Lab 1

Lab 1

In this module, we've learned how to create separate header files so that we can call them without having to place them into the `main` file. This will dramatically reduce the length of code needed in the `main` file. Make sure your `main` file (`lab1.cpp`) and your header file (`class.h`) contain the code below.

main

```cpp
#include "class.h"

int main() {

  //add code below this line

  Greeting g("Hello world");
  cout << g.GetGreeting() << endl;
  g.SetGreeting("Hi world");
  cout << g.GetGreeting() << endl;

  //add code above this line

  return 0;

}
```

header

```cpp
#ifndef CLASS_H
#define CLASS_H
#include <iostream>
using namespace std;

//add class definitions below this line

class Greeting {
  public:
    Greeting(string g) {
      greeting = g;
    }

    string GetGreeting() {
      return greeting;
    }

    void SetGreeting(string new_greeting) {
      greeting = new_greeting;
    }

    void PrintGreeting(){
      cout << GetGreeting() << endl;
    }

  private:
    string greeting;
};

//add class definitions above this line

#endif
```

Now, try to add to the existing `class.h` by including a class called `Farewell`. This class should include all of the equivalent functions of `Greeting`.

For example, if:

```cpp
Farewell f("Goodbye world");
cout << f.GetFarewell() << endl;
f.SetFarewell("Bye world");
cout << f.GetFarewell() << endl;
```

Then the result should be:

```
Goodbye world
Bye world
```

Lab 2

Lab 2

You are provided the following header and `main` files:

▼ point.h

```
#ifndef SLOPE_H
#define SLOPE_H

//add definitions below this line



//add definitions above this line

#endif
```

▼ slope.h

```
#ifndef SLOPE_H
#define SLOPE_H

//add definitions below this line



//add definitions above this line

#endif
```

▼ lab2.cpp

```cpp
#include <iostream>
using namespace std;
#include "point.h"
#include "slope.h"

int main() {

  //add code below this line



  //add code above this line

  return 0;

}
```

In this lab, you'll be working with these three files. The idea is to create a struct called `point` within `point.h`, then create a static function called `CalculateSlope` within `slope.h`, and finally run a few commands within `lab2.cpp` to print some results.

point.h

In this header file, we will create a struct called `point` which contains just two attributes, `int x` and `int y`. Remember, structs are public by default which means they are easily accessible.

```cpp
//add definitions below this line

struct point {
  int x;
  int y;
};

//add definitions above this line
```

slope.h

In this header file, we will create a class called `Slope`. This class only has one static member function called `CalculateSlope`. `CalculateSlope` takes in two `point` structures and returns the calculated slope between them.

```
//add definitions below this line

class Slope {
  public:
    static double CalculateSlope(point a, point b) {
      return ( (double) (b.y - a.y) / (double) (b.x - a.x) );
    }
};

//add definitions above this line
```

lab2.cpp

Now it's time to test our header files within main. We are going to create two point structures, assign values to their attributes, then call CalculateSlope on the two points. Note that we do not need to create a Slope object before calling CalculateSlope since it is a static function. Simply use the scope resolution operator :: to access the function as shown in the code below.

```
//add code below this line

point a;
point b;
a.x = 0;
a.y = 0;
b.x = 2;
b.y = 2;
cout << Slope::CalculateSlope(a, b) << endl;

//add code above this line
```

challenge

Try these variations:

- Replace the code in main with:

```
//add code below this line

point a;
point b;
a.x = 1;
a.y = 2;
b.x = 10;
b.y = 20;
cout << Slope::CalculateSlope(a, b) << endl;

//add code above this line      —>2
```

- Change the entire `lab2.cpp` to look like this:

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

struct point {
  int x;
  int y;
};

class Slope {
  public:
    static double CalculateSlope(point a, point b) {
      return ( (double) (b.y - a.y) / (double) (b.x - a.x)
         );
    }
};

int main() {

  //add code below this line

  point a;
  point b;
  a.x = 1;
  a.y = 2;
  b.x = 10;
  b.y = 20;
  cout << Slope::CalculateSlope(a, b) << endl;

  //add code above this line

  return 0;

}
```

Note that including the header files produces the same result as including the struct `point` and class `Slope` within `main`. However, the header files enable the `main` program to be less cluttered with definitions.

Lab Challenge

Problem

Create a `BankAccount` struct in the IDE to the left which has two `double` attributes `checking` and `savings`. Create a function called `ToString` within the struct that prints a representation of a `BankAccount` struct which includes these attributes.

You **MUST** use `struct` within your code in order to receive credit for this challenge.

▼ **Given Code**

```cpp
#include <iostream>
#include <iomanip>
using namespace std;

//add definitions below this line




//add definitions above this line

int main() {

  //DO NOT EDIT code below this line

  BankAccount account1;
  account1.checking = 2432;
  account1.savings = 89.52;
  BankAccount account2;
  account2.checking = 1998;
  account2.savings = 239.43;
  account1.ToString();
  account2.ToString();

  //DO NOT EDIT code above this line

  return 0;

}
```

▼ **Setting decimal places**

You can use the code `cout << setprecision(2) << fixed` followed by a specified value to set that value to two decimal places.

Testing Your Code

The code in `main` is used to test your definitions. **DO NOT EDIT** this code!

```
//DO NOT EDIT code below this line

BankAccount account1;
account1.checking = 2432;
account1.savings = 89.52;
BankAccount account2;
account2.checking = 1998;
account2.savings = 239.43;
account1.ToString();
account2.ToString();

//DO NOT EDIT code above this line
```

Expected Result

```
BankAccount[checking=2432.00, savings=89.52]
BankAccount[checking=1998.00, savings=239.43]
```