

18CSC304J/ Compiler Design

Submitted By:- ANANNYA P. NEOG (RA1911003010367)

Exp-9: Computation of LR(0) in Item sets

Aim:- To write code to compute LR(0) in Item sets

Codes:-

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

//Variables used in most of the other modules.
char items[30][100][100];
char augmented_grammar[100][100], terminals[10], nonterminals[10];
int no_of_productions = 0, no_of_states = 0, no_of_items[30], no_of_terminals = 0,
no_of_nonterminals = 0;
char FIRST[2][10][10];
char FOLLOW[10][10];

//Variables used only in this module.
int state_index = 0, goto_state_index = 0, closure_item_index = 0;
int check(char c)
{
    int i;
    for(i = 0; i < no_of_terminals; i++)
        if(terminals[i] == c)
            return 1;
    return 0;
}

void generate_terminals()
{
    int i, j;
    int index = 0;
    for(i = 0; i < no_of_productions; i++)
    {
        for(j = 0; augmented_grammar[i][j] != '>'; j++);
        j++;
        for(; augmented_grammar[i][j] != '\0'; j++)
        {
            if(augmented_grammar[i][j] < 65 || augmented_grammar[i][j] > 90)
            {
                if(!check(augmented_grammar[i][j]))
                {
                    terminals[index] = augmented_grammar[i][j];
                    no_of_terminals++;
                    index++;
                }
            }
        }
    }
}
```

```

    }
}
terminals[index] = '$';
no_of_terminals++;
index++;
terminals[index] = '\0';
}

int check2(char c, int index)
{
    int i;
    for(i = 0; i < index; i++)
        if(nonterminals[i] == c)
            return 1;
    return 0;
}

void generate_nonterminals()
{
    int i, index = 0;
    for(i = 0; i < no_of_productions; i++)
        if(!check2(augmented_grammar[i][0], index))
        {
            nonterminals[index] = augmented_grammar[i][0];
            index++;
        }
    no_of_nonterminals = index;
    nonterminals[index] = '\0';
}

void initialize_items()
{
    generate_terminals();
    generate_nonterminals();
    int i;
    for(i = 0; i < 30; i++)
        no_of_items[i] = 0;
}

void generate_item(char *s, char *t)
{
    int i;
    for(i = 0; i < 3; i++)
        t[i] = s[i];
    t[i] = '.';
    if(s[i] != '@')
        for(; i < strlen(s); i++)
            t[i+1] = s[i];
            t[i+1] = '\0';
}

```

```

int item_found(char *s)
{
    //Check for items in a state.
    int i;
    for(i = 0; i < closure_item_index; i++)
    {
        if(!strcmp(s, items[state_index][i])) //If the strings match.
            return 1;
    }
    return 0;
}

int isterminal(char s)
{
    int i;
    for(i = 0; i < no_of_terminals; i++)
        if(s == terminals[i])
            return 1;
    return 0;
}

void closure(char *s)
{
    int i, j;
    for(i = 0; s[i] != '.'; i++)
        i++;
    if(!item_found(s))
    {
        strcpy(items[state_index][closure_item_index], s);
        closure_item_index++;
        // printf("%s\n", items[state_index][closure_item_index-1]);
    }

    if(s[i] == s[0] && s[i-2] == '>') //To avoid infinite loop due to left recursion.
        return;
    if(isterminal(s[i]))
        return;
    else
    { //Not a terminal
        for(j = 0; j < no_of_productions; j++)
        {
            char temp[100];
            if(augmented_grammar[j][0] == s[i])
            {
                generate_item(augmented_grammar[j], temp);
                closure(temp);
            }
        }
    }
}

```

```

}

int Goto1(char s, char temp[][100])
{ //Find Goto on symbol s. GOTO(goto_state_index, s)
    int i, j;
    int n = 0;
    char t, temp2[100];
    if(s == '\0')
    {
        return n;
    }
    for(i = 0; i < no_of_items[goto_state_index]; i++)
    {
        strcpy(temp2, items[goto_state_index][i]);
        for(j = 0; temp2[j] != '.'; j++);
        if(temp2[j+1] == '\0')
            continue;
        if(temp2[j+1] == s)
        {
            t = temp2[j];
            temp2[j] = temp2[j+1];
            temp2[j+1] = t;
            strcpy(temp[n], temp2);
            n++;
        }
    }

    return n;
}

int state_found(char *s)
{ //Checks for existence of same state.
    int i;
    for(i = 0; i < state_index; i++)
    {
        if(!strcmp(s, items[i][0])) //Compare with the first item of each state.
            return 1;
    }
    return 0;
}

int transition_item_found(char * t_items, char s, int t_index)
{
    int i;
    for(i = 0; i < t_index; i++)
        if(s == t_items[i])
            return 1;
    return 0;
}

```

```

void compute_closure_goto()
{
    char temp[100][100], transition_items[100];
    int i, no_of_goto_items, j, transition_index = 0;
    generate_item(augmented_grammar[0], temp[0]);
    closure(temp[0]);
    no_of_items[state_index] = closure_item_index;
    closure_item_index = 0;
    state_index++;
    //state_index is 1 now.
    while(goto_state_index < 30)
    {
        transition_index = 0;
        transition_items[transition_index] = '\0';
        for(i = 0; i < no_of_items[goto_state_index]; i++)
        {
            for(j = 0; items[goto_state_index][i][j] != '.'; j++);
            j++;
            if(!transition_item_found(transition_items,
                items[goto_state_index][i][j], transition_index))
            {
                transition_items[transition_index] =
                    items[goto_state_index][i][j];
                transition_index++;
            }
        }

        transition_items[transition_index] = '\0';
        for(i = 0; i < transition_index; i++)
        {
            int add_flag = 0;
            no_of_goto_items = Goto1(transition_items[i], temp);
            for(j = 0; j < no_of_goto_items; j++)
            {
                if(!state_found(temp[j]))
                {
                    add_flag = 1;
                    closure(temp[j]);
                }
                else
                    break;
            }

            if(add_flag)
            {
                no_of_items[state_index] = closure_item_index;
                closure_item_index = 0;
                state_index++;
            }
        }
    }
}

```

```

        goto_state_index++;
    }

    no_of_states = state_index;
}

void print()
{
    int i, j;
    printf("\nNumber of states = %d.\n", no_of_states);
    for(i = 0; i < no_of_states; i++)
    {
        printf("\nItems in State %d...\n", i);
        for(j = 0; j < no_of_items[i]; j++)
            printf("%s\n", items[i][j]);
    }
}

void start()
{
    char str[100];
    printf("Enter number of productions:");
    scanf("%d", &no_of_productions);
    printf("Enter the productions...\n");
    int i;
    for(i = 1; i <= no_of_productions; i++)
        scanf("%s", augmented_grammar[i]);
    printf("\n\nAugmented Grammar is...\n\n");
    strcpy(augmented_grammar[0], "Z->");
    str[0] = augmented_grammar[1][0];
    str[1] = '\0';
    strcat(augmented_grammar[0], str);
    no_of_productions++;
    for(i = 0; i < no_of_productions; i++)
        printf("%s\n", augmented_grammar[i]);
    initialize_items();
    compute_closure_goto();
    print();
}

int main()
{
    start();
    return 0;
}

```

```
main.c
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4
5 //Variables used in most of the other modules.
6 char items[30][100][100];
7 char augmented_grammar[100][100], terminals[10], nonterminals[10];
8 int no_of_productions = 0, no_of_states = 0, no_of_items[30], no_of_terminals = 0,
9 no_of_nonterminals = 0;
10 char FIRST[2][10][10];
11 char FOLLOW[10][10];
12
13 //Variables used only in this module.
14 int state_index = 0, goto_state_index = 0, closure_item_index = 0;
15 int check(char c)
16 {
17     int i;
18     for(i = 0; i < no_of_terminals; i++)
19         if(terminals[i] == c)
20             return 1;
21     return 0;
22 }
23
24 void generate_terminals()
25 {
26     int i, j;
27     int index = 0;
28     for(i = 0; i < no_of_productions; i++)
29     {
30         for(j = 0; augmented_grammar[i][j] != '>'; j++);
31         j++;
32         for(; augmented_grammar[i][j] != '\0'; j++)
33         {
34             if(augmented_grammar[i][j] < 65 || augmented_grammar[i][j] > 90)
35             {
36                 if(!check(augmented_grammar[i][j]))
37                 {
38                     terminals[index] = augmented_grammar[i][j];
39                     no_of_terminals++;
40                     index++;
41                 }
42             }
43         }
44     }
45     terminals[index] = '$';
46     no_of_terminals++;
47     index++;
48     terminals[index] = '\0';
49 }
50
51 int check2(char c, int index)
52 {
53     int i;
54     for(i = 0; i < index; i++)
55         if(nonterminals[i] == c)
56             return 1;
57     return 0;
58 }
59
60 void generate_nonterminals()
61 {
62     int i, index = 0;
63     for(i = 0; i < no_of_productions; i++)
64         if(!check2(augmented_grammar[i][0], index))
65         {
66             nonterminals[index] = augmented_grammar[i][0];
67             index++;
68         }
69     no_of_nonterminals = index;
70     nonterminals[index] = '\0';
71 }
72
```

```
main.c
38
39     terminals[index] = augmented_grammar[i][j];
40     no_of_terminals++;
41     index++;
42 }
43 }
44 }
45 terminals[index] = '$';
46 no_of_terminals++;
47 index++;
48 terminals[index] = '\0';
49 }
50
51 int check2(char c, int index)
52 {
53     int i;
54     for(i = 0; i < index; i++)
55         if(nonterminals[i] == c)
56             return 1;
57     return 0;
58 }
59
60 void generate_nonterminals()
61 {
62     int i, index = 0;
63     for(i = 0; i < no_of_productions; i++)
64         if(!check2(augmented_grammar[i][0], index))
65         {
66             nonterminals[index] = augmented_grammar[i][0];
67             index++;
68         }
69     no_of_nonterminals = index;
70     nonterminals[index] = '\0';
71 }
72
```

 Run

 Debug

 Stop

 Share

 Save


 Beautify





Language C


main.c


```
73 void initialize_items()
74 {
75     generate_terminals();
76     generate_nonterminals();
77     int i;
78     for(i = 0; i < 30; i++)
79         no_of_items[i] = 0;
80 }
81
82 void generate_item(char *s, char *t)
83 {
84     int i;
85     for(i = 0; i < 3; i++)
86         t[i] = s[i];
87         t[i] = '.';
88         if(s[i] != '@')
89             for(; i < strlen(s); i++)
90                 t[i+1] = s[i];
91                 t[i+1] = '\0';
92 }
93
94 int item_found(char *s)
95 {
96     //Check for items in a state.
97     int i;
98     for(i = 0; i < closure_item_index; i++)
99     {
100         if(!strcmp(s, items[state_index][i])) //If the strings match.
101             return 1;
102     }
103     return 0;
104 }
105
```


 Run


 Debug

 Stop

 Share

 Save

 Beautify



Language C

main.c

```
106 int isterminal(char s)
107 {
108     int i;
109     for(i = 0; i < no_of_terminals; i++)
110         if(s == terminals[i])
111             return 1;
112     return 0;
113 }
114
115 void closure(char *s)
116 {
117     int i, j;
118     for(i = 0; s[i] != '.'; i++);
119     i++;
120     if(!item_found(s))
121     {
122         strcpy(items[state_index][closure_item_index], s);
123         closure_item_index++;
124         // printf("%s\n", items[state_index][closure_item_index-1]);
125     }
126
127     if(s[i] == s[0] && s[i-2] == '>') //To avoid infinite loop due to left recursion.
128         return;
129     if(isterminal(s[i]))
130         return;
131     else
132     { //Not a terminal
133         for(j = 0; j < no_of_productions; j++)
134         {
135             char temp[100];
136             if(augmented_grammar[j][0] == s[i])
137             {
138                 generate_item(augmented_grammar[j], temp);
139                 closure(temp);
140             }
141         }
142     }
143 }
144
```



```
main.c
144
145 int Goto1(char s, char temp[][100])
146 { //Find Goto on symbol s. GOTO(goto_state_index, s)
147     int i, j;
148     int n = 0;
149     char t, temp2[100];
150     if(s == '\0')
151     {
152         return n;
153     }
154     for(i = 0; i < no_of_items[goto_state_index]; i++)
155     {
156         strcpy(temp2, items[goto_state_index][i]);
157         for(j = 0; temp2[j] != '.'; j++)
158             if(temp2[j+1] == '\0')
159                 continue;
160             if(temp2[j+1] == s)
161             {
162                 t = temp2[j];
163                 temp2[j] = temp2[j+1];
164                 temp2[j+1] = t;
165                 strcpy(temp[n], temp2);
166                 n++;
167             }
168     }
169     return n;
170 }
171
172
173 int state_found(char *s)
174 { //Checks for existence of same state.
175     int i;
176     for(i = 0; i < state_index; i++)
177     {
178         if(!strcmp(s, items[i][0])) //Compare with the first item of each state.
179             return 1;
180     }
181     return 0;
182 }
```

```
main.c
184 int transition_item_found(char *t_items, char s, int t_index)
185 {
186     int i;
187     for(i = 0; i < t_index; i++)
188         if(s == t_items[i])
189             return 1;
190     return 0;
191 }
192
193 void compute_closure_goto()
194 {
195     char temp[100][100], transition_items[100];
196     int i, no_of_goto_items, j, transition_index = 0;
197     generate_item(augmented_grammar[0], temp[0]);
198     closure(temp[0]);
199     no_of_items[state_index] = closure_item_index;
200     closure_item_index = 0;
201     state_index++;
202     //state_index is 1 now.
203     while(goto_state_index < 30)
204     {
205         transition_index = 0;
206         transition_items[transition_index] = '\0';
207         for(i = 0; i < no_of_items[goto_state_index]; i++)
208         {
209             for(j = 0; items[goto_state_index][i][j] != '.'; j++)
210                 j++;
211             if(!transition_item_found(transition_items,
212                                     items[goto_state_index][i][j], transition_index))
213             {
214                 transition_items[transition_index] =
215                     items[goto_state_index][i][j];
216                 transition_index++;
217             }
218         }
219
220         transition_items[transition_index] = '\0';
221         for(i = 0; i < transition_index; i++)
222         {
```

RunDebugStopShareSaveBeautify

Language C

```
main.c
223     int add_flag = 0;
224     no_of_goto_items = Goto1(transition_items[i], temp);
225     for(j = 0; j < no_of_goto_items; j++)
226     {
227         if(!state_found(temp[j]))
228         {
229             add_flag = 1;
230             closure(temp[j]);
231         }
232         else
233             break;
234     }
235
236     if(add_flag)
237     {
238         no_of_items[state_index] = closure_item_index;
239         closure_item_index = 0;
240         state_index++;
241     }
242 }
243
244 goto_state_index++;
245 }
246
247 no_of_states = state_index;
248 }
249
250 void print()
251 {
252     int i, j;
253     printf("\nNumber of states = %d.\n", no_of_states);
254     for(i = 0; i < no_of_states; i++)
255     {
256         printf("\n\nItems in State %d...\n\n", i);
257         for(j = 0; j < no_of_items[i]; j++)
258             printf("%s\n", items[i][j]);
259     }
260 }
261
```

RunDebugStopShareSaveBeautify

Language C

```
main.c
261
262 void start()
263 {
264     char str[100];
265     printf("Enter number of productions:");
266     scanf("%d", &no_of_productions);
267     printf("Enter the productions...\n");
268     int i;
269     for(i = 1; i <= no_of_productions; i++)
270         scanf("%s", augmented_grammar[i]);
271     printf("\n\nAugmented Grammar is...\n\n");
272     strcpy(augmented_grammar[0], "Z->");
273     str[0] = augmented_grammar[1][0];
274     str[1] = '\0';
275     strcat(augmented_grammar[0], str);
276     no_of_productions++;
277     for(i = 0; i < no_of_productions; i++)
278         printf("%s\n", augmented_grammar[i]);
279     initialize_items();
280     compute_closure_goto();
281     print();
282 }
283
284 int main()
285 {
286     start();
287     return 0;
288 }
289
```

Output:-

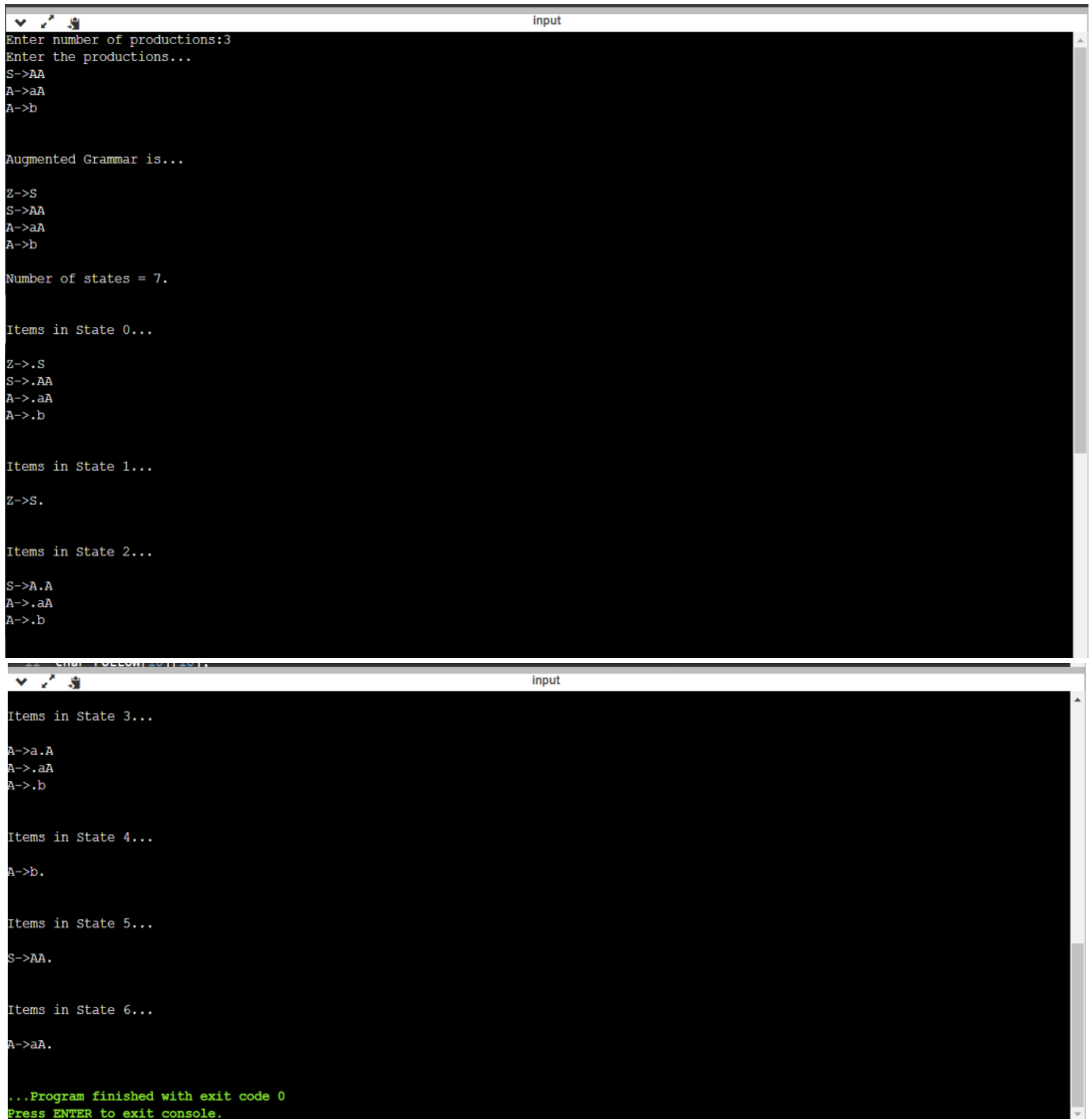
i) When input is:

S->AA

A->aA

A->b

as shown in the following output



```
input
Enter number of productions:3
Enter the productions...
S->AA
A->aA
A->b

Augmented Grammar is...

Z->S
S->AA
A->aA
A->b

Number of states = 7.

Items in State 0...

Z-> .S
S-> .AA
A-> .aA
A-> .b

Items in State 1...

Z->S.

Items in State 2...

S->A.A
A-> .aA
A-> .b

Items in State 3...

A->a.A
A-> .aA
A-> .b

Items in State 4...

A->b.

Items in State 5...

S->AA.

Items in State 6...

A->aA.

...Program finished with exit code 0
Press ENTER to exit console.
```

ii) When input is:

E->E+T

E->T

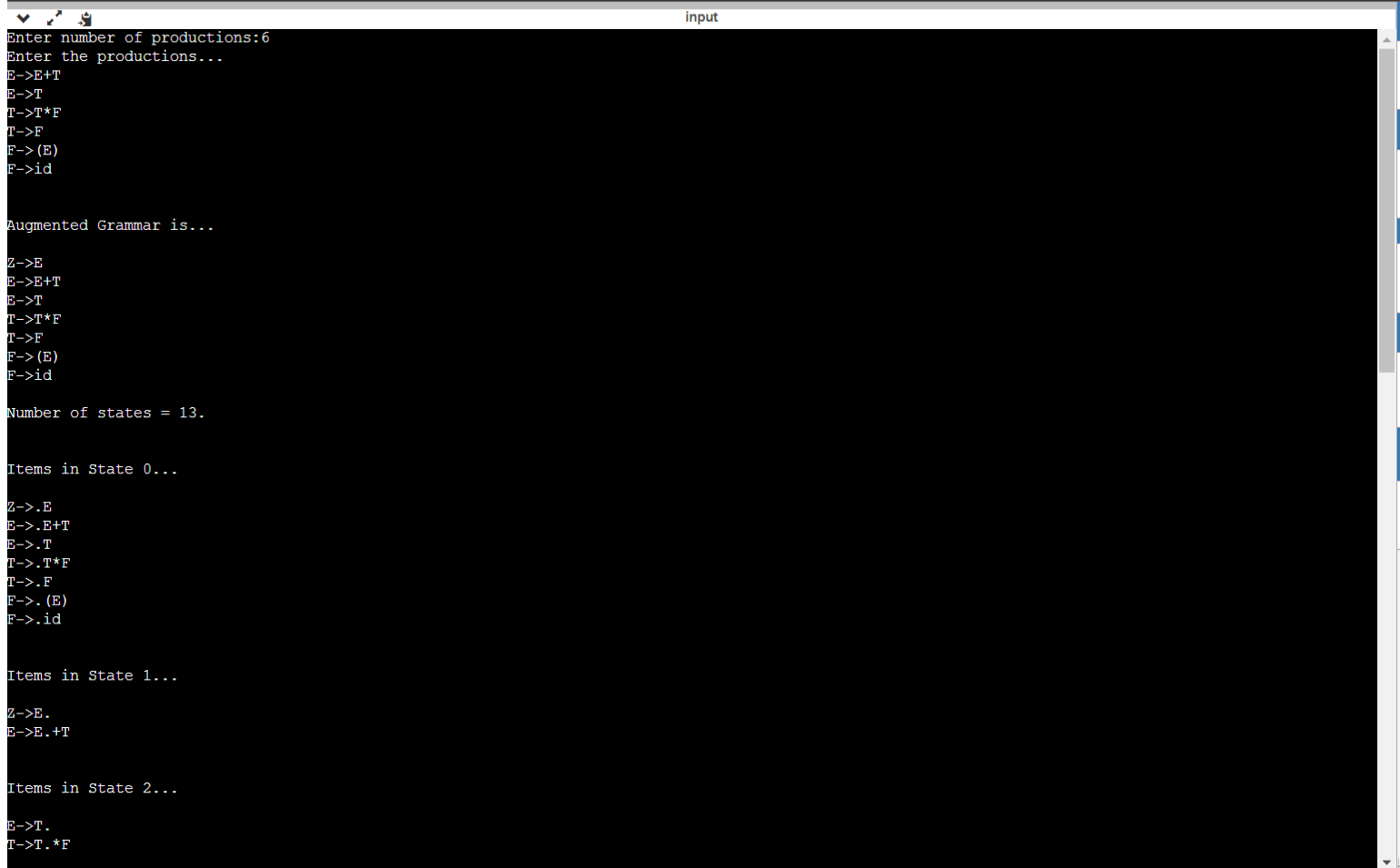
T->T*F

T->F

F->(E)

F->id

as shown in the following output



```
input
Enter number of productions:6
Enter the productions...
E->E+T
E->T
T->T*F
T->F
F->(E)
F->id

Augmented Grammar is...

Z->E
E->E+T
E->T
T->T*F
T->F
F->(E)
F->id

Number of states = 13.

Items in State 0...

Z->.E
E->|.E+T
E->|.T
T->|.T*F
T->|.F
F->|. (E)
F->|.id

Items in State 1...

Z->E.
E->E|. +T

Items in State 2...

E->T.
T->T|. *F
```

```
input
Items in State 3...
T->F.

Items in State 4...
F->(.E)
E->.E+T
E->T.
T->T*F
T->F
F->.(E)
F->.id

Items in State 5...
F->i.d

Items in State 6...
E->E+.T
T->T*F
T->F
F->.(E)
F->.id

Items in State 7...
T->T*.F
F->.(E)
F->.id

Items in State 8...
F->(E.)
E->E+.T
```

```
input
Items in State 9...
F->id.

Items in State 10...
E->E+T.
T->T*F

Items in State 11...
T->T*F.

Items in State 12...
F->(E) .

...Program finished with exit code 0
Press ENTER to exit console.
```