1. 面向对象的程序设计思想是什么?
答：把数据结构和对数据结构进行操作的方法封装形成一个个的对象。

2. 什么是类?
答：把一些具有共性的对象归类后形成一个集合，也就是所谓的类。

3. 对象都具有的二方面特征是什么？分别是什么含义?
答：对象都具有的特征是：静态特征和动态特征。
静态特征是指能描述对象的一些属性，动态特征是指对象表现出来的行为

4. 在头文件中进行类的声明，在对应的实现文件中进行类的定义有什么意义?
答：这样可以提高编译效率，因为分开的话只需要编译一次生成对应的.obj 文件后，再次应用该类的地方，这个类就不会被再次编译，从而大大提高了效率。

5. 在类的内部定义成员函数的函数体，这种函数会具备那种属性?
答：这种函数会自动为内联函数，这种函数在函数调用的地方在编译阶段都会进行代码替换。

6. 成员函数通过什么来区分不同对象的成员数据？为什么它能够区分?
答：通过 this 指针来区分的， 因为它指向的是对象的首地址。

7. C++编译器自动为类产生的四个缺省函数是什么?
答：默认构造函数，拷贝构造函数，析构函数，赋值函数。

8. 拷贝构造函数在哪几种情况下会被调用?
答：1.当类的一个对象去初始化该类的另一个对象时；
2.如果函数的形参是类的对象，调用函数进行形参和实参结合时；
3.如果函数的返回值是类对象，函数调用完成返回时。

9. 构造函数与普通函数相比在形式上有什么不同？（构造函数的作用，它的声明形式来分析）
答：构造函数是类的一种特殊成员函数，一般情况下，它是专门用来初始化对象成员变量的。
构造函数的名字必须与类名相同，它不具有任何类型，不返回任何值。

10. 什么时候必须重写拷贝构造函数?
答：当构造函数涉及到动态存储分配空间时，要自己写拷贝构造函数，并且要深拷贝。

11. 构造函数的调用顺序是什么?
答：1.先调用基类构造函数
2.按声明顺序初始化数据成员
3.最后调用自己的构造函数。

12. 哪几种情况必须用到初始化成员列表?

答：类的成员是常量成员初始化；
类的成员是对象成员初始化，而该对象没有无参构造函数。
类的成员为引用时。

13. 什么是常对象?
答：常对象是指在任何场合都不能对其成员的值进行修改的对象。

14. 静态函数存在的意义?
答：静态私有成员在类外不能被访问，可通过类的静态成员函数来访问；
当类的构造函数是私有的时，不像普通类那样实例化自己，只能通过静态成员函数来调用构造函数。

15. 在类外有什么办法可以访问类的非公有成员?
答：友元，继承，公有成员函数。

16. 什么叫抽象类?
答：不用来定义对象而只作为一种基本类型用作继承的类。

17. 运算符重载的意义?
答：为了对用户自定义数据类型的数据的操作与内定义的数据类型的数据的操作形式一致。

18. 不允许重载的 5 个运算符是哪些?
答：
1. .*（成员指针访问运算符号）
2. ：：域运算符
3. Sizeof 长度运算符号
4. ？：条件运算符号
5. .（成员访问符）

19. 运算符重载的三种方式?
答：普通函数，友元函数，类成员函数。

20. 流运算符为什么不能通过类的成员函数重载？一般怎么解决?
答：因为通过类的成员函数重载必须是运算符的第一个是自己，而对流运算的重载要求第一个参数是流对象。一般通过友元来解决。

21. 赋值运算符和拷贝构造函数的区别与联系?
答：相同点：都是将一个对象 copy 到另一个中去。
不同点：拷贝构造函数涉及到要新建立一个对象。

22. 在哪种情况下要调用该类的析构函数?
答：对象生命周期结束时。

23. 对象间是怎样实现数据的共享的?

答：通过类的静态成员变量来实现的。静态成员变量占有自己独立的空间不为某个对象所私有。

24. 友元关系有什么特性?
答：单向的，非传递的，不能继承的。

25. 对对象成员进行初始化的次序是什么?
答：它的次序完全不受它们在初始化表中次序的影响，只有成员对象在类中声明的次序来决定的。

26. 类和对象之间的关系是什么?
答：类是对象的抽象，对象是类的实例。

27. 对类的成员的访问属性有什么?
答：public，protected，private。

28. const char *p,char * const p;的区别
如果 const 位于星号的左侧，则 const 就是用来修饰指针所指向的变量，即指针指向为常量;
如果 const 位于星号的右侧，const 就是修饰指针本身，即指针本身是常量。

29. 是不是一个父类写了一个 virtual 函数，如果子类覆盖它的函数不加 virtual ,也能实现多态?
virtual 修饰符会被隐形继承的。
virtual 可加可不加,子类覆盖它的函数不加 virtual ,也能实现多态。

30. 函数重载是什么意思？它与虚函数的概念有什么区别?
函数重载是一个同名函数完成不同的功能，编译系统在编译阶段通过函数参数个数、参数类型不同，函数的返回值来区分该调用哪一个函数，即实现的是静态的多态性。但是记住：不能仅仅通过函数返回值不同来实现函数重载。而虚函数实现的是在基类中通过使用关键字 virtual 来申明一个函数为虚函数，含义就是该函数的功能可能在将来的派生类中定义或者在基类的基础之上进行扩展，系统只能在运行阶段才能动态决定该调用哪一个函数，所以实现的是动态的多态性。它体现的是一个纵向的概念，也即在基类和派生类间实现。

31. 构造函数和析构函数是否可以被重载,为什么?
答：构造函数可以被重载，析构函数不可以被重载。因为构造函数可以有多个且可以带参数，而析构函数只能有一个，且不能带参数。

32. 如何定义和实现一个类的成员函数为回调函数?
答：
所谓的回调函数，就是预先在系统的对函数进行注册，让系统知道这个函数的存在，以后当某个事件发生时，再调用这个函数对事件进行响应。
定义一个类的成员函数时在该函数前加 CALLBACK 即将其定义为回调函数，函数的实现和普通成员函数没有区别

33. 虚函数是怎么实现的?
答：简单说来使用了虚函数表.

34. 抽象类不会产生实例，所以不需要有构造函数。 错

35. 从一个模板类可以派生新的模板类，也可以派生非模板类。 对

36. main 函数执行以前，还会执行什么代码?
答案：全局对象的构造函数会在 main 函数之前执行。

37. 当一个类 A 中没有生命任何成员变量与成员函数,这时 sizeof(A)的值是多少，如果不是零，请解释一下编译器为什么没有让它为零。（Autodesk）
答案：肯定不是零。举个反例，如果是零的话，声明一个 class A[10]对象数组，而每一个对象占用的空间是零，这时就没办法区分 A[0],A[1]…了。

38. delete 与 delete []区别：
delete 只会调用一次析构函数，而 delete[]会调用每一个成员的析构函数。

39. 子类析构时要调用父类的析构函数吗?
会调用，
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

40. .继承优缺点。
1、类继承是在编译时刻静态定义的，且可直接使用，
2、类继承可以较方便地改变父类的实现。
缺点：
1、因为继承在编译时刻就定义了，所以无法在运行时刻改变从父类继承的实现
2、父类通常至少定义了子类的部分行为，父类的任何改变都可能影响子类的行为
3、如果继承下来的实现不适合解决新的问题，则父类必须重写或被其他更适合的类替换。这种依赖关系限制了灵活性并最终限制了复用性。

41. 解释堆和栈的区别。
栈区（stack）— 由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。
堆:一般由程序员分配释放， 若程序员不释放，程序结束时可能由 OS 回收 。

42. 一个类的构造函数和析构函数什么时候被调用,是否需要手工调用?
答：构造函数在创建类对象的时候被自动调用，析构函数在类对象生命期结束时，由系统自动调用。

43. 何时需要预编译：
总是使用不经常改动的大型代码体。
程序由多个模块组成，所有模块都使用一组标准的包含文件和相同的编译选项。在这种情况

下，可以将所有包含文件预编译为一个预编译头。

44. 多态的作用?
主要是两个：
1. 隐藏实现细节，使得代码能够模块化；扩展代码模块，实现代码重用；
2. 接口重用：为了类在继承和派生的时候，保证使用家族中任一类的实例的某一属性时的正确调用

45. 虚拟函数与普通成员函数的区别？内联函数和构造函数能否为虚拟函数?
答案：区别：虚拟函数有 virtual 关键字，有虚拟指针和虚函数表，虚拟指针就是虚拟函数的接口，而普通成员函数没有。内联函数和构造函数不能为虚拟函数。

46. 构造函数和析构函数的调用顺序? 析构函数为什么要虚拟?
答案：构造函数的调用顺序：基类构造函数—对象成员构造函数—派生类构造函数；析构函数的调用顺序与构造函数相反。析构函数虚拟是为了防止析构不彻底，造成内存的泄漏。

47. .C++中类型为 private 的成员变量可以由哪些函数访问?
只可以由本类中的成员函数和友员函数访问

48. 请说出类中 private，protect，public 三种访问限制类型的区别
private 是私有类型，只有本类中的成员函数访问;protect 是保护型的，本类和继承类可以访问;public 是公有类型，任何类都可以访问.

49. 类中成员变量怎么进行初始化?
可以通过构造函数的初始化列表或构造函数的函数体实现。

50. 在什么时候需要使用“常引用”？
如果既要利用引用提高程序的效率，又要保护传递给函数的数据不在函数中被改变，就应使用常引用。

51. 引用与指针有什么区别?
答 、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。

52. 描述实时系统的基本特性
53. 答 、在特定时间内完成特定的任务，实时性与可靠性。

54. 全局变量和局部变量在内存中是否有区别？如果有，是什么区别?
答 、全局变量储存在静态数据区，局部变量在堆栈中。

55. 堆栈溢出一般是由什么原因导致的?
答 、没有回收垃圾资源

56. 什么函数不能声明为虚函数?
答 构造函数（constructor）

57. .IP 地址的编码分为哪俩部分?
答 IP 地址由两部分组成，网络号和主机号。

58. .不能做 switch()的参数类型是：
答 、switch 的参数不能为实型。

59. 如何引用一个已经定义过的全局变量?
答 、可以用引用头文件的方式，也可以用 extern 关键字，如果用引用头文件方式来引用某个在头文件中声明的全局变理，假定你将那个变写错了，那么在编译期间会报错，如果你用 extern 方式引用时，假定你犯了同样的错误，那么在编译期间不会报错，而在连接期间报错

60. 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?
答 、c 用宏定义，c++用 inline

61. C++是不是类型安全的?
答案：不是。两个不同类型的指针之间可以强制转换（用 reinterpret cast)

62. 当一个类 A 中没有生成任何成员变量与成员函数,这时 sizeof(A)的值是多少，请解释一下编译器为什么没有让它为零。
答案：为 1。举个反例，如果是零的话，声明一个 class A[10]对象数组，而每一个对象占用的空间是零，这时就没办法区分 A[0],A[1]…了。

63. 简述数组与指针的区别?
数组要么在静态存储区被创建（如全局数组），要么在栈上被创建。指针可以随时指向任意类型的内存块。
(1)修改内容上的区别
char a[] = “hello”;
a[0] = ‘X’;
char *p = “world”; // 注意 p 指向常量字符串
p[0] = ‘X’; // 编译器不能发现该错误，运行时错误
(2) 用运算符 sizeof 可以计算出数组的容量（字节数）。sizeof(p),p 为指针得到的是一个指针变量的字节数，而不是 p 所指的内存容量。

64. C++函数中值的传递方式
有三种方式：值传递、指针传递、引用传递

65. 内存的分配方式
分配方式有三种，
1、静态存储区，是在程序编译时就已经分配好的，在整个运行期间都存在，如全局变量、常量。

2、 栈上分配，函数内的局部变量就是从这分配的，但分配的内存容易有限。
3、堆上分配，也称动态分配，如我们用 new,malloc 分配内存，用 delete,free 来释放的内存。

66. extern“C”有什么作用?
Extern “C”是由C＋＋提供的一个连接交换指定符号，用于告诉C＋＋这段代码是C函数。这是因为C++编译后库中函数名会变得很长，与C生成的不一致，造成C＋＋不能直接调用C函数，加上 extren “c”后，C++就能直接调用C函数了。

67. 用什么函数开启新进程、线程。
答案：
线程：CreateThread/AfxBeginThread 等
进程：CreateProcess 等

68. SendMessage 和 PostMessage 有什么区别
答案：SendMessage 是阻塞的，等消息被处理后，代码才能走到 SendMessage 的下一行。PostMessage 是非阻塞的，不管消息是否已被处理，代码马上走到 PostMessage 的下一行。

69. CMemoryState 主要功能是什么
答案：查看内存使用情况，解决内存泄露问题。

70. 26、#include <filename.h> 和 #include “filename.h” 有什么区别?
答：对于#include <filename.h> ，编译器从标准库路径开始搜索 filename.h
对于#include “filename.h” ，编译器从用户的工作路径开始搜索 filename.h

71. 处理器标识#error 的目的是什么?
答：编译时输出一条错误信息，并中止编译。

72. #if!defined(AFX_..._HADE_H)
#define(AFX_..._HADE_H)
……
#endif 作用?
防止该头文件被重复引用。

73. 在定义一个宏的时候要注意什么?
定义部分的每个形参和整个表达式都必须用括号括起来，以避免不可预料的错误发生

74. 数组在做函数实参的时候会转变为什么类型?
数组在做实参时会变成指针类型。

75. 系统会自动打开和关闭的 3 个标准的文件是?
(1) 标准输入----键盘---stdin
(2) 标准输出----显示器---stdout

(3) 标准出错输出----显示器---stderr

76. .在 Win32 下 char, int, float, double 各占多少位?
(1) Char 占用 8 位
(2) Int 占用 32 位
(3) Float 占用 32 位
(4) Double 占用 64 位

77. strcpy()和 memcpy()的区别?
strcpy()和 memcpy()都可以用来拷贝字符串，strcpy()拷贝以’\0’结束，但 memcpy()必须指定拷贝的长度。

78. 说明 define 和 const 在语法和含义上有什么不同?
(1) #define 是 C 语法中定义符号常量的方法，符号常量只是用来表达一个值，在编译阶段符号就被值替换了，它没有类型；
(2) Const 是 C++语法中定义常变量的方法，常变量具有变量特性，它具有类型，内存中存在以它命名的存储单元，可以用 sizeof 测出长度。

79. 说出字符常量和字符串常量的区别，并使用运算符 sizeof 计算有什么不用?
字符常量是指单个字符，字符串常量以‘\0’结束，使用运算符 sizeof 计算多占一字节的存储空间。

80. 简述全局变量的优缺点?
全局变量也称为外部变量，它是在函数外部定义的变量，它属于一个源程序文件，它保存上一次被修改后的值，便于数据共享，但不方便管理，易引起意想不到的错误。

81. 总结 static 的应用和作用?
（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。

82. .总结 const 的应用和作用?
（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”。

83. 什么是指针？谈谈你对指针的理解?
指针是一个变量，该变量专门存放内存地址；
指针变量的类型取决于其指向的数据类型，在所指数据类型前加*
指针变量的特点是它可以访问所指向的内存。

84. 什么是常指针，什么是指向常变量的指针?
常指针的含义是该指针所指向的地址不能变，但该地址所指向的内容可以变化，使用常指针可以保证我们的指针不能指向其它的变量，
指向常变量的指针是指该指针的变量本身的地址可以变化，可以指向其它的变量，但是它所指的内容不可以被修改。指向长变量的指针定义，

85. 函数指针和指针函数的区别?
函数指针是指指向一个函数入口的指针；
指针函数是指函数的返回值是一个指针类型。

87. 简述 Debug 版本和 Release 版本的区别?
Debug 版本是调试版本，Release 版本是发布给用户的最终非调试的版本，

88. 指针的几种典型应用情况?
int *p[n];-----指针数组，每个元素均为指向整型数据的指针。
int (*)p[n];------p 为指向一维数组的指针，这个一维数组有 n 个整型数据。
int *p();----------函数带回指针，指针指向返回的值。
int (*)p();------p 为指向函数的指针。

89. static 函数与普通函数有什么区别?
static 函数在内存中只有一份，普通函数在每个被调用中维持一份拷贝

90. struct(结构) 和 union(联合)的区别?
1. 结构和联合都是由多个不同的数据类型成员组成, 但在任何同一时刻, 联合中只存放了一个被选中的成员（所有成员共用一块地址空间）, 而结构的所有成员都存在（不同成员的存放地址不同）。
2. 对于联合的不同成员赋值, 将会对其它成员重写, 原来成员的值就不存在了, 而对于结构的不同成员赋值是互不影响的。

91. class 和 struct 的区别?
struct 的成员默认是公有的，而类的成员默认是私有的。

92. 简述枚举类型?
枚举方便一次定义一组常量，使用起来很方便；

93. assert()的作用?
ASSERT()是一个调试程序时经常使用的宏，在程序运行时它计算括号内的表达式，如果表达式为 FALSE (0), 程序将报告错误，并终止执行。如果表达式不为 0，则继续执行后面的语句。这个宏通常原来判断程序中是否出现了明显非法的数据，如果出现了终止程序以免导致严重后果，同时也便于查找错误。

94. 局部变量和全局变量是否可以同名?
能，局部会屏蔽全局。要用全局变量，需要使用"::"(域运算符)。

95. 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。

96. 在什么时候使用常引用?
如果既要利用引用提高程序的效率，又要保护传递给函数的数据不在函数中被改变，就应使用常引用。

97. 类的声明和实现的分开的好处?
1. 起保护作用；
2. 提高编译的效率。

98. windows 消息系统由哪几部分构成?
由一下 3 部分组成：
1. 消息队列：操作系统负责为进程维护一个消息队列，程序运行时不断从该消息队列中获取消息、处理消息；
2. 消息循环：应用程序通过消息循环不断获取消息、处理消息。
3. 消息处理：消息循环负责将消息派发到相关的窗口上使用关联的窗口过程函数进行处理。

99. 什么是消息映射?
消息映射就是让程序员指定 MFC 类（有消息处理能力的类）处理某个消息。然后由程序员完成对该处理函数的编写，以实现消息处理功能。

100. 什么是 UDP 和 TCP 的区别是什么?
TCP 的全称为传输控制协议。这种协议可以提供面向连接的、可靠的、点到点的通信。
UDP 全称为用户报文协议，它可以提供非连接的不可靠的点到多点的通信。
用 TCP 还是 UDP，那要看你的程序注重哪一个方面？可靠还是快速?

101. winsock 建立连接的主要实现步骤?
答：
服务器端：socket()建立套接字，绑定（bind）并监听（listen），用 accept（）等待客户端连接, accept（）发现有客户端连接，建立一个新的套接字，自身重新开始等待连接。该新产生的套接字使用 send() 和 recv（）写读数据，直至数据交换完毕，closesocket()关闭套接字。

客户端：socket()建立套接字，连接（connect）服务器，连接上后使用 send()和 recv（），在套接字上写读数据，直至数据交换完毕，closesocket()关闭套接字。

102. 进程间主要的通讯方式?
信号量，管道，消息，共享内存

103. 构成 Win32 API 函数的三个动态链接库是什么?
答：内核库，用户界面管理库，图形设备界面库。

104. 创建一个窗口的步骤是?
答：填充一个窗口类结构->注册这个窗口类->然后再创建窗口->显示窗口->更新窗口。

105. 模态对话框和非模态对话框有什么区别?
答：1.调用规则不同：前者是用 DoModal()调用，后者通过属性和 ShowWindow()来显示。
2. 模态对话框在没有关闭前用户不能进行其他操作，而非模态对话框可以。
3. 非模态对话框创建时必须编写自己的公有构造函数，还要调用 Create()函数。

106. 从 EDIT 框中取出数据给关联的变量，已经把关联的变量的数据显示在 EDIT 框上的函数是什么?
答： UpdateData(TRUE)，Updatedata(FALSE).

107. 简单介绍 GDI?
答；GDI 是 Graphics Device Interface 的缩写，译为：图形设备接口；是一个在 Windows 应用程序中执行与设备无关的函数库，这些函数在不同的输出设备上产生图形以及文字输出。

108. windows 消息分为几类？并对各类做简单描述。
1.窗口消息：与窗口相关的消息，除 WM_COMMAND 之外的所有以 WM_开头的消息；
2.命令消息；用于处理用户请求，以 WM_COMMAND 表示的消息；
3.控件通知消息：统一由 WM_NOTIFT 表示，
4.用户自定义消息。

109. 如何自定义消息?
使用 WM_USER 和 WM_APP 两个宏来自定义消息，

110. 简述 Visual C++ 、Win32 API 和 MFC 之间的关系?
(1) Visual C+是一个以 C++程序设计语言为基础的、集成的、可视化的编程环境；
(2) Win32 API 是 32 位 Windows 操作系以 C/C++形式提供的一组应用程序接口；
(3) MFC 是对 Win32 API 的封装，简化了开发过程。

111.怎样消除多重继承中的二义性?
1. 成员限定符

2. 虚基类

112 什么叫静态关联，什么叫动态关联
在多态中，如果程序在编译阶段就能确定实际执行动作，则称静态关联，
如果等到程序运行才能确定叫动态关联。

113 多态的两个必要条件
1.一个基类的指针或引用指向一个派生类对象，
2.虚函数

114.什么叫智能指针?
当一个类中，存在一个指向另一个类对象的指针时，对指针运算符进行重载，那么当前类对象可以通过指针像调用自身成员一样调用另一个类的成员。

115.什么时候需要用虚析构函数?
当基类指针指向用 new 运算符生成的派生类对象时，delete 基类指针时，派生类部分没有释放掉而造成释放不彻底现象，需要虚析构函数。

116. MFC 中，大部分类是从哪个类继承而来?
CObject

117.什么是平衡二叉树?
答：左右子树都是平衡二叉树，而且左右子树的深度差值的约对值不大于 1

118.语句 for(； 1 ；)有什么问题？它是什么意思?
答：无限循环，和 while(1)相同。

119. 派生新类的过程要经历三个步骤
1 吸收基类成员
2.改造基类成员
3.添加新成员

121. TCP/IP 建立连接的过程
在 TCP/IP 协议中，TCP 协议提供可靠的连接服务，采用三次握手建立一个连接。
第一次握手：建立连接时，客户端发送连接请求到服务器，并进入 SYN_SEND 状态，等待服务器确认；
第二次握手：服务器收到客户端连接请求，向客户端发送允许连接应答，此时服务器进入 SYN_RECV 状态；
第三次握手：客户端收到服务器的允许连接应答，向服务器发送确认，客户端和服务器进入通信状态，完成三次握手

122. .memset ,memcpy 的区别
memset 用来对一段内存空间全部设置为某个字符，一般用在对定义的字符串进行初始化为'\0'。

memcpy 用来做内存拷贝，你可以拿它拷贝任何数据类型的对象，可以指定拷贝的数据长度；

123. 在 C++ 程序中调用被 C 编译器编译后的函数，为什么要加 extern “C”？
答：C++语言支持函数重载，C 语言不支持函数重载。函数被 C++编译后在库中的名字
与 C 语言的不同。假设某个函数的原型为： void foo(int x, int y);该函数被 C 编译器编译后在库中的名字为_foo ， 而 C++编译器则会产生像_foo_int_int 之类的名字。C++提供了 C 连接交换指定符号 extern“C”来解决名字匹配问题。

124 怎样定义一个纯虚函数？含有纯虚函数的类称为什么？
在虚函数的后面加=0，含有虚函数的类称为抽象类。

125. 已知 strcpy 函数的原型是：
char * strcpy(char * strDest,const char * strSrc);不调用库函数，实现 strcpy 函数。
答案：

```
char *strcpy(char *strDest, const char *strSrc)
{
if ( strDest == NULL || strSrc == NULL)
return NULL ;
if ( strDest == strSrc)
return strDest ;
char *tempptr = strDest ;
while( (*strDest++ = *strSrc++) != ‘\\0’)
;
return tempptr ;
}
```

126. 已知类 String 的原型为：

```
class String
{
public:
String(const char *str = NULL); // 普通构造函数
String(const String &other); // 拷贝构造函数
~ String(void); // 析构函数
String & operate =(const String &other); // 赋值函数
private:
char *m_data; // 用于保存字符串
};
```

请编写 String 的上述 4 个函数。
答案：

```
String::String(const char *str)
{
```

```
if ( str == NULL ) //strlen 在参数为 NULL 时会抛异常才会有这步判断
{
m_data = new char[1] ;
m_data[0] = '' ;
}
else
{
m_data = new char[strlen(str) + 1];
strcpy(m_data,str);
}
}
String::String(const String &other)
{
m_data = new char[strlen(other.m_data) + 1];
strcpy(m_data,other.m_data);
}
String & String::operator =(const String &other)
{
if ( this == &other)
return *this ;
delete []m_data;
m_data = new char[strlen(other.m_data) + 1];
strcpy(m_data,other.m_data);
return *this ;
}
String::~ String(void)
{
delete []m_data ;
}
```

127. 类成员函数的重载、覆盖和隐藏区别

答案：

成员函数被重载的特征：

（1）相同的范围（在同一个类中）；

（2）函数名字相同；

（3）参数不同；

（4）virtual 关键字可有可无。

覆盖是指派生类函数覆盖基类函数，特征是：

（1）不同的范围（分别位于派生类与基类）；

（2）函数名字相同；

（3）参数相同；

（4）基类函数必须有 virtual 关键字。

“隐藏”是指派生类的函数屏蔽了与其同名的基类函数，规则如下：

（1）如果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（注意别与重载混淆）。
（2）如果派生类的函数与基类的函数同名，并且参数也相同，但是基类函数没有 virtual 关键字。此时，基类的函数被隐藏（注意别与覆盖混淆）

128. 如何打印出当前源文件的文件名以及源文件的当前行号?
答案：
```
cout << __FILE__ ;
cout<<__LINE__ ;
```
__FILE__和__LINE__是系统预定义宏，这种宏并不是在某个文件中定义的，而是由编译器定义的。

129. 文件中有一组整数，要求排序后输出到另一个文件中
答案：
```
void Order(vector<int> &data) //起泡排序
{
int count = data.size() ;
int tag = false ;
for ( int i = 0 ; i < count ; i++)
{
for ( int j = 0 ; j < count - i - 1 ; j++)
{
if ( data[j] > data[j+1])
{
tag = true ;
int temp = data[j] ;
data[j] = data[j+1] ;
data[j+1] = temp ;
}
}
if ( !tag )
break ;
}
}
void main( void )
{
vector<int>data;
ifstream in("c:\\data.txt");
if ( !in)
{
cout<<"file error!";
exit(1);
}
int temp;
```

```
while (!in.eof())
{
in>>temp;
data.push_back(temp);
}
in.close();
Order(data);
ofstream out("c:\\result.txt");
if ( !out)
{
cout<<"file error!";
exit(1);
}
for ( i = 0 ; i < data.size() ; i++)
out<<data[i]<<" ";
out.close();
}
```

130. 一个链表的结点结构

```
struct Node
{
int data ;
Node *next ;
};
typedef struct Node Node ;
```

已知链表的头结点 head,写一个函数把这个链表逆序 (Intel)

```
Node * ReverseList(Node *head) //链表逆序
{
if ( head == NULL || head->next == NULL )
return head;
Node *p1 = head ;
Node *p2 = p1->next ;
Node *p3 = p2->next ;
p1->next = NULL ;
while ( p3 != NULL )
{
p2->next = p1 ;
p1 = p2 ;
p2 = p3 ;
p3 = p3->next ;
}
p2->next = p1 ;
head = p2 ;
```

```
return head ;
}
```
131. 一个链表的结点结构
```
struct Node
{
int data ;
Node *next ;
};
typedef struct Node Node ;
```
已知两个链表 head1 和 head2 各自有序，请把它们合并成一个链表依然有序。
```
Node * Merge(Node *head1 , Node *head2)
{
if ( head1 == NULL)
return head2 ;
if ( head2 == NULL)
return head1 ;
Node *head = NULL ;
Node *p1 = NULL;
Node *p2 = NULL;
if ( head1->data < head2->data )
{
head = head1 ;
p1 = head1->next;
p2 = head2 ;
}
else
{
head = head2 ;
p2 = head2->next ;
p1 = head1 ;
}
Node *pcurrent = head ;
while ( p1 != NULL && p2 != NULL)
{
if ( p1->data <= p2->data )
{
pcurrent->next = p1 ;
pcurrent = p1 ;
p1 = p1->next ;
}
else
{
pcurrent->next = p2 ;
pcurrent = p2 ;
```

```
p2 = p2->next ;
}
}
if ( p1 != NULL )
pcurrent->next = p1 ;
if ( p2 != NULL )
pcurrent->next = p2 ;
return head ;
}
```

132.已知两个链表 head1 和 head2 各自有序，请把它们合并成一个链表依然有序，这次要求用递归方法进行。 (Autodesk)
答案：

```
Node * MergeRecursive(Node *head1 , Node *head2)
{
if ( head1 == NULL )
return head2 ;
if ( head2 == NULL)
return head1 ;
Node *head = NULL ;
if ( head1->data < head2->data )
{
head = head1 ;
head->next = MergeRecursive(head1->next,head2);
}
else
{
head = head2 ;
head->next = MergeRecursive(head1,head2->next);
}
return head ;
}
```

133. 分析一下这段程序的输出 (Autodesk)

```
class B
{
public:
B()
{
cout<<"default constructor"<<endl;
}
~B()
{
cout<<"destructed"<<endl;
```

```
}
B(int i):data(i)
{
cout<<"constructed by parameter" << data <<endl;
}
private:
int data;
};
B Play( B b)
{
return b ;
}
int main(int argc, char* argv[])
{
B temp = Play(5);
return 0;
}
```

133 将“引用”作为函数参数有哪些特点?

（1）传递引用给函数与传递指针的效果是一样的。这时，被调函数的形参就成为原来主调函数中的实参变量或对象的一个别名来使用，所以在被调函数中对形参变量的操作就是对其相应的目标对象（在主调函数中）的操作。

（2）使用引用传递函数的参数，在内存中并没有产生实参的副本，它是直接对实参操作；而使用一般变量传递函数的参数，当发生函数调用时，需要给形参分配存储单元，形参变量是实参变量的副本；如果传递的是对象，还将调用拷贝构造函数。因此，当参数传递的数据较大时，用引用比用一般变量传递参数的效率和所占空间都好。

（3）使用指针作为函数的参数虽然也能达到与使用引用的效果，但是，在被调函数中同样要给形参分配存储单元，且需要重复使用"*指针变量名"的形式进行运算，这很容易产生错误且程序的阅读性较差；另一方面，在主调函数的调用点处，必须用变量的地址作为实参。而引用更容易使用，更清晰。

134. 什么时候需要“引用”?
流操作符（<<、>>）和赋值操作符（=）的返回值、拷贝构造函数的参数、赋值操作符的参数、其它情况都推荐使用引用。

135.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实行 protection(private, protected,public)

2. 继承：广义的继承有三种实现形式：实现继承（指使用基类的属性和方法而无需额外编

码的能力）、可视继承（子窗体使用父窗体的外观和实现代码）、接口继承（仅使用属性和方法，实现滞后到子类实现）。前两种（类继承）和后一种（对象组合=>接口继承以及纯虚函数）构成了功能复用的两种方式。

3. 多态：是将父对象设置成为和一个或更多的他的子对象相等的技术，赋值之后，父对象就可以根据当前赋值给它的子对象的特性以不同的方式运作。简单的说，就是一句话：允许将子类类型的指针赋值给父类类型的指针。

136.求下面函数的返回值（微软）

```
int func(x)
{
int countx = 0;
while(x)
{
countx ++;
x = x&(x-1);
}
return countx;
}
```

假定 x = 9999。 答案：8
思路：将 x 转化为 2 进制，看含有的 1 的个数。

137、写出下列代码的输出内容

```
#i nclude<stdio.h>
int inc(int a)
{
return(++a);
}
int multi(int*a,int*b,int*c)
{
return(*c=*a**b);
}
typedef int(FUNC1)(int in);
typedef int(FUNC2) (int*,int*,int*);
void show(FUNC2 fun,int arg1, int*arg2)
{
INCp=&inc;
int temp =p(arg1);
fun(&temp,&arg1, arg2);
printf("%d\n",*arg2);
}
```

```
main()
{
int a;
show(multi,10,&a);
return 0;
}
```
答：110

138。编写一个 C 函数，该函数在一个字符串中找到可能的最长的子字符串，且该字符串是由同一字符组成的。

```
char * search(char *cpSource, char ch)
{
        char *cpTemp=NULL, *cpDest=NULL;
        int iTemp, iCount=0;
        while(*cpSource)
        {
                if(*cpSource == ch)
                {
                        iTemp = 0;
                        cpTemp = cpSource;
                        while(*cpSource == ch)
++iTemp, ++cpSource;
                        if(iTemp > iCount)
iCount = iTemp, cpDest = cpTemp;
       if(!*cpSource)
break;
                }
                ++cpSource;
 }
 return cpDest;
}
```

139。请编写一个 C 函数，该函数在给定的内存区域搜索给定的字符，并返回该字符所在位置索引值。

```
int search(char *cpSource, int n, char ch)
{
        int i;
        for(i=0; i<n && *(cpSource+i) != ch; ++i);
        return i;
}
```

140.一个单向链表，不知道头节点,一个指针指向其中的一个节点，问如何删除这个指针指向的节点?

将这个指针指向的 next 节点值 copy 到本节点，将 next 指向 next->next,并随后删除原 next 指向的节点。

141、用指针的方法，将字符串“ABCD1234efgh”前后对调显示

```
#i nclude <stdio.h>
#i nclude <string.h>
#i nclude <dos.h>
int main()
{
   char str[] = "ABCD1234efgh";
   int length = strlen(str);
   char * p1 = str;
   char * p2 = str + length - 1;
   while(p1 < p2)
   {
      char c = *p1;
      *p1 = *p2;
      *p2 = c;
      ++p1;
      --p2;
   }
   printf("str now is %s\n",str);
   system("pause");
   return 0;
}
```

142、有一分数序列：1/2,1/4,1/6,1/8……，用函数调用的方法，求此数列前 20 项的和

```
#i nclude <stdio.h>
double getValue()
{
   double result = 0;
   int i = 2;
   while(i < 42)
   {
      result += 1.0 / i;//一定要使用 1.0 做除数，不能用 1，否则结果将自动转化成整数，即 0.000000
      i += 2;
   }
   return result;
}
int main()
{
   printf("result is %f\n", getValue());
   system("pause");
   return 0;
}
```

143、有一个数组 a[1000]存放 0--1000;要求每隔二个数删掉一个数，到末尾时循环至开头继续进行，求最后一个被删掉的数的原始下标位置。

以 7 个数为例：
{0,1,2,3,4,5,6,7} 0-->1-->2（删除）-->3-->4-->5(删除)-->6-->7-->0（删除），如此循环直到最后一个数被删除。
方法 1：数组

```
#i nclude <iostream>
using namespace std;
#define null 1000
int main()
{
int arr[1000];
for (int i=0;i<1000;++i)
arr[i]=i;
int j=0;
int count=0;
while(count<999)
{
while(arr[j%1000]==null)
j=(++j)%1000;
j=(++j)%1000;
while(arr[j%1000]==null)
j=(++j)%1000;
j=(++j)%1000;
while(arr[j%1000]==null)
j=(++j)%1000;
arr[j]=null;
++count;
}
while(arr[j]==null)
j=(++j)%1000;
cout<<j<<endl;
return 0;
}
```

方法 2：链表

```
#i nclude<iostream>
using namespace std;
#define null 0
struct node
{
int data;
node* next;
};
int main()
{
node* head=new node;
```

```
head->data=0;
head->next=null;
node* p=head;
for(int i=1;i<1000;i++)
{
node* tmp=new node;
tmp->data=i;
tmp->next=null;
head->next=tmp;
head=head->next;
}
head->next=p;
while(p!=p->next)
{
p->next->next=p->next->next->next;
p=p->next->next;
}
cout<<p->data;
return 0;
}
```

方法 3：通用算法

```
#i nclude <stdio.h>
#define MAXLINE 1000   //元素个数
/*
MAXLINE   元素个数
a[]       元素数组
R[]       指针场
suffix    下标
index     返回最后的下标序号
values    返回最后的下标对应的值
start     从第几个开始
K         间隔
*/
int find_n(int a[],int R[],int K,int& index,int& values,int s=0) {
  int suffix;
  int front_node,current_node;
  suffix=0;
    if(s==0) {
    current_node=0;
    front_node=MAXLINE-1;
 }
    else {
    current_node=s;
```

```
    front_node=s-1;
    }
      while(R[front_node]!=front_node) {
         printf("%d\n",a[current_node]);
         R[front_node]=R[current_node];
         if(K==1) {
           current_node=R[front_node];
           continue;
         }
         for(int i=0;i<K;i++){
            front_node=R[front_node];
         }
         current_node=R[front_node];
      }
 index=front_node;
 values=a[front_node];
 return 0;
}
int main(void) {
int a[MAXLINE],R[MAXLINE],suffix,index,values,start,i,K;
suffix=index=values=start=0;
K=2;
for(i=0;i<MAXLINE;i++) {
a[i]=i;
R[i]=i+1;
}
R[i-1]=0;
find_n(a,R,K,index,values,2);
printf("the value is %d,%d\n",index,values);
return 0;
}
```

144、指出下列程序有什么错误：

```
void test2()
{
  char string[10], str1[10];
  int i;
  for(i=0; i<10; i++)
  {
    str1[i] = 'a';
  }
  strcpy( string, str1 );
}
```

解答：对试题 2，如果面试者指出字符数组 str1 不能在数组内结束可以给 3 分；如果面试者指出 strcpy(string, str1)调用使得从 str1 内存起复制到 string 内存起所复制的字节

数具有不确定性可以给 7 分，在此基础上指出库函数 strcpy 工作方式的给 10 分；
str1 不能在数组内结束:因为 str1 的存储为：{a,a,a,a,a,a,a,a,a,a},没有'\0'(字符串结束符)，所以不能结束
strcpy(char *s1,char *s2)他的工作原理是，扫描 s2 指向的内存，逐个字符付到 s1 所指向的内存，直到碰到'\0',因为 str1 结尾没有'\0'，所以具有不确定性，不知道他后面还会付什么东东。
正确应如下

```
void test2()
{
  char string[10], str1[10];
  int i;
  for(i=0; i<9; i++)
  {
    str1[i] = 'a'+i; //把 abcdefghi 赋值给字符数组
  }
  str[i]='\0';//加上结束符
  strcpy( string, str1 );
}
```

145、实现 strcmp

```
int StrCmp(const char *str1, const char *str2)
{
     assert(str1 && srt2);
while(*str1 && *str1++ = = *str2++);
return *str1-*str2;
}
```

146.符串 A 和 B,输出 A 和 B 中的最大公共子串。
比如 A="aocdfe" B="pmcdfa" 则输出"cdf"

```
*/
//Author: azhen
#i nclude<stdio.h>
#i nclude<stdlib.h>
#i nclude<string.h>
char *commanstring(char shortstring[], char longstring[])
{
int i, j;
char *substring=malloc(256);
if(strstr(longstring, shortstring)!=NULL)                    // 如果……，那么返回 shortstring
return shortstring;
for(i=strlen(shortstring)-1;i>0; i--)                //否则，开始循环计算
{
for(j=0; j<=strlen(shortstring)-i; j++){
memcpy(substring, &shortstring[j], i);
substring[i]='\0';
```

```
if(strstr(longstring, substring)!=NULL)
return substring;
}
}
return NULL;
}

main()
{
char *str1=malloc(256);
char *str2=malloc(256);
char *comman=NULL;
gets(str1);
gets(str2);
if(strlen(str1)>strlen(str2))                    //将短的字符串放前面
comman=commanstring(str2, str1);
else
comman=commanstring(str1, str2);
printf("the longest comman string is: %s\n", comman);
}
```

147、写一个函数比较两个字符串 str1 和 str2 的大小，若相等返回 0，若 str1 大于 str2 返回 1，若 str1 小于 str2 返回－1

```
int strcmp ( const char * src,const char * dst)
{
	int ret = 0 ;
	while( ! (ret = *(unsigned char *)src - *(unsigned char *)dst) && *dst)
{
		++src;
++dst;
}
	if ( ret < 0 )
		ret = -1 ;
	else if ( ret > 0 )
		ret = 1 ;
	return( ret );
}
```

148、判断一个字符串是不是回文

```
int IsReverseStr(char *aStr)
{
int i,j;
int found=1;
if(aStr==NULL)
return -1;
```

```
j=strlen(aStr);
for(i=0;i<j/2;i++)
if(*(aStr+i)!=*(aStr+j-i-1))
{
found=0;
break;
}
return found;
```

```
149 #include main()
{
 int c[3][3]={1,2,3,4,5,6,7,8,9};
for(int i=0;i<3;i++)
for(int j=0;j<3;j++)
printf("%ld\n",&c[j]);
printf("------------------------\n");
printf("%ld\n",(c+1));
printf("%ld\n",(*c+1));
printf("%ld\n",&c[0][0]);
printf("%ld\n",**c);
printf("%ld\n",*c[0]);
if(int(c)==int(*c)) printf("equl");
}
```

为什么 c，*c 的值相等，（c+1），（*c+1）的值不等 c，*c，**c，代表什么意思?

参考答案：

c 是第一个元素的地址，*c 是第一行元素的首地址，其实第一行元素的地址就是第一个元素的地址，这容易理解。**c 是提领第一个元素。

为什么 c，*c 的值相等?

int c 因为直接用 c 表示数组 c[0][0] printf("%ld\n",*c[0]);语句已将指针移到数组头。int(*c)表示 c0 的值为 1，所以相等。数组 c 的存放空间示意如下：（机器中是行优先存放的） c[0][0] c[0][1] c[0][2] c[1][0] c[1][1] c[1][2] c[2][0] c[2][1] c[2][2] c 是一个二维数组名，实际上它是一个指针常量，不能进行自加、自减运算，即：c++、c--、++c、--c 都是不允许的；

c： 数组名；是一个二维指针，它的值就是数组的首地址，也即第一行元素的首地址（等于 *c），也 等于第一行第一个元素的地址（ & c[0][0]）；可以说成是二维数组的行指针。*c： 第一行元素的首地址；是一个一维指针，可以说成是二维数组的列指针。**c：二维数组中的第一个元素的值；即：c[0][0] 所以： c 和 *c 的值是相等的，但他们两者不能相互赋值，（类型不同）；（c + 1）：c 是行指针，（c + 1）是在 c 的基础上加上二维数组一行的地址长度，即从&c[0][0] 变到了&c[1][0]；（*c + 1）：*c 是列指针，（*c + 1）是在*c 的基础上加上二数组一个元素的所占的长度，即从 &c[0][0]变到了 &c[0][1] 从而（c + 1）和（*c + 1）的值就不相等了

150、定义 int **a[3][4], 则变量占有的内存空间为：_____ 参考答案：

int **p; /*16 位下 sizeof(p)=2, 32 位下 sizeof(p)=4*/ 总共 3*4*sizeof(p)

151、写出判断 ABCD 四个表达式的是否正确, 若正确, 写出经过表达式中 a 的值
int a = 4;
(A)a += (a++); (B) a += (++a) ;(C) (a++) += a;(D) (++a) += (a++);
a = ?
答：C 错误，左侧不是一个有效变量，不能赋值，可改为(++a) += a;
改后答案依次为 9,10,10,11

152、某 32 位系统下, C++程序，请计算 sizeof 的值
char str[] = “http://www.ibegroup.com/”
char *p = str ;
int n = 10;
请计算
（1）sizeof (str) = ？
（2）sizeof (p) = ？
（3）sizeof (n) = ？
void Foo (char str[100]){
请计算
sizeof(str) = ？（4）
}
void *p = malloc(100);
请计算
sizeof (p) = ？（5）
答：（1）17（2）4（3）4（4）4（5）4

153、回答下面的问题
(1).Void GetMemory(char **p, int num){
*p = (char *)malloc(num);
}
void Test(void){
char *str = NULL;
GetMemory(&str, 100);
strcpy(str, "hello");
printf(str);
}
请问运行 Test 函数会有什么样的结果?
答：输出“hello”

154、 void Test(void){
char *str = (char *) malloc(100);
strcpy(str, “hello”);
free(str);
if(str != NULL){

strcpy(str, “world”);
printf(str);
}
}
请问运行 Test 函数会有什么样的结果?
答：输出“world”

155、 char *GetMemory(void){
char p[] = "hello world";
return p;
}
void Test(void){
char *str = NULL;
str = GetMemory();
printf(str);
}
请问运行 Test 函数会有什么样的结果?
答：无效的指针，输出不确定

156、编写 strcat 函数
已知 strcat 函数的原型是 char *strcat (char *strDest, const char *strSrc);
其中 strDest 是目的字符串，strSrc 是源字符串。
（1）不调用 C++/C 的字符串库函数，请编写函数 strcat
答：
VC 源码：
char * __cdecl strcat (char * dst, const char * src)
{
char * cp = dst;
while(*cp)
cp++;
while(*cp++ = *src++) ;
return(dst);
}

157、strcat 能把 strSrc 的内容连接到 strDest，为什么还要 char * 类型的返回值?
答：方便赋值给其他变量

158、MFC 中 CString 是类型安全类么?
答：不是，其它数据类型转换到 CString 可以使用 CString 的成员函数 Format 来转换

159.C++中什么数据分配在栈或堆中?
答：栈: 存放局部变量，函数调用参数,函数返回值，函数返回地址。由系统管理
堆: 程序运行时动态申请，new 和 malloc 申请的内存就在堆上

160、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。

161、 int i=10, j=10, k=3; k*=i+j; k 最后的值是?
答：60，此题考察优先级，实际写成： k*=(i+j);，赋值运算符优先级最低

162、do……while 和 while……do 有什么区别?
答 、前一个循环一遍再判断，后一个判断以后再循环

163、请写出下列代码的输出内容
```
#i nclude
main()
{
int a,b,c,d;
a=10;
b=a++;
c=++a;
d=10*a++;
printf("b，c，d：%d，%d，%d"，b，c，d）;
return 0;
}
```
答 、10，12，120

164.在 c 语言库函数中将一个字符转换成整型的函数是 atol()吗，这个函数的原型是什么?
答 、函数名: atol
功 能: 把字符串转换成长整型数
用 法: long atol(const char *nptr);
程序例:
```
#include
#include
int main(void)
{
   long l;
   char *str = "98765432";
   l = atol(lstr);
   printf("string = %s integer = %ld\n", str, l);
   return(0);
```

}

165. 以下三条输出语句分别输出什么?
char str1[] = "abc";
char str2[] = "abc";
const char str3[] = "abc";
const char str4[] = "abc";
const char* str5 = "abc";
const char* str6 = "abc";
cout << boolalpha << (str1==str2) << endl; // 输出什么?
cout << boolalpha << (str3==str4) << endl; // 输出什么?
cout << boolalpha << (str5==str6) << endl; // 输出什么?
答：分别输出 false,false,true。str1 和 str2 都是字符数组，每个都有其自己的存储区，它们的值则是各存储区首地址，不等；str3 和 str4 同上，只是按 const 语义，它们所指向的数据区不能修改。str5 和 str6 并非数组而是字符指针，并不分配存储区，其后的“abc”以常量形式存于静态数据区，而它们自己仅是指向该区首地址的指针，相等。

166 以下代码中的两个 sizeof 用法有问题吗?
void UpperCase(char str[]) // 将 str 中的小写字母转换成大写字母
{
for(size_t i=0; i<sizeof(str)/sizeof(str[0]); ++i)
if('a'<=str[i] && str[i]<='z')
str[i] -= ('a'-'A');
}
char str[] = "aBcDe";
cout << "str 字符长度为: " << sizeof(str)/sizeof(str[0]) << endl;
UpperCase(str);
cout << str << endl;

答：函数内的 sizeof 有问题。根据语法，sizeof 如用于数组，只能测出静态数组的大小，无法检测动态分配的或外部数组大小。函数外的 str 是一个静态定义的数组，因此其大小为 6，函数内的 str 实际只是一个指向字符串的指针，没有任何额外的与数组相关的信息，因此 sizeof 作用于上只将其当指针看，一个指针为 4 个字节，因此返回 4。

167 非 C++内建型别 A 和 B，在哪几种情况下 B 能隐式转化为 A?
答:
a. class B : public A { ……} // B 公有继承自 A，可以是间接继承的
b. class B { operator A( ); } // B 实现了隐式转化为 A 的转化
c. class A { A(const B&); } // A 实现了 non-explicit 的参数为 B（可以有其他带默认值的参数）构造函数
d. A& operator= (const A&); // 赋值操作，虽不是正宗的隐式类型转换，但也可以勉强算一个
168.以下代码有什么问题?
struct Test

```
{
Test( int ) {}
Test() {}
void fun() {}
};
void main( void )
{
Test a(1);
a.fun();
Test b();
b.fun();
}
```

答：变量 b 定义出错。按默认构造函数定义对象，不需要加括号。

169 以下代码有什么问题?

```
cout << (true?1:"1") << endl;
```

答：三元表达式“？:”问号后面的两个操作数必须为同一类型。

170 以下代码能够编译通过吗，为什么?

```
unsigned int const size1 = 2;
char str1[ size1 ];
unsigned int temp = 0;
cin >> temp;
unsigned int const size2 = temp;
char str2[ size2 ];
```

答：str2 定义出错，size2 非编译器期间常量，而数组定义要求长度必须为编译期常量。

171. 以下代码中的输出语句输出 0 吗，为什么?

```
struct CLS
{
int m_i;
CLS( int i ) : m_i(i) {}
CLS()
{
CLS(0);
}
};
CLS obj;
cout << obj.m_i << endl;
```

答：不能。在默认构造函数内部再调用带参的构造函数属用户行为而非编译器行为，亦即仅执行函数调用，而不会执行其后的初始化表达式。只有在生成对象时，初始化表达式才会随

相应的构造函数一起调用。

172 C++中的空类，默认产生哪些类成员函数?
答:
```
class Empty
{
public:
Empty(); // 缺省构造函数
Empty( const Empty& ); // 拷贝构造函数
~Empty(); // 析构函数
Empty& operator=( const Empty& ); // 赋值运算符
Empty* operator&(); // 取址运算符
const Empty* operator&() const; // 取址运算符 const
};
```

173 以下两条输出语句分别输出什么?
```
float a = 1.0f;
cout << (int)a << endl;
cout << (int&)a << endl;
cout << boolalpha << ( (int)a == (int&)a ) << endl; // 输出什么?
float b = 0.0f;
cout << (int)b << endl;
cout << (int&)b << endl;
cout << boolalpha << ( (int)b == (int&)b ) << endl; // 输出什么?
```

答：分别输出 false 和 true。注意转换的应用。(int)a 实际上是以浮点数 a 为参数构造了一个整型数，该整数的值是 1，(int&)a 则是告诉编译器将 a 当作整数看（并没有做任何实质上的转换）。因为 1 以整数形式存放和以浮点形式存放其内存数据是不一样的，因此两者不等。对 b 的两种转换意义同上，但是 0 的整数形式和浮点形式其内存数据是一样的，因此在这种特殊情形下，两者相等（仅仅在数值意义上）。
注意，程序的输出会显示(int&)a=1065353216，这个值是怎么来的呢？前面已经说了，1 以浮点数形式存放在内存中，按 ieee754 规定，其内容为 0x0000803F（已考虑字节反序）。这也就是 a 这个变量所占据的内存单元的值。当(int&)a 出现时，它相当于告诉它的上下文：“把这块地址当做整数看待！不要管它原来是什么。”这样，内容 0x0000803F 按整数解释，其值正好就是 1065353216（十进制数）。
通过查看汇编代码可以证实“(int)a 相当于重新构造了一个值等于 a 的整型数”之说，而(int&)的作用则仅仅是表达了一个类型信息，意义在于为 cout<<及==选择正确的重载版

174、请简述以下两个 for 循环的优缺点（5 分）

```
for (i=0; i<N; i++)
```

```
{
if (condition)
    DoSomething();
else
    DoOtherthing();
}
 if (condition)
{
for (i=0; i<N; i++)
    DoSomething();
}
else
{
        for (i=0; i<N; i++)
    DoOtherthing();
}
```

优点：程序简洁
缺点：多执行了 N-1 次逻辑判断，并且打断了循环“流水线”作业，使得编译器不能对循环进行优化处理，降低了效率。

优点：循环的效率高
缺点：程序不简洁

175

```
void GetMemory(char *p)
{
p = (char *)malloc(100);
}
void Test(void)
{
char *str = NULL;
GetMemory(str);
strcpy(str, "hello world");
printf(str);
}
```

请问运行 Test 函数会有什么样的结果?
答：程序崩溃。
因为 GetMemory 并不能传递动态内存，
Test 函数中的 str 一直都是 NULL。
strcpy(str, "hello world");将使程序崩溃。

176

```
 char *GetMemory(void)
```

```
{
char p[] = "hello world";
return p;
}
void Test(void)
{
char *str = NULL;

str = GetMemory();
printf(str);
}
```

请问运行 Test 函数会有什么样的结果?

答：可能是乱码。

因为 GetMemory 返回的是指向“栈内存”的指针，该指针的地址不是 NULL，但其原现的内容已经被清除，新内容不可知。

177

```
void GetMemory2(char **p, int num)
{
*p = (char *)malloc(num);
}
void Test(void)
{
char *str = NULL;
GetMemory(&str, 100);
strcpy(str, "hello");
printf(str);
}
```

请问运行 Test 函数会有什么样的结果?

答:

（1）能够输出 hello

（2）内存泄漏

178

```
 void Test(void)
{
char *str = (char *) malloc(100);
    strcpy(str, “hello”);
    free(str);
    if(str != NULL)
    {
      strcpy(str, “world”);
      printf(str);
    }
```

```
}
```
请问运行 Test 函数会有什么样的结果?
答：篡改动态内存区的内容，后果难以预料，非常危险。
因为 free(str);之后，str 成为野指针，
if(str != NULL)语句不起作用。

179、请阅读以下一段程序，并给出答案。

```
class A
{
public:
    A(){ doSth(); }
    virtual void doSth(){printf("I am A");}
};
class B:public A
{
public:
    virtual void doSth(){ printf("I am B");}
};
B b;
```

执行结果是什么？为什么?
答：执行结果是 I am A
因为 b 对象构造时调用基类 A 的构造函数 A()，得此结果。

180 实现双向链表删除一个节点 P，在节点 P 后插入一个节点，写出这两个函数;
答：双向链表删除一个节点 P

```
template<class type> void list<type>::delnode(int p)
{
 int k=1;
 listnode<type> *ptr,*t;
 ptr=first;

 while(ptr->next!=NULL&&k!=p)
 {
  ptr=ptr->next;
     k++;
 }
    t=ptr->next;
 cout<<"你已经将数据项 "<<t->data<<"删除"<<endl;

 ptr->next=ptr->next->next;
 length--;
 delete t;
```

```
}
```

在节点 P 后插入一个节点：

```
template<class type> bool list<type>::insert(type t,int p)
{
 listnode<type> *ptr;
 ptr=first;

 int k=1;
 while(ptr!=NULL&&k<p)
 {
  ptr=ptr->next;
  k++;
 }
 if(ptr==NULL&&k!=p)
  return false;
 else
 {
   listnode<type> *tp;
   tp=new listnode<type>;
   tp->data=t;
   tp->next=ptr->next;
   ptr->next=tp;
   length++;

   return true;
 }
}
```

181. 完成下列程序

```
*

*.*.

*..*..*..

*...*...*...*...

*....*....*....*....*....

*.....*.....*.....*.....*.....*.....

*......*......*......*......*......*......*......
```

..........*.....*.....*.....*.....*.....*.....*.....

```
#i nclude<iostream>
using namespace std;

const int n = 8;

main()
{
  int i;
  int j;
  int k;

  for(i = n; i >= 1; i--)
  {
    for(j = 0; j < n-i+1; j++)
    {
      cout<<"*";
      for(k=1; k < n-i+1; k++)
      {
        cout<<".";
      }
    }
    cout<<endl;
  }
  system("pause");
}
```

182 完成程序，实现对数组的降序排序

```
#include <iostream>
using namespace std;

void sort(int* arr, int n);

int main()

{
  int array[]={45,56,76,234,1,34,23,2,3};
  sort(array, 9);
  for(int i = 0; i <= 8; i++)//曾经在这儿出界
    cout<<array[i]<<" ";
```

```
  cout<<endl;
  system("pause");
}

void sort(int* arr, int n)
{
  int temp;
  for(int i = 1; i < 9; i++)
  {
    for(int k = 0; k < 9 - i; k++)//曾经在这儿出界
    {
      if(arr[k] < arr[k + 1])
        {
          temp = arr[k];
          arr[k] = arr[k + 1];
          arr[k + 1] = temp;
        }
    }
  }
}
```

183. 以下两条输出语句分别输出什么？[C++难]

```
float a = 1.0f;
cout << (int)a << endl;
cout << (int&)a << endl;
cout << boolalpha << ( (int)a == (int&)a ) << endl; // 输出什么？
float b = 0.0f;
cout << (int)b << endl;
cout << (int&)b << endl;
cout << boolalpha << ( (int)b == (int&)b ) << endl; // 输出什么？
```

```
1
1065353216
boolalpha0
0
0
boolalpha1
```

51. 以下反向遍历 array 数组的方法有什么错误？[STL 易]

```
vector array;
array.push_back( 1 );
array.push_back( 2 );
array.push_back( 3 );
for( vector::size_type i=array.size()-1; i>=0; --i ) // 反向遍历 array 数组
{
```

```
    cout << array[i] << endl;
}
```

184 写一个函数，完成内存之间的拷贝。[考虑问题是否全面]
答:

```
void* mymemcpy( void *dest, const void *src, size_t count )
{
    char* pdest = static_cast<char*>( dest );
    const char* psrc = static_cast<const char*>( src );
    if( pdest>psrc && pdest<psrc+cout ) 能考虑到这种情况就行了
    {
        for( size_t i=count-1; i!=-1; --i )
                pdest[i] = psrc[i];
    }
    else
    {
        for( size_t i=0; i<count; ++i )
            pdest[i] = psrc[i];
    }
    return dest;
}
int main( void )
{
    char str[] = "0123456789";
    mymemcpy( str+1, str+0, 9 );
    cout << str << endl;

    system( "Pause" );
    return 0;
}
```

185 对于C++中类(class) 与结构(struct)的描述正确的为:
A,类中的成员默认是 private 的,当是可以声明为 public,private 和 protected,
结构中定义的成员默认的都是 public;
B,结构中不允许定义成员函数,当是类中可以定义成员函数;
C,结构实例使用 malloc() 动态创建,类对象使用 new 操作符动态分配内存;
D,结构和类对象都必须使用 new 创建;
E,结构中不可以定义虚函数,当是类中可以定义虚函数.
F,结构不可以存在继承关系,当是类可以存在继承关系.
答:A,D,F

186. 两个互相独立的类:ClassA 和 ClassB, 都各自定义了非静态的公有成员函数 PublicFunc() 和非静态的私有成员函数 PrivateFunc();
现在要在 ClassA 中增加定义一个成员函数 ClassA::AdditionalPunction(ClassA

a,ClassB b);则可以在 AdditionalPunction(ClassA x,ClassB y)的实现部分(函数功能体内部)

出现的合法的表达是最全的是:

A,x.PrivateFunc();x.PublicFunc();y.PrivateFunc();y.PublicFunc();

B,x.PrivateFunc();x.PublicFunc();y.PublicFunc();

C,x.PrivateFunc();y.PrivateFunc();y.PublicFunc();

D,x.PublicFunc();y.PublicFunc();

答:B

186.C++程序下列说法正确的有:

A,对调用的虚函数和模板类都进行迟后编译.

B,基类与子类中函数如果要构成虚函数,除了要求在基类中用 virtual 声名,而且必须名字相同且参数类型相同返回类型相同

C,重载的类成员函数都必须要:或者返回类型不同,或者参数数目不同,或者参数序列的类型不同.

D,静态成员函数和内联函数不能是虚函数,友员函数和构造函数也不能是虚函数,但是析构函数可以是虚函数.

答:A

187 头文件的作用是什么?

答：一、通过头文件来调用库功能。在很多场合，源代码不便（或不准）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，而不必关心接口怎么实现的。编译器会从库中提取相应的代码。

二、头文件能加强类型安全检查。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。

188、以下为 Windows NT 下的 32 位 C++程序，请计算 sizeof 的值（10 分）

```
char  str[] = “Hello” ;
char   *p = str ;
int     n = 10;
```

请计算

sizeof (str) = 6 （2 分）

sizeof (p) = 4 （2 分）

sizeof (n) = 4 （2 分）void Func (char str[100])

{

请计算

sizeof(str) = 4 （2 分）

}

```
void *p = malloc( 100 );
```

请计算

sizeof (p) = 4 （2分）

3 写出下列程序的运行结果。

```
unsigned int i=3;
cout<<i * -1;
```

189.写出下列程序所有可能的运行结果。

```
int a;
int b;
int c;

void F1()
{
b=a*2;
a=b;
}

void F2()
{
c=a+1;
a=c;
}

main()
{
a=5;
//Start F1,F2 in parallel
F1(); F2();
printf("a=%d\n",a);
}a=11
```

190 一个链表的操作，注意代码的健壮和安全性。要求：
（1）增加一个元素；
（2）获得头元素；
（3）弹出头元素（获得值并删除）。

191.unsigned short array[]={1,2,3,4,5,6,7};
int i = 3;
*(array + i) = ?
答:

4

192
class A
{
  virtual void func1();
  void func2();
}
Class B: class A
{
  void func1(){cout << "fun1 in class B" << endl;}
  virtual void func2(){cout << "fun2 in class B" << endl;}
}
A, A 中的 func1 和 B 中的 func2 都是虚函数.
B, A 中的 func1 和 B 中的 func2 都不是虚函数.
C, A 中的 func2 是虚函数.，B 中的 func1 不是虚函数.
D, A 中的 func2 不是虚函数，B 中的 func1 是虚函数.

答:
A

193 输出下面程序结果。
#include <iostream.h>
class A
{
public:
 virtual void print(void)
 {
    cout<<"A::print()"<<endl;
 }
};
class B:public A
{
public:
 virtual void print(void)
 {
   cout<<"B::print()"<<endl;
 };
};
class C:public B
{
public:
 virtual void print(void)
 {

```
  cout<<"C::print()"<<endl;
 }
};
void print(A a)
{
   a.print();
}
void main(void)
{
   A a, *pa,*pb,*pc;
   B b;
   C c;

   pa=&a;
   pb=&b;
   pc=&c;

   a.print();
   b.print();
   c.print();

   pa->print();
   pb->print();
   pc->print();

   print(a);
   print(b);
   print(c);
}
```

```
A::print()
A::print()
B::print()
C::print()
A::print()
B::print()
C::print()
A::print()
A::print()
A::print()
```

194.程序改错

```
class mml
```

```
{
  private:
    static unsigned int x;
  public:
    mml(){ x++; }
    mml(static unsigned int &) {x++;}
    ~mml{x--;}
  pulic:
    virtual mon() {} = 0;
    static unsigned int mmc(){return x;}
    ......

};
class nnl:public mml
{
  private:
    static unsigned int y;
  public:
    nnl(){ x++; }
    nnl(static unsigned int &) {x++;}
    ~nnl{x--;}
  public:
    virtual mon() {};
     static unsigned int nnc(){return y;}
    ......
};
```

代码片断:

```
mml* pp = new nnl;
.........
delete pp;
```

A:

基类的析构函数应该为虚函数

```
virtual ~mml{x--;}
```

195.101 个硬币 100 真、1 假，真假区别在于重量。请用无砝码天平称两次给出真币重还是假币重的结论。

答:

101 个先取出 2 堆,

33,33

第一次称,如果不相等,说明有一堆重或轻

那么把重的那堆拿下来,再放另外 35 个中的 33
如果相等,说明假的重,如果不相等,新放上去的还是重的话,说明假的轻(不可能新放上去的轻)

第一次称,如果相等的话，这 66 个肯定都是真的,从这 66 个中取出 35 个来,与剩下的没称过的 35 个比
下面就不用说了

方法二:
第 3 题也可以拿 A(50),B(50)比一下，一样的话拿剩下的一个和真的比一下。
如果不一样，就拿其中的一堆。比如 A(50)再分成两堆 25 比一下，一样的话就在 B(50)中，不一样就在 A(50)中，结合第一次的结果就知道了。

196. 写出程序结果:
```
void Func(char str[100])
{
  printf("%d\n", sizeof(str));
}
```

答:
4
分析:
指针长度

197. int id[sizeof(unsigned long)];
这个对吗？为什么??
答:
对
这个 sizeof 是编译时运算符，编译时就确定了
可以看成和机器有关的常量。

198、 sizeof 应用在结构上的情况
请看下面的结构：
```
struct MyStruct
{
double dda1;
char dda;
int type
};
```
对结构 MyStruct 采用 sizeof 会出现什么结果呢？sizeof(MyStruct)为多少呢？也许你会这样求：
sizeof(MyStruct)=sizeof(double)+sizeof(char)+sizeof(int)=13
但是当在 VC 中测试上面结构的大小时，你会发现 sizeof(MyStruct)为 16。你知道为什么在 VC 中会得出这样一个结果吗？

其实，这是 VC 对变量存储的一个特殊处理。为了提高 CPU 的存储速度，VC 对一些变量的起始地址做了"对齐"处理。在默认情况下，VC 规定各成员变量存放的起始地址相对于结构的起始地址的偏移量必须为该变量的类型所占用的字节数的倍数。下面列出常用类型的对齐方式(vc6.0,32 位系统)。
类型
对齐方式（变量存放的起始地址相对于结构的起始地址的偏移量）
Char
偏移量必须为 sizeof(char)即 1 的倍数
int
偏移量必须为 sizeof(int)即 4 的倍数
float
偏移量必须为 sizeof(float)即 4 的倍数
double
偏移量必须为 sizeof(double)即 8 的倍数
Short
偏移量必须为 sizeof(short)即 2 的倍数
各成员变量在存放的时候根据在结构中出现的顺序依次申请空间，同时按照上面的对齐方式调整位置，空缺的字节 VC 会自动填充。同时 VC 为了确保结构的大小为结构的字节边界数（即该结构中占用最大空间的类型所占用的字节数）的倍?

199
#include "stdafx.h"
 Y n P }2{&k O v H `,o0
#define SQR(X) X*X
int main(int argc, char* argv[])
{
int a = 10;
int k = 2;
int m = 1;
a /= SQR(k+m)/SQR(k+m);
printf("%d\n",a);
return 0;
}
这道题目的结果是什么啊?
define 只是定义而已，在编择时只是简单代换 X*X 而已，并不经过算术法则的
a /= k+m*k+m/k+m*k+m;
=>a /= (k+m)*1*(k+m);
=>a = a/9;
=>a = 1;

200.下面的代码有什么问题?

void DoSomeThing(...)
{

```
char* p;
p = malloc(1024); // 分配 1K 的空间
2y x
if (NULL == p)
return;
p = realloc(p, 2048); // 空间不够，重新分配到 2K
if (NULL == p)
return;
}
```

A:

p = malloc(1024);　应该写成：p = (char *) malloc(1024);

没有释放 p 的空间，造成内存泄漏。

201 下面的代码有什么问题？并请给出正确的写法。

```
void DoSomeThing(char* p)
{
char str[16];
int n;
assert(NULL != p);
sscanf(p, "%s%d", str, n);
if (0 == strcmp(str, "something"))
{
}
}
```

A:

sscanf(p, "%s%d", str, n);　这句该写成：sscanf(p, "%s%d", str, &n);

202.下面代码有什么错误?

```
Void test1()
{

char string[10];
char *str1="0123456789";
strcpy(string, str1);

}
```

数组越界

203.下面代码有什么问题?

```
Void test2()

{
```

```
char string[10], str1[10];

for(i=0; i<10;i++)
{
    str1[i] ='a';
}
strcpy(string, str1);
}
}
```
数组越界

204 下面代码有什么问题?LUPA 开源社区+j H2B F,c U

```
Void test3(char* str1)
{
char string[10];
if(strlen(str1)<=10)
{
   strcpy(string, str1);
}
}
```
==数组越界
==strcpy 拷贝的结束标志是查找字符串中的\0 因此如果字符串中没有遇到\0 的话 会一直复制，直到遇到\0,上面的 123 都因此产生越界的情况
建议使用 strncpy 和 memcpy

205.写出运行结果:

```
{
   char str[] = "world"; cout << sizeof(str) << ": ";
   char *p    = str;     cout << sizeof(p) << ": ";
   char i     = 10;      cout << sizeof(i) << ": ";
   void *pp   = malloc(10); cout << sizeof(p) << endl;
}
```
6：4：1：4

206.C 和 C++有什么不同?
从机制上：c 是面向过程的（但 c 也可以编写面向对象的程序）；c++是面向对象的，提供了类。但是，c++编写面向对象的程序比 c 容易
从适用的方向：c 适合要求代码体积小的，效率高的场合，如嵌入式；c++适合更上层的，复杂的； llinux 核心大部分是 c 写的，因为它是系统软件，效率要求极高。
从名称上也可以看出，c++比 c 多了+，说明 c++是 c 的超集；那为什么不叫 c+而叫 c++呢，是因为 c++比

c 来说扩充的东西太多了，所以就在 c 后面放上两个+；于是就成了 c++
C 语言是结构化编程语言，C++是面向对象编程语言。LUPA 开源社区 } n*r2C/M8f
C++侧重于对象而不是过程，侧重于类的设计而不是逻辑的设计。

207 在不用第三方参数的情况下，交换两个参数的值

```
#include <stdio.h>
void main()
{
      int i=60;
       int j=50;
       i=i+j;
       j=i-j;
)
       i=i-j;
       printf("i=%d\n",i);
       printf("j=%d\n",j);
}
```

方法二：

```
i^=j;
j^=i;
i^=j;
```

方法三：

```
// 用加减实现，而且不会溢出
a = a+b-(b=a)
```

208.下面的函数实现在一个固定的数上加上一个数，有什么错误，改正

```
int add_n(int n)
{
static int i=100;
i+=n;
return i;
}
```

答:
因为 static 使得 i 的值会保留上次的值。
去掉 static 就可了

209. union a {

```
int a_int1;
double a_double;
int a_int2;
};
typedef struct
{ a a1;
```

```
char y;
} b;

class c
{
double c_double;
b b1;
a a2;
};
```
输出 cout<<sizeof(c)<<endl;的结果?
答:
VC6 环境下得出的结果是 32
我(sun)在 VC6.0+win2k 下做过试验:
int-4
float-4
double-8
指针-4

210. unsigned short array[]={1,2,3,4,5,6,7};
int i = 3;
*(array + i) = ?
答:4

211. class A
{
virtual void func1();
void func2();
}
Class B: class A
{
void func1(){cout << "fun1 in class B" << endl;}
virtual void func2(){cout << "fun2 in class B" << endl;}
}
A, A 中的 func1 和 B 中的 func2 都是虚函数.
B, A 中的 func1 和 B 中的 func2 都不是虚函数.
C, A 中的 func2 是虚函数.，B 中的 func1 不是虚函数.
D, A 中的 func2 不是虚函数，B 中的 func1 是虚函数.
答:
A

212 输出下面程序结果。
```
#include <iostream.h>
class A
```

```
{
public:
virtual void print(void)
{
    cout<<"A::print()"<<endl;
}
};

class B:public A
{
public:
 virtual void print(void)
{
   cout<<"B::print()"<<endl;

};
};
class C:public
{
public:
virtual void print(void)
{
   cout<<"C::print()"<<endl;
}
};
void print(A a)
{
   a.print();
}

void main(void)
{
   A a, *pa,*pb,*pc;
   pa=&a;
   pb=&b;
   pc=&c;
   a.print();
   b.print();
   c.print();
  a->print();
   pb->print();
   pc->print();

   print(a);
```

```
  print(b);
   print(c);
}
```

A:
A::print()

B::print()
C::print()

A::print()

B::print()

C::print()

A::print()

A::print()

A::print()

213 C++语言是在___ C _____语言的基础上发展起来的。
214 C++语言的编译单位是扩展名为____ .cpp_____的____程序_____文件。
215. 行尾使用注释的开始标记符为____ //_____。
216 多行注释的开始标记符和结束标记符分别为_____ /*_____和___ */_____。
217. 用于输出表达式值的标准输出流对象是____ cout_____。
218 用于从键盘上为变量输入值的标准输入流对象是__ cin_____。
219. 一个完整程序中必须有一个名为____ main____的函数。
220 一个函数的函数体就是一条____复合_____语句。
221. 当执行 cin 语句时，从键盘上输入每个数据后必须接着输入一个___空白_____符，然后才能继续输入下一个数据。
222 在 C++程序中包含一个头文件或程序文件的预编译命令为____#include _____。
223. 程序中的预处理命令是指以___#___字符开头的命令。
224. 一条表达式语句必须以___分号___作为结束符。
225. 在#include 命令中所包含的头文件，可以是系统定义的头文件，也可以是___用户（或编程者_____定义的头文件。
226. 使用#include 命令可以包含一个头文件，也可以包含一个__程序____文件。
227. 一个函数定义由__函数头_____和__函数体_____两部分组成。
228. 若一个函数的定义处于调用它的函数之前，则在程序开始可以省去该函数的__原型（或声明）____语句。
229. C++头文件和源程序文件的扩展名分别为__.h ___和___.cpp ___。
230. 程序文件的编译错误分为____警告（warning）____和____致命(error) ____两类。

231. 当使用___ void ____保留字作为函数类型时，该函数不返回任何值。
232. 当函数参数表用___ void __保留字表示时，则表示该参数表为空。
233. 从一条函数原型语句“int fun1(void);”可知，该函数的返回类型为_____，该函数带有_____个参数。
234. 当执行 cout 语句输出 endl 数据项时，将使 C++显示输出屏幕上的光标从当前位置移动到___下一行_____的开始位置。
235. 假定 x=5，y=6，则表达式 x++*++y 的值为___35_____。
236. 假定 x=5，y=6，则表达式 x--*--y 的值为___25_____。
237. 假定 x=5，y=6，则执行表达式 y*=x++计算后，x 和 y 的值分别为___6___和___30 _____。
238. 假定 x=5，y=6，则执行表达式 y+=x--计算后，x 和 y 的值分别为____4__和___11___。
239. C++常数 0x145 对应的十进制值为___325 ___。
240. C++常数 0345 对应的十进制值为____ 229__。
241. 十进制常数 245 对应的十六进制的 C++表示为____0xF5___。
242. 十进制常数 245 对应的八进制的 C++表示为___0365 ___。
243. signed char 类型的值域范围是__-128__至___+127 __之间的整数。
244. int 和 float 类型的数据分别占用___ 4___和____ 4___个字节。
245. float 和 double 类型的数据分别占用____ 4___和_____8___个字节。
246. bool 和 char 类型的数据分别占用_____1____和____1___个字节。
247. unsigned short int 和 int 类型的长度分别为____ 2___和____4___。
248. 字符串”This\’s a book.\n”的长度为_____ 15____。
249. 字符串”\nThis\’s a pen\n\n”的长度为_____ 15_____。
250. 在 C++中存储字符串”abcdef”至少需要___7 _____个字节。
251. 在 C++中存储字符串”a+b=c”至少需要_____6 ___个字节。
252. 假定 x 和 y 为整型，其值分别为 16 和 5，则 x%y 和 x/y 的值分别为___1_____和____3____。
253. 假定 x 和 y 为整型，其值分别为 16 和 5，则 x/y 和 double(x)/y 的值分别为____3____和___3.2____。
254. 假定 x 是一个逻辑量，则 x && true 的值为___ x ____。
255. 假定 x 是一个逻辑量，则 x || true 的值为_____ true(或 1)_____。
256. 假定 x 是一个逻辑量，则 x && false 的值为____ false(或 0) ___。
257. 假定 x 是一个逻辑量，则 x || false 的值为 x。
258. 假定 x 是一个逻辑量，则!x || false 的值为____!x ____。
259. 假定 x 是一个逻辑量，则 x && !x 的值为____ false(或 0)____。
260. 假定 x 是一个逻辑量，则 x || !x 的值为____ true(或 1)___。
261. 设 enum Printstatus{ready,busy,error}; 则 cout<<busy 的输出结果是_____1___。
262. 设 enum Printstatus{ready=2,busy,error}; 则 cout<<busy 的输出结果是_____3____。
263. 常数-4.205 和 6.7E-9 分别具有___4_____和____2___位有效数字。
264. 枚举类型中的每个枚举值都是一个____枚举常量_____，它的值为一个___整数____。
265. 常数 100 和 3.62 的数据类型分别为____ int ___和_____ double ___。
266. 若 x=5, y=10, 则计算 y*=++x 表达式后，x 和 y 的值分别为___6___和__60 ___。

267. 假定 x 和 ch 分别为 int 型和 char 型，则 sizeof(x)和 sizeof(ch)的值分别为__4__和__1__。
268. 假定 x=10，则表达式 x<=10?20:30 的值为__ 20 __。
269. 表达式 sqrt(81)和 pow(6,3)的值分别为___9 ___和___216___。
270. 含随机函数的表达式 rand()%20 的值在___0__至___ 19 __区间内。
271. 在 switch 语句中，每个语句标号所含关键字 case 后面的表达式必须是___常量___。
272. 在 if 语句中，每个 else 关键字与它前面同层次并且最接近的____ if ____关键字相配套。
273. 作为语句标号使用的 C++保留字 case 和 defaule 只能用于___ switch ___语句的定义体中。
274. 执行 switch 语句时，在进行作为条件的表达式求值后，将从某个匹配的标号位置起向下执行，当碰到下一个标号位置时（停止/不停止）___不停止__执行。
275. 若 while 循环的“头”为“while(i++<=10)”，并且 i 的初值为 0，同时在循环体中不会修改 i 的值，则循环体将被重复执行__11___次后正常结束。
276. 若 do 循环的“尾”为“while(++i<10)”，并且 i 的初值为 0，同时在循环体中不会修改 i 的值，则循环体将被重复执行___10 ___次后正常结束。
277. 当在程序中执行到 break 语句时，将结束本层循环类语句或 switch 语句的执行。
278. 当在程序中执行到___ continue___语句时，将结束所在循环语句中循环体的一次执行。
279. 在程序中执行到__ return ___语句时，将结束所在函数的执行过程，返回到调用该函数的位置。
280. 在程序执行完____主(或 main)__函数调用后，将结束整个程序的执行过程，返回到 C++集成开发窗口。
281. 元素类型为 int 的数组 a[10]共占用___ 40___字节的存储空间。
282. 元素类型为 double 的二维数组 a[4][6]共占用____192__字节的存储空间。
283. 元素类型为 char 的二维数组 a[10][30]共占用___300__字节的存储空间。
284. 存储字符’a’和字符串”a”分别需要占用_____1___和____2 ___个字节。

285

```
#include "stdafx.h"
#define SQR(X) X*X
int main(int argc, char* argv[])
 {
int a = 10;
int k = 2;
int m = 1;
a /= SQR(k+m)/SQR(k+m);
printf("%d\n",a);
return 0;
}
```

这道题目的结果是什么啊?
define 只是定义而已，在编择时只是简单代换 X*X 而已，并不经过算术法则的
a /= (k+m)*(k+m)/(k+m)*(k+m);
=>a /= (k+m)*1*(k+m);

=>a = a/9;
=>a = 1;

286. 以面向对象方法构造的系统，其基本单位是_____对象___。
287. 每个对象都是所属类的一个__实例__。
288. C++支持两种多态性：___编译____时的多态性和____运行__时的多态性。
289. 在 C++中，编译时的多态性是通过___重载___实现的，而运行时的多态性则是通过___虚函数____实现的。
290. 对于类中定义的任何成员，其隐含访问权限为___ private（或私有）__。
291. 对于结构中定义的任何成员，其隐含访问权限为__ public(或公有)_。
292. 若在类的定义体中给出了一个成员函数的完整定义，则该函数属于__内联__函数。
293. 为了避免在调用成员函数时修改对象中的任何数据成员，则应在定义该成员函数时，在函数头的后面加上__ const __关键字。
294. 若只需要通过一个成员函数读取数据成员的值，而不需要修改它，则应在函数头的后面加上__ const __关键字。
295. 判断一个字符串是不是回文

```
int IsReverseStr(char *aStr)
{
int i,j;
int found=1;
if(aStr==NULL)
return -1;
j=strlen(aStr);
for(i=0;i<j/2;i++)
if(*(aStr+i)!=*(aStr+j-i-1))
{
found=0;
break;
}
return found;
}
```

296. .写出判断 ABCD 四个表达式的是否正确, 若正确, 写出经过表达式中 a 的值(3 分)

```
int a = 4;
```

(A)a += (a++); (B) a += (++a) ;(C) (a++) += a;(D) (++a) += (a++);
a = ?
答：C 错误，左侧不是一个有效变量，不能赋值，可改为(++a) += a;
改后答案依次为 9,10,10,11

298. 动态连接库的两种方式?
答：调用一个 DLL 中的函数有两种方法：
1. 载入时动态链接（load-time dynamic linking），模块非常明确调用某个导出函数，使得他们就像本地函数一样。这需要链接时链接那些函数所在 DLL 的导入库，导入库向系统提供了载入 DLL 时所需的信息及 DLL 函数定位。

2．运行时动态链接（run-time dynamic linking），运行时可以通过 LoadLibrary 或 Loa
dLibraryEx 函数载入 DLL。DLL 载入后，模块可以通过调用 GetProcAddress 获取 DLL 函数的出口地址，然后就可以通过返回的函数指针调用 DLL 函数了。如此即可避免导入库文件了。

299．请写出下列代码的输出内容
#i nclude
main()
{
int a,b,c,d;
a=10;
b=a++;
c=++a;
d=10*a++;
printf("b，c，d：%d，%d，%d"，b，c，d）;
return 0;
}
答 、10，12，120

300．设有以下说明和定义：
typedef union {long i; int k[5]; char c;} DATE;
struct data { int cat; DATE cow; double dog;} too;
DATE max;
则语句 printf("%d",sizeof(struct date)+sizeof(max));的执行结果是?
答 、结果是：___52____。DATE 是一个 union, 变量公用空间. 里面最大的变量类型是 int[5], 占用 20 个字节. 所以它的大小是 20
data 是一个 struct, 每个变量分开占用空间. 依次为 int4 + DATE20 + double8 = 32.
所以结果是 20 + 32 = 52.
当然...在某些 16 位编辑器下, int 可能是 2 字节,那么结果是 int2 + DATE10 + double8 = 20

301． 以下代码有什么问题?
cout << (true?1:"1") << endl;
答：三元表达式“？:”问号后面的两个操作数必须为同一类型。

302．以下代码能够编译通过吗，为什么?
unsigned int const size1 = 2;
char str1[size1];
unsigned int temp = 0;
cin >> temp;
unsigned int const size2 = temp;
char str2[size2];
答：str2 定义出错，size2 非编译器期间常量，而数组定义要求长度必须为编译期常量。

303. 以下反向遍历 array 数组的方法有什么错误?

```
vector array;
array.push_back( 1 );
array.push_back( 2 );
array.push_back( 3 );
for( vector::size_type i=array.size()-1; i>=0; --i ) // 反向遍历 array 数组
{
cout << array[i] << endl;
}
```

答：首先数组定义有误，应加上类型参数：vector<int> array。其次 vector::size_type 被定义为 unsigned int，即无符号数，这样做为循环变量的 i 为 0 时再减 1 就会变成最大的整数，导致循环失去控制。

304. 以下代码中的输出语句输出 0 吗，为什么?

```
struct CLS
{
int m_i;
CLS( int i ) : m_i(i) {}
CLS()
{
CLS(0);
}
};
CLS obj;
cout << obj.m_i << endl;
```

答：不能。在默认构造函数内部再调用带参的构造函数属用户行为而非编译器行为，亦即仅执行函数调用，而不会执行其后的初始化表达式。只有在生成对象时，初始化表达式才会随相应的构造函数一起调用。