This test has 4 questions. You have 75 minutes. The exam is closed book, except that you are allowed to use a one page cheatsheet. 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.”

-------------------------------
Signature

<table>
<thead>
<tr>
<th>Problem</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>
0. Miscellaneous.

(a) Write your name and Princeton 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. Number systems.

   instead of the decimal representation of the number.

(a) Hexadecimal: $FE_{16}$, decimal unsigned: 254, decimal signed: $-2$.

(b) In Java, a byte is an 8-bit signed two’s complement number. What is the output of the following code fragment (it should be 10 lines)

```java
public static void ByteMystery(){
    byte sum = 1;
    byte pow = 1; // byte is like int but uses only 8 bits
    for (int i = 1; i <= 10; i++) {
        pow = (byte) (pow * 2);
        sum = (byte) (sum + pow);
        System.out.println( i + " " + pow + " " + sum);
    }
}
```

OUTPUT:

```
i = 1 pow = 2 sum = 3
i = 2 pow = 4 sum = 7
i = 3 pow = 8 sum = 15
i = 4 pow = 16 sum = 31
i = 5 pow = 32 sum = 63
i = 6 pow = 64 sum = 127
i = 7 pow = -128 sum = -1
i = 8 pow = 0 sum = -1
i = 9 pow = 0 sum = -1
i = 10 pow = 0 sum = -1
```
2. Java basics.

Write a complete program SecondLargest.java that reads in a sequence of real values between 0 and 1 from standard input and prints out the largest and second largest values. Assume that you have access to the library StdIn.java. Your answer will be graded for correctness and clarity.

```java
public class SecondLargest{
    public static void main(String[] args) {
        double x, lx, slx; // input, largest, second largest
        lx = -1;
        slx = -1;
        // assumes you want two largest numbers, even if repeated. // repeats will be processed // i.e., if largest value is repeated, will be both lx and slx
        while (!StdIn.isEmpty()){  
            x = StdIn.readDouble();
            if (x >= lx) {
                // shift both
                slx = lx;
                lx = x;
            } 
            else if (x > slx)
                slx = x;
        }
        System.out.print("Largest = " + lx);
        System.out.println("Second largest =" + slx);
    }
}
```
3. **Recursive graphics.**

(a) Write a program that draws a recursive pattern consisting of squares, such that each square at depth \( n - 1 \) has a depth \( n \) square centered in its upper left and lower right corners, and the side length of a depth \( n \) square is half of the side length of the squares at depth \( n - 1 \). (Hint: this pattern should look quite familiar given the H-tree assignment).

The output for \( N = 1, 2, 3, 4 \) is given on page after the next one.

Your window should be 512x512 pixels and the \( N = 1 \) square should be 256x256 pixels. The depth of recursion \( N \) is given by the user as a command line argument. Make sure your program handles the case \( N = 0 \) (nothing drawn). Also, handle the illegal input \( N < 0 \) (print an error message and draw nothing).

Below we provide the basic initialization calls to get you started.
This problem permits multiple solutions. We provide just one possibility below. The answer to part (b) is specific to this solution.

```java
public class Squares{
    public static void DrawSquareTree(int centerx, int centery, int radius){
        StdDraw.go(centerx - radius, centery - radius);
        StdDraw.penDown();
        StdDraw.go(centerx - radius, centery + radius);
        StdDraw.go(centerx + radius, centery + radius);
        StdDraw.go(centerx + radius, centery - radius);
        StdDraw.go(centerx - radius, centery - radius);
        StdDraw.penUp();
        StdDraw.pause(100);
    }

    public static void recur(int centerx, int centery, int size, int n){
        if (n == 0)
            return;
        int radius = size / 2;
        DrawSquareTree(centerx, centery, radius);
        recur(centerx - radius, centery + radius, radius, n - 1);
        recur(centerx + radius, centery - radius, radius, n - 1);
    }

    public static void main (String [] args){
        int N = Integer.parseInt(args[0]);
        int SIZE = 512;
        StdDraw.create(SIZE, SIZE);
        int radius = SIZE / 2;
        recur(radius, radius, radius, N);
    }
}
```
(b) On the example for $N = 3$, mark the order in which the squares will be drawn by your program. Write the appropriate number next to each square (so the first square that your program draws would be marked 1, second 2, etc.).
4. **TOY.** Suppose you are given the following TOY code. The values in variable RA and RB are set before this piece of code is executed.

```
10: 7101 R1 <- 01
11: 7200 R2 <- 00
12: 7C00 RC <- 00
13: 23B2 R3 <- RB - R2
14: C31A if (R3 == 0) pc <- 1A
15: 14A2 R4 <- RA + R2
16: A504 R5 <- mem[R4]
17: 1CC5 RC <- RC + R5
18: 1221 R2 <- R2 + R1
19: C013 if (0 == 0) pc <- 13
1A: BCFF mem[FF] <- RC
1B: 0000 halt
```

(a) What does this program do? In particular, what is the interpretation of registers RA and RB? What is stored in register RC when the program finishes?

The program sums RB consecutive elements in memory, with the address of the first element given by the contents of RA. The sum is stored in register RC.

(b) Write a piece of Java code that is equivalent to the TOY program (i.e., Java code that does “the same thing”). Assume arguments a and b, corresponding to registers RA and RB, are passed as parameters:

```java
public static void TOYMystery(int[] a, int b){
    int sum = 0;
    for (int i = 0; i < b; i++){
        sum = sum + a[i];
    }
    System.out.println(sum);
}
```
(c) Suppose the memory locations F0–FA are set as follows:

F0: 0001
F1: 0002
F2: 0001
F3: 0000
F4: 0003
F5: 0002
F6: 0001
F7: 0004
F8: 000D
F9: 0001
FA: 0003

and registers RA and RB as

RA <- F1
RB <- 5

What is the output of the program?

0008
This page intentionally left blank.
TOY REFERENCE CARD

INSTRUCTION FORMATS

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>Format 2:</td>
<td>opcode</td>
<td>d</td>
<td>addr</td>
</tr>
</tbody>
</table>

ARITHMETIC and LOGICAL operations

1: add \[ R[d] \leftarrow R[s] + R[t] \]
2: subtract \[ R[d] \leftarrow R[s] - R[t] \]
3: and \[ R[d] \leftarrow R[s] \& R[t] \]
4: xor \[ R[d] \leftarrow R[s] \oplus R[t] \]
5: shift left \[ R[d] \leftarrow R[s] \ll R[t] \]
6: shift right \[ R[d] \leftarrow R[s] \gg R[t] \]

TRANSFER between registers and memory

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

CONTROL

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

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