**COMPILERS ASSIGNMENT-PHASE 2**

Group members :

Simran Sahni: 2017B5A70856H

Radhesh Sarma: 2017B4A70886H

Chatrik Singh Mangat: 2017B5A70822H

Ashi Sinha: 2017B5A71149H

The parsing algorithm we have used is the **LL(1) parsing algorithm.** A top-down parser that uses a one-token lookahead is called an LL(1) parser.

* The first L indicates that the input is read from left to right
* The second L says that it produces a left-to-right derivation.
* And the 1 says that it uses one lookahead token

We first make a parse table for the implementation of our parser based on the First and Follow symbols of non-terminals present in the grammar. This parse table is given as input to the code. Parse table defines what character to expect and the next rule to follow, depending on the grammar. The parser follows this table to determine what token to parse.

Our parser accepts or rejects strings depending on whether they adhere to the rules of the grammar or not . The parser will not stop in case of bad input or error due to the error handling mechanisms incorporated and will continue to parse the input incase of error.

The program would run as python parser.py test1.txt

**Our Grammar :**

0 P ➝ S D  
1  P ➝ D  
2  S ➝ K D V D V  
3  S ➝ if F O V then V O F G  
4  S ➝ L S  
5  K ➝ true  
6  K ➝ false  
7  K ➝ return  
8  K ➝ void  
9  K ➝ main  
10  K ➝ for  
11  K ➝ while  
12  K ➝ until  
13  T ➝ int  
14  T ➝ float  
15  T ➝ boolean  
16  T ➝ string  
17  L ➝ while S { }  
18  L ➝ for S { }  
19  L ➝ until S { }  
20  C ➝ if S { }  
21  C ➝ else S { }  
22  V ➝ T O  
23  V ➝ id  
24  O ➝ +  
25  O ➝ -  
26  O ➝ %  
27  O ➝ /  
28  O ➝ \*  
29  O ➝ ==  
30  O ➝ >  
31  O ➝ <  
32  O ➝ >=  
33  O ➝ <=  
34  O ➝ !=  
35  O ➝ &&  
36  O ➝ ||  
37  O ➝ !  
38  O ➝ ?  
39  O ➝ :  
40  O ➝ =  
41  O ➝ "  
42  O ➝ $  
43  D ➝ {  
44  D ➝ }  
45  D ➝ (  
46  D ➝ )  
47  D ➝ [  
48  D ➝ ]  
49  D ➝ ,  
50  D ➝ ;  
51  F ➝ n  
52  G ➝ else V O F endif

**Firsts and Follows for the non-terminals :**

**Firsts :**

P : if,{,},(,),[,],,,;,true,false,return,void,main,for,while,until  
S : if, true , false ,return ,void ,main, for, while, until  
K : true, false, return, void, main, for, while, until  
T : int, float, boolean, string  
L : while, for, until  
C : if, else  
V : id ,int, float, Boolean , string  
O : +,-,%,/,\*,==,>,<,>=,<=,!=,&&,||,!,?,:,=,",$  
D : {,},(,),[,],,,;  
F : n  
G : else

**Follows :**

P : $  
S : {,},(,),[,],,,;  
K : {,},(,),[,],,,;  
T : +,-,%,/,\*,==,>,<,>=,<=,!=,&&,||,!,?,:,=,",$  
L : if, true, false, return, void, main, for, while, until  
C : if, true , false ,return ,void ,main, for, while, until  
V : {,},(,),[,],,,;,then,+,-,%,/,\*,==,>,<,>=,<=,!=,&&,||,!,?,:,=,",$  
O : id,int,float,boolean,string,n,{,},(,),[,],,,;,then,+,-,%,/,\*,==,>,<,>=,<=,!=,&&,||,!,?,:,=,",$  
D : $,id, int, float, boolean, string  
F : +,-,%,/,\*,==,>,<,>=,<=,!=,&&,||,!,?,:,=,",$,else,endif  
G : {,},(,),[,],,,;

**PARSE TABLE :**

**(As per the grammar rule number listed above)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | if | then | true | false | return | void | main | for | while | until | int | float | boolean | string | { | } | else | id | + | - | % | / | \* | == | > | < | >= | <= | != | && | || | ! | ? | : | = | “ | $ | ( | ) | [ | ] | , | ; | n | endif |
| P | 0 |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | 1 | 1 | 1 | 1 | 1 |  |  |
| S | 3 |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| K |  |  | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| T |  |  |  |  |  |  |  |  |  |  | 13 | 14 | 15 | 16 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| L |  |  |  |  |  |  |  | 18 | 17 | 19 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| C | 20 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 21 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| V |  |  |  |  |  |  |  |  |  |  | 22 | 22 | 22 | 22 |  |  |  | 23 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| O |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 |  |  |  |  |  |  |  |  |
| D |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 43 | 44 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 45 | 46 | 47 | 48 | 49 | 50 |  |  |
| F |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 51 |  |
| G |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 52 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**Error recovery actions used :**

* Auto-correction: Auto-corrects new unidentified symbols by making use of look-ahead symbols.
* Synchronization: Parser pops the stack until it synchronizes the stack with the current input symbol.
* Search: Search through the current stack to see if there are any productions that could match the current token, if there is then pop the stack to that non-terminal
* Continue consuming tokens: If stack has only one production left, continue parsing the input tokens till it matches with that production.
* Inserting tokens: Insert a terminal for continuation of parsing.
* Removes invalid tokens :Removes tokens which are not part of our grammar

**…………………………………………………………………………………………………………………………………………………………………………………………..**