This lab exercise helps you to demonstrate your understanding of key object-oriented programming features: data abstraction/encapsulation, polymorphism, inheritance, and dynamic binding. We will create a foundational framework for a simple game that involves character interactions in a fantasy setting.

Below is the description of the program for the game that explains different classes and functions. After every class description, there is a section **“Assignment Task”.** Implement these changes and upload the completed program with run.

**Assignment Details:**

1. **Weapon Class:**
   1. Create a class named Weapon.
   2. Each weapon should have 2 protected members name and damage. Basically, when a weapon hits a character, it applies damage.
   3. There are three member functions
      1. displayInfor() - This function displays the weapon name and damage
      2. getdamage() - This function returns a damage. This demosntrate abstraction of data – *‘damage'*
      3. specialAttack() - Pure virtual function – This function needs to be overriden in derived class. It adds a certain integer value on top of damage.
   4. There are two derived classes: Sword and Staff
   5. Sword : specialAttack() needs to be overridden to return damage + 5 in this derived class
   6. Staff: specialAttack() needs to be overridden to return damage + 10 in this derived class. Overriding functions is an example of polymorphism.
   7. **Assignment Task :** 
      1. Complete the code for Staff class. It should be similar to Sword class with 2 functions – constructor and a special attack. Special attack function should return damage + 5 value.
2. **Game Character Classes:**
   1. **Purpose**: Serves as the base class for all characters in the game.
   2. **Attributes**:
      1. name: A string representing the character's name.
      2. health: An integer representing the character's health points.
      3. weapon: A reference to a Weapon object, allowing each character to equip a weapon. This is an example of encapsulation
   3. **Methods**:
      1. Constructor: Initializes the character's name, health, and weapon.
      2. displayInfo(): A function (not implemented here) to display character details.
      3. equipWeapon(Weapon& w): Allows the character to equip a new weapon.
      4. attack(Character& op): A virtual function allowing characters to attack each other.
      5. specialAttack(Character& op): A virtual function for a unique attack method (to be defined in derived classes).
   4. getHit(int damage): Reduces the character's health by the damage taken.
   5. **Assignment Task:** Character class is given.
3. **Warrior Class**:
   1. **Purpose**: A derived class that represents warrior characters, inheriting from Character.
   2. **Attributes**:
      1. strength: An integer representing the warrior's physical strength.
   3. **Methods**:
      1. Constructor: Initializes the warrior's name, health, weapon, and strength.
      2. attack(Character& op): Overrides the base class attack method, providing specific attack behavior for warriors.
      3. specialAttack(Character& op): Overrides to define a special attack unique to warriors.
      4. getHit(int damage): Overrides the base class method to define how warriors react to damage.
      5. set Strength(int s): A setter method for modifying the warrior's strength.
   4. **Assignment Task:** Warrior Class is given.
4. **Wizard Class**:
   1. **Purpose**: Another derived class representing wizard characters, also inheriting from Character.
   2. **Attributes**:
      1. magicPower: An integer representing the wizard's magical abilities.
   3. **Methods**:
      1. Constructor: Initializes the wizard's name, health, weapon, and magic power.
      2. attack(Character& op): Overrides the attack method to provide specific behavior for wizards.
      3. specialAttack(Character& op): Implements a unique special attack for wizards.
      4. getHit(int damage): Overrides to define how wizards handle incoming damage.
      5. healingSpell(): A method (not shown here) that likely allows the wizard to heal themselves or others.
      6. setMagicPower(int mp): A setter method for adjusting the wizard's magic power.
   4. **Assignment Task:** Write the definition of derived class based in Character.hpp file. Complete the definitions of all the functions of derived classes in Character.cpp file. Function definitions are already placed .
5. **GameTest.cpp** -
   1. Run the final program that creates objects of the above classes and demonstrates all the different concepts of OOP. There is one simple test case already written.
   2. **Assignment Task**: Create an object of a new weapon and change the warrior object using equipweapon. Change weapons and attack again

**Submission Guidelines:**

* Submit your C++ code cpp and hpp files
* Also upload proof of running your program.