COMP 251 L1: 1999 Fall Semester Final

Date:	Saturday, Dec 18, 1999
Time Allowed:	3 hours
Instructions:	(a) There are 8 questions in this paper.
	(b) Attempt ALL questions.
	(c) Write ALL answers in the blue book provided.
	(d) The total mark is 100.

Problem 1 [7%] Context Free Grammar.

Consider the following CFG in BNF notation:

```
<S> ::= <S><S> | (<S>) | ()
```

where <S> is the starting non-terminal.

(a) [2%] The grammar is ambiguous. Why? Support your answer by showing the ambiguous generations of one example expression.

(b) [5%] Rewrite the grammar (still in BNF) so that the **same** language may be generated unambiguously.

Problem 2 [12%] Concepts.

Explain the main difference between each of the following pairs with an example unless otherwise stated.

(a) [4%] static scope vs. dynamic scope

(b) [4%] dangling pointers vs. pointers causing memory leaks

(c) [2%] strongly typed language vs. weakly typed language (here just name an example language in each case)

(d) [2%] compilation vs. interpretation (here just name an example language in each case)

Problem 3 [10%] SML Programming I.

Write an SML function called `distribute` that takes two lists:

$$[x_1, x_2, \ldots, x_n] \text{ and } [y_1, y_2, \ldots, y_k]$$

and returns the following list:

$$[[x_1, y_1], [x_1, y_2], \ldots, [x_1, y_k], [x_2, y_1], \ldots, [x_2, y_k], \ldots, [x_n, y_1], \ldots, [x_n, y_k]]$$

For example, `distribute([1,2,3], [6,7])` returns the following list:

[[1,6], [1,7], [2,6], [2,7], [3,6], [3,7]]

You may assume that each list has **<u>at least one</u>** item.

Problem 4 [8%] SML Programming II.

Consider the following SML function definitions:

```
exception EmptyList;

fun mystery(F, nil) = raise EmptyList
  | mystery(F, [a]) = a
  | mystery(F, x::xs) = F(x, mystery(F,xs));

fun foo(x, y) = if x+y > 91 then (x+y)-91 else x+y;
```

(a) [6%] What are the types of the functions `mystery` and `foo`?

(b) [2%] What is the output of `mystery(foo, [20,16,78,90,10])`? Please show how you arrive at the result.

Problem 5 [12%] PROLOG Programming I.

(a) [6%] Implement the following **<u>three</u>** predicates about stacks in PROLOG. A stack is represented by a list with the head of the list representing the top of the stack. The stack is also polymorphic so that items on the stack can be of any datatype.

- pop(<original stack>, <popped item>, <resulting stack>)
- push(<original stack>, <pushed item>, <resulting stack>)
- num_items(<stack>, <number of items>)

(b) [4%] Write **<u>ONE SINGLE</u>** PROLOG query to push the character string, Y2K, the integer, 1999, and the atom, bug, onto the stack in that order, and then do a pop, and finally find out the number of items left on the stack. Be careful of how to represent a string, an integer, and an atom in PROLOG.

(c) [2%] Write down the PROLOG output of the query of part (b).

Problem 6 [18%] PROLOG Programming III.

Consider the following PROLOG program:

```
t1:  is_true(1).

s1:  student(a).
s2:  student(b).
s3:  student(c).

l1:  loves(a,b).
l2:  loves(a,c).

h1:  hates(X) :- loves(X,Y), !, is_true(0).
h2:  hates(_).

r1:  q1 :- student(X), hates(X).
r2:  q2 :- hates(X), student(X).
```

(a) [16%] Draw PROLOG search trees for queries q1 and q2. Show the rules and MGUs (most general unifiers) used, and number the search nodes in the order by which they would be generated by the PROLOG interpreter.

(b) [2%] What are the answers for the two queries?

Problem 7 [13%] Java Programming: Inheritance.

Consider the following hierarchy of numbers:

```
 number
   |
rational
   |
integer
```

Define a corresponding Java class inheritance hierarchy: `MyNumber` (base class), `MyRational` (a class derived from `MyNumber`), and `MyInteger` (a class derived from `MyRational` and `MyNumber`) so that the **main()** program on page 5 will run properly.

Problem 8 [20%] Java: Generic Programming and Exception Handling.

One of the strengths of object-oriented programming is that it allows generic programming. In this problem, you are going to write <u>one</u> definition of a max(x, y) method which may run on objects of the following 3 classes: BM_Int, BM_Double, and BM_Str and returns the greater object between x and y. The max() method is defined in the class Max. Parts of the program including the main() method is given. Some details are given as follows:

1. Each of the 3 supported classes, BM_Int, BM_Double, and BM_Str uses an interface called Comparable which has only one method with the following function prototype:
   public int compare(Comparable y)
2. Each of the 3 supported classes has only:
   - one private data member of type int, double, and String respectively.
   - one constructor with an input of its data member type to set its data member.
   - a public accessor method, get().
   - a public inherited method from the Comparable interface, compare(Comparable y), which return an integer greater than zero, equal to zero, or less than zero if the caller is greater than, equal to, or less than y respectively.
3. The Max class has only 2 methods, max() and main(). max() checks if the inputs are objects of the 3 permitted classes; if they are, it returns the greater of them, otherwise raises an exception of Unsupported_Exception class.
4. The exception class, Unsupported_Exception, only has one constructor with no argument.
5. max() of the Max class handles only exceptions of Unsupported_Exception class by a complete try block which should consists of 3 different phrases: It raises the exception when inputs to max() are of wrong classes, handles it with a printout of "Gotcha!" and then passes it to the caller, and wraps up by another printout of "Good luck!".

You are required to

(a) [18%] Complete other parts of the program as indicated by the word "**ACTION**" or "...".

(b) [2%] Write the output of the program. You may ignore outputs generated not directly from the given codes.

Java Program of Problem 7

```
class NumberTest
{
    public static void main(String[] args)
    {
        MyNumber[] x = new MyNumber[3];
        x[0] = new MyRational(10,11);          // this gets a rational 10/11
        x[1] = new MyInteger(20);              // this gets an integer 20
        x[2] = new MyRational(10,7);           // this gets a rational 10/7

        for (int i=0; i < 3;i++)               // this will print out the numbers
            x[i].print();                      // one by one as: 10/11 20 10/7
    }
}
```

Incomplete Java Program of Problem 8

```
import BM_Char;                // Import the class definition of BM_Char class

class Unsupported_Exception ...
// ACTION: Complete this class definition

... Comparable
// ACTION: Complete this interface definition

class BM_Int ...
// ACTION: Complete this class definition

class BM_Double ...
// ACTION: Complete this class definition

class BM_Str ...
// ACTION: Complete this class definition
```

```
class Max
{
    static Comparable max(Comparable x, Comparable y)
        /* ACTION: Fill in the correct syntax here to complete exception handling */
        {
            /* the syntax: <object> instanceof <class>
             * checks if <object> belongs to <class> */
            if (x instanceof BM_Int || x instanceof BM_Double || x instanceof BM_Str)
            {
                return (x.compare(y) > 0) ? x : y;
            }
            else
            {
                try
                {
                    /* ACTION: Raise the exception, Unsupported_Exception,
                     * with the message: "Unsupported Class" */
                }
                /* ACTION: Fill in the correct syntax for the 2nd phrase
                 * of a try block */
                {
                    System.out.println("Gotcha!");
                    // ACTION: Pass the exception to Java's runtime system
                }
                /* ACTION: Fill in the correct syntax for the 3rd phrase
                 * of a try block */
                {
                    System.out.println("Good luck!");
                }
            }
        }

    public static void main(String[] args)
        throws Unsupported_Exception
        {
            System.out.println( ((BM_Double) max(new BM_Double(3.4),
                                                  new BM_Double(5.6))).get());
            System.out.println( ((BM_Int) max(new BM_Int(8),
                                               new BM_Int(6))).get());
            System.out.println( ((BM_Char) max(new BM_Char('r'),
                                                new BM_Char('m'))).get());
            System.out.println( ((BM_Str) max(new BM_Str("good"),
                                               new BM_Str("luck"))).get());
        }
}
```