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 vs. protected 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 class with 3 different Print functions. One public, one private and one protected. Note that the name of the function does not matter, it is the location under which it appears that specifies its access level.

class Parent
{
    public:
        int P1, P2;
        Parent() { P1 = 5; P2 = 10; } // Constructor
        void PublicPrint() { cout << "P1 = " << P1 << "n" << "P2 = " << P2 << endl; }
    private:
        void PrivatePrint() { cout << "P1 = " << P1 << "n" << "P2 = " << P2 << endl; }
    protected:
        void ProtectedPrint() { cout << "P1 = " << P1 << "n" << "P2 = " << P2 << endl; }
};
```
Now place the following lines in your main program and run it.

```cpp
Parent myParentObj;
myParentObj.PublicPrint();
myParentObj.PrivatePrint();
myParentObj.ProtectedPrint();
```

Compile the above code and note that you will get a couple of syntax errors. That is because `myParentObj` does not have access to Private or Protected functions of the class!

Comment off the two lines with syntax error and then...

![Screenshot of the code and output](image)

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

**Step 3:** So, what goes is it to have a private or protected function? Well, the answer to that is that private functions can be called from within a class. For example, replace the following `PublicPrint()` function in the Parent class.

```cpp
//void PublicPrint() { cout << "P1 = " << P1 << "\n" << "P2 = " << P2 << endl; }
void PublicPrint() { PrivatePrint(); }
```

If you run the program again, you will note that a public function (in fact any function inside the class) can call its private. If you want, you can try the same with protected functions. (see the lines below)

```cpp
//void PublicPrint() { cout << "P1 = " << P1 << "\n" << "P2 = " << P2 << endl; }
//void PublicPrint() { PrivatePrint(); }
void PublicPrint() { ProtectedPrint(); }
```

So, you can see the clear difference between Public and the other two access modifiers.

The next question is what is the difference between Private and Protected?

To answer this we need to bring in inheritance back into the picture. Protected member are accessible to the subclasses (child classes) that inherit from the “Parent” class. However, even the child classes are not able to access the private functions of a Parent class. Let’s see this below.
Step 4: Copy the following Child class right below the Parent class and the main() function.

class Child : public Parent  //inherit from Parent class
{
    int   C1;
public:
    Child() { C1 = 15; }
    void print() { Parent::PublicPrint(); cout << "C1 = " << C1 << endl; }
};

Now place the following line in your main() program. Right before the system() line.

    Child myChild;
    myChild.print();

Now compile and run your code again. You should see

Screenshot 2: Run the program and include a Screen shot of the code and output after this step.

Note what is happening here.
1) The child is calling its own constructor to initialize its private variable “C” to 15.
2) Then the child calls its public print() function.
3) The child’s public print method then calls the Parent::PublicPrint() method, and prints “C” on its own. See line below:

    void print() { Parent::PublicPrint(); cout << "C1 = " << C1 << endl; }

Step 5: Now let’s try to replace the line above with the following:

    //void print() { Parent::PublicPrint(); cout << "C1 = " << C1 << endl; }
    void print() { Parent::PrivatePrint(); cout << "C1 = " << C1 << endl; }

Will this work? If not, why not?
Well, the answer is that `Parent::PrivatePrint();` is a private function! And therefore, it can not be accessed outside of parent (not even by one of it sub-classes!)

Now, comment off the second attempt and try a third attempt by calling the Parent’s ProtectedPrint() function.

```cpp
void print() { Parent::ProtectedPrint(); cout << "C1 = " << C1 << endl; }
```

Now compile and run your code again. You should see the same results as calling the PublicPrint() function. The child class has access to the parent’s protected members!

**Screenshot 3: Run the program and include a Screen shot of the code and output after this step.**
Complete Code.
If you missed some instruction and the program is not working, below is the complete code for this lab.

```cpp
// Understanding the use of public, private, and Protected member functions
// in inherited classes.

#include <iostream>
using namespace std;
//------------------------------------------------------
class Parent
{
public:
    int P1, P2;
    Parent() { P1 = 5; P2 = 10; }  // Constructor
    //void PublicPrint() { cout << "P1 = " << P1 << "\n" << "P2 = " << P2 << "end1; }
    void PublicPrint() { ProtectedPrint(); }
private:
    void PrivatePrint() { cout << "P1 = " << P1 << "\n" << "P2 = " << P2 << "end1; }
protected:
    void ProtectedPrint() { cout << "P1 = " << P1 << "\n" << "P2 = " << P2 << "end1; }
};
//------------------------------------------------------
class Child : public Parent   //inherit from Parent class
{
    int C1;
public:
    Child() { C1 = 15; }
    //void print() { Parent::PublicPrint(); cout << "C1 = " << C1 << "end1; }
    void print() { Parent::ProtectedPrint(); cout << "C1 = " << C1 << "end1; }
};
//------------------------------------------------------
void main()
{
    Parent  myParentObj;
    myParentObj.PublicPrint();
    //myParentObj.PrivatePrint();
    //myParentObj.ProtectedPrint();

    Child myChild;
    myChild.print();
    system("pause");
}