Day1

using System;

namespace HelloWorldApplication

{

class HelloWorld

{

static void Main(string[] args)

{

/\* 我的第一个 C# 程序\*/

Console.WriteLine("Hello World");

Console.ReadKey();

}

}

}

这是一个Hello World程序

System是命名空间，是本程序用到的“库”

（声明**namespace**是一系列的类**，HelloWorldApplication**命名空间包含了类**HelloWorld**

（声明class，类HelloWorld包含了程序使用的**数据**和**方法**声明。类一般包含多个方法，方法=行为，有一个是Main方法

（定义main方法，是所有C#程序的**入口点**。Main方法说明当执行时，类将做什么动作。

Console.WriteLine(“Hello World!”);指定了类的**具体行为**。它是Console类的一个方法。

类比理解：

**类**像是一个人，方法是他能做的事。他可以让别的人（类）帮他做事（调用其他类的方法）。**命名空间**（避免对象来自不同地方，但名字相同引起的错误）是不同的班级，每个班都可以有个小明，对本班人来说不会引起误解。只需要在喊他之前说明喊的是几班的小明就可以了。

扩展阅读：

关于namespace：（C++的规范写法问题）

1. <iostream>和<iostream.h>格式不一样，前者没有后缀，实际上，在你的编译器include文件夹里面可以看到，二者是两个文件，打开文件就会发现，里面的代码是不一样的。后缀为.h的头文件c++标准已经明确提出不支持了，早些的实现将标准库功能定义在全局空间里，声明在带.h后缀的头文件里，c++标准为了和C区别开，也为了正确使用命名空间，规定头文件不使用后缀.h。 因此，当使用<iostream.h>时，相当于在c中调用库函数，使用的是全局命名空间，也就是早期的c++实现；当使用<iostream>的时候，该头文件没有定义全局命名空间，必须使用namespace std；这样才能正确使用cout。

也就是说<iostream>是不包含namespace std中的内容的，而<iostream.h>是包含的，所以使用.h的话不写using namespace std也可以正确调用cout。

1. 所谓命名空间，是一种将程序库名称封装起来的方法，它就像在各个程序库中立起一道道围墙。

这一段实际上是C++的内容，但是我一直没有理解到位，贴下代码：

//DISPLAYNamespaceDemonstration

#include <iostream>

using namespace std ;

namespace savitch1

{

    void greeting();

}

namespace savitch2

{

    void greeting();

}

void big\_greeting();

int main()

{

    {

        using namespace savitch2 ;

        //使用savictch2、std、全局三个命名空间

        greeting();

    }

    {

        using namespace savitch1 ;

        //使用savitch1、std、全局三个命名空间

        greeting();

    }

    big\_greeting();

    //使用了std和全局两个命名空间

    return0 ;

}

namespace savitch1

{

    void greeting()

    {

        cout<<"Hellofromnamespacesavitch1.\n" ;

    }

}

namespace savitch2

{

    void greeting()

    {

        cout<<"Greetingsfromnamespacesavitch2.\n" ;

    }

}

void big\_greeting()

{

    cout<<"ABigGlobalHello!\n" ;

}

Day2

**C#基本语法**

任何C#程序中的第一条语句都是 using System;

Class关键字用来声明一个类

**成员变量&成员函数**：变量是类的属性/数据成员，用于存储数据；函数是执行任务的语句。

**实例化**：Rectangle r = new Rectangle();

**标识符**：

1. 第一个字符不是数字，只能由字母、下划线、数字组成
2. 不包含空格&符号
3. 不能是C#关键字

**关键字**：

编译器预定义的保留字。@关键字可作为标识符。

**保留**关键字/**上下文**关键字（提供特定含义，但不是保留字，有的在多个上下文中有特殊含义）

**数据类型**

* 值类型（Value types）可直接分配给一个值bool/byte/char/decimal/double/float/int等
* 引用类型（Reference types）对象/指针
* 指针类型（Pointer types）

**引用-对象（Object）类型**

所有数据类型的**终极基类**。值类型->对象类型：**装箱**；对象类型->值类型：**拆箱。**

object obj;

obj = 100; // 这是装箱

**引用-动态（Dynamic）类型**

可存储任何类型的值在动态数据类型变量中。类型检查在运行时发生。

声明动态类型：

dynamic <variable\_name> = value;

eg：

dynamic d = 20;

动态类型与对象类型相似，但是对象类型变量的类型检查是在编译时发生的，而动态类型变量的类型检查是在运行时发生的。

**引用-字符串（String）类型**

两种形式：**引号**和 **@引号**（逐字字符串，将转义字符\当做普通字符）

**eg：**

String str = "w3cschool.cc";

@"w3cschool.cc";

string str = @"C:\Windows";

**C#类型转换**

隐式：安全的，小类型->大类型，如小的整数类型->大的整数类型，派生类->基类

显式：用户使用显示函数（强制转换运算符）

i = (int)d;

**C#类型转换方法**

**ToBoolean/ToChar/ToDouble等，调用时使用：变量.方法（）即可。这个用过很多次了。**

Console.WriteLine(i.ToString());