**LAB RECORD**

**BACHELOR OF TECHNOLOGY**

**B.Tech. CS&E \_\_\_\_\_ Semester ( VI )**

**(Academic Session – 2021 - 22 )**

**Course Title : Compiler Construction**

**Course Code : CSE304**

**Enrollment No. : A7605220004**

**Name of Student : Humra Khan**

**Date of Submission : \_\_ \_\_ / \_\_ \_\_ /\_\_ \_\_ \_\_ \_\_**

**Signature of Student :**

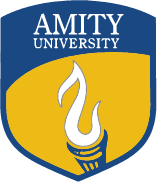
**Grade/Marks Obtained :**

**Faculty Name & Signature : Dr. Pawan Singh**

**Department of Computer Science & Engineering**

**Amity School of Engineering & Technology**

**Amity University, Lucknow Campus**



Department of Computer Science & Engineering

Amity School of Engineering & Technology

Amity University, Lucknow Campus

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| **Lab Session** | **Date** | **Signature of Faculty** | **Remarks** |
| 1.Write a program in C/C++ to accept the regular expression-  i. ab\*  ii. a\* + ab\*  iii. (a\* + ab\*)\* + abc  iv. (a/b)\*abb | 23.1.23 |  |  |
| 2. Write a program in C/C++ to design a lexical analyzer to identify integer, operators, keywords. | 23.2.23 |  |  |
| 3. i. Write C programs to eliminate left recursion from the production of grammar provided by the user.  Ii. Write C programs to eliminate left recursion from the grammar provided by the user. | 13.2.23 |  |  |
| 4. i. Program for implementation of Recursive Descent Parser for language given below  E-> TE’  E’-> +TE’ | -TE’ | null  T-> FT’  T’-> \*FT’| /FT’ | null  F-> id/ (E)/ num  ii. To perform recursive descent parsing using multi characters and whitespace in input. | 3.4.23 |  |  |
| 6. |  |  |  |
| 7. |  |  |  |
| 8 |  |  |  |
| 9. |  |  |  |
| 10. |  |  |  |

**PROGRAM 1**

**i)**

**Aim:-** Write a program to accept regular expression ab\*

**Software used:-** onlinegdb.com (Online compiler)

**Procedure:-** Go to debug -> run or press CTRL + F9 to run the program.

#include <stdio.h>

#include <string.h>

int main()

{

    char s[20];

    int state=0, i=0;

    printf("Enter: ");

    gets(s);

    while(s[i]!='\0')

    {

        switch(state)

        {

            case 0:

            if(s[i]=='a')

            {

                state++;

                i++;

            }

            else

                state=3;

            break;

            case 1:

            if(s[i]=='b')

            {

                i++;

            }

            else

            {

                state=3;

            }

            break;

        }

        if(state==3)

        break;

    }

    if(state==3)

        printf("not Accepted");

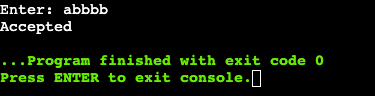
    else

        printf("Accepted");

    return 0;

}

**Output-**



**ii)**

**Aim:-** Write a program to accept regular expression a\* + ab\*

**Software used:-** onlinegdb.com (Online compiler)

**Procedure:-** Go to debug -> run or press CTRL + F9 to run the program.

#include <stdio.h>

#include <string.h>

int main()

{

char s[20];

int state=0, i=0;

printf("Enter: ");

gets(s);

while(s[i]!='\0')

{

switch(state)

{

case 0:

if(s[i]=='a')

{

i++;

if(s[i]=='a')

{

state=1;

}

else if(s[i]=='b')

{

state=2;

}

}

else

state=5;

break;

case 1:

if(s[i]=='a')

{

i++;

}

else

{

state=5;

}

break;

case 2:

if(s[i]=='b')

{

i++;

}

else

{

state=5;

}

break;

}

if(state==5)

break;

}

if(state==5)

printf("not Accepted");

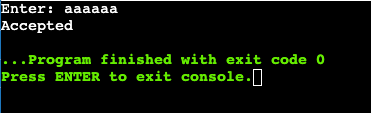
else

printf("Accepted");

return 0;

}

**Output-**



**iii)**

**Aim:-** Write a program to accept regular expression (a\* + ab\*)\* + abc

**Software used:-** onlinegdb.com (Online compiler)

**Procedure:-** Go to debug -> run or press CTRL + F9 to run the program.

#include <stdio.h>

#include <string.h>

int main()

{

char s[20];

int state=0, i=0;

printf("Enter: ");

gets(s);

while(s[i]!='\0')

{

switch(state)

{

case 0:

if(s[i]=='a')

{

if(s[0]=='a' && s[1]=='b' && s[2]=='c')

{

i=i+3;

}

i++;

if(s[i]=='b')

{

state=1;

}

}

else

state=5;

break;

case 1:

if(s[i]=='b')

{

i++;

if(s[i]=='a')

{

state=0;

}

}

else

{

state=5;

}

break;

}

if(state==5)

break;

}

if(state==5)

printf("not Accepted");

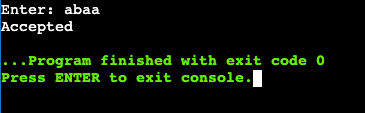
else

printf("Accepted");

return 0;

}

**Output-**



**iv)**

**Aim:-** Write a program to accept regular expression (a + b)\*abb

**Software used:-** onlinegdb.com (Online compiler)

**Procedure:-** Go to debug -> run or press CTRL + F9 to run the program.

#include <stdio.h>

#include <string.h>

int main()

{

char s[20];

int state=0, i=0;

printf("Enter: ");

gets(s);

int l=strlen(s);

while(s[i]!='\0')

{

switch(state)

{

case 0:

if(i==l-4)

state=1;

if(s[i]=='a')

{

i++;

if(s[i]=='b')

{

state=0;

}

}

else if(s[i]=='b')

{

i++;

if(s[i]=='a')

{

state=0;

}

}

else

state=5;

break;

case 1:

if(s[i]=='a')

{

i++;

state=2;

}

else

{

state=5;

}

break;

case 2:

if(s[i]=='b')

{

i++;

state=3;

}

else

{

state=5;

}

break;

case 3:

if(s[i]=='b')

{

i++;

}

else

{

state=5;

}

break;

}

if(state==5)

break;

}

if(state==5)

printf("not Accepted");

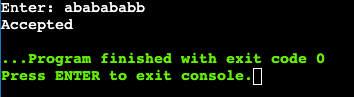
else

printf("Accepted");

return 0;

}

**Output-**



**PROGRAM 2**

**Aim:-** Write a program in C/C++ to design a lexical analyzer to identify integer, operators, keywords.

**Software used:-** onlinegdb.com (Online compiler)

**Procedure:-** Go to debug -> run or press CTRL + F9 to run the program.

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<ctype.h>

int fail(int);

void idorkeyword(char str[]);

void main(void)  
{ int i,j,state,l;  
char s[100],temp[10],c; i=0;j=0;state=0;l=0; clrscr(); printf("Enter the expression:"); scanf("%[^\n]",s);

l=strlen(s);  
while(i<=l)  
{  
switch(state)

{  
case 0:

c=s[i];  
if(c==' ')

{state=0;i++;}  
else if(c=='<')

{state=1;i++;}

else if(c=='=')

{state=5;i++;}

else if(c=='>')

{state=6;i++;}

else state=fail(state);

break;

case 1: c=s[i];

if(c=='=')

{state=2;i++;}

else if(c=='>')

{state=3;i++;}

else state=4;

break;

case 2:

printf("RELOP\_LE ");

i++;state=9;

break;

case 3:

printf("RELOP\_NE ");

i++;state=9;

break;

case 4:

printf("RELOP\_LT ");

state=9;

break;

case 5:

printf("RELOP\_EQ ");

i++;state=9;

break;

case 6:

c=s[i]; i

f(c=='=')

{state=7;i++;}

else state=8;

break;

case 7:

printf("RELOP\_GE ");

i++;state=9;break;

case 8:

printf("RELOP\_GT ");

state=9;

break;

case 9:

c=s[i];

if(isalpha(c))

{state=10;i++;temp[j]=c;}

else state=fail(state);

break;

case 10:

c=s[i];

if(isalpha(c))

{state=10;i++;j++;temp[j]=c;}

else if(isdigit(c))

{state=10;i++;j++;temp[j]=c;}

else

state=11;

break;

case 11:

j++;temp[j]='\0';i

dorkeyword(temp);

j=0;

state=12;

break;

case 12:

c=s[i];

if(isdigit(c))

{state=13;i++;}

else

state=fail(state);

break;

case 13:

c=s[i];

if(isdigit(c))

{state=13;i++;}

else if(c=='.')

{state=14;i++;}

else if(c=='E')

{state=16;i++;}

else state=19;

break;

case 14:

c=s[i];

if(isdigit(c))

{state=15;i++;}

else

printf("Error ");

break;

case 15:

c=s[i];

if(isdigit(c))

{state=15;i++;}

else if(c=='E')

{state=16;i++;}

else state=19;

break;

case 16: c=s[i];

if(c=='+'||c=='-')

{state=17;i++;}

else if(isdigit(c))

{state=18;i++;}

else printf("Error ");

break;  
case 17:

c=s[i];

if(isdigit(c))

{state=18;i++;}

else printf("Error ");

break;

case 18:

c=s[i];

if(isdigit(c)){state=18;i++;}

else

state=19;break;

case 19:

printf("NUM ");

state=0;

break; }

}  
getch();

}

int fail(int start)  
{  
switch(start)

{  
case 0:

start=9;

break;

case 9:

start=12;

break;

case 12:

start=0;break;

}

return(start);

}

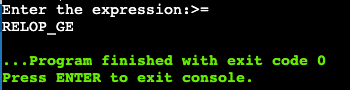
void idorkeyword(char str[10])  
{ char \*key1="if",\*key2="then",\*key3="else";  
if ( strcmp(str,key1)==0||strcmp(str,key2)==0||strcmp(str,key3)==0)

printf("%s ",str);  
else

printf("ID ");

}

**Output-**



**PROGRAM 3**

**i)Aim:-** Write C programs to eliminate left recursion from productions of the grammar provided by the user.

**Software used:-** onlinegdb.com (Online compiler)

**Procedure:-** Go to debug -> run or press CTRL + F9 to run the program.

**PROGRAM LOGIC:-** A grammar is left recursive if it has a nonterrninal A such that there is a derivation A -> Ac for some string a. Top-down parsing methods cannot handle left-recursive grammars, so it transformation that eliminates left recursion is needed. A left-recursive pair of productions

A -> Aα | β could be replaced by the non-left-recursive productions

A -> βA'

A' -> αA' | ε

#include<stdio.h>

#include<string.h>

void main() {

char input[100],\*l,\*r,\*temp,tempprod[20],productions[25][50];

int i=0,j=0,flag=0;

printf("Enter the productions: ");

scanf("%s",input);

l = strtok(input,"->");

r = strtok(NULL,"->");

temp = strtok(r,"|");

while(temp) {

if(temp[0] == l[0]) {

flag = 1;

sprintf(productions[i++],"%s'->%s%s'\0",l,temp+1,l);

}

else

sprintf(productions[i++],"%s->%s%s'\0",l,temp,l);

temp = strtok(NULL,"|");

}

sprintf(productions[i++],"%s’-> \356\0",l);

if(flag == 0)

printf("The given productions don't have Left Recursion");

else

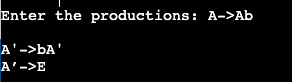
for(j=0;j<i;j++) {

printf("\n%s",productions[j]);

}

}

**Output-**



**ii)Aim:-** Write C programs to eliminate left recursion from the grammar provided by the user.

**Software used:-** onlinegdb.com (Online compiler)

**Procedure:-** Go to debug -> run or press CTRL + F9 to run the program.

**PROGRAM LOGIC:-** A grammar is left recursive if it has a nonterrninal A such that there is a derivation A -> Ac for some string a. Top-down parsing methods cannot handle left-recursive grammars, so it transformation that eliminates left recursion is needed. A left-recursive pair of productions

A -> Aα | β could be replaced by the non-left-recursive productions

A -> βA'

A' -> αA' | ε

#include<stdio.h>

#include<string.h>

void main() {

char \*l,\*r,\*temp,tempprod[20],productions[25][50];

int i=0,j=0,flag=0, n,k=0;

printf("\nEnter the No. of productions: ");

scanf("%d", &n);

char input[n][100];

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

{

printf("\nEnter the productions: ");

scanf("%s",input[k]);

}

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

{

l = strtok(input[k],"->");

r = strtok(NULL,"->");

temp = strtok(r,"|");

while(temp) {

if(temp[0] == l[0]) {

flag = 1;

sprintf(productions[i++],"%s'->%s%s'\0",l,temp+1,l);

}

else

sprintf(productions[i++],"%s->%s%s'\0",l,temp,l);

temp = strtok(NULL,"|");

}

sprintf(productions[i++],"%s’->\356\0",l);

if(flag == 0)

printf("The given productions don't have Left Recursion");

else

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

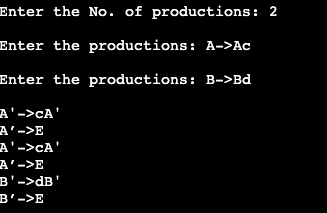
printf("\n%s",productions[j]);

}

}

}

**Output-**



**PROGRAM 4**

**i)Aim:-** Program for implementation of Recursive Descent Parser for language given below

E-> TE’

E’-> +TE’ | -TE’ | null

T-> FT’

T’-> \*FT’| /FT’ | null

F-> id/ (E)/ num

**Software used:-** onlinegdb.com (Online compiler)

**Procedure:-** Go to debug -> run or press CTRL + F9 to run the program.

#include <conio.h>

#include<stdio.h>

#include<string.h>

char input[100];

char prod[10][5];

int pos=-1,st=0,len=0,flag=1;

char id,num;

void E();

void Ed();

void T();

void Td();

void F();

void advance();

void main()

{

printf("\nEnter the input string for parsing:\n");

gets(input);

len=strlen(input);

input[len]='$';

advance();

E();

if(pos==len&&flag==1)

printf("Accepted");

else

printf("Not Accepted");

getch();

}

void advance()

{

pos++;

//printf("%d",pos);

if(pos<len) {

if(input[pos]>='0'&&input[pos]<='9')

{

id='\0';

num=input[pos];

}

if((input[pos]>='A'&&input[pos]<='Z')||input[pos]>='a'&&input[pos]<='z')

{

id=input[pos];

num='\0';

}

}

}

void E()

{

//strcpy(prod[++st],"E->TE' ");

T();

Ed();

}

void Ed()

{

int p=1;

if(input[pos]=='+'||input[pos]=='-')

{

p=0;

//strcpy(prod[++st],"E'->+TE'||-TE' ");

advance();

T();

Ed();

}

if (p==1)

{

}

}

void T()

{

F();

Td();

}

void Td()

{ int p=1;

if(input[pos]=='\*'||input[pos]=='/')

{ p=0;

advance();

F();

Td();

}

if (p==1)

{}

}

void F()

{

if(input[pos]==id)

{

//strcpy(prod[++st],"F->id ");

advance();

}

else if(input[pos]=='(')

{ //strcpy(prod[++st],"F->(E) ");

advance();

E();

if(input[pos]==')')

{

//

advance();

}

}

else if(input[pos]==num)

{

//strcpy(prod[++st],"F->num ");

advance();

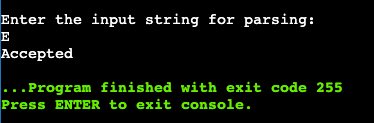
}

else

flag=0;

}

**Output-**



**ii)Aim:-** To perform recursive descent parsing using multi characters and whitespace in input.

**Software used:-** onlinegdb.com (Online compiler)

**Procedure:-** Go to debug -> run or press CTRL + F9 to run the program.

#include<stdio.h>

#include<conio.h>

#include<string.h>

char prod[50][50];

char input[100];

int pos=-1;

int flag=0;

int st=-1;

int reject=0,k=0;

char id[100],num[100];

void E();

void Ed();

void T();

void Td();

void F();

void advance();

void E()

{

//strcpy(prod[++st]," E->TE' ");

T();

Ed();

}

void Ed()

{

int p=1;

if(input[pos]=='+')

{

p=0;

strcpy(prod[++st]," E'->+TE' ");

advance();

T();

Ed();

}

else if(input[pos]=='-')

{

p=0;

strcpy(prod[++st]," E'->-TE' ");

advance();

T();

Ed();

}

if(p==1)

{

strcpy(prod[++st]," E'->null ");

}

}

void T()

{

//strcpy(prod[++st]," T->FT' ");

F();

Td();

}

void Td()

{

int p=1;

if(input[pos]=='\*')

{

p=0;

strcpy(prod[++st]," E'->\*FT' ");

advance();

F();

Td();

}

else if(input[pos]=='/')

{

p=0;

strcpy(prod[++st]," T'->/FT' ");

advance();

F();

Td();

}

if(p==1)

{

strcpy(prod[++st]," T'->null ");

//advance();

}

}

void F()

{

if(input[pos]=='(')

{

advance();

E();

if(input[pos]==')')

{

advance();

strcpy(prod[++st],"F->(E)");

}

}

else if(input[pos]==id[k] || input[pos]==num[k])

{

advance();

}

}

void advance()

{

flag=0;

pos++;

k=0;

if (input[pos]>='0'&&input[pos]<='9')

{

do

{

if(input[pos]>='a'&&input[pos]<='z')

{

reject=1;

}

num[k]=input[pos];

k++;

pos++;

flag=1;

}

while (input[pos]>='0'&&input[pos]<='9');

}

else if(input[pos]>='a'&&input[pos]<='z')

{ pos++;

flag=1;

do

{

id[k]=input[pos];

pos=pos+1;

k++;

}while((input[pos]>='a'&&input[pos]<='z')||(input[pos]>='0'&&input[pos]<='9'));

}

if(flag==1)

{

pos--;

k--;

}

}

void main()

{

int l;

printf("Enter String \n");

gets(input);

l=strlen(input);

input[l]='$';

advance();

E();

if(pos==l &&reject!=1)

{

printf("Accepted");

}

else

printf("rejected");

getch();

}

**Output-**

