CS 571
Quiz 1 Annotated Solution

Sep 19 **Closed book**
15 points **Closed notes**

Important Reminder: As per the course Academic Honesty Statement, cheating of any kind will minimally result in receiving an F letter grade for the entire course.

For each of the following questions, select a **single** alternative on the grid-sheet. Please ensure that you have filled-in your name in the bubbles on the provided grid-sheet; the B-number is not required.

There are 7 questions with 2-points per question; there is 1-point for submitting the quiz.

1. Which of the following statements about standard regular expression notation is false?

   (a) The infix `.` binary operator is used to denote the concatenation of two regular expressions.

   (b) The infix `|` binary operator is used to denote the alternation of two regular expressions.

   (c) The postfix `*` unary operator is used to denote 0-or-more repetitions of a regular expression.

   (d) The postfix `?` unary operator is used to denote an optional regular expression.

   (e) The postfix `+` unary operator is used to denote 1-or-more repetitions of a regular expression.

Answer: (a).

Standard regular expression uses the juxtaposition of two regular expressions to denote concatenation.

2. Which of the following programming languages is very different from the others in terms of syntax?

   (a) Algol

   (b) Pascal

   (c) Lisp

   (d) C

   (e) C++

   Answer: (c).

   Lisp uses a parenthesized prefix notation whereas all the other languages use a more traditional syntax.

3. Which of the following regular expressions describes strings of one-or-more `a`'s followed **optionally** by a single `b` followed by zero-or-more `a`'s (ignore whitespace added for readability within each regex)?

   (a) `a? b* a+`

   (b) `a a* b? a*`

   (c) `a* b? a* a`

   (d) `a+ b? a+`

   (e) `a* b+ a+`

   Answer: (b).

   (b) consists of one `a` followed by 0-or-more `a`'s followed by an optional `b` followed by 0-or-more `a`'s which is clearly what is required. The other alternatives are incorrect: (a) only allows a prefix of 0-or-1 `a`'s; (c) allows a prefix of 0-or-more `a`'s; (d) requires a suffix of one-or-more `a`'s; (e) requires one-or-more `b`'s.

4. Which of the following languages over the vocabulary {`a`, `b`} is not expressible using standard regular expressions?

(a) Strings whose length is exactly 5. Examples include aabab, babab and aaaaa.

(b) Strings whose length must be a multiple of 3. Examples include the empty string, aba, aabbab.

(c) Strings of length less-than-or-equal-to 8 which consist of a sequence of a's followed by an equal number of b's. Examples include the empty string, ab and aaabbb.

(d) Strings of arbitrary length containing 1-or-more a's followed by 0-or-more b's. Examples include aaa, a and abbb.

(e) Strings of arbitrary length which consist of a sequence of a's followed by an equal number of b's. Examples include the empty string, ab and aaaabbbb.

Answer: (e).

(e) requires that a number of a's be balanced using the same number of b's but regular expressions cannot be used to specify arbitrary balanced constructs. All the other alternatives can be expressed using regexps's: `[ab][ab][ab][ab][ab]` for (a), `([ab][ab][ab])*` for (b), `|ab|aabb|aaabbb|aaaabbbb` for (c) and `a+b*` for (d).

5. Given the following CFG over the set of terminal symbols NUMBER, #, ^, !, (and):

```
exp
  : exp '#' term
  | term
  ;
term
  : factor '^' term
  | factor
  ;
factor
  : factor '!'
  | '(' exp ')'
  | NUMBER
  ;
```

Which of the following statements about the precedence and associativity of the operators `#`, `^`, and `!` is true?

(a) `^` has lowest precedence, followed by `#` with higher precedence, followed by `!` with highest precedence. `#` is right-associative, while `^` is left-associative.

(b) `#` has lowest precedence, followed by `^` with higher precedence, followed by `!` with highest precedence. `#` is left-associative, while `^` is right-associative.

(c) `!` has lowest precedence, followed by `^` with higher precedence, followed by `#` with highest precedence. `#` is left-associative, while `^` is right-associative.

(d) `#` has lowest precedence, followed by `^` with higher precedence, followed by `!` with highest precedence. `#` is right-associative, while `^` is left-associative.

(e) `!` has lowest precedence, followed by `^` with higher precedence, followed by `#` with highest precedence. `#` is right-associative, while `^` is left-associative.

Answer: (b).

The operators occur in the grammar in order `#`, `^`, `!` with `#` closest to the start symbol. Hence the precedences increase in the order `#`, `^`, `!`. Since `#` occurs in a left-recursive rule it is left-associative, while `^` is right-associative since it occurs in a right-recursive rule.

6. Which of the following describes the language consisting of n `a`'s followed by exactly n `b`'s for $n >= 0$?

   (a) The regular expression `a*b*`.

   (b) The CFG:

   ```
   S
     : 'a' S 'b'
     | //empty
     ;
   ```

   (c) The regular expression `a+b+`.

(d) The CFG:

```
S
  : 'a' S 'b'
  | 'a' 'b'
  ;
```

(e) The CFG:

```
S
  : 'a' 'a' S 'b' 'b'
  | //empty
  ;
```

Answer: (b).

In (b), the 2nd alternative allows for the empty string; the first alternative allows some number of `a`'s followed by an equal number of `b`'s.

(a) and (c) cannot be acceptable since regexp's cannot describe balanced constructs of arbitrary length, (d) does not allow $n = 0$ while (e) requires even n.

7. Which of the following statements is false?

   (a) A *scanner* converts a stream of characters into a stream of tokens.

   (b) A recursive-descent parser must have a parsing function for each non-terminal in the grammar.

   (c) The stack frame for a function activation will typically contain the return address for that activation.

   (d) If a grammar permits a derivation containing a step with ambiguity about which non-terminal should be expanded next, then the grammar is defined to be *ambiguous*.

   (e) The `match()` function of a recursive-descent parser must match the current terminal, else signal an error.

   Answer: (d).

   An ambiguous grammar is one which allows multiple parse trees. It is perfectly normal for derivations to permit multiple non-terminals at any step; for a non-ambiguous grammar the same parse tree will result irrespective of the order in which those non-terminals are expanded.