# 面向对象程序设计C++ 复习题

一、选择题

1.不是面向对象程序设计的基本特征是（ D ）。

A、抽象; B、封装; C、继承; D、类；

2. 如下对类CPoint的说明，其中错误的是（ B ）。

A class CPoint{

B int a=100;

public:

C CPoint(int val);

D ~ CPoint( ); };

A、A； B、B； C、C； D、D；

3.使用new运算符为char数组动态分配10个存储空间，正确的是（ B ）。

A、char\*pre=new char (10); B、char\*pre=new char [10];

C、char\*pre=new char [10](10); D、char \*pre=new char;

4.以下有关类与对象的描述中错误的是（ C ）。

A、类是一种类型，它封装了数据和操作; B、对象是类的实例;

C、一个类的对象只有一个; D、一个对象必属于某个类;

5.下列函数声明中，默认参数设置正确的是（A ）。

A、int max(int x,int y=1,int z=2); B、int max(int x=1,int y,int z=3);

C、int max(int x,int y=2,int z); D、int max(int x=1,int z=3,int y);

6.假设有定义const char \*n=“this is a book”; 错误的是（ A ）。

A、n[3]=‘a’; B、n=“lin”; C、n=new char[10]; D、n=new char(‘a’);

7.以下有关运算符重载的描述中，正确的是（ D ）。

A、C++语言允许在重载运算符时改变运算符的操作数个数;

B、C++语言允许在重载运算符时改变运算符的优先级;

C、C++语言允许在重载运算符时改变运算符的结合性;

D、C++语言允许在重载运算符时改变运算符原来的功能;

8.在类外部可被访问的成员有（ B ）。

A、类所有的成员; B、类公有的成员; C、类私有的成员; D、类保护的成员;

9.以下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，不能默认;

10.以下关于析构函数的描述中，正确的是（ C ）。

A、析构函数可以重载; B、析构函数名与类名完全相同;

C、析构函数不能指定返回类型; D、析构函数可以带参数;

11.重载函数满足的条件的是（ D ）。

A、必须有不同的参数个数; B、对应的参数类型必须不相同;

C、A和B必须同时满足; D、A和B只要满足一个即可;

12.类声明时，以下描述的正确是（ C ）。

A、可以在类的声明中给数据成员赋初值； B、构造函数的名称可以与类名不同；

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；

13.友元函数的作用之一是（ D ）。

A、实现数据的共享; B、加强类的封装;

C、实现数据的隐藏性; D、提高程序的运行效率;

14.构造函数是在（ C ）时被执行。

A、程序编译; B、创建类; C、创建对象; D、程序装入内存;

15.下面关于友元函数的描述中，错误的是（ D ）。

A、关键字friend用于声明友元; B、一个类中的成员函数可以是另一类的友元;

C、友元函数访问对象的成员不受访问属性的影响;

D、友元函数通过this指针访问对象成员;

16.当一个派生类保护继承一个基类时，基类中所有的公有成员和保护成员成为派生类的（ C ）。

A、public成员; B、private成员; C、protected成员; D、友元;

17.在创建派生类的对象时，构造函数的执行顺序是（ D ）。

A、对象成员构造函数、基类构造函数、派生类本身的构造函数;

B、派生类本身的构造函数、基类构造函数、对象成员构造函数;

C、基类构造函数、派生类本身的构造函数、对象成员构造函数;

D、基类构造函数、对象成员构造函数、派生类本身的构造函数;

18.关于纯虚函数与抽象类的描述中，错误的是（ C ）。

A、纯虚函数是一种特殊的函数，它允许没有具体的实现;

B、抽象类是指具有纯虚函数的类;

C、一个基类的声明中有纯虚函数，该基类的派生类一定不再是抽象类;

D、抽象类只能作为基类来使用，其纯虚函数的实现由派生类给出;

19.若类A和类B的定义如下：

class A

{ int x,y；

public:

void get()； };

class B:public A

{

protected:

int z；

public:

void make()； }；

void B::make()

{ z=x\*y； }

则其中（ D ）是非法的表达式。

A、void get()； B、int z； C、void make()； D、k=x\*y；

20.下列关于继承与派送的描述中，错误的是（ D ）。

A、析构函数不能被继承; B、派生类是基类的组合;

C、派生类的成员除了它自己的成员外，还包含了它的基类的成员;

D、派生类中继承的基类成员的访问权限到派生类保持不变;

21.有关派生类中虚函数原型的描述中，正确的是（ D ）。

A、函数类型可以与基类中虚函数的类型不同; B、参数个数可以与基类中虚函数的原型不同;

C、参数类型可以与基类中虚函数的原型不同; D、以上说明都不对;

22.在保护继承的情况下，下面描述正确的是（ A ）。

A、基类中私有成员在派生类中不可直接访问;

B、基类中公有成员和保护成员在派生类中访问属性保持不变;

C、通过派生类对象可以访问基类中公有的成员;

D、在派生类中能够直接访问基类中私有成员;

23.在下面有关静态成员的描述中，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时，要用static前缀; D、静态数据成员只能在类外定义;

24.关于this指针的描述正确的是（ B ）。

A、this指针必须显示声明; B、创建一个对象后，this指针就指向该对象;

C、成员函数拥有this指针; D、静态成员函数拥有this指针;

25.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;

26.已知类A中的一个成员函数的说明为：void Set(A &a);则该函数的参数“A &a”的含义是（ C ）。

A、指向A的指针为a; B、将变量a的地址赋给类A;

C、类A对象引用a用作函数的形参; D、变量A与a按位与后作函数参数;

27. C++是对C语言作了很多改进，即从面向过程变成为面向对象的主要原因是（ D ）。

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、规定函数说明符必须用原型； D、引进了对与对象的概念；

28. 下列程序的输出结果是（ B ）。

#include<iostream>

using namespace std;

int main()

{

int n[][3]={1,2,3,4,5,6};

int (\*p)[3];

p=n;

cout<<p[0][0]<<","<<\*(p[0]+1)<<","<<(\*p)[2]<<endl;

return 0;

}

A、1，3，5; B、1，2，3; C、2，4，6; D、1，3，6;

29. 有关C++编译指令，以下叙述正确的是（ B ）。

A、C++每行可以写多条编译指令; B、#include指令中的文件名可含有路径信息;

C、C++的编译指令可以以#或//开始;

D、C++中不管＃if后的常量表达式是否为真，该部分都需要编译;

30. 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；

31. 要禁止修改指针p本身，又要禁止修改p所指向的数据，这样的指针应定义为（ D ）。

A、const char \*p=“ABCD”； B、char \*const p=“ABCD”；

C、char const \*p=“ABCD”； D、const char \* const p=“ABCD”；

32. 下列语句中正确的是（ D ）。

A、int a, &ra=a, &&ref=ra; B、int a[4], int &ref[4]=a;

C、int a, &ra=a,＆\*refp=ra; D、int a, &ra=a, int\*pi=&ra;

33. 编写C++程序一般需经过的几个步骤依次是（ B ）。

A、编辑、调试、编译、连接； B、编辑、编译、连接、运行；

C、编译、调试、编辑、连接； D.编译、编辑、连接、运行；

34. 使用对象地址作为实参传给形参，下列说法正确的是（ D ）。

A、实参是形参的备份； B、实参与形参无联系；

C、形参是实参的备份； D、实参与形参是同一对象；

35. 在（ A ）情况下适宜采用内联函数。

A、函数代码小，频繁调用； B、函数代码多，频繁调用；

C、函数体含有递归语句； D、函数体含有循环语句；

36. 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；

37. 关于new运算符的下列描述中，错误的是（ D ）。

A、它可以用来动态创建对象和对象数组；

B、使用它创建的对象和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；

38. 假定AA为一个类，a()为该类公有的成员函数，x为该类的一个对象，则访问x对象中成员函数a()的格式为（ B ）。

A、x.a； B、x.a()； C、x-> a()； D、（\*x）.a()；

39. 下面关于访问类A的私有数据成员的说法，错误的是（ C ）。

A、类A的友元函数可以访问类A的私有成员；

B、类A的友元类中的非静态成员函数可以访问类A的私有成员；

C、类A的嵌套类中的非静态成员函数可以访问类A的私有成员；

D、类A的非静态成员函数可以访问类A的私有成员；

40. 有下列函数调用语句，函数fun1( )的实参个数是（ B ）。

fun1(a+b, (y=9, y\*x), fun2(y, n, k));

A、2； B、3； C、5； D、6；

41、面向对象的\_\_\_\_B\_\_是一种信息隐蔽技术，目的在于将对象的使用者与设计者分开。不允许使用者直接存取对象的属性，只能通过有限的接口与对象发生联系

A 多态性 B 封装性 C 继承性 D 重用性

42、下述静态成员的特性中，\_\_\_D\_\_\_是错误的。

A 静态成员函数不能利用this指针

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有的

43、下列关于继承的描述中，\_\_\_D\_\_\_是错误的。

A 析构函数不能被继承

B 派生类是基类的组合

C 派生类的成员除了它自己的成员外，还包含了它的基类的成员

D 派生类中继承的基类成员的访问权限到派生类保持不变

44、下列循环体执行\_\_C\_\_\_\_次。

#include<iostream.h>

void main()

{ int x=-10,k=0;

while(++x){ k++; }

cout<<k;

}

A 无限 B 10 C 9 D 11

45、下列常量中，\_\_\_\_D\_\_不是字符常量？

A ’\005’ B ’\n’ C ’c’ D “a”

46、若已知char str[20];，有语句cin>>str;

当输入为：This is a program所得的结果是str=\_\_\_B\_\_

A This is a program B This C This is D This is a

47、使用new为int数组动态分配10个存储空间，下面哪个语句正确。B

A int \*p=new; B int \*p=new int[10]; C int \*p=new int[]; D int \*p=new int;

48、下列关键字，\_\_\_\_C\_\_不是类的访问权限控制符。

A public B private C class D protected

49、下面定义的共用体长度是\_\_\_D\_\_\_字节。

union M

{ int a;

char b;

float c;

double d;

};

A 1 B 2 C 4 D 8

50、从键盘输入两个整型数2和3，输入数据格式：a＝2，b＝3符合要求的输入语句是\_\_C\_\_\_\_。

A scanf（“％d，％d”，&a，&b）；

B scanf（“％d：％d”，&a，&b）；

C scanf（“a＝％d，b＝％d”，&a，&b）；

D scanf（“％d ％d”，&a，&b）；

51、关于虚函数的描述中，\_\_\_D\_\_\_是正确的。

A. 虚函数是一个静态成员函数

B. 虚函数是一个非成员函数

C. 虚函数既可以在函数说明时定义，也可以在函数实现时定义

D. 派生类的虚函数与基类中对应的虚函数具有相同的参数个数和类型

52、下列语句中，\_\_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;

53、有关运算符重载正确的描述是\_\_\_D\_\_\_

A C++语言允许在重载运算符时改变运算符的操作数个数

B C++语言允许在重载运算符时改变运算符的优先级

C C++语言允许在重载运算符时改变运算符的结合性

D C++语言允许在重载运算符时改变运算符原来的功能

54、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

55、假设已经定义好了一个类Student，现在要定义类Derived，它是从Student类私有派生的，定义类Deriver的正确写法是：\_\_C\_\_\_\_

A class Derived：：Student private{//…..};

B class Derived：Student public{//…..};

C class Derived：private Student {//…..};

D class Derived：：public Student {//…..};

56、下面关于对构造函数的描述中，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 构造函数必须带有返回值

D 构造函数必须定义，不能默认

57、下列语句中错误的是：\_\_\_\_D\_\_

A int \*p=new int;

B int \*p=new int[10];

C int \*p=new int(10);

D int \*p=new int[40](10);

58、假设已经有定义：char \*const name=”chen”；下面的语句中正确的是\_\_A\_\_\_\_

A name[3]=’q’;

B name=”lin”;

C name=new char[5];

D name=new char(‘q’);

59、在下面存储类中， ( C ) 对象的可见性与生存期不一致。

A. 外部类 B. 自动类 C. 内部静态类 D. 寄存器类

60、在下面存储类中，（ A ）的对象不是局部变量。

A. 外部静态类 B. 自动类 C. 函数形参 D. 寄存器类

61、关于局部变量，下面说法正确的是( C )。

A. 定义该变量的程序文件中的函数都可以访问

B. 定义该变量的函数中的定义处以下的任何语句都可以访问

C. 定义该变量的复合语句中的定义处以下的任何语句都可以访问

D. 定义该变量的函数中的定义处以上的任何语句都可以访问

62、一个类的静态数据成员所表示属性 ( C )。

A. 是类的或对象的属性 B. 只是对象的属性

C. 只是类的属性 D. 类和友元的属性

63、类的静态成员的访问控制（ D ）。

A. 只允许被定义为private

B. 只允许被定义为private或protected

C. 只允许被定义为public

D. 可允许被定义为private、protected或public

64、静态成员函数对类的数据成员访问（ B ）。

A. 是不允许的 B. 只允许是静态数据成员

C. 只允许是非静态数据成员 D. 可允许是静态数据成员或非静态数据成员

65、被非静态成员函数访问的类的数据成员( A )。

A. 可以是非静态数据成员或静态数据成员 B. 不可能是类的静态数据成员

C. 只能是类的非静态数据成员 D. 只能是类的静态数据成员

66、静态数据成员的初始化是在（ D ）中进行的。

A. 构造函数 B. 任何成员函数

C. 所属类 D. 全局区

67、当将一个类A或函数f()说明为另一个类B的友元后，类A或函数f()能够直接访问类B的（ D ）。

A. 只能是公有成员 B. 只能是保护成员

C. 只能是除私有成员之外的任何成员 D. 具有任何权限的成员

68、引入友元的主要目的是为了（ C ）。

A. 增强数据安全性 B. 提高程序的可靠性

C. 提高程序的效率和灵活性 D. 保证类的封装性

69、 一个类的成员函数也可以成为另一个类的友元函数，这时的友元说明（ A ）。

A. 需加上类域的限定 B. 不需加上类域的限定

C. 类域的限定可加可不加 D. 不需要任何限定

70、在下面的一维数组定义中，哪一个有语法错误。（ C ）

A. int a[]={1,2,3}; B. int a[10]={0};

C. int a[]; D. int a[5];

72、在下面的字符数组定义中，哪一个有语法错误。（ D ）。

A. char a[20]=”abcdefg”; B. char a[]=”x+y=55.”;

C. char a[15]; D. char a[10]=’5’;

73、在下面的二维数组定义中，正确的是( C )。

A. int a[5][]; B. int a[][5];

C. int a[][3]={{1,3,5},{2}}; D. int a[](10);

74、假定一个二维数组的定义语句为“int a[3][4]={{3,4},{2,8,6}};”，则元素a[1][2]的值为（ C ）。 ({3，4} 是a[0][]的 意思是a[0][0] = 3; a[0][1] = 4 同样 a[1][0] = 2 ; a[1][1] = 8; a[1][2] = 6;

A. 2 B. 4 C. 6 D. 8

75、假定一个二维数组的定义语句为“int a[3][4]={{3,4},{2,8,6}};”，则元素a[2][1]的值为（ A ）。

A. 0 B. 4 C. 8 D. 6

76、若定义了函数 double \*function(), 则函数function的返回值为（ B ）。

A. 实数型 B.实数的地址 C.指向函数的指针 D.函数的地址

77、以下函数的返回结果是（ A ）。

int function(char \*x) {

char \*p=x; while(\*p++); return(p-x-1);

}

A. 求字符串的长度 B.将字符串x连接到字符串p后面

C. 将字符串x复制到字符串p中 D.将字符串x反向存放

78、设有如下函数定义

int f(char \*s) {

char \*p=s;

while(\*p!=’\0’) p++;

return(p-s);

}

在主函数中用cout<<f(“good”)调用上述函数, 则输出结果为（ B ）。

A. 3 B. 4 C. 5 D. 6

79、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;

80、变量s的定义为“char \*s=”Hello world!”;”，要使变量p指向s所指向的同一个字符串，则应选取（ A ）。

A. char \*p=s; B. char \*p=&s; C. char \*p;p=\*s; D. char \*p; p=&s

81、关于void 指针，下列说法正确的是（ C ）。

A. void 指针就是未指向任何数据的指针

B. void 指针就是已定义而未初始化的指针

C. 指向任何类型数据的指针可直接赋值给一个void 指针

D. void 指针值可直接赋给一个非void 指针

82、假定一条定义语句为“int a[10], x, \*pa=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（ D ）。

A. x=pa[3]; B. x=\*(a+3); C. x=a[3]; D. x=\*pa+3;

83、假定有定义“int b[10]; int \*pb;”，则不正确的赋值语句为（ D ）。

A. pb=b; B. pb=&b[0]; C. \*pb=new int; D. pb=b[5];

84、假定p是具有double类型的指针变量，则表达式++p使p的值（以字节为单位）增加（ C ）。

A. 1 B. 4 C. sizeof(double) D. sizeof(p)

85、假定p指向的字符串为”string”，则cout<<p+3的输出结果为（ C ）。

A. string B. ring C. ing D. i

86、假定p指向的字符串为”string”，若要输出这个字符串的地址值，则使用（ D ）。

A. cout<<\*s; B. cout<<s; C. cout<<&s; D. cout<<(void \*)s;

87、定义p并使p指向动态空间中的包含30个整数的数组所使用的定义语句为（ A ）。

A. int \*p=new int[30]; B. int \*p=new int(30);

C. int \*p=new [30]; D. \*p=new int[30];

88、假定p是具有int\*\*类型的指针变量，则给p赋值的正确语句为（ B ）。

A. p=new int; B. p=new int\*; C. p=new int\*\*; D. p=new int[10];

89、要使语句“p=new int[10][20];”能够正常执行，p应被事先定义为（ D ）。

A. int \*p; B. int \*\*p; C. int \*p[20]; D. int(\*p)[20];

90、假定变量a和pa定义为“double a[10], \*pa=a;”，要将12.35赋值给a中的下标为5的元素，不正确的语句是（ D ）。

A. pa[5]=12.35; B. a[5]=12.35;

C. \*(pa+5)=12.35; D. \*(a[0]+5)=12.35;

91、假定变量b和pb定义为“int b[10], \*pb=b;”，要将24赋值给b[1]元素中，不正确的语句是（ C ）。

A. \*(pb+1)=24; B. \*(b+1)=24; C. \*++b=24; D. \*++pb=24;

92、假定指针变量p定义为“int \*p=new int(100);”，要释放p所指向的动态内存，应使用语句（ A ）。

A. delete p; B. delete \*p; C. delete &p; D. delete []p;

93、假定指针变量p定义为“int \*p=new int[30];”，要释放p所指向的动态内存，应使用语句（ D ）。

A. delete p; B. delete \*p; C. delete &p; D. delete []p;

94、当类中一个字符指针成员指向具有n个字节的存储空间时，它所能存储字符串的最大长度为( C )。

A. n B. n+1 C. n-1 D. n-2

95、在一个用链表实现的队列类中，假定每个结点包含的值域用elem表示，包含的指针域用next表示，链队的队首指针用elemHead表示，队尾指针用elemTail表示，若链队非空，则进行插入时必须把新结点的地址赋给( D )。

A. elemHead B. elemTail

C. elemHead->next和elemHead D. elemTail->next和elemTail

96、在一个用链表实现的队列类中，假定每个结点包含的值域用elem表示，包含的指针域用next表示，链队的队首指针用elemHead表示，队尾指针用elemTail表示，若链队为空，则进行插入时必须把新结点的地址赋给( C )。

A. elemHead B. elemTail

C. elemHead和elemTail D. elemHead或elemTail

97、当类中的一个整型指针成员指向一块具有n\*sizeof(int)大小的存储空间时，它最多能够存储( A )个整数。

A. n B. n+1 C. n-1 D. 1

98、假定AB为一个类，则执行 “AB \*px=new AB[n];”语句时将( A )。

A. 动态分配一个数组 B. 动态分配一个对象

C. 静态分配一个数组 D. 静态分配一个对象

99、设px是指向一个类对象的指针变量，则执行 “delete px;”语句时，将自动调用该类的( C )。

A. 无参构造函数 B. 带参构造函数 C. 析构函数 D. 拷贝构造函数

100、当一个类对象离开它的作用域时，系统自动调用该类的( D )。

A. 无参构造函数 B. 带参构造函数 C. 拷贝构造函数 D. 析构函数

101、假定一个类对象数组为A[n]，当离开它定义的作用域时，系统自动调用该类析构函数的次数为( C )。

A. 0 B. 1 C. n D. n-1

102、假定AB为一个类，则执行 “AB a[10];”语句时调用该类无参构造函数的次数为( D )。

A. 0 B. 1 C. 9 D. 10

103、假定AB为一个类，则执行 “AB \*px=new AB[n];”语句时调用该类无参构造函数的次数为( A )。

A. n B. n-1 C. 1 D. 0

104、假定AB为一个类，则执行 “AB a, b(3), \*p;”语句时共调用该类构造函数的次数为( A )。

A. 2 B. 3 C. 4 D. 5

105、假定AB为一个类，则执行 “AB a(2), b[3], \*p[4];”语句时共调用该类构造函数的次数为( B )。

A. 3 B. 4 C. 5 D. 9

106、假定AB为一个类，则执行“AB a, b(2), c[3], \*p=&a;”语句时共调用该类无参构造函数的次数为( D )

A. 5 B. 6 C. 3 D. 4

107、假定AB为一个类，则执行“AB \*p=new AB(1,2);”语句时共调用该类构造函数的次数为( B )。

A. 0 B. 1 C. 2 D. 3

108、假定AB为一个类，px为指向该类的一个含有n个对象的动态数组的指针，则执行“delete []px;”语句时共调用该类析构函数的次数为( C )。

A. 0 B. 1 C. n D. n+1

109．在c++中，类之间的继承关系具有( c )。

(a) 自反性 (b) 对称性 (c) 传递性 (d) 反对称性

110．下列关于类的继承描述中，( d )是正确的。

(a) 派生类公有继承基类时，可以访问基类的所有数据成员，调用所有成员函数。

(b) 派生类也是基类，所以它们是等价的。

(c) 派生类对象不会建立基类的私有数据成员，所以不能访问基类的私有数据成员。

(d) 一个基类可以有多个派生类，一个派生类可以有多个基类。

111．当一个派生类公有继承一个基类时，基类中的所有公有成员成为派生类的( a )。

(a) public成员 (b) private成员 (c) protected成员 (d) 友员

4．当一个派生类私有继承一个基类时，基类中的所有公有成员和保护成员成为派生类的( b )。

(a) public成员 (b) private成员 (c) protected成员 (d) 友员

112．当一个派生类保护继承一个基类时，基类中的所有公有成员和保护成员成为派生类的( c )。

(a) public成员 (b) private成员 (c) protected成员 (d)友员

113．不论派生类以何种方式继承基类，都不能直接使用基类的( b )。

(a) public 成员 (b) private成员

(c) protected成员 (d) 所有成员

114．下面描述中，错误的是( d )。

(a) 在基类定义的public成员在公有继承的派生类中可见，也能在类外被访问。

(b) 在基类定义的protected成员在私有继承的派生类中可见。

(c) 在基类定义的公有静态成员在私有继承的派生类中可见。

(d) 访问声明可以在公有继承派生类中把基类的public成员声明为private成员。

115．在c++中，可以被派生类继承的函数是( a )。

(a) 成员函数 (b)构造函数 (c) 析构函数 (d)友员函数

116．在创建派生类对象时，构造函数的执行顺序是( d )。

(a) 对象成员构造函数—基类构造函数—派生类本身的构造函数

(b) 派生类本身的构造函数—基类构造函数—对象成员构造函数

(c) 基类构造函数—派生类本身的构造函数—对象成员构造函数

(d) 基类构造函数—对象成员构造函数—派生类本身的构造函数

117．当不同的类具有相同的间接基类时，( c )。

(a) 各派生类无法按继承路线产生自己的基类版本

(b) 为了建立惟一的间接基类版本，应该声明间接基类为虚基类

(c) 为了建立惟一的间接基类版本，应该声明派生类虚继承基类

(d) 一旦声明虚继承，基类的性质就改变了，不能再定义新的派生类

118．关于函数模板，描述错误的是( a )。

(a) 函数模板必须由程序员实例化为可执行的函数模板

(b) 函数模板的实例化由编译器实现

(c) 一个类定义中，只要有一个函数模板，则这个类是类模板

(d) 类模板的成员函数都是函数模板，类模板实例化后，成员函数也随之实例化

119．下列的模板说明中，正确的是( c )。

(a) template < typename T1, T2 >

(b) template < class T1, T2 >

(c) template < typename T1, typename T2 >

(d) template ( typedef T1, typedef T2 )

120．假设有函数模板定义如下：

　template <typename T>

　Max( T a, T b ,T &c)

{ c　= a + b ; }

下列选项正确的是( b )。

(a) int x, y; char z ; (b) double x, y, z ;

Max( x, y, z ) ; Max( x, y, z ) ;

(c) int x, y; float z ; (d) float x; double y, z ;

Max( x, y, z ); Max( x, y, z ) ;

121．关于类模板，描述错误的是( a )。

(a) 一个普通基类不能派生类模板

(b) 类模板可以从普通类派生，也可以从类模板派生

(c) 根据建立对象时的实际数据类型，编译器把类模板实例化为模板类

(d) 函数的类模板参数需生成模板类并通过构造函数实例化

122．说明虚函数的关键字是（ B ）。

A. inline B. virtual C. define D. static

123．在每个C++程序中都必须包含有这样一个函数，该函数的函数名为（ A ）。

A. main B. MAIN C. name D. function

124．cout是某个类的标准对象的引用，该类是（ A ）。

A. ostream B. istream C. stdout D. stdin

125．如果在类外有函数调用CPoint::func()；则函数func()是类CPoint的（ C ）。

A. 私有静态成员函数 B. 公有非静态成员函数

C. 公有静态成员函数 B. 友元函数

126. 如果class类中的所有成员在定义时都没有使用关键字public、private或protected，则所有成员缺省定义为（ C ）。

A. public B. protected C. private D．static

127．一个类的所有对象共享的是（ D ）。

A. 私有数据成员 B. 公有数据成员

C. 保护数据成员 D. 静态数据成员

128．动态联编所支持的多态性称为（ D ）。

A. 虚函数 B. 继承

C. 编译时多态性 D. 运行时多态性

129．定义类模板时要使用关键字（ D ）。

A. const B. new C. delete D. template

130．对虚基类的定义（ A ）。

A. 不需要使用虚函数 B. 必须使用虚函数

C. 必须使用private D. 必须使用public

131．类型转换函数（ A ）。

A. 不能带有参数 B. 只能带一个参数

C. 只能带2个参数 D. 只能带3个参数

132、下列描述中正确的是（ A ）。

A） 不是每个C++程序都必须包含预处理命令#include

B）C++程序必须有return 语句

C）C++程序中的变量必须定义在主程序内部

D）C++程序中所使用的符号常量可以不进行定义

133. C++源程序文件的扩展名为（ A ）。

选项A） .CPP

选项B）.C

选项C）.DLL

选项D）.EXE

134有关C++编译指令，以下叙述正确的是（ B ）。

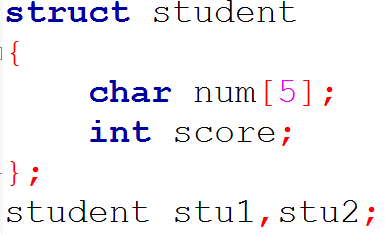
A）C++每行可以写多条编译指令

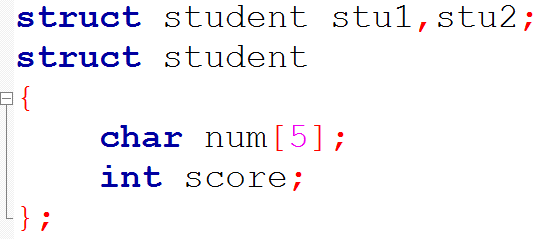
B）#include指令中的文件名可含有路径信息

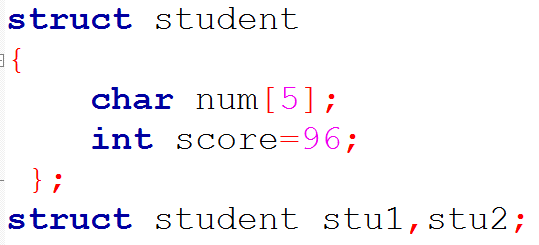
C）C++的编译指令可以以#或//开始

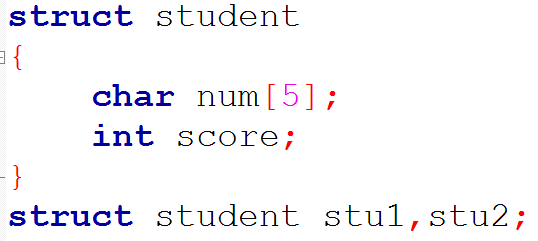
D）C++中不管＃if后的常量表达式是否为真，该部分都需要编译

135、以下各项用于定义结构体类型，并定义结构体变量，其中正确的是（ A ）。

A）

B）

C）

D）

136、函数定义为Fun(int &i)，有定义变量int n = 100，则下面调用正确的是（ C ）。

A） Fun(20)；

B）Fun(20 + n)；

C）Fun(n)；

D）Fun(&n)；

137假如已定义函数：int max(int x，int y);以下函数调用语句错误的是（ C ）。

A）cout<<max(3，5)<<endl；

B）max(6，9)；

C）max(5，7)=4;

D）int k；k=max(12，4);

138、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；

139假设已经定义了类Student，其构造函数有两个，一个不带参数，一个带两个参数都是int类型的，则下面描述错误的是（ A ）。

选项A）int n; int const&m=n; 则m是n的常引用

选项B）int n; const int &m=n; 则m是n的常引用

选项C）Student const s1；则s1是常对象

选项D）const Student s2(23，30)；则s2是常对象

140、下面描述错误的是（ C ）。

选项A）常成员函数可以访问常数据成员

选项B）普通成员函数可以访问和修改普通数据成员的值

选项C）普通成员函数可以访问和修改常数据成员的值

选项D）常成员函数可以访问但不可以修改普通数据成员和常数据成员的值

141、如果有基类Base和派生类Derived，并分别定义了基类对象b和派生类对象d，下面描述错误的是（ D ）。

选项A）对于语句：b=d；即派生类对象d赋值给基类对象b，用派生类对象中从基类继承来的数据成员，逐个赋值给基类对象的数据成员

选项B）对于语句：Base &br=d；即派生类对象可以初始化基类对象的引用

选项C）对于语句：Base \*br=&d；即派生类对象地址可以赋值给指向基类对象的指针

选项D）如果函数的形参是派生类对象，在调用函数时可以用基类对象作为实参

142、假设类Base中重载了+运算符，则对于Base类的两个对象a和b，则表达式a+b，C++编译器在处理它时，将其解释成函数调用表达式为（ B ）。

选项A）operator(a+b)；

选项B）operator+(a，b)；

选项C）+(a，b)；

选项D）operator a+b；

143、下面描述错误的是（ D ）。

选项A）对于双目运算符而言，成员运算符重载函数参数表中含有一个参数

选项B）对于双目运算符而言，友元运算符重载函数参数表中含有两个参数

选项C）对于单目运算符而言，成员运算符重载函数参数表中没有参数

选项D）对于单目运算符而言，友元运算符重载函数参数表中没有参数

144、下面描述错误的是（ D ）。

选项A）在赋值表达式a=b的情况下，赋值运算符右边b的值需转为a类型后进行赋值

选项B）当char或short类型变量与int类型变量进行运算时，将char或short类型转换为int类型

选项C）当表达式a+b两个操作数类型不一致时，在运算前，级别低的类型自动转换为级别高的类型

选项D）当表达式中操作数类型不一致，运算时，不需要进行类型转换

## 二、判断题

1、类的静态数据成员需要在定义每个类的对象时进行初始化。F

2、当将一个类S定义为另一个类A的友元类时，类S的所有成员函数都可以直接访问类A的所有成员。T

3、静态数据成员必须在类外定义和初始化。T

4、静态成员函数可以引用属于该类的任何函数成员F。

5、友元函数是在类声明中由关键字friend修饰说明的类的成员函数。F

6、友元函数访问对象中的成员可以不通过对象名F

类的静态数据成员需要在定义每个类的对象时进行初始化。F

7、当将一个类S定义为另一个类A的友元类时，类S的所有成员函数都可以直接访问类A的所有成员。T

8、静态数据成员必须在类外定义和初始化。T

9、静态成员函数可以引用属于该类的任何函数成员F。

10、友元函数是在类声明中由关键字friend修饰说明的类的成员函数。F

11、友元函数访问对象中的成员可以不通过对象名F

构造函数可以声明为虚函数。（F　）

12、．在析构函数中调用虚函数时，采用动态束定。（F　）

13、．在一个成员函数内调用一个虚函数时，对该虚函数的调用进行动态束定。（T　）

14、公有继承可以实现子类型。（T　）

15、构造函数可以声明为纯虚函数。（F　）

16、.如果派生类的成员函数的原型与基类中被定义为虚函数的成员函数原型相同，那么，这个函数自动继承基类中虚函数的特性。对

17、虚函数不能是类的静态成员。对

18、重定义虚函数的派生类必须是公有继承的。对

19、对于从基类继承的虚函数，派生类也可以不进行重定义。对

20、纯虚函数与函数体为空的虚函数等价。错

21、多态类提供了一种动态多分支派送机制。对

22、作为虚函数隐含参数的this指针，决定了虚函数调用时执行的代码。对

23．虚函数在基类和派生类之间定义，且要求函数原型完全一致。 （ T ）

24．抽象类可以用来直接创建对象。 （ F ）

25．内联函数中可以出现递归语句。 （ F ）

26．模板类与类模板的意义完全相同。 （ F ）

27．常对象只能调用常成员函数。 （ T ）

28．重载函数要求函数有相同的函数名，但具有不同的参数序列。 （ T ）

## 三、分析程序输出结果

3、分析下列程序的输出结果。

#include <iostream>

using namespace std;

class Student

{

string name;

int age;

int score;

public:

Student( )

{

name=" ";

age=0;

score=0;

cout<<"Student( )"<<endl;

}

Student(string n,int m,int s )

{

name=n;

age=m;

score=s;

cout<<"Student(string n,int m,int s )"<<endl;

}

~Student( )

{

cout<<"~Student( )"<<endl;

}

};

class Teacher

{

string name;

int age;

public:

Teacher( )

{

name=" ";

age=0;

cout<<"Teacher( )"<<endl;

}

Teacher(string n,int m)

{

name=n;

age=m;

cout<<"Teacher(string n,int m)"<<endl;

}

~Teacher( )

{

cout<<"~Teacher( )"<<endl;

}

};

class TS

{

private:

int number;

Student s;

Teacher t;

public:

TS( )

{

number=0;

cout<<"TS( )"<<endl;

}

TS(string n1,int m1,int s,string n2,int m2,int num ):s(n1,m1,s),t(n2,m2)

{

number=num;

cout<<"TS(string n1,int m1,int s,string n2,int m2,int num ):s(n1,m1,s),t(n2,m2)"<<endl;

}

TS(string n1,int m1,int s,int num ):s(n1,m1,s)

{

number=num;

cout<<"TS(string n1,int m1,int s,int num ):s(n1,m1,s)"<<endl;

}

TS(string n2,int m2,int num ):t(n2,m2)

{

number=num;

cout<<" TS(string n2,int m2,int num ):t(n2,m2)"<<endl;

}

~TS( )

{

cout<<"destructing ts"<<endl;

}

};

void main( )

{

TS ts1;

cout<<"--------以上是定义ts1输出的情况----------"<<endl;

TS ts2("张同学",19,96,"王老师",36,1);

cout<<"--------以上是定义ts2输出的情况----------"<<endl;

TS ts3("姚同学",18,67,2);

cout<<"---------以上是定义ts3输出的情况---------"<<endl;

TS ts4("李老师",42,3);

cout<<"----------以上是定义ts4输出的情况--------"<<endl;

cout<<"------in main---------"<<endl;

}

4、分析下列程序的输出结果。

#include <iostream.h>

#include<stdlib.h>

class A

{

private:

static int total;

public:

A ( )

{ total++;

cout<<"Constructor called"<<" "<<total<<endl;

}

~A( )

{ total--;

cout<<"destructor called"<<" "<<total<<endl;

}

int get( ){ return total;}

};

int A::total=0;

void main( )

{

A a1,a2;

cout<<a1.get()<<" "<<"objects in existence\n";

A \*p=new A;

if(!p)

{ cout<<"Allocationg error\n";

exit(0);

}

cout<<a1.get()<<" "<<"objects in existence\n";

delete p;

cout<<a1.get()<<" "<<"objects in existence\n";

}

5、分析下列程序的输出结果。

#include <iostream.h>

class Array

{ private:

int \*m,size;

public:

Array(int x)

{ size=x;

m=new int[x];

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

}

~Array()

{ delete []m;}

void print()

{ int i;

for(i=0;i<size;i++)cout<<\*(m+i)<<endl;

}

};

void main()

{

Array obj1(5);

obj1.print();

}

6、分析下列程序的输出结果。

#include <iostream.h>

class Example

{ private:

int i;

public:

Example(int n)

{ i=n;

cout<<"Example(int n)"<<endl;

}

Example(const Example &obj)

{ i=obj.i;

cout<<"Example(const &obj)"<<endl;

}

~Example()

{ cout<<"~Example()"<<endl;}

int get\_i()

{ return i;}

};

int sqr\_it(Example obj)

{ cout<<"sqr\_it(Example obj) "<<endl;

return obj.get\_i()\*obj.get\_i();

}

void main( )

{ Example x(10);

cout<<x.get\_i()<<endl;

cout<<sqr\_it(x)<<endl;

cout<<"in main()"<<endl;

}

7、分析下列程序的输出结果。

#include <iostream.h>

class Student

{

private:

string name;

int score;

public:

Student(string n,int s )//学生类构造函数

{

name=n;

score=s;

cout<<"constructing student"<<endl;

}

~Student( )//学生类析构函数

{

cout<<"denstructing student"<<endl;

}

};

class Teacher

{

private:

string name;

int score;

public:

Teacher( string n,int s)//教师类构造函数

{ cout<<"constructing teacher"<<endl; }

~Teacher( )//教师类析构函数

{ cout<<"destructing teacher"<<endl; }

};

class TS

{ private:

Student student;//学生对象数据成员

Teacher teacher; //教师对象数据成员

public:

TS( )//教师学生类构造函数

{ cout<<"costructing ts"<<endl; }

~TS( )//教师学生类析构函数

{ cout<<"destructing ts"<<endl; }

};

void main( )

{ TS ts;

cout<<"in main"<<endl;

}

8、分析下列程序的输出结果。

#include <iostream.h>

class A

{ public:

A( )

{ cout<<"A( )"<<endl; }

virtual ~A( )

{ cout<<"~A( )"<<endl; }

virtual void f( )

{ cout<<"A::f( )."<<endl; }

};

class B : public A

{ public:

B()

{ cout<<"B()"<<endl; }

~B( )

{ cout<<"~B( )"<<endl; }

};

class C : public B

{ public:

C( )

{ cout<<"C( )"<<endl; }

~C( )

{ cout<<"~C( )"<<endl; }

void f( )

{ cout<<"C::f( )."<<endl; }

};

void main( )

{

A \*a=new C;

a->f( );

delete a;

}

9. 分析下列程序的输出结果。

#include<iostream>

using namespace std;

class Base1

{

public:

Base1()

{

cout<<"Base1()"<<endl;

}

};

class Base2

{

public:

Base2()

{

cout<<"Base2()"<<endl;

}

};

class Level1:public Base2,virtual public Base1

{

public:

Level1()

{

cout<<"Level1()"<<endl;

}

};

class Level2:public Base2,virtual public Base1

{

public:

Level2()

{

cout<<"Level2()"<<endl;

}

};

class Top:public Level1,virtual public Level2

{

public:

Top()

{

cout<<"Top()"<<endl;

}

};

int main()

{

Top obj;

return 0;

}

10分析下列程序的输出结果。

#include<iostream>

using namespace std;

class XXX

{

public:

int x;

XXX(int i)

{

x=i;

}

void f( )

{

cout<<"class XXX: void f():"<<x<<endl;

}

};

class YYY : public XXX

{

public:

int x;

YYY(int i):XXX(i+100)

{

x=i;

}

void f( )

{

cout<<"class YYY: void f():"<<x<<endl;

}

};

void main ()

{

YYY obj(10);

obj.f( ); //访¤?问¨º函¡¥数ºy Y::f( )

obj.XXX::f( ); //访¤?问¨º函¡¥数ºy X::f( )

cout<<"obj.x"<<obj.x<<endl;

cout<<"obj.XXX::x"<<obj.XXX::x<<endl;

}