Comp 11 Lectures

Mike Shah

Tufts University

June 7, 2017
Functions
Functions 1 - An Introduction
A Code Example

Let's say I have the following code below

```cpp
#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";
    return 0;
}
```

Listing 1: A function that adds two to a number
The code is redundant, so let us use a loop we learned about before. Now we have saved ourselves 10 lines.

```cpp
#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
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.

```cpp
#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
```cpp
#include <iostream>

// We have created a function
void addAndPrint(int a, int b) {
    for (int i = 0; i < 10; ++i) {
        std::cout << (a + b) << " \n";
    }
}

int main() {
    // Now only 2 lines in our main body of code.
    addAndPrint(2, 2);
    addAndPrint(9, 3);

    return 0;
}
```

Listing 4: A function that adds and prints
A Closer Look Function

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

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
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.
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

```c
int cosineOfZero = \cos(0);
```

Note that when I refer to \textit{return}, that corresponds to a special keyword in \texttt{C++} telling us what value the function is returning. If there is no return, then a result is not returned from the function.

\textbf{void}

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

¹We will learn how to return multiple pieces of data and custom data types later
return_type — What sort of data are we working with?
unique_function_name — How do we use the function
A list of parameters (could also be no parameters)
A return_type that matches the functions return type
^ We do not want to lie to the computer

code:

```java
return_type unique_function_name(parameterType parameterName, ...) {
    return return_type
}
```

Listing 6: The skeletal structure of a function.
Another example - square

```cpp
#include <iostream>

int square(int value) {
    int result = value * value;
    return result;
}

int main() {
    // square(4) is called, and the result stored in squareOfFour
    // Remember, we evaluate the right-hand side of = first, then assign the value to the left-hand side.
    int squareOfFour = square(4);
    std::cout << "The square of 4 is: " << squareOfFour << "\n";
    return 0;
}
```

Listing 7: A function with one parameter
// Return type is an 'int'
// One parameter 'value' that takes 'int' values.
// We call the function with square. This is a new keyword we define.

int square(int value) {
    // Create a variable in local scope.
    // Perform a multiplication.
    int result = value * value;
    // We return from our function a result that can be stored elsewhere.
    return result;
}

Listing 8: A function that returns an integer
#include <iostream>

int square(int value) {
    int result = value * value;
    return result;
}

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

Listing 9: First 15 squares
Removing one line in square

```cpp
#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 \textit{square}(4.3)?
The problem is we have a type mismatch.

Remember that C++ is expecting an integer, and it may truncate the result.
Function Overloading
This idea is that we can have multiple functions named the same thing, 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?
```cpp
#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;
}
```

Listing 11: Overloaded functions
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

Listing 12: Square Template

template <class myType> myType square (myType a) {
    return a*a;
}

- 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

```cpp
#include <iostream>

// Templated square function
template <class myType> myType square (myType a) {
    return a*a;
}

int main (){
    // The brackets here are the template.
    // we put in the template that we want.
    int a = square<int>(5);
    long b = square<long>(5);
    float c = square<float>(5.5);
    double d = square<double>(5.6435322);

    return 0;
}

Listing 13: Clean code
```
```cpp
#include <iostream>

// Templated square function
template <class myType> myType square (myType a) {
    return a*a;
}

int main (){
    // The brackets here are the template.
    // we put in the template that we want.
    std::cout << square<int>(5) << " \n";
    std::cout << square<long>(5) << " \n";
    std::cout << square<float>(5.5) << " \n";
    std::cout << square<double>(5.6435322)
    return 0;
}
```

Listing 14: Clean code
In-Class Activity

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!


2\(^{\text{nd}}\) 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!