C++类和对象复习笔记

1. 1. 基本概念

1.1 类与对象的关系

- **类（Class）**：具有相同属性和行为的对象的抽象描述
  - 包含：数据成员（属性）+ 成员函数（行为）
  - 是一种**用户自定义数据类型**
- **对象（Object）**：类的实例，类的具体体现

1.2 面向对象三大特性

1. **封装性**：将数据和操作数据的方法组合在一起
2. **继承性**：从已有类派生新类，复用代码
3. **多态性**：同一接口的不同实现（函数重载、运算符重载、虚函数）

---

2. 2. 类的定义与结构

2.1 类的组成

```
class ClassName {
private:    // 私有成员：只能类内访问
    // 数据成员
public:     // 公有成员：类内外都可访问
    // 构造函数、析构函数、成员函数
protected:  // 保护成员：类内和派生类可访问
    // 成员变量和函数
};
```

Fence 1

2.2 成员函数定义方式

类内定义：

```
class MyClass {
public:
    void func() { /* 函数体 */ }  // 内联函数
};
```

Fence 2

类外定义：

```
class MyClass {
public:
    void func();  // 类内声明
};
void MyClass::func() { /* 函数体 */ }  // 类外定义
```

Fence 3

2.3 对象的内存分配

- **类内定义函数**：每个对象都有完整的数据区和代码区
- **类外定义函数**：对象只分配数据区，代码区共享
- **重点**：不同对象的数据成员内容不同，但成员函数代码相同

---

3. 3. 构造函数与析构函数 ⭐⭐⭐

3.1 构造函数

3.1.1 基本特点

- **函数名与类名相同**
- **无返回值类型**（连void都不写）
- **对象创建时自动调用**
- **可以重载**

3.1.2 构造函数类型

1. **默认构造函数**

```
ClassName() {}              // 用户定义
ClassName() = default;      // 使用系统默认
```

Fence 4

2. **参数化构造函数**

```
ClassName(int x, int y) : m_x(x), m_y(y) {}  // 初始化列表
```

Fence 5

3. **拷贝构造函数**

```
ClassName(const ClassName& other);
```

Fence 6

3.1.3 重要规则

- 如果定义了任何构造函数，系统不再提供默认构造函数
- 建议单参数构造函数使用`explicit`防止隐式转换

3.2 析构函数

```
~ClassName() {}
```

Fence 7

- **函数名：~类名**
- **无参数，无返回值**
- **不能重载**（一个类只能有一个）
- **对象销毁时自动调用**

3.3 构造/析构执行顺序 ⭐

构造顺序：基类 → 成员对象（声明顺序） → 当前类
析构顺序：当前类 → 成员对象（声明逆序） → 基类

---

4. 4. 拷贝构造函数 ⭐ ⭐

4.1 调用时机（三种情况）

1. **对象声明时初始化**

```
ClassName obj2(obj1);
ClassName obj2 = obj1;  // 注意：这是初始化，不是赋值
```

Fence 8

2. **对象作为函数参数**（值传递）
3. **函数返回对象**（返回值）

4.2 浅拷贝 vs 深拷贝

浅拷贝：系统默认，直接复制数据成员的值

- 问题：指针成员会指向同一内存，析构时重复释放

深拷贝：用户自定义，为指针成员分配新内存

```
ClassName(const ClassName& other) {
    // 为指针成员分配新内存并复制内容
}
```

Fence 9

---

5. 5. 特殊成员类型

5.1 常成员 ⭐

5.1.1 常对象

```
const ClassName obj;
```

Fence 10

- 成员变量不可修改（mutable除外）
- **只能调用常成员函数**

5.1.2 常成员函数

```
int getValue() const { return value; }  // 不能修改成员变量
```

Fence 11

5.1.3 常数据成员

```
class MyClass {
    const int value;
public:
    MyClass(int v) : value(v) {}  // 必须用初始化列表
};
```

Fence 12

5.2 静态成员 ⭐⭐

5.2.1 静态数据成员

```
class MyClass {
    static int count;  // 类内声明
};
int MyClass::count = 0;  // 类外定义和初始化
```

Fence 13

- **所有对象共享**
- **必须在类外初始化**
- **访问方式**：`ClassName::member` 或 `object.member`

5.2.2 静态成员函数

```
static void func() {}
```

Fence 14

- **没有this指针**
- **只能访问静态成员**
- **通过类名直接调用**

---

6. 友元 ⭐

6.1 友元函数

```
class MyClass {
    friend void friendFunc(MyClass& obj);  // 友元函数声明
    friend void OtherClass::memberFunc();  // 其他类成员函数为友元
};
```

Fence 15

6.2 友元类

```
class A {
    friend class B;  // B是A的友元类
};
```

Fence 16

6.3 友元特点

- **破坏封装性**，但提高效率
- **关系单向**：A是B的友元，B不一定是A的友元
- **不能传递**：A是B的友元，B是C的友元，A不是C的友元
- **不受访问控制符影响**

---

7. 运算符重载 ⭐⭐

7.1 重载方式

1. **成员函数方式**

```
ClassName operator+(const ClassName& other) const;
```

Fence 17

2. **友元函数方式**

```
friend ClassName operator+(const ClassName& a, const ClassName& b);
```

Fence 18

7.2 重载限制

- **不能重载的运算符**：`. :: ?: .* sizeof`
- **只能用成员函数重载**：`= () [] ->`
- **不能改变**：优先级、结合性、操作数个数
- **不能创造新运算符**

---

8. 指向成员的指针 ⭐

8.1 指向数据成员的指针

```
class X {
public:
    int a;
    void f(int);
};

int main() {
    int X::* pmi = &X::a;  // 指向成员变量的指针
    X objx;
    objx.*pmi = 10;        // 通过指针访问成员
    cout << objx.a << endl;
}
```

Fence 19

8.2 指向成员函数的指针

```
void (X::* pmf)(int) = &X::f;  // 指向成员函数的指针
(objx.*pmf)(5);                // 通过指针调用成员函数
```

Fence 20

语法格式：

- 声明：`类型名 类名::*指针名`
- 赋值：`&类名::成员名`
- 使用：`对象.*指针` 或 `对象指针->*指针`

---

9. 类之间的关系 ⭐

9.1 组合关系（包含对象成员）

```
class Point {
    int x, y;
public:
    Point(int a, int b) : x(a), y(b) {}
};

class Circle {
    Point center;  // 对象成员
    int radius;
public:
    Circle(int x, int y, int r) : center(x, y), radius(r) {}
};
```

Fence 21

构造/析构顺序：

- 构造：先构造成员对象，再构造当前对象
- 析构：先析构当前对象，再析构成员对象

9.2 嵌套类

```
class Outer {
public:
    class Inner {  // 公有嵌套类
    public:
        void func() {}
    };
private:
    class PrivateInner {  // 私有嵌套类
        void func() {}
    };
};

// 使用方式
Outer::Inner obj;  // 公有嵌套类可以在类外使用
```

特点：

Fence 22

- 公有嵌套类可在类外使用：`外部类::嵌套类 对象名;`
- 私有嵌套类只能在外部类内使用
- 使用不便，不建议多用

---

10. 委托构造函数 ⭐

```
class Box {
    double length, width, height;
public:
    Box(double lv, double wv, double hv) : length{lv}, width{wv}, height{hv}
{}
    Box(double side) : Box(side, side, side) {}  // 委托构造
};
```

Fence 23

特点：

- 一个构造函数调用另一个构造函数
- 在初始化列表中调用
- 避免代码重复

---

11. 简单数据结构（考试可能涉及）

11.1 链表（List）

- 动态数据结构
- 节点包含数据和指向下一节点的指针
- 常见操作：插入、删除、查找

11.2 栈（Stack）

- 后进先出（LIFO）
- 主要操作：push（入栈）、pop（出栈）、top（查看栈顶）

11.3 队列（Queue）

- 先进先出（FIFO）
- 主要操作：enqueue（入队）、dequeue（出队）

---

12. 重要补充知识点

12.1 this指针

- **隐式参数**：指向调用对象的指针
- **静态成员函数没有this指针**
- **使用场景**：区分参数和成员变量、链式调用

12.2 初始化列表 ⭐

```
ClassName(int a, int b) : member1(a), member2(b) {}
```

Fence 24

- **必须使用初始化列表的情况**：
  - const成员
  - 引用成员
  - 没有默认构造函数的对象成员

12.3 explicit关键字

```
explicit ClassName(int x);  // 防止隐式转换
```

Fence 25

---

13. 易错知识点总结 ⚠️

1. **拷贝构造 vs 赋值运算**

```
ClassName obj2 = obj1;  // 拷贝构造
obj2 = obj1;            // 赋值运算
```

Fence 26

2. **常对象只能调用常成员函数**
3. **静态成员函数不能访问非静态成员**
4. **构造函数不能是虚函数，析构函数可以是虚函数**
5. **友元关系不具有传递性和对称性**
6. **成员初始化顺序由声明顺序决定，不是初始化列表顺序**
7. **指向成员的指针使用 .* 和 ->* 运算符**
8. **包含对象成员的类，构造时先构造成员对象**

---

14. 考试重点提醒 🎯

14.1 选择题重点

- 构造/析构函数调用时机和顺序
- 常成员、静态成员的特性
- 友元的特点和限制
- 拷贝构造函数的调用时机

14.2 编程题重点

- 类的完整定义（构造、析构、拷贝构造）
- 运算符重载的实现
- 静态成员的使用
- 包含对象成员的类设计
- 指向成员的指针操作
- 委托构造函数的使用

14.3 调试题重点

- 内存泄漏问题（深浅拷贝）
- 常成员函数调用问题
- this指针的使用
- 初始化列表的正确使用