Topic 4

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

An Array of Questions?

In the **`main`** function of that last program, there was some repetitive code to display each question and check the responses.

It would be nicer if all questions were collected in a single array, though some are base `Question`s and some are `ChoiceQuestion`**s**.

You could then loop to present them to the user:

First Attempt at Code for an Array of Questions

```
const int QUIZZES = 2;
Question quiz[QUIZZES];
quiz[0].set_text("Who was the inventor of C++?");
quiz[0].set_answer("Bjarne Stroustrup");

ChoiceQuestion cq;
cq.set_text("In which country was the inventor of C++ born?");
cq.add_choice("Australia", false);
...
quiz[1] = cq;

for (int i = 0; i < QUIZZES; i++)
{
   quiz[i].display();
   cout << "Your answer: ";
   getline(cin, response);
   cout << quiz[i].check_answer(response) << endl;
}
```

The Slicing Problem

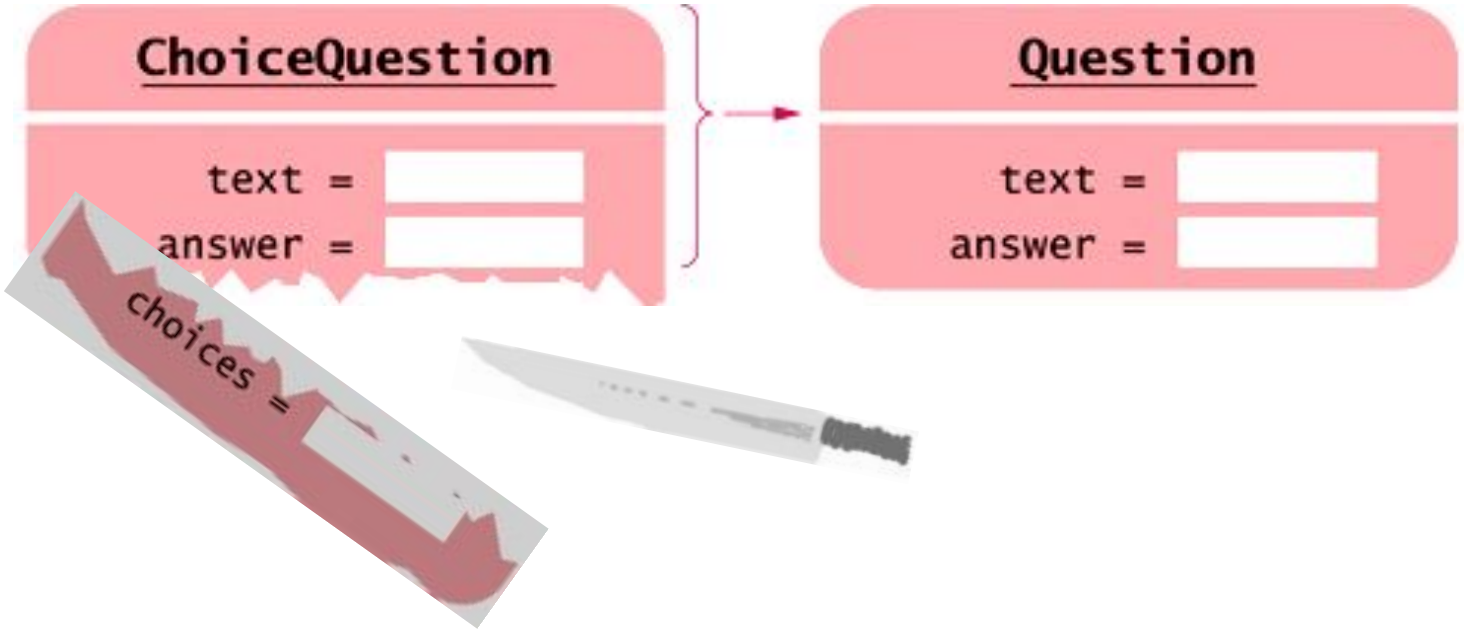
However, is it really working?

Here's a run of the program:

```
Who was the inventor of C++?
Your answer: Bjarne Stroustrup
true
In which country was the inventor of C++ born?
Your answer:
```

Where are the choices for the ChoiceQuestion?

The Slicing Problem (2)



The array `quiz` holds `Question`s. The compiler realizes that a `ChoiceQuestion` is a special case of a `Question`. Thus it permits :

```
quiz[1] = cq;
```

However, a **`ChoiceQuestion`** object has 3 data members, whereas a `Question` has just 2. There is no room to store the derived-class data in the array.

That data simply gets sliced away when you assign a derived-class object to a base-class variable.

Use Pointers Instead

To access objects from *different* classes
in a class hierarchy, use an array of pointers to objects
instead of an array of objects.

(to avoid slicing).

Pointers to the various objects all have the same size:
the size of a memory address.

Pointers to Base and Derived Classes

Pointers to base classes can hold pointers
to *ANY* object publicly derived from it

– as far down the inheritance chain as you want to go.

The opposite will not work:

Assigning a base pointer to a derived pointer location
will generate a compiler error.

Pointers to Base and Derived Classes (2)

To manage all of these, use a

```
vector<Question*> qv;
```

or an array of

```
Question* quiz[2];
```

and store

only pointers to the different kinds of **`Questions`**

- Fill-in-the-blank
- Numeric
  Free response
- Choice (single)
  - Choice (multiple – inherits from Choice (single)
- Essay

Code for Pointers to Base and Derived Classes

Notice the use of **new** and **->** :

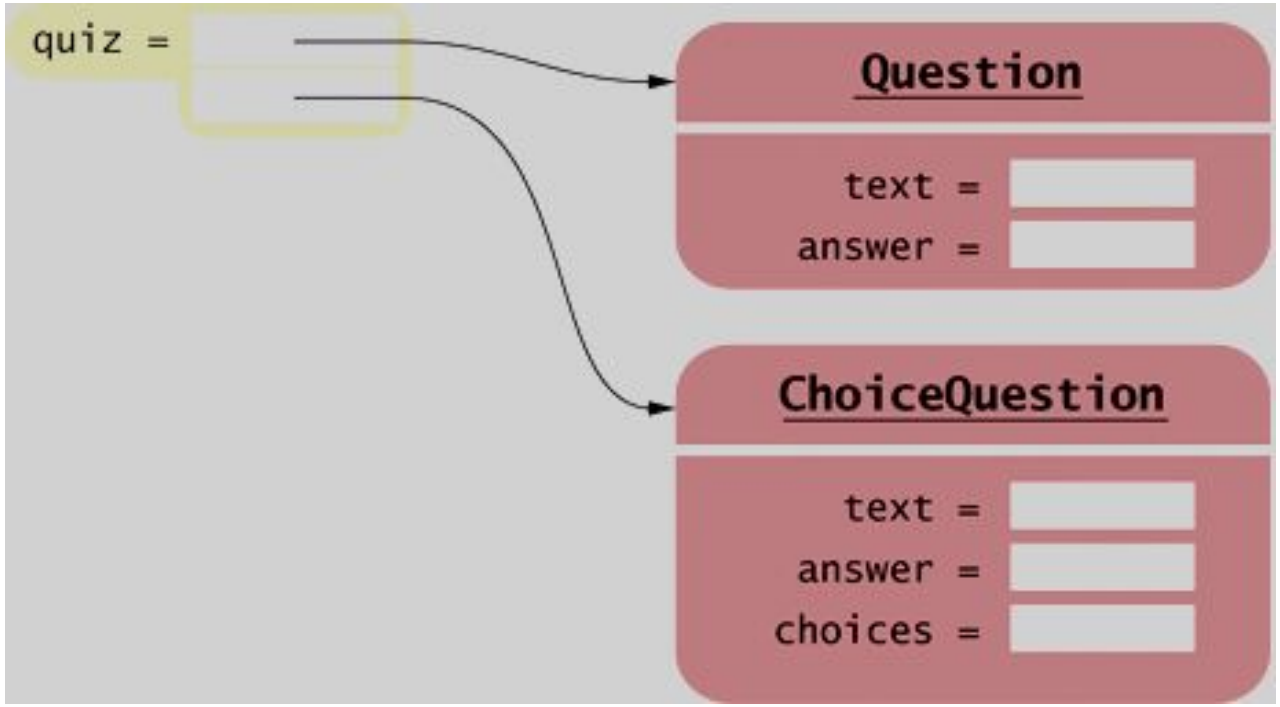
```
Question* quiz[2];


quiz[0] = new Question;
quiz[0] -> set_text("Who was the inventor of C++?");
quiz[0] -> set_answer("Bjarne Stroustrup");

ChoiceQuestion* cq_pointer = new ChoiceQuestion;
cq_pointer -> set_text("In which country… …C++ born?");
cq_pointer -> add_choice("Australia", false);
...
quiz[1] = cq_pointer;
```

Diagram of Pointers to Base to Manage Base and Derived

```
Question* quiz[2];
quiz[0] = new Question;
quiz[1] = new ChoiceQuestion;
```



The Code to Display All the Questions in the Pointer Array

The code to present all questions – any kind of **`Question`** – is:

```
for (int i = 0; i < QUIZZES; i++)
{
   quiz[i] -> display();
   cout << "Your answer: ";
   getline(cin, response);
   cout << quiz[i] -> check_answer(response) << endl;
}
```

Virtual Functions: Motivation

When you call the **`display`** member function on a **`Question*`** pointer that currently contains a pointer to a **`ChoiceQuestion`**, you want the **`choices`** to be displayed.

Virtual Functions: Motivation (2)

But that's not what happens.

For reasons of efficiency, by default, the call

```
quiz[i]->display();
```

always calls **`Question::display`**
because the type of **`quiz[i]`** is **`Question*`**.

Virtual Functions: Mechanism

In C++, you must alert the compiler
that the function call needs to *not* be the default,

that the function should be the one *in* the thing pointed to.

(How?)

You use the **`virtual`** reserved word for this purpose.

Virtual Functions Must Be Declared in the Base Class

The **`virtual`** reserved word must be used in the base class.

```
class Question
{
public:
   Question();
   void set_text(string question_text);
   void set_answer(string correct_response);
   virtual bool check_answer(string response) const;
   virtual void display() const;
private:
...
};
```

All functions with the same name and parameter types in derived classes are then automatically virtual.

Virtual Functions: Derived Classes

Although not needed, it is considered good practice to write the **`virtual`** reserved word in the derived-class functions in the derived-class interface.

```
class ChoiceQuestion : public Question
{
public:
   ChoiceQuestion();
   void add_choice(string choice, bool correct);
   virtual void display() const;
private:
...
};
```

You do **NOT** Write `virtual` in the Function Definition

You do not write **`virtual`** in the function definition, only in the interface header in the `class xxx{};` statement:

```
void Question::display() const
{
   cout << text << endl;
}
```

When a virtual function is called, the compiled code determines the type of the implicit parameter at run time. The appropriate function for that object is then called. For example:

```
quiz[i]->display();
```

always calls the function belonging to the actual type of the object to which **`quiz[i]`** points — either **`Question::display`** or **`ChoiceQuestion::display`**.

Polymorphism

The quiz vector collects a mixture of all kinds of **`Questions`**.

Such a collection is called *polymorphic*
(literally, “of multiple shapes”).

Objects in a polymorphic collection have some commonality but are not necessarily of the same type.

Inheritance is used to express this commonality.

`virtual` functions enable variations in behavior.

Polymorphism (2)

Each object knows
on its own
how to carry out *its* specific version
of these general tasks:

"Display the question"
(my way)
and
"Check a response"
(my way).

Polymorphism (3)

Suppose we want to have a new
kind of question for calculations,
where we are willing to accept an approximate answer.

All we need to do is to define a new class **`NumericQuestion`**,
with its own **`check_answer`** function.

Then we can populate any quiz vector with a
mixture of plain questions, choice questions,
and these new numeric questions.

They will fit in just fine because:
they IS-A Questions.

Complete Program: Polymorphism & Virtual Functions (1)

```
// question.h
#ifndef QUESTION_H
#define QUESTION_H

#include <string>
using namespace std;

class Question
{
public:
   /**
      Constructs a question with empty question and answer.
   */
   Question();

   /**
      @param question_text the text of this question
   */
   void set_text(string question_text);

   /**
      @param correct_response the answer for this question
   */
   void set_answer(string correct_response);
```

Complete Program: Polymorphism & Virtual Functions (2)

```
   /**
      @param response the response to check
      @return true if the response was correct,
      false otherwise
   */
   virtual bool check_answer(string response) const;

   /**
      Displays this question.
   */
   virtual void display() const;

private:
   string text;
   string answer;
};

#endif
```

Complete Program: Polymorphism & Virtual Functions (3)

```
// choicequestion.h

#ifndef CHOICEQUESTION_H
#define CHOICEQUESTION_H

#include <vector>
#include "question.h"

class ChoiceQuestion : public Question
{
public:
   /**
      Constructs a choice question with no choices.
   */
   ChoiceQuestion();
   /**
      Adds an answer choice to this question.
      @param choice the choice to add
      @param correct true if this is the correct choice,
      false otherwise
   */
   void add_choice(string choice, bool correct); virtual void display() const;

private:
   vector<string> choices;
};
#endif
```

Complete Program: Polymorphism & Virtual Functions (4)

```
// sec04/demo.cpp

#include <iostream>
#include "question.h"
#include "choicequestion.h"
int main()
{
   string response;
   cout << boolalpha;

   // Make a quiz with two questions
   const int QUIZZES = 2;
   Question* quiz[QUIZZES];

   quiz[0] = new Question;
   quiz[0]->set_text("Who was the inventor of C++?");
   quiz[0]->set_answer("Bjarne Stroustrup");
```

Complete Program: Polymorphism & Virtual Functions (5)

```
   ChoiceQuestion* cq_pointer = new ChoiceQuestion;
   cq_pointer->set_text("In which country was the inventor of C++
born?");
   cq_pointer->add_choice("Australia", false);
   cq_pointer->add_choice("Denmark", true);
   cq_pointer->add_choice("Korea", false);
   cq_pointer->add_choice("United States", false);
   quiz[1] = cq_pointer;

   // Check answers for all questions
   for (int i = 0; i < QUIZZES; i++)
   {
      quiz[i]->display();
      cout << "Your answer: ";
      getline(cin, response);
      cout << quiz[i]->check_answer(response) << endl;
   }
   return 0;
}
```