95

University of Swaziland

Department of Computer Science

Final Examination

2015/16

Title of Paper: Programming Languages

Course Number: CS343

Time Allowed: Three (3) hours

Instruction: ***ANSWER ALL QUESTIONS***

You are not allowed to open this paper until you have been told to do so by the invigilator.

Question 1

a) Discuss the following:

   i. Type inference [2]

   ii. Lazy evaluation [2]

   iii. Currying [2]

   iv. Pattern matching [3]

b) Reduce the following λ-calculus expression to normal form:

   i. ((λx.((λy.(x y))x))(λz.w)) [3]

   ii. ((λf.((λg.((f f)g))(λh.(k h))))(λx.(λy.y))) [7]

   iii. (λg.((λf.((λx.(f (x x)))(λx.(f (x x))))) g)) [6]

Question 2

a) Why are computers designed to follow a small set of binary coded instructions instead of instructions given in natural language? [3]

b) Discuss the differences between the following:

   i. Compiler and interpreter [4]

   ii. Syntax and semantics [2]

   iii. Operational semantics and formal semantics [4]

   iv. Axiomatic semantics and denotational semantics [6]

c) For each of the following give examples of how they may be used (NOT type definitions) using a language of your choice:

   i. Collections [3]

   ii. Compounds [3]

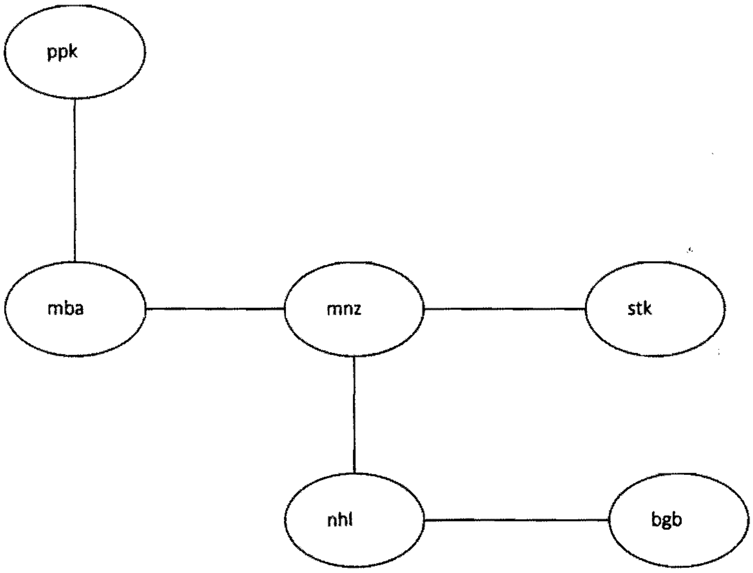
Question 3

a) Discuss the differences between:

i. Imperative and declarative paradigms [6]

ii. Structured and object-oriented paradigms [7]

b)



i. The adjoining map shows 5 roads connecting 6 towns in Swaziland. Represent information about the roads by writing <u>5 Prolog facts</u>, each of the form:

```
road(Town1, Town2)
```

where `Town1` and `Town2` are the two towns connected by the road. [5]

ii. Define a Prolog rule of the form:

```
nearby(Town, Num) :- ...
```

that succeeds when `Num` is the number of towns directly linked to the given `Town`. E.g. based on the above map, the query `nearby(mnz, 3)` must succeed. [7]

Question 4

Write Haskell functions that can perform the following:

a) Write an expression to produce a list of all even integers between 50 and 100, inclusive. [3]

b) Write a function, ***opposite***, that takes a list of strings and returns a new list of strings by adding the prefix "un" to each element of the given list. e.g. ***opposite ["happy", "equal"]*** should return ***["unhappy", "unequal"]***. [7]

c) Write a recursive function, ***absent***, that takes 2 arguments: a string and a list of strings. It should return True if the string is not found in the list, or False otherwise. e.g. ***absent "a" ["aa", "b"]*** should return ***True***. [7]

d) Write a recursive function, ***uniqueStr***, that takes a list of strings and returns a list of the unique elements (i.e. without duplicates). e.g. ***uniqueStr ["a", "aa", "a", "b", "aa"]*** should return ***["a", "aa", "b"]***. [8]

■■■