

COLLEGE OF ENGINEERING & MANAGEMENT PUNNAPRA

Under Co-operative Academy of Professional Education
Estd. by Govt. of Kerala
VADACKAL(PO), ALAPPUZHA - 688003



LABORATORY RECORD

BTech Computer Science & Engineering

CS 431 Compiler Design Lab

Year 2020-2021

Name : VISHNU SURESH

Semester :S7 Branch :CSE

Roll No :22 KTU ID : PRP17CS026

October 28, 2021

Department of Computer Science & Engineering
COLLEGE OF ENGINEERING & MANAGEMENT PUNNAPRA
Under Co-operative Academy of Professional Education, Estd. by Govt. of Kerala
VADACKAL(PO), ALAPPUZHA - 688003

CERTIFICATE



This is to certify that it is a bonafied record of practical work done by Sri/Kum.VISHNU SURESH bearing the KTU ID: PRP17CS026 of 7th Semester B Tech Computer Science & Engineering in the CS 431 Compiler Design laboratory during the academic year 2020-2021 under our supervision.

Staff-in-Charge

Head of Department

Date:

Contents

Title	Page No	Date	Signature
1 Lexical Analyzer	4	_____	_____
2 ϵ – closure of an NFA	8	_____	_____
3 Operator Precedence Parser	12	_____	_____
4 Recursive Descent parser	18	_____	_____
5 Shift Reduce Parser	23	_____	_____
6 Constant Propagation	28	_____	_____
7 Intermediate code generation	31	_____	_____
8 Conversion from three address code to 8086 assembly language instructions	35	_____	_____
9 Lexical analyser using Lex tool	38	_____	_____
10 Validate arithmetic expression using LEX and YACC tool	42	_____	_____
11 Validate variable using LEX and YACC tool	46	_____	_____
12 Calculator using LEX and YACC tool	49	_____	_____

Program No:1

Date :29/09/2020

1. Lexical Analyzer

Aim

Design and implement lexical analyser for a given language using C and the lexical analyser should ignore redundant spaces, tabs and new lines

Algorithm

input: read a c program

output: print wheather the lexemes are digit,keyword, identifier,special operators

step1: start the program

step2: declare all the file pointers and variables

step3: read the input program and store it in the file f

step4: separete each stream

step5: check whether the string is digit,keyword,identifier or special symbols

step6: print the numbers,keywords,identifiers and special symbols of the input programs

Program

```
#include<string.h>
#include<ctype.h>
#include<stdio.h>
void main()
{
    FILE *f;
    char c,str[10];
    int t=1,n=0,i=0;
    printf("Enter the c program \n");
    f=fopen("input.txt","w");
    while((c=getchar())!=EOF)
    {
        putc(c,f);
    }
    fclose(f);
```

```

f=fopen("input.txt","r");
while((c=getc(f))!=EOF)
{
    if(isdigit(c))
    {
        n=c-48;
        c=getc(f);
        while(isdigit(c))
        {
            n=n*10+(c-48);
            c=getc(f);
        }
        printf("\n%d is a number ",n);
        ungetc(c,f);
    }
    else if(isalpha(c))
    {
        str[i++]=c;
        c=getc(f);
        while(isdigit(c)||isalpha(c)||c=='_'||c=='$')
        {
            str[i++]=c;
            c=getc(f);
        }
        str[i++]='\0';
        if(strcmp("printf",str)==0||strcmp("main",str)
            ==0||strcmp("void",str)==0||strcmp("auto",str)
            ==0||strcmp("break",str)==0||strcmp("case",str)
            ==0||strcmp("char",str)==0||strcmp("const",str)
            ==0||strcmp("continue",str)==0||strcmp("default
            ",str)==0||strcmp("do",str)==0||strcmp("double
            ",str)==0||strcmp("else",str)==0||strcmp("enum
            ",str)==0||strcmp("extern",str)==0||strcmp("
            float",str)==0||strcmp("for",str)==0||strcmp("
            goto",str)==0||strcmp("if",str)==0||strcmp("int
            ",str)==0||strcmp("long",str)==0||strcmp("
            register",str)==0||strcmp("return",str)==0||
            strcmp("short",str)==0||strcmp("signed",str)

```

```

        ==0||strcmp("sizeof",str)==0||strcmp("static",
        str)==0||strcmp("struct",str)==0||strcmp("
        switch",str)==0||strcmp("typedef",str)==0||
        strcmp("union",str)==0||strcmp("unsigned",str)
        ==0||strcmp("void",str)==0||strcmp("volatile",
        str)==0||strcmp("while",str)==0)
    {

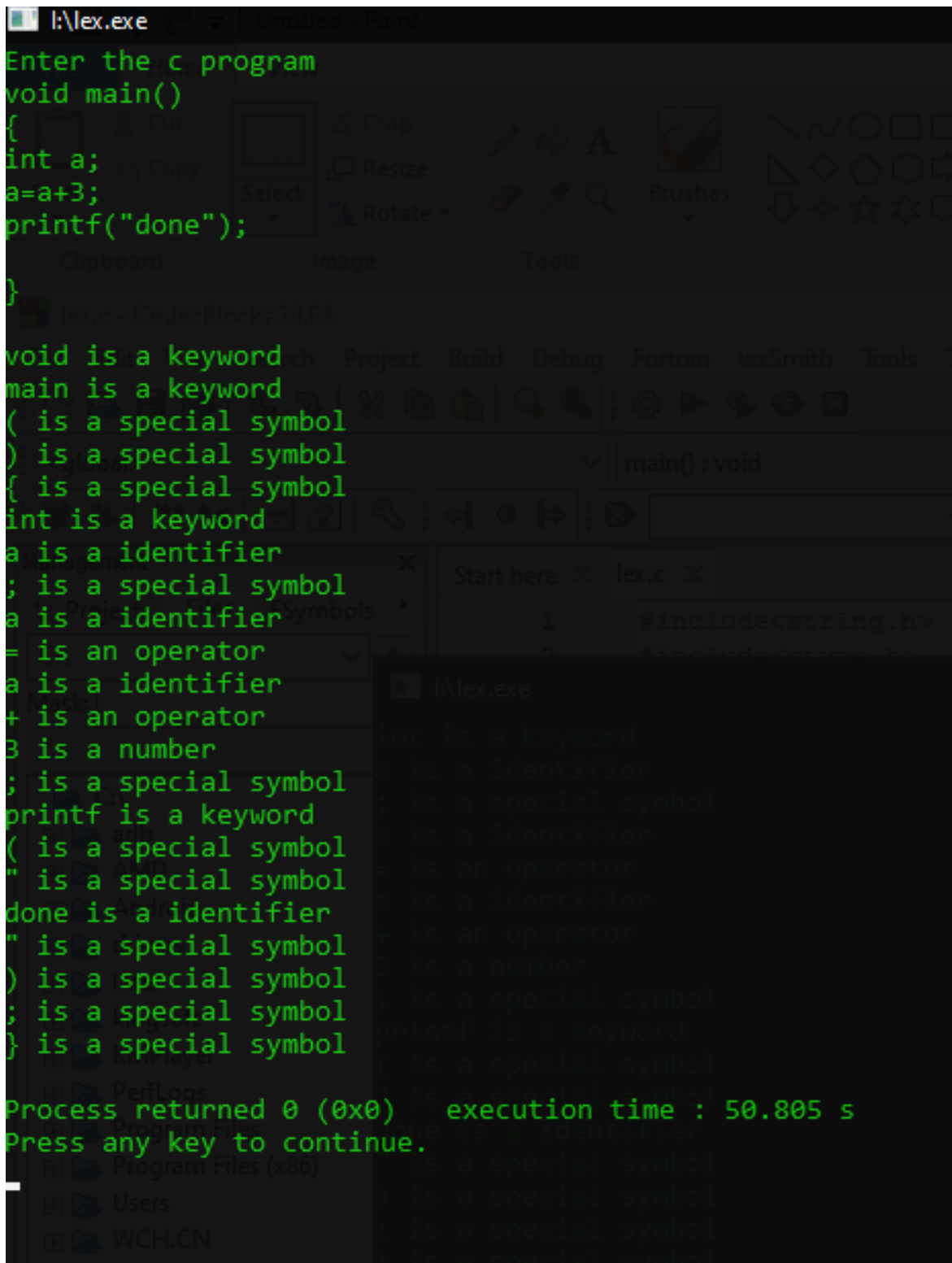
        printf("\n%s is a keyword",str);
    }

    else
    {
        printf("\n%s is a identifier",str);
    }
    ungetc(c,f);
    i=0;
}

else if(c==' '||c=='\t'||c=='\n')
{
    t=1;
}
else if(c=='+'||c=='-'||c=='*'||c=='/'||c=='='||c
    =='=='||c=='!='||c=='>'||c=='>='||c=='<'||c=='<=')
{
    printf("\n%c is an operator",c);
}
else
{
    printf("\n%c is a special symbol",c);
}
}
printf("\n");
fclose(f);
}

```

Output



```
I:\lex.exe
Enter the c program
void main()
{
int a;
a=a+3;
printf("done");
}

lex.c - CodeBlocks 20.03
void is a keyword
main is a keyword
( is a special symbol
) is a special symbol
{ is a special symbol
int is a keyword
a is a identifier
; is a special symbol
a is a identifier
= is an operator
a is a identifier
+ is an operator
3 is a number
; is a special symbol
printf is a keyword
( is a special symbol
" is a special symbol
done is a identifier
" is a special symbol
) is a special symbol
; is a special symbol
} is a special symbol

Process returned 0 (0x0)   execution time : 50.805 s
Press any key to continue.
```

Result

Program is executed and output is obtained

Program No:2

Date :05/10/2020

2. ϵ – closure of an NFA

Aim

Write program to find ϵ – closure of all states of any given NFA with ϵ transition should ignore redundant spaces, tabs and new lines

Algorithm

Input : Transition's and states of an NFA

Output : Epsilon closure of the given NFA

step1: Start the program

step2: Declare all the file pointers and variables

step3: Fetch the input transition from input.dat and states

step4: Check the transition and states , store them in an array

step5: If input state and given transitions match goto step 5.1 else step6

step 5.1 : check Epsilon transitions for the state , if found

store in in an array read next

step6: Check the next state until no states left , if next state found go to step5 else 7

step 7 : Print the Epsilon closure of the given transitions from the array

step 8 : End

Program

```
#include<stdio.h>
```

```
#include<string.h>
```

```
char result[20][20],copy[10],states[20][20];
```

```
void add_state(char a[10],int i){
```

```
    strcpy(result[i],a);
```

```
}
```



```

void display(int n){
    int k=0;
    printf("\n Epsilon closure of %s = { ",copy);
    while(k < n){
        printf(" %s",result[k]);
        k++;
    }
    printf(" } \n\n\n");
}

int main(){
    FILE *INPUT;
    INPUT=fopen("input.dat","r");
    char state[10];
    int end,i=0,n,k=0;
    char state1[10],input[10],state2[10];
    printf("\n Enter the no of states: ");
    scanf("%d",&n);
    printf("\n Enter the states \n");
    for(k=0;k<n;k++){
        scanf("%s",states[k]);

    }

    for( k=0;k<n;k++){
        i=0;
        strcpy(state,states[k]);
        strcpy(copy,state);
        add_state(state,i++);
        while(1){
            end = fscanf(INPUT,"%s%s%s",state1,
                input,state2);
            if (end == EOF ){
                break;
            }

            if( strcmp(state,state1) == 0 ){
                if( strcmp(input,"e") == 0 ) {

```

```

                                add_state(state2,i++);
                                }
                                }
                                }
                                display(i);
                                rewind(INPUT);
                                }

                                return 0;
                                }

```

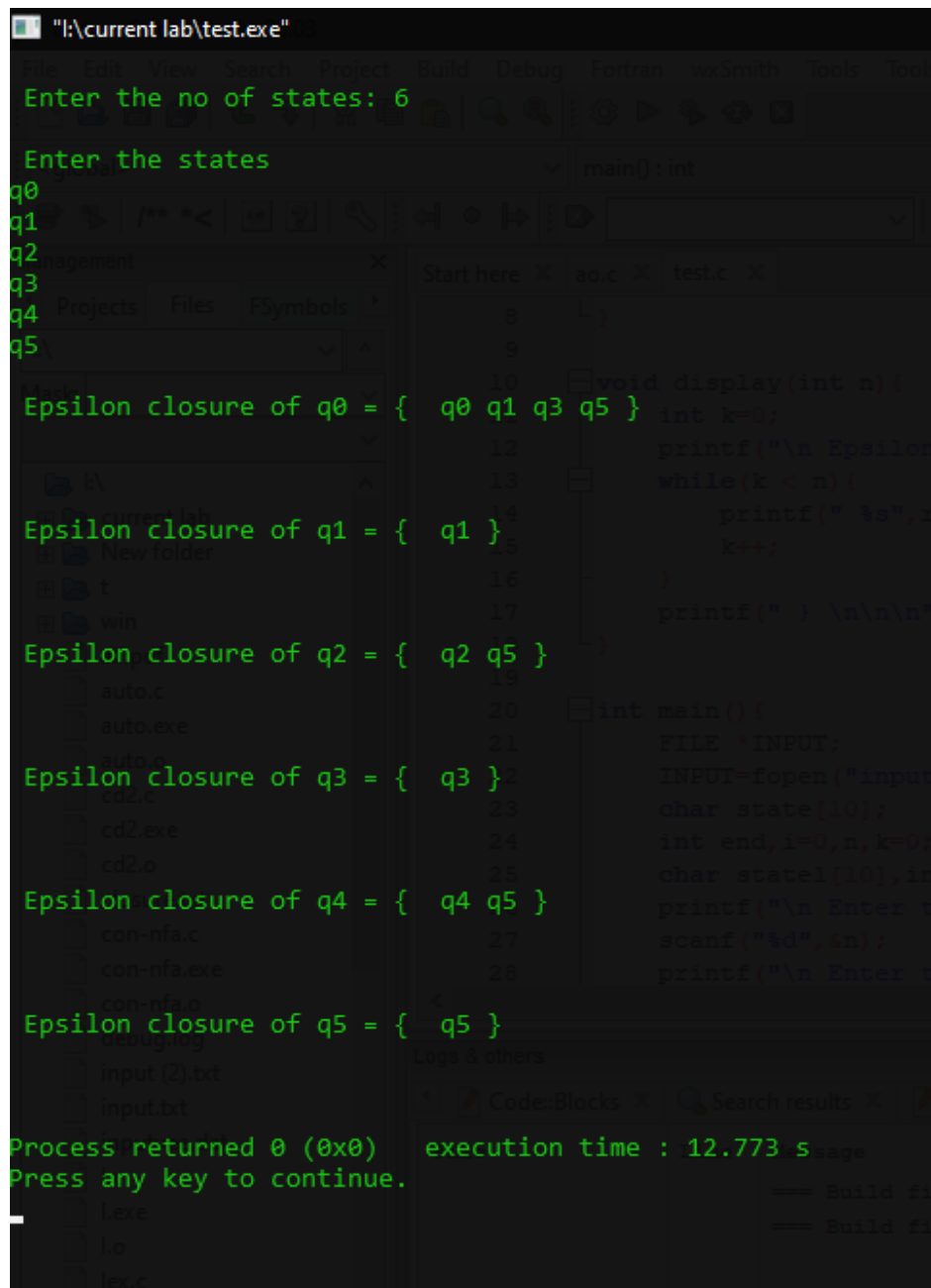
input.dat

```

q0 e q1
q0 e q3
q0 e q5
q1 a q2
q2 e q5
q3 b q4
q4 e q5

```

Output



```
"I:\current lab\test.exe"
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools
Enter the no of states: 6
Enter the states
q0
q1
q2
q3
q4
q5
Epsilon closure of q0 = { q0 q1 q3 q5 }
Epsilon closure of q1 = { q1 }
Epsilon closure of q2 = { q2 q5 }
Epsilon closure of q3 = { q3 }
Epsilon closure of q4 = { q4 q5 }
Epsilon closure of q5 = { q5 }
Process returned 0 (0x0)
Press any key to continue.
execution time : 12.773 s
```

Result

Program is executed and output is obtained

Program No:4

Date :20/10/2020

3. Operator Precedence Parser

Aim

Develop an operator precedence parser for a given language

Algorithm

Input : An expression

Output : Whether the given expression is accepted or not

step1 : Start

Step2 : Allocate the needed variables and stack

step3 : Scan the precedence parsing table which is defined in our program , Record the grammar symbols according to their possibility.

step4 : Read the expression and add \$ to the end

step5 : do steps until string is completely processed

step6 : Now scan the input string from left right until the > is encountered

step7 : Scan towards left over all the equal precedence until the first left most < is encountered

step8 : Make sure that everything between left most < and right most > is a handle , else go to step5

step9 : End of do

step10 : \$ element \$ means parsing is successful, print " Accepted" else print "Not Accepted"

step11 : End

Program

```
#include<stdio.h>
```

```
#include<string.h>
```

```
char *input;
```

```
int i=0;
```

```
char lasthandle[6],stack[50],handles[][5]={")E(", "E*E", "E+E", "i", "E^E"};
```

```

int top=0,l;
char prec[9][9]={
    // Parsing table defanition
    /*stack + - * / ^ i ( ) $ */

    /* + */ '>>', '>>', '<<', '<<', '<<', '<<', '<<', '>>', '>>',
    /* - */ '>>', '>>', '<<', '<<', '<<', '<<', '<<', '>>', '>>',
    /* * */ '>>', '>>', '>>', '>>', '<<', '<<', '<<', '>>', '>>',
    /* / */ '>>', '>>', '>>', '>>', '<<', '<<', '<<', '>>', '>>',
    /* ^ */ '>>', '>>', '>>', '>>', '<<', '<<', '<<', '>>', '>>',
    /* i */ '>>', '>>', '>>', '>>', '>>', 'e', 'e', '>>', '>>',
    /* ( */ '<<', '<<', '<<', '<<', '<<', '<<', '<<', '>>', 'e',
    /* ) */ '>>', '>>', '>>', '>>', '>>', 'e', 'e', '>>', '>>',
    /* $ */ '<<', '<<', '<<', '<<', '<<', '<<', '<<', '<<', '>>',

};

int getindex(char c)
{
switch(c)
{
case '+':return 0;
case '-':return 1;
case '*':return 2;
case '/':return 3;
case '^':return 4;
case 'i':return 5;
case '(':return 6;
case ')':return 7;
case '$':return 8;
}
}

```

```
}
```

```
int shift()
{
    stack[++top]=*(input+i++);
    stack[top+1]='\0';
}
```

```
int reduce()
{
    int i,len,found,t;
    for(i=0;i<5;i++)
    {
        len=strlen(handles[i]);
        if(stack[top]==handles[i][0]&&top+1>=len)
        {
            found=1;
            for(t=0;t<len;t++)
            {
                if(stack[top-t]!=handles[i][t])
                {
                    found=0;
                    break;
                }
            }
            if(found==1)
            {
                stack[top-t+1]='E';
                top=top-t+1;
                strcpy(lasthandle,handles[i]);
                stack[top+1]='\0';
                return 1;
            }
        }
    }
    return 0;
}
```

```

void dispstack()
{
int j;
for(j=0;j<=top;j++)
    printf("%c",stack[j]);
}

```

```

void dispinput()
{
int j;
for(j=i;j<l;j++)
    printf("%c",*(input+j));
}

```

```

void main()
{
int j;

input=(char*)malloc(50*sizeof(char));
printf("\nEnter the string\n");
scanf("%s",input);
input=strcat(input,"$");
l=strlen(input);
strcpy(stack,"$");
printf("\nSTACK\tINPUT\tACTION");
while(i<=l)
{
    shift();
    printf("\n");
    dispstack();
    printf("\t");
    dispinput();
}
}

```

```

printf("\tShift");
if(prec[getindex(stack[top])][getindex(input[i])
    ]=='>')
    {
        while(reduce())
            {
                printf("\n");
                dispstack();
                printf("\t");
                dispinput();
                printf("\tReduced: E->%s",lasthandle)
                    ;
            }
    }
}

if(strcmp(stack,"$E$")==0)
    printf("\nAccepted;");
else
    printf("\nNot Accepted;");
}

\begin{lstlisting}

```


Output

```
"I:\current lab\4\oparser.exe"

Enter the string
i+i-i*i

STACK   INPUT   ACTION
$i      +i-i*i$  Shift
$E      +i-i*i$  Reduced: E->i
$E+     i-i*i$  Shift
$E+i    -i*i$   Shift
$E+E    -i*i$   Reduced: E->i
$E      -i*i$   Reduced: E->E+E
$E-     i*i$    Shift
$E-i    *i$     Shift
$E-E    *i$     Reduced: E->i
$E-E*   i$      Shift
$E-E*i  $        Shift
$E-E*E  $        Reduced: E->i
$E-E    $        Reduced: E->E*E
$E-E$   $        Shift
$E-E$   $        Shift
Not Accepted;
Process returned 14 (0xE)   execution time : 11.380 s
Press any key to continue.
```

Result

Program is executed and output is obtained

Program No:6

Date :17/11/2020

4. Recursive Descent parser

Aim

Construct Recursive Descent parser for an expression

Algorithm

Input: Expression to be evaluated

Output: Expression in accepted or rejected

Main function

Step1: Set all required variables

Step2: Read the input expression

Step3: Call function E and return value here

Step4: If the expression is fully processed and flag value
is 0 go to step5 else goto step6

Step5: Print Expression is accepted goto step 7

Step6: Print Expression is rejected

Step7: End

Function E

Step1: Call function T and return value here

Step2: Call function Tdash and return value here

Step3: Return value to the caller

Step4: End

Function Edash

Step1: If current input value pointed by the pointer is +
then goto step2 else goto step5
Step2: Increment the pointing value
Step3: Call function T
Step4: Call function Edash
Step5: Return value to the caller
Step6: End

Function T

Step1: Call function F
Step2: Call function Tdash
Step3: Return value to the caller
Step4: End

Function Tdash

Step1: If current input value pointed by the pointer is *
then goto step2 else goto step5
Step2: Increment the pointing value
Step3: Call function F
Step4: Call function Tdash
Step5: Return value to the caller
Step6: End

Function F

Step1: If current input value pointed by the pointer is a
then goto step2 else goto step 8
Step2: Increment the pointing value goto step
Step3: If current input value pointed by the pointer is (
then goto step4 else goto step 8
Step4: Increment the pointing value call Function E

Step5: If current input value pointed by the pointer is)
 then goto step4 else goto step7
 Step6: Increment the pointing value
 Step7: Set flag as 1
 Step8: Set flag as 1
 Step9: End

Program

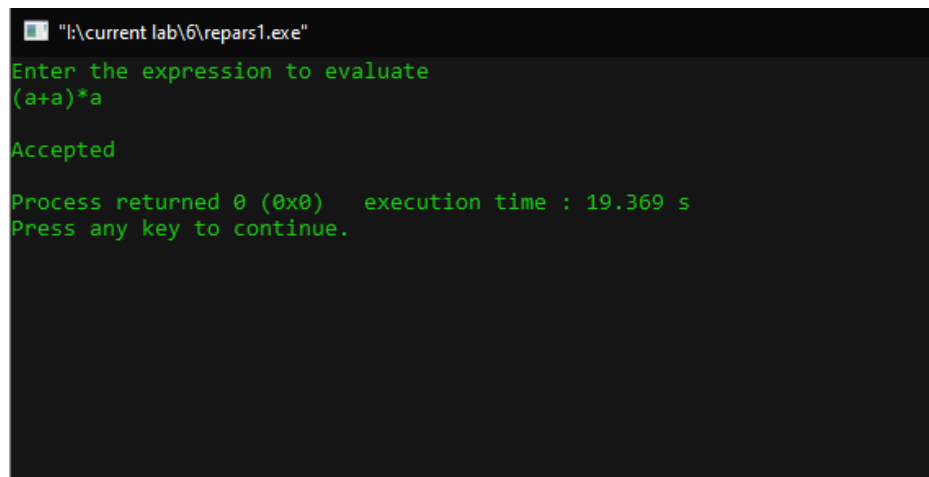
```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
char input[10];
int i,flag;
void E();
void T();
void Edash();
void Tdash();
void F();
main()
{
    i=0;
    flag=0;
    printf("Enter the expression to evaluate \n");
    gets(input);
    E();
    if(strlen(input)==i&&flag==0)
        printf("\nAccepted\n");
    else
        printf("\nRejected\n");
}
void E()
{
    T();
    Edash();
}
void Edash()
{
    if(input[i]=='+')
    {
```

```

        i++;
        T();
        Edash();
    }
}
void T()
{
    F();
    Tdash();
}
void Tdash()
{
    if(input[i]=='*')
    {
        i++;
        F();
        Tdash();
    }
}
void F()
{
    if(input[i]=='a')
        i++;
    else if(input[i]=='(')
    {
        i++;
        E();
        if(input[i]==')')
            i++;
        else flag=1;
    }
    else
        flag=1;
}

```

Output



```
"I:\current lab\6\repars1.exe"
Enter the expression to evaluate
(a+a)*a
Accepted
Process returned 0 (0x0)   execution time : 19.369 s
Press any key to continue.
```

Result

Program is executed and output is obtained

Program No:7

Date :15/12/2020

5. Shift Reduce Parser

Aim

Construct a shift reduce parser for a given language

Algorithm

Algorithm:Shift Reduce Parser

Input: Read an arithmetic expression.

Output: Print whether the expression is accepted or not.

Step1: Start the program.

Step2: Declare all the variables,input buffer,stack.

Step3: Read the expression and store it in an input buffer.

Step4: Move symbols from input buffer to stack one at a time.

Step5: If stack contain symbols matching production rules then reduce.

Step6: Print the Stack,Input Buffer,Action at each step.

Step7: Continue steps 4-6 until input buffer is empty.

Step8: Check if stack contain E at last.

Step9: If so print Accepted else Not Accepted.

Step10: Stop.

Program

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int z = 0, i = 0, j = 0, c = 0;
char a[16], h1[20], stack[15], h2[10];
void check()
{
    strcpy(h1,"REDUCE TO E -> ");
    for(z=0;z<c;z++)
    {
        if(stack[z] == 'a')
```

```

    {
        printf("%sa", h1);
        stack[z] = 'E';
        stack[z + 1] = '\0';
        printf("\n%s\t%s$\t", stack, a);
    }
}
for(z=0;z<c;z++)
{
    if(stack[z] == '(' && stack[z + 1] == 'E' && stack[
        z + 2] == ')')
    {
        printf("%s(E)", h1);
        stack[z]='E';
        stack[z + 1]='\0';
        stack[z + 1]='\0';
        printf("\n%s\t%s$\t", stack, a);
        i = i - 2;
    }
}
for(z=0;z<c;z++)
{
    if(stack[z] == 'E' && stack[z + 1] == '*' && stack[
        z + 2] == 'E')
    {
        printf("%sE*E", h1);
        stack[z] = 'E';
        stack[z + 1] = '\0';
        stack[z + 2] = '\0';
        printf("\n%s\t%s$\t", stack, a);
        i = i - 2;
    }
}

for(z=0;z<c;z++)
{
    if(stack[z] == 'E' && stack[z + 1] == '+' && stack[
        z + 2] == 'E')

```



```

    {
        printf("%sE+E", h1);
        stack[z]='E';
        stack[z + 1]='\0';
        stack[z + 1]='\0';
        printf("\n$s\t%s$\t", stack, a);
        i = i - 2;
    }
}
for(z=0;z<c;z++)
{
    if(stack[z] == 'E' && stack[z + 1] == '-' && stack[
        z + 2] == 'E')
    {
        printf("%sE-E", h1);
        stack[z]='E';
        stack[z + 1]='\0';
        stack[z + 1]='\0';
        printf("\n$s\t%s$\t", stack, a);
        i = i - 2;
    }
}
for(z=0;z<c;z++)
{
    if(stack[z] == 'E' && stack[z + 1] == '/' && stack[
        z + 2] == 'E')
    {
        printf("%sE/E", h1);
        stack[z]='E';
        stack[z + 1]='\0';
        stack[z + 1]='\0';
        printf("\n$s\t%s$\t", stack, a);
        i = i - 2;
    }
}

return ;
}
int main()

```

```

{
    printf("GRAMMAR is -\nE->a \nE->E*E \nE->E+E \nE->E-E \nE->E/E \nE->(E)\n");
    printf("enter the string\n");
    gets(a);
    c=strlen(a);
    strcpy(h2,"SHIFT");
    printf("\nstack \t input \t\t action");
    printf("\n$\t%s$\t", a);
    for(i = 0; j < c; i++,j++)
    {
        printf("%s", h2);
        stack[i] = a[j];
        stack[i + 1] = '\0';
        a[j]=' ';
        printf("\n$%s\t%s$\t", stack, a);
        check();

    }
    if(stack[0] == 'E' && stack[1] == '\0')
        printf("Accept\n");
    else
        printf("Reject\n");
}

```

Output

```
"I:\current lab\sr par\sr.exe"
GRAMMAR is -
E->a
E->E*E
E->E+E
E->E-E
E->E/E
E->(E)
enter the string
(a+a)+a

stack    input    action
$        (a+a)+a$  SHIFT
$(       a+a)+a$  SHIFT
$(a      +a)+a$  REDUCE TO E -> a
$(E      +a)+a$  SHIFT
$(E+     a)+a$  SHIFT
$(E+a    )+a$  REDUCE TO E -> a
$(E+E    )+a$  REDUCE TO E -> E+E
$(E      )+a$  SHIFT
$(E)     +a$  REDUCE TO E -> (E)
$E       +a$  SHIFT
$E+      a$   SHIFT
$E+a     $    REDUCE TO E -> a
$E+E     $    REDUCE TO E -> E+E
$E       $    Accept

Process returned 0 (0x0)   execution time : 15.281 s
Press any key to continue.
```

Result

Program is executed and output is obtained

Program No:9

Date :24/11/2020

6. Constant Propagation

Aim

Write a program to perform constant propagation.

Algorithm

Algorithm : Constant Propagation

Input:Read an input expressions

Output:Perform constant propagation and print the result

Step 1:Start the program

Step 2:Declare the variable and two dimensional array

Step 3:Read no:of expression and repeat the step4 to step 8
until all the expression are processed

Step 4:Read each expression and checks whether the RHS of a
expression is a digit or not

Step 5: If it is a digit then store that digit into a
variable 'f' and the variable name into 't'

Step 6:If the next expression variable name is same as the
previous then ignore that expression

Step 7:Check whether the next expression matches with the
value in the variable 'f' then replace that with 't'
otherwise process till the expression is completed then
print the expression itself.

Step 8: Stop

Program

```
#include<stdio.h>
#include<string.h>
void main()
{
    int n,i,j,k,f;
    char s[10][10],t;
    printf("Enter the no.of statements ");
    scanf("%d",&n);
```

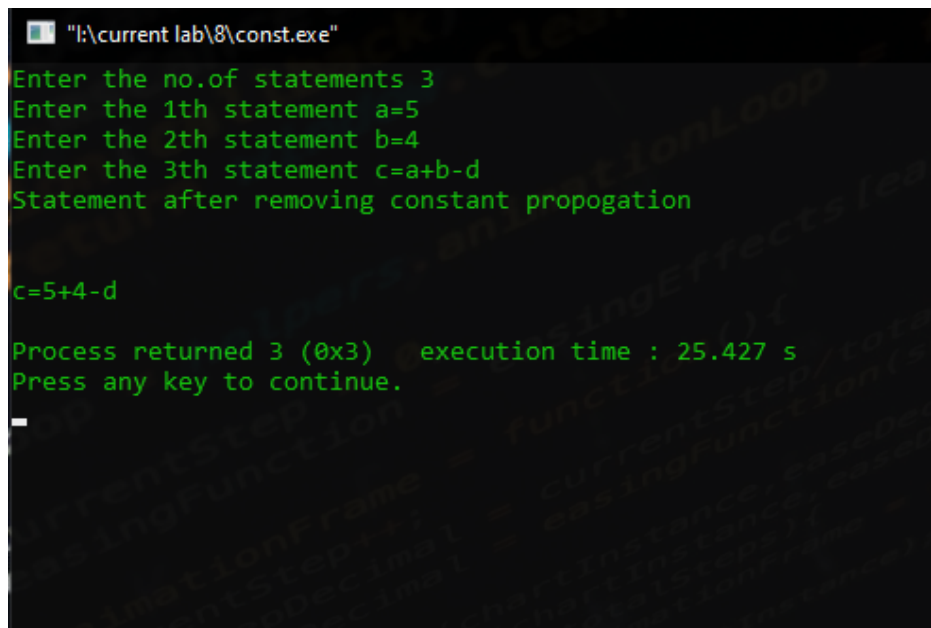
```

for(i=0;i<n;i++)
{
    printf("Enter the %dth statement ",i+1);
    scanf("%s",s[i]);
}
for(i=0;i<n;i++)
{
    if(isdigit(s[i][2]) && s[i][3]=='\0')
    {
        f=s[i][2];
        t=s[i][0];
        s[i][0]='\0';
        for(j=i+1;j<n;j++)
        {

            if(s[j][0]==t)
            {
                s[j][0]='\0';
                break;
            }
            k=2;
            while(s[j][k]!='\0')
            {
                if(s[j][k]==t)
                {
                    s[j][k]=f;
                }
                k=k+1;
            }
        }
    }
}
printf("Statement after removing constant
propagation\n");
for(i=0;i<n;i++)
{
    printf("%s\n",s[i]);
}
}

```

Output



```
"I:\current lab\8\const.exe"
Enter the no.of statements 3
Enter the 1th statement a=5
Enter the 2th statement b=4
Enter the 3th statement c=a+b-d
Statement after removing constant propogation

c=5+4-d

Process returned 3 (0x3)   execution time : 25.427 s
Press any key to continue.
-
```

Result

Program is executed and output is obtained

Program No:10
Date : 12/01/2021

7. Intermediate code generation

Aim

Implement intermediate code generation for simple expressions

Algorithm

Input: Input the expression

Output: Print the intermediate code of the expression

Step1 : Start.

Step2 : Input the expression and store it in an array 'a'.

Step3 : Find the length of the expression and store it in 'n'.

Step4 : Initialize j=0

Step5 : if j<n goto step 6 else goto step 25.

Step6 : Initialize i=0

Step7 : If i<n goto step 8 else goto step 15

Step8 : If a[i] is equal to '*' or a[i] is equal to '/' goto step 9 else goto step 7.

Step9 : Assign d=i-1

Step10 : If(a[d]is equal to ' ' then goto step 11 else goto step12.

Step11 : decrement d by 1 and goto step 10.

Step12 : Print the values stored in c equal to the values in a[d],a[i],a[i+1].

Step13 : Replace the value in a[d] by c , a[i] by space, a[i+1] by space and increment c.

Step14 : Increment i by 1 and goto step 7.

Step15 : Initialize i=0.

Step16 : If i<n goto step 17 else goto step 15.

Step17 : If a[i] is equal to '+' or a[i] is equal to '-' goto step 18 else goto step 16.

Step18 : Assign d=i-1.

Step19 : If(a[d]is equal to ' ' then goto step 20 else goto step21.

Step20 : decrement d by 1 and goto step 19.

Step21 : Print as the values stored in c equal to the values in a[d],a[i],a[i+1].

Step22 : Replace the value in a[d] by c , a[i] by space, a[i+1] by space and increment c.

Step23 : Increment i by 1 and goto step 16.

Step24 : Increment j by 1 and goto step 5.

Step25 : Stop.

Program

```
#include<stdio.h>
#include<string.h>
void main()
{
    int j=0,i=0,n=0,d=0;
    char a[20],c='A';
    printf("Enter the code : ");
    scanf("%s",a);
    while(a[i]!='\0')
    {

        n=n+1;
        i=i+1;

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

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

            if(a[i]=='*' || a[i]=='/')
            {
                d=i-1;
                while(a[d]==' ')
                {

                    d=d-1;

                }
                if(d!=(i-1))
                {
                    printf("%c=%c%c%c\n",c,a[d],a[i],a[i+1]);
```



```

        a[d]=c;
        a[i]=' ';
        a[i+1]=' ';
        c=c+1;
        continue;
    }
    else if(d==(i-1))
    {
        printf("%c=%c%c%c\n",c,a[i-1],a[i],a[i+1]);
        a[i-1]=c;
        a[i]=' ';
        a[i+1]=' ';
        c=c+1;
        continue;
    }
}
for(i=0;i<n;i++)
{
    if(a[i]=='+' || a[i]=='-')
    {
        d=i-1;
        while(a[d]!=' ')
        {
            d=d-1;
        }
        if(d!=(i-1))
        {
            printf("%c=%c%c%c\n",c,a[d],a[i],a[i+1]);
            a[d]=c;
            a[i]=' ';
            a[i+1]=' ';
            c=c+1;
            continue;
        }
    }
}

```


Program No:13
Date : 12/01/2021

8. Conversion from three address code to 8086 assembly language instructions

Aim

Write a c program to implement the back end of the compiler to take the three address code and produced 8086 assembly language instructions that can be assembled and the run using an assembler. The target assembly instructions can be simple mov, add etc

Algorithm

Input: Input the three address code

Output: Print the corresponding 8086 assembly language instruction

Step1 : Start.

Step2 : Input the three address code and store it in 'a'.

Step3 : Store a[3] in 'ch'

Step4 : If 'ch is equal to '+' then print MOV R0 a[2] , ADD R0 a[4] , MOV a[0] R0 and goto step 9.

Step5 : If 'ch is equal to '-' then print MOV R0 a[2] , SUB R0 a[4] , MOV a[0] R0 and goto step 9.

Step6 : If 'ch is equal to '*' then print MOV R0 a[2] , MUL R0 a[4] , MOV a[0] R0 and goto step 9.

Step7 : If 'ch is equal to '/' then print MOV R0 a[2] , DIV R0 a[4] , MOV a[0] R0 and goto step 9.

Step8 : Print INVALID

Step9 : Stop.

Program

```
#include<stdio.h>
#include<string.h>
void main()
{
    char a[10],ch;
    printf("Enter the three address code:\n");
    gets(a);
```

```

ch=a[3];
switch(ch)
{
    case '+' :
        printf("\nMOV R0,%c",a[2]);
        printf("\nADD R0,%c",a[4]);
        printf("\nMOV %c,R0\n",a[0]);
        break;
    case '-' :
        printf("\nMOV R0,%c",a[2]);
        printf("\nSUB R0,%c",a[4]);
        printf("\nMOV %c,R0\n",a[0]);
        break;
    case '*' :
        printf("\nMOV R0,%c",a[2]);
        printf("\nMUL R0,%c",a[4]);
        printf("\nMOV %c,R0\n",a[0]);
        break;
    case '/' :
        printf("\nMOV R0,%c",a[2]);
        printf("\nDIV R0,%c",a[4]);
        printf("\nMOV %c,R0\n",a[0]);
        break;
    default : printf("INVALID\n") ;
}
}

```

Output

```
$ ./a.out
enter the three address code:
s=q*w

MOV R0,q
MUL R0,w
MOV s,R0
$ █
```

Result

The program is executed and the output is obtained and verified.

Program No:11
Date :14/01/2021

9. Lexical analyser using Lex tool

Aim

Implementation of lexical analyser using Lex tool

Algorithm

Algorithm:Lexical analyser using Lex Tool

Input: A C-program

Output: Print the lexemes

Step1 : Start.

Step2 : Define the definition section if any.

Step3 : Define the rule section with regular expression for preprocessor,keyword,identifier,function,operators etc.

Step4 : Define the subroutine part.

Step5 : Store the file containing the C program to a file pointer.

Step6 : Store it to yyin.

Step7 : call yylex().

Step8 : invoke yywrap() to check end of the file.

Step9 : Stop

Program

```
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* { printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}
int |
float |
char |
double |
while |
for |
do |
if |
break |
continue |
```

```

void |
switch |
case |
long |
struct |
const |
typedef |
return |
else |
printf |
goto {printf("\n\t%s is a KEYWORD",yytext);}
{identifier}\(\) {printf("\n\nFUNCTION\n\t%s",yytext);}
\{ { printf("\n BLOCK BEGINS");}
\} { printf("\n BLOCK ENDS");}
{identifier}\([[0-9]*\])? { printf("\n %s IDENTIFIER",
    yytext);}
[0-9]+ {printf("\n\t%s is a NUMBER",yytext);}
= {printf("\n\t%s is an ASSIGNMENT OPERATOR",yytext);}
\<= |
\>= |
\< |
== |
\!= |
\> { printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
\+ |
\- |
\* |
\% |
\/ {printf("\n\t%s is a ARITHMETIC OPERATOR",yytext);}
\; |
\: |
\, |
\" |
\\ |
\( |
\) |
\' {printf("\n\t%s is a SPECIAL OPERATOR",yytext);}
\\n |
\\t |

```

```

\\0 {printf("\n\t%s is a ESCAPE SEQUENCE",yytext);}
\s |
%f |
%d |
%c {printf("\n\t%s is a FORMAT SPECIFIER",yytext);}
%%
int main()
{
    FILE *file;
    file = fopen("prg.c","r");
    if(!file)
    {
        printf("could not open file \n");
        exit(0);
    }
    yyin = file;

    yylex();
    return 0;
}
int yywrap()
{
    return 1;
}

```

prg.c

```

#include<stdio.h>
void main(int)
{
    printf("hai");
    printf("0");
}

```


Output

```

[root@3rd3y3 cd_lab]# lex lex.l
[root@3rd3y3 cd_lab]# cc lex.yy.c
[root@3rd3y3 cd_lab]# ./a.out

#include<stdio.h> is a PREPROCESSOR DIRECTIVE

    void is a KEYWORD
main IDENTIFIER
    ( is a SPECIAL OPERATOR
    int is a KEYWORD
    ) is a SPECIAL OPERATOR

BLOCK BEGINS

    printf is a KEYWORD
    ( is a SPECIAL OPERATOR
    " is a SPECIAL OPERATOR
hai IDENTIFIER
    " is a SPECIAL OPERATOR
    ) is a SPECIAL OPERATOR
    ; is a SPECIAL OPERATOR

    printf is a KEYWORD
    ( is a SPECIAL OPERATOR
    " is a SPECIAL OPERATOR
    0 is a NUMBER
    " is a SPECIAL OPERATOR
    ) is a SPECIAL OPERATOR
    ; is a SPECIAL OPERATOR

BLOCK ENDS

[root@3rd3y3 cd_lab]#
```

Result

Program is executed and output is obtained

Program No: 12.a

Date :15/01/2021

10. Validate arithmetic expression using LEX and YACC tool

Aim

Write a program to recognise a valid arithmetic expression that use the operator +, -, * and / using LEX and YACC tool.

Algorithm

Input: Read an arithmetic expression

Output: Print whether the expression is valid or not.

Step1 : Start.

Step2 : Define the definition section of lex and yacc if any.

Step3 : Define the rule section of lex with numbers.

%%

```
[0-9]+ {  
    yylval=atoi(yytext);  
    return NUMBER;  
}
```

%%

Step4 : Define the rule section of yacc with arithmetic operators.

%%

```
A:E{  
E:E'+'E  
|E'-'E  
|E'*'E  
|E'/'E  
| '('E')'  
| NUMBER
```

;

%%

Step5 : Define the main function in yacc such that, print valid if the expression is valid.
 Step6 : Call yyerror() in yacc if the expression is invalid
 .
 Step7 : call yyparse().
 Step8 : Invoke yywrap() in lex to check end of the file.
 Step9 : Stop

Program

expval.l

```
%{
    /* Definition section*/
    #include "y.tab.h"
    extern yylval;
}%

%%
[0-9]+ {
    yylval=atoi(yytext);
    return NUMBER;

}

[\\t]+ ;

\\n { return 0; }
. { return yytext[0]; }

%%
int yywrap()

{

return 1;

}
```

expval.y

```

%{
    int valid=0;
#include<stdio.h>
}%

%token NUMBER
%left '+' '-'
%left '*' '/'
%%
A:E{}
E:E'+'E
|E'-'E
|E'*'E
|E'/'E
|'('E')'
| NUMBER

;
%%

    int main()
    {
        printf("Enter the expression\n");
        yyparse();
        if(valid==0)
            printf("\nValid expression!\n");

    }
    int yyerror()

{
    valid=1;
    printf("\nInvalid expression!\n");
    return 0;

}

```

Output

```
user@user-To-be-filled-by-0-E-M:~$ ./a.out
Enter the expression
3++5

Invalid expression!
user@user-To-be-filled-by-0-E-M:~$ ./a.out
Enter the expression
3+7*5

Valid expression!
user@user-To-be-filled-by-0-E-M:~$
```

Result

The program is executed and the output is obtained and verified.

Program No: 12.b

Date :15/01/2021

11. Vaildate variable using LEX and YACC tool

Aim

Write a program to recognise a valid variable which starts with letter followed by any number of letters and digits.

Algorithm

Input: Read an input variable.

Output: Print whether the variable is valid or not.

Step1 : Start.

Step2 : Define the definition section of lex and yacc if any.

Step3 : Define the rule section of lex with alphabet followed by digits or alphabets.

%%

[a-zA-Z_][a-zA-Z_0-9]* return letter;

%%

Step4 : Define the rule section of yacc with letter from the lex.

%%

start : s

s : letter ;

%%

Step5 : Define the main function in yacc such that, print variable if the variable is valid.

Step7 : Call yyparse().

Step6 : Call yyerror() in yacc if the variable is invalid.

Step8 : Invoke yywrap() in lex to check the end .

Step9 : Stop

Program

validvar.l

%{

```

        #include "y.tab.h"
    %}

    %%
    [a-zA-Z_][a-zA-Z_0-9]* return letter;
    . return yytext[0];
    \n return 0;
    %%
    int yywrap()
    {
    return 1;
    }

    validvar.y

    %{

        #include<stdio.h>

        int valid=1;

    %}

    %token letter

    %%

    start : s

    s : letter ;

    %%

    int yyerror()
    {
        printf("\nIts not in a variable form!\n");
        valid=0;
        return 0;
    }

```

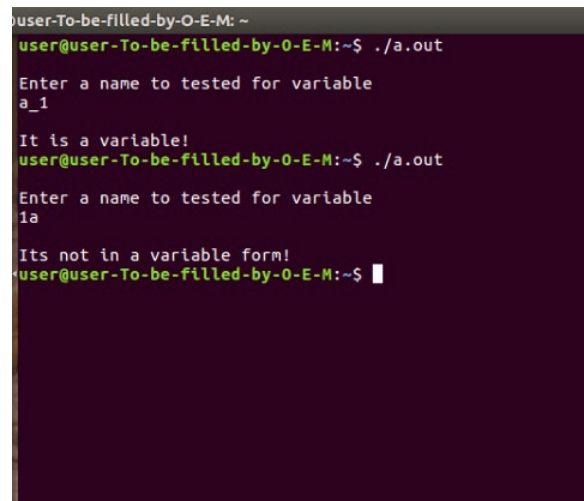
```

int main()
{
    printf("\nEnter a name to tested for variable \n");
    yyparse();
    if(valid)

    {
        printf("\nIt is a variable!\n");
    }
}

```

Output



```

user-To-be-filled-by-O-E-M: ~
user@user-To-be-filled-by-O-E-M:~$ ./a.out
Enter a name to tested for variable
a_1
It is a variable!
user@user-To-be-filled-by-O-E-M:~$ ./a.out
Enter a name to tested for variable
1a
Its not in a variable form!
user@user-To-be-filled-by-O-E-M:~$ █

```

Result

The program is executed and the output is obtained and verified.

Program No: 12.c

Date :14/01/2021

12. Calculator using LEX and YACC tool

Aim

Write a program to implement a calculator using LEX and YACC tool.

Algorithm

Input: Read an arithmetic expression

Output: Print whether the expression is valid or not.

Step1 : Start.

Step2 : Define the definition section of lex and yacc if any.

Step3 : Define the rule section of lex with numbers.

%%

```
[0-9]+ {  
        yylval=atoi(yytext);  
        return NUMBER;  
    }
```

%%

Step4 : Define the rule section of yacc with arithmetic operators and print the result by performing the operation.

%%

A:E

```
{  
    printf("\nResult=%d\n", $$);  
}
```

E:E'+'E {\$\$=\$1+\$3;}

|E'-'E {\$\$=\$1-\$3;}

|E'*'E {\$\$=\$1*\$3;}

|E'/'E {\$\$=\$1/\$3;}

|E'%'E {\$\$=\$1%\$3;}

|'('E')' {\$\$=\$2;}

| NUMBER {\$\$=\$1;}

;

```

%%
Step5 : Define the main function in yacc and call yyparse()
.
Step6 : Call yyerror() in yacc if the expression is invalid
.
Step7 : Invoke yywrap() in lex to check the end.
Step8 : Stop

```

Program

cal.l

```

%{
#include<stdio.h>
#include"y.tab.h"
extern int yylval;
}%
%%
[0-9]+ {
        yylval=atoi(yytext);
        return NUMBER;

    }

[\\t] ;

[\\n] return 0;
. return yytext[0];

```

```

%%

int yywrap()
{
    return 1;
}

```

cal.y

```

%{
#include<stdio.h>
int flag=0;
}%

```

```

%token NUMBER

%left '+' '-'

%left '*' '/' '%'

%left '(' ')'
%%

A:E
{
    printf("\nResult=%d\n", $$);
    return 0;
}
E:E '+' E {$$=$1+$3;}

|E '-' E {$$=$1-$3;}

|E '*' E {$$=$1*$3;}

|E '/' E {$$=$1/$3;}

|E '%' E {$$=$1%$3;}

| '(' E ')' {$$=$2;}

| NUMBER {$$=$1;}

;

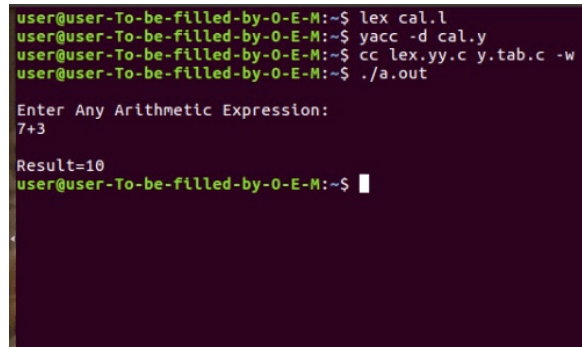
%%

void main()
{
    printf("\nEnter Any Arithmetic Expression:\n");
    yyparse();
}

```

```
void yyerror()  
{  
    printf("\nInvalid\n");  
}
```

Output



```
user@user-To-be-filled-by-0-E-M:~$ lex cal.l  
user@user-To-be-filled-by-0-E-M:~$ yacc -d cal.y  
user@user-To-be-filled-by-0-E-M:~$ cc lex.yy.c y.tab.c -w  
user@user-To-be-filled-by-0-E-M:~$ ./a.out  
  
Enter Any Arithmetic Expression:  
7+3  
  
Result=10  
user@user-To-be-filled-by-0-E-M:~$
```

Result

The program is executed and the output is obtained and verified.