S05mid1-4. **Combinational Logic (8 points)**

Consider a **3-bit** binary number X represented in 2's complement format.

(a) Write out the truth table for the following Boolean function of X: the absolute value of X is greater than 2

(b) Write out the sum-of-products form of this Boolean function.

(c) Now, using AND, OR, and NOT gates, draw a combinational circuit of the same function. Your AND and OR gates may have any number of inputs.

2. **Regular Expressions, Deterministic Finite State Automata (6 points)**

We have the three letter alphabet { a, b, c } and the language of all strings that start and end with a.

Here are some examples of strings, and whether they are in the language:

Yes	No
a	empty string
abca	abc
abacaa	baca

a) Which one of these Regular Expressions generates all strings that start and end with a? Circle the roman numeral that goes with your answer.

i) `a* ( a | b | c )* a`

ii) `a ( a | b | c )* a`

iii) `a ( ( b | c )* a )*`

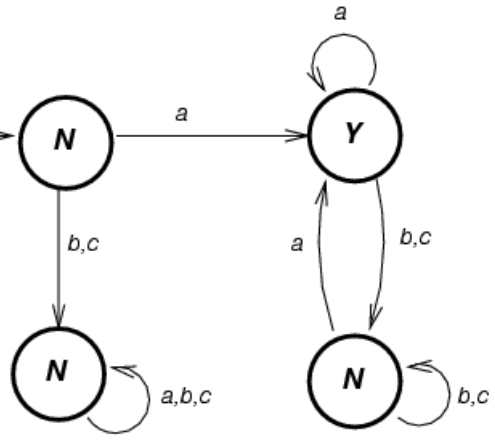
iv) `a* ( ( b | c )* a )*`

v) `a ( b | c )* a`

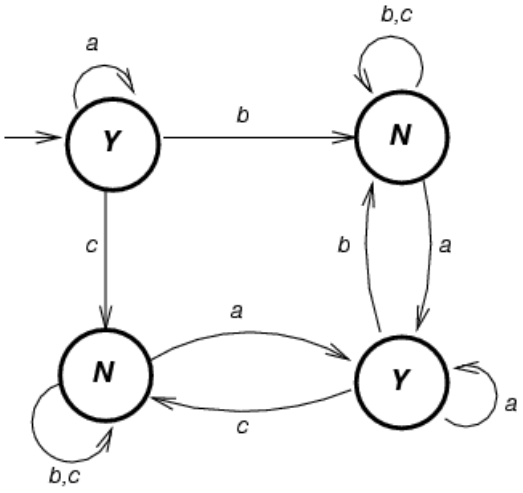
2. **RE, DFA continued**

b) Which one of the following DFA accepts all strings that start and end with a? Circle the roman numeral that goes with your answer.

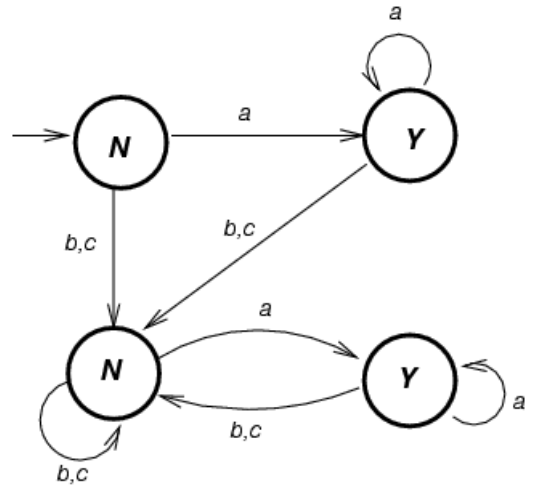
i)



ii)



iii)



3. **Linked Lists (6 points)** Assume you have access to the private Node class:

```
private class Node {
  double value;
  Node next;
}
```

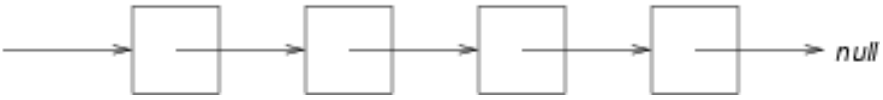
Consider the following method which operates on linked lists:

```
public boolean linky_dink (Node head) {
   Node a,b;
   a = head;
   if (a == null) return true;
   b = a.next;

   while ( b != null && b != a ) {
      b = b.next;
      if (b == null) return true;
      b = b.next;
      a = a.next;
   }

   return (b == null);
}
```

(a) What does `linky_dink` return on the following lists? Circle your answer.

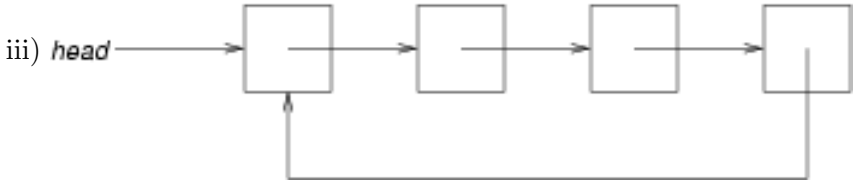
i) 

returns true returns false does not return

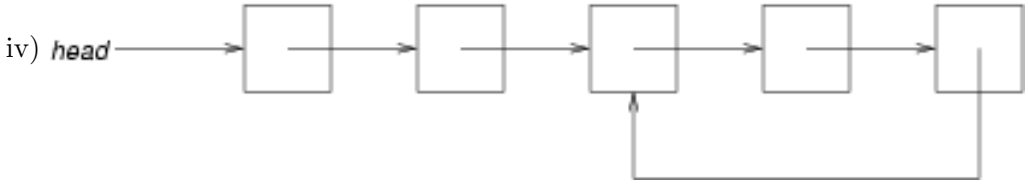
ii) 

returns true returns false does not return

3. **Linked Lists continued**

iii) 

returns true returns false does not return

iv) 

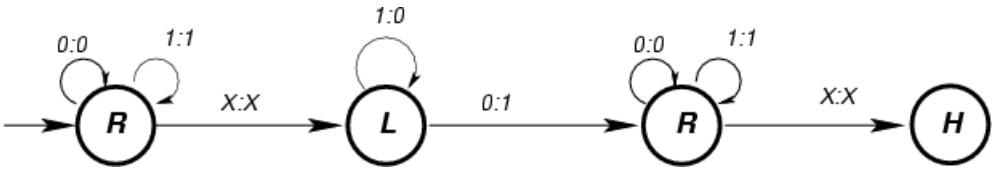
returns true returns false does not return

(b) What does `linky_dink` do?

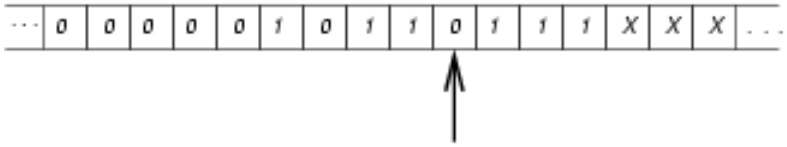
(c) If your linked list has N nodes, what is the complexity of `linky_dink`? Circle your answer.

N $N \log N$ N^2 2^N

7. **Turing Machine (4 points)**



a) The Turing Machine above starts in the leftmost state. If this Turing Machine is run on the tape below, with the tape head starting at the position marked by the arrow, what will be the contents of the tape when it halts, AND where will the head be?
Write your answer in the empty tape below.



b) What computation does this Turing Machine perform?

8. **Data Structures (3 points)** Circle your answer.

   Circle the data structure that is most appropriate choice for the described problem.

   (a) Store and retrieve student records, which have unique usernames.
   Array Linked List Binary Search Tree Symbol Table

   (b) Store all student grades and retrieve all grades higher than 90.
   Linked List Binary Search Tree Symbol Table Stack

   (c) Implement the Back button in a browser
   Queue Binary Search Tree Stack Circular Linked List

9. **True or False (6 points)** Circle your answer.

T F (a) P is the set of search problems solvable in Polynomial time by a deterministic Turing Machine.

T F (b) NP is the set of search problems not solvable in Polynomial time by a deterministic Turing Machine.

T F (c) For proper encapsulation, instance variables should always be declared public.

T F (d) Because the Halting Problem is unsolvable, it is impossible to tell if *your* TSP program for Assignment 6 has an infinite loop.

T F (e) A Universal Turing Machine can compute anything that any other Turing Machine could possibly compute.

T F (g) If P equals NP, then the Traveling Salesperson Problem can be solved in polynomial time by a deterministic Turing Machine.

T F (h) If P does not equal NP, then there is no case of the Traveling Salesperson Problem for which you can find the optimal tour in polynomial time.

T F (j) Factoring is known to be in NP but has not been proven to be NP-complete, so the discovery of a polynomial-time algorithm for factoring would mean that P equals NP.

T F (k) Factoring is known to be in NP but has not been proven to be NP-complete, so no polynomial-time algorithm for factoring is possible.

Practice Programming Exam

Part 1. Write a data type `Account.java` for maintaining bank accounts by implementing the following API.

```
public class Account
---------------------------------------
       Account(int init) // constructor, initialize account balance to init
  void deposit(int amt)  // deposit amt into account
  void withdraw(int amt) // withdraw amt from account if there is enough balance
                         // otherwise, print error message and withdraw nothing

  void transfer(int amt, Account b) // transfer amt to account b if there is
                                    // enough balance otherwise, print error
                                    // message and transfer nothing
   int getBalance()      // get current balance
```

Assume all arguments sent to the methods are ≥ 0. Make sure your methods match the given API and comments and the sample output (below).

Here is a test client you can use as the main to test your class.

```
   public static void main(String[] args) {
      Account princeton = new Account(100000000);
      Account student1 = new Account (1000);
      Account student2 = new Account (1000);
      System.out.println("Student1 account has " + student1.getBalance());
      System.out.println("Student2 account has " + student2.getBalance());
      System.out.println("Princeton account  has " + princeton.getBalance());
      student1.withdraw(100);
      student2.withdraw(500);
      student1.transfer(800, princeton);
      student2.transfer(800, princeton);
      System.out.println("Student1 account  has " + student1.getBalance());
      System.out.println("Student2 account has " + student2.getBalance());
      System.out.println("Princeton account has " + princeton.getBalance());
   }
}
```

The test client given in `main` outputs:

```
Student1 account has 1000
Student2 account has 1000
Princeton account has 100000000
Insufficient funds
Student1 account has 100
Student2 account has 500
Princeton account has 100000800
```

Part 2. Write a client program WorkStudy.java that takes the name of 2 data files as command-line inputs. The first file, studentAccounts.txt, is a list of pairs of student ID's (String) and account balances (int). For each ID-balance pair, read it in, create an Account, and put it in a symbol table. The second file, studentPayroll.txt, is a list of pairs of student ID's (String) and direct deposit payments (int). Read in each ID and payment and deposit the payment into the student's Account. Not every student necessarily gets a payment, and a student may receive more than one payment. At the end, print out each student ID and account balance.

```
% java WorkStudy studentAccounts.txt studentPayroll.txt
abe 184
brg 1180
kaw 575
mda 1050
rls 1078
```

Input file format. Both files consist of a sequence of lines containing the student ID (a String) and an amount of money (an int) separated by whitespace. To test your client, you will need to create small data files. (The files below correspond to the output in the sample run above.)

```
% more studentAccounts.txt
abe  100
brg 1000
mda 1050
rls  978
kaw  500

% more studentPayroll.txt
abe   84
brg   60
rls  100
brg  120
kaw   75
```

Something new! How can you iterate through every key in a symbol table? To travel through all the symbol table information, you will need to use a different type of `for()` loop. If your key data type is a String, and your symbol table variable name is `st` use:

```
for (String s : st) { // body of your for loop here }
```