**C++编程规范**

**2010年6月22日**

**目录**

[1 介绍 3](#_Toc264989265)

[1.1 文档目的 3](#_Toc264989266)

[1.2 动机 3](#_Toc264989267)

[2 一般性原则 3](#_Toc264989268)

[3 文件结构 4](#_Toc264989269)

[3.1 一般性问题 4](#_Toc264989270)

[3.2 头文件 4](#_Toc264989271)

[3.3 源文件的结构。 6](#_Toc264989272)

[3.4 目录结构 7](#_Toc264989273)

[4 命名和书写规范 7](#_Toc264989274)

[4.1 标识符的命名规则 7](#_Toc264989275)

[4.2 类 8](#_Toc264989276)

[4.3 函数 9](#_Toc264989277)

[4.4 数据成员 10](#_Toc264989278)

[4.5 指针和引用 10](#_Toc264989279)

[4.6 宏定义 11](#_Toc264989280)

[4.7 const常量 11](#_Toc264989281)

[4.8 枚举变量 11](#_Toc264989282)

[4.9 全局变量 12](#_Toc264989283)

[5 表达式和基本语句 12](#_Toc264989284)

[5.1 运算符的优先级 12](#_Toc264989285)

[5.2 复合表达式 12](#_Toc264989286)

[5.3 if 语句 13](#_Toc264989287)

[5.4 循环语句的效率 15](#_Toc264989288)

[5.5 for 语句的循环控制变量 16](#_Toc264989289)

[5.6 switch语句 16](#_Toc264989290)

[6 排版 16](#_Toc264989291)

[6.1 空格与对齐 16](#_Toc264989292)

[6.2 注释 21](#_Toc264989293)

[7 内存管理 22](#_Toc264989294)

[7.1 创建与销毁 22](#_Toc264989295)

[7.2 数组与指针 22](#_Toc264989296)

## 介绍

### 文档目的

本文档规定了C++编程过程中的编码规范，本文档的目的是为了帮助本学院的学生，教师更好的完成程序设计任务。所有的程序作者都应该遵循此文档所规定的编码规范。如果你对于文档中规范有不同的见解或更好的解决方案，在确保自己的解决方案经过充分的记录和测试的前提下，请及时通知本文的作者，以便对该文档相应的规则做出必要的更改。

### 动机

为了对软件的开发进行适当的规范化，特制定本规范，其根本目的，是为了保证程序具有良好的、一致的结构，以期提高程序的可读性及可维护性，方便程序的测试、维护、升级等工作，同时，也用以培养学生书写规范代码的习惯。

## 一般性原则

为了书写出清晰而易维护的代码，以下的一般性原则可以在所有的情况下应用到:

* + 重用性。软件工程师的一个目标就是通过重复使用代码来避免编写新的代码。因为重新使用已有的代码可以降低成本、增加代码的可靠性并提高它们的一致性。
  + 可维护性。一段风格良好的代码肯定具有很好的可读性。如果代码的维护者不明白你所书写的代码的含义，那么代码的维护就会成为一件十分困难而耗时的工作，所以在你编写代码的时候，一定要时刻的考虑到你所书写的代码别人是不是能够真正理解。
  + 模块化，封装和信息隐藏。如果一个模块的代码过长或者过于复杂，一定要考虑你的模块是不是需要重新组织一下，或者将一个模块分解成多个模块。
  + 不要对使用你代码的用户抱有太多的幻想。要让代码保护自己不被错误使用。
  + 书写注释。注释可以帮助别人更好的理解你的代码，尽管这一条经常被软件开发人员忽视，但是对于代码的作者，使用者或者维护人员来说，代码注释在提高代码的可读性和可维护性方面是十分必要的。
  + 保持代码风格的一致性，无论是在文件级别，模块级别，还是工程级别，请保持代码风格的一致性。
  + 变量、函数、类和名字空间的命名要使用有意义的英文单词，且命名不要过长，简明易懂为佳，尽量不要使用缩写，更不要使用汉语拼音。

## 文件结构

### 一般性问题

* + 文件名使用字母、数字、下划线(不推荐)的组合来命名，但命名要有意义。
  + 所有的目录名，文件名一律用小写字母书写。
  + 一个文件中只能定义或者实现一个类，唯一的例外是小型的帮助类或私有的内部类可以和它的主类定义在同一个文件中。
  + 文件中不允许包含废弃不用的源代码。

### 头文件

* + 为了防止头文件被重复引用，应当用#ifndef/#define/#endif结构产生预处理块。
  + 用#include <filename> 格式来引用标准库的头文件（编译器将从标准库目录开始搜索）。
  + 用#include "filename.h" 格式来引用非标准库的头文件（编译器将从当前目录开始搜索，若没有找到则从标准库目录搜索）。
  + 一般情况下，将构造函数和析构函数声明在类的开头位置。
  + 将从父类继承的函数放在不同的位置，并添加注释说明这些成员函数是从哪个类继承而来的
  + 声明写在头文件中，定义写在源文件中。
  + 避免使用全局变量，尽量不要在头文件中出现象extern int value 这类声明。
  + 尽量用类中公共的静态成员函数取代全局函数。
  + 类中的访问控制符排列顺序为public、protected、private。用相同访问控制符修饰的方法和数据成员尽量写到一起。

|  |
| --- |
| C++头文件的结构 |
| //------------------------------------------------------------  //  // Copyright © 2009, Software College of Hebei Normal University  //  // File Name: filename.h  // Description:  //  //  // Author:  // Date:  // Version:  //-----------------------------------------------------------  #ifndef GRAPHICS\_H    // 防止graphics.h被重复引用  #define GRAPHICS\_H    #include <math.h>      // 引用标准库的头文件  …  #include "myheader.h"      // 引用非标准库的头文件  …  class Common  {  public:  static void Function1(…);    // 全局函数声明  …  };  class Box              // 类结构声明  {  public:  Box();  ~Box();  int getSize(void) const;  string getName(void) const;  protected:  int calculateSide(void);  private:  int m\_intSide;  };  #endif |

### 源文件的结构。

* + 书写注释，源文件的注释是为这个模块的维护者书写的，源文件的注释应该满足以下条件：
    1. 注释要描述所有类级别的实现，详细描述复杂的和容易产生歧义的实现部分。
    2. 成员函数的作用，目的以及返回值和参数都需要添加必要的注释部分。
    3. 注释的更改和更新要同步，对于核心代码使用块注释，不仅写明是什么，还要写清为什么，这样更便于日后维护。
    4. 避免使用尾注释，除非是对变量声明的注释。
  + 在同一个源文件中写一个类的实现部分。
  + 全局函数和成员函数的定义要和头文件中的声明顺序一致。
  + 声明函数时不要省略形参变量名，且同函数的参数名要和头文件中保持一致。

|  |
| --- |
| C++源文件的结构 |
| // 版权和版本声明见示例1-1，此处省略。    #include "graphics.h"         // 引用头文件  …    // 全局函数的实现体  void Function1(…)  {  …  }    // 类成员函数的实现体  void Box::Draw(…)  {  …  } |

### 目录结构

* + 如果一个软件的头文件数目比较多（如超过十个），通常应将头文件和定义文件分别保存在不同的目录，以便于维护。

## 命名和书写规范

### 标识符的命名规则

* + 不允许使用汉语拼音命名，更不要使用汉语，尽量英文来命名标识符，因为计算机语言的保留字都用英文命名，而且在国际化的组织中，英文一般作为主要的沟通语言。
  + 用描述性的文字来命名标识符，如果你不能为一个标识符命名，你一定要考虑这个标识符是否还有存在的必要。
  + 尽量避免名字中出现数字编号，如Value1,Value2等，除非逻辑上的确需要编号。
  + 类，结构的名字开头字母要大写
  + 标识符的命名要清晰、明了，有明确含义，同时使用完整的单词或大家基本可以理解的缩写，避免使人产生误解。
  + 命名类时应使用名词，命名函数时使用动词或动宾短语或isXXX等。

### 类

* + 必须以大写开头，后面字母反映具体含义，例如：Student, Point。
  + 接口类必须以大写"I"开头，代表 Interface。
  + 类中成员的声明顺序严格按照访问控制符public, protected 和private的顺序。
  + 将数据成员放在类的private访问控制符下，非特殊情况，不允许在protected和public的限定符下声明数据成员。
  + 尽量的用class声明一个类，而不是struct。
  + 将析构函数声明为虚函数，以保证父类可以正确释放子类的资源。
  + 尽可能减少友元friend的使用。
  + 尽可能的为类提供一个默认构造函数。
  + 用inline函数代替宏定义。
  + 如果类A和类B毫不相关，不可以为了使B的功能更多些而让B继承A的功能和属性。
  + 若在逻辑上B是A的“一种”（a kind of ），则允许B继承A的功能和属性。例如男人（Man）是人（Human）的一种，男孩（Boy）是男人的一种。那么类Man可以从类Human派生，类Boy可以从类Man派生。
  + 若在逻辑上A是B的“一部分”（a part of），则不允许B从A派生，而是要用A和其它东西组合出B。
  + 采用以函数为中心的版式，将public类型的函数写在前面，而将private类型的数据写在后面，采用这种版式的程序员主张类的设计“以行为为中心”，重点关注的是类应该提供什么样的接口。

|  |  |
| --- | --- |
| 以数据为中心版式 | 以函数为中心的版式（推荐的版式） |
| class A  {   private:  int i, j;  float x, y;      …   public:  void Func1(void);  void Func2(void);  …  } | class A  {   public:  void Func1(void);  void Func2(void);  …   private:  int i, j;  float x, y;      …  } |

### 函数

* + 声明函数时，参数的书写要完整，不要贪图省事只写参数的类型而省略参数名字。
  + 避免函数有太多的参数，参数个数尽量控制在5个以内。如果参数太多，在使用时容易将参数类型或顺序搞错。
  + 尽量不要使用类型和数目不确定的参数。
  + 通过函数默认值减少函数重载的个数。
  + 函数参数以小写a开头(例如：aPara),表示这个变量是参数(argument)变量，从而使参数和临时变量，数据成员分开

例子：

// testclass.h:

//----------------------------------------------------------------

// Test class description.

//----------------------------------------------------------------

class TestClass

{

public:

int memberFunction( int **aNumber** );

...

// testclass.cpp:

...

// -----------------------------------------------------------

// Implementation description.

// -----------------------------------------------------------

int TetrisClass::memberFunction( int **aNumber** )

{

...

}

* + 如果参数是指针，且仅作输入用，则应在类型前加const，以防止该指针指向的对象在函数体内被意外修改。
  + 如果输入参数以值传递的方式传递对象，则宜改用“const &”方式来传递，这样可以省去临时对象的构造和析构过程，从而提高效率。
  + 不要省略返回值的类型。
  + 函数名字与返回值类型在语义上不可冲突。
  + 有时候函数原本不需要返回值，但为了增加灵活性如支持链式表达，可以附加返回值。
  + 在函数体的“入口处”，对参数的有效性进行检查，检查时提倡使用assert断言函数。
  + 函数名称涉及多个单词时，从第二个单词开始，首字母大写。如：getName()。

### 数据成员

* + 用m\_开头来命名数据成员，例如: int m\_intValue;

### 指针和引用

* + 应当将修饰符 \* 和＆紧靠变量名

示例：

char \*name;

int   \*x, y; // 此处y不会被误解为指针

TestClass \*pointer; // pointer to TestClass

void Modify( Tint &aInteger ); // reference parameter used

### 宏定义

* + 宏名全部使用大写，多个单词时，使用下划线隔开，如：NAME\_SIZE。
  + 尽量避免使用对象宏，替代时，一般用const常量替代对象宏，用内联函数或模板替代函数宏。

### const常量

* + 在C++ 程序中只使用const常量而不使用宏常量，即const常量完全取代宏常量。
  + 常量定义以大写字母K开头，代码中的字符，大于0，小于-1的整型变量及其他非整型变量一律用const常量定义

例如：

     const int KMagicNumber = 100;

     const string KUserName = "username";

* + const常量定义在源文件中，尽量避免在头文件中定义const常量
  + 如果某一const常量与其它const常量密切相关，应在定义中包含这种关系，而不应给出一些孤立的值。

例如：

const float   KRadius = 100;

const float   KDiameter = KRadius \* 2;

### 枚举变量

* + 枚举变量定义应该包含在特定的类中。
  + 枚举和它的成员的定义，应该采用有意义的命名方式，避免出现歧义。
  + 枚举成员以E开头。
  + 枚举常量不会占用对象的存储空间，它们在编译时被全部求值。枚举常量的缺点是：它的隐含数据类型是整数，其最大值有限，且不能表示浮点数（如PI=3.14159）。

例子 :

class TestClass

{

...

enum TestType

{

ETestEumFist,

ETestEumSecond,

...

};

...

};

### 全局变量

* + 尽量避免应用全局变量，如果有特殊情况必须定义的话，全局变量以小写g开头

## 表达式和基本语句

### 运算符的优先级

* + 一元运算符 + - \* 的优先级高于对应的二元运算符。
  + 如果代码行中的运算符比较多，用括号确定表达式的操作顺序，避免使用默认的优先级。

例如：

word = (high << 8) | low

if ((a | b) && (a & c))

### 复合表达式

* + 不要编写太复杂的复合表达式。

例如：

   i = a >= b && c < d && c + f <= g + h ;   // 复合表达式过于复杂

* + 不要有多用途的复合表达式。

例如：

d = (a = b + c) + r ;

该表达式既求a值又求d值。应该拆分为两个独立的语句：

a = b + c;

d = a + r;

* + 不要把程序中的复合表达式与“真正的数学表达式”混淆。

例如：

if (a < b < c)              // a < b < c是数学表达式而不是程序表达式

并不表示

if ((a<b) && (b<c))

而是成了令人费解的

if ( (a<b)<c )

### if 语句

* + 不可将布尔变量直接与TRUE、FALSE或者1、0进行比较。

根据布尔类型的语义，零值为“假”（记为FALSE），任何非零值都是“真”（记为TRUE）。TRUE的值究竟是什么并没有统一的标准。例如Visual C++ 将TRUE定义为1，而Visual Basic则将TRUE定义为-1。

假设布尔变量名字为flag，它与零值比较的标准if语句如下：

if (flag) // 表示flag为真

if (!flag)    // 表示flag为假

其它的用法都属于不良风格，例如：

     if (flag == TRUE)

     if (flag == 1 )

     if (flag == FALSE)

     if (flag == 0)

* + 应当将整型变量用"=="或"！="直接与0比较。

     假设整型变量的名字为value，它与零值比较的标准if语句如下：

if (value == 0)

if (value != 0)

不可模仿布尔变量的风格而写成

if (value)         // 会让人误解 value是布尔变量

if (!value)

* + 不可将浮点变量用"=="或"！="与任何数字比较。

     千万要留意，无论是float还是double类型的变量，都有精度限制。所以一定要避免将浮点变量用"=="或"！="与数字比较，应该设法转化成">="或"<="形式。

     假设浮点变量的名字为x，应当将

if (x == 0.0) // 隐含错误的比较

转化为

if ((x>=-EPSINON) && (x<=EPSINON))

其中EPSINON是允许的误差（即精度）。

* + 应当将指针变量用"=="或"！="与NULL比较。

     指针变量的零值是“空”（记为NULL）。尽管NULL的值与0相同，但是两者意义不同。假设指针变量的名字为p，它与零值比较的标准if语句如下：

         if (p == NULL)     // p与NULL显式比较，强调p是指针变量

         if (p != NULL)

不要写成

         if (p == 0)  // 容易让人误解p是整型变量

         if (p != 0)

     或者

if (p)             // 容易让人误解p是布尔变量

     if (!p)

### 循环语句的效率

* + 在多重循环中，如果有可能，应当将最长的循环放在最内层，最短的循环放在最外层，以减少CPU跨切循环层的次数。

|  |  |
| --- | --- |
| 低效率：长循环在最外层 | 高效率：长循环在最内层 |
| for (row=0; row<100; row++)  {  for ( col=0; col<5; col++ )  {  sum = sum + a[row][col];  }  } | for (col=0; col<5; col++ )  {  for (row=0; row<100; row++)  {      sum = sum + a[row][col];  }  } |

* + 如果循环体内存在逻辑判断，并且循环次数很大，宜将逻辑判断移到循环体的外面。

|  |  |
| --- | --- |
| 效率低但程序简洁 | 效率高但程序不简洁 |
| for (i=0; i<N; i++)  {  if (condition)      doSomething();  else      doOtherthing();  } | if (condition)  {  for (i=0; i<N; i++)  doSomething();  }  else  {  for (i=0; i<N; i++)      doOtherthing();  } |

### for 语句的循环控制变量

* + 不可在for 循环体内修改循环变量，防止for 循环失去控制。
  + 循环变量应定义在循环内，如for(int i = 0 ; i < 10; i++)。

### switch语句

* + 每个case语句的结尾不要忘了加break，否则将导致多个分支重叠（除非有意使多个分支重叠）。
  + 如果使用多个分支重叠，请在每个分支后面加注释说明“//fall through”。
  + 为提升代码的可读性，多于2条语句的case分支需要使用花括号“{}”标注作用域。
  + 不要忘记最后那个default分支。即使程序真的不需要default处理，也应该保留语句   default : break; 这样做并非多此一举，而是为了防止别人误以为你忘了default处理。

## 排版

### 空格与对齐

* + 程序块要采用缩进风格编写，缩进的空格数为4个
  + 在每个类声明之后、每个函数定义结束之后都要加空行, 在一个函数体内，逻揖上密切相关的语句之间不加空行，其它地方应加空行分隔。
  + 示例：

|  |  |
| --- | --- |
| 函数之间的空行 | 函数内部的空行 |
| // 空行  void function1(…)  {  statement1;  }  // 空行  void function2(…)  {  statement1;  }  // 空行  void function3(…)  {  statement1;  } | // 空行  while (condition)  {      statement1;      // 空行      if (condition)      {          statement2;      }      else      {          statement3;      }  // 空行      statement4;  } |

* + 较长的语句（>80字符）要分成多行书写，长表达式要在低优先级操作符处划分新行，操作符放在新行之首，划分出的新行要进行适当的缩进，使排版整齐，语句可读。

示例：

|  |
| --- |
| if ((very\_longer\_variable1 >= very\_longer\_variable12)  && (very\_longer\_variable3 <= very\_longer\_variable14)  && (very\_longer\_variable5 <= very\_longer\_variable16))  {      doSomething();  } |
| virtual Matrix MultiplyMatrix ( Matrix leftMatrix,                                  Matrix rightMatrix ); |
| for ( very\_longer\_initialization;        very\_longer\_condition;        very\_longer\_update )  {       dosomething();  } |

* + 不允许把多个短语句写在一行中，即一行只写一条语句；if、for、while、do等语句自占一行，执行语句不得紧跟其后。

示例：

|  |  |
| --- | --- |
| 风格良好的代码行 | 风格不良的代码行 |
| int width;    // 宽度  int height;   // 高度  int depth;    // 深度 | int width, height, depth; // 宽度高度深度 |
| x = a + b;  y = c + d;  z = e + f; | x = a + b;   y = c + d; z = e + f; |
| if (width < height)  {  dosomething();  } | if (width < height) dosomething(); |
| for (initialization; condition; update)  {  dosomething();  }  // 空行  other(); | for (initialization; condition; update)       dosomething();  other(); |

* + 对齐只使用空格键，不使用TAB键。

说明：以免用不同的编辑器阅读程序时，因TAB键所设置的空格数目不同而造成程序布局不整齐。

* + 函数或过程的开始、结构的定义及循环、判断等语句中的代码都要采用缩进风格，case语句下的情况处理语句也要遵从语句缩进要求。
  + 在多个结构嵌套时，尾部的大括号极容易混淆，请在每个尾部的大括号后说明。

示例：

for(int i = 0; i < 10 ; i++)

{

for(int k = 0 ; k < 10 ; k++)

{

if(k > 3)

{

……

}//if 结束

}循环k 结束

}循环i结束

* + 尽可能在定义变量的同时初始化该变量（就近原则）

示例：

int width = 10;    // 定义并初绐化width

int height = 10;   // 定义并初绐化height

int depth = 10;    // 定义并初绐化depth

* + 程序块的分界符（如C/C++语言的大括号‘{’和‘}’）应各独占一行并且位于同一列，同时与引用它们的语句左对齐。在函数体的开始、类的定义、结构的定义、枚举的定义以及if、for、do、while、switch、case语句中的程序都要采用如上的缩进方式。

示例：

|  |  |
| --- | --- |
| 风格良好的对齐 | 风格不良的对齐 |
| void Function(int x)  {  … // program code  } | void Function(int x){  … // program code  } |
| if (condition)  {  … // program code  }  else  {  … // program code  } | if (condition){  … // program code  }  else {  … // program code  } |
| for (initialization; condition; update)  {  … // program code  } | for (initialization; condition; update){  … // program code  } |
| While (condition)  {  … // program code  } | while (condition){  … // program code  } |

### 注释

* + 一般情况下，源程序有效注释量必须在20％以上，注释的原则是有助于对程序的阅读理解，在该加的地方都加了，注释不宜太多也不能太少，注释语言必须准确、易懂、简洁。
  + 版权和版本的声明位于头文件和定义文件的开头

|  |
| --- |
| 版权和版本的声明 |
| //---------------------------------------------------------------  //  // Copyright (c) 2009, Software College of Hebei Normal University  //  // File name: filename.h  //  // Description:  //  // Author:  // Date:  // Version:  //--------------------------------------------------------------- |

* + 函数头部应进行注释，列出：函数的目的/功能、输入参数、输出参数、返回值、调用关系（函数、表）等。

|  |
| --- |
| 函数的注释 |
| //-------------------------------------------------------------  //  // Function: //Function name  //  // Description: filename.h  //  //------------------------------------------------------------- |

* + 边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除。
  + 注释的内容要清楚、明了，含义准确，防止注释二义性。
  + 避免在注释中使用缩写，特别是非常用缩写。

## 内存管理

### 创建与销毁

* + 不要混用C和C++的内存操作函数和操作符
  + 用new申请内存之后，应该立即检查指针值是否为NULL。防止使用指针值为NULL的内存。
  + 用delete释放了内存或对象之后，立即将指针设置为NULL，防止产生“野指针”。
  + 回收用 new[] 分配的一组对象的内存空间的时候用 delete[]
  + 动态内存的申请与释放必须配对，防止内存泄漏。

### 数组与指针

* + 不要忘记为数组和动态内存赋初值。防止将未被初始化的内存作为右值使用。
  + 避免数组或指针的下标越界，特别要当心发生“多1”或者“少1”操作。