void关键字的使用规则：

1. 如果函数没有返回值，那么应声明为void类型；
2. 如果函数无参数，那么应声明其参数为void；
3. 如果函数的参数可以是任意类型指针，那么应声明其参数为void *；
4. void不能代表一个真实的变量；

void体现了一种抽象，这个世界上的变量都是“有类型”的

1.概述

许多初学者对C/C++语言中的void及void指针类型不甚理解，因此在使用上出现了一些错误。本文将对void关键字的深刻含义进行解说，并详述void及void指针类型的使用方法与技巧。

2.void的含义

void的字面意思是“**无类型**”，**void ***则为“无类型指针”，void *可以指向任何类型的数据。

void几乎只有“注释”和限制程序的作用，因为从来没有人会定义一个void变量，让我们试着来定义：

```
    void a;
```

这行语句编译时会出错，提示“illegal use of type 'void'”。不过，即使void a的编译不会出错，它也没有任何实际意义。

void真正发挥的作用在于：　　（1）**对函数返回的限定**；　　（2）** 对函数参数的限定**。

我们将在第三节对以上二点进行具体说明。

众所周知，如果指针p1和p2的类型相同，那么我们可以直接在p1和p2间互相赋值；如果p1和p2指向不同的数据类型，则必须使用强制类型。转换运算符把赋值运算符右边的指针类型转换为左边指针的类型。

例如：

```
float *p1;
int *p2;
p1 = p2;
```

其中p1 = p2语句会编译出错，提示“'=' : cannot convert from 'int *' to 'float *'”，必须改为：

```
p1 = (float *)p2;
```

而void *则不同，任何类型的指针都可以直接赋值给它，无需进行强制类型转换：

```
void *p1;
int *p2;
p1 = p2;
```

但这并不意味着，void *也可以无需强制类型转换地赋给其它类型的指针。因为“无类型”可以包容“有类型”，而“有类型”则不能包

容“无类型”。道理很简单，我们可以说“男人和女人都是人”，但不能说“人是男人”或者“人是女人”。下面的语句编译出错：

```
void *p1;//人
int *p2;//男人
p2 = p1;//不能说  “人是男人”   因而报错
```

提示“'=' : cannot convert from 'void *' to 'int *'”。

3.void的使用

下面给出void关键字的使用规则：

规则一 如果函数没有返回值，那么应声明为void类型

在C语言中，凡不加返回值类型限定的函数，就会被编译器作为返回整型值处理。但是许多程序员却误以为其为void类型 。例如

```
add ( int a, int b )
{
  return a + b;
}
int main(int argc, char* argv[])
{
  printf ( "2 + 3 = %d", add ( 2, 3) );
}
```

程序运行的结果为输出：　　2 + 3 = 5　　这说明不加返回值说明的函数的确为int函数 。

> 林锐博士《高质量C/C++编程》中提到：“C++语言有很严格的类型安全检查，不允许上述情况（指函数不加类型声明）发生”。可是编译
>
> 器并不一定这么认定，譬如在Visual C++6.0中上述add函数的编译无错也无警告且运行正确，所以不能寄希望于编译器会做严格的类型检查。

因此，为了避免混乱，我们在编写C/C++程序时，对于任何函数都必须一个不漏地指定其类型。如果函数没有返回值，一定要声明为void类型。这既是程序良好可读性的需要，也是编程规范性的要求。另外，加上void类型声明后，也可以发挥代码的“自注释”作用。代码的“自注释”即代码能自己注释自己。

规则二：如果函数无参数,那么应声明其参数为void

在C++语言中声明一个这样的函数：

```
int function(void)
{
  return 1;
}
```

则进行下面的调用是不合法的：

```
function(2);
```

因为在C++中，函数参数为void的意思是**这个函数不接受任何参数。**

我们在Turbo C 2.0中编译：

```
#include "stdio.h"
fun()
{
  return 1;
}
main()
{
  printf("%d",fun(2));
  getchar();
}
```

编译正确且输出1，这说明，在C语言中，可以给无参数的函数传送任意类型的参数，但是在C++编译器中编译同样的代码则会出错。**在C++中，不能向无参数的函数传送任何参数**，出错提示“'fun' : function does not take 1 parameters”。

所以，无论在C还是C++中，若函数不接受任何参数，一定要指明参数为void。

规则三 小心使用void指针类型

按照ANSI(American National Standards Institute)标准，不能对void指针进行算法操作，即下列操作都是不合法的：

```
void * pvoid;
pvoid++; //ANSI：错误
pvoid += 1; //ANSI：错误
//ANSI标准之所以这样认定，是因为它坚持：进行算法操作的指针必须是确定知道其指向数据类型大小的。
//例如：
int *pint;
pint++; //ANSI：正确
```

pint++的结果是使其增大sizeof(int)。(在VC6.0上测试是sizeof(int)的倍数)

但是大名鼎鼎的GNU(GNU's Not Unix的缩写)则不这么认定，它指定**void ****的算法操作与char **一致。

因此下列语句在GNU编译器中皆正确：

```
pvoid++; //GNU：正确
pvoid += 1; //GNU：正确
```

pvoid++的执行结果是其增大了1。(在VC6.0上测试是sizeof(int)的倍数)

在实际的程序设计中，为迎合ANSI标准，并提高程序的可移植性，我们可以这样编写实现同样功能的代码：

```
void * pvoid;
(char *)pvoid++; //ANSI：正确；GNU：正确
(char *)pvoid += 1; //ANSI：错误；GNU：正确
```

GNU和ANSI还有一些区别，总体而言，GNU较ANSI更“开放”，提供了对更多语法的支持。但是我们在真实设计时，还是应该尽可能地迎合ANSI标准。

规则四如果函数的参数可以是任意类型指针，那么应声明其参数为void *

典型的如内存操作函数memcpy和memset的函数原型分别为：

```
void * memcpy(void *dest, const void *src, size_t len);
void * memset ( void * buffer, int c, size_t num );
```

> 这样，任何类型的指针都可以传入memcpy和memset中，这也真实地体现了内存操作函数的意义，因为它操作的对象仅仅是一片内存，而不论这片内存是什么类型。如果memcpy

> 和memset的参数类型不是void *，而是char *，那才叫真的奇怪了！这样的memcpy和memset明显不是一个“纯粹的，脱离低级趣味的”函数！

下面的代码执行正确：

```
//示例：memset接受任意类型指针
int intarray[100];
memset ( intarray, 0, 100*sizeof(int) ); //将intarray清0
//示例：memcpy接受任意类型指针
int intarray1[100], intarray2[100];
memcpy ( intarray1, intarray2, 100*sizeof(int) ); //将intarray2拷贝给intarray1
```

有趣的是，memcpy和memset函数返回的也是void *类型，标准库函数的编写者是多么地富有学问啊！

规则五 void不能代表一个真实的变量

下面代码都企图让void代表一个真实的变量，因此都是错误的代码：

```
void a; //错误
function(void a); //错误
```

void体现了一种抽象，这个世界上的变量都是“有类型”的，譬如一个人不是男人就是女人（还有人妖？）。

void的出现只是为了一种抽象的需要，如果你正确地理解了面向对象中“抽象基类”的概念，也很容易理解void数据类型。正如不能给抽

象基类定义一个实例，我们也不能定义一个void（让我们类比的称void为“抽象数据类型”）变量。

4.总结

小小的void蕴藏着很丰富的设计哲学，作为一名程序设计人员，对问题进行深一个层次的思考必然使我们受益匪浅。不论什么类型的指针(void*, char*, int*, float*...) 默认初始值都是0xCCCCCCCC//这个应该各个编译器都不同的,这个针对vc6

```
#include<iostream.h>
#include <memory.h>
//#include   <string.h>
void main()
{
  void *p1;
  int a = 10;
  int *p2 = &a;
  cout << p1 << endl;
  cout << (int)*p2 << endl;
  p1 = p2;
```

```
    cout << *(int*)p1 << endl;//!!!!!!!    用空类型操作输出值！
    cout << (int)*p2 << endl;
}
```

> 输出： 0xCCCCCCCC 10 10 10

在声明同时赋值NULL，在delete后立即设置为NULL。

在debug版本下指针默认初始值为0xCCCCCCCC，在Release版本下初始值为0x0000000A，（在我电脑上VC6.0）。对于指针如果暂时没有合适的初始化值，就应该把它置为NULL（0）。 对于好的编程习惯来说，declare一个指针，则初始化为NULL，如果是类成员 则在构造函数中initialize，当对指针使用delete时候，则置它为NULL.

0xCCCCCCCC只是在debug状态下VC生成的未定义过的指针值，用来提示这个指针是未被初始化的，在release状态下不会等于这个值（除非巧合）。对于指针如果暂时没有合适的初始化值，就应该把它置为NULL（0）。