← C++ 判断 C++ 数字 →

C++ 函数

函数是一组一起执行一个任务的语句。每个 C++ 程序都至少有一个函数，即主函数 **main()** ，所有简单的程序都可以定义其他额外的函数。

您可以把代码划分到不同的函数中。如何划分代码到不同的函数中是由您来决定的，但在逻辑上，划分通常是根据每个函数执行一个特定的任务来进行的。

函数**声明**告诉编译器函数的名称、返回类型和参数。函数**定义**提供了函数的实际主体。

C++ 标准库提供了大量的程序可以调用的内置函数。例如，函数 **strcat()** 用来连接两个字符串，函数 **memcpy()** 用来复制内存到另一个位置。

函数还有很多叫法，比如方法、子例程或程序，等等。

定义函数

C++ 中的函数定义的一般形式如下：

```
return_type function_name( parameter list )
{
   body of the function
}
```

在 C++ 中，函数由一个函数头和一个函数主体组成。下面列出一个函数的所有组成部分：

返回类型：一个函数可以返回一个值。**return_type** 是函数返回的值的数据类型。有些函数执行所需的操作而不返回值，在这种情况下，return_type 是关键字 **void**。

函数名称：这是函数的实际名称。函数名和参数列表一起构成了函数签名。

参数：参数就像是占位符。当函数被调用时，您向参数传递一个值，这个值被称为实际参数。参数列表包括函数参数的类型、顺序、数量。参数是可选的，也就是说，函数可能不包含参数。

函数主体：函数主体包含一组定义函数执行任务的语句。

实例

以下是 **max()** 函数的源代码。该函数有两个参数 num1 和 num2，会返回这两个数中较大的那个数：

```
// 函数返回两个数中较大的那个数

int max(int num1, int num2)
```

分类导航

HTML / CSS
JavaScript
服务端
数据库
移动端
XML 教程
ASP.NET
Web Service
开发工具
网站建设

```
{
   // 局部变量声明
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

函数声明

函数**声明**会告诉编译器函数名称及如何调用函数。函数的实际主体可以单独定义。

函数声明包括以下几个部分：

```
return_type function_name( parameter list );
```

针对上面定义的函数 max()，以下是函数声明：

```
int max(int num1, int num2);
```

在函数声明中，参数的名称并不重要，只有参数的类型是必需的，因此下面也是有效的声明：

```
int max(int, int);
```

当您在一个源文件中定义函数且在另一个文件中调用函数时，函数声明是必需的。在这种情况下，您应该在调用函数的文件顶部声明函数。

调用函数

创建 C++ 函数时，会定义函数做什么，然后通过调用函数来完成已定义的任务。

当程序调用函数时，程序控制权会转移给被调用的函数。被调用的函数执行已定义的任务，当函数的返回语句被执行时，或到达函数的结束括号时，会把程序控制权交还给主程序。

调用函数时，传递所需参数，如果函数返回一个值，则可以存储返回值。例如：

实例

```
#include <iostream>
using namespace std;

// 函数声明
int max(int num1, int num2);

int main ()
{
   // 局部变量声明
   int a = 100;
   int b = 200;
   int ret;
```

```
   // 调用函数来获取最大值
   ret = max(a, b);

   cout << "Max value is : " << ret << endl;

   return 0;
}

// 函数返回两个数中较大的那个数
int max(int num1, int num2)
{
   // 局部变量声明
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

把 max() 函数和 main() 函数放一块，编译源代码。当运行最后的可执行文件时，会产生下列结果：

```
Max value is : 200
```

函数参数

如果函数要使用参数，则必须声明接受参数值的变量。这些变量称为函数的**形式参数**。

形式参数就像函数内的其他局部变量，在进入函数时被创建，退出函数时被销毁。

当调用函数时，有两种向函数传递参数的方式：

调用类型	描述
传值调用	该方法把参数的实际值复制给函数的形式参数。在这种情况下，修改函数内的形式参数对实际参数没有影响。
指针调用	该方法把参数的地址复制给形式参数。在函数内，该地址用于访问调用中要用到的实际参数。这意味着，修改形式参数会影响实际参数。
引用调用	该方法把参数的引用复制给形式参数。在函数内，该引用用于访问调用中要用到的实际参数。这意味着，修改形式参数会影响实际参数。

默认情况下，C++ 使用**传值调用**来传递参数。一般来说，这意味着函数内的代码不能改变用于调用函数的参数。之前提到的实例，调用 max() 函数时，使用了相同的方法。

参数的默认值

当您定义一个函数，您可以为参数列表中后边的每一个参数指定默认值。当调用函数时，如果实际参数的值留空，则使用这个默认值。

这是通过在函数定义中使用赋值运算符来为参数赋值的。调用函数时，如果未传递参数的值，则会使用默认值，如果指定了值，则会忽略默认值，使用传递的值。请看下面的实例：

实例

```
#include <iostream>
using namespace std;

int sum(int a, int b=20)
{
  int result;

  result = a + b;

  return (result);
}

int main ()
{
   // 局部变量声明
   int a = 100;
   int b = 200;
   int result;

   // 调用函数来添加值
   result = sum(a, b);
   cout << "Total value is :" << result << endl;

   // 再次调用函数
   result = sum(a);
   cout << "Total value is :" << result << endl;

   return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Total value is :300
Total value is :120
```

Lambda 函数与表达式

C++11 提供了对匿名函数的支持,称为 Lambda 函数(也叫 Lambda 表达式)。

Lambda 表达式把函数看作对象。Lambda 表达式可以像对象一样使用，比如可以将它们赋给变量和作为参数传递，还可以像函数一样对其求值。

Lambda 表达式本质上与函数声明非常类似。Lambda 表达式具体形式如下:

```
[capture](parameters)->return-type{body}
```

例如：

```
[](int x, int y){ return x < y ; }
```

如果没有返回值可以表示为：

```
[capture](parameters){body}
```

例如：

```
[]{ ++global_x; }
```

在一个更为复杂的例子中，返回类型可以被明确的指定如下：

```
[](int x, int y) -> int { int z = x + y; return z + x; }
```

本例中，一个临时的参数 z 被创建用来存储中间结果。如同一般的函数，z 的值不会保留到下一次该不具名函数再次被调用时。
如果 lambda 函数没有传回值（例如 void），其返回类型可被完全忽略。
在Lambda表达式内可以访问当前作用域的变量，这是Lambda表达式的闭包（Closure）行为。 与JavaScript闭包不同，C++变量传递有传值和传引用的区别。可以通过前面的[]来指定：

```
[]      // 没有定义任何变量。使用未定义变量会引发错误。
[x, &y] // x以传值方式传入（默认），y以引用方式传入。
[&]     // 任何被使用到的外部变量都隐式地以引用方式加以引用。
[=]     // 任何被使用到的外部变量都隐式地以传值方式加以引用。
[&, x]  // x显式地以传值方式加以引用。其余变量以引用方式加以引用。
[=, &z] // z显式地以引用方式加以引用。其余变量以传值方式加以引用。
```

另外有一点需要注意。对于[=]或[&]的形式，lambda 表达式可以直接使用 this 指针。但是，对于[]的形式，如果要使用 this 指针，必须显式传入：

```
[this]() { this->someFunc(); }();
```

← C++ 判断　　C++ 数字 →

- JavaScript 实例
- Ajax 实例
- jQuery 实例
- XML 实例
- Java 实例

- HTML ASCII 字符集
- HTML ISO-8859-1
- HTML 实体符号
- HTML 拾色器
- JSON 格式化工具

- Windows10 MYSQ...
- Docke 镜像加速
- Debian Docker 安装
- C 库函数 –...
- Linux groupadd ...
- CSS var() 函数

- 免责声明
- 关于我们
- 文章归档

关注微信

反馈/建议