



C++语言程序设计

第八章 编程规范

王焦乐

<http://faculty.hitsz.edu.cn/jlwang>



哈尔滨工业大学（深圳）
机电工程与自动化学院



C++语言程序设计...

群号：599477959



扫一扫二维码，加入群聊。





本章主要内容

- 头文件
- 变量作用域
- 编程特性
- 命名约定
- 注释
- 格式



头文件

头文件保护

- 所有头文件都应该使用**#define**防止头文件被多重包含。
- 命名格式: `<PROJECT>_<PATH>_<FILE>_H_`
- 例: 项目foo 中的头文件foo/src/bar/baz.h

```
#ifndef FOO_BAR_BAZ_H_
```

```
#define FOO_BAR_BAZ_H_
```

```
...
```

```
#endif // FOO_BAR_BAZ_H_
```



头文件

内联函数

- inline
- 只有当函数只有10行甚至更少时才会将其定义为内联函数
- 内联那些包含循环或switch语句的函数是得不偿失的，除非在大多数情况下，这些循环或switch 语句从不执行
- 递归函数不应该被声明为内联

```
inline int max(int a, int b)
{
    return a>b ? a : b;
}
```

```
inline int fibonacci(int n)
{
    if (n==1 || n==0)
        return 1;
    else
        return fibonacci(n-1)+fibonacci(n-2);
}
```



头文件

函数参数顺序

- 定义函数时，参数顺序为：**输入**参数在前，**输出**参数在后
- **输入**参数一般传值或**常数引用** (const references)
- **输出**参数或输入/输出参数为**非常数指针** (non-const pointers) 或引用

```
void swap(int *a, int *b)
{
    ...
}
```

```
void swap(int &a, int &b)
{
    ...
}
```

```
void add(const int &a, const int &b, int *c)
{
    c=a+b;
}
```

```
void add(const int &a, const int &b, int &c)
{
    c=a+b;
}
```



头文件

头文件包含顺序

- 将包含次序标准化可增强可读性、避免隐藏依赖 (hidden dependencies) , 次序如下: C 库、C++库、其他库的.h、项目内的.h。
- 例: dir/foo.cc 的主要作用是执行或测试dir2/foo2.h 的功能, foo.cc 中包含头文件的次序如下:
 - dir2/foo2.h (优先位置)
 - C 系统文件
 - C++系统文件
 - 其他库头文件
 - 本项目内头文件



变量作用域

局部变量

- 将函数变量尽可能置于最小作用域内
- 尽可能靠近首次使用位置
- 在声明变量时将其初始化

```
int i;  
i=f(); // 坏——初始化和声明分离  
int j=g(); // 好——初始化时声明
```

- 如果变量是一个**对象**，声明在循环外

```
for (int i = 0; i < 1000000; ++i)  
{  
    Foo f; // 构造函数和析构函数分别调用1000000次!  
    f.DoSomething(i);  
}
```

```
Foo f;  
for (int i = 0; i < 1000000; ++i)  
{  
    f.DoSomething(i);  
}
```



变量作用域

全局变量

- `class` 类型的全局变量是被禁止的
- 内建类型的全局变量是允许的
- 多线程代码中非常数全局变量也是被禁止的
- 永远不要使用函数返回值初始化全局变量。
- 对于全局的字符串常量，使用C风格的字符串，而不要使用STL 的字符串：
 - `const char kFrogSays[] = "ribbet";`



编程特性

降低耦合

- 一个函数只做一件事。
- 不要过早在意细节优化。
- 先用最清楚的**逻辑**描述出程序的**框架**。
- 减少全局变量的数量。
- 试图面向对象。



前置自增和自减

- 不考虑返回值的话，前置自增 ($++i$) 通常要比后置自增 ($i++$) 效率更高
- 因为后置的自增自减需要对表达式的值 i 进行一次拷贝
- 如果 i 是迭代器或其他非数值类型，拷贝的代价是比较大的。
- 对迭代器和模板类型来说，要使用前置自增（自减）。



预处理宏

- 宏意味着你和编译器看到的代码是不同的，因此可能导致异常行为，尤其是当宏存在于全局作用域中。
- 使用宏时要谨慎，尽量以内联函数、枚举和常量代替之。
- `#define MAX 1024` `const int MAX=1024;`
- `#define ADD(a,b) (a+b)`
- `inline int ADD(int a, int b){return a+b;}`



尽量用const 和inline 而不用#define

- 尽量用编译器而不用预处理。
- `#define max(a,b) ((a) > (b) ? (a) : (b))`
- `int a = 5, b = 0;`
- `max(++a, b);` // a 的值增加了? 次
- `max(++a, b+10);` // a 的值增加了? 次
- `template<class T>`
- `inline const T& max(const T& a, const T& b)`
- `{ return a > b ? a : b; }`



编程特性

尽量用 `<iostream>` 而不用 `<stdio.h>`

- `scanf` 和 `printf` 不是类型安全的，而且没有扩展性。
- 类型安全和扩展性是C++的基石

尽量用 `new` 和 `delete` 而不用 `malloc` 和 `free`

- 构造函数和析构函数
- 析构函数里对指针成员调用 `delete`

0 和 NULL

- 整数用 `0`
- 实数用 `0.0`
- 指针用 `NULLPTR`
- 字符 (串) 用 `'\0'`



命名约定

- **最重要的一致性规则是命名管理，命名风格直接可以直接确定命名实体是：类型、变量、函数、常量、宏等等**
 - **无需查找实体声明，便可了解含义。**
- **命名规则具有一定随意性，但相比按个人喜好命名，一致性更重要，所以不管你怎么想，规则总归是规则。**
- **团队合作使用统一的约定**



命名约定

通用命名规则

- 函数命名、变量命名、文件命名应具有描述性，不要过度缩写
- 类型和变量应该是名词
- 函数名可以用“命令性”动词
- 除约定俗称的缩写外，尽量不适用缩写
 - `int num_completed_connections`
 - `int n` `int n_comp_conns`
 - `void OpenFile()`



命名约定

文件名

- 文件名要全部小写，可以包含下划线（_）或短线（-），按项目约定来。
如：
 - my_useful_class.cc
 - my-useful-class.cc
 - myusefulclass.cc
- 通常，尽量让文件名更加明确，
 - http_server_logs.h
 - logs.h
- 定义类时文件名一般成对出现，如foo_bar.h 和foo_bar.cpp，对应类FooBar。



命名约定

类型命名

- 类型命名每个单词以**大写字母**开头，**不包含下划线**
 - MyExcitingClass
 - MyExcitingEnum。
- 所有类型命名使用相同约定
 - 类
 - 结构体
 - 类型定义 (typedef)
 - 枚举



命名约定

变量命名

- 变量名一律小写，单词间以下划线相连，类的成员变量以下划线结尾，如
 - `int my_exciting_local_variable`
 - `int my_exciting_member_variable_`
- 全局变量可以以`g_`为前缀
 - `int g_stu_num`
- 所有编译时常量（无论是局部的、全局的还是类中的）和其他变量保持些许区别，`k`后接大写字母开头的单词：
 - `const int kDaysInAWeek = 7;`



格式

- 保证代码可读性
- Code Tells You How, Comments Tell You Why
- 风格：使用//或/* */，统一就好。
- 文件：在每一个文件开头加入版权公告，然后是文件内容描述。
- 类：每个类的定义要附着描述类的功能和用法的注释
- 函数：声明处注释描述函数功能，定义处描述函数实现。
- 代码：对于实现代码中巧妙的、晦涩的、有趣的、重要的地方加以注释。

```
#include <iostream>

using namespace std;

int main()
{
    const int kArraySize=10;
    int array_test[kArraySize]={0};

    // some comments here
    for (int i=0; i<kArraySize; ++i)
    {
        if (0==array_test[i])
        {
            array_test[i]=i;
            cout<<array_test[i]<<endl;
        } // end if
        else
        {
            cout<<array_test[i]<<endl;
        } // end else
    } //end for

    return 0;
}
```