Sample Midterm Exam

Answer all *six* questions.

Question 1 [3 points]

Give a regular definition for each of the following languages over $\Sigma = \{0, 1, \ldots, 9\}$.

1. All strings that contain no leading zeros (e.g., 00 and 012 are invalid but 0 and 12 are valid).
2. All strings that represent even numbers (strings with leading zeros are allowed).
3. All strings that contain three consecutive 1s (strings with leading zeros are allowed).

Question 2 [3 points]

Consider the following grammar.

```
E  -> E or T | T
T  -> T and NF | NF
NF -> not NF | F
F  -> ( E ) | x
```

Construct a parse tree for the sentence:

```
not not x or not x and x
```

Question 3 [4 points]

What is dynamic binding? Discuss briefly its pros and cons.

Question 4 [8 points]

Designing an electronic voting machine is a challenging task. Lets consider a very simplified voting machine. Assume that only two attributes of a candidate, namely, the name and number of votes, are of interest here and no two candidates have the same name. Implement in an OOP language of your choice a VotingMachine class that have the following methods:

1. `addCandidate(name)`: add a candidate to the list.
2. `castVote(name)`: increment the number of votes for the candidate.

Question 5 [6 points]

Define the following functions in F#. No built-ins on association lists can be used.

1. `assoc xs ys`: Create an association list from two lists `xs` and `ys`, where `xs` is a list of characters and `ys` is a list of the same length of integers. For example,

   ```
   assoc ['a';'b';'c'] [1;2;3]
   ```

   yields

   ```
   [('a', 1); ('b', 2); ('c', 3)]
   ```

2. `ass1 alist x`: Return the value associated with `x` in the association list `alist`. For example,

   ```
   ass1 [('a', 1); ('b', 2); ('c', 3)] 'b'
   ```

   yields 2.

Question 6 [6 points]

Define in F# the following functions on `BinaryTree` defined as follows:

```
type BinaryTree =
     | Void
     | Node of int*BinaryTree*BinaryTree
```

A binary tree is either `Void` (empty tree) or `Node(value,left,right)`, where `value` is the value stored in the node, `left` is the left subtree, and `right` is the right subtree.

1. `depth tree`: Return the depth of the given tree. The depth of an empty tree is 0, and the depth of a non-empty tree is the depth of the deepest subtree plus one.

2. `sum tree`: Return the sum of the nodes in the tree.