Problem Statement 1: Basic Character Skill Usage

Design a simple game system where different character types use skills.

1. **Base `Character` Class:** Create a base class `Character` with `char name[32]`, `int health`, `int mana`, and an `int type_id` (e.g., `WARRIOR_TYPE`, `MAGE_TYPE` enums). It should also have an `int last_action_id` (e.g., `NONE_ACTION`, `ATTACK_ACTION`, `HEAL_ACTION`).
2. **Derived Character Classes:**
   - `Warrior` inherits from `Character`. Has an `int armor`.
   - `Mage` inherits from `Character`. Has an `int intelligence`.
3. **`SkillEngine` Class:** Implement a `SkillEngine` class with a `static` method `void useSkill(Character* character)`. This method should determine which basic skill a character uses based on their type and simple stats.
   - **Moderate Logic:** The `useSkill` method will use an `if-else if` structure based on `character->type_id`.
     - If `WARRIOR_TYPE`:
       - If `health` is below 30, set `last_action_id = HEAL_ACTION` (simulating using a potion).
       - Else, set `last_action_id = ATTACK_ACTION`.
     - If `MAGE_TYPE`:
       - If `mana` is below 20, set `last_action_id = NONE_ACTION` (cannot cast).
       - Else, set `last_action_id = ATTACK_ACTION` (simulating casting a spell).
   - The method should print what action the character performs. No virtual functions for skill usage.

Problem Statement 2: Simple Order Fulfillment Status

Create a system to track the status of different types of orders.

1. **Base `Order` Class:** Create a base class `Order` with a `char order_id[16]`, and an `int type_id` (e.g., `ONLINE_ORDER_TYPE`, `PHONE_ORDER_TYPE` enums). It also has an `int status_id` (e.g., `PENDING_STATUS`, `PROCESSING_STATUS`, `COMPLETED_STATUS`, `CANCELLED_STATUS` enums).
2. **Derived Order Classes:**
   - `OnlineOrder` inherits from `Order`. Has a `bool payment_received`.
   - `PhoneOrder` inherits from `Order`. Has a `bool customer_confirmed`.
3. **`OrderProcessor` Class:** Implement an `OrderProcessor` class with a `static` method `void updateOrderStatus(Order* order)`. This method advances the order's status based on its type and simple conditions.
   - **Moderate Logic:** The `updateOrderStatus` method will use an `if-else if` structure based on `order->type_id` and its current `status_id`.
     - If `ONLINE_ORDER_TYPE`:
       - If `status_id` is `PENDING_STATUS` and `payment_received` is true, change `status_id` to `PROCESSING_STATUS`.
       - If `status_id` is `PROCESSING_STATUS`, change `status_id` to `COMPLETED_STATUS`.
     - If `PHONE_ORDER_TYPE`:
       - If `status_id` is `PENDING_STATUS` and `customer_confirmed` is true, change `status_id` to `PROCESSING_STATUS`.

        - If `status_id` is `PROCESSING_STATUS`, change `status_id` to `COMPLETED_STATUS`.
    - The method should print the order's ID and its new status. No virtual functions for status updates.

Problem Statement 3: Package Handling with Weight-Based Fees

Develop a basic system for handling different types of packages and calculating simple fees.

1. **Base `Package` Class:** Create a base class `Package` with a `char tracking_id[16]`, `double weight_kg`, and an `int type_id` (e.g., `STANDARD_PACKAGE_TYPE`, `OVERNIGHT_PACKAGE_TYPE` enums).
2. **Derived Package Classes:**
    - `StandardPackage` inherits from `Package`. Has a `bool fragile`.
    - `OvernightPackage` inherits from `Package`. Has a `bool signature_required`.
3. **`ShippingCalculator` Class:** Implement a `ShippingCalculator` class with a `static` method `double calculateShippingCost(Package* package)`.
    - **Moderate Logic:** The `calculateShippingCost` method will use an `if-else if` structure based on `package->type_id` and its weight.
        - If `STANDARD_PACKAGE_TYPE`:
            - Base cost: $5.00 + (`weight_kg` * $1.50).
            - If `fragile` is true, add an additional $2.00 fee.
        - If `OVERNIGHT_PACKAGE_TYPE`:
            - Base cost: $15.00 + (`weight_kg` * $3.00).
            - If `signature_required` is true, add an additional $5.00 fee.
    - The method should return the calculated cost. No virtual functions for cost calculation.

Problem Statement 4: Device Health Check

Create a simple system to check the health of different types of electronic devices.

1. **Base `Device` Class:** Create a base class `Device` with a `char device_name[32]`, `int battery_level`, and an `int type_id` (e.g., `PHONE_TYPE`, `LAPTOP_TYPE` enums).
2. **Derived Device Classes:**
    - `Phone` inherits from `Device`. Has a `bool sim_card_detected`.
    - `Laptop` inherits from `Device`. Has a `bool external_power_connected`.
3. **`HealthMonitor` Class:** Implement a `HealthMonitor` class with a `static` method `char* getDeviceStatus(Device* device)`. (Return a static char array or string literal for simplicity).
    - **Moderate Logic:** The `getDeviceStatus` method will use an `if-else if` structure based on `device->type_id` and simple attribute checks.
        - If `PHONE_TYPE`:
            - If `battery_level` < 10, return "Critical Battery".
            - Else if `sim_card_detected` is false, return "No SIM Card".
            - Else, return "Operational".
        - If `LAPTOP_TYPE`:
            - If `battery_level` < 5, return "Battery Extremely Low".
            - Else if `external_power_connected` is false and `battery_level` < 20, return "Charge Recommended".

        - Else, return "Operational".
    - No virtual functions for status checking.

Problem Statement 5: Animal Sound Production

Design a simple system where different animals produce sounds.

1. **Base `Animal` Class:** Create a base class `Animal` with a `char species[32]`, `int age`, and an `int type_id` (e.g., `DOG_TYPE`, `CAT_TYPE` enums).
2. **Derived Animal Classes:**
    - `Dog` inherits from `Animal`. Has a `bool has_collar`.
    - `Cat` inherits from `Animal`. Has a `bool is_sleeping`.
3. **`SoundProducer` Class:** Implement a `SoundProducer` class with a `static` method `void makeSound(Animal* animal)`.
    - **Moderate Logic:** The `makeSound` method will use an `if-else if` structure based on `animal->type_id`.
        - If `DOG_TYPE`:
            - If `age` < 1 (puppy), print "Yip!".
            - Else if `has_collar` is true, print "Woof!".
            - Else, print "Grrr...".
        - If `CAT_TYPE`:
            - If `is_sleeping` is true, print "Purrr...".
            - Else if `age` > 10 (old cat), print "Meow!".
            - Else, print "Hiss!".
    - No virtual functions for sound production.
