# C 和 C++ 区别 （函数/类/struct/class）

## new delete和malloc free区别：

前者是运算符，后者是库函数，malloc和free只负责分配和释放内存空间，new在分配内存空间后会自动调用对象的构造函数，delete在释放内存空间前会自动调用对象的析构函数。

## explicit

class Point {

public:

int x, y;

explicit Point(int x = 0, int y = 0)

: x(x), y(y) {}

};

void displayPoint(const Point& p)

{

cout << "(" << p.x << ","

<< p.y << ")" << endl;

}

int main()

{

//报错，explicit声名Point对象为显示的，不能隐式调用

displayPoint(1);

Point p(1);

}

# 智能指针

在写程序的时候使用指针容易出现内存泄漏，智能指针超出了自身作用域就会自动调用析构函数，自动释放资源。

## 常用接口

T\* get(); //获取封装在内部的原始指针

T& operator\*(); //等价于指针的\*

T\* operator->(); //等价于指针的->

T& operator=(const T& val);

T\* release(); //释放指针，返回释放前的指针

void reset (T\* ptr = nullptr); //将内部指针置为ptr

## auto\_ptr

c++11已弃用

auto\_ptr<std::string> p1 (new string ("hello"));

auto\_ptr<std::string> p2;

p2 = p1; //auto\_ptr 不会报错

std::cout << \*p1 << std::endl; //报错，p1已经被p2抢占

## unique\_ptr（唯一型）

auto\_ptr<std::string> p1 (new string ("hello"));

auto\_ptr<std::string> p2;

p2 = p1; //报错，unique\_ptr保证同一时间只有一个智能指针指向该对象，//不允许p2抢占p1

## shared\_ptr（共享型，强引用）

shared\_ptr是为了解决auto\_ptr, unique\_ptr在对象所有权上的局限性(auto\_ptr是独占的)，在使用引用计数的机制上提供了可以共享所有权的智能指针。shared\_ptr 实现共享式拥有概念，多个智能指针可以指向相同对象，该对象和其相关资源会在“最后一个引用被销毁”时候释放。它使用计数机制来表明资源被几个指针共享。

可以通过成员函数use\_count()来查看资源的所有者个数，除了可以通过 new 来构造，还可以通过传入auto\_ptr, unique\_ptr, weak\_ptr来构造。当我们调用release()时，当前指针会释放资源所有权，计数减1。当计数等于 0时，资源会被释放。

## weak\_ptr（弱引用）

weak\_ptr是用来解决shared\_ptr循环引用问题的，它指向一个shared\_ptr对象，但它的构造和析构都不会影响计数的大小，可以使用lock函数转化为shared\_ptr。

循环引用的一个例子是有两个类A和B，A类中有一个shared\_ptr<B>类型的数据成员，B类中有一个shared\_ptr<A>类型的数据成员，这两个数据成员相互指向A和B的实例化对象，这样两个shared\_ptr的计数都是2，在程序结束时，两个shared\_ptr的计数都无法降低至0，会导致A和B的资源都不会释放。此时使用weak\_ptr代替其中一个shared\_ptr，这样有一个计数就是1，程序结束时一个计数就会降低至0，释放资源，然后另一个shared\_ptr计数也会降低至0，释放资源。