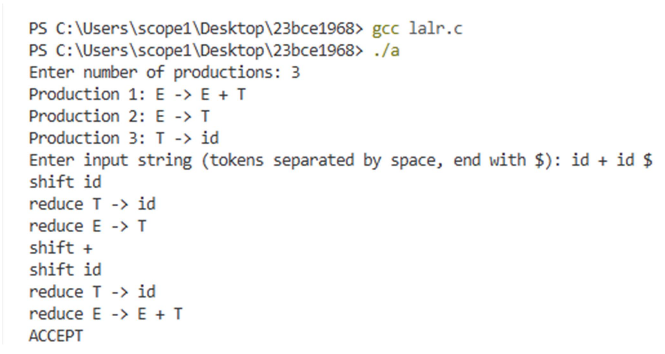
BCSE307P
Compiler Design Lab
LALR Parsing

Name: ADITYARAJ SHRIVASTAVA
Reg No: 23BCE1968
Slot: L5 + L6

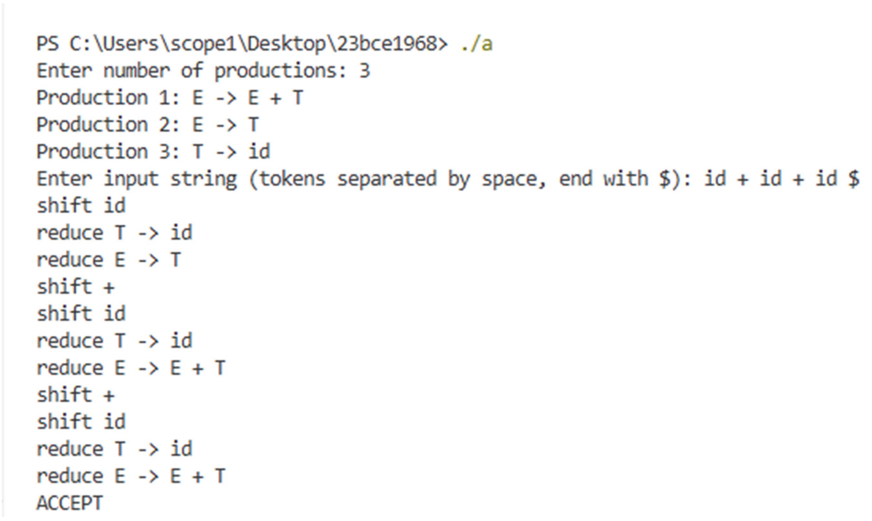
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXSYM 256
#define MAXP 200
#define MAXSTATES 2000

typedef struct {
    char *lhs;
    char **rhs;
    int rlen;
} Prod;

typedef struct {
    int action;
    int num;
} Act;

Prod prods[MAXP];
char *sym[MAXSYM];
int term[MAXSYM];
int nonterm[MAXSYM];
int P = 0;
int symc = 1;
Act ACTION[MAXSTATES][MAXSYM];
int GOTO[MAXSTATES][MAXSYM];

int sym_index(const char *s) {
    for (int i = 0; i < symc; i++)
        if (sym[i] && strcmp(sym[i], s) == 0)
            return i;
    sym[symc] = strdup(s);
    return symc++;
}

int add_prod(char *lhs, char **rhs, int rlen) {
    prods[P].lhs = strdup(lhs);
    prods[P].rhs = malloc(sizeof(char*) * rlen);
    for (int i = 0; i < rlen; i++)
        prods[P].rhs[i] = strdup(rhs[i]);
    prods[P].rlen = rlen;
    return P++;
}

void build_demo_table() {
    for (int i = 0; i < MAXSTATES; i++)
        for (int j = 0; j < MAXSYM; j++) {
            ACTION[i][j].action = 0;
            ACTION[i][j].num = 0;
            GOTO[i][j] = -1;
        }

    int id = sym_index("id");
    int plus = sym_index("+");
    int dollar = sym_index("$");
    int E = sym_index("E");
    int T = sym_index("T");
    term[id] = term[plus] = term[dollar] = 1;
    nonterm[E] = nonterm[T] = 1;

    ACTION[0][id].action = 1; ACTION[0][id].num = 5;
    GOTO[0][E] = 1;
    GOTO[0][T] = 2;
    ACTION[1][plus].action = 1; ACTION[1][plus].num = 3;
    ACTION[1][dollar].action = 3;
    ACTION[2][plus].action = 2; ACTION[2][plus].num = 2;
    ACTION[2][dollar].action = 2; ACTION[2][dollar].num = 2;
    ACTION[3][id].action = 1; ACTION[3][id].num = 5;
    GOTO[3][T] = 4;
    ACTION[4][plus].action = 2; ACTION[4][plus].num = 1;
    ACTION[4][dollar].action = 2; ACTION[4][dollar].num = 1;
    ACTION[5][plus].action = 2; ACTION[5][plus].num = 3;
    ACTION[5][dollar].action = 2; ACTION[5][dollar].num = 3;

    char *r1[] = {"E", "+", "T"}; add_prod("E", r1, 3);
    char *r2[] = {"T"}; add_prod("E", r2, 1);
    char *r3[] = {"id"}; add_prod("T", r3, 1);
}

int main() {
    int n;
    char line[1024];
    printf("Enter number of productions: ");
    scanf("%d", &n);
    getchar();

    for (int i = 0; i < n; i++) {
        printf("Production %d: ", i+1);
        fgets(line, 1024, stdin);
        line[strcspn(line, "\n")] = 0;
```

```c
}

build_demo_table();

printf("Enter input string (tokens separated by space, end with $): ");
fgets(line, 1024, stdin);
line[strcspn(line, "\n")] = 0;

char *tokens[256];
int tcnt = 0;
char *tok = strtok(line, " ");
while (tok) {
    tokens[tcnt++] = tok;
    tok = strtok(NULL, " ");
}

int stack[1000];
int stp = 0;
stack[stp++] = 0;
int ip = 0;

while (1) {
    int s = stack[stp-1];
    int a = sym_index(tokens[ip]);
    int act = ACTION[s][a].action;

    if (act == 1) {
        printf("shift %s\n", tokens[ip]);
        stack[stp++] = ACTION[s][a].num;
        ip++;
    } else if (act == 2) {
        int pno = ACTION[s][a].num;
        int rlen = prods[pno-1].rlen;
        for (int k = 0; k < rlen; k++) stp--;
        int top = stack[stp-1];
        int Aidx = sym_index(prods[pno-1].lhs);
        stack[stp++] = GOTO[top][Aidx];
        printf("reduce %s ->", prods[pno-1].lhs);
        for (int z = 0; z < rlen; z++)
            printf(" %s", prods[pno-1].rhs[z]);
        printf("\n");
    } else if (act == 3) {
        printf("ACCEPT\n");
        break;
    } else {
            printf("ERROR\n");
            break;
        }
    }
    return 0;
}
```

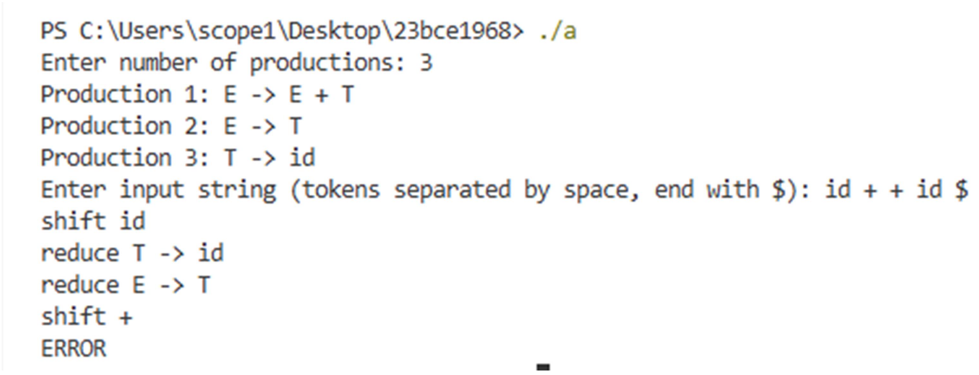
OUTPUT:



```
PS C:\Users\scope1\Desktop\23bce1968> gcc lalr.c
PS C:\Users\scope1\Desktop\23bce1968> ./a
Enter number of productions: 3
Production 1: E -> E + T
Production 2: E -> T
Production 3: T -> id
Enter input string (tokens separated by space, end with $): id + id $
shift id
reduce T -> id
reduce E -> T
shift +
shift id
reduce T -> id
reduce E -> E + T
ACCEPT
```



```
PS C:\Users\scope1\Desktop\23bce1968> ./a
Enter number of productions: 3
Production 1: E -> E + T
Production 2: E -> T
Production 3: T -> id
Enter input string (tokens separated by space, end with $): id + id + id $
shift id
reduce T -> id
reduce E -> T
shift +
shift id
reduce T -> id
reduce E -> E + T
shift +
shift id
reduce T -> id
reduce E -> E + T
ACCEPT
```



```
PS C:\Users\scope1\Desktop\23bce1968> ./a
Enter number of productions: 3
Production 1: E -> E + T
Production 2: E -> T
Production 3: T -> id
Enter input string (tokens separated by space, end with $): id + + id $
shift id
reduce T -> id
reduce E -> T
shift +
ERROR
```