Assignement 2, Due 11:59 pm, Mon Sep 11

Part I In Programming Example of section 2, Complex numbers are implemented. Extend the definition of the class complexType given in the book by overloading the following operators:

- : for subtraction

/ : for division

~ : complex conjugation

! : absolute value

(See exercises 12-15 for more detailed description). Notice that - and / are binary operators whereas ~ and ! are unary operators.

Implement - and / as non-member operators but ~ and ! as member operators. Use the pointer **this** in the implementations of ~ .

You may use the files given in the book, but make sure you modify the test program so that you thoroughly test the new operations you define. Remember to follow the guidelines stated in the first assignment.

Part II: Do programming project 4 on page 125 in 2 ways: First using inheritance as described in the book. Second using composition where the class circleType includes the class pointType. You may use the test program in the class folder FILESFORLABS\Lab2 (for the inheritance version), or you can write a test program similar to it.

Remember the general requirements that are expected in every lab assignment in this course that are explained in Lab 1 (repeated below)

Writing a reasonable test program with good user interface *is **always a default requirement for all programming assignments in this course****. Since this course is also about "program design", this is an essential point you need to keep in mind. So, this requirement is understood and will not be repeated for future assignments. Please always pay attention to this requirement. Having solved a problem correctly may not be good enough to get full points. You need to write a good test program and design a good user interface as well. A good test program and a good user interface are not fully prescribed and they may change from program to program. It is something you need to think about for each assignment. An obvious example of a good user interface would be giving the user a chance to repeat a computation before exiting the program, in fact letting the user repeat as long as they like. Also make sure that your program tests all aspects of the assignment. Another point to consider is that asking too much input from the user is not convenient.

You will be working in pairs for the programming assignments in this course. At the top of ***each program*** you submit in this course, you need to include the following information:

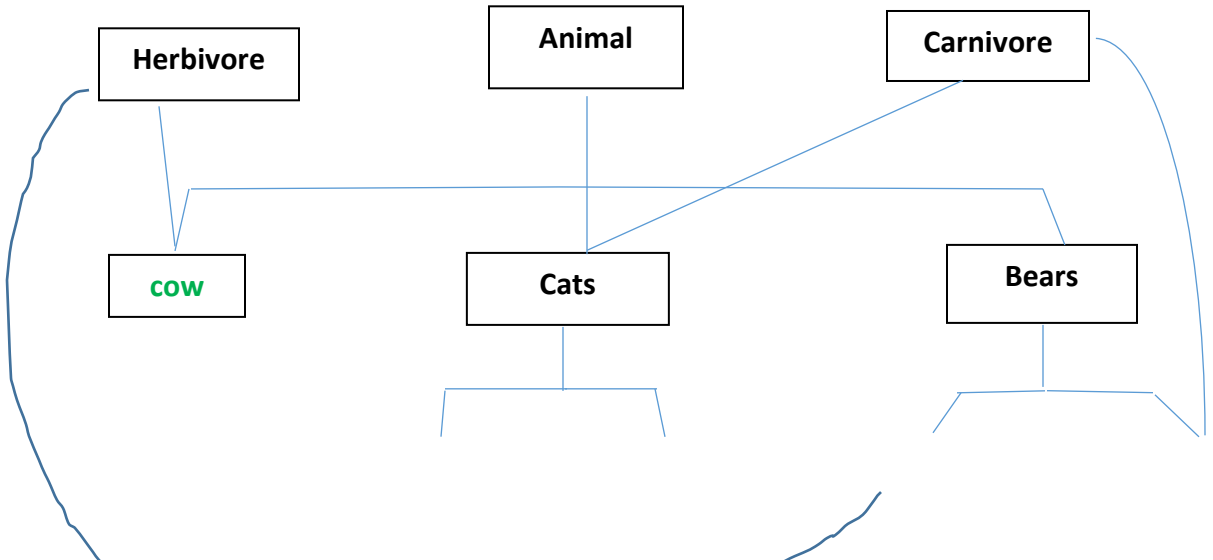
- The names of students who wrote the programs
- A description of the program
- A statement saying that each person contributed about equally to the development of the program and you followed the guidelines for Pair Programming.

*Please note that the example programs in the textbook can often be improved in terms of testing purposes and the user interface.

The Following Project is for Your Own Practice, not to be Submitted for Grade.

I highly recommend that you do them.

Part III



Define classes that satisfy the conditions below and the hierarchy in the diagram above. Don't forget that writing a good test program/user interface is part of the assignment

- A declaration of an object, for a class in the bottom of the hierarchy, should produce a message consisting of the names of all the classes that directly or indirectly are related to the object.
- A declaration of an object for a class not in the bottom of the hierarchy should prompt the compiler to issue an error message (think about how to satisfy this condition.). For example, the statement `Animal a;` should cause a compiler error message.
- Each defined object should be able to respond to the following functions:
  talk-- A parameter-free function that produces a message from the object.
  xivore-- A parameter-free function that produces one of "carnivore" or "herbivore" that reflects the kind of object in discourse.
- Each of the classes is allowed to explicitly declare at most two member functions, with at most one of the two not being a constructor.
- The body of each member function should be a single command of the form cout<<" ....";
- A string containing the name of a class may appear only within members of the class. (That is, if A is the name of a class and x is a string "A" referring to A, then x may only be included within members of A)

  Example: The program

```
#include ...
//missing definitions
int main( )
{
        leopard l;
        l.talk( );
        l.xivore( );
        return 0;
}
```

should produce an output of the following form

```
animals carnivore cats leopard
leooo
carnivore
```