第四次实验报告

这个任务是一个面向对象编程（C++）的综合模拟题，模拟的是一个虚拟动物园的日常运营。我们需要实现类的继承体系、封装与多态，以及模拟一个动物园每天的活动。由于任务非常复杂，所以先对任务的详细**整理、解释与实现规划**：

一、总体目标

- 使用 C++ 创建一套完整的面向对象系统，模拟动物园每天的运营。
- 使用封装、继承、多态等面向对象设计原则。
- 实现类结构、成员函数、模拟逻辑、状态管理与终止条件。

注意事项

1. 使用三文件架构，包含： `zoo.h` ， `zoo.cpp` ， `main.cpp` 。
2. 题目已给出的数据

动物食物数据

Animal Food	Starting Amount	Price (per unit)
Peanuts	10,000	20 cents
Carrots	7,000	30 cents
Bananas	4,000	50 cents

每种动物的食物容量(指动物在进食多少食物后会造成脏乱)

Each Animal	Food Capacity (units of food)
Elephant	750
Giraffe	500
Monkey	300

3. 评分要求中会扣分的情况：
(a) 内存泄漏。(b) 缺少缩进。(c) 缺少注释。(d) 变量或函数命名不当。(e) 访问修饰符使用不当。
(f) 声明和/或实现放置不当（例如全部放在 Main.cpp 中）。

二、类设计（包含成员与职责）

根据题目要求，结合面向对象设计原则，优化类结构与功能划分如下：

1. Zoo 类 (动物园)

成员变量：

- `AnimalEnclosure* enclosures`： 动态数组（大象、长颈鹿、猴子围栏各一个）
- `ZooKeeper keeper`： 饲养员（唯一实例）
- `FoodSeller seller`： 食物售卖员（唯一实例）
- `vector<Visitor*> visitors`： 访客列表（存储 Adult/Child 对象指针）
- `int openDays`： 动物园已开放的天数
- `Money totalRevenue`： 总收入
- `ExitReason exitReason`： 枚举型关闭原因（FOOD_SOLD_OUT/KEEPER_QUIT）

方法：

- **simulateDay()**：执行单日模拟逻辑（成人与儿童购票、喂食、围栏清洁）
- **runSimulation()**：循环调用 `simulateDay()` 直到终止条件触发
- **checkTerminationConditions()**：每日结束检查是否食物售罄/饲养员辞职

2. Money 类 (货币)

成员变量：

- `float amount`： 按浮点数精确计算，显示时转换为美元美分（如 `$12.34` ）

重载运算符：

- `+`，`-`，`*`，`/`： 支持货币与浮点的算术运算

- `<`, `==`： 支持金额比较

3. AnimalFood 类 (动物食物)

成员变量:

- `FoodType type`： 枚举（`PEANUTS`, `CARROTS`, `BANANAS`）
- `int quantity`： 当前库存量

4. AnimalEnclosure 类 (动物围栏)

成员变量:

- `Animal** animals`： 动态分配的动物指针数组（根据类型）
- `int dirtLevel`： 当前脏污度（初始0）
- `bool isOpen`： 是否对游客开放
- `int daysClosed`： 累计关闭天数
- `AnimalType type`： 围栏内动物类型（`ELEPHANT`, `GIRAFFE`, `MONKEY`）

方法:

- **addDirt(int units)**：按喂食量增加脏污度（`dirtLevel += units`）
- **closeForCleaning()**： 关闭围栏并重置脏污度
- **shouldClean()**： 检查是否需要关闭（返回 `dirtLevel > 2000`）

注意事项

题目明确要求 1 头大象、2 只长颈鹿、3 只猴子，需要在 Zoo 或 AnimalEnclosure 的构造中明确这些数量（比如 new Elephant[1], new Giraffe[2] 等）。

5. Animal 类 (抽象动物基类)

成员变量:

- `string name`： 动物名字（如 `"Dumbo"`）
- `float weight`： 体重（千克）

- `int foodEaten`：当日已摄入食物量

纯虚函数：

- **virtual FoodType getPreferredFood() = 0**
- **virtual int getFoodCapacity() = 0**

核心方法：

- **eat(int amount)**：更新 `foodEaten`，若超过容量则触发 `foodEaten = 0`（多态由子类实现）

6. Elephant / Giraffe / Monkey 类（具体动物）

成员变量（附加）：

- `float trunkLength`（大象象鼻长度）
- `float neckLength`（长颈鹿脖子长度）
- `float armLength`（猴子手臂长度）

纯虚函数实现：

```
// Elephant 类示例
FoodType getPreferredFood() override { return PEANUTS; }
int getFoodCapacity() override { return 750; }
```

7. Person 类（抽象人员基类）

成员变量：

- `string name`：姓名
- `int age`：年龄

继承关系

题面要求：ZooKeeper、FoodSeller、Visitor 都必须派生自抽象基类 Person，以保证统一的姓名/年龄属性和多态行为。

```
class ZooKeeper : public Person { … };
class FoodSeller : public Person { … };
class Visitor    : public Person { … };
```

8. ZooKeeper 类 (饲养员)

成员变量:

- `int cleaningDays`：累计清洁天数（初始0）

方法:

- **cleanEnclosure(AnimalEnclosure& enclosure)**：重置围栏状态， `cleaningDays++`

注意事项

ZooKeeper::cleanEnclosure() 会重置一个围栏的脏污度，但是要注意「一天只能清洁一个围栏」的限制。
题面要求因为只有一个饲养员，每日最多只能关闭并清洁 **一个** 围栏，其他围栏必须保持开放。可以在模拟函数里面实现这个限制。

9. FoodSeller 类 (食物售卖员)

静态成员:

- `static map<FoodType, float> prices`：初始化静态食物价格（`{PEANUTS:0.20, ...}`）

成员变量:

- `AnimalFood peanuts`：花生（初始库存10000）
- `AnimalFood carrots`：胡萝卜（初始库存7000）
- `AnimalFood bananas`：香蕉（初始库存4000）
- `Money revenue`：销售收入

核心方法:

- **sellFood(Adult& adult, FoodType type, int units)**：更新库存与收入

10. Visitor / Adult / Child 类（游客）

Visitor 类 (基类):

- `int passID`：唯一通行证编号
- **virtual void feedAnimal(AnimalEnclosure& enclosure)** = 0（仅 Child 可喂食）

Adult 类:

- `Money budget`：携带金额（初始 $10~20 随机）
- **buyTickets(int childCount)**：计算成人票（1）与儿童票总费用（0.4×childCount)
- **buyFood()**：根据剩余预算均分购买三种食物（`peanuts`，`carrots`，`bananas`），调用 `FoodSeller::sellFood()` 更新库存与收入
- **giveFood()**：将购买的食物分配给儿童（`Child::receiveFood()`）
- **feedAnimal()**：调用 `Visitor::feedAnimal()`，但实际不执行任何操作（Adult 不喂食）

Child 类:

- `int peanutsOwned`：持有的花生数量
- `int carrotsOwned`：持有的胡萝卜数量
- `int bananasOwned`：持有的香蕉数量
- **feedAnimal()**：若围栏开放，按食物类型递减库存并触发 `enclosure.addDirt()`
- **receiveFood()**：接收成人分配的食物（`peanutsOwned`，`carrotsOwned`，`bananasOwned`）

关键设计验证

1. 【继承与多态】
   - `Adult` 与 `Child` 继承自 `Visitor`，覆盖 `feedAnimal` 方法（仅 `Child` 可调用）
   - `Animal` 的 `getPreferredFood()` 和 `getCapacity()` 由子类多态实现
2. 【静态价格表】

```
map<FoodType, float> FoodSeller::prices = {{PEANUTS,0.2}, {CARROTS,0.3}, {BANANAS,0.5}};
```

3. 【饲养员辞职条件】

```
bool Zoo::checkKeeperQuit() {
  return keeper.getCleaningDays() >= 10; // 累计天数达10即退出
}
```

4. 【食物分配规则】
   在 `Adult::buyFood()` 中，剩余预算分成三份购买三种食物（与围栏状态无关）：

```
int buyPeanuts = remainingBudget / 3 / FoodSeller::getPrice(PEANUTS);
// 类似计算胡萝卜和香蕉
```

功能流程图

每日模拟流程：
Generate Visitors → Adults购票 → 买食物（均分三份）→儿童喂食 →
记录脏污度 → 每日结束检查围栏 → 清洁处理 → 更新关闭天数

此设计严格遵循原题要求且消除了先前冲突点，通过清晰的多态划分和封装保证可维护性。

三、模拟逻辑规划（Zoo::simulateDay）

生成访客

- 成人：随机 20–40
- 每个成人带 1–3 个儿童
- 成人预算金钱：10–20 美元（随机）

售票 + 买食物

- 成人买通行证（1 美元/成人，0.4 美元/儿童）
- 剩余的钱按照相同比例买3种食物。**如果围栏关闭，也需要购买。**
- 售卖员更新库存与收入
- 食物均匀分配给儿童

动物喂食

- 儿童对动物喂食（前提是围栏是开启状态）
- 动物吃食物、脏乱度累加
- 超出阈值 → 围栏变脏，食量重置

清洁管理

- 饲养员检查围栏
- 若脏乱度 > 2000 → 关闭围栏进行清洁
- 动物会随机生病，生病的概率由脏乱度决定，越高越容易生病。如果生病就要显示 （动物名）生病了！ 。生病的动物会在一周之后自行好转，好转的信息也要输出。
- 饲养员清洁次数+1；超过 10 次则辞职

终止条件检查

- 或 饲养员辞职
- 或 某种食物完全用完

四、模拟结果统计

- 总共营业了多少天
- 关闭原因（饲养员辞职/ 食物用完）
- 总成人数、儿童数
- 售卖员总收入
- 饲养员清洁天数
- 每个围栏关闭了几天
- 动物的生病情况（包含名字、种类、生病的日期等信息）

五、建议结构拆分

- `zoo.h`：包含所有类声明与接口
- `zoo.cpp`：类函数实现、模拟逻辑
- `main.cpp`：主函数，创建 Zoo 实例并调用 `runSimulation()`

六、反思及对比

在本项目中，若不使用面向对象编程（OOP），将会面临以下问题：

1. **功能堆叠**：所有数据和逻辑只能靠全局变量与函数组合，缺乏封装，导致各模块耦合严重。
2. **扩展困难**：新增一种动物、一个角色或功能时，往往需要重写多个函数逻辑而非继承或覆盖。
3. **维护困难**：围栏、动物、食物等行为没有统一抽象，导致代码冗长重复，极易出错。
4. **缺乏多态**：无法优雅处理如动物喂食等行为的差异，只能通过冗长的 `if-else` 分支实现。

不使用 OOP 的伪代码示例（C++ 风格）

```
// 定义所有动物的数据结构
struct Animal {
    string name;
    float weight;
    int foodEaten;
    string foodType;
};

Animal elephants[10];
Animal giraffes[5];
Animal monkeys[8];

// 动物喂食函数（非多态）
void feedAnimal(Animal &a, int foodAmount) {
    a.foodEaten += foodAmount;
    if (a.foodEaten > 100) {
        a.weight += 1.5;
    }
}

// 围栏状态用数组和整数表示
int elephantEnclosureDirt = 0;
int giraffeEnclosureDirt = 0;
int monkeyEnclosureDirt = 0;

// 清洁逻辑（重复代码）
void cleanEnclosure(string type) {
    if (type == "elephant") elephantEnclosureDirt = 0;
    else if (type == "giraffe") giraffeEnclosureDirt = 0;
    else if (type == "monkey") monkeyEnclosureDirt = 0;
}

// 售票逻辑（全局处理）
float totalRevenue = 0.0;
void sellTickets(int adults, int children) {
    totalRevenue += adults * 1.0 + children * 0.4;
}

// 动物园模拟主逻辑
void simulateDay() {
    int numAdults = rand() % 20 + 20;
```

```
    int numChildren = 0;

    for (int i = 0; i < numAdults; ++i) {
        int c = rand() % 4;
        numChildren += c;
        sellTickets(1, c);
    }

    // 喂食全部动物
    for (int i = 0; i < 10; ++i)
        feedAnimal(elephants[i], 5);
    for (int i = 0; i < 5; ++i)
        feedAnimal(giraffes[i], 3);
    for (int i = 0; i < 8; ++i)
        feedAnimal(monkeys[i], 2);

    elephantEnclosureDirt += 100;
    giraffeEnclosureDirt += 80;
    monkeyEnclosureDirt += 60;

    if (elephantEnclosureDirt > 2000)
        cleanEnclosure("elephant");
    // 其他类似逻辑重复处理...
}
```

与面向对象实现对比

项目维度	非 OOP 实现	OOP 实现
数据封装	全局变量散乱、结构体简单	类内封装、成员职责清晰
功能扩展	需手动添加新逻辑到多个函数	通过继承/多态重载简化扩展
行为管理	`if-else` 或者 `switch-case` 判断行为分支	虚函数和多态机制
代码维护	修改一处易影响其他地方	修改封装类时对外部影响较小
可读性与组织性	低，函数混杂，代码重复多	高，结构分层明确，易于模块化

小结

通过上述伪代码对比，我们可以明显看到：

- 面向对象设计让复杂系统更清晰、模块化；
- 非 OOP 实现虽然短期上手容易，但长期维护和扩展代价极高；
- 使用类的封装、继承和多态可以有效地组织和管理逻辑，是大型系统的必要设计。

因此，**本实验强化了我们对面向对象编程优势的理解和实践能力**，尤其在复杂模拟系统中，OOP 能大幅降低代码复杂度与维护成本。

七、总结

这是一个面向对象模拟系统设计题，主要考察 OOP 的三大特性、类结构设计能力、系统模拟能力以及C++语言特性（如重载、继承、多态）。代码分层清晰，逻辑严密是关键。

八、附录

实验结果（输出）展示

```
===== Day 13 =====
Child1-1 fed 8 units of food to animals in the Elephant enclosure.
Child1-2 fed 7 units of food to animals in the Elephant enclosure.
Child1-3 fed 4 units of food to animals in the Giraffe enclosure.
Child2-1 fed 3 units of food to animals in the Monkey enclosure.
Child2-2 fed 3 units of food to animals in the Monkey enclosure.
Child2-3 fed 7 units of food to animals in the Elephant enclosure.
Child3-1 fed 5 units of food to animals in the Elephant enclosure.
Child3-2 fed 3 units of food to animals in the Giraffe enclosure.
Child3-3 fed 4 units of food to animals in the Elephant enclosure.
Child4-1 fed 10 units of food to animals in the Monkey enclosure.
Child5-1 fed 9 units of food to animals in the Giraffe enclosure.
Child5-2 fed 9 units of food to animals in the Giraffe enclosure.
Child6-1 fed 8 units of food to animals in the Elephant enclosure.
Child6-2 fed 5 units of food to animals in the Giraffe enclosure.
Child7-1 fed 8 units of food to animals in the Elephant enclosure.
Child7-2 fed 8 units of food to animals in the Elephant enclosure.
Child8-1 fed 10 units of food to animals in the Elephant enclosure.
Child9-1 fed 10 units of food to animals in the Giraffe enclosure.
Child10-1 fed 10 units of food to animals in the Elephant enclosure.
Child11-1 fed 8 units of food to animals in the Monkey enclosure.
Child12-1 fed 7 units of food to animals in the Giraffe enclosure.
Child12-2 fed 4 units of food to animals in the Monkey enclosure.
Child13-1 fed 3 units of food to animals in the Monkey enclosure.
Child13-2 fed 8 units of food to animals in the Elephant enclosure.
Child13-3 fed 7 units of food to animals in the Elephant enclosure.
Child14-1 fed 8 units of food to animals in the Elephant enclosure.
Child14-2 fed 5 units of food to animals in the Giraffe enclosure.
Child14-3 fed 4 units of food to animals in the Giraffe enclosure.
Child15-1 fed 10 units of food to animals in the Giraffe enclosure.
Child16-1 fed 4 units of food to animals in the Monkey enclosure.
Child16-2 fed 6 units of food to animals in the Giraffe enclosure.
Child16-3 fed 5 units of food to animals in the Giraffe enclosure.
Child17-1 fed 6 units of food to animals in the Monkey enclosure.
Child17-2 fed 10 units of food to animals in the Giraffe enclosure.
Child18-1 fed 10 units of food to animals in the Elephant enclosure.
Child18-2 fed 7 units of food to animals in the Giraffe enclosure.
Child19-1 fed 6 units of food to animals in the Giraffe enclosure.
```

Child19-2 fed 8 units of food to animals in the Elephant enclosure.
Child19-3 fed 8 units of food to animals in the Elephant enclosure.
Child20-1 fed 7 units of food to animals in the Giraffe enclosure.
Child20-2 fed 10 units of food to animals in the Elephant enclosure.
Child21-1 fed 10 units of food to animals in the Monkey enclosure.
Visitors today: 63 (21 adults, 42 children)
Food remaining: Peanuts=583, Carrots=797, Bananas=363
Zoo revenue today: $774.20

===== Day 14 =====
Child1-1 fed 6 units of food to animals in the Giraffe enclosure.
Child1-2 fed 9 units of food to animals in the Elephant enclosure.
Child1-3 fed 6 units of food to animals in the Giraffe enclosure.
Child2-1 fed 8 units of food to animals in the Giraffe enclosure.
Child2-2 fed 5 units of food to animals in the Monkey enclosure.
Child3-1 fed 3 units of food to animals in the Monkey enclosure.
Child3-2 fed 3 units of food to animals in the Monkey enclosure.
Child4-1 fed 4 units of food to animals in the Giraffe enclosure.
Child4-2 fed 6 units of food to animals in the Elephant enclosure.
Child4-3 fed 4 units of food to animals in the Giraffe enclosure.
Child5-1 fed 10 units of food to animals in the Elephant enclosure.
Child6-1 fed 8 units of food to animals in the Elephant enclosure.
Child6-2 fed 5 units of food to animals in the Giraffe enclosure.
Child6-3 fed 4 units of food to animals in the Giraffe enclosure.
Child7-1 fed 10 units of food to animals in the Giraffe enclosure.
Child8-1 fed 10 units of food to animals in the Giraffe enclosure.
Child9-1 fed 3 units of food to animals in the Giraffe enclosure.
Child9-2 fed 3 units of food to animals in the Giraffe enclosure.
Child9-3 fed 3 units of food to animals in the Giraffe enclosure.
Child10-1 fed 3 units of food to animals in the Giraffe enclosure.
Child10-2 fed 3 units of food to animals in the Giraffe enclosure.
Child10-3 fed 4 units of food to animals in the Elephant enclosure.
Child11-1 fed 10 units of food to animals in the Giraffe enclosure.
Child12-1 fed 10 units of food to animals in the Giraffe enclosure.
Child12-2 fed 10 units of food to animals in the Elephant enclosure.
Child13-1 fed 7 units of food to animals in the Giraffe enclosure.
Child13-2 fed 10 units of food to animals in the Elephant enclosure.
Child14-1 fed 10 units of food to animals in the Giraffe enclosure.
Child15-1 fed 10 units of food to animals in the Giraffe enclosure.
Child16-1 fed 3 units of food to animals in the Giraffe enclosure.
Child16-2 fed 5 units of food to animals in the Elephant enclosure.
Child16-3 fed 4 units of food to animals in the Elephant enclosure.
Child17-1 fed 7 units of food to animals in the Monkey enclosure.

Child18-1 fed 6 units of food to animals in the Monkey enclosure.
Child19-1 fed 4 units of food to animals in the Giraffe enclosure.
Child19-2 fed 2 units of food to animals in the Monkey enclosure.
Child19-3 fed 2 units of food to animals in the Monkey enclosure.
Child20-1 fed 10 units of food to animals in the Elephant enclosure.
Child21-1 fed 6 units of food to animals in the Giraffe enclosure.
Child21-2 fed 5 units of food to animals in the Giraffe enclosure.
Child22-1 fed 5 units of food to animals in the Monkey enclosure.
Child22-2 fed 5 units of food to animals in the Monkey enclosure.
Child23-1 fed 10 units of food to animals in the Giraffe enclosure.
Child24-1 fed 10 units of food to animals in the Giraffe enclosure.
Child25-1 fed 7 units of food to animals in the Giraffe enclosure.
Child25-2 fed 10 units of food to animals in the Elephant enclosure.
Child26-1 fed 10 units of food to animals in the Elephant enclosure.
Child26-2 fed 10 units of food to animals in the Elephant enclosure.
Child27-1 fed 4 units of food to animals in the Giraffe enclosure.
Child27-2 fed 5 units of food to animals in the Elephant enclosure.
Child27-3 fed 2 units of food to animals in the Monkey enclosure.
Child28-1 fed 7 units of food to animals in the Elephant enclosure.
Child28-2 fed 6 units of food to animals in the Giraffe enclosure.
Child28-3 fed 3 units of food to animals in the Monkey enclosure.
Child29-1 fed 10 units of food to animals in the Giraffe enclosure.
Child30-1 fed 4 units of food to animals in the Monkey enclosure.
Child30-2 fed 6 units of food to animals in the Giraffe enclosure.
Child31-1 fed 10 units of food to animals in the Giraffe enclosure.
Child32-1 fed 10 units of food to animals in the Giraffe enclosure.
Child33-2 fed 3 units of food to animals in the Monkey enclosure.
Child34-2 fed 7 units of food to animals in the Giraffe enclosure.
Child35-1 fed 7 units of food to animals in the Monkey enclosure.
Child37-1 fed 5 units of food to animals in the Monkey enclosure.
Child37-2 fed 4 units of food to animals in the Monkey enclosure.
Zookeeper Bob cleaned the Giraffe enclosure.
Visitors today: 106 (37 adults, 69 children)
Food remaining: Peanuts=0, Carrots=272, Bananas=57
Zoo revenue today: $838.80

SIMULATION ENDED: Peanuts has been sold out.

====== Zoo Simulation Summary ======
Days operated: 14
Exit reason: Food sold out (Peanuts)
Total revenue: $838.80
Food seller revenue: $5989.90

```
Enclosure status:
  Elephant enclosure was closed for 1 days.
  Giraffe enclosure was closed for 1 days.
  Monkey enclosure was closed for 0 days.
=== Zoo Simulation Ended ===
请按任意键继续. . .
```

code

zoo.h

```cpp
#ifndef ZOO_H
#define ZOO_H

#include <map>
#include <string>
#include <vector>


// 枚举定义
enum FoodType {
    PEANUTS,
    CARROTS,
    BANANAS
};

enum AnimalType {
    ELEPHANT_ENCLOSURE,
    GIRAFFE_ENCLOSURE,
    MONKEY_ENCLOSURE
};

enum ExitReason {
    NONE,
    FOOD_SOLD_OUT,
    KEEPER_QUIT
};

// 前向声明
class Animal;
class AnimalEnclosure;
class Visitor;
class Adult;
class Child;

// ====================== Money 类 ======================
class Money {
private:
    float amount;

public:
    Money(float val);
    Money();
```

```
    Money operator+(const Money& other) const;
    Money operator-(const Money& other) const;
    Money operator*(float multiplier) const;
    Money operator/(float divisor) const;
    bool operator<(const Money& other) const;
    bool operator==(const Money& other) const;

    std::string toString() const;
    float getAmount() const;
};

// ====================== AnimalFood 类 ======================
class AnimalFood {
private:
    FoodType type;
    int quantity;

public:
    AnimalFood(FoodType t, int qty);
    AnimalFood();

    FoodType getType() const;
    int getQuantity() const;
    void reduceQuantity(int amount);
    bool isEmpty() const;
};

// ====================== Animal 类 (抽象基类) ======================
class Animal {
private:
    std::string name;
    float weight;
    int foodEaten;

public:
    Animal(std::string n, float w);
    virtual ~Animal() {}

    std::string getName() const;
    float getWeight() const;
    void eat(int amount);
    int getFoodEaten() const;
```

```
    // 纯虚函数
    virtual FoodType getPreferredFood() = 0;
    virtual int getFoodCapacity() = 0;
};

// ==================== Elephant 类 ====================
class Elephant : public Animal {
private:
    float trunkLength;

public:
    Elephant(std::string n, float w, float tLength);

    FoodType getPreferredFood() override;
    int getFoodCapacity() override;
};

// ==================== Giraffe 类 ====================
class Giraffe : public Animal {
private:
    float neckLength;

public:
    Giraffe(std::string n, float w, float nLength);

    FoodType getPreferredFood() override;
    int getFoodCapacity() override;
};

// ==================== Monkey 类 ====================
class Monkey : public Animal {
private:
    float armLength;

public:
    Monkey(std::string n, float w, float aLength);

    FoodType getPreferredFood() override;
    int getFoodCapacity() override;
};

// ==================== Person 类 (抽象基类) ====================
class Person {
```

```
private:
    std::string name;
    int age;

public:
    Person(std::string n, int a);
    virtual ~Person() {}

    std::string getName() const;
    int getAge() const;
};

// ==================== ZooKeeper 类 ====================
class ZooKeeper : public Person {
private:
    int cleaningDays;

public:
    ZooKeeper(std::string n, int a);

    void cleanEnclosure(AnimalEnclosure& enclosure);
    int getCleaningDays() const;
};

// ==================== FoodSeller 类 ====================
class FoodSeller : public Person {
private:
    AnimalFood peanuts;
    AnimalFood carrots;
    AnimalFood bananas;
    Money revenue;

    static std::map<FoodType, float> prices;

public:
    FoodSeller(std::string n, int a);

    Money sellFood(Adult& adult, FoodType type, int units);
    bool isFoodSoldOut() const;
    std::string getSoldOutFoodName() const;
    Money getRevenue() const;
    int getFoodAmount(FoodType type) const;
```

```
    static float getPrice(FoodType type);
};

// ==================== Visitor 类 (抽象基类) ====================
class Visitor : public Person {
private:
    int passID;
    static int nextID;

public:
    Visitor(std::string n, int a);
    virtual ~Visitor() {}

    int getPassID() const;
    virtual void feedAnimal(AnimalEnclosure& enclosure) = 0;
};

// ==================== Adult 类 ====================
class Adult : public Visitor {
private:
    Money budget;

public:
    Adult(std::string n, int a);

    Money getBudget() const;
    Money buyTickets(int childCount);
    void buyFood(FoodSeller& seller, std::vector<Child*>& children);
    void giveFood(std::vector<Child*>& children, int peanuts, int carrots, int bananas);
    void feedAnimal(AnimalEnclosure& enclosure) override;
};

// ==================== Child 类 ====================
class Child : public Visitor {
private:
    int peanutsOwned;
    int carrotsOwned;
    int bananasOwned;

public:
    Child(std::string n, int a);

    void receiveFood(int peanuts, int carrots, int bananas);
```

```
    void feedAnimal(AnimalEnclosure& enclosure) override;
};

// ====================== AnimalEnclosure 类 ======================
class AnimalEnclosure {
private:
    Animal** animals;
    int animalCount;
    AnimalType type;
    int dirtLevel;
    int daysClosed;

public:
    bool isOpen;  // 公开变量，以便Child类可以直接访问

    AnimalEnclosure(AnimalType t);
    ~AnimalEnclosure();

    void addDirt(int units);
    void closeForCleaning();
    void reopen();
    bool shouldClean() const;
    std::string getEnclosureTypeName() const;
    AnimalType getEnclosureType() const;
    int getDirtLevel() const;
    int getDaysClosed() const;
};

// ====================== Zoo 类 ======================
class Zoo {
private:
    AnimalEnclosure** enclosures;  // 动物围栏数组
    ZooKeeper* keeper;  // 饲养员
    FoodSeller* seller;  // 食物售卖员
    std::vector<Visitor*> visitors;  // 访客列表
    int openDays;  // 开放天数
    Money totalRevenue;  // 总收入
    ExitReason exitReason;  // 退出原因

public:
    Zoo();
    ~Zoo();
```

```
    void simulateDay();
    void runSimulation();
    void checkTerminationConditions();
    void printSummary();
};

#endif  // ZOO_H
```

zoo.cpp

```cpp
#include "zoo.h"

#include <algorithm>
#include <iomanip>
#include <iostream>
#include <random>
#include <sstream>


using namespace std;

// ==================== Money 类实现 ====================
Money::Money(float val) : amount(val) {}

Money::Money() : amount(0.0f) {}

Money Money::operator+(const Money& other) const {
    return Money(amount + other.amount);
}

Money Money::operator-(const Money& other) const {
    return Money(amount - other.amount);
}

Money Money::operator*(float multiplier) const {
    return Money(amount * multiplier);
}

Money Money::operator/(float divisor) const {
    if (divisor == 0) {
        throw runtime_error("Division by zero");
    }
    return Money(amount / divisor);
}

bool Money::operator<(const Money& other) const {
    return amount < other.amount;
}

bool Money::operator==(const Money& other) const {
    return abs(amount - other.amount) < 0.001f;  // 浮点数比较
}
```

```
string Money::toString() const {
    ostringstream oss;
    oss << "$" << fixed << setprecision(2) << amount;
    return oss.str();
}

float Money::getAmount() const {
    return amount;
}

// ==================== AnimalFood 类实现 ====================
AnimalFood::AnimalFood(FoodType t, int qty) : type(t), quantity(qty) {}

AnimalFood::AnimalFood() : type(PEANUTS), quantity(0) {}

FoodType AnimalFood::getType() const {
    return type;
}

int AnimalFood::getQuantity() const {
    return quantity;
}

void AnimalFood::reduceQuantity(int amount) {
    if (amount <= quantity) {
        quantity -= amount;
    } else {
        quantity = 0;
    }
}

bool AnimalFood::isEmpty() const {
    return quantity <= 0;
}

// ==================== Animal 类实现 ====================
Animal::Animal(string n, float w) : name(n), weight(w), foodEaten(0) {}

string Animal::getName() const {
    return name;
}

float Animal::getWeight() const {
```

```
    return weight;
}

void Animal::eat(int amount) {
    foodEaten += amount;
    // 若食物超过容量，则重置食物量
    if (foodEaten >= getFoodCapacity()) {
        foodEaten = 0;
    }
}

int Animal::getFoodEaten() const {
    return foodEaten;
}

// ====================== 具体动物类实现 ======================
// Elephant 实现
Elephant::Elephant(string n, float w, float tLength)
    : Animal(n, w), trunkLength(tLength) {}

FoodType Elephant::getPreferredFood() {
    return PEANUTS;
}

int Elephant::getFoodCapacity() {
    return 750;
}

// Giraffe 实现
Giraffe::Giraffe(string n, float w, float nLength)
    : Animal(n, w), neckLength(nLength) {}

FoodType Giraffe::getPreferredFood() {
    return CARROTS;
}

int Giraffe::getFoodCapacity() {
    return 500;
}

// Monkey 实现
Monkey::Monkey(string n, float w, float aLength)
    : Animal(n, w), armLength(aLength) {}
```

```
FoodType Monkey::getPreferredFood() {
    return BANANAS;
}

int Monkey::getFoodCapacity() {
    return 300;
}

// ====================== Person 类实现 ======================
Person::Person(string n, int a) : name(n), age(a) {}

string Person::getName() const {
    return name;
}

int Person::getAge() const {
    return age;
}

// ====================== ZooKeeper 类实现 ======================
ZooKeeper::ZooKeeper(string n, int a) : Person(n, a), cleaningDays(0) {}

void ZooKeeper::cleanEnclosure(AnimalEnclosure& enclosure) {
    enclosure.closeForCleaning();
    cleaningDays++;
    cout << "Zookeeper " << getName() << " cleaned the "
         << enclosure.getEnclosureTypeName() << " enclosure." << endl;
}

int ZooKeeper::getCleaningDays() const {
    return cleaningDays;
}

// ====================== FoodSeller 类实现 ======================
// 静态成员初始化
map<FoodType, float> FoodSeller::prices = {
    {PEANUTS, 0.2f},
    {CARROTS, 0.3f},
    {BANANAS, 0.5f}};

FoodSeller::FoodSeller(string n, int a) : Person(n, a),
                                          peanuts(PEANUTS, 10000),
```

```
                                        carrots(CARROTS, 7000),
                                        bananas(BANANAS, 4000),
                                        revenue(0.0f) {}

Money FoodSeller::sellFood(Adult& adult, FoodType type, int units) {
    AnimalFood* targetFood = nullptr;

    switch (type) {
    case PEANUTS:
        targetFood = &peanuts;
        break;
    case CARROTS:
        targetFood = &carrots;
        break;
    case BANANAS:
        targetFood = &bananas;
        break;
    }

    if (!targetFood) {
        return Money(0.0f);
    }

    // 确保不能卖出超过库存的食物
    int actualUnits = min(units, targetFood->getQuantity());
    if (actualUnits <= 0) {
        return Money(0.0f);
    }

    // 计算价格
    Money cost(actualUnits * prices[type]);

    // 更新库存和收入
    targetFood->reduceQuantity(actualUnits);
    revenue = revenue + cost;

    return cost;
}

bool FoodSeller::isFoodSoldOut() const {
    return peanuts.isEmpty() || carrots.isEmpty() || bananas.isEmpty();
}
```

```
string FoodSeller::getSoldOutFoodName() const {
    if (peanuts.isEmpty()) return "Peanuts";
    if (carrots.isEmpty()) return "Carrots";
    if (bananas.isEmpty()) return "Bananas";
    return "None";
}

Money FoodSeller::getRevenue() const {
    return revenue;
}

float FoodSeller::getPrice(FoodType type) {
    return prices[type];
}

int FoodSeller::getFoodAmount(FoodType type) const {
    switch (type) {
    case PEANUTS:
        return peanuts.getQuantity();
    case CARROTS:
        return carrots.getQuantity();
    case BANANAS:
        return bananas.getQuantity();
    default:
        return 0;
    }
}

// ====================== Visitor 类实现 ======================
int Visitor::nextID = 1;

Visitor::Visitor(string n, int a) : Person(n, a), passID(nextID++) {}

int Visitor::getPassID() const {
    return passID;
}

// ====================== Adult 类实现 ======================
Adult::Adult(string n, int a) : Visitor(n, a), budget(10.0f + static_cast<float>(rand() % 1001)

Money Adult::getBudget() const {
    return budget;
}
```

```cpp
Money Adult::buyTickets(int childCount) {
    Money cost(1.0f + 0.4f * childCount);

    if (budget.getAmount() >= cost.getAmount()) {
        budget = budget - cost;
        return cost;
    }

    return Money(0.0f);
}

void Adult::buyFood(FoodSeller& seller, vector<Child*>& children) {
    if (children.empty() || budget.getAmount() <= 0.0f) {
        return;
    }

    // 将预算均分为三份
    float budgetPerType = budget.getAmount() / 3.0f;

    // 购买三种食物
    int peanutUnits = static_cast<int>(budgetPerType / seller.getPrice(PEANUTS));
    int carrotUnits = static_cast<int>(budgetPerType / seller.getPrice(CARROTS));
    int bananaUnits = static_cast<int>(budgetPerType / seller.getPrice(BANANAS));

    // 扣钱并购买
    Money peanutCost = seller.sellFood(*this, PEANUTS, peanutUnits);
    Money carrotCost = seller.sellFood(*this, CARROTS, carrotUnits);
    Money bananaCost = seller.sellFood(*this, BANANAS, bananaUnits);

    // 更新预算
    budget = budget - peanutCost - carrotCost - bananaCost;

    // 获取实际购买的单位数
    peanutUnits = static_cast<int>(peanutCost.getAmount() / seller.getPrice(PEANUTS));
    carrotUnits = static_cast<int>(carrotCost.getAmount() / seller.getPrice(CARROTS));
    bananaUnits = static_cast<int>(bananaCost.getAmount() / seller.getPrice(BANANAS));

    // 将食物分配给孩子
    giveFood(children, peanutUnits, carrotUnits, bananaUnits);
}

void Adult::giveFood(vector<Child*>& children, int peanuts, int carrots, int bananas) {
```

```
    int childCount = children.size();
    if (childCount == 0) return;

    // 计算每个孩子分得的食物数量
    int peanutsPerChild = peanuts / childCount;
    int carrotsPerChild = carrots / childCount;
    int bananasPerChild = bananas / childCount;

    // 余数
    int peanutRemainder = peanuts % childCount;
    int carrotRemainder = carrots % childCount;
    int bananaRemainder = bananas % childCount;

    // 分配食物给每个孩子
    for (int i = 0; i < childCount; i++) {
        int extraPeanuts = (i < peanutRemainder) ? 1 : 0;
        int extraCarrots = (i < carrotRemainder) ? 1 : 0;
        int extraBananas = (i < bananaRemainder) ? 1 : 0;

        children[i]->receiveFood(
            peanutsPerChild + extraPeanuts,
            carrotsPerChild + extraCarrots,
            bananasPerChild + extraBananas);
    }
}

void Adult::feedAnimal(AnimalEnclosure& enclosure) {
    // 成人不喂食动物
}

// ==================== Child 类实现 ====================
Child::Child(string n, int a) : Visitor(n, a),
                                peanutsOwned(0),
                                carrotsOwned(0),
                                bananasOwned(0) {}

void Child::receiveFood(int peanuts, int carrots, int bananas) {
    peanutsOwned += peanuts;
    carrotsOwned += carrots;
    bananasOwned += bananas;
}

void Child::feedAnimal(AnimalEnclosure& enclosure) {
```

```
if (!enclosure.isOpen) {
    return;  // 围栏关闭则不喂食
}

// 获取围栏内动物喜欢的食物类型
FoodType preferredFood = PEANUTS;  // 默认值

// 根据围栏类型确定动物喜欢的食物
switch (enclosure.getEnclosureType()) {
case ELEPHANT_ENCLOSURE:
    preferredFood = PEANUTS;
    break;
case GIRAFFE_ENCLOSURE:
    preferredFood = CARROTS;
    break;
case MONKEY_ENCLOSURE:
    preferredFood = BANANAS;
    break;
}

// 根据食物类型决定喂食数量
int foodAmount = 0;
switch (preferredFood) {
case PEANUTS:
    foodAmount = min(peanutsOwned, 10);  // 最多喂10个单位
    peanutsOwned -= foodAmount;
    break;
case CARROTS:
    foodAmount = min(carrotsOwned, 10);
    carrotsOwned -= foodAmount;
    break;
case BANANAS:
    foodAmount = min(bananasOwned, 10);
    bananasOwned -= foodAmount;
    break;
}

// 如果有食物可喂，则喂食并增加围栏脏污度
if (foodAmount > 0) {
    enclosure.addDirt(foodAmount);
    cout << getName() << " fed " << foodAmount << " units of food to animals in the "
         << enclosure.getEnclosureTypeName() << " enclosure." << endl;
}
```

```
}

// ==================== AnimalEnclosure 类实现 ====================
AnimalEnclosure::AnimalEnclosure(AnimalType t) : type(t), dirtLevel(0), isOpen(true), daysClose
    // 根据类型初始化动物数组
    switch (type) {
    case ELEPHANT_ENCLOSURE:
        animalCount = 1;
        animals = new Animal*[animalCount];
        animals[0] = new Elephant("Dumbo", 2000.0f, 1.5f);
        break;
    case GIRAFFE_ENCLOSURE:
        animalCount = 2;
        animals = new Animal*[animalCount];
        animals[0] = new Giraffe("Melman", 1200.0f, 2.8f);
        animals[1] = new Giraffe("Gerry", 1300.0f, 3.0f);
        break;
    case MONKEY_ENCLOSURE:
        animalCount = 3;
        animals = new Animal*[animalCount];
        animals[0] = new Monkey("Kong", 80.0f, 0.8f);
        animals[1] = new Monkey("Cheeks", 70.0f, 0.7f);
        animals[2] = new Monkey("Banana", 75.0f, 0.75f);
        break;
    default:
        animalCount = 0;
        animals = nullptr;
        break;
    }
}

AnimalEnclosure::~AnimalEnclosure() {
    // 释放动物对象内存
    for (int i = 0; i < animalCount; i++) {
        delete animals[i];
    }
    delete[] animals;
}

void AnimalEnclosure::addDirt(int units) {
    dirtLevel += units;
}
```

```
void AnimalEnclosure::closeForCleaning() {
    if (isOpen) {
        isOpen = false;
        dirtLevel = 0;
        daysClosed++;
    }
}

void AnimalEnclosure::reopen() {
    isOpen = true;
}

bool AnimalEnclosure::shouldClean() const {
    return dirtLevel > 2000;
}

string AnimalEnclosure::getEnclosureTypeName() const {
    switch (type) {
    case ELEPHANT_ENCLOSURE:
        return "Elephant";
    case GIRAFFE_ENCLOSURE:
        return "Giraffe";
    case MONKEY_ENCLOSURE:
        return "Monkey";
    default:
        return "Unknown";
    }
}

AnimalType AnimalEnclosure::getEnclosureType() const {
    return type;
}

int AnimalEnclosure::getDirtLevel() const {
    return dirtLevel;
}

int AnimalEnclosure::getDaysClosed() const {
    return daysClosed;
}

// ====================== Zoo 类实现 ======================
Zoo::Zoo() : openDays(0),
```

```
            totalRevenue(0.0f),
            exitReason(NONE) {
    // 创建围栏
    enclosures = new AnimalEnclosure*[3];
    enclosures[0] = new AnimalEnclosure(ELEPHANT_ENCLOSURE);
    enclosures[1] = new AnimalEnclosure(GIRAFFE_ENCLOSURE);
    enclosures[2] = new AnimalEnclosure(MONKEY_ENCLOSURE);

    // 创建工作人员
    keeper = new ZooKeeper("Bob", 35);
    seller = new FoodSeller("Alice", 28);
}

Zoo::~Zoo() {
    // 释放内存
    for (int i = 0; i < 3; i++) {
        delete enclosures[i];
    }
    delete[] enclosures;

    delete keeper;
    delete seller;

    // 释放访客内存
    for (auto visitor : visitors) {
        delete visitor;
    }
    visitors.clear();
}

void Zoo::simulateDay() {
    openDays++;
    cout << "\n===== Day " << openDays << " =====" << endl;

    // 清空昨天的访客
    for (auto visitor : visitors) {
        delete visitor;
    }
    visitors.clear();

    // 1. 生成今天的访客
    random_device rd;
    mt19937 gen(rd());
```

```
uniform_int_distribution<> adultDistribution(20, 40);
uniform_int_distribution<> childDistribution(1, 3);

int adultCount = adultDistribution(gen);

// 创建成人访客
vector<Adult*> adults;
for (int i = 0; i < adultCount; i++) {
    string name = "Adult" + to_string(i + 1);
    Adult* adult = new Adult(name, 25 + rand() % 30);
    adults.push_back(adult);
    visitors.push_back(adult);

    // 每个成人带1-3个孩子
    int childCount = childDistribution(gen);
    vector<Child*> children;

    for (int j = 0; j < childCount; j++) {
        string childName = "Child" + to_string(i + 1) + "-" + to_string(j + 1);
        Child* child = new Child(childName, 5 + rand() % 10);
        children.push_back(child);
        visitors.push_back(child);
    }

    // 2. 成人购票
    Money ticketCost = adult->buyTickets(childCount);
    totalRevenue = totalRevenue + ticketCost;

    // 3. 成人购买食物并分配给孩子
    adult->buyFood(*seller, children);
}

// 4. 孩子喂食动物
for (auto visitor : visitors) {
    Child* child = dynamic_cast<Child*>(visitor);
    if (child) {
        // 随机选择一个围栏喂食
        int enclosureIndex = rand() % 3;
        child->feedAnimal(*enclosures[enclosureIndex]);
    }
}

// 5. 检查围栏状态并清洁
```

```
    bool keeperCleanedToday = false;
    for (int i = 0; i < 3; i++) {
        if (enclosures[i]->shouldClean() && !keeperCleanedToday) {
            keeper->cleanEnclosure(*enclosures[i]);
            keeperCleanedToday = true;
        } else if (!enclosures[i]->isOpen) {
            // 如果围栏已关闭，则重新开放
            enclosures[i]->reopen();
        }
    }

    // 6. 打印当天状态
    cout << "Visitors today: " << visitors.size() << " ("
         << adults.size() << " adults, "
         << visitors.size() - adults.size() << " children)" << endl;

    cout << "Food remaining: Peanuts=" << seller->getFoodAmount(PEANUTS)
         << ", Carrots=" << seller->getFoodAmount(CARROTS)
         << ", Bananas=" << seller->getFoodAmount(BANANAS) << endl;

    cout << "Zoo revenue today: " << totalRevenue.toString() << endl;

    // 检查终止条件
    checkTerminationConditions();
}

void Zoo::runSimulation() {
    while (exitReason == NONE) {
        simulateDay();
    }

    // 打印总结
    printSummary();
}

void Zoo::checkTerminationConditions() {
    // 检查食物是否售罄
    if (seller->isFoodSoldOut()) {
        exitReason = FOOD_SOLD_OUT;
        cout << "\nSIMULATION ENDED: " << seller->getSoldOutFoodName() << " has been sold out."
    }

    // 检查饲养员是否辞职
```

```
    if (keeper->getCleaningDays() >= 10) {
        exitReason = KEEPER_QUIT;
        cout << "\nSIMULATION ENDED: Zookeeper " << keeper->getName()
             << " has quit after cleaning for " << keeper->getCleaningDays() << " days." << endl
    }
}

void Zoo::printSummary() {
    cout << "\n====== Zoo Simulation Summary ======" << endl;
    cout << "Days operated: " << openDays << endl;

    cout << "Exit reason: ";
    switch (exitReason) {
    case FOOD_SOLD_OUT:
        cout << "Food sold out (" << seller->getSoldOutFoodName() << ")" << endl;
        break;
    case KEEPER_QUIT:
        cout << "Zookeeper quit after " << keeper->getCleaningDays() << " cleaning days" << endl
        break;
    default:
        cout << "Unknown" << endl;
    }

    cout << "Total revenue: " << totalRevenue.toString() << endl;
    cout << "Food seller revenue: " << seller->getRevenue().toString() << endl;

    cout << "Enclosure status:" << endl;
    for (int i = 0; i < 3; i++) {
        cout << "  " << enclosures[i]->getEnclosureTypeName()
             << " enclosure was closed for " << enclosures[i]->getDaysClosed() << " days." << en
    }
}
```

main.cpp

```cpp
#include <cstdlib>
#include <ctime>
#include <iostream>

#include "zoo.h"


int main() {
    // 设置随机数种子
    srand(static_cast<unsigned int>(time(nullptr)));

    std::cout << "=== Zoo Simulation Starting ===" << std::endl;

    // 创建动物园实例
    Zoo zoo;

    // 运行模拟
    zoo.runSimulation();

    std::cout << "=== Zoo Simulation Ended ===" << std::endl;

    system("pause");


    return 0;
}
```