1:指针（*）、引用（&）、解引用（*）、取地址（&）、的概念和区别

概念：

指针指向一块内存，指针保存的是内存的地址；引用是变量的别名，本质是引用该变量的地址。

解引用是取指针指向的地址的内容，取地址是获得变量在内存中的地址。

区别：

（1）引用使用是无需解引用，指针需解引用。

（2）引用不能为空，指针可以为空。

（3）引用在定义时被初始化一次，之后不可变；指针指向的值和本身的值是可变的，也就是说指针只是一块地址，地址里的东西可变。

（4）程序会给指针变量分配内存区域，而引用不需要分配内存区域。

2:memset、memcpy 和 strcpy 的区别

memcpy 是内存拷贝函数，可以拷贝任何数据类型的对象，例如 memcpy(b, a, sizeof(b))。

strcpy 只能拷贝字符串，遇到’\0′结束拷贝。

memset 用来对一段内存空间全部设置为某个字符，例如:char a[100];memset(a, ’’, sizeof(a))。

3:struct 和 class 的区别,struct 与 union 的区别

struct 和 class 都是声明类的关键字

区别是：

（1）在默认情况下,struct 的成员变量是公共(public)的；在默认情况下，class 的成员变量是私有(private)的。

（2）struct 保证成员按照声明顺序在内存中存储。class 不能保证。

（3）对于继承来说，class 默认是 private 继承，struct 默认是 public 继承。

区别是（union 和 class 同理）：

（1）一个 union 类型的变量，所有成员变量共享一块内存，该内存的大小有这些成员变量中长度最大的一个来决定，

struct 中成员变量内存都是独立的。

（2）union 分配的内存是连续的，而 struct 不能保证分配的内存是连续的。

struct 与 union 的区别：

（1）struct 与 union 都由多个不同的数据类型成员组成。但是， union 的成员共享存储空间一块地址空间,而 struct 的成员则不是。 （2）因此，对于 union 的一个成员赋值，那么其它成员会重写，而 struct 则不会

4：指针在 16 位机、32 位机、64 位机分别占用多少个字节

16 位机--2 字节、32 位机--4 字节、64 位机--8 字节。

5：如何引用一个已经定义过的全局变量？区别是什么

如果在同一个文件中，直接引用即可。

如果不在同一个文件，有两种方式：

（1）直接引用头文件就可以使用了。

（2）用 extern 关键字重新声明一下。

6：全局变量可不可以定义在可被多个.C 文件包含的头文件中？因为全局变量的作用域是整个源程序，可以声明多次，但是只能定义一次。变量的声明一般放在头文件中，那么变的定义可以放在头文件中吗？在实际的编程中一般很少在头文件中定义全局变量，因为多次引用可能重定义。**7：do……while 和 while……do 有什么区别？**

do...while 先执行循环再判断条件，while...do 先判断条件再执行循环。

8：对于一个频繁使用的短小函数,在 C 语言中应用什么实现,在 C++中应用什么实现?

C 用宏定义，C++用 inline。**9：main 函数执行以前，会执行什么代码？**

全局对象的构造函数会在 main 函数之前执行，比如 int a；初始化为 0。

10：main 主函数执行完毕后，会执行什么代码？

可以，使用 on_exit 注册的函数会在代码执行完毕后执行：

```
void main( void )
{
String str(“zhanglin”);
on_exit( fn1 );
on_exit( fn2 );
on_exit( fn3 );
on_exit( fn4 );
printf( “This is executed first.\n” );
}
int fn1()
{
printf( “next.\n” );
return 0;
}
```

11：局部变量能否和全局变量重名？

可以，但是局部会屏蔽全局。要用全局变量，需要使用域作用符“::”。

12：描述内存分配方式以及它们的区别?1．从静态存储区域分配。该存储区域在程序编译的时候就已经分配好了，这块内存在程序的整个运行期间都存在。例如全局变量，static 变量。

2． 在栈上创建。在执行函数时，函数的局部变量存储在该区域，函数执行结束时会释放该存储空间。栈内存分配运算内置于处理器的指令集。

3． 从堆上分配，亦称动态内存分配。程序在运行的时候用 malloc 或 new 申请任意多少的内存，程序员自己负责在何时用 free 或 delete 释放内存。 动态内存的生存期由程序员决定，使用非常灵活。

13：类的成员函数重载、覆盖和隐藏的概念和区别？概念：重载是指再同一个作用域内，有几个同名的函数，但是参数列表的个数和类型不同。 函数覆盖是指派生类函数覆盖基类函数，函数名、参数类型、返回值类型一模一样。派生类的对象会调用子类中的覆盖版本，覆盖父类中的函数版本。 “隐藏”是指派生类的函数屏蔽了与其同名的基类函数。覆盖和重载的区别：函数是否处在不同的作用域，参数列表是否一样；基类函数是否有 virtual 关键字。

隐藏和覆盖的区别：

（1）派生类的函数与基类的函数同名，但是参数不同。此时，不论有无 virtual 关键字，基类的函数将被隐藏。

（2）派生类的函数与基类的函数同名，参数也相同，但是基类函数没有 virtual 关键字。此时，基类的函数被隐藏有 virtual，就是覆盖。

如果子类覆盖父类的函数但是不加 virtual ,也能实现多态，由于 virtual 修饰符会被隐形继承，但是尽量加上。

14：static 关键字？

为什么引入 static：

编译器为局部变量在栈上分配空间，但是函数执行结束时会释放掉。所以 static 可以解决函数执行完后变量值被释放的问题，static 变量存储在静态存储区。

缺点：

不安全，因为作用域是全局的。

优点：

因为静态变量对所有对象所公有的，可以节省内存，提高时间效率。

静态数据成员使用方式：

<类名>::<静态成员名>

作用：

（1）可以用于全局变量的定义：为该变量分配静态存储区。程序运行结束前不会被释放。 （2）声明和定义静态成员函数：表示该函数为静态函数，只能在本文件中被调用。 （3）定义静态局部变量：只被初始化一次，只有程序运行结束才会释放。区别是作用域的范围。

15：const 与#define 的概念和优缺点？

const 用来定义常量、修饰函数参数、修饰函数返回值，可以避免被修改，提高程序的健壮性。

define 是宏定义，在编译的时候会进行替换，这样做的话可以避免没有意义的数字或字符串，便于程序的阅读。

区别：

1. const 定义的数据有数据类型，而宏常量没有数据类型。编译器可以对 const 常量进行类型检查。而对宏定义只进行字符替换，没有类型安全检查，所以字符替换时可能出错。

例子：

写一个“标准”宏 MIN，这个宏输入两个参数并返回较小的一个。

```
#define MIN(A,B) ((A) <= (B) (A) : (B))
```

least = MIN(a, b);

下面的关键字 const 是什么含意：

const int a;//a 是一个常整型数

int const a;//a 是一个常整型数

const int *a;//a 是一个指向常整型数的指针

int * const a;//a 是一个指向整型数的常指针

int const * a const;//a 是一个指向常整型数的常指针

static 关键字的作用：

（1）函数体内 static 变量的作用范围为该函数体，不同于 auto 变量，该变量的内存只被分配一次，因此其值在下次调用时仍维持上 次的值；

（2）在模块内的 static 全局变量可以被模块内所用函数访问，但不能被模块外其它函数访问；

（3）在模块内的 static 函数只可被这一模块内的其它函数调用，这个函数的使用范围被限制在声明它的模块内；

（4）在类中的 static 成员变量属于整个类所拥有，对类的所有对象只有一份拷贝；

（5）在类中的 static 成员函数属于整个类所拥有，这个函数不接收 this 指针，因而只能访问类的 static 成员变量。

const 关键字的作用：

（1）欲阻止一个变量被改变，可以使用 const 关键字。在定义该 const 变量时，通常需要对它进行初始化，因为以后就没有机会再去改变它了；

（2）对指针来说，可以指定指针本身为 const，也可以指定指针所指的数据为 const，或二者同时指定为 const；

（3）在一个函数声明中，const 可以修饰形参，表明它是一个输入参数，在函数内部不能改变其值；

（4）对于类的成员函数，若指定其为 const 类型，则表明其是一个常函数，不能修改类的成员变量；

（5）对于类的成员函数，有时候必须指定其返回值为 const 类型，以使得其返回值不为“左值”。例如：

16：堆栈溢出的原因？

没有回收垃圾资源

栈溢出：

一般都是由越界访问导致的。例如局部变量数组越界访问或者函数内局部变量使用过多，超出了操作系统为该进程分配的栈的大小。

堆溢出：

由于堆是用户申请的，所以溢出的原因可能是程序员申请了资源但是忘记释放了。

17：刷新缓冲区方式?

换行刷新缓冲区：printf(“\n”);程序结束刷新缓冲区：return 0;**18：类和对象的两个基本概念？**类的作用或概念：用来描述一组具有相似属性的东西的对象的一种数据结构。类中有数据成员的声明和定义，有成员函数的实现代码。对象就是类的实例化。计算机中想要使用类，只能进行实例化。

19：介绍一下 STL，详细说明 STL 如何实现 vector。

STL 是标准模版库，由容器算法迭代器组成。

STL 有以下的一些优点：

（1）可以很方便的对一堆数据进行排序（调用 sort（））；

（2）调试程序时更加安全和方便；

（3）stl 是跨平台的，在 linux 下也能使用。

vector 实质上就是一个动态数组，会根据数据的增加,动态的增加数组空间。

什么是容器。如何实现？

容器是一种类类型，用来存储数据

STL 有 7 种主要容器：vector,list,deque,map,multimap,set,multiset.

20：变量的声明和定义有什么区别

变量的声明是告诉编译器我有某个类型的变量，但不会为其分配内存。但是定义会分配了内存。

21：简述#define #endif 和#ifndef 的作用

这三个命令一般是为了避免头文件被重复引用。

#ifndef CH_H //意思是如果没有引用 ch.h

#define CH_H //引用 ch.h

#endif //否则不需要引用

22：引用与指针有什么区别？

（1） 引用必须被初始化，指针不必。

（2） 引用初始化以后不能被改变，指针可以改变所指的对象。

（3） 不存在指向空值的引用，但是存在指向空值的指针。

23：C++继承机制？

n 类成员的访问控制方式

public：类本身、派生类和其它类均可访问；

protected：类本身和派生类均可访问，其它类不能访问；

private（默认）：类本身可访问，派生类和其它类不能访问。

继承成员的访问控制规则

——由父类成员的访问控制方式和继承访问控制方式共同决定

private+public（protectd，private）=>不可访问

pubic（protected）+public=>public（protected）

public（protected）+protected=>protected

public（protected）+private（默认）=>private

C++中的模板和 virtual 异同？ ==>?

private 继承和 public 继承区别？ ==>?

24：什么是内存泄露？C++内存泄漏检测内存泄露是指程序中动态分配了内存，但是在程序结束时没有释放这部分内存,从而造成那一部分内存不可用的情况

有一些内存泄漏的检测工具，比如 BoundsChecker。

静态内存泄漏通过工具或者仔细检查代码找到泄漏点。

动态的内存泄漏很难查，一般通过在代码中加断点跟踪和 Run-Time 内存检测工具来查找。

内存泄漏的检测可以分以下几个步骤：

（1）看代码 new 之后是否 delete，就是申请了静态内存用完是否释放。看析构函数是否真的执行，如果没有真正执行，就需要动态释放对象；

（2）让程序长时间运行，看任务管理器对应程序内存是不是一直向上增加；

（3）使用常用内存泄漏检测工具来检测内存泄漏点。

25：头文件的作用是什么？

（1）头文件用于保存程序的声明。 （2）通过头文件可以来调用库函数。因为有些代码不能向用户公布，只要向用户提供头文件和二进制的库即可。用户只需要按照头文件中的接口声明来调用库功能，编译器会从库中提取相应的代码。 （3）如果某个接口被实现或被使用时，其方式与头文件中的声明不一

致，编译器就会指出错误，这一简单的规则能大大减轻程序员调试、改错的负担。

26：函数模板与类模板有什么区别？

答：函数模板的实例化是由编译程序在处理函数调用时自动完成的，而类模板的实例化必须由程序员在程序中显式地指定。

函数模板是模板的一种，可以生成各种类型的函数实例：

```
template

Type min( Type a, Type b )

{

return a < b ? a : b;

}
```

参数一般分为类型参数和非类型参数：

类型参数代表了一种具体的类型

非类型参数代表了一个常量表达式

27：system（”pause”）的作用？

调用 DOS 的命令，按任意键继续，和 getchar（）差不多；省去了使用 getchar（）；区别是一个属于系统命令，一个属于 c++标准函数库。

28：析构函数和虚函数的用法和作用？

析构函数是类成员函数，在类对象生命期结束的时候，由系统自动调用，释放在构造函数中分配的资源。

虚函数是为了实现多态。含有纯虚函数的类称为抽象类,不能实例化对象,主要用作接口类

Test(int j):pb(j),pa(pb+5) { }

~Test() { cout<<"释放堆区 director 内存空间 1 次"; }

析构函数的特点：

1. 函数名称固定：~类名( ) 2. 没有返回类型，没有参数 3. 不可以重载，一般由系统自动的调用

29：编写一个标准 strcpy 函数

可以拿 10

```
char * strcpy( char *strDest, const char *strSrc )//为了实现链式操作，将目的地址返回，加 3 分!
{
assert( (strDest != NULL) &&(strSrc != NULL) );
char *address = strDest;
while( (*strDest++ = * strSrc++) != ‘\0’ );
return address;
}
```

30：new、delete；malloc、free 关系

new 和 delete 是一组，new 用调用构造函数来实例化对象和调用析构函数释放对象申请的资源。

malloc 和 free 是一对，malloc 用来申请内存和释放内存，但是申请和释放的对象只能是内部数据类型。

区别：

malloc 与 free 是 C++/C 语言的标准库函数，new/delete 是 C++的运算符。

maloc/free 只能操作内部数据类型。

31：delete 与 delete []区别 都是用来调用析构函数的：

（1）delete 只会调用一次析构函数，delete[]会调用每一个成员的析构函数。

（2）delete 与 new 配套，delete []与 new []配套，用 new 分配的内存用 delete 删除用 new[]分配的内存用 delete[]删除

32：继承优缺点 优点：

继承可以方便地改变父类的实现，可以实现多态，子类可以继承父类的方法和属性。

缺点：

破坏封装，子类和父类可能存在耦合。

子类不能改变父类的接口。

33：C 和 C++有什么不同?

（1）c 是面向过程的，也就是说更偏向逻辑设计；c++是面向对象的，提供了类，偏向类的设计。

（2）c 适合要求代码体积小的，效率高的场合，如比如嵌入式。

34：析构函数的调用次序，子类析构时要调用父类的析构函数吗？

析构函数调用的次序是：先派生类的析构后基类的析构，也就是说在基类的的析构调用的时候,派生类的信息已经全部销毁了定义一个对象时先调用基类的构造函数、然后调用派生类的构造函数；**35：什么是“野指针”？**野指针指向一个已删除的对象或无意义地址的指针。与空指针不同，野指针无法通过简单地判断是否为 NULL 避免，而只能通过养成良好的编程习惯来尽力避免。造成的主要原因是：指针变量没有被初始化，或者指针 p 被 free 或者 delete 之后，没有置为 NULL。**36：常量指针和指针常量的区别？**常量指针：是一个指向常量的指针。可以防止对指针误操作而修改该常量。
指针常量：是一个常量，且是一个指针。指针常量不能修改指针所指向的地址，一旦初始化，地址就固定了，不能对它进行移动操作。但是指针常量的内容是可以改变。

37：sizeof 的概念(作用)，举例

sizeof 是 C 语言的一种单目操作符，并不是函数。sizeof 以字节的形式返回操作数的大小。

（1）sizeof(int **a[3][4])

int **p; //16 位下 sizeof(p)=2, 32 位下 sizeof(p)=4

总共 3*4*sizeof(p)

（2）若操作数具有类型 char、unsigned char 或 signed char，其结果等于 1。

（3）当操作数是指针时，sizeof 依赖于系统的位数　（4）当操作数具有数组类型时，其结果是数组的总字节数。　（5）联合类型操作数的 sizeof 是其最大字节成员的字节数。

结构类型操作数的 sizeof 是这种类型对象的总字节数（考虑对齐问题时）。这样做可以提高程序的性能，避免访问两次内存；　（6）如果操作数是函数中的数组形参或函数类型的形参，sizeof 给出其指针的大小。

sizeof 和 strlen（）的区别：① sizeof 是运算符，计算数据所占的内存空间；strlen（）是一个函数，计算字符数组的字符数；
② sizeof 可以用类型作参数；strlen（）只能用 char*作参数，必须是以‘/0’结束
③ 数组做 sizeof 的参数不退化,传递给 strlen 就退化为指针了;
④ sizeof 操作符的结果类型是 size_t，它在头文件中 typedef 为 unsigned int 类型。该类型保证能容纳实现建立的最大对象的字节大小。

38：如果 NULL 和 0 作为空指针常数是等价的，那到底该用哪一个？

#define NULL 0

按道理说，null 和 0,没有区别，但为何要多此一举呢，

（1）什么是空指针常量?

0、0L、'\0'、3 - 3、0 * 17 以及 (void*)0 都是空指针常量。

（2）什么是空指针?

如果 p 是一个指针变量，则 p = 0;、p = 0L;、p = '\0';、p = 3 - 3;、p = 0 * 17; 中的任何一种赋值操作之后， p 都成为一个空指针，由系统保证空指针不指向任何实际的对象或者函数。

（3）什么是 NULL?

即 NULL 是一个标准规定的宏定义，用来表示空指针常量。因此，除了上面的各种赋值方式之外，还可以用 p = NULL; 来使 p 成为一个空指针。

（4）空指针指向了内存的什么地方？

标准并没有对空指针指向内存中的什么地方这一个问题作出规定，一般取决于系统的实现。我们常见的空指针一般指向 0 地址，即空指针的内部用全 0 来表示。

空指针的“逻辑地址”一定是 0，对于空指针的地址，操作系统是特殊处理的。并非空指针指向一个 0 地址的物理地址。

在实际编程中不需要了解在我们的系统上空指针到底是一个 0 指针还是非 0 地址，我们只需要了解一个指针是否是空指针就可以了——编译器会自动实现其中的转换，为我们屏蔽其中的实现细节。

（5）可以用 memset 函数来得到一个空指针吗？

这个问题等同于：如果 p 是一个指针变量，那么 memset(&p, 0, sizeof(p)); 和 p = 0;是等价的吗？

答案是否定的，虽然在大多数系统上是等价的，但是因为有的系统存在着“非零空指针” （nonzero null pointer），所以这时两者不等价。

（6）可以定义自己的 NULL 的实现吗？兼答"NULL 的值可以是 1、2、3 等值吗？"类似问题

NULL 是标准库中的一个符合上述条件的保留标识符。所以，如果包含了相应的标准头文件而引入了 NULL 的话，则再在程序中重新定义 NULL 为不同的内容是非法的，其行为是未定义的。也就是说，如果是符合标准的程序，其 NULL 的值只能是 0，不可能是除 0 之外的其它值，比如 1、2、3 等。

（7）malloc 函数在分配内存失败时返回 0 还是 NULL?

malloc 函数是标准 C 规定的库函数。在标准中明确规定了在其内存分配失败时返回的是空指针

39：如果 NULL 定义成#define NULL ((char *)0) 难道不就可以向函数传入不加转换的 NULL 了吗？

不行。因为有的机器不同类型数据的指针有不同的内部表达。如果是字符指针的函数没有问题，但对于其它类型的指针参数仍然有问题，而合法的构造如 FILE *fp = NULL;则会失败。
如果定义#define NULL ((void *)0)除了潜在地帮助错误程序运行以外，这样的定义还可以发现错误使用 NULL 的程序。无论如何，ANSI 函数原型确保大多数指针参数在传入函数时正确转换。

40：使用非全零的值作为空指针内部表达的机器上，NULL 是如何定义的？

跟其它机器一样：定义为 0 。当程序员请求一个空指针时，无论写“0” 还是“NULL”，都是有编译器来生成适合机器的空指针的二进制表达形式。因此，在空指针的内部表达不为 0 的机器上定义 NULL 为 0 跟在其它机器上一样合法：编译器在指针上下文看到的未加修饰的 0 都会被生成正确的空指针。

41：NULL 是什么，它是怎么定义的？

很多人不愿意在程序中到处出现未加修饰的 0。因此定义了预处理宏 NULL 为空指针常数，通常是 0 或者((void *)0) 。希望区别整数 0 和空指针 0 的人可以在需要空指针的地方使用 NULL。

42：用“if(p)” 判断空指针是否可靠？如果空指针的内部表达不是 0 会怎么样？

表达式中要求布尔值时：如果表达式等于 0 则认为该值为假。if(p) 等价于 if(p != 0)。

43：怎样在程序里获得一个空指针？

在指针上下文中的常数 0 会在编译时转换为空指针。
```
char *p = 0;
if(p != 0)
```

44：空指针到底是什么？

空指针表示“未分配” 或者“尚未指向任何地方” 的指针。
空指针在概念上不同于未初始化的指针。空指针可以确保不指向任何对象或函数；而未初始化指针则可能指向任何地方。

45：我能否用 void 指针作为参数，使函数按引用接受一般指针？**

不可移植。C 中没有一般的指针的指针类型。void* 可以用作一般指针只是因为当它和其它类型相互赋值的时候，如果需要，它可以自动转换成其它类型；但是，如果试图这样转换所指类型为 void* 之外的类型的 void** 指针时，这个转换不能完成。

46：我有一个 char * 型指针刚好指向一些 int 型变量，我想跳过它们。 为什么((int *)p)++; 不行？

类型转换的实质“把这些二进制位看作另一种类型，并作相应的对待”；((int *)p)++是一个转换操作符，根据定义它只能生成一个右值(rvalue)。而右值既不能赋值，也不能用++ 自增。正确的做法:p = (char *)((int *)p + 1);

47：*p++ 自增 p 还是 p 所指向的变量？

p++ 和(p++) 等价。要自增 p 指向的值，使用(*p)++，或者++*p。

48：我想声明一个指针并为它分配一些空间，代码 char *p; *p = malloc(10) 的问题；

你所声明的指针是 p，而不是*p，当你操作指针本身时，你只需要使用指针的名字即可:p = malloc(10);

49：int i=7; printf(“%d\n”, i++ *i++);的值？

i++*i++=49

50：枚举和#define 有什么不同？1）：#define 是在预编译阶段进行简单替换。枚举常量则是在编译的时候确定其值。
2）：一般在编译器里，可以调试枚举常量，但是不能调试宏常量。
3）：枚举可以一次定义大量相关的常量，而#define 宏一次只能定义一个。
51：C++文件编译与执行的四个阶段

第一阶段：预处理阶段。根据文件中的预处理指令来修改源文件的内容。如#include 指令，作用是把头文件的内容添加到.cpp 文件中。

第二阶段：编译阶段，将其翻译成等价的中间代码或汇编代码。

第三阶段：汇编阶段，把汇编语言翻译成目标机器指令。

第四阶段：是链接，例如，某个源文件中的函数可能引用了另一个源文件中定义的某个函数；在程序中可能调用了某个库文件中的函数。

52：声明 struct x1 { . . . }; 和 typedef struct { . . . } x2; 有什么不同？

第一种形式声明了一个“结构标签”；

第二种声明了一个“类型定义”。

主要的区别是第一种方式定义结构体变量需要写“struct x1”而引用第一种，而第二种方式定义结构体变量不需要使用 struct 关键字。

53：以下的初始化有什么区别？

char a[] = “string literal”; char *p= “string literal”
用作数组初始值，它指明该数组中字符的初始值。

第二种情况会转化为一个无名的静态字符数组，可能会存储在只读内存中，这就是造成它不一定能被修改。第二个声明把 p 初始化成指向无名数组的第一个元素。为了编译旧代码，有的编译器有一个控制字符串是否可写的开关。

54：对于没有初始化的变量的初始值可以作怎样的假定？如果一个全局变量初始值为“零”，它可否作为空指针或浮点零？

对于具有“静态” 生存期的未初始化全局变量可以确保初始值为零，如果是指针会被初始化为正确的空指针，如果是浮点数会被初始化为 0.0 。
对于局部变量，如果没有显示地初始化，则包含的是垃圾内容。
用 malloc() 和 realloc() 动态分配的内存也可能包含垃圾数据，因此必须由调用者正确地初始化。

55：函数指针的定义是什么？

是一个指向函数的指针。看例子：
A)，char * (*fun1)(char * p1,char * p2);//fun1 不是函数名，而是一个指针变量，它指向一个函数。这个函数有两个指针类型的参数，函数的返回值也是一个指针。
B)，char * *fun2(char * p1,char * p2);//是个二级指针
C)，char * fun3(char * p1,char * p2);//函数的返回值为 char *类型

56：int *p = NULL 和*p = NULL 有什么区别？

int *p = NULL;//定义一个指针变量 p，其指向的内存里面保存的是 int 类型的数据；在定义变量 p 的同时把 p 的值设置为 0×00000000，而不是把*p 的值设置为 0×00000000

```
int *p;
*p = NULL;
```
给*p 赋值为 NULL，即给 p 指向的内存赋值为 NULL；但是由于 p 指向的内存可能是非法的，所以调试的时候编译器可
能会报告一个内存访问错误。

```
int i = 10;
int *p = &i;
*p = NULL;
```
在编译器上调试一下，我们发现 p 指向的内存由原来的 10 变为 0 了；而 p 本身的值， 即内存地址并没有改变。

57：介绍一下#error 预处理

#error 预处理指令的作用是，编译程序时，只要遇到#error 就会生成一个编译错误提示消息，并停止编译。其语法格式为：
```
#error error-message
```
注意，宏串 error-message 不用双引号包围。遇到#error 指令时，错误信息被显示，可能同时还显示编译程序作者预先定义的其他内容。

58：用变量 a 给出下面的定义

a) int a; //一个整型数

b) int *a; //一个指向整型数的指针

c) int **a; //一个指向指针的的指针，它指向的指针是指向一个整型数

d) int a[10]; //一个有 10 个整型数的数组

e) int *a[10]; //一个有 10 个指针的数组，该指针是指向一个整型数的

f) int (*a)[10]; //一个指向有 10 个整型数数组的指针

g) int (*a)(int); //一个指向函数的指针，该函数有一个整型参数并返回一个整型数

h) int (*a[10])(int); // 一个有 10 个指针的数组，该指针指向一个函数，该函数有一个整型参数并返回一个整型数

59：分别给出 BOOL，int，float，指针变量 与“零值”比较的 if 语句（假设变量名为 var）

BOOL 型变量：if(!var)

int 型变量： if(var==0)

float 型变量：

const float EPSINON = 0.00001;

if ((x >= - EPSINON) && (x <= EPSINON)

指针变量： if(var==NULL)

60：什么是预编译？何时需要预编译？预编译又称为预处理 ， 是做些代码文本的替换工作。处理 # 开头的指令 ， 比如拷贝 #include 包含的文件代码，#define 宏定义的替换 ， 条件编译等。
c 编译系统在对程序进行通常的编译之前，先进行预处理。 c 提供的预处理功能主要有以下三 种： 1 ）宏定义 2 ）文件包含 3 ）条件编译

61：内联函数与宏有什么区别

内联函数在编译时展开，宏在预编译时展开
在编译的时候内联函数可以直接被嵌入到目标代码中，而宏只是一个简单的文本替换
内联函数可以完成诸如类型检测、语句是否正确等编译功能，宏就不具备这样的功能
inline 函数是函数，宏不是函数。
62：iostream 与 iostream.h 的区别#include <iostream.h>非标准输入输出流
#include <iostream>标准输入输出流
有“.h”的就是非标准的，C 的标准库函数，无“.h”的，就要用到命令空间，是 C++的。**63：namespace 的使用**因为标准库非常的庞大，所程序员在选择的类的名称或函数名时就很有可能和标准库中的某个名字相同。所以为了避免这种情况所造成的名字冲突，就把标准库中的一切都被放在名字空间 std 中。
C＋＋标准程序库中的所有标识符都被定义于一个名为 std 的 namespace 中。
1、直接指定标识符。例如 std::ostream
2、使用 using 关键字。
using std::cout;

using std::endl;
3、最方便的就是使用 using namespace std;
64：堆与栈的区别（1）一个是静态的，一个是动态的，堆是静态的，由用户申请和释放，栈是动态的，保存程序的局部变量（2）申请后系统的响应不同
栈：只要栈的剩余空间大于申请空间，系统就为程序提供内存，否则将抛出栈溢出异常
堆：当系统收到程序申请时，先遍历操作系统中记录空闲内存地址的链表，寻找第一个大于所申请空间的堆结点，然后将该结点从空间结点链表中删除，并将该结点的空间分配给程序。
（3）申请大小限制的不同
栈：在 windows 下，栈的大小一般是 2M，如果申请的空间超过栈的剩余空间时，将提示 overflow。
堆：堆是向高地址扩展的数据结构，是不连续的内存区域。这是由于系统是用链表来存储的空闲内存地址的，自然是不连续的，而链表的遍历方向是由低地址向高地址。堆的大小受限于计算机系统中有效的虚拟内存。由此可见，堆获得的空间比较灵活，也比较大。 **65：含参数的宏与函数的优缺点**
宏： 优点：在预处理阶段完成，不占用编译时间，同时，省去了函数调用的开销，运行效率高
缺点：不进行类型检查，多次宏替换会导致代码体积变大，而且由于宏本质上是字符串替换，故可能会由于一些参数的副作用导致得出错误的结果。
函数：优点：有类型检查，比较安全。缺点：函数调用需要参数、返回地址等的入栈、出栈开销，效率没有带参数宏高
宏与内联函数的区别
内联函数和宏都是在程序出现的地方展开，内联函数不是通过函数调用实现的，是在调用该函数的程序处将它展开（在编译期间完成的）；宏同样是；
不同的是：内联函数可以在编译期间完成诸如类型检测，语句是否正确等编译功能；宏就不具有这样的功能，而且宏展开的时间和内联函数也是不同的（在运行期间展开）

66：多态的作用？
（1）可以隐藏实现的细节，使得代码模块化；方便扩展代码；

（2）可以实现接口重用。

67：类的静态成员和非静态成员有何区别？
静态成员则是属于这个类的非静态成员是属于每个对象的

68：C++纯虚函数,虚函数，虚函数的实现,什么是虚指针？

纯虚函数是在基类中声明的虚函数，它在基类中没有定义，但要求任何派生类都要定义自己的实现方法。
virtual void f()=0;//是一个接口，子类必须实现这个接口虚指针或虚函数指针是虚函数的实现细节。带有虚函数的每一个对象都有一个虚指针指向该类的虚函数表。虚函数 ：虚函数是在基类中被声明为 virtual，并在派生类中重新

定义的成员函数，可实现成员函数的动态覆盖（Override）
纯虚函数和虚函数的区别是，纯虚函数子类必须实现。 纯虚函数的优点：
（1）可以实现多态特性
（2）定义一个标准的接口，在派生类中必须予以重写以实现多态性。
抽象类 ：包含纯虚函数的类称为抽象类。由于抽象类包含了没有定义的纯虚函数，所以不能定义抽象类的对象。

多态性可分为两类：静态多态和动态多态。函数重载和运算符重载实现的多态属于静态多态，动态多态性是通过虚函数实现的。

虚函数与构造函数，析构函数，成员函数的关系

为什么基类析构函数是虚函数？

编译器总是根据类型来调用类成员函数。但是一个派生类的指针可以安全地转化为一个基类的指针。这样删除一个基类的指针的时候，C++不管这个指针指向一个基类对象还是一个派生类的对象，调用的都是基类的析构函数而不是派生类的。如果你依赖于派生类的析构函数的代码来释放资源，而没有重载析构函数，那么会有资源泄漏。

为什么构造函数不能为虚函数

虚函数采用一种虚调用的方法。需调用是一种可以在只有部分信息的情况下工作的机制。如果创建一个对象，则需要知道对象的准确类型，因此构造函数不能为虚函数。

如果虚函数是有效的，那为什么不把所有函数设为虚函数？

不行。因为每个虚函数的对象都要维护一个虚函数表，因此在使用虚函数的时候都会产生一定的系统开销，这是没有必要的。

69：面向对象的三个基本特征，并简单叙述之？
1. 封装：将客观事物抽象成类，每个类对自身的数据和方法。封装可以使得代码模块化，目的是为了代码重用
2. 继承：子类继承父类的方法和属性，继承可以扩展已存在的代码，目的是为了代码重用
3. 多态：允许将子类类型的指针赋值给父类类型的指针。

70：什么是“&（引用）”？申明和使用“引用”要注意哪些问题？

引用就是某个目标变量的“别名”。注意事项：（1）申明一个引用的时候，必须要对其进行初始化。（2）初始化后，该引用名不能再作为其他变量名的别

名。（3）引用本身不占存储单元，系统不给引用分配存储单元。（4）返回引用时，在内存中不产生被返回值的副本

（5）不能返回局部变量的引用。主要原因是局部变量会在函数返回后被销毁.

71：引用与多态的关系？ 引用就是对象的别名。引用主要用作函数的形参。引用必须用与该引用同类型的对象初始化： 引用是除指针外另一个可以产生多态效果的手段。这意味着，一个基类的引用可以指向它的派生类实例。
int ival = 1024;int &refVal = ival; const 对象的引用只能是 const 类型的：const int ival = 1024;const int &refVal = ival; 多态是通过虚函数实现的。

72：指针和引用有什么区别；为什么传引用比传指针安全？如果我使用常量指针难道不行吗？
(1) 引用在创建的同时必须初始化，保证引用的对象是有效的，所以不存在ＮＵＬＬ引用；而指针在定义的时候不必初始化，所以，指针则可以是 NULL，可以在定义后面的任何地方重新赋值。
(2) 引用一旦被初始化为指向一个对象，它就不能被改变为另一个对象的引用；而指针在任何时候都可以改变为指向另一个对象.
(3) 引用的创建和销毁并不会调用类的拷贝构造函数

因为不存在空引用，并且引用一旦被初始化为指向一个对象，它就不能被改变为另一个对象的引用，所以比指针安全。
由于 const 指针仍然存在空指针，并且有可能产生野指针，所以还是不安全

73：参数传递有几种方式；实现多态参数传递采用什么方式，如果没有使用某种方式原因是什么？
传值，传指针或者引用

74：拷贝构造函数相关问题，深拷贝，浅拷贝，临时对象等。
深拷贝意味着拷贝了资源和指针，而浅拷贝只是拷贝了指针，没有拷贝资源
这样使得两个指针指向同一份资源，可能造成对同一份析构两次，程序崩溃。
而且浪费时间，并且不安全。
临时对象的开销比局部对象小些。

75：构造函数的特点

构造函数只在建立对象的时候自动被调用一次
构造函数必须是公共的，否则无法生成对象
构造函数只负责为自己的类构造对象

在构造函数中初始化变量
Person::Person( ) : name(“Jack”), age(30)
{
…
}

76：面向对象如何实现数据隐藏

定义类来实现数据隐藏：

成员函数和属性的类型：

私有成员 private
保护成员 protected
公共成员 public

77：字符指针、浮点数指针、以及函数指针这三种类型的变量哪个占用的内存最大？为什么？

所有指针变量占用内存单元的数量都是相同的。

78：C++是不是类型安全的？

不是。两个不同类型的指针之间可以强制转换.

79：const char*, char const*, char *const 的区别是什么？

把一个声明从右向左读，* 读成指向
char * const cp;//cp 是常指针，指向 char 类型的数据
const char * cp;//cp 是 char 类型的指针，指向 const char
char const * p;//C++里面没有 const*的运算符，所以 const 属于前面的类型。

80:什么是模板和宏？模板怎么实现？模板有什么缺点和优点？模版特化的概念，为什么特化？

标准库大量采用了模板技术。比如容器。

模板是一个蓝图，它本身不是类或函数。编译器用模板产生指定的类或函数的特定类型版本。模版的形参分为类型形参和非类型形参类型形参就是表示类型的形参，跟在关键字 typename 后非类型形参用来表示常量表达式

81：空指针和悬垂指针的区别？

空指针是指被赋值为 NULL 的指针；delete 指向动态分配对象的指针将会产生悬垂指针。

空指针可以被多次 delete，而悬垂指针再次删除时程序会变得非常不稳定；

使用空指针和悬垂指针都是非法的，而且有可能造成程序崩溃，如果指针是空指针，尽管同样是崩溃，但和悬垂指针相比是一种可预料的崩溃。

(a)指针数组和数组指针,函数指针和指针函数相关概念

指针数组：用于存储指针的数组

int* a[4]

元素表示：*a[i]

数组指针：指向数组的指针

int (*a)[4]

元素表示：(*a)[i]

指针函数：函数返回类型是某一类型的指针，int *f(x，y);

指针函数与函数指针表示方法的不同。最简单的辨别方式就是看函数名前面的指针*号有没有被括号（）包含，如果被包含就是函数指针，反之则是指针函数。

函数指针：是指向函数的指针变量，即本质是一个指针变量。

int (*f) (int x); /* 声明一个函数指针 */ 类型说明符 (*指针的变量名)(参数)

f=func; /* 将 func 函数的首地址赋给指针 f */

指向函数的指针包含了函数的地址

指针的指针： 例如：char ** cp;
如果有三个星号，那就是指针的指针的指针，依次类推。 指针的指针需要用到

指针的地址。
char c='A';
char *p=&c;
char **cp=&p;
通过指针的指针，不仅可以访问它指向的指针，还可以访问它指向的指针所指向的数据：
char *p1=*cp;
char c1=**cp; 指向指针数组的指针： char *Names[]={ Bill,Sam,0};
char **nm=Names;
while(*nm!=0) printf(%s\n,*nm++);
先用字符型指针数组 Names 的地址来初始化指针 nm。每次 printf()的调用都首先传递指针 nm 指向的字符型指针，然后对 nm 进行自增运算使其指向数组的下一个元素(还是指针)。

82：什么是智能指针？

当类中有指针成员时，一般有两种方式来管理指针成员：

(1)每个类对象都保留一份指针指向的对象的拷贝；

(2)使用智能指针，从而实现指针指向的对象的共享。实质是使用计数器与对象相关联，这样做可以保证对象正确的删除，避免垂悬指针。

每次创建类的新对象时，初始化指针并将引用计数置为 1；当对象作为另一对象的副本而创建时，拷贝构造函数拷贝指针并增加与之相应的引用计数；对一个对象进行赋值时，赋值操作符减少左操作数所指对象的引用计数，并增加右操作数所指对象的引用计数；调用析构函数时，构造函数减少引用计数。

83：C++空类默认有哪些成员函数？

默认构造函数、析构函数、复制构造函数、赋值函数

84：哪一种成员变量可以在一个类的实例之间共享？

答：static 静态成员变量

85：什么是多态？多态有什么作用？如何实现的？多态的缺点？

多态就是一个接口，多种方法。所以说，多态的目的则是为了实现接口重用。也就是说，不论传递过来的究竟是那个类的对象，函数都能够通过同一个接口调用到适应各自对象的实现方法。

C++的多态性是通过虚函数来实现的，虚函数允许子类重新定义成员函数，而子类重新定义父类的做法称为覆盖(override)，或重写。而重载则是允许有多个同名的函数，而这些函数的参数列表不同，允许参数个数不同，参数类型不同。编译器会根据函数列表的不同，而生成一些不同名称的预处理函数，来实现同名函数的重载。但这并没有体现多态性。

多态与非多态的实质区别就是函数的地址是运行时确定还是编译时确定。如果函数的调用在编译器编译期间就可以确定函数的调用地址，并生产代码，是静态的。而如果函数调用的地址在运行时才确定，就是动态的。

最常见的用法就是声明基类的指针，利用该指针指向任意一个子类对象，调用相应的虚函数，可以根据指向的子类的不同而实现不同的方法。如果没有使用虚函数的话，即没有利用 C++多态性，则利用基类指针调用相应的函数的时候，将总被限制在基类函数本身，而无法调用到子类中被重写过的函数

86：虚函数表解析和内存布局

虚函数表

虚函数是通过一张虚函数表来实现的。就像一个地图一样，指明了实际所应该调用的函数的地址。

这里我们着重看一下这张虚函数表。C++的编译器保证了虚函数表的指针存在于对象实例中最前面的位置（为了性能）。因此我们可以通过对象实例的地址得到这张虚函数表，然后通过遍历其中函数指针，并调用相应的函数。

为什么可以由父类的指针调用子类的对象的虚函数：

```
Derive d;//Derive 是 Base 的子类

Base *b1 = &d;//这必须使用父类的指针？？？

b1->f(); //Derive::f()
```

87：公有继承、受保护继承、私有继承

1）公有继承时，派生类对象可以访问基类中的公有成员，派生类的成员函数可以访问基类中的公有和受保护成员；公有继承时基类受保护的成员，可以通过派生类对象访问但不能修改。

2）私有继承时，基类的成员只能被直接派生类的成员访问，无法再往下继承；

3）受保护继承时，基类的成员也只被直接派生类的成员访问，无法再往下继承。

88：有哪几种情况只能用构造函数初始化列表而不能用赋值初始化？

答：const 成员，引用成员

89：C++如何阻止一个类被实例化？一般在什么时候将构造函数声明为 private？

1）将类定义为抽象基类或者将构造函数声明为 private；

2）不允许类外部创建类对象，只能在类内部创建对象

90：类使用 static 成员的优点，如何访问？

（1）static 成员的名字是在类的作用域中，因此可以避免与其他类的成员或全局对象名字冲突；

（2）可以实施封装。static 成员可以是私有成员，而全局对象不可以；

（3） static 成员是与特定类关联的，可清晰地显示程序员的意图。

91：static 数据成员和 static 成员函数

（1）static 数据成员：
static 数据成员独立于该类的任意对象而存在；static 数据成员（const static 数据成员除外）在类定义体内声明，必须在类外进行初始化。不像普通数据成员，static 成员不能在类的定义体中初始化，只能在定义时才初始化。static 数据成员定义放在 cpp 文件中，不能放在初始化列表中。Const static 成员可就地初始化。

变量定义：用于为变量分配存储空间，还可为变量指定初始值。程序中，变量有且仅有一个定义。

变量声明：用于向程序表明变量的类型和名字。

（2）static 成员函数： 在类的外部定义，Static 成员函数没有 this 形参，它可以直接访问所属类的 static 成员，不能直接使用非 static 成员。因为 static 成员不是任何对象的组成部分，所以 static 成员函数不能被声明为 const。同时，static 成员函数也不能被声明为虚函数。

92：C++的内部连接和外部连接

编译单元：当编译 cpp 文件时，预处理器首先递归包含头文件，形成一个编译单元。这个编译单元会被编译成为一个与 cpp 文件名同名的目标文件(.o 或是.obj)。连接程序把不同编译单元中产生的符号联系起来，构成一个可执行程序。

内部连接：如果一个名称对于它的编译单元来说是局部的，并且在连接时不会与其它编译单元中的同样的名称相冲突，那么这个名称有内部连接：

a)所有的声明

b)名字空间(包括全局名字空间)中的静态自由函数、静态友元函数、静态变量的定义

c)enum 定义

d)inline 函数定义(包括自由函数和非自由函数)

e)类的定义

f)名字空间中 const 常量定义

g)union 的定义

外部连接:在一个多文件程序中，如果一个名称在连接时可以和其它编译单元交互，那么这个名称就有外部连接。

以下情况有外部连接:

a)类非 inline 函数总有外部连接。包括类成员函数和类静态成员函数

b)类静态成员变量总有外部连接。

c)名字空间(包括全局名字空间)中非静态自由函数、非静态友元函数及非静态变量

93：变量的分类，全局变量和局部变量有什么区别？实怎么实现的？操作系统和编译器是怎么知道的？static 全局变量与普通的全局变量有什么区别？static 局部变量和普通局部变量有什么区别？static 函数与普通函数有什么区别？

1）变量可以分为：全局变量、局部变量、静态全局变量、静态局部变量
全局变量在整个工程文件内都有效；静态全局变量只在定义它的文件内有效局部变量在定义它的函数内有效，这个函数返回会后失效。
静态局部变量只在定义它的函数内有效，只是程序仅分配一次内存，函数返回后，该变量不会消失，直到程序运行结束后才释放；全局变量和静态变量如果没有手工初始化，则由编译器初始化为 0。局部变量的值不可知。
静态全局变量是定义存储类型为静态型的外部变量,其作用域是从定义点到程序结束,所不同的是存储类型决定了存储地点,静态型变量是存放在内存的数据区中的,它们在程序开始运行前就分配了固定的字节,在程序运行过程中被分配的字节大小是不改变的.只有程序运行结束后,才释放所占用的内存.

变量的作用域：
形参变量只在被调用期间才分配内存单元，调用结束立即释放。 这一点表明形参变量只有在函数内才是有效的， 离开该函数就不能再使用了。局部变量也称为内部变量。其作用域仅限于函数内， 离开该函数后再使用这种变量是非法的。
全局变量也称为外部变量，它不属于哪一个函数，它属于一个源程序文件。其作用域是整个源程序。在函数中使用全局变量，一般应作全局变量说明。 只有在函数内经过说明的全局变量才能使用。全局变量的说明符为 extern。 但在一个函数之前定义的全局变量，在该函数内使用可不再加以说明。 对于全局变量还有以下几点说明：
外部变量可加强函数模块之间的数据联系， 但是又使函数要依赖这些变量，因而使得函数的独立性降低。从模块化程序设计的观点来看这是不利的， 因此在不必要时尽量不要使用全局变量。
在同一源文件中，允许全局变量和局部变量同名。在局部变量的作用域内，全局变量不起作用。

变量的存储方式可分为“静态存储”和“动态存储”两种。
静态存储变量通常是在变量定义时就分定存储单元并一直保持不变， 直至整个程序结束。动态存储变量是在程序执行过程中，使用它时才分配存储单元， 使用完毕立即释放。 如果一个函数被多次调用，则反复地分配、 释放形参变量的存储单元。从以上分析可知， 静态存储变量是一直存在的， 而动态存储变量则时而存在时而消失。我们又把这种由于变量存储方式不同而产生的特性称变量的生存期。 生存期表示了变量存在的时间。 生存期和作用域是从时间和空间这两个不同的角度来描述变量的特性，这两者既有联系，又有区别。 一个变量究竟属于哪一种存储方式， 并不能仅从其作用域来判断，还应有明确的存储类型说明。

从作用域看：

全局变量具有全局作用域。全局变量只需在一个源文件中定义，就可以作用于所有的源文件。当然，其他不包含全局变量定义的源文件需要用 extern 关键字再次声明这个全局变量。
静态局部变量具有局部作用域，它只被初始化一次，自从第一次被初始化直到程序运行结束都一直存在，它和全局变量的区别在于全局变量对所有的函数都是可见的，而静态局部变量只对定义自己的函数体始终可见。
局部变量也只有局部作用域，它是自动对象（auto），它在程序运行期间不是一直存在，而是只在函数执行期间存在，函数的一次调用执行结束后，变量被撤销，其所占用的内存也被收回。
静态全局变量也具有全局作用域，它与全局变量的区别在于如果程序包含多个文件的话，它作用于定义它的文件里，不能作用到其它文件里，即被 static 关键字修饰过的变量具有文件作用域。这样即使两个不同的源文件都定义了相同名字的静态全局变量，它们也是不同的变量。

从分配内存空间看：
全局变量，静态局部变量，静态全局变量都在静态存储区分配空间，而局部变量在栈里分配空间。
全局变量本身就是静态存储方式，静态全局变量当然也是静态存储方式。这两者在存储方式上并无不同。这两者的区别虽在于非静态全局变量的作用域是整个源程序，当一个源程序由多个源文件组成时，非静态的全局变量在各个源文件中都是有效的。而静态全局变量则限制了其作用域，即只在定义该变量的源文件内有效，在同一源程序的其它源文件中不能使用它。由于静态全局变量的作用域局限于一个源文件内，只能为该源文件内的函数公用，因此可以避免在其它源文件中引起错误。
1)、静态变量会被放在程序的静态数据存储区(全局可见)中，这样可以在下一次调用的时候还可以保持原来的赋值。这一点是它与堆栈变量和堆变量的区别。
2)、变量用 static 告知编译器，自己仅仅在变量的作用范围内可见。这一点是它与全局变量的区别。

程序的局部变量存在于（堆栈）中，全局变量存在于（静态区 ）中，动态申请数据存在于（ 堆）中。

94：应用程序在运行时的内存包括代码区和数据区，其中数据区又包括哪些部分？

对于一个进程的内存空间而言，可以在逻辑上分成 3 个部份：代码区，静态数据区和动态数据区。

动态数据区一般就是“堆栈”。 栈是一种线性结构，堆是一种链式结构。进程的每个线程都有私有的“栈”。

全局变量和静态变量分配在静态数据区，本地变量分配在动态数据区，即堆栈中。程序通过堆栈的基地址和偏移量来访问本地变量。

95：C++里面是不是所有的动作都是 main()引起的？如果不是，请举例。

比如全局变量的初始化，就不是由 main 函数引起的：

```
class A{};

A a; //a 的构造函数限执行

int main() {}
```

96：异常框架

异常存在于程序的正常功能之外，并要求程序立即处理。 C++ 的异常处理包括： 1. throw 表达式，错误检测部分使用这种表达式来说明遇到了不可处理的错误。2. try 块，错误处理部分使用它来处理异常。try 语句块以 try 关键字开 始，并以一个或多个 catch 子句结束。在 try 块中执行的代码所抛出（throw）的异常，通常会被其中一个 catch 子句处理。3. 由标准库定义的一组异常类，用来在 throw 和相应的 catch 之间传递有关的错误信息。 throw 表达式：if (!item1.same_isbn(item2))throw runtime_error("Data must refer to same ISBN"); try 块：try {program-statements} catch (exception-specifier) {handler-statements} catch (exception-specifier) {handler-statements} 函数在寻找处理代码的过程中退出在复杂的系统中，程序的执行路径也许在遇到抛出异常的代码之前，就已经经过了多个 try 块。抛出一个异常时，首先要搜索 的是抛出异常的函数。如果没有找到匹配的 catch，则终止这个函数的执行，并在调用这个函数的函数中寻找相配的 catch。如果仍然没有找到相应的处理代码，该函数同样要终止，搜索调用它的函数。直到找到适当类型的 catch 为止。