C++模拟试题-1-2 解答分析

1．c++系统预定了 4 个用于标准数据流的对象，下列选项中不属于此类对象的是(_____)

A.cout

B.cin

C.cerr

D.cset

- **正确答案：** 【D】
- **题目分析：**
  C++ 标准库在 `<iostream>` 头文件中预定义了四个用于处理标准数据流的对象：
  - `cin`：标准输入流对象，通常关联到键盘。
  - `cout`：标准输出流对象，通常关联到屏幕。
  - `cerr`：标准错误流对象，非缓冲，通常关联到屏幕，用于输出错误信息。
  - `clog`：标准错误流对象，缓冲，通常关联到屏幕，也用于输出错误信息。
    选项 D 中的 `cset` 并非标准数据流对象。
- **知识点：**
  - **标准 I/O 流对象：** `cin`（类型 `istream`）、`cout`（类型 `ostream`）、`cerr`（类型 `ostream`）、`clog`（类型 `ostream`）。
  - **`<iostream>` 头文件：** 使用这些对象前需要包含此头文件。
  - **缓冲与非缓冲：** `cerr` 是非缓冲的，输出会立即显示；`cout` 和 `clog` 通常是缓冲的，输出内容可能先存储在缓冲区中，待缓冲区满或特定条件下再实际输出。
- **易错点：**
  - 将标准流对象与 `<iomanip>` 头文件中的 I/O 操纵符（如 `setw`，`setprecision`，`fixed`，`endl` 等）混淆。`cset` 并非标准库的一部分。
  - 忘记 `clog` 也是一个标准错误流对象，或者不清楚 `cerr` 和 `clog` 的区别（缓冲与非缓冲）。

2．设有说明:char w;int x;float y;double z;则表达式 w*x+z-y 值的数据类型为(_____)

A.float

B.char

C.int
D.double

- **正确答案：** 【D】
- **题目分析：**
  C++ 在进行混合类型的算术运算时，会遵循一套隐式类型转换规则，通常是向精度更高（表示范围更大）的类型转换，以避免数据丢失。
  i. `w` (char) 和 `x` (int): `w` 会被提升 (promoted) 为 `int` 类型。 `w * x` 的结果是 `int` 类型。
  ii. `(w * x)` (int) 和 `z` (double): `(w * x)` 的结果 (int) 会被转换为 `double` 类型。 `(w * x) + z` 的结果是 `double` 类型。
  iii. `((w * x) + z)` (double) 和 `y` (float): `y` (float) 会被转换为 `double` 类型。 `((w * x) + z) - y` 的最终结果是 `double` 类型。
  因此，整个表达式的最终数据类型是 `double` 。
- **知识点：**
  - **隐式类型转换 (Type Promotion/Conversion):** 在算术表达式中，较小（精度较低）的数据类型会自动转换成较大（精度较高）的数据类型。
  - **类型转换层次 (部分)：** `char -> short -> int -> long -> long long -> float -> double -> long double` 。
  - 当不同类型的操作数进行运算时，结果类型通常是参与运算的类型中精度最高的那一个。
- **易错点：**
  - 忽略字符类型 (`char`) 在算术运算中会先转换成 `int` 类型。
  - 错误地判断类型转换的优先级，例如可能误认为结果是 `float` 。
  - 不清楚 `double` 的精度高于 `float` 。

3．下列对析构函数的描述中,正确的是(____)

A.一个类中只能定义一个析构函数
B.析构函数名与类名不同
C.析构函数的定义只能在类体内
D.析构函数可以有一个或多个参数

- **正确答案：** 【A】
- **题目分析：**
  - **A. 一个类中只能定义一个析构函数：** 正确。析构函数不能重载，因此一个类最多只能有一个析构函数。
  - **B. 析构函数名与类名不同：** 错误。析构函数的名称是在类名前加上波浪号 `~` (例如，类名为 `MyClass` ，则析构函数名为 `~MyClass`)。

- **C. 析构函数的定义只能在类体内：** 错误。析构函数可以在类体内定义，也可以在类体外定义（在类内声明，类外通过作用域解析运算符 `::` 实现）。
- **D. 析构函数可以有一个或多个参数：** 错误。析构函数不能有任何参数。

- **知识点：**
  - **析构函数的作用：** 主要用于对象销毁前执行清理工作，如释放动态分配的内存、关闭文件等。
  - **析构函数的特征：**
    - 名称为 `~类名` 。
    - 没有返回类型（连 `void` 都不写）。
    - 不能有参数。
    - 不能被重载（因此一个类只有一个）。
    - 通常声明为 `public` 。
    - 当对象的生命周期结束时（例如，局部对象离开其作用域， `delete` 指向动态对象的指针），析构函数会被自动调用。
- **易错点：**
  - 将析构函数的特性与构造函数混淆（构造函数可以重载，可以有参数）。
  - 忘记析构函数名前的 `~` 符号。
  - 认为析构函数可以像普通成员函数一样被随意调用（虽然语法上可以显式调用，但通常不推荐，应由系统自动管理）。

4．下列定义数组的语句中正确的是(_____)

A.#define size 10 char str1[size],str2[size+2];
B.char str[];
C.int num['10'];
D.int n=5; int a[n][n+2];

- **正确答案：** 【A】
- **题目分析：**
  - **A. `#define size 10 char str1[size],str2[size+2];`：** 正确。 `#define` 是一个预处理指令，它会在编译前将代码中所有的 `size` 替换为 `10` 。因此，该语句等效于 `char str1[10], str2[10+2];` 即 `char str1[10], str2[12];` ，这是合法的数组定义。
  - **B. `char str[];`：** 错误（在大多数情况下）。如果这是在函数外部或内部的独立声明，没有初始化，编译器无法确定数组大小，会导致编译错误。这种形式仅在以下情况有效：
    - 作为函数参数声明： `void func(char str[])`
    - 在声明的同时进行初始化： `char str[] = "hello";` (编译器会自动计算大小)
    - 作为 `extern` 声明，表示数组在别处定义。

- **C. `int num['10'];`：** 错误。数组的大小必须是一个整型常量表达式。`'10'` 是一个字符常量（更准确地说，是一个多字符常量，其值是实现定义的，通常不是程序员期望的整数 10），而不是整数 `10`。应写为 `int num[10];`。
- **D. `int n=5; int a[n][n+2];`：** 错误（在标准 C++中）。在标准 C++中，数组的维度必须是编译时常量。`n` 是一个变量，即使它用常量初始化。这种语法称为变长数组 (VLA)，是 C99 标准的一部分，并在某些 C++编译器中作为扩展支持，但不是标准 C++特性。若要使其在 C++中合法，`n` 应声明为 `const int n = 5;`。

- **知识点：**
  - **数组定义：** 数据类型 数组名[常量表达式];
  - **常量表达式：** 数组的大小必须在编译时就能确定。这可以是字面常量 (如 `10`)、`const` 常量、枚举常量或 `sizeof` 表达式等。
  - **`#define` 预处理指令：** 用于定义宏，进行文本替换。
  - **变长数组 (VLA)：** C 语言特性，非标准 C++。
- **易错点：**
  - 使用变量作为数组大小（标准 C++中应使用 `const` 变量）。
  - 数组大小表达式类型错误，如使用字符常量代替整型常量。
  - 不完全理解 `char str[];` 的合法使用场景。

5．关于 this 指针使用说法正确的是(____)

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。

- **正确答案：** 【A】
- **题目分析：**
  - **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：** 正确。`this` 指针是 C++中非静态成员函数的一个隐含参数，它指向调用该成员函数的对象。通过 `this` 指针，成员函数可以访问特定对象的数据成员。类的所有对象共享同一份成员函数的代码，但每个对象有自己独立的数据成员副本。`this` 指针使得共享的代码能够操作不同对象的数据。
  - **B. 保证基类私有成员在子类中可以被访问：** 错误。`this` 指针本身不改变访问控制规则。基类的 `private` 成员在子类中始终是不可直接访问的。
  - **C. 保证基类保护成员在子类中可以被访问：** 错误。虽然基类的 `protected` 成员可以在子类中被访问，但这由 `protected` 访问修饰符决定，而不是 `this` 指针的功能。`this` 指针在子类成员函数中指向子类对象，通过它可以访问子类能访问的成员（包括继承来的 `protected` 和 `public` 成员）。

    - **D. 保证基类公有成员在子类中可以被访问：** 错误。原因同 C，访问权限由 `public` 修饰符决定。
- **知识点：**
    - **`this` 指针：**
        - 在类的非静态成员函数内部，`this` 是一个指向调用该函数的对象的指针。
        - 它的类型是 `类类型* const`（一个指向类类型的常量指针，即 `this` 指针本身的值不能被修改，但它指向的对象可以被修改）。
        - 主要用途是区分同名的成员变量和局部变量/参数，或者在成员函数中返回对象自身的引用或指针。
        - 静态成员函数没有 `this` 指针，因为静态成员函数不与任何特定对象关联。
    - **对象的数据共享与独立：** 类的所有对象共享成员函数的代码（存储在代码段），但每个对象有其独立的数据成员（存储在各自的内存空间）。
- **易错点：**
    - 混淆 `this` 指针的作用与访问控制修饰符（`private`，`protected`，`public`）的作用。`this` 指针用于标识当前对象，而访问控制修饰符决定成员的可访问性。
    - 认为 `this` 指针可以突破封装性。

6．所谓多态性是指(____)

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.一个对象调用不同名称的函数
D.一个对象调用不同名称的对象

- **正确答案：** 【B】
- **题目分析：**
  多态性（Polymorphism）是面向对象编程的核心概念之一，字面意思是“多种形态”。在 C++中，多态性主要指运行时多态，它允许不同类的对象对相同的消息（即函数调用）做出不同的响应（即执行不同的函数实现）。
    - **A. 不同的对象调用不同名称的函数：** 这不是多态，只是正常的函数调用。
    - **B. 不同的对象调用相同名称的函数：** 正确。这是多态的典型表现。例如，基类指针指向不同的派生类对象，通过该指针调用一个虚函数，会执行各自派生类中该虚函数的版本。
    - **C. 一个对象调用不同名称的函数：** 这也不是多态，一个对象可以调用它所拥有的多个不同名称的成员函数。
    - **D. 一个对象调用不同名称的对象：** 这种说法不准确，对象之间通常是通过成员函数进行交互。
- **知识点：**

- **多态性 (Polymorphism)：** 允许将接口与实现分离。主要分为编译时多态和运行时多态。
  - **编译时多态 (静态多态)：** 通过函数重载和模板实现。在编译阶段确定调用哪个函数。
  - **运行时多态 (动态多态)：** 通过继承和虚函数实现。在程序运行时，根据对象的实际类型来确定调用哪个函数。这是通常意义上所说的多态。
- **实现运行时多态的条件：**
  a. 类之间存在继承关系。
  b. 基类中声明了虚函数 (`virtual` 关键字)。
  c. 通过基类指针或引用指向派生类对象，并调用虚函数。

- **易错点：**
  - 将函数重载（编译时多态）与虚函数实现的运行时多态混淆。虽然都是“相同名称，不同行为”，但机制和发生时间不同。
  - 不理解多态的核心在于“相同的接口，不同的实现”，即调用形式一致，但具体执行的代码随对象类型而异。

7．派生类构造函数的执行顺序是先执行**____** 的构造函数，然后执行成员对象的构造函数，最后执行 ____ 的构造函数。

- **正确答案：** 第一个空：【基类】；第二个空：【派生类自身】
- **题目分析：**
  派生类对象在创建时，其构造函数的执行遵循特定的顺序，以确保对象的所有组成部分都被正确初始化：
  i. **调用基类的构造函数：**
    - 如果派生类的构造函数初始化列表中显式指定了调用哪个基类构造函数，则调用指定的。
    - 如果没有显式指定，则调用基类的默认构造函数。
    - 如果存在多重继承，基类构造函数的调用顺序取决于它们在派生类继承列表中的声明顺序。
    - 如果存在虚拟继承，虚基类的构造函数会优先被调用，并且只调用一次。
  ii. **调用成员对象的构造函数：**
    - 如果类中包含其他类的对象作为成员（成员对象），则会按照这些成员对象在类中声明的顺序调用它们的构造函数。
    - 可以在派生类构造函数的初始化列表中为成员对象指定构造函数。
  iii. **执行派生类自身的构造函数体：**
    - 在所有基类和成员对象都构造完毕后，才执行派生类构造函数体内的代码。
- **知识点：**
  - **构造函数执行顺序：** 基类 -> 成员对象 -> 派生类自身。

- **构造函数初始化列表：** 用于显式调用基类构造函数和初始化成员对象。推荐使用初始化列表进行初始化，尤其对于 `const` 成员和引用成员，它们必须在初始化列表中初始化。
- **析构函数执行顺序：** 与构造函数相反：派生类自身 -> 成员对象 -> 基类。

- **易错点：**
  - 记错构造顺序，特别是成员对象构造与派生类自身构造的先后。
  - 在多重继承或虚拟继承时，对基类构造函数的调用顺序不清晰。
  - 忘记如果不在初始化列表中显式调用基类构造函数，编译器会尝试调用基类的默认构造函数，如果基类没有默认构造函数且需要参数，则会导致编译错误。

8．C++语言程序的注释可以出现在程序中的任何地方，一个注释以 **_____** 作为开始和结束的标记。

- **正确答案：** 【 `/*` 和 `*/` 】 (题目提供的答案 `/**/` 是这种注释的完整形式)
- **题目分析：**

  C++ 支持两种类型的注释：

  i. **块注释 (Block Comment)：** 以 `/*` 开始，以 `*/` 结束。可以跨越多行。题目描述的“以...作为开始和结束的标记”指的就是这种注释。

  ii. **行注释 (Line Comment)：** 以 `//` 开始，直到该行末尾。只能注释单行。

  题目要求填写开始和结束标记，因此 `/*` 是开始标记， `*/` 是结束标记。题目给出的答案 `/**/` 是一个完整的块注释示例。
- **知识点：**
  - **注释的作用：** 提高代码的可读性，对代码功能、逻辑进行解释说明。注释内容会被编译器忽略，不参与程序执行。
  - **块注释 `/* ... */`：**
    - 可以用于多行注释。
    - 可以用于行内注释的一部分，例如 `int x = 10; /* 初始化x */` 。
    - 注意块注释不能嵌套（标准 C++不允许，虽然某些编译器可能支持，但不可移植）。例如 `/* 外部注释 /* 内部注释 */ 外部注释 */` ，第一个 `*/` 会结束整个注释。
  - **行注释 `// ...`：**
    - 从 `//` 开始到行尾的所有内容都是注释。
    - 使用方便，常用于单行解释或临时屏蔽代码。
- **易错点：**
  - 块注释未正确配对 `/*` 和 `*/` ，导致大段代码被意外注释掉或产生编译错误。
  - 在块注释中尝试嵌套块注释。
  - 混淆两种注释的用法，例如试图用 `//` 进行多行注释（需要每行都加 `//` ）。

9．下列程序在构造函数和析构函数中申请和释放类的数据成员 int *a，申请时使用形参 b 初始化 a，请填空。

```
class A
{
public:
A(int b);
~A();
private:
int *a;
};

A::A(int b) // 构造函数定义，注意这里题目中 A::(int b) 应该是 A::A(int b)
{________ ;} // 第一个空

A::~A()
{________ ;} // 第二个空
```

- **正确答案：**
  - 第一个空： `a = new int(b);`
  - 第二个空： `delete a;`
- **题目分析：**
  - **构造函数 `A::A(int b)`：** 负责对象的初始化。题目要求申请类的数据成员 `int *a` 并使用形参 `b` 初始化 `a` 指向的整数。因此，需要使用 `new int(b)` 来动态分配一个 `int` 类型的内存空间，并将其初始化为 `b` 的值，然后将这块内存的地址赋给指针 `a`。
  - **析构函数 `A::~A()`：** 负责对象的清理工作，特别是释放在构造函数中动态分配的资源。由于 `a` 指向的是通过 `new` 分配的单个 `int` 对象，因此应使用 `delete a;` 来释放这块内存，防止内存泄漏。
- **知识点：**
  - **构造函数：** 用于创建和初始化对象。
  - **析构函数：** 用于对象销毁前进行资源清理。
  - **`new` 运算符：** 用于动态内存分配。
    - `new T`：分配一个 T 类型的对象，并返回其地址。
    - `new T(initializer)`：分配一个 T 类型的对象，并用 `initializer` 初始化它。
    - `new T[size]`：分配一个包含 `size` 个 T 类型对象的数组，并返回首元素地址。
  - **`delete` 运算符：** 用于释放由 `new` 分配的内存。
    - `delete pointer`：释放由 `new T` 分配的单个对象。
    - `delete[] pointer`：释放由 `new T[size]` 分配的对象数组。

    - **RAII (Resource Acquisition Is Initialization)：** 资源获取即初始化。这是一种 C++编程技术，将资源的生命周期与对象的生命周期绑定。资源在对象构造时获取，在对象析构时释放。本题就是 RAII 的一个简单示例。
- **易错点：**
    - 在构造函数中忘记初始化指针 `a`，或者只分配内存未初始化值（如 `a = new int;` 如果题目要求初始化）。
    - 在析构函数中忘记 `delete a;`，导致内存泄漏。
    - 如果 `a` 是指向数组（例如 `a = new int[size];`），则析构函数中应使用 `delete[] a;`。本题中 `a` 指向单个 `int`，所以用 `delete a;`。
    - 对 `new int(b)` 和 `new int[b]` 的混淆：前者是分配一个 `int` 并初始化为 `b`，后者是分配一个大小为 `b` 的 `int` 数组。

10．C++对 C 语言做了很多改进，即从面向过程变成为面向对象的主要原因是(____)

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.引进了类和对象的概念

- **正确答案：** 【D】
- **题目分析：**
  C++ 最核心的、区别于 C 语言的特性，并使其成为一门面向对象编程语言的基石，就是引入了**类 (class)** 和 **对象 (object)** 的概念。
    - **A. 增加了一些新的运算符：** 如 `new`，`delete`，`::` 等，这些是支持面向对象特性的辅助，但不是根本原因。
    - **B. 允许函数重载,并允许设置缺省参数：** 这些是 C++对 C 函数功能的增强，属于过程式编程范畴的改进，有助于提高编程便利性和代码表达能力，但不是转向面向对象的根本原因。函数重载可以看作是多态性的一种早期形式（编译时多态），但面向对象的核心是运行时多态和封装、继承。
    - **C. 规定函数说明符必须用原型：** 这是为了增强类型检查和代码的健壮性，也是对 C 语言的改进，但与面向对象的核心思想关联不大。
    - **D. 引进了类和对象的概念：** 类是创建对象的蓝图，它封装了数据（成员变量）和操作这些数据的方法（成员函数）。对象是类的实例。这是面向对象编程（OOP）的基础，OOP 的三大特性——封装、继承、多态——都是基于类和对象的概念展开的。
- **知识点：**
    - **面向对象编程 (OOP)：** 一种编程范式，使用“对象”来设计软件和应用程序。

    - **类 (Class)：** 用户定义的数据类型，包含了数据成员和成员函数。
    - **对象 (Object)：** 类的实例。
    - **OOP 三大特性：**
        - **封装 (Encapsulation)：** 将数据和操作数据的方法捆绑在一起，并对外部隐藏对象的内部实现细节。
        - **继承 (Inheritance)：** 允许创建新类（派生类）来重用、扩展和修改现有类（基类）的行为。
        - **多态 (Polymorphism)：** 允许不同类的对象对相同的消息做出不同的响应。
- **易错点：**
    - 将 C++的某些语法改进（如函数重载、默认参数）误认为是其成为面向对象语言的核心原因。这些特性虽然重要，但它们服务于或补充了面向对象的整体框架。

11．下列有关模板和继承的叙述正确的是(_____)

A.模板和继承都可以派生出一个类系
B.从类系的成员看,模板类系的成员比继承类系的成员较为(_____)稳定
C.从动态性能看, 继承类系比模板类系具有更多的动态特性
D.相同类模板的不同实例一般没有联系,而派生类各种类之间有兄弟父子等关系

- **正确答案：** 【D】
- **题目分析：**
    - **A. 模板和继承都可以派生出一个类系：** 说法不准确。继承明确地创建具有父子关系的类层次结构（类系）。模板本身是参数化类型，其实例化是产生具体的类，这些实例化的类之间通常没有直接的继承关系，除非模板参数本身是相关的类。
    - **B. 从类系的成员看,模板类系的成员比继承类系的成员较为(_____)稳定：** 这个说法比较模糊，"稳定"的含义不明确。模板实例化出的类，其成员由模板定义和具体类型参数决定。继承类系的成员则遵循继承规则。难以一概而论谁更“稳定”。
    - **C. 从动态性能看, 继承类系比模板类系具有更多的动态特性：** 正确。继承（特别是与虚函数结合使用时）是实现运行时多态的基础，允许在运行时根据对象的实际类型确定行为，这是典型的动态特性。模板主要在编译时进行类型替换和代码生成，产生的是静态的、类型特化的代码，其多态性（如模板特化、SFINAE）更多体现在编译期。
    - **D. 相同类模板的不同实例一般没有联系,而派生类各种类之间有兄弟父子等关系：** 正确。例如， `std::vector<int>` 和 `std::vector<double>` 是同一个类模板 `std::vector` 的两个不同实例（具体类）。它们之间没有继承关系。而通过继承，如 `class Derived : public Base {}` ， `Derived` 和 `Base` 之间有明确的父子关系。如果 `Derived1` 和 `Derived2` 都继承自 `Base` ，它们之间是兄弟关系。
- **知识点：**

- **模板 (Templates)：** C++的泛型编程机制，允许编写与类型无关的代码。类模板用于创建泛型类，函数模板用于创建泛型函数。模板在编译时被实例化。
- **继承 (Inheritance)：** 面向对象的核心机制之一，允许一个类（派生类）继承另一个类（基类）的属性和方法。建立了类之间的 "is-a" 关系。
- **运行时多态 vs. 编译时多态：** 继承和虚函数主要用于实现运行时多态。模板主要实现编译时多态（参数化多态）。
- **类实例化：** 模板实例化是根据具体的类型参数生成实际的类或函数的过程。

- **易错点：**
  - 混淆模板和继承的作用和机制。模板关注代码复用和对不同类型的通用性，继承关注类之间的层次关系和行为扩展/覆盖。
  - 认为模板实例化后的不同类之间也存在类似继承的关系。

12．关于 new 运算符的下列描述中,错误的是(____)

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

- **正确答案：** 【D】
- **题目分析：**
  - **A. 它可以用来动态创建对象和对象数组：** 正确。 `new T` 创建单个对象， `new T[N]` 创建对象数组。
  - **B. 使用它创建的对象或对象数组可以使用运算符 delete 删除：** 正确。 `delete ptr` 删除单个对象， `delete[] ptr` 删除对象数组。 `new` 和 `delete` (以及 `new[]` 和 `delete[]`) 必须配对使用。
  - **C. 使用它创建对象时要调用构造函数：** 正确。当使用 `new` 创建一个类的对象时，会先分配内存，然后自动调用该类的相应构造函数来初始化对象。
  - **D. 使用它创建对象数组时必须指定初始值：** 错误。使用 `new T[N]` 创建对象数组时：
    - 如果类 `T` 有用户定义的构造函数，编译器会尝试调用默认构造函数（无参构造函数或所有参数都有默认值的构造函数）来初始化数组中的每个元素。如果类没有可访问的默认构造函数，则编译会失败。
    - 对于内置类型（如 `int` , `char` ）或没有用户定义构造函数的聚合类型，如果不提供初始化，数组元素的值是未定义的（对于全局或静态数组，会被零初始化）。
    - C++11 及以后版本允许使用初始化列表来初始化动态分配的数组，例如
      `int* arr = new int[3]{1, 2, 3};` 或

`MyClass* obj_arr = new MyClass[2]{ {arg1}, {arg2} };` 。但这不是“必须”的，不指定初始值也是合法的（只要默认构造可行或不需要）。

- **知识点：**
  - `new` **运算符：** 动态内存分配。
  - `delete` **运算符：** 动态内存释放。
  - **构造函数调用：** `new` 创建对象时会自动调用构造函数。
  - **对象数组的初始化：**
    - 默认构造：如果类有默认构造函数，会被用于初始化数组元素。
    - 无默认构造：如果类没有默认构造函数，且尝试创建对象数组而不提供初始化，会导致编译错误。
    - C++11 初始化列表：可以为动态数组提供初始值。
- **易错点：**
  - 认为创建动态数组时必须像静态数组那样提供每个元素的初始值。
  - 忘记如果类没有默认构造函数，则不能直接 `new T[N]` （除非使用 C++11 的初始化列表且能满足构造需求）。
  - 混淆 `delete` 和 `delete[]` 。

13．下列运算符中,在 C++语言中不能重载的是(____)

A.* (乘法或解引用)
B.>= (大于等于)
C.:: (作用域解析)
D./ (除法)

- **正确答案：** 【C】
- **题目分析：**
  C++允许重载大部分运算符，但有少数几个运算符是不能被重载的。
  - **A.** `*` **：** 可以重载（作为一元解引用或二元乘法）。
  - **B.** `>=` **：** 可以重载（作为比较运算符）。
  - **C.** `::` **(作用域解析运算符)：** 不能重载。它用于指定命名空间或类的作用域，其语义是固定的。
  - **D.** `/` **：** 可以重载（作为除法运算符）。

  不能重载的运算符包括：
  i. `.` (成员访问运算符)
  ii. `.*` (成员指针访问运算符)
  iii. `::` (作用域解析运算符)
  iv. `?:` (条件运算符/三元运算符)

v. `sizeof` (长度运算符)
vi. `typeid` (类型识别运算符)
vii. `static_cast` , `dynamic_cast` , `const_cast` , `reinterpret_cast` (C++类型转换运算符)
viii. `alignof` (C++11, 对齐方式运算符)
ix. `noexcept` (C++11, 异常说明运算符)

- **知识点：**
  - **运算符重载 (Operator Overloading)：** 允许用户为自定义类型（类或枚举）重新定义 C++标准运算符的行为。
  - **可重载与不可重载的运算符：** 大部分运算符都可以重载，但少数具有特殊语法或编译期含义的运算符不可重载。
  - **重载规则：**
    - 不能创建新的运算符。
    - 不能改变运算符的优先级和结合性。
    - 不能改变运算符所需的操作数个数。
    - 至少有一个操作数是用户定义的类型（类类型或枚举类型）。
    - `=` , `[]` , `()` , `->` 必须作为类的非静态成员函数重载。
- **易错点：**
  - 记不清哪些运算符不能重载，特别是 `.` 和 `::` 。
  - 试图改变运算符的原始特性（如操作数个数、优先级）。

14．已知 int i=0, x=1, y=0；在下列选项使 i 的值变成 1 的语句是(____)

A.if(x&&y) i++ ;
B.if(x==y) i++ ;
C.if(x||y) i++ ;
D.if(!x) i++ ;

- **正确答案：** 【C】
- **题目分析：**
  初始值： `i = 0` , `x = 1` , `y = 0` 。目标是使 `i` 的值变成 `1` ，即 `i++` 被执行。
  - **A. `if( x && y ) i++ ;`**
    - `x && y` (逻辑与): `1 && 0` 。在 C++中，非零值为 `true` ，零值为 `false` 。
    - `true && false` 结果为 `false` (0)。
    - `if(false)` 条件不成立， `i++` 不执行。 `i` 仍为 `0` 。
  - **B. `if( x == y ) i++ ;`**
    - `x == y` (等于比较): `1 == 0` 结果为 `false` (0)。

        - `if(false)` 条件不成立， `i++` 不执行。 `i` 仍为 `0` 。
    - **C. `if( x || y ) i++ ;`**
        - `x || y` (逻辑或): `1 || 0` 。
        - `true || false` 结果为 `true` (1)。
        - `if(true)` 条件成立， `i++` 执行。 `i` 变为 `1` 。
    - **D. `if( !x ) i++ ;`**
        - `!x` (逻辑非): `!1` 。
        - `!true` 结果为 `false` (0)。
        - `if(false)` 条件不成立， `i++` 不执行。 `i` 仍为 `0` 。
- **知识点：**
    - **逻辑运算符：**
        - `&&` (逻辑与)：两边都为 `true` 时结果为 `true` 。
        - `||` (逻辑或)：两边至少一个为 `true` 时结果为 `true` 。
        - `!` (逻辑非)： `true` 变 `false` ， `false` 变 `true` 。
    - **关系运算符：** `==` (等于)。
    - **`if` 语句：** 根据条件表达式的真假决定是否执行后续语句。在 C++中，非零值被视为 `true` ，零值被视为 `false` 。
    - **短路求值：**
        - 对于 `A && B` ，如果 `A` 为 `false` ，则 `B` 不会被求值。
        - 对于 `A || B` ，如果 `A` 为 `true` ，则 `B` 不会被求值。
- **易错点：**
    - 混淆赋值运算符 `=` 和比较运算符 `==` 。
    - 对逻辑运算符的真值表理解不清。
    - 忘记 C++中非零即为真的规则。

15．sizeof(float)是(____)

A.一个双精度型表达式
B.一个整型表达式
C.一种函数调用
D.一个不合法的表达式

- **正确答案：** 【B】
- **题目分析：**

  `sizeof` 是 C/C++中的一个运算符（不是函数），它返回一个对象或类型所占用的内存字节数。
    - `sizeof(float)` ：返回 `float` 类型在当前系统上占用的字节数。

- 这个返回值是一个编译时确定的常量，其类型是 `std::size_t` （通常是无符号整型，如 `unsigned int` 或 `unsigned long` ）。因此， `sizeof(float)` 的结果是一个整型值。
- **A. 一个双精度型表达式：** 错误。 `sizeof` 返回的是大小，是整数。
- **B. 一个整型表达式：** 正确。其结果是一个整数，表示字节数。
- **C. 一种函数调用：** 错误。 `sizeof` 是一个运算符，尽管其使用形式 `sizeof(...)` 看起来像函数调用，但它在编译时求值。
- **D. 一个不合法的表达式：** 错误。 `sizeof(type)` 是合法的。

- **知识点：**
  - **`sizeof` 运算符：**
    - 返回其操作数所占内存空间的大小，单位是字节。
    - 操作数可以是一个类型名（如 `sizeof(int)` ）或一个表达式（如 `sizeof x` 或 `sizeof(x)` ）。
    - 如果操作数是表达式， `sizeof` 不会计算表达式的值（除非是变长数组，C 语言特性）。
    - 结果类型是 `std::size_t` ，定义在 `<cstddef>` (或 `<stddef.h>`) 中。
  - **编译时求值：** `sizeof` 的结果通常在编译时确定。
- **易错点：**
  - 认为 `sizeof` 误认为是一个函数。
  - 不清楚 `sizeof` 返回值的类型（是整型，具体为 `std::size_t` ）。
  - 认为 `sizeof` 会执行其表达式操作数。

16．下面叙述不正确的是(_____)

A.基类的保护成员在派生类中仍然是保护的成员
B.基类的保护成员在公有派生类中仍然是保护的
C.基类的保护成员在私有派生类中仍然是私有的
D.对基类成员的访问必须是无二义性

- **正确答案：** 【A】
- **题目分析：**
  访问控制在继承中的变化规则：
  - **`public` 继承：**
    - 基类的 `public` 成员在派生类中仍然是 `public` 。
    - 基类的 `protected` 成员在派生类中仍然是 `protected` 。
    - 基类的 `private` 成员在派生类中不可直接访问。
  - **`protected` 继承：**
    - 基类的 `public` 成员在派生类中变成 `protected` 。
    - 基类的 `protected` 成员在派生类中仍然是 `protected` 。

        - 基类的 `private` 成员在派生类中不可直接访问。
    - **`private` 继承：**
        - 基类的 `public` 成员在派生类中变成 `private`。
        - 基类的 `protected` 成员在派生类中变成 `private`。
        - 基类的 `private` 成员在派生类中不可直接访问。

    现在分析选项：

    - **A. 基类的保护成员在派生类中仍然是保护的成员：** 这个说法不完全正确，因为它没有指明继承方式。如果是 `private` 继承，基类的保护成员在派生类中会变成 `private`。所以这句话作为通用描述是错误的。
    - **B. 基类的保护成员在公有派生类中仍然是保护的：** 正确。根据 `public` 继承规则。
    - **C. 基类的保护成员在私有派生类中仍然是私有的：** 正确。根据 `private` 继承规则，基类的 `public` 和 `protected` 成员在派生类中都变为 `private`。
    - **D. 对基类成员的访问必须是无二义性：** 正确。这是指在多重继承（尤其是菱形继承）中，如果多个基类有同名成员，直接访问该成员会导致二义性，需要使用作用域解析运算符明确指定。例如 `Derived::Base1::member`。

    题目要求选出“不正确”的叙述。A 选项的表述不够严谨，没有限定继承方式，因此在某些情况下（如私有继承）是不成立的。

- **知识点：**
    - **继承方式：** `public`，`protected`，`private`。
    - **访问控制修饰符：** `public`，`protected`，`private`。
    - **继承中访问权限的变化：** 派生类成员对基类成员的访问权限，以及派生类对象外部对继承来的成员的访问权限，都受到继承方式和基类成员自身访问权限的共同影响。派生类中成员的最终访问权限是基类中访问权限和继承方式中较严格的那个。
    - **二义性访问：** 在多重继承中可能出现，需要通过类名限定来解决。
- **易错点：**
    - 忘记不同继承方式下访问权限的具体变化规则。
    - A 选项具有迷惑性，因为它在公有继承和保护继承下是成立的，但在私有继承下不成立。题目问的是一般性叙述。

17．决定 C++语言中函数的返回值类型的是(____)

A.return 语句中的表达式类型
B.调用该函数时系统随机产生的类型
C.调用该函数时的主调用函数类型
D.在定义该函数时所指定的数据类型

- **正确答案：** 【D】

- **题目分析：**
  函数的返回值类型是在函数定义或声明时明确指定的。
  - **A. `return` 语句中的表达式类型：** `return` 语句中表达式的类型必须能够隐式转换为函数定义时声明的返回类型，或者完全匹配。如果不能转换或不匹配（且没有合适的转换），会导致编译错误。`return` 语句的表达式类型本身不决定函数的返回类型，而是要服从函数声明的返回类型。
  - **B. 调用该函数时系统随机产生的类型：** 错误。函数返回类型是静态确定的。
  - **C. 调用该函数时的主调用函数类型：** 错误。调用者的类型与被调用函数的返回类型无关。
  - **D. 在定义该函数时所指定的数据类型：** 正确。例如 `int myFunction() { ... }`，这里 `int` 就是在函数定义时指定的返回类型。函数声明（原型）也必须指定相同的返回类型。
- **知识点：**
  - **函数定义/声明：** `返回类型 函数名(参数列表);`
  - **`return` 语句：** 用于从函数中返回一个值。返回的值的类型必须与函数声明的返回类型兼容。
  - **`void` 返回类型：** 表示函数不返回值。
  - **类型兼容性与转换：** `return` 表达式的类型如果与函数声明的返回类型不完全一致，编译器会尝试进行隐式类型转换。
- **易错点：**
  - 认为 `return` 语句写了什么类型，函数就是什么返回类型，而忽略了函数头部的声明。
  - 对于 `void` 函数，如果 `return` 语句后跟了表达式，会导致编译错误（C++中 `return;` 是合法的，但 `return value;` 不合法）。

18．下面对于友元函数描述正确的是(____)

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.友元函数破坏了类的封装性和隐藏性
D.友元函数不能访问类的私有成员

- **正确答案：** 【C】
- **题目分析：**
  - **A. 友元函数的实现必须在类的内部定义：** 错误。友元函数只是在类内部声明（使用 `friend` 关键字），其定义通常在类外部，像普通函数一样。也可以在类内部定义，但它仍然不是成员函数。
  - **B. 友元函数是类的成员函数：** 错误。友元函数不是类的成员函数。它没有 `this` 指针，不能通过对象名直接调用（除非它是另一个类的成员函数，并且那个类的对象可以调用它）。
  - **C. 友元函数破坏了类的封装性和隐藏性：** 正确。封装性是指将数据和操作数据的方法捆绑，并对外部隐藏实现细节。友元函数被授予了访问类中所有成员（包括 `private` 和 `protected`

成员）的特权，这在一定程度上打破了封装的界限，因为它允许非成员函数直接访问类的内部状态。因此，应谨慎使用友元。
    - **D. 友元函数不能访问类的私有成员：** 错误。友元函数的主要目的就是为了能够访问类的私有和保护成员。
- **知识点：**
    - **友元 (Friend)：**
        - **友元函数：** 在类中使用 `friend` 关键字声明的非成员函数，它可以访问该类的所有成员（`public`，`protected`，`private`）。
        - **友元类：** 一个类可以声明另一个类为友元，那么友元类的所有成员函数都可以访问声明它的那个类的所有成员。
    - **声明位置：** 友元声明在类定义内部，但友元函数本身不是类的成员。
    - **作用：** 提供一种受控的方式来突破封装，允许特定的外部函数或类访问内部数据，通常用于实现一些紧密相关的类之间的协作，或者重载某些需要访问类私有成员的运算符（如流插入 << 和流提取 >> 运算符通常被重载为友元函数）。
    - **封装性：** 面向对象的重要原则，友元是其一个可控的“例外”。
- **易错点：**
    - 认为友元函数是成员函数。
    - 不理解友元为何以及如何在一定程度上破坏封装性。
    - 忘记友元函数的定义通常在类外。

19．不能作为重载函数的调用的依据是(_____)

A.参数个数
B.参数类型
C.函数类型
D.函数名称

- **正确答案：** 【C】 (题目选项中的“函数类型”通常指函数的返回类型)
- **题目分析：**
  函数重载是指在同一个作用域内，可以定义多个同名函数，但它们的参数列表必须不同。编译器在调用重载函数时，会根据传递的实参来选择最匹配的函数版本。
  判断函数是否构成重载以及调用哪个重载版本的依据是**函数签名 (Function Signature)**，在C++中，函数签名主要包括：
    i. **函数名**
    ii. **参数个数**
    iii. **参数类型** (包括 `const` / `volatile` 修饰符)
    iv. **参数顺序**

函数的返回类型不参与重载决策。 即不能定义两个仅返回类型不同而参数列表完全相同的函数。

- **A. 参数个数：** 是重载的依据。例如 `void func(int a)` 和 `void func(int a, int b)`。
- **B. 参数类型：** 是重载的依据。例如 `void func(int a)` 和 `void func(double a)`。
- **C. 函数类型 (通常指返回类型)：** 不是重载的依据。例如， `int func(int a)` 和 `void func(int a)` 不能构成重载，会导致编译错误。
- **D. 函数名称：** 是重载的前提，重载的函数必须同名。但仅凭函数名称无法区分调用哪个版本，还需要参数列表。

因此，不能作为重载函数调用依据的是函数的返回类型。

- **知识点：**
  - **函数重载 (Function Overloading)：** 允许在同一作用域内声明几个功能类似的同名函数，但是这些同名函数的形式参数（指参数的个数、类型或者顺序）必须不同。
  - **函数签名：** 用于区分不同函数的唯一标识，在 C++中主要由函数名和参数列表构成。返回类型不属于函数签名的一部分（用于重载决策时）。
  - **重载决策：** 编译器根据函数调用时提供的实参类型、数量和顺序来匹配最佳的重载函数。
- **易错点：**
  - 错误地认为函数返回类型可以用来区分重载函数。
  - 对函数签名的构成理解不完整。
  - `typedef` 或类型别名不会创建新的类型，因此 `void func(int)` 和 `typedef int MyInt; void func(MyInt)` 不是重载。

20．面向对象的模型中，最基本的概念是对象和**____**。

- **正确答案：** 【类】
- **题目分析：**
  面向对象编程 (Object-Oriented Programming, OOP) 的核心思想是围绕“对象”进行的。而对象是由“类”创建出来的。
  - **类 (Class)：** 是对一类具有相同属性和行为的事物的抽象描述。它是一个模板或蓝图，定义了对象的结构（数据成员）和行为（成员函数）。
  - **对象 (Object)：** 是类的具体实例。每个对象都拥有类所定义的属性和行为。

  因此，在面向对象的模型中，类和对象是最基本、最核心的概念。其他概念如封装、继承、多态都是建立在这两者之上的。
- **知识点：**
  - **类 (Class)：** 抽象，模板，定义属性和方法。
  - **对象 (Object)：** 具体，实例，拥有属性和执行方法。
  - **面向对象的基本要素：** 除了类和对象，还包括抽象、封装、继承、多态。
- **易错点：**

- 可能会想到其他 OOP 概念如继承或多态，但题目问的是“最基本”的概念，类和对象是构建其他一切的基础。

21．在 VC 中，若定义一个函数的返回类型为 void，以下叙述正确的是(_____)

A.函数返回值需要强类型转换
B.函数不执行任何操作
C.函数本身没有返回值
D.函数不能修改实际参数的值

- **正确答案：** 【C】
- **题目分析：**
  `void` 作为函数的返回类型，意味着该函数在执行完毕后不向调用者返回任何值。
  - **A. 函数返回值需要强类型转换：** 错误。 `void` 函数没有返回值，所以谈不上类型转换。如果试图将 `void` 函数的“结果”赋给变量，会导致编译错误。
  - **B. 函数不执行任何操作：** 错误。 `void` 返回类型只说明函数不返回值，但函数体内部可以执行各种操作，如修改全局变量、修改通过引用或指针传递的参数、进行 I/O 操作等。
  - **C. 函数本身没有返回值：** 正确。这是 `void` 返回类型的确切含义。
  - **D. 函数不能修改实际参数的值：** 错误。函数是否能修改实际参数的值，取决于参数的传递方式：
    - **值传递 (Pass by Value)：** 函数接收的是实参的副本，修改形参不影响实参。
    - **引用传递 (Pass by Reference)：** 函数接收的是实参的引用（别名），修改形参会直接修改实参。
    - **指针传递 (Pass by Pointer)：** 函数接收的是实参地址的副本，通过指针解引用可以修改实参指向的内容。
      函数的返回类型是 `void` 与其是否能修改参数无关。
- **知识点：**
  - **`void` 返回类型：** 表示函数不返回任何值。
  - **函数副作用：** `void` 函数通常通过产生副作用来完成其工作（例如，打印输出、修改传入的引用/指针参数、修改全局状态等）。
  - **参数传递方式：** 值传递、引用传递、指针传递。
- **易错点：**
  - 将“不返回值”与“不执行操作”或“不能修改参数”混淆。

22．函数参数的默认值不允许为(____)

A.全局常量
B.直接常量
C.局部变量
D.函数调用

- **正确答案：** 【C】
- **题目分析：**
  函数参数的默认值必须是在函数声明时就能确定的编译时常量或表达式。
  - **A. 全局常量：** 允许。例如 `const int G_VAL = 10; void func(int x = G_VAL);`
  - **B. 直接常量：** 允许。例如 `void func(int x = 10);`
  - **C. 局部变量：** 不允许。局部变量的生命周期和作用域仅限于其所在的块或函数，在函数声明（尤其是在头文件中被其他文件包含时）或在函数被调用时，该局部变量可能不存在或不可见，其值也不确定。默认参数的值需要在编译时绑定。
  - **D. 函数调用：** 允许，但该函数调用的结果必须在编译时是可确定的（例如，一个 `constexpr` 函数的调用），或者该函数调用本身能产生一个适合作为默认参数的值且在上下文中合法。更常见的是，如果一个函数调用结果是常量表达式，那么它是可以的。例如 `int getDefault() { return 5; } void func(int x = getDefault());` (这里 `getDefault()` 的结果在运行时确定，但如果 `getDefault` 是 `constexpr` 且能在编译期求值，则更佳)。不过，一般简单的函数调用作为默认值是可以的，只要其返回类型匹配。

  严格来说，默认参数是在编译时确定的。局部变量的值在编译时是未知的，并且其作用域有限，因此不能作为默认参数。
- **知识点：**
  - **默认参数 (Default Arguments)：** 允许在函数声明中为一个或多个参数指定默认值。如果调用函数时未提供这些参数的实参，则使用默认值。
  - **设置规则：**
    - 默认参数必须从右向左设置。即如果某个参数有默认值，则其右边的所有参数也必须有默认值。
    - 默认值在函数声明中指定，通常在头文件中。如果在定义和声明中都指定，它们必须相同。
    - 默认值必须是编译时可确定的（常量、全局变量/常量、或能在编译时求值的表达式）。
- **易错点：**
  - 试图使用非静态成员变量作为默认参数（除非在类内定义且上下文允许）。
  - 默认参数的设置顺序错误。
  - 不理解默认参数值是在编译期绑定，而非运行时。

23．下列的描述中(_____)是错误的。

A.使用全局变量可以从被调用函数中获取多个操作结果
B.局部变量可以初始化,若不初始化,则系统默认它的值为 0
C.当函数调用完后,静态局部变量的值不会消失
D.全局变量若不初始化,则系统默认它的值为 0

- **正确答案：** 【B】
- **题目分析：**
  - **A. 使用全局变量可以从被调用函数中获取多个操作结果：** 正确。被调用函数可以修改全局变量的值，调用者在函数返回后可以访问这些修改后的全局变量，从而间接获取多个结果。但这通常不是推荐的做法，因为它破坏了封装性并增加了代码的耦合度。
  - **B. 局部变量可以初始化,若不初始化,则系统默认它的值为 0：** 错误。
    - 局部变量（非静态）如果未显式初始化，其值是未定义的（不确定的、随机的垃圾值）。访问未初始化的局部变量是未定义行为。
    - 只有静态存储期的变量（全局变量、静态全局变量、静态局部变量）在未显式初始化时，会被系统默认初始化为零值（对于数值类型是 0，指针是 `nullptr` ，布尔是 `false` 等）。
  - **C. 当函数调用完后,静态局部变量的值不会消失：** 正确。静态局部变量（用 `static` 修饰的局部变量）在程序运行期间只初始化一次（通常在第一次执行到其声明时）。它的生命周期持续到程序结束，其值在函数调用之间保持不变。
  - **D. 全局变量若不初始化,则系统默认它的值为 0：** 正确。全局变量（以及静态全局变量和静态局部变量）具有静态存储期，如果程序员没有提供初始值，编译器会自动将其初始化为对应类型的零值。
- **知识点：**
  - **变量的存储期 (Storage Duration)：**
    - **自动存储期 (Automatic)：** 局部变量（非 `static` ）。在进入其作用域时创建，离开作用域时销毁。未初始化则值不确定。
    - **静态存储期 (Static)：** 全局变量、命名空间作用域变量、类的静态成员变量、函数的静态局部变量。在程序开始时创建（或第一次使用时），在程序结束时销毁。未初始化则零初始化。
    - **线程存储期 (Thread)：** C++11 引入， `thread_local` 。
    - **动态存储期 (Dynamic)：** 通过 `new` 分配的内存。通过 `delete` 释放。
  - **变量的初始化：**
    - 显式初始化：程序员提供初始值。
    - 默认初始化：对于静态存储期变量，若无显式初始化，则进行零初始化。对于自动存储期变量，若无显式初始化，其值未定义（除非是类类型且有默认构造函数）。
- **易错点：**
  - 混淆局部变量和静态/全局变量的默认初始化规则。这是一个非常常见的错误。

- 认为所有未初始化的变量都会被置为 0。

24．已知 int a[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }, *p = a ;则不能表示数组 a 中元素的式子是(____)

A.*a

B.*p

C.a

D.a[p-a]

- **正确答案：** 【C】
- **题目分析：**

  `int a[10]` 是一个包含 10 个整数的数组。

  `*p = a;` 意味着指针 `p` 指向数组 `a` 的第一个元素 `a[0]`。

  在 C++中，数组名在大多数表达式中会隐式转换为其首元素的地址。

  - **A. `*a`：**
    - `a` (数组名) 在这里会转换为指向 `a[0]` 的指针。
    - `*a` 就是对 `a[0]` 的地址进行解引用，得到 `a[0]` 的值，即 `0`。这是数组 `a` 中的一个元素。**可以表示。**
  - **B. `*p`：**
    - `p` 指向 `a[0]`。
    - `*p` 就是对 `p` 指向的地址（即 `a[0]` 的地址）进行解引用，得到 `a[0]` 的值，即 `0`。这是数组 `a` 中的一个元素。**可以表示。**
  - **C. `a`：**
    - `a` 本身，当它不作为 `sizeof` 的操作数、`&` (取地址) 的操作数，或用于初始化字符数组的字符串字面量时，它代表数组首元素的地址。它是一个地址（指针常量），而不是数组中的一个元素值。**不能表示数组 a 中元素的值，它表示的是地址。**
  - **D. `a[ p-a ]`：**
    - `p` 是指向 `a[0]` 的指针。
    - `a` (数组名) 转换为指向 `a[0]` 的指针。
    - `p - a` 是指针相减。两个指向同一数组中元素的指针相减，结果是它们之间元素的个数。这里 `p` 和 `a` 都指向 `a[0]`，所以 `p - a` 的结果是 `0` (类型通常是 `std::ptrdiff_t`)。
    - 所以 `a[ p-a ]` 等价于 `a[0]`，其值为 `0`。这是数组 `a` 中的一个元素。**可以表示。**

  因此，不能表示数组 `a` 中元素值的是 `C. a`，因为它表示的是数组的首地址。
- **知识点：**
  - **数组名与指针：** 数组名在多数情况下隐式转换成指向其首元素的指针。
  - **指针算术：**

        - 指针加/减整数： `ptr + n` 指向 `ptr` 后面第 `n` 个元素。
        - 指针相减： `ptr1 - ptr2` 得到两个指针间元素的数量差（必须指向同一数组或其末尾之后）。
    - **解引用运算符 `*`：** 获取指针所指向对象的值。
    - **数组下标运算符 `[]`：** `arr[i]` 等价于 `*(arr + i)`。
- **易错点：**
    - 混淆数组名本身（代表地址）和数组元素（代表值）。
    - 对 `a[p-a]` 这种组合形式不熟悉，未能正确分析指针运算。
    - 认为 `a` 直接代表 `a[0]` 的值。

25. 已知 `int a[3][3] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };` 则不能表示数组元素 `a[2][1]` 的地址是

选项：

A. `&a[2][1]`

B. `(a[2]+1)`

C. `a[2]+1`

D. `(a+2)+1`

正确答案： B

解析：

- `a[2][1]` 是数组的第三行第二列元素，其地址可以用 `&a[2][1]` 表示，选项 A 正确。
- `a[2]` 是第三行的首地址， `a[2]+1` 是第三行第二列的地址，选项 C 正确。
- `a+2` 是第三行的首地址， `(a+2)+1` 是第四行的首地址，与 `a[2][1]` 的地址无关，选项 D 错误。
- `(a[2]+1)` 是一个表达式，其结果是 `a[2][1]` 的地址，但语法上不能直接表示地址，选项 B 错误。

26. 设 `char *s1,*s2;` 分别指向两个字符串，判断字符串 `s1` 和 `s2` 是否相等的表达式为

选项：

A. `s1=s2`

B. `s1==s2`

C. `strcpy(s1,s2)==0`

D. `strcmp(s1,s2)==0`

正确答案： D
解析：

- `s1=s2` 是赋值操作，不是比较，选项 A 错误。
- `s1==s2` 是比较指针地址，不是字符串内容，选项 B 错误。
- `strcpy(s1,s2)` 是字符串复制函数，返回值与比较无关，选项 C 错误。
- `strcmp(s1,s2)` 是比较字符串内容，返回 0 表示相等，选项 D 正确。

27. 在类定义的外部，可以被访问的成员有

选项：
A. 所有类成员
B. `private` 或 `protected` 的类成员
C. `public` 的类成员
D. `public` 或 `private` 的类成员
正确答案： C
解析：

- `public` 成员可以在类外部访问，选项 C 正确。
- `private` 和 `protected` 成员只能在类内部或派生类中访问，选项 A、B、D 错误。

28. 若 `class B` 中定义了一个 `class A` 的类成员 `A a`，关于类成员的正确叙述是

选项：
A. 在类 `B` 的成员函数中可以访问 `A` 类的私有数据成员
B. 在类 `B` 的成员函数中可以访问 `A` 类的保护数据成员
C. 类 `B` 的构造函数可以调用类 `A` 的构造函数做数据成员初始化
D. 类 `A` 的构造函数可以调用类 `B` 的构造函数做数据成员初始化
正确答案： C
解析：

- 类 `B` 不能访问类 `A` 的私有成员，选项 A 错误。
- 类 `B` 不能访问类 `A` 的保护成员，除非 `B` 是 `A` 的派生类，选项 B 错误。
- 类 `B` 的构造函数可以调用类 `A` 的构造函数初始化成员 `a`，选项 C 正确。
- 类 `A` 的构造函数不能调用类 `B` 的构造函数，选项 D 错误。

29. 要求用友员函数重载的运算符是

选项：

A. `=`

B. `[]`

C. `<<`

D. `()`

正确答案： C

解析：

- `=`、`[]`、`()` 运算符只能通过成员函数重载，选项 A、B、D 错误。
- `<<` 运算符通常通过友元函数重载，选项 C 正确。

30. 将以下程序写成三目运算表达式是

题目：

```
if(a>b) max=a;
else max=b;
```

正确答案： `max=(a>b)?a:b;`

解析：

- 三目运算符 `?:` 的语法为 `条件 ? 表达式1 : 表达式2`。
- 将 `if-else` 语句转换为三目运算符即可。

31. 下面程序的结果为

题目：

```
#include <iostream>
void main() {
    int a=1, b=2;
    bool c=1;
    if((a>b)||c) cout<<"true";
    else cout<<"false";
}
```

正确答案： true
解析：

- a>b 为 false ， c 为 true ， false || true 结果为 true 。

32. 在创建派生类对象时，构造函数的执行顺序是

选项：
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
正确答案： D
解析：

- 派生类对象的构造函数执行顺序：基类构造函数 → 对象成员构造函数 → 派生类构造函数。

33. 以下程序的执行结果为

题目：

```
#include <iostream>
using namespace std;
class base {
public:
    virtual void who() { cout<<"base class"; }
};
class derive1 : public base {
public:
    void who() { cout<<"derive1 class"; }
};
class derive2 : public base {
public:
    void who() { cout<<"derive2 class"; }
};
void main() {
    base obj1, *p;
    derive1 obj2;
    derive2 obj3;
    p=&obj1; p->who();
    p=&obj2; p->who();
    p=&obj3; p->who();
}
```

正确答案： `base class derive1 class derive2 class`
解析：

- `who()` 是虚函数，通过基类指针调用时会根据实际对象类型执行派生类的重写函数。

34. 虚析构函数的作用是

选项：
A. 虚基类必须定义虚析构函数
B. 类对象作用域结束时释放资源
C. `delete` 动态对象时释放资源
D. 无意义
正确答案： C
解析：

- 虚析构函数确保通过基类指针删除派生类对象时，派生类的析构函数会被调用，选项 C 正确。

35. 若一个类中含有纯虚函数，则该类称为

选项：

A. 基类

B. 纯基类

C. 抽象类

D. 派生类

正确答案： C

解析：

- 含有纯虚函数的类称为抽象类，不能实例化，选项 C 正确。

36. 下列流类中可以用于处理文件的是

选项：

A. `ios`

B. `iostream`

C. `strstream`

D. `fstream`

正确答案： D

解析：

- `fstream` 专门用于文件操作，选项 D 正确。

37. 作为语句标号使用的 C++ 保留字 `case` 和 `default` 只能用于**_____**语句的定义体中。

正确答案： `switch`

解析：

- `case` 和 `default` 是 `switch` 语句的组成部分。

38. 存储字符 'a' 和字符串 "a" 分别需要占用**_____和_____**个字节。

正确答案： 1 或 2
解析：

- 字符 'a' 占用 1 个字节，字符串 "a" 占用 2 个字节（包括结束符 \0）。

39. 若有定义 struct AA {int a; char b; double c;}x; ，则 x 占用空间大小为**_____**字节。

正确答案： 13
解析：

- int 占用 4 字节， char 占用 1 字节， double 占用 8 字节，总大小为 13 字节。

40. 若 y 是 x 的引用，则 &y 和 &x 的值，即为变量**_____**的地址。

正确答案： 相等 或 x
解析：

- 引用是变量的别名， &y 和 &x 的值相同，都是 x 的地址。

41. 执行**_____**操作将释放由 p 所指向的动态分配的数据空间。

正确答案： delete p
解析：

- delete 用于释放动态分配的内存。

42. 已知语句 `cout<<s;` 的输出是 `"apple"`，则执行语句 `cout<<s+2;` 的输出结果为**_____**。

正确答案： `ple`
解析：

- `s+2` 表示从字符串的第三个字符开始输出，即 `"ple"`。

43. 基类和派生类的关系称为**_____**。

正确答案： 继承
解析：

- 派生类继承基类的成员，这种关系称为继承。

44. 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**_____**关键字。

正确答案： `const`
解析：

- `const` 成员函数保证不会修改对象的数据成员。

45. 为了释放类对象中指针成员所指向的动态存储空间，则需要为该类定义**_____**。

正确答案： 析构函数
解析：

- 析构函数用于释放动态分配的资源。

46. 假定 `AB` 为一个类，则执行 `AB a[10];` 语句时，系统自动调用该类构造函数的次数为**____**。

正确答案： `10`
解析：

- 创建数组时会为每个元素调用构造函数。

47. 对类中引用成员的初始化只能通过在构造函数中给出的**____**来实现。

正确答案： 参数初始化表
解析：

- 引用成员必须在构造函数的参数初始化表中初始化。

48. 假定要把 `aa` 定义为 `AB` 类中的一个引用整数型数据成员，则定义语句为**____**。

正确答案： `int& aa;`
解析：

- 引用成员的定义格式为 `类型& 成员名;`。

49. 定用户为类 `AB` 定义了一个构造函数 `AB(int aa=0):a(aa){}` ，则定义该类的对象时，可以有**____**种不同的定义格式。

正确答案： `2`
解析：

- 可以不带参数或带一个参数调用构造函数。

50. 定用户为类 `AB` 定义了一个构造函数 `AB(int aa):a(aa){}` ，则定义该类的对象时，有**____**种定义格式。

正确答案： 1
解析：

- 必须带一个参数调用构造函数。

51. 定用户为类 `AB` 定义了一个构造函数 `AB(int aa=0, int bb=0) {a=aa; b=bb;}` ，则定义该类的对象时，可以有**____**种不同的定义格式。

正确答案： 3
解析：

- 可以不带参数、带一个参数或带两个参数调用构造函数。

52. 假定用户只为类 `AB` 定义了一个构造函数 `AB():a(0),b(0){}` ，则定义该类对象 `x` 的定义语句 `AB x;` 是**__**(正确/错误)的。

正确答案： 正确
解析：

- 构造函数 `AB()` 是无参构造函数，可以直接通过 `AB x;` 定义对象。

53. 假定 `AB` 为一个类，则该类的拷贝构造函数的函数头为**____**。

正确答案： AB(AB&)
解析：

- 拷贝构造函数的参数是同类对象的引用，函数头为 `AB(AB&)` 。

54. 假定 AB 为一个类，该类中含有一个指向动态数组空间的指针成员 pa ，则在该类的析构函数中应该包含有一条**_____**语句。

正确答案： delete []pa;
解析：

- 动态数组需要使用 delete [] 释放内存。

55. 派生类的成员函数可以直接访问基类的公有和保护成员，不能直接访问基类的**_____**成员。

正确答案： 私有
解析：

- 派生类不能直接访问基类的私有成员。

56. 若要保证一个公共的基类在派生类中只产生一个基类子对象，则必须都以**_____**的方式直接继承它。

正确答案： 虚基类
解析：

- 虚基类可以解决多重继承中的二义性问题，确保基类子对象唯一。

57. 引进虚基类的根本目的是为了消除**_____**。

正确答案： 二义性
解析：

- 虚基类的主要作用是消除多重继承中的二义性。

58. 若运行时给变量 x 输入 12 ，则以下程序的运行结果是(_____)

题目：

```
#include <iostream>
void main() {
    int x, y;
    cin >> x;
    y = x > 12 ? x + 10 : x - 12;
    cout << y;
}
```

正确答案： 0
解析：

- x=12 ， x>12 为 false ， y=x-12=0 。

59. 一个类可以从直接或间接的祖先中继承所有属性和方法。采用这个方法提高了软件的**_____**。

正确答案： 可重用性
解析：

- 继承机制提高了代码的可重用性。

60. 静态成员函数可以直接访问类的**_____成员，不能直接访问类的_____**成员。

正确答案： 静态或非静态
解析：

- 静态成员函数只能直接访问静态成员，不能直接访问非静态成员。

61. 函数模板： `template <class T> T add(T x, T y) { return x + y; }` 下列对 `add` 函数的调用不正确的是(____)

选项：

A. `add<>(1, 2)`

B. `add(1, 2)`

C. `add(1.0, 2)`

D. `add(1.0, 2.0)`

正确答案： C

解析：

- `add(1.0, 2)` 中参数类型不一致，模板无法推断类型。

62. 在公有派生情况下，有关派生类对象和基类对象的关系，下列叙述不正确的是(____)

选项：

A. 派生类的对象可以赋给基类的对象

B. 派生类的对象可以初始化基类的引用

C. 派生类的对象可以直接访问基类中的成员

D. 派生类的对象的地址可以赋给指向基类的指针

正确答案： C

解析：

- 派生类对象不能直接访问基类的私有成员。

63. 在 C++ 语言中每个类都有一个**____**指针，该指针指向正在调用成员函数的对象。

正确答案： `this`

解析：

- `this` 指针指向当前对象。

64. 在 C++ 中， `cin` 是一个(____)

选项：

A. 类

B. 对象

C. 模板

D. 函数

正确答案： B

解析：

- `cin` 是 `istream` 类的对象。

65. 下列叙述中，不属于结构化程序设计方法的主要原则的是(____)

选项：

A. 自顶向下

B. 由底向上

C. 模块化

D. 限制使用 `goto` 语句

正确答案： B

解析：

- 结构化程序设计的主要原则是自顶向下、模块化和限制使用 `goto` 语句。

66. 关于构造函数 `A()` 的拷贝构造函数正确的是(____)

选项：

A. `A(A* B);`

B. `A(AB);`

C. `A(A& B);`

D. `A(A);`

正确答案： C

解析：

- 拷贝构造函数的参数是同类对象的引用，函数头为 `A(A&)` 。

67. 派生类的成员一般分为两部分，一部分是**_____**，另一部分是自己定义的新成员。

正确答案： 从基类继承的成员
解析：

- 派生类的成员包括从基类继承的成员和自身定义的成员。

68. (_____) 不是构造函数的特征

选项：
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
正确答案： D
解析：

- 构造函数没有返回值类型，选项 D 错误。

69. 以下对 C++ 语言函数的有关描述中，正确的是(_____)

选项：
A. 在 C++ 语言中调用函数，只能把实参的值传给形参，形参的值不能传送给实参
B. C++ 语言函数既可以嵌套定义又可以递归调用
C. 函数必须无返回值，否则不能使用函数
D. 函数必须有返回值，返回值类型不定
正确答案： A
解析：

- 函数调用时，实参的值传递给形参，形参的值不会影响实参。

70. 可以把具有相同属性的一些不同对象归类，称为**_____**。

正确答案： 对象类
解析：

- 具有相同属性的对象归类为对象类。

71. C++ 语言中的多态性是在编译时通过**_____和模板体现的，在运行时是通过_____**体现的。

正确答案： 函数重载或虚函数
解析：

- 编译时多态通过函数重载和模板实现，运行时多态通过虚函数实现。

72. 常数据成员和静态数据成员在使用前共同的要求是要进行**_____**。

正确答案： 初始化
解析：

- 常数据成员和静态数据成员必须在使用前初始化。

73. 建立派生类对象时，3 种构造函数分别是 a(基类的构造函数)、b(成员对象的构造函数)、c(派生类的构造函数)，这 3 种构造函数的调用顺序为(_____)

选项：
A. abc
B. acb
C. cab
D. cba
正确答案： A
解析：

- 构造函数的调用顺序：基类构造函数 → 成员对象构造函数 → 派生类构造函数。

74. 表达式 `x.operator+(y.operator++(0))` 还可以写成**____**。

正确答案： `x + y++` 或 `x + (y++)`
解析：

- `operator++(0)` 表示后置自增运算符，等价于 `y++` 。

75. 每个 C++ 程序中都必须有且仅有一个

选项：
A. 类
B. 预处理命令
C. 主函数
D. 语句
正确答案： C
解析：

- 每个 C++ 程序必须有且仅有一个 `main()` 函数。

76. C++ 中封装性、继承性和**____**是面向对象思想的主要特征。

正确答案： 多态性
解析：

- 面向对象的主要特征是封装性、继承性和多态性。

77. 以下叙述中正确的是(____)

选项：
A. 构成 C++ 语言程序的基本单位是类
B. 可以在一个函数中定义另一个函数

C. `main()` 函数必须放在其他函数之前
D. 所有被调用的函数一定要在调用之前进行定义
正确答案： A
解析：

- C++ 程序的基本单位是类。

78. 类的析构函数的作用是(____)

选项：
A. 一般成员函数的初始化
B. 类的初始化
C. 对象的初始化
D. 删除类创建的对象
正确答案： D
解析：

- 析构函数用于释放对象占用的资源。

79. 下面关于类和对象的描述中，错误的是(____)

选项：
A. 类就是 C 语言中的结构体类型，对象就是 C 语言中的结构体变量
B. 类和对象之间的关系是抽象和具体的关系
C. 对象是类的实例，一个对象必须属于一个已知的类
D. 类是具有共同行为的若干对象的统一描述体
正确答案： A
解析：

- 类和结构体在 C++ 中有本质区别，类支持封装、继承和多态。

80. 以下程序中，错误的行为是(____)

题目：

```
#include <iostream>
class A {
public:
    int n = 2;
    A(int val) {}
    ~A() {}
};
void main() {
    A a(0);
}
```

正确答案： (5)
解析：

- 类成员不能在声明时直接初始化，选项 (5) 错误。

81. 有如下函数定义： `void func(int a, int& b) { a++; b++; }` 若执行代码段： `int x = 0, y = 1; func(x, y);` 则变量 `x` 和 `y` 值分别是(_____)

选项：
A. 0 和 1
B. 0 和 1
C. 0 和 2
D. 1 和 2
正确答案： C
解析：

- `a` 是值传递， `x` 不变； `b` 是引用传递， `y` 变为 `2` 。

82. 关于友元的概念错误的是(_____)

选项：
A. 友元函数没有 `this` 指针
B. 调用友元函数时必须在它的实参中给出要访问的对象
C. 一个类的成员函数也可以作为另一个类的友元函数

D. 只能在类的公有段声明友元

正确答案： D

解析：

- 友元可以在类的任何段声明，选项 D 错误。

83. 下列函数的运行结果是(_____)

题目：

```
#include <iostream>
int f(int a, int b) {
    int c;
    if (a > b) c = 1;
    else if (a == b) c = 0;
    else c = -1;
    return c;
}
void main() {
    int i = 2, j = 3;
    int p = f(i, j);
    cout << p;
}
```

正确答案： -1

解析：

- i=2 ， j=3 ， i<j ， c=-1 。

84. 下列运算符中全都可以被友元函数重载的是(_____)

选项：

A. = , + , - , \

B. [] , + , () , new

C. -> , + , * , >>

D. << , >> , + , *

正确答案： D
解析：

- `<<`，`>>`，`+`，`*` 都可以通过友元函数重载。

85. 在类的定义中，用于为对象分配内存空间，对类的数据成员进行初始化并执行其他内部管理操作的函数是(_____)

选项：
A. 友元函数
B. 虚函数
C. 构造函数
D. 析构函数
正确答案： C
解析：

- 构造函数用于初始化对象。

86. 若执行下面的程序时，从键盘上输入 `5` 和 `2`，则输出结果是(_____)

```
#include <iostream>
void main() {
    int a, b, k;
    std::cin >> a >> b;
    k = a;
    if (a < b) k = a % b;
    else k = b % a;
    std::cout << k << std::endl;
}
```

- **正确答案：C**
- **分析：**
  - 输入 `a=5`， `b=2`。
  - `a < b` 为假，执行 `k = b % a`，即 `k = 2 % 5`，结果为 `2`。
- **易错点：** 混淆取模运算的顺序。

87. 关于虚函数下面说的正确的是(_____)

- **正确答案：D**
- **分析：**
  - 虚函数不能是 `static` 的，因为 `static` 函数属于类而非对象。
- **易错点：** 混淆虚函数与 `static` 函数的特性。

88. C++中，封装是借助于(_____)达到的

- **正确答案：B**
- **分析：**
  - 封装是通过类来实现的，类将数据和方法封装在一起。
- **易错点：** 混淆封装与其他概念。

89. 下列关于 C++关键字的说法中正确的是(_____)

- **正确答案：D**
- **分析：**
  - 关键字是 C++中预先定义并实现一定功能的一类单词。
- **易错点：** 混淆关键字与用户定义的标识符。

90. C++中类的定义的结束声明的符号是(_____)

- **正确答案：C**
- **分析：**
  - 类定义以分号 `;` 结束。
- **易错点：** 忽略类定义结束符号。

91. 以下 C++语言中用于单行注释的是(_____)

- **正确答案：A**
- **分析：**
  - 单行注释使用 `//`。
- **易错点：** 混淆单行注释与多行注释。

92. 派生类对象对其基类中的(_____)成员是可访问的

- **正确答案：D**
- **分析：**
  - 保护继承的保护成员在派生类中可访问。

- **易错点：** 混淆不同继承方式下的访问权限。

93. 在重载一个运算符时，如果其参数表中有一个参数，则说明该运算符是(____)

- **正确答案：D**
- **分析：**
  - 一元成员运算符或一元友元运算符都可能有一个参数。
- **易错点：** 混淆运算符重载的参数数量。

94. `this` 指针是 C++实现(____)的一种机制

- **正确答案：B**
- **分析：**
  - `this` 指针用于封装，指向当前对象。
- **易错点：** 混淆 `this` 指针的作用。

95. 执行语句 `for(i=1;i++<4;);` 后变量 `i` 的值是(____)

- **正确答案：C**
- **分析：**
  - `i++<4` 在每次循环后 `i` 递增，循环结束时 `i=5`。
- **易错点：** 忽略 `i++` 的后置递增特性。

96. 下列程序的输出结果是(____)

```
#include <iostream>
int min(int a, int b) {
    if (a < b) return a;
    else return b;
}
void main() {
    std::cout << min(1, min(2, 3)) << std::endl;
}
```

- **正确答案：B**
- **分析：**
  - `min(2, 3)` 返回 `2`，`min(1, 2)` 返回 `1`。
- **易错点：** 忽略嵌套函数调用的顺序。

97. 下面对静态数据成员的描述中，正确的是(____)

- **正确答案：D**
- **分析：**
  - 静态数据成员可以直接用类名调用。
- **易错点：** 混淆静态数据成员的访问方式。

98. 对于下面的程序，说法正确的是(____)

```
#include <iostream>
void main() {
    int x = 3, y = 4, z = 2;
    if (x = y + z) std::cout << "x=y+z";
    else std::cout << "x!=y+z";
}
```

- **正确答案：D**
- **分析：**
  - `x = y + z` 赋值后 `x=6`，条件为真，输出 `"x=y+z"`。
- **易错点：** 混淆赋值与比较操作。

99. 不能重载的运算符是(____)

- **正确答案：C**
- **分析：**
  - `.*` 运算符不能重载。
- **易错点：** 忽略不能重载的运算符。

100. 对表达式 `for(表达式1;;表达式3)` 可理解为(____)

- **正确答案：B**
- **分析：**
  - 省略的条件表达式默认为真，等价于 `for(表达式1;1;表达式3)`。
- **易错点：** 忽略 `for` 循环的默认条件。