在C++中，采用操作符new来分配堆内存，采用delete来释放分配的堆内存。

例：程序heap\_test1

p = new int[3];

for (int i = 0; i != 3; ++i)

{

p[i] = i;

}

for (int j = 0; j != 3; ++j)

{

std::cout << p[j] << std::endl;

}

delete [] p;

p = NULL;

动态创建的堆内存指针，如果通过++，--等方式改变了指向时，在delete释放内存时，必须把指针指向它最初的位置，否则会发生错误。

int num = 4;

int data\_num = 9;

float\* input = new float[num \* data\_num];

float\* backup\_ptr = input; // 备份指针

for (int i = 0; i != data\_num; ++i)

{

input += num;

}

// input = input - num \* data\_num;

input = backup\_ptr; // 把input指针指向它初始的位置

delete [] input;

input = NULL;

C++中的内存管理：

内存分配方式有3种：

1. 从静态存储区域分配。内存在程序编译的时候就分配好了，这些内存在程序的整个运行期间都存在。如全局变量，static变量等。
2. 在栈上分配。在函数执行期间，函数内部变量（包括形参）的存储单元都创建在栈上，函数结束时这些存储单元自动释放（栈清退）。
3. 从堆（heap）或自由存储空间上分配，亦称动态内存分配。程序运行期间用malloc()或new申请任意数量的内存，程序员自己掌握释放内存的恰当时机（使用free()或delete）。如果使用堆栈存储和静态存储就能满足程序要求，就不要使用堆内存分配。

内存的使用：

1. 用malloc或new申请内存之后，应该立即检查指针值是否为NULL或者进行异常处理，以防止使用值为NULL的指针。
2. 不要忘记初始化指针、数组或动态内存，防止将未初始化的内存作为右值使用。
3. 避免数组或指针的下标越界，尤其是在for循环中，特别要当心发生“多1”或者“少1”操作。
4. 动态内存的申请与释放必须配对，防止内存泄露。
5. 用free或delete释放了内存之后，立即将指针设置为NULL，防止产生“野指针”，即悬空指针。

free或delete只是把指针指向的内存给释放掉，而并没有把指针本身删除掉，这时一定要把指针设置为NULL。因为这时的指针为无效指针，但不一定为NULL。

在函数内部定义指针变量，然后分配动态内存。当函数结束时，指针会消亡，但其指向的内存仍然存在，这就会造成内存泄露。

1. 指针消亡了，并不代表它所指向的内存会被自动释放；
2. 内存被释放了，并不代表指针消亡或成了NULL。

野指针，即悬空指针，为指向“非法”内存的指针。

野指针产生的原因主要有3种：

1. 指针变量没有初始化。它的默认值是随机的，会乱指一气。所以指针变量在创建时一定要初始化，要么指向NULL，要么指向有效内存。
2. 指针p被free()或delete之后，没有置为NULL，让人误以为p仍然有效。
3. 指针超越了变量的作用范围。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 文件名 ：test.cpp

/\* 作者 ：刘珅珅

/\* 日期 ：2012/4/23

/\* 功能描述 ：野指针测试

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

using namespace std;

class Test

{

public:

void Func()

{

cout << "Test::func()" << endl;

}

};

int main( int argc, char \*argv[] )

{

Test \*pTest = NULL;

if ( pTest == NULL )

{

Test test;

pTest = &test;

}

// pTest为野指针，但程序不会报错，原因是test虽然退栈

// 但其析构函数并没有清除test的内存，test的内存空间

// 仍然在函数的堆栈上。

pTest->Func();

return 0;

}