Name ________________________
Student Number________________________

Software Engineering 2A04
Answer Key

DAY CLASS — Dr. William M. Farmer
DURATION OF EXAMINATION: 3 Hours
MCMASTER UNIVERSITY FINAL EXAMINATION — December 2001

(1) [2 pts.] The professional responsibilities of a software engineer are essentially the same as the professional responsibilities of any other kind of engineer. Is this statement true or false?

A.) $\boxed{\text{True.}}$

B.) False.

(2) [2 pts.] Suppose that P is a guarded program list of the form

$$(G_1 \to P_1 | \cdots | G_n \to P_n).$$

If $G_1, \ldots, G_n$ cover all cases and there is some i with $1 \leq i \leq n$ such that the program P_i satisfies the specification S when G_i is true, then P satisfies S. Is this statement true or false?

A.) True.

B.) $\boxed{\text{False.}}$

(3) [2 pts.] Procedures defined using recursion are not generally as space efficient as procedures defined using iteration. Is this statement true or false?

A.) $\boxed{\text{True.}}$

B.) False.

(4) [2 pts.] An actual description of a product is intended to describe the attributes that the product is required to possess. Is this statement true or false?

A.) True.

B.) $\boxed{\text{False.}}$

(5) [2 pts.] The "secret" of a module is its interface. Is this statement true or false?

A.) True.

B.) $\boxed{\text{False.}}$

Continued on page 2

(6) [2 pts.] An exception is usually defined as an event that causes a program to abort. Is this statement true or false?

- A.) True.
- B.) $\boxed{\text{False.}}$

(7) [2 pts.] The wider the scope of a variable, the more descriptive its name needs to be. Is this statement true or false?

- A.) $\boxed{\text{True.}}$
- B.) False.

(8) [2 pts.] Wild random testing is good for checking the reliability of a software product. Is this statement true or false?

- A.) True.
- B.) $\boxed{\text{False.}}$

(9) [2 pts.] According to Frederick Brooks's mythical man-month idea, the worst way to help a software project that is behind schedule is to

- A.) $\boxed{\text{Add more workers to the project team.}}$
- B.) Reschedule the project over a longer period of time.
- C.) Cut back on the goals of the project.
- D.) Reuse completed work from other projects.

(10) [2 pts.] The idea behind Harlan Mill's surgical team proposal for software development is that

- A.) Fluff in a software product must be surgically removed.
- B.) $\boxed{\text{With the right kind of development team, the benefits of a one-person team can be realized on a project too big for one person.}}$
- C.) A development team should have only one leader.
- D.) Serious design flaws in a software product are best fixed by a small repair team lead by an experienced software "surgeon".

(11) [2 pts.] According to Brooks, why should one plan to throw away the first implementation of a product?

- A.) The first implementation will usually not use the latest technology.
- B.) The first implementation will have many bugs that can never be found.
- C.) After seeing the first implementation, the client will usually ask for a completely new implementation.
- D.) $\boxed{\text{In one way or another, the first implementation will eventually be thrown away anyway.}}$

Continued on page 3

(12) [2 pts.] According to Brooks, what is the most important consideration in system design?

A.) Ease of use.

B.) Ease of implementation.

C.) $\boxed{\text{Conceptual integrity.}}$

D.) Cost of design materials.

(13) [2 pts.] How many interface functions are in the rectangle module for Lab Exercise 2?

A.) 0.

B.) $\boxed{1.}$

C.) 8.

D.) 18.

(14) [2 pts.] In Lab Exercise 4, what value is returned when `rectangle(n)` is executed immediately after `START(n)` is executed?

A.) -1.

B.) 0.

C.) $\boxed{\texttt{FALSE.}}$

D.) `TRUE`.

(15) [2 pts.] In Lab Exercise 5, if `GA(n)` returns the value 4, which "S" procedure was called most recently?

A.) `S2P`.

B.) `SPHW`.

C.) `SPHA`.

D.) $\boxed{\texttt{S2P, SPHW, or SPHA.}}$

(16) [2 pts.] The rectangle module of Lab Exercise 3 can store at most ____ rectangles at the same time.

A.) 1.

B.) $\boxed{2.}$

C.) 8.

D.) 100.

Continued on page 4

(17) [2 pts.] In Lab Exercise 4, the interface procedure `GW` will never return the value

A.) $\boxed{\text{-2.}}$

B.) -1.

C.) 1.

D.) 2.

(18) [2 pts.] According to the definitions given in Lab Exercise 2, `AinB` implies

A.) $\boxed{\neg\texttt{tangent}.}$

B.) `AisB`.

C.) `disjoint`.

D.) `overlap`.

(19) [2 pts.] Which software structure is useful for dividing a product into subproducts (i.e, subsets of the product)?

A.) Data flow.

B.) State transition.

C.) $\boxed{\text{Uses hierarchy.}}$

D.) All of the above.

(20) [2 pts.] Which kind of function specification is always deterministic?

A.) $\boxed{\text{Definitional.}}$

B.) Relational.

C.) Axiomatic.

D.) All of the above.

(21) [2 pts.] Any ____ that satisfies the ____ should be acceptable.

A.) Design, requirements.

B.) Implementation, design.

C.) Implementation, requirements.

D.) $\boxed{\text{All of the above.}}$

Continued on page 5

(22) [2 pts.] Software testing can usually be used to show the ____ of a software product.

A.) Correctness.

B.) $\boxed{\text{Incorrectness.}}$

C.) Trustworthiness.

D.) All of the above.

(23) [2 pts.] Development is performed in a top-down manner in the ____ software "lifecycle" model.

A.) $\boxed{\text{Refinement.}}$

B.) Incremental.

C.) Spiral.

D.) Prototyping.

(24) [2 pts.] An axiomatic input/output MIS provides a good basis for

A.) Greybox testing an implementation of the MIS.

B.) Clearbox testing an implementation of the MIS.

C.) Simulating an implementation of the MIS.

D.) $\boxed{\text{Mathematically verifying that an implementation satisfies the MIS.}}$

(25) [2 pts.] As a data structure, a constant has ____ mutators.

A.) $\boxed{\text{0.}}$

B.) 1.

C.) More than 1.

D.) More than 2.

(26) [2 pts.] It is often useful to view a procedure with side effects as a

A.) Turing machine.

B.) $\boxed{\text{Finite state machine.}}$

C.) An expression in a language.

D.) As the proof of a formula.

Continued on page 6

(27) [2 pts.] Suppose an Oberon module includes the following declarations:

```
VAR y: REAL;

PROCEDURE Achilles(VAR x: REAL);
  BEGIN
    x := -1;
  END Achilles;
```

Then `Achilles(y)` is a

A.) Call by name.

B.) Call by value.

C.) $\boxed{\text{Call by reference.}}$

D.) Call by variation.

(28) [2 pts.] Which of the following statements is true?

A.) $(\mathrm{I}\, x : \mathbf{Z} \,.\, 0 \neq x \wedge x * x = x + x) = 2$.

B.) $(\lambda\, x : \mathbf{Z} \,.\, x - y)(y) = 0$.

C.) $-6 \leq \mathsf{if}(A, -4, 4) \leq 6$ where A is some formula.

D.) $\boxed{\text{All of the above.}}$

(29) [2 pts.] The interface for the lists abstract data type (ADT) presented in class includes

```
PROCEDURE Member(i: INTEGER, k: List): INTEGER;
```

This interface function is a ____ for the ADT.

A.) Constructor.

B.) $\boxed{\text{Selector.}}$

C.) Mutator.

D.) All of the above.

(30) [2 pts.] Which is not a means of abstraction?

A.) A lambda expression $\lambda\, x : \alpha \,.\, t$.

B.) $\boxed{\text{An Oberon IF statement.}}$

C.) An Oberon procedure declaration.

D.) An Oberon module declaration.

Continued on page 7

(31) [5 pts.] In class we discussed the following interface for an abstract data type (ADT) of stacks:

```
TYPE Stack;
CONST Bottom: Stack;
PROCEDURE Push(i: INTEGER, s: Stack): Stack;
PROCEDURE Top(s: Stack): INTEGER;
PROCEDURE Pop(s: Stack): Stack;
```

Using the interface functions of this interface, write a full Oberon procedure declared as

```
PROCEDURE Reverse(s: Stack): Stack;
```

that, given a stack s as input, returns a stack that contains the members of s in reverse order.

Answer:

```
PROCEDURE Reverse(s: Stack): Stack;
  VAR s1: Stack;
  BEGIN
    s1 := Bottom;
    WHILE s # Bottom DO
      s1 := Push(Top(s),s1)
      s := Pop(s);
    END;
    RETURN s1;
  END Reverse;
```

(32) [5 pts.] Write an interface for an Oberon module for an abstract data type (ADT) of 2-dimensional vectors. The interface functions of the interface should include:

A.) A constructor that, given two `REAL` numbers x and y, returns a vector whose coordinates are x and y.

B.) Appropriate selectors.

C.) An interface function for adding two vectors.

D.) An interface function for multiplying a vector by a `REAL` number.

Continued on page 8

Answer:

```
TYPE Vector;
PROCEDURE Make(x,y: REAL): Vector;
PROCEDURE GetX(v: Vector): REAL;
PROCEDURE GetY(v: Vector): REAL;
PROCEDURE Add(u,v: Vector): Vector;
PROCEDURE Muliply(x: REAL; v: Vector): Vector;
```

(33) [10 pts.] Below is a before/after MIS for a module that stores a *set* of `REAL`s. Also below is an Oberon module that is intended to implement the MIS by representing the set as a linked list. The Oberon module contains 5 mistakes. Circle the lines of code that contain the mistakes. 2 points will be awarded if a circle contains a mistake. Only the first 5 circles will be considered; the rest will be ignored. Any circle that contains more than one line will also be ignored.

Before/after MIS:

A.) Imported modules: none.

B.) Interface:

```
PROCEDURE Reset();
PROCEDURE Empty(): BOOLEAN;
PROCEDURE IsMember(r: REAL): BOOLEAN;
PROCEDURE Adjoin(r: REAL);
PROCEDURE Remove(r: REAL): BOOLEAN
```

C.) Exceptions: none

D.) State constants: none

E.) State variables: $S : \mathsf{sets}(\mathtt{REAL})$ (initially S is the empty set $\emptyset$)

F.) Behavior rules:

Name	Input	Output	Transition
`Reset`			$S' = \emptyset$
`Empty`		$S = \emptyset$	
`IsMember`	$r : \mathtt{REAL}$	$r \in S$	
`Adjoin`	$r : \mathtt{REAL}$		$S' = S \cup \{r\}$
`Remove`	$r : \mathtt{REAL}$	$r \in S$	$S' = S - \{r\}$

Continued on page 9

Oberon implementation with mistakes:

```
MODULE Set;

(* INTERFACE *)

(*
PROCEDURE Reset();
PROCEDURE Empty(): BOOLEAN;
PROCEDURE IsMember(r: REAL): BOOLEAN;
PROCEDURE Adjoin(r: REAL);
PROCEDURE Remove(r: REAL): BOOLEAN;
*)

(* IMPLEMENTATION *)

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

VAR set: LinkedList;        (* Set as a linked list *)

PROCEDURE Reset*();
  BEGIN
    set := NIL;
  END Reset;

PROCEDURE Empty*(): BOOLEAN;
  BEGIN
    RETURN set = 0;
  END Empty;

PROCEDURE IsMember*(r: REAL): BOOLEAN;
  VAR p:     LinkedList;
  BEGIN
    p := set;
    WHILE p # NIL DO
      IF p^.item = r THEN
        RETURN TRUE;
      ELSE
        p := p^.next;
      END;
    END;
    RETURN TRUE;
  END IsMember;
```

Continued on page 10

```
PROCEDURE Adjoin*(r: REAL);
  VAR p: LinkedList;
  BEGIN
    NEW(p);
    p^.item := r;
    p^.next := set;
    set := p;
  END Adjoin;

PROCEDURE Remove*(r: REAL): BOOLEAN;
  VAR p1,p2: LinkedList;
  BEGIN
    p1 := NIL;
    p2 := set;
    WHILE p2 # NIL DO
      IF p2^.item = r THEN
        IF p1 = NIL THEN
          set := p1^.next;
        ELSE
          p1^.next := p2^.next;
        END;
        RETURN TRUE;
      ELSE
        p1 := p1^.next;
        p2 := p2^.next;
      END;
    END;
    RETURN FALSE;
  END Remove;

BEGIN
  Empty();
END Set.
```

Continued on page 11

Corrected Oberon implementation:

```
MODULE Set;

(* INTERFACE *)

(*
PROCEDURE Reset();
PROCEDURE Empty(): BOOLEAN;
PROCEDURE IsMember(r: REAL): BOOLEAN;
PROCEDURE Adjoin(r: REAL);
PROCEDURE Remove(r: REAL): BOOLEAN
*)

(* IMPLEMENTATION *)

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

VAR set: LinkedList;        (* Set as a linked list *)

PROCEDURE Reset*();
  BEGIN
    set := NIL;
  END Reset;

PROCEDURE Empty*(): BOOLEAN;
  BEGIN
    RETURN set = NIL;
  END Empty;

PROCEDURE IsMember*(r: REAL): BOOLEAN;
  VAR p:     LinkedList;
  BEGIN
    p := set;
    WHILE p # NIL DO
      IF p^.item = r THEN
        RETURN TRUE;
      ELSE
        p := p^.next;
      END;
    END;
    RETURN FALSE;
  END IsMember;
```

Continued on page 12

```
PROCEDURE Adjoin*(r: REAL);
  VAR p: LinkedList;
  BEGIN
    NEW(p);
    p^.item := r;
    p^.next := set;
    set := p;
  END Adjoin;

PROCEDURE Remove*(r: REAL): BOOLEAN;
  VAR p1,p2: LinkedList;
  BEGIN
    p1 := NIL;
    p2 := set;
    WHILE p2 # NIL DO
      IF p2^.item = r THEN
        IF p1 = NIL THEN
          set := p2^.next;
        ELSE
          p1^.next := p2^.next;
        END;
        RETURN TRUE;
      ELSE
        p1 := p2;
        p2 := p2^.next;
      END;
    END;
    RETURN FALSE;
  END Remove;

BEGIN
  Reset();
END Set.
```

Continued on page 13

(34) [20 pts.] Below is a before/after MIS for a module that stores a number of bank accounts. Write a complete Oberon module named `Bank` that implements the MIS. Points will be taken off for any irrelevant extra code.

Before/after MIS:

A.) Imported modules: none.

B.) Interface:

```
PROCEDURE Balance(n: INTEGER): REAL;
PROCEDURE Deposit(n: INTEGER; r: REAL);
PROCEDURE Withdraw(n: INTEGER; r: REAL);
```

C.) Exceptions: BadArgs : `BOOLEAN`

D.) State constants:

capacity : `INTEGER` $(0 < \mathsf{capacity})$.

penalty : `REAL` $(0 < \mathsf{penalty})$.

E.) State variables: b : `ARRAY` capacity `OF REAL`

Initially, $\forall\, i : \texttt{INTEGER} \,.\, 0 \leq i < \mathsf{capacity} \supset b[i] = 0$.

F.) Behavior rules:

`Balance`	
Input	$n : \texttt{INTEGER}$
Output	$b[n]$
Transition	
Exception	$n < 0 \;\vee\; \mathsf{capacity} \leq n \;\supset\; \mathsf{BadArgs}$
`Deposit`	
Input	$n : \texttt{INTEGER},\ r : \texttt{REAL}$
Output	
Transition	$b'[n] = b[n] + r$ $\wedge \forall\, i : \texttt{INTEGER} \,.\, i \neq n \supset b'[i] \simeq b[i]$
Exception	$n < 0 \;\vee\; \mathsf{capacity} \leq n \;\vee\; r \leq 0 \;\supset\; \mathsf{BadArgs}$
`Withdraw`	
Input	$n : \texttt{INTEGER},\ r : \texttt{REAL}$
Output	
Transition	$b'[n] = b[n] - \mathsf{if}(r \leq b[n],\ r,\ \mathsf{penalty})$ $\wedge \forall\, i : \texttt{INTEGER} \,.\, i \neq n \supset b'[i] \simeq b[i]$
Exception	$n < 0 \;\vee\; \mathsf{capacity} \leq n \;\vee\; r \leq 0 \;\supset\; \mathsf{BadArgs}$

Recall that $a \simeq b$ means $a = b$ or both a and b are undefined.

Continued on page 14

Answer:

```
MODULE Bank;

IMPORT Out;

(* INTERFACE *)

(*
PROCEDURE Balance(n: INTEGER): REAL;
PROCEDURE Deposit(n: INTEGER; r: REAL);
PROCEDURE Withdraw(n: INTEGER; r: REAL);
*)

(* IMPLEMENTATION *)

CONST capacity = 5000;             (* Number of bank accounts *)
       penalty = 8.50;             (* Penalty for insufficient funds *)

VAR b: ARRAY capacity OF REAL;     (* Array of bank accounts *)

PROCEDURE BadArgsException();
  BEGIN
    Out.String("Insufficient funds!");
  END BadArgsException;

PROCEDURE Init();
  VAR i: INTEGER;
  BEGIN
    FOR i := 0 TO capacity - 1 DO
      b[i] := 0;
    END;
  END Init;

PROCEDURE Balance*(n: INTEGER): REAL;
  BEGIN
    IF (n < 0) OR (capacity <= n) THEN
      BadArgsException();
    ELSE
      RETURN b[n];
    END;
  END Balance;
```

Continued on page 15

```
PROCEDURE Deposit*(n: INTEGER; r: REAL);
  BEGIN
    IF (n < 0) OR (capacity <= n) OR (r <= 0) THEN
      BadArgsException();
    ELSE
      b[n] := b[n] + r;
    END;
  END Deposit;

PROCEDURE Withdraw*(n: INTEGER; r: REAL);
  BEGIN
    IF (n < 0) OR (capacity <= n) OR (r <= 0) THEN
      BadArgsException();
    ELSE
      IF r <= b[n] THEN
        b[n] := b[n] - r;
      ELSE
        b[n] := b[n] - penalty;
      END;
    END;
  END Withdraw;

BEGIN
  Init();
END Bank.
```

The End