对于常数据成员，下面描述正确的是 【 BA 】

A. 常数据成员必须被初始化，并且不能被修改

B. 常数据成员可以不初始化，并且不能被修改

C. 常数据成员可以不初始化，并且可以被修改

D. 常数据成员必须被初始化，并且可以被修改

答案是 A. 常数据成员必须被初始化，并且不能被修改。

常数据成员在对象创建时必须被初始化，并且一旦被初始化后，其值不能被修改。这是因为常数据成员被声明为const，表示其值是常量，在对象的生命周期内是不可变的。

如果常数据成员没有在构造函数的初始化列表中进行初始化，会导致编译错误。一旦常数据成员被初始化，它的值就被确定，并且不能在对象的生命周期内进行修改。

因此，选项 B、C 和 D 都是错误的描述。只有选项 A 描述了常数据成员的正确特性，即必须被初始化且不能被修改

在C++中，使用流进行输入输出，其中用于定义文件输入流对象的类是 【 B 】

1. ios B. ifstream C. ofstream D. istrstream

答案是 B. ifstream。在C++中，用于定义文件输入流对象的类是**ifstream**。它是**istream**类的派生类，用于从文件中读取数据。**ifstream**类提供了一些用于读取文件的成员函数，如**open()**用于打开文件、**close()**用于关闭文件、**getline()**用于读取一行文本等。其他选项的描述如下：

* A. **ios**是一个基类，用于定义流的基本操作和状态。它不是用于定义文件输入流对象的类。
* C. **ofstream**是用于定义文件输出流对象的类。它是**ostream**类的派生类，用于向文件中写入数据。
* D. **istrstream**是用于从字符数组中进行输入的类，而不是用于定义文件输入流对象的类。

下列关于异常的叙述错误的是【 D 】

A. 只要是编程者认为是异常的都可当异常抛出

B. 硬件故障也可当异常抛出

C. 运行错属于异常

D. 编译错属于异常，可以抛出

答案是 D. 编译错属于异常，可以抛出。

选项 D 的描述是错误的。编译错误（compile errors）不属于异常。编译错误是在编译阶段发生的错误，通常是由于语法错误、类型错误、缺少声明等问题导致的。这些错误在编译器检查源代码时就会被捕捉到，并且会导致编译过程中断，无法生成可执行文件。

异常（exceptions）是在程序运行时发生的错误或异常情况，它们与编译错误是不同的概念。运行时错误包括但不限于：除零错误、内存访问错误、无效的操作、文件不存在等。当程序运行时遇到异常情况，可以通过抛出异常并在适当的位置进行捕获和处理。

其他选项的描述如下：

* A. 只要是编程者认为是异常的都可以抛出。这是正确的，编程者可以根据程序逻辑和需求自行决定在何处抛出异常。
* B. 硬件故障也可以当作异常抛出。这是正确的，某些硬件故障情况也可以通过抛出异常来通知程序进行相应处理。
* C. 运行错误属于异常。这是正确的，运行时错误是在程序运行过程中发生的异常情况，可以通过异常处理机制进行处理。

在面向对象的程序设计中，基类可以为其派生类提供一组公共的行为，而各个派生类可以为这组公共行为提供不同的实现，从而使得派生类的对象在执行这些公共行为时会有不同的表现，这种特性称为面向对象的【 D 】

A. 封装性 B. 消息性 C. 继承性 D. 多态性

答案是 D. 多态性。

多态性（Polymorphism）是面向对象程序设计中的一个重要特性，它允许不同的对象以各自独特的方式对共享的行为进行响应。

通过继承关系，基类可以定义一组公共的行为（方法或函数），而派生类可以继承这些公共行为并提供自己的实现。当使用基类的指针或引用来操作派生类的对象时，可以根据实际对象的类型调用相应的方法，这就是多态性的体现。

多态性使得程序可以以一种统一的方式处理不同的对象，根据对象的实际类型来决定具体调用哪个类的方法，从而实现代码的灵活性和可扩展性。

其他选项的描述如下：

* A. 封装性（Encapsulation）是面向对象编程中的另一个重要特性，它将数据和操作封装在对象中，隐藏了内部实现细节，通过公共接口访问对象。封装性确保数据的安全性和一致性，并提供了良好的模块化和代码复用。
* B. 消息性（Message passing）是一种描述对象之间通信的概念，它表示对象之间通过发送消息来交互和执行操作。消息性是面向对象编程中的基本概念之一，但不是描述基类和派生类关系的特性。
* C. 继承性（Inheritance）是面向对象编程中的重要特性，它允许一个类（派生类）从另一个类（基类）继承属性和行为。继承性允许派生类重用基类的代码，并在此基础上进行扩展和修改。尽管继承性为多态性提供了基础，但继承本身并不是描述派生类对象在执行公共行为时有不同表现的特性。

关于域的描述中，错误的是【 C 】

A. 一个类是一个独立的域

B. 派生类域被自动嵌套在基类类域中

C. 域操作符::可以被重载

D. 不同域中可以定义同名变量和函数

错误的描述是C. 域操作符::可以被重载。

域操作符::用于访问类的静态成员和命名空间的成员，它本身不可以被重载。在C++中，重载操作符是指对类的成员函数或全局函数进行重载，但域操作符::不是一个可以重载的操作符。它的功能是固定的，用于限定命名空间或类的作用域。

A. 一个类是一个独立的域： 这是正确的描述。在面向对象编程中，类被视为一个独立的域，它可以包含成员变量、成员函数和其他类相关的元素。类提供了封装数据和行为的能力，它定义了对象的属性和操作。

B. 派生类域被自动嵌套在基类类域中： 这是正确的描述。在继承关系中，派生类继承了基类的成员，包括成员变量和成员函数。当在派生类中访问成员时，可以直接使用成员名，而无需显式地指定基类的名称。编译器会自动在派生类的域中查找成员，如果找不到则会继续在基类的域中查找。

C. 域操作符::可以被重载： 这是错误的描述，如前面所解释的。

D. 不同域中可以定义同名变量和函数： 这是正确的描述。不同的域可以定义具有相同名称的变量和函数，因为每个域都有自己的作用域。当访问变量或函数时，编译器会根据当前的作用域来确定使用哪个域中的定义。这种机制可以实现命名空间的概念，避免名称冲突和重复。

下列对变量的引用中错误的是【 C 】

A. int a; int &p = a;

B. char a; char \*p = &a;

C. int a; int &p; p = a;

D. float a; float p=a;

错误的是C. int a; int &p; p = a;

在C语言中，引用必须在声明的同时初始化，并且不能单独声明一个引用变量。选项C中的代码尝试在声明引用p时未进行初始化，然后在后面的语句中尝试将a赋值给p，这是错误的语法。正确的方式是在声明引用时进行初始化，如选项A中的代码所示。

选项A. int a; int &p = a; 这是正确的引用的使用方式，p是a的引用，即p和a指向相同的内存位置。

选项B. char a; char \*p = &a; 这是使用指针的方式，p指向a的内存位置。

选项D. float a; float p = a; 这是将a的值赋给p，而不是使用引用或指针。p是一个新的变量，它的值与a相同，但它们是独立的变量。

下面【 A 】的叙述不符合赋值兼容规则。

A. 基类的对象可以赋值派生类的对象

B. 派生类指针可以指向基类对象

C. 派生类的对象可以初始化基类的对象

D. 派生类的对象的地址可以赋值给指向基类的指针

正确的回答是：A

A. 基类的对象可以赋值派生类的对象

这叙述不符合赋值兼容规则。根据赋值兼容规则，在C++中，可以将派生类对象赋值给基类对象，但不能将基类对象直接赋值给派生类对象。这是因为派生类对象可能包含额外的成员或重写的成员函数，而基类对象无法完全表示派生类的特性。

B. 派生类指针可以指向基类对象

这是符合赋值兼容规则的。派生类指针可以指向基类对象，因为派生类对象是基类对象的扩展。

C. 派生类的对象可以初始化基类的对象

这是符合赋值兼容规则的。可以使用派生类对象初始化基类对象，只会保留基类的成员和属性，而派生类特有的成员和属性将被忽略。

D. 派生类的对象的地址可以赋值给指向基类的指针

这也是符合赋值兼容规则的。派生类对象的地址可以赋值给指向基类的指针，因为派生类对象包含了基类对象的所有成员和属性。这样的指针可以用于访问基类对象的成员和调用基类对象的方法

构造函数不具备的特征的是【 D 】

A.构造函数的函数名与类名相同

B.构造函数可以重载

C.构造函数可以设置默认参数

D.构造函数必须指定返回类型

正确的回答是：D

D. 构造函数必须指定返回类型

这是错误的描述。构造函数不需要指定返回类型，包括void类型。构造函数在创建对象时被调用，它们的主要目的是初始化对象的成员变量和执行其他必要的初始化操作。因此，构造函数没有返回类型，包括没有指定返回类型。

关于虚函数的描述中，正确的是【 D 】

A. 虚函数是一个static类型的成员函数

B. 虚函数是一个友元函数

C. 基类中说明了虚函数后，派生类中与其对应的函数必须再次说明为虚函数

D. 派生类的虚函数与基类的虚函数具有相同的参数个数和类型

正确的描述是：

D. 派生类的虚函数与基类的虚函数具有相同的参数个数和类型。

虚函数是在基类中声明为虚函数的成员函数。当派生类继承基类并重写虚函数时，派生类的虚函数必须具有与基类的虚函数相同的函数签名，包括参数个数和类型。这确保了在运行时通过基类指针或引用调用虚函数时，可以正确地动态绑定到派生类的实现。

另外三个选项的描述是错误的：

A. 虚函数不是一个static类型的成员函数。虚函数是通过基类的指针或引用进行动态绑定的，而static成员函数是在类级别上调用，并且不能被派生类重写。

B. 虚函数也不是一个友元函数。友元函数是在类外部定义的，可以访问类的私有成员，但它们不是虚函数。

C. 派生类中不需要再次说明虚函数。在基类中将函数声明为虚函数后，在派生类中对于相同的函数名，在函数声明前不需要再次使用关键字"virtual"进行说明。重写基类的虚函数会自动使派生类中的函数成为虚函数。

下列关于静态数据成员的说法，不正确的是【 C 】

A. 类中定义的公用静态数据成员，可以通过类的对象来访问

B. 类中定义的所有静态数据成员，都必须在类外初始化

C. 静态数据成员不是所有对象所共用的

D. 普通的成员函数可以直接访问类中的静态数据成员

错误的描述是：

C. 静态数据成员不是所有对象所共用的。

静态数据成员是类的成员，它们被所有类的对象所共享。每个类的对象都可以访问和修改相同的静态数据成员。静态数据成员在内存中只有一份拷贝，无论创建多少个类的对象，它们都共享相同的静态数据成员。

其他选项的描述是正确的：

A. 类中定义的公用静态数据成员可以通过类的对象来访问。可以使用对象名或作用域解析运算符(::)来访问公用静态数据成员。

B. 类中定义的所有静态数据成员都必须在类外初始化。静态数据成员在类外部进行初始化，并且在类定义内部只是声明，不能在类定义内部直接初始化。

D. 普通的成员函数可以直接访问类中的静态数据成员。成员函数可以通过类的对象或类的名称来访问静态数据成员。

下列关于运算符重载的描述中，正确的是【 D 】

A. 运算符重载可以改变操作数的个数

B. 运算符重载可以改变优先级

C. 运算符重载可以改变结合性

D. 运算符重载不可以改变语法结构

正确的描述是：

D. 运算符重载不可以改变语法结构。

运算符重载允许程序员重新定义已有运算符的行为，以适应自定义类型的操作。然而，运算符重载不能改变运算符的语法结构，即不能更改运算符的操作数个数、优先级或结合性。运算符的操作数个数、优先级和结合性是由语言规范确定的，并且运算符重载的目的是为了保持语言的一致性和可预测性。

其他选项的描述是错误的：

A. 运算符重载可以改变操作数的个数。运算符重载时，操作数的个数必须与原始运算符的操作数个数相同。不能通过运算符重载来改变操作数的个数。

B. 运算符重载不能改变优先级。运算符的优先级是由语言规范定义的，无法通过运算符重载来修改。

C. 运算符重载不能改变结合性。运算符的结合性也是由语言规范定义的，无法通过运算符重载来改变运算符的结合性。

有关内联函数的描述，正确的是【 A 】

A. 内联函数可以提高程序效率

B. 内联函数的调用过程与普通函数一样

C. 只要用 inline 关键字声明的函数都是内联函数

D. 全局函数不能声明为内联函数

正确的描述是：

A. 内联函数可以提高程序效率。

内联函数的主要目的是在调用点展开函数代码，避免了函数调用的开销。通过将函数的代码嵌入到调用点处，可以减少函数调用的开销，从而提高程序的执行效率。内联函数适用于短小的函数或频繁调用的函数。

其他选项的描述是错误的：

B. 内联函数的调用过程与普通函数不一样。内联函数在调用时，不会像普通函数一样创建函数的副本和保存现场，而是直接将函数的代码插入到调用点处，类似于宏展开。这样可以减少函数调用的开销。

C. 不是所有用**inline**关键字声明的函数都是内联函数。**inline**关键字只是对编译器的提示，表明函数可以作为内联函数进行处理。编译器可以选择忽略这个提示，不将函数作为内联函数处理，特别是对于函数体比较复杂的情况。

D. 全局函数可以声明为内联函数。与局部函数一样，全局函数也可以使用**inline**关键字进行声明为内联函数。内联函数的适用范围不限于局部函数，可以包括全局函数和类的成员函数。

下列操作符中不能被重载的是【 D 】

A. ->

B. \*

C. %

D. ?:

正确的描述是：

D. ?: 操作符不能被重载。

条件运算符（?:），也称为三元运算符，用于条件判断和选择。它是一个特殊的操作符，无法通过运算符重载来改变其行为或定义新的操作。

其他选项的描述是错误的：

A. -> 操作符可以被重载。箭头操作符（->）用于通过指针访问类的成员，它可以被重载，以定义自定义的指针访问行为。

B. \* 操作符可以被重载。解引用操作符（）用于获取指针所指向的对象，在类中可以通过重载操作符来定义自定义的解引用行为。

C. % 操作符可以被重载。取模运算符（%）用于取得两个数的余数，它可以被重载，以定义自定义类型之间的取模操作。

如果一个函数的声明是int fun(char\*)，则下列函数声明中，错误地重载了该函数的是【 B 】

A. int fun (char\*, int)

B. void fun (char\*)

C. int fun (int)

D. void fun (char\*, int)

错误地重载了该函数的是：

B. void fun (char\*)

原始函数的声明是 int fun(char\*)，而选项 B 中的函数声明是 void fun(char\*)。在函数重载中，函数名称相同但参数列表不同。尽管返回类型不同，但在函数重载中，仅仅改变返回类型是不足以进行重载的，因此选项 B 错误地重载了该函数。

其他选项的描述是正确的：

A. int fun (char\*, int)：参数列表包含一个额外的 int 类型参数，正确地重载了原始函数。

C. int fun (int)：参数列表与原始函数不同，正确地重载了原始函数。

D. void fun (char\*, int)：参数列表包含一个额外的 int 类型参数，正确地重载了原始函数。

C++函数的重载条件如下：

1. 函数名称相同：函数重载要求函数名称相同。
2. 参数列表不同：函数重载要求函数的参数列表不同。参数列表可以包括参数的类型、顺序和个数的差异。
3. 返回类型可以相同也可以不同：函数重载可以具有相同的返回类型，也可以具有不同的返回类型。返回类型不是函数重载的决定因素。

注意：函数重载不能仅仅通过返回类型的差异进行区分。如果两个函数具有相同的名称和参数列表，但返回类型不同，则会导致编译错误。

函数重载的目的是允许使用相同的函数名称来执行不同的操作或处理不同类型的数据，以提高代码的可读性和灵活性

下面是关于派生类构造函数的执行顺序的叙述：①执行派生类的构造函数函数体中的内容。②按照基类被继承时声明的顺序（从左向右）执行基类的构造函数。③按照内嵌成员对象在类中声明的顺序执行它们的构造函数。上述执行步骤的正确排序是【D】

A. ①②③ B. ②①③ C. ③②① D. ②③①

正确的排序是：

D. ②③①

派生类构造函数的执行顺序是按照以下步骤进行：

② 按照基类被继承时声明的顺序（从左向右）执行基类的构造函数。首先，派生类会调用基类的构造函数，以初始化从基类继承的成员。

③ 按照内嵌成员对象在类中声明的顺序执行它们的构造函数。在基类构造函数执行完毕后，会按照内嵌成员对象在类中的声明顺序执行它们的构造函数，以初始化成员对象。

① 执行派生类的构造函数函数体中的内容。在基类构造函数和成员对象构造函数执行完毕后，派生类构造函数会执行自身的构造函数体，以完成对派生类成员的初始化。

因此，正确的排序是②③①。首先执行基类构造函数，然后执行内嵌成员对象的构造函数，最后执行派生类构造函数体

设A是一个有不带参数的构造函数的类，fun是一个如下定义的函数：A fun(A &a){return a;}则在执行程序段{A a ; fun(a); }时，对A类的：①构造函数，②析构函数，③拷贝构造函数的调用顺序是【 B 】

A. ①③③②②② B. ①③②②

C. ①②③② D. ①②③③②②

在公有派生类的成员函数不能直接访问基类中继承来的某个成员，则该成员一定是基类中的【 A 】

A. 私有成员 B. 公有成员

C. 保护成员 D. 保护成员或私有成员

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【 C 】

A. 函数体 B. 对象 C. 类 D. 数据块

关于运算符重载，下列表述中正确的是【 B 】

A. 可以通过运算符重载来创建C++中原来没有的运算符

B. 在类型转换函数的定义中不需要声明返回类型

C. 运算符函数的返回类型不能声明为基本数据类型

D. C++已有的任何运算符都可以重载

正确的描述是：

B. 在类型转换函数的定义中不需要声明返回类型

解释如下：

A. 运算符重载可以用来重新定义现有的运算符的行为，但不能创建新的运算符。C++中已经定义了一组运算符，只能对这些运算符进行重载。

B. 在类型转换函数（重载类型转换运算符）的定义中，不需要显式声明返回类型。返回类型由函数的返回值类型隐式确定。

C. 运算符函数的返回类型可以是基本数据类型，也可以是自定义的类类型。运算符函数的返回类型是根据实际需求和操作结果来确定的。

D. C++并非所有的运算符都可以重载。只有一部分运算符可以被重载，例如赋值运算符、算术运算符、关系运算符等。而一些运算符如作用域解析运算符(::)和条件运算符(?:)等是不能被重载的。

因此，选项 B 正确地描述了运算符重载的情况。

以下是一个类型转换函数的示例，它重载了类型转换运算符，返回类型被隐式确定，无需显式声明返回类型：



在上述代码中，**MyInt**类重载了类型转换运算符 **operator int()**，它允许将 **MyInt** 对象隐式转换为 **int** 类型。在 **main** 函数中，将 **MyInt** 对象 **myInt** 赋值给 **int** 类型的变量 **x**，由于存在类型转换运算符的重载，**MyInt** 对象被自动转换为 **int** 类型并赋值给 **x**，最终输出的结果是 **5**。

这个例子展示了在类型转换函数的定义中不需要声明返回类型，返回类型由函数的返回值类型隐式确定。

下列关于静态数据成员的特性的叙述中，错误的是【 C 】

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有的

D. 初始化静态数据成员时，要在其名称前加类名和作用域运算符

错误的是：

C. 静态数据成员不是所有对象所共有的

静态数据成员是类的成员，而不是对象的成员。它被所有类的对象所共享，即所有对象共享同一个静态数据成员的内存空间。

正确的描述是：

A. 说明静态数据成员时前边要加修饰符static

静态数据成员在声明时需要加上 static 修饰符，以指示该成员是静态的。

B. 静态数据成员要在类体外进行初始化

静态数据成员需要在类体外进行初始化，可以在类外部的命名空间范围内进行初始化。

D. 初始化静态数据成员时，要在其名称前加类名和作用域运算符

在类外部初始化静态数据成员时，需要使用类名和作用域运算符来指定静态数据成员的名称。例如：**ClassName::staticMemberName = value;**。

为了使类中的某个成员不能被类的对象通过成员操作符访问，则不能把该成员的访问权限定义为【 AD 】。

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

为了使类中的某个成员不能被类的对象通过成员操作符访问，不能把该成员的访问权限定义为：

A. public：公有成员可以通过类的对象以及类的派生类对象通过成员操作符访问。

D. static：静态成员可以通过类名和作用域运算符访问，无需通过对象访问。

因此，将成员的访问权限定义为 protected 或 private 可以限制类的对象通过成员操作符直接访问该成员。

在用关键字class定义的类中，以下叙述正确的是【 A 】

A. 在类中，不作特别说明的数据成员均为私有类型

B. 在类中，不作特别说明的数据成员均为公有类型

C. 类成员的定义必须是成员变量定义在前，成员函数定义在后

D. 类的成员定义必须放在类定义体内部

正确的叙述是：

A. 在用关键字class定义的类中，不作特别说明的数据成员均为私有类型。

解释如下：

A. 在C++中，使用关键字class定义的类，默认情况下，类中的数据成员是私有的，即不作特别说明的数据成员默认为私有类型。

B. 不正确。在class定义的类中，不作特别说明的数据成员默认为私有类型，而不是公有类型。

C. 不正确。类成员的定义顺序可以灵活地安排，可以先定义成员变量，再定义成员函数，或者相反。

D. 不正确。类的成员定义可以放在类定义体内部，也可以在类定义体外部进行，只需在类定义体内提供成员的声明即可。

因此，选项 A 正确地描述了在用关键字class定义的类中，不作特别说明的数据成员均为私有类型。

下列情况中，不会调用复制构造函数的是【 C 】

A. 函数的返回值是类的对象，函数执行返回调用时

B. 函数的形参是类的对象，调用函数进行形参和实参相结合

C. 将类的一个对象赋值给该类的另一个对象时

D. 用一个对象去初始化同一类的另一个新对象时

不会调用复制构造函数的情况是：

C. 将类的一个对象赋值给该类的另一个对象时。

解释如下：

A. 当函数的返回值是类的对象时，函数执行完毕并返回时会调用复制构造函数，将函数内部的对象复制给函数外部的对象。

B. 当函数的形参是类的对象时，在调用函数时，实参会传递给形参，此过程中会调用复制构造函数来创建形参对象。

D. 当用一个对象去初始化同一类的另一个新对象时，会调用复制构造函数，将一个对象的值复制给另一个对象。

因此，选项 C 是不会调用复制构造函数的情况，因为在对象赋值的过程中，会调用赋值运算符重载函数而不是复制构造函数。

下列关于抽象类描述不正确的是【 D 】

A. 抽象类不能定义对象

B. 其派生类没有定义纯虚函数，则仍是抽象类。”

C. 其派生类定义所有纯虚函数，则是具体类

D. 不能定义指向抽象类的指针变量

D. 不能定义指向抽象类的指针变量。

这个描述是不正确的。在C++中，可以定义指向抽象类的指针变量。抽象类可以作为一种接口来使用，可以声明指向抽象类类型的指针或引用，并使用它们来访问派生类的对象。这样的指针或引用可以用于实现多态性，即在运行时根据对象的实际类型来调用相应的成员函数。

A. 抽象类不能定义对象。这个描述是正确的。抽象类是一种不能被实例化的类，它仅作为其他类的基类存在。它的目的是为了提供一个共享的接口或行为规范，具体的实现由派生类来完成。因此，不能直接创建抽象类的对象。

B. 其派生类没有定义纯虚函数，则仍是抽象类。这个描述是正确的。如果一个类继承自抽象类但没有实现其所有的纯虚函数，那么该派生类仍然是一个抽象类，因为它没有提供对所有纯虚函数的具体实现。只有在派生类中实现了所有纯虚函数，才可以将其实例化为具体的类对象。

C. 其派生类定义所有纯虚函数，则是具体类。这个描述是正确的。当一个抽象类的所有纯虚函数在派生类中都得到了具体的实现，那么这个派生类就不再是抽象类，而是一个具体的类。它可以被实例化并直接使用。

总结：

* 抽象类不能定义对象，只能作为基类被继承。
* 派生类如果没有定义纯虚函数，仍然是抽象类。
* 派生类如果定义了所有纯虚函数，就成为具体类，可以实例化为对象。
* 在C++中可以定义指向抽象类的指针变量，用于实现多态性。

下列关于常成员和常对象的叙述中，正确的是【 A 】

A. 通过常对象只能调用其常成员函数

B. 常成员函数只能由常对象调用

C.通 过常成员函数可以修改常成员数据的值

D. 常成员函数可以通过类名来调用

A. 通过常对象只能调用其常成员函数。

这个描述是正确的。常对象是指通过使用**const**关键字声明的对象，它被限制为只能调用常成员函数。常成员函数是在类中使用**const**关键字声明的成员函数，它承诺不会修改对象的状态。

B. 常成员函数只能由常对象调用。

这个描述是不正确的。常成员函数可以由常对象和非常对象调用。常成员函数在定义时添加了**const**关键字，表示该函数在调用过程中不会修改对象的状态。因此，即使是非常对象也可以调用常成员函数。

C. 通过常成员函数可以修改常成员数据的值。

这个描述是不正确的。常成员函数被声明为**const**，它们承诺不会修改对象的状态，包括对象的常成员数据。在常成员函数中，对于常成员数据的修改操作是不允许的。

D. 常成员函数可以通过类名来调用。

这个描述是不正确的。常成员函数不能通过类名直接调用，它们只能通过对象来调用。类名只能用于访问静态成员函数或静态成员变量。常成员函数是与对象相关的成员函数，需要通过对象来调用。

窗体顶端

关于成员函数特征的下列叙述中，错误的是【 A 】

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

A. 成员函数一定是内联函数。

这个描述是错误的。成员函数不一定是内联函数。内联函数是在编译器决定是否进行内联展开的函数，它通常在类定义中定义并且在类外部声明为**inline**，但并非所有成员函数都会被内联展开。一般来说，编译器会根据一些因素（如函数的复杂性、调用频率等）来决定是否内联展开函数。

B. 成员函数可以重载。

这个描述是正确的。成员函数可以像全局函数一样进行重载。函数重载是指在同一个作用域内，可以有多个同名函数，但它们的参数列表必须不同，可以有不同的参数类型、不同的参数个数或者不同的参数顺序。

C. 成员函数可以设置缺省参数值。

这个描述是正确的。成员函数可以像全局函数一样设置缺省参数值。缺省参数值是指在函数定义时为参数指定的默认值，当调用函数时如果没有提供对应的参数值，则会使用默认值。

D. 成员函数可以是静态的。

这个描述是正确的。成员函数可以是静态的。静态成员函数是与类关联而不是与类的实例对象关联的函数。它们可以直接通过类名来调用，而不需要创建类的对象。静态成员函数没有隐含的**this**指针，因此无法访问非静态成员变量。

下面关于友元函数和友元类的叙述中，错误的是【C】

A. 如果在类A中把类B的某个函数声明为自己的友元函数，那么在类A的定义之前一定要有类B的完整定义

B. 如果在类A中把类B声明为自己的友元类，那么在类A的定义之前可以仅有类B的前向声明，而不必有类B的完整定义

C. 如果将类B声明为类A的友元，那么类A也就成为类B的友元

D. 如果将类B声明为类A的友元，那么类B中的函数可以访问类A的全部成员

A. 如果在类A中把类B的某个函数声明为自己的友元函数，那么在类A的定义之前一定要有类B的完整定义。这个描述是正确的。在类A中将类B的函数声明为友元函数时，需要知道类B的完整定义，以便正确识别和访问该函数。

B. 如果在类A中把类B声明为自己的友元类，那么在类A的定义之前可以仅有类B的前向声明，而不必有类B的完整定义。这个描述也是正确的。当类A将类B声明为友元类时，只需要知道类B的名称和基本信息，可以使用前向声明来满足这个要求。

C．友元关系不是相互的

D.将类B 声明为类A 的友元类，那么类B 中的所有成员函数都是类A 的友元函数，可以访问类A 的所有成员

程序中重载了C++流运算符<<，它是【 】

A.用于输出操作的成员函数

B.用于输入操作的成员函数

C.用于输入操作的非成员函数

D.用于输出操作的非成员函数

D. 用于输出操作的非成员函数。

C++流运算符**<<**通常被重载为用于输出操作，它可以将对象的内容输出到输出流中。这种重载是通过定义一个非成员函数来实现的，该函数接受一个输出流对象和要输出的对象作为参数，并将对象的内容输出到输出流中。这样，我们可以使用**<<**运算符来方便地输出自定义类型的对象。因此，选项D是正确的描述。

下列字符常量的写法中错误的是【 C 】

1. ‘\105’ B. ‘\*’ C. “” D. ‘\n’

C. ""

这个写法是错误的。在C++中，双引号**""**表示一个空字符串，而不是字符常量。字符常量使用单引号**''**进行表示，而不是双引号。

其他选项的写法是正确的：

A. '\105'：这是一个八进制转义序列，表示字符代码为105的字符。

B. '\*'：这是一个普通的字符常量，表示字符**\***。

D. '\n'：这是一个转义序列，表示换行符（newline）字符。

下列关于静态数据成员的描述中，正确的是【 A 】

A. 静态数据成员是类的所有对象共享的数据

B. 类的每个对象都有自己的静态数据成员

C. 类的不同对象有不同的静态数据成员

D. 静态数据成员不能通过类的对象调用

A. 静态数据成员是类的所有对象共享的数据

正确的描述是A. 静态数据成员是类的所有对象共享的数据。静态数据成员是指属于类本身而不是类的任何特定对象的成员变量。它们在类的所有对象之间共享，可以通过类名直接访问，而不需要创建类的对象。

B. 类的每个对象都有自己的静态数据成员：这个描述是错误的。静态数据成员是类的所有对象共享的，它们不属于类的任何特定对象。因此，每个对象并没有自己的静态数据成员。

C. 类的不同对象有不同的静态数据成员：这个描述也是错误的。静态数据成员是类级别的，它们在所有对象之间共享，因此不同的对象之间并没有不同的静态数据成员。

D. 静态数据成员不能通过类的对象调用：这个描述是不准确的。静态数据成员可以通过类的对象来访问，但是它们更常见地通过类名直接访问，因为它们属于类本身而不是类的对象。这样可以更清楚地表达静态数据成员的特性，同时也可以避免混淆。

设A是一个类，且语句“A\*p=newA[10];”是能够正确执行的，则下列关于该语句的执行的叙述最恰当的是【 D 】

A. 类A有不带参数的构造函数，且构造函数被调用10次

B. 类A有拷贝构造函数，且拷贝构造函数被调用10次

C. 类A有这样的构造函数，其每个参数都有默认的形参值，且该构造函数被调用10次

D. A和C皆有可能。

D. A和C皆有可能。

根据提供的语句"A\*p = new A[10];"，我们可以推断出以下两种情况：

A. 类A有不带参数的构造函数，且构造函数被调用10次：这意味着在创建A类型的动态数组时，将调用A类的不带参数的构造函数10次来初始化数组的每个元素。

C. 类A有这样的构造函数，其每个参数都有默认的形参值，且该构造函数被调用10次：这意味着在创建A类型的动态数组时，将调用A类的带有默认形参值的构造函数10次来初始化数组的每个元素。

因此，根据提供的信息，A和C都是可能的情况。

如果派生类中有指针类型的数据成员，该数据成员指向的内存空间需要动态申请，并且需要在构造函数中初始化该数据成员，在析构函数中回收该成员可能指向的内存空间。此时，如果存在通过基类的指针删除派生类的对象的情况，则需要【 B 】

A. 将基类的构造函数声明为虚函数

B. 将基类的析构函数声明为虚函数

C. 将派生类的构造函数声明为虚函数

D. 将派生类的析构函数声明为纯虚函数

在类 object 中，以下定义会出现编译错误的是【 C 】

A. static object a;

B. object a;

C. static object \*pa;

D. object\* pa;

运算符“>”被友元重载，则表达式“obj1>obj2”被编译器解释为【 A 】

A. operator > (obj1,obj2)

B. > (obj1,obj2)

C. obj2.operator > (obj1)

D. obj1.operator > (obj2)

答案是 A. operator > (obj1, obj2)。

当运算符">"被友元重载时，表达式"obj1 > obj2"将被编译器解释为"operator > (obj1, obj2)"。

友元函数是在类外部定义的函数，但被声明为类的友元，可以访问类的私有成员。当一个运算符被声明为友元函数并重载时，它可以实现对两个对象之间的操作。

在这种情况下，">"运算符被重载为一个友元函数，接受两个参数，即obj1和obj2。所以，编译器会将表达式"obj1 > obj2"解释为"operator > (obj1, obj2)"，并调用相应的友元函数来执行比较操作。

其他选项的描述如下：

* B. ">" (obj1, obj2)：这种语法是不正确的，不符合运算符重载的语法。
* C. obj2.operator > (obj1)：这种语法是错误的，运算符重载函数不是类的成员函数，所以不会使用成员函数调用的语法。
* D. obj1.operator > (obj2)：这种语法是错误的，运算符重载函数不是类的成员函数，所以不会使用成员函数调用的语法。

关于友元，下列说法错误的是【 A 】

A. 如果类A是类B的友元，则类B也是类A的友元

B. 如果函数fun()被说明为类A的友元，那么在fun()中可以访问类A的私有成员

C. 友元关系不能被继承

D. 如果类A是类B的友元，那么类A的所有成员函数都是类B的友元

答案是 A. 如果类A是类B的友元，则类B也是类A的友元。

说法 A 是错误的。友元关系不具有传递性，即如果类A是类B的友元，不意味着类B也是类A的友元。友元关系是单向的，类A可以访问类B的私有成员，但类B无法访问类A的私有成员，除非在类B中将类A也声明为友元。

面向对象系统具有【 A 】 、继承性和多态性三种基本特性。

A. 封装性

B. 消息性

C. 维护性

D. 动态性

从基类 public 派生一个类，基类 protected 成员成为派生类【 B 】 成员。

A. public

B. protected

C. private

D. virtual

答案是 B. protected。

当从基类以 public 方式派生一个类时，基类的 protected 成员会成为派生类的 protected 成员。

在 C++ 中，继承关系中的访问控制有三种类型：public、protected 和 private。当使用 public 继承时，基类的 public 成员在派生类中仍然是 public 成员，可以被外部访问。基类的 protected 成员在派生类中成为 protected 成员，只能被派生类及其派生类的成员函数访问。基类的 private 成员在派生类中不可见，不能直接访问。

所以，当以 public 方式派生一个类时，基类的 protected 成员会成为派生类的 protected 成员。

下列关于虚函数的描述中，正确的是【 C 】

A.虚函数可以是一个static类型的成员函数

B.虚函数可以是非成员函数

C.基类中说明了虚函数后，派生类中其对应的函数可以不必说明为虚函数

D.派生类的虚函数与基类的虚函数具有不同的参数个数和类型

答案是 C. 基类中说明了虚函数后，派生类中其对应的函数可以不必说明为虚函数。

选项 C 是正确的。当基类中的成员函数被声明为虚函数后，派生类中对应的函数可以选择是否继续声明为虚函数。如果派生类中的函数与基类中的虚函数具有相同的函数签名（即相同的函数名、参数列表和常量性），那么它将自动成为虚函数，不需要再次声明为虚函数。

其他选项的描述如下：

* A. 虚函数不能是 static 类型的成员函数。虚函数是用于实现动态多态性的，而 static 成员函数是属于类本身的，不与特定对象相关联，无法实现运行时的动态绑定。
* B. 虚函数必须是类的成员函数，它们不能是非成员函数。虚函数是依赖于类的对象进行动态绑定的，非成员函数无法直接与类的对象关联。
* D. 派生类的虚函数与基类的虚函数应具有相同的参数个数和类型。虚函数的重写要求派生类中的虚函数与基类中的虚函数具有相同的函数签名（函数名、参数列表和常量性都相同），以确保正确的运行时动态绑定。参数个数和类型必须匹配。

this指针存在的目的是【 C 】

A. 保证基类共有成员在子类中可以被访问

B. 保证基类保护成员在子类中可以被访问

C. 保证每个对象拥有自己的数据成员，但共享处理这些数据成员的代码

D. 保证基类私有成员在子类中可以被访问

答案是 C. 保证每个对象拥有自己的数据成员，但共享处理这些数据成员的代码。

this指针是一个隐含的指针，指向当前对象的地址。它在每个非静态成员函数中都是可用的，用于访问当前对象的成员。

this指针的存在主要是为了解决对象的成员函数与对象的数据成员之间的关联问题。通过this指针，每个对象可以访问自己的数据成员，保证了每个对象拥有独立的数据存储空间。同时，通过this指针，对象可以共享处理这些数据成员的代码，避免了为每个对象复制一份相同的代码。

其他选项的描述如下：

* A. this指针并不是为了保证基类共有成员在子类中可以被访问的。访问权限是通过继承和访问控制符来管理的，与this指针无关。
* B. this指针并不是为了保证基类保护成员在子类中可以被访问的。访问权限也是通过继承和访问控制符来管理的，与this指针无关。
* D. this指针并不是为了保证基类私有成员在子类中可以被访问的。私有成员对于子类是不可访问的，无论是否使用this指针。

以下成员函数表示纯虚函数的是【 C + B 】

A. virtual int vf (int);

B. void vf (int) = 0;

C. virtual void vf () = 0;

D. virtual void vf (int) {}

纯虚函数是在基类中声明的虚函数，但没有提供具体的实现。它只有函数的声明部分，以 "= 0" 结尾，表示该函数是纯虚函数。派生类必须实现纯虚函数，否则派生类也将成为抽象类。

选项 A 中的函数声明不包含 "= 0"，表示它是一个普通的虚函数，不是纯虚函数。

选项 B 中的函数声明以 "= 0" 结尾，表示它是一个纯虚函数。这是一个有效的表示纯虚函数的方式。

选项 D 中的函数声明不包含 "= 0"，表示它是一个普通的虚函数，而不是纯虚函数。它提供了具体的实现，派生类不需要强制实现该函数。

对于一组功能相同但处理数据类型不同的函数可以通过【 B 】抽象为同一段代码。

A. 类模板

B. 函数模板

C. 函数重载

D. 虚函数

C++标准库中的类、结构以及相关函数等被定义在名字空间 【 D 】中。

A. cplus

B. news

C. std

D. namespace

已知在一个类体中包含如下函数原型: Volume operator-(Volume)const;，

下列关于这个函数的叙述中，错误的是【 B 】

A. 这是运算符-的重载运算符函数

B. 这个函数所重载的运算符是一个一元运算符

C. 这是一个成员函数

D. 这个函数不改变类的任何数据成员的值

答案是 B. 这个函数所重载的运算符是一个一元运算符。

选项 B 是错误的。给定的函数原型 "Volume operator-(Volume) const;" 是一个减法运算符的重载函数，但它是一个二元运算符，而不是一元运算符。

其他选项的描述如下：

* A. 这是运算符-的重载运算符函数。这是正确的，函数原型指定了对减法运算符 "-" 的重载。
* C. 这是一个成员函数。这是正确的，函数原型中的 "const" 关键字表示这是一个成员函数，并且它不会修改对象的数据成员。
* D. 这个函数不改变类的任何数据成员的值。这是正确的，函数原型中的 "const" 关键字表示这个函数是一个常量成员函数，它不会修改类的数据成员的值。

在下列函数原型中，可以作为类AA构造函数的是【 D 】

A. void AA(int); B. int AA();

C. AA(int)const; D. AA(int);

下面四个选项中，【 A 】是用来声明虚函数的。

A. virtual B. public

C. include D. using namespace

实现运行时的多态性要使用【 D 】

A. 重载函数 B. 构造函数

C. 析构函数 D.虚函数

要实现动态联编，必须通过【 A 】调用虚函数

A. 对象指针 B. 成员名限定

C. 对象名 D. 派生类名

以下【 C 】成员函数表示纯虚函数

1. virtual int vf(int); B. void vf(int)=0; C. virtual void vf()=0; D. virtual void vf(int)

在表达式 x+yz中， + 是作为成员函数重载的运算符， 是作为非成员函数重载的运算符。则 operator+ 有\_\_\_ 个参数，operator\* 有\_\_\_ 参数。【 C 】

A. 2、2 B. 2、1 C. 1、2 D. 1、1

一个类的友元函数或友元类可以访问该类的【 C 】

A. 私有成员 B. 保护成员

C. 所有成员 D. 公有成员

对C++语言和C语言的兼容性，其中描述正确的是【 B 】

A. C兼容C++

B. C++兼容C

C. C++不兼容C

D.C++部分兼容C

关于公有继承，下列说法错误的是【 D 】

A. 基类的公有成员和保护成员被继承后作为派生类的公有成员和保护成员

B. 派生类的其他成员可以直接访问基类的公有成员和保护成员

C. 派生类成员和对象都无法访问基类的私有成员

D. 派生类的对象可以访问基类的私有成员

关于new运算符的下列描述中，【 D 】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.使用它创建数组时必须指定初始值

面向对象程序设计数据与【 A 】放在一起，作为一个相互依存、不可分割的整体来处理。

A. 对数据的操作 B. 信息

C. 数据隐藏 D. 数据抽象

在类中说明的成员可以使用关键字的是【 A 】

A. public B. extern C. cpu D. register

下列关于C++语言类的描述中错误的是【 C 】

A. 类用于描述事物的属性和对事物的操作

B. 类与类之间通过封装而具有明确的独立性

C. 类与类之间必须是平等的关系，而不能组成层次结构

D. 类与类之间必须是平等的关系，而不能组成层次结

下面关于类中概念的描述中错误的是【 D 】

A. 类是抽象数据类型的实现

B. 类是具有共同行为的若干对象的统一描述体

C.类是创建对象的模板

D. 类就是C语言中的结构体类型

作用域运算符的功能是【 C 】

A. 标识作用域的级别

B. 指出其大小

C. 标识某个成员是属于哪一个类

D. 给出的范围

下列关于模板的叙述中，错误的是【 B 】

A. 在一定条件下函数模板的实参可以省略

B. 类模板不能有数据成员

C. 在模板声明中用< >括起来的部分是模板的形参表

D. 模板声明中的第一个符号总是关键字template

下列关于C++函数的叙述中，正确的是【 D 】

A. 每个函数都必须有参数

B. 每个函数都必须返回一个值

C. 函数在被调用之前可以不先声明

D. 函数可以自己调用自己

下列带缺省值参数的函数说明中，正确的说明是【 A 】

A. int Fun(int x, int y=2, int z=3);

B. int Fun(int x=1, int y, int z);

C. int Fun(int x=1, int y, int z=3);

D. int Fun(int x, int y=2, int z);

假定A为一个类，f()为该类公有的函数成员，a1为该类的一个对象，则\n访问a1对象中函数成员f()的格式为( B )。

1. a1.f B. a1.f() C. a1->f() D.（a1）.f ()

下列关于运算符重载的叙述中，正确的是【 B 】

A. 通过运算符重载，可以定义新的运算符

B.有的运算符只能作为成员函数重载

C. 若重载运算符+，则相应的运算符函数名是+

D. 重载二元运算符时，必须声明两个形参\n\n

假定AA为一个类，a()为该类公有的函数成员，x为该类的一个对象，则访问x对象中函数成员a()的格式为【 B 】

A. x.a B.x.a() C.x->a D. x->a()

下列关于构造函数的特点，其中描述错误的是【 B 】。

A. 构造函数是一种成员函数，它具有一般成员函数的特点

B. 构造函数必须指明其类型

C. 构造函数的名称与其类名相同

D. 一个类中可以定义一个或多个构造函数

假设Class Y: public X，即类Y是类X的派生类，则定义一个Y类的对象时和删除Y类对象时，调用构造函数和析构函数的次序分别为【 A 】

A. X,Y；Y,X B. X,Y；X,Y

C. Y,X；X,Y D. Y,X；Y,X

对使用new申请的存储空间，释放时必须使用【 C 】

1. free B. release C. delete D. malloc

下列叙述错误的是【 D 】。

A. catch（…）语句可捕获所有类型的异常

B．一个try语句可以有多个catch语句

C．catch（…）语句可以放在catch语句组的中间

D．程序中try语句与catch语句是一个整体，缺一不可

所谓的数据封装就是将数据和对这些数据相关的操作组装在一起，形成一个实体，这个实体我可可以称为【 A 】

A. 类 B. 函数 C. 对象 D. 数据块

有关类和结构体的关系，错误的是【 B 】

A. 结构体一般只有数据成员，通常不会加入函数方法

B. 类一般只有函数方法，通常不会包含数据成员

C. 类的成员默认都是私有的

D. 结构体的成员默认都是公开的

类的构造函数被自动调用执行的情况是在定义类的【 C 】

A.成员函数时 B.数据成员时

C.对象时 D.友元函数时

关于this指针使用说法正确的是【 A 】

A. 保证每个对象拥有自己的数据成员，但共享处理这些数据的代码

B. 保证基类私有成员在子类中可以被访问。

C. 保证基类保护成员在子类中可以被访问。

D. 保证基类公有成员在子类中可以被访问。

下面对静态数据成员的描述中，正确的是【 C 】

A. 静态数据成员可以在类体内进行初始化

B. 静态数据成员不可以被类的对象调用

C. 静态数据成员可以直接用类名调用

D. 静态数据成员不能受private控制符的作用

下面对友元函数的描述中正确的是【 C 】

A. 友元函数的实现必须在类的内部定义

B. 友元函数是类的成员函数

C. 在类中必须用friend声明友元函数

D.友元函数不能访问类的私有成员

如果类A被说明成类B的友元，则【 D 】

A. 类B也一定是类A的友元

B. 类B的成员即类A的成员

C. 类A的成员即类B的成员

D. 类A的成员函数可以访问类B的成员

有关多态性说法不正确的是【 C 】

A. C++语言的多态性分为编译时的多态性和运行时的多态性

B. 编译时的多态性可通过函数重载实现

C. 运行时的多态性可通过模板和虚函数实现

D. 实现运行时多态性的机制称为动态多态性

对基类和派生类的关系描述中，错误的是【 B 】

A. 派生类是基类的具体化

B. 基类继承了派生类的属性

C. 派生类是基类定义的延续

D. 派生类是基类的特殊化

所谓多态性是指 【 B 】

A. 不同的对象调用不同名称的函数

B. 不同的对象调用相同名称的函数

C. 一个对象调用相同名称的函数

D. 一个对象调用不同名称的对象

下列打开文件的表达式中，错误的是【 A 】

A. cout.open(“C:\vc\abc.txt”,ios::binary);

B. ifstream ifile (“C:\vc\abc.txt”);

C. fstream iofile;iofile.open(“abc.txt”,ios::ate);

D. ofstream ofile; ofile.open(“C:\vc\abc.txt”,ios::binary);

以下有关析构函数的叙述，不正确的是【 A 】

A. 析构函数和构造函数一样可以有形参

B. 析构函数名前必须冠有符号“~”

C. 析构函数不允许用返回值

D. 在一个类只能定义一个析构函数

对类的构造函数和析构函数描述正确的是【 B 】

A. 构造函数不能重载，析构函数可以重载

B. 构造函数可以重载，析构函数不能重载

C. 构造函数可以重载，析构函数也可以重载

D. 构造函数不能重载，析构函数也不能重载

下列描述中，成员函数中包括【 B 】的类肯定是抽象类。

A. virtual void draw();

B. virtual void draw() = 0;

C. virtual void draw(){}

D. void draw() = 0;

下列关于C++函数的叙述中，正确的是【 C 】

A. 每个函数至少要具有一个参数

B. 每个函数都必须返回一个值

C. 函数在被调用之前必须先声明

D.函数不能自己调用自己

下列关于类和对象的叙述中，错误的是【 A 】

A. 一个类只能有一个对象

B. 对象是类的具体实例

C. 类是对某一类对象的抽象

D. 类和对象的关系是一种数据类型与变量的关系

假定 A 为一个类，则执行 A a1 ；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

通常，拷贝构造函数的形参是【 D 】

A. 本类对象名 B. 本类对象的指针名

C. 本类对象的成员名 D. 本类对象的引用名

下列不是类的成员函数的是【 B 】

A. 构造函数 B. 友元函数

C. 析构函数 D. 拷贝构造函数

下面描述中，表达错误的是【 A 】

A. 公用继承时基类中的private成员在派生类中仍是private的

B. 公用继承时基类中的public成员在派生类中仍是public的

C. 公用继承时基类中的protected成员在派生类中仍是protected的

D. 私有继承时基类中的public成员在派生类中是private的

带默认参数的函数原型声明若有：void fun(int a,int b=7,char z=’’);下面函数调用中，不合法的是【 C 】

A. fun(1); B. fun(1,2);

C. fun( ); D. fun(1,2,‘a’);

cout是I0流库预定义的【 B 】

A. 类 B. 对象 C. 包含文件 D. 常量

C++中解决命名冲突的机制是【 D 】

A.虚基类 B.虚函数 C.函数重载 D.名称空间

下面关于命名空间的说法，错误的是【 A 】

A. 一个命名空间对应多个命名作用域

B. 一个命名空间中的标识符命名作用域相同

C. 一个命名空间中可以集合很多不同的标识符

D. 命名空间的引入让程序员可以在不同的模块中使用相同名字表示不同事物

关键字【 C 】说明对象或变量初始化后不会被修改。

A. static B. public C. const D.inline

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

对基类和派生类的关系描述中，错误的是【 B 】

A.派生类是基类的具体化

B.基类继承了派生类的属性

C.派生类是基类定义的延续

D.派生类是基类的特殊化

下列关于多继承二义性的描述中，错误的是【 B 】

A. 一个派生类的两个基类中都有某个同名成员，派生类中这个成员的访问可能出现二义性。

B. 基类和派生类中同时出现的同名函数，也存在二义性问题。

C. 解决二义性的最常用的方法是对成员名的限定法

D. 一个派生类是从两个基类派生出来的，而这两个基类又有一个共同的基类，对该基类成员进行访问时，可能出现二义性。

下列有关C++类的说法中，不正确的是【 D 】

A. 类是一种用户自定义的数据类型

B. 只有类中的成员函数或类的友元函数才能存取类中的私有成员

C. 在类中，如果不做特别说明，所有成员的访问权限均为私有的

D. 在类中，如果不做特别说明，所有成员的访问权限均为公用的

C++ 中对于类中定义的成员,其默认的访问权限为【 C 】

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

下列关于类的访问权限的描述中,错误的是【 D 】

A. 说明为公有的成员可以被程序中的任何代码访问

B. 说明为私有的成员只能被类的成员和说明为友元类的成员函数访问

C. 说明为保护的成员,除了能被本身的成员函数和说明为友元类的成员函数访问外,该类的派生类的成员也可以访问

D. 类的所有成员都可以被程序中的任何代码访问

判断题

构造函数、析构函数能被继承或取地址（×）

构造函数可以重载，但析构函数只有一个（√）

返回值类型为引用类型的函数可以返回局部变量（×）

静态成员函数没有this指针（√）

静态成员函数可以直接访问非静态成员变量（×）

//由于静态成员函数没有this指针，所以不能直接访问非静态成员变量，必须通过向函数传递当前对象作为参数才可以访问当前对象的成员变量

const类型的变量声明时可以不用初始化（×）

const类型的对象只能访问const类型的成员函数，构造函数、析构函数除外（√）

const类型的对象如果作为实参，只能传递给const类型的形参（√）

volatile类型的对象只能访问volatile类型的成员函数，构造函数、析构函数除外（√）

复合类调用构造函数时，成员对象的构造函数调用顺序取决于复合类的构造函数初始化列表（×）

//成员对象的构造函数调用顺序取决于其在复合类的声明顺序

作为友元的类或类的函数，必须先声明后使用（√）

友元关系可以继承（×）

//友元关系不继承、不传递、不对称

重载函数允许只有返回值类型不同（×）

//重载函数可以在参数个数、参数类型、是否为const类型上有不同，但如果只有返回值类型不同，则调用函数时有歧义

可以重载C++中不存在的运算符（×）

运算符重载可以改变优先级（×）

运算符重载不改变操作数个数（√）

不能重载的运算符有 ？: :: . # ## .\* ->\* \*

返回值为临时对象的重载运算符有 后缀++ 后缀-- + - \* / %

返回值为引用类型的重载运算符有 前缀++ 前缀-- = [] () << >>

必须以友元方式重载的运算符有 << >>

必须以成员函数方式重载的运算符有 = [] () ->

派生类对象可用在基类对象使用的任何场合，如派生类对象可以赋值给基类对象、基类指针或基类引用，但此时这一基类的对象、指针或引用只能访问基类成员

复合类或派生类的定义前需要包含成员对象所在类或基类的定义

派生类构造函数的执行顺序：基类的构造函数（都为或都不为虚基类时按继承时声明的顺序）-> 内嵌成员对象的构造函数（按成员对象在类中声明的顺序）-> 自身类的构造函数

派生类对象在调用其基类的构造函数时，若其有多个在同一层次上的基类，则虚基类的构造先于非虚基类的构造

派生类和基类的虚函数原型可以不一致（×）

//如果仅是同名，则只是普通重载；如果仅是返回值类型不同，则程序会出错

虚函数可以为static类型（×）

//静态函数、内联函数、构造函数不可以是虚函数

在普通函数调用虚函数时，实际调用的是this指针指向的对象所在类的虚函数（√）

在基类的构造函数或析构函数中调用虚函数，调用的是基类的虚函数（√）

虚特性不可以继承（×）

实现多态的两种方式：重载、虚函数

纯虚函数是空函数（×）

函数调用的匹配原则：

① 寻找完全匹配的函数

② 函数模板实例化后匹配

③ 重载函数参数隐式类型转化后匹配

④ 在上述某一步找到两个及以上匹配的函数或三步都找不到匹配的函数时，则程序出错

析构函数不可以主动调用（×）

//主动调用析构函数后，程序最终还会再次调用析构函数

友元函数的定义必须在类外（×）

//友元函数的定义可以在类内或类外

静态成员函数的定义在类外时，需要加上static关键字（×）

在类中定义的函数是内联函数（√）

//类中的内联函数实现有两种方式，一种是在类中进行定义；另一种是在类外进行定义，函数定义前加上incline关键字。注意第二种方式的函数定义需要放在头文件

定义引用类型的变量时必须初始化（√）

在调用派生类的拷贝构造函数时，其基类的构造函数不会被调用（×）

40.在异常处理机制中，throw的变量一定会被catch（×）