Name _____

Student number _____

SE 2A04 Fall 2001

Surprise Quiz 2 Answer Key

Instructor: W. M. Farmer

You have 20 minutes to complete this quiz consisting of 3 pages and 1 question. Good luck!

Below is a before/after MIS for a module that stores an abstract array of REALs. Also below is an Oberon module that is intended to implement the MIS by representing the abstract array as a linked list. The Oberon module contains 10 mistakes (some are purely mistakes of syntax). Circle the lines of code that contain the mistakes according to the following rules:

(1) 10 points will be awarded if a circle contains a mistake.

(2) 5 points will be deducted if a circle does not contain a mistake.

(3) A circle that contains more than one line will be ignored.

(4) If there are $x > 10$ circles, $10 \cdot (x - 10)$ points will be deducted.

Before/after MIS:

(1) Imported modules: none.

(2) Interface:

```
PROCEDURE Construct(a: ARRAY OF REAL);
PROCEDURE Select(i: LONGINT): REAL;
PROCEDURE Mutate(i: LONGINT; r: REAL);
```

(3) Exceptions: BadIndex.

(4) State constants: none.

(5) State variables: $\mathsf{array} : \texttt{LONGINT} \to \texttt{REAL}$ (initially array is the empty function)

(6) Behavior rules:

Name	Input	Output	Transition
`Construct`	a : `ARRAY OF REAL`		$\mathsf{array}' = \lambda\, i : \texttt{LONGINT} \,.\, a[i]$
`Select`	i : `LONGINT`	$\mathsf{array}(i)$	
`Mutate`	i : `LONGINT`, r : `REAL`		$\mathsf{array}' = \lambda\, j : \texttt{LONGINT} \,.\, \mathsf{if}(i = j, r, \mathsf{array}(j))$

Oberon implementation with mistakes:

```
MODULE AbstractArray;

(* INTERFACE *)

(* PROCEDURE Construct(a: ARRAY OF REAL);
   PROCEDURE Select(i: LONGINT): REAL;
   PROCEDURE Mutate(i: LONGINT; r: REAL); *)

(* IMPLEMENTATION *)

(* Types *)

TYPE LinkedList = POINTER TO LinkedListRec;
     LinkedListRec =
       RECORD
         item: REAL;
         next: LinkedList;
       END;

(* Variables *)

VAR   array:  LinkedListRec;     (* Abstract array as a linked list *)
CONST length: LONGINT;           (* Length of abstract array *)

(* Exceptions: *)

PROCEDURE BadIndexException();
  BEGIN
    HALT(33);                    (* Abort program *)
  END BadIndexException;

(* Interface functions *)

PROCEDURE Construct*(a: ARRAY OF REAL);
  VAR index: LONGINT;
      p: LinkedList;
  BEGIN
    index := -1;
    length := LEN(a);
    WHILE index <= length DO
      IF index = 0 THEN
        p^.next := MakeRecord(a[index]);
        array := p;
      ELSE
        p^.next := MakeRecord(a[index]);
        p := p^.next;
      END;
      index := index + 1;
    END;
  END Construct;
```

```
PROCEDURE Select*(i: LONGINT): REAL;
  VAR p: LinkedList;
  BEGIN
    p := FindRecord(i);
    RETURN p^.item;
  END Select;

PROCEDURE Mutate*(i: LONGINT; r: REAL);
  VAR p: LinkedList;
  BEGIN
    p := FindRecord(i);
    p^.item := r;
    RETURN p^.item;
  END Mutate;

(* Local functions *)

PROCEDURE MakeRecord(r: REAL): LinkedList;
  VAR p: LinkedList;
  BEGIN
    p := NEW(r);
    p^.item := r;
    p^.next := NIL;
    RETURN p;
  END MakeRecord;

PROCEDURE FindRecord(i: LONGINT): LinkedList;
  VAR index: LONGINT;
      p:     LinkedList;
  BEGIN
    IF (i < 0) OR (length <= i) THEN
      BadIndexException();
    ELSE
      index := 0;
      p:= array;
      WHILE index < i DO
        p := p^.previous;
        index := index - 1;
      END;
      RETURN p;
    END;
  END FindRecord;

PROCEDURE Init();
  BEGIN
    array  := 0;
    length := 0;
  END Init;

BEGIN
  Init();
END AbstractArray.
```

Corrected Oberon implementation:

```
MODULE AbstractArray;

(* INTERFACE *)

(* PROCEDURE Construct(a: ARRAY OF REAL);
   PROCEDURE Select(i: LONGINT): REAL;
   PROCEDURE Mutate(i: LONGINT; r: REAL); *)

(* IMPLEMENTATION *)

(* Types *)

TYPE LinkedList = POINTER TO LinkedListRec;
     LinkedListRec =
       RECORD
         item: REAL;
         next: LinkedList;
       END;

(* Variables *)

VAR array:  LinkedList;        (* Abstract array as a linked list *)
    length: LONGINT;           (* Length of abstract array *)

(* Exceptions: *)

PROCEDURE BadIndexException();
  BEGIN
    HALT(33);  (* Abort program *)
  END BadIndexException;

(* Interface functions *)

PROCEDURE Construct*(a: ARRAY OF REAL);
  VAR index: LONGINT;
      p: LinkedList;
  BEGIN
    index := 0;
    length := LEN(a);
    WHILE index < length DO
      IF index = 0 THEN
        p := MakeRecord(a[index]);
        array := p;
      ELSE
        p^.next := MakeRecord(a[index]);
        p := p^.next;
      END;
      index := index + 1;
    END;
  END Construct;
```

```
PROCEDURE Select*(i: LONGINT): REAL;
  VAR p: LinkedList;
  BEGIN
    p := FindRecord(i);
    RETURN p^.item;
  END Select;

PROCEDURE Mutate*(i: LONGINT; r: REAL);
  VAR p: LinkedList;
  BEGIN
    p := FindRecord(i);
    p^.item := r;
  END Mutate;

(* Local functions *)

PROCEDURE MakeRecord(r: REAL): LinkedList;
  VAR p: LinkedList;
  BEGIN
    NEW(p);
    p^.item := r;
    p^.next := NIL;
    RETURN p;
  END MakeRecord;

PROCEDURE FindRecord(i: LONGINT): LinkedList;
  VAR index: LONGINT;
      p:     LinkedList;
  BEGIN
    IF (i < 0) OR (length <= i) THEN
      BadIndexException();
    ELSE
      index := 0;
      p:= array;
      WHILE index < i DO
        p := p^.next;
        index := index + 1;
      END;
      RETURN p;
    END;
  END FindRecord;

PROCEDURE Init();
  BEGIN
    array  := NIL;
    length := 0;
  END Init;

BEGIN
  Init();
END AbstractArray.
```