HW Submission requirements:
1) Put your name and ID number on the top of EACH assignment (in comments for programs)
2) Name each file the name written in blue above each problem
3) Put all the files into a folder and zip it
3) Name the zipped folder with all assignments HW#.zip (-5 points for incorrect name)

NOTES:

1. **INCLUDE A README FOR EACH PROGRAM *(-10 points) Remember a README should let the user (my TA) know how to run your program***
2. **MAKE SURE YOUR PROGRAM IS PROPERLY INDENTED. *(-10 points)***
3. **MAKE SURE ANY COMMENTS YOU INCLUDE ARE MEANINGFUL *(-10 points) Do not just put random comments on every line of code***
4. **DO NOT CODE EVERYTHING IN MAIN *(Automatic 0)***
5. **INCLUDE UML DIAGRAMS (ACTIVITY AND CLASS) FOR EACH PROGRAM**

NOTE: For the programs, part of the grading rubric will be putting functionality in the classes. For example, if you make classes but do not put functionality (or very little functionality), points will be deducted. Do not just make a class to make a class and then put all the work in main-the work should be contained in functions in the classes. Example: if you have a program where a person orders from a restaurant, you could have a Person class and a Restaurant class. The functionality of actually ordering can be a function in the Person class NOT something that happens in the main.

Problem 1 (15 points)-True/False Submit a document called Answers.doc

*Answer the following true/false questions. You must correctly state **WHY** your answer is true or false in order to receive credit.*

Code fragment:

```
class Flower{
protected:
	string location;
public:
	explicit Flower(string location)
	{
		this->location=location;
	}
	virtual bool pick_flower()=0;
};

class Rose:public Flower{
	int num_thorns;

public:
	Rose(string location, int num_thorns=5):Flower(location)
	{
		this->location=location;
```

```
class Table{

public:
	Vase* v1;
	shared_ptr<Vase> v2;
```

```
            this->num_thorns=num_thorns;
        }

        bool pick_flower(){
            return(num_thorns<5);
        }
        void operator << (int n)
        {
            num_thorns+=n;
        }

};

class Person{
        shared_ptr<Flower> f;

        shared_ptr<Flower> acquire_flower(string flo, string loc){
            if(flo=="rose")
            {
                shared_ptr<Rose> r=make_shared<Rose>(loc);
                return r;
            }

            else
            {
                shared_ptr<Edelweiss> r=make_shared<Edelweiss>(loc);
                return r;
            }
        }


public:
        void add_flower(Vase &vs, string flower_type, string loc)
        {
            vs.v.push_back(acquire_flower(flower_type, loc));
            Message::message_me(loc);
        }

        shared_ptr<Flower> get_flower()
        {
            return f;
        }

        void set_flower(shared_ptr<Flower> f)
        {
            this->f=f;
        }

};
```

1. We can say for sure from the code alone that *Edelweiss* is a derived class from Flower.

2. We can assume from the code that the *Vase* class has a vector of *Flower* objects.

3. There are 4 examples of polymorphism in the above code.

4. In the main, a line like *Rose* r1=new Rose(3);* would be a valid line of code.

5. We could change the parameter of *set_flower()* to *Flower *f* and still have the same program.

6. If we had the line *Person p1*; in the main, *p1<<3;* would be a valid line of code.

7. If we had the line *Rose* r1=new Rose("Seattle");* in the main, *r1<<3* would be a valid line of code.

8. You can see an example of all the three of the main principles of object oriented programming in the above code alone.

9. The *Rose* class inherits a constructor from the *Flower* class.

10. If the *Table* class has a third member variable *Vase * v2*, we would need to delete a total of three pointers when we are done with a *Table* object.

11. If the *Sunflower* class has protected inheritance from the *Flower* class and we include the line *using Flower::Flower;* the constructor will not be accessible outside of the *Daffodil* class (you can assume it is not abstract).

12. If we have a *Daffodil* class with protected inheritance from the *Flower* class and we don't implement the *pick_flower()* function in the *Daffodil* class, a line like *Daffodil d1("Texas");* will be possible in the main.

13. We could change the overloaded operator << to ! and the program would still function the same.

14. We could create an enum called *Flowerstuff* and change the member variable *int num_thorns* to *Flowerstuff num_thorns*.

15. We cannot say for sure that *message_me()* is a static function because it could also possibly be called by an object.

Note: for most of the coding portion you will be "putting a GUI" on top of past programs you have already created

Problem 2 (20 points)-Write a program. Submit a folder named Weight that contains the main function (in a file called weight_main.cpp) and any header files you include. Do not forget to include a makefile. AUTOMATIC 0 IF YOU CODE EVERYTHING IN weight_main.cpp or do not include a makefile)

Recreate problem 2 (weight.cpp) from HW 1 using a GUI. This means the movement throughout the program should be through the GUI ONLY (not text based).

Problem 3 (20 points)-Write a program. Submit a folder named Shape that contains the main function (in a file called shape_main.cpp) and any header files you include. Do not forget to include a makefile. AUTOMATIC 0 IF YOU CODE EVERYTHING IN shape_main.cpp or do not include a makefile)

Recreate program 3 (shapes.cpp) from HW 1 using a GUI. This means the movement throughout the program should be through the GUI ONLY (not text based).

Problem 4 (20 points)-Write a program. Submit a folder named Money that contains the main function (in a file called money_main.cpp) and any header files you include. Do not forget to include a makefile. AUTOMATIC 0 IF YOU CODE EVERYTHING IN money_main.cpp)

Recreate problem 2 (money.cpp) from HW 2 using a GUI. This means the movement throughout the program should be through the GUI ONLY (not text based).

Problem 5 (25 points)-Write a program. Submit a folder named Closet that contains the main function (in a file called closet_main.cpp) and any header files you include. Do not forget to include a makefile. AUTOMATIC 0 IF YOU CODE EVERYTHING IN closet_main.cpp)

The movement throughout the program should be through the GUI ONLY (not text based).

Marko wants to clean out and sell some of the items in his closet. He has asked you to create a program to help him do this. The program should allow him to enter the products he wants to sell (along with the price) and then continuously allow customers to look for a product and buy it. Marko should be able to check at any time the amount of money he has made from sales.

Possible TEXT-BASED Sample Run 1 (you will be implementing a GUI, not a text based program):

Hello closet cleaner! Enter your name: Marko
How many items do you want to sell? 3

Enter item and price: belt 3.50
Enter item and price: shoes 8.75
Enter item and price: pants 4.25

****
Marko's Closet!
****

Hello shopper! What item are you looking for? shoes
We have shoes for $8.75. Would you like to buy them? yes
Ok!

Hello shopper! What item are you looking for? sunglasses
Sorry! We don't have that.

Hello shopper! What item are you looking for? check balance
Hello Marko! So far you have made $8.75.

Hello shopper! What item are you looking for? exit
Bye!