CMSC 129
Programming Exercises – Non-Recursive Predictive Parsing

I. Objective

To implement a Non-Recursive Predictive Parser that checks if the string of tokens is valid based on the syntactic rules given.

II. Major Program Details

1. The program must read a production file (***.prod*** extension) containing the list of productions for the grammar.
    a. The first column is the line number of a particular production.
    b. The second column is the list of all the non-terminal symbols.
    c. The third column is the production for the corresponding non-terminal symbol.
    d. Instead of using the OR symbol "|" for non-terminals having two or more productions, the productions are now placed on separate lines.
    e. Each symbol in the production is separated by at least one horizontal space. This will make the process of reading the file easier for you.
    f. The empty symbol is represented by the symbol e.
2. It must also read a parse table file (***.ptbl*** extension) containing the parse table content for the grammar.
    a. The first column is the list of all the non-terminal symbols.
    b. The second column and each succeeding columns are labeled individually with all the terminal symbols plus the input right end marker.
    c. The content of each cell will be either a number that corresponds to the production number from the productions file or a blank.
3. The program must check if the string of tokens specified in the input field is valid under the grammar represented by the input parse table and the productions list. The tokens in the input field are separated by some whitespace (trailing, in between; one or more horizontal/vertical spaces).
4. The series of steps while checking the input must be placed in an output file (***.prsd*** extension).
    a. The output filename (*outname* in the format) should be asked from the user and then appended with the productions filename (*prodname* in the format): ***outname_prodname***.
    b. The first row of the solution table shows the initial state of the parsing process. The succeeding rows correspond to the steps in the parsing process.
    c. The first column shows the content of the stack at every step in the process.
    d. The second column shows the content of the input buffer at every step in the process.
    e. The third column shows the action performed for the process.
5. After checking the input, the result should be **VALID** if the input is indeed correct under the grammar, and should be **INVALID** otherwise.
6. The productions, parse table, and output files content will be in CSV format.

III. Program Process Flow

- Once program is running, the main UI should show up immediately.
- Checking of validity of input can be done only when a valid file has been loaded for the parse table, a valid file has been loaded for the productions, and an input has been specified in the input field. The parse table and the productions files should be of the same name. Only ***.ptbl*** and ***.prod*** file extensions should be identified by the program for loading.
- Whenever a new file is to be loaded (successful/ unsuccessful) in the program, the Parsing solution display should be set to empty.
- Whenever an unsuccessful attempt (cancelled or invalid file encountered) is made in loading a file, the previously successfully loaded file should still be valid for use. This is for the files with the same extension.
- For the displays, the content of the successfully loaded file should replace the previously loaded content of the file with the same extension.
- Any time during program execution, the program user should be able to load files and process.
- The output file is to be saved to the same directory as the input file.
- Loading of file from any accessible directory.

IV. Sample Input

rules.ptbl

```
,id,+,*,(,),$
E,1,,,1,,
E',,2,,,3,3
T,4,,,4,,
T',,6,5,,6,6
F,8,,,7,,
```

rules.prod

```
1,E,T E'
2,E',+ T E'
3,E',e
4,T,F T'
5,T',* F T'
6,T',e
7,F,(E)
8,F,id
```

(NOTE: CSV files are text files used for storing tables. Each cell is separated by a comma.)

V. Sample Output

test_rules.prsd

```
E $,id + id * id $
T E' $,id + id * id $,Output E > T E'
F T' E' $,id + id * id $,Output T > F T'
id T' E' $,id + id * id $,Output F > id
T' E' $,+ id * id $,Match id
E' $,+ id * id $,Output T' > e
+ T E' $,+ id * id $,Output E' > + T E'
T E' $,id * id $,Match +
F T'E' $,id * id $,Output T > F T'
id T' E' $,id * id $,Output F > id
T' E' $,* id $,Match id
* F T' E' $,* id $,Output T' > * F T'
F T' E' $,id $,Match *
id T' E' $,id $,Output F > id
T' E' $,$,Match id
E' $,$,Output T' > e
$,$,Output E' > e
,,Match $
```

NOTE: If the result of parsing is VALID, the output file should contain the complete solution of the parsing. If the result of parsing is INVALID, the output file should contain the partial solution of the parsing to the point where parsing can no longer proceed. Then add another line stating an error (under the Action column).

The output file content is interpreted as follows (example for *test_rules.prsd*):

$E\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	
$TE'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $E \to TE'$
$FT'E'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $T \to FT'$
$\mathbf{id}\ T'E'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $F \to \mathbf{id}$
$T'E'\$$	$+\ \mathbf{id} * \mathbf{id}\$$	match **id**
$E'\$$	$+\ \mathbf{id} * \mathbf{id}\$$	output $T' \to \epsilon$
$+\ TE'\$$	$+\ \mathbf{id} * \mathbf{id}\$$	output $E' \to + TE'$
$TE'\$$	$\mathbf{id} * \mathbf{id}\$$	match +
$FT'E'\$$	$\mathbf{id} * \mathbf{id}\$$	output $T \to FT'$
$\mathbf{id}\ T'E'\$$	$\mathbf{id} * \mathbf{id}\$$	output $F \to \mathbf{id}$
$T'E'\$$	$* \mathbf{id}\$$	match **id**
$* FT'E'\$$	$* \mathbf{id}\$$	output $T' \to * FT'$
$FT'E'\$$	$\mathbf{id}\$$	match *
$\mathbf{id}\ T'E'\$$	$\mathbf{id}\$$	output $F \to \mathbf{id}$
$T'E'\$$	$\$$	match **id**
$E'\$$	$\$$	output $T' \to \epsilon$
$\$$	$\$$	output $E' \to \epsilon$

VI. Suggested Interface (with all the parts shown)

Productions:
rules.prod

ID	NT	P
1	E	T E'
2	E'	+ T E'
3	E'	e
4	T	F T'
5	T'	* F T'
6	T'	e
7	F	(E)
8	F	id

Parse Table:
rules.ptbl

	id	+	*	(	)	$
E	1			1		
E'		2			3	3
T	4			4		
T'		6	5		6	6
F	8			7		

LOADED: rules.prod **Load**

INPUT id + id * id **Parse**

PARSING: Valid. Please see test_rules.prsd.

Stack	Input Buffer	Action
E $	id + id * id $	
T E' $	id + id * id $	Output E > T E'
F T' E' $	id + id * id $	Output T > F T'
id T' E' $	id + id * id $	Output F > id
T' E' $	+ id * id $	Match id
E' $	+ id * id $	Output T' > e
+ T E' $	+ id * id $	Output E' > + T E'
T E' $	id * id $	Match +
F T'E' $	id * id $	Output T > F T'
id T' E' $	id * id $	Output F > id
T' E' $	* id $	Match id
* F T' E' $	* id $	Output T' > * F T'
F T' E' $	id $	Match *
id T' E' $	id $	Output F > id
T' E' $	$	Match id
E' $	$	Output T' > e
$	$	Output E' > e
		Match $