malloc/free与new/delete的区别

相同点：

> 都可用于申请动态内存和释放内存

不同点：

> （1）操作对象有所不同。
>
> malloc与free是C++/C 语言的标准库函数，new/delete 是C++的运算符。对于非内部数据类的对象而言，光用maloc/free 无法满足动态对象的要求。对象在创建的同时要自动执行构造函数， 对象消亡之前要自动执行析构函数。由于malloc/free 是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加malloc/free。
>
> （2）用法上也有所不同。
>
> 函数malloc 的原型如下：
>
> ```
> void * malloc(size_t size);
> ```
>
> 用malloc 申请一块长度为length 的整数类型的内存，程序如下：
>
> ```
> int *p = (int *) malloc(sizeof(int) * length);
> ```
>
> 我们应当把注意力集中在两个要素上：“类型转换”和“sizeof”。
>
> 1. malloc 返回值的类型是void *，所以在调用malloc 时要显式地进行类型转换，将void * 转换成所需要的指针类型。
> 2. malloc 函数本身并不识别要申请的内存是什么类型，它只关心内存的总字节数。
>
> 函数free 的原型如下：
>
> ```
> void free( void * memblock );
> ```
>
> **为什么free 函数不象malloc 函数那样复杂呢？**
>
> 这是因为指针p 的类型以及它所指的内存的容量事先都是知道的，语句free(p)能正确地释放内存。如果p 是NULL 指针，那么free对p 无论操作多少次都不会出问题。如果p 不是NULL 指针，那么free 对p连续操作两次就会导致程序运行错误。

new/delete 的使用要点：

运算符new 使用起来要比函数malloc 简单得多：

```
int *p1 = (int *)malloc(sizeof(int) * length);
int *p2 = new int[length];
```

> 这是因为new 内置了sizeof、类型转换和类型安全检查功能。对于非内部数据类型的对象而言，new 在创建动态对象的同时完成了初始化工作。如果对象有多个构造函数，那么new 的语句也可以有多种形式。

如果用new 创建对象数组，那么只能使用对象的无参数构造函数。例如

```
Obj *objects = new Obj[100];       // 创建100 个动态对象
```

不能写成

```
Obj *objects = new Obj[100](1);        // 创建100 个动态对象的同时赋初值1
```

在用delete 释放对象数组时，留意不要丢了符号‘[]’。例如

```
delete []objects; // 正确的用法
delete objects; // 错误的用法
```

后者相当于delete objects[0]，漏掉了另外99 个对象。

---

1. new自动计算需要分配的空间，而malloc需要手工计算字节数

1. new自动计算需要分配的空间，而malloc需要手工计算字节数

2. new是类型安全的，而malloc不是，比如：

```
int* p = new float[2]; // 编译时指出错误
int* p = malloc(2*sizeof(float)); // 编译时无法指出错误
```

   new operator 由两步构成，分别是 **operator new** 和 **construct**

3. operator new对应于malloc，但operator new可以重载，可以自定义内存分配策略，甚至不做内存分配，甚至分配到非内存设备上。而malloc无能为力

4. new将调用constructor，而malloc不能；delete将调用destructor，而free不能。

5. malloc/free要库文件支持，new/delete则不要。

---

1. 本质区别

   > malloc/free是C/C++语言的标准库函数，new/delete是C++的运算符。对于用户自定义的对象而言，用maloc/free无法满足动态管理对象的要求。对象在创建的同时要自动执行构造函数，对象在消亡之前要自动执行析构函数。由于malloc/free是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于malloc/free。因此C++需要一个能完成动态内存分配和初始化工作的运算符new，以及一个能完成清理与释放内存工作的运算符delete。

2. 联系

   > 既然new/delete的功能完全覆盖了malloc/free，为什么C++还保留malloc/free呢？因为C++程序经常要调用C函数，而C程序只能用malloc/free管理动态内存。如果用free释放“new创建的动态对象”，那么该对象因无法执行析构函数而可能导致程序出错。如果用delete释放“malloc申请的动态内存”，理论上讲程序不会出错，但是该程序的可读性很差。所以new/delete、malloc/free必须配对使用。

实例代码

```
class Obj
{
public:
    Obj( )
    { cout  <<  "Initialization"  <<  endl; }
    ~ Obj( )
    { cout  <<  "Destroy" <<  endl; }
    void Initialize( )
    { cout  <<  "Initialization"  <<  endl; }
    void  Destroy( )
    { cout  <<  "Destroy"  <<  endl; }
}obj;

void  UseMallocFree( )
{
    Obj   * a  =  (Obj  *) malloc( sizeof ( obj ) );      //  allocate memory
    a -> Initialize();                                    //  initialization
    // ???
    a -> Destroy();                                        // deconstruction
    free(a);                                               // release memory
}

void  UseNewDelete( void )
{
    Obj   * a  =   new  Obj;
    // …
    delete a;
}
```

> 类Obj的函数Initialize实现了构造函数的功能，函数Destroy实现了析构函数

的功能。函数UseMallocFree中，由于malloc/free不能执行构造函数与析构函数，必须调用成员函数Initialize和Destroy来完成“构造”与“析构”。所以我们不要用malloc/free来完成动态对象的内存管理，应该用new/delete。由于内部数据类型的“对象”没有构造与析构的过程，对它们而言malloc/free和new/delete是等价的。