# 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:-

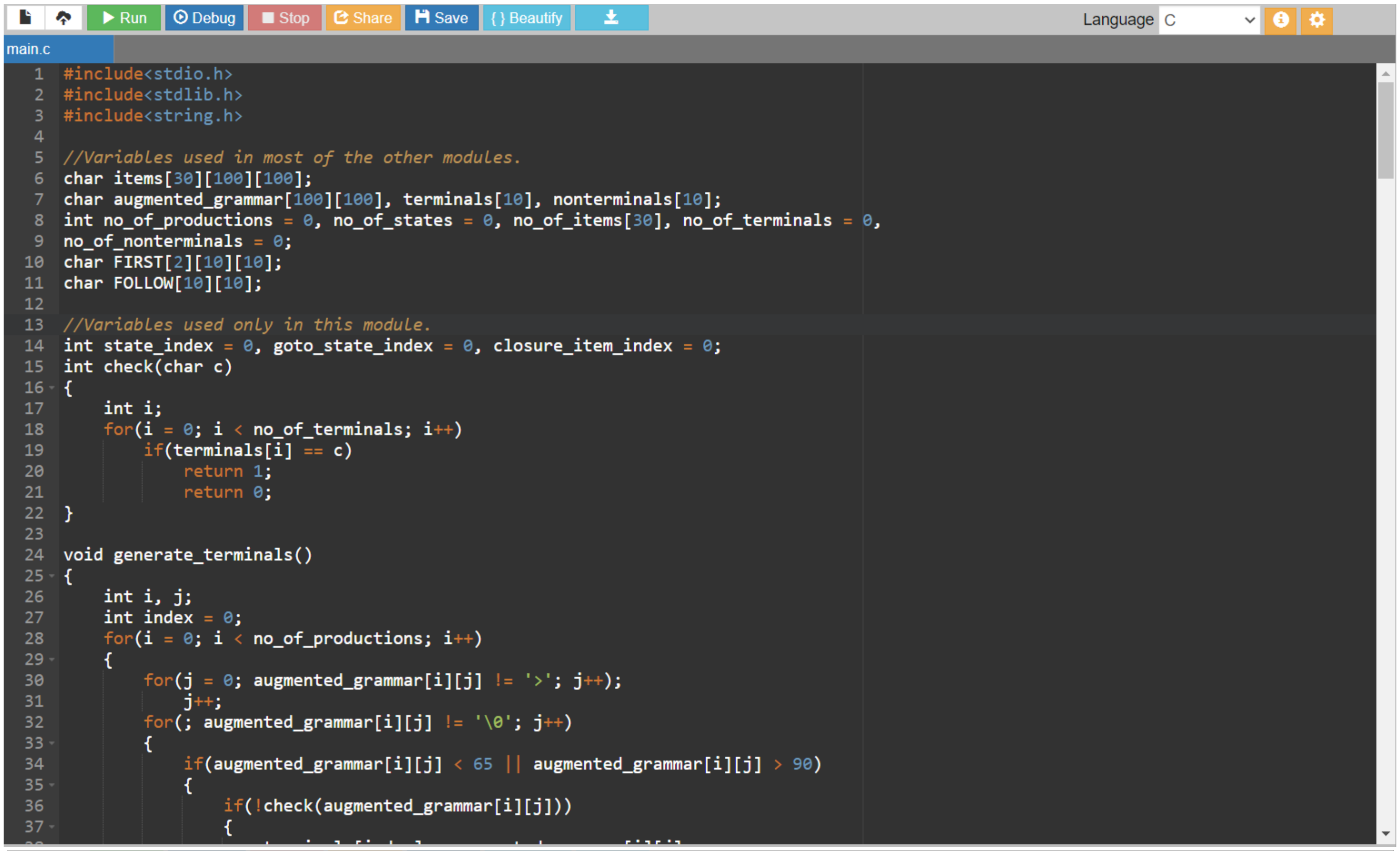
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

//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';
}
```

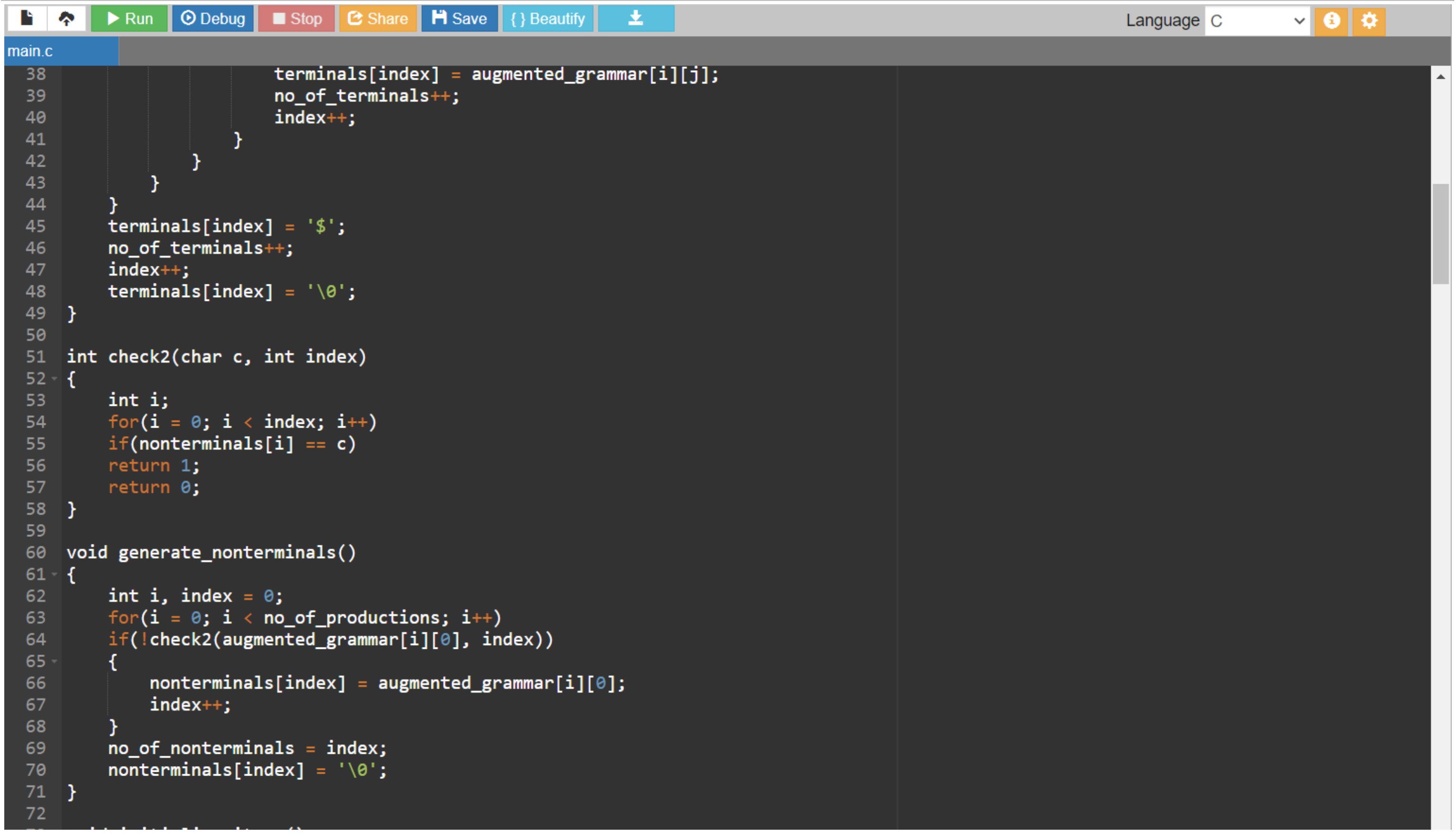
```
int item_found(char *s)
{
   //Check for items in a state.
   int i;
   for(i = 0; i < closure_item_index; i++)
   {
      if(!strcmp(s, items[state_index][i])) //If the strings match.
   return 1;
   }
   return 0;
}

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 existance 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++;
         }
      }
```

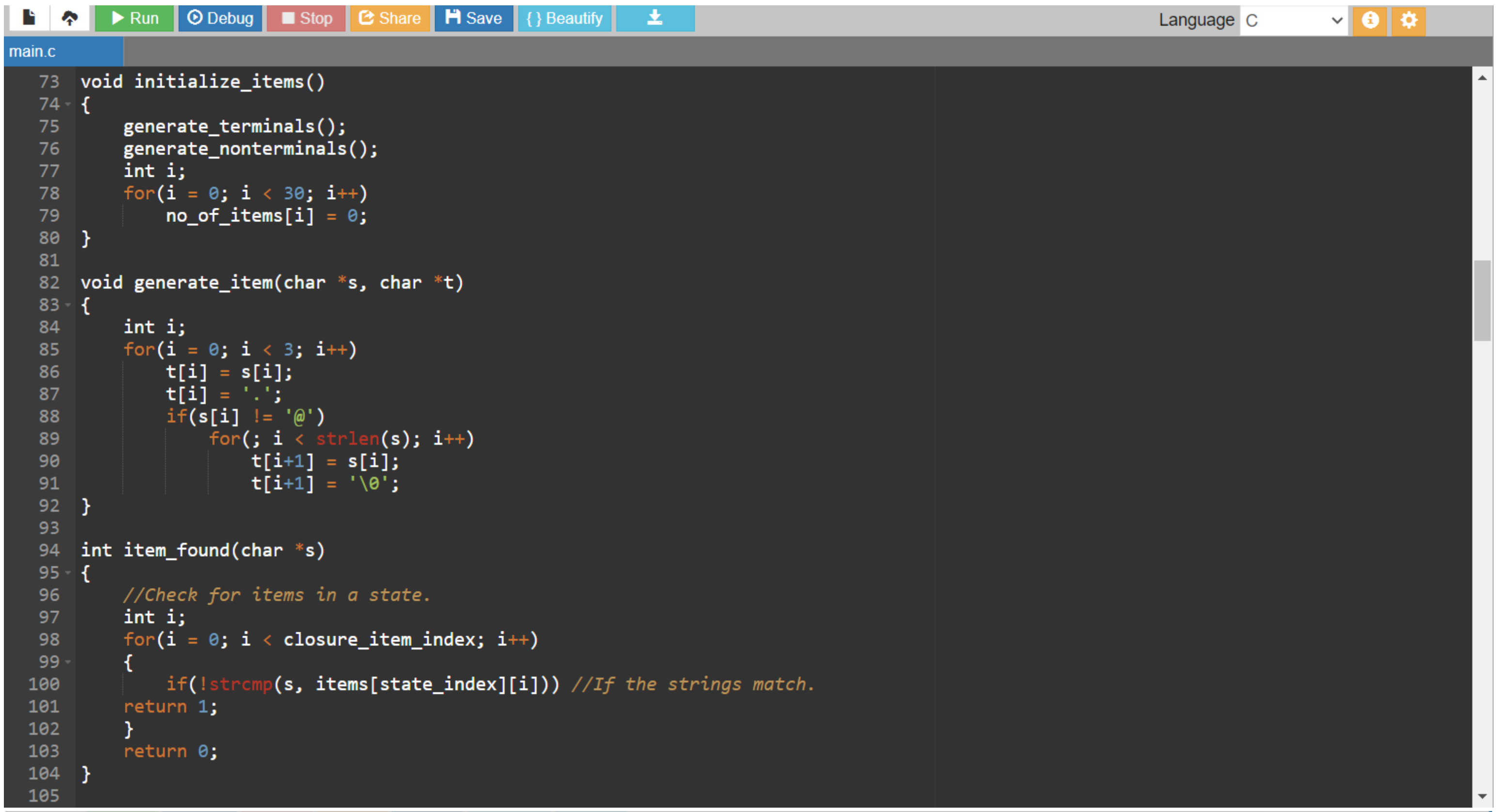
```
            goto_state_index++;
        }

    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("\n\nItems in State %d...\n\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;
}
```



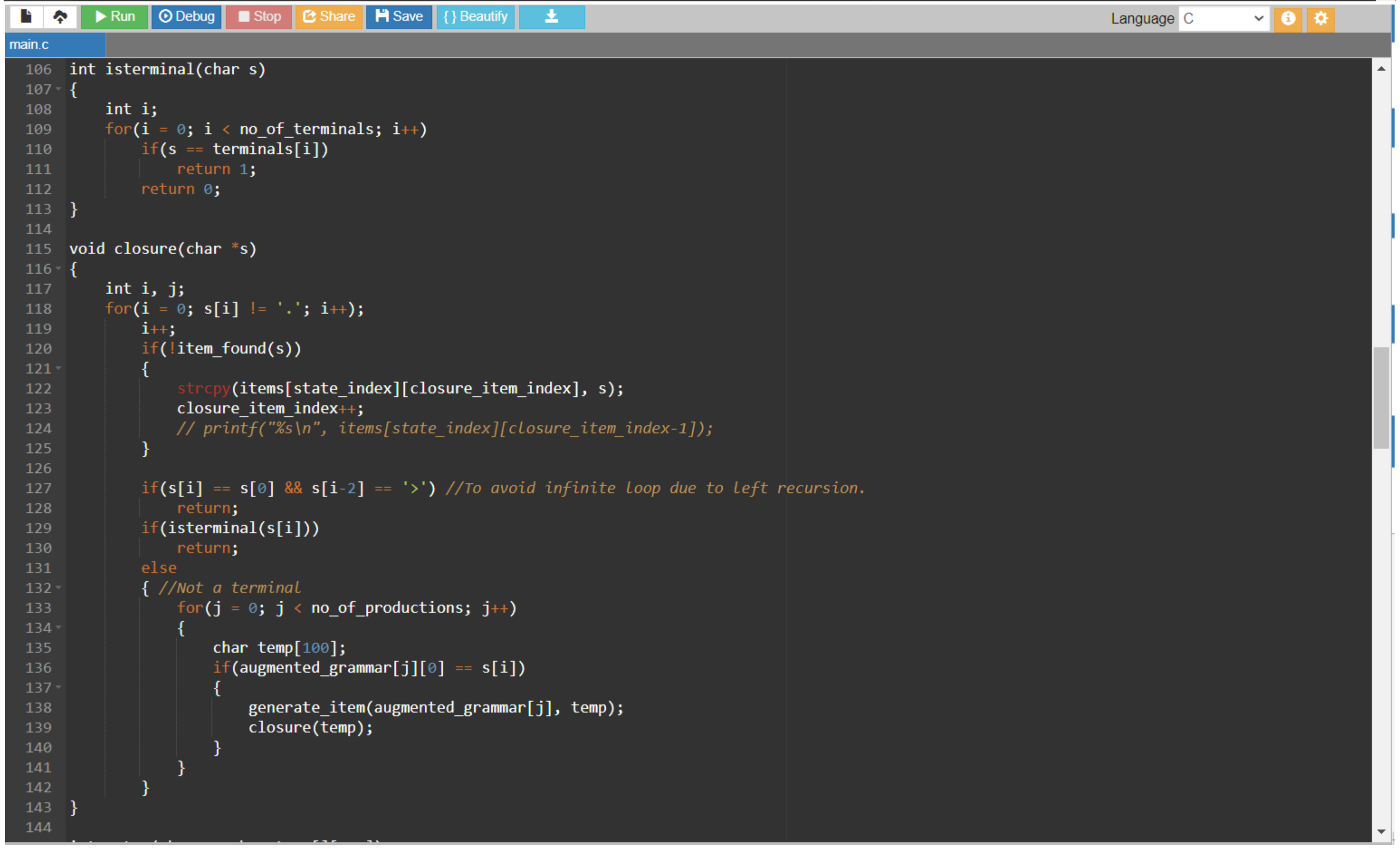
```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

//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]))
                {
```



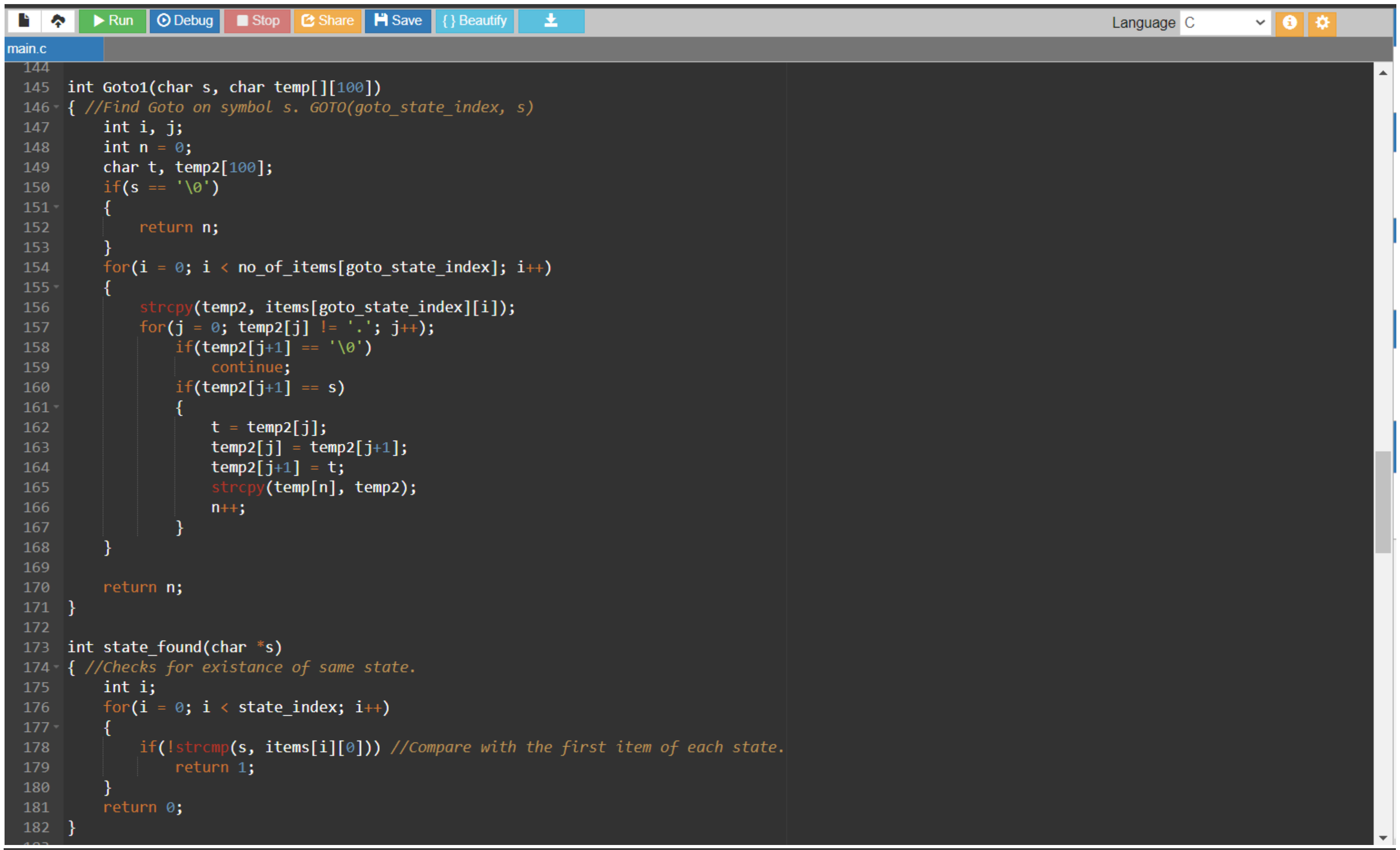
```c
                    terminals[index] = augmented_grammar[i][j];
                    no_of_terminals++;
                    index++;
                }
            }
        }
    }
    terminals[index] = '$';
    no_of_terminals++;
    index++;
    terminals[index] = '\0';
}

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';
}
```



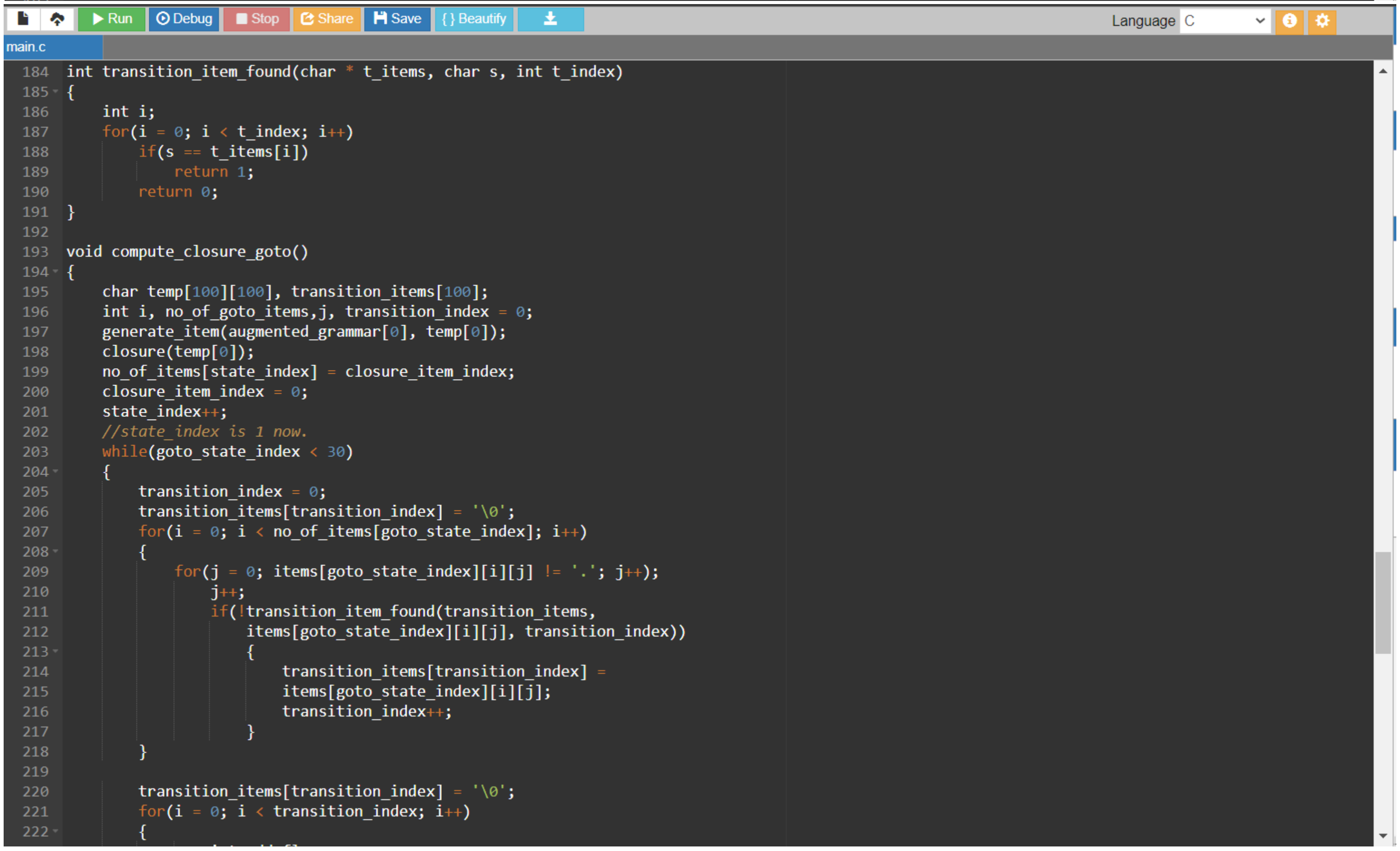
```c
void initialize_items()
{
    generate_terminals();
    generate_nonterminals();
    int i;
    for(i = 0; i < 30; i++)
        no_of_items[i] = 0;
}

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;
}

```



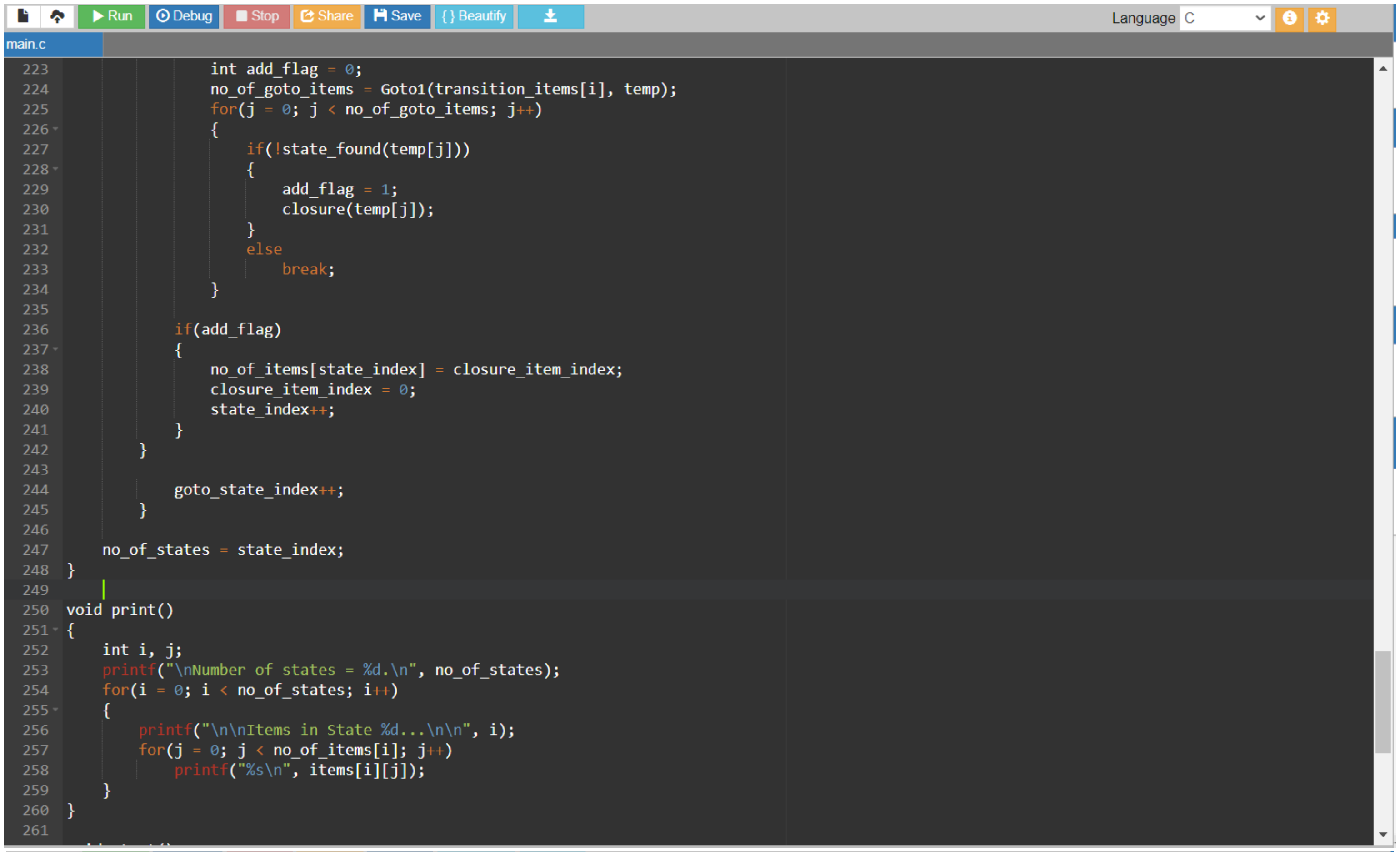
```c
int isterminal(char s)
{
    int i;
    for(i = 0; i < no_of_terminals; i++)
        if(s == terminals[i])
            return 1;
        return 0;
}

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);
                }
            }
        }
}

```



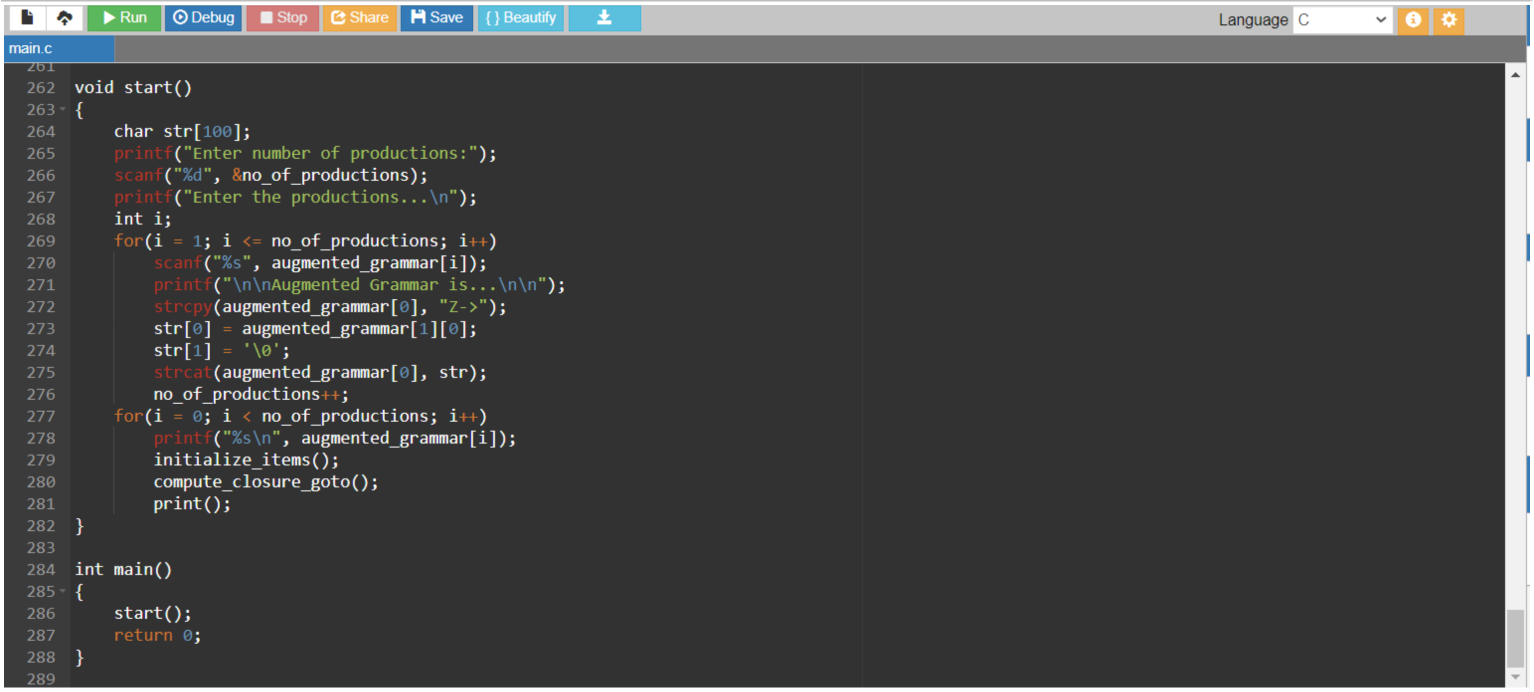
```c
int Goto1(char s, char temp[][100])
{ //Find Goto on symbol s. GOTO(goto_state_index, s)
    int i, j;
    int n = 0;
    char t, temp2[100];
    if(s == '\0')
    {
        return n;
    }
    for(i = 0; i < no_of_items[goto_state_index]; i++)
    {
        strcpy(temp2, items[goto_state_index][i]);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



```c
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++)
        {
```



```c
                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("\n\nItems in State %d...\n\n", i);
        for(j = 0; j < no_of_items[i]; j++)
            printf("%s\n", items[i][j]);
    }
}

```



```c
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;
}

```

## Output:-

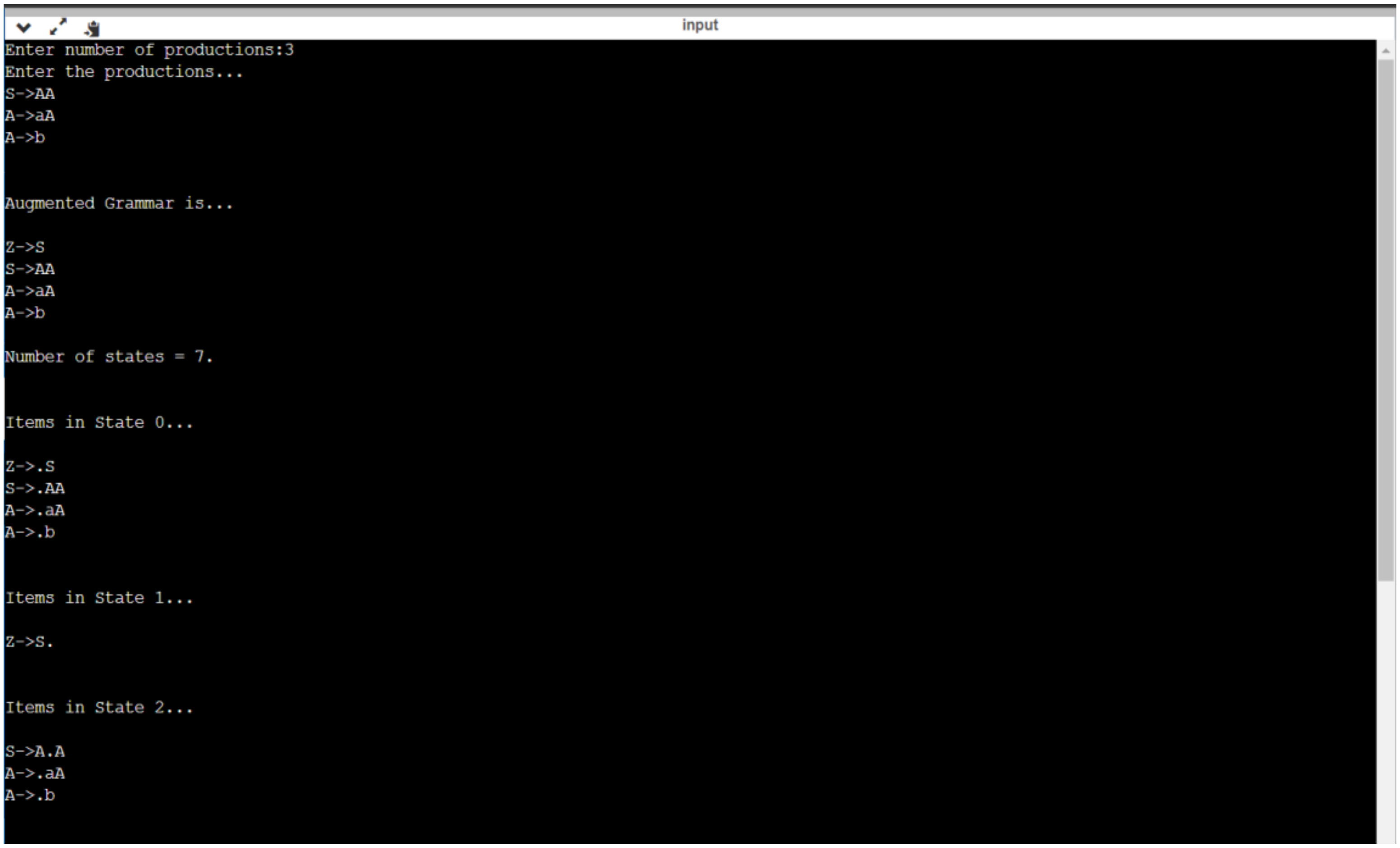
i) When input is:

**S->AA**

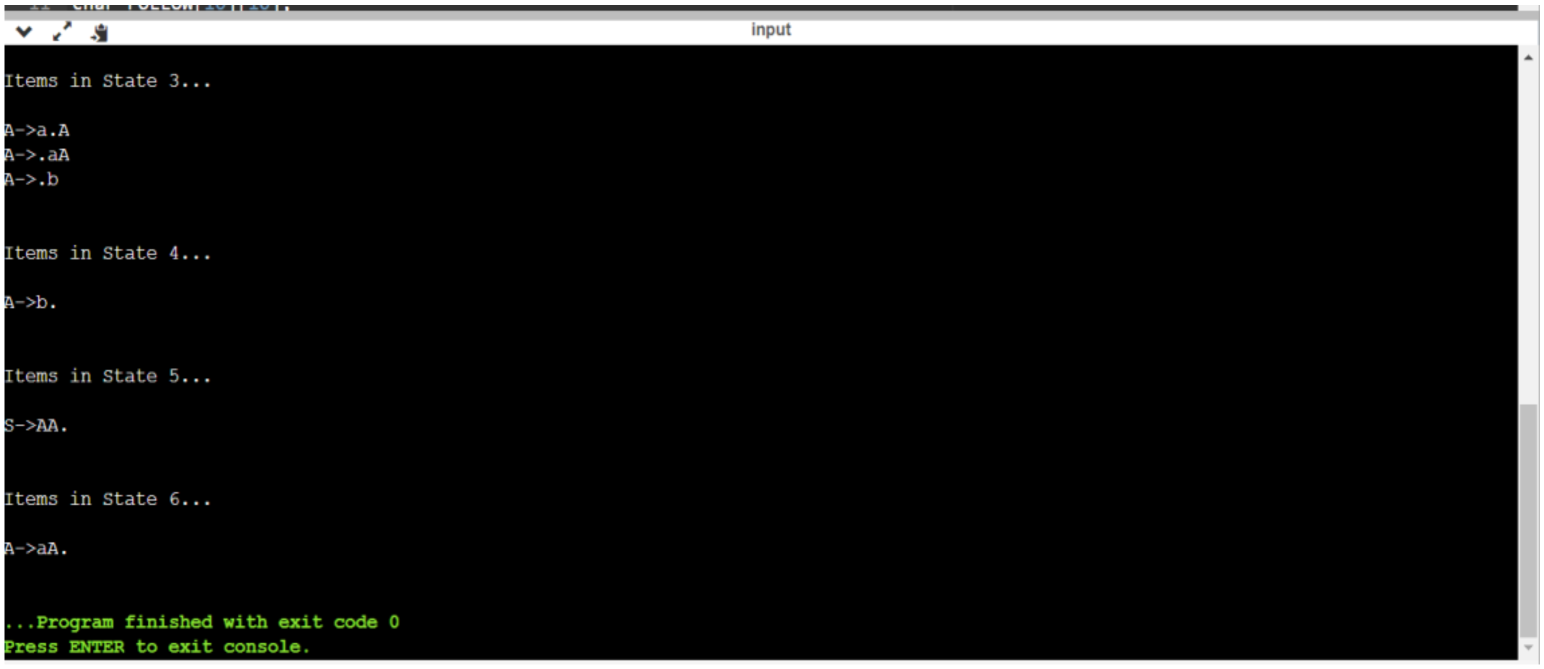
**A->aA**

**A->b**

as shown in the following output



```
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**

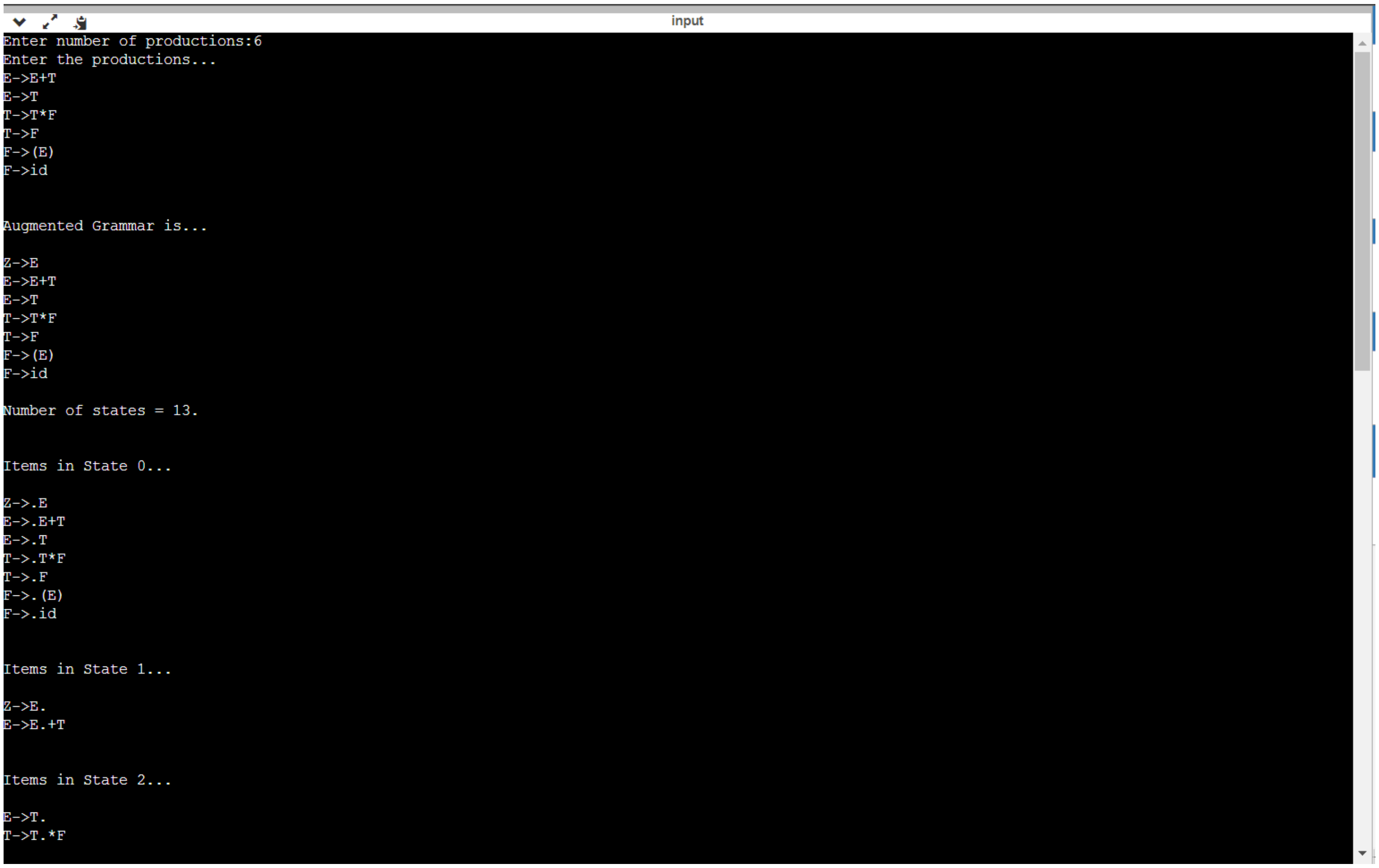
**T->T*F**

**T->F**

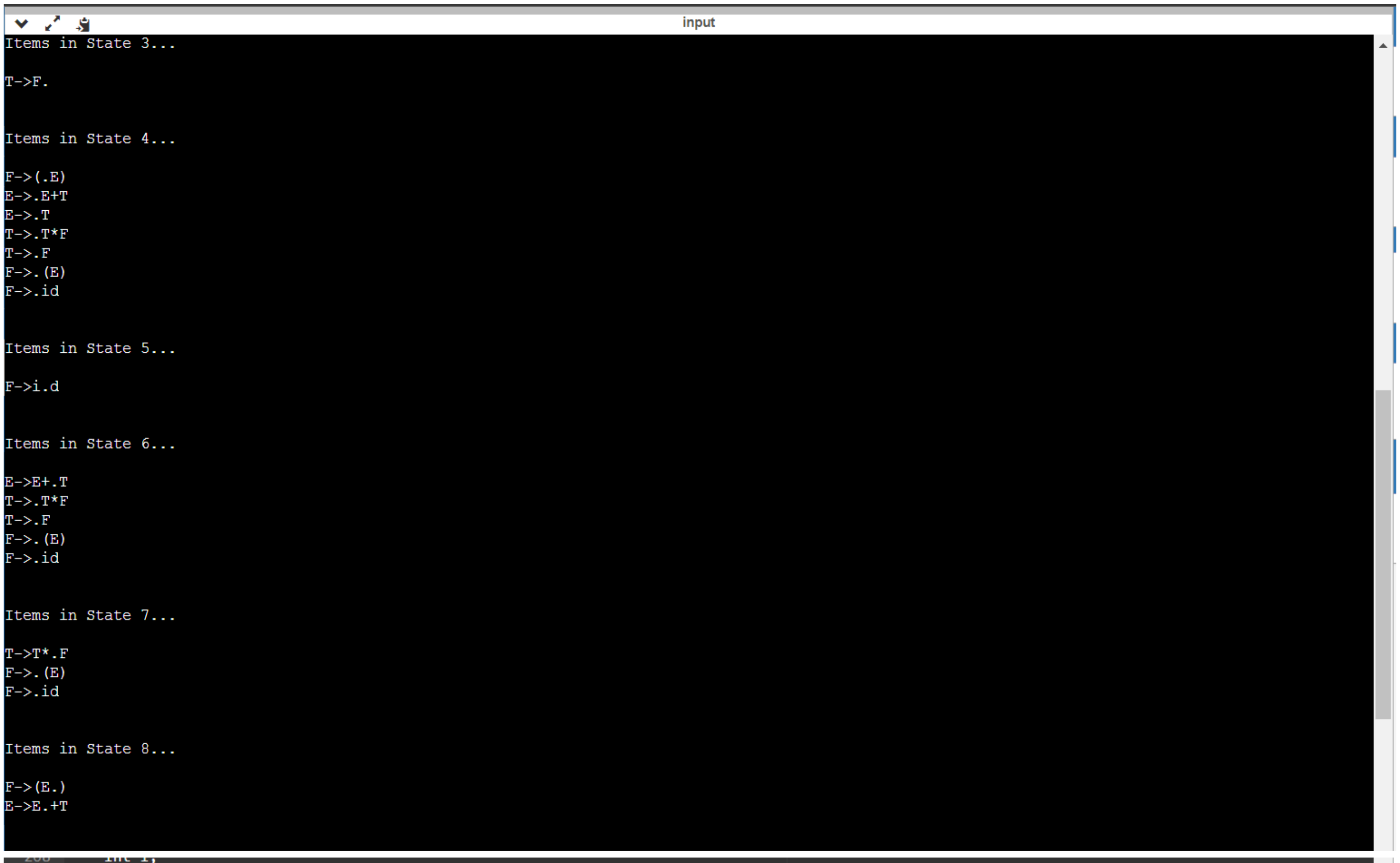
**F->(E)**

**F->id**

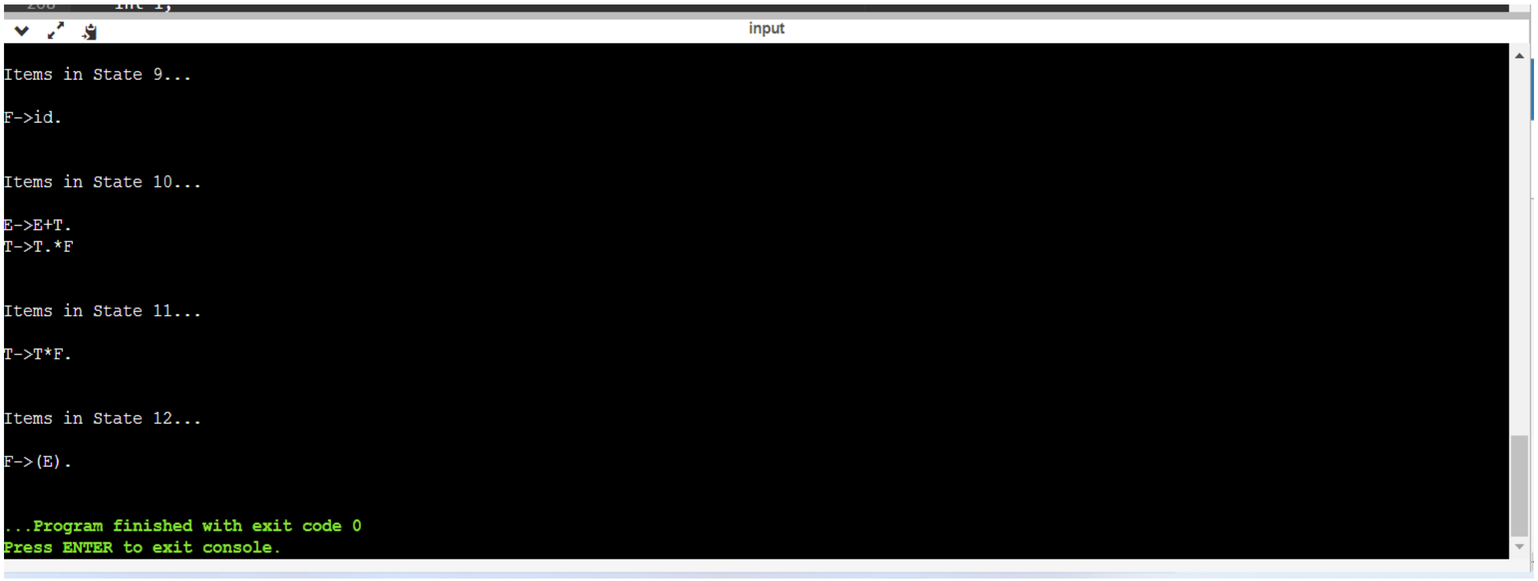
as shown in the following output



```
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
```



```
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
```



```
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.
```