University of Toronto
csc148F—Introduction to Computing, Fall 1996

First Term Test — Horton's 6:00 section

Duration: 50 minutes

Aids allowed: None

Make sure that your examination booklet has 6 pages (including this one). Write your answers in the spaces provided. Write legibly.

Family Name: SOLUTIONS **Lecturer:** ____________

First Name: ____________ **Tutor:** ____________

Student # : ____________

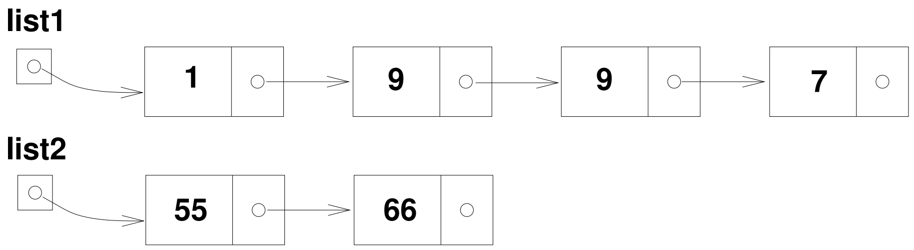
1.	______	/ 7
2.	______	/ 4
3.	______	/ 19
Total	______	/ 30

Question 1

[7 marks in total]

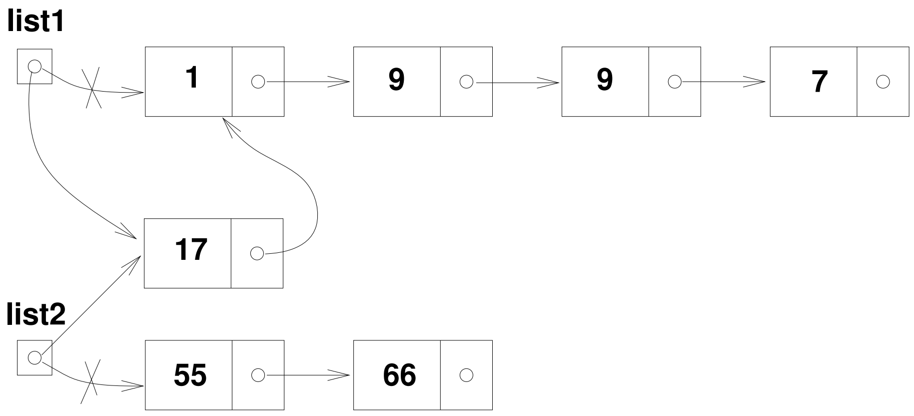
Assume that the type node below has been declared, and that some OOT code has been executed to create the linked lists shown below.

```
type node:
    record
      element : int
      next: ^node
    end record
```



(a) [5 marks]
Trace the following code and modify the diagram above to show the state of the linked lists and the variables list1 and list2 after the code has been executed:

```
new list2
list2 -> element := 17
list2 -> next := list1
list1 := list2
```



(b) [2 marks]
If the statement

```
put list1 -> element
```

were added at the end of the above code, what output would result?

```
17
```

Question 2

[4 marks in total]

(a) [2 marks]
The binary search procedure used in class requires 8 complete iterations of its loop if searching in a list of 256 elements. How many does it require if searching in a list of 2048 elements? Circle the best answer.

80 $\boxed{11}$ 256 8 24 2048 1780

(b) [2 marks]
The linear search procedure used in class requires 256 complete iterations of its loop if searching a list of 256 elements, none of which matches the searchkey. How many does it require if searching a list of 2048 elements, none of which matches the searchkey? Circle the best answer.

80 11 256 8 24 $\boxed{2048}$ 1780

Question 3

[19 marks in total]

Note: The parts of this question can be answered independently of each other.

As defined in class, an *Ordered Set* is composed of a finite (and non-negative) number of elements coming from an ordered domain, and contains no two identical elements. Below is a signature for a module that implements the ordered set ADT.

```
unit
module orderedSet

    import elementType
    export sizeOf, insert, retrieve, delete

    %  Returns the number of elements in the ordered set (0 if it's empty).
    function sizeOf : int

    % Given an item, adds it to the ordered set.  Has no effect if the item
    % already occurs in the ordered set.
    procedure insert (element : elementType)

    % If rank is between 1 and the size of the ordered set inclusive,
    % returns with answer set to the element of that rank from the ordered set,
    % and success set to true.  Otherwise, returns with success set to false.
    %     This procedure does not delete; it only reports on what is in the
    % set.
    procedure retrieve (rank : int, var success : boolean,
            var answer : elementType)

    % If rank is between 1 and the size of the ordered set inclusive,
    % deletes the element of that rank from the ordered set, and returns with
    % success set to true.  Otherwise, returns with success set to false.
    procedure delete (rank : int, var success : boolean)

end orderedSet
```

(a) [5 marks]

Suppose we want to define a function `isMember` in the main program, and that `isMember`'s purpose is to determine whether a given item is a member of the ordered set. The function has been started on the next page. Complete the function by filling in the two boxes.

Remember that this function is to go *outside* the module.

```
% This is the elementType that orderedSet will import:
type elementType : int

import orderedSet

% This function returns true if the given item already occurs in the ordered set,
% and false otherwise.
function isMember (element : elementType) : boolean

    % Retrieve the elements of the set in order by rank.  If any one of them
    % matches the element, then it is a member of the set, so return true.


                 +-----------------------+
    for i : 1 .. |   orderedSet.sizeOf   |
                 +-----------------------+
        var success : boolean
        var answer : elementType
        +-----------------------------------------------------------------------+
        |   orderedSet.retrieve(i, success, answer)                             |
        |   % NB: success is guaranteed to come back set to true in this case   |
        +-----------------------------------------------------------------------+
        if answer = element then
            result true
        end if

    end for

    % No element of the set matched the element we were searching for, so it
    % is not a member of the set.  Return false.
    result false

end isMember
```

(b) [4 marks]
We could have simply exported `numElements` from the module, rather than export a `sizeOf` function. Would this have been a better or worse design? Circle one.

BETTER $\boxed{\text{WORSE}}$

Explain your answer:

A module should "hide" implementation details, such as variables. That is, it should not export them. It should instead export a subprogram that provides any needed access to the information stored in the variable.

If we were to export numElements, then any program that used the module might come to rely on the fact that there is an int variable within the module, called numElements, that stores the number of items in the queue. This would constrain any future changes to the module unnecessarily.

CONTINUED

(c) [10 marks]
One way to implement the ordered set ADT is with a sorted linked list. The element of rank 1 would be stored in the first node in the linked list, the element of rank 2 in the second, and so on. An additional variable stores the number of elements currently in the ordered set. Below are type and variable declarations for this data structure.

```
type node :
    record
        entry : elementType
        next : ^node
    end record
var front : ^node := nil
var numElements : int := 0
```

Complete the module's retrieve procedure, assuming that the sorted linked list data structure just described has been declared appropriately within the module. Assume that the data structure has been properly maintained (*e.g.*, `numElements` correctly indicates the number of elements in the linked list).

Remember that this function is *inside* the module.

```
procedure retrieve (rank : int, var success : boolean, var answer : elementType)

     if (rank >= 1) and (rank <= numElements) then

           success := true
           var temp : ^node := front
           for i: 1..(rank-1)
                temp  := temp -> next
           end for
           answer := temp -> entry

     else
           success := false
     end if

end retrieve
```