CSE401 Midterm, October 24, 2008

- Open book, open note
- Closed electronics, closed neighbor
- 50 minutes
- Put your name on every page (at the top)
- Legibility is a plus – if I can't read your answer, I won't try to read your mind

- *Don't open until you are told to*

1. (20 points total, 4 points/question) True or false – and briefly (about one sentence) explain your answer
   a. Natural language (e.g., English) can be accurately parsed using a context-sensitive grammar.

   b. In some compilers, the scanning phase finishes completely before the parsing phase is invoked.

   c. Given a context-free concrete syntax, there is precisely one corresponding abstract syntax for the grammar.

   d. Given a non-deterministic finite state automata and an equivalent deterministic finite state automata, the deterministic one has more states.

   e. A symbol table contains the run-time values of variables defined in the associated scope.

2. (30 points, 10 points each) The following language features are from real languages. Explain which of the three phases – lexing, parsing, semantic analysis/typechecking – is most directly affected by these features, why and in what way.

   a. Dynamic scoping – references to variables not declared directly in scope are associated with the most recent binding made during program execution.

   b. Procedures in most languages accept multiple parameters but only allow at most one return value. A few languages allow multiple return values, usually using a form like
   `[a, b, c] = f ()`
   where function **`f`** returns three values, **`a`**, **`b`** and **`c`**

   c. C is among the languages that depend heavily on a powerful preprocessor, which takes as input a mixture of C and preprocessor commands (e.g., **`#define`**, **`#include`**) and produce as output "naked" C with all preprocessor commands expanded. Examples of use of the preprocessor include defining macros such as
   `#define min(X, Y)  ((X) < (Y) ? (X) : (Y))`[1]
   which allows **`min(X,Y)`** to be used anywhere in the program, with the preprocessor expanding every use for later compilation.
   `min(a,b)` is expanded to **`((a) < (b) ? (a) : (b))`** and
   `min(f(v),g(w))` is expanded to **`((f(v)) < (g(w)) ? (f(v)) : (g(w)))`**

[1] **`bool ? exp1 : exp2`** is (roughly) C syntax for an expression that evaluates **`bool`** and if it's true returns **`exp1`** and otherwise returns **`exp2`**

3. (35 points total) Consider the following bizarre programming language, Whitespace, in which all non-whitespace characters are ignored and only spaces **`[Space]`**, tabs **`[Tab]`** and linefeeds **`[LF]`** are considered syntax. (http://compsoc.dur.ac.uk/whitespace/index.php for your later pain or pleasure, if you wish.)

   - Whitespace is a stack-based language that supports arbitrary length binary integers that are represented as a series of **`[Space]`** and **`[Tab]`**, terminated by a **`[LF]`**. **`[Space]`** represents the binary digit 0, **`[Tab]`** represents 1. The sign of a number is given by its first character, **`[Space]`** for positive and **`[Tab]`** for negative.
     - For example, a positive 2 (decimal) would be represented as
       `[Space][Tab][Space][LF]`
   - The instruction set consists of stack manipulations operations (these start with a **`[Space]`**), arithmetic operations (these start with a **`[Tab][Space]`**), heap access (starting with a **`[Tab][Tab]`**), flow control (starting with **`[LF]`**), and I/O (starting with **`[Tab][LF]`**).
     - For example, a push operation is represented by
       `[Space][Space]`
       so pushing a 2 onto the stack would be represented by
       `[Space][Space] [Space][Tab][Space][LF]`
     - For example, a jump to Label if the top of the stack is zero would be represented by
       `[Tab][Space]` *`Label`*
       where *`Label`* designates an instruction that is marked by *`Label`*, which is an integer defined using a **Mark** instruction
       `[Space][Space]` *`Label`*

a. (20 points total) From a compiler's point of view, concisely but clearly describe two distinct issues that would affect the scanner and the parser – in particular, consider the lexical and syntactic structures of the language?

b. (5 points) Concisely argue whether a top-down or bottom-up parser would be more appropriate for parsing Whitespace.

c. (5 points) Concisely describe one run-time error that can happen during execution of a Whitespace program.

d. (5 points) Concisely describe a semantic check that could be useful for Whitespace.

4. (15 points total) Consider the following grammar, which defines a language that contains all odd-length palindromes over the alphabet **a, b, c** where **c** appears only as the middle character.

```
[0] S ::= P
[1] P ::= a P a
[2] P ::= b P b
[3] P ::= c
```

The following is a largely correct – exactly two entries are incorrect – shift-reduce table for this grammar.

State	Input Symbol				Goto
	a	b	c	$	P
0	S2	S3	S4		2
1				accept	
2	S2	S3	S4		5
3	S2	S3	S4		6
4	R3	R3			
5		S8			
6	S7				
7	R1	R1	R1	R1	
8	R2	R2	R2	R2	

Consider parsing this input: abcba$ A partial initial trace of the shift-reduce process is:

• S0
S0 a • S2
S0 a S2 b • S3
S0 a S2 b S3 c • S4
S0 a S2 b • S3 P
S0 a S2 b • S6 P

a. (10 points) Which two entries in the table are incorrect, and how can you correct them? (Hint: continuing the parsing process will help you identify them.)

b. (5 points) Finish the shift-reduce process with the fixed table.