This test has 10 questions, weighted as indicated. The exam is closed book, except that you are allowed to use a one page cheatsheet (front and back). No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided.

Print your name, login ID, and precept number on this page (now), and write out and sign the Honor Code pledge before turning in this paper. Note: It is a violation of the Honor Code to discuss this midterm exam question with anyone until after everyone in the class has taken the exam. You have 50 minutes to complete the test.

“I pledge my honor that I have not violated the Honor Code during this examination.”

Signature

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/5</td>
</tr>
<tr>
<td>2</td>
<td>/7</td>
</tr>
<tr>
<td>3</td>
<td>/6</td>
</tr>
<tr>
<td>4</td>
<td>/6</td>
</tr>
<tr>
<td>5</td>
<td>/10</td>
</tr>
<tr>
<td>6</td>
<td>/6</td>
</tr>
<tr>
<td>7</td>
<td>/8</td>
</tr>
<tr>
<td>8</td>
<td>/12</td>
</tr>
<tr>
<td>9</td>
<td>/5</td>
</tr>
<tr>
<td>10</td>
<td>/5</td>
</tr>
<tr>
<td></td>
<td>/70</td>
</tr>
</tbody>
</table>
1. Boolean Algebra and Combinational Circuits (5 points).

A. (3 points) In the blank at left of each of the boolean expressions below, mark T if the expression is equivalent to \( x+y \). Otherwise, mark F.

\[ \quad \quad xy + (x+y) \]
\[ \quad \quad xy (x+y) \]
\[ \quad \quad xy + (x'y')' \]

B. (2 points) Circle the one circuit below that computes \( x+y \).

![Circuit Diagrams](image)
2. **Programming languages** (7 points). Write *YES* or *NO* in each of the seven unshaded boxes below to indicate whether or not the indicated programming language has the indicated property.

<table>
<thead>
<tr>
<th></th>
<th>can pass a function as an argument to another function</th>
<th>must declare types of variables</th>
<th>automatic garbage collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Python</td>
<td></td>
<td></td>
<td><strong>YES</strong></td>
</tr>
</tbody>
</table>
3. **REs/DFAs (6 points)**. By circling *exactly one entry in each cell* in the table below, indicate which of the strings at the top are in the language described by the RE/DFA at the left.

<table>
<thead>
<tr>
<th></th>
<th>010101</th>
<th>111110</th>
<th>000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(0 \mid 11)^*$</td>
<td>match</td>
<td>mismatch</td>
<td>mismatch</td>
</tr>
</tbody>
</table>

![DFA Diagram](image)

<table>
<thead>
<tr>
<th></th>
<th>010101</th>
<th>111110</th>
<th>000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
</tr>
<tr>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
</tbody>
</table>
4. Linked structures (6 points). Examine the following code and answer the questions below:

```java
public class Node {
    private String name;
    private Node next;

    public Node(String s, Node n) {
        name = s; next = n;
    }

    public static void main(String[] args) {
        Node a = new Node("Alice", null);
        Node b = new Node("Bob", a);
        Node e = new Node("Eve", b);

        Node cur = e;
        Node temp = null;

        while (cur != null) {
            Node next = cur.next;
            cur.next = temp;
            temp = cur;
            cur = next;
        }
    }
}
```

A. (3 points) In the diagram below, draw arrows to indicate the node references after the first three statements (which create the nodes) in main() have been executed. If a node reference is null, mark the box with an X.

```
Alice   Eve

Bob
```

B. (3 points) In the diagram below, draw arrows to indicate the node references after the while loop in main() completes execution.

```
Alice   Eve

Bob
```
5. Creating Data Types (10 points).

Recall the Point data type from the TSP assignment. Your task for this question is to develop an API for a LineSegment abstract data type by filling in the blanks below the given instance method descriptions with full method signatures (the constructor signature is provided for you). For each method you must provide an access modifier, a return type, and parameter types and names when warranted, but not the code that implements the method. We have provided the method names; your job is to fill in the blanks around them. Use the Point data type when it is logical to do so. For reference, a line segment is a part of a line that is bounded by two distinct end points, and contains every point on the line between its end points.

A. Constructor: create a line segment with two given end points.

   public LineSegment(Point x, Point y)

B. Return the string representation of this line segment.

   __________________________toString_______________________________

C. Return true if the given point is contained by this line segment; false otherwise.

   __________________________contains_______________________________

D. Return the slope of this line segment.

   __________________________slope___________________________________

E. Return N equally-spaced points that are contained by this line segment.

   __________________________sample_________________________________

F. Return the point at the intersection of this line segment and the given line segment (null if none exists).

   __________________________intersection_____________________________
6. **BSTs (6 points).** In the blanks provided write *yes* next to any of the following sequences that could be the sequence of keys examined in a search for the letter *M* in some binary search tree containing the letters of the alphabet. Write *no* next to any sequence that could not result from such a search.

A. _________ A B C Z L M.
B. _________ C U F N P K M.
C. _________ G Q H I K R M.
D. _________ Z C Y F K P M.
E. _________ E U G P K H M.

7. **Computability/Intractability (8 points).** For each of the computational problems below, indicate its difficulty by writing the most appropriate choice of *X* (not computable), *NPC* (NP-complete), *P* (tractable), or *DFA* (solvable by a DFA) in the blank at left.

A. _________ Checking whether any given program contains a virus.
B. _________ Compiling any given Java program.
C. _________ Determining whether there exists a set of inputs for which any given combinational circuit computes 1.
D. _________ Checking whether any given string is a valid e-mail address.
E. _________ Checking whether any given DNA sequence contains ACGGAC.
F. _________ Finding an optimal traveling salesperson tour through any given set of points.
G. _________ Solving Post’s correspondence problem.
H. _________ Checking whether any given string has an equal number of 0s and 1s.
8. TOY/Turing (12 points). Consider the following ten simple operations that might be performed on a 16-bit 2s complement binary integer (ignore overflow).

0. No-op (no change).
1. Flip all bits (0s to 1s and 1s to 0s).
4. AND with 1.
5. Add 1.
7. Shift left 2.
8. XOR with all 1s.

A. (6 points) Indicate which of the operations is performed on the contents of R7 by each of the three TOY code sequences below by writing a number between 0 and 9 in the box to the left of each. Assume the TOY sequences are independent; they do not come after one another.

```
7101 R1 <- 0001
2201
4772
1771 R7 <- R7 + R1
```

B. (6 points) Indicate which of the operations is performed by each of the Turing machines below by writing a number between 0 and 9 in the box to the left of each. As usual, we omit transitions to the same state that do not change the symbol. The input is a (2s complement) binary number on the tape (surrounded by #s). The head starts on the left bit. The output is the (2s complement) binary value of the final tape contents.
**TOY REFERENCE CARD**

**INSTRUCTION FORMATS**

<table>
<thead>
<tr>
<th>. . . .</th>
<th>. . . .</th>
<th>. . . .</th>
<th>. . . .</th>
<th>. . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format 1:</td>
<td>opcode</td>
<td>d</td>
<td>s</td>
<td>t</td>
</tr>
<tr>
<td>Format 2:</td>
<td>opcode</td>
<td>d</td>
<td>addr</td>
<td>(7-9, C-F)</td>
</tr>
</tbody>
</table>

**ARITHMETIC and LOGICAL operations**

1: add \[ R[d] <- R[s] + R[t] \]
2: subtract \[ R[d] <- R[s] - R[t] \]
3: and \[ R[d] <- R[s] \& R[t] \]
4: xor \[ R[d] <- R[s] ^ R[t] \]
5: shift left \[ R[d] <- R[s] << R[t] \]
6: shift right \[ R[d] <- R[s] >> R[t] \]

**TRANSFER between registers and memory**

7: load address \[ R[d] <- addr \]
8: load \[ R[d] <- \text{mem[addr]} \]
9: store \[ \text{mem[addr]} <- R[d] \]
A: load indirect \[ R[d] <- \text{mem[R[t]]} \]
B: store indirect \[ \text{mem[R[t]]} <- R[d] \]

**CONTROL**

0: halt \[ \text{halt} \]
C: branch zero \[ \text{if } (R[d] == 0) \text{ pc } <- \text{addr} \]
D: branch positive \[ \text{if } (R[d] > 0) \text{ pc } <- \text{addr} \]
E: jump register \[ \text{pc } <- R[d] \]
F: jump and link \[ R[d] <- \text{pc}; \text{pc } <- \text{addr} \]

Register 0 always reads 0.
Loads from \text{mem[FF]} come from stdin.
Stores to \text{mem[FF]} go to stdout.
9. Circuits (5 points). Identify the circuits below by writing one of the identifiers **DECODER, MAJORITY, MEMORY BIT, ODD PARITY** or **XOR** above each of them. (Inputs are at the top and outputs are on the right or at the bottom.)

![Circuit Diagram]

10. CPU (5 points). Match each term on the left with a phrase on the right by writing a letter in each blank space. Use each letter once and only once.

A. SR flip-flop  _______ holds instruction address during execution

B. IR  _______ performs arithmetic operations

C. bus  _______ set of parallel wires

D. ALU  _______ two cross-coupled NOR gates

E. PC  _______ holds instruction during execution