Assignment 6

U20CS005
BANSI MARAKANA

Write a program to construct SLR (1) parse table for the following grammar and check whether the given input can be accepted or not.
Grammar:
S → CC
C → Cc | d

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

using namespace std;

int no_t, no_nt;
char terminals[100] = {}, non_terminals[100] = {}, goto_table[100][100];
char reduce[20][20], follow[20][20], fo_co[20][20], first[20][20];
char a[100] = {}, b = {};
struct state
{
    int prod_count;
    char prod[100][100] = {{}};
};

void add_dots(struct state *I)
{
    for (int i = 0; i < I->prod_count; i++)
    {
        for (int j = 99; j > 3; j--)
            I->prod[i][j] = I->prod[i][j - 1];
        I->prod[i][3] = '.';
    }
}

void augument(struct state *S, struct state *I)
{
    if (I->prod[0][0] == 'S')
        strcpy(S->prod[0], "Z->.S");
```

```cpp
    else
    {
        strcpy(S->prod[0], "S->.");
        S->prod[0][4] = I->prod[0][0];
    }
    S->prod_count++;
}

void get_prods(struct state *I)
{
    cout << "Enter the number of productions:\n";
    cin >> I->prod_count;
    cout << "Enter the number of non terminals:" << endl;
    cin >> no_nt;
    cout << "Enter the non terminals one by one:" << endl;
    for (int i = 0; i < no_nt; i++)
        cin >> non_terminals[i];
    cout << "Enter the number of terminals:" << endl;
    cin >> no_t;
    cout << "Enter the terminals (single lettered) one by one:" << endl;
    for (int i = 0; i < no_t; i++)
        cin >> terminals[i];
    cout << "Enter the productions one by one in form (S->ABc):\n";
    for (int i = 0; i < I->prod_count; i++)
        cin >> I->prod[i];
}

bool is_non_terminal(char a)
{
    if (a >= 'A' && a <= 'Z')
        return true;
    else
        return false;
}

bool in_state(struct state *I, char *a)
{
    for (int i = 0; i < I->prod_count; i++)
        if (!strcmp(I->prod[i], a))
            return true;
    return false;
}

char char_after_dot(char a[100])
{
    char b;
    for (int i = 0; i < strlen(a); i++)
        if (a[i] == '.')
        {
            b = a[i + 1];
            return b;
        }
    return -1;
}

char *move_dot(char b[100], int len)
{
    strcpy(a, b);
    for (int i = 0; i < len; i++)
    {
        if (a[i] == '.')
        {
            swap(a[i], a[i + 1]);
            break;
        }
    }
    return &a[0];
}

bool same_state(struct state *I0, struct state *I)
{

    if (I0->prod_count != I->prod_count)
        return false;

    for (int i = 0; i < I0->prod_count; i++)
    {
        int flag = 0;
        for (int j = 0; j < I->prod_count; j++)
            if (strcmp(I0->prod[i], I->prod[j]) == 0)
```

```
                flag = 1;
        if (flag == 0)
            return false;
    }
    return true;
}

void closure(struct state *I, struct state *I0)
{

    for (int i = 0; i < I0->prod_count; i++)
    {
        b = char_after_dot(I0->prod[i]);
        if (is_non_terminal(b))
        {
            for (int j = 0; j < I->prod_count; j++)
            {
                if (I->prod[j][0] == b)
                {
                    if (!in_state(I0, I->prod[j]))
                    {
                        strcpy(I0->prod[I0->prod_count], I->prod[j]);
                        I0->prod_count++;
                    }
                }
            }
        }
    }
}

void goto_state(struct state *I, struct state *S, char a)
{
    int time = 1;
    for (int i = 0; i < I->prod_count; i++)
    {
        if (char_after_dot(I->prod[i]) == a)
        {
            if (time == 1)
                time++;
            strcpy(S->prod[S->prod_count], move_dot(I->prod[i],
strlen(I->prod[i])));
            S->prod_count++;
        }
    }
}

void print_prods(struct state *I)
{
    for (int i = 0; i < I->prod_count; i++)
        printf("%s\n", I->prod[i]);
    cout << endl;
}

bool in_array(char a[20], char b)
{
    for (int i = 0; i < strlen(a); i++)
        if (a[i] == b)
            return true;
    return false;
}

char c[20] = {};
char *chars_after_dots(struct state *I)
{

    for (int i = 0; i < I->prod_count; i++)
        if (!in_array(c, char_after_dot(I->prod[i])))
            c[strlen(c)] = char_after_dot(I->prod[i]);
    return &a[0];
}

char d[100] = {};
void cleanup_prods(struct state *I)
{

    for (int i = 0; i < I->prod_count; i++)
        strcpy(I->prod[i], d);
    I->prod_count = 0;
}
```

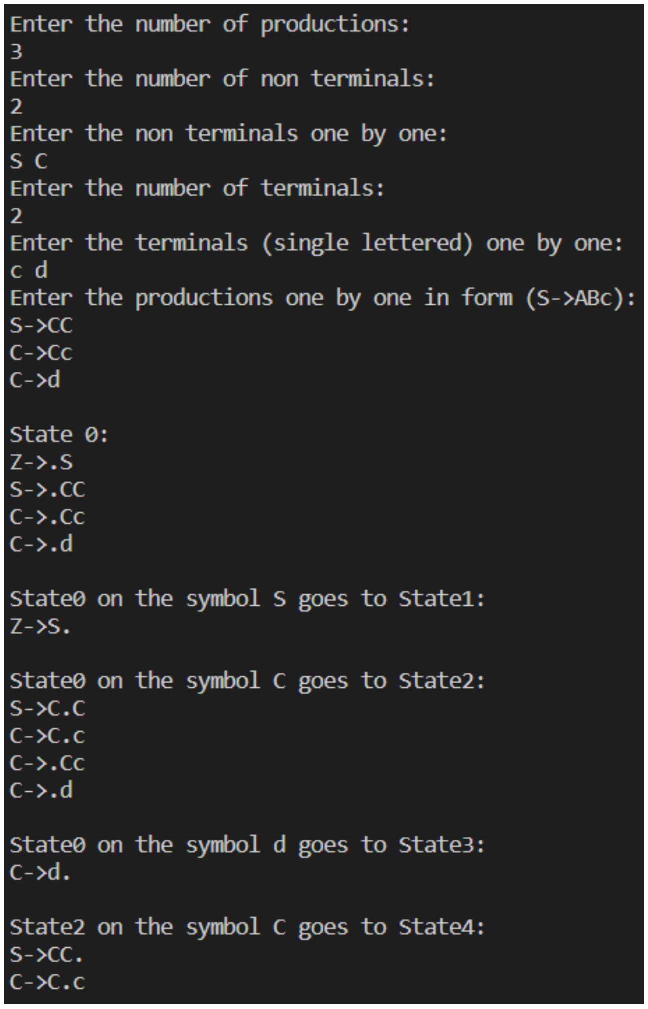
```cpp
int return_index(char a)
{
    for (int i = 0; i < no_t; i++)
        if (terminals[i] == a)
            return i;
    for (int i = 0; i < no_nt; i++)
        if (non_terminals[i] == a)
            return no_t + i;
    return -1;
}

void print_shift_table(int state_count)
{
    cout << endl << "********Shift Actions*********" << endl << endl;
    cout << "\t";
    for (int i = 0; i < no_t; i++)
        cout << terminals[i] << "\t";
    for (int i = 0; i < no_nt; i++)
        cout << non_terminals[i] << "\t";
    cout << endl;
    for (int i = 0; i < state_count; i++)
    {
        int arr[no_nt + no_t] = {-1};
        for (int j = 0; j < state_count; j++)
            if (goto_table[i][j] != '~')
                arr[return_index(goto_table[i][j])] = j;
        cout << "I" << i << "\t";
        for (int j = 0; j < no_nt + no_t; j++)
        {
            if (i == 1 && j == no_t - 1)
                cout << "ACC" << "\t";
            if (arr[j] == -1 || arr[j] == 0)
                cout << "\t";
            else
            {
                if (j < no_t)
                    cout << "S" << arr[j] << "\t";
                else
                    cout << arr[j] << "\t";
                }
            }
            cout << "\n";
        }
}

int get_index(char c, char *a)
{
    for (int i = 0; i < strlen(a); i++)
        if (a[i] == c)
            return i;
    return -1;
}

void add_dot_at_end(struct state *I)
{
    for (int i = 0; i < I->prod_count; i++)
        strcat(I->prod[i], ".");
}

void add_to_first(int n, char b)
{
    for (int i = 0; i < strlen(first[n]); i++)
        if (first[n][i] == b)
            return;
    first[n][strlen(first[n])] = b;
}

void add_to_first(int m, int n)
{
    for (int i = 0; i < strlen(first[n]); i++)
    {
        int flag = 0;
        for (int j = 0; j < strlen(first[m]); j++)
            if (first[n][i] == first[m][j])
                flag = 1;
        if (flag == 0)
            add_to_first(m, first[n][i]);
    }
}
```

```
void add_to_follow(int n, char b)
{
    for (int i = 0; i < strlen(follow[n]); i++)
        if (follow[n][i] == b)
            return;
    follow[n][strlen(follow[n])] = b;
}

void add_to_follow(int m, int n)
{
    for (int i = 0; i < strlen(follow[n]); i++)
    {
        int flag = 0;
        for (int j = 0; j < strlen(follow[m]); j++)
            if (follow[n][i] == follow[m][j])
                flag = 1;
        if (flag == 0)
            add_to_follow(m, follow[n][i]);
    }
}

void add_to_follow_first(int m, int n)
{
    for (int i = 0; i < strlen(first[n]); i++)
    {
        int flag = 0;
        for (int j = 0; j < strlen(follow[m]); j++)
            if (first[n][i] == follow[m][j])
                flag = 1;
        if (flag == 0)
            add_to_follow(m, first[n][i]);
    }
}

void find_first(struct state *I)
{
    for (int i = 0; i < no_nt; i++)
        for (int j = 0; j < I->prod_count; j++)
            if (I->prod[j][0] == non_terminals[i])
```

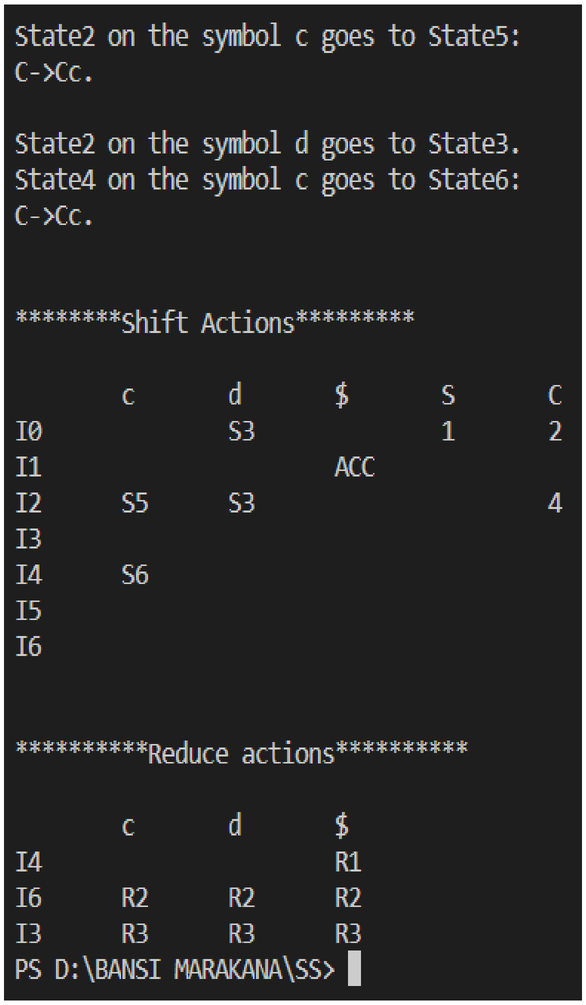
```
                if (!is_non_terminal(I->prod[j][3]))
                    add_to_first(i, I->prod[j][3]);
}

void find_follow(struct state *I)
{
    for (int i = 0; i < no_nt; i++)
        for (int j = 0; j < I->prod_count; j++)
            for (int k = 3; k < strlen(I->prod[j]); k++)
                if (I->prod[j][k] == non_terminals[i])
                    if (I->prod[j][k + 1] != '\0')
                        if (!is_non_terminal(I->prod[j][k + 1]))
                            add_to_follow(i, I->prod[j][k + 1]);
}

int get_index(int *arr, int n)
{
    for (int i = 0; i < no_t; i++)
        if (arr[i] == n)
            return i;
    return -1;
}

void print_reduce_table(int state_count, int *no_re, struct state *temp1)
{
    cout << "**********Reduce actions**********" << endl << endl;
    cout << "\t";
    int arr[temp1->prod_count][no_t] = {-1};
    for (int i = 0; i < no_t; i++)
        cout << terminals[i] << "\t";
    cout << endl;
    for (int i = 0; i < temp1->prod_count; i++)
    {
        int n = no_re[i];
        for (int j = 0; j < strlen(follow[return_index(temp1->prod[i][0])
- no_t]); j++)
            for (int k = 0; k < no_t; k++)
                if (follow[return_index(temp1->prod[i][0]) - no_t][j] ==
terminals[k])
                    arr[i][k] = i + 1;
        cout << "I" << n << "\t";
        for (int j = 0; j < no_t; j++)
            if (arr[i][j] != -1 && arr[i][j] != 0 && arr[i][j] <
state_count)
                cout << "R" << arr[i][j] << "\t";
            else
                cout << "\t";
        cout << endl;
    }
}

int main()
{
    struct state init;
    struct state temp;
    struct state temp1;
    int state_count = 1;
    get_prods(&init);
    temp = init;
    temp1 = temp;
    add_dots(&init);

    for (int i = 0; i < 100; i++)
        for (int j = 0; j < 100; j++)
            goto_table[i][j] = '~';

    struct state I[50];
    augument(&I[0], &init);
    closure(&init, &I[0]);
    cout << "\nState 0:\n";
    print_prods(&I[0]);

    char characters[20] = {};
    for (int i = 0; i < 20; i++)
        characters[i] = '~';
    for (int i = 0; i < state_count; i++)
    {
        char characters[20] = {};
        for (int z = 0; z < I[i].prod_count; z++)
            if (!in_array(characters, char_after_dot(I[i].prod[z])))
```

```
                characters[strlen(characters)] =
char_after_dot(I[i].prod[z]);

        for (int j = 0; j < strlen(characters); j++)
        {
            goto_state(&I[i], &I[state_count], characters[j]);
            closure(&init, &I[state_count]);
            int flag = 0;
            for (int k = 0; k < state_count - 1; k++)
            {
                if (same_state(&I[k], &I[state_count]))
                {
                    cleanup_prods(&I[state_count]);
                    flag = 1;
                    cout << "State" << i << " on the symbol " <<
characters[j] << " goes to State" << k << ".\n";
                    goto_table[i][k] = characters[j];
                    ;
                    break;
                }
            }
            if (flag == 0)
            {
                state_count++;
                cout << "State" << i << " on the symbol " << characters[j]
<< " goes to State" << state_count - 1 << ":\n";
                goto_table[i][state_count - 1] = characters[j];
                print_prods(&I[state_count - 1]);
            }
        }
    }

    int no_re[temp.prod_count] = {-1};
    terminals[no_t] = '$';
    no_t++;

    add_dot_at_end(&temp1);
    for (int i = 0; i < state_count; i++)
        for (int j = 0; j < I[i].prod_count; j++)
            for (int k = 0; k < temp1.prod_count; k++)
```

```
                if (in_state(&I[i], temp1.prod[k]))
                    no_re[k] = i;

    find_first(&temp);
    for (int l = 0; l < no_nt; l++)
        for (int i = 0; i < temp.prod_count; i++)
            if (is_non_terminal(temp.prod[i][3]))
                add_to_first(return_index(temp.prod[i][0]) - no_t,
return_index(temp.prod[i][3]) - no_t);

    find_follow(&temp);
    add_to_follow(0, '$');
    for (int l = 0; l < no_nt; l++)
    {
        for (int i = 0; i < temp.prod_count; i++)
        {
            for (int k = 3; k < strlen(temp.prod[i]); k++)
            {
                if (temp.prod[i][k] == non_terminals[l])
                {
                    if (is_non_terminal(temp.prod[i][k + 1]))
                        add_to_follow_first(l, return_index(temp.prod[i][k
+ 1]) - no_t);
                    if (temp.prod[i][k + 1] == '\0')
                        add_to_follow(l, return_index(temp.prod[i][0]) -
no_t);
                }
            }
        }
    }
    print_shift_table(state_count);
    cout << endl << endl;
    print_reduce_table(state_count, &no_re[0], &temp1);
}
```



```
Enter the number of productions:
3
Enter the number of non terminals:
2
Enter the non terminals one by one:
S C
Enter the number of terminals:
2
Enter the terminals (single lettered) one by one:
c d
Enter the productions one by one in form (S->ABc):
S->CC
C->Cc
C->d

State 0:
Z->.S
S->.CC
C->.Cc
C->.d

State0 on the symbol S goes to State1:
Z->S.

State0 on the symbol C goes to State2:
S->C.C
C->C.c
C->.Cc
C->.d

State0 on the symbol d goes to State3:
C->d.

State2 on the symbol C goes to State4:
S->CC.
C->C.c
```



```
State2 on the symbol c goes to State5:
C->Cc.

State2 on the symbol d goes to State3.
State4 on the symbol c goes to State6:
C->Cc.


********Shift Actions*********

        c       d       $       S       C
I0              S3              1       2
I1                      ACC
I2      S5      S3                      4
I3
I4      S6
I5
I6


**********Reduce actions**********

        c       d       $
I4                      R1
I6      R2      R2      R2
I3      R3      R3      R3
PS D:\BANSI MARAKANA\SS> 
```