

Concordia University  
Department of Computer Science

Final Examination

**Course:** COMP 245/2 All Sections

**Date:** Saturday, 7 December 1996.

**Time:** 14:00 – 17:00

**Instructor:** P. Dénommée, S.S. Ghaderpanah

**Coordinator:** Greg Butler

**Special Instructions and Information**

- You may use writing and drawing materials only.
- Pocket calculators are not allowed.
- There are eight questions altogether. Answer **FIVE** of them only. Each question carries 20% of the total mark.
- Start each question on a fresh page. Write the numbers of the questions that you have chosen to do on the front of the examination booklet.
- Write programs in Pascal.
- Each answer should be a full and complete answer.
- Hand in this examination with your booklets.

# COMP 245

## Final Examination, Fall 1996

Answer FIVE questions only.

**Question 1.** The sum of the first  $N$  odd numbers is  $N^2$ . For example

$$1 + 3 + 5 + 7 = 16 = 4^2.$$

Use this fact to write an algorithm which computes  $N^2$  by adding odd numbers.

- (a) [5%] Give suitable pre-condition and post-condition for the algorithm.
- (b) [5%] Write the pseudocode of your algorithm, including the pre-condition, post-condition, and loop invariant.
- (c) [5%] Prove that your algorithm terminates.
- (d) [5%] Prove that your algorithm is correct.

**Question 2.** The recursive definition of multiplication of two integers  $M$  and  $N$  is described below:

$$\begin{array}{ll} M = 0 \Rightarrow & M \times N = 0 \\ M \text{ is even} \Rightarrow & M \times N = (M \div 2) \times (2 \times N) \\ M \text{ is odd} \Rightarrow & M \times N = N + (M-1) \times N \end{array}$$

Its recursive algorithm can be found as follows:

```
function Mul ( M, N : integer): integer;
begin { Pre: M ≥ 0 }
  if M = 0
  then Mul := 0
  else if even( M )
  then Mul := Mul( M ÷ 2, 2 × N )
  else Mul := N + Mul( M - 1, N )
end; { Post: Mul = M × N }
```

- (a) [5%] Trace the execution of the above algorithm on the input  $M = 5, N = 3$ .
- (b) [5%] What is the inductive hypothesis to prove the above algorithm using an inductive proof.
- (c) [10%] Give details of the inductive proof using the above hypothesis for the above algorithm.

**Question 3.** The following declarations define a type for binary trees with an integer at each node.

```

type
  TreePtr = ↑ TreeNode;
  TreeNode =
    record      IntVal: integer;
              Left, Right: TreePtr
    end;

```

(a) [6%] Write a recursive function with signature

```
function Sum(t: TreePtr): integer;
```

such that the value of  $Sum(t)$  is the sum of the integers in the nodes of the tree  $t$ . If  $t$  is empty, this value should be zero.

(b) [6%] Write a recursive function with signature

```
function Height(t: TreePtr): integer;
```

such that the value of  $Height(t)$  is zero if  $t$  is empty, otherwise it is one greater than the maximum heights of the left and right subtrees of  $t$ .

(c) [8%] A *heap* is a binary tree with the property that the number at a node is greater than the numbers in the subtrees of the node. Write a recursive function with signature

```
function IsHeap(t: TreePtr): boolean;
```

such that  $IsHeap(t)$  returns true if and only if  $t$  is a heap.

Question 4. Answer the following questions about complexity.

- (a) [5%] Express the complexity of the following function in big-O notation

$$\frac{n!}{10^{1000}} + 10^{100} n^{10000} + 1000000000^{100n}$$

- (b) [5%] Prove that  $n \log(n)$  is not  $O(n)$ .
- (c) [10%] Give the time and space complexity of the following function.

```
function f( x:integer ): real;
var tmp: real;
begin
  if x = 0 then
    f := 10.0
  else begin
    tmp := f( x div 2 );
    f := x * tmp;
  end;
end;
```

Question 5. Consider the following recursive function for constructing a list of integers. Note that it uses the *Cons* function from the course notes. The notation  $(x_1, x_2, \dots, x_k)$  denotes a list with  $k$  entries where the  $i$ -th entry is  $x_i$ . So  $x_1$  is the first entry in the list, and  $x_k$  is the last entry.

```
function Sequence( n : integer ) : List;
begin { Pre: n ≥ 0 }
  if n = 0 then
    Sequence := nil
  else
    Sequence := Cons( n, Sequence( n - 1 ) )
end; { Post: Sequence = (n, n-1, n-2, ..., 1) }
```

- (a) [5%] Trace the execution of the function call *Sequence*(3).
- (b) [5%] Prove that the function terminates, provided  $n \geq 0$ .
- (c) [10%] Prove the correctness of the function, provided  $n \geq 0$ .

Question 6. Consider the following algorithm to compute the minimum entry  $M$  in an array of integers  $A[1..N]$ .

```

{ Pre:  $A[1..N]$  is an array of integers and  $N \geq 1$  }
 $M := A[1]$ ;
 $k := 1$ ;
{ Loop invariant:  $M$  is the minimum element in  $A[1..k]$  }
while  $k \neq N$  do
   $k := k + 1$ ;
  if  $M > A[k]$  then
     $M := A[k]$ 
  else
    skip
{ Post:  $M$  is the minimum element in  $A[1..N]$  }

```

- (a) [5%] Complete the following definition of the predicate  $Min$  by supplying the value for “?”, and then restate the loop invariant using the predicate  $Min$ .

The predicate  $Min(x, A[Lo..Hi])$  is true precisely when  $x$  is the minimum value of the entries in the array segment  $A[Lo..Hi]$ .

$$Min(x, A[Lo..Hi]) \equiv (x \in A[Lo..Hi]) \wedge (x ? A[Lo..Hi])$$

- (b) [5%] Prove that the loop invariant is true after the two initialisation statements have been executed.
- (c) [5%] Prove that the loop body maintains the loop invariant.
- (d) [5%] Prove that the algorithm is correct.

**Question 7.** A *stack* is a data structure which can store values and which provides the following three operations.

- *Create* makes a new, empty stack.
- *Empty* returns true if and only if the stack is empty.
- *Push* adds a new entry to a stack.
- *Pop* retrieves an entry from a stack.

A stack must have the *last-in-first-out* property.

Assume the following Pascal declarations for a stack of characters:

```

const StackSize = 20;
type StackType =
  record
    SP : 0..StackSize;
    Store : array [1..StackSize] of Char
  end;

```

Write Pascal programs for each of the stack operations, using the following signatures.

(a) [5%]

```

procedure Create(var Stack : StackType);

```

(b) [5%]

```

function Empty(Stack : StackType) : Boolean;

```

(c) [5%]

```

procedure Push(var Stack : StackType; Item : Char);

```

(d) [5%]

```

procedure Pop(var Stack : StackType; var Item : Char);

```

Question 8. Here are definitions for a linked list in which each node contains an integer.

```

type
  List = ↑ Node;
  Node =
    record
      Head: integer;
      Tail: List
    end;

```

Using these definitions, write the following functions and procedures.

(a) [5%] The function *Create* has signature

```
function Create : List;
```

and returns an empty list.

(b) [5%] The function *First* has signature

```
function First ( T: List): integer;
```

and returns the first component of the list *T*, assuming that *T* is not empty.

(c) [5%] The function *Find* has signature

```
function Find ( T: List, N: integer): List;
```

and returns a pointer to a node containing *N* if *N* is in the list *T*, and returns the null pointer if *N* is not in the list *T*.

(d) [5%] The procedure *InsertAtStart* has signature

```
procedure InsertAtStart ( N: integer, var T: List );
```

and inserts a node with integer *N* at the start of the list *T*.

# COMP 245

## Final Examination, Fall 1992

Answer FIVE questions only.

**Question 1.** An array of integers  $A[1..n]$  is *sorted in ascending sequence* ("sorted" for short) if, for any integers  $i$  and  $j$  such that  $1 \leq i < j \leq N$ , we have  $A[i] \leq A[j]$ .

- (a) [5%] Show that, according to the definition, the one-element array  $A[1..1]$  is sorted.
- (b) [5%] Give pre- and post-conditions for the statement

insert  $k$  into  $A[i..j]$

such that the program

```
for  $i := 2$  to  $N$  do
  insert  $A[i]$  into  $A[1..i-1]$ 
```

will terminate with the array  $A[1..N]$  sorted.

- (c) [5%] Write a sort procedure based on the pseudocode in part (b).
- (d) [5%] Give the time complexity of your procedure. Use order notation and explain your answer.

### Question 2.

The Pascal type Buffer is defined by

type Buffer = array [ 1..6 ] of Char;

- (a) [8%] Write a procedure  $\text{Convert}(N, B)$  which converts an integer to an array of characters according to the following specification.

**Entry Conditions**  $N$  is an integer and  $0 \leq N \leq 999999$ .  $B$  is of type Buffer.

**Exit Conditions** The buffer  $B$  contains the digits of the decimal representation of  $N$ , right-justified with leading blanks. For example, if  $N = 1357$  then, on exit, the buffer  $B$  contains  $\_ \_ 1357$ , where  $\_$  denotes a blank.

- (b) [12%] A number is a *palindrome* if it reads the same forwards as backwards. Some numbers are both perfect squares and palindromes. For example,  $11^2 = 121$ . Write a program which tests the following hypothesis: if  $0 \leq N \leq 1000$ , then  $N^2$  is a palindrome only if  $N$  is a palindrome.

```

type
  List = ↑Node;
  Node = record
    head : integer;
    tail : List
  end;
function ABC( x, y : List ) : List;
var t : List;
begin
  if x = nil then
    ABC := y
  else begin
    t := x↑.tail;
    x↑.tail := y;
    ABC := ABC( t, x )
  end
end; {ABC}

```

4. This question is about lists of integers. We write  $(m, n, \dots)$  to denote the list containing the integers  $m, n, \dots$  and  $()$  to denote the empty list. Here are suitable type definitions for lists.

```

type
  List = ↑ Node;
  Node =
    record
      Head: integer;
      Tail: List
    end;

```

- (a) Write a Pascal function with the signature

```
function Extend (K: List; E: integer): List;
```

such that the following two equations are satisfied:

$$\text{Extend}((), e) = (e)$$

$$\text{Extend}((e_1, \dots, e_n), e) = (e_1, \dots, e_n, e).$$

- (b) Suppose that  $L$  is the list  $(1, 2, 3, 4)$  and that the function *Mystery* is coded as below. What is the result of computing *Mystery*( $L$ )? What does *Mystery* do in the general case?

```

function Mystery (var K: List): List;
begin
  if K = nil
  then Mystery := nil
  else Mystery := Extend(Mystery(K↑.Tail), K↑.Head)
end;

```

5. The following declarations define a type for lists in which each node contains an integer.

```
type
  ListPtr = ↑ ListNode;
  ListNode =
    record
      IntVal : integer;
      Next : ListPtr
    end
```

- (a) Write a function which finds the largest number in a given non-empty list.
- (b) Assume that  $X$  and  $Y$  are two lists in which the numbers are arranged in ascending order and which contain no duplicates. Write a function which constructs a third list,  $Z$ , which contains all of the numbers in  $X$  and  $Y$  and which has the same properties. For example, if

$$X = (2, 3, 6, 9, 12)$$

and

$$Y = (3, 7, 9, 11)$$

then your function should return

$$Z = (2, 3, 6, 7, 9, 11, 12).$$

6. A set of integers can be represented using a singly linked list of integers organized in ascending order. Design algorithms for the following procedures:

```
function Member( n : integer; S : List ) : boolean;
{ Return the value true if and only if  $n \in S$  }
```

```
function Size( S : List ) : integer;
{ Return the size  $|S|$  of the set  $S$  }
```

```
function Union( S1, S2 : List ) : List;
{ Return a new list which represents  $S1 \cup S2$  }
{ S1 and S2 remain unchanged }
```

```
function Intersect( S1, S2 : List ) : List;
{ Return a new list which represents  $S1 \cap S2$  }
{ S1 and S2 remain unchanged }
```

```
function Difference( S1, S2 : List ) : List;
{ Return a new list which represents  $S1 \setminus S2$  }
{ S1 and S2 remain unchanged }
```

What are the average time complexities of your algorithms?

7. The following declarations define a type for binary trees with an integer at each node.

```
type
  TreePtr = ↑ TreeNode;
  TreeNode =
    record
      IntVal: integer;
      Left, Right: TreePtr
    end;
```

- (a) Write a recursive function with signature

```
function Sum(t : TreePtr) : integer;
```

such that the value of  $Sum(t)$  is the sum of the integers in the nodes of the tree  $t$ . If  $t$  is empty, this value should be zero.

- (b) Write a recursive function with signature

```
function Height(t : TreePtr) : integer;
```

such that the value of  $Height(t)$  is zero if  $t$  is empty, otherwise it is one greater than the maximum heights of the left and right subtrees of  $t$ .

8. The following definitions define a type for binary trees. Each node in a binary tree has a *level*. The level of the root node is 0 and the level of every other node is one greater than the level of its parent.

```
type
  Ptr = ↑ Node;
  Node =
    record
      Val : integer;
      Left, Right : Ptr
    end;
```

- (a) Write a Pascal function  $Copy$  such that, if  $P$  is a pointer to a binary tree, then  $Copy(P)$  returns a new binary tree with the same structure and node values as  $P$ . (A function which simply performs the assignment  $Copy := P$  is not acceptable, because it returns a pointer to the original tree, not a *new* tree.)
- (b) Write a Pascal function  $PartSum$  such that, if  $P$  is a pointer to a binary tree and  $N$  is a natural number, then  $PartSum(P, N)$  returns the sum of all nodes down to and including level  $N$  of tree  $P$ .

COMP 245  
Alternate Final Examination, Winter 1993

Answer FIVE questions only.

Question 1. Here are definitions for a linked list in which each node contains an integer.

```
type
  List = ↑ Node;
  Node =
    record
      Head: integer;
      Tail: List
    end;
```

Using these definitions, write the following functions and procedures.

(a) [5%] The procedure *Create* has signature

```
function Create ( var T: List );
```

and creates an empty list *T*.

(b) [5%] The procedure *Insert* has signature

```
procedure Insert ( var T: List; N: integer );
```

and adds a new node containing the integer *N* to the front of the list *T*.

(c) [10%] The function *Member* has signature

```
function Member ( T: List; N: integer ): boolean;
```

and returns true if and only if *N* occurs as an entry in the list *T*.

**Question 2.** Suppose that  $A[1..N]$  is an array of integers, and no two entries of the array are equal. This question concerns the problem of finding the *second largest* component of  $A$ . For example, if  $A$  contains the values

5 3 -6 9 2 7 6 1

then the second largest value is 7.

Assume the following predicate  $Count(A[Lo..Hi], x, k)$ , which is true if and only if  $k$  equals the number of entries in the array segment  $A[Lo..Hi]$  which are greater than  $x$ .

$$Count(A[Lo..Hi], x, k) \equiv k = \text{size of the set } \{A[j] \mid (Lo \leq j \leq Hi) \wedge (A[j] > x)\}$$

- (a) [5%] What is the shortest length of an array for which it makes sense to define the second largest entry?
- (b) [5%] Using  $Count$ , define a predicate  $Largest(A[Lo..Hi], x)$ , which is true if and only if  $x$  is the largest entry of the array segment  $A[Lo..Hi]$ . Remember to include that  $x$  must be a member of the array segment.
- (c) [5%] Using  $Count$ , define a predicate  $Second(A[Lo..Hi], x)$ , which is true if and only if  $x$  is the second largest entry of the array segment  $A[Lo..Hi]$ . Remember to include that  $x$  must be a member of the array segment.
- (d) [5%] Write an algorithm to find the second largest entry of the array  $A[1..N]$ . The problem must be solved in one scan of the array.

**Question 3.** Suppose that  $B[1..N]$  is an array of integers, which are in no particular order. This question concerns the problem of arranging the entries in the array so that all the odd entries come before all the even entries. For example, if  $B$  when input contains the values

8 5 3 24 -6 9 2 7 6

then  $B$  at the end might be arranged as

7 5 3 9 -6 24 2 8 6

Consider the following outline of an algorithm in pseudocode

```

{ Precondition: ..... }
R := N+1;
L := 1;
{ Invariant I: (1 ≤ L ≤ R ≤ N+1) and
every element of B[1..(L-1)] is odd and
every element of B[R..N] is even }
while L < R do
  if B[L] is odd then
    ...statements S1 ...
  else
    ...statements S2 ...
{ Postcondition: ..... }

```

- (a) [5%] If  $I$  is actually a loop invariant, then what is a suitable postcondition for the algorithm?
- (b) [5%] Supply suitable statements  $S_1$  and  $S_2$  which complete the algorithm and preserve the loop invariant.  $S_1$  and  $S_2$  may actually be sequences of statements.
- (c) [5%] Verify that  $I$  is actually a loop invariant for your completed algorithm.
- (d) [5%] What, if any, is a suitable precondition for the algorithm?

Question 4. Consider the following pseudocode

```

var N : integer;
procedure S( M, x, y, z : integer );
begin
  if M > 0 then begin
    S( M - 1, x, z, y );
    write( x, z, M );
    S( M - 1, y, x, z );
  end;
end;
begin
  read( N );
  S( N, 1, 2, 3 );
end.

```

- (a) [5%] What is the output when  $N$  is 3?
- (b) [5%] How many calls to  $S$  are made when  $N$  is 3?
- (c) [5%] Explain why the algorithm always terminates.
- (d) [5%] How many calls to  $S$  are made for a general value of  $N$ ?