndMultiple(int m, int n)

{

int x, y, result1 = 0, result2 = 0;

if (m > n)

{

x = n;

y = m;

}

else

{

x = m;

y = n;

}

for (int i = 1; i <= x; i++)

{

if (m % i == 0 && n % i == 0)

{

result1 = i;

}

}

Console.WriteLine("{0}和{1}的最小公倍数是：{2}", m, n, result1);

for (int j = y; j <= x \* y; j++)

{

if (j % m == 0 && j % n == 0)

{

result2 = j;

break;

}

}

Console.WriteLine("{0}和{1}的最大公约数是：{2}", m, n, result2);

}

}

//单链表结点类,采用泛型

public class Node<T>

{

private T data; //数据域,当前结点的数据

private Node<T> next; //引用域,即下一结点

//构造器：数据域+引用域，普通结点

public Node(T item, Node<T> p)

{

data = item;

next = p;

}

//构造器：引用域，头结点

public Node(Node<T> p)

{

next = p;

}

//构造器：数据域，尾结点

public Node(T val)

{

data = val;

next = null;

}

//构造器：无参数

public Node()

{

data = default(T);

next = null;

}

//数据域属性

public T Data

{

get

{

return data;

}

set

{

data = value;

}

}

//引用域属性

public Node<T> Next

{

get

{

return next;

}

set

{

next = value;

}

}

}

//链表类，包含链表定义及基本操作方法

public class MyLinkList<T>

{

private Node<T> head; //单链表的头结点

//头结点属性

public Node<T> Head

{

get

{

return head;

}

set

{

head = value;

}

}

//构造器

public MyLinkList()

{

head = null;

}

//求单链表的长度

public int GetLength()

{

Node<T> p = head;

int len = 0;

while (p != null)

{

++len;

p = p.Next;

}

return len;

}

//清空单链表

public void Clear()

{

head = null;

}

//判断单链表是否为空

public bool IsEmpty()

{

if (head == null)

{

return true;

}

else

{

return false;

}

}

//在单链表的末尾添加新元素

public void Append(T item)

{

Node<T> q = new Node<T>(item);

Node<T> p = new Node<T>();

if (head == null)

{

head = q;

return;

}

p = head;

while (p.Next != null)

{

p = p.Next;

}

p.Next = q;

}

//在单链表的第i个结点的位置前插入一个值为item的结点

public void Insert(T item, int i)

{

if (IsEmpty() || i < 1 || i > GetLength())

{

Console.WriteLine("LinkList is empty or Position is error!");

return;

}

if (i == 1)

{

Node<T> q = new Node<T>(item);

q.Next = head;

head = q;

return;

}

Node<T> p = head;

Node<T> r = new Node<T>();

int j = 1;

while (p.Next != null && j < i)

{

r = p;

p = p.Next;

++j;

}

if (j == i)

{

Node<T> q = new Node<T>(item);

q.Next = p;

r.Next = q;

}

}

//在单链表的第i个结点的位置后插入一个值为item的结点

public void InsertPost(T item, int i)

{

if (IsEmpty() || i < 1 || i > GetLength())

{

Console.WriteLine("LinkList is empty or Position is error!");

return;

}

if (i == 1)

{

Node<T> q = new Node<T>(item);

q.Next = head.Next;

head.Next = q;

return;

}

Node<T> p = head;

int j = 1;

while (p != null && j < i)

{

p = p.Next;

++j;

}

if (j == i)

{

Node<T> q = new Node<T>(item);

q.Next = p.Next;

p.Next = q;

}

}

//删除单链表的第i个结点

public T Delete(int i)

{

if (IsEmpty() || i < 0 || i > GetLength())

{

Console.WriteLine("LinkList is empty or Position is error!");

return default(T);

}

Node<T> q = new Node<T>();

if (i == 1)

{

q = head;

head = head.Next;

return q.Data;

}

Node<T> p = head;

int j = 1;

while (p.Next != null && j < i)

{

++j;

q = p;

p = p.Next;

}

if (j == i)

{

q.Next = p.Next;

return p.Data;

}

else

{

Console.WriteLine("The " + i + "th node is not exist!");

return default(T);

}

}

//获得单链表的第i个数据元素

public T GetElem(int i)

{

if (IsEmpty() || i < 0)

{

Console.WriteLine("LinkList is empty or position is error! ");

return default(T);

}

Node<T> p = new Node<T>();

p = head;

int j = 1;

while (p.Next != null && j < i)

{

++j;

p = p.Next;

}

if (j == i)

{

return p.Data;

}

else

{

Console.WriteLine("The " + i + "th node is not exist!");

return default(T);

}

}

//在单链表中查找值为value的结点

public int Locate(T value)

{

if (IsEmpty())

{

Console.WriteLine("LinkList is Empty!");

return -1;

}

Node<T> p = new Node<T>();

p = head;

int i = 1;

while (!p.Data.Equals(value) && p.Next != null)

{

p = p.Next;

++i;

}

return i;

}

//显示链表

public void Display()

{

Node<T> p = new Node<T>();

p = this.head;

while (p != null)

{

Console.Write(p.Data + " ");

p = p.Next;

}

}

}

}