CS 162
Intro to Programming II

Inheritance I
Basic Idea

Parent/base/super class
More general

Child/derived/sub class
More specific

Character
Wizard
Inheritance

The derived class (child class) inherits:

- All the member variables of the base class (or parent class)
- All the member functions of the base class (or parent class)

Why is inheritance useful?

- Code reuse (character has code that is in common with derived classes)
/* Character.hpp */

class Character {
  public:
    Character(std::string nameValue, int hitPointsValue, int strengthValue, int intelligenceValue);
    std::string getName() { return name; }
    int getHitPoints() { return hitPoints; }
    int getStrength() { return strength; }
    int getIntelligence() { return intelligence; }
    void setHitPoints(int hitPointsValue) { hitPoints = hitPointsValue; }
    void attack(Character& c);

  private:
    std::string name;
    int hitPoints;
    int strength;
    int intelligence;
};
Implementation

/* Character.cpp */
Character::Character(std::string nameValue, int hitPointsValue, int strengthValue, int intelligenceValue) :
  name(nameValue), hitPoints(hitPointsValue), strength(strengthValue), intelligence(intelligenceValue) {
}
void Character::attack(Character& c) {
  int damage = rand() % strength;
  int newHitPoints = c.getHitPoints() - damage;
  if( newHitPoints < 0 )
    newHitPoints = 0;
  c.setHitPoints(newHitPoints);
}
Inheritance

/* Wizard.hpp */
class Wizard : public Character {
public:
    Wizard(std::string nameValue, int hitPointsValue,
           int strengthValue, int intelligenceValue,
           int magicPointsValue);
    void heal(Character& c, int numHitPoints);
private:
    int magicPoints;
};

It is derived (inherits) from Character
Inheritance

/* Wizard.hpp */
class Wizard : public Character {
public:
    Wizard(std::string nameValue, int hitPointsValue,
           int strengthValue, int intelligenceValue,
           int magicPointsValue);
    void heal(Character& c, int numHitPoints);
private:
    int magicPoints;
};

It inherits from Character
Variables- name, hitPoints, strength, intelligence
Functions- getName(), getHitPoints(), getStrength(),
            getIntelligence(), setHitPoints
Inheritance

/* Wizard.hpp */
class Wizard : public Character {
public:
    Wizard(std::string nameValue, int hitPointsValue,
           int strengthValue, int intelligenceValue,
           int magicPointsValue);
    void heal(Character& c, int numHitPoints);
private:
    int magicPoints;
};

It has its own constructor
Inheritance

/* Wizard.hpp */
class Wizard : public Character {
public:
    Wizard(std::string nameValue, int hitPointsValue,
            int strengthValue, int intelligenceValue,
            int magicPointsValue);
    void heal(Character& c, int numHitPoints);
private:
    int magicPoints;
};

It adds function heal()
And variable magicPoints
/* Wizard.cpp */
Wizard::Wizard(std::string nameValue, int hitPointsValue,
    int strengthValue, int intelligenceValue,
    int magicPointsValue) :
    Character(nameValue, hitPointsValue, strengthValue, intelligenceValue),
    magicPoints(magicPointsValue) {
}
void Wizard::heal(Character& c, int numHitPoints) {
    int newHitPoints = c.getHitPoints() + numHitPoints;
    c.setHitPoints(newHitPoints);
}
Suppose you wrote the Wizard constructor without the call to the Character constructor:

```cpp
Wizard::Wizard(std::string nameValue, int hitPointsValue, int strengthValue,
               int intelligenceValue,
               int magicPointsValue) :
    magicPoints(magicPointsValue) {
}
```

What happens?
Constructor

• It calls the default constructor for Character i.e. Character()
• If the default constructor for Character doesn’t exist, it won’t compile
• Recommendation: Always explicitly invoke the base class constructor in the initialization section of the derived class constructor
• The call to the base class constructor is the first thing done in the derived class constructor
Inheritance

• The red part of the code below from Wizard.cpp seems strange.
  
  ```cpp
  void Wizard::heal(Character& c, int numHitPoints) {
    int newHitPoints = c.getHitPoints() + numHitPoints;
    c.setHitPoints(newHitPoints);
  }
  ```

• Why not just call c.hitPoints?
  
  ```cpp
  void Wizard::heal(Character& c, int numHitPoints) {
    int newHitPoints = c.HitPoints + numHitPoints;
    c.setHitPoints(newHitPoints);
  }
  ```
Inheritance

• The red part of the code below from Wizard.cpp seems strange.
  
  void Wizard::heal(Character& c, int numHitPoints) {
    int newHitPoints = c.getHitPoints() + numHitPoints;
    c.setHitPoints(newHitPoints);
  }

• Why not just call c.hitPoints?
  
  void Wizard::heal(Character& c, int numHitPoints) {
    int newHitPoints = c.HitPoints + numHitPoints;
    c.setHitPoints(newHitPoints);
  }

Not allowed as hitPoints is private
Access

• **Private** member variables and member functions are inherited but not accessible to the derived class
  – The derived class needs to call the public accessors and mutators of the base class to change private member variables

• **Protected** member variables and member functions are accessible by:
  – The class itself
  – And any derived classes
Interface

/* Character.hpp */
class Character {

public:
    Character(std::string nameValue, int hitPointsValue, int strengthValue, int intelligenceValue);
    std::string getName() { return name; }
    int getHitPoints() { return hitPoints; }
    int getStrength() { return strength; }
    int getIntelligence() { return intelligence; }
    void setHitPoints(int hitPointsValue) { hitPoints = hitPointsValue; }
    void attack(Character& c);

protected:
    std::string name;
    int hitPoints;
    int strength;
    int intelligence;

};
Now accessible to derived classes
Inheritance

• The Wizard class inherits the following functions from the Character class
  • `int getName()`
  • `int getHitPoints()`
  • `int getStrength()`
  • `int getIntelligence()`
  • `void setHitPoints(int hitPointsValue)`
  • `void attack(Character& c)`;

• Any of the above can be called by a Wizard object
Inheritance

• This code will call the attack() function from the Character class
• You can redefine the attack() function in the Wizard class
• In fact, you can redefine any inherited function from Character

Wizard w(“Gandalf”,100,10,10,10);
Character c(“Bob”,100,10,10);
w.attack(c);
class Wizard : public Character {
    public:
        void attack(Character& c);
    private:
        /* etc. */
};
Overload vs Redefining

• Important distinction-
  Redefining a function definition:
  • the new function definition in the derived class
    has the same number and types of parameters

Overloading a function:
• The function in the derived class has a different
  number of parameters or different types of
  parameters from the function in the base class
Overloading

- You can overload a new attack() function for a Wizard object
- Note it has two parameters
- Wizard objects now have 2 attack() functions

```cpp
class Wizard : public Character {
public:
    void attack(Character& c);
    void attack(Character& c, bool useMagic);
};
```
Inheritance

• Suppose the Wizard class had only one attack function declared in Wizard.hpp:
  
  \[
  \text{void attack(Character& c, bool useMagic);}
  \]

• Then the Wizard class would still have two attack functions
  1. One attack function (with two parameters) from the Wizard class
  2. Another attack function (with one parameter) from the Character class