National Institute Of Technology Surathkal Mangalore Karnataka-575025

Department Of Information Technology

Lab Assignment:- 09

Name:- Chikkeri Chinmaya

Roll Number:- 211IT017

Branch:- Information Technology (B.Tech)

Section :- S13

Course:-Automata And Compiler Design (IT252)

Submitted To:-

Anupama H C Mam

C++

```c
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <assert.h>

#define MAXBUF 4096

typedef struct Production {
    char nonterminal;
    char symbols[20];
    int size;
} Production;

typedef struct ProductionSet {
    Production productions[20];
    int size;
    char starting_symbol;
} ProductionSet;

void production_set_init(ProductionSet *p) {
    p->size = 0;
}

void production_init(Production *p) {
    p->size = 0;
}

void production_print(Production *p) {
    if (p->size == 0) {
        printf("@");
        return;
    }

    for (int symbol = 0; symbol < p->size; symbol++) {
        printf("%c", p->symbols[symbol]);
    }
}

void insert_production(ProductionSet *p, char nonterminal, char *input) {
    int i;
    p->size++;

    for (i = 0; input[i] != '\0' && input[i] != '\n'; i++) {
        p->productions[p->size - 1].symbols[i] = input[i];
    }
```

```
    p->productions[p->size - 1].size = i;
    p->productions[p->size - 1].nonterminal = nonterminal;
}

void grammar_print(ProductionSet *set) {
    for (int i = 0; i < set->size; i++) {
        printf("%c -> ", set->productions[i].nonterminal);
        production_print(&set->productions[i]);
        printf("\n");
    }
}

int grammar_input_check(ProductionSet *set, char *input) {
    char sr_stack[256];
    int sr_stack_top = 0;
    int input_offset = 0;
    char curr_input;
    int iteration = 0;

    printf("ITERATION\tSTACK\t\t\tCURR. INPUT\t\tACTION\n");

    while (1) {
        if (input[input_offset] == '\n' || input[input_offset] == '\0') {
            curr_input = '$';
        } else {
            curr_input = input[input_offset];
        }

        printf("%d\t\t[ ", iteration);
        for (int i = 0; i < sr_stack_top; i++) {
            printf("%c ", sr_stack[i]);
        }
        printf("]\t\t\t%c\t\t\t", curr_input);

        if (curr_input == '$') {
            printf("Input Exhausted. No additional handles found.\n");

            if (sr_stack_top != 1) {
                printf("Stack symbol count is not 1. Input Rejected.\n");
                return 0;
            }

            if (sr_stack[0] != set->starting_symbol) {
                printf("Final Stack Symbol is not Starting Symbol '%c'. Input
Rejected.\n", set->starting_symbol);
                return 0;
            }
```

```
            printf("Final Stack Symbol is Starting Symbol '%c'. Input
Accepted.\n", set->starting_symbol);
            return 1;
        }

        printf("Shifting '%c' to stack.\n", curr_input);
        sr_stack_top++;
        sr_stack[sr_stack_top - 1] = curr_input;
        input_offset++;

        if (input[input_offset] == '\n' || input[input_offset] == '\0') {
            curr_input = '$';
        } else {
            curr_input = input[input_offset];
        }

        int prod;
        int equal = 1;

        while (equal) {
            for (prod = 0; prod < set->size; prod++) {
                equal = 1;
                int k = set->productions[prod].size - 1;

                for (int j = sr_stack_top - 1; j >= 0 && k >= 0; j--, k--) {
                    if (set->productions[prod].symbols[k] != sr_stack[j]) {
                        equal = 0;
                        break;
                    }
                }

                if (k != -1) {
                    equal = 0;
                }

                if (equal) {
                    break;
                }
            }

            if (equal) {
                iteration++;
                printf("%d\t\t[ ", iteration);
                for (int i = 0; i < sr_stack_top; i++) {
                    printf("%c ", sr_stack[i]);
                }
                printf("]\t\t\t%c\t\t\tReducing Handle '", curr_input);
                production_print(&set->productions[prod]);
```

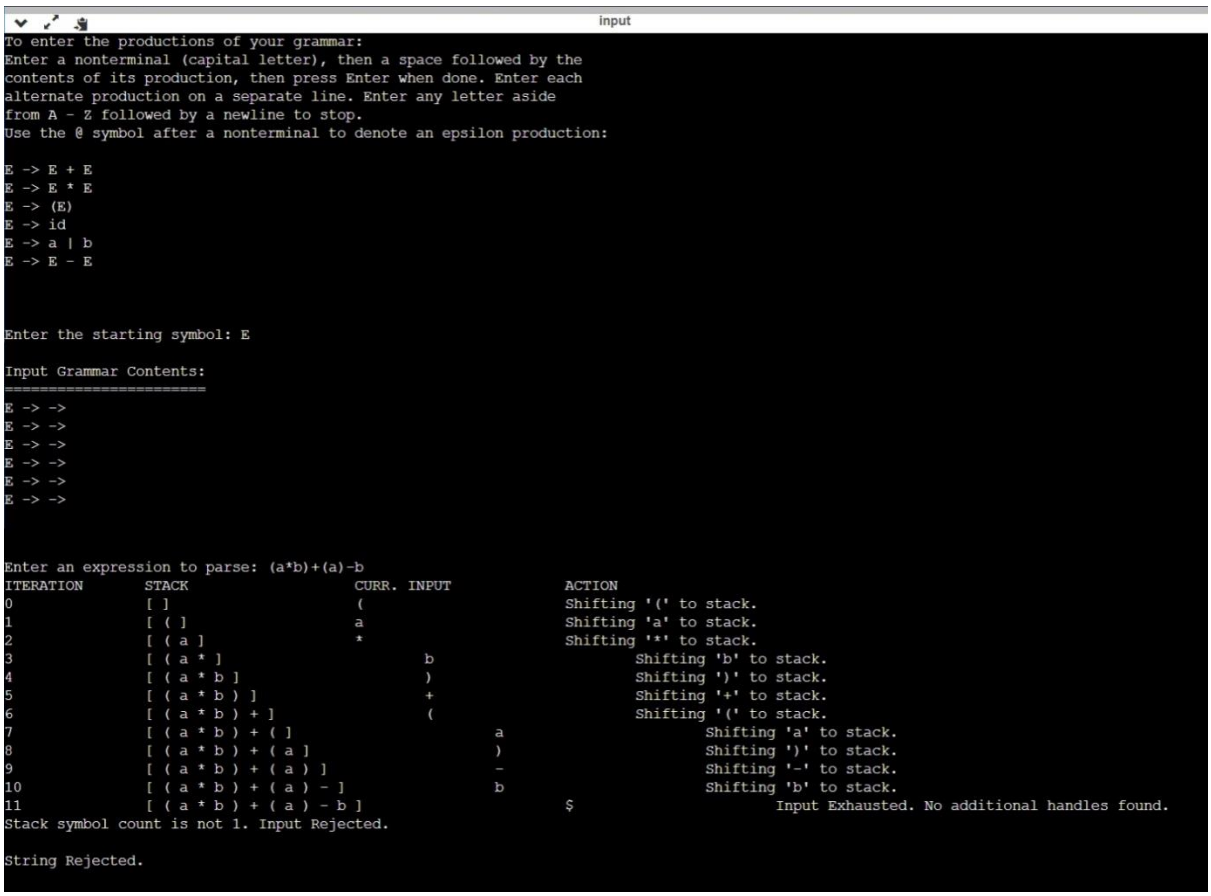
```
                printf("' to '%c'\n", set->productions[prod].nonterminal);

                for (int i = 0; i < set->productions[prod].size; i++) {
                    sr_stack_top--;
                }

                sr_stack_top++;
                sr_stack[sr_stack_top - 1] = set-
>productions[prod].nonterminal;
            } else {
                break;
            }
        }

        iteration++;
    }

    return -1;
}

// Input Flush
void flush() {
    int c;
    while (((c = getchar()) != EOF) && (c != '\n'));
}

int main() {
    ProductionSet g;

    char ch;
    char buf[MAXBUF];

    printf("To enter the productions of your grammar:\n");
    printf("Enter a nonterminal (capital letter), then a space followed by
the\n"
           "contents of its production, then press Enter when done. Enter
each\n"
           "alternate production on a separate line. Enter any letter aside\n"
           "from A - Z followed by a newline to stop.\n");
    printf("Use the @ symbol after a nonterminal to denote an epsilon
production:\n\n");

    production_set_init(&g);

    while (1) {
        ch = getchar();
        if (ch < 'A' || ch > 'Z') {
            break;
```

```
        }

        scanf("%s", buf);
        insert_production(&g, ch, buf);

        // Flush input
        flush();
    }

    // Flush input
    flush();

    printf("\nEnter the starting symbol: ");

    scanf("%c", &ch);

    // Flush input
    flush();

    g.starting_symbol = ch;

    printf("\nInput Grammar Contents:\n"
           "=======================\n");

    grammar_print(&g);

    printf("\n");

    printf("\nEnter an expression to parse: ");
    fgets(buf, MAXBUF, stdin);
    int verdict = grammar_input_check(&g, buf);

    printf("\n");

    if (verdict == 1) {
        printf("String Accepted.\n");
    } else {
        printf("String Rejected.\n");
    }

    return 0;
}
```

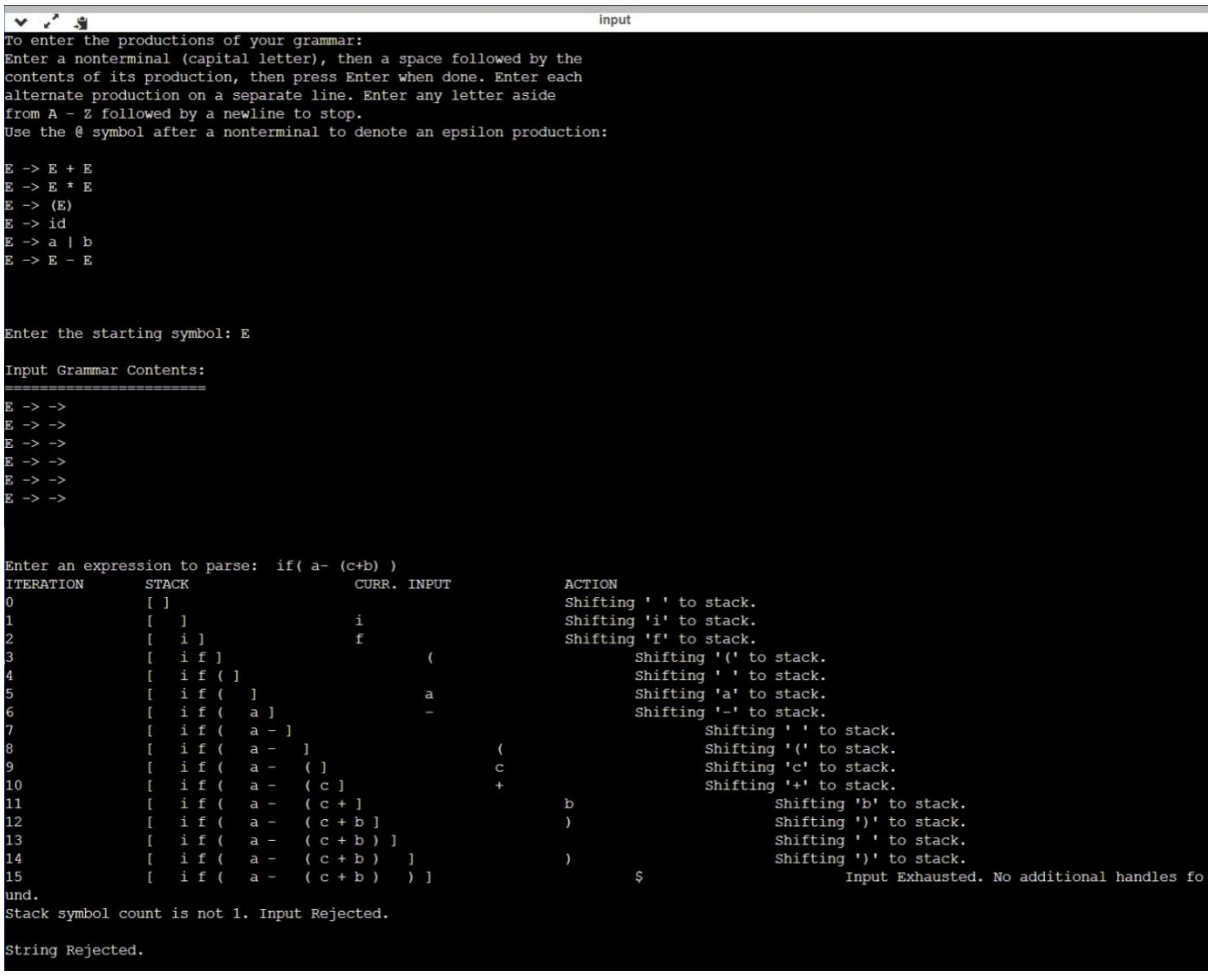
OUTPUT:-

For Input :

1) (a*b)+(a)-b



2) if(a- (c+b))


input
To enter the productions of your grammar:
Enter a nonterminal (capital letter), then a space followed by the
contents of its production, then press Enter when done. Enter each
alternate production on a separate line. Enter any letter aside
from A - Z followed by a newline to stop.
Use the @ symbol after a nonterminal to denote an epsilon production:
E -> E + E
E -> E * E
E -> (E)
E -> id
E -> a | b
E -> E - E
Enter the starting symbol: E
Input Grammar Contents:
E -> ->
E -> ->
E -> ->
E -> ->
E -> ->
E -> ->
Enter an expression to parse: if(a- (c+b))
ITERATION STACK CURR. INPUT ACTION
0 [ ] Shifting ' ' to stack.
1 [ ] i Shifting 'i' to stack.
2 [i] f Shifting 'f' to stack.
3 [i f] (Shifting '(' to stack.
4 [i f ( ] Shifting ' ' to stack.
5 [i f ( ] a Shifting 'a' to stack.
6 [i f (a] - Shifting '-' to stack.
7 [i f (a -] Shifting ' ' to stack.
8 [i f (a -] (Shifting '(' to stack.
9 [i f (a - ( ] c Shifting 'c' to stack.
10 [i f (a - (c] + Shifting '+' to stack.
11 [i f (a - (c +] b Shifting 'b' to stack.
12 [i f (a - (c + b]) Shifting ')' to stack.
13 [i f (a - (c + b)] Shifting ' ' to stack.
14 [i f (a - (c + b)]) Shifting ')' to stack.
15 [i f (a - (c + b))] $ Input Exhausted. No additional handles fo
und.
Stack symbol count is not 1. Input Rejected.
String Rejected.