Functions

May 13, 2018

All C++ functions (except the special case of the main function) must have:

- A declaration: this is a statement of how the function is to be called.
- A definition: this is the statement(s) of the task the function performs when called

C++ functions can:

- Accept parameters, but they are not required
- Return values, but a return value is not required
- Can modify parameters, if given explicit direction to do so

'Function Syntax:
retVariableType functionName(parameter1, parameter2, . . . ,parameterN) { statement(s); }'
For example:
`void printValues() { std::cout<<"Hello World!"; }`
This function accepts no parameters and returns no value.
When it is called in a program, it will execute the statement
`std::cout<<"Hello World!";`
Function Declaration: function is declared with the syntax:
returnVariableType functionName(parameter1, parameter2, . . . ,parameterN);
Calling the function
Functions are called using their function name with any parameters specified in the declaration and definition.
For example:
`int main() { ... functionName(parameter1, parameter2, ..., parameterN); ... }`

1 Print

The function should print the message, "Functions".

```
In [1]: #include <iostream>
        void printMessage()
        {
            std::cout<<"Functions";
        }
        printMessage()
```

Functions

Print the product of two integers

```
In [2]: void printMessage(int m1, int m2)
        {
            std::cout<<m1<<" * "<<m2<<" = "<<m1 * m2;
        }
        printMessage(4, 5)
```

```
4 * 5 = 20
```

It is quite common to put functions into a header file. It makes your main program look cleaner. On this programming quiz, move the function declaration and function definition to the header file called main.cpp.

```
In [ ]: /*
        main.cpp

        #include "main.hpp"

        int main()
        {
            printProduct(4, 5);
            return 0;
        }

        main.hpp

        #include<iostream>

        void printProduct(int m1, int m2)
        {
            std::cout<<m1<<" * "<<m2<<" = "<<m1 * m2;
        }
        /*
```

2 Return

Functions with return values require the declaration and definition state the variable type of the return. The syntax is:

```
returnVariableType functionName(parameter1, parameter2, ..., parameterN) {
variarble type returnVariable;       statement(s);      return variable; }
```

For example:

```
int calculateSum(int m1, int m2) {      int sum;      sum = m1 + m2;      return
sum; }
```

We can, of course, shorten this to the more succinct:

```
int calculateSum(int m1, int m2) {      return m1 + m2; }
```

```
In [3]: // Create an add function
        float add(float m1, float m2)
        {
            return m1 + m2;
        }
        std::cout<<add(4.0, 5.0);
```

```
9
```

```
In [4]: // Create an substruction function
        float sub(float m1, float m2)
        {
            return m1 - m2;
        }
        std::cout<<sub(4.0, 5.0);
```

```
-1
```

```
In [5]: // Create an multiplication function
        float mul(float m1, float m2)
        {
            return m1 * m2;
        }
        std::cout<<mul(4.0, 5.0);
```

```
20
```

```
In [6]: // Create an division function
        float divi(float m1, float m2)
        {
            return m1 / m2;
        }
        std::cout<<divi(4.0, 5.0);
```

```
0.8
```

2.1 Pass variable by reference

The second method for effecting variables outside of their scope, is to pass by reference.

Passing by reference refers to passing the address of the variable rather than the variable. Then when we make a change in a function, we are changing the value at the address, not the variable. Once the value is changed at its address, any access to that address will retrieve the new value.

We know how to access the address of any variable. We learned it in our Pointers lesson.

So, now we can use that knowledge. In the program below, we'll pass the variable by reference.

```
In [7]: void increment_point(int &input)//Note the addition of '&'
        {
            input++; //**Note the LACK OF THE addition of '&'**
            std::cout<<"In the function call a = "<<input<<"\n";
        }
```

```
In [8]: int alpha = 1;
        std::cout<<"Before the function call alpha = "<<alpha<<"\n";
        increment_point(alpha);
        std::cout<<"After the function call alpha = "<<alpha<<"\n";
```

```
Before the function call alpha = 1
In the function call a = 2
After the function call alpha = 2
```

```
In [ ]: #include <iostream>

        void calculate(float in1, float in2, char op, float &ans)
        {
            switch(op)
            {
                case '+': ans = in1 + in2;
                          break;
                case '-': ans = in1 - in2;
                          break;
                case '*': ans = in1 * in2;
                          break;
                case '/': ans = in1 / in2;
                          break;
                default:  std::cout<<"Illegal operation\n";
            }
        }
        void printEquation(float input1,float input2, char operation, float result)
        {
            std::cout<<input1<<" "<<operation<<" "<<input2<<" = "<<result<<"\n";
        }

        char operation = '/';
        float alpha = 9.8;
        float beta = 2.3;
        float res;

        calculate(alpha, beta, operation, res);
        printEquation(alpha, beta, operation, res);
```

2.2 Arrays as parameters

C++ does not allow arrays to be passed to functions, but, as we have seen, it does allow pointers to be passed.

There are three methods for passing an array by reference to a function:

```
void functionName(variableType *arrayName)
void functionName(variableType arrayName[length of array])
void functionName(variableType arrayName[])
```

Let's look at each method in the program below.

In [1]:
```
#include <iostream>
#include <iomanip>

//Pass the array as a pointer
void arrayAsPointer(int *array, int size)
{
    std::cout<<std::setw(5);
    for(int i=0; i<size; i++)
        std::cout<<array[i]<<" ";
    std::cout<<"\n";
}
//Pass the array as a sized array
void arraySized(int array[3], int size)
{
    std::cout<<std::setw(5);
    for(int i=0; i<size; i++)
        std::cout<<array[i]<<" ";
    std::cout<<"\n";
}
//Pass the array as an unsized array
void arrayUnSized(int array[], int size)
{
    std::cout<<std::setw(5);
    for(int i=0; i<size; i++)
        std::cout<<array[i]<<" ";
    std::cout<<"\n";
}
```

In [4]:
```
const int size = 3;
int array[size] = {33,66,99};
    //We are passing a pointer or reference to the array
    //so we will not know the size of the array
    //We have to pass the size to the function as well
arrayAsPointer(array, size);
arraySized(array, size);
arrayUnSized(array, size);
```

```
33 66 99
33 66 99
33 66 99
```

2.2.1 Assignment

For this assignment, we would like you to write a function that searches an array for a value.

If the array contains the value, return the index where the key is located. If the array does not contain the value, return a -1.

```
In [7]: #include <iostream>

        int search(int *array, int size, int searchKey)
        {
            int found = -1;
            for(int i=0; i<size;i++)
            {
                if(array[i] == searchKey)
                {
                    found=i;
                }
            }
            return found;
        }

        const int size_arr = 4;
        int new_array[] = {345,75896,2,543};
        int searchKey = 543;
        std::cout<<"Found at: "<<search(new_array, size_arr, searchKey);
```

```
Found at: 3
```

3 Best Practices

When passing variables that are **not** going to be modified in the function, define the variable as a 'const' so that it cannot be changed by the function.

For example: In this function, we are passing an integer variable called input. We want the function to use it and not modify it. So we will give it a variable type that is a const int.

Function Delcaration

int doubleInput(const int input)

Function Definition

int doubleInput(const int input) { int h = input *2; return h; }

This code will not compile:

I have modified the variable input. In the declaration and definition I said this value would not be modified.

int doubleInput(const int input) { input++; int h = input *2; return h; }

Below is the error message I get when I try to compile this code.