# Pre-Lab for Lab 3

## Lab Overview

In this lab, we’ll learn about declaring, calling and defining functions. Functions are very useful, they let us write code that can be used over and over again. By using *parameters* with our functions, we can also use the functions to produce different results each time we call them!

## Pre-Lab Submission

Complete this Pre-Lab and submit it as a PDF by 23:59 the night before your lab section for Lab XX.

* Complete the following sections of this document before submitting:
  + Lab Prep 1: Declaring a Function
  + Lab Prep 2: Defining a Function
  + Lab Prep 3: Calling functions and using their return values

## Learning Goals

List of learning goals:

* Declaring Functions
  + With pass by value parameters
  + With pass by reference parameters
  + With return values
* Defining Functions
* Calling functions
  + Passing parameters
  + Using return values

### Goal 1: Declaring Functions

We declare a function using a *prototype*. A prototype is like a promise. It tells the compiler that we will *implement* the function later. In our class, we'll use prototypes to help make our code easier to read. They let us *declare* functions at the top of our file without having a big chunk of code up there (they're usually only one line long). We can then implement our main routine, and then *after that*, all the functions we implemented or defined.

In general, our code should look like this:



Function Declaration

Function Definition

Figure : Standard layout of a c++ file. Note the prototypes (function declarations) are after the constants and before the main routine.

##### Declaring a function

When we declare a function we must specify the following:

* The return type
* The name or identifier
* The parameters
  + These are all the inputs
  + Identifiers (the name that is used inside the function)
  + Types

#### Exemplars

##### Declaring a function that returns one value and takes one input

We’re going to declare a function that will compute the area of a circle.

**Input**

* Radius of the circle
  + Type should also be a floating point number

**Output**

* Need to return an area:
  + Numerical
  + For accuracy, a floating point number is best

**Name**

* A meaningful name will make it clear what the function does

Figure 2: Description of the function we’re going to declare

float getArea(float radius);

//we could use double, as well, for both return value

//and parameter

//note function name is an action (get)

//and uses camel case

Figure 3: Example of declaring a function to compute and return the area of a circle.

##### Declaring a function that returns two values and takes one input

Figure 4: Description of the function we’re going to declare

We’re going to declare a function that will compute both the area and perimeter of a circle.

**Input**

* Radius of the circle

**Output**:

* Need to return an area and a perimeter! But we can only return one thing. So let’s still return the area:
  + Numerical
  + For accuracy, a floating point number is best
* We also need to return the perimeter
  + Use a *pass by reference* parameter which will alter anything passed into it

**Name**

* A meaningful name will make it clear what the function does

float getAreaAndPerimeter(float radius, float &perimeter);

//the second parameter is pass by reference

//this lets us get a second value back from function

//by passing in a variable that will hold the perimeter

Figure 3: Example of declaring a function to compute and return two values.

##### Declare a function with one input and no return values

Figure 5: Example of declaring a function to output a number to the user.

void printArea(float myArea);

//we use *void* to indicate no return value

//calling it *printArea* makes it clear that value is being

//*printed out* to the user

Figure 4: Example of declaring a function to output a number to the user.

We’re going to declare a function that will output a formatted number to the user.

**Input**

* The area to be formatted

**Output**:

* None. Even though we’re *outputting* a value to the user, this is not the same as the function output.

**Name**

* A meaningful name will make it clear what the function does

##### Declare a function with no inputs and one return value

Figure 6: Example of declaring a function to get a number from the user.

We’re going to declare a function that will prompt the user for a floating point number and return it.

**Input**

* none

**Output**:

* the number the user entered

**Name**

* A meaningful name will make it clear what the function does

Figure 7: Example of declaring a function to get a number from the user.

float getRadius();

//since we need to pass in zero (0) parameters

//just put empty () for the parameter list

Lab Prep 1: Declaring a Function

Complete the given examples by filling in the boxes with your solution. In each case, carefully read the description and complete the code so that it matches the description. Refer to the exemplars provided above for hints about how to proceed.

##### Declare a function with one parameter and one return value

Declare a function that will compute and return the length of a tree

**Inputs:**

* Height
* Radius

**Output:**

* Length: Length = √ (radius2 + height2)

**Name:**

* Call it *getTreeLength*

Your answer:

##### Declare a function with no inputs and two return values

Declare a function that will get the height and radius of the tree

**Inputs:**

* none

**Output:**

* Height: gathered from user, returned as the return value
* Radius: gathered from user, and returned as *pass by reference* parameter

**Name:**

* Call it *getTreeValues*

Your answer:

##### Declare a function with no inputs and no return values.

Declare a function that will print out a description of the program to the user

**Inputs:**

* none

**Output:**

* none

**Name:**

* Call it *printOverview*

Your answer

### Goal 2: Defining Functions

Before we can call a function, we must define it. Function definitions go *after* the main routine. When defining function, we need to ensure the function definition matches the declaration’s signature:

* Return type
* Identifier
* Parameter type(s) and order

To make testing easier, we’re going to break defining functions down into two steps:

* Stubbing out (the bare essentials so we can call the function)
* Filling out the function body (actually adding the functionality)

**Stubbing out functions**:

**Stubbing out a function** means we're going to define the functions, but with only the absolute minimum amount of code needed. They won’t work, but we can call them. For functions that return void (like the one that prints out the height, length, radius, and surface area) the function only needs:

* its signature
* an empty code block
  + open a {
  + close the }

For functions that return a value (like the one that computes and returns the surface area) you need to also return some default value.

* its signature
* a code block with a return value
  + open a {
  + return a value that matches the return type
  + close the }

#### Exemplars

##### Stubbing out a function with no return value

void printArea(float myArea){

//since we return nothing, we can replace the ';'

//with an opening '{' and closing '}'

}

Figure 9: Stubbing out a function with no return value.

Figure 8: Stubbing out a function with no return value

Stub out the following function:

void printArea(float myArea);

##### Stubbing out a function with a return value

Figure 10: Stubbing out a function with a return value

Stub out the following function:

float getArea(float radius);

float getArea(float radius){

float area = 0.0;

return area;

}

Figure 11: Stubbing out a function with a return value. We have to return a value, and the value should match the return value of the function’s signature. Since we’re going to compute and return the area for this function we might as well define, initialize and return the correct variable.

Lab Prep 2: Defining Functions

Complete the given examples by filling in the box with your solution. In each case, carefully read the description and complete the code so that it matches the description. Refer to the exemplars provided above for hints about how to proceed.

##### Stub out a function that returns a value

Stub out your function that will compute and return the length of a tree. Make sure that its signature matches what you declared above.

Your answer:

##### Stub out a function that doesn’t return a value

Stub out your function that will print out a description of the program to the user. Make sure that its signature matches what you declared above.

Your answer:

### Goal 3: Calling Functions

Once a function is declared and defined (or stubbed out), we should be able to call the function. In our lab, we’ll be calling these functions from the main routine. When we call a function, we need to remember the following:

* the function’s name or identifier must match exactly
  + they’re case sensitive!
* If the function takes parameters:
  + The passed in value or variable must match the type in the declaration
* If the function returns a value:
  + Make sure to *catch* the return value in a variable!

#### Exemplars

##### Calling a function that has no return value and takes one parameter

Figure 13: Example of calling a function that takes a floating point number as a parameter

int main(){

printArea(2.4);

return 0;

}

Figure 12: Description of the function that is being called with a literal value passed in as an argument

Call the following function:

void printArea(float myArea);

pass in the literal value 2.4 for the area

##### Calling a function that has a return value and takes one parameter

Figure 12: Description of the function that is being called with a variable passed in as an argument, and the return value saved in another variable

Call the following function:

float getArea(float radius);

Declare a variable to hold the answer.

Declare a variable to hold the passed in radius

Figure 13: Example of calling a function that takes a floating point number as a parameter, and returns one. Note that the name of the variable being pass in (myRadius) **does not** have to match the name of the parameter (radius)

int main(){

float myArea = 0.0;//declare and initialize variable

float myRadius = 10.0;//the radius of the circle

myArea = getArea(myRadius);//myArea now contains the

//computed value

return 0;

}

##### Calling a function that returns two values

Figure 12: Description of the function that is being called with a variable passed in as an argument, and the return value saved in another variable

Call the following function:

float getAreaAndPerimeter(float radius, float &perimeter);

Declare a variable to hold the area.

Declare a variable to hold the perimeter

Declare a variable to hold the passed in radius

Figure 13: Example of calling a function that takes a floating point number as a parameter, and returns two. Note that the name of the variables being pass in (myRadius, myPerimeter) **do not** have to match the name of the parameters (radius, perimeter)

int main(){

float myArea = 0.0;//declare and initialize variable

float myPerimeter = 0.0;//will hold perimeter

float myRadius = 15.0;//the radius of the circle

myArea = getAreaAndPerimeter(myRadius, myPerimeter);

return 0;

}

Lab Prep 3: Calling a Function

Complete the given examples by filling in the box with your solution. In each case, carefully read the description and complete the code so that it matches the description. Refer to the exemplars provided above for hints about how to proceed.

##### Calling a function that returns a value

Call your function that will compute and return the length of a tree.

Your answer:

##### Calling a function that doesn’t return a value

Call your function that will print out a description of the program to the user.

Your answer:

##### Calling a function that returns two values

Call your *getTreeValues* function that will return the height and radius of the tree.

Have the function

* return the radius as the return value
* return the height as a *pass by reference* parameter

Your answer: