MALASPINA UNIVERSITY-COLLEGE
FINAL EXAMINATION APRIL 1996
COMPUTING SCIENCE CSCI 161

NAME ______________________________

REG. NUMBER ______________________________

LEGAL SIGNATURE ______________________________

Instructor: Dr. P Walsh **Duration:** 3 Hours.

LATE WRITE

TOTAL VALUE	100			
QUESTION	1	2	3	4
VALUE	20	40	10	30
MARK				

Instructions

- Students must count the number of pages in this examination paper before beginning to write, and report any discrepancy immediately to the invigilator.
- This examination paper consists of 19 pages not including this cover page.
- Print your name and reg. number on the top of this examination paper. Sign the examination paper in the space provided. Answer all questions in the space provided. All questions relate to material covered in class or in the lab.
- This is a CLOSED BOOK - NO NOTES examination.
- Calculators are NOT permitted.

QUESTION 1. (20 marks)
Consider the PStack module interface specification (located at the back of this examination paper). Your task is to implement PStack to specification.

Implementation Details:
You must use an array to store the elements in the stack. Local declarations/definitions and some code are given on the following pages. You must use the given code to advantage and, you are not allowed to change the given code in any way.

```
(********** PStack module---implementation **********)

[inherit('PStackInt')]

module PStackImp(output);

(***** local constants *****)

(***** local types *****)

(***** local variables *****)

var     stackArray: array[1..PSMAXSIZ] of integer;
        siz: integer;

(***** local procedures and functions *****)

(***** exported procedures and functions *****)

[global] procedure PSInit;
        begin
        siz:=0
        end;
```

```
[global] procedure PSPush(x:integer;var status:PSStatus);

        begin






        end;

[global] procedure PSPop(var status:PSStatus);

        begin






        end;
```

```
[global] function PSTop(var status:PSStatus):integer;


        begin










        end;


[global] function PSDepth:integer;


        begin


        end;

[global] procedure PSDump;
        var i: integer;

        begin
        for i:=1 to siz do
                writeln(stackArray[i])
        end;

end.
```

QUESTION 2. (40 marks)
Consider the `DSSet` module interface specification (located at the back of this examination paper). Your task is to implement to specification the subprograms `DSSAdd`, `DSSDel` , `DSSIsMem` and `DSSSize`.

Implementation Details:
You must use a linked list to store the elements of the set. Local declarations/definations and some code are given on the following pages. Study the given code carefully!

Each node in the linked list contains an array. Set elements are stored in these arrays. Newly allocated nodes must be inserted at the head of the linked list. The first node in the list may contain up to `MAXNODESIZ` elements. All other nodes in the list **must** contain exactly `MAXNODESIZ` elements. The following is a representation of the list containing the elements {1,7,9,0,6,8,2}.

```
 head
------        -------------       ---------------
|  --|-----> |8|2| | | |-|-----> |1|7|9|0|6|nil|
------        -------------       ---------------


 siz       firstNodeSiz
------        ------
| 7  |        | 2  |
------        ------
```

You must use the given code to advantage and, you are not allowed to change the given code in any way.

The access routine `DSSInit` does NOT have to dispose of any memory associated with an existing list. The access routine `DSSDel` must dispose of the memory associated with a node containing no elements.

Hint: when adding an element to the set, a new node must be allocated if `(siz=0) or (firstNodeSiz=MAXNODESIZ)`.

```
(********** DSSet module---implementation **********)

[inherit('DSSetInt')]

module DSSetImp(output);

(***** local constants *****)

const
        MAXNODESIZ=5;
        NOTINNODE=0;
        NOTINSET=nil;

(***** local types *****)

type
        NodeArray=array[1..MAXNODESIZ] of integer;
        NodePtr=^Node;
        Node=record
              dataArray:NodeArray;
              next:NodePtr
        end;

(***** local variables *****)

var     head:NodePtr;
        siz:integer;            (* total number of elements in the set *)
        firstNodeSiz:integer;   (* number of elements in the first
                                   node of the linked list *)
        nPtr:NodePtr;           (* a pointer to a node *)
        nIndex:integer;         (* an index into a dataArray *)
```

```
(***** local procedures and functions *****)

(*
** Assumptions:
**      none
** Behaviour:
**      if x is in ptr^.dataArray then
**              return the its index position in dataArray
**      else
**              return NOTINNODE
*)
function FindInd(ptr: NodePtr; x:integer):integer;
        var     i, lastInNodeIndex: integer;
                found: boolean;

        begin
        i:=1;
        found:=false;
        FindInd:=0;
        if ptr=head then
                lastInNodeIndex:=firstNodeSiz
        else
                lastInNodeIndex:=MAXNODESIZ;
        while (not found) and (i <= lastInNodeIndex) do
                begin
                if ptr^.dataArray[i]=x then
                        begin
                        found:=true;
                        FindInd:=i
                        end
                else
                        i:=i+1
                end
        end;
```

```
(*
** Assumptions:
**      none
** Behaviour:
**      if x is in the set then
**              ptr:=a pointer to the node where x resides
**              ind:=the index into ptr^.dataArray where x is located
**      else
**              ptr:=NOTINSET
*)
procedure Find(x:integer; var ptr:NodePtr; var ind:integer);
        var found: boolean;

        begin
        ptr:=head;
        found:=false;
        while (not found) and (ptr<>nil) do
                begin
                ind:=FindInd(ptr, x);
                if ind<>NOTINNODE then
                        found:=true
                else
                        ptr:=ptr^.next
                end
        end;

(***** exported procedures and functions *****)

[global] procedure DSSInit;
        begin
        head:=nil; (* note, existing nodes are not disposed *)
        siz:=0;
        firstNodeSiz:=0
        end;
```

```
[global] procedure DSSAdd(x:integer;var status:DSSStatus);

        begin










        end;
```

```
[global] procedure DSSDel(x:integer;var status:DSSStatus);

        begin










        end;
```

```
[global] function DSSIsMem(x:integer):boolean;



        begin



        end;


[global] function DSSSize:integer;



        begin



        end;

[global] procedure DSSDump;
        var i, lastInNodeIndex:integer;
            ptr: NodePtr;

        begin
        ptr:=head;
        while ptr<>nil do
                begin
                if ptr=head then
                        lastInNodeIndex:=firstNodeSiz
                else
                        lastInNodeIndex:=MAXNODESIZ;
                for i:=1 to lastInNodeIndex do
                        writeln(ptr^.dataArray[i]);
                ptr:=ptr^.next
                end
        end;

end.
```

QUESTION 3. (10 marks)
(a) Show the result of inserting 2, 4, 1, 5, 9, 3, 6, 7 into an initially empty binary search tree. Note, insert 2 first, then 4 etc.

(b) Using the tree from part (a), show the result of deleting 3, 5, and 2. Note, delete 3 then 5 and finally 2.

QUESTION 4. (30 marks)
Consider the `TISet` module interface specification (located at the back of this examination paper). Your task is to implement to specification the local subprograms `Find`, `GetHeight`, `Dump` and `Thread`.

Implementation Details:
The elements of the set are stored in a binary search tree. Local declarations/definitions and some code are given on the following pages. You must use the given code to advantage and, you are not allowed to change the given code in any way. You may assume that `itHead` is initialized to nil when `SEQ` mode is entered.

```
(********** TISet module---implementation **********)

[inherit('TISetInt')]

module TISetImp(output);

(***** local types *****)

type    TreePtr=^Tree;
        Tree=record
                data:integer;
                left:TreePtr;   (* left and right are used by the *)
                right:TreePtr;  (* binary search tree *)
                next:TreePtr    (* next is used to thread the tree *)
                end;

(***** local variables *****)

var     root: TreePtr;
        itHead: TreePtr;
        siz: integer;
        mode: TISMode;
        tempStatus: TISStatus;
```

```
(***** local procedures and functions *****)

(*
** Assumptions:
**    none
** Behaviour:
**      if x is in the set then
**              return a pointer to the node containing x
**      else
**              return  nil
*)
function Find(x:integer; tree: TreePtr):TreePtr;


        begin










        end;
```

```
(*
** Assumptions:
**      none
** Behaviour:
**      return the height of the node pointed to by tree
**      note, the height of a nil tree is -1
*)
function GetHeight(tree: TreePtr):integer;


        begin











        end;


(*
** Assumptions:
**      none
** Behaviour:
**      Display a post-order listing of the elements of s
*)
procedure Dump(tree: TreePtr);


        begin




        end;
```

```
(*
** Assumptions:
**      none
** Behaviour:
**      Build a linked-list thread through nodes of the tree
**      itHead will point to the head of the list
**      and next (see Tree type definition) will point
**      to the next node in the list
**      The linked list is to be used by the iterator
*)
procedure Thread(tree: TreePtr);



        begin









        end;

...
end.
```