PART 1

Write a valid SPLaFA program with at least 20 statements. The size of the window should be no bigger than 300 by 300.

PART 2

The purpose of this assignment is to write a parser for the SPLaFA programming language (see the project 1 handout for a description of the tokens in the SPLaFA programming language). Your program will read in a data file containing a SPLaFA program, and will determine if it is a syntactically correct SPLaFA program using an SLR(1) parser. In project 3 you will continue to build on this project, writing an interpreter.

The SPLaFA programming language has a program definition (shown first) and five types of statements:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>size $i , j$ [ stmts ]</td>
<td>program definition - defines size of animation window (width $i$, height $j$)</td>
</tr>
<tr>
<td>$v = a$ ;</td>
<td>an assignment statement</td>
</tr>
<tr>
<td>line $v , a , b , c , d$ color ;</td>
<td>draw a line from point $(a, b)$ to point $(c, d)$</td>
</tr>
<tr>
<td>rect $v , a , b , c , d$ color ;</td>
<td>draw a rectangle with top left corner at point $(a, b)$, width $c$ and height $d$</td>
</tr>
<tr>
<td>move $v , a , b$ ;</td>
<td>move $v$ $a$ spaces in the x-direction and $b$ spaces in the y-direction</td>
</tr>
<tr>
<td>for $v = b$ to $c$ by $d$ do stmts end ;</td>
<td>assign $v$ the value $b$, (<em>) execute stmts, add $d$ to $v$, if $v \leq c$ then repeat starting at (</em>)</td>
</tr>
</tbody>
</table>

where $v$ is a variable, $a$, $b$, $c$, and $d$ are either variables or integers, $i$ and $j$ are integers, color is one of 8 valid colors, and stmts represent 1 or more valid statements.
CFG for the SPLaFA Programming Language

(1)  <Program> → <Sizestmt> [ <List> ]
(2)  <Sizestmt> → size int int
(3)  <List> → <Statement> ;
(4)  <List> → <List> <Statement> ;
(5)  <Statement> → line var <Type> <Type> <Type> <Type> color
(6)  <Statement> → rect var <Type> <Type> <Type> <Type> color
(7)  <Statement> → move var <Type> <Type>
(8)  <Statement> → for var = <Type> to <Type> by <Type> do <List> end
(9)  <Statement> → var = <Type>
(10) <Type> → var
(11) <Type> → int

where “var” represents a variable and “int” represents an integer. The productions are numbered.
DESCRIPTION OF YOUR PROGRAM

Given a SPLaFA program, your task is to 1) scan the program and identify all its parts (or tokens) and 2) parse the program using an SLR parser and identify if it is syntactically correct. If it is, then produce a list of rules that will produce a rightmost derivation (in reverse) of the program.

Part 1 - The Scanner

The purpose of the scanner is to find the next token in your program, enter its value into the symbol table (a data structure that handles searches and insertions), and return 1) the location of the tokens value in the symbol table, and 2) a unique symbol, called the token type, which indicates the type of the token. If a value already exists in the data structure, don’t reenter it. This part was done in Project 1.

Part 2 - The Parser

You are to write an SLR(1) parser to produce rightmost derivations of SPLaFA programs. The parser (called the driver in project 1) will call the scanner whenever it needs the next token in the SPLaFA program. The token type will be shifted onto the SLR parsing stack. The location of the token’s value in the symbol table will be ignored for now. It will be used in project 3.

An SLR(1) Parse Table for the SPLaFA programming language and its associated transition diagram are attached to this handout. The parse table can be stored as a two-dimensional array. The columns are labeled by symbols in the grammar (both terminals and nonterminals) and an end-of-string marker ($) (in this case, the end-of-string marker represents the end of a SPLaFA program or end-of-file marker). The rows are labeled by state numbers from the transition diagram. Each entry in the array represents one of four actions. There may be additional information stored in the entry.

Actions:

- ERROR - The SPLaFA program is not syntactically correct.
- ACCEPT - The SPLaFA program is syntactically correct.
- SHIFT - Shift the input symbol (lookahead) and state number onto the stack. A state number must be stored in this array entry.
- REDUCE - Replace the righthand side (rhs) of the rewrite rule that is on top of the stack with its lefthand side (lhs). You might want to store some representation of the rule in this array entry.

The file to create the SLR(1) Parse Table in this handout is called parsedata and is available on the assignment page web site. You may read in this file in front of a SPLaFA program to create the data entries in the parse table. The format of the file is: 1 row of headers for the terminals, 47 rows of entries, 1 row of headers for the variables, and 47 rows of entries. Each row in the table first has the number of the row, except the header rows, which have
no entry. Items are separated by “&”. If there is no entry in a column, then there will be two adjacent &’s.

An SLR(1) parser when applied to a string in the language it represents will produce a rightmost derivation (in reverse order) of the string. In order to list the rules in the order they would be used in a rightmost derivation (starting with the start symbol), the rules must be stacked. Whenever a REDUCE action is encountered in the parse table, store the rule on a rule stack (this is a different stack than the parsing stack). When the starting rule is encountered, print all the rules on the stack. Thus, for each SPLaFA program that is syntactically correct, print the production rules that would derive the SPLaFA program. Whenever there is an error, you do not need to show the rules encountered, but you can if you want.

The parsing routine accesses a parser stack. Terminals and variables from the grammar, and state numbers can appear on this stack.

Consider the following SPLaFA program.

```plaintext
size 300 200 [
    rect fred 10 20 40 70 red ;
    move fred 10 0 ;
]
```

This SPLaFA program can be derived by applying the following production rules (using the first letter of each variable except using Z for Sizestmt):

<table>
<thead>
<tr>
<th>RULES</th>
<th>DERIVATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>P → Z [ L ]</td>
<td>Z [ L ]</td>
</tr>
<tr>
<td>L → L S ;</td>
<td>Z [ L S ; ]</td>
</tr>
<tr>
<td>S → move var T T</td>
<td>Z [ L move var T T ; ]</td>
</tr>
<tr>
<td>T → int</td>
<td>Z [ L move var T 0 ; ]</td>
</tr>
<tr>
<td>T → int</td>
<td>Z [ L move var 10 0 ; ]</td>
</tr>
<tr>
<td>L → S ;</td>
<td>Z [ S ; move var 10 0 ; ]</td>
</tr>
<tr>
<td>S → rect var T T T color</td>
<td>Z [ rect fred T T T T red ; move fred 10 0 ; ]</td>
</tr>
<tr>
<td>T → int</td>
<td>Z [ rect fred T T T 70 red ; move fred 10 0 ; ]</td>
</tr>
<tr>
<td>T → int</td>
<td>Z [ rect fred T T 40 70 red ; move fred 10 0 ; ]</td>
</tr>
<tr>
<td>T → int</td>
<td>Z [ rect fred T 20 40 70 red ; move fred 10 0 ; ]</td>
</tr>
<tr>
<td>T → int</td>
<td>Z [ rect fred 10 20 40 70 red ; move fred 10 0 ; ]</td>
</tr>
<tr>
<td>Z → size int int</td>
<td>size 300 200 [rect fred 10 20 40 70 red ; move fred 10 0 ; ]</td>
</tr>
</tbody>
</table>

Note: An SLR parser will generate these rules in reverse order.

**INPUT:**

The format of the data file is the same as it was in project 1.
A data file consists of one SPLaFA program. You may assume that SPLaFA programs contain valid tokens. Sample data files will be available on the assignment page web site. In addition you should create your own data files for testing.

OUTPUT:

For each syntactically correct SPLaFA program, you should list the production rules that form a rightmost derivation of the program (starting with the $P$ rule, show the last rule found first, i.e. show the rules in the order they appear in the previous example (just the rules, you do not need to show the derivation)). Indicate whether or not the SPLaFA program is syntactically correct. If it is not correct, then do not show the rules.

THE PROGRAM AND ITS SUBMISSION

Your program should be written in Java 5 or C++ and use Eclipse.

Your program will be graded on style as well as content. Style will count for 20% of your grade.

Appropriate style for this course includes:

- **Modularity** - Your program should be divided into classes. Comments should be included for each method to explain the purpose of the method.
- **Liberal use of comments** - In addition to the comment for each method, each nontrivial section of code (for example a loop) should have a comment describing its purpose. Comments should not merely echo the code.
- **Readability** - Your program should use the indentation and spacing appropriately to make it easily readable. Your comments should be clearly distinguishable from the code.
- **Appropriate variable names** - Give variables names that describe their function.
- **Understandable output** - Your program should indicate its input as well as its output in a clear and readable manner. Remember, the output from your program is the only indication that it works!

The remaining of your grade is based on meeting the specifications of the assignment. If you do not get your program correctly running, for partial credit you may generate output that identifies which part of your program works correctly. This output must also be clearly understandable or no credit will be given!

Include a README file that describes how to run your program. Your program should work for several data files, so you may want to prompt the user to type in the name of the data file.

Name your SPLaFA program with yourInitials.splafa. Submit Part 1 under `project2part1` and submit your program with Eclipse and Ambient under CompSci 140 `project2part2`.

Programs should be submitted by midnight (23:59:59) on the due date.
LATE PENALTIES

Project 2 can be turned in by midnight on the following days with the following penalties:

- Thursday, March 26 3 pts off
- Thursday, April 2 6 pts off

See me immediately if the project is not complete by April 4!

Grammar for SPLaFA: In shorter notation (use first symbol of each variable or terminal except use Z for Sizestmt), and adding a new start symbol:

1. $P' \rightarrow P$
2. $P \rightarrow Z[L]$
3. $Z \rightarrow \text{s i i}$
4. $L \rightarrow S;$
5. $L \rightarrow LS;$
6. $S \rightarrow lvTTTc$
7. $S \rightarrow rvTTTc$
8. $S \rightarrow mvTT$
9. $S \rightarrow f v = T t T b T d L e$
10. $S \rightarrow v = T$
11. $T \rightarrow v$
12. $T \rightarrow i$