S.NO: 37

Harsha Vardhan

EXP.NO-1 2:

lexical Analyzer to identify identifier, constants, operators using c program.

Aim: To develop a lexical Analyzer to identify identifier, constants, operators, using c progra

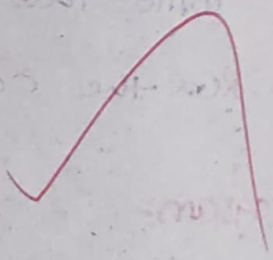
Algorithm:

1. Start the program
2. import some modules
3. Declare variable
4. write the operations required
5. Run the code

Program:

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
int main()
{
  int i, ic=0, m, cc=0 oc=0, j;
  char b[30], operators[30], identifier[30],
                              constants[30],
  printf("Enter the string");
```

```
scanf("%.[^\n]s", &b);
for(i=0;i<strlen(b);i++)
{
if(isspace(b[i]))
   {
  continue;
}
else if(isalpha[b(i)])
    {
identifier[ic]=b[i];
ic++;
}
else if(isdigit(b[i]))
    {
n=(b[i]-'0');
i=i+1;
while(isdigit(b[i]))
        {
   n=n*10+(b[i]-'0');
     i++;
}
 i=i-1;
 constants[cc]=n;
  cc++;
  }
```



Harsha S.NO:37

192011494L

```
else if (b[i] == '=')
    {
    operator[oc] = '=';
    oc++;
}
    }
}
printf("identifier:");
for(i=0;i<cc;i++)
    {
    printf("%d", constant[i]);
}
printf("\n operator:");
    for(i=0;i<oc;i++)
        {
    printf("%c", operator[i]);
    }
}
```

8 AUG 22

1920114941

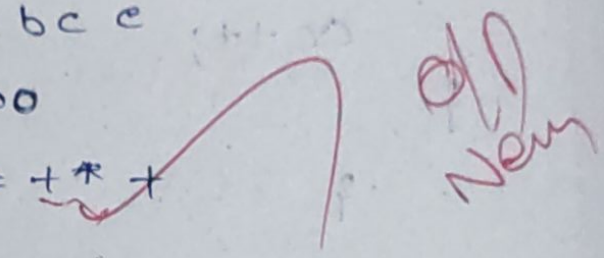
output:-

enter the string : a = b + c * c + 100

identifiers : a b c c

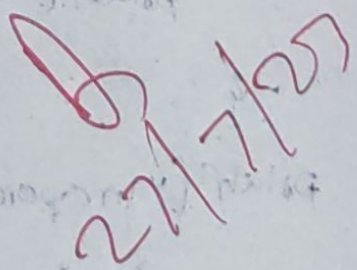
constants : 100

operators : = + * +



Result:-

lexical Analyzer to identify identifiers, constants and operators has been identified a developed lexical Analyzer



EXP NO2 Identify whether a given line is comment or not using C

Aim: To Develop a lexical Analyser to identify whether a given line is a comment or not using C

Program:

```
#include <stdio.h>
#include <conio.h>
int main
```

Algorithm:

1. Start the program
2. Declare input statement
3. Declare variable
4. Read the code

program:

```
#include <stdio.h>
#include <conio.h>
int main()
{
    char com[30];
    int i=1, a=0;
    printf("Enter comment:");
    gets(com);
    if(com[0]=='/')
```

```
{
if (com[i]=='/')
    printf("\n It is a comment");
else if (com[i]=='*')
  {
    for(i=2;i<=30;i++)
    {
    for(com[i]=='*'&&com[i+1]=='/')
      {
        printf("\n It is a comment");
        a=1;
        break;
      }
      else
      continue;
    }
    if(a==0)
        printf("It is not a comment");
  }
  else
    printf("It is not a comment");
}
else
  printf("It is not a comment");
}
```

192011494L

output:

Input: Enter commad : //hello

output: It is a comment

o/p
Verified

Result

By developing a lexical Analyzer identify whether a given line is Comment or not using C

28/7/25

EXNO:1 Ignore spaces, comment using C

Aim: To design a lexical Analyser for given language which ignore redundant spaces, tabs and new lines and ignore comments using C.

Algorithm:

1. Start the program
2. Declare the input statement
3. import the required modules
4. Declare the variables
5. Run the codes

Program

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
int isKeyword(char buffer[]){
char keywords[32][10]=
{"main","auto","break","case","char","const"
"default","do","double","float","goto","int"
"long",4};
```

```
int i, flag=0;
for(i=0;i<32;++i){
if(strcmp(keywords[i],buffer)==0){
flag=1;
break;
}
}
return flag;
}
int main(){
char ch, buffer[15], operator[]="+-*/%=";
FILE *fp;
printf("error while opening the file\n");
exit(0);
}
while((ch=fgetc(fp))!=EOF){
    for(i=0;i<6;++i)
if(ch==operator[i])
printf("%c is operator\n",ch);
```

```
printf("%s is identifier \n", buffer);
}
}
fclose(fp);
return 0;
}
```

Input:

```
main()
{
int a,b,c;
c= b+c;
Printf("%d", c);
}
```

Output:

main is keyword
int is keyword
a is identifier
b is identifier
c is identifier
(is identifier
= is operator
+ is operator
) is operator

o/p varies

1980114941

printf → Keyword

d is identifier

c is identifier

Result :

28/7/23

Ignoring the redundant space, tab and new line and comment has been ignored by developed lexical Analyzer using C.

EXPT 4

PROGRAM

Validate operators to recognize +,-*/, using regular expression in C.

Aim:

To develop a lexical Analyzer to validate operator to recognize the operator +,-,*,/, using regular arithmetic operator using C.

Algorithm:

1. import the module
2. declare the variable
3. initialize the operation
4. Run the code

Program:

```
#include <stdio.h>
#include <conio.h>
int main()
{
    char s[5];
    printf("\n Enter any operator:");
    gets(s);
    switch(s[0]):
    {
        case '+':
            if(s[1]=='+')
```

```
        printf("\n greater than or equal ");
    else
        printf("Greater than ");
    break;
case '<';
    if (s[i] == '=')
        printf("\n less than or equal ");
    else
        printf("less than ");
    break;
case '&';
    if (s[i] == '&')
        printf("logical AND");
    else
        printf("Bitwise AND");
    break;
case '|';
    if (s[i] = '|')
    printf("logical OR");
case '+':
    printf("Addition");
    break;
```

```
case '+':
    printf("addition");
    break;
case '-':
    printf("subtraction");
    break;
case '*':
    printf("multiplication");
    break;
case '/':
    printf("division");
    break;
case '%':
    printf("modulus");
    break;
default:
    printf("Not a operator");
}
}
```

Output Enter any operator; <= less than or equal.

Result

Recognizing the valid operator normal Arithematic operator like +, -, *, / can be identified using

EXP NO:5 Find the number of white space and newline character using C

Aim: To design a lexical-Analyzer to find the number of whitespace and newline Character using C.

Algorithm:

1. Start the program
2. Import the modules
3. declare the variable
4. Run the code and execute the code

Program:

```
#include<stdio.h>
int main()
{
 char str[100];
int words=0, newlin=0, character=0,
scanf("%s[^\n]",&str);
 for (int i=0; str[i]!='\0';i++)
  {
  if(str[i]==' ')
    {
    words++;
  }
  else if(str[i]=='\n')
```

```
{
  newline ++;
  word ++;
}
else if (str[i] != ' ' && str[i] != '\n')
  character ++;
}
}
if(character > 0)
{
  word ++;
  newline ++;
}
printf("Total number of word : %d \n", word);
printf("Total number of line : %d \n", newline);
printf("Total number of character : %d\n",
                                    character);
return 0;
}
```

Output

```
void main()
{
  int a;
  int b;
```

192a14441

```
a = b+c;
c = d*e;
}
```

all words

Total number of words : 18

Total number of lines : 7

28/7/25

Result:- number of white spaces and newline characters is identified and Removed.

EXP.NO:6 Test whether a given identifier is valid or Not using c

Aim:- To develop lexical Analyser to test whether a given identifier is valid or not using c.

Program:-

```
#include <stdio.h>
#include<conio.h>
#include <ctype.h>
int main()
{
  char a[10];
  int flag i=1;
  printf("Enter the identifier:\n");
  gets(a);
  if (isalpha(a[0]))
     flag =1;
  else
     printf("Not a valid identifier\n");
     while (a[i]!='\0');
{
```

```
if (! isdigit (a[i]) && ! isalpha(a[i]))
{
flag = 0;
break;
}
i++;
}
if (flag==1)
printf("In valid identifier");
```

Output:

Enter an identifier : abc123

valid identifier.

Result: given identifier is valid or not has been identified

28/7/23

EXPT NO: 3

RA2011494

To Find First () - predictive paper for the given grammer.

Aim: To find First() - predictive parse for the grammer

S → AaAb / BbBa

A → ε

B → ε

Algorithm:

1. Start the code and import the modu
2. declare the variable
3. write the Required code
4. Run the code

Program:

```
#include<stdio.h>
#include<conio.h>
void First(char[],char);
void addToResultSet(char[],char);
int numOfProductions;
char productionSet[10][10];
int main()
{
int i,
```

192011494L

```
char choice;
Char C;
printf("How many number of productions");
Scanf("%d", &numof production);
for (i=0 ;i<numOfproduction ;i++);
{
printf("Enter Production Number %d:i+1)
scanf("%s",production[i]);
}
do
Void Find (char * Result,Char C)
{
int i ,j,k;
char SubResult[20];
int foundException;
SubResult[0]='!0';
Result[0]='0'
for (i=0 ;i<numofproduction ;i++).
{
if (production Hes add A to First(x)-
```

```
return;
}
void addToResultSet(char Result[],char
                    val)
{
int k;
for(k=0;Result[k]!='\0';k++).
    if(Result[k]==val)
        return;
    Result[k]=val
    Result[k+1]='\0'.
}
```

Output:

How many number of productions:4

Enter production Number 1: S=AaBb

Enter production Number 2: S=BbBa

Enter production Number 3: A=$

Enter production Number 4: B=$

Find the FIRST of :S

FIRST(S) = {a b}

press 'y' to continue :y

Find the First of :A

First(A) = {ε}.

press 'y' to continue :y

Find the First of :B

First(B) = {ε}

press 'y' to continue :n.

Result

To Find FIRST () → predictive parser

for the given grammar is done

28/7/25

Experiment No.

To find Follow() - predictive parser for the given number.

Aim: To write a c-program to find follow () - predictive parser for the given grammer.

$S \rightarrow AaAb / BbBa$

$A \rightarrow \epsilon$

$B \rightarrow \epsilon$

Algorithm:

1. Start the program
2. Import the packages
3. Declare the variables
4. Write required code.
5. Run the program.

program:

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
int count,n=0
char production[10][10],array[10];
void findfirst(char c);
```

```
void find_follow(char ch);

void array_manipulation(char ch);

int main ()
{
 int count;
 char option, ch;
  printf (" Enter Total Number of
                    productions :%d");
 scanf ("%d", &limit);

 for (count=0; count<limit; count++)
  {
  printf ("Value of production number %d
  scanf ("%c", production[count]);
  }
 do
 {
  x=0;
    Printf ("Enter the production variable
    find follow:
 }
}

Void FIRST (char ch)
 {
   int i,k;
```

```
if (!isupper(c1))
{
  Array_manipulation(c1);
}
for(k=0;k<count;k++)
{
  if(production[k][0]==c1]
  {
    find_follow(production[i][0]);
  }
  else if(isupper(production[k][2]))
  {
  }
  }
}
void Array_multiplication(char c1)
{
  int count;
  for(count=0;count<=k;count++)
  {
    if(array[count]==c1)
    {
      return;
    }
}
```

```
array[x++]=c;
}
```

output:-

```
Enter total number of production :4
Value of Production Number[1]:S=AaAb
Value of Production Number[2]:S=BbBa
value of production Number[3]:A=$
Enter production value to find follow:S
follow value of S:{$}
To continue,press Y:y
Enter production value to find follow:A
follow value of A:{a,b}
To continue,press Y:y
Enter production value to find follow B
follow value of B:{ba}
To continue,press Y:n
```

28/1/23

Result: c-program is written to find follow() predictive parser for the given grammer.

EXNO:7 implement a 'c program to eliminate left recursion from a grammar

Aim: To implement a 'c program to eliminate left Recursion from a given CFG.

S → (L)/a

L → L,S/S

Algorithm:

1. Start the code
2. import all packages required
3. Declare the variables
4. Run the code and execute it

Program:

```
#include <stdio.h>
#include<string.h>
#define size 10
int main() {}
{
  char non-terminal;
  char beta, alpha;
  int num;
  char production[10][size];
  printf("Enter the number of productions");
  Scanf("%d", &num);
  printf("Enter the grammer as E->E+A
```

192044944

```
for (int i=0 ;i<num ;i++){
        scanf ("%s", production[i]);
    }

if (non_terminal == production [i] [index]) {
alpha = production [i] [index+1];
printf (" is left recursive");
while (production [i] [index] != 0 && Production
                                [i] [index] != '|')
    index++
    beta = production [i] [index+1];
    printf ("Grammar without left recursion");
    }
    else
        printf ("cant be reduced");
    }
    else
        printf (" is not left recursive.
    index =3;
    }
}
```

output :-

Enter the number of productions 2

Enter the grammar as E - E+A

S -> (L) | a

L -> L,S | S.

Grammar :: S->(L) | a is not left recursive

Grammar :: L -> L,S | S is left recursive.

Grammar without left Recursion:

L -> SL'

L' -> ,L' | E

Result :-

Left Recursion eliminator from a is implemented by using c.

28/1/23

Exp.No: Eliminate left factoring from a given CFG. 19201142LL

Aim:- To implement a C program to eliminate left factoring from a given CFG.

S → iEts / iEtses / a

Algorithm:

* Start the code and import the package.
* Declare the variables.
* Assign the variables.
* Save and Run the code.

Program:

```
# include<stdio.h>
# include<string.h>
int main()
{
  char
gram[20], part1[20], part2[20], modifiedGram[20],
int i,j=0, k=0, l=0, pos;
printf("Enter Production:S->");
gets(gram);
for(i=0;gram[i]!='|';i++,j++).
    part1[j]=gram[i];
```

```
for(i=++pos ; i=0; gram[i] != '\0' ; i++, pos++)
    part2[i] = gram[i];
part2[i] = '\0';

for (i=0; i<strlen(part1) || i<strlen(part2); i++)
{
  if (part1[i] == part2[i]).
    {
      modified Gram[k] = part[i];
      k++;
      pos = pos+1;
    }
}
  modified Gram[k] = 'x';
  modified Gram[++k] = '\0'
  new Gram[i] = '\0';
  printf("\n modified gram");
  printf("\n %s", new gram);
}
```

192011434L

Output:-

Enter Production: S -> iEtS | iEtSeS | a

S -> iEtSX

X -> eS | a

28/11/25

Result:-

left-factoring from a given CFG is eliminated using C.

EXP 10 C program to perform Symbol table operations

Aim: To implement a c program to perform Symbol table operations.

Algorithm:

* Start the code and import package
* Declare the variables.
* Assign the variable and write the logic
* Save and Run the Code.

Program:

```
#include <stdio.h>
#include< stdlib.h>
#include< string.h>
int cnt=0;
Struct Symtb
{
  Char label [20];
    int addr;
};
Sy[30];
Void insert();
int search (char*);
Void display()
```

```
void main()
{
int ch, val;
char lab[10];

do
 {
  printf(" 1.insert 2. display 3.search, 4.modify.
    5. exit ");
  scanf("%d", &ch);
  switch(ch)
  {
     case 1:
        insert();
        break;
     case 2:
          display();
          break;

     case 3:
printf("Enter the label");
          scanf("%s", lab);
          val = search(lab);
          if(val == 1)
          printf("lable is found");
     else
        printf("label is not found");
```

```
        break;

        case 4:
            modify();
            break;
        }
    } while (ch<5)
}
void input()
{
int val;
    char lab[10];
    int symbol;
    printf("Enter the label");
    return;
    }
}
void modify()
{
  int val, ad, i;
  char lab[10];
  printf("enter the label");
  scanf("%s", lab);
   val = search(lab);
   if (val==0)
   printf("no such symbol");
```

19201A04U

```
else
{
printf("label is found");
printf("enter the address");
scanf("%d", &add);
for(i=0; i<cnt; i++)
{
if(strcmp(sy[i].label, lab)==0)
}
void display()
{
int i;
for(i=0; i<cnt; i++).
}
```

output:

1. insert
2. display
3. search
4. modify
5. exit

1.
enter the label a
enter the address 1000

Result: Symbol table operation are performed by using c program.

EXPT NO:12 Construct Recursive descent Parsing for given grammer.

19201149

Aim:- To construct recursive descent parsing for the given grammer.

Algorithm:-

* Start the compiler and import packages.
* Declare the variables
* Assign the value for the variables.
* Run the code and get output

Program:-

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
Char input[100];
int i,l;
void main()
{
clrscr();printf("Recursion descent Parsing for
the following grammar:");
gets(input);
{
if(input[i+1]=='0')
printf("\n String is not accepted");
```

```
else
printf("String not accepted");
getch();
}
E()
{
if(T())
{
if(EP())
return (1);
else
}
return(0);
}
if (EP())
return(1);
else
return (0)
}
else
return(0);
if TP())
return (1);
else
return 0);
}
else
return(0);
```

```
}
else.
return(1);

F()
if(input[i]=='('
{
i++;
if (input[i]==')')
{
i++;
return(1);
}
else.
return(0);
}
else if(input[i]>='a' && input[i]<='z'||
{
i++;
return (1);
}
else.
return(0);
}
```

output:

Recursive decent parsing for the following grammar.

E → TE'

E' → +TE' | @

T → FT'

T' → *FT' | @

F → (E) | ID

Enter the string to be checked (a+b)*c

String is accepted.

Enter the string to be checked : a |c+d

String is NOT ACCEPTED

28/11/23

Result:

Recursion decent Parsing for the given grammer is done using c program.

EXP.NO:13 Top down parsing technique or Bottom up Parsing technique

19201149

Aim:- To implement either Top down parsing technique or Bottom up parsing technique to check whether the given input string is valid according to the grammar or not

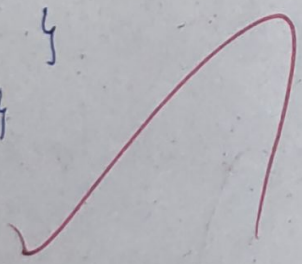
Algorithm

1. import the in-built Package
2. Declare the variables
3. Assign the variable
4. Run the code and execute

Program

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
char string[50];
int flag,count=0;
printf("The grammar is: S->aA ; S->Sb ; S->ab");
printf("Enter the string to be checked:");
gets(string)
if(string[0]=='a'){
flag=0;
for(count =1; string[count-1]
```

```
if ((flag==1)&&(string[count]=='a'))
    printf("The string does not belong to the
            specified grammar.");
    break;
}
else if (string[count]=='a')
    continue;else
if (flag==1)&&(string[count]=='b'){
    break;
}
}
}
}
```



Output:

The grammar is :S->aA / S->bB / S->ab.

Enter the string to be checked:

abb

String accepted

Result:

Top down parsing technique or Bottop up parsing technique is done By using C-Program.

EXP.NO:14 Shift reduce parsing in C Programming

19201142

Aim:- To implement the concept of shift reduce parsing in c programming.

Algorithm

* import the packages.
* Declare the variables.
* Assign the variables
* Save and Run the code

program

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
char ip_sym [15], stack [15]; int ip_ptr =0
St_ptr =0; len ;i;
Char at[15];
void check ();
int main()
{
printf(" SHIFT REDUCE PARSER");
print("\t\t\t");
temp [0] = ip_sym [ip_ptr];
```

1920/14926

```
stack[st_ptr] = ip_sym[ip_ptr];

int flag=0; temp2[0] = stack[st_ptr]; temp2[0]='\0';

if((!strcpy(stack,"E+E"))||(!strcmpi(stack,"E
{
strcpy(stack,"E"); st_ptr=0; if(!strcmpi(stack,"E
printf("\n %s\t\t%s\t\t E->E+E", stack, ip_sym);
else
if(!strcmpi(stack ip_sym);else
printf("\n%s\t\t %s\t\t E->E+E",
        stack,ip_sym); flag=1;
if(!strcmpi(stack,"E")&&ip_ptr==len)
{
printf("\n$%s\t\t$\t\t %s\t\t accept", stack, ip_sym);
getch();
exit(0);
}
if(flag==0)
{
printf("\n%s\t\t%s\t\t reject", stack, ip_sym);
exit(0);
}
return 0;
```

}

output:-

SHIFT REDUCE PARSER

GRAMMER

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$	- -
$a	+b$	shift a
$E	+b$	E→a
$E+	$	shift b
$E	$	E→E+E
$E	$	Accept

Result:- shift reduce parser in c prog is done.

EXP NO:-15 Implement the operator Precedence parsing. 19201104041

Aim:- To implement the operator precede-ce parsing By using c program

Algorithm:-

* Start code and implement or import various package needed.
* Declare the variables.
* Assign the variables.
* Run the code and Execute it.

Program:-

```
#include <stdio.h>
#include <conio.h>
char *input;
int i=0;
char lasthandle[6], stack[50], handles[][5]=
{"E+E","E*E","i","E^E"};
int top=0, l;
char Prec[9][9]={
int getindex(char c)
```

```
{
 switch (c)
  case '$': return 0;

  case '-': return 1;
  case '*': return 2;
  case '/': return 3;
  case '^': return 4;
  case '(': return 5;
  case ')': return 6;
  case 'i': return 8;
 }
}

 int Shift ()
 {
 stack [++top ]= * (input + i++);
 }
 int reduce ()
  {
  int i, len, found;
  {
  int (i=0 ; i<5 ; i++)
    {
    length = strlen (handle[i]);
```

```
if (prec[getindex(stacktop)][getindex(
        (input)[i]=='$')
{
while(reduce())
    {
    printf("\n");
    display();
    printf("Reduced : E -> %s", lhs[rhs]);
    }
}
}
printf("\n Not accepted");
}
```

output:-

Enter the string

i*(i+i)*$

STACK	INPUT	ACTION
$i	*i+(i+i)*i$	Reduce E->i
$E	*(i+i)*i$	Shift
$E*	(i+i)*i$	Reduced E->i
$E*(	i+i)*i$	Shift

$E $ Reduced E→E*E 1920114941

$E $ shift

Accepted $ $ → E

-Accepted.

Result:-

operator precedence parser by using

c-program is implemented.

EXP: NO: 16 Three Address code representation for the given input statement.

1920114941

Aim: write a c-program to generate the three Address code Representation for the given input statement

Algorithm:

1. import package and Run start the code.
2. Declare the variables.
3. Assign the variable.
4. Run the code and execute it.

Program:

```
#include <stdio.h>
#include<conio.h>
#include<stdlib.h>
struct three
{
char data[10], temp[7]; 4
[30];
int main()
{
char d1[7], d2[7]="t";
int i=0, j=1, len=0
fi=fopen("new.txt","r");
f2=fopen("out.txt","w");
while(!feof(fi)&&ch[len]!=EOF)
len++
```

```
strcpy(d1," ");
strcpy(d2,"t");
strcat(i,d1);
if(!strcmp(S[i+1].data,"+"));
fprintf(f2,"%s=%s+%s",S[i].temp,S[i-1].temp,S[i+1].
data);
else if(!scmp(S[i+1].data,"-")
fprintf(f2,"%s=%s-%s",S[i].temp,S[i-1].temp,
S[i+1].data);
strcpy(d1," ");
strcpy(d2,"t");
j++;
}
fprintf(f2,"%s=%s",S[0].data,S[i-1].temp
fclose(f1);
fclose(f2)
getch();
}
```

Program 4

Output:

Input : Source.txt

out = in1 + in2 * in3 - in4

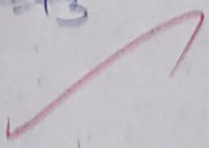
Output: out.txt

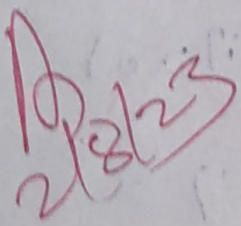
t1 = in1 + in2

t2 = t1 * in3

t3 = t2 - in4

out = t3





Result:

Three-Address code generator Representation for the given input statement

Exp.No:17 Lexical Analyzer to scan and count the no of characters, words and lines in a file. 19/2/24

Aim: Implementing a lexical analyzer to scan and count the number of characters, words and lines in a file.

Algorithm:

1. Import the packages
2. declare the variables.
3. Assign the variables.
4. Save and Run the code.

Program:

```
#include<stdio.h>
int main()
{
  char str[100];
  int word=0, newline=0, character=0;
  scanf("%[^~]", &str);
  for(int i=0; str[i]!='\0'; i++)
    {
      if(str[i]==' ')
        {
          word++;
        }
      else if(str[i]=='\n')
        word++;
    }
  else if(str[i]>='a' && str[i]<='z')
}
```

```
else if(str[i]!=' ' && str[i]!='\n'){
  character++;
}
}
if(character>0)
{
  word++;
  newline++;
}
printf("Total number of words:%d",word);
printf("Total number of lines:%d",newline);
printf("Total number of characters:%d",character);
return 0;
}
```

output:

```
void main()
{
int a;
int b;
a=b+c;
c=d*e
}
```

Total number of words : 18

Total number of lines :-

10/8/22

Result:- to implement lexical analyzer to scan and count the number of characters, words, and lines in a file

Roll 4941

EX.NO:11 C program to implement the Back end of the compiler.

Aim: C program to implement the Back end of the compiler.

Algorithm:

1. import the package
2. Declare the variables.
3. Assign the variables
4. Run the code and execute

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
int n,i,j;
char a[50][50];
printf("enter the no: intermediate code:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter the 3address code %d:",i+1);
for(i=0;i<6;i++
{
printf("enter the 3 address code
for(i=0;i<6;i++)
{
scanf("%c",&a[i][j]);
```

PROGRAM

```
{
printf("\n mov %c ,R %d ",a[i][3],i);
if(a[i][4]==':')
{
printf("\n sub %c ,R%d ",a[i][5]);
}
if a[i][4]=='+')
{
printf("\n add ,%c ,R %d ",a[i][5],i);
}
if (a[i][4] == '*')
{
if(a[i][4]=='/')
{
printf("\n div %c ,R%d",a[i][5],i);
}
printf("\n mov R%d %c",i,a[i][0]);
printf("\n");
}
return 0;
}
```

2/8/22

output:-

enter the no: intermediate code:2

enter the 3 address code : 1:a=b+c

enter the 3 address code : 2 d=a*c

-the generated code.

mov b,R0.

add c,R0

mov R0,a

mov ..,R1

Muld,R1

mov R1,d

Result: Implementation of Back end of the compiler is done.

EXP.NO: 12. Operator to compute LEADING operator precedence for the given grammar. Mohamed

Aim: Operator to compute LEADING operator precedence for the given grammar.

Algorithm:

1. import package and start with the code.
2. Declare the variable
3. Assign the variable
4. Run the code and execute it

Program:

```
#include <conio.h>
#include <stdio.h>
char prod[] = "EETTFF";
char arr[6][3] = {{'E','+','T'},{'T','0'},{'T','*'}
char stack[5][2];
int top = -1;
void index(char pro, char re){
int i;
for(i=0; i<18; i++){
if (arr[i][0] == pro && arr[i][1] == re){
for (i=0; i<18; i++){
printf("\n");
for (j=0; j<3; j++)
```

```
        printf("%c \n",arr[i][j]);
    }
    getch();
    printf("\n\n");
    for(i=0;i<18;i++){
        if(pri!=arr[i][0]){
        pri=arr[i][0];
        printf("\n\t%c -> ",pri);
        }
        if(arr[i][1]=='T'){
            printf("%c ",arr[i][j]);
        }
    getch();
    }
```

output:-

E + T

E * T

E (T

E) F

E i T

E $ F

F * F

F i T

T * T

T (T

T) F

T i T

T $ F

F ->+ *(i)

F-> (i

T -> *(i

Rewrite:

C-program to compute leading of operator precedence for the given grammar.

EXP:NO:8 C program to Compution TRAILING operator precedes Parser for the given gram

Aim: C program to compute TRAILING- oper precedes parser for the given gram

E → E+T | T

T → T*F | F

F → (E) | id

Algorithm:

1. import the package
2. Declare the variable
3. Assign the variable
4. Run the code and Execute

Program:

```
#include<conio.h>
#include<stdio.h>
Char arr[10][3] = {{'E','+','('},
Char prod[6] = "EE TTFF";
Char rel[6][3] = {{'E','+','F'},
Char stack[5][2];
int top = -1;
void push(char pro, char rel){
    int i;
    for(i=0; i<1; i++){
```

```
if(arr[i][0] == pro && arr[i][1] == rel
{
for (i=0; i<if; i++){
  printf("\n");
  for (j=0; j<3; j++)
    printf("%c\t", arr[i][j]);
}
printf("\n\n");
for(i=0; i<if; i++){
  if(pri != arr[i][0]){
    printf("\n%c -> ", pri);
}
if(arr[i][2] == 'T')
}
if(arr[i][1] == 'T')
  printf("%c ", arr[i][0]);}
}
```

outputs

E + F

E * F

E (F

E) F

E i F

RB0114941

E $ F

F + F

F (F

F) F

F $ F

T * F

T) F

T i F

T $ F

E ->

F->

T->

Result

C program to compute Pairlow operator precedence parser for the given gramm

Expno:-21 Count the number of characters, number of lines & number of words.

192011491

Aim: To find and count the number of characters, number of lines & number of words.

Input source program:-

```
#include <stdio.h>
int main()
{
int num1, num2, sum;
printf("Enter two integers:");
scanf("%d %d", &number1, &number2);
sum = number1 + number2;
printf("%d + %d = %d", num1, num2, sum);
return 0;
}
```

Algorithm:

1. write a input program
2. Now import package
3. write code for counting characters.
4. Run the code.

Program (count-lines.l)

```
%{
int nchar, nword, nline;
%}
```

```
\n {nline ++; nchar ++;}
[^ \t\n]+ {nword ++; nchar += yyleng;}
{nchar ++;}

int yywrap (void){
return 1;
}
int main (int argc, char *argv[]){
YYin = fopen (argv[1], "r");
printf("Number of Characters = %d \n",
                        nchar);
printf("Number of words = %d \n", nword);
printf("Number of lines = %d \n", nline);
fclose (yyin);
}
```

Output:

```
G:\flex>flex count_line.l

G:\flex>gcc lex.yy.c

Number of Characters = 233

Number of words = 33

Number of lines = 10
```

192011494LL

Result Count of number of characters, number of line & number of words and done by using lex program.