CS 571
Quiz 2 Annotated Solution

Oct 5 **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 and B-number in the bubbles on the provided grid-sheet.

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

1. Which of the following languages over the vocabulary of parentheses {`(`, `)`} is not expressible using standard regular expressions?

   (a) Strings whose length is exactly 3. Examples include `)((`, `)()` and `())`.

   (b) Strings of even length. Examples include the empty string, `((`, `()()` and `(())`.

   (c) Strings of length less-than-or-equal-to 4 which consist of balanced parentheses. Examples include the empty string, `()` and `()()`.

   (d) Strings of even length containing 1-or-more `(`'s followed by 0-or-more `)`'s. Examples include `((`, `()))` and `(())`.

   (e) Strings of even length which consist of balanced parentheses. Examples include the empty string, `()` and `)(())`.

Answer: (e).

Regular expressions cannot describe arbitrary balanced constructs. Hence (e) is not expressible. (a) and (c) can be described by a regex which simply enumerates all possibilities. (b) can be described by the regex `( \(\( | \(\) | \)\( | \)\) )*` and (d) can be described by the regex `\( ( \(\( )* ( \(|\) ) ( \)\) )*`.

2. Given the following CFG over the set of terminals {`NUM`, `ID`, `','`, `';'`}:

```
list
  : ID tail
  ;
tail
  : ',' NUM tail
  | ',' ID tail
  | /* empty */
  | ';'
  ;
```

Which one of the following describes the language defined by the above CFG most precisely?

(a) Lists of `ID`'s and `NUM`'s separated by `','` and terminated by `';'`.

(b) Lists of `ID`'s and `NUM`'s starting with a `ID`, separated by `','` and optionally terminated by `';'`.

(c) Lists of `ID`'s and `NUM`'s separated by `','`.

(d) Lists of `ID`'s and `NUM`'s separated by `','` and optionally terminated by `';'`.

(e) Lists of `ID`'s and `NUM`'s starting with a `ID`, separated by `','` and terminated by `';'`.

Answer: (b)

The rule for `list` requires all sentences in the language to start with `ID`. The first two rules for `tail` allow 0-or-more additional `ID`'s or `NUM`'s preceeded by `,`. The last two rules for `tail` have the list terminated by an optional `;`.

3. In Javascript, *hoisting* refers to:

   (a) Moving `var` declarations to the start of a block.

   (b) Moving `var` declarations to the start of a function.

   (c) Moving `var` declarations to the `window` object.

   (d) Moving `let` declarations to the start of a block.

   (e) Moving `let` declarations to the start of a function.

   Answer: (b).

   This is the standard definition of the term *hoisting* within the Javascript community. This behavior can surprise programmers coming to Javascript from a language where variables have block scope.

4. Heap allocation is the only alternative for

   (a) Entities having a lifetime equal to that of the entire program.

   (b) Entities having a lifetime equal to that of a function activation.

   (c) Entities having a lifetime equal to that of a block activation.

   (d) Entities which are function parameters.

   (e) Entities which have indeterminate lifetime.

   Answer: (e).

   Static allocation can be used for (a). Stack allocation can be used for (b), (c) and (d). However, if an entity has indeterminate lifetime then heap allocation is the only alternative.

5. Temporary variables within a stack frame refer to

   (a) Variables declared to be temporary by the programmer.

   (b) Variables introduced by the programmer to implement the exchange of the values of two variables.

   (c) Variables which contain references to heap data.

   (d) Variables introduced by the compiler to hold the variables of a function while it is active.

(e) Variables introduced by the compiler to hold intermediate values computed while evaluating an expression.

Answer: (e).

6. Assuming that all declarations introduced using `var` in the following pseudo-code are dynamically scoped, what will be the output of the following program?

```
var a = 11;

f() { var a = 22; h(); }

h() { print a; }

print a; f(); print a;
```

(a) `11 11 11`
(b) `11 22 11`
(c) `11 22 22`
(d) `11 11 22`
(e) `22 11 22`

Answer: (b).

The first print prints out the value `a` defined at the top-level. The second print within `h()` prints out the dynamic value of `a` which is defined as `22` within the activation of `f()` which called `h()`. The final print again prints out the top-level value.

7. Which one of the following statements is false?

(a) Scheme is dynamically scoped.
(b) In some languages, there can be *holes* within the scope of a variable.
(c) A context-free grammar can describe nested constructs.
(d) In C, the declaration `struct S;` can be used to make a forward-reference to a structure type.

(e) Static allocation cannot be the sole allocation strategy in the presence of recursive functions.

Answer: (a).

Scheme is lexically scoped.

Languages with nested blocks which allow inner blocks to redeclare variables declared in outer blocks can have scopes with holes. A CFG can indeed describe nested constructs using rules like $A \rightarrow AXA$. In C, `struct S;` can be used for a forward-declaration; in fact, this can be used for implementing ADT's in C. With only static allocation, recursive functions are not possible since there is no storage to restore the previous values of the variables within a function after a recursive call.