The Conceptual Side of Classes (cont)

- When we define a Student we might do the following:

```cpp
class Student
{
    string name;
    string address;
    string localPhone;
    int studentID;
};
```

- Here we have defined a class of people... a Student
- When we allocate a variable of type Student...
  - We actually “create” one “member” of the class Student.
  - Hmmm... a class of people...

The Conceptual Side of Classes (cont)

- Sometimes multiple classes have similarities:

```cpp
class Student
{
    string name;
    string address;
    string localPhone;
    int studentID;
};
class Instructor
{
    string name;
    string address;
    string phone;
    string employeeID;
};
```

The Conceptual Side of Classes (cont)

- Sometimes the similarities are common to a broader class than the class being defined

- In the case of Student and Instructor, consider the common fields:
  - name
  - address
  - phone

- Suppose we create a class called “Person”, as follows:

```cpp
class Person
{
    string name;
    string address;
    string phone;
};
```

The Conceptual Side of Classes (cont)

- Now maybe you’d think that we could do this:

```cpp
class Student
{
    Person imAPerson;
    int studentID;
};
class Instructor
{
    Person imAPerson;
    int employeeID;
};
```

- We can, in fact do this.
  - But then any instance would have to access fields in Person through the imAPerson member variable.
Inheritance

- A better way to do this is with Inheritance
- In C++, when one class inherits another
  - all public (and protected) member variables in the "base class" are accessible from the "derived class" as if they were declared right in the derived class.
- In our example:
  - Person is the base class
  - Student is the derived class
- To declare Student as being a derivation of Person, do this:

```cpp
class Student : public Person
{
    int studentID;
};
```

```cpp
class Instructor : public Person
{
    int employeeID;
};
```

Inheritance (cont)

- Now, given the following declarations:

```cpp
class Person
{
public:
    string name;
    string address;
    string localPhone;
};
```

```cpp
class Student : public Person
{
public:
    int studentID;
};
```

Let's see this in action:

```cpp
int main()
{
    Student aStudent;
    aStudent.name = "Jon Doe";   // Defined in Person
    aStudent.address = "12 Park Place"; // Defined in Person
    aStudent.phone = "555-1212"; // Defined in Person
    aStudent.studentID = 442221; // Defined in Student
    ...
}
```

Protected Members

- A derived class may access any of the public members of the base class, and so can anyone else using the base class directly.
- A derived class may NOT access any of the private members of the base class, nor anyone else using the base class directly.
- A derived class may access any of the protected members of the base class, but no one using the base class directly may access them.
- To mark a member variable or function as protected, do the following:

```cpp
class Person
{
    protected:
        string name;
        string address;
        string phone;
};
```

Protected Members (cont)

- To clarify, when a member function or variable follows a protected keyword:
  - Only member functions defined in a derived class may access the protected member functions/variables in the base class
  - All other classes (not derived from the base class) may not access the protected member functions/variables
- Let's look at some code:

```cpp
class Person
{
public:
    void setInfo(string Name,string Addr,string Phone);
protected:
    string name;
    string address;
    string phone;
};
```
Protected Members (cont)
* Now Consider a Derived Class...

```cpp
class Student : public Person {
public:
    void printInfo();
    int getID() { return studentID; }
private:
    int studentID;
};
```

```cpp
void Student::printInfo()
{
    cout << "Name: " << name << endl; // name, address and
    cout << "Addr: " << address << endl; // phone are defined
    cout << "Phone: " << phone << endl; // in the base class
}
```

Protected Members (cont)
* Finally, let’s use it...

```cpp
int main()
{
    Student aStudent;
    aStudent.name = "Joe Student";  // ??
    aStudent.address = "166 Phelps Lane"; // ??
    aStudent.phone = "555-1212"; // ??
    aStudent.printInfo();
}
```

Protected Members (cont)
* But they can be accessed inside of the derived class

```cpp
void Student::printInfo()
{
    cout << "Name: " << name << endl; // name, address and
    cout << "Addr: " << address << endl; // phone are defined
    cout << "Phone: " << phone << endl; // in the base class
}
```

* The Person class had its own public method for setting info:

```cpp
void Person::setInfo(string Name, string Addr, string Phone)
{
    name = Name;
    addr = Addr;
    phone = Phone;
}
```

Protected Members (cont)
* So the right way to do it (in this particular case) is:

```cpp
int main()
{
    Student aStudent;
    // Now set the information. Remember, setInfo() is
    // defined in the "Person" class:
    aStudent.setInfo("Joe Student","166 Phelps Lane",
                        "555-1212");
    aStudent.printInfo();
}
```

* Let’s see this in action...

Cleaning Up Our Implementation
* You might think that the Person class should print its own data:

```cpp
class Person
{
public:
    void setInfo(string Name, string Addr, string Phone);
    void printInfo();
private:
    string name;
    string address;
    string phone;
};
```

```cpp
void Person::printInfo()
{
    cout << "Name: " << name << endl;
    cout << "Addr: " << address << endl;
    cout << "Phone: " << phone << endl;
}
```
cleaning up our implementation

- That makes a certain amount of sense...

```cpp
class Instructor : public Person
{
private:
  int employeeID;
};
```

```cpp
int main()
{
  Instructor anInstructor;
anInstructor.setInfo("Ron BiMagoli","120 Maple Ave", "555-1313");
anInstructor.printInfo();
}
```

- Would work just as well (without having to define printInfo() in each derived class)

Cleaning Up Our Implementation

- But what about things we might want to print out in a derived class that aren’t present in the base class?
  - studentID field in the Student class.
  - employeeID field in the Employee class.
  - Is there any way to include them in the Person::printInfo() member function?
  - Not really, but we can do the next best thing.
  - We could have a special definition of printInfo which is used when we’re dealing with a Student class instance

```cpp
void Student::printInfo()
{
  cout << "Student ID: " << studentID << endl;
  // Hmmmm, how can I call the printInfo() from Person?
}
```

```cpp
void Person::printInfo()
{
  cout << "Name: " << name << endl;
  cout << "Addr: " << address << endl;
  cout << "Phone: " << phone << endl;
}
```

- Let’s find out...

Demonstration #3

Redefining Base Class Member Functions

- Let’s consider the following code:

```cpp
void printPersonInfo(Person &aPerson)
{
  aPerson.printInfo();
}
```

```cpp
void printPersonInfo(Person &aPerson)
{
  aPerson.printInfo();
}
```

- Let’s consider the following code:

```cpp
int main()
{
  Student aStudent;
  Instructor anInstructor;
  aStudent.setInfo("Joe Student","1 E Main St","555-1212");
aStudent.studentID = 33445;
anInstructor.setInfo("Ron D","120 Maple Ave","555-1313");
anInstructor.employeeID = 12345;
printPersonInfo(aStudent);
printPersonInfo(anInstructor);
}
```

- Open the window

Overriding

- Yes, it does work.
- Whenever a derived class defines a member function that is also defined in the base class it is said that the definition in the derived class overrides the definition in the base class.
- In our previous example, Student::printInfo() overrides Person::printInfo()
- However, consider the case where we’d like to write a function that can take a Person as an argument and will cause that person’s printInfo method to be invoked:

```cpp
void printPersonInfo(Person &aPerson)
{
  aPerson.printInfo();
}
```

```cpp
int main()
{
  Student aStudent;
  Instructor anInstructor;
  aStudent.setInfo("Joe Student","1 E Main St","555-1212");
aStudent.studentID = 33445;
anInstructor.setInfo("Ron D","120 Maple Ave","555-1313");
anInstructor.employeeID = 12345;
printPersonInfo(aStudent);
printPersonInfo(anInstructor);
}
```

- Open the window

Overriding (cont)

- Let’s consider the following code:
Demonstration #4

Redefining Base Class Member Functions II

Overriding (cont)
- So, wait a minute. Did the compiler forget that we overrode Person::printInfo() in the derived class Student?
- No, it's only doing what it was told to do!
- Recall that we didn't get any complaints from the compiler when we passed an Instructor and a Student in to the function printPersonInfo(Person &).
- It's legal to do that, since Instructor and Student are derived from Person, the compiler thinks we want to treat whatever argument is passed in as a Person.
- And, since inside the scope of printPersonInfo the argument passed is an instance of a Person, Person::printInfo() is used when we call aPerson.printInfo().
- Well, doesn't that make overriding somewhat useless?

Overriding (cont)
- We'll find out more, next lecture!

Lecture 10

Final Thoughts