11. (简答题, 50分)按要求编写Lambda表达式：

.NET已定义的Func<out TResult>是一个泛型委托，它封装一个方法，该方法不具有参数，且返回由 TResult 参数指定的类型的值。它的使用，举一例如下：

Func<int> f1 = () => DateTime.Now.Year;

//Lambda表达式无参数，返回int值，符合委托定义

Console.WriteLine(f1());

该泛型委托有好几个同名兄弟类型，如Func<in T, out TResult>，Func <in T1, in T2, out TResult>等（直到T16）等，仿照上面的例子编写Lambda表达式，并调用它们：

（1）写一个Lambda表达式赋给Func<int, double> f2，使其完成取平方根功能（可用Math.Sqrt方法）。

（2）写一个Lambda表达式赋给Func<int, int, double> f3，使两个整数参数先平方，再相加，再取平方根。

（3）Predicate<T>泛型委托代表一类函数，它们对参数实施判断并返回bool值，实际上表示一种条件。写一个Predicate<int>型Lambda表达式，并调用它。

（4）Action、Action<T>、Action<T1,T2>…等委托封装的方法，有0至16个参数，但没有返回值，试写出Action<String>和Action<int, List<int>>的Labmda表达式，并调用它们。

（调用它们时，如果需要参数，则自行生成；可以和Java中的Lambda表达式比较一下）

**解答**

（1）编写Lambda表达式并赋给Func<int, double>，使其完成取平方根功能：

|  |  |
| --- | --- |
|  | Func<int, double> f2 = x => Math.Sqrt(x); |
|  | Console.WriteLine(f2(9)); // 输出3.0 |

（2）编写Lambda表达式并赋给Func<int, int, double>，使两个整数参数先平方，再相加，再取平方根：

|  |  |
| --- | --- |
|  | Func<int, int, double> f3 = (x, y) => Math.Sqrt(x \* x + y \* y); |
|  | Console.WriteLine(f3(3, 4)); // 输出5.0 |

（3）编写Predicate<int>型Lambda表达式，并调用它：

|  |  |
| --- | --- |
|  | Predicate<int> isEven = x => x % 2 == 0; |
|  | Console.WriteLine(isEven(4)); // 输出True |
|  | Console.WriteLine(isEven(5)); // 输出False |

（4）编写Action<String>和Action<int, List<int>>的Lambda表达式，并调用它们：

|  |  |
| --- | --- |
|  | Action<string> printString = s => Console.WriteLine(s); |
|  | printString("Hello, World!"); // 输出Hello, World! |
|  |  |
|  | Action<int, List<int>> addToList = (number, list) => list.Add(number); |
|  | List<int> numbers = new List<int>(); |
|  | addToList(5, numbers); |
|  | addToList(10, numbers); |
|  | Console.WriteLine(string.Join(", ", numbers)); // 输出5, 10 |

Lambda表达式在C#中是一种非常简洁且强大的方式，用于表示匿名函数。它们可以直接赋值给委托类型的变量，并在需要时调用。在上面的例子中，Lambda表达式被用来创建符合特定委托类型签名的函数，这些函数可以执行各种操作，如计算平方根、判断奇偶性、打印字符串以及向列表中添加元素等。

与Java中的Lambda表达式相比，C#的Lambda表达式在语法上略有不同，但基本概念和用途是相似的。两者都允许以简洁的方式表示匿名函数，并作为参数传递给其他方法或赋值给变量。