1	2	3	4	5	6	7	Total

Name:

SI 413 Fall 2013: Midterm Exam

There are seven problems on this exam. Each problem starts on a separate page. Keep in mind that many problems have multiple parts. You must complete all seven problems. Do your best in the time allotted.

Try to keep your answers to each problem in the boxes provided. If you need more space for any problem, you may work on the back of the page, but be sure to *indicate clearly that you have done so*.

If you aren't sure about an answer, you are always free to add more explanation in an attempt at partial credit. But don't waste all your time on a single problem!

(The space below is reserved for any bribes or humorous drawings that will curry favor with your professor.)

Name:

Problem 1

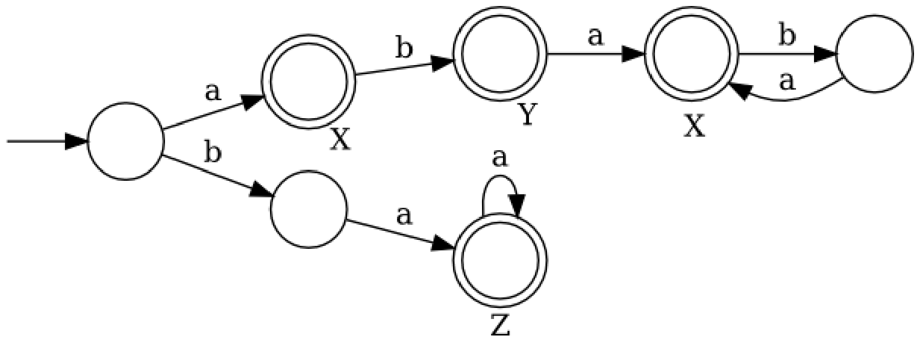
What are the main stages of compilation, and what does each stage produce? (Fill in the 7 blanks in the table below. If you don't remember the "standard name" for something, you can describe it in words for some partial credit.)

Compilation Stage	Output from that stage
(initial)	Source code
	Compiled executable

Name:

Problem 2

Here is a DFA for a scanner. The only valid characters are `a` and `b`, and the three possible types of tokens are called X, Y, and Z. Assume that there is a trap state (not drawn) that all unspecified transitions go to.



(a) Indicate how the following input would be tokenized by this scanner:

`a b a b a a b b a`

(b) Write a scanner specification (regular expressions for each token type) that could have generated the DFA above.

Name:

Problem 3

Consider the following grammar with tokens `U`, `V`, and `W`:

$$\begin{aligned} \mathit{start} &\to \texttt{U}\ p\ \texttt{V} \\ p &\to p\ q \\ p &\to q \\ q &\to \texttt{W}\ \texttt{W} \end{aligned}$$

(a) Fill in the following table with the PREDICT and FOLLOW sets for this grammar.

	PREDICT	FOLLOW
start		
p		
q		

(b) Is this grammar LL(1)? Why or why not?

Name:

Problem 4

Consider the following grammar with tokens `X`, `Y`, and `Z`:

$$
\begin{array}{rcl}
\mathit{start} & \rightarrow & a \\
a & \rightarrow & b \\
a & \rightarrow & a\ b \\
b & \rightarrow & \mathtt{X} \\
b & \rightarrow & \mathtt{Y}\ a\ \mathtt{Z}
\end{array}
$$

(a) Drawn below is a state in the CFSM for this grammar. Draw all the transitions out of this state, and the states to which those transitions go.
(You do *not* need to continue on and build the entire CFSM.)

$$
\boxed{\begin{array}{rl}
\mathit{start} \rightarrow & a\ \bullet \\
a \rightarrow & a\ \bullet\ b \\
b \rightarrow & \bullet\ \mathtt{X} \\
b \rightarrow & \bullet\ \mathtt{Y}\ a\ \mathtt{Z}
\end{array}}
$$

(continued on the next page)

(b) What kind of a conflict does the *original* CFSM state from part (a) represent?

(c) Can this conflict be resolved with 1 token of look-ahead? Explain why or why not.

Name:

Problem 5

Here's the same grammar from Problem 4 again:

$$\begin{aligned} \textit{start} &\to a \\ a &\to b \\ a &\to a\ b \\ b &\to \texttt{X} \\ b &\to \texttt{Y}\ a\ \texttt{Z} \end{aligned}$$

I want you to parse the following token stream bottom-up: **X Y X Z**.

There are a total of 11 "parsing moves", each either shift or reduce. For each one, circle shift or reduce and show the stack of tokens and non-terminals at that step. On the right, draw the total parse tree that results. The first 4 moves are filled in for you.

	circle one	Stack contents
1	shift reduce	
2	shift reduce	
3	shift reduce	
4	shift reduce	
5	shift reduce	
6	shift reduce	
7	shift reduce	
8	shift reduce	
9	shift reduce	
10	shift reduce	
11	shift reduce	

The tree:

Name:

Problem 6

Below is a program in a language just like C, except that:

- The only operators are, in order of precedence from highest to lowest, `~`, `$`, and `?`.
- These are all binary operators.
- Operators `~` and `?` are right-associative, while `$` is left-associative.

Draw the AST for the following program:

```
while (1 $ x $ 2) {
  x = x ? 4 ~ 5 $ 6 ? 7;
}
```

Name:

Problem 7

(a) In scanning the string "`x--y`" in C/C++/Java, the "`--`" is a single token rather than two "`-`" tokens. Why? AND what is the name for this?

(b) How is the notion of *look-ahead* used differently in scanners, recursive descent parsers, and SLR parsers?

(c) Say I have a programming language that can be parsed either by an LL(1) recursive descent parser or an SLR(0) automatically-generated parser. List one advantage of the LL(1) parser and one advantage of going with the SLR(0) parser.

(d) What does it mean to say that functions are "first-class" objects in Scheme? Give an example to show that Java does not share this property.