Name:
Login Name:
Preceptor Name:
Precept Number:

Computer Science 126 — Final Exam
01/21/2000 — 1:30pm - 4:30pm

This exam has 9 questions. The weight of each question is printed in the table below and next to each question. Do all of your work on these pages (use the back for scratch space), giving the answer in the space provided. Put your name on each page (now). Sign the Honor Code pledge.

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

Question		Worth	Earned
1	Linked lists and recursion	11	
2	Binary search trees	9	
3	Arrays, pointers, and complexity	10	
4	Java	8	
5	Combinational circuits and two's complement arithmetic	11	
6	FSAs and grammar	11	
7	Automata	11	
8	TOY architecture	12	
9	ADT, TOY, compilers, two's complement, and operating systems	17	
Total		100	

Name:

1. Linked Lists and Recursion [11]

Assume the following linked list definition.

```
typedef struct node *link;
struct node {
    int account;
    float amount;
    link next;
};
```

a) Write a function that returns a new list with a new node added to the beginning of a given list. The prototype is given below. The new node is to contain **`account`** and **`amount`** in it.

```
link
insertHead (link list, int account, float amount)
{









}
```

b) Complete the following recursive function which returns the sum of the **`amount`** values in the list:

```
float
totalAmount(link list)
{
    if (list == NULL)

        return _____________________________________________;

    return _____________________________________________;
}
```

Name:

c) What does the following program fragment print?

```
link
foo(link list, int a)
{
    if (list == NULL)
        return NULL;
    if (list->account <= a) {
        list->next = foo(list->next, a);
        if (list->account == a)
            return list->next;
    }
    return list;
}

/* just prints the list */
void
print(link list)
{
    link t;
    for (t = list; t != NULL; t = t->next)
        printf("[%d %f] ", t->account, t->amount);
    printf("\n");
}

int
main()
{
    link list = NULL;
    list = insertHead(list, 4000, 50.0);
    list = insertHead(list, 3000, 20.0);
    list = insertHead(list, 2000, 70.0);
    list = insertHead(list, 1000, 40.0);
    print(list);
    print(foo(list, 2500));
    print(foo(list, 3000));
}
```

Name:

2. Binary Search Trees [9]

a) Insert the following keys into an empty binary search tree in the given order. Draw the resulting tree.

50, 14, 60, 65, 33, 25, 6, 4, 12, 40

b) Print the keys in the above tree using postorder traversal.

Name:

c) For a **general binary search tree** with **more than two nodes** of **distinct keys** (not necessarily the tree on the previous page), answer the following questions.

c.1) Under what circumstances, if any, can the preorder and postorder traversals produce the same result?

c.2) Under what circumstances, if any, can the inorder and postorder traversals produce the same result?

c.3) Under what circumstances, if any, can the preorder and inorder traversals produce the same result?

Name:

3. Arrays, Pointers, and Complexity [10]

```
void
g(int a[], int asize)
{
    int *p, *q;
    int t;
    for (p = &a[asize-1]; p >= &a[0]; p--) {
        for (q = &a[0]; q < p; q++) {
            if (*q > *(q+1)) {
                t = *q;
                *q = *(q+1);
                *(q+1) = t;
            }
        }
        printf("%d ", *p);
    }
}

main()
{
    int a[8];
    int i;
    for (i = 0; i < 8; i++) a[i] = 8-i;
    g(a, 8);
}
```

a) What does this program print?

Name:

b) What is the complexity of the function **`g()`** in “big O” notation?

c.1) Is this function in the complexity class P?

c.2) Is this function in the complexity class NP?

c.3) Is this function in the complexity class NP-Complete?

Name:

4. Java [8]

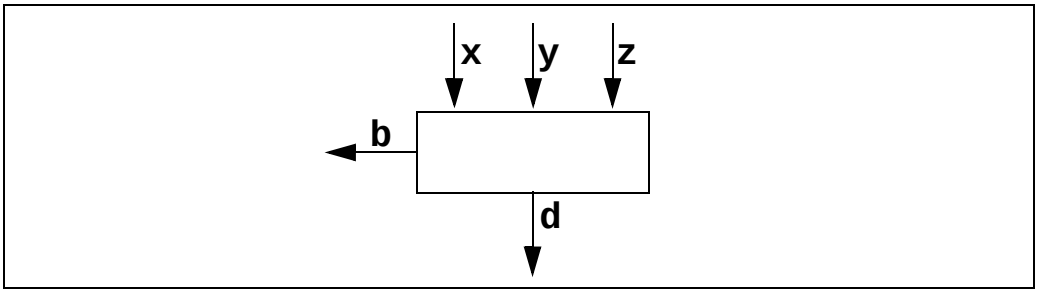
What does the following program print?

```
public class A {
    int x,y;
    public A() {x = 1; y = 2;}
    public void change(int x) {this.x = x;}
    public int foo() {return x*y;}
    public int bar() {return x;}
}

public class B extends A {
    int x = 5;
    public int foo() {return x+y;}
    public int baz() {return x;}
}

class Test {
    public static void main(String[] args) {
        B b = new B();
        B x = b;
        System.out.println(b.foo());
        b.change(10);
        System.out.println(b.foo());

        System.out.println(b.bar());
        System.out.println(b.baz());

        System.out.println(x.bar());
        System.out.println(x.baz());
    }
}
```

Name:

5. Combinational Circuits and Two's Complement Arithmetic [11]

We first construct a one-bit (bit-slice) subtracter. Its interface is as the following:



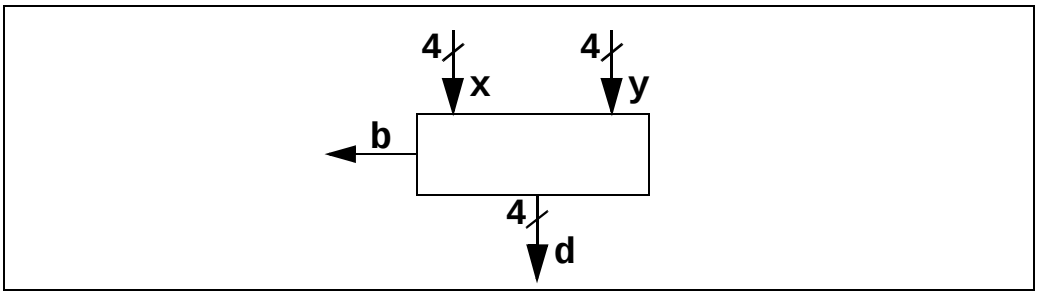
It has three inputs and two outputs: **x** is the bit to subtract from; **y** is the bit to subtract; **z** is the input borrow bit that is also subtracted from **x**; **d** is the output difference bit; and **b** is the output borrow bit.

a) Derive all the necessary truth tables.

b) Derive all the output boolean expressions.

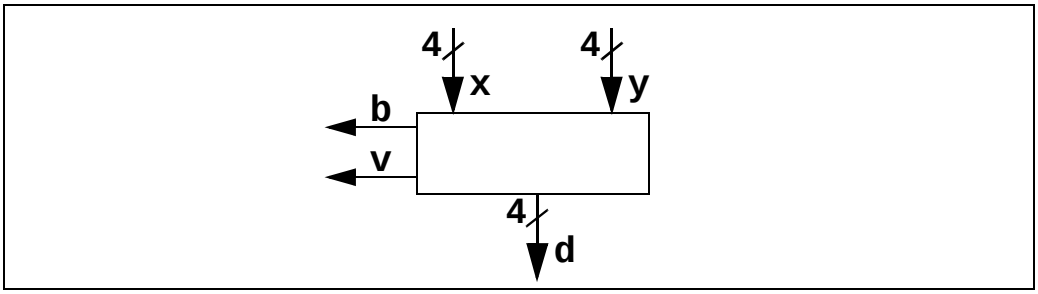
Name:

c) We now want to construct a 4-bit subtracter that has the following interface:



Recall that the notation used in this illustration means that **x** is a 4-bit wire $x_3x_2x_1x_0$. Similarly, $y=y_3y_2y_1y_0$ and $d=d_3d_2d_1d_0$. Illustrate how to use the bit-slice subtracter of the previous page to build a 4-bit subtracter.

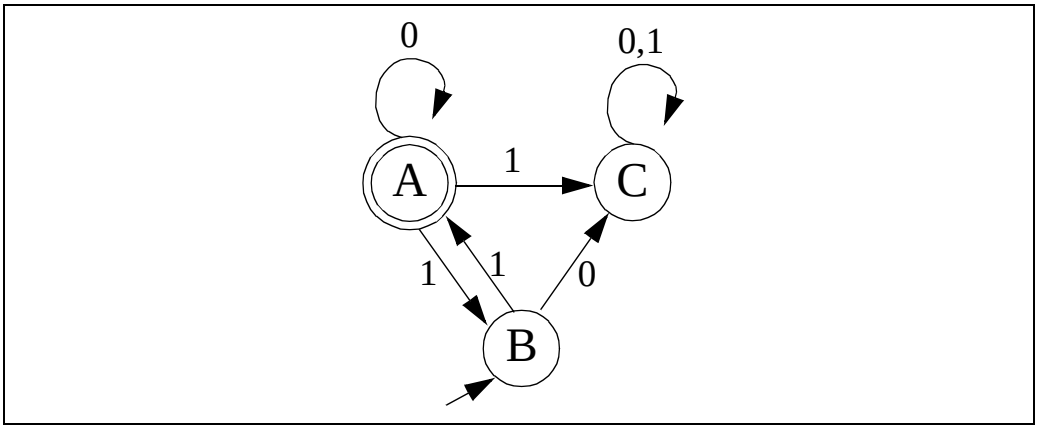
d) We now want to add an overflow output bit **v** to the subtracter (The inputs **x** and **y** are two two's complement integers):



Recall that the overflow bit is 1 iff the answer of the subtracter does not fit in 4 bits. Are the borrow bit **b** and the overflow bit **v** the same?

Name:

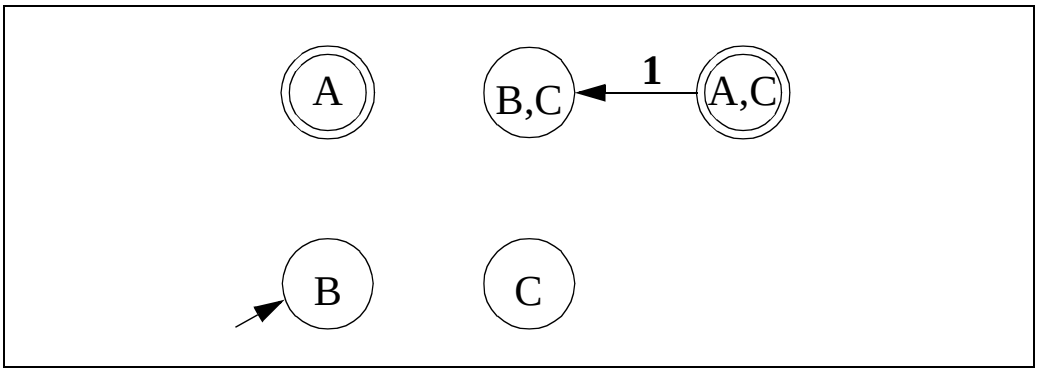
6. FSAs and Grammar [11]



In this non-deterministic FSA, **B** is the start state, and **A** is the accept state.

a) Give the Type 3 grammar that corresponds to this machine.

b) Convert the non-deterministic FSA (of part a) into a deterministic FSA by filling in the missing transition edges in the following graph.

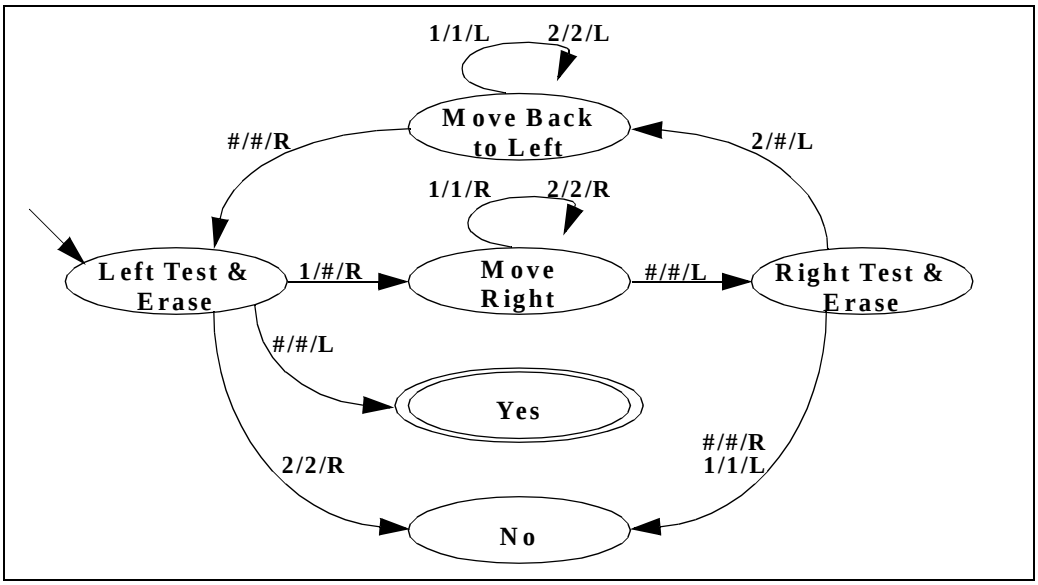


c) Which one of the following regular expressions does **not** describe this FSA?

A) 10*(110*)*
B) (10*)|(10*110*(110*)*)
C) (10*)|(10*11(0*11)*)

Name:

7. Automata [11]



In the Turing machine illustrated above, The state labeled with "Left Test & Erase" is the start state; and "Yes" is the accept state.

a.1) Does the machine accept the following string (given the initial read head position)?

#	1	2	#	#	#
	↑				

a.2) Answer the same question for the following string.

#	1	1	#	#	#
	↑				

a.3) Answer the same question for the following string.

#	1	2	1	#	#
	↑				

a.4) Answer the same question for the following string.

#	1	1	2	2	#
	↑				

b) Characterize the strings that are accepted by this Turing machine.

c) Does there exist a push down automaton that can recognize the language accepted by this Turing machine?

d) Does there exist a finite state automaton that can recognize the language accepted by this Turing machine?

Name:

8. TOY Architecture [12]

Assume that each of the following operations in the TOY datapath takes the stated amount of time (in nanoseconds):

	Operation	Time (ns)
1	**fetching an instruction from memory**	**0.8**
2	**reading from the register file**	**0.8**
3	**executing any ALU operation**	**0.8**
4	**accessing data memory**	**0.9**
5	**writing the result back into the register file**	**0.8**
6	**all other delays (those of MUXes, controls, wires, etc.)**	**0**

(Each of these operations must finish within a single clock cycle.)

a) How much time (in nanoseconds) does each of the following instructions **minimally need** to complete?

a.1) `9923  r1 <- mem[r2+r3]`

a.2) `1114  r1 <- r1+r4`

a.3) `A923  mem[r2+r3] <- r1`

b)

b.1) What is the minimum cycle time for a single cycle design?

b.2) What is the minimum cycle time for a multicycle design?

Name:

c) Given the following TOY program:

```
1A: 9923    r1 <- mem[r2+r3]
1B: 1114    r1 <- r1+r4
1C: 1114    r1 <- r1+r4
......                        } 8 times
22: 1114    r1 <- r1+r4
23: A923    mem[r2+r3] <- r1
```

c.1) Assume a single cycle design and the cycle time of b.1), how much time does this program **actually take**?
c.2) Assume a multicycle design and the cycle time of b.2), how much time does this program **actually take**?
c.3) Using the same assumptions, is one of these two designs **always** faster than the other for **any** program?

Name:

9. ADT, TOY, Compilers, Two's Complement Arithmetic, Operating Systems [17]

Consider the following arithmetic expression:

```
(1*2)-(3*(9-2))
```

a) Give the postfix representation of this expression.

b) Evaluate this postfix expression using a stack. Give the content of the stack when the stack is tallest. Graphically point out the top of the stack.

c) Give the syntax tree that our TOY compiler constructs for this expression.

Name:

d) Fill in the missing pieces of the machine code generated by our TOY compiler for this expression.

```
1A: B101        R1 <- 1
1B:
1C: 3112        R1 <- R1 * R2
1D: B203        R2 <- 3
1E: B309        R3 <- 9
1F: B402        R4 <- 2
20:
21:
22: 2112        R1 <- R1 - R2
23: 4102        print R1
24: 0000        halt
```

e) Give the content of **R1** in hexadecimal when TOY halts after executing the above program.

f) Change the arithmetic expression to the following c statement

```
a = (a*2)-(3*(9-2))
```

where **a** is an integer variable that is stored at memory location **0x00**. Assume **R0** holds the value 0. What changes should the TOY compiler make to the machine code of part d)?

g) We decided to add multiprogramming support to TOY. To do this, 1) we added timer interrupts to TOY, and 2) we wrote a small TOY Operating System that takes over control when a timer interrupt occurs. Of the two above TOY programs [program in part d) and program in part f)], can we run multiple instances of them simultaneously?

A) Program d only

B) Program f only

C) Both

D) Neither

Appendix. TOY Instruction Set

INSTRUCTION FORMATS	
Format 1: opcode, r0, r1, and r2 Format 2: opcode, r0, and 8-bit addr Indexed addressing (for format 2): if leading bit of r0 digit is 1, then addr = r1 + r2	
TRANSFER between registers and memory	
9: load A: store B: load address	r0 <- mem[addr] mem[addr] <- r0 r0 <- addr
ARITHMETIC operations	
1: add 2: subtract 3: multiply	r0 <- r1 + r2 r0 <- r1 - r2 r0 <- r1 * r2
LOGICAL operations	
C: xor D: and E: shift right F: shift left	r0 <- r1 ^ r2 r0 <- r1 & r2 r0 <- r0 >> addr r0 <- r0 << addr
CONTROL	

0: halt
4: system call
5: jump
6: jump if positive
7: jump and count
 	
8: jump and link
	halt
	print r0 on tty
	pc <- addr
	if (r0 > 0) pc <- addr
	r0--
	if (r0 != 0) pc <- addr
	r0 <- pc
	pc <- addr

Feel free to tear out this sheet.