Topic 2

1. Inheritance hierarchies
2. Implementing derived classes
3. Overriding member functions
4. Virtual functions and polymorphism

Implementing Derived Classes

Now for those different kinds of questions.

Each of the different kinds of questions *IS-A* Question

so we code by starting with the base class (**`Question`**) and then we write code for what makes the different types *special versions* of more general **`Question`** type.

Derived Classes Inherit All Data and Functions from the Base

Through inheritance, each of the derived classes has the data members and member functions set up in class `Question`.

– plus “specialness” which is not inherited, but added in the definition of each derived class

(We don’t rewrite the member functions)

(code reuse in action)

Derived Classes: `ChoiceQuestion`

```
class ChoiceQuestion : public Question
{
public:
    // New and changed member
    // functions will go here
private:
    // Additional data members
    // will go here
};
```

The `:` denotes inheritance

`ChoiceQuestion` – Analysis of the Problem

After a programmer has set the question text and the several multiple choice answers the **`ChoiceQuestion`** object is asked to display something like:

```
In which country was the inventor of C++ born?
1: Australia
2: Denmark
3: Korea
4: United States
```

`ChoiceQuestion` must have:

`ChoiceQuestion` must have:

- Storage for the various choices for the answer
  - **`Question`** has the question text and correct answer, not these
- A member function for adding another choice
- A display function
  - The designer of the **`Question`** class could not have known how to display this sort of multiple choice question. It only has the question itself, not the choices.
  - In the **`ChoiceQuestion`** class you will have to *rewrite* the display function **`display`**.
    - This is called ***overriding*** a member function.

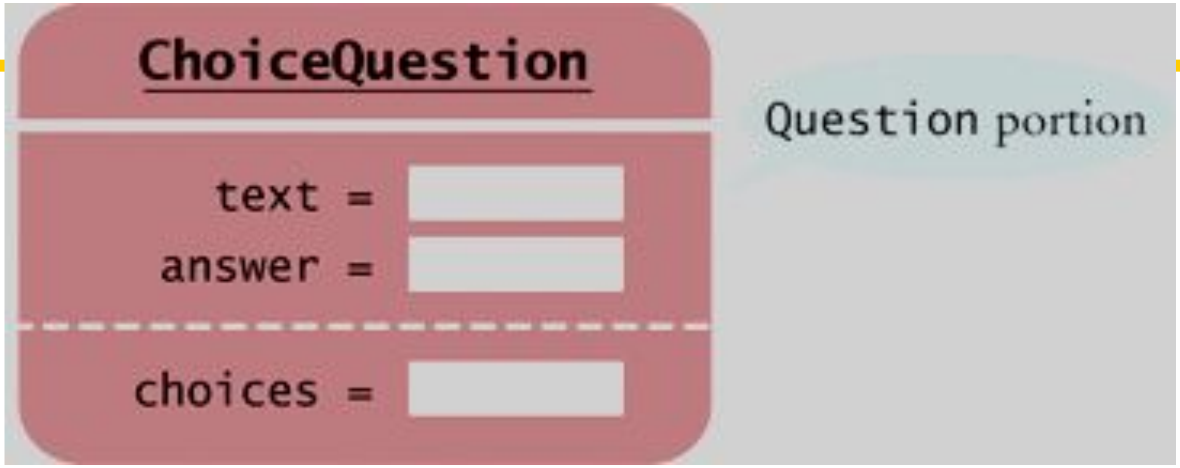
Derived Classes: `ChoiceQuestion` Code

After specifying the class you are inheriting from, you only write the differences:

```
class ChoiceQuestion : public Question
{
public:
   ChoiceQuestion();
   void add_choice(string choice, bool correct);
   void display() const;
private:
   vector<string> choices;
};
```

Derived Class Diagram

```
class Question
{
public:
   Question();
   void set_text(string
   question_text);
   void set_answer(string
   correct_response);
   bool check_answer(string response)
   const;
   void display() const;
private:
   string text;
   string answer;
};
class ChoiceQuestion : public Question
{
public:
   ChoiceQuestion();
   void add_choice(string choice, bool
   correct);
   void display() const;
private:
   vector<string> choices;
};
```



ChoiceQuestion is *one* type,
made of two parts:
inherited `(text, answer)`
and new `(choices)`.

Derived Classes Cannot Directly Read/Write Private Base Data

The derived class inherits all data members
and all functions that it does not override.

Consider:

```
Choice_question choice_question;
choice_question.set_answer("2");
//calls public member function of base class

choice_question.answer = "2"; //ERROR
// will not compile - private data only
   accessible in member functions of base
```

Big C++ by Cay Horstmann

Derived Classes and Private Data of the Base Class

This means that when you are writing the **`ChoiceQuestion`** member functions, you cannot directly access any private data members in **`Question`**.

The derived class functions, just like any other function, can only use the public interface of the base class.

add_choice Member Function

```
void ChoiceQuestion::add_choice(string choice, bool correct)
{
   choices.push_back(choice);
   if (correct) //change answer to this one's number
   {
      // Convert choices.size() to string
   string num_str = to_string(choices.size());

      // Set num_str as the answer, using public function:
      set_answer(num_str);
   }
}
```

Practice It: Derived Classes from `Critter (1)`

Here is the Critter class, from which we will derive others:

(file `critter.h`)

```
class Critter
{
public:
   Critter(); //Constructs a critter at position 0 with blank history.
   string get_history() const; /** @return the history */
   void add_history(string event); /**Adds to the history
      @param event the event to add to the history */

   int get_position() const;
   void move(int steps); // @param steps the number of steps to move.
   void act(); //The action of this critter in one pass of simulation.

private:
   int position;
   vector<string> history;
};
```

Practice It: Derived Classes from `Critter` (2)

Define a class `Sloth` derived from `Critter`. Sloths alternate between eating and sleeping. Add the word "eat" or "sleep" to the history each time the `act` function is called.

```
#include <iostream>
using namespace std;
#include "critter.h"
/**   A sloth eats and sleeps.*/
class Sloth : public Critter
{
public:
   Sloth();
   . . .
private:
   . . .
};
Sloth::Sloth(){   . . .}
```

Practice It: Derived Classes from `Critter` (3)

Define a derived class `NervousCritter` from `Critter`. A nervous critter moves nervously between positions 0 and 1. In the `act` function, carry out the appropriate move.

```
#include <iostream>
using namespace std;
#include "critter.h"
/** A nervous critter moves back and forth between
positions 0 and 1.*/
class NervousCritter . . .
{
public:
   . . .
};

. . .
```

Common Error 10.1: Private Inheritance

Here is the class definition for `ChoiceQuestion` again.
It has one small error.   Can you find it?

```
class ChoiceQuestion : Question
{
public:
   ChoiceQuestion();
   void add_choice(string choice, bool correct);
   void display() const;
private:
   vector<string> choices;
};
```

Common Error: Private Inheritance

If you do not specify `public` inheritance,
you get *private inheritance.*

```
class ChoiceQuestion : __________ Question
{
public:
   ChoiceQuestion();
   void add_choice(string choice, bool correct);
   void display() const;
private:
   vector<string> choices;
};
```

Private inheritance: only member functions of `ChoiceQuestion` get to call member functions of `Question`.

Whenever a `main()` invokes a Question member function on a ChoiceQuestion object, the ***compiler will flag it as an error:***

Common Error: Replicating Base Class Members

A derived class has no direct access to base class private data. The following code therefore won't compile, with an "unknown identifier in this scope: text" error message:

```
ChoiceQuestion::ChoiceQuestion(string quest_txt)
{
   text = quest_txt; //text is in the base class
}
```

When some programmers encounter that compiler error,

they just start hacking…

Common Error: Replicating Base Class Members (2)

And an “easy” fix seems to be to add the data member that the compiler is complaining about.

```
class ChoiceQuestion : public Question
{
 ChoiceQuestion::ChoiceQuestion(string quest_txt)
 ...
 private:
   vector<string> choices;
   string text; //hacking addition, a mistake
}
```

Now it compiles, but it doesn’t set the correct text! Such a `ChoiceQuestion` object has 2 data members named `text.` The constructor sets one, and the display function uses the other.

Instead of replicating a base-class data member, you need to call a member function to initialize it: **`set_text(quest_txt)`**

Inheritance Is For Behaviors, Not Values

Consider a program that tracks fuel efficiency of cars by logging the distance traveled and the refueling amounts.

Some cars are hybrids. Should you create a derived class HybridCar? Not in this application.

Hybrids don’t behave any differently than other cars when it comes to driving and refueling. They just have better MPG. A single Car class with a data member

```
double miles_per_gallon;
```

is entirely sufficient.

However, in a program showing repairs of different kinds of vehicles, you need a separate class HybridCar, as their repairs behave differently.

Calling the Base-Class Constructor (1)

A derived-class constructor can only initialize the data members of the derived class.

But the base-class data members also need to be initialized.

Unless you specify otherwise, the base-class data members are initialized with the default constructor of the base class.

Calling the Base-Class Constructor (2)

If you want to use another base-class constructor, you use an **initializer list.** For example, suppose the Question base class had a constructor setting the question text. Here is a derived-class constructor calling that base-constructor:

```
ChoiceQuestion::ChoiceQuestion(string question_text)
   : Question(question_text)
{ . . .
}
```

The constructor of a derived class can supply arguments to a base-class constructor.

The base-class constructor acts before the derived class code inside the { }.