CS 162
Intro to Programming II

Polymorphism Ia

Virtual Functions

A pointer to a derived (child) class is type-compatible with a pointer to its base (parent) class eg.

```
Wizard *w1 = new Wizard(“Larry",100, 20,10,100);

Character *c1 = w1;
```

Virtual Functions

You can call any member function of the Character class using the c1 object eg.

```
Wizard *w1 = new Wizard(“Larry",100, 20,10,100);

Character *c1 = w1;

std::cout << c1->getName();
std::cout << c1->getStrength();
std::cout << c1->getIntelligence();
std::cout << c1->getHitPoints();
```

Virtual Functions

You can call any member function of the Character class using the c1 object eg.

```
Wizard *w1 = new Wizard(“Larry",100, 20,10,100);

Character *c1 = w1;

c1->heal(Bob,100);  <<< Will not compile.
                    Heal is part of Wizard class
w1->heal(Bob,100);  <<< This will work
```

Virtual Functions

- What if you call the attack function?
- Which gets called, the one in Wizard or the one in Character?

```
Wizard *w1 = new Wizard(“Larry",100, 20,10,100);

Character *c1 = w1;
c1->attack(Bob);
```

The attack function in Character is called.

Virtual Functions

```
Wizard *w1 = new Wizard(“Larry",100, 20,10,100);

Character *c1 = w1;
c1->attack(Bob);
```

- This may not be what you intend.
- c1 points to a Wizard object and you want that attack function called.

Virtual Functions

You can change this be making the attack function declaration virtual in Character by making this change to the .hpp file:

```
/* Character.hpp */
class Character {
Public:
   Virtual void attack(Character& c);
   /* … */
}
```

You do not need the virtual keyword in the .cpp file

Virtual Functions

By making this change the following code now calls the attack function from the Wizard class:

```
Wizard *w1 = new Wizard(“Larry",100, 20,10,100);

Character *c1 = w1;

c1->attack(Bob);
```

Virtual Functions

- The previous code is an example of polymorphism
- Polymorphism allows the attack() function to take different forms depending on the actual type of the derived class
- More generally, polymorphism allows you to write a generic piece of code that can **be applied to objects of different types**

Virtual Functions

- Virtual functions allow late binding (also called dynamic binding) which is a key part of polymorphism
- Late binding is the technique of waiting until run time (not compile time) to determine which implementation of a function to run
- As opposed to static binding
- See page 943

Virtual Functions

```
/* Character.hpp */
class Character {
Public:
   Virtual void attack(Character& c);
   /* … */
}
```

`attack(Character& c)` was declared virtual in the parent class it will automatically be declared virtual in all derived classes. You do not need to declare it virtual yourself.

Note: The text recommends doing it anyway.

Virtual Functions

- Which is which, for changing the behavior of inherited functions?
  - Overriding refers to doing this change to a virtual function
  - Redefining refers to doing this change to a non-virtual function
  - Overloading refers to the definition of different functions within the same class with the same name and different parameter lists.