基础C++程序分析与设计

马瑞新 田琳琳 赖晓晨 编著

大连理工大学出版社
DALIAN UNIVERSITY OF TECHNOLOGY PRESS

基础C++程序分析与设计

马瑞新　田琳琳　赖晓晨　编著

大连理工大学出版社

图书在版编目(CIP)数据

基础C++程序分析与设计/马瑞新，田琳琳，赖晓晨编著．—大连：大连理工大学出版社，2007.3
ISBN 978-7-5611-3498-6

Ⅰ.基… Ⅱ.①马… ②田… ③赖… Ⅲ.C语言—程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字(2007)第027323号

大连理工大学出版社出版
地址：大连市软件园路80号　邮编：116023
发行：0411-84708842　邮购：0411-84703636　传真：0411-84701466
E-mail：dzcb@dutp.cn　URL：http://www.dutp.cn
大连理工印刷有限公司印刷　　大连理工大学出版社发行

幅面尺寸：185mm×260mm　　印张：12.75　　字数：293千字
2007年3月第1版　　2007年3月第1次印刷

责任编辑：高智银　　责任校对：达　理
封面设计：宋　蕾

定价：22.80元(含1CD)

前　言

仔细想想，与其把这篇短文当作本书序言，不如把它当作一篇引导初学者步入C++殿堂的方法论。八年并不算很长的编程经验，使我感触颇深。痛苦迷茫以及成功后的喜悦一直交替伴随着我。爱好编程的我，在学习期间也被很多很多学习的困难疑惑所阻挡，甚至不止一次地想放弃。让我初次领略到作为一名程序员应该有快乐与喜悦的是C++语言。这些年为了工作，也学习过很多其他的语言，包括时下正在流行的Java与C#等等。在这里不得不说的是，作为一名程序员，一名能够适应当前中国工作环境的程序员，要学的还远不止这些，包括数据系统，等等。

很早就听人说过，如果你是一名程序员，如果你没有学过C++语言，那么就不能算作是一名真正的程序员。这句话也许有点夸张，不过当你学习过它以后就不得不承认这句话真的有那么几分道理。C++语言是由C语言发展而来的一种新的支持面向对象的语言，从一开始接触它，它的魅力就深深地吸引着我，起初我和很多想要学习它的人一样，觉得它很难懂，当时为了看懂、想明白一些现在看来真的很容易的例子时，真的有点让我苦闷。

作为一种灵活性高，体系庞大，支持面向对象的高级语言，C++的确比其他语言更难学习。很多正在学习它的在校大学生，以及很多正在从事编程工作想自学它的程序员，对于它的入门及必要知识点掌握苦恼不已。

其实学习C++和学习其他知识一样，并没有很多特别的要求，需要的只是那么一点点耐心，那么一点点忍耐力，以及遇到困难挫折不轻易屈服的精神。更重要的一点是你必须有一本好的、适合你的入门书籍指引你逐渐进步。在行内，很多人是不愿意和你分享学习经验的，一来工作任务繁重，再者很多人觉得这样会给自己带来更多的竞争对手。这些我都亲身经历过，正是因为此类的种种感受，于是我由内心而发，真心实意地愿意帮助那些正在学习C++语言、想入门的朋友们。

从一开始学习计算机语言，几乎所有的人都会问到，我该如何入门，入门后又该如何继续持久地进步下去呢？首先我要说的是，计算机语言的入门，无论是C/C++还是Java、C#，它们都和高等数学没有太多联系，计算机语言是一种逻辑的文字描述，体现逻辑的是思想，当你通过理解一些计算机语言所描述的，并不算难的逻辑问题后，你就已经掌握了语言本身，并且真正入门了。

当然，计算机语言毕竟是和数学有密切联系的产物。在计算机世界里他们彼此依赖，谁也离不开谁。当你正确理解编程思想，掌握必要知识点，入门之后，如果想做一名优秀的程序员而不是单单做一名程序的“拼装工”，你就不得不对数学知识进行进一步的系统学习。对于C++的学习，创建一条由入门到深入最后到精通的可持续学习并不断发展的道路，概括起来可以是以下顺序：

(1)学习一些基本的C++语言知识，例如：什么是变量，什么是函数。

(2)C++语言入门，基本知识点的掌握。

(3)高等数学及工程数学的系统学习，结合实际了解并使用C++的各类常用的标准库。此外平时可以找一些广受好评的具备一定深度的C++教材看一下，进一步理解C++的高级编程

精髓，以及看一些计算机原理和数据结构方面的书籍。

(4)学习包括在各类操作系统下编程的必要知识，以Windows操作系统为例，需要学习掌握Windows API，以及高效开发的MFC，VCL等在内的其他知识。

学习语言必须要通过编程，要成为一个编程高手，有哪些途径呢？下面是成为编程高手的二十二条军规，只要你像军人一样遵守这些军规，那么你已经是战场上的常胜将军了。

(1)大学生活丰富多彩，会令你一生都难忘，但难忘有很多种，你可以因为学了很多东西而难忘，也会因为什么都没学到而难忘！

(2)计算机专业是一个很枯燥的专业，但既来之、则安之，只要你努力学，也会发现其中的乐趣。

(3)记住：万丈高楼平地起！基础很重要，尤其是专业基础课，只有打好基础才能学得更深。

(4)C++语言是基础，很重要，如果你不学好C++语言，那么什么高级语言你都学不好。

(5)C语言与C++语言是两回事。就像大熊猫和小熊猫一样，只是名字很像。

(6)请学习好专业课《数据结构》、《计算机组成原理》，不要刚开始就拿着一本VC在看，你连面向对象都搞不清楚，看VC没有任何用处。

(7)对编程有一定的认识后，就可以学习C++了。是C++而不是VC，这两个也是两码事。C++是一门语言，而VC教程则是讲解如何使用MFC类库，学习VC应建立在充分了解C++的基础之上。看VC的书，是学不了C++语言的。

(8)学习编程的秘诀是：编程，编程，再编程。

(9)认真学习每一门专业课，那是你今后的饭碗。

(10)在学校的实验室就算你做错一万次程序都不会有人骂你，如果在公司你试试看！所以多去实验室上机，现在错得多了，毕业后就错得少了。

(11)从现在开始，在写程序时就要养成良好的习惯。

(12)不要漏掉书中任何一个练习题——请全部做完并记录下解题思路。

(13)你会买好多参考书，那么请把书上的程序例子亲手输入到电脑上实践，即使配套光盘中有源代码。

(14)VC、C#、.NET这些东西都会过时，不会过时的是数据结构和优秀的算法。

(15)记住：书到用时方恨少。不要让这种事发生在你身上，在学校你有充足的时间和条件读书，多读书，如果有条件多读原版书，你要知道，当一个翻译者翻译一本书时，他会不知不觉把他的理念写进书中，那本书就会变得像鸡肋。

(16)我还是要强调认真听专业课，因为有些课像《数据结构》、《编译原理》、《操作系统》等等，这种课老师讲一分钟能让你明白的内容，你自己看要看好几个月，有的甚至看了好几年都看不明白。

(17)抓住在学校里的各种实践的机会，要为自己积累经验，就业时经验比什么都有用。

(18)多去图书馆，每个学校的图书馆都有很多好书等你去看。

(19)编程不是技术活，而是体力活。

(20)如果你决定了要当一名好的程序员，那么请你放弃游戏，除非你是那种每天只要玩游戏就能写出好程序的天才。

(21)你要有足够的韧性和毅力。

(22)找到只属于你自己的学习方法。不要盲目地追随别人的方法，适合自己的才是最好

的。

千里之行，始于足下。路虽然很长，困难也会很多，不过你一旦入门，一定会觉得面前豁然开朗，会不断地激励着你学习下去的。同时也建议程序高手们，把你的优秀的经验和思想奉献出来，帮助更多需要的人，毕竟思想是需要沟通的，知识是需要共享的，快乐是需要传递的。

关于本书

本书是为软件学院量身打造的，主要讨论C++程序设计的基础部分，不涉及C++高级应用部分，包括类、继承、多态、模板等，这是在C++高级程序设计课程中要学习的内容。这样做的目的就是想通过扎实的基础训练，让初学者一步步走入编程殿堂，为后续课程打下良好的基础。配套光盘中包括本书所有案例的代码，方便读者学习、程序调试使用。

由于编写水平和时间有限，有不当之处请批评指正。

张宪超

2007 年 2 月

目　录

第1章　编程修养

通常一本书的第一章应该是对整体内容的一个大概介绍，然而学习程序设计，特别是对初学者，形成良好的编程修养是首要基础。正像张君宝当年什么武功都不会，每天都听觉远师父念诵金刚经，若干年后突然大彻大悟，成为一代宗师张三丰。当我们一开始就养成良好的编程修养并坚持下去，一定有一天也会仰天长笑，成为一名优秀的程序员。

什么是优秀的程序员？是懂得很多技术细节？还是懂底层编程技术？还是编程速度比较快？我觉得都不是。对于一些技术细节和底层的技术来说，只要看帮助，查资料就能找到，对于速度快，只要编得多也就熟能生巧了。

好的程序员应该有以下几方面的素质：

- 有钻研精神，勤学善问、举一反三；
- 有积极向上的态度，有创造性思维；
- 有与人积极交流沟通的能力，有团队精神；
- 谦虚谨慎，戒骄戒躁；
- 写出的代码质量高，包括：代码的稳定、易读、规范、易维护、专业。

这些都是程序员的修养，这里的“编程修养”，也就是上述中的最后一点。如果要了解一个作者，我会看他所写的文章；如果要了解一个画家，我会看他所画的图画；如果要了解一个工人，我会看他所做出来的产品；同样，如果要了解一个程序员，首先我最想看的就是他的程序代码。程序代码可以看出一个程序员的素质和修养，程序就像一个作品，有素质有修养的程序员的作品必然是一幅精美的图画，一首美妙的歌曲，一本赏心悦目的小说。我看过许多程序，没有注释，没有缩进，胡乱命名的变量名，等等。我把这种人统称为没有修养的程序员，这种程序员，是在做创造性的工作吗？不，完全就是在搞破坏，他们与其说是在编程，还不如说是在对源程序进行“加密”，这种程序员，见一个就应该开除一个，因为他编程序所创造的价值，远远小于需要在上面进行维护的价值。

程序员应该有程序员的修养，哪怕再累，再没时间，也要对自己编的程序负责。我宁可要那种动作慢，技术一般，但有良好的编写程序风格的程序员，也不要那种技术强、动作快的“搞破坏”的程序员。有句话叫“字如其人”，我想从程序上也能看出一个程序员的优劣。因为，程序是程序员的作品，作品的好坏直接关系到程序员的声誉和素质。而“修养”好的程序员一定能做出好的程序和软件。

有个成语叫“独具匠心”，意思是做什么都要做得很专业，很用心，如果你要做一个“匠”，也就是造诣高深的人，那么，从一件很简单的作品上就能看出你有没有“匠”的特性，我觉得做一个程序员不难，但要做一个“程序匠”就不简单了。编程序很简单，但编出有质量的程序就难了。

在这里不讨论过深的技术，只想在一些容易让人忽略的问题上说一说，虽然这些问题可能很细微，但如果你不注意这些细微之处的话，那么它将会极大地影响整个软件的质量，以及整个软件工程的实施，所谓“千里之堤，毁于蚁穴”，“细微之处见真功”，真正能体现一个程序员的功底恰恰在这些细微之处。

这就是程序员的编程修养。我总结了在用C++语言(主要是C++语言基础部分)进行程序写作上的27个“修养”,通过这些,你可以写出高质量的程序,同时也会让看你程序的人啧啧称道,那些看过你程序的人一定会说:“这个人的编程修养不错”。

下面对这些规定进行详细解释。

1. 版权和版本

好的程序员会给自己的每个函数,每个文件,都注上版权和版本。

对于C++的文件,文件头应该有类似这样的注释:

```
/*********************************************************************************
*
*   文件名:network.cpp
*   文件描述:网络通讯函数集
*
*   创建人: Mike, 2006 年 2 月 3 日
*
*   版本号:1.0
*
*   修改记录:
*
*
*********************************************************************************/
```

而对于函数来说,应该也有类似于这样的注释:

```
/* ================================================================
*
* 函数名:XXX
*
* 参   数:
*
*         type name [IN] : descriptions
*
* 功能描述:
*
*         ......
*
* 返回值:成功 true,失败 false
*
* 抛出异常:
*
* 作      者:Mike,2006/4/2
*
*
================================================================ */
```

这样的描述可以让人对一个函数、一个文件有一个总体的认识,对提高代码的易读性和易维护性有很大的好处。这是好的作品产生的开始。

2. 缩进、空格、换行、空行、对齐

(1)缩进。缩进应该是每个程序员都会做的,只要学过程序就应该知道这个,但是我仍然

看过不缩进的程序，或是乱缩进的程序，如果你的公司还有写程序不缩进的程序员，请毫不犹豫地开除他吧，并以破坏源码罪起诉他，还要他赔偿读过他程序的人的精神损失费。缩进，这是不成文规矩，再重提一下，一个缩进一般是一个 Tab 键或是 4 个空格。（最好用 Tab 键）

(2)空格。空格能给程序带来什么损失吗？没有有效的利用空格可以让你的程序读起来更加赏心悦目，而不是一堆表达式挤在一起。看看下面的代码：

```
ha=(ha*128+*key++)%tabPtr->size;          //不加空格，不提倡使用

ha = ( ha * 128 + *key++)%tabPtr->size;    //加空格，建议使用
```

有空格和没有空格的感觉不一样吧。一般来说，语句中要在各个操作符间加空格，函数调用时，要在各个参数间加空格。如下面这种加空格的和不加空格的：

```
if ((hProc=OpenProcess(PROCESS_ALL_ACCESS,FALSE,pid))==NULL)  //不加空格，不提倡使用
{
}

if ( ( hProc = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid) ) == NULL )    //加空格，建议使用
{
}
```

(3)换行。不要把语句都写在一行上，这样很不好。如：

```
for(i=0;i<len;i++) if((a[i]<'0'||a[i]>'9')&&(a[i]<'a'||a[i]>'z')) break;
```

这种既无空格，又无换行的程序在写什么啊？加上空格和换行吧。

```
for ( i=0; i<len; i++)
{
    if ( ( a[i] < '0' || a[i] > '9' ) && ( a[i] < 'a' || a[i] > 'z' ) )
    {
        break;
    }
}
```

好多了吧？有时候，函数参数多的时候，最好也换行，如：

```
CreateProcess ( NULL,
                cmdbuf,
                NULL,
                NULL,
                bInhH,
                dwCrtFlags,
                envbuf,
                NULL,
                &siStartInfo,
                &prInfo
                );
```

条件语句也应该在必要时换行：

```
if ( ch >= '0' || ch <= '9' ||
    ch >= 'a' || ch <= 'z' ||
    ch >= 'A' || ch <= 'Z' )
```

(4)空行。不要不加空行，空行可以区分不同的程序块，程序块间，最好加上空行。如：

```
HANDLE hProcess;
PROCESS_T procInfo;

/* open the process handle */
if((hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid)) == NULL)
{
    return LSE_MISC_SYS;
}

memset(&procInfo, 0, sizeof(procInfo));
procInfo.idProc = pid;
procInfo.hdProc = hProcess;
procInfo.misc = MSCAVA_PROC;

return(0);
```

(5)对齐。用 Tab 键对齐一些变量的声明或注释,同样会让你的程序好看一些。如:

```
typedef struct _pt_man_t_ {
    int     numProc;   /* Number of processes                    */
    int     maxProc;   /* Max Number of processes                */
    int     maxProc;   /* Max Number of processes                */
    int     numEvnt;   /* Number of events                       */
    int     maxEvnt;   /* Max Number of events                   */
    HANDLE* pHndEvnt;  /* Array of events                        */
    DWORD   timeout;   /* Time out interval                      */
    HANDLE  hPipe;     /* Namedpipe                              */
    TCHAR   usr[MAXUSR]; /* User name of the process             */
    int     numMsg;    /* Number of Message                      */
    int     Msg[MAXMSG]; /* Space for intro process communicate  */
} PT_MAN_T;
```

好看的代码会让人的心情愉快,读代码也就不累,工整、整洁的程序代码,通常更让人欢迎,也更让人称道。现在的硬盘空间这么大,不要让你的代码挤在一起,这样它们会抱怨你虐待它们的。好了,用"缩进、空格、换行、空行、对齐"装饰你的代码吧,让它们从没有秩序的土匪中变成一排排整齐有秩序的正规部队吧。

3. 程序注释

养成写程序注释的习惯,这是每个程序员所必须要做的工作。我看过那种几千行,却居然没有一行注释的程序。这就如同在公路上驾车却没有路标一样。用不了多久,连自己都不知道自己的意图了,还要花上几倍的时间才能看明白,这种浪费别人和自己时间的人,是最为可耻的。注释的书写也能看出一个程序员的功底。一般来说你需要至少写这些地方的注释:文件的注释、函数的注释、变量的注释、算法的注释、功能块的程序注释。主要就是记录你这段程序是干什么的,你的意图是什么,你这个变量是用来做什么的,等等。

不要以为注释好写,有一些算法是很难说或写出来的,只能意会,我承认有这种情况的时候,但你也要写出来,正好可以训练一下自己的表达能力。而表达能力正是那种闷头搞技术的技术人员最缺的,你有再高的技术,如果你表达能力不行,你的技术将不能得到充分的发挥。因为,这是一个团队的时代。下面是几个注释的技术细节:

(1)对于行注释("//")比块注释("/* */")要好的说法,我并不是很同意。因为一些老版本的C++编译器并不支持行注释,所以为了你的程序的移植性,请你还是尽量使用块注释。

(2)你也许会为块注释的不能嵌套而不舒服,那么你可以用预编译来完成这个功能。使用"# if 0"和"# endif"括起来的代码,将不被编译,而且还可以嵌套。

4. 函数的[in][out]参数

我经常看到这样的程序:

```
FuncName(char * str)
{
    int len = strlen(str);
    ......
}

char * GetUserName(user * pUser)
{
    return pUser->name;
}
```

不! 请不要这样做。你应该先判断一下传进来的那个指针是不是为空。如果传进来的指针为空的话,那么,你的一个大的系统就会因为这一个小的函数而崩溃。一种更好的技术是使用断言(assert),这里我就不多说这些技术细节了。当然,如果是在C++中,引用要比指针好得多,但你也需要对各个参数进行检查。

写有参数的函数时,首要工作就是要对传进来的所有参数进行合法性检查。而对于传出的参数也应该进行检查,这个动作当然应该在函数的外部,也就是说,调用完一个函数后,应该对其传出的值进行检查。当然,检查会浪费一点时间,但为了整个系统不至于出现"非法操作"或是"Core Dump"的系统级的错误,多花这点时间还是很值得的。

5. 对系统调用的返回进行判断

继续上一条,对于一些系统调用,比如打开文件,我经常看到,许多程序员对 fopen 返回的指针不做任何判断,就直接使用了。然后发现文件的内容怎么也读不出来,或是怎么也写不进去。还是判断一下吧:

```
fp = fopen("log.txt", "a");
if ( fp == NULL )
{
    cout << "Error: open file error\n";
    return false;
}
```

其他还有许多,比如:socket 返回的 socket 号,new 返回的内存。请对这些系统调用返回的东西进行判断。

6. if 语句对出错的处理

先看一段程序代码:

```
if ( ch >= '0' && ch <= '9' )
{
    /* 正常处理代码 */
}
else
```

```
{
    /* 输出错误信息 */
    cout << "error ......\n";
    return ( false );
}
```

这种结构很不好，特别是如果“正常处理代码”很长时，对于这种情况，最好不要用 else。先判断错误，如：

```
if ( ch < '0' || ch > '9' )
{
    /* 输出错误信息 */
    cout << "error ......\n";
    return ( false );
}

/* 正常处理代码 */
......
```

这样的结构，不是很清楚吗？突出了错误的条件，让别人在使用你的函数的时候，第一眼就能看到不合法的条件，于是就会更加下意识地避免。

7. 在堆上分配内存

可能许多人对内存分配上的“栈 stack”和“堆 heap”还不是很明白。包括一些科班出身的人也不明白这两个概念。我不想过多地说这两个东西。简单的来讲，stack 上分配的内存系统自动释放，heap 上分配的内存，系统不释放，哪怕程序退出，那一块内存还是在那里。stack 一般是静态分配内存，heap 上一般是动态分配内存。由 new 系统函数分配的内存就是从堆上分配内存。从堆上分配的内存一定要自己释放。用 delete 释放，不然就是所谓的“内在泄露”或“内存漏洞”(Memory Leak)。于是，系统的可分配内存会随 new 越来越少，直到系统崩溃。还是来看看“栈内存”和“堆内存”的差别吧。

栈内存分配：

```
char * AllocStrFromStack()
{
    char pstr[100];
    return pstr;
}
```

堆内存分配：

```
char * AllocStrFromHeap(int len)
{
    char * pstr;
    if ( len <= 0 )
        return NULL;
    return ( char * ) new int [len ];
}
```

对于第一个函数，那块 pstr 的内存在函数返回时就被系统释放了。于是所返回的 char * 什么也没有。而对于第二个函数，是从堆上分配内存，所以哪怕是程序退出时，也不释放，所以第二个函数返回的内存没有问题，可以被使用。但一定要调用 delete 释放，不然就是内存泄漏！在堆上分配内存很容易造成内存泄漏，这是C++的最大“克星”，如果你的程序要稳定，那

么就不要出现 Memory Leak。所以，我还是要在这里千叮咛万嘱咐，在使用 new 函数时千万要小心。记得有一个 UNIX 上的服务应用程序，大约由几百个C++文件编译而成，运行测试良好，等使用时，每隔三个月系统就要死机一次，搞得许多人焦头烂额，查不出问题所在，只好每隔两个月人工手动重启系统一次。出现这种问题就是 Memery Leak 在作怪了，在C++程序中这种问题总是会发生，所以你一定要小心。一个 Rational 的检测工作——Purify，可以帮你测试你的程序有没有内存泄漏。

做过许多C++工程的程序员，都会对 new 有些“感冒”。当你什么时候在使用 new 时，有一种轻度的紧张和惶恐的感觉时，你就具备了这方面的修养了。

对于 new 和 delete 的操作有以下规则：

(1)配对使用，有一个 new，就应该有一个 delete。

(2)尽量在同一层上使用，不要像上面那种，new 在函数中，而 delete 在函数外。最好在同一调用层上使用这两个函数。

(3)new 分配的内存一定要初始化。delete 后的指针一定要设置为 NULL。

注：虽然现在的操作系统(如：UNIX 和 Windows 2000/NT)都有进程内存跟踪机制，也就是如果你有没有释放的内存，操作系统会帮你释放。但操作系统依然不会释放你程序中所有产生了 Memory Leak 的内存，所以，最好还是你自己来做这个工作。有的时候不知不觉就出现 Memory Leak 了，而且在几百万行的代码中找无异于海底捞针。

8. 变量的初始化

变量一定要被初始化再使用。C/C++编译器在这个方面不会像 Java 一样帮你初始化，这一切都需要你自己来，如果你使用了没有初始化的变量，结果未知。好的程序员从来都会在使用变量前初始化变量。如：

(1)对 new 分配的内存进行清零操作。

(2)对一些栈上分配的 struct 或数组进行初始化，最好也是清零。

不过话又说回来，初始化也会造成系统运行时间有一定的开销，所以，也不要对所有的变量做初始化，这个也没有意义。好的程序员知道哪些变量需要初始化，哪些则不需要。

如以下这种情况，则不需要。

```
char * pstr; /* 一个字符串 */
pstr = ( char* ) new int[50];
if ( pstr == NULL )
    exit(0);
strcpy( pstr, "Hello" );
```

但如果是下面一种情况，最好进行内存初始化。指针是一个危险的东西，一定要初始化。

```
char ** pstr; /* 一个字符串数组 */
pstr = ( char ** ) new int[50];
if ( pstr == NULL )
    exit(0);
/* 让数组中的指针都指向 NULL */
memset( pstr, 0, 50 * sizeof(char*) );
```

而对于全局变量和静态变量，一定要声明时就初始化。因为你不知道它第一次会在哪里被使用。所以使用前初始化这些变量是比较不现实的，一定要在声明时就初始化它们。如：

```
Links * plnk = NULL;      /* 对于全局变量 plnk 初始化为 NULL */
```

9. .h 文件和.cpp 文件的使用

.h 文件和.cpp 文件怎么用呢？一般来说，.h 文件中是声明(declare)，.cpp 文件中是定

义(define)。因为.cpp 文件要编译成库文件(Windows 下是.obj/.lib,UNIX 下是.o/.a),如果别人要使用你的函数,那么就要引用你的.h 文件,所以,.h 文件中一般是变量、宏定义、枚举、结构和函数接口的声明,就像一个接口说明文件一样。而.cpp 文件则是实现细节。.h 文件和.cpp 文件最大的用处就是声明和实现分开。这个特性应该是公认的了,但我仍然看到有些人喜欢把函数写在.cpp 文件中,这种习惯很不好。而且,如果在.h 文件中写上函数的实现,你还得在 makefile 中把头文件的依赖关系也加上去,这个就会让你的 makefile 很不规范。最后,有一个最需要注意的地方就是:带初始化的全局变量不要放在.h 文件中! 例如有一个处理错误信息的结构:

```
char * errmsg[] = {
        /* 0 */    "No error",
        /* 1 */    "Open file error",
        /* 2 */    "Failed in sending/receiving a message",
        /* 3 */    "Bad arguments",
        /* 4 */    "Memory is not enough",
        /* 5 */    "Service is down; try later",
        /* 6 */    "Unknown information",
        /* 7 */    "A socket operation has failed",
        /* 8 */    "Permission denied",
        /* 9 */    "Bad configuration file format",
        /* 10 */   "Communication time out",
        ......
    };
```

请不要把这个东西放在头文件中,因为如果你的这个头文件被 5 个函数库(.lib 或是.a)所用到,于是它就被链接在这 5 个.lib 中,而如果你的一个程序用到了这 5 个函数库中的函数,并且这些函数都用到了这个出错信息数组。那么这份信息将有 5 个副本存在于你的执行文件中。如果你的这个 errmsg 很大的话,而且你用到的函数库更多的话,你的执行文件也会变得很大。正确的写法应该把它写到.cpp 文件中,然后在各个需要用到 errmsg 的.cpp 文件头上加上 extern char * errmsg[]; 的外部声明,让编译器在链接时才去管它,这样一来,就只会有一个 errmsg 存在于执行文件中,而且,这样做很利于封装。我曾遇到过的最疯狂的事,就是在我的目标文件中,这个 errmsg 一共有 112 个副本,执行文件有 8M 左右。当我把 errmsg 放到.cpp 文件中,并为 1000 多个.cpp 文件加上了 extern 的声明后,所有的函数库文件大小都下降了 20%左右,而我的执行文件只有 5M 了,一下子少了 3M 啊。

10. 出错信息的处理

你会处理出错信息吗? 它并不是简单的输出。看下面的示例:

```
if ( p == NULL )
{
    cout << "ERR: The pointer is NULL\n";
}
```

告别学生时代的编程吧。这种编程很不利于维护和管理,出错信息或是提示信息,应该统一处理,而不是像上面这样,写成一个“硬编码”。第 9 条对这方面的处理做了一部分说明。如果要管理错误信息,那就要有以下的处理:

```
/* 声明出错代码 */
#define    ERR_NO_ERROR        0 /* No error                    */
```

```
#define    ERR_OPEN_FILE      1 /* Open file error              */
#define    ERR_SEND_MESG      2 /* sending a message error      */
#define    ERR_BAD_ARGS       3 /* Bad arguments                */
#define    ERR_MEM_NONE       4 /* Memory is not enough         */
#define    ERR_SERV_DOWN      5 /* Service down try later       */
#define    ERR_UNKNOW_INFO    6 /* Unknown information          */
#define    ERR_SOCKET_ERR     7 /* Socket operation failed      */
#define    ERR_PERMISSION     8 /* Permission denied            */
#define    ERR_BAD_FORMAT     9 /* Bad configuration file       */
#define    ERR_TIME_OUT       10 /* Communication time out      */

/* 声明出错信息 */
char * errmsg[] = {
    /* 0 */          No error",
    /* 1 */          "Open file error",
    /* 2 */          "Failed in sending/receiving a message",
    /* 3 */          "Bad arguments",
    /* 4 */          "Memory is not enough",
    /* 5 */          "Service is down; try later",
    /* 6 */          "Unknown information",
    /* 7 */          "A socket operation has failed",
    /* 8 */          "Permission denied",
    /* 9 */          "Bad configuration file format",
    /* 10 */         "Communication time out",
};

/* 声明错误代码全局变量 */
long errno = 0;

/* 打印出错信息函数 */
void perror( char * info)
{
    if ( info )
    {
        cout << errmsg[errno] << "\n";
        return;
    }
    cout << errmsg[errno] << "\n";
}
```

这个基本上是 ANSI 的错误处理实现细节了，于是当程序中有错误时你就可以这样处理：

```
bool CheckPermission( char * userName )
{
    if ( strcpy(userName, "root") ! = 0 )
    {
        errno = ERR_PERMISSION_DENIED;
        return (FALSE);
```