函数

2021年11月2日 16:02

1、函数每个形参的类型都要写出来。即使某个形参不被使用，也必须为它提供一个实参。

2、static：在定义局部变量时前面加上static可定义为局部静态对象。

局部静态对象在程序的执行路径第一次经过对象定义语句时初始化，并且直到程序终止才被销毁，在此期间即使对象所在的函数结束执行也不会对它有影响。

3、函数声明也称作函数原型。函数应该在头文件中声明而在源文件中定义。

4、如果函数无需改变引用形参的值，强烈建议将其声明为常量引用。

尽量使用常量引用

> 把函数不会改变的形参定义成（普通的）引用是一种比较常见的错误，这么做带给函数的调用者一种误导，即函数可以修改它的实参的值。此外，使用引用而非常量引用也会极大地限制函数所能接受的实参类型。就像刚刚看到的，我们不能把 const 对象、字面值或者需要类型转换的对象传递给普通的引用形参。

5、 int &arr[10]; 将arr声明成引用的数组

int (&arr)[10]; arr是具有10个整数的整型数组的引用

int *matrix[10]; 10个指针构成的数组

int (*matrix)[10]; 指向含有10个整数的数组的指针

6、主函数 int main(int argc, char* argv[]) {...}

第一个形参argc代表数组中字符串的数量。第二个形参argv是一个数组，它的元素是指向C风格字符串的指针，即argv指向char*。

当使用argv中的实参时，一定要记得可选的实参从argv[1]开始；argv[0]保存程序的名字，而非用户输入。

7、在含有return语句的循环后面也应该有一条return语句，如果没有的话该程序就是错误的。很多编译器都无法发现此类错误。

8、不要返回局部对象的引用或指针。main函数不能调用它自己。

9、

引用返回左值

函数的返回类型决定函数调用是否是左值（参见 4.1.1 节，第 121 页）。调用一个返回引用的函数得到左值，其他返回类型得到右值。可以像使用其他左值那样来使用返回引用的函数的调用，特别是，我们能为返回类型是非常量引用的函数的结果赋值：

```
char &get_val(string &str, string::size_type ix)
{
    return str[ix];              // get_val 假定索引值是有效的
}
int main()
{
    string s("a value");
    cout << s << endl;           // 输出 a value
    get_val(s, 0) = 'A';         // 将 s[0]的值改为 A
    cout << s << endl;           // 输出 A value

    return 0;
}
```

10、声明一个返回数组指针的函数：

举个具体点的例子，下面这个 func 函数的声明没有使用类型别名：

```
int (*func(int i))[10];
```

可以按照以下的顺序来逐层理解该声明的含义：

- func(int i)表示调用 func 函数时需要一个 int 类型的实参。
- (*func(int i))意味着我们可以对函数调用的结果执行解引用操作。
- (*func(int i))[10]表示解引用 func 的调用将得到一个大小是 10 的数组。
- int (*func(int i))[10]表示数组中的元素是 int 类型。

使用尾置返回类型

在 C++11 新标准中还有一种可以简化上述 func 声明的方法，就是使用**尾置返回类型**（trailing return type）。任何函数的定义都能使用尾置返回，但是这种形式对于返回类型比较复杂的函数最有效，比如返回类型是数组的指针或者数组的引用。尾置返回类型跟在形参列表后面并以一个->符号开头。为了表示函数真正的返回类型跟在形参列表之后，我们在本应该出现返回类型的地方放置一个 auto：

```
// func 接受一个 int 类型的实参，返回一个指针，该指针指向含有 10 个整数的数组
auto func(int i) -> int(*)[10];
```

因为我们把函数的返回类型放在了形参列表之后，所以可以清楚地看到 func 函数返回的是一个指针，并且该指针指向了含有 10 个整数的数组。

使用decltype

还有一种情况，如果我们知道函数返回的指针将指向哪个数组，就可以使用 decltype 关键字声明返回类型。例如，下面的函数返回一个指针，该指针根据参数 i 的不同指向两个已知数组中的某一个：

```
int odd[] = {1,3,5,7,9};
int even[] = {0,2,4,6,8};
// 返回一个指针，该指针指向含有 5 个整数的数组
decltype(odd) *arrPtr(int i)
{
    return (i % 2) ? &odd : &even; // 返回一个指向数组的指针
}
```

arrPtr 使用关键字 decltype 表示它的返回类型是个指针，并且该指针所指的对象与 odd 的类型一致。因为 odd 是数组，所以 arrPtr 返回一个指向含有 5 个整数的数组的指针。有一个地方需要注意：decltype 并不负责把数组类型转换成对应的指针，所以 decltype 的结果是个数组，要想表示 arrPtr 返回指针还必须在函数声明时加一个*符号。

10、如果同一作用域内的几个函数名字相同但形参列表不同（形参数量或形参类型不同），我们称之为重载函数。（注意是形参类型不是形参名字）

main函数不能重载。

11、重载和const形参

如 6.2.3 节（第 190 页）介绍的，顶层 const（参见 2.4.3 节，第 57 页）不影响传入函数的对象。一个拥有顶层 const 的形参无法和另一个没有顶层 const 的形参区分开来：

```
Record lookup(Phone);
Record lookup(const Phone);      // 重复声明了 Record lookup(Phone)

Record lookup(Phone*);
Record lookup(Phone* const);     // 重复声明了 Record lookup(Phone*)
```

在这两组函数声明中，每一组的第二个声明和第一个声明是等价的。

另一方面，如果形参是某种类型的指针或引用，则通过区分其指向的是常量对象还是非常量对象可以实现函数重载，此时的 const 是底层的：

```
// 对于接受引用或指针的函数来说，对象是常量还是非常量对应的形参不同
// 定义了 4 个独立的重载函数
Record lookup(Account&);         // 函数作用于 Account 的引用
Record lookup(const Account&);   // 新函数，作用于常量引用

Record lookup(Account*);         // 新函数，作用于指向 Account 的指针
Record lookup(const Account*);   // 新函数，作用于指向常量的指针
```

在上面的例子中，编译器可以通过实参是否是常量来推断应该调用哪个函数。因为 const 不能转换成其他类型（参见 4.11.2 节，第 144 页），所以我们只能把 const 对象（或指向 const 的指针）传递给 const 形参。相反的，因为非常量可以转换成 const，所以上面的 4 个函数都能作用于非常量对象或者指向非常量对象的指针。不过，如 6.6.1 节（第 220 页）将要介绍的，当我们传递一个非常量对象或者指向非常量对象的指针时，编译器会优先选用非常量版本的函数。

12、在不同的作用域中无法重载函数名。因为如果我们在内层作用域中声明名字，它将隐藏外层作用域中声明的同名实体。
在C++语言中，名字查找发生在类型检查之前。

13、默认实参
我们可以为一个或多个形参定义默认值，不过需要注意的是，一旦某个形参被赋予了默认值，它后面的所有形参都必须有默认值。
尽量让不怎么使用默认值的形参出现在前面，而让那些经常使用默认值的形参出现在后面。
在给定的作用域中，一个形参只能被赋予一次默认实参。使用默认实参的方法只需要在调用函数时省略该实参就可以了。
通常，应该在函数声明中指定默认实参，并将该声明放在合适的头文件中。

14、内联函数
将函数指定为内联函数（inline），通常就是将它在每个调用点上“内联地”展开。内联函数可避免函数调用的开销。
在函数的返回类型前面加上关键字inline，就可以将函数声明为内联函数。一般来说，内联机制用于优化规模较小、流程直接、频繁调用的函数。
内联说明只是向编译器发出的一个请求，编译器可以选择忽略这个请求。

15、constexpr函数
constexpr函数是指能用于常量表达式的函数。定义constexpr函数的方法与其他函数类似，不过要遵循几项约定：函数的返回类型及所有形参的类型都得是字面值类型，而且函数题中必须有且只有一条return语句。
constexpr函数不一定返回常量表达式。

16、内联函数和constexpr函数通常定义在头文件中。

17、assert预处理宏
assert是一种预处理宏，所谓预处理宏其实是一个预处理变量，它的行为有点类似于内联函数。
assert宏使用一个表达式作为它的条件：assert（expr）；首先对expr求值，如果表达式为假，assert输出信息并终止程序的运行。如果表达式为真，assert什么也不做。
assert宏定义在cassert头文件中。和预处理变量一样，宏名字在程序内必须唯一。含有cassert头文件的程序不能再定义名为assert的变量、函数或者其他实体。
除了assert之外，现代C++程序很少再使用预处理宏了。

18、NDEBUG预处理变量
assert的行为依赖于一个名为NDEBUG的预处理变量的状态。如果定义了NDEBUG，则assert什么也不做。默认状态下没有定义NDEBUG，此时assert将执行运行时检查。
我们可以使用一个#define语句定义NDEBUG，从而关闭调试状态。

19、函数指针
//pf只想一个函数，该函数的参数是两个const string的引用，返回值是bool类型。
bool (*pf)(const string& , const string&);

*pf 两端的括号必不可少。如果不写这对括号，则 pf 是一个返回值为 bool 指针的函数：

```
// 声明一个名为 pf 的函数，该函数返回 bool*
bool *pf(const string &, const string &);
```

函数指针形参

和数组类似（参见 6.2.4 节，第 193 页），虽然不能定义函数类型的形参，但是形参可以是指向函数的指针。此时，形参看起来是函数类型，实际上却是当成指针使用：

```
// 第三个形参是函数类型，它会自动地转换成指向函数的指针
void useBigger(const string &s1, const string &s2,
               bool pf(const string &, const string &));
// 等价的声明：显式地将形参定义成指向函数的指针
void useBigger(const string &s1, const string &s2,
               bool (*pf)(const string &, const string &));
```

我们可以直接把函数作为实参使用，此时它会自动转换成指针：

```
// 自动将函数 lengthCompare 转换成指向该函数的指针
useBigger(s1, s2, lengthCompare);
```

返回指向函数的指针

和数组类似（参见 6.3.3 节，第 205 页），虽然不能返回一个函数，但是能返回指向函数类型的指针。然而，我们必须把返回类型写成指针形式，编译器不会自动地将函数返回类型当成对应的指针类型处理。与往常一样，要想声明一个返回函数指针的函数，最简单的办法是使用类型别名：

```
using F = int(int*, int);          // F 是函数类型，不是指针
using PF = int(*)(int*, int);      // PF 是指针类型
```

其中我们使用类型别名（参见 2.5.1 节，第 60 页）将 F 定义成函数类型，将 PF 定义成指向函数类型的指针。必须时刻注意的是，和函数类型的形参不一样，返回类型不会自动地转换成指针。我们必须显式地将返回类型指定为指针：

```
PF f1(int);            // 正确：PF 是指向函数的指针，f1 返回指向函数的指针
F f1(int);             // 错误：F 是函数类型，f1 不能返回一个函数
F *f1(int);            // 正确：显式地指定返回类型是指向函数的指针
```

当然，我们也能用下面的形式直接声明 f1：

```
int (*f1(int))(int*, int);
```

按照由内向外的顺序阅读这条声明语句：我们看到 f1 有形参列表，所以 f1 是个函数；f1 前面有*，所以 f1 返回一个指针；进一步观察发现，指针的类型本身也包含形参列表，因此指针指向函数，该函数的返回类型是 int。

出于完整性的考虑，有必要提醒读者我们还可以使用尾置返回类型的方式（参见 6.3.3 节，第 206 页）声明一个返回函数指针的函数：

```
auto f1(int) -> int (*)(int*, int);
```

20、牢记当我们将decltype用于某个函数时，它返回函数类型而非指针类型。因此，我们显式地加上*以表明我们需要返回指针，而非函数本身。