<empty page to realign pages, flip-over to start worksheet Q1>

CSE 401 - LR Parsing Worksheet - Week 3

Problem 1

a. Using the technique shown in lecture, construct the LR(0) state machine for this grammar. Remember to show the set of items that correspond to each state, including both any initial items and the resulting closure.

```
0. S’::= S $
1. S ::= a Z
2. S ::= b
3. Z ::= a
4. Z ::= b S
```

b. Based on your state machine, build the corresponding LR(0) parse table. Start by filling in the ACTION and GOTO headers with the grammar's terminals and non-terminals, respectively, then give each state in your state machine a number and use it to fill out one row of the table.

STATE	**ACTION**	**GOTO**

c. Finally, use your table to parse the provided input, keeping track of both the stack and the remaining input at each step in a table such as the one below. For clarity, you will probably find it easiest to push both the current state and the corresponding symbol onto the stack at each point, although a real parser would only need to keep track of the states. The initial state (s_1) has already been inserted onto the stack for you.

STACK	**INPUT**	**ACTION**
`$ 1`	`a b a b b $`	

Problem 2

a. Using the technique shown in lecture, construct the LR(0) state machine for this grammar. Remember to show the set of items that correspond to each state, including both any initial items and the resulting closure.

```
0. S'::= S $
1. S ::= a X a
2. S ::= b X
3. X ::= c
4. X ::= S c
```

b. Based on your state machine, build the corresponding LR(0) parse table. Start by filling in the ACTION and GOTO headers with the grammar's terminals and non-terminals, respectively, then give each state in your state machine a number and use it to fill out one row of the table.

STATE	**ACTION**	**GOTO**

c. Finally, use your table to parse the provided input, keeping track of both the stack and the remaining input at each step in a table such as the one below. For clarity, you will probably find it easiest to push both the current state and the corresponding symbol onto the stack at each point, although a real parser would only need to keep track of the states. The initial state (s_1) has already been inserted onto the stack for you.

STACK	**INPUT**	**ACTION**
`$ 1`	`a b c c a $`	

Problem 3

a. Using the technique shown in lecture, construct the LR(0) state machine for this grammar. Remember to show the set of items that correspond to each state, including both any initial items and the resulting closure.

```
0. S'::= S $
1. S ::= ( E )
2. E ::= E && V
3. E ::= V
4. V ::= ! ( E )
5. V ::= a
```

b. Based on your state machine, build the corresponding LR(0) parse table. Start by filling in the ACTION and GOTO headers with the grammar's terminals and non-terminals, respectively, then give each state in your state machine a number and use it to fill out one row of the table.

STATE	ACTION	GOTO

c. Finally, use your table to parse the provided input, keeping track of both the stack and the remaining input at each step in a table such as the one below. For clarity, you will probably find it easiest to push both the current state and the corresponding symbol onto the stack at each point, although a real parser would only need to keep track of the states. The initial state (s_1) has already been inserted onto the stack for you.

STACK	INPUT	ACTION
`$ 1`	`( ! ( a && a ) ) $`	