11.**SLR**

Program(1):

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<unistd.h>

int i, j, k, m, n=0, o, p, ns=0, tn=0, rr=0, ch=0;

char cread[15][10], gl[15], gr[15][10], temp, templ[15], tempr[15][10], \*ptr, temp2[5];

char dfa[15][10];

struct states {

char lhs[15], rhs[15][10];

int n;

} I[15];

int compstruct(struct states s1, struct states s2) {

int t;

if(s1.n != s2.n)

return 0;

if(strcmp(s1.lhs, s2.lhs) != 0)

return 0;

for(t = 0; t < s1.n; t++)

if(strcmp(s1.rhs[t], s2.rhs[t]) != 0)

return 0;

return 1;

}

void moreprod() {

int r, s, t, l1 = 0, rr1 = 0;

char \*ptr1, read1[15][10];

for(r = 0; r < I[ns].n; r++) {

ptr1 = strchr(I[ns].rhs[l1], '.');

t = ptr1 - I[ns].rhs[l1];

if(t + 1 == strlen(I[ns].rhs[l1])) {

l1++;

continue;

}

temp = I[ns].rhs[l1][t + 1];

l1++;

for(s = 0; s < rr1; s++)

if(temp == read1[s][0])

break;

if(s == rr1) {

read1[rr1][0] = temp;

rr1++;

} else

continue;

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

if(gl[s] == temp) {

I[ns].rhs[I[ns].n][0] = '.';

I[ns].rhs[I[ns].n][1] = '\0';

strcat(I[ns].rhs[I[ns].n], gr[s]);

I[ns].lhs[I[ns].n] = gl[s];

I[ns].lhs[I[ns].n + 1] = '\0';

I[ns].n++;

}

}

}

}

void canonical(int l) {

int t1;

char read1[15][10], rr1 = 0, \*ptr1;

for(i = 0; i < I[l].n; i++) {

temp2[0] = '.';

ptr1 = strchr(I[l].rhs[i], '.');

t1 = ptr1 - I[l].rhs[i];

if(t1 + 1 == strlen(I[l].rhs[i]))

continue;

temp2[1] = I[l].rhs[i][t1 + 1];

temp2[2] = '\0';

for(j = 0; j < rr1; j++)

if(strcmp(temp2, read1[j]) == 0)

break;

if(j == rr1) {

strcpy(read1[rr1], temp2);

read1[rr1][2] = '\0';

rr1++;

} else

continue;

for(j = 0; j < I[0].n; j++) {

ptr = strstr(I[l].rhs[j], temp2);

if(ptr) {

templ[tn] = I[l].lhs[j];

templ[tn + 1] = '\0';

strcpy(tempr[tn], I[l].rhs[j]);

tn++;

}

}

for(j = 0; j < tn; j++) {

ptr = strchr(tempr[j], '.');

p = ptr - tempr[j];

tempr[j][p] = tempr[j][p + 1];

tempr[j][p + 1] = '.';

I[ns].lhs[I[ns].n] = templ[j];

I[ns].lhs[I[ns].n + 1] = '\0';

strcpy(I[ns].rhs[I[ns].n], tempr[j]);

I[ns].n++;

}

moreprod();

for(j = 0; j < ns; j++) {

if(compstruct(I[ns], I[j]) == 1) {

I[ns].lhs[0] = '\0';

for(k = 0; k < I[ns].n; k++)

I[ns].rhs[k][0] = '\0';

I[ns].n = 0;

dfa[l][j] = temp2[1];

break;

}

}

if(j < ns) {

tn = 0;

for(j = 0; j < 15; j++) {

templ[j] = '\0';

tempr[j][0] = '\0';

}

continue;

}

dfa[l][j] = temp2[1];

printf("\n\nI%d :", ns);

for(j = 0; j < I[ns].n; j++)

printf("\n\t%c -> %s", I[ns].lhs[j], I[ns].rhs[j]);

ns++;

tn = 0;

for(j = 0; j < 15; j++) {

templ[j] = '\0';

tempr[j][0] = '\0';

}

}

}

void main() {

int l;

for(i = 0; i < 15; i++) {

I[i].n = 0;

I[i].lhs[0] = '\0';

I[i].rhs[0][0] = '\0';

dfa[i][0] = '\0';

}

printf("Enter the number of productions: ");

scanf("%d", &n);

printf("Enter the grammar productions (e.g., S -> S+T or S T):\n");

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

printf("LHS for production %d: ", i + 1);

scanf(" %c", &gl[i]);

printf("RHS for production %d: ", i + 1);

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

}

printf("THE GRAMMAR IS AS FOLLOWS\n");

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

printf("\t\t\t\t%c -> %s\n", gl[i], gr[i]);

I[0].lhs[0] = 'Z';

strcpy(I[0].rhs[0], ".S");

I[0].n++;

l = 0;

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

temp = I[0].rhs[l][1];

l++;

for(j = 0; j < rr; j++)

if(temp == cread[j][0])

break;

if(j == rr) {

cread[rr][0] = temp;

rr++;

} else

continue;

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

if(gl[j] == temp) {

I[0].rhs[I[0].n][0] = '.';

strcat(I[0].rhs[I[0].n], gr[j]);

I[0].lhs[I[0].n] = gl[j];

I[0].n++;

}

}

}

ns++;

printf("\nI%d :\n", ns - 1);

for(i = 0; i < I[0].n; i++)

printf("\t%c -> %s\n", I[0].lhs[i], I[0].rhs[i]);

for(l = 0; l < ns; l++)

canonical(l);

printf("\n\n\t\tDFA TABLE IS AS FOLLOWS\n\n\n");

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

printf("I%d : ", i);

for(j = 0; j < ns; j++)

if(dfa[i][j] != '\0')

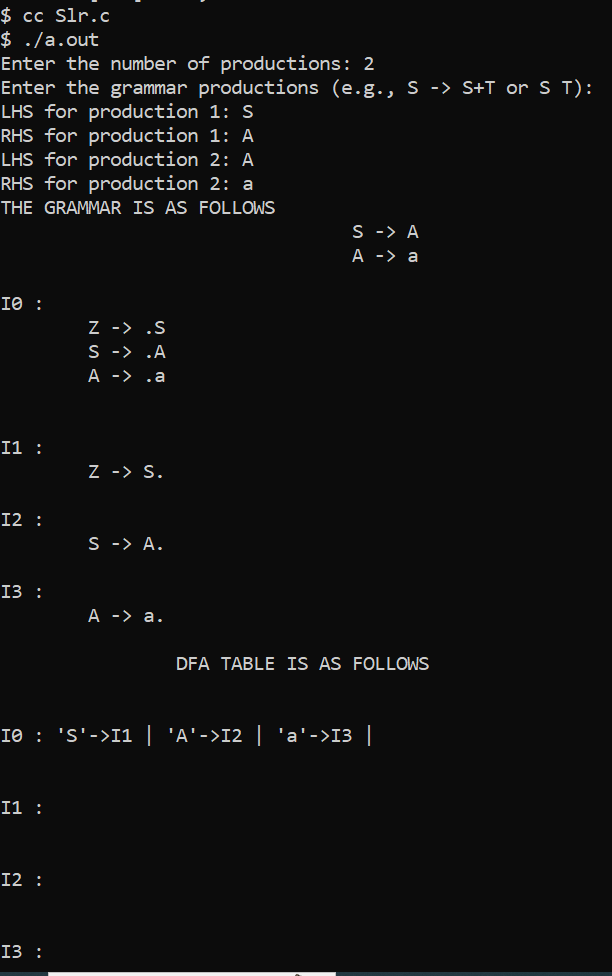
printf("'%c'->I%d | ", dfa[i][j], j);

printf("\n\n\n");

}

}

Output:



Program(2):

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

char terminals[100] = {};

int no\_t;

char non\_terminals[100] = {};

int no\_nt;

char goto\_table[100][100];

char reduce[20][20];

char follow[20][20];

char fo\_co[20][20];

char first[20][20];

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");

else {

strcpy(S->prod[0], "S->.");

S->prod[0][4] = I->prod[0][0];

}

S->prod\_count++;

}

void get\_prods(struct state \*I) {

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

scanf("%d", &I->prod\_count);

printf("Enter the number of non terminals:\n");

scanf("%d", &no\_nt);

printf("Enter the non terminals one by one:\n");

for (int i = 0; i < no\_nt; i++)

scanf(" %c", &non\_terminals[i]);

printf("Enter the number of terminals:\n");

scanf("%d", &no\_t);

printf("Enter the terminals (single lettered) one by one:\n");

for (int i = 0; i < no\_t; i++)

scanf(" %c", &terminals[i]);

printf("Enter the productions one by one in form (S->ABc):\n");

for (int i = 0; i < I->prod\_count; i++) {

scanf("%s", I->prod[i]);

}

}

int is\_non\_terminal(char a) {

return (a >= 'A' && a <= 'Z');

}

int in\_state(struct state \*I, char \*a) {

for (int i = 0; i < I->prod\_count; i++) {

if (!strcmp(I->prod[i], a))

return 1;

}

return 0;

}

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 '\0';

}

char\* move\_dot(char b[100], int len) {

static char a[100];

strcpy(a, b);

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

if (a[i] == '.') {

char temp = a[i];

a[i] = a[i + 1];

a[i + 1] = temp;

break;

}

}

return a;

}

int same\_state(struct state \*I0, struct state \*I) {

if (I0->prod\_count != I->prod\_count)

return 0;

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 0;

}

return 1;

}

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

char a;

for (int i = 0; i < I0->prod\_count; i++) {

a = char\_after\_dot(I0->prod[i]);

if (is\_non\_terminal(a)) {

for (int j = 0; j < I->prod\_count; j++) {

if (I->prod[j][0] == a) {

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]);

printf("\n");

}

int in\_array(char a[20], char b) {

for (int i = 0; i < strlen(a); i++)

if (a[i] == b)

return 1;

return 0;

}

char\* chars\_after\_dots(struct state \*I) {

static char a[20];

memset(a, 0, sizeof(a));

for (int i = 0; i < I->prod\_count; i++) {

if (!in\_array(a, char\_after\_dot(I->prod[i]))) {

a[strlen(a)] = char\_after\_dot(I->prod[i]);

}

}

return a;

}

void cleanup\_prods(struct state \*I) {

char a[100] = {};

for (int i = 0; i < I->prod\_count; i++)

strcpy(I->prod[i], a);

I->prod\_count = 0;

}

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) {

printf("\n\*\*\*\*\*\*\*\*Shift Actions\*\*\*\*\*\*\*\*\*\n\n");

printf("\t");

for (int i = 0; i < no\_t; i++)

printf("%c\t", terminals[i]);

for (int i = 0; i < no\_nt; i++)

printf("%c\t", non\_terminals[i]);

printf("\n");

for (int i = 0; i < state\_count; i++) {

int arr[no\_nt + no\_t];

memset(arr, -1, sizeof(arr));

for (int j = 0; j < state\_count; j++) {

if (goto\_table[i][j] != '~') {

arr[return\_index(goto\_table[i][j])] = j;

}

}

printf("I%d\t", i);

for (int j = 0; j < no\_nt + no\_t; j++) {

if (i == 1 && j == no\_t - 1)

printf("ACC\t");

if (arr[j] == -1 || arr[j] == 0)

printf("\t");

else {

if (j < no\_t)

printf("S%d\t", arr[j]);

else

printf("%d\t", arr[j]);

}

}

printf("\n");

}

}

int get\_index\_char(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\_set(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\_set(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(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) {

printf("\*\*\*\*\*\*\*\*\*\*Reduce actions\*\*\*\*\*\*\*\*\*\*\n\n");

printf("\t");

int arr[temp1->prod\_count][no\_t];

memset(arr, -1, sizeof(arr));

for (int i = 0; i < no\_t; i++) {

printf("%c\t", terminals[i]);

}

printf("\n");

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;

}

}

printf("I%d\t", n);

for (int j = 0; j < no\_t; j++) {

if (arr[i][j] != -1 && arr[i][j] != 0 && arr[i][j] < state\_count)

printf("R%d\t", arr[i][j]);

else

printf("\t");

}

printf("\n");

}

}

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]);

printf("\nI0:\n");

print\_prods(&I[0]);

char characters[20] = {};

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;

printf("I%d on reading the symbol %c goes to I%d.\n", i, characters[j], k);

goto\_table[i][k] = characters[j];

break;

}

}

if (flag == 0) {

state\_count++;

printf("I%d on reading the symbol %c goes to I%d:\n", i, characters[j], state\_count - 1);

goto\_table[i][state\_count - 1] = characters[j];

print\_prods(&I[state\_count - 1]);

}

}

}

int no\_re[temp.prod\_count];

memset(no\_re, -1, sizeof(no\_re));

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);

}

}

}

}

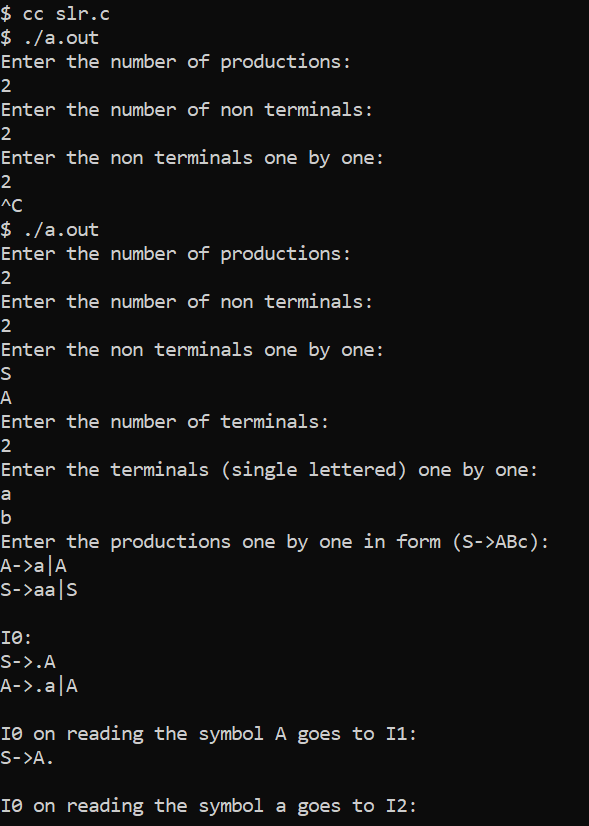
print\_shift\_table(state\_count);

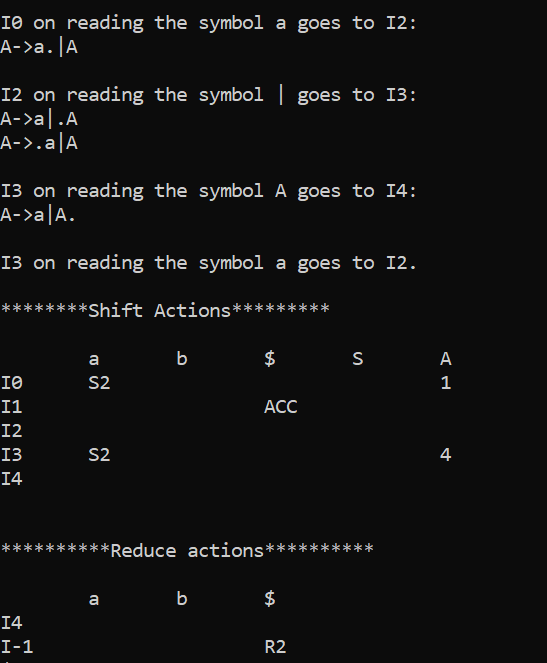
printf("\n\n");

print\_reduce\_table(state\_count, no\_re, &temp1);

}

Output:



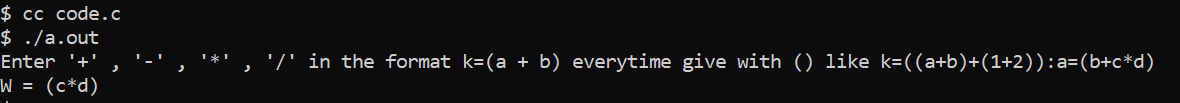


12.**Three Address Code**

Program(1):

#include<stdio.h>  
#include<string.h>  
  
char string[100];  
char temp\_char[]={'W','E','M','P'};  
int t\_count=0;  
int is\_variable(char check){  
                if(check >= 'a' && check <= 'z' || check >='A' && check <='Z' || check >='0' && check <= '9')  
                       return 1;  
               return 0;  
}  
int is\_assignment(char check){  
        return check == '=';  
}  
int is\_operator(char check){  
        return check == '+' || check == '-' || check == '\*' || check == '/' || check == '%';  
}  
int replace(char string[100],int i ,char \*stack ,int \*top,int to\_be\_runned){  
  
                printf("%c = (",temp\_char[t\_count]);  
                int j=2;  
                while(j>=0){  
                        printf("%c",stack[\*top-j]);  
                        stack[\*top-j]='\0';  
                        j--;  
                }  
  
                printf(")\n");  
                \*top=\*top-2;  
                stack[\*top] =temp\_char[t\_count++];  
  
}int main(){  
        printf("Enter '+' , '-' , '\*' , '/' in the format k=(a + b) everytime give with () like k=((a+b)+(1+2)):");  
        scanf(" %s",string);  
        char stack[10];  
int top=-1;  
        if(!is\_variable(string[0])){  
                printf("Error\n");  
        }  
        if(!is\_assignment(string[1])){  
                printf(" assignment operator error !\n");  
        }  
        if(string[2] == '\0'){  
                return 0;  
        }  
        stack[++top] = string[2];  
        int i=2;  
        int to\_be\_runned=0;  
        while((top>-1||string[i] !='\0')){  
                if(string[i] == '('){  
                        to\_be\_runned++;  
                        i++;  
                }  
                else if(is\_variable(string[i]) || is\_operator(string[i])){  
                        stack[++top] = string[i++];  
                }  
                else if(string[i] == ')'){  
                        replace(string , i ,stack ,&top,to\_be\_runned);  
                        to\_be\_runned--;  
                        i++;  
                }  
  
                if(to\_be\_runned == 0){  
                        break;}  
  
        }  
  
}

Output:



Program(2):

#include <stdio.h>

#include <string.h>

#include <ctype.h>

int temp\_count = 1;

void process\_expression(char operand1, char operator, char operand2) {

printf("t%d = %c %c %c\n", temp\_count, operand1, operator, operand2);

temp\_count++;

}

void parse\_expression(char\* expr) {

printf("t%d = c \* d\n", temp\_count);

int t1 = temp\_count++;

printf("t%d = b + t%d\n", temp\_count, t1);

int t2 = temp\_count++;

printf("t%d = t2\n", temp\_count);

}

int main() {

printf("Enter expression: ");

char string[100];

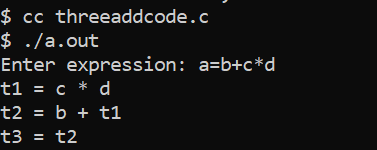
scanf(" %[^\n]", string);

parse\_expression(string);

return 0;

}

Output:



13.**Code Optimization**

Program:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_INSTR 100

#define MAX\_LEN 50

typedef struct {

char lhs[MAX\_LEN];

char rhs[MAX\_LEN];

} Instruction;

Instruction instructions[MAX\_INSTR];

int instrCount = 0;

void eliminateCommonExpressions() {

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

for (int j = i + 1; j < instrCount; j++) {

if (strcmp(instructions[i].rhs, instructions[j].rhs) == 0) {

strcpy(instructions[j].rhs, instructions[i].lhs);

}

}

}

}

void eliminateDeadCode() {

int used[MAX\_INSTR] = {0};

for (int i = instrCount - 1; i >= 0; i--) {

for (int j = 0; j < instrCount; j++) {

if (strcmp(instructions[i].lhs, instructions[j].rhs) == 0) {

used[i] = 1;

break;

}

}

}

int newInstrCount = 0;

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

if (used[i]) {

instructions[newInstrCount++] = instructions[i];

}

}

instrCount = newInstrCount;

}

void displayCode() {

printf("\nIntermediate code:\n");

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

printf("%s = %s\n", instructions[i].lhs, instructions[i].rhs);

}

}

int main() {

printf("Enter the number of instructions: ");

scanf("%d", &instrCount);

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

printf("Enter LHS value: ");

scanf("%s", instructions[i].lhs);

printf("Enter RHS value: ");

scanf("%s", instructions[i].rhs);

}

displayCode();

eliminateCommonExpressions();

printf("\nAfter eliminating common expressions:\n");

displayCode();

eliminateDeadCode();

printf("\nAfter Dead Code Elimination:\n");

displayCode();

return 0;

}

Output:

