1. malloc/free，new/delete的区别
2. malloc/free 是c/c++标准库函数，new/delete是c++的运算符。它们都可用于申请动态内存和释放动态内存，但是new能够自动分配空间大小，而malloc需要计算字节数
3. malloc/free是库函数，malloc只能分配空间，不会自动调用构造函数。Free只能释放空间，不会自动调用析构函数。
4. new/delete是运算符，能完成动态内存分配与释放，以及自动调用构造函数与析构函数
5. new是类型安全的，而malloc不是。例如：

int\* p = new float[2]; //编译出错

int\* p = malloc(2\*sizeof(float)); //编译无法指出错误

1. malloc申请内存失败返回空指针，new申请内存失败会抛出异常。
2. new/delete的内部实现
3. new的实现

内存的分配：

CA\* p = operator new(sizeof(CA)); // 分配堆内存

CA::CA(p1)；//调用构造函数

当调用operator new[] 分配数组对象时，编译器会增加4或8字节的空间用来保存所分配的数组对象的数量。数组对象根据这个数量进行构造函数的调用

unsigned long\* ps = operator new[](20\*sizeof(CA)+sizeof(unsigned long)); //多分配unsigned long的空间

\*ps = 20; //保存对象的数量

CA\* p1 = (CA\*)(ps+1)；

CA\* pt = p1;

for (int i = 0; i < \*p; ++i)

{

CA::CA(pt);

pt += 1;

}

1. delete实现

CA::~CA(p1); //调用析构函数

operator delete(p); //释放内存

1. placement 技术

系统默认的new关键字除了分配堆内存还进行构造函数的调用，而实际中我们可能由一些已经预先分配好的内存区域，如何在这些已分配的内存来构建一个对象。

char buf1[100]; //预申请一块栈内存

CA \*p1 = (CA\*)buf1;

CA::CA(p1); //在栈内存上调用构造函数，会填充虚表指针。

p1->foo(); // foo为虚函数

p1->m\_a = 10;

char \*buf2 = new char[sizeof(CA)]; //预申请一块堆内存

CA \*p2 = (CA\*)buf2;

CA::CA(p2); //堆内存上调用构造函数

p2->foo();

p2->m\_a = 20;

p1->~CA();

p2->~CA();

delete[] buf2;

使用placement new 和placement delete

char buf1[100];

CA \*p1 = new(buf1) CA(10) //调用 operator new(size\_t, void\*)

p1->foo();

char \*buf2 = new char[sizeof(CA)];

CA \*p2 = new(buf2) CA(20); //调用 operator new(size\_t, void\*)

p2->foo();

p1->~CA();

operator delete(p1, buf1); //调用 operator delete(void\*, void\*) 还内存

p2->~CA();

operator delete(p2, buf2); //调用 operator delete(void\*, void\*)

delete[] buf2;

1. 早绑定/晚绑定与多态

编译时多态（静态多态）：通过重载函数实现，函数调用的地址在编译期间就能决定。称为早绑定。

运行时多态（动态多态）：通过虚函数实现，函数调用的地址需要在运行时才确定。称为晚绑定。

对于普通函数，当使用类的指针调用成员函数时，普通函数由指针类型决定，而虚函数由指针指向的实际类型决定。

1. vector与list的实现原理
2. vector相当于一个动态数组，size函数返回已用空间大小，capacity函数返回总空间大小。当size与capacity相等时，说明vector目前的空间已用完。如果再添加新元素，则会引起vector空间的动态增长。这个自动增长包括重新分配内存空间、拷贝原空间、释放原空间。

reserve(n)预先分配一块指定大小的内存空间，resize只改变元素的数目，不改变vector的容量。

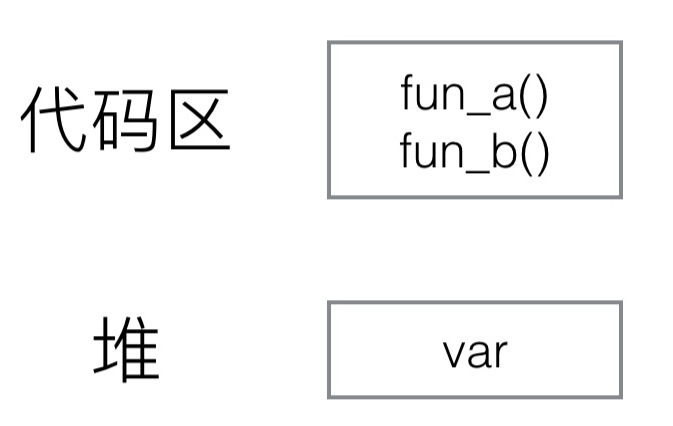
1. list 双向链表，信息块info，前驱指针Pre，后继指针Post。使用非连续的内存空间。可在两端push、pop。
2. 类中的默认函数

构造函数、析构函数、拷贝构造函数、默认重载赋值运算符函数、默认重载取地址运算符函数、默认重载取地址运算符const函数、默认移动构造函数（C++11）、默认重载移动赋值操作符函数（C++11）

1. 对象的内存布局
2. 对于一个只包含非静态成员变量和普通成员函数的类，如

class C {void fun\_a(); void fun\_b(); int var};

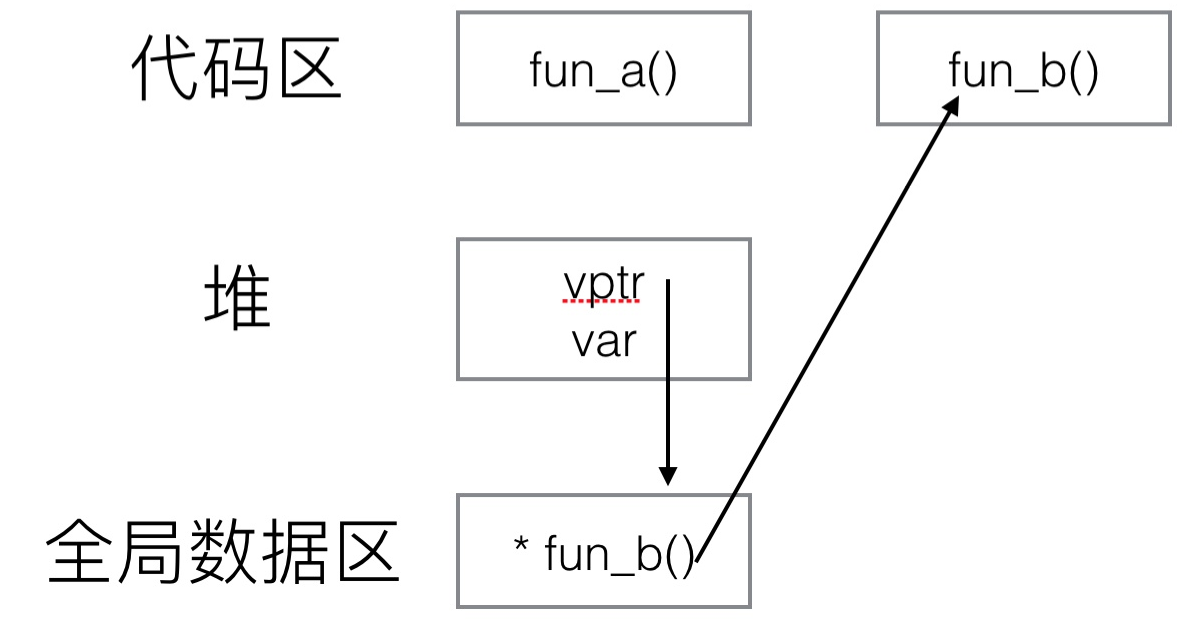
其内存布局如下图



其中成员函数放在代码区，为该类的所有对象公有，即不管新建多少个该类的对象，所对应的都是同一个函数存储区的函数。而成员变量则为各个对象所私有，即每新建一个对象都会新建一块内存区用来存储var值。

1. 对于 class D{void func\_a(); virtual void func\_b(); int var;};

其内存布局如下图：



如果sizeof(D的对象)会比sizeof(C的对象)大4个字节。多出来的这4个字节就是虚函数表指针，指向“虚函数表”，表中数据为函数指针，存储了虚函数fun\_b具体实现所对应的位置。析构函数设置为虚函数是为了当使用基类指针指向子类对象时，对基类指针进行delete操作，能够调用到子类的析构函数。

虚函数表是同一个类的所有对象公有的，只有成员变量和虚函数表指针是每个对象私有的，sizeof也只包括vptr和var所占内存的大小（sizeof不计算公有的成员，如static 成员变量及所有的成员函数代码）。vptr的初始化发生在调用构造函数时。

1. struct 对齐

8、C++ static的几种用法