UNIVERSITY OF TORONTO
Faculty of Arts and Science

April/May Examinations 1998

CSC148S

Duration — 3 hours

Aids allowed: None.

- **Make sure your examination booklet has 19 pages (including this one).**
- Write your answers in the spaces provided. Do not feel that you must use all of the space provided. If you run out of space on any question, use the back of a page, and draw an arrow to point this out.
- You will be rewarded for concise, well-thought-out answers, rather than long rambling ones.
- Write legibly. Unreadable answers will be given 0.

- Throughout the exam, take tree height to mean the number of *nodes* on the longest past from the root to any leaf.
- When asked to write code, comments are not necessary.

Family Name: ____________________ Given Names: ____________________

Student #: ____________________ Lecturer: ____________________

1.	________	/ 8
2.	________	/ 14
3.	________	/ 7
4.	________	/ 18
5.	________	/ 13
6.	________	/ 22
7.	________	/ 8
8.	________	/ 15
Total	________	/ 105

Question 1

[8 marks in total]

Consider the following class:

```
class SNode {
        public static int i = 0;
        public String s;
        public SNode link;

        public SNode (String initialValue) {
                s = initialValue;
                link = null;
                i++;
        }
}
```

a. [1 mark]
The `SNode()` method is a constructor. When is it called? Circle the one best answer.

1. Whenever we construct an instance of `SNode`.
2. **Correct answer:** Whenever we construct an instance of `SNode` using a `String` parameter.
3. Whenever we declare a variable of type `SNode`.
4. Whenever we declare a variable of type `SNode` using a `String` parameter.
5. Never; the default constructor is called instead.
6. None of the above. This code has a syntax error because method `SNode` lacks a return value.

b. [1 mark]
If we define a class without a constructor, Java provides a default constructor. Which variables in the class does the default constructor affect, and what does it do to them?

```
It affects the instance variables.
It sets them all to zero, null, false, as appropriate for their type.
```

c. [1 mark]
Variable `i` is "static". What does this mean?

```
It is a "class variable".
So there is one copy of this variable shared by all instances of the class (rather
than one copy per instance).
```

d. [1 mark]
Write a single line of code that shows how to access static variable `i` from outside of class `SNode`.

```
// We refer to i through the class name (SNode), rather than
// through the name of an instance:
SNode.i = 13;
```

e. [4 marks]
The program below runs, but does not work. It fails to create a linked list with two nodes: one containing the string “Anna”, and one containing “Jameel”. Fix the bug by writing on the code below.

YOU MAY USE THIS SPACE FOR ROUGH WORK:

```
(The problem is due to the fact that parameters are passed by value in Java.  So when
we pass myList to insert, its value is sent to front (i.e., a reference to the first
node in the list, or null, is sent to front).  We can change front all we want inside
method insert, but this has no effect on myList!  The solution is to return the revised
front of the list from the method, and store it in myList.)
```

ONLY THIS PART WILL BE MARKED:

```
class Tester {

     // Old version: public static void insert (String name, SNode front) {
     public static SNode insert (String name, SNode front) {

          SNode newOne = new SNode(name);

          newOne.link = front;

          // Old version: front = newOne;
          return newOne;

     }

     public static void main (String[] args) {

          SNode myList = null;

          // Old version: insert ("Anna", myList);
          myList = insert ("Anna", myList);

          // Old version: insert ("Jameel", myList);
          myList = insert ("Jameel", myList);

          // At this point, we want to have a linked list with 2 nodes.
     }
}
```

Question 2

[14 marks in total]

Suppose we are designing software for the University of Toronto, and that we already have written two classes: `FacultyMember` and `Student`. The details of these two classes are irrelevant to this question.

Now consider a class called `Course`, for keeping track of information about a University course. The data members of the class will be:

- the professor (an instance of class `FacultyMember`)
- the course name (a instance of class `String`)
- a set of students taking the course (an instance of class `StudentSet`)

Class `StudentSet` is for keeping track of a set of `Student`s. Although one would probably not want duplicate entries, class `StudentSet` will not try to prevent them. This class is to be implemented using a singly-linked list, and an appropriate `Node` class has already been written.

An outline for all of this code appears below and on the next page. On the next page, add code to do the things listed below. A large part of the marks will be for good design.

- Declare any instance variables needed for class `StudentSet`.
- Declare instance variables for the 3 data members (described above) in class `Course`.
- Add any methods to class `StudentSet` that will be needed by class `Course`'s `enrolStudent()` method. Do not write a complete `StudentSet` class — just what is needed by `enrolStudent()`.
- Complete method `enrolStudent()` in class `Course`. It should add a given `Student` to the set of `Student`s enrolled in the course. Do not bother checking to avoid duplicates in the set.
- Assume that all of these classes have been fully implemented. Now completely define a class called `LimitedEnrolmentCourse`. `LimitedEnrolmentCourse` works just like class `Course`, except that it has a limit on the number of students, and it will not enrol any more students if the class is full. Include in your code an instance variable for this limit, and a constructor that sets it.

Don not change or add to this code:

```
class FacultyMember {
        // Details omitted.
}
```

```
class Student {
        // Details omitted.
}
```

```
class Node {
        public Student s;
        public Node next;

        public Node (Student stud) {
                s = stud;
                next = null;
        }
}
```

Add your code on the next page.

```
// A set of students.  Allows duplicates.
class StudentSet {
         // These instance variables are part of the solution.
         private Node front;
         private int count;

         // This method is part of the solution.
         public void addStudent(Student s) {

            Node temp = new Node(s);
            temp.next = front;
            front = temp;
         }

        // Assume that a constructor and other needed methods are included here.
}

class Course {
         // These instance variables are part of the solution.
         private FacultyMember professor;
         private String courseName;
         private StudentSet classMembers;

         // Add s to the set of Students enrolled in this course.
         public void enrolStudent (Student s) {

            // The body of this method is part of the solution.
            classMembers.addStudent(s);
         }

        // Assume that a constructor and other needed methods are included here.
}

class LimitedEnrolmentCourse extends Course {
         // The entire body of this class is part of the solution.

         private int maxClass;
         private int classSize;

         public LimitedEnrolmentCourse (int limit) {
            maxClass = limit;
            classSize = 0;
         }

         public void enrolStudent (Student s) {

               if (classSize < maxClass) {
                     super.enrolStudent(s);
                     classSize++;
               }
         }
 }
```

Question 3

[7 marks in total]

Consider the silly classes show below. If we run the `SillyTester` class shown on the right, (*i.e.* if we run its `main` method), it runs without crashing. Show what the output would be.

```
class Snorfle {
    private int foo;
    public Snorfle () {
            foo = 5;
    }
    public void setFoo (int x) {
            foo = x;
    }
    public int getFoo () {
        return foo;
    }
    public void messFoo (int a) {
            foo = foo * a;
    }
}

class Badoom extends Snorfle {
    public void messFoo (int a) {
            setFoo(getFoo() + a);
    }
}

class Pirk extends Badoom {
    public void messFoo (int a) {
            super.messFoo (100);
    }
}
```

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

        Snorfle s1 = new Snorfle();
        s1.messFoo(2);
        System.out.println("A: " + s1.getFoo());

        Badoom b = new Badoom();
        b.messFoo(2);
        System.out.println("B: " + b.getFoo());

        Pirk p = new Pirk();
        p.messFoo(2);
        System.out.println("C: " + p.getFoo());

        Snorfle s2 = s1;
        s2.setFoo(13);
        s1.messFoo(3);
        System.out.println("D: " + s1.getFoo()+ " "
                                 + s2.getFoo());

        s2 = new Badoom();
        s2.messFoo(50);
        System.out.println("E: " + s1.getFoo()+ " "
                                 + s2.getFoo());
    }
}
```

YOU MAY USE THIS SPACE FOR ROUGH WORK:

OUTPUT (ONLY THIS WILL BE MARKED):

```
A: 10
B: 7
C: 105
D: 39 39
E: 39 55
```

Question 4

[18 marks in total]

Consider a new abstract data type called StringSet, which has the following data and operations.

DATA:

- a set of zero or more strings, each of length at least 1 and made up of the 26 lower-case letters from a to z.

OPERATIONS:

- exists: given a string, returns true if it is in the set, and false otherwise.
- insert: adds a given string to the set; if already there, does nothing.
- delete: removes a given string from the set; if the string is not in the set, does nothing.
- size: returns the number of strings in the set.

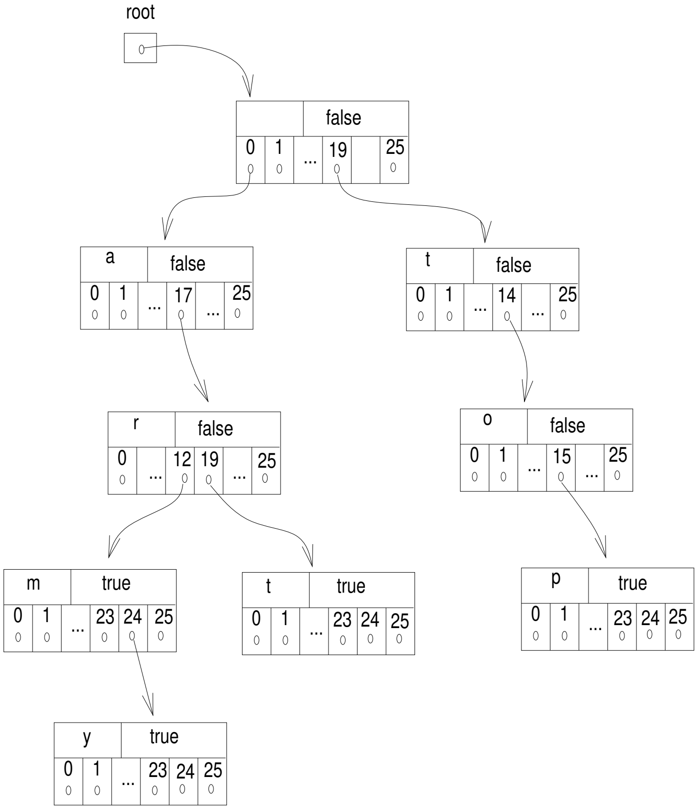
a. [5 marks]
Write the interface for a `StringSet` class that implements this ADT; that is, write the declarations for the public methods provided by the class. Do not write the method bodies.

```
    class StringSet {

         // A correct solution might omit this constructor.
         public StringSet () {

         public boolean exists (String s) {

         public void insert (String s) {

         public void delete (String s) {

         public int size () {
}
```

Now suppose that class `StringSet` is implemented by a special kind of tree. Each node stores a single letter plus one reference for each letter of the alphabet. If you follow any path in the tree starting from the root, the sequence of letters encountered defines a string. Each node also stores a boolean called `present` to indicate whether or not the string from the root to that node is considered to be in the set.

The root node in such a tree is special: it doesn't store a character, and so doesn't contribute a character to any of the strings stored in the tree.

The figure below shows such a tree, into which the strings "arm", "army", "art", and "top" have been inserted. The strings "a", "ar", "t", and "to" are not considered to be in this tree because their `present` members are set to false.



b. [3 marks]
Suppose we start with an empty `StringSet` and insert the following strings:

"robert", "edward", "ed", "geraldine", "andrew", "roberta".

How many nodes will the tree have?	29
How many leaves?	4
What will the height of the tree be?	10

Now, assume that the following class, for creating nodes in the tree, has already been defined:

```
class TreeNode{
     public char letter;
     public boolean present;
     public TreeNode[] child;
}
```

c. [2 marks]
Write declarations for the instance variables needed to implement the StringSet ADT with a tree as described above.

```
private TreeNode root;
```

d. [6 marks]
Complete the code begun below for the `exists` method. It uses a helper method called `charPos()`. The call `charPos(aChar)` returns an integer between 0 and 25 corresponding to the letter stored in `aChar`, where `a` corresponds to 0, `b` to 1, `c` to 2, etc. Recall that method `charAt(i)` in class `String` returns the character at position `i` in the string.

Note that this is meant to be a non-recursive implementation of the method. Do not try to complete it by making it recursive.

```
public boolean exists (String s) {
     int i = 0;

     TreeNode p =     root        ;

     while ( i < s.length() &&
             p.child[ charPos(s.charAt(i)) ] != null ) {


        p = p.child[ charPos(s.charAt(i)) ];
        i++;


     }

     if (      p.present  &&  i == s.length()     ) {

        return true;


     }
     else {

        return false;


     }
}
```

e. [2 marks]
Using big-Oh notation give the worst-case time complexity of `exists(s)` as a function of n, the length of the string `s`. NOTE: You may be able to do this part even if you have not done any of the previous ones.

O(n)

Question 5

[13 marks in total]

Recall the `SetOfInts` class from Assignment 2. Below is the interface for that class. Note that a new method called `contains()` has been included.

```
// Store a set of integers.
public class SetOfInts {

    // Ensure there I contain exactly one occurrence of integer i.
    public void insert(int i) {

    // Return true if I contain i, false otherwise.
    public boolean contains(int i) {

    // Return true if I am empty, false otherwise.
    public boolean isEmpty() {

    // Return one integer that I contain. (It doesn't matter which one.)
    // Precondition: I am not empty.
    public int getOne() {

    // Ensure that I do not contain integer i.
    public void delete(int i) {

    // Return a string representation of me.
    public String toString() {

}
```

In this question, you will add a new operation to the `SetofInts` class: set intersection. An `intersect()` method with the following signature will go *inside* the `SetOfInts` class:

```
    // Return the set intersection of me and set t.
    public SetOfInts intersect (SetOfInts t) {
```

The call `s.intersect(t)`, where `s` and `t` are both `SetOfInts`, should return a new `SetOfInts` containing those integers that occur in both `s` and `t`. For example, if `s` contains 1, 2, and 3, and `t` contains 3, 0, 2, and 5, then `s.intersect(t)` should return a new `SetofInts` that contains the integers 2 and 3.

a. [2 marks]

Consider an implementation that stores a SetofInts as an unsorted linked list. The following algorithm can be used to compute `s.intersect(t)`: Create a new `SetOfInts r` to hold the result. For each integer in `s`, if this integer belongs to set `t`, add it to the result set `r`.

Give the worst-case time-complexity of this algorithm, as a function of n (the number of elements in set `s`) and m (the number of elements in set `t`).

`O(n * m)`

b. [6 marks]
Assume that the following class, for nodes in the linked list, has already been defined:

```
// A node in the linked list of integers.
class SetOfIntsNode {
    public int data;
    public SetOfIntsNode next;
}
```

and that the `SetOfInts` class contains the following instance variable:

```
// The pointer to my first element.
private SetOfIntsNode first;
```

Write the `intersect()` method using the algorithm described in part (a).

```
// Return the set intersection of me and set t.
public SetOfInts intersect (SetOfInts t) {

      SetOfInts r = new SetOfInts();

      SetOfIntsNode temp = first;

      while (first != null) {

         if (t.contains(first.data))
              r.insert(first.data);

         first = first.next;
      }

      return r;
}


Alternative solution.  Not as good because it destroys the set
that is being intersected with t.

      SetOfInts r = new SetOfInts();

      while (! isEmpty() )

         int num = getOne();

         if (t.contains(num))
              r.insert(num);

         delete(num);
      }

      return r;
```

c. [3 marks]
Now consider a different implementation of class `SetOfInts`: a linked list *sorted* in non-decreasing order.

Suppose class `SetOfInts` contains an additional method called `getFirst()`, with no parameters, that returns (and removes from the linked list) the first element of the linked list. Describe an algorithm for the intersection method that takes advantage of the fact that the linked list is sorted. Your new algorithm must have worst-case time complexity $O(n + m)$, where n is the number of elements in set `s` and m is the number of elements in set `t`.

```
A merge-like process will solve this problem in O(n+m).  We start at the front of each
linked list, and work through the two lists in synch.  When we find an element that
is in both, we add it to the result and move on; when we don't, we just move on.
```

d. [2 marks]
Explain why your new algorithm is $O(n + m)$. You do not need all the space below to answer this.

```
It is O(n+m) because it requires us to look at each node only once and there are n
+ m nodes in total.
   (For the other method, every time we check membership in set t, we start at the
beginning of its linked list.  This means we look at the nodes in that linked list
many times.)
```

Question 6

[22 marks in total]

Assume we have written a class called `Stack` which implements the stack ADT. The elements of the stack are instances of `Object`. Our stack class has methods `push()`, `pop()`, `isEmpty()` and `size()`. The `size()` method returns the number of elements currently on the stack.

Consider the following class, which makes use class `Stack`.

```
class Client {

   // Delete Object o from Stack s.
   // Precondition:  o is not null.
   public static void delete (Object o, Stack s) {

         int sizeBefore = s.size();
         Stack temp = new Stack() ;
         Object element ;

         element = s.pop() ;
         while( ! element.equals(o) ) {
               temp.push (element);
               element = s.pop();
         }

         while( !temp.isEmpty() ) {
               Object tempElement = temp.pop();
               s.push (tempElement);
         }
   }

   public static void main (String[] args) {
         Stack myStack = new Stack();
         myStack.push("any");
         myStack.push("dot");
         myStack.push("car");
         myStack.push("ball");
         delete("dot", myStack);

         while( !myStack.isEmpty() ) {
               System.out.println( myStack.pop() );
         }
   }
}
```

a. [2 marks]

Trace execution of the `main()` method, and show what the output would be.

```
ball
car
any
```

b. [4 marks]
Method `delete()` needs two more preconditions. What are they?

PRECONDITION: `o occurs on the stack s at least once.`

PRECONDITION: `s is not empty.`

c. [4 marks]
For the first loop in `delete()`, give a loop invariant that best expresses the relationship between `sizeBefore`, `s.size()`, and `temp.size()`.

`sizeBefore = s.size() + temp.size() + 1`

d. [4 marks; correct answers earn +1, incorrect ones −1]
Assuming the preconditions of method `delete()` are met, which of the following are loop invariants for the *second* loop in `delete()`? (Don't worry about whether or not they are interesting, only if they are correct.) Indicate your answer for each expression by circling yes or no.

`s.size()` $<$ `sizeBefore`	$\boxed{\text{YES}}$	NO
`temp.size()` $\leq$ `s.size()`	YES	$\boxed{\text{NO}}$
`temp.size()` > 0	YES	$\boxed{\text{NO}}$
`temp.size()` ≥ 0	$\boxed{\text{YES}}$	NO

e. [4 marks; correct answers earn +1, incorrect ones −1]
Assuming the preconditions of method `delete()` are met, which of the following are postconditions for method `delete()`? (Don't worry about whether or not they are interesting, only if they are correct.) Indicate your answer for each expression by circling yes or no.

`s.size()` $=$ `sizeBefore`	YES	$\boxed{\text{NO}}$
`s.size()` $<$ `sizeBefore`	$\boxed{\text{YES}}$	NO
`temp.size()` > 0	YES	$\boxed{\text{NO}}$
`temp.size()` $= 0$	$\boxed{\text{YES}}$	NO

f. [4 marks]
Give post conditions which correctly and fully specify what method `delete()` does (assuming the preconditions are met).

`s is unaltered except that the topmost instance of o has been removed.`

Question 7

[8 marks]

Assume the following class has been defined:

```
// For creating nodes in a binary tree of Objects.
class BTNode {
     public Object data;
     public BTNode left, right;
}
```

Write a recursive method that returns the number of nodes in a given tree that have exactly two children. The method's interface has already been written for you.

```
public static int twoKids (BTNode root) {

     if (root == null)

         return 0;

     else if (root.left != null && root.right != null)

         return twoKids (root.left) + twoKids(root.right) + 1;

     else

         return twoKids (root.left) + twoKids(root.right);
}
```

Question 8

[15 marks in total]

The following method uses the same `BTNode` class as in Question 7. It returns the sum of all values in a binary search tree that are between `low` and `high` inclusive. It takes advantage of the order in the search tree to reduce the number of nodes that are considered.

```
static int sum (BTNode root, int low, int high) {

   if (root == null)
      return 0;
   else if (root.data < low)
      return sum (root.right, low, high);
   else if (root.data > high)
      return sum (root.left, low, high);
   else
      return root.data + sum (root.right, low, high) + sum (root.left, low, high);

}
```

a. [5 marks]
A **complete tree** is one in which all nodes have two children, and all leaves are at the same level. Let $AC(h)$ represent the number of additions and comparisons performed, in the worst case, by a call to `sum()` with a *complete* tree of height h. Write a recurrence relation for $AC(h)$. Do not solve the recurrence.

```
AC(0) = 1

AC(n) = 5 + AC(h-1) + AC(h-1)    if n >= 1
```

b. [10 marks]
Let $S(h)$ denote the statement

> "If `sum(root, low, high)` is called, where `root` is a reference to the root of a binary search tree of height h, then the call returns and the value returned is the sum of the node values in the tree between low and high inclusive."

On the next two pages is an outline for a proof that $S(h)$ is true for all $h \geq 0$. Complete the proof. For this part, do not assume that the tree is complete.

You may not need to use all the space available.

If there is a step that you need to prove but are unable to, **just say so and carry on** as if you had proven it.

Base Case: Prove that $\boxed{S(0) \text{ is true}}$

```
- I.e., prove that if sum(root, low, high) is called, where
  root is a reference to the root of a binary search tree of height 0,
  then the call returns and the value returned is the sum of the node
  values in the tree between low and high inclusive.

- Consider a call to sum(root, low, high), where
  root is a reference to the root of a binary search tree of height 0.

- If the tree has height 0, then root must be null.

- So the first if-condition succeeeds, and we immediately return 0.

- And since there are no nodes in the tree, 0 *is* the sum of the node
  values in the tree between low and high inclusive.

- So S(0) is true.
```

Let k be an arbitrary integer $\boxed{\geq 0}$

Induction Hypothesis: Assume that $\boxed{S(j) \text{ is true, for all } 0 \leq j \leq k}$

Induction Step: Prove that $\boxed{S(k+1) \text{ is true}}$

```
- I.e., prove that if sum(root, low, high) is called, where
  root is a reference to the root of a binary search tree of height k+1,
  then the call returns and the value returned is the sum of the node
  values in the tree between low and high inclusive.

- Consider a call to sum(root, low, high), where
  root is a reference to the root of a binary search tree of height k+1.

- Since k >= 0, (k+1) >= 1.  That is, the tree has height >= 1.

- Then root must not be null.

- So the first if-condition fails, and we continue on.

- There are 3 possible cases:
```

```
Case 1: root.data < low
          - Thus, the second if-condition succeeds and we
            return sum (root.right, low, high).
          - root.right is the root of a binary search tree of
            height k.
          - Since S(k) is true (by the induction hypothesis),
            the recursive call will return and the value returned
            will be the sum of the node values in that right subtree
            between low and high inclusive.
          - Because root.data < low, root.data should not be included
            in the sum to be returned.
          - Because the tree is a BST, all nodes in root's left subtree
            are also < low, and should not be included in the sum
            to be returned.
          - So the sum of the node values in that right subtree
            between low and high inclusive, which is the sum returned,
            is the sum of the node values in the whole tree
            between low and high inclusive.

Case 2: root.data > high
          Analogous argument to case 1.

Case 3: root.dat >= low and root.data <= high
          - Thus all the if-conditions fail and we do the final
            else, returning
                root.data +
                sum (root.right, low, high) + sum (root.left, low, high)
          - root.right is the root of a binary search tree of
            height k, as is root.left.
          - Since S(k) is true (by the induction hypothesis),
            both recursive calls will return and the value returned
            will be the sum of the node values in that subtree
            between low and high inclusive.
          - Because root.dat >= low and root.data <= high, it should
            be included in the sum to be returned.
          - So the sum of the node values in the whole tree between
            low and high inclusive is root.data plus the sum of the node
            values in the left subtree between low and high inclusive
            plus the sum of the node values in the right subtree between
            low and high inclusive.
          - And this is the value returned.

- So in any case, the call returns and the value returned is the sum of the node
  values in the whole tree between low and high inclusive.

- So S(k+1) is true.
```

Induction Conclusion: By induction, $S(h)$ is true for all integers $h \geq 0$.

END OF EXAM; Total pages = (19)