C++期末考试题库

一、单选题:

1. 能作为C++程序的基本单位是（ C ）。

A. 字符　　B。 语句　　C。 函数　　D. 源程序文件

2. 程序中主函数的名字为（ A ）。

A. main　　B。 MAIN　　C. Main　　D。 任意标识符

3. 关于C++与C语言的关系的描述中，(D）是错误的。

A。 C语言是C++的一个子集；　　B。 C语言与C++是兼容的；

C。 C++对C语言进行了一些改进；　D。 C++和C语言都是面向对象的

4。 可用作C++语言用户标识符的一组标识符是（ B)。

A。 void　define　+WORD　　B。 a3_b3　_123　YN

C。 for　-abc　Case　　D。 2a　DO　sizeof

5. 存储以下数据，占用存储字节最多的是(D)。

A. 0　　B。 ‘0’　　C。 “0”　　D。 0。0

6. 设int　a=12；则执行完语句a+=a＊a;后，a的值是（ C ）。

A. 12　　B。 144　　C. 156　　D. 288

7。 假设在程序中 a、b、c 均被定义成整型,所赋的值都大于1，则下列能正确表示代数式$\frac{1}{abc}$的表达式是（ D ）。

A。 1。0/a＊b*c　　B。 1/（a＊b＊c)　　C. 1/a/b/(float)c　　D. 1。0/a/b/c

8. 以下说法中正确的是（B)。

A. C++程序总是从第一个定义的函数开始执行

B. C++程序总是从main函数开始执行

C。 C++函数必须有返回值，否则不能使用函数

D。 C++程序中有调用关系的所有函数必须放在同一个程序文件中

9. 下面有关构造函数的描述中，正确的是（B ）.

A。 构造函数可以带有返回值　　B。 构造函数的名字与类名完全相同

C. 构造函数必须带有参数　　D. 构造函数必须定义,不能缺省

10。10.在声明类时，下面的说法正确的是（　）。

A. 可以在类的声明中给数据成员赋初值

B。 数据成员的数据类型可以是register

C. private，public,protected可以按任意顺序出现

D. 没有用private,public，protected定义的数据成员是公有成员

11. 在下面有关析构函数特征的描述中，正确的是（ A ）。

A。 一个类可以有多个析构函数　　B。 析构函数与类名完全相同

C. 析构函数不能指定返回类型　　D. 析构函数可以有一个或多个参数

12。构造函数是在（B ）时被执行的.

A. 程序编译　　B。 创建对象　　C. 创建类　　D. 程序装入内存

13。 下面有关静态成员函数的描述中，正确的是（　B　）

A. 在静态成员函数中可以使用this指针

B。 在建立对象前,就可以为静态数据成员赋值

C。 静态成员函数在类外定义是,要用static前缀

D. 静态成员函数只能在类外定义

14。下面有关友员函数的描述中，真确的说法是（A ）

A. 友员函数是独立于当前类的外部函数

B. 一个友员函数不可以同时定义为两个类的友员函数

C. 友员函数必须在类的外部进行定义

D. 在类的外部定义友员函数时必须加上 friend 关键字

15。友员的作用之一是（ A ）

A。 提高程序的运行效率　　B。 加强类的封装

C。 实现数据的隐蔽性　　D. 增加成员函数的种类

16。使用派生类的主要原因是(A)

A。 提高代码的可重用性　　B。 提高程序的运行效率

C. 加强类的封装性　　D. 实现数据的隐藏

17。在 C++中继承方式有几中(C)

A。 1　　B。 2　　C。 3　　D. 4

18.假设已经定义好了类 student,现在要定义类 derived，它是从 student 私有派生的，则定义类 derived 的正确写法是（ D ）

A. class derived :: student private{ //…….}

B。 class derived :: student public { //…….}

C. class derived :: public student { //…….}

D。 class derived :: private student { //…….}

19.派生类的对象对它的基类成员中(C）是可以访问的。

A.公有继承的公有成员;　　B.公有继承的私有成员;

C.公有继承的保护成员;　　D.私有继承的公有成员。

20。 类中定义的成员默认为（A ）访问属性。

A. public　　B。 private　　C. protected　　D. friend

21。下列存储类标识符中,要求通过函数来实现一种不太复杂的功能，并且要求加快执行速度，选用（ A ）合适。

A。 内联函数;　B. 重载函数;　　C。 递归调用;　D. 嵌套调用。

22.C++对 C 语言作了很多改进，下列描述中(D）使得 C 语言发生了质变，从面向过程变成了面向对象.

A. 增加了一些新的运算符;　　B。 允许函数重载，并允许设置缺省参数;

C。 规定函数说明必须用原型;　　D. 引进了类和对象的概念;

23。按照标识符的要求，(A）符号不能组成标识符。

A. 连接符　　B. 下划线　　C。大小写字母　　D。 数字字符

24.下列变量名中,（ A ）是合法的。

A CHINA　　B byte—size　　C double　　D A+a

25.下列各种函数中，(C）不是类的成员函数。

A 构造函数　B 析构函数　　C 友元函数　　D 拷贝构造函数

26。下列(A ）是正确的语句。

A ;　　B a=17　　C x+y　　D cout〈〈"\n"

27.不能作为函数重载判断的依据的是(B)

A.返回类型　　B。 const　　C.参数个数　　D。 参数类型

28.通常，拷贝构造函数的参数是(C)

A.某个对象名　　B。某个对象的成员名

C.某个对象的引用名　　D。某个对象的指针名

29.下面对静态数据成员的描述中,正确的是（C ）

A 类的不同对象有不同的静态数据成员值

B 类的每个对象都有自己的静态数据成员

C 静态数据成员是类的所有对象共享的数据

D 静态数据成员不能通过类的对象调用

30。假定 AB 为一个类,则执行 AB x;语句时将自动调用该类的(B)

A。有参构造函数　B.无参构造函数　C.拷贝构造函数　D。赋值构造函数

31.C++程序从上机到得到结果的几个操作步骤依次是（　B　）.

A。 编译、编辑、连接、运行　　B。 编辑、编译、连接、运行

C. 编译、运行、编辑、连接　　D. 编辑、运行、编辑、连接

32.假定一条定义语句为“int a[10]， x， ＊p=a;”，若要把数组 a 中下标为 3 的元素值赋给 x，则不正确的语句为(A　)。

A。 x=p[3];　　B。 x=＊(a+3);　　C。 x=a［3］;　　D。 x=*p+3;

33。关于封装，下列说法中不正确的是(D　)。

A. 通过封装，对象的全部属性和操作结合在一起,形成一个整体

B. 通过封装,一个对象的实现细节被尽可能地隐藏起来（不可见）

C。 通过封装，每个对象都成为相对独立的实体

D。 通过封装,对象的属性都是不可见的

34。预处理命令在程序中都是以（ **B**)符号开头的。

A. ＊　　B。 #　　C. &　　D. @

35.存储以下数据，占用存储字节最少的是(B　)。

A。 0　　B。 ‘0’　　C。 “0”　　D。 0。0

36。程序运行中需要从键盘上输入多于一个数据时，各数据之间应使用(D)符号作为分隔符。

A. 空格或逗号　　B. 逗号或回车　　C. 逗号或分号　　D。 空格或回车

37。假定变量 m 定义为“int m=7;”，则定义变量 p 的正确语句为（ B)。

A。 int p=&m;　　B。 int *p=&m;　　C. int &p=＊m;　　D. int ＊p=m;

38。下面的哪个保留字不能作为函数的返回类型？ C　　。

A。 void　　B. int　　C。 new　　D。 long

39。采用重载函数的目的是（ B　）.

A.实现共享　　B.减少空间　　C.提高速度　　D。使用方便，提高可读性

40。假定 AB 为一个类，则(C　）为该类的拷贝构造函数的原型说明.

A。 AB（AB x);　　B. AB(int x）;　　C. AB（AB& x）;　　D。 void AB（AB & x）;

41、 C++对 C 语言作了很多改进，下列描述中（D　）使得 C 语言发生了质变，从面向过程变成了面向对象.

A、增加了一些新的运算符;　　B、允许函数重载,并允许设置缺省参数;

C、规定函数说明必须用原型;　D、引进了类和对象的概念;

42、 所谓数据封装就是将一组数据和与这组数据有关操作组装在一起,形成一个实体,这实体也就是（A）.

A.类　　B。对象　　C.函数体　　D.数据块

43、 关于 new 运算符的下列描述中，(C) 是错误的。
A、 它可以用来动态创建对象和对象数组；
B、使用它创建的对象或对象数组可以使用运算符 delete 删除；
C、使用它创建对象时要调用构造函数；
D、使用它创建对象数组时必须指定初始值；
44、 (D) 不是构造函数的特征。
A、构造函数的函数名与类名相同； B、构造函数可以重载；
C、构造函数可以设置缺省参数； D、构造函数必须指定类型说明。
45、 假定一个类的构造函数为 B(int x,int y){a=x——；b=a＊y——；},则执行 B x(3,5); 语句后，x.a 和 x。b 的值分别为（C ）
A、 3 和 5 B、 5 和 3 C、 3 和 15 D、20 和 5
46、 关于成员函数特征的下列描述中，(A)是错误的。
A、成员函数一定是内联函数； B、成员函数可以重载；
C、成员函数可以设置缺省参数值； D、成员函数可以是静态的；
47、 在公有继承的情况下，基类成员在派生类中的访问权限（B ）。
A、受限制 B、保持不变 C、受保护 D、不受保护
48、 友元的作用是（ A)。
A、提高程序的运用效率； B、加强类的封装性;
C、实现数据的隐藏性； D、增加成员函数的种类；
49、 在 C++中，关于下列设置缺省参数值的描述中，(B)是正确的。
A、不允许设置缺省参数值；
B、在指定了缺省值的参数右边,不能出现没有指定缺省值的参数；
C、只能在函数的定义性声明中指定参数的缺省值;
D、设置缺省参数值时，必须全部都设置；
50、关于 delete 运算符的下列描述中，(C）是错误的。
A、它必须用于 new 返回的指针；
B、使用它删除对象时要调用析构函数;
C、对一个指针可以使用多次该运算符；
D、指针名前只有一对方括号符号，不管所删除数组的维数。
51、const int *p 说明不能修改(C)。
A、p 指针； B、p 指针指向的变量；
C、p 指针指向的数据类型； D、上述 A、B、C 三者；
52、当需要打开 A 盘上的以 xxk。dat 文件用于输入时，则定义文件流对象的语句为 （B)。
A、fstream fin (“A:xxk.dat”, 1) B、ifstream fin(“A:xxk.dat”, ios::nocreate)
C、ofstream fin(“A：xxk.dat”) D、ifstream fin(“A：xxk。dat”, ios:：app)
53、派生类的对象对它的基类成员中（A ）是可以访问的。
A、公有继承的公有成员； B、公有继承的私有成员;
C、公有继承的保护成员； D、私有继承的公有成员；
54、假定一个字符串的长度为 n，则定义存储该字符串的字符数组的长度至少为（C)。
A、n-1 B、n C、 n+1 D、n+2
55、 在 int a=3; *p=&a；中，*p 的值是(D)。
A、变量 a 的地址值 B、无意义 C、变量 p 的地址值 D、 3
56、 下列常量中,（ D）不是字符常量.
A、’\005’ B、’\n’ C、 ’ c’ D、“a”

57、在 int a=3，＊p=＆a;中,*p 的值是（ D)。
A)变量 a 的地址值 B）无意义 C）变量 p 的地址值 D） 3
58、以下 4 个选项中，不能看作一条语句的是（B ）。
A) if(b==0) m=1; n=2; B） a=0,b=0,c=0; C） if(a>0); D) {;}
59、(D)不是构造函数的特征。
A)构造函数的函数名与类名相同 B)构造函数可以重载
C）构造函数可以设置缺省参数 D）构造函数必须指定类型说明
60、以下程序段中与语句 k=a〉b? 1：0；功能等价的是（D)。
A) if（a〉b) k=1; B） if(a>b) k=0
C)if（a>b) k=1; D) if（a〈b) k=0;else k=0; else k=1;
61、下列常量中,（ D)不是字符常量.
A)’\005’ B)’\n’ C） ’c’ D)″a″
62、表示“x 大于 1 而小于 10”的逻辑表达式是（B)。
A）1<x<10 B）x>1&&x<10 C） !(x<=1||x〉=10) D)x〉1||x〈10
63、关于成员函数特征的下列描述中，(A)是错误的.
A)成员函数一定是内联函数 B）成员函数可以重载
C)成员函数可以设置缺省参数值 D）成员函数可以是静态的
64、有以下程序
#include 〈iostream.h〉
void main（ ）
{ int i，s=0;
for(i=1;i<10; i+=2) s+=i;
cout〈<s;}
程序执行后的输出结果是（C)。
A） 自然数 1～9 的累加和 B) 自然数 1~10 的累加和
C） 自然数 1～9 中的奇数之和 D） 自然数 1～10 中的偶数之和
65、设有定义:int n=0,＊p=＆n，＊＊q=&p;则以下选项中，正确的赋值语句是（D)。
A) p=1; B） ＊q=2; C) q=p; D) *p=5;
66、(A)能正确输出结果：C++。
A） char s[]=”C++″;cout<<s<<endl;
B）char s[3]=″C++”;cout〈〈s〈<endl;
C）char s[3]={’C’,’+’,’+’,’\0’};cout<<s〈〈endl;
D）char s[3]={’C’,’+’,’+’};cout<〈s〈<endl;
67、有以下程序段
int a[10]={1,2,3，4,5，6，7,8，9,10},*p=a，b;
b=p[5];
b 中的值是（ B)。
A）5 B）6 C） 8 D） 9
68、有以下程序
#include <iostream。h〉
void main（ ）
{ char *p[10]={”abc”，”aabdfg″,″dcdbe”,″abbd”,”cd”};
cout<〈p[3]〈<endl;

}
执行后输出结果是 B
A)dcdbe B） abbd C） abc D） abb
69、下列静态数据成员的特性中，(C) 是错误的.
A） 说明静态数据成员时前边要加修饰符 static
B） 静态数据成员要在类体外进行初始化
C） 静态数据成员不是所有对象所共用的
D)引用静态数据成员时,要在其名称前加〈类名>和作用域运算符
70、有以下语句
struct S
{ int g;
char h; } T;
则下面叙述中不正确的是(C) .
A）S 是结构体名 B）该结构体包含 2 个成员
C）S 是 struct 类型的变量 D）T 是 struct S 类型的变量
71、派生类的对象对它的基类成员中（ A）是可以访问的.
A）公有继承的公有成员
B）公有继承的私有成员
C）公有继承的保护成员
D）私有继承的公有成员
72、实现运行时的多态性用 。D
A、重载函数 B、构造函数 C、析构函数 D、虚函数
73. 下列变量名中，(A)是合法的。
A. CHINA B。 byte—size C. double D. A+a
74. 在 int b[] [3] = {{1} ,{3,2}, {4, 5,6}, {0}}; b [2] [2] 的值是（D)。
A。 0 B。 2 C。 5 D. 6
75。 下列各运算符中,（B）优先级最高.
A。 +（双目） B。 *（单目） C。 <= D。 *=
76. 下列 for 循环的次数为 (B)。
for(int i(0), x=0;!x&＆i<=5;i++)
A。 5 B. 6 C 。 1 D. 无限
77. 下述关于 break 语句的描述中，(C)是正确的。
A. break 语句可用于循环体内，它将退出该重循环。
B. break 语句可用于开关语句中，它将退出开关语句。
C。 break 语句可用于 if 体内,它将退出 if 语句。
D. break 语句在一个循环体内可以出现多次。
78。 在一个被调用函数中，关于 return 语句使用的描述，(D)是错误的。
A。 被调用函数中可以不用 return 语句。
B. 被调用函数中可以使用多个 return 语句。
C。 被调用函数中,如果有返回值，就一定要有 return 语句。
D. 被调用函数中，一个 return 语句可返回多个值给调用函数.
79。 在 C++中，关于下列设置参数默认的描述中，(C) 是正确的.
A。 不允许设置参数的默认值。
B。 设置参数默认值只能在定义函数时设置.

C. 设置参数默认值时,应该是先设置右边的再设置左边的。
D。 设置参数默认值时，应该全部参数都设置.
80。 下列存储类标识符中，要求通过函数来实现一种不太复杂的功能,并且要求加快执行速度，选用(A) 合适。
A。 内联函数 B。 重载函数 C. 递归调用 D. 嵌套调用
81。 下列的各类函数中，(C) 不是类的成员函数.
A. 构造函数 B. 析构函数 C . 友元函数 D。 拷贝初始化构造函数
82. (D)不是构造函数的特征
A。 构造函数的函数名与类名相同 B. 构造函数可以重载
C. 构造函数可以设置缺省参数 D。 构造函数必须指定类型说明
83. f1(int)是类 A 的公有成员函数,p 是指向成员函数 f1() 的指针，采用 (C) 是正确的。
A. p=f1 B。 p=A:: f1 C。 p=A:: f1() D。 p=f1 ()
84. 下列定义中, (A) 是定义指向数组的指针 p。
A. int *p[5] B。 int (*p)[5] C。 (int *)p[5] D. int *p[ ]
85. 循环 while(int i=0) i-—; 执行次数是(A)。
A。 0 B. 1 C 。 5 D . 无限
86。 设 int x; ,则经过 (C) 后，语句*px=0; 可将 x 值置为 0。
A。 int * px; B . int const * px=&x;
C. int * const px=&x; D. const int * px=&x;
87。 设 void f1 (int *m, long & n);int a; long b; 则以下调用合法的是 (B)。
A。 f1(a, b) ; B. f1(&a, b);
C。 f1(a, &b); D. f1(&a, &b)
88. 关于对象概念的描述中，(A) 是错误的。
A. 对象就是 C 语言中的结构变量;
B. 对象代表着正在创建的系统中的一个实体;
C。 对象是一个状态和操作(或方法)的封装体;
D. 对象之间的信息传递是通过消息进行的;
89. 在下列 double 型常量表示中，(A) 是错误的。
A. E15; B。 。35; C. 3E5; D。 3E—5
90. 下列给字符数组进行初始化中，(A) 是正确的。
A. char s1[ ]=”12345abcd″; B. char s2 [3]=” xyz″;
C. char s3[][3]={ ‘a’ ,’x’ ,’ y’ }; D。 char s4 [2 [3] ={″xyz″,″mnp”};
91。 对于 int *pa [5] ;的描述, (D)是正确的。
A。 pa 是一个指向数组的指针，所指向的数组是 5 个 int 型元素;
B。 pa 是一个指向某个数组中第 5 个元素的指针，该元素是 int 型变量;
C。 pa [5] 表示某个数组的第 5 个元素的值;
D. pa 是一个具有 5 个元素的指针数组，每个元素是一个 int 型指针;
92。 下列各运算符中，(A) 优先级最低。
A. ? : B。 | C. | | D。 ! =
93. 下列 for 循环的循环体执行次数为 (D)。
for(int i(0), j (10);i=j=4;i++,j- —)
A. 0; B。 1; C。 4; D. 无限;
94. 下述关于开关语句的描述中，(A) 是正确的。

A。 开关语句中 default 子句可以没有，也可以有一个;
B. 开关语句中每个语句序列中必须有 break 语句;
C。 开关语句中 default 子句只能放在最后;
D。 开关语句中 case 子句后面的表达式可以是整形表达式。
95。 下列存储类标识符中,（ C ）的可见性与存在性不一值。
A. 外部类; B. 自动类; C。 内部静态类; D. 寄存器类。
96. 下述静态数据成员的特征中,（ D）是错误的。
A。 说明静态数据成员时前边要加修饰符 static;
B。 静态数据成员要在类体外进行初始化;
C。 引用静态数据成员时，要在静态数据成员名前加〈类名〉和作用域运算符;
D。 静态数据成员不是所有对象所共用的.
97. （ A)是析构函数的特征。
A。 一个类中只能定义一个析构函数; B。 析构函数与类名不同;
C. 析构函数的定义只能在类体内; D。 析构函数可以有各个或多个参数。;
98。 已知：p 是一个指向类 A 数据成员 m 的指针，A1 是类 A 的一个对象。如果要给 m 赋值为 5,（ C ）是正确的。
A。 A1.p=5; B。 A1-〉p=5;
C. A.*p=5; D. ＊A1。p=5;
99. 关于 new 运算符的下列描述中,（ D)是错的。
A。 它可以用来动态创建对象和对象数组;
B。 使用它创建的对象或对象数组可以使用运算符 delete 删除;
C. 使用它创建对象时要调用构造函数;
D. 使用它创建对象数组时必须指定初始值。
100. 派生类的构造函数的成员初始化列中，不能包含（C ）.
A。 基类的构造函数;
B。 派生类中子对象的初始化;
C。 基类的子对象初始化;
D. 派生类中一般数据成员的初始化
101。 派生类的对象对它的基类成员中（ A)是可以访问的.
A. 公有继承的公有成员;
B. 公有继承的私有成员;
C. 公有继承的保护成员;
D. 私有继承的公有成员。
102. C++类体系中，不能被派生类继承的有（ A）.
A.构造函数 B。虚函数 C。静态成员函数 D.赋值操作函数
103. 下面标识符中正确的是（A ）。
A. _abc B. 3ab C. int D. +ab
104. 下列哪项（ D ）不是面向对象程序设计的主要特征?
a。封装 b。继承 c。多态 d.结构
105. 若有以下定义，则说法错误的是(D)。
int a=100,＊p=&a ;
A. 声明变量 p,其中＊表示 p 是一个指针变量
B. 变量 p 经初始化，获得变量 a 的地址
C. 变量 p 只可以指向一个整形变量

D．变量 p 的值为 100

106．对数组名作函数的参数,下面描述正确的是(B)。

A．数组名作函数的参数，调用时将实参数组复制给形参数组。

B．数组名作函数的参数，主调函数和被调函数共用一段存储单元。

C．数组名作参数时，形参定义的数组长度不能省略。

D．数组名作参数，不能改变主调函数中的数据。

107．关于静态成员的描述中，(　B　　)是错误的。

A.　静态成员可分为静态数据成员和静态成员函数。

B.　静态数据成员定义后必须在类体内进行初始化。

C.　静态数据成员初始化不使用其构造函数。

D.　静态数据成员函数中不能直接引用非静态成员。

108．下列关于构造函数的描述中，错误的是(　D).

A．构造函数可以设置默认的参数　B．构造函数在定义类对象的自动执行

C．构造函数可以是内联函数　　D．构造函数不可以重载

109．假设 OneClass 为一个类，则该类的拷贝初始化构造函数的声明语句为（ C ）。

A. OneClass (OneClass p);　　B. OneClass& (OneClass p);

C. OneClass(OneClass & p);　　D。 OneClass (OneClass *p);

110．如果类 A 被说明成类 B 的友元，则（ D　）。

A。类 A 的成员即类 B 的成员。

B。类 B 的成员即类 A 的成员。

C。类 A 的成员函数不得访问类 B 的成员　.

D.类 B 不一定是类 A 的友元。

111、关于对象和类的关系,说法正确的是（ C　）。

A） 同属于一类的对象，具有相同的数据成员和成员函数。

B) 对象是具体，是类的对象，同其他变量一样，先定义后使用。

C） 同一类的不同对象，其具有的操作可不同，具体的操作也不同.

D) 不同类的对象，可有相同的操作.

112、下列关于 C++函数的说明中正确的是（ D　）。

A) 内联函数就是定义在另一函数体内部的函数。

B） 函数体的最后一条语句必须是 RETURN 语句。

C） 标准 C++要求在调用一个函数之前，必须先声明其原型.

D) 编译器会根据函数的返回值类型和参数表来区分函数的不同重载形式。

113、下列关于虚函数的说明中，正确的是（　B ）。

A) 从虚基类继承的函数都是虚函数。

B） 虚函数不得是静态成员函数。

C) 只能通过指针和引用调用虚函数.

D） 抽象类中的成员函数都是虚函数.

114、下列符号中可以用作 C++标识符的是（ A　）。

A) _radius　　B） foo～bar　C) else　　D）　3room

115、下列语句中错误的是（　B ）。

A） const int buffer=256;　　B） const int int temp;

C） const double *point;　　D) const double *rt=new double(5。5)

116、下列关于实参和形参说法错误的是（　D ）.

A)实参可以是变量、常量、或表达式。

B）实参与形参的类型必须一致,否则会发生“类型不匹配"的错误。
C）实参对形参的数据传递是单向传递，调用结束后,实参单元被释放。
D)形参必须是变量 。
117、关于内联函数说法错误的是（ C ）。
A）不是任何一个函数都可定义成内联函数。
B）内联函数的函数体内不能含有复杂的结构控制语句。
C)递归函数可以被用来作为内联函数.
D）内联函数一般适合于只有 1～5 行语句的小函数。
118、关于保护继承的说法正确的是（ D ）.
A）基类的公有成员、私有成员可被子类继承下来，而且性质不变。
B）基类的公有成员、私有成员可被子类继承下来，而且性质改变为保护成员。
C）基类的公有成员、私有成员可被子类继承下来，而且性质均改变为私有成员。
D）基类的公有成员、私有成员可被子类继承下来，性质不变，私有成员不被继承。
119、关于函数的返回值说法正确的是（ A ）。
A）由 return 语句返回时，只带回一值,其类型在函数定义时确定.
B）其类型由调用表达式决定。
C)函数可以没有返回值，这时在函数定义，函数的类型说明就没必要了。
D)函数调用就要有返回值，否则调用就没意义了。
120、关于结构化程序设计方法说法错误的是（ D ）。
A）在数据处理过程中，采用的是自顶向下、分而治之的方法。
B）将整个程序按功能划分为几个可独立编程的子过程模块。
C）以“对象”或“数据”为中心.
D）数据和处理数据的过程代码是分离的、相互独立的实体。
121、运算符重载是对已有的运算符赋予多重含义，因此（ C ）。
A）可以对基本类型（如 int 类型）的数据，重新定义“+”运算符的含义。
B）可以改变一个已有运算符的优先级和操作数个数。
C）只能重载 C++中已经有的运算符,不能定义新运算符。
D）C++中已经有的所有运算符都可以重载.
122、关于 C++程序说法不正确的是(D)。
A) C++程序由函数构成,但只有一个 main ()函数 。
B) C++程序中 main()函数可以在程序的任何位置 。
C) C++程序由 main（）函数开始执行,由 main()结束执行。
D) main()都是没有参数的。
123、下面有关重载函数的说法中正确的是(C)。
A）重载函数必须具有不同的返回值类型。
B)重载函数形参个数必须不同。
C)重载函数必须有不同的形参列表。
D）重载函数名可以不同。

二、填空：

1。 C++语言的头文件与源程序文件扩展名分别是(。h ）和（ .cpp)。
2。 在 C++程序中使用基本输入与输出流时需要包含的头文件名是（ iostr eam ）.
3. 在 C++语言中，惟一的一个三目运算运算符是（ ？： ）.
4。 C++中当一个函数无返回值时,则函数的类型是(void)。

5。 一个类中可以有（多）个构造函数，只能有（ 一 ）个析构函数。

6。一般情况下,按照面向对象的要求,把类中的数据成员（属性）定义为（ 私有 ）权限，

而把成员函数(方法)定义为（公有)权限.

7.在定义类的对象时,C++程序将自动调用该对象的(构造)函数初始化对象自身.在撤销类的对象时，C++程序将自动调用该对象的（ 析构 ）函数.

8.类继承中,缺省的继承方式是（ 私有继承 ）.

9．在C++语言中，用转义字符序列（ ’\n’ ）或操纵符（ endl ）表示输出一个换行符。

10．表达式a=a＋1表示成增量表达式为（ ++a)。

11．当不需要函数返回任何值时，则应把该函数类型定义为（ void)。

12．用于输出表达式值的标准输出流对象是（ cout)。，用于从键盘上为变量输人值的标准输人流对象是(cin)。

13、变量分为全局和局部两种，(全局变量默认初始化为0)。变量没有赋初值时，其值是不确定的。

14、假定类AB中有一个公用属性的静态数据成员bb，在类外不通过对象名访问该成员bb的写法为（ AB::bb)。

15、类的成员分为__数据成员__和__成员函数__。

16、一个__类__和__对象__的关系,正如基本数据类型与该类型的变量一样，如int x;。

17、对一个类中的数据成员的初始化可以通过构造函数中的__赋值__实现,也可以通过构造函数中的__初始化列表__实现。

18、类有两种用法：一种是__定义对象__，即生成类的对象；另一种是通过__派生__，派生出新的类。

19、C++语言提供的__多态__机制允许一个派生类继承多个基类，即使这些基类是相互无关的。

20、声明虚函数的方法是在基类中的成员函数原型前加上关键字__virtual__。

21、如果一个类中有一个或多个纯虚函数，则这个类称为__虚基类__.

22、静态数据成员在定义或说明时，前面要加上关键字__static__.

23、如果成员函数不需要访问类的__非静态__成员,则可以把它声明为静态成员函数。

24、友元可以是__全局函数__，也可以是__其他类的成员函数__。

25、若需要把一个函数“void fun ();”定义为一个类A的友元函数,则应在类A的定义中加入一条语句:__friend void fun ()；__。

26、运算符函数中的关键字是__operator__,它和__运算符__一起组成该运算符函数的函数名。

27、类中运算符重载一般采用两种形式：__成员函数__和__友元函数__。

28、面向对象程序设计的3大机制为：__封装性__、__继承性__和__多态性__。2、类的访问权限有__public__、__private__和__protected__三种.

29、构造函数是__对象__被创建时自动执行，对象消失时自动执行的成员函数称为__析构函数__。

30、如果类A继承了类B，则类A称为__基类__，类B称为__派生类__.

31、如果一个特定的类型S当且仅当它提供了类型T的行为时，则称类型S是类型T的____子类型__.

32、在类定义中,将__=0__置于虚函数的函数原型的末尾可以声明该函数为纯虚函数。

33、类的静态成员分为__静态数据成员__和__静态成员函数__。

34、友元函数的声明可以放在类的__私有__部分，也可以放在类的__共有__部分,它们是没有区别的。
35、如果说类 B 是类 A 的友元类，则类 B 的所有成员函数都是类 A 的__成员函数__。
36、设 a 和 b 是两个整型变量，我们用 a+b 的形式求这两个变量的和；设 c 和 d 为浮点型变量，我们用 c+d 的形式求这两个变量的和.显然运算符“+”具有不同的用途，这是__运算符重载__的例子.
37、对于双目运算符，若重载为类的成员函数，有 1__个参数；若重载为友元函数，则有__2 个参数.
38、 当建立__一个新的对象___，程序自动调用该类的构造函数。
39、 在 C++中有二种参数传递方式:__传值___ 和___引用__ 。
40、模板对处理数据的类型的要求不同可以分为两种类型：__函数模板__ 和 __ 类模板__。
41、 异常是通过_检查（try)__ 、_抛出（throw）__和__捕捉(catch）__ 来实现的。
42、 虚函数实现了 C++的__多态__ 机制，类实现了 C++的___封装__ 机制。
43、 面向对象程序设计的___继承___机制提供了重复利用程序资源的一种途径。
44、C++语言程序设计的三种基本结构是:_顺序结构_ 、 _选择结构_ 、 __循环结构__ .
45、为了避免嵌套的条件语句 if-else 的二义性，C++语言规定 else 与__if __配对使用。
46、定义函数时，在函数的类型前加上修饰词 ___ inline ___，指明将该函数定义为内联函数.
47、有说明语句：＊p；则＊p++运算首先访问 ___＊p___,然后使__p____ 的值加 1.
48、执行 int *p=new int 操作，可以得到一个动态分配整型对象的__指针____。
49、有说明语句：int *p;则（＊p）++运算首先访问 ___＊p___，然后使 __＊p____的值加 1。
50. C++目标程序经___编译链接___后生成扩展名为 exe 的可执行程序文件。
51. 16 位计算机中整型占___2_____个字节存储空间。
52. 要在屏幕上显示"Hello,world!"应执行语句 cout〈〈_____"Hello,world!"；_____。
53. 表达式 8&3 的结果是___0_____.
54. 作为语句的函数调用是通过____函数的副作用____来体现的.
55. 执行 if (x〉=0||x<=0) cout〈<”abcd"; else cout〈<”wxyz”;屏幕上显示的是____abcd____。
56. 设一维整型数组 data 已进行了初始化，则其元素个数可由操作符 sizeof 通过表达式____ sizeof(data) / sizeof(int) ___计算出来.
57. 若一全局变量只允许本程序文件中的函数使用，则定义它时应加上修饰符__ static _____。
58. 设有 int w[3] [4];，pw 是与数组名 w 等价的数组指针，则 pw 应初始化为____ int ＊[4] pw = w; _____。
59。要使引用 pr 代表变量 char＊p;,则 pr 应初始化为_____char*& pr=p; _____。
60. 在 C++中封装性、继承性和_____多态性
61. 假设 int a=1，b＝2;则表达式 a+++--b 的值为 __2____。
62. C++语言支持的两种多态性分别是编译时的多态性和 __运行时___的多态性。
63. 设有如下程序结构：

```
class Box
 { … };
void main()
 {Box A，B,C; }
```

该程序运行时调用___3__次构造函数；调用 ___3__次析构函数。
64. 目前，有两种重要的程序设计方法,分别是：___面向过程__和___面向对象___。
65.函数重载时要求同名函数的_参数个数__或_参数类型__不同,否则无法确定是哪个函数。
66. 静态数据成员是类的所有对象中的____共享__成员，而非静态数据成员是属于___一个具体___对象的。
67。设 A 为 test 类的对象且赋有初值,则语句 test B(A) ; 表示__将对象 A 复制给对象 B ___.
68.面向对象程序设计方法中的每个对象都具有_属性__和___方法__两方面的特征。
69。类的静态成员是_的所有对象中共享_的成员,不是_某个对象___的成员，静态成员在___类的声明中__进行引用性说明，在_其文件作用域的其他地方_进行定义性说明。
70。“封装”指的是_将有关的数据和操作代码封装在一个对象中,形成一个基本单位，各个对象之间相互独立，互不干扰，且将对象中某些部分对外隐藏_。
71。可以让新类继承已定义的类的_数据成员_和_成员函数_，这个新类称为__继承_，原来的类称为___基类__。新类可以从一个类中派生，这叫__单继承__，也可以从多个类中派生，称为__多重继承_。
72.指针类型变量用于存储__另一变量的地址__，在内存中它占有____一个__存储单元。
73。类有三种继承方式，分别是：__公有继承_、__保护继承__、___私有继承_.

四、写出程序运行结果

1、#include 〈iostream。h〉
```
int a[] ={2, 4,6,8,10} ;
int &index (int i)
 {   return a [i];
}
void main ()
{
    int i;
    index (3) =12;
    for (i=0;i<=4; i++)
    cout< 〈a [i] 〈<" ";
}
```
结果 2 4 6 12 10

2、#include 〈iostream.h〉
```
void f(int *m, int n)
 {
    int temp;
    temp=*m;
    *m=n;
    n=temp;
}
void main()
{
```

```
    int a=5,b=10;
    f(&a,b);
    cout 〈<a<<”  ” 〈〈b 〈<endl;
}
10 10

3、＃include <iostream.h>
int i=15;
void main ()
{
    int i;
    i=100;
    ::i=i+1;
    cout 〈<::i< 〈endl;
}
101
4、#include  〈iostream。h>
class toy
{
public:
    toy(int q, int p)
     {
        quan = q;
        price = p;
    }
    int get_quan ()
     {
        return quan;
    }
    int get_price()
     {
        return price;
    }
private:
    int quan, price;
} ;
main ()
 {
    toy op [3][2]={
    toy (10,20), toy(30,48),
    toy (50, 68), toy(70,80),
    toy (90,16), toy (11,120),
} ;
    int i;
```

```
    for (i=0; i<3; i++)
    {
         cout< <op[i][0]。get_quan()<<",";
         cout<<op [i] [0] .get_price() <<" \n" ;
         cout<<op [i] [1] .get_quan () <<", ";
         cout< <op [i] [1]。get_price() <<"\n";
    }
    cout<<" \n";
    return 0;
}
10，20
30,48
50,68
70,80
90，16
11，120

5、#include  <iostream。h>
class example
 {
public:
    example(int n)
      {
        i=n;
        cout< <"Constructing\n ";
    }
    ~example ()
     {
        cout <<" Destructing\n";
    }
    int get_i()
     {
        return i;
    }
private:
    int i;
};
int sqr_it(example o)
{
    return o.get_i() * o.get_i() ;
}
main ()
{
    example x (10) ;
```

```
    cout< <x。get_i() < <endl;
    cout<<sqr_it (x)<<endl;
    return 0;
}
Constructing
10
Destructing
100
10
Destructing
```

6、#include <iostream.h>

```
   class Test
   {
   private:
       int x;
   public:
       Test()
       {
cout<<“构造函数被执行”<<endl;
         x=0;
       }
~Test () {cout<<“析构函数被执行”<<endl;
       void print()  {cout <<“x=” <<x<<endl;  }
   } ;
   void main()
    {
       Test obj1, obj2;
obj1。print () ;
obj2.print ();
   }
构造函数被执行
构造函数被执行
x=0
x=0
析构函数被执行
析构函数被执行
```

7、＃include <iostream.h>

```
   class A
    {
   public:
      A(int *s)  {cout< <s <<endl;}
   };
```

```
   class B:public A
   {
   public:
      B(char *s1, char *s2) :A (s1)
       {
          cout<<s2<<endl;
       }
   };
   class C:public A
    {
   public:
      C(char *s1,char *s2):A(s1)
      {
           cout <<s2 <<endl;
      }
   };
   class D:public B,C
   {
   public:
      D(char *s1, char *s2, char *s3,char *s4) :B(s1,s2), C (s3, s4)
      {
           cout <<s4< <endl;
      }
   };
   void main()
    {
      D d  ( “class A”,"class B”,” class C",” class D") ;
   }
class A
class B
class C
class D
class D
```

8、#include <iostream。h>

```
   class Base
    {
   public:
      virtual void disp ()  {cout <<” base class”<<endl;}
   };
   class Derivel:public Base
   {
   public:
      void disp() {cout< <” derivel class”  <<endl;}
```

```
};
class Derive2:public Base
{
public:
   void disp ()  {cout 〈〈” derive2 class”<<endl;}
} ;
void main()
{
   Base *p;
   Base b;
   Derive1 d1;
   Derive2 d2;
   p=&b;
   p—〉 disp () ;
   p=&d1;
   p—〉 disp();
   p=&d2;
   p—〉 disp();
}
base class
derivel class
derive2 class
```

9、#include 〈iostream.h〉

```
 class Sample
{
private:
   int x;
   static int y;
public:
   Sample (int a);
   void print ();
} ;
Sample:: Sample(int a)
 {
    x=a;
    y ++;
}
void Sample:: print ()
 {
    cout< 〈” x=” < 〈x 〈<” , y=″ 〈〈y 〈<endl;
}
int Sample:: y=25;
void main ()
```

```
    {
        Sample s1 (5) ;
        Sample s2(10);
        s1.print();
        s2.print() ;
    }
x=5, y=27
x=10, y=27

10、#include <iostream。h>
    class Sample
    {
    private:
        int x;
    public:
        Sample  () { }
        Sample(int a)  {x=a; }
        void disp()  {cout <<" x=" < <x <<endl;}
        friend Sample operator+( Sample &s1, Sample &s2);
    } ;
    Sample operator+ ( Sample &s1,  Sample &s2)
     { return Sample (s1.x+s2.x) ;}
    void main ()
    {
        Sample obj1(10);
        Sample obj2(20);
        Sample obj3;
        obj3=obj1+obj2;
        obj3。disp () ;
    }
x=30

11、#include  <iostream。h>
    class Test
     {
    private:
        int x;
    public:
        Test()
         {
cout <<"构造函数被执行"  <<endl;
          x=0;
        }
        void print()
```

```
        {
          cout 〈〈“x=” 〈〈x〈 〈endl;
        }
   };
   void main()
   {
       Test obj1, obj2;
obj1。print ();
obj2。print ();
   }
```

构造函数被执行
构造函数被执行
x=0
x=0

12、＃include 〈iostream.h〉

```
   class A
   {
   protected:
      int x;
   public:
      A(int x)
      {
         A::x=x;
         cout〈〈”class A”〈 〈endl;
      }
   };
   class B
   {
   private:
      A a1;
   public:
      B (int x) :a1 (x)
      {
          cout 〈〈”class B”〈 〈endl;
      }
   };
   class C: public B
   {
   private:
      A a2;
   public:
      C(int x) :B(x),a2(x)
      {
```

```
            cout 〈〈”class C″〈〈endl;
        }
    };
    class D:public C
     {
    public:
        D(int x)：C(x)
         {
            cout 〈<”class D”<<endl;
        }
    };
    void main()
     {
        D dobj（10);
    }
class A
class B
class A
class C
class D
```

13、

```
#include 〈iostream.h〉
    class Point
     {
    private:
        int x;
        int y;
    public:
        Point (int a, int b)
        {
            x=a;
            y=b;
        }
        virtual int area()  {return 0;}
    };
    class Rectangle:public Point
     {
    private:
        int length;
        int width;
    public:
        Rectangle（int a, int b, int l, int w）: Point(a,b)
        {
            length=l;
```

```
        width=w;
    }
    virtual int area()  { return length*width;}
};
void disp (Point &p)
{
    cout<<"面积是:"<<p.area()<<endl;
}
void main ()
{
    Rectangle rect (3,5, 7, 9);
    Disp(rect) ;
}
```

面积是：63

14、#include <iostream.h>

```
 class Sample
 {
private:
    int x;
    static int y;
public:
    Sample (int a);
    void print() ;
};
Sample::  Sample (int a)
 {
     x=a;
     y=x++;
}
void Sample: :print ()
 {cout<<"x="<<x<<",y="<<y<<endl;}
int Sample: :y=25;
void main ()
 {
     Sample s1(5);
     Sample s2 (10);
     s1.print() ;
     s2.print ();
}
```

x=6,y=10

x=11,y=10

15、#include <iostream.h>

```
   class Sample
   {
   private:
      int x;
   public:
      Sample () {}
      Sample  (int a)
      {
         x=a;
      }
      void disp ()  {cout<<" x=" <<x<<endl;}
      Sample operator+( Sample &s);
   };
   Sample Sample::  operator+( Sample &s)
   {
      return Sample (x+s.x);
   }
   void main()
   {
       Sample obj1 (20) ;
       Sample obj2(20);
       Sample obj3;
obj3=obj1+obj2;
       obj3.disp();
   }
x=40
```

```
16、# include <iostream。h>
class A  {
int a , b ;
public :
A ( )  { a=b=0;  }
A ( int aa , int bb )  :  a(aa),b(bb)  {
cout < <" a=" <<a<<" , " <<" b=" <<b< <endl;
}
~A ( ) {cout  <<" Destructor! " < <endl; }
};
void main () {
A x ,  y (2 , 3);
}
 a=2, b=3
Destructor!
Destructor!
```

```
17、 #include <iostream。h>
class R
{
public:
R(int r1,int r2)  {R1=r1; R2=r2;}
void print() ;
void print() const;
private:
int R1,R2;
} ;
void R:: print()
 {
cout <<R1 <<”: ” <<R2<<endl;
}
void R::print () const
 {
cout <<R1<<”; ”<<R2< <endl;
}
void main ()
{
   R a (5, 4) ;
a。print ();
const R b (20,52) ;
b.print();
}
5: 4
20; 52

18、#include <iostream。h>
class A
 {
public:
virtual void act1 ();
void act2 ()  {act1 (); }
};
void A:: act1()
{
cout <<”A:: act1 () called. ″ <<endl;
}
class B : public A
{
public:
void act1 () ;
};
```

```
void B:: act1()
{
cout 〈〈”B::act1()  called。 ”<<endl;
}
void main()
 {
B b;
b.act2 ();
}
     # include  〈iostream.h〉
       void fun (int, int, int * );
       void main ()
       {
     int x, y,z;
         fun (2,3, &x);
         fun (4, x, &y);
         fun (x, y,&z) ;
         cout 〈〈x 〈〈', ' 〈〈y 〈〈',' < 〈z< 〈endl;
        }
        void fun (int a, int b,int  *  c )
        { b*=a; *c=b-a;}
B:: act1 () called.

19、#include  〈iostream.h〉
class AA
 {  public:
     AA(int i, int j)
            {A=i;B=j;cout 〈〈”Constructor\n” ;}
         AA (AA &obj)
            {A=obj.A+1; B=obj.B+2;cout 〈〈"Copy_Constructor\n"; }
         ～AA ()
           {cout 〈〈”Destructor\n"; }
        void print()
         {cout< 〈”A="< 〈A< 〈", B=”〈<B<<endl;  }
  private:
      int A, B;
};
void main ()
 {   AA a1(2, 3);
    AA a2(a1);
     a2。print();
     AA *pa=new AA(5,6) ;
     pa—>print ();
   delete pa;
```

```
}
Constructor
Copy_Constructor
A=3,B=5
Constructor
A=5,B=6
Destructor
Destructor
Destructor
```

20、＃include 〈iostream.h〉

```
void ff(int x), ff (double x) ;
void main ()
{   float a (88。18);
ff (a);
char b(' a');
ff (b) ;
 }
void ff(int x)
{   cout 〈〈" ff(int): "<<x< 〈endl;  }
void ff(double x)
{   cout 〈<"ff (double): "  〈<x 〈<endl;}
```

ff(double) : 88。18
ff (int) : 97

21、＃include 〈iostream。h〉

```
class AA
 {   public:
      AA (int i, int j)
          {A=i;B=j;cout 〈〈"Constructor\n"; }
      AA (AA &obj)
          {A=obj.A+1; B=obj.B+2; cout 〈〈" Copy_Constructor\n" ;}
      ~AA ()
         {cout 〈<" Destructor\n"; }
      void print()
       {cout 〈〈"A=" 〈<A 〈〈",B=" <<B 〈〈endl; }
   private:
      int A,B;
};
void main()
 {   AA a1(2, 3);
    AA a2(a1);
    a2.print ();
```

```
    AA *pa=new AA(5,6) ;
    pa->print() ;
    delete pa;
  }
Constructor
Copy_Constructor
A=3,B=5
Constructor
A=5,B=6
Destructor
Destructor
Destructor
```

22、#include <iostream.h>

```
void  Swap ( int &a,  int & b);
void main()
{
    int x ( 10 ), y( 7 ) ;
    cout<<" x=" <<x<<"  y=" <<y<<endl;
    Swap( x ,  y ) ;
    cout <<"x="<<x<<"  y=" <<y <<endl;
}

void Swap(int & a, int & b)
{   int  temp;  temp = a ;  a=b ;   b=temp ;  }
x=10  y=7
x=7  y=10
```

23、#include <iostream。h>

```
class A
 {
public:
    A();
    A(int i, int j);
    ~A () {cout <<" Donstructor。\n"; }
    void print();
private:
    int a,b;
};
A: :A()
 {  a=b=10; cout <<" Default constructor.\n" ;}
A:: A (int i,int j)
{ a=i,b=j;cout<<" Constructor。\n" ;}
void A:: print()
```

```
 {cout 〈〈” a=” <<a 〈〈",b="< 〈b 〈<endl;}
void main ()
 {
    A m, n (15,18);
    m。print();
    n.print() ;
}
```

Default constructor。
Constructor。
a=10,b=10
a=15, b=18
Donstructor。
Donstructor

```
24、#include 〈iostream。h〉
class Sample
{
int n;
static int sum;
public:
Sample (int x){n=x;}
void add () {sum+=n; }
void disp()
 {
cout 〈〈"n=” 〈〈n< 〈” , sum=” 〈<sum<<endl;
}
};
int Sample: :sum=0;
void main()
 {
Sample a(2),b (3), c (5);
a。add() ;
a.disp();
b.add() ;
b。disp();
c。add();
c。disp();
}
```

n=2,sum=2
n=3, sum=5
n=5,sum=10

25、＃include<iostream。h>

```
class Sample
{
int x;
public:
Sample () {};
Sample (int a){x=a;}
Sample (Sample &a) {x=a.x++ +10; }
void disp (){cout<<" x=" < <x <<endl;}
};
void main()
{
Sample s1 (2) ,s2(s1) ;
s1.disp();
s2。disp();
}
x=3
x=12
```

```
26、 #include <iostream.h>
class A
{
public:
A(char *s) { cout << s << endl; }
~A () {}
} ;
class B:public A
{
public:
B(char *s1, char *s2) :A(s1)
{
cout << s2 << endl;
}
};
class C: public A
 {
public:
C(char *s1,char *s2) : A (s1)
{
cout << s2 < < endl;
}
} ;
class D: public B,public C
 {
public:
```

```
D (char *s1,char *s2,char *s3, char *s4) :B (s1,s2),C (s1,s3)
{
cout << s4 << endl;
}
void main ()
{
D d("class A", "class B", "class C","class D");
}
class A
class B
class A
class C
class D
```

27、#include<iostream.h>

```
class Sample
{
int n;
public:
Sample (int i){n=i;}
friend int add(Sample &s1,Sample &s2);
};
int add (Sample &s1, Sample &s2)
{
return s1。n+s2.n;
}
void main()
 {
Sample s1 (10), s2 (20) ;
cout <<add (s1, s2) <<endl;
}
```

30

28、＃include<iostream.h>

```
class Sample
 {
int x,y;
public:
Sample(){x=y=0; }
Sample (int a,int b) {x=a;y=b;}
~Sample ()
{
if (x==y)
```

```
cout 〈〈” x=y”〈 〈endl;
else
cout〈 〈” x!=y″ 〈〈endl;
}
void disp()
 {
cout 〈〈″x=″ 〈〈x〈 〈″, y=″〈 〈y〈 〈endl;
}
};

void main()
{
Sample s1(2, 3);
s1.disp();
}
x=2, y=3
x!=y
```

五、指出下列程序的错误，并说明为什么：

```
1、＃include 〈iostream.h〉
class Student{
    int sno;
    int age;
    void printStu ();
    void setSno (int d);
};
void printStu()
 {
    cout<<” \nStudent No. is ”<<sno<<”,    ”;
    cout 〈〈” age is ”〈〈age 〈〈”.″ 〈〈endl;
}
void setSno(int s)
 {
    sno=s;
}
void setAge(int a)
{
    age=a;
}
void main ()
{
    Student lin;
    lin.setSno(20021);
```

```
    lin。setAge(20);
    lin。printStu ();
}
```

1)void printStu() 应该改为 void Student：:printStu ()
2)void setSno (int s) 应该改为 void Student:：setSno (int s)
3)Student 的定义中应加入成员函数的声明语句 void setAge(int a);
4) void setAge(int a) 应该改为 void Student：:setAge(int a)
5)应将成员函数定义为 public 类型 在前面加 public:
2、

```
#include 〈iostream.h〉
class Point {
public：
    int x, y;
private：
    Point(){x=1;y=2;}
} ;
void main ()
{
    Point cpoint;
    cpoint.x=2;
}
```

2.应将成员 x 定义为 public

3、＃include<iostream.h>

```
class A
 {
int *a;
public：
A(int x)
 {
a=new int(x) ;
cout 〈<” ＊a=″< 〈*a;
}
};
void main ()
 {
A x (3),*p;
p=new A (5) ;
delete p;
}
```

4、# include 〈iostream。h〉

```
class base
 {
```

```
public:
virtual int func ()  { return 0;  }};
class derived: public base
{
public: int func ()  { return 100; }
} ;
void main ()
{
derived d;
base& b = d;
cout < < b。func ()   <<  endl;
cout << b.base:: func ()   << endl;
}
```

5、#include <iostream.h>

```
#define pi=3.1416;
const float r=3.2;
void main ()
{
float s1, s2, c1,c2,r1;
c1=pi＊r＊r;
s1=2*pi*r;
r=2.8;
c2=pi*r*r;
s2=2*pi*r;
cout <c2<s2;
}
```

1）＃define pi=3。1416; 应改为 #define pi 3。1416

2）修改了 const 类型的变量 r 应改为 static float r=3。2;

3）cout<c2<s2; 应改为 cout< <c2<<” ″<<s

六、程序填空：

1、

```
 class A
{
int  *  a;
int n;
public :
A (int nn=0): n (nn)
{
if (n==0) a=0;
else a=__ a= (int  * ) malloc(n*sizeof (int) );
_ ;  // 分配长度为 n 的动态数组
```

```
}
void Seta（int * aa)
{
for (int i=0;  i 〈n; i++)  a[i]=aa[i];
}
_____～A（）;___// 定义析构造函数，释放动态数组空间
};
```

2、程序输出结果为

a转换为A
b转换为B

请将程序补充完整

```
＃include 〈iostream.h〉
class Sample
 {
char c1,c2;
public：
Sample    (1)
void disp（）
{
cout〈〈c1〈〈”转换为”〈〈c2〈〈endl;
}
};
void main（）
{
   (2)
a。disp（）;
b.disp（);
}
```

2。

1)

```
(char c)
{
    c1=c;
    if（c1>='a' &&c1<='z')
        c2=c1+' A'—' a';
}
```

2)

```
Sample a(' a'), b('b'）;
```

3、程序输出结果为8 8

请将程序补充完整

```
#include<iostream。h>
```

```
class c {
   (5)
   int x;
   c (int px=10)
    {x=px;}
  void setx (int a)  {x=a;}
     (6)
};
   (7)
 { cp (11) ; c*q; q=&p;
  int p:: c＊cptr;  int(c::＊fptr) ();
  p。setx(8) ;
  cptr=&c::x;
  fptr=c:: getx;
   cout< <q—>*cptr <<endl;
   cont <<(q—> ＊fptr) () ;}
```

4。

5)

public:

6)

```
int getx()  {return x;  }
```

7)

```
void main ()
```

5、 程序输出结果为counstructing object:x=1

请将程序补充完整

```
＃include <iostream.h>
class Sample
 {
int x;
public:
Sample(int a)
 {
___(4) __
cout <<"constructing object:x=" <<x<<endl;
}
};
void func(int n)
 {
___ (5) __
}
void main()
```

```
{
func (1);
func (10);
}
```

6.

4)

```
x=a;
```

5)

```
static Sample obj (n);
```

6、程序输出结果为：

A=2，B=7

A=5,B=7

请把程序补充完整

```
#include 〈iostream。h〉
class Sample
{
int A;
static int B;
public:
Sample(int a) {A=a,B+=a;}
static void func (Sample s);
};
void Sample：:func (Sample s)
 {
___ (6)__
}
int Sample:: B=0;
void main()
{
Sample s1 (2) ,s2 (5);
___ (7) __
Sample:: func(s2);
}
```

7.

6)

```
cout〈〈”A=” 〈〈a 〈〈”，B=”<<b<<endl;
```

7)

```
Sample ：:func(s1);
```

九、编程题:

1. 定义个 datetime 类,使其对象可以显示当前系统时间和日期 。

```
#include 〈time.h〉
#include 〈stdio.h〉
```

```
class datetime
{
public:
    int year;
    int month;
    int day;
    int hour;
    int min;
    int sec;
    datetime ()
    {
        struct tm* ptm;
        time_t m;
        time (&m);
        ptm = localtime(&m);
        year = ptm->tm_year+1900;
        month = ptm->tm_mon+1;
        day = ptm-〉 tm_mday;
        hour = ptm—〉 tm_hour;
        min = ptm—〉 tm_min;
        sec = ptm-〉 tm_sec;
    }
    void output ()
    {
        printf (”%.4d/%。2d/%。2d%.2d: %.2d: %。2d\n",year, month, day,hour,
min,sec);
    }
};
void main (void)
{
    datetime d;
    d。output ();
}
```

2、设计一个汽车类 Vehicle,包含数据成员车轮数和车重,由它派生出类 Car 和类 Truck，前者包含载客数,后者包含载重量。编写程序实现.

```
＃include<iostream。h>
class vehicle // 定义汽车类
 {
protected:
int wheels;  // 车轮数
float weight; // 重量
public:
vehicle(int wheels,float weight);
```

```
int get_wheels ();
float get_weight();
float wheel_load();
void show() ;
};
class car: public vehicle // 定义小车类
{
int passenger_load; // 载人数
public:
car(int wheels,float weight, int passengers=4);
int get_passengers ();
void show() ;
};
class truck:public vehicle // 定义卡车类
{
int passenger_load; // 载人数
float payload; // 载重量
public:
truck(int wheels,float weight,int passengers=2, float max_load=24000。00);
int get_passengers();
float efficiency () ;
void show ();
} ;
vehicle: :vehicle (int wheels, float weight)
 {
vehicle: :wheels=wheels;
vehicle:: weight=weight;
}
int vehicle:: get_wheels()
 {
return wheels;
}
float vehicle:: get_weight ()
 {
return weight/wheels;
}
void vehicle::show ()
 {
cout < < ”车轮:″ << wheels << ”个″ << endl;
cout < < ”重量: ″ < < weight < < ”公斤″ << endl;
}
car:: car(int wheels, float weight,
int passengers) : vehicle (wheels, weight)
 {
```

```
passenger_load=passengers;
}
int car：:get_passengers ()
{
return passenger_load;
}
void car：:show()
 {
cout <<” 车型:小车” << endl;
vehicle：:show();
cout 〈〈 "载人：” 〈〈 passenger_load 〈〈 ”人” << endl;
cout << endl;
}
truck： : truck(int wheels， float weight,int passengers， float max_load)：
vehicle(wheels,weight)
 {
passenger_load=passengers;
payload=max_load;
}
int truck::get_passengers ()
{
return passenger_load;
}
float truck::efficiency ()
{
return payload/(payload+weight);
}
void truck：:show()
{
cout 〈<"车型：卡车” 〈〈 endl;
vehicle:: show ();
cout 〈〈 ”载人：" 〈〈 passenger_load 〈〈 ”人” 〈〈 endl;
cout 〈〈 ”效率:” 〈< efficiency () 〈< endl;
cout 〈< endl;
}
void main ()
{
car car1 (4，2000，5);
truck tru1 (10，8000,3，340000);
cout << "输出结果” << endl;
car1. show ();
tru1. show () ;
}
```

3、设计一个点类 Point，包含点的坐标 x，y 两个数据成员,采用友元类的方式分别计算两点间的水平距离和垂直距离。

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

class Point;//先声明类型 Point
int horizontalDistance(const Point& first,  const Point& second); //水平距离函数声明
int verticalDistance (const Point& first, const Point& second);//垂直距离函数声明

class Point
 {
private:
 int x; //横坐标
 int y;//纵坐标
public:
 Point (int x=0, int y = 0) //构造函数
 {
  this->x = x;
  this->y = y;
 }
 friend int horizontalDistance (const Point& first, const Point& second);
 friend int verticalDistance (const Point& first,  const Point& second);
};
//水平距离函数定义
 int  horizontalDistance (const Point& first,  const Point& second)
{
 if (first。x -second.x >=0)
  return first.x-second.x;
 else
  return second.x—first。x;
}
 //垂直距离函数定义
int  verticalDistance (const Point& first,  const Point& second)
{
 if(first.y —second.y 〉=0)
  return first。y—second。y;
 else
  return second.y—first。y;
}
int main()
 {
 Point a (1, 2);    //定义三个 Point 对象
```

```
 Point b (0,0);
 Point c (—1, -1) ;
 //测试
 cout< ( horizontalDistance (a,b) (<endl;
 cout (< horizontalDistance (a,c) < (endl;
 cout< ( verticalDistance (a,b) <<endl;
 cout<< verticalDistance(a,c) (<endl;
return 0;
}
```

4、设计一个点类 Point，包含横、纵两个坐标数据 x,y，由它派生出圆类 Circle,并添加一个半径数据 r，求其面积 area。

```
＃include〈iostream〉        //＃include〈iomanip>//此头文件与下面的 setprecision (3)对应，可同时加上，控制输出精度
using namespace std;
#define PI 3。1415926
//点类如下:
class Point
{
public:
 Point()
  {}
 Point(float x1,float y1)
  {
  x=x1;
  y=y1;
 }
 virtual float area () =0;
 virtual float perim()=0; //周长
private:
 float x;
 float y;
};

class Circle: public Point
{
public:
 Circle (float x1,float y1, float r1) ;
 float area();
 float perim ();
private:
 float r;
};
```

```
Circle:: Circle (float x1, float y1, float r1):Point (x1,y1)
 {
 r=r1;
}
float Circle: :area()
{
 return PI*r*r;
}
float Circle:: perim ()
{
 return 2*PI*r;
}
int main()
{
 Point *p;
 Circle circle (0, 0,3); //前两个为point (0, 0), 第三个常量"3"为圆的半径。
 p=&circle;
 cout<<"Circle 的周长是:"<<p->perim()<<endl;
 cout <<"Circle 的面积是:"<<p->area() <<endl;
}
```

5、设计一个点类Point，包含两个坐标数据成员x，y和一个构造函数；再设计一个友元函数distance（）用于求两点之间的距离。

```
#include<iostream>
using namespace std;
class Point;//先声明类型Point
int horizontalDistance(const Point& first, const Point& second); //水平距离函数声明
int verticalDistance (const Point& first, const Point& second);//垂直距离函数声明
class Point
 {
private:
 int x;//横坐标
 int y;//纵坐标
public:
 Point (int x=0, int y = 0)//构造函数
 {
  this-> x = x;
  this->y = y;
 }
 friend int horizontalDistance (const Point& first, const Point& second) ;
};
```

```
//水平距离函数定义
 int  horizontalDistance（const Point& first, const Point& second)
 {
 if(first。x -second.x >=0)
  return first。x-second。x;
 else
  return second.x—first。x;
}
int main（）
 {
 Point a(1，2）;//定义三个 Point 对象
 Point b（0，0）;
 Point c(—1,-1);
 cout<〈 horizontalDistance(a,b) <<endl;
 cout〈〈 horizontalDistance（a,c)  〈<endl;
 return 0;
}
```

6、编写 class cA 的派生类 class cAB，增加成员函数,用于求圆的周长

```
class cA
 {
int r;
public:
cA（int x){ r=x; }
double area( ) { return r*r*3.14; }
int get_r ( )  { return r; }
} ;

class cAB：public Ca

{cAB（int x):cA(x) { }

double girth( )  { return 2*get_r ( ) *3.14; }};
```

7、定义一个处理日期的类 TDate，它有 3 个私有数据成员:Month,Day,Year 和若干个公有成员函数，并实现如下要求：①构造函数重载；②成员函数设置缺省参数；③定义一个友元函数来打印日期（3 分)；④可使用不同的构造函数来创建不同的对象。

```
＃include 〈stdio。h>

class TDate

 {

public:

 TDate（）;          //构造函数
```

```
 TDate（int nMoth,int nDay,int nYear）；  //构造函数重载
 void SetDay（int nDay=1);      //三个设置某个成员变量的成员函数，都带有默认值
 void SetMonth(int nMonth=1);
 void SetYear(int nYear=2001)；
 void SetDate（int nMoth，int nDay，int nYear）;//一个非静态成员函数
 friend void PrintDate(TDate cTdate)；          //友员函数
private：
   int m_nMonth;
   int m_nDay;
   int m_nYear;
}；
TDate::TDate()
 {
 m_nDay=1;
 m_nMonth=1;
 m_nYear=2000；
}
TDate：：TDate（int nMoth,int nDay,int nYear）
 {
 m_nYear=nYear;
 m_nDay=nDay;
 m_nMonth=nMoth;
}
void TDate：:SetDate(int nMoth，int nDay,int nYear)
 {
 m_nYear=nYear;
 m_nDay=nDay;
 m_nMonth=nMoth;
}
```

```
void TDate::SetDay（int nDay/*=1＊/)
 {
  m_nDay=nDay；
}
void TDate：：SetMonth(int nMonth/*=1*/)
{
  m_nMonth=nMonth;
}
void TDate:：SetYear(int nYear/*=2000*/)
{
  m_nYear=nYear；
}
void PrintDate（TDate cTDate）
{
  printf("Date is:%d-％d-％d"，cTDate。m_nYear,cTDate.m_nMonth，cTDate.m_nDay)；
}
void main（)
 {
  TDate cTdate;
  cTdate.SetDate(6，1，2012）; //使用成员函数
  cTdate.SetDay（10);
  TDate CMyDate（6,1，2012)；//重载的构造函数生成对象实例
  PrintDate(CMyDate)；      //使用友员函数
}
```

8、定义个 datetime 类，使其对象可以显示当前系统时间和日期 。

编写一个程序,该程序的输入有三个整数,找出其中最大的两个整数并输出这两个整数的和

```
#include〈time。h〉
#include〈stdio.h〉
class datetime
 {
public:
```

```
    int year;
    int month;
    int day;
    int hour;
    int min;
    int sec;
    datetime ()
    {
        struct tm* ptm;
        time_t m;
        time (&m);
        ptm = localtime (&m) ;
        year = ptm—>tm_year+1900;
        month = ptm—〉 tm_mon+1;
        day = ptm—>tm_mday;
        hour = ptm-〉 tm_hour;
        min = ptm-〉 tm_min;
        sec = ptm—>tm_sec;
    }
    void output()
     {
        printf ("%.4d/%.2d/%.2d%.2d:%.2d: %.2d\n",year, month,day,hour, min,
sec);
    }
} ;
void main (void)
{
    datetime d;
    d。output();
}
```

十、 简答题

1、类的公有成员和私有成员有何区别?

C++中的类里的成员函数是可以是私有函数或公有函数.两者的区别是,私有函数只可以在类里面调用,而公有函数其他类也可以调用

私有成员只可以在本类中使用不能在其他类中使用，而公有成员都可以使用

2、引入类的静态成员有何意义？

3、什么是多态性？它能带来什么样的好处？

1。 什么是多态

多态是C++中的一个重要的基础,可以这样说,不掌握多态就是C++的门外汉。然而长期以来,C++社群对于多态的内涵和外延一直争论不休.大有只见树木不见森林之势.多态到底是怎么回事呢？说实在的，我觉的多态这个名字起的不怎么好（或是译的不怎么好)。要是我给起名的话,我就给它定一个这样的名字——“调用’同名函数’却会因上下文不同会有不同的实现的一种机制".这个名字长是长了点儿，可是比“多态”清楚多了。看这个长的定义，我

们可以从中找出多态的三个重要的部分。一是“相同函数名”，二是“依据上下文”，三是“实现却不同”。嘿，还是个顺口溜呢.我们且把它们叫做多态三要素吧.

2。　　多态带来的好处

多态带来两个明显的好处：一是不用记大量的函数名了，二是它会依据调用时的上下文来确定实现.确定实现的过程由C++本身完成另外还有一个不明显但却很重要的好处是：带来了面向对象的编程。

4、运算符重载能带来什么好处？

5、怎样理解类与对象的含义?类与对象的关系是什么？

类是现实世界或思维世界中的实体在计算机中的反映,它将数据以及这些数据上的操作封装在一起.

对象是具有类类型的变量.

类是对象的抽象,而对象是类的具体实例.类是抽象的,不占用内存，而对象是具体的，占用存储空间。类是用于创建对象的蓝图，它是一个定义包括在特定类型的对象中的方法和变量的软件模板。

6、为什么要在基类中定义虚函数？

7、引入友元的目的是什么？

友元(成员)函数特点：定义在类外部，但是需要在类体内进行说明；说明时在前面加上关键字 friend class A{public:　。。。friend (B：:) double GetDistance（A start，A stop)；　。。...}；友元（成员)函数作用：提高程序的运行效率,可以访问类中的保护和私有成员；

友元类特点：class A{.。.。.。　friend class B;　。。。..。}；

友元类作用：能够允许一个类中的所有成员函数都能够访问另一个类中的私有成员

具体来说：为了使其他类的成员函数直接访问该类的私有变量即：允许外面的类或函数去访问类的私有变量和保护变量，从而使两个类共享同一函数

优点：能够提高效率，表达简单、清晰

缺点:友元函数破坏了封装机制，尽量不使用成员函数,除非不得已的情况下才使用友元函数。

8、什么是运算符重载？是否所有运算符都能进行重载？

运算作用在复数类对象上应该完成什么功能这就是运算符重载。

重载函数存在于类的继承，是指对你类已有的函数重新定义。重载要注意函数的返回类型及参数表要完全相同。

9、析构函数有什么作用?

解答要点：

对象销毁时,需要调用析构函数.在多态调用时，是用基类的指针访问派生类的对象。如果析构函数是非虚函数，则基类指针只能访问基类的析构函数，而不能访问派生类的析构函数，导致派生类对象销毁时，没有调用派生类的析构函数，只是调用了基类的析构函数。如果把析构函数定义成虚函数,则可克服这个问题.

10、拷贝构造函数在哪几种情况下调用？

解答要点:

用一个对象初始化另一个对象时

当用对象作为函数参数传递时

当函数返回对象时

11、函数重载与函数覆盖有什么不同，它们与多态有什么关系？

解答要点:

函数重载是指函数名相同，而函数的参数个数或类型不同；覆盖是指在派生类中成员函数与基类成员函数的函数名、参数个数、类型与返回值均相同；C++中正是通过虚函数的覆盖，实现多态的功能.

12、C++继承是如何工作的？

解答要点：

继承使得派生类能够使用基类的公有和保护成员,从而实现代码的复用，派生类可以增加成员，也可以隐藏和覆盖基类的成员.对于公有继承，基类成员的访问权限在派生类保持不变。