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, let's 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: To understand why “forward declaration” is necessary let's do the following.

Add the line: `func1(1, 'a');` to main. Try compiling (ctrl-F7, or using the menu BUILD, select COMPILE), but (not running) the program.

```cpp
void main()
{
    func1(1, 'a');
    system("pause");
}
```

Note that you should get a Syntax ERROR.

Screenshot 1: Include a Screen shot of the code and output after this step.
What is the reason for this? Well at this point it is because the function has not been declared yet. So,
lets declare it. Place the following code below the main() function, and recompile (not run) the code.
Do you get any syntax errors?

```cpp
void func1(int var1, char var2)
{
    cout << var1 << var2 << endl;
}
```

Now, move the func1() above the main() function and recompile. Note that the syntax error goes away!
Why is that. The reason for this is that the function is defined before (ahead of) it being used in main()
The only problem with putting all the functions before the main() function is that it is not very easy to
find main. We are used to seeing the main() function at the beginning. So, instead of moving all the
function definitions before main() we create forward declarations for each function before main(). This
way we are specifying the function signature (its name, variable that are passed to it, etc.) to the
compiler, before we use them in main() or any other subsequent functions. This tell the compiler what
to look for and how to do syntax checking, but it is only a promise that the function will be available
later. (if we don’t deliver on the promise we could still have an error!)
So, lets comment off the func1() and instead just put in a forward declaration for the function above
main, then compile the code again (Ctrl-F7) don’t try to run it yet.

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

// Forward Declaration
void func1(int var1, char var2);

void main()
{
    func1(1, 'a');
    system("pause");
}
```

Note that we get no syntax errors. The compiler is happy! Since it has been provided a forward
declaration for the func1(). However, if you now try to compile and run the above code, you will notice
that the “LINKER” will not be able to find func1() and will provide a link error.
Screenshot 2: Include a Screen shot of the code and output after this step.

Now uncomment the func1() function and recompile and run the program. Everything should be back to normal.

Step 2: Passing parameters to functions (By value) (By reference) and (by address). In most cases as a modern C++ programmer you will see pass by-value and by-reference. However, much legacy code uses pass by-address which was the approach used by most “C” programmers, and you will likely see this approach in upper level classes such as operating systems, etc.

**By value:**

```cpp
void fun1(int x)
{
    x = 10;
}
```

**By reference:**

```cpp
void fun2(int &x)
{
    x = 20;
}
```

**By address:**

```cpp
void fun3(int *x)
{
    *x = 30;
}
```
Copy the above function definitions to the bottom of your program. Then add forward declarations for each function before the main() function. Finally, add the following lines to your main program.

```cpp
//2. Passing parameters(by value, by reference, by address)
int x = 0;
fun1(x);
cout << "By Value :" << x << endl;

fun2(x);
cout << "By Reference:" << x << endl;

fun3(&x);
cout << "By Address :" << x << endl;
```

Screenshot 3: Include a screenshot of the code and output after this step.

Step 3: One of the things that makes Functions “functions” is their ability to return a value. Otherwise some languages consider them to be “procedures” which simply do something and then return. Functions, do something and then return a value. Up to just recently, we were only able to return simple data types from a function. With the introduction of STRUCT and CLASS, we now know that we are able to return those types as well. Let’s review this topic!

First, let’s place the following “struct” and “class” definitions above our forward declaration.

```cpp
struct Employee
{
    string Name;
    double HourlyWage;
    double HoursWorked;
    double GrossPay;
};
```
class Car
{
private:// Private Data
    string color;
    int year;
    string make;
    string model;

public: // Public Methods
    Car() { color = "black"; year = 1920; make = "ford"; model = "t"; }
    void SetColor(string clr) { color = clr; }
    void Print(void)
    {
        cout << "Color = " << color << endl;
        cout << "Year  = " << year << endl;
        cout << "Make  = " << make << endl;
        cout << "Model = " << model << endl;
    }
};

If you get a syntax error just by copying the code above, FIX IT. It may be complaining about the fact that you are using a “string” object without including the string header.

Now, let’s place the following forward declarations in our program:

int returnSimpleType();
Employee returnStruct();
Car returnObject();

Next let’s try to copy the code for the above functions to the bottom of our program. Note that each function below returns a different data type (int), (Employee), and (Car)!

//-------------------------------
int returnSimpleType()
{
    int x = 5;
    return(x);
}

Employee returnStruct()
{
    Employee myEmp{"mary", 10.0, 20.0, 200.00};
    return(myEmp);
}

Car returnObject()
{
    Car myCar;
    myCar.Print();
    myCar.SetColor("YELLOW");
    return(myCar);
}
Finally, let's call these functions from our main() program. To better understand what each one does:

```
//3. Returning values (simple, struct, object)
int newInt = returnSimpleType();
cout << "NewInt = " << newInt << endl;

Employee newEmp = returnStruct();
cout << "Employee Name = " << newEmp.Name << endl;

Car newCar = returnObject();
newCar.Print();
```

**Screenshot 4:** Include a screen shot of the code and output after this step. It should look something like:

![Screenshot of code and output](image-url)
Complete Code.
If you missed some instruction and the program is not working, below is the complete code for this lab.

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

struct Employee
{
    string Name;
    double HourlyWage;
    double HoursWorked;
    double GrossPay;
};

class Car
{
private:// Private Data
    string color;
    int year;
    string make;
    string model;

public: // Public Methods
    Car() { color = "black"; year = 1920; make = "ford"; model = "t"; }
    void SetColor(string clr) { color = clr; }
    void Print(void)
    {
        cout << "Color = " << color << endl;
        cout << "Year  = " << year << endl;
        cout << "Make  = " << make << endl;
        cout << "Model = " << model << endl;
    }
};

// Forward Declaration
void func1(int var1, char var2);
void fun1(int x);
void fun2(int &x);
void fun3(int *x);
int returnSimpleType();
Employee returnStruct();
Car returnObject();

void main()
```
//1. Forward declaration (why)
func1(1, 'a');

//2. Passing parameters(by value, by reference, by address)
int x = 0;
fun1(x);
cout << "By Value    : " << x << endl;

fun2(x);
cout << "By Reference: " << x << endl;

fun3(&x);
cout << "By Address  : " << x << endl;

//3. Returning values(simple, struct, object)
int newInt = returnSimpleType();
cout << "NewInt = " << newInt << endl;

Employee newEmp = returnStruct();
cout << "Employee Name = " << newEmp.Name << endl;

Car newCar = returnObject();
newCar.Print();

system("pause");

void func1(int var1, char var2)
{
    cout << var1 << var2 << endl;
}

void fun1(int x)
{
    x = 10;
}

void fun2(int &x)
{
    x = 20;
}

void fun3(int *x)
{
    *x = 30;
}

int returnSimpleType()
{
    int x = 5;
    return(x);
Employee returnStruct()
{
    Employee myEmp{"mary", 10.0, 20.0, 200.00};
    return(myEmp);
}

Car returnObject()
{
    Car myCar;
    myCar.Print();
    myCar.SetColor("YELLOW");
    return(myCar);
}

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