第29章　动态分配函数

本章介绍动态分配函数，这些函数是从C语言中继承而来的。这些函数的核心是malloc( )和free( )。每次调用malloc( )时，都要分配一部分剩余的自由存储区；而每次调用free( )时，都把存储区返给系统。内存是从该自由存储区分配的，这个存储区称为堆（heap）。动态分配函数的原型在<cstdlib>中定义，而C程序必须使用头文件stdlib.h。

所有的C++编译器至少包括以下4个动态分配函数：calloc( ), malloc( ), free( )和realloc( )。

然而，一般的编译器总是包含这些函数的几种变体以适应不同的选择和环境，可以查看所使用的编译器的文档。

虽然C++支持我们在这里介绍的动态分配函数，但是一般来说，这些函数并不在C++程序中使用，这是因为C++提供了动态分配运算符new和delete。使用动态分配运算符有几个优点：第一，new能够为分配的数据类型自动分配适当大小的存储区；第二，它可以返回指向该存储区的适当的指针类型；第三，new和delete可以被重载。因为new和delete优于基于C的动态分配函数，所以建议在C++程序中使用这些运算符。

29.1　calloc函数

```
#include <cstdlib>
void *calloc(size_t num, size_t size);
```

calloc( )函数把存储区的大小分配为num * size。也就是说，calloc( )给具有num个对象的数组分配足够的内存，其中每个对象的大小为size。

calloc( )函数返回一个指向所分配的存储区的第一个字节的指针。如果存储区的大小不足以满足需求，函数返回一个空指针。在试图使用该返回值之前，验证一下该值不为空是非常重要的。

与calloc( )相关的函数有free( )，malloc( )和realloc( )。

29.2　free函数

```
#include <cstdlib>
void free(void *ptr);
```

free( )函数把ptr所指的存储区返回给堆，从而使该存储区可以用于将来的分配操作。

这里有一个强制性的要求，即只能通过先前由任何一种动态分配系统的函数（malloc( )或calloc( )）分配的指针调用free( )。使用无效的指针调用该函数很可能破坏内存管理机制并导致系统崩溃。

与free( )相关的函数有calloc( )，malloc( )和realloc( )。

29.3 malloc 函数

```
#include <cstdlib>
void *malloc(size_t size);
```

malloc( )函数返回一个指向大小为 size 的存储区的第一个字节的指针，这个存储区是从堆中分配的。如果堆中的存储区不足以满足需求，malloc( )将返回一个空指针。在试图使用该返回值之前，验证一下该值不为空是非常重要的。试图使用一个空指针将导致系统崩溃。

与 malloc( )相关的函数有 free( )，realloc( )和 calloc( )。

29.4 realloc 函数

```
#include <cstdlib>
void *realloc(void *ptr, size_t size);
```

realloc( )函数将 ptr 所指的先前分配的存储区大小改变为 size 指定的大小。size 可以比原来的值大，也可以比原来的值小。因为 realloc( )为了改变存储区大小必须要移动存储块，所以该函数返回一个指向存储块的指针。如果出现这种情况，老存储块中的内容将被复制到新存储块中——不会丢失任何信息。

如果 ptr 为空，realloc( )只分配 size 个字节的存储区并返回一个指向该存储区的指针。如果 size 为 0，ptr 所指的存储区为自由存储区。

如果堆中的自由存储区不足以分配 size 个字节，函数将返回一个空指针，而且原来的存储块将保持不变。

与 realloc( )相关的函数有 free( )，malloc( )和 calloc( )。