实验一

1. 尝试用命令行方式编译一个简单的控制台程序为可执行PE文件。

(1) 先用“记事本”编写一个简单的控制台应用程序p1.cs，该程序中只包含可供测试的几行简单代码，例如为：

using System;

using System.Threading;

class test

{

static void Main()

{

int x, y;

x=Int32.Parse(Console.ReadLine());

y=Int32.Parse(Console.ReadLine());

Console.WriteLine("{0}+{1}={2}",x, y, x+y);

Console.ReadLine();

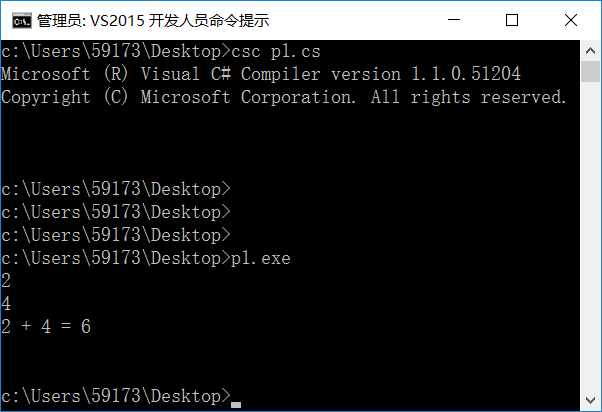
}

}

(2) 使用csc.exe 程序以及适当的参数执行编译命令，产生一个可执行PE程序。

(3) 执行该PE程序，观察是否符合预期。

(4) 观察用以上方法编译时，如果代码中有错误，系统如何输出错误信息？

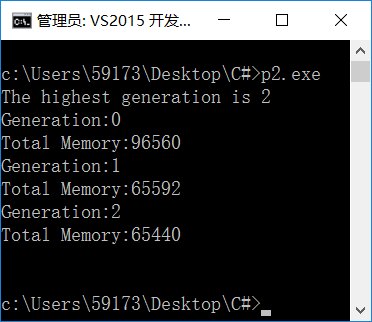


2. 使用 GC类控制 .NET的垃圾回收操作。

(1) 按照教材上【例1-3】，建立C#的控制台应用程序。

(2) 编译调试该程序，观察、记录控制台输出。

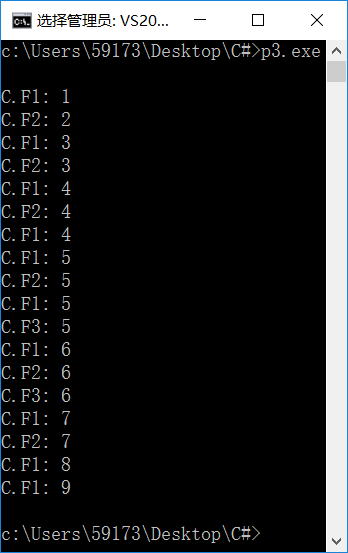
(3) 如果要使MakeSomeGarbage方法中产生的对象不会被回收，应该如何修改 MakeSomeGarbage中的代码？修改后实际运行时，观察程序输出的相关信息是否与预计一致（需要与修改前输出的内存占有数据相比较）。



3. 学习使用委托

(1) 按照教材上【例1-5】，建立C#的控制台应用程序。

(2) 编译调试该程序，观察、记录控制台输出。



\*4. 不要利用Visual Studio.NET IDE环境的“工具箱”、“属性窗口”，最好只使用“笔记本”写代码，编写一个“Hello,World!”的Windows应用程序。

(1) 该Windows程序窗体上有一个按钮。当单击按钮时，弹出对话框显示“Hello,World!”。

(2) 在“命令行窗口”下执行CSC将C#代码文件编译为PE程序。

(3) 尽量用最少的代码编写该程序，并且只用一个源文件。

**提示：**

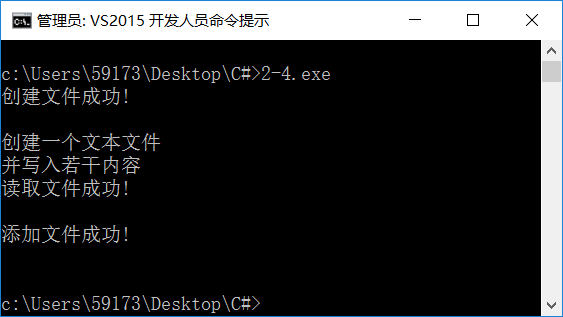
可在Form\_Load等事件中执行 Button button1=new Button(); 创建该按钮，对该按钮，要将其加入到Form的Controls集合中。然后对按钮的Text、Position等属性赋值。最后记得要为单击按钮的事件设置一个方法，详见教材上§1.5.2中有关叙述。

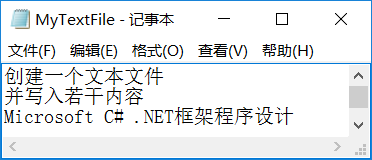
5. 使用StreamReader和StreamWrite类操纵文本文件。

(1) 按照教材上【例2-4】，建立C#的控制台应用程序。

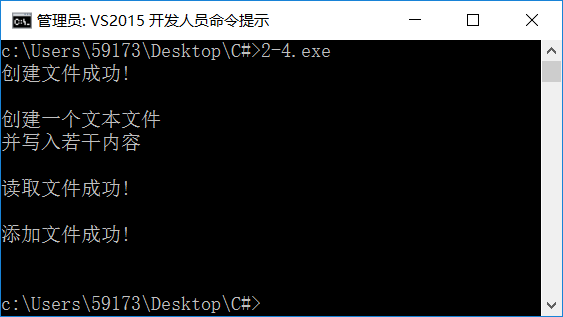
(2) 编译调试该程序，观察、记录控制台输出。（程序中使用的文件路径和名称可调整）

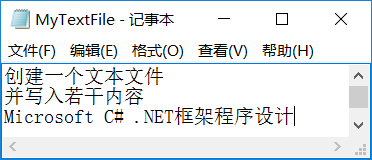
(3) 修改一下程序中代码，将ReadLine 方法替换为使用ReadToEnd方法，但仍然可以输出同样的结果。





将ReadLine替换成为ReadToEnd之后的结果:



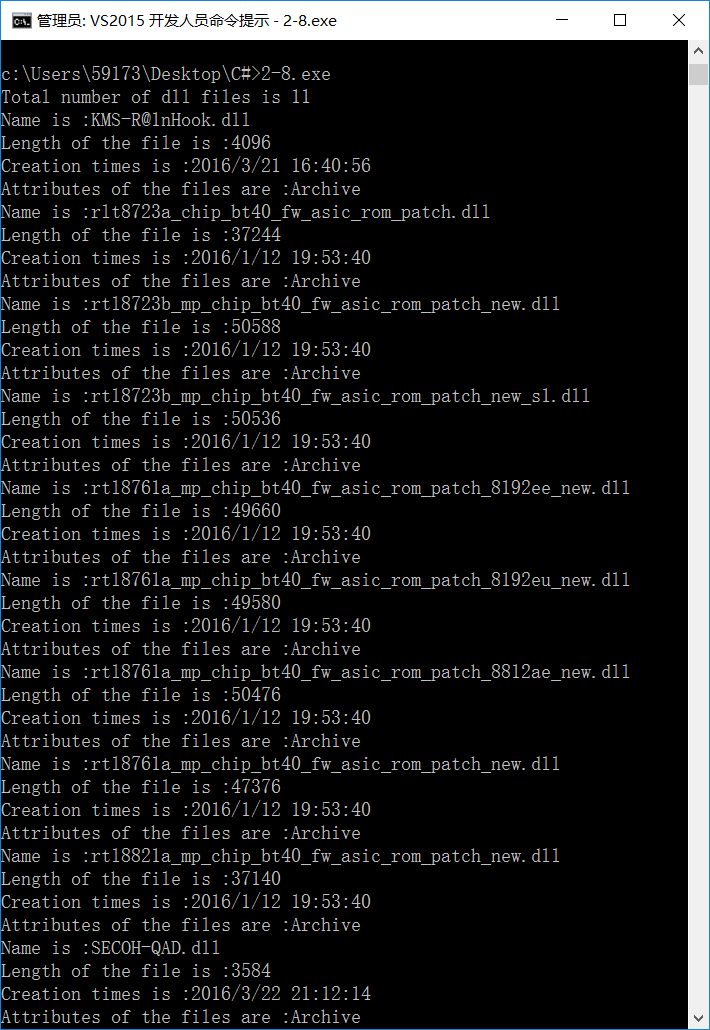


6 使用 DirectoryInfo 类和 FileInfo 类，对文件系统进行操作。

(1) 按照教材上【例2-8】，建立C#的控制台应用程序。

(2) 编译调试该程序，观察、记录控制台输出。（程序中使用的文件夹名称可以酌情调整）

(3) 修改或补充该程序，当指定一个文件夹后，程序可输出该文件夹及其各级子目录内所包含的文件的总数。



搜索给定目录下的所有子目录和文件的个数:

核心代码:

public static void Main()

{

string directory = "c:\\Users\\59173\\Thunder";

Console.WriteLine("The parent directory we search is \n" + directory);

SearchFile(directory);

Console.ReadLine();

}

//查找算法

public static void SearchFile(string directory)

{

int countfile = 0;

int countdir = 0;

DirectoryInfo dir = new DirectoryInfo(directory);

FileSystemInfo[] f = dir.GetFileSystemInfos("\*", SearchOption.AllDirectories);

foreach (FileSystemInfo i in f)

{

if (i is DirectoryInfo)

{

countdir++;

SearchFile(i.FullName);

}

else if (i is FileInfo)

{

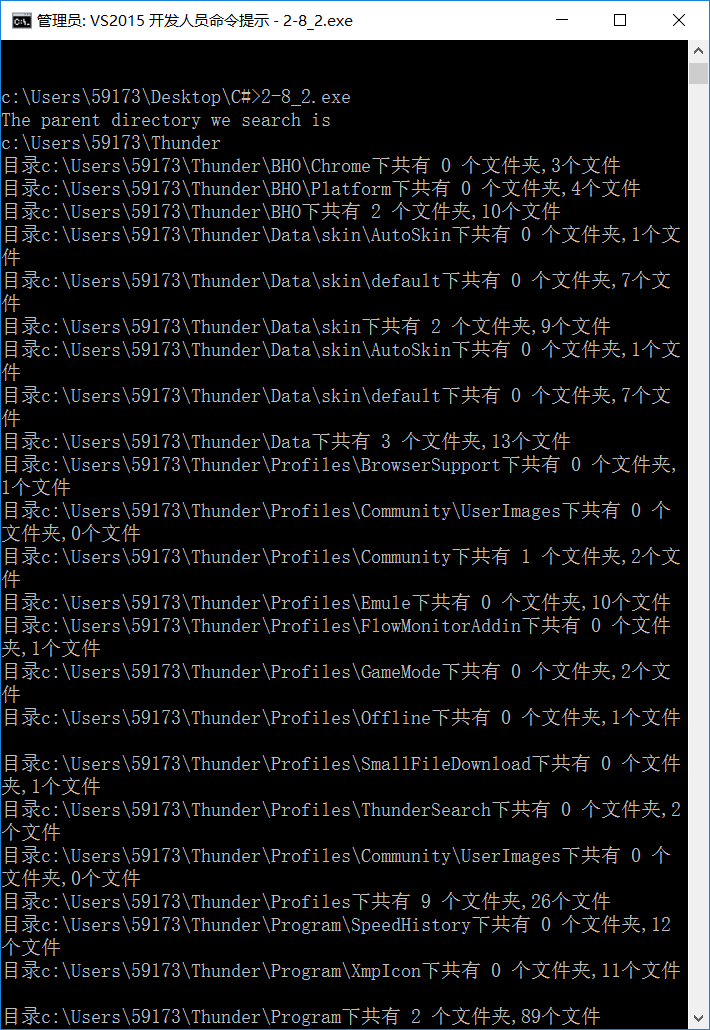
countfile++;

}

}

Console.WriteLine("目录" + directory + "下共有 {0} 个文件夹,{1}个文件", countdir, countfile);

}

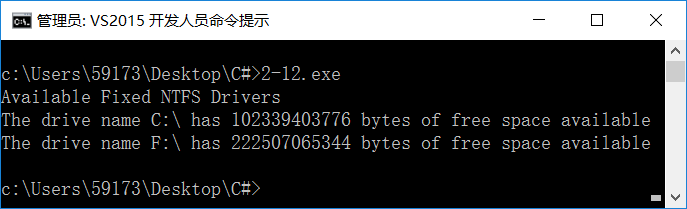


7. 使用DriveInfo类探测本机上各个驱动器。

(1) 按照教材上【例2-12】，建立C#的控制台应用程序。

(2) 编译调试该程序，观察、记录控制台输出。

(3)　修改或补充该程序，使能找到本机上所有U盘中可用空间最大的那一个。



最大的U盘信息:

核心代码:

DriveInfo[] drives = DriveInfo.GetDrives();

Console.WriteLine("Available Fixed NTFS Drivers");

DriveInfo[] rmbdisk = new DriveInfo[10];

int index = 0;

foreach (DriveInfo drive in drives)

{

if (drive.DriveType == DriveType.Removable)

rmbdisk[index++] = drive;

}

rmbdisk[index] = null;

long max = 0;

DriveInfo maxspace = null;

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

{

if (rmbdisk[i].AvailableFreeSpace >= max)

{

max = rmbdisk[i].AvailableFreeSpace;

maxspace = rmbdisk[i];

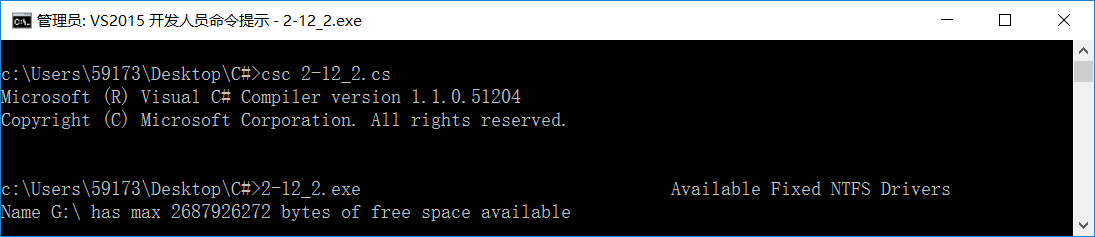
}

}

Console.WriteLine("Name {0} has max {1} bytes of free space available",

maxspace.Name, maxspace.AvailableFreeSpace);

Console.ReadLine();

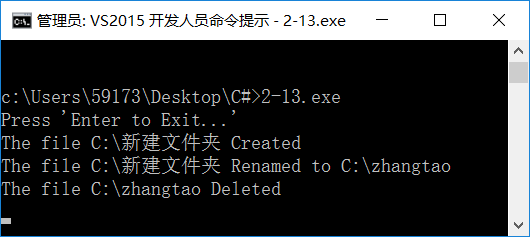


8. 使用FileSystemWatche类，监视C盘根目录上的各种文件操作。

(1) 按照教材上【例2-13】建立C#的控制台应用程序。

(2) 编译调试该程序，记录测试过程。

\*(3) 修改或补充该程序，使得文件改名操作不允许（或每次改名后总能自动改回去）。



9. 写一个实验报告，包含以上各个实验中的关键内容。

对于第六题，主要用到了递归来遍历目录，如果子文件还是目录，则继续遍历，分别用两个变量来记录文件夹的数量和普通文件的数量。在程序结束的时候将详细信息打印出来。

对于第七题中，我主要用到了用一个DriveInfo的数组保存从总的驱动设备信息中筛选出移动设备的编号。然后再分别比较他们剩余的空间大小，选出最大的一个驱动设备。并进行打印。

注：

1. 实验题是按照部分利用课余时间布置的，规定题目量在实验课内完不成属正常现象，希望课外能再抽一点时间去完成。
2. 凡带\*的题目（或小题）为选做。
3. 有不会做的题，可以看书、上网查资料或请老师给予指导。同学间也可以互相交流，但不要直接抄作业。除了实验课内找老师提问外，[课外可以用guowyy@126.com](mailto:课外可以用guowyy@126.com)邮箱联系郭老师。
4. 每次实验都要求交一个报告。报告格式没有特别规定，但开头处要写入学号、姓名、班级、课程等相关信息。报告中每个规定的题目都要有所反映，一般可按题目顺序排列。对每个题，要简要叙述一下问题和解题思路、调试过程、实验步骤和结果等。关键的代码片断要粘贴入报告中（如与教材上完全一致的部分，可适当省略）。另外，每个重要运行结果都要使用截屏方法将画面贴入报告（每个题至少一至二幅图片）。如有其它经验体会、或自己觉得有较高技巧的代码等，也可写入报告。
5. 记住，实验报告本质上应该是写给自己（或团队内部成员）看的，交给老师只是便于督促和改进。为了对自己负责，请不要以消极应付的态度来对待。