Asn 4 – Due Dec 5, 2016 (3:00pm – HARD DEADLINE)

Problem 1 [12 marks]

Write a C++ "Frog" class that contains the following (private attributes):

- `age` (integer): this is the age of the frog in months. It should be initialized to 0.
- `length` (integer): this is the length of the frog in mm. It should be initialized to 3.
- `tadpole` (boolean): this is a true/false attribute. Frogs that are less than 3 months are tadpoles.

The public methods (aside from the constructor and destructors) of the class are as follows:

- `void eat():` this method adds 1mm to the length of the frog.
- `void grow():` this method adds 1 month to the age of the frog.
- `void report():` this prints out all the attributes of the frog object. For example, a call to report() may be printed as follows.

```
age: 5 months old
length: 5mm
tadpole? No
```

In your main function, write a test case for the following:

1. Create a new frog
2. Call the `report()` function
3. Call the `eat()` function three times
4. Call the `report()` function
5. Call the `grow()` function three times
6. Call the `report()` function

Asn 4 – Due Dec 5, 2016 (3:00pm – HARD DEADLINE)

Problem 2 [6 marks]

Let ∑={ 1, 2, 3, $ }. Draw an FSM diagram which outputs 1 whenever the current sum is greater than or equal to 5 (and 0 otherwise). If the sum exceeds 5, the excess amount will be carried over to the next iteration. The end of the sequence will be indicated by the symbol '$'. For example, if the sequence is:

```
123112$
```

The output will be

```
001001
```

because the sum from 1,2,3 is 6. There is 1 in excess that is carried over to the next iteration.

Another way to look at this FSM is that it outputs 1, n times where n = sum of the sequence/5. In the example given above, the sum is 10 and 1 is output n=2 times.

Problem 3 [16 marks] For each of the given alphabets, draw a FSM diagram that accepts (ends up at an accepting state) the following language descriptions:

a) **[4]** Let ∑={ a, b, c } be the alphabet: The language of all strings with an even total number of a's and an even total number of b's. Both must be even. The number of c's can be anything. Examples: Your machine should accept (end up at an accepting state): abcabc, acca, abbcbba, ccccc, , but reject: abc, cccaccc, bbabb.

b) **[4]** Let ∑={ 1, 2, 3 } be the alphabet: The language that starts with 1 and ends with odd length, OR starts with 2 and has even length, OR starts with 3 and ends with 3.

c) **[4]** Let ∑={ 0, 1 } be the alphabet: The language in which every odd position of the string is 1. For example, your machine should accept: 01010111111 (1's found in even positions are ok :)

d) **[4]** Let ∑={ 0, 1 } be the alphabet: The language that contains at least two 0's and at most one 1.