实验一

1.

namespace Exp1\_20170324

{

class Crypt

{

public static void Test()

{

Console.WriteLine("请输入字符串： ");

string text = Console.ReadLine();

Console.WriteLine("请输入加密字符：");

char key = Console.ReadKey().KeyChar;

Console.Write("\n加密后的字符串为：");

string encrypted = "";

foreach (char c in text)

{

char a = (char)(c ^ key);

encrypted += a;

}

Console.WriteLine(encrypted);

Console.Write("解码后的字符串为：");

string decrypted = "";

foreach (char c in encrypted)

{

char a = (char)(c ^ key);

decrypted += a;

}

Console.WriteLine(decrypted);

}

}

}

2.

namespace Exp1\_20170324

{

class MeanTool

{

public static double Mean(double a, double b, double c)

{

return (a + b + c) / 3;

}

public static void Mean(double a, double b, double c, ref double e, ref double f)

{

e = (a + b + c) / 3;

f = Math.Sqrt((a \* a + b \* b + c \* c) / 3);

}

public static void Mean2(double a, double b, double c, out double e, out double f)

{

e = (a + b + c) / 3;

f = Math.Sqrt((a \* a + b \* b + c \* c) / 3);

}

public static void Mean(out double v1, out double v2, params double[] numbers)

{

double sum = 0, multiply = 1, sum1 = 0;

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

{

sum += numbers[i];

}

v1 = sum / numbers.Length;

sum = 0;

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

{

sum += numbers[i] \* numbers[i];

}

v2 = Math.Sqrt(sum / numbers.Length);

}

}

class Ex3

{

public static void Test()

{

double a = 2, b = 3;

double v1, v2;

Console.WriteLine("函数1：{0}", MeanTool.Mean(a, b, 4));

Console.WriteLine("函数2：");

MeanTool.Mean(2, 3, 4, ref a, ref b);

Console.WriteLine("{0}, {1}", a, b);

Console.WriteLine("函数3：");

MeanTool.Mean2(2, 3, 4, out v1, out v2);

Console.WriteLine("{0}, {1}", v1, v2);

Console.WriteLine("函数4：");

MeanTool.Mean(out v1, out v2, 2, 3, 4, 5, 6);

Console.WriteLine("{0}, {1}", v1, v2);

}

}

}

3.

class Program

{

static void Main()

{

Random rd = new Random();

int value = rd.Next();

string str = value.ToString();

Console.WriteLine("原来的数字：" + str);

Console.WriteLine("原来的数字长度" + str.Length);

string[] reps = new string[] {"zero", "one", "two", "three", "four", "five",

"six", "seven", "eight", "nine"};

string newstr = str;

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

{

newstr = newstr.Replace(i.ToString(), reps[i]);

}

Console.WriteLine("替换后的字符串长度" + newstr.Length);

Console.WriteLine("替换后的字符串" + newstr);

string[] splits = newstr.Split('o');

foreach (string s in splits)

{

Console.WriteLine(s);

}

}

}

4.

class Delegates

{

//创建委托类型

public delegate bool NumberPredicate(int number);

static void Main(string[] args)

{

//生成20个随机数，放在数组numbers中并打印这些数

int[] numbers;

numbers = new int[20];

Random rd = new Random();

Console.WriteLine("数组包含以下值：");

for(int i=0; i<20; i++)

{

numbers[i] = rd.Next(100);

Console.Write(numbers[i] + "\t");

}

Console.WriteLine();

//生成委托实例

NumberPredicate evenPredicate = IsEven;

//利用委托变量调用IsEven

Console.WriteLine("通过委托变量调用IsEven方法: {0}",evenPredicate(numbers[0]));

//选出偶数并输出

Console.WriteLine("数组包含以下偶数：");

FilterArray(numbers, evenPredicate);

//选出素数并输出

Console.WriteLine("数组包含以下素数：");

FilterArray(numbers, IsPrime);

}

//选择满足predicate的数组元素并打印出来

private static void FilterArray(int[] intArray, NumberPredicate predicate)

{

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

{

int value = intArray[i];

if (predicate(value))

{

Console.Write(value + "\t");

}

}

Console.WriteLine();

}

//偶数判断函数

private static bool IsEven(int number)

{

return (number % 2 == 0);

}

//判断是否素数

private static bool IsPrime(int number)

{

if (number <= 1)

return false;

for (int i = 2; i <= Math.Sqrt(number); i++)

{

if (number % i == 0)

return false;

}

return true;

}

}

实验二

1.

public struct Point2d

{

public double x;

public double y;

public static readonly Point2d Origin = new Point2d();

public Point2d(double x, double y)

{

this.x = x;

this.y = y;

}

public override string ToString()

{

return String.Format("Point(x: {0}, y:{1})", x, y);

}

}

abstract class Shape

{

public abstract double Area { get; }

public abstract double Perimeter { get; }

public abstract bool Contains(double x, double y);

}

class Rect : Shape

{

public Point2d TopLeft { get; set; }

public double Width { get; set; }

public double Length { get; set; }

public override double Area

{

get

{

return Width \* Length;

}

}

public override double Perimeter

{

get

{

return 2 \* ( Width + Length);

}

}

public override bool Contains(double x, double y)

{

double subx = x - TopLeft.x;

double suby = y - TopLeft.y;

if (subx >= 0 && subx <= Length && suby >= 0 && suby <= Width)

return true;

else

return false;

}

public override string ToString()

{

return string.Format("Rect [ TopLeft: {0}, Length: {1}, Width: {2}, Area: {3}]", TopLeft, Length, Width, Area);

}

}

class Circle : Shape

{

public Point2d Center { get; set; }

public double Radius { get; set; }

public Circle()

: this(Point2d.Origin, 1)

{

}

public Circle(Point2d center, double radius)

{

Center = center;

Radius = radius;

}

public override double Area

{

get { return Math.PI \* Radius \* Radius; }

}

public override double Perimeter

{

get { return 2 \* Math.PI \* Radius; }

}

public override bool Contains(double x, double y)

{

double distance = Math.Sqrt((Center.x - x) \* (Center.x - x) + (Center.y - y) \* (Center.y - y));

return distance <= Radius;

}

public override string ToString()

{

return string.Format("Circle [ Center: {0}, Radius: {1}, Area: {2}]", Center, Radius, Area);

}

}

class GraphicsTest

{

public static void Main(string[] argv)

{

Point2d p1 = new Point2d();

Console.WriteLine(p1);

Point2d p2 = new Point2d(1, 1);

Console.WriteLine(p2);

Rect rect = new Rect { TopLeft = p1, Length = 2, Width = 3 };

Console.WriteLine(rect);

Circle circle = new Circle(p2, 2);

Console.WriteLine(circle);

}

}

2．单例类

public class Singleton

{

private static Singleton instance;

private Singleton() { }

public static Singleton GetInstance()

{

if(instance == null)

{

instance = new Singleton();

}

return instance;

}

}

class Test

{

public static void Main(string[] argv)

{

Singleton s1 = Singleton.GetInstance();

Singleton s2 = Singleton.GetInstance();

Console.WriteLine(s1 == s2);

}

}

3.

namespace Exp2\_20170407

{

class Person : IComparable

{

public string Name { get; set; }

public int Salary { get; set; }

public DateTime Birthday { get; set; }

public int CompareTo(object p)

{

Person op = p as Person;

return Name.CompareTo(op.Name);

}

public override string ToString()

{

return string.Format("{0} : Salary = {1}; Birthday = {2}", Name, Salary, Birthday);

}

}

class PersonComparer : IComparer

{

public int Type { get; set; }

public PersonComparer(int type)

{

Type = type;

}

public int Compare(object a, object b)

{

Person pa = a as Person;

Person pb = b as Person;

if (Type == 0)

return pa.Salary - pb.Salary;

else

return pa.Birthday.CompareTo(pb.Birthday);

}

}

public class Ex2\_4

{

public static void Test()

{

Person[] persons = new Person[] {

new Person { Name = "Damon", Salary = 10, Birthday = new DateTime(1990, 5, 1) },

new Person { Name = "Niki", Salary = 15 , Birthday = new DateTime(1995, 10, 4) },

new Person { Name = "Ayrton", Salary = 12 , Birthday = new DateTime(1992, 6, 23) },

new Person { Name = "Graham", Salary = 13 , Birthday = new DateTime(1994, 9, 15) }

};

Console.WriteLine("Order by name:");

Array.Sort(persons);

foreach (var p in persons)

Console.WriteLine(p);

Console.WriteLine();

Console.WriteLine("Order by salary:");

Array.Sort(persons, new PersonComparer(0));

foreach (var p in persons)

Console.WriteLine(p);

Console.WriteLine();

Console.WriteLine("Order by birthday:");

Array.Sort(persons, new PersonComparer(1));

foreach (var p in persons)

Console.WriteLine(p);

Console.WriteLine();

}

}

}

实验三

1. 创建一个集合类Stuff，它是Employee类的集合

public class Employee

{

public string Name { get; set; }

public int Salary { get; set; }

public DateTime Birthday { get; set; }

public override string ToString()

{

return string.Format("Employee({0} : Salary = {1}; Birthday = {2:D})", Name, Salary, Birthday);

}

}

public class StuffDict : DictionaryBase

{

public void Add(Employee employee)

{

Dictionary.Add(employee.Name, employee);

}

public void Remove(Employee employee)

{

Dictionary.Remove(employee.Name);

}

public Employee this[string name]

{

get

{

return (Employee)Dictionary[name];

}

set

{

Dictionary[name] = value;

}

}

public Employee GetOldest()

{

Employee oldest = null;

foreach(Employee emp in Dictionary.Values)

{

if (oldest == null)

{

oldest = emp;

}

else if(oldest.Birthday.CompareTo(emp.Birthday) > 0)

{

oldest = emp;

}

}

return oldest;

}

}

public class StuffList

{

List<Employee> emps = new List<Employee>();

public void Add(Employee employee)

{

emps.Add(employee);

}

public void Remove(Employee employee)

{

emps.Remove(employee);

}

public Employee this[string name]

{

get

{

foreach(Employee emp in emps)

{

if (emp.Name == name)

return emp;

}

return null;

}

set

{

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

{

if (emps[i].Name == name)

emps[i] = value;

}

}

}

public Employee GetOldest()

{

Employee oldest = null;

foreach (Employee emp in emps)

{

if (oldest == null)

{

oldest = emp;

}

else if (oldest.Birthday.CompareTo(emp.Birthday) > 0)

{

oldest = emp;

}

}

return oldest;

}

}

2.改写Employee类

public class Employee2

{

public string Name { get; set; }

public int? Salary { get; set; }

public DateTime? Birthday { get; set; }

public Employee2(string name)

{

Name = name;

}

public Employee2(string name, int salary, DateTime birthday)

{

Name = name;

Salary = salary;

Birthday = birthday;

}

public override string ToString()

{

string str = string.Format("Employee({0}", Name);

if (Salary.HasValue)

str += string.Format(" : Salary = {0}; ", Salary);

if (Birthday.HasValue)

str += string.Format("Birthday = {0:D}", Birthday);

str += ")";

return str;

}

}

3．

class Test

{

public static void Main(string[] argv)

{

ArrayList al = new ArrayList();

Random r = new Random();

for (int i = 0; i < 20; i++)

{

al.Add(r.Next(10));

}

Console.WriteLine("初始：");

PrintList(al);

while (al.IndexOf(5) >= 0)

{

al.Remove(5);

}

Console.WriteLine("删除5后：");

PrintList(al);

ArrayList al2 = new ArrayList();

for (int i = 0; i < 3; i++)

{

al.Insert(3, r.Next(10));

}

Console.WriteLine("加三个数后：");

PrintList(al);

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

{

int value = (int)al[i];

if (value == 1)

al[i] = 10;

}

Console.WriteLine("以10代1后：");

PrintList(al);

}

public static void PrintList(ArrayList al)

{

foreach (int n in al)

{

Console.Write(n + "\t");

}

Console.WriteLine();

}

4.

class Test

{

public static void Main(string[] argv)

{

string str = @" class Test

{

public static void Main(string[] argv)

{

ArrayList al = new ArrayList();

Random r = new Random();

for (int i = 0; i < 20; i++)

{

al.Add(r.Next(10));

}

Console.WriteLine(""初始：\"");

PrintList(al);

while (al.IndexOf(5) >= 0)

{

al.Remove(5);

}

Console.WriteLine(""删除5后："");

PrintList(al);

ArrayList al2 = new ArrayList();

for (int i = 0; i < 3; i++)

{

al.Insert(3, r.Next(10));

}

Console.WriteLine(""加三个数后："");

PrintList(al);

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

{

int value = (int)al[i];

if (value == 1)

al[i] = 10;

}

Console.WriteLine(""以10代1后："");

PrintList(al);

}

public static void PrintList(ArrayList al)

{

foreach (int n in al)

{

Console.Write(n + ""\t"");

}

Console.WriteLine();

}

";

string[] keys = { "Console", "static", "foreach", "int", "if" };

Dictionary<string, int> counts = new Dictionary<string, int>();

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

{

int n1 = str.IndexOf(keys[i]);

int count = 0;

while (n1 > 0)

{

count++;

n1 = str.IndexOf(keys[i], n1 + 1);

}

counts[keys[i]] = count;

}

foreach(var pair in counts)

{

Console.WriteLine("{0}:\t{1}", pair.Key, pair.Value);

}

}

实验四

1. 编写一个静态类MyExtensions，扩展.NET Framework基本类型的功能。

namespace Exp2\_20170407

{

static class MyExtensions

{

public static string Reverse(string str)

{

string result = "";

foreach(char c in str)

{

result.Insert(0, c.ToString());

}

return result;

}

public static bool IsPalindrome(this string str)

{

for (int i = 0; i < str.Length / 2; i++)

{

if (str[i] != str[str.Length - i - 1])

return false;

}

return true;

}

//字符串方法或代数方法实现

public static int ReverseDigits(this int number)

{

int result = 0;

while (true)

{

int i = number % 10;

result += i;

number /= 10;

if (number == 0)

break;

result \*= 10;

}

return result;

}

}

public class Ex2\_1

{

public static void Main()

{

string s = "abc";

Console.WriteLine(s + " is " + (s.IsPalindrome()?"":"not ") + "palindrome");

s = "abcba";

Console.WriteLine(s + " is " + (s.IsPalindrome() ? "" : "not ") + "palindrome");

int i = 1234;

Console.WriteLine("Reverse of " + i + " is " + i.ReverseDigits());

}

}

}

1. 编写Lambda表达式

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

Func<int, double> f2 = (a) => Math.Sqrt(a);

Console.WriteLine(f2(16));

Func<int, int, double> f3 = (a, b) => { return Math.Sqrt(a \* a + b \* b); };

Console.WriteLine(f3(6, 8));

Func<String, int> f4 = x => x.Length;

Console.WriteLine(f4("Hello"));

Action<String> a1 = x => Console.WriteLine(x);

a1("Hello");

List<int> list = new List<int> { 1, 2, 3 };

Action<int, List<int>> ActionList = (a, lst) =>

{

foreach (int c in list)

{

a += c;

}

Console.WriteLine(a);

};

ActionList(10, list);

Predicate<int> pred = x =>

{

if (x > 0)

{

Console.WriteLine("{0}为正数", x);

return true;

}

Console.WriteLine("{0}为负数", x);

return false;

};

pred(-8); pred(12);

Predicate<String> p1 = x => x.StartsWith("A");

Console.WriteLine(p1("Apple"));

}

}

}

1. 利用LINQ技术查询Racer对象

IList <Racer> racers = Racer.GetChampions();

var query = from r in racers

where r.Country == "Brazil"

orderby r.Wins descending

select r;

foreach (Racer r in query)

{

Console.WriteLine("{0:A}", r);

}

Console.WriteLine();

query = racers.Where(r => r.Country == "Brazil")

.OrderByDescending(r => r.Wins);

foreach (Racer r in query)

{

Console.WriteLine("{0:A}", r);

}

Console.WriteLine();

var query2 = from r in racers

where r.Wins > 25

orderby r.Wins descending

select r;

foreach (Racer r in query2)

{

Console.WriteLine("{0:A}", r);

}

Console.WriteLine();

foreach (Racer r in racers)

{

if (r.Country == "UK")

r.Country = "United Kingdom";

}

foreach (Racer r in query2)

{

Console.WriteLine("{0:A}", r);

}

Console.WriteLine();

var query3 = from r in racers

where r.Starts >= 100 && r.Wins >= 20

select r;

foreach (Racer r in query3)

{

Console.WriteLine("{0:A}", r);

}

Console.WriteLine();

var query4 = from r in racers

where r.Country == "Germany"

select r.Wins;

Console.WriteLine(query4.Sum());

Console.WriteLine();

var query5 = from r in racers

select new

{

Name = r.FirstName + " " + r.LastName,

Ratio = r.Wins / r.Starts

};

foreach (var value in query5)

{

Console.WriteLine(value);

}

实验5

1. 要求用户输入一个目录名或文件名。

class Test

{

public static void Main(string[] argv)

{

string path = Console.ReadLine();

if (Directory.Exists(path))

{

string[] dirs = Directory.GetDirectories(path);

foreach (string dir in dirs)

{

Console.WriteLine("[{0}]", dir);

}

string[] files = Directory.GetFiles(path);

foreach (string file in files)

{

OutputFile(file);

}

}

}

private static void OutputFile(string file)

{

string[] suffix = { "B", "K", "M", "G", "T" };

FileInfo fi = new FileInfo(file);

long size = fi.Length;

int index = 0;

while(size >= 1024)

{

size = size >> 10;

index++;

}

Console.WriteLine(fi.Name + " : " + size + suffix[index]);

/\* if (size < 1024)

{

Console.WriteLine(fi.Name + ":" + size + "B");

return;

}

size = size >> 10;

if (size < 1024)

{

Console.WriteLine(fi.Name + ":" + size + "K");

return;

}

size = size >> 10;

if (size < 1024)

{

Console.WriteLine(fi.Name + ":" + size + "M");

return;

}

size = size >> 10;

if (size < 1024)

{

Console.WriteLine(fi.Name + ":" + size + "G");

return;

}\*/

}

1. 在Code目录下存放着一些文件。遍历该目录，统计其中各个接口的出现次数，保存在SortedDictionary<string, int>中。现假定接口名称都如IList、ICacheManager（以I字符开头，第二字符大写）。

class Test

{

public static void Main(string[] argv)

{

string path = Console.ReadLine();

if (Directory.Exists(path))

{

string[] dirs = Directory.GetDirectories(path);

foreach (string dir in dirs)

{

Console.WriteLine("[{0}]", dir);

}

string[] files = Directory.GetFiles(path);

foreach (string file in files)

{

OutputFile(file);

}

}

}

private static void OutputFile(string file)

{

string[] suffix = { "B", "K", "M", "G", "T" };

FileInfo fi = new FileInfo(file);

long size = fi.Length;

int index = 0;

while(size >= 1024)

{

size = size >> 10;

index++;

}

Console.WriteLine(fi.Name + " : " + size + suffix[index]);

/\* if (size < 1024)

{

Console.WriteLine(fi.Name + ":" + size + "B");

return;

}

size = size >> 10;

if (size < 1024)

{

Console.WriteLine(fi.Name + ":" + size + "K");

return;

}

size = size >> 10;

if (size < 1024)

{

Console.WriteLine(fi.Name + ":" + size + "M");

return;

}

size = size >> 10;

if (size < 1024)

{

Console.WriteLine(fi.Name + ":" + size + "G");

return;

}\*/

}

1. 写一个记事本程序：
2. 窗口可用DockPanel进行布局，让菜单和工具栏都位于顶部，即：

DockPanel.Dock="Top"。

1. 文本文件的编辑可以使用TextBox控件。
2. 使用命令绑定，让菜单项和工具栏同时与一个操作相关联。

在MainWindow.xaml的Window标签下加：

<Window.CommandBindings>

<CommandBinding Command="ApplicationCommands.New" Executed="NewCommand\_Executed"/>

<CommandBinding Command="ApplicationCommands.Open" Executed="OpenCommand\_Executed"/>

<CommandBinding Command="ApplicationCommands.Save" Executed="SaveCommand\_Executed"/>

</Window.CommandBindings>

在菜单项添加：

<MenuItem Header="新建(\_N)" Command="New"/>

在工具栏添加：

<Button Content="新建" Command="New"/>

就可绑定命令。同时Ctrl+O等键盘组合也默认与Open命令相绑定。

1. 添加bool类型\_saved字段，标记当前内容是否已保存。
2. 打开文件时，弹出打开文件对话框，操作代码如下：

OpenFileDialog dlg = new OpenFileDialog();

dlg.DefaultExt = "\*.txt";

dlg.Filter = "Text Files (\*.txt)|\*.txt";

bool? result = dlg.ShowDialog();

if (result == true)

{

string fileName = dlg.FileName;

自此可对该文件名进行操作。

private void NewCommand\_Executed(object sender, ExecutedRoutedEventArgs e)

{

MessageBox.Show("New");

}

private void OpenCommand\_Executed(object sender, ExecutedRoutedEventArgs e)

{

OpenFileDialog dlg = new OpenFileDialog();

dlg.DefaultExt = "\*.txt";

dlg.Filter = "Text Files (\*.txt)|\*.txt|Word Files (\*.docx)|\*.docx";

bool? result = dlg.ShowDialog();

if (result == true)

{

string fileName = dlg.FileName;

StreamReader sr = new StreamReader(fileName);

txtEdit.Text = sr.ReadToEnd();

}

}

private void SaveCommand\_Executed(object sender, ExecutedRoutedEventArgs e)

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "文本文件|\*.txt|所有文件|\*.\*";

saveFileDialog.FilterIndex = 0;

bool? result = saveFileDialog.ShowDialog();

if (result == true)

{

string strFile = saveFileDialog.FileName;

StreamReader sr = new StreamReader(strFile);

txtEdit.Text = sr.ReadToEnd();

}

}

}

}

XAML代码：

<Window.CommandBindings>

<CommandBinding Command="ApplicationCommands.New" Executed="NewCommand\_Executed"/>

<CommandBinding Command="ApplicationCommands.Open" Executed="OpenCommand\_Executed"/>

<CommandBinding Command="ApplicationCommands.Save" Executed="SaveCommand\_Executed"/>

</Window.CommandBindings>

<DockPanel >

<Menu DockPanel.Dock="Top">

<MenuItem Header="文件(\_F)">

<MenuItem Header="新建(\_N)" Command="New" />

<MenuItem Header ="打开(\_O)" Command="Open"/>

<MenuItem Header ="保存(\_S)" Command="Save"/>

</MenuItem>

</Menu >

<ToolBar DockPanel.Dock="Top">

<Button Command="New">新建</Button>

<Button Command="Open">打开</Button>

<Button Command="Save">保存</Button>

</ToolBar>

<TextBox DockPanel.Dock="Bottom" Name="txtEdit"/>

</DockPanel >

</Window>