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

Question 1

a) Give a short definition of three *concepts of programming languages* out of the following list:

Value, Storage, Binding, Abstraction, Encapsulation, Type System, Sequencer.

[6 marks]

b) Write a recursive equation describing binary trees over integers. Give three values which belong to the solution of your equation; give three values which do not belong to the solution of your equation. Argue, for each example, why it is – or why it is not – an element of the solution.

[10 marks]

c) Strings are supported by all modern programming languages, but no consensus has emerged among language designers on how string should be classified. What consequences has it for the programmer to have strings as a primitive types or as a structured types respectively?

Hint: SML has strings as primitive types, while Pascal and Ada see them as array of char.

[9 marks]

Question 2

a) Name three *programming paradigms*. For each paradigm describe the underlying 'abstract machine' and name at least one language following this paradigm.

[6 marks]

b) Explain in terms of examples what it means for a function abstraction to be

i) side-effect free, and

ii) referentially transparent.

[4 marks]

c) Implement different modules of *sets*:

Hint: 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 "sets over integers" as an abstract data type. The operations on sets should include constructors `emptyset`, `addElement`, which adds an integer to a set, and a function `inSet`, which tests if an integer is an element of a set.

ii) Write a module "sets over integers" as an object class. The operations on sets should include `emptyset`, `addElement`, which adds an integer to a set, and a function `inSet`, which tests if an integer is an element of a set.

iii) Turn one of your implementation into a generic module, where the type of the set entries is a parameter.

[15 marks]

Question 3

a) State two *principles of good language design.*

[4 marks]

b) Consider the parameter passing mechanism for function abstractions and procedure abstractions:

i) Explain the copy-mechanism: how can it be used to realise the different *modes* of parameter passing?

ii) Explain how an alias may arise in the context of parameter passing. Why are aliases seen as a problem?

[8 marks]

c) i) Consider the following Ada program:

```
type Month = (January, February, March, April, May, June,
              July, August, September, October, November, December);
type Day is range 1 .. 31;
diary:   array (Month,Day) of DiaryEntry;
keyword: String(1..10);
keydate: Date := <default value>

function matches (x:DiaryEntry; k: String(1..10)) return Boolean is
  ...
end matches

search:
for m in Month loop
  for d in Day loop
    if matches(diary(m,d)),keyword)
      keydate := (m,d)
      exit search,
    end if
  end loop
end loop
```

Rewrite the block "search" without an exit, such that your program immediately stops if the entry has been found. Which program is easier to analyse and easier to understand?

[3 marks]

ii) Consider the following implementation of queues in Ada as generic module:

```
generic
  capacity: in Positive;
  type ITEM is private;
package queue_class is
  function is_empty return Boolean;
  function is_full  return Booean;
  procedure append (i: in Item);
  procedure remove (i: out Item);
end queue_class;

package body queue_class is
  FullQueue, EmptyQueue: exception;
  items: array (0..capacity -1) of ITEM;
  size, front, rear: Integer range -1 .. capacity-1;

  function is_empty return Boolean is ... end is_empty;
  function is_full  return Booean is ... end is_full;
  procedure append (i: in Item) is ... end append;
  procedure remove (i: out Item) is .. end remove;

begin
  front:=0;
  rear:=-1
end
```

Write functions `is_empty` and `is_full` and procedures `remove` and `append` such that

- the call of `remove` with an empty queue raises an exception `EmtpyQueue`, and
- that the call of `append` with a full queue raises an exception `FullQueue`.

[10 marks]