COS 126 **General Computer Science** **Spring 2006**

Exam 2 Solutions

1. **Deconstructing Hello, World.**

   G C B A E F

2. **Strings, references and debugging.**

   ```
   Elis
   Tigers
   Tigers
   Big Red
   ```

   In the constructor, the local variable `name` shadows the instance variable `name`, so the instance variable is always initialized to `"Elis"`.

   The `setName()` method works as intended.

   Changing what string `x` points to will not change `team`.

   In Java, strings are immutable, so `x.replaceAll()` has no effect on `x`. To change `x`, you would use `x = x.replaceAll("Red", "Green");` .

3. **Data types.**

   (a)

   ```
   public class Point {
      private double x, y;

      public Point(double x, double y) {
         this.x = x;
         this.y = y;
      }

      public double distanceTo(Point b) {
         double dx = b.x - x;
         double dy = b.y - y;
         return Math.sqrt(dx*dx + dy*dy);
      }

      public String toString() {
         return "(" + x + ", " + y + ")";
      }
   }
   ```

(b)

```
// find furthest point from a
Point furthest = a;
while (!StdIn.isEmpty()) {
   double bx = StdIn.readDouble();
   double by = StdIn.readDouble();
   Point b = new Point(bx, by);
   if (a.distanceTo(b) > a.distanceTo(furthest))
      furthest = b;
}
```

A common optimization is to store the furthest distance encountered so far in a separate variable to avoid recomputing each time.

4. **Encapsulation.**

1. The data type didn't use the access modifier `private` with the instance variable `temp[]`, so the client can directly manipulate it.

   ```
   gauge.temp[0] = -1.0;
   ```

2. Arrays are mutable, so changing the array in the client will change the instance variable. To protect against such access, the method `setTemp()` should store a *copy* of the array.

   ```
   gauge.setTemp(t);
   t[0] = -1.0;
   ```

3. Arrays are mutable, so changing the array in the client will change the instance variable. To protect against such access, the method `setTemp()` should return a *copy* of the array.

   ```
   t = gauge.getTemp();
   t[0] = -1.0;
   ```

4. The client changes the instance variable `N`, thereby defeating the check for negative values in `isConsistent()`. This defect is created because instance variable `N` was not declared `private` and because the constant 86,400 is hardwired into `isConsistent()` (which uses this constant instead of `N`).

   ```
   gauge.N = 0;
   t[0] = -1;
   gauge.setTemp(t);
   gauge.N = 86400;
   ```

5. **Linked structures.**

   A C B B

   Note that `findB()` only works on the doubly linked list if the element you are searching for is in the list; otherwise it will go into an infinite loop if the element is strictly between two consecutive values.

6. **DFAs and regular expressions.**

   (a) All strings of a's and b's with an odd number of a's and an even number of b's.

   (b) Kleene's theorem says that DFAs and regular expressions are equivalent: given any DFA, you can construct a RE that matches the same set of strings; given any RE, you can build a DFA that recognizes the same set of strings. If you actually want to see such a RE, here is one!

   ```
   ((a + ba(aa)*b)(b(aa)*b)* (a + ba(aa)*b) + b(aa)*b)* (a + ba(aa)*b)(b(aa)*b)*
   ```

   (c) `GCG(CGG|AGG)+CTG`

7. **Turing machines.**

   (a) The downward transition is `0:F`. The upward transition is `0:T`.

   The Turing machine scans to find the rightmost bit in the second binary integer. Then, it scans to find the rightmost bit in the first binary integer. It overwrites the first bit with a T or F depending on the bitwise OR of the two bits. It then repeats the process with the next pair of bits. Finally, the Turing machine cleans up after itself, replacing the T's with 1's and F's with 0's.

   (b) N^2. Summing up each bit takes time proportional to the number of bits N.

8. **Data structures.**

   For each problem on the left, put the letter of the *best* matching data structure on the right.

   C Maintain a database of information about registered vehicles, indexed by VINs (vehicle identification numbers).

   B Simulate the behavior of cars at a traffic light.

   A Evaluate arithmetic expressions.

   D Represent the interconnections between neurons in the human brain.

   A. Stack

   B. Queue

   C. Symbol table

   D. Graph

   E. Adjacency list

9. **Intractability, Universality, and Computabaility.**

F The undecidability of the halting problem is a statement about Turing machines: it is *not* applicable to real computers.

T The Church-Turing thesis is a theory about our universe: it cannot be proven mathematically, but it can be falsified.

T The Turing machine is a universal model of computation: with a Turing machine we can solve any decision problem that can be solved with a DFA or with a Pentium M running Linux 2.6.16.

T P is the class of yes-no problems for which a polynomial-time Java program could, in principle, be written.

F NP is the class of yes-no problems for which *no* polynomial time Java program could ever be written.

T NP is the class of yes-no problems for which, in principle, you could write a Java program to check a given proposed solution in polynomial-time.

F If you discover a polynomial-time algorithm for any problem in NP, then all problems in NP are solvable in polynomial-time.

F Most Princeton computer science faculty believe that P = NP.

T Since FACTOR is a problem in NP, any instance of FACTOR can be restated as an instance of 3-SAT (3-satisfiability).

10. **Circuits.**

(a)

x_2	x_1	x_0	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(b) $f = x_2x_1'x_0 + x_2x_1x_0' + x_2x_1x_0$.

(c) Observe that $f = x_2(x_1 + x_0)$ and draw the resulting circuit.