



第二章 认识程序基本结构

主讲教师：同济大学电子与信息工程学院 陈宇飞
同济大学电子与信息工程学院 龚晓亮



目录

- 进入C++
- C++语句
- 函数初探



目录

- 进入C++基本结构

- main() 函数
- C++注释
- C++预处理器和iostream文件
- 头文件名
- 名称空间
- 使用cout进行C++输出
- C++源代码的格式化



2.1 C++程序基本结构

2.1.1 编程的基本结构

```
// myfirst.cpp--displays a message

#include <iostream>
using namespace std;
int main()
{
    cout << "Hello world!";
    cout << endl;
    cout << "You won't regret it!" << endl;
    return 0;
}

// a PREPROCESSOR directive
// make definitions visible
// function header
// start of function body
// message
// start a new line
// more output

// end of function body
```

问：这个程序有几个部分？





2.1 C++程序基本结构

2.1.1 编程的基本结构

// myfirst.cpp--displays a message

1 #include <iostream>2

3 using namespace std;

4 int main()

{

cout << "Hello world!";

cout << endl;

cout << "You won't regret it!" << endl;

return 0;

}

5// a PREPROCESSOR directive

// make definitions visible

// function header

// start of function body

// message

// start a new line

// more output

// end of function body

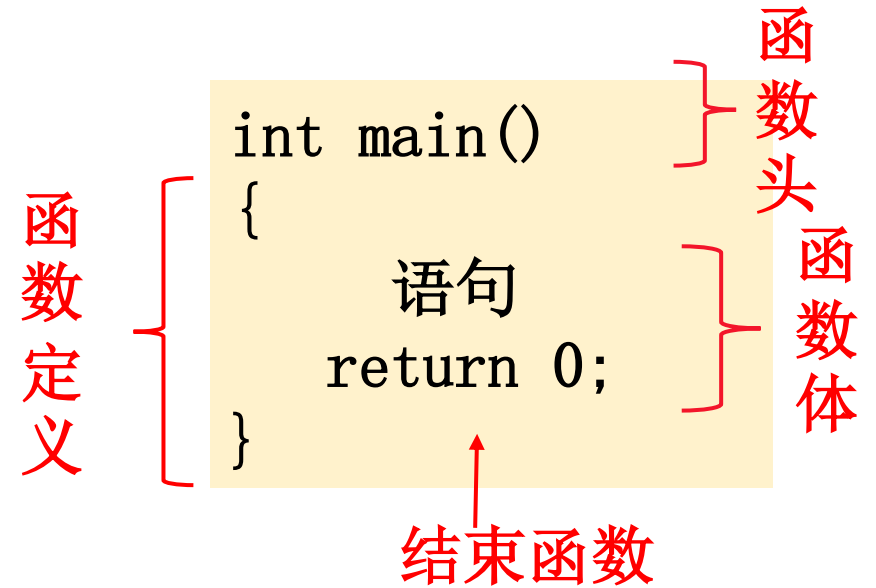
1. 预编译指令; 2. 头文件; 3. 名称空间; 4. main() 函数; 5. 注释



2.1 C++程序基本结构

2.1.2 C++ main() 函数

- ✓函数头对函数与程序其他部分之间的接口进行了总结
- ✓函数体是指出函数应该做什么的计算机指令
- ✓main() 中最后一条语句叫作返回语句 (return statement)，它结束该函数
- ✓位于函数前面的部分叫作**函数返回类型**，它描述的是从函数返回给调用它的函数的信息

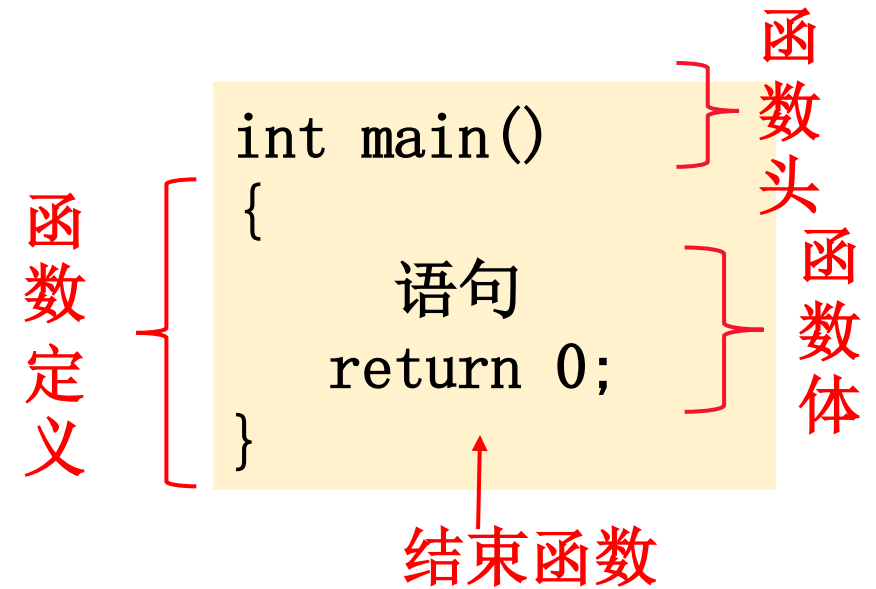




2.1 C++程序基本结构

2.1.2 C++ main() 函数

- ✓ 函数名后括号中的部分叫作**形参列表** (argument list) 或者 **参数列表** (parameter list)，它描述的是从调用函数传递给被调用的函数的信息
- ✓ C++独立程序必须包含一个名为main() 的函数（不是Main()、MAIN()或mane()，**大小写和拼写都要正确**）
- ✓ int main()（括号中可以包含void）：空或void表明main()没有参数
- ✓ main()可以带参





2.1 C++程序基本结构

2.1.2 C++ main() 函数

- ✓一般而言，编写一个能运行在操作系统上的程序，都需要一个main函数
- ✓main函数意味着建立一个独立进程，main函数是C++程序的入口，又是程序的出口
- ✓main函数必须返回int值, 如果想让程序拥有很好的可移植性，请一定要用int main()。如果没有明确写出，默认返回0
- ✓main函数的返回值用于说明程序的退出状态，如果返回0，表示程序正常退出，如果返回非0，则代表程序异常退出

注意：最好不要太多的细节逻辑直接放在主函数内，这样不利于维护和扩展。主函数应该尽量简洁，具体的实现细节应该封装到被调用的子函数里去。



2.1 C++程序基本结构

2.1.2 C++ main() 函数

- main函数的形式

(1) main函数参数列表为空

✓ 不需要从命令行中获取参数

✓ int main() 是UNIX和Linux中的默许写法

```
int main()
{
    // 函数体
    return 0;
}
```

```
int main(void)
{
    // 函数体
    return 0;
}
```



2.1 C++程序基本结构

2.1.2 C++ main() 函数

- main函数的形式

- (2) main函数参数列表带两个参数

- ✓ 需要从命令行中获取参数

- ✓ `int main(int argc, char* argv[])`

- 是UNIX和Linux中的标准写法

- ✓ `argc`为整数

- ✓ `argv`为指针的指针，`char** argv`、`char* argv[]`、`char argv[][]`

- ✓ 没有任何参数输入的条件下，数组`argv`并不为空，而是含有一个参数，也就是当前执行文件的路径及文件名，这个可执行文件存储在`argv[0]`中

```
int main(int argc, char* argv[])
{
    // 函数体
    return 0;
}
```

此处了解即可，详见后续进阶课程



2.1 C++程序基本结构

2.1.2 C++ main() 函数

- main函数的执行前后

(1) main函数执行前

一些全局变量、对象和静态变量、对象的空间分配和初始值在执行main函数之前

(2) main函数执行后

main函数执行完后，还要去执行一些诸如释放空间、释放资源使用权等操作，全局对象的析构函数会在main函数之后执行，用atexit注册的函数也会在main之后执行



2.1 C++程序基本结构

2.1.3 C++注释

(1) 常规注释

✓C++支持单行注释和多行注释。注释中的所有字符会被编译器忽略

✓C++注释一般有两种：

`//.....` 一般用于单行注释。注释以`//`开始，直到行末为止

`/*.....*/` 一般用于多行注释。注释以`/*`开始，以`*/`终止

```
cout << endl;      // start a new line
/* a main function
int main()
{
    return 0;
} */
```



2.1 C++程序基本结构

2.1.3 C++注释

(2) 条件编译注释

✓块注释符（/*...*/）是不可以嵌套使用的

✓**#if 0...#endif** 属于条件编译，0即为参数。可以使用**#if 0...#endif** 来实现注释，且可以实现嵌套，格式为：

```
#if 0
    code
#endif
```

```
#if condition
    code1
#else
    code2
#endif
```

✓如果condition条件为true执行code1，否则执行code2



2.1 C++程序基本结构

2.1.4 C++预处理器和iostream文件

- ✓C++预处理器（Preprocessor）是C++编译前的一个组件，负责在**编译之前对源代码进行处理**。它可以执行各种文本替换和条件编译，以及定义符号常量和宏函数等操作。
 - ✓预处理的代码通常以**#**开头，包括以下常用指令：
 - #define: 定义符号常量或宏函数
 - #include: 导入其它文件，通常在C++中用于导入头文件
 - #ifdef和#endif: 条件编译，根据条件判断是否编译某段代码
 - #error: 用来产生编译错误信息
- ```
#include <iostream> // a PREPROCESSOR directive
```
- ✓实际上，是将源代码文件和iostream文件组合成一个复合文件，编译的下阶段将使用该文件



## 2.1 C++程序基本结构

### 2.1.4 C++预处理器和iostream文件

- ✓ iostream文件是C++标准库的一部分，它提供了定义输入输出操作的类和对象，以便C++程序能够实现简单的命令行界面，包括向控制台输出、获取用户输入、读写文件等功能
- ✓ iostream文件包含两个主要类：istream和ostream。istream类是输入流类，提供一些方法来输入设备读取数据，而ostream类是输出流类，提供一些方法来想输出设备写入数据

```
#include <iostream>
using namespace std;
int main()
{
 cout << "Hello world!";
 return 0;
}
```



# 2.1 C++程序基本结构

## 2.1.5 头文件名

- ✓ 像iostream这样的文件叫作包含文件（include file），也叫作头文件（header file）-由于它们被包含在文件起始处
- ✓ C++编译器自带了很多头文件，每个头文件都支持一组特定的工具

| 头文件类型   | 约定          | 示例         | 说明                                  |
|---------|-------------|------------|-------------------------------------|
| C++旧式风格 | 以.h结尾       | iostream.h | C++程序可以使用                           |
| C旧式风格   | 以.h结尾       | math.h     | C、C++程序可以使用                         |
| C++新式风格 | 没有扩展名       | iostream   | C++程序可以使用，使用namespace std           |
| 转换后的C   | 加上前缀c，没有扩展名 | cmath      | C++程序可以使用，可以使用不是C的特性，如namespace std |





## 2.1 C++程序基本结构

### 2.1.6 命名空间

- ✓命名空间是C++中**为了解决命名冲突**所引申出的一种解决方案
- ✓命名空间的原理是将全局作用域划分为一个一个的命名空间，每个命名空间是一个独立的作用域，在不同命名空间内部定义的名字彼此之间互相不影响，从而**有效避免命名污染**

```
#include <stdio.h>
#include <stdlib.h>
int rand = 10;
int main()
{
 printf ("%d\n",rand) ;
 return 0;
}
```





## 2.1 C++程序基本结构

### 2.1.6 命名空间

- ✓定义命名空间，需要使用到 namespace 关键字，后面跟命名空间的名字，然后接一对 {} 即可，{} 中即为命名空间的成员
- ✓在命名空间作用域内，可以包含：变量、对象以及它们的初始化、枚举常量、函数声明以及函数定义、类、结构体声明与实现、模板、其他命名空间。

```
namespace Curry1
{
 int a = 10;
 int func1(int a, int b)
 {
 return a+b;
 }
 struct Node
 {
 struct Node* next;
 int val;
 };
}
```



## 2.1 C++程序基本结构

### 2.1.6 命名空间

- ✓命名空间可以可以在**全局作用域**或者其他命名空间内部定义(**嵌套定义**)，但不能在函数、结构体或类内部定义，且要保证命名空间之间不会出现名字冲突

```
namespace Curry1
{
 int a = 10;
 namespace Curry2
 {
 int b;
 void func2()
 {
 //...
 }
 }
 //...
}
```

# 2.1 C++程序基本结构

## 2.1.6 命名空间

- ✓每个命名空间是一个作用域，定义在命名空间中的实体成为命名空间成员，命名空间中的每个名字必须是该命名空间中唯一实体，但不同命名空间可以具有同名成员
- ✓同一个工程中允许存在多个相同名称的命名空间，**编译器最后会合成同一个命名空间中**

```
namespace Curry1
```

```
{
 int a = 10;
 void func1()
 {
 //...
 }
 //...
}
```

```
namespace Curry1
```

```
{
 int b;
 int Add(int a, int b)
 {
 return a+b;
 }
 //...
}
```





# 2.1 C++程序基本结构

## 2.1.6 命名空间

命名空间有三种使用方式:

(1) **命名空间名称::xxx**

✓这种命名空间的方式是三种中**最繁琐**，需要一个个去指定命名空间域，但同时它是**最安全**的，避免空间被污染

✓**::xxx** 表示引用全局命名空间成员

```
#include<cstdio>
int a = 100;
namespace Curry1
{
 int a = 30;
 int Add(int a, int b)
 {
 return a + b;
 }
}
int main()
{
 int a = 10;
 printf("%d\n", a);
 printf("%d\n", ::a);
 printf("%d\n", Curry1::a);
 printf("%d\n", Curry1::Add(1, 2));
 return 0;
}
```

|     |
|-----|
| 10  |
| 100 |
| 30  |
| 3   |



# 2.1 C++程序基本结构

## 2.1.6 命名空间

命名空间有三种使用方式:

(2) **using 命名空间名称::xxx**

✓将std C++标准库中经常使用的少数几个成员使用using 命名空间名称::xxx部分展开, 使用一部分, 暴露一部分。在一定程度上减少了命名空间被污染

```
#include<iostream>
using std::cout;
using std::endl;
int main()
{
 cout << "hello world!"<< endl;
 return 0;
}
```

## 2.1 C++程序基本结构

### 2.1.6 命名空间

命名空间有三种使用方式：

#### (3) `using namespace` 命名空间名称

- ✓这种方式**最为方便**，直接令整个命名空间成员都有效，但同时这种全局展开的方式是**最不安全**的。如果我们定义跟库重名的类型/对象/函数，就存在冲突问题
- ✓在日常练习中，建议直接**`using namespace std`**即可，很方便
- ✓但是项目开发中代码较多、规模大，就容易出现冲突问题。在项目开发中使用时，建议使用**`std::cout`**或者**`using std::cout`**指定展开常用库中的对象/类型等方式

```
#include<iostream>
using namespace std;
int main()
{
 cout << "hello world!"<< endl;
 return 0;
}
```



## 2.1 C++程序基本结构

### 2.1.6 命名空间

在C++新标准汇总<iostream>与<iostream.h>的区别

- ✓使用<iostream.h>调用的是C中的库函数，使用的是全局命名空间
- ✓使用<iostream>时，该文件没有定义全局命名空间，必须加上using namespace std;才能使用C++标准库中的类、函数

```
#include<iostream>
using namespace std;
int main()
{
 cout << "hello world!"<< endl;
 return 0;
}
```





## 2.1 C++程序基本结构

### 2.1.7 使用cout进行C++输出

#### 流的基本概念

✓流的含义：流是来自设备或传给设备的一个数据流，由一系列**字节**组成，按顺序排列（**也称为字节流**）

✓C语言用 printf/scanf 等**函数**来实现输入和输出  
通过 #include <stdio.h> 来调用

✓C++通过cin和cout的**流对象**来实现  
通过 #include <iostream> 来调用

cout: 输出流对象      <<: 流插入运算符

cin: 输入流对象      >>: 流提取运算符

```
#include<iostream>
using namespace std;
int main()
{
 cout << "hello world!"<< endl;
 return 0;
}
```



## 2.1 C++程序基本结构

### 2.1.7 使用cout进行C++输出

输出流的基本操作

格式: `cout << 表达式1 << 表达式2 << ... << 表达式n;`

- ✓ 插入的**数据存储在缓冲区**中, 不是立即输出, 要等到缓冲区满 (不同系统大小不同) 或者碰到换行符 ("**\n**" / **endl**) 或者强制立即输出 (**flush**) 才一齐输出
- ✓ 显示用引号括起来的字符串时, 通常使用**换行符\n**, 在其他情况下则使用**控制符endl**

```
cout << "hello" << endl;
cout << "hello\n";
cout << "hello" << "\n";
cout << "hello" << '\n';
```

换行符的多种形式



# 2.1 C++程序基本结构

## 2.1.7 使用cout进行C++输出

### 输出流的基本操作

- ✓ 默认的输出设备是显示器(可更改, 称**输出重定向**, 进阶课内容)
- ✓ 一个cout语句可写为若干行, 或者若干语句

```
cout << "This is a C++ program." << endl;
```

```
cout << "This is " << "a C++ " << "program." << endl;
```

```
cout << "This is "
 << "a C++ "
 << "program."
 << endl;
```

一个语句分4行  
前三行无分号

```
cout << "This is ";
cout << "a C++ ";
cout << "program.";
cout << endl;
```

4个语句  
每行有分号



## 2.1 C++程序基本结构

### 2.1.7 使用cout进行C++输出

#### 输出流的基本操作

✓一个插入运算符只能输出一个值

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

int main()
{
 int a=10, b=15, c=20;
 cout << a << b << c;
 return 0;
}
```

Microsoft Visual Studio 调试控制台  
101520

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

int main()
{
 int a=10, b=15, c=20;
 cout << a, b, c;
 return 0;
}
```

Microsoft Visual Studio 调试控制台  
10



## 2.1 C++程序基本结构

### 2.1.8 C++源代码的风格

```
// myfirst.cpp--displays a message
#include <iostream>
using namespace std;

int main()
{
 cout << "Hello world!";
 cout << endl;
 cout << "You won't regret it!" << endl;
 return 0;
}
```

// a PREPROCESSOR directive  
// make definitions visible  
  
// function header  
// start of function body  
// message  
// start a new line  
// more output  
  
// end of function body



## 2.1 C++程序基本结构

### 2.1.8 C++源代码的风格

- ✓目的：使代码更加易读、易于维护
  - ✓常见的C++代码格式规则（Primer书）：
    - 每条语句占一行
    - 每个函数都有一个开始花括号和一个结束花括号，这两个花括号各占一行
    - 函数中的语句都相对于花括号进行缩进
    - 与函数名称相关的圆括号周围没有空白
- 前三条规则旨在确保代码清晰易读；第四条规格帮助区分函数和一些也使用圆括号的C++内置结构（如循环）



## 2.1 C++程序基本结构

### 2.1.8 C++源代码的风格

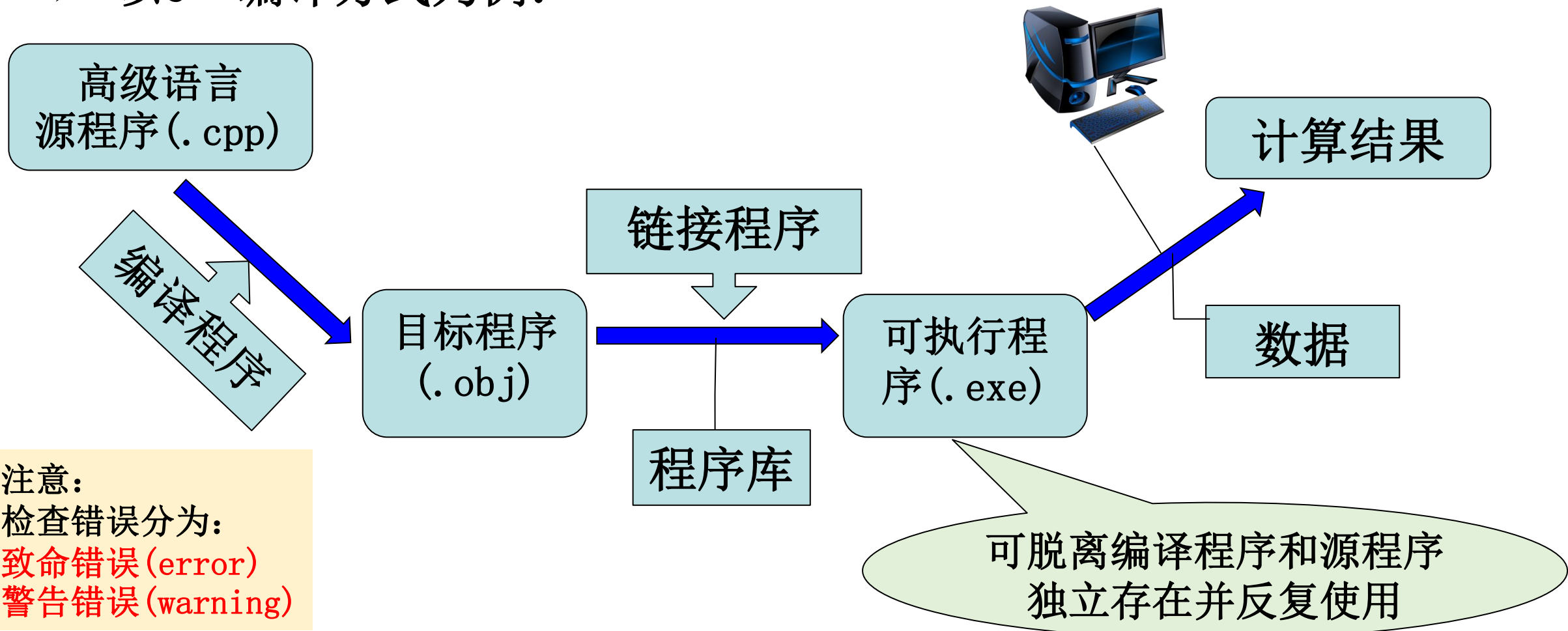
✓常见的C++代码格式规则（补充说明）：

- ① 缩进：使用空格或制表符将代码块缩进，通常是四个空格或一个制表符
- ② 大括号：在控制结构（如if、for、while等）中使用大括号。注意，即使只有一条语句，也应该使用大括号，这样可以避免潜在的错误
- ③ 换行：在较长的语句中使用换行符，以便阅读。通常是在函数调用、运算符、逗号等位置换行
- ④ 空格：在运算符前后、函数调用的参数前后、关键字与括号之间等位置使用空格，以便于阅读

# 2.1 C++程序基本结构

## 2.1.9 编程的基本过程

➤ 以C++编译方式为例:







# 目录

- 进入C++
- C++语句
- 函数初探



# 目录

- C++语句
  - 声明语句与变量
  - 赋值语句
  - 其他C++语句

```
// carrots.cpp -- food processing program
// uses and displays a variable
#include <iostream>
using namespace std;

int main()
{
 int carrots; // declare an integer variable
 carrots = 25; // assign a value to the variable
 cout << "I have ";
 cout << carrots; // display the value of the variable
 cout << " carrots.";
 cout << endl;
 carrots = carrots - 1; // modify the variable
 cout << "Crunch, crunch. Now I have " << carrots << " carrots." << endl;
 return 0;
}
```



## 2.2 C++语句

### 2.2.1 声明语句与变量

- ✓ 计算机是一种精确的、有条理的机器。要将信息项存储在计算机中，必须指出信息的**存储位置**和所需的**内存空间**
- ✓ 在C++中，使用**声明语句**来指出存储类型并提供位置标签

```
int carrots;
```

- ✓ `int` 整数类型，`carrots`是变量，C++中，所有变量都必须声明，尽可能在首次使用变量前声明它
- ✓ 程序中的声明语句叫作定义声明（defining declaration）语句，简称为定义（definition）。这意味着**编译器将为变量分配内存空间**

变量定义语法格式：  
数据类型 变量名称 = 初值



## 2.2 C++语句

### 2.2.1 声明语句与变量

- ✓ 较为复杂情况下，还有引用声明（reference declaration），这些声明命令计算机使用在其他地方定义的变量。这意味着**编译器将为不分配内存空间**
- ✓ 通过使用**extern关键字**声明变量名而不定义它。extern告诉编译器变量在其他地方定义了

```
extern int carrots;
```

- ✓ 同一个变量，只能有一个定义，但可以在多个文件中进行声明



## 2.2 C++语句

### 2.2.2 赋值语句

✓赋值语句将值赋给存储单元

```
carrots = 25;
```

✓符号=叫作**赋值运算符**

```
carrots = carrots - 1;
```

(1) C++（和C）特性-**可以连续使用赋值运算符**

赋值将**从右向左**进行

```
int steinway;
int baldwin;
int yamaha;
yamaha = baldwin = steinway = 88;
```

(2) 进行赋值运算时，如果赋值运算符两边的数据类型不同，系统会**自动进行类型转换**



## 2.2 C++语句

### 2.2.3 其他C++语句

```
// getinfo.cpp -- input and output
#include <iostream>
using namespace std;
int main()
{
 int carrots;
 cout << "How many carrots do you have?" << endl;
 cin >> carrots; // C++ input
 cout << "Here are two more. ";
 carrots = carrots + 2; // the next line concatenates output
 cout << "Now you have " << carrots << " carrots." << endl;
 return 0;
}
```



# 2.2 C++语句

## 2.2.3 其他C++语句

### 输入流的基本操作

格式: cin >> 变量1 >> 变量2 >> ... >> 变量n; cin >> carrots;

- ✓ 键盘输入的**数据存储在缓冲区**中, 不是立即被提取, 要等到缓冲区满(不同系统大小不同)或碰到**回车符**才进行提取
- ✓ 默认的输入设备是键盘(可更改, 称**输入重定向**, 进阶课程内容)
- ✓ 一行输入内容可分为若干行, 或者若干语句

|                          |                   |           |               |
|--------------------------|-------------------|-----------|---------------|
| cin >> a >> b >> c >> d; |                   |           |               |
| cin >> a                 | 1个语句分4行<br>前3行无分号 | cin >> a; | 4个语句<br>每行有分号 |
| >> b                     |                   | cin >> b; |               |
| >> c                     |                   | cin >> c; |               |
| >> d;                    |                   | cin >> d; |               |





## 2.2 C++语句

### 2.2.3 其他C++语句

#### 输入流的基本操作

✓ 一个提取运算符只能输入一个值

```
#include <iostream>
using namespace std;
int main()
{
 int a, b, c;
 cin >> a >> b >> c;
 cout << a << b << c;
 return 0;
}
```

Microsoft Visual Studio 调试控制台

```
10 20 30
102030
```

```
#include <iostream>
using namespace std;
int main()
{
 int a, b, c;
 cin >> a, b, c;
 cout << a << ' ' << b << ' ' << c;
 return 0;
}
```



## 2.2 C++语句

### 2.2.3 其他C++语句

#### 输入流的基本操作

✓ 一个提取运算符只能输入一个值

不同编译器结果差异本学期暂不深入讨论

VS

```
error C4700: 使用了未初始化的局部变量 "c"
error C4700: 使用了未初始化的局部变量 "b"
```

Dev

D:\Workspace\VS2019-Demo\cpp-demo\cpp-demo.exe

```
10 20 30
10 4242416 7929704
```

cin >> a, b, c;

逗号表达式，只有a被cin，但b和c是随机值

```
#include <iostream>
using namespace std;
int main()
{
 int a, b, c;
 cin >> a, b, c;
 cout << a << ' ' << b << ' ' << c;
 return 0;
}
```



## 2.2 C++语句

### 2.2.3 其他C++语句

#### 输入流的基本操作

- ✓ 提取运算符后必须跟变量名, 不能是常量/表达式等  
(因为提取数据后会改变 >> 后的值, 因此 >> 后面必须是变量)

```
#include <iostream>
using namespace std;
int main()
{
 int a=1, b=1, c=1;
 cin >> a + 10;
 cout << a << ' ' << b << ' ' << c << endl;
 return 0;
}
```

输出

显示输出来源(S): 生成

1>C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tools\MSVC\14.31.31103\include\istream(186,39): message : 或 “std::basic\_istream<char, std::char\_traits<char>>”  
1>C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tools\MSVC\14.31.31103\include\istream(181,39): message : 或 “std::basic\_istream<char, std::char\_traits<char>>”  
1>C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tools\MSVC\14.31.31103\include\istream(176,39): message : 或 “std::basic\_istream<char, std::char\_traits<char>>”  
1>D:\Workspace\VS2022-demo\demo-cpp\demo.cpp(6,18): message : 或 “内置 C++ operator>>(int, int)”  
1>D:\Workspace\VS2022-demo\demo-cpp\demo.cpp(6,18): message : 尝试匹配参数列表“(std::istream, int)”时  
1>已完成生成项目“demo-cpp.vcxproj”的操作 - 失败。  
生成: 成功 0 个, 失败 1 个, 最新 0 个, 跳过 0 个

```
int a=1, b=1, c=1;
cin >> a + 10;
cout << a << ' ' << b << ' ' << c << endl;
return 0;
}
```

上百个错误, 而且提示系统文件出错, 没装好? 病毒?



## 2.2 C++语句

### 2.2.3 其他C++语句

#### 输入流的基本操作

- ✓ 提取运算符后必须跟变量名, 不能是常量/表达式等  
(因为提取数据后会改变 >> 后的值, 因此 >> 后面必须是变量)

```
#include <iostream>
using namespace std;
int main()
{
 int a=1, b=1, c=1;
 cin >> a + 10;
}
```

不要慌, 移到开头, 第一个错  
一定在你的程序里!!!

输出

显示输出来源(S): 生成

已启动生成...

1>—— 已启动生成: 项目: cpp-demo, 配置: Debug Win32

1>cpp-demo.cpp

1>D:\WorkSpace\VS2019-Demo\cpp-demo\cpp-demo.cpp(6,18): error C2678: 二进制“>>”: 没有找到接受“std::istream”类型的左操作数的运算符(或没有可接受的转换)

1>C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\include\istream(301,39): message : 可能是“std::basic\_istream<char, std::char\_traits<char>>

1>C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\include\istream(297,39): message : 或“std::basic\_istream<char, std::char\_traits<char>>

1>C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\include\istream(293,39): message : 或“std::basic\_istream<char, std::char\_traits<char>>

1>C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\include\istream(289,39): message : 或“std::basic\_istream<char, std::char\_traits<char>>



## 2.2 C++语句

### 2.2.3 其他C++语句

#### 输入流的基本操作

- ✓ 提取运算符后必须跟变量名, 不能是常量/表达式等  
(因为提取数据后会改变 >> 后的值, 因此 >> 后面必须是变量)

```
#include <iostream>
using namespace std;
int main()
{
 int a=1, b=1, c=1;
 cin >> a + 10; (编译时语法错)
 cout << a << ' ' << b << ' ' << c << endl;
 return 0;
}
```



## 2.2 C++语句

### 2.2.3 其他C++语句

后续课程/实践深入讨论

#### 输入流的基本操作

- ✓ 输入终止条件为回车、空格、非法输入
- ✓ 系统会自动根据cin后变量的类型按最长原则来读取合理数据
- ✓ 变量读取后，系统会判断输入数据是否超过变量的范围，若超过则置内部的错误标记并返回一个不可信的值（不同编译器处理不同）
- ✓ cin输入完成后，通过cin.good()/cin.fail()可判断本次输入是否正确

| 输入              | cin.good()返回 | cin.fail()返回 |
|-----------------|--------------|--------------|
| 正确范围+回车/空格/非法输入 | 1            | 0            |
| 错误范围+回车/空格/非法输入 | 0            | 1            |
| 非法输入            | 0            | 1            |



# 目录

- 进入C++
- C++语句
- 函数初探



# 目录

- 函数初探
  - 程序结构的基本形式
  - 函数的组成
  - 函数的分类
  - 用户定义的函数

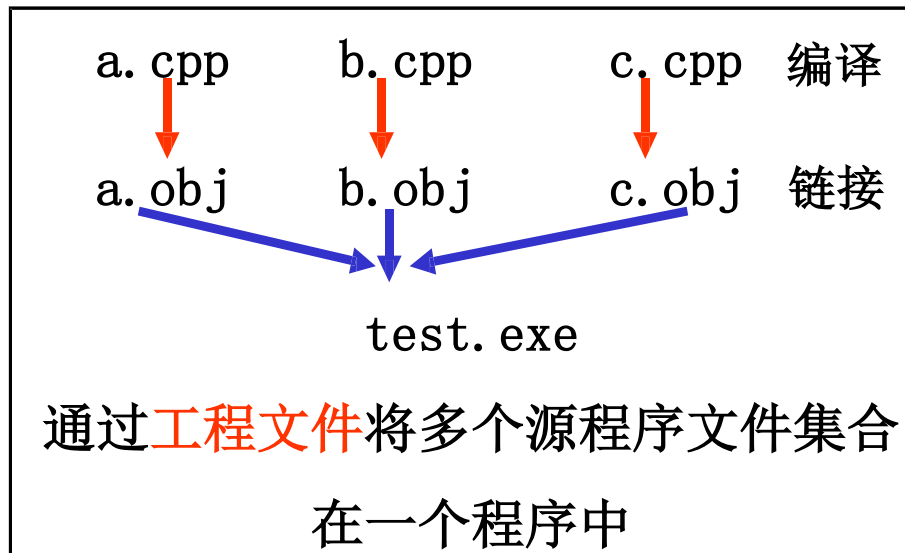




## 2.3 函数初探

### 2.3.1 程序结构的基本形式

- ✓ C++程序由函数组成，函数是C++程序的基本单位
- ✓ 一个C++程序可由若干源程序文件 (\*.cpp) 组成，每个源程序文件可以包含若干函数



```
#include<iostream>
using namespace std;
int main()
{
 cout << "hello world!"<< endl;
 return 0;
}
```



## 2.3 函数初探

### 2.3.1 程序结构的基本形式

- ✓ 有且仅有一个名为main()的函数，称为主函数，程序的执行从它开始
- ✓ C++提供许多库函数（已做好的）
- ✓ 系统提供的库函数和自己编写的函数调用方法相同

```
#include <iostream>
using namespace std;
int max(int x, int y)
{
 int z;
 if (x>y) z=x;
 else z=y;
 return (z);
}

int main()
{
 int a, b, m;
 cin >> a >> b;
 m=max(a, b);
 cout<<"max="<<m<<' \n' ;
 return 0;
}
```



## 2.3 函数初探

### 2.3.2 函数的组成

函数返回类型    函数名（形式参数表）

{

函数体（局部变量的定义、函数体的可执行部分）

}

```
int max(int x, int y)
{
 int z;
 if (x>y) z=x;
 else z=y;
 return (z);
}
```



## 2.3 函数初探

### 2.3.2 函数的组成

- ✓ 从main()开始执行，函数相互间的位置不影响程序的正确性


```
#include <iostream>
using namespace std;
int max(int x, int y)
{ ...
}
int main()
{ ...
}
```

```
#include <iostream>
using namespace std;
int main()
{ ...
}
int max(int x, int y)
{ ...
}
```

## 2.3 函数初探

### 2.3.2 函数的组成

- ✓ 函数平行定义，嵌套调用
- ✓ 函数由语句组成，一个语句以;  
结尾（**必须有**）
- ✓ 语句分为定义语句和执行语句，  
定义语句用于声明某些信息，执  
行语句用于完成特定的操作



```
#include <iostream>
using namespace std;
int max(int x, int y) //函数定义
{
 int z; //定义语句
 if (x>y) z=x; //执行语句
 else z=y; //执行语句
 return (z);
}
```

```
int main()
{
 int a,b,m;
 cin >> a >> b;
 m=max(a, b); //函数调用
 cout<<"max="<<m<<' \n' ;
 return 0;
}
```



## 2.3 函数初探

### 2.3.2 函数的组成

✓ 书写格式自由：可以一行多个语句，也可以一个语句多行（以\分行）

```
#include <iostream>
using namespace std;
int main()
{
 cout << "This is a C++ program";
 return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{ cout << "This is a C++ program";return 0;}
```

```
#include <iostream>
using namespace std;
int main()
{ cout \
 << \
 "This is a C++ program";
 return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{ cout
 <<
 "This is a C++ program";
 return 0;
}
```

编译器  
均通过



## 2.3 函数初探

### 2.3.3 函数的分类

✓ 有返回值的

```
#include<iostream>
using namespace std;
int main()
{
 cout << "hello world!"<< endl;
 return 0;
}
```

✓ 没有返回值的

```
#include<iostream>
using namespace std;
void main()
{
 cout << "hello world!"<< endl;
}
```



## 2.3 函数初探

### 2.3.3 函数的分类

- ✓ 有参数的函数

```
int sum(int x, int y)
{
 return a + b;
}
```

- ✓ 默认参数的函数  
如果调用该函数时没有传递相应参数，则使用默认值

```
int sum(int x=0, int y=0)
{
 return a + b;
}
```

- ✓ 不接受参数的函数

```
int rand(void); //prototype of a function that takes no arguments
MyGuess = rand(); //function call with no arguments
```





## 2.3 函数初探

### 2.3.3 函数的分类

- ✓ 没有返回值的函数

```
void bucks(double);
//prototype for function with no return value
```

由于它不返回值，因此不能将该函数调用放在赋值语句或其他表达式中

```
bucks(1234.56);
//function call, no return value
```

- ✓ 函数重载 (function overloading)
  - ✓ 内联函数 (inline function)
  - ✓ 函数模板 (function template) ...
- } 后续课程内容



## 2.3 函数初探

### 2.3.4 用户定义的函数

#### 函数格式

- ✓ 用户定义的函数与main函数的定义采用的格式相同，含函数头和花括号中的函数体

```
type functionname (argumentlist)
{
 statements
}
```

- ✓ C++不允许将函数定义嵌套在另一个函数中。每个函数定义都是独立的，所有函数创建都是平等的



## 2.3 函数初探

### 2.3.4 用户定义的函数

```
// ourfunc.cpp -- defining your own function
#include <iostream>
using namespace std;
```

```
void simon(int); // function prototype for simon()
```

```
int main()
{
 simon(3); // call the simon() function
 cout << "Pick an integer: ";
 int count;
 cin >> count;
 simon(count); // call it again
 cout << "Done!" << endl;
 return 0;
}
```

```
void simon(int n)
// define the simon() function
{
 cout << "Simon says touch your toes "
 << n << " times." << endl;
}
// void functions don't need return
statements
```

用户定义的函数simon()的源代码位于main()函数的后面



## 2.3 函数初探

### 2.3.4 用户定义的函数

#### 函数命名约定

- ✓ 选择取决于开发团队、使用的技术或库以及程序员个人的品味和喜好

```
Myfunction()
myfucntion()
myFunction()
my_function()
my_funcnt()
```

- ✓ 精确、让人一目了然的个人命名约定是良好的软件工程师的标志，它将在整个编程生涯中都会起到很好的作用



## 2.3 函数初探

### 2.3.4 用户定义的函数

```
// convert.cpp -- converts stone to pounds
#include <iostream>
using namespace std;
int stonetolb(int); // function prototype
int main()
{
 int stone;
 cout << "Enter the weight in stone: ";
 cin >> stone;
 int pounds = stonetolb(stone);
 cout << stone << " stone = ";
 cout << pounds << " pounds." << endl;
 return 0;
}
```

```
int stonetolb(int sts)
{
 return 14 * sts;
}
```



## 2.3 函数初探

### 2.3.4 用户定义的函数

- ✓ 函数原型描述了函数接口
- ✓ 函数`stonetolb()`特性:
  - 有函数头和函数体
  - 接受一个参数
  - 返回一个值
  - 需要一个原型

返回结果相同，但下面的版本更容易理解和修改，它将计算和返回分开

```
int stonetolb(int);
// function prototype
```

```
int stonetolb(int sts)
{
 return 14 * sts;
}
```

```
int stonetolb(int sts)
{
 int pounds=14 * sts;
 return pounds;
}
```



# 总结

- 进入C++

(C++程序基本结构)

- 预编译指令
- 头文件
- 名称空间
- main() 函数
- 注释

- C++语句

- 声明语句
- 变量
- 赋值语句
- 流和关键字  
(cout和cin)

- 函数初探

- 函数头
- 函数的参数
- 函数的返回值
- 函数的调用