S05mid1-4. **Combinational Circuits**

```
(a) decimal  x2 x1 x0  |X|>2
      0      0  0  0    0
      1      0  0  1    0
      2      0  1  0    0
      3      0  1  1    1
     -4      1  0  0    1
     -3      1  0  1    1
     -2      1  1  0    0
     -1      1  1  1    0
```

```
(b)  x2'x1x0 + x2x1'x0' + x2x1'x0

(c)  The circuit has three NOT gates to provide a', b' and c'.

     Three three-input AND gates have inputs matching the terms
     in the equation, and output going to a three-input OR gate
     whose output is f.
```

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

   a) The answer, iii) generates all desired strings and only desired strings.

      i) can generate a string that starts with b.

      ii) cannot generate a single a.

      iv) can generate a string that starts with b.

      v) cannot generate a single a.

   b) The answer, i) accepts all desired strings and only desired strings.

      ii) accepts the empty string.

      iii) accepts strings that start with b.

3. **Linked Lists (6 points)**

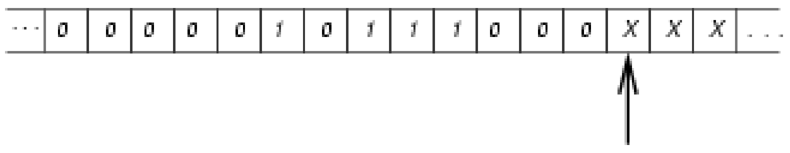
   (a)
   - i) returns true
   - ii) returns true
   - iii) returns false
   - iv) returns false

   (b) `linky_dink` returns true for a null-terminated linked list. It returns false for a circular linked list, even if the circular part is preceded by a straight path.

   (c) N

   For a null terminated linked list, b will traverse each node once before the method returns true. For a circular linked list, b which is traveling twice as quickly as a, will catch up to a in a constant number of circuits of the length N list.

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

   a) 

   b) The Turing Machine adds 1 to the binary number on the tape.

8. **Data Structures (3 points)**

   (a) Symbol Table

   (b) Binary Search Tree

   (c) Stack

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

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

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

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

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

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

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.

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.

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

```
public class Account {
   private int balance;

   //constructor, initializing account balance to init
   public Account(int init) {
      this.balance = init;
   }

   // deposit amt into account
   public void deposit(int amt) {
      this.balance = this.balance + amt;
   }

   // withdraw amt from account if there is enough balance
   // otherwise, print an error message and withdraw nothing
   public void withdraw(int amt) {
      if (amt <= this.balance)
         this.balance = this.balance - amt;
      else
         System.out.println("Insufficient funds");
   }

   // transfer amt to the account b if there is enough balance
   // otherwise, print an error message and transfer nothing
   public void transfer(int amt, Account b) {
      if (amt <= this.balance) {
         this.balance = this.balance - amt;
         b.balance = b.balance + amt;
      }
      else
         System.out.println("Insufficient funds");
   }

   // get current balance
   public int getBalance() {
      return this.balance;
   }
}
```

Part 2

Note - a simpler version of this would be to ignore Account and just treat the balances as doubles. Presumably, in the real world, Account would have more info than just the account balance.

```
public class WorkStudy {
  public static void main(String[] args) {
    In accounts = new In(args[0]);
    In payroll  = new In(args[1]);

    ST<String, Account> st = new ST<String, Account>();

    // make the symbol table
    while(!accounts.isEmpty()) {
     String id = accounts.readString();
     Account acc = new Account(accounts.readInt());
     st.put(id, acc);
    }

    // update the student accounts
    while(!payroll.isEmpty()) {
      String stud = payroll.readString();
      int pay  = payroll.readInt();
      // pull up the student's account
      Account acc = st.get(stud);
      // deposit the pay
      acc.deposit(pay);
      // put back student's account with new balance
      st.put(stud, acc);
    }

    // output all student balances - NEW FOR LOOP
    for (String s : st) {
      System.out.println(s + " " + st.get(s).getBalance());
    }

  }
}
```