OOP：（object oriented programming）面向对象语言；

OO语言（面向对象语言）三大特性：继承，封装，多态；

::预操作运算符：std::cout，在std函数库下直接实行cout操作；

**一、名字空间**

namespace：可以进行嵌套，嵌套里面和外面内容不同；

1、名字空间可以解决函数、全局变量名重复的问题，包在不同名字空间里的重复名字函数实际是两个完全无关函数；

2、名字空间允许嵌套，嵌套内部和外部没有任何关系；

3、名字空间（不是重名函数）允许重名，重名的名字空间内容会被合并进其中一个空间；

4、一个命名空间就定义了一个新的作用域，命名空间的内容仅仅局限于此个空间。它解决了C语言中只能用文件划分作用域的缺陷；

5、使用using关键字，可以引用一个名字空间，使得这个名字空间内的所有内容对自己可见；

6、如果两个名字空间里分别有相同的两个函数，那么即使我同时用using引用了两个名字空间，那么也不会报错，除非我使用了这个函数（因为使用函数会产生歧义，也就是不知道该调那个函数）。

using namespace std;

->实现std::cout<<.....；

**二、缺省参数**

函数的参数列表允许存在被赋值的参数，这些参数具备以下特点：

1、传入自己的值时，使用自己的值，否则使用函数中指定的值；

void func(int a = 10,int b = 20,int c = 30)

{

cout << a << endl;

cout << b << endl;

cout << c << endl;

}

使用函数之前，参数必定得有自己的指定值，否则不能调用时不给参数；

2、若函数参数没有自己的指定值，那么函数中缺省参数必须在函数参数末尾；

3、缺省参数不能在函数声明和定义中同时出现，在声明和定义中取其一出现即可；声明可以有很多个，但是定义只有一个；

4、缺省值必须是常量或者全局变量。

**三、函数重载**

1、C++允许在同一作用域中声明几个功能类似的同名函数，这些同名函数的形参列表(参数个数或类型或顺序)必须不同，常用来处理实现功能类似数据类型不同的问题；

2、函数名相同时，参数相同就不能实现函数重载，不会考虑返回值是否不同；

3、extern "C"表示某些函数按照C风格进行函数编译；也可以用大括号将部分代码括起来进行C风格编译；

4、C++中函数重载底层是怎么处理的？C++在进行编译时会用@加上域名和函数名；

**四、名字修饰**

1、在C语言中，名字修饰只是在函数名之前加下划线，所以只要函数名相同，就会导致冲突；

2、在C++中，名字修饰是“？函数名@域名1@域名2...@@参数列表@Z”的格式构成的，包含：a：函数名，b：所在域，c：参数列表，所以在C++中，以上三个必须完全相同，才会出现冲突，这就是函数重载的原理；

**五、引用**

1、引用是给一个变量起别名，两个名字都指一个变量，操作谁，在结果上是一致的；

2、引用是代替指针完成跨栈操作的，所以它具备指针跨栈的一切特点；

3、引用的底层实现实际是指针；

4、在语法概念上引用就是一个别名，没有独立空间，和其引用实体共用同一块空间，在底层实现上实际是有空间的，因为引用是按照指针方式来实现的。

**特点：**

1、引用在定义时，必须初始化；

2、一个变量可以有多个引用；

3、一个引用一旦引用了一个变量，就不能再引用其他变量了。

引用比指针使用起来相对安全，因为指针有可能存在其他指向，但是引用不会；

**六、内联函数**

【相对于宏较安全，能使用内联函数就不用宏】

1、调用时不创建新栈而直接在调用处展开的函数叫做内联函数，关键字为inline；

2、内联函数是一个对编译器的建议，如果函数过于复杂，编译器会不接受你的建议，而处理成普通的函数；

3、inline函数可以代替带参数的宏。

**七、auto关键字（C++11新增）**

数组的元素类型不能包含auto类型；

1、auto是一个类型修饰符，他会根据你给变量初始化的值的类型来决定自己是什么类型；

2、auto定义的变量必须初始化，否则不知道auto是什么类型；

3、auto可以连续定义变量，类型由第一个定义的变量决定，后续不能违背这个类型，但是可以定义对应类型的指针和引用；

注:

auto不能做函数的参数类型；

auto不能参与数组的类型；

C++11后，auto会失去原本的含义；

**八、类与对象**

1、C++中的结构体struct可以看做是一个域；struct是一个简化版的类；

2、结构体里面的函数，属于这个域，但是不占结构体的空间；

3、结构体在C和C++中的不同：

（1）在C++中，用结构体定义变量不需要加struct，而C中不行；

（2）在C++中，可以定义空结构体，大小为1，而C中不行；

（3）在C++中，可以在结构体中声明甚至实现函数，在C中只能放函数指针；

（4）在C++中，成员函数直接可以访问本结构体的成员变量而无需传入，在C中，函数和结构体并无直接关联；

**注：**

（1）成员函数不影响结构体的大小，因为成员函数是放在公共区域的，只是在这个结构体域中而已；

（2）进行结构体内存对齐，是要进行空间换时间，保证跳转的次数减少，效率变高；

4、类

类是一类特殊的结构体，在上述的结构体中加入public：,把struct改成class就成了一个简单的类；

5、访问限定符

public：正常访问；

private：只能在类内部访问；

protected：目前跟private没有区别；

**特点：**

（1）访问限定符只在编译阶段生效，编译好后，在运行阶段并没有限定；

（2）结构体其实也能使用访问限定符，只是一般不去使用，因为它是用来兼容C结构体的；

（3）结构体中默认是public，而类中默认是private；

6、this指针（C++关键字）

（1）一般成员函数中都含有一个this指针，这个指针指向调用这个成员函数的对象；

（2）成员函数中，this指针中的成员不需要this指针访问，直接访问即可；

（3）this的类型是当前类类型的指针，this是一个常量指针，不能更改指向；

（4）this指针是成员函数栈里的临时变量，随栈的存在而存在；

（5）this指针可以为空，不能调用修改成员变量的成员函数，成员函数在公共区域里；

（6）写在类内部的成员函数默认inline（内联函数）；

7、类的6个默认成员函数

（1）构造函数：只要对象创造，就会自动调用构造函数，不用手动(显示)调用；

a.当你自己写了构造函数后，系统就不会给你提供默认构造函数；

b.构造函数函数名是类名，或者就是跟类名相同的函数就是构造函数，不存在返回值；

c.构造函数可以存在参数，它与其他的构造函数是以函数重载的方式共同存在；

（2）拷贝构造函数：指的是参数为本类其他对象的引用的构造函数，它在给对象初始化成本类其他对象时调用，系统会自动提供一个拷贝构造函数；

（3）析构函数：是当一个栈被销毁前调用的，在C++中，当一个函数栈被销毁前，会调用栈中每一个对象的析构函数；

a.析构函数不存在参数，也不存在返回值；

b.它的名字的类名前加波浪线~；

c.系统会自动提供一个什么都不做的析构函数；

d.浅拷贝：直接复制内存；

e.深拷贝：当成员中有指向堆的指针，就必须重新给该指针分配空间，然后将目标对象指针所指空间的内容拷贝到新分配的空间。（如果不这样做，会导致两个指针指向同一片空间，从而在析构中多次释放）。

（4）赋值重载

（5）普通对象

（6）const 对象

**九、初始化列表**

1、狭义初始化

在定义变量的时候直接进行初始化的这种行为叫做狭义初始化：

int a = 3;

2、广义初始化

第一次给变量赋值就叫做初始化的情况叫广义初始化

int a;

...//跟a无关的代码

a = 3

初始化列表相当于狭义初始化，而构造函数内部相当于广义初始化

所以初始化列表可以解决一些只能用狭义初始化进行初始化的变量，

例如：

（1）const变量

（2）引用

（3）没有无参构造的类的对象

//修饰this指针

explicit：阻止单参构造的不规范调用

单参数的构造函数，可以用=直接调用，例如：

假设CT类中有一个单参数的构造函数，参数类型为int或int相关类型，那么，“CT a = 3;” 这种写法就是被允许的，但是这种写法十分别扭，看上去好像直接把3赋给了a，为了避免这种写法，可以在构造函数加explicit，使得这样的写法变得无效。

**十、赋值运算符重载**

operator=

运算符重载：

将运算符看成函数，把他的几目当成参数，通过参数的类型识别出对应的操作方法，相当于函数重载。

运算符重载有指定的规则，规则根据运算符来制定。

类会自动提供一个赋值运算符的重载(4)，执行的是浅拷贝，跟拷贝构造相同

**十一、const成员函数**

const加在成员函数的末尾，代表这个函数中的this是const修饰的

如果一个对象是const对象，那么它不能调用非const的成员函数

**十二、取地址运算符重载**

类会自动提供两个取地址运算符重载，一个是针对普通对象的，一个是针对const对象的

**十三、静态成员**

静态成员跟类走不跟对象走，类在他在，而一般成员是对象在他才在。所以静态成员可以通过类名直接调用，而普通成员必须通过对象调用。

静态成员也有private，public，protected之分

1、静态成员变量

1、所有对象共享，无论谁改了，所有的一起改

2、存储在全局区，不占用类的空间，所以取sizeof的时候不算在内

3、赋初值只能在类外，赋值时不加static，用“类型 类名::变量名 = n” 直接赋值

2、静态成员函数

只能直接访问静态成员变量，无法访问其他的普通成员，因为他没有this指针

当静态成员函数引用本类的非静态成员的时候，应该加对象名和成员运算符“.”；

**十四、友元函数**

友元就是让一个外部函数或者外部类能访问我的私有成员；

友元打破了原有的权限制度，所以十分危险，不建议使用；

内部类：

在另一个类内声明的类；

它属于外部类域中的一员，实例化需要使用域操作运算符（::），同时它有权限的限制。（private的类声明不能直接在外面声明对象）；

外部类是内部类的友元，但是内部类不是外部类的友元；

**十五、内部类**

特性：

1、内部类可以定义在外部类的public、protected、private都是可以的。

2、注意内部类可以直接访问外部类中的static、枚举成员，不需要外部类的对象/类名。

1. sizeof(外部类)=外部类，和内部类没有任何关系。

内存管理：

const，int，i等在栈里面；

static在全局区；

内存中“常量区”以及“代码区”是只读区域；

动态分配，使用“堆”进行；

无名变量在堆上生成；

动态内存分配：

一、内存区域：

1、栈，堆，全局区，常量区，代码区；

栈：系统自动分配的空间，只要不特殊声明，就定义在栈区，函数的区域也在栈上，栈是向下增长；

堆：使用动态内存分配的方式，可以申请堆空间，用完要手动释放；

全局区：静态变量、全局变量（static）；

常量区：代码中的数字、字符等常量，例如'a'等；

代码区：存放可执行代码，避免频繁的读硬盘。

2、data，bss，text

data：（数据区）堆栈，以及已初始化的全局变量（除零以外的初始化）；

bss：未初始化的变量，或者初始化为0的变量，不占用可执行文件；

text：文章段（只读段），包括常量区和代码区；

二、动态内存分配的常见函数和运算符

函数：C语言中stdlib.h中提供的系列函数

void \*malloc(size\_t)

void \*calloc(size\_t,size\_t)

malloc不会进行初始化，calloc会初始化空间为0，所以在追求效率的时候，使用malloc；

void \*realloc(void \* p,size\_t,size)

1、申请size个字节的空间；

2、将p中的值拷贝到刚申请的空间中；

3、释放p，返回刚申请的空间；

void free(void \*)

运算符：new delete

通过动态内存分配实例化对象的时候，new/delete会自动调用构造/析构函数，而malloc/free不会，所以在给类分配空间是，最好使用new/delete

三、operator new与operator delete函数

new和delete是用户进行动态内存申请和释放的操作符；

operator new和operator delete是系统提供的全局函数；

new在底层调用operator new全局函数来申请空间；

delete在底层通过operator delete全局函数来释放空间；

operator new实际上也是通过malloc来申请空间，如果malloc申请空间成功就直接返回，否则执行用户提供的空间不足应对措施，如果用户提供该措施就继续申请，否则就抛异常；

operator delete最终通过free来释放空间；

四、new和delete的实现原理

其中new/delete申请和释放的是单个元素的空间，new[]和delete[]申请的是连续空间；

new在申请空间失败后会抛异常，malloc会返回NULL；

new的原理：

1、调用operator new函数申请空间；

2、在申请的空间上执行构造函数，完成对象的构造；

delete的原理：

1、在空间上执行析构函数，完成对象中资源的清理工作；

2、调用operator delete函数释放对象的空间；

new T[N]的原理：

1、调用operator new[]函数，在operator new[]中实际调用operator new函数完成N个对象空间的申请；

2、在申请的空间上执行N次构造函数；

delete[]的原理：

1、在释放的对象空间上执行N次析构函数，完成N个对象中资源的清理；

2、调用operator delete[]释放空间，实际在operator delete[]中调用operator delete来释放空间；

五、malloc/free和new/delete的区别

共同点：都是从堆上申请空间，并且需要手动释放；

不同点：

1、malloc和free是函数，new和delete是操作符；

2、malloc申请的空间不会初始化，new可以初始化；

3、malloc申请空间时，需要手动计算空间大小并传递，new只需在其后跟上空间的类型即可；

4、malloc申请空间失败后，返回值是NULL，因此使用是必须判空，new不需要，但是new需要捕获异常；

5、malloc的返回值为void\*，在使用时必须强转，new不需要，因为new后跟的是空间的类型；

6、申请自定义类型对象时，malloc/free只会开辟空间，不会调用构造函数与析构函数，而new在申请空间后会调用析构函数完成对象的初始化，delete在释放空间前会调用析构函数完成空间中资源的清理；

7、new/delete比malloc/free的效率稍微低点，因为new/delete的底层封装了malloc/free;

六、new定位表达式

1、通过new定位表达式申请的空间实际并没有开辟新空间，所以无需通过delete释放，如果需要调用析构函数，可以用显示调用的方式，例如：通过pd->~Date()的方式析构。

malloc、calloc、realloc之间的区别：

1、malloc

函数原型：void\* malloc(unsigned size);

函数功能：在堆内存中分配一块长为size字节的连续区域，size为内存空间长度；

2、calloc

函数原型：void\* calloc(size\_t numElements,size\_t sizeOfElement);

函数功能：与malloc相似，参数sizeOfElement为单位元素长度；numElement为元素个数，即在内存中申请numElement\*sizeOfElement字节大小的内存空间；

3、realloc

函数原型：void\* realloc（void\* ptr,unsigned newsize）;

函数功能：

使用realloc函数为ptr重新分配大小为size的内存空间。

（1）对ptr进行判断，如果ptr为NULL，则函数相当于malloc(new\_size)，试着分配大小为new\_size的内存，成功将地址返回，否则返回NULL。如果ptr不为NULL，则进入（2）

（2）查看ptr是不是在堆中，如果不是的话会抛出realloc invalid pointer异常，如果ptr在堆中，则查看new\_size大小，如果new\_size大小为0，则相当于free(ptr)，将ptr指向的内存空间释放，返回NULL。如果new\_size小于原大小，则ptr中的数据可能会丢失，此时只有把new\_size大小的数据保存起来；如果size等于原大小，等于什么也没有做；如果size大于原大小，则查看ptr指向的位置还有没有足够的连续内存空间，如果有的话，分配更多的空间，返回的地址和ptr相同，如果没有的话，在更大的空间内查找，如果找到size大小的空间，将旧的内容拷贝在新的内存中，把旧的内存释放掉，返回新地址，否则返回NULL。

malloc、calloc、realloc之间的区别：

1、是否会对申请的内存空间初始化：

函数malloc不能初始化所分配的内存空间，函数calloc会将分配的内存空间中的每一位都初始化为0；

2、功能上的区别：

malloc与calloc用来动态分配内存空间，而realloc则是对给定的指针所指向的内存空间进行扩大或者缩小。

一、++，--的重载

在前的重载不需要参数

在后的重载需要一个象征性的int参数

二、string类的应用

构造

默认构造

传入一个字符串

拷贝构造

n个字符

size、length

这两个函数没有参数，实现原理相同，返回值为字符串的长度

capacity

没有参数，返回值是容器的容量

（容量是16的倍数-1，容量只会增大不会缩小）

empty

判断容器是否为空，如果为空返回1否则返回0

clear

清空字符串

reserve

调整容量（只会向大调整，为了避免频繁扩容导致的效率损失）

参数是要调整到的容量大小，如果比已有的容量小，则无效。

resize

强制调整字符串长度，参数是要调整的大小，第二个参数是缺省参数，是向大调整时，扩展部分的空间要赋的值，如果不给第二个参数，那么默认给0，resize会先调用reserve。

append

可以对字符串进行追加，类似+=，但也可以传n个字符的形式

重载：[] += + = > < >= <= == !=

vector（顺序表） 矢量容器；

重载：

insert（三种）

erase（两种）

一组数组中数据出现三次进行查找，查找出只出现一次的数字；

sum = (sum ^ n) &~ tmp;

tmp = (tmp ^ n) &~ sum;

1、删除排序数组中的重复项

2、在一组数组中找出只出现一次的两个数

3、连续子数组的最大和

1、非类型模板参数

模板在定义的时候可以在尖括号里定义一个常量，作为参数，例如：template<class T, size\_t N = 10>。

\*常量的类型只能是：整型（及相关类型）、指针、引用。

\*传入的时候必须是const类型的数据。

2、模板的特化

（1）函数模板的特化

针对某个函数模板，特化它其中的某种情况

\*特化必须针对一个已经存在的函数模板，不能单独存在。

\*特化不影响调用优先级。

（2）类模板的特化

写法和作用都跟函数模板的特化相似。

\*写实拷贝：写的时候拷贝；

输出流：cerr直接输出在屏幕上

IP地址由四个char字符存入，大小为四字节。

main函数可以直接递归