COMP 141: Midterm Exam Sample Questions

(1) What is von Neumann architecture?

(2) Enumerate the four main programming paradigms and differentiate them.

(3) Define what the syntax and semantics of a PL are.

(4) Explain the functionality of each of the following.

  (a) Compiler
  (b) Assembler
  (c) Linker
  (d) Loader
  (e) Interpreter
  (f) Scanner
  (g) Parser
  (h) Evaluator

(5) Explain what the following criteria for PL design are.

  (a) Readability
  (b) Writability
  (c) Expressiveness
  (d) Reliability

(6) In Pascal, functions can return scalar and pointer types but cannot return structured types like sets, files, and arrays. Lack of which criterion this refers to?

(7) Consider the following CFG, called G_1.

$$\begin{aligned} &\textit{expr} ::= \textit{expr} \ @ \ \textit{expr} \mid \textit{expr} \ \# \ \textit{expr} \mid (\textit{expr}) \mid \texttt{SYMBOL} \\ &\texttt{SYMBOL} = \texttt{A} \mid \texttt{B} \mid \texttt{C} \end{aligned}$$

  (a) Consider the following derivations for the expression `A # B @ C`.

$$\begin{aligned} \textit{expr} &\Rightarrow \textit{expr} \ \# \ \textit{expr} \Rightarrow \texttt{SYMBOL} \ \# \ \textit{expr} \Rightarrow \texttt{A} \ \# \ \textit{expr} \Rightarrow \texttt{A} \ \# \ \textit{expr} \ @ \ \textit{expr} \Rightarrow \texttt{A} \ \# \ \texttt{SYMBOL} \ @ \ \textit{expr} \\ &\Rightarrow \texttt{A} \ \# \ \texttt{B} \ @ \ \textit{expr} \Rightarrow \texttt{A} \ \# \ \texttt{B} \ @ \ \texttt{SYMBOL} \Rightarrow \texttt{A} \ \# \ \texttt{B} \ @ \ \texttt{C} \end{aligned}$$

$$\begin{aligned} \textit{expr} &\Rightarrow \textit{expr} \ @ \ \textit{expr} \Rightarrow \textit{expr} \ \# \ \textit{expr} \ @ \ \textit{expr} \Rightarrow \texttt{SYMBOL} \ \# \ \textit{expr} \ @ \ \textit{expr} \Rightarrow \texttt{A} \ \# \ \textit{expr} \ @ \ \textit{expr} \\ &\Rightarrow \texttt{A} \ \# \ \texttt{SYMBOL} \ @ \ \textit{expr} \Rightarrow \texttt{A} \ \# \ \texttt{B} \ @ \ \textit{expr} \Rightarrow \texttt{A} \ \# \ \texttt{B} \ @ \ \texttt{SYMBOL} \Rightarrow \texttt{A} \ \# \ \texttt{B} \ @ \ \texttt{C} \end{aligned}$$

  Give the parse tree that corresponds to each of the these derivations.

  (b) What are the ASTs for the parse trees given in the previous question?

  (c) Give the disambiguated version of G_1, called G_2, considering the following precedence cascade among the operators:

- The highest precedence is for parentheses,
- the second highest precedence is for `#`,
- the least precedence is for `@`.

(d) Give the single parse tree that can be generated using G_2.

(e) What would be the AST for the parse tree in the previous question?

(f) Redefine G_2 using EBNF.

(8) Conisder the following CFG rule.

$$e ::= e \blacktriangle \texttt{A} \mid \texttt{A}$$

(a) Is ▲ defined as left-recursive or right-recursive?

(b) Redefine the CFG to make ▲ be left-recursive if it is already right-recursive. Redefine the CFG to make ▲ be right-recursive if it is already left-recursive.

(c) For each of the left-recursive and right-recursive definitions in the two previous questions, give the EBNF definition.

(9) Multiplying numbers in a list:

(a) Define a recursive function in Haskell that receives a list of integers and multiplies the elements together. (You may assume that if the list is empty, the result is 1).

(b) What would be the inferred type of this function?

(10) Define a function `cprod` in Haskell that receives three lists in input and returns the Cartesian product of them. For example, `cprod [1,2] ["a","c"] [True]` must return `[(1,"a", True), (1,"b",True), (2,"a",True), (2,"b",True)]`. *Hint*: Use list comprehension.

(11) Give an example where strict and lazy evaluations end in different results.