Lab objective

The goal of this lab is to revisit functions and take inventory of what we know, also review the functions in the context of some of the topics that we have covered (e.g. structs, classes, etc):

1. Forward declaration? (why)
2. Passing parameters (by value, by reference, by address)
3. Returning values (simple, struct, object)
4. Function with default parameters
5. Template Functions
6. Private vs. public functions
7. Inline functions

Step 1) To begin with, lets create the following C++ program and place it in a file called main.cpp

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

void main()
{
    system("pause");
}
```

Step 2: Let’s create functions with default parameters. Create the following function and place at the bottom of your program.

```cpp
void showVolume(int l, int h, int w)
{
    cout << "L = " << l << endl;
    cout << "H = " << h << endl;
    cout << "W = " << w << endl;
    cout << "Volume: " << l * h * w << endl;
}
```

Now place the following forward declaration for the above function and place it in the forward declaration section of your program. (above main())

```cpp
// Forward Declaration
// Note the default parameters.
// They should be defined either in the forward declaration or
// function definition. (not both)
// Also, note that default parameters should be given from
// right to left. with no gaps!

void showVolume(int l, int h = 2, int w = 1);
```
Now Place the following code in the main() function and compile and run the code.

```cpp
//4. Function with default parameters
showVolume(5);
showVolume(5, 3);
showVolume(5, 3, 2);
```

**Screenshot 1:** Run the program and include a Screenshot of the code and output after this step.

**Remember:**
- Default arguments can be specified for “call by value” parameters only.
- During the call, if the corresponding argument is omitted, it is replaced by the default value.
- Default values should be given either in the forward declaration or the function definition (whichever appears first. Not both, some compilers complain!)
- Default parameters must be from right to left. Right most parameter must be given a default value, then the one to its left, and so on. (no gaps!)
- Watch out for ambiguity!

**Step 3:** Let’s talk about “template functions”. Sometimes it is useful to have the same function deal with multiple types. For example why should we have to write two swap() functions? Once for swapping integers, once for swapping characters, etc. Similarly, why should we write sort() function for sorting integers and another function for sorting doubles, and a third one for sorting characters? Well, the answer is we don’t need to, if we can create a generic or “Template” function with a generic or “Template” type!

**Consider the code below:**

```cpp
// Note: A template function is really not a function
// until it is instantiated. In other words,
// it is more like a class definition or struct
// definition, or forward declaration.
// It is only a definition until is
// instantiated by the user. At that time, the
// proper TYPE is passed to the definition and a
// COPY of the template function is made and added to the
// code and compiled into the program.

template<typename TYPE>
TYPE GenericSum(TYPE a, TYPE b)
{
    cout << a + b << endl;
    return(a + b);
}
```
Note the above code is precisely like a forward declaration (or struct or class definition), so it should be placed either in a “.h” file or with the rest of the forward declarations.

Place the code above between your forward declarations and the main() function.

Now place the following code in the main() program.

```cpp
//5. Template Functions
int intResult = -1;
float floatResult = -1.0;

cout << "---Make a template function for adding two integers" << endl;
intResult = GenericSum<int>(5, 10);
cout << "Sum: " << intResult << endl;

cout << "---Make a template function for adding two floats" << endl;
floatResult = GenericSum<float>(5.7, 10.9);
cout << "Sum: " << floatResult << endl;
```

Note the above code will instantiate two different version of the GenericSum() function. Once using <int> as the data TYPE, and the second time using <float> as the data TYPE.

**Screenshot 1: Run the program and include a Screen shot of the code and output after this step.**

You should see something like the following screen shot.
Complete Code. If you missed some instruction and the program is not working, below is the complete code for this lab.

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

//-----------------------------------------------
// Forward Declaration

// Note the default parameters.
// They should be defined either in the forward declaration or
// function definition. (not both)
// Also, note that default parameters should be given from
// right to left. with no gaps!

void showVolume(int l, int h = 2, int w = 1);

//-----------------------------------------------

// Template GenericSum Function will take
// Int, float, double, etc and return the sum
// of two parameters passed to it.

// Note: A template function is really not a function
// until it is instantiated. In other words,
// it is more like a class definition or struct
// definition. It is only a definition until is
// instantiated by the user. At that time, the
// proper TYPE is passed to the definition and a
// COPY of the template function is made and added to the
// code and compiled into the program.

template<typename TYPE>
TYPE GenericSum(TYPE a, TYPE b)
{
    cout << a + b << endl;
    return(a + b);
}

void main()
{

    //4. Function with default parameters
    showVolume(5);
    showVolume(5, 3);
    showVolume(5, 3, 2);

    //5. Template Functions
```
int intResult = -1;
float floatResult = -1.0;

cout << "---Make a template function for adding two integers" << endl;
intResult = GenericSum<int>(5, 10);
cout << "Sum: " << intResult << endl;

cout << "---Make a template function for adding two floats" << endl;
floatResult = GenericSum<float>(5.7, 10.9);
cout << "Sum: " << floatResult << endl;

system("pause");

//-----------------------------------------------
void showVolume(int l, int h, int w)
{
    cout << "L = " << l << endl;
    cout << "H = " << h << endl;
    cout << "W = " << w << endl;
    cout << "Volume: " << l * h * w << endl;
}