Lab: Parameters

The Slope Formula

You are going to write a function that takes in 4 **doubles** as parameters. These parameters represent two sets of coordinate points labeled as `x1`, `y1`, `x2`, and `y2`. The function will then use these points to calculate the slope that their line creates and then prints that slope to the user.

Function Header

First, we need to set up the function header. As usual, we will start off with our return type. Since the result is simply printing the slope, we will use `void` as the return type. Additionally, we'll name the function `GetSlope()`.

```
void GetSlope() {
}
```

Parameters

The function should take 4 **doubles** as parameters named `x1`, `y1`, `x2`, and `y2`.

```
void GetSlope(double x1, double y1, double x2, double y2) {
}
```

Printing the Slope

The final step is to print the slope, but we'll need the slope formula in order to do that. The slope formula is defined as `(y2 - y1) / (x2 - x1)`.

```
cout << (y2 - y1) / (x2 - x1) << endl;
```

Testing the Function

In order to use a function, you’ll need to call it by specifying its name within the `main()` function. Note that the function requires parameters so we’ll need to provide some arguments in order for the function to work properly. Let’s use the points `(3, 2)` and `(5, 6)` as our coordinates which correspond to `(x1, y1)` and `(x2, y2)` respectively. Lastly, it is a best practice to include `return 0` as the last statement inside `main()`.

```
int main() {
  GetSlope(3, 2, 5, 6);
  return 0;
}
```

The `GetSlope()` function will apply the slope formula `(6 - 2) / (5 - 3)` and print the result `2.0` to the user. Make sure to also include documentation so that other users can understand your function.

▼ **Code**

```
/**
 * This function prints the slope between two sets
 * of coordinate points
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return No return value
 */
void GetSlope(double x1, double y1, double x2, double y2) {
  cout << (y2 - y1) / (x2 - x1) << endl;
}

int main() {
  GetSlope(3, 2, 5, 6);
  return 0;
}
```

Lab: Variable Scope

Local and Global Variables

For this lab, we are going to be adding local and global variables to our previously created `GetSlope()` function. Remember that **global** variables exist *outside* of functions while **local** variables exist *inside* functions. Depending on how you declare your local and global variables, they will behave differently per situation.

Global Variables

First, let's add some global variables to our program.

```
double input1;
double input2;
double input3;
double input4;
```

The GetSlope() Function

As from before, the function will still take 4 **doubles** as parameters named `x1`, `y1`, `x2`, and `y2`. However, we're going to implement two different calculations within the function. Specifically, we are going to calculate the difference between the **y** coordinates first, then calculate the difference between the **x** coordinates. These calculations will then be assigned to **local** variables called `y_change` and `x_change`. Finally, the function will print the quotient between `y_change` and `x_change`, which is also the slope itself.

```
void GetSlope(double x1, double y1, double x2, double y2) {
  double y_change = y2 - y1;
  double x_change = x2 - x1;
  cout << y_change / x_change << endl;
}
```

Testing the Function

To make things more dynamic, we’ll actually make use of a `cin` within the `main()` function. `cin` will take in inputs from the user and assign them to our 4 global variables `input1`, `input2`, `input3`, and `input4`. These inputs will later correspond to `x1`, `y1`, `x2`, and `y2`. Having `cin` enables the user to decide what the coordinate points will be.

```
int main() {
  cout << "Enter first x coordinate: " << endl;
  cin >> input1;
  cout << "Enter first y coordinate: " << endl;
  cin >> input2;
  cout << "Enter second x coordinate: " << endl;
  cin >> input3;
  cout << "Enter second y coordinate: " << endl;
  cin >> input4;

  GetSlope(input1, input2, input3, input4);
}
```

You’ll notice that you have access to the Terminal which you will use to input any coordinate points you want. If you enter `3`, `2`, `5`, and `6` respectively, you should get `2` since `cout` removes all trailing zeros. Click the `TRY IT` button to compile and run the program.

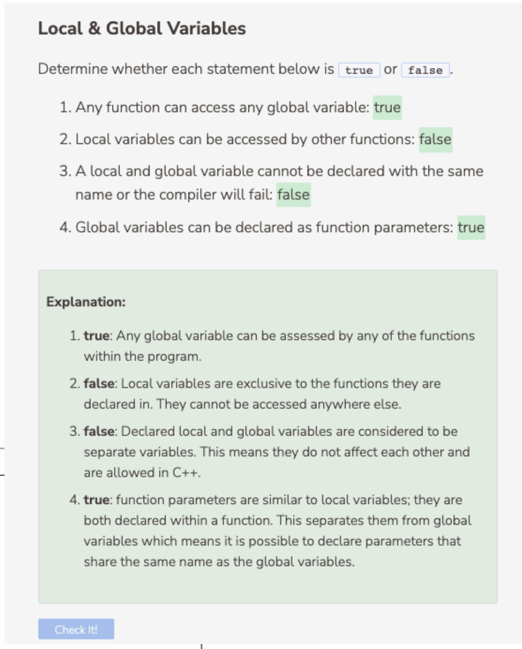
▼ **Code**

```
double input1; //global
double input2; //global
double input3; //global
double input4; //global

/**
 * This function prints the slope between two sets
 * of coordinate points by calculating their coordinate
 * changes separately
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return No return value
 */
void GetSlope(double x1, double y1, double x2, double y2) {
  double y_change = y2 - y1;
  double x_change = x2 - x1;
  cout << y_change / x_change << endl;
}

int main() {
  cout << "Enter first x coordinate: " << endl;
  cin >> input1;
  cout << "Enter first y coordinate: " << endl;
  cin >> input2;
  cout << "Enter second x coordinate: " << endl;
  cin >> input3;
  cout << "Enter second y coordinate: " << endl;
  cin >> input4;

  GetSlope(input1, input2, input3, input4);
}
```



Local & Global Variables

Determine whether each statement below is `true` or `false`.

1. Any function can access any global variable: true
2. Local variables can be accessed by other functions: false
3. A local and global variable cannot be declared with the same name or the compiler will fail: false
4. Global variables can be declared as function parameters: true

Explanation:

1. **true**: Any global variable can be assessed by any of the functions within the program.
2. **false**: Local variables are exclusive to the functions they are declared in. They cannot be accessed anywhere else.
3. **false**: Declared local and global variables are considered to be separate variables. This means they do not affect each other and are allowed in C++.
4. **true**: function parameters are similar to local variables; they are both declared within a function. This separates them from global variables which means it is possible to declare parameters that share the same name as the global variables.

info

Program Flow

After the program is initiated, the global variables will be created first. Next, `main()` will run. Although commonly written last, `main()` will always be the first function to run by default in C++. The lines of code within `main()` will be executed in the order of their appearance.

Lab: Return Values

Returning a Value

When the result of a function is simply a print statement, it is considered to be a `void` function. `void` functions do not have a return type, meaning they do not return data back to the user. To return data, use the keyword `return` followed by the data. Note that functions with `return` must be declared with same data type as the data that they return. For example, a function that returns an `double` must be declared in the header as a `double` function.

```
double GetSlope(double x1, double y1,
                double x2, double y2) { //replace void with
        double
  double y_change = y2 - y1;
  double x_change = x2 - x1;
  return y_change / x_change; //returns a double
}
```

Modifying the Return Value

Notice that our function returns a single double, which is nice but not extremely helpful when it comes to determining *rise* and *run* for slopes (`rise / run`). Let's say we want instead to express the slope in the `rise / run` format. `rise` is the change in `y` values and `run` is the change in `x` values. Unfortunately, we can't simply do `return y_change + " / " + x_change`. Why? Because the `" / "` is a string which is not compatible with the current `return` value of `double`. One way around this is to convert the doubles into strings. Doing so will force us to change our `double` function into a `string` function.

```
string GetSlope(double x1, double y1,
                double x2, double y2) { //replace double with
        string
  double y_change = y2 - y1;
  double x_change = x2 - x1;
  return to_string(y_change) + " / " + to_string(x_change);
        //returns a string
}
```

Notice how we need to use `to_string()` to convert our doubles into strings.

Completing the Program

Now just copy over the rest of the program that we had previously written.

```
double input1;
double input2;
double input3;
double input4;

/**
 * This function returns the slope between two sets
 * of coordinate points by calculating their coordinate
 * changes separately
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return A string expression of the slope in rise / run format
 */
string GetSlope(double x1, double y1,
                double x2, double y2) {
  double y_change = y2 - y1;
  double x_change = x2 - x1;
  return to_string(y_change) + " / " + to_string(x_change);
}

int main() {
  cout << "Enter first x coordinate: " << endl;
  cin >> input1;
  cout << "Enter first y coordinate: " << endl;
  cin >> input2;
  cout << "Enter second x coordinate: " << endl;
  cin >> input3;
  cout << "Enter second y coordinate: " << endl;
  cin >> input4;

  GetSlope(input1, input2, input3, input4);
}
```

Printing the Slope

If we try to run the program, we will not see anything printed to the screen. Why? Because there is no print statement anywhere within the code. All the program does is calculate and return values. Returning values and printing them are **not** the same thing. Therefore, we need to include a print statement if we want to actually see the output. However, we cannot just include a print statement within our function because it is a `string` function, not a `void` one. Fortunately, we can use our `main()` function to print our desired output.

```
double input1;
double input2;
double input3;
double input4;

/**
 * This function returns the slope between two sets
 * of coordinate points by calculating their coordinate
 * changes separately
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return A string expression of the slope in rise / run format
 */
string GetSlope(double x1, double y1,
                double x2, double y2) {
  double y_change = y2 - y1;
  double x_change = x2 - x1;
  return to_string(y_change) + " / " + to_string(x_change);
}

int main() {
  cout << "Enter first x coordinate: " << endl;
  cin >> input1;
  cout << "Enter first y coordinate: " << endl;
  cin >> input2;
  cout << "Enter second x coordinate: " << endl;
  cin >> input3;
  cout << "Enter second y coordinate: " << endl;
  cin >> input4;

  cout << GetSlope(input1, input2, input3, input4) << endl;
  //prints what is returned by the GetSlope() function
}
```

Lab: Helper Functions

Purpose of Helper Functions

When a function calls another function, it is using that function to help it perform a particular action. **Helper** functions provide users with more flexibility when it comes to developing functions. Additionally, helper functions help reduce code repetition because the same action only has to be written once. Let’s start by creating a few helper functions that will help us with other functions later.

```
/**
 * This function returns the difference in y values
 *
 * @param y1 A double of the first y-coordinate
 * @param y2 A double of the second y-coordinate
 * @return The difference of y1 and y2 as a double
 */
double GetRise(double y1, double y2) {
  return y2 - y1;
}

/**
 * This function returns the difference in x values
 *
 * @param x1 A double of the first x-coordinate
 * @param x2 A double of the second x-coordinate
 * @return The difference of x1 and x2 as a double
 */
double GetRun(double x1, double x2) {
  return x2 - x1;
}
```

Above, we have two functions. One that calculates the *rise* of a slope and another that calculates the *run* of a slope. These two helper functions will come in handy in out later slope calculations.

Using Helper Functions

```
/**
 * This function returns the slope in decimal form
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return A double expression of the slope
 */
double GetSlopeDecimal(double x1, double y1,
                       double x2, double y2) {
  return GetRise(y1, y2) / GetRun(x1, x2);
}

/**
 * This function returns the slop in fraction form
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return A string expression of the slope in rise / run format
 */
string GetSlopeFraction(double x1, double y1,
                        double x2, double y2) {
  return to_string(GetRise(y1, y2)) + " / " +
        to_string(GetRun(x1, x2));
  //need to convert doubles to strings!
}
```

Notice how within the two functions above `GetSlopeDecimal()` and `GetSlopeFraction()`, the previous helper functions `GetRise()` and `GetRun()` are called. Having 4 functions at our disposal provides us with a flexibility that a single function cannot offer. In this program, we can get the slope in its decimal form and its fraction form in addition to the rise and run individually. If we wanted, we can continue to build more into this program.

Complete and Run the Program

Copy over the rest of the program and then test it.

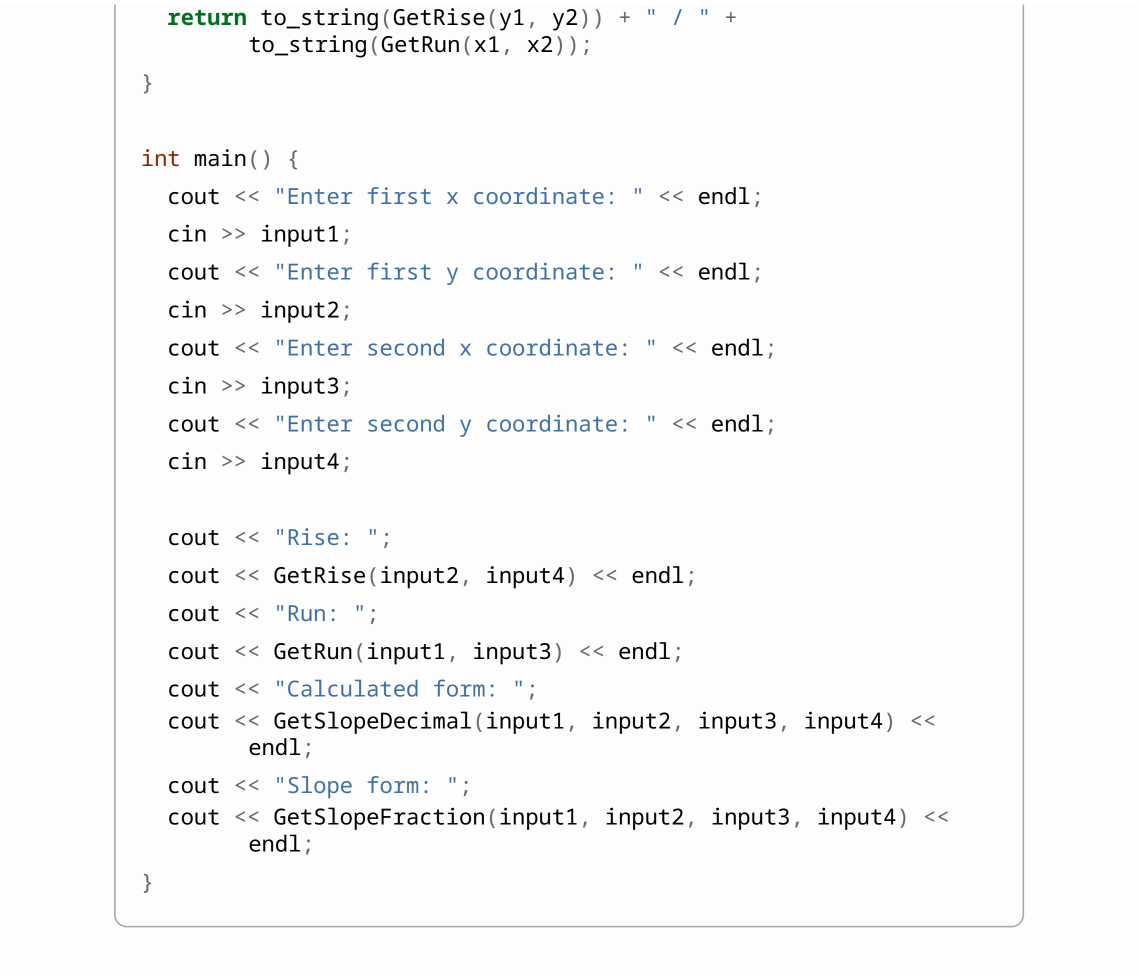
```
double input1;
double input2;
double input3;
double input4;
```

```
/**
 * This function returns the difference in y values
 *
 * @param y1 A double of the first y-coordinate
 * @param y2 A double of the second y-coordinate
 * @return The difference of y1 and y2 as a double
 */
double GetRise(double y1, double y2) {
  return y2 - y1;
}

/**
 * This function returns the difference in x values
 *
 * @param x1 A double of the first x-coordinate
 * @param x2 A double of the second x-coordinate
 * @return The difference of x1 and x2 as a double
 */
double GetRun(double x1, double x2) {
  return x2 - x1;
}

/**
 * This function returns the slope in decimal form
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return A double expression of the slope
 */
double GetSlopeDecimal(double x1, double y1,
                       double x2, double y2) {
  return GetRise(y1, y2) / GetRun(x1, x2);
}

/**
 * This function returns the slop in fraction form
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return A string expression of the slope in rise / run format
 */
string GetSlopeFraction(double x1, double y1,
                        double x2, double y2) {
```



```
  return to_string(GetRise(y1, y2)) + " / " +
        to_string(GetRun(x1, x2));
}

int main() {
  cout << "Enter first x coordinate: " << endl;
  cin >> input1;
  cout << "Enter first y coordinate: " << endl;
  cin >> input2;
  cout << "Enter second x coordinate: " << endl;
  cin >> input3;
  cout << "Enter second y coordinate: " << endl;
  cin >> input4;

  cout << "Rise: ";
  cout << GetRise(input2, input4) << endl;
  cout << "Run: ";
  cout << GetRun(input1, input3) << endl;
  cout << "Calculated form: ";
  cout << GetSlopeDecimal(input1, input2, input3, input4) <<
        endl;
  cout << "Slope form: ";
  cout << GetSlopeFraction(input1, input2, input3, input4) <<
        endl;
}
```

Lab Challenge: Greeting Machine

Create a Greeting Machine

You are going to develop a function that takes a vector of strings as a parameter, iterates through that vector and greets every element in it with `"Hello"` followed by a **newline**.

Existing Code:

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

//add code below this line



//add code above this line

int main(int argc, char** argv) {
  vector<string> names;
  for (int i = 1; i < argc; i++) {
    names.push_back(argv[i]);
  }
  SayHello(names);
}
```

Requirements

- You **should not** make any changes to the code that already exists. If you accidentally delete any existing code, you can copy and paste the entire program from above.
- You can use **any** number of additional functions you'd like but you **must** include at least the function called `SayHello()` in your code.

Compile and test your code with a few different values

create

Test with Alan & Bob

▼ **Expected Output**

```
Hello Alan
Hello Bob
```

create

Test with 1, 2 & 3

▼ **Expected Output**

```
Hello 1
Hello 2
Hello 3
```

create

Test with Codio

▼ **Expected Output**

```
Hello Codio
```