COS 126 General Computer Science Spring 2004

Exam 1

This test has 8 questions worth a total of 50 points. You have 120 minutes. The exam is closed book, except that you are allowed to use a one-page cheat-sheet. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. **Write out and sign the Honor Code pledge before turning in the test.**

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

Problem	Score
0	
1	
2	
3	
4	
5	
6	
7	
Total	

Name:

NetID:

Precept:

1	MF	10:00	Paul
2	MF	11:00	Diego
3	MF	1:30	Yilei
4	MF	2:30	Chi
5	M	7:30	Donna
	F	2:30	
6	MF	1:30	Randy

0. Miscellaneous. (2 points)

(a) Write your name and NetID in the space provided on the front of the exam, and circle your precept number.
(b) Write and sign the honor code on the front of the exam.

1. Loops and conditionals. (5 points)

Consider the following code segment:

```
int i=0, j=0;
for(i = 0; i < 3; i++) {
      for(j = 0; j < 5; j++){
            if((i*j)%2 == 0)
                  System.out.print("*");
            else
                  System.out.print("+");
            }
            System.out.println();
      }
```

What is the output generated by these nested loops?

2. Recursion, debugging (6 points)

Consider the following program:

```
public class Series{
     public static int func(int j){
          if (j==1) return 1;
          return 2*func(j-1)+5*func(j-2);
     }

     public static void main(String[] args) {
          int N=Integer.parseInt(args[0]);
          if (N<0) {
               System.out.println("invalid argument");
               return;
          }
          System.out.println(func(N));
     }
}
```

(a) Draw the recursion tree for **func(3)** for the original version of the program (before you debug it). You only need to draw the tree up to 3 levels, which means the height of the recursion tree should be no greater than 3.

(b) From the recursion tree in (a), do you see a problem with the program? Explain what is the problem and how to fix it.

3. Arrays. (7 points)

This is a cipher program to encode and decode an integer.

```
public class Cipher {
     public static void main(String[] args){
          int[] encode_table = {0, 7, 1, 9, 2, 8, 3, 5, 6, 4};
          int[] decode_table = {_, _, _, _, _, _, _, _, _, _};
          int[] table, list;
          int i, x, length;

          if (args[0].equals("encode"))
               table = encode_table;
          else {
               if (! args[0].equals("decode")){
                    System.out.println("Bad operation");
                    System.exit(1);
               }
               table = decode_table;
          }

          x = Integer.parseInt(args[1]);
          if (x <= 0){
               System.out.println("Integer must be positive");
               System.exit(1);
          }

          length = args[1].length();
          list = new int[length];
          for (i=0; i<length; i++){
               list[i] = x%10;
               x = x/10;
          }

          for (i=length-1; i>=0; i--)
               System.out.print(table[list[i]]);
          System.out.println();
     }
}
```

(a) What is the output of the following command?

java Cipher encode 1394

(b) This program uses an array `encode_table` to generate the cipher and another array `decode_table` to restore the original integer from the cipher. What is the correct value for decode_table?

4. Recursion: (9 points)

The following questions are about the functions 'ping' and 'pong' given below:

```
public static boolean ping(int n) {


     if (n == 0) return false;
     else return pong(n-1);
}

public static boolean pong(int n) {
     if (n == 0) return true;
     else return ping(n-1);
}
```

(a) What do the statements below print to the terminal:

```
System.out.println(ping(2));
System.out.println(pong(4));
```

(b) Explain in one sentence what is the relationship between 'n' and the return value of the 'ping' and 'pong' functions.

(c) In practice, what happens if you try to run 'ping(-1)' in your computer? (Be specific but brief)

(d) Change the 'ping' function to make it work for negative numbers. (Write your one-liner fix in the space indicated by the arrow).

(e) Going back to item (c), the original code without your modifications. What happens if you run 'pong(-1)' in a computer that is fast enough, can make as many recursive calls as it needs, and uses the 2's complement representation for integers? (Hint: an integer on this powerful machine is still represented with a finite number of bits.)

5. Number Representations (4 points)

Assume 16-bit signed two's complement numbers in this set of questions.

(a) Convert 300_{10} from decimal to hexadecimal

(b) Convert -57_{10} from decimal to hexadecimal
(note the negative sign in the decimal representation)

(c) Convert $00A3_{16}$ from hexadecimal to decimal

(d) Convert $FFE0_{16}$ from hexadecimal to decimal

6. Combinational Circuits (8 points)

The well-known Fibonnacci numbers are 1, 1, 2, 3, 5, 8, 13, ... Let the Boolean variables `a`, `b`, and `c` together represent a 3-bit non-negative binary number (that is, not in 2's-complement representation). Let `c` be the least significant bit (that is, write the number as `abc`). Let F be a Boolean variable that indicates whether the number represented by `a`, `b`, and `c` is a Fibonnacci number. (F = 1 if it is, and 0 otherwise.)

(a) Write out the truth table for the function F.

(b) Write out the sum-of-products formula for F (with no simplifications).

(c) Draw a circuit using AND, OR, and NOT gates that takes inputs `a`, `b`, and `c` and generates output F (with no simplifications). You may use AND and OR gates with more than 2 inputs if you need to.

(d) For a tiny amount of extra credit, simplify the formula and the circuit.

7. TOY programming (9 points)
(a) A "nop" instruction is a single instruction that can be inserted into an arbitrary location of a TOY program without influencing the execution of the program (assuming the programmer takes care of editing jobs such as properly changing the target of branch instructions after insertion if necessary). In other words, a nop instruction does not alter the register file state, memory state, or input/output of the machine. Several TOY instructions can be used as nop instructions. Give three such instructions. These three instructions must have different opcodes. At least one of them should not depend on R[0].

(b) Consider the following TOY program:
An integer array is stored starting at memory location 50. The length of the array is stored at memory location 4F.

```
10: 7101    ________________________
11: 884F    ________________________
12: 7250
13: 1328    R[3] <- R[2] + R[8]
14: 2331    R[3] <- R[3] - R[1]
15: 2423    R[4] <- R[2] - R[3]
16: D41E    ________________________
17: A502    ________________________
18: A603
19: B503    ________________________
1A: B602
1B: 1221    R[2] <- R[2] + R[1]
1C: 2331    R[3] <- R[3] - R[1]
1D: C015    ________________________
1E: 0000    halt

4F: 0005
50: 1141
51: 092C
52: 0653
53: 1EAD
54: 0EEF
```

(b.1) Fill in the blank lines with instruction pseudocode in the style given by the TOY instruction set reference sheet.

(b.2) Upon termination of this TOY program, give the contents of the memory at the following locations:

```
4F:
50:
51:
52:
53:
54:
```

TOY REFERENCE CARD

```
INSTRUCTION FORMATS

           | . . . . | . . . . | . . . . | . . . .|
Format 1:  | opcode  |    d    |    s    |    t   | (0-6, A-B)
Format 2:  | opcode  |    d    |       addr       | (7-9, C-F)


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] <- mem[addr]
9: store             mem[addr] <- R[d]
A: load indirect     R[d] <- mem[R[t]]
B: store indirect    mem[R[t]] <- R[d]

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


Register 0 always reads 0.
Loads from mem[FF] come from stdin.
Stores to mem[FF] go to stdout.
```