Convert a Structure to a Class

Highlights of this lab:

In this lab you will learn how convert a structure to a class implementation.

- What is the difference between a member function and a non-member function?

Lab Exercise: 

- Convert a Cat structure with helper functions into a class implementation.

Click the little computer above for a detailed description.
NOTE: Your lab instructor will tell you what will be marked for this lab.

What is the difference between a member function and a non-member function?

There are some major differences.

1. A non-member function always appears ***outside*** of a class and does NOT use the scope resolution operator (`::`).

   By contrast, the prototype for the member function appears ***inside*** the body of the class. When defining the function (in the .cpp file), we use the scope resolution operator to identify that that function is a member of a particular class.

   For instance, if we had a class called `StudentClass` and wanted to define the implementation of a `printAll` function, we could write:

```
void StudentClass::printAll () const  //Member Function
{
       //details of this implementation
       //printing name, id and marks
}
```

   The `StudentClass::` specifies that the `printAll` function belongs to the "StudentClass" class.

   By contrast, a non-member function has no scope resolution operator. It does not belong to a class. In the implementation file, our non-member function would be written as:

```
void printStudent (const StudentStruct& stu)  //Non-member Function
{
       //details of this implementation
       //printing name, id and marks
}
```

2. Another difference between member functions and non-member functions is how they are called (or invoked) in the main routine. Consider the following segment of code:

```
int main()
{
       int i;
       StudentClass stuC;  //declare a object of StudentClass
       StudentStruct stuS; //declare an instance of StudentStruct

       stuC.printAll (); //invoking the member function
       printStudent (stuS);  //invoking the non-member function
}
```

   Notice how the member function uses dot notation. The non-member does not.

3. If you are passing an instance of a structure to a function and you convert your code to a class implementation, you will have to make some adjustments. Consider the print functions in both the structure and class implementations:

```
void printStudent (const StudentStruct& c)  //Non-member Function
{
    cout << endl;
    cout << "Name: " << c.name << endl;
    cout << "ID: " << c.id << endl;
    for (int i=0; i < 3; i++)
    {
        cout << "Test " << i+1 << ": " << c.mark[i] << " " << endl;
    }
}
```

   Remember with a member function implementation, you will have direct access to the values within the instance (on the left-hand side of the dot). The above code will be modified in two ways:

   i. You no longer need to pass one instance as an argument.
   ii. You do not need to use dot notation inside the function to access that instance's data.

```
void StudentClass::printAll () const  //Member Function
{
    cout << endl;
    cout << "Name: " << name << endl;
    cout << "ID: " << id << endl;
    for (int i=0; i < 3; i++)
    {
        cout << "Test " << i+1 << ": " << mark[i] << " " << endl;
    }
}
```

A side-by-side example is below:

Implementation as a Class

```
#include <string>
#include <iostream>
using namespace std;

//-----------------------------------------------
// This would be the StudentClass.h file
//-----------------------------------------------
class StudentClass
{
    private:
      string name;
      int id;
      int mark[3];
    public:
      //getters and setters are missing
      //default constructor is also missing
      StudentClass (string aName, int anId, int mark1, int mark2, int mark3);
      void printAll () const;
};
//-----------------------------------------------
// This would be the StudentClass.cpp file
//-----------------------------------------------
StudentClass::StudentClass (string aName,
            int anId, int mark1, int mark2, int mark3)
{
     name = aName;
     id = anId;
     mark[0] = mark1;
     mark[1] = mark2;
     mark[2] = mark3;
}

void StudentClass::printAll () const
{
   cout << endl;
   cout << "Name: " << name << endl;
   cout << "ID: " << id << endl;
   for (int i=0; i < 3; i++)
   {
       cout << "Test " << i+1 << ": " << mark[i] << " " << endl;
   }
}

//-----------------------------------------------
// This would be the main.cpp file
//-----------------------------------------------
int main ()
{
  StudentClass averageStudent ("John Doe", 299999999, 65, 65, 65);
  averageStudent.printAll ();

  return 0;
}
```

Implementation as a Structure

```
#include <string>
#include <iostream>
using namespace std;

//-----------------------------------------------
// This would be the StudentStruct.h file
//-----------------------------------------------
struct StudentStruct
{
     string name;
     int id;
     int mark[3];
};
void initStruct (StudentStruct &stu, string aName,
            int anId, int mark1, int mark2, int mark3);
void printStudent (const StudentStruct &c);
//-----------------------------------------------
// This would be the StudentStruct.cpp file
//-----------------------------------------------
void initStruct (StudentStruct &stu, string aName,
            int anId, int mark1, int mark2, int mark3)
{
     stu.name = aName;
     stu.id = anId;
     stu.mark[0] = mark1;
     stu.mark[1] = mark2;
     stu.mark[2] = mark3;
}

void printStudent (const StudentStruct &c)
{
    cout << endl;
    cout << "Name: " << c.name << endl;
    cout << "ID: " << c.id << endl;
    for (int i=0; i < 3; i++)
    {
        cout << "Test " << i+1 << ": " << c.mark[i] << " " << endl;
    }
}

//-----------------------------------------------
// This would be the main.cpp file
//-----------------------------------------------
int main ()
{
  StudentStruct averageStudent;
  initStruct (averageStudent, "John Doe", 299999999, 65, 65, 65);
  printStudent (averageStudent);

  return 0;
}
```

---

4. Lab Exercise — Convert a Structure to a Class

Convert a structure into a class implementation.

Details:

- Start with the code provided to you
- Modify all of the functions so that they are member functions. Use the scope resolution operator (::) to show that an implementation of a function is part of a class
- Change the initialization function into a constructor with arguments.
- Add a default constructor that initializes things to 0 or empty string. Please note that you should use a for loop to initialize all the values of `furColours`
- Add one getter to return the first `furColour`.
- Modify the calls in main to reflect the change in the functions.
- Be sure to add the keyword `const` after functions that do not change the data.
- If you want, you can turn the `isTaller` function into an overloaded `operator>`.

Here are two sample runs that you can try in the lab:

```
hercules[5]% ./main
Please describe the cat
Please enter a length: 48
Please enter a height: 25
Please enter a tail length: 31
Please enter an eye colour: blue
Please enter a description of the fur (long, medium, short, none): short
Please enter the colours of the fur (separated by a space or a newline
character). Add "done" at the end: grey white done
The average cat's first colour is orange
-------------------------------------------------------------
This is myCat:
Length: 48  Height: 25  Tail Length: 31
Eye Colour: blue  Fur Classification: short
Cat Colours: grey white
My cat is taller than the average cat
hercules[6]%
hercules[6]% ./main
Please describe the cat
Please enter a length: 44
Please enter a height: 22
Please enter a tail length: 30
Please enter an eye colour: yellow
Please enter a description of the fur (long, medium, short, none): long
Please enter the colours of the fur (separated by a space or a newline
character). Add "done" at the end: black white orange done
The average cat's first colour is orange
-------------------------------------------------------------
This is myCat:
Length: 44  Height: 22  Tail Length: 30
Eye Colour: yellow  Fur Classification: long
Cat Colours: black white orange
My cat is shorter than the average cat
My cat is a calico
```