COS 126	General Computer Science	Fall 2004

Exam 1

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`

Problem	Score
0	
1	
2	
3	
4	

Total	

Name:

NetID:

Precept: 7 Thu 1:30
8 Tue 1:30

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.**

   (a) Assume we are using 8-bit integers. What is the decimal and hexadecimal representation of the binary number 11111110_2. Use both signed two's complement and unsigned interpretation for the decimal. You may use a mathematical expression instead of the decimal representation of the number.

   (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)

```
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);
    }
}
```

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.

```
public class SecondLargest {
    public static void main (String[] args) {




    }
}
```

3. **Recursive graphics.**

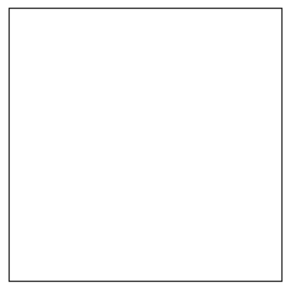
   (a) Write a program that draws a recursive pattern consisting of squares, such that each square at depth $n-1$ has a depth n square centered in its upper left and lower right corners, and the side length of a depth n square is half of the side length of the squares at depth $n-1$. (Hint: this pattern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

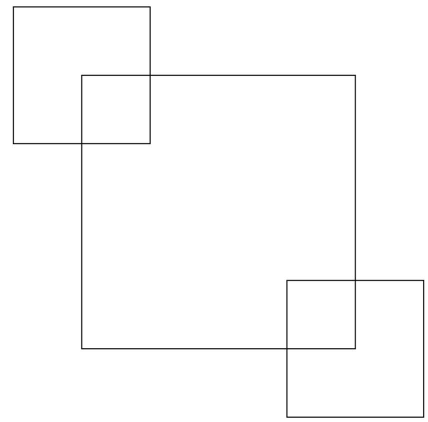
   Your window should be 512x512 pixels and the $N = 1$ square should be 256x256 pixels. The depth of recursion N is given by the user as a command line argument. Make sure your program handles the case $N = 0$ (nothing drawn). Also, handle the illegal input $N < 0$ (print an error message and draw nothing).

   Below we provide the basic initialization calls to get you started.

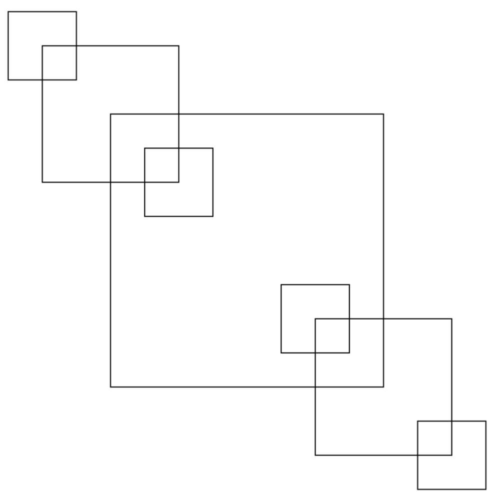
```
public class Squares {

















    public static void main (String [] args){
        int N = Integer.parseInt(args[0]);
        // Do error checking



       //Initialize the graphics
        int SIZE = 512;
        StdDraw.create(SIZE, SIZE);

       //Draw the recursive pattern



    }
}
```



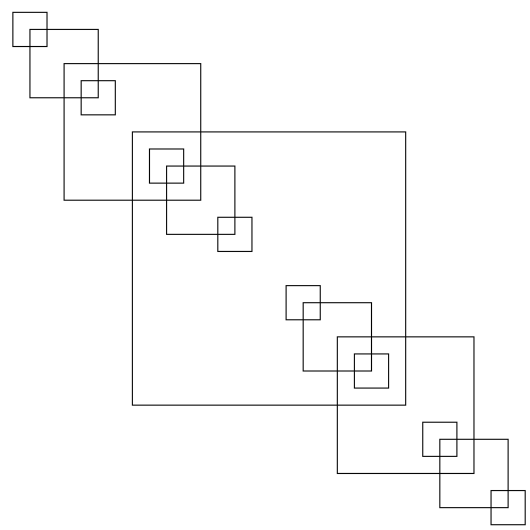
`java Squares 1`



`java Squares 2`



`java Squares 3`



`java Squares 4`

(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?

(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:

```
public static void TOYMystery(int [] a, int b){



}
```

(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?

This page intentionally left blank.

```
            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.
```