CPSC 323 Project 2 Report

1. Authors

- Eugene Chei - eugenechei@csu.fullerton.edu
- Belal Skaik - bskaik1@csu.fullerton.edu
- Jack Loague - jloague@csu.fullerton.edu

2. Running the Program (Windows)

1) Ensure that Python is installed on your computer (Download: https://www.python.org/downloads/).
2) Download the Project 2 repository (Repository link: https://github.com/jackloague1/CPSC-323-Project-2).
3) Within the command prompt, navigate to the directory of the lr_parser.py file.
4) Within the command prompt, run the command "python lr_parser.py" to run the program.

3. Code Documentation

1) First, in the main function, the program prompts the user for a string input.
2) The input string is then passed to the tokenize function, which divides the contents of the string into individual tokens and returns them in a list.
3) A stack object is then created using the ParsingStack() class.
4) The tokens list, the stack object, the context free grammar dictionary, and the parsing table dictionary are then passed to the parse_line function, which defines the logic of the stack implementation.
5) Within the parse_line function, a while loop is performed where during each iteration, the current action to be performed is determined based on the last element of stack object's items list, and the current token. The current token and the current action will be recorded in a list using the stack object's record_history function. A different process will occur depending on the action:
    - If the current action is undefined, the while loop will end and the parse_line function will return a value of "Rejected", meaning the input string was rejected. The program will then return to the main function.
    - If the current action is a shift action, the current token and the shift number indicated by the action will be appended to the end of the stack object's items list.
    - If the current action is a reduce action, the production rule based on the number of the reduce action will be retrieved from the context free grammar dictionary. The number of elements on the right hand side of the production rule will be doubled,

and this value will be used to determine how many elements to pop from the stack object's items list. Next, the goto number will be found based on the left hand side symbol of the production rule, and the last element of the stack object's items list. The left hand side symbol and the goto number will then be appended to the stack object's items list. If there is no goto number found for the left hand side symbol, the while loop will be exited, and the parse_line function will return a value of "Rejected", meaning the input string was rejected.. The program will then return to the main function.

- If the current action is an accept action, the while loop will end, and the parse_line function will return a value of "Accepted", meaning the input string was accepted. The program will then return to the main function.

6) Back in the main function, the steps of the stack implementation will be retrieved through the stack object's history list. The contents of this list will be displayed in the console. The result of the string being either accepted or rejected will also be displayed in the console.

4. Results

Test input # 1 - (id+id)*id$

```
Enter the expression to parse: (id+id)*id$
Step                Stack               Input               Action
----                -----               -----               ------
1                   0                   ( id + id ) * id $  Action: S4
2                   0 (                 ( id + id ) * id $  None
3                   0 ( 4               ( id + id ) * id $  None
4                   0 ( 4               id + id ) * id $    Action: S5
5                   0 ( 4 id            id + id ) * id $    None
6                   0 ( 4 id 5          id + id ) * id $    None
7                   0 ( 4 id 5          + id ) * id $       Action: R6
8                   0 ( 4 id            + id ) * id $       None
9                   0 ( 4               + id ) * id $       None
10                  0 ( 4 F             + id ) * id $       None
11                  0 ( 4 F 3           + id ) * id $       None
12                  0 ( 4 F 3           + id ) * id $       Action: R4
13                  0 ( 4 F             + id ) * id $       None
14                  0 ( 4               + id ) * id $       None
15                  0 ( 4 T             + id ) * id $       None
16                  0 ( 4 T 2           + id ) * id $       None
17                  0 ( 4 T 2           + id ) * id $       Action: R2
18                  0 ( 4 T             + id ) * id $       None
19                  0 ( 4               + id ) * id $       None
20                  0 ( 4 E             + id ) * id $       None
21                  0 ( 4 E 8           + id ) * id $       None
22                  0 ( 4 E 8           + id ) * id $       Action: S6
23                  0 ( 4 E 8 +         + id ) * id $       None
24                  0 ( 4 E 8 + 6       + id ) * id $       None
25                  0 ( 4 E 8 + 6       id ) * id $         Action: S5
26                  0 ( 4 E 8 + 6 id    id ) * id $         None
27                  0 ( 4 E 8 + 6 id 5  id ) * id $         None
28                  0 ( 4 E 8 + 6 id 5  ) * id $            Action: R6
29                  0 ( 4 E 8 + 6 id    ) * id $            None
30                  0 ( 4 E 8 + 6       ) * id $            None
31                  0 ( 4 E 8 + 6 F     ) * id $            None
32                  0 ( 4 E 8 + 6 F 3   ) * id $            None
33                  0 ( 4 E 8 + 6 F 3   ) * id $            Action: R4
34                  0 ( 4 E 8 + 6 F     ) * id $            None
35                  0 ( 4 E 8 + 6       ) * id $            None
36                  0 ( 4 E 8 + 6 T     ) * id $            None
37                  0 ( 4 E 8 + 6 T 9   ) * id $            None
38                  0 ( 4 E 8 + 6 T 9   ) * id $            Action: R1
39                  0 ( 4 E 8 + 6 T     ) * id $            None
40                  0 ( 4 E 8 + 6       ) * id $            None
41                  0 ( 4 E 8 +         ) * id $            None
42                  0 ( 4 E 8           ) * id $            None
```

43	0 (4 E	) * id $	None
44	0 (4	) * id $	None
45	0 (4 E	) * id $	None
46	0 (4 E 8	) * id $	None
47	0 (4 E 8	) * id $	Action: S11
48	0 (4 E 8)	) * id $	None
49	0 (4 E 8) 11	) * id $	None
50	0 (4 E 8) 11	* id $	Action: R5
51	0 (4 E 8)	* id $	None
52	0 (4 E 8	* id $	None
53	0 (4 E	* id $	None
54	0 (4	* id $	None
55	0 (	* id $	None
56	0	* id $	None
57	0 F	* id $	None
58	0 F 3	* id $	None
59	0 F 3	* id $	Action: R4
60	0 F	* id $	None
61	0	* id $	None
62	0 T	* id $	None
63	0 T 2	* id $	None
64	0 T 2	* id $	Action: S7
65	0 T 2 *	* id $	None
66	0 T 2 * 7	* id $	None
67	0 T 2 * 7	id $	Action: S5
68	0 T 2 * 7 id	id $	None
69	0 T 2 * 7 id 5	id $	None
70	0 T 2 * 7 id 5	$	Action: R6
71	0 T 2 * 7 id	$	None
72	0 T 2 * 7	$	None
73	0 T 2 * 7 F	$	None
74	0 T 2 * 7 F 10	$	None
75	0 T 2 * 7 F 10	$	Action: R3
76	0 T 2 * 7 F	$	None
77	0 T 2 * 7	$	None
78	0 T 2 *	$	None
79	0 T 2	$	None
80	0 T	$	None
81	0	$	None
82	0 T	$	None
83	0 T 2	$	None
84	0 T 2	$	Action: R2
85	0 T	$	None
86	0	$	None
87	0 E	$	None
88	0 E 1	$	None
89	0 E 1	$	Action: acc
90	0 E 1	$	Accept

Accepted

Test input # 2 - id*id$

```
Enter the expression to parse: id*id$
Step                Stack               Input               Action
----                -----               -----               ------
1                   0                   id * id $           Action: S5
2                   0 id                id * id $           None
3                   0 id 5              id * id $           None
4                   0 id 5              * id $              Action: R6
5                   0 id                * id $              None
6                   0                   * id $              None
7                   0 F                 * id $              None
8                   0 F 3               * id $              None
9                   0 F 3               * id $              Action: R4
10                  0 F                 * id $              None
11                  0                   * id $              None
12                  0 T                 * id $              None
13                  0 T 2               * id $              None
14                  0 T 2               * id $              Action: S7
15                  0 T 2 *             * id $              None
16                  0 T 2 * 7           * id $              None
17                  0 T 2 * 7           id $                Action: S5
18                  0 T 2 * 7 id        id $                None
19                  0 T 2 * 7 id 5      id $                None
20                  0 T 2 * 7 id 5      $                   Action: R6
21                  0 T 2 * 7 id        $                   None
22                  0 T 2 * 7           $                   None
23                  0 T 2 * 7 F         $                   None
24                  0 T 2 * 7 F 10      $                   None
25                  0 T 2 * 7 F 10      $                   Action: R3
26                  0 T 2 * 7 F         $                   None
27                  0 T 2 * 7           $                   None
28                  0 T 2 *             $                   None
29                  0 T 2               $                   None
30                  0 T                 $                   None
31                  0                   $                   None
32                  0 T                 $                   None
33                  0 T 2               $                   None
34                  0 T 2               $                   Action: R2
35                  0 T                 $                   None
36                  0                   $                   None
37                  0 E                 $                   None
38                  0 E 1               $                   None
39                  0 E 1               $                   Action: acc
40                  0 E 1               $                   Accept
Accepted
```

Test input # 3 - (id*)$

```
Enter the expression to parse: (id*)$
Step                Stack               Input               Action
----                -----               -----               ------
1                   0                   ( id * ) $          Action: S4
2                   0 (                 ( id * ) $          None
3                   0 ( 4               ( id * ) $          None
4                   0 ( 4               id * ) $            Action: S5
5                   0 ( 4 id            id * ) $            None
6                   0 ( 4 id 5          id * ) $            None
7                   0 ( 4 id 5          * ) $               Action: R6
8                   0 ( 4 id            * ) $               None
9                   0 ( 4               * ) $               None
10                  0 ( 4 F             * ) $               None
11                  0 ( 4 F 3           * ) $               None
12                  0 ( 4 F 3           * ) $               Action: R4
13                  0 ( 4 F             * ) $               None
14                  0 ( 4               * ) $               None
15                  0 ( 4 T             * ) $               None
16                  0 ( 4 T 2           * ) $               None
17                  0 ( 4 T 2           * ) $               Action: S7
18                  0 ( 4 T 2 *         * ) $               None
19                  0 ( 4 T 2 * 7       * ) $               None
20                  0 ( 4 T 2 * 7       ) $                 Syntax error
21                  0 ( 4 T 2 * 7       ) $                 Reject
Rejected
```