🎯 九、考试重点提醒

9.1 必考知识点

1. **类大小计算**：只算成员变量，空类1字节
2. **拷贝构造参数必须是引用**：避免无限递归
3. **赋值运算符只能是成员函数**：编译器限制
4. **流操作符用全局函数**：保持自然语法
5. **六大默认成员函数**：构造、析构、拷贝构造、赋值重载、两个取地址

9.2 运算符重载重点

1. **前置vs后置++/--**：参数和返回值的区别
2. **复合赋值vs二元运算符**：实现方式和复用模式
3. **比较运算符链式实现**：用基础运算符实现其他运算符
4. **必须是成员函数的运算符**：= [] () ->
5. **不能重载的运算符**：:: . .* ?: sizeof typeid

9.3 易错点提醒

- **流操作符为什么用全局函数**：保持cout << obj语法
- **友元的作用**：让外部函数访问私有成员
- **this指针的使用**：返回*this支持链式操作
- **const成员函数**：不修改对象状态的函数应该加const
- **自赋值检测**：if(this != &other) 避免自己给自己赋值# C++类知识点复习笔记

📚 一、类的基本概念

1.1 类的实例化

- **关键要点**：类在实例化前不占用内存空间
- **必须先实例化，才能访问类对象的内部成员**
- **实例化过程**：从类模板创建具体对象的过程

1.2 类对象大小计算

🔍 核心规则

- **只计算成员变量的大小，不计算成员函数大小**

- **原因**：成员函数存储在函数表中，类中只保存函数地址
- **空类大小**：如果类没有成员变量，大小为**1个字节**（标识对象存在）
- **内存对齐**：遵循结构体内存对齐规则

```cpp
class Empty {};            // 大小为1字节
class WithData {
    int a;                 // 大小为4字节（只计算成员变量）
    void func() {}         // 函数不计入大小
};
```

📋 二、类的默认成员函数

2.1 六大默认成员函数

空类实际上并不空，编译器会自动生成6个默认成员函数：

1. **构造函数**
2. **析构函数**
3. **拷贝构造函数**
4. **赋值运算符重载**
5. **取地址重载**（普通对象）
6. **const取地址重载**（const对象）

2.2 构造函数类型

🎯 无参构造函数 vs 全缺省构造函数

- **无参构造函数**：`ClassName(){}`
- **全缺省构造函数**：所有参数都有默认值

  ```cpp
  ClassName(Type1 param1 = value1, Type2 param2 = value2)
  ```

- **特点**：全缺省构造函数可以不传参数调用，也可以传递部分或全部参数

🎯 拷贝构造函数

```cpp
NN(const NN& a) {}  // 标准形式
```

🔴 **重要特性：**

- **参数必须是引用类型**（不能是值传递）
- **通常加const修饰**，防止意外修改
- **参数只能有一个**，且必须是同类型对象的引用
- **为什么必须用引用**：如果用值传递，会无限递归调用拷贝构造函数

⚙️ 三、运算符重载

3.1 重载基本规则

- **必须有一个类类型参数**
- **内置类型运算符含义不能改变**
- **作为成员函数重载时，形参比操作数少1个**（this指针隐式传递）

3.2 重载方式分类

🎯 成员函数重载 vs 全局函数重载

```cpp
// 成员函数重载（推荐用于大多数运算符）
class NN {
    NN operator+(int day);           // 成员函数形式
    NN& operator+=(int day);         // 成员函数形式
    bool operator==(const NN& other); // 成员函数形式
};

// 全局函数重载（推荐用于流操作符）
ostream& operator<<(ostream& out, const NN& d);  // 全局函数
istream& operator>>(istream& in, NN& d);          // 全局函数
```

🔴 **选择原则：**

- **成员函数**：适用于修改左操作数的运算符（+=, -=, ++, --, = 等）
- **全局函数**：适用于不修改操作数或需要类型转换的运算符（<<, >>, +, -, == 等）

3.3 赋值运算符重载（重点）

🎯 标准格式

```cpp
类型名& operator=(const 类型名& other) {
    if (this != &other) {  // 检测自赋值
        // 执行赋值操作
    }
    return *this;
}
```

🔴 关键要点：

- **参数类型**：`const 类型名&`（引用提高效率）
- **返回类型**：`类型名&`（引用提高效率，支持连续赋值）
- **返回值**：`*this`
- **只能重载成类的成员函数**，不能重载成全局函数
- **自赋值检测**：`if (this != &other)`

📝 编译器默认行为：

- 对内置类型：逐字节值覆盖
- 对自定义类型：调用该类型的赋值运算符
- 涉及资源管理时，建议自定义实现

3.3 流操作符重载（<< 和 >>）

🎯 输出流重载（<<）

```cpp
friend ostream& operator<<(ostream& out, const NN& d);

// 实现（全局函数）
ostream& operator<<(ostream& out, const NN& d) {
    out << d._year << "/" << d._month << "/" << d._day;
    return out;  // 支持连续输出
}
```

🎯 输入流重载（>>）

```cpp
friend istream& operator>>(istream& in, NN& d);

// 实现（全局函数）
istream& operator>>(istream& in, NN& d) {
    in >> d._year >> d._month >> d._day;
    return in;  // 支持连续输入
}
```

🔴 **为什么用全局函数：**

- 保持 `cout << obj` 的自然语法
- 如果用成员函数，语法变成 `obj << cout`（不符合习惯）
- 使用友元函数访问私有成员

3.4 其他常用运算符重载

前置和后置递增/递减

```cpp
// 前置++：返回引用
NN& operator++() {
    *this += 1;
    return *this;
}

// 后置++：参数int用于区分，返回旧值
NN operator++(int) {
    NN tmp(*this);
    *this += 1;
    return tmp;
}
```

3.7 流操作符重载（<< 和 >>）

3.8 取地址运算符重载

cpp

```cpp
// 声明
NN* operator&();
const NN* operator&() const;

// 实现
NN* NN::operator&() {
    return this;
}

const NN* NN::operator&() const {
    return this;
}
```

🎯 函数调用运算符 operator()

cpp

```cpp
// 声明
return_type operator()(参数列表);

// 示例：仿函数
class Add {
public:
    int operator()(int a, int b) {
        return a + b;
    }
};
// 使用：Add add; int result = add(3, 5);
```

🎯 下标运算符 operator[]

cpp

```cpp
// 声明
Type& operator[](索引类型 index);
const Type& operator[](索引类型 index) const;

// 示例
char& operator[](int index) {
    return data[index];
}
```

🎯 不能重载的运算符

- **::** (作用域解析)
- **.** (成员访问)
- **.*** (成员指针访问)
- **?:** (三目运算符)
- **sizeof** (求大小)
- **typeid** (类型信息)

🎯 必须重载为成员函数的运算符

- **=** (赋值)
- **[]** (下标)
- **()** (函数调用)
- **->** (成员访问)

3.9 特殊运算符重载

🎯 函数调用运算符 operator()

```cpp
// 声明
return_type operator()(参数列表);

// 示例：仿函数
class Add {
public:
    int operator()(int a, int b) {
        return a + b;
    }
};
// 使用：Add add; int result = add(3, 5);
```

🎯 下标运算符 operator[]

cpp

```cpp
// 声明
Type& operator[](索引类型 index);
const Type& operator[](索引类型 index) const;

// 示例
char& operator[](int index) {
    return data[index];
}
```

🎯 不能重载的运算符

- **::** (作用域解析)
- **.** (成员访问)
- **.*** (成员指针访问)
- **?:** (三目运算符)
- **sizeof** (求大小)
- **typeid** (类型信息)

🎯 必须重载为成员函数的运算符

- **=** (赋值)
- **[]** (下标)
- **()** (函数调用)
- **->** (成员访问)

3.10 运算符重载设计模式

🔴 复用模式（推荐）

cpp

```cpp
// 用复合赋值实现二元运算符
NN NN::operator+(int day) {
    NN temp(*this);
    temp += day;     // 复用+=
    return temp;
}

// 用==实现!=
bool NN::operator!=(const NN& other) {
    return !(*this == other);
}

// 用<实现其他比较运算符
bool NN::operator>(const NN& other) {
    return other < *this;
}
```

📋 六、类的其他重要特性

6.1 静态成员

cpp

```cpp
class MyClass {
    static int count;        // 静态成员变量
    static void func();      // 静态成员函数
};
// 类外定义
int MyClass::count = 0;
```

6.2 友元

cpp

```cpp
class MyClass {
    friend class FriendClass;           // 友元类
    friend void friendFunction();       // 友元函数
    friend ostream& operator<<(ostream& out, const MyClass& obj);
};
```

6.3 this指针

- **指向当前对象的指针**
- **在成员函数中隐式传递**
- **静态成员函数没有this指针**
- **用于区分成员变量和参数同名的情况**

6.4 初始化列表

cpp

```cpp
class MyClass {
    const int x;
    int& y;
public:
    // 构造函数初始化列表
    MyClass(int a, int& b) : x(a), y(b) {
        // 构造函数体
    }
};
```

6.5 explicit关键字

cpp

```cpp
class MyClass {
public:
    explicit MyClass(int x);  // 防止隐式转换
};
```

🎯 七、重要设计原则

- **不修改内容的成员函数尽量加const修饰**
- **会修改内容的成员函数不要加const**
- **const修饰的函数构成函数重载**

7.1 const使用原则

- **目的**：让外部函数访问类的私有成员
- **常用场景**：流操作符重载
- **声明**：在类内用 `friend` 关键字声明

7.2 友元函数使用

- **传参优化**：能用引用就用引用（避免拷贝）
- **const修饰**：不修改的参数加const
- **返回值优化**：支持链式操作时返回引用

7.3 参数和返回值设计

上述笔记中的完整代码实现展示了一个日期类（NN类）的完整实现，包含：

- 构造函数和参数验证
- 各种运算符重载
- 日期计算逻辑
- 输入输出流重载

这个例子很好地展示了C++类设计的最佳实践。

📝 八、代码示例总结