Broadview
www.broadview.com.cn

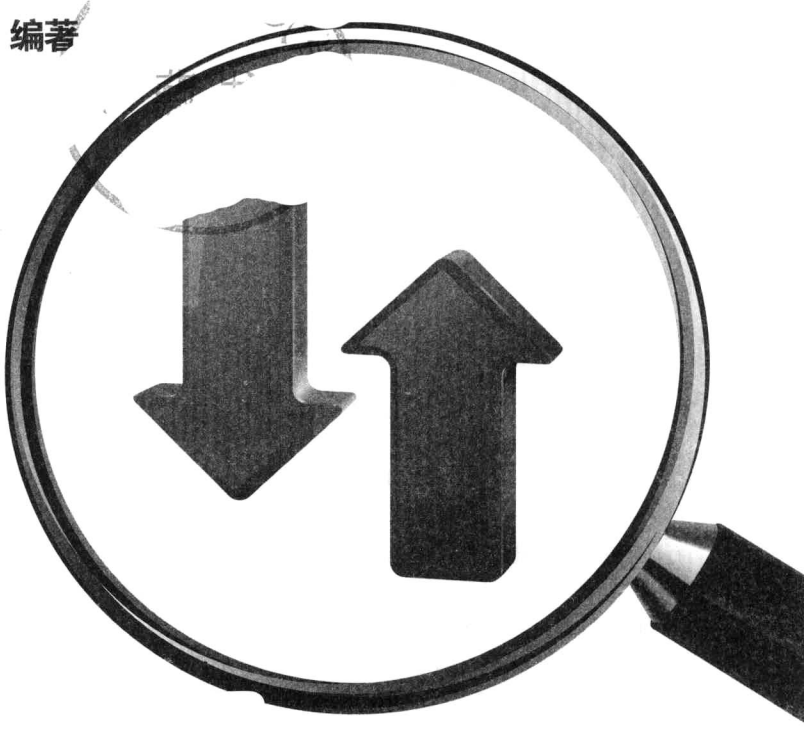
C/C++ 程序缺陷与优化

于秀山 许 峰 李华莹
刘 然 于长钺 杨玲萍 编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

C/C++ 程序缺陷与优化

于秀山　许　峰　李华莹
刘　然　于长钺　杨玲萍　编著



電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

程序设计可谓是一个汗牛充栋的话题。与传统的C/C++程序设计方面的书籍不同，本书从另外一个视角——程序缺陷的角度来探讨程序设计与优化。

本书从作者所从事的软件测试项目中精选了与C/C++语言有关的程序缺陷，主要包括编码风格、内存管理、内存泄漏、缓冲区溢出、指针使用、安全等方面。对于每一种缺陷，通过实例分析了缺陷产生的原因，并给出了具体的修改和优化方法。面对这些缺陷，程序员会有一种似曾相识、相见恨晚的感觉。通过这些缺陷，程序员能够跳出固有的程序设计思维定式，使其翻然醒悟，茅塞顿开。

本书适合于有一定编程经验的软件开发人员和测试人员使用，也可作为高等院校计算机相关专业高级程序设计及软件测试课程教材。

图书在版编目（CIP）数据

C/C++程序缺陷与优化 / 于秀山等编著. —北京：电子工业出版社，2014.4
ISBN 978-7-121-22632-8

Ⅰ. ①C… Ⅱ. ①于… Ⅲ. ①C语言－程序设计 Ⅳ.①TP312

中国版本图书馆CIP数据核字（2014）第047653号

责任编辑：郭　立
印　　刷：三河市双峰印刷装订有限公司
装　　订：三河市双峰印刷装订有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　　邮编：100036
开　　本：787×980　1/16　印张：17.5　字数：383千字
印　　次：2014年4月第1次印刷
定　　价：49.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

C/C++程序设计是一个既古老又时尚的话题，其古老性表现在几乎任何一个程序员都对其有所了解，都有过使用该语言开发软件的经验；其时尚性表现在虽然历经几十年的演变，这两种语言依然经久不衰，仍然在各个领域得到广泛使用，C/C++程序设计几乎成为每一个程序员的必修课。

虽然大多数程序员都经过了系统的程序设计方面的培训，但编写的软件中仍然存在大量的缺陷，甚至是很低级的缺陷，这些缺陷严重影响了软件质量。

“软件中为何还会存在这样的缺陷？”这是令管理者和程序员经常困惑的一个问题，也是笔者所关注的问题。笔者长期从事软件测试方面的工作，亲历了大量各式各样的软件缺陷，这些缺陷使笔者萌生了从另外一个角度透视程序设计的想法。

与传统的 C/C++程序设计方面的书籍不同，本书从另外一个视角——程序设计缺陷的角度来探讨程序设计。程序员长期形成的习惯性思维，使其难以觉察到自身在程序设计方面存在的问题，可谓“不识庐山真面目，只缘身在此山中”。本书列举了大量来自实际项目中出现的软件缺陷，这些缺陷就像一面镜子，面对这些缺陷，程序员会有一种似曾相识、相见恨晚的感觉。通过这些缺陷，程序员能够跳出固有的程序设计思维定式，使其翻然醒悟，茅塞顿开。

前车之覆，后车之鉴，期望本书能够使读者充分借鉴前人在 C/C++程序设计方面的经验教训，快速提升自己的程序设计水平。

本书由于秀山、许峰、李华莹、刘然、于长钺、杨玲萍编著。在本书的编写过程中，尹浩、严少清、董昕、刘怡静同志参与了部分章节的编写工作，在此向他们表示衷心感谢。

鉴于作者才疏学浅，书中难免有遗漏和错误之处，敬请读者斧正。

作　者

2013 年秋于北京

目　录

第 1 章 语言使用基本问题

C/C++是一种介于汇编语言和高级语言之间的中级语言，自 C/C++语言问世以来，由于其具有运行速度快、可移植、容易理解、便于阅读和书写等特点，使得这两种语言经久不衰，成为很多程序员的首选编程语言。然而，C/C++语言，特别是 C 语言也存在一些固有的缺点，例如，C 语言不是强类型语言，它允许将整型变量赋值给字符型变量、C 语言缺少对字符串和记录的处理、C 语言缺少运行期间检查；为了追求效率，在运行期间，C 语言不检查诸如数组越界之类的错误；C 语言中有相当多的地方容易使程序员产生误解，例如，运算符优先级规则；C 语言中有很多地方的定义是不完备的，程序执行结果可能因编译器的不同而改变，某些情况下，即使在同一个编译器内也会根据上下文而发生变化。上述缺点使得 C/C++程序会产生各种不同的缺陷。其中，有些是变量名字书写错误、有多余的分号等这样的简单缺陷，有些是算法理解方面的复杂缺陷。

使用 C/C++语言可以编写出布局良好的、结构化的、表达性强的程序，也可以编写出不当的和特别难以理解的程序。在本章中，笔者重点介绍在 C/C++语言使用方面存在的基本问题，这些问题看似简单，似乎不屑一顾，而事实上，这些问题在具体项目中频繁出现，严重影响了软件质量。

1.1 变量使用问题

变量是程序处理的基本数据对象之一，它用于保存程序中不断变化的值、从外部接收的数据、保存中间结果及最终结果等。程序中所使用的任何变量和数据都必须遵循先定义后使用的原则。使用未初始化的变量是很常见的程序错误，编译器把变量存放在内存中的某个位置，如果变量没有正确初始化，则该变量存储的数据是未知的，在程序中直接使用该变量可能引起计算错误，甚至软件崩溃等问题。虽然许多编译器在检查出使用未初始化的变量时都会给出一个警告（Warning），但并不要求程序员必须修改此问题，而且，没有一个编译器能够发现所有使用未初始化变量的错误。

1. 未明确定义变量值

所有的变量在使用前都应该有明确定义的值。

例 1：

```
1   void foo( ) {
2   int b;
3   b++;
4   }
```

例 1 中第 3 行语句，变量 b 的值未明确定义却被使用。

修改方法：在第 2 行语句后面增加给变量 b 赋值的语句，如：b＝0;

2. 未初始化所有变量

为了避免使用没有初始化的变量，程序中所有的变量都应该初始化。

如果不初始化声明的整数变量，将得到一个随机值；如果不初始化声明的静态变量，编译器会自动将其初始化为 0；如果不初始化声明的指针变量，那么它所指向的内容是无法确定的，在这种情况下使用该指针变量时，将产生不可预料的后果。

例 2：

```
1   f()
2   {
3       int  *t; // 错误
4       …
5   }
```

例 2 中，f()函数仅仅声明了一个指向整数的指针 t，却没有对 t 进行初始化，指针变量 t 指向的地址将是不确定的，如果其他程序调用 f()函数，使用其中没有初始化的指针变量 t，则可能产生不可预料的错误。

修改方法：将第 3 行语句修改为：int *t=1;

3. 存在多余的变量

程序中定义并初始化了局部变量，但该变量未被使用。

例 3：

```
1   class A {
2   public:
3       void displBalance();
4   };
```

```
5   void foo( ) {
6      A a;       // 错误
7   }
8   void func( ) {
9      int i = 0;  // 错误
10  }
```

修改后的程序如例 4 所示。

例 4：

```
1   class A {
2   public:
3      void displBalance( );
4   };

5   void foo( ) {
6      A a;
7      a.displBalance( );
8   }

9   void func( ) {
10     int i = 0;
11     i++;
12  }
```

4. 使用未初始化的堆内存

程序中使用内存分配函数 malloc()为结构体分配堆内存后，堆内存没有初始化却被使用。

例 5：

```
1   struct student{
2     int num;
3     char[10] name;
4   };
5   int  main(int t)
6   {
7     struct student *s=malloc(sizeof(struct s));
8     int t;
9     t= s->num; // 错误
10  }
```

例 5 中第 9 行语句，结构体 s 还没有初始化却被使用。

5. 读取结构体中未赋值的局部变量

程序中结构体内的局部变量没有被赋值，但却读取该成员的值或将该成员作为参数传递给某个函数。

例 6：

```
1   struct s
2   {
3     int a;
4     int b;
5   };
6   main()
7   {
8     s x;
9     x.b=0;
10    max(x.a,1); // 错误
11  }
```

例 6 中第 10 行语句，结构体 x 中的成员变量 a 还没有赋值，却作为参数传递给函数 max()。

6. 没有使用构造函数初始化类成员

类中所有的成员变量没有在类构造函数中初始化。

例 7：

```
1   class c
2   {
3     private: int i;
4              int j;
5              bool flag;
6     public: c()
7       {
8         i=0;
9         j=1;
10      }
11  };
```

例 7 中，类 c 的构造函数中缺少对成员变量 flag 的赋值语句。

7. 引用被初始化为地址可变的对象

引用总是指向一个对象，它只能在定义时被初始化一次，之后不可改变。引用和指针之间的区别如下。

（1）指针指向一块内存，它的内容是所指内存的地址，而引用是某块内存的别名。指针是一个实体，而引用仅是一个别名；

（2）使用引用时无须解引用（dereference），而指针需要解引用（*ptr，ptr 为指针）；

（3）引用只能在定义时被初始化一次，之后不可改变，而指针可改变；

（4）引用没有 const 类型，而指针有 const 类型，且 const 类型的指针不可改变；

（5）引用不能为空，而指针可以为空。

例 8：

```
1   void tmp(int* b)
2   {
3      if (b==NULL)
4       {
5          b=new int;
6          *b=200;
7       }
8      else
9       {
10        *b=100;
11      }
12  }
13  main()
14  {
15    int a=0;
16    tmp(&a);
17    int* c=NULL;
18    tmp(&c);    // 指针可以为空，但引用不能为空
19    return;
20  }
```

例 8 的第 18 行语句中，变量 c 是一个空指针，因为引用不能为空，所以“&c”不正确。

修改方法：将第 18 行语句修改为：`tmp(c);`

8. 静态成员未初始化

类中的静态数据成员是所有类对象共享的内容，其存放的是所有对象的值，而不是某个对象的值。

静态数据成员的初始化在类体外进行，初始化时不需要加访问权限符，因为静态数据成员是类的数据成员，所以在初始化时应指出其类名。

例 9：

```
1   class T
2    {
3     public:
4       T (int a,int b);
5         bb( );
6       private:
7         int x,y;
8         static int s;
9    };
10  T::T (int a,int b)//构造函数的实现部分
11  {
12    x=a;
13    y=b;
14  }
15  void T::bb( )//成员函数的实现部分
16  {
17    s=s+x+y;
18    cout<<"s="<<s<
19  }
20  main( )
21  {
22     T  t1(10,20), t2(5,3);
23     t1.bb( );
       t2.bb( );
24  }
```

例 9 中，T 类中有一个静态数据成员 s，该数据成员应该在类体外被定义。

修改方法：在第 10 行语句之前，增加为静态数据成员 s 赋初值的语句：`int T::s=0;`

9. 赋值运算符（operator=）未给所有的变量赋值

程序中虽然使用赋值运算符（operator=），但没有给所有的变量赋值。

例 10：

```
1   class A
2   {
3      private:
```

```
4       int _x, _y, _z;
5       public:
6       A( ) { }
7       A& operator=( const A& a )
8       {
9           _x = a._x;
10          _y = a._y;
11          return *this;
12      }
13  };
```

例 10 中，第 7 行语句使用“operator=”的本意是给所有的成员赋值，但事实上仅仅对类 A 中的成员变量“_x、_y”进行了赋值，没有对类 A 中所有的成员变量进行赋值。

修改方法：在第 10 行语句后面增加语句：`_z = a._z;`

10. 工程头文件中包含变量的定义

头文件中最好只声明变量，而不定义变量。因为工程中的头文件会被.c 或.cpp 文件多次包含，如果头文件中包含变量的定义，势必造成变量的重复定义。因此，变量的定义应该写在.c 或.cpp 文件内。

例 11：

```
头文件 header.h
    int t=2; // 错误
```

例 11 中，头文件 header.h 包含了对变量 t 的初始化语句，头文件 header.h 如果多次被源代码（.c 或.cpp）文件包含，编译时该变量将被重复定义。

修改方法：将 int t=2 语句修改为：`int t;`

11. 无符号整数初始化为负数

无符号整数不能识别负数，所以初始化无符号整数为负数是错误的。

例 12：

```
1   F()
2   {
3       unsigned int y = -21; // 错误
4   }
```

例 12 中第 3 行语句，无符号整数 y 被错误赋值为负数“-21”。

修改方法：将有符号整数赋值为负数，即将第 3 行语句修改为：signed int y = -21;

一般情况下，有符号整数和无符号整数在内存中均占用 16 位，但这两种整数类型的取值范围是不同的，有符号整数的取值范围是–32768 到 32767，无符号整数的取值范围是 0 到 65535。

例 13：

```
1   bool fun(size_t cbSize)
2   {
3     if(cbSize > 1024)
4     rerurn false;
5     char *pBuf = new char[cbSize- 1];              //未对 new 的返回值进行检查
6     memset(pBuf, 0x90, cbSize- 1);
7       …
8     return true;
9   }
```

例 13 中，第 5 行语句调用 new()分配内存后，未对调用结果的正确性进行检查。如果 cbSize 为 0，则“cbSize – 1”为–1，但是 memset()函数中第 3 个参数本身是无符号数，因此会将–1 视为正的 0xFFFFFFFF，导致执行此函数时程序崩溃。

12. 显式调用未定义的构造函数

如果程序员自定义的类中没有构造函数，编译器将为该类产生一个默认的构造函数，用于创建类对象时调用，但这个编译器产生的构造函数是无参函数，函数体为空，不做任何操作。为了给类中各成员变量赋初值，应该在创建的类中自定义构造函数。

例 14：

```
1   class A
2    {
3      public:
4         int b;
5         int a;
6    };
7   A* f()
8    {A *a
9      A *a=new A();
10     return new A();
11  }
```

例 14 中，类 A 中并没有定义自己的构造函数，而在 f()函数中调用“A()”。

修改方法：在第 5 行语句后面增加对构造函数的声明语句：

```
A(){ int b=1;  int a=2;};
```

13. 通过赋值方式初始化类的常量成员

调用类的构造函数时，构造函数将参数值传递给类中相应的数据成员。当构造函数的参数值是常量（const）时，常量参数值只能通过初始化方式传递给类中相应的数据成员。

例 15：

```
1   class A {
2   public:
3      A( const char *file, const char *path )
4      {
5         myFile = file;  // 错误
6         myPath = path;  // 错误
7      }
8   private:
9         string myFile;
10        string myPath;
11  };
```

例 15 中的第 5 行语句和第 6 行语句，因为类 A 的构造函数 A()中两个参数值都是常量，所以应该通过初始化方式而非赋值方式给类中成员传值。修改后的程序如例 16 所示。

例 16：

```
1   class A {
2   public:
3      A( const char *file, const char *path ) :
          myFile(file), myPath(path) {};
4   private:
5      string myFile;
6      string myPath;
7   };
```

例 17：

```
1   class  student
2   {
3        public:
4          student();
5        protected:
6          const  int  a;
7          const  int  &b;
8   };
```

```
9    student::student(int  i, int  j) // 错误
10   {
11      a=i;
12      b=j;
13   }
```

例 17 中第 9 行语句，类中常量成员初始化操作没有在初始化表里完成。

修改方法：将第 9 行语句修改为：

```
student::student(int i, int j):a(i),b(j){};
```

14. 构造函数内变量初始化和声明的顺序不一致

C++语言中，构造函数有一种特殊的初始化方式——“初始化列表”，构造函数初始化列表以一个冒号开始，接着是以逗号分隔的数据成员列表，每个数据成员后面跟一个放在括号中的初始化值。如果类存在继承关系，派生类必须在其初始化列表里调用其基类的构造函数。

初始化时，编译器会记录成员变量在初始化列表中的初始化顺序，以保证析构函数能够正确释放成员变量。

类的构造函数按照成员在类中的声明顺序执行初始化操作，因此，如果类中构造函数的初始化列表和成员变量的声明顺序不一致，势必造成析构函数释放成员变量的顺序错误。

例 18：

```
1    class A
2    {
3    public:
4       A( int x ) : a( x ), b( a ) {};// 错误
5    private:
6       int b;
7       int a;
8    };
9    class B : public A
10   {
11   public:
12      B( int );
13   private:
14      int t;
15      float u;
16   };
17   B::B( int y ) : u( y ), A( 1 ), t( u ) { }; // 错误
```

例 18 中，类 A 的初始化列表（第 4 行语句）中成员变量的初始化顺序是先 a 成员，再 b 成员；

而类 A 中成员变量声明的顺序是先 b 成员，再 a 成员，两者初始化顺序不一致。第 17 行语句中也存在同样的错误。

修改方法：将第 4 行语句修改为：

```
A( int x ) : b( x ), a (b) {};
```

第 17 行语句修改为：

```
B::B( int y ) : t(y), A( 1 ), u( t ) { };
```

15. 构造函数内成员变量初始化顺序错误

类中构造函数按照成员在类中的声明顺序执行初始化操作，如果忽略了这一点，将导致初始化错误。

例 19：

```
1   class X
2   {
3   public:
4      X( int y );
5   private:
6      int i;
7      int j;
8    };
9   inline X::X( int y ) : j( y ), i( j ) {} ; // 错误
```

例 19 中，构造函数类 X 中成员变量的初始化顺序是先 i，再 j，第 9 行语句初始化列表中将成员变量 i 初始化为值 j ，而此时成员变量 j 还没有被初始化。

修改方法：将第 9 行语句修改为：

```
inline X::X( int y int x ) : j( y ), i( x ) {};
```

16. 外部变量重复定义

工程里同一外部变量在不同源文件中被重复定义时，不同的编译器对这种情况可能有不同的处理方式。为了避免外部变量被重复定义，应保证程序中同一外部变量只能定义一次。

例 20：

```
在 sourc1.c 中定义 int  a=1;
在 sourc2.c 中声明  extern  int  a; //a 是一个外部变量
在 sourc3.c 中定义 int  a=0;
```

当把 sourc1、sourc2 和 sourc3 源代码同时编译链接在一个工程里时，此时源文件 sourc2.c 中引用的外部变量 a 有两个值（在 sourc1 和 sourc3 中均有定义）。