Comp 11 Lectures

Mike Shah

Tufts University

June 7, 2017

Functions

Functions 1 - An Introduction

A Code Example

Lets say I have the following code below

```
#include<iostream>

int main(){
  // Print result of 2+2 10 times
  std::cout << (2+2) << "\n";
  std::cout << (2+2) << "\n";
  std::cout << (2+2) << "\n";
  std::cout << (2+2) << "\n";
  std::cout << (2+2) << "\n";
  std::cout << (2+2) << "\n";
  std::cout << (2+2) << "\n";
  std::cout << (2+2) << "\n";
  std::cout << (2+2) << "\n";
  std::cout << (2+2) << "\n";
  return 0;
}
```

Listing 1: A function that adds two to a number

Modified with loops

The code is redundant, so let us use a loop we learned about before. Now we have saved ourselves 10 lines.

```
#include<iostream>

int main(){
  // Print result of 2+2 10 times
  for(int i =0; i < 10; ++i){
      std::cout << (2+2) << "\n";
    }

  return 0;
}
```

Listing 2: A function hat adds two to a number

Capturing a computation

- But what if we want to perform another computation besides (2+2)? We still have to create another loop. So we can actually refactor our code further.

```
#include<iostream>

int main(){
  // Print result of 2+2 10 times
  for(int i =0; i < 10; ++i){
      std::cout << (2+2) << "\n";
    }
    // Print result of 9+3 10 times
  for(int i =0; i < 10; ++i){
      std::cout << (9+3) << "\n";
    }

  return 0;
}
```

Listing 3: A function that adds two to a number

A First Function

```
1  #include <iostream>
2
3  // We have created a function
4  void addAndPrint(int a, int b){
5    for(int i =0; i < 10; ++i){
6        std::cout << (a+b) << "\n";
7      }
8  }
9
10 int main(){
11   // Now only 2 lines in our main body of code.
12   addAndPrint(2,2);
13   addAndPrint(9,3);
14
15   return 0;
16 }
```

Listing 4: A function that adds and prints

A Closer Look Function

```
1 // function has a name 'addAndPrint'
2 // function takes in two parameters a and b
3 // We can reuse this function as often as we want after we
      create it once.
4 void addAndPrint(int a, int b){
5   for(int i =0; i < 10; ++i){
6       std::cout << (a+b) << "\n";
7     }
8 }
```

Listing 5: Modular code

- We have seen functions already, **int main()** is a special function where we start a program.
- We can have more functions however, that allow us to reuse code.
- You have also seen functions before in math

Functions in math

$\cos(0) = 1$

We are familiar with what the cosine function does. It takes an input in degrees(zero), and then gives us an output (1).

- Functions are the same in programming. We take an input, and we get an output.
- Functions use the input to generate the appropriate output.

Return types and new void data type

- Functions can return one piece of data of the types we have learned about. [1]
- When functions return a result, we often store this result in a variable.
- e.g. $\cos(0) = 1$, and we can store the value 1 in a variable
  int cosineOfZero = cos(0);
- Note that when I refer to *return*, that corresponds to a special keyword in C++ telling us what value the function is returning. If there is no return, then a result is not returned from the function.

void

If a function does not return data, then we return a type of 'void' symbolizing that no result is returned.

[1] We will learn how to return multiple pieces of data and custom data types later

Function Structure

```
1 return_type - What sort of data are we working with?
2 unique_function_name - How do we use the function
3 A list of parameters ( could also be no parameters)
4 A return_type that matches the functions return type
5   ^ We do not want to lie to the computer
6
7 return_type unique_function_name(paramterType parameterName
      ,...){
8   return return_type
9 }
```

Listing 6: The skeletal structure of a function.

Another example - square

```
1  #include <iostream>
2
3  int square(int value){
4   int result = value*value;
5   return result;
6  }
7
8  int main(){
9   // square(4) is called, and the result stored in
       squareOfFour
10  // Remember, we evaluate the right-hand side of = first,
       then assign the value to the left-hand side.
11  int squareOfFour = square(4);
12  std::cout << "The square of 4 is: " << squareOfFour << "\n";
13
14  return 0;
15 }
```

Listing 7: A function with one parameter

square(int value) annotated

```
1  // Return type is an 'int'
2  // One parameter 'value' that takes 'int' values.
3  // We call the function with square. This is a new keyword we
       define.
4  int square(int value){
5   // Create a variable in local scope.
6   // Perform a multiplication.
7   int result = value*value;
8   // We return from our function a result that can be stored
       elsewhere.
9   return result;
10 }
```

Listing 8: A function that returns an integer

Computing the first 15 squares

```
1  #include <iostream>
2
3  int square(int value){
4   int result = value*value;
5   return result;
6  }
7
8  int main(){
9
10  for(int i = 0; i < 15; ++i){
11   std::cout << i << ":" << square(i) << "\n";
12  }
13
14  return 0;
15 }
```

Listing 9: First 15 squares

Removing one line in square

```
#include <iostream>

int square(int value){
//  int result = value*value ; No need for this variable here
 return (value*value);
}

int main(){

 for(int i = 0; i < 15; ++i){
  std::cout << i << ":" << square(i) << "\n";
 }

 return 0;
}
```

Listing 10: We actually do not need the result variable from the previous example but it may improve readability

Why use functions?

- Allow us to write modular code that can be reused. You can share your functions with friends!
- Functions can be tested to help increase confidence our software works as intended.
- Functions can be composed to allow more complex functions to be built
- Functions make our software cleaner and more readable. No longer will we write one large block of code!

Question about Square

- What if we want to find the square of a number like 4.3?
- Will it work if I call *square(4.3)*?

Using Square

- The problem is we have a type mismatch.
- Remember that C++ is expecting an integer, and it may truncate the result.

Function Overloading

Function Overloading

- This idea is that we can have multiple functions named the samething, but have different parameters or return types.
- All we have to do is create another function.
- What data type should it accept?
- What data type should it return?

Overloaded Square functions

```
#include <iostream>

// takes in a int, and returns an int
int square(int value){
 return (value*value);
}
// This takes in a float and returns a float
float square(float value){
 return (value*value);
}

int main(){

  // C++ will select the correct square function based on its
    // type.
  std::cout << i << ":" << square(4) << "\n";
  std::cout << i << ":" << square(4.3) << "\n";

 return 0;
}
```

General Advice

- Choose the most generic return type and value so you do not have to define multiple functions.
- There is a tool that can save us time, and this is called a template. Specifically a function template.

Function Templates

```
1 template <class myType> myType square (myType a) {
2  return a*a;
3 }
```

Listing 12: Square Template

- template <class myType> - The syntax for templates. My type in this case is a placeholder, sort of a variable.
- Anywhere myType is, you would substitute in 'int', 'float', 'double' for example.

Function Template Usage

```
1  #include <iostream>
2
3  // Templated square function
4  template <class myType> myType square (myType a) {
5   return a*a;
6  }
7
8  int main(){
9      // The brackets here are the template.
10     // we put in the template that we want.
11     int a = square<int>(5);
12     long b = square<long>(5);
13     float c = square<float>(5.5);
14     double d = square<double>(5.6435322);
15
16    return 0;
17 }
```

Listing 13: Clean code

Function Template Usage (with output)

```
1  #include <iostream>
2
3  // Templated square function
4  template <class myType> myType square (myType a) {
5   return a*a;
6  }
7
8  int main(){
9      // The brackets here are the template.
10     // we put in the template that we want.
11     std::cout << square<int>(5) << "\n";
12     std::cout << square<long>(5) << "\n";
13     std::cout << square<float>(5.5) << "\n";
14     std::cout << square<double>(5.6435322)
15
16    return 0;
17 }
```

Listing 14: Clean code

In-Class Activity

http://www.mshah.io/comp/11/activities/activity4.pdf

Activity Discussion

Review of what we learned

- (At least) Two students
- Tell me each 1 thing you learned or found interesting in lecture.

5-10 minute break

To the lab!

Lab: `http://www.mshah.io/comp/11/labs/lab3.pdf`
[2]

[2] You should have gotten an e-mail and hopefully setup an account at `https://www.eecs.tufts.edu/~accounts` prior to today. If not–no worries, we'll take care of it during lab!