CS 311
Concepts of Programming Languages
(Answer 2 questions out of 3)

Question 1

a) Consider the description of for-loops in Compaq Fortran on the following two pages.

I) Write do-constructs in Compaq Fortran counting an integer variable `K` through the values

(i) 1, 2, 3, 4, 5

(ii) 10, 20, 30, 40, 50

(iii) 50, 40, 30, 20, 10

within the loop (e.g., in the first problem stated above the value of `K` during the first iteration is 1, during the final iteration the value of `K` is 5). Compute the 'iteration count' for your code. (INT is a rounding function into the type integer; it does not change integer values.)

II) Is it possible to write a non-terminating do-construct with a Do variable of type integer? Give references to the respective text lines supporting your answer.

III) Can the Do variable be changed within the iteration? Give a reference to the respective text line.

IV) What happens if you assign the value of the Do variable after leaving the loop? Give a reference to the respective text line.

[15 marks]

b) Let X and Y be sets. Prove the following:

I) $X + Y \cong Y + X$.

II) $X \times Y \cong Y \times X$.

[10 marks]

7.5.2.1 Iteration Loop Control

DO iteration loop control takes the following form:

```
    do-var = expr1, expr2 [, expr3]
```

do-var
Is the name of a scalar variable of type integer or real. It cannot be the name of an array element or structure component.

expr
Is a scalar numeric expression of type integer or real. If it is not the same type as do-var, it is converted to that type.

Rules and Behavior

A DO variable or expression of type real is a deleted feature in Fortran 95; it was obsolescent in Fortran 90. Compaq Fortran fully supports features deleted in Fortran 95.

The following steps are performed in iteration loop control:

1. The expressions expr1, expr2, and expr3 are evaluated to respectively determine the initial, terminal, and increment parameters.

   The increment parameter (expr3) is optional and must not be zero. If an increment parameter is not specified, it is assumed to be of type default integer with a value of 1.

2. The DO variable (do-var) becomes defined with the value of the initial parameter (expr1).

3. The iteration count is determined as follows:

   ```
   MAX(INT((expr2 - expr1 + expr3)/expr3), 0)
   ```

   The iteration count is zero if either of the following is true:

   ```
   expr1 > expr2 and expr3 > 0
   expr1 < expr2 and expr3 < 0
   ```

4. The iteration count is tested. If the iteration count is zero, the loop terminates and the DO construct becomes inactive. If the iteration count is nonzero, the range of the loop is executed.

```
47
48      5. The iteration count is decremented by one, and the DO variable
49         is incremented by the value of the increment parameter, if any.
50
51  After termination, the DO variable retains its last value (the one it
52  had when the iteration count was tested and found to be zero).
53
54  The DO variable must not be redefined or become undefined during
55  execution of the DO range.
56
57  If you change variables in the initial, terminal, or increment
58  expressions during execution of the DO construct, it does not affect
59  the iteration count. The iteration count is fixed each time the DO
60  construct is entered.
61
62  Examples
63
64  The following example specifies 25 iterations:
65
66    DO K=1,50,2
67
68  K=49 during the final iteration, K=51 after the loop.
69
70  The following example specifies 27 iterations:
71
72    DO J=50,-2,-2
73
74  J=-2 during the final iteration, J=-4 after the loop.
75
76  The following example specifies 9 iterations:
77
78    DO NUMBER=5,40,4
79
80  NUMBER=37 during the final iteration, NUMBER=41 after the loop. The
81  terminating statement of this DO loop must be END DO.
```

From: *Compaq Fortran*, Language Reference Manual, Compaq Computer Corporation Houston, Texas, September 1999.

Question 2

a) Draw a diagram showing the lifetime of the variables in the following Pascal program:

```
program P;
  var m: Integer;

  procedure R (n:Integer);
  begin
    if n > 0 then R (n-1)
  end

begin
  R(2)
end.
```

Hint: Note that the procedure `R` is recursively called three times. Note also that the variable `n` is local to `R`.

[5 marks]

b) Write a function abstraction that is not referentially transparent. In which way does the presence of a non referentially transparent function make programs more difficult to understand? Argue with the help of your function abstraction.

[6 marks]

c) Implement different modules that realise a telephone book which stores names (as strings) and telephone numbers (as integers).

(Both levels, the interface and the implementation of the interface, need to be present; BUT it is not necessary to give the details of the procedure and function *bodies* in the implementation part.)

I) Write a module "Telephonebook" as an abstract data type. The operations should include constructors

- `emptybook`,
- `addEntry`, which adds a name together with a telephone number,

and a function

- `search`, which returns the number of a person (in the case that there is no matching entry, the result is 0).

II) Write a module "Telephonebook" as an object class. The operations should include

- `emptybook`,
- `addEntry`, which adds a name together with a telephone number, and
- a function `search`, which returns the number of a person (in the case that there is no matching entry, the result is 0).

[14 marks]

Question 3

a) List 3 different forms of declaration, illustrate each form by an example, and explain what binding your example describes.

[6 marks]

b) Consider the following procedure in Pascal:

```
procedure multiply (var m,n: Integer);
begin
  m := m * n;
  writeln(m,n)
end
```

Note that in Pascal the keyword `var` encodes the definitional parameter mechanism.

Assume that the variable `i` has the value 2 and that the variable `j` has the value 3.

What is printed to the screen by the calls

I) `multiply (i,j)` and

II) `multiply (i,i)` ?

Give a short explanation how you worked out your results.

Now suppose that instead the copy mechanism is used. What is printed then to the screens by the calls

III) `multiply (i,j)` and

IV) `multiply (i,i)` ?

Give a short explanation how you worked out your results.

[8 marks]

...

c) Consider the following implementation fragment that realises stacks of integers in Ada as a generic module:

```
generic
 Capacity: Positive;
package stack_class is
  function is_empty return Boolean;
  function top return Integer;
  procedure push (element: in Integer);
  procedure pop;
end stack_class;

package body stack_class is
  FullStack, EmptyStack: exception;
  type List_type is array (1..Capacity) of Integer;
  type Stacktype is
    record
      list: List_type;
      index: Integer range 0..Capacity;
    end record;
  stck: Stacktype;

 function is_empty return Boolean is
  begin ... end is_empty;

  function top return Integer is
  begin ... end top;

  procedure push (element: in Integer) is
  begin ... end push;

  procedure pop is
  begin ... end pop;

begin
   ...
end stack_class;
```

Write functions `is_empty` and `top` and procedures `pop` and `push` such that

- the call of `pop` and the call of `top` with an empty stack raises an exception `EmtpyStack`, and
- that the call of `push` with a full stack raises an exception `FullStack`.

Add also appropriate exception handlers.

[11 marks]