OOP HW4

Notes

2026 年 5 月 8 日

Question 3

Which among the following is mandatory condition for operators overloading?

- **A. Overloaded operator must be member function of the left operand**
- B. Overloaded operator must be member function of the right operand
- C. Overloaded operator must be member function of either left or right operand
- D. Overloaded operator must not be dependent on the operands

解析：正确选项是 **A**。在 C++ 面向对象题库的标准语境中，本题特指**“重载为成员函数”**的前提条件。为了让编译器识别重载并隐含传递 `this` 指针，运算符必须作为表达式左侧操作数的成员函数（即左操作数充当调用函数的对象）。如果是友元函数重载则无此限制，但在该类题库中 A 为标准的预期答案。

Question 4

When the operator to be overloaded becomes the left operand member then _____

- A. The right operand acts as implicit object represented by *this
- **B. The left operand acts as implicit object represented by *this**
- C. Either right or left operand acts as implicit object represented by *this
- D. *this pointer is not applicable in that member function

解析：正确选项是 **B**。当二元运算符（如 `+`）被重载为类的成员函数时，执行 `A + B` 实际上是被编译器转换为 `A.operator+(B)`。因此，**左操作数**（A）成为了隐含的调用对象，在成员函数内部由 `*this` 指针来代表。

Question 5

If the left operand is pointed by *this pointer, what happens to other operands?

- A. Other operands are passed as function return type
- B. Other operands are passed to compiler implicitly
- C. Other operands must be passed using another member function
- **D. Other operands are passed as function arguments**

解析：正确选项是 **D**。在成员函数形式的二元运算符重载中，因为左操作数已经作为隐含的 `*this` 指针存在，所以**右操作数**（即 other operands）必须显式地作为**函数参数（Function arguments）**传递进该成员函数中。例如声明为：`ReturnType operator+(const ClassName& right_operand);`。

Question 8

Which object' s members can be called directly while overloading operator function is used (In function definition)?

- **A. Left operand members**
- B. Right operand members
- C. All operand members
- D. None of the members

解析：正确选项是 **A**。由于重载的成员函数是由**左操作数**（即 `*this`）主动调用的，所以在重载函数的函数体内部，你可以直接访问左操作数的成员变量（例如直接写 `val`，编译器会自动将其解析为 `this->val`）。而右操作数的成员不能直接写名字，必须通过传递进来的参数对象加点号来访问（例如 `rhs.val`）。

补充笔记：运算符的重载方式选择 (Member vs Friend)

在 C++ 中，运算符重载的选择并非随意，编译器对不同运算符有严格的规定。以下为四大类核心场景：

1. 必须且只能用“成员函数 (Member Function)”重载

这类运算符与对象的内存状态或生命周期高度绑定，左操作数必须是该类的对象。

- **包含：**赋值 `=`、下标 `[]`、函数调用 `()`、成员访问 `->`
- **记忆口诀：**“等号、中括号、小括号、箭头”这四个是类的“私有财产”，绝不允许非成员函数插手。

2. 通常必须用“友元函数 / 非成员函数 (Friend Function)”重载

这类运算符的左操作数通常不是自定义类，而是标准库类（如 `ostream` 或 `istream`）。

- **包含：**流插入 `<<`、流提取 `>>`
- **核心逻辑：**为了保持 `cout << obj;` 这种符合直觉的写法，左操作数必须是 `ostream` 对象，因此只能写成全局非成员函数：`operator<<(ostream& out, const MyClass& obj)`。通常将其声明为 **friend** 以便访问类内的私有成员。

3. 两者皆可（Member 或 Friend 都可以）

绝大多数常规的二元运算符都属于这一类。

- **包含：**算术 (`+`, `-`, `*`), 关系 (`==`, `<`), 复合赋值 (`+=`), 自增自减 (`++`, `--`) 等。
- **最佳实践经验：**
  - **改变对象自身状态的**（如 `+=`, `++`）：强烈建议用 **Member function**。
  - **不改变双方状态且具对称性的**（如 `+`, `==`）：强烈建议用 **Friend function**。因为友元函数允许左右两边的操作数都进行隐式类型转换（例如 `obj + 5` 和 `5 + obj` 均可顺利编译）。

4. 绝对不能被重载的运算符

为了保证 C++ 基础语法的解析不崩溃，以下 5 个运算符**绝对禁止**重载：

- 成员访问 `.`、成员指针访问 `.*`、作用域解析 `::`、三目条件 `?:`、获取大小 `sizeof`。

终极总结速查表

运算符类别	示例	必须 Member?	必须 Friend?	两者皆可?
类的核心行为	=, [], (), ->	✓ 是	× 否	× 否
标准 IO 流操作	<<, >> (配合 cin/cout)	× 否	✓ 是	× 否
修改对象自身状态	+=, -=, ++, --	× 否	× 否	✓ 可以 (推荐 Member)
对称计算与比较	+, -, ==, <	× 否	× 否	✓ 可以 (推荐 Friend)
底层语法解析	., ::, ?:, sizeof	-	-	严禁重载

1 Operator Overloading & Polymorphism Concepts

Concept 1: Assignment Operator

赋值运算符可以重载，但是无论参数为何种类型，赋值运算符都必须重载为成员函数，并且因为返回的是左值，所以返回值的类型必须是该类的 _____。

答案： 引用（或 类名 &）
解析：赋值运算符 = 为了支持连续赋值操作（如 a = b = c;），其返回值必须是可以被修改的左值。因此，它必须返回调用该运算符的对象自身的引用（即 return *this;），其标准函数签名通常为 ClassName& operator=(const ClassName& rhs);。

Concept 2: Parameter Counts & Polymorphism Definition

运算符重载为类的成员函数时,函数参数个数比原来的运算符个数 _____,当重载为类的友元函数时,参数个数与原来运算符个数 _____。多态指不同对象接收 _____ 时产生的 _____。

答案： 少一个、相同、相同的消息、不同行为
解析：

- 参数个数：重载为成员函数时，左操作数充当了隐藏的 this 指针，所以显式参数少一个；重载为友元函数时无隐藏指针，所有操作数都要显式传入。
- 多态本质：面向对象中的多态，核心定义就是“不同的对象接收到相同的消息（即调用同名接口），由于内部具体实现不同，从而表现出不同的响应行为”。

Concept 3: Overloading Methods

运算符重载函数的两种主要方式是 _____、_____。

答案： 成员函数、友元函数（或普通全局函数）

Concept 4: Types of Polymorphism (Execution)

从运行的角度多态可分为 _____ 和 _____。

答案： **静态多态**（编译时多态）、**动态多态**（运行时多态）

解析：静态多态主要依赖函数重载、运算符重载和模板（Template）在编译期决断；动态多态则主要依赖继承和虚函数（`virtual`）在运行期决断。

Concept 5: Function Overloading

函数重载就是一种 _____ ，相同的函数名，对应多个不同的函数体。

答案： **静态多态**（或编译时多态）

解析：编译器在编译阶段，就能根据传入实参的“类型、数量、顺序”组成的函数签名，唯一确定需要绑定哪一个具体的函数版本，无需拖延到运行时。

Concept 6: Overloading Polymorphism

函数重载和 _____ 都属于重载多态。

答案： **运算符重载**

解析：广义的多态可分为四大类：重载多态（函数/运算符重载）、强制多态（类型转换）、包含多态（虚函数）、参数多态（模板）。

Concept 7: Coercion Polymorphism

强制类型转换是通过 _____ 来实现的。

答案： **类型转换函数**（如 `operator int()`）或 **强制多态**

解析：在 C++ 类的语境下，实现隐式或强制类型转换靠的是自定义的类型转换重载函数。

Concept 8: Binding Phases

按照联编所进行的阶段不同,可分为两种不同的联编方法:_____ 和 _____。

答案： **静态联编**（早期联编/编译时联编）、**动态联编**（晚期联编/运行时联编）

解析：联编（Binding）即将函数调用语句与具体的函数体内存代码绑定在一起的过程。

Concept 9: Dynamic Binding Core

动态联编对函数的选择不是基于指针或者引用，而是基于 _____ ，在编译、链接过程中无法解决的绑定问题要等到程序开始运行之后再确定。

答案： **指针或引用所指向对象的实际类型**

解析：这是动态多态的灵魂。例如 `Base* ptr = new Derived();`，`ptr` 的静态类型是基类指针，但它真正在内存中指向的**实际类型**是派生类。动态联编会通过查找虚函数表（vtable），在运行时精准调用派生类重写的虚函数。

Concept 10: Copy Constructor vs. Assignment Operator (拷贝构造与赋值重载的易错区分)

在 C++ 中，“=”符号在不同的上下文中有两种完全不同的含义，这是考试和面试中最常见的陷阱。核心的判断标准只有一个：赋值号左边的对象是否已经存在？

1. 拷贝构造函数 (Copy Constructor): `A(const A &)`

- **核心动作：初始化 (Initialization)**
- **触发条件：**正在**创建**一个全新的对象，并用一个同类型的、已存在的对象来初始化它。此时该对象在内存中刚刚被分配空间。
- **典型代码：** `A d = a;`（在 C++ 编译器眼中，这行代码与 `A d(a);` 完全等价，绝不会调用 `operator=`）。
- **其他隐式调用场景 (极易考)：**
  - 将对象作为实参，**按值传递**给函数参数时（例如调用 `void func(A obj);`）。
  - 函数**按值返回**一个局部对象时（例如 `A func() { return a; }`）。

2. 赋值运算符重载 (Assignment Operator): `A& operator=(const A &)`

- **核心动作：赋值 (Assignment)**
- **触发条件：**针对一个**已经存在**（在之前的代码中早就已经被构造函数创建好）的对象，修改或覆盖它的状态。
- **典型代码：** `c = a;`（前提是 `c` 在这行代码之前就已经通过诸如 `A c;` 声明过了）。

【一句话防坑口诀】：

> 带有类型名声明的“=”（例如 `A d = a;`）是在生孩子，**必定调用拷贝构造**；
> 没有类型名、单独使用的“=”（例如 `c = a;`）是在换衣服，**必定调用赋值重载**。