

**Compiler Design Lab**

**SUBMITTED BY: SUBMITTED TO:**

Name: Harsh Agrawal Dr. Ajay Prasad

SAP ID: 500107136

Enrollment No: R2142220326

Course- B.Tech. (CSE)

Batch: 4 (AI/ML) (Non- Hons)

**Assignment 7**

1. WAP to implement an Operator Precedence Parser for a given language in C

Sol-

Code-

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Function to exit and print "Not operator grammar"

void f() {

printf("Not operator grammar\n");

exit(0);

}

int main() {

char grm[20][20];

int i, n, j, flag = 0;

// Taking number of productions from the user

printf("Enter number of productions: ");

scanf("%d", &n);

printf("Enter the productions:\n");

for (i = 0; i < n; i++) {

scanf("%s", grm[i]);

}

for (i = 0; i < n; i++) {

j = 2; // Reset index for each production

// If the RHS starts with '$', exit immediately

if (grm[i][j] == '$') {

f();

}

// Check for the presence of operators

int has\_operator = 0;

while (grm[i][j] != '\0') { // Iterate until end of production

char c = grm[i][j];

// If an operator is found, set flag

if (c == '+' || c == '-' || c == '\*' || c == '/') {

has\_operator = 1;

}

// Move to next character

j++;

}

// If no operator is found, it's not an operator grammar

if (!has\_operator) {

f();

}

flag = 1; // Indicating it's an operator grammar

}

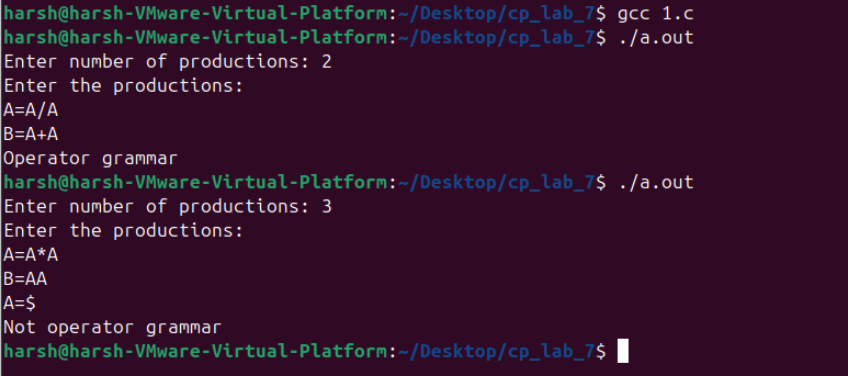
if (flag == 1)

printf("Operator grammar\n");

return 0;

}

Output-



1. WAP to implement Shift-Reduce Parser in C

Sol-

Code-

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int z = 0, i = 0, j = 0, c = 0;

char a[16], ac[20], stk[15], act[10];

void check() {

strcpy(ac, "REDUCE TO E -> ");

for (z = 0; z < c; z++) {

if (stk[z] == '4') {

printf("%s4", ac);

stk[z] = 'E';

stk[z + 1] = '\0';

printf("\n$%s\t%s$\t", stk, a);

}

}

for (z = 0; z < c - 2; z++) {

if (stk[z] == '2' && stk[z + 1] == 'E' && stk[z + 2] == '2') {

printf("%s2E2", ac);

stk[z] = 'E';

stk[z + 1] = '\0';

stk[z + 2] = '\0';

printf("\n$%s\t%s$\t", stk, a);

i = i - 2;

}

}

for (z = 0; z < c - 2; z++) {

if (stk[z] == '3' && stk[z + 1] == 'E' && stk[z + 2] == '3') {

printf("%s3E3", ac);

stk[z] = 'E';

stk[z + 1] = '\0';

stk[z + 1] = '\0';

printf("\n$%s\t%s$\t", stk, a);

i = i - 2;

}

}

}

int main() {

printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");

strcpy(a, "32423");

c = strlen(a);

strcpy(act, "SHIFT");

printf("\nstack \t input \t action");

printf("\n$\t%s$\t", a);

for (i = 0; j < c; i++, j++) {

printf("%s", act);

stk[i] = a[j];

stk[i + 1] = '\0';

a[j] = ' ';

printf("\n$%s\t%s$\t", stk, a);

check();

}

check();

if (stk[0] == 'E' && stk[1] == '\0')

printf("Accept\n");

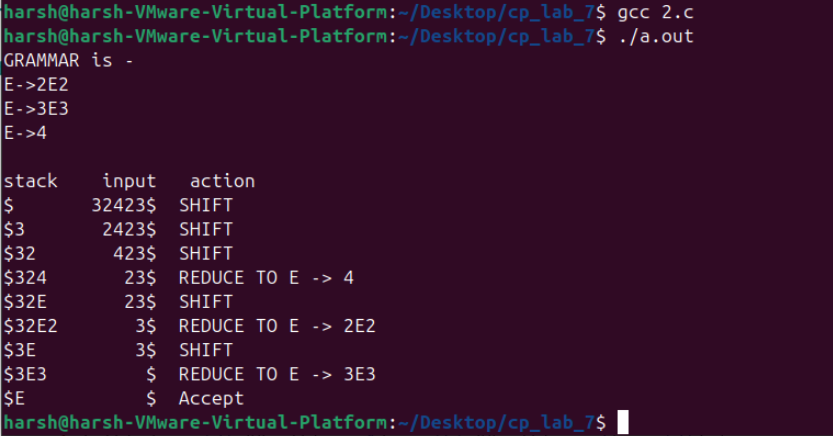
else

printf("Reject\n");

return 0;

}

Output-



1. WAP to derive a LALR Bottom Up Parser for the given language in C

Sol-

#include <stdio.h>

#include <string.h>

#define MAX 100

char stack[MAX];

int top = -1;

char input[MAX];

int ip = 0;

// Predefined grammar rules

const char \*grammar[] = {

"E->E+E", // Rule 0

"E->E\*E", // Rule 1

"E->(E)", // Rule 2

"E->i" // Rule 3

};

void push(char ch) {

stack[++top] = ch;

}

void pop(int count) {

top -= count;

}

void display() {

for (int i = 0; i <= top; i++)

printf("%c", stack[i]);

}

void shift() {

push(input[ip]);

ip++;

}

int reduce() {

if (top >= 2 && stack[top - 2] == 'E' && stack[top - 1] == '\*' && stack[top] == 'E') {

printf("Reduce using: %s", grammar[1]); // E -> E \* E

pop(2);

stack[top] = 'E';

return 1;

}

if (top >= 2 && stack[top - 2] == 'E' && stack[top - 1] == '+' && stack[top] == 'E') {

printf("Reduce using: %s", grammar[0]); // E -> E + E

pop(2);

stack[top] = 'E';

return 1;

}

if (top >= 2 && stack[top - 2] == '(' && stack[top - 1] == 'E' && stack[top] == ')') {

printf("Reduce using: %s", grammar[2]); // E -> (E)

pop(2);

stack[top] = 'E';

return 1;

}

if (top >= 0 && stack[top] == 'i') {

printf("Reduce using: %s", grammar[3]); // E -> i

stack[top] = 'E';

return 1;

}

return 0;

}

void parse() {

printf("\nStack\t\tInput\t\tAction\n");

while (1) {

printf("\n");

display();

printf("\t\t%s\t\t", &input[ip]);

if (top == 0 && stack[top] == 'E' && input[ip] == '$') {

printf("Accept\n");

return;

}

if (input[ip] != '$') {

printf("Shift");

shift();

}

while (reduce()); // Keep reducing until no further reduction is possible

}

}

int main() {

printf("Predefined Grammar Rules:\n");

for (int i = 0; i < 4; i++) {

printf("%s\n", grammar[i]);

}

printf("\nEnter the input string: ");

scanf("%s", input);

strcat(input, "$");

parse();

return 0;

}

Output-

