**Exp9:** **Write a C Program for implementation of LR Parsing algorithm to accept a given input string.**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

//Global Variables

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

// Modify array size to increase

// length of string to be parsed

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

// This Function will check whether

// the stack contain a production rule

// which is to be Reduce.

// Rules can be E->2E2 , E->3E3 , E->4

void check()

{

// Copying string to be printed as action

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

// c=length of input string

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

{

//checking for producing rule E->4

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

{

printf("%s4", ac);

stk[z] = 'E';

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

//printing action

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

}

}

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

{

//checking for another production

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$%s\t\t%s$\t", stk, a);

i = i - 2;

}

}

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

{

//checking for E->3E3

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

stk[z + 2] == '3')

{

printf("%s3E3", ac);

stk[z]='E';

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

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

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

i = i - 2;

}

}

return ; //return to main

}

//Driver Function

int main()

{

char str[50];

printf("Consider the CFG -\nE->2E2 \nE->3E3 \nE->4\n");

printf("\nEnter input string from given CFG: ");

scanf("%s",str);

// a is input string

strcpy(a,str);

// strlen(a) will return the length of a to c

c=strlen(a);

// "SHIFT" is copied to act to be printed

strcpy(act,"SHIFT");

// This will print Labels (column name)

printf("\nStack \t\tInput \t\t Action");

// This will print the initial

// values of stack and input

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

// This will Run upto length of input string

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

{

// Printing action

printf("\t%s", act);

// Pushing into stack

stk[i] = a[j];

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

// Moving the pointer

a[j]=' ';

// Printing action

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

// Call check function ..which will

// check the stack whether its contain

// any production or not

check();

}

// Rechecking last time if contain

// any valid production then it will

// replace otherwise invalid

check();

// if top of the stack is E(starting symbol)

// then it will accept the input

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

printf("Accept\n");

else //else reject

printf("Reject\n");

}

**OUTPUT1**

GRAMMAR is -

E->2E2

E->3E3

E->4

Enter input string from given CFG: 2324232

stack input action

$ 2324232$ SHIFT

$2 324232$ SHIFT

$23 24232$ SHIFT

$232 4232$ SHIFT

$2324 232$ REDUCE TO E -> 4

$232E 232$ SHIFT

$232E2 32$ REDUCE TO E -> 2E2

$23E 32$ SHIFT

$23E3 2$ REDUCE TO E -> 3E3

$2E 2$ SHIFT

$2E2 $ REDUCE TO E -> 2E2

$E $ Accept

GRAMMAR is -

E->2E2

E->3E3

E->4

**OUTPUT2**

Enter input string from given CFG: 1234

stack input action

$ 1234$ SHIFT

$1 234$ SHIFT

$12 34$ SHIFT

$123 4$ SHIFT

$1234 $ REDUCE TO E -> 4

$123E $ Reject

=== Code Execution Successful ===

**Exp10: Write a C program for the implementation of a Shift Reduce Parser using Stack Data Structure to accept a given input string of a given grammar.**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

char ip\_sym[15],stack[15]; int ip\_ptr=0,st\_ptr=0,len,i; char temp[2],temp2[2]; char act[15];

void check();

int main()

{

printf("\n\n\t Shift Reduce Parser\n");

printf("\n\t\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\*");

printf("\nGrammar\n");

printf("E->E+E\nE->E/E\n");

printf("E->E\*E\nE->a/b");

printf("\nEnter the Input Symbol:");

scanf("%s",ip\_sym);

printf("\n\n\t Stack Implementation Table");

printf("\n Stack\t\t Input Symbol\t\t Action");

printf("\n $\t\t\t%s$\t\t\t --",ip\_sym);

strcpy(act,"shift");

temp[0]=ip\_sym[ip\_ptr];

temp[1]='\0';

strcat(act,temp);

len=strlen(ip\_sym);

for(i=0;i<=len-1;i++)

{

stack[st\_ptr]=ip\_sym[ip\_ptr];

stack[st\_ptr+1]='\0';

ip\_sym[ip\_ptr]=' ';

ip\_ptr++;

printf("\n$%s\t\t%s$\t\t\t%s", stack,ip\_sym,act);

strcpy(act,"shift");

temp[0]=ip\_sym[ip\_ptr];

temp[1]='\0';

strcat(act,temp);

check();

st\_ptr++;

}

st\_ptr++;

check();

return 0;

}

void check()

{

int flag=0;

temp2[0]=stack[st\_ptr];

temp[1]='\0';

if((!strcmp(temp2,"a"))||(!strcmp(temp2,"b")))

{

stack[st\_ptr]='E';

if(!strcmp(temp2,"a"))

printf("\n$%s\t\t%s$\t\t\tE->a",stack,ip\_sym);

else printf("\n$%s\t\t%s$\t\t\tE->a",stack,ip\_sym);

flag=1;

}

if((!strcmp(temp2,"+"))||(strcmp(temp2,"\*"))||(!strcmp(temp2,"/")))

{

flag=1;

}

if((!strcmp(stack,"E+E"))||(!strcmp(stack,"E/E"))||(!strcmp(stack,"E\*E")))

{

strcpy(stack,"E");

st\_ptr=0;

if(!strcmp(stack,"E+E"))

printf("\n$%s\t\t%s$\t\t\tE->E+E",stack,ip\_sym);

else

if(!strcmp(stack,"E/E"))

printf("\n$%s\t\t\t%s$\t\tE->E/E",stack,ip\_sym);

else printf("\n$%s\t\t%s$\t\t\tE->E\*E",stack,ip\_sym);

flag=1;

}

if(!strcmp(stack,"E")&&ip\_ptr==len)

{

printf("\n$%s\t\t%s$\t\t\tAccept",ip\_sym);

exit(0);

}

if(flag==0)

{

printf("\n %s \t\t\t %s \t\t Reject",stack,ip\_sym);

}

return;

}

**OUTPUT1**

Shift Reduce Parser

\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\*

Grammar

E->E+E

E->E/E

E->E\*E

E->a/b

Enter the Input Symbol:a+b$

Stack Implementation Table

Stack Input Symbol Action

$ a+b$$ --

$a +b$$ shifta

$E +b$$ E->a

$E+ b$$ shift+

$E+b $$ shiftb

$E+E $$ E->a

$E $$ E->E\*E

$E$ $ shift$

=== Code Execution Successful ===