Ex No.1

Develop a lexical Analyzer to identify, constants, operators using C program.

Aim:- To develop lexical Analyzer to identify, constants, operators using C program.

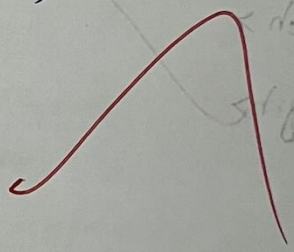
Algorithm:-

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

Program:-

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
int main()
{
  int i, ic=0, m, cc=0, oc=0, j;
  char b[30], operators[30], identifiers[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]))
{
  identifiers[ic]=b[i];
  ic++;
}
else if (isdigit(b[i]))
{
  m=(b[i]-'0');
  i=i+1;
  while (isdigit(b[i]))
  {
    m=m*10+(b[i]-'0');
    i++;
  }
  i=i-1;
  constants[cc]=m;
  cc++;
}
else
{
  if(b[i]=='*')
  {
    operators[oc]='*';
    oc++;
  }
  else if(b[i]=='-')
  {
    operators[oc]='-';
    oc++;
  }
  else if(b[i]=='+')
  {
```



```
        operators[oc] = '+';
        oc++;
    }
    else if(b[i]=='=')
    {
        operators[oc] = '=';
        oc++;
    }
        }
    }
printf("identifiers: ");
for(j=0;j<ic;j++)
{
    printf("%c", identifiers[j]);
}
printf("\n constants: ");
for(j=0; j<cc; j++)
{
    printf("\n operator: ");
    for(j=0; j<oc; j++)
    {
        printf("%c", operators[j]);
    }
}
```

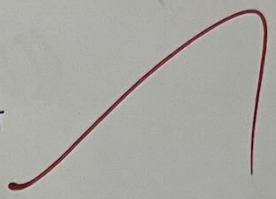
Output:-

enter the string: a-b+c*e+100

identifiers : a b c e

constants : 100

operators : = + * +



⊙/10
Verify

B
27/7/23

Result:- A C program to identify constants, operators and identifiers is developed.

(2) Develop a lexical Analyzer to identify whether a given line is Comment or not.

Aim:- To develop a lexical Analyzer to identify the given line is comment or not.

Algorithm:-

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

Program:-

```
#include <stdio.h>
#include <conio.h>
int main()
{
  char com[30];
  int i=2, a=0
  printf("\n Enter comment: ");
  gets(com);
  if(com[0]==' / ')
  {
    if(com[1]==' / ')
    {
      printf("\n It is comment");
    }
    else if(com[1]==' * ')
    {
      for(i=2; i<=30; i++)
      {
```

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

Output:

I/P: Enter a comment: //hello

O/P: It is a comment

I/P: Enter a comment: hello

O/P: It is not a comment.

28/7/25

Result:- Using C program, we developed a lexical Analyzer to check it is Comment or Not.

③ Design a lexical Analyzer for given language should ignore the redundant spaces, tabs and new lines and ignore comm

Aim:- To design a Analyzer for given language should ignore the redundant spaces, tabs and new lines & ignore comments

Algorithm:-

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

Program:-

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
int isKeyword(char buffer[]){
char keywords[32][10] = {"main", "auto", "break", "case", "char", "const",
"continue", "default", "do", "double", "else", "enum", "extern", "float",
"for", "goto", "if", "int", "long", "register", "return", "short",
"signed", "sizeof", "static", "struct", "switch", "typedef", "unsigned",
"void", "printf", "while"};
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], operators[] = "+-*/%=";
    FILE *fp;
    int i,j=0;
    fp=fopen("lex_input.txt","r");
    if(fp==NULL){
        printf("error while opening the file\n");
        exit(0);
    }
    while((ch=fgetc(fp))!=EOF){
        for(i=0;i<6;i++){
            if(ch==operators[i])
                printf("%c is operator\n",ch);
        }
        if(isalnum(ch)){
            buffer[j++]=ch;
        }
        else if((ch==' '||ch=='\n')&&(j!=0)){
            buffer[j]='\0';
            j=0;
            if(isKeyword(buffer)==1)
                printf("%s is keyword\n",buffer);
            else
                printf("%s is identifier\n",buffer);
        }
    }
```

```
    fclose(fp);
    return 0;
}
```

Input: 3lex _input.txt

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

Output:-

main is Keyword
int is Keyword
a,b,c is indentifier
= is operator
+ is operator
c is indentifier
print is Keyword
% is operator
d is indentifier
c is indentifier.

o/p verified

28/7/23

Result:- Designed a lexical Analyzer for given language should ignore redundant spaces, tabs and new lines and ignore comments.

④ Design a lexical Analyzer to validate operators to recognize the operators +, -, *, / using regular arithmetic operations

Aim:- To design a Analyzer to validate operators +, -, *, / using C programing.

Algorithm:-

* Start the code and import the packages.
* Declare the variables.
* Assign the variables.
* Save and 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("\n Greater than");
     break;
     case '<':
     if(s[1]=='=')
```

```
        printf("\n Less than or equal");
    else
        printf("\n Equal to");
    break;
case '=':
    if(s[i]=='=')
        printf("\n Equal to");
    else
        printf("\n Bit Not");
    break;
case '&':
    if(s[i]=='&')
        printf("\n Logical AND");
    else
        printf("\n Bitwise AND");
    break;
case '|':
    if(s[i]=='|')
        printf("\n Logical OR");
    else
        printf("\n Bitwise OR");
    break;
case '+':
    printf("\n Addition");
    break;
case '-':
    printf("\n Subtraction");
    break;
```

```
        case '*':
            printf("\n Multiplication");
            break;
        case '%':
            printf("Modulus");
            break;
        default:
            printf("\n Not an operator");
    }
}
```

Output:-

Enter any operator: <=

Less than or equal

8/1 verified

28/1/23

Result:- Design an Analyzer to identify the operators +, -, *, / using C program.

5 Design a lexical

Aim:- To desig
whitespaces &

Algorithm:-

- Start the
* Declare v
* Assign vari
* Save and

Program:-

```
#include <std
int main()
{
    char str
    int words
    scanf("%
    for(int i
    {
        if(str
        {
        }
    }
    els
    {
```

(5) Design a lexical Analyzer to find the no. of whitespaces & newline characters

Aim:- To design a lexical Analyzer to find the no. of whitespaces & newline characters.

Algorithm:-

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

Program:-

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

```
        characters++;
    }
}
if(characters >0)
{
    words++;
    newline++;
}
printf("Total no. of words: %d\n", words);
printf("Total no. of lines: %d\n", newline);
printf("Total no. of characters: %d\n", characters);
return 0;
}
```

Output:-

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

Total no. of words: 18

Total no. of lines: 7

Result:- Designed a lexical Analyzer to find newline characters & whitespaces using C program.

(6) Develop a lexical Analyzer to test whether a given identifier is valid or not

Aim:- To Develop a Analyzer to test whether a given identifier is valid or not.

Algorithm:-

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

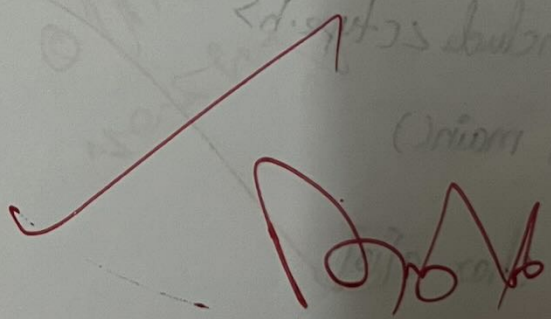
Program:-

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

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

Output:

Enter an identifier : abc123
Valid identifier



Result:- Developed a Analyzer to test whether a given identifier is valid or not using C.

(7) Write a C program for FIRST() - predictive parser for given grammar

Aim:- To write a C program to find FIRST() - predictive parser for given grammar.

Algorithm:-

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

S → AaAb/BbBa
A → ε
B → ε

Program:-

```
#include <stdio.h>
#include <ctype.h>
void FIRST(char[], char);
void addToResultSet(char[], char);
int numOfProductions;
char productionSet[10][10];
int main()
{
    int i;
    char choice;
    char c;
    char result[20];
```

```
scanf("%d", &numofProductions);
for(i=0;i<numofProductions;i++)
{
  printf("Enter productions Number %d: ", i+1);
  scanf("%s", production Set[i]);
}
do
{
  printf("\n find the first of: ");
  scanf("%c", &c);
  FIRST(result, c);
  printf("\n FIRST(%c) = {", c);
  for(i=0; result[i]!='\0'; i++)
  printf("%c", result[i]);
  printf("}\n");
  printf("press 'y' to continue: ");
  scanf("%c", &choice);
}
while(choice=='y'||choice=='Y');
}
Void FIRST(char* Result, char c)
{
  int i, j, k;
```

```
    char subResult[20];
    int foundEpsilon;
    subResult[0]='\0';
    Result[0]='\0';
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 no. of productions ? : 4

Enter production no. 1 : S=AaAb

Enter production no. 2 : S=BbBa

Enter production no. 3 : A=$

Enter production no. 4 : B=$

Find FIRST of : S

FIRST(S) = { $ a b }

Press 'y' to continue : y

Result: A C program for FIRST-predictive parser for given grammar is developed.

(8) Write a C program to FOLLOW() - predictive parser of grammar

Aim: To write a C program to FOLLOW() - predictive parser for the given grammar.

Algorithm:

* Start the code and import production/packages.
* Declare the variables.
* Assign the variables.
* Start and Run the code.

$S \rightarrow AaAb \mid BbBa$

$A \rightarrow \varepsilon$

$B \rightarrow \varepsilon$

Program:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void main() {
    int limit, x=0;
    char production[10][10], array[10];
    void find_first(char ch);
    void find_follow(char ch);
    void Array_Manipulation(char ch);
```

FOLLOW() - predictive

packages.

```
int main()
{
    int count;
    char option, ch;
    printf("\n Enter Total no. of Productions: ");
    scanf("%d", &limit);
    for(count=0; count<limit; count++)
    {
        printf("\n Value of Production No. [%d]: \t", count+1);
        scanf("%s", production[count]);
    }
    do
    {
        x=0;
        printf("\n Enter production value to find FOLLOW: \t");
        scanf("%c", &ch);
        find_follow(ch);
        printf("\n Follow Value of %c: \t{", ch);
        for(count=0; count<x; count++)
        {
            printf("%c", array[count]);
        }
        printf("}\n");
        printf("To Continue, Press Y: \t");
        scanf("%c", &option);
    }
```

```
void Array_Manipulation(char ch)
{
    int count;
    for(count=0; count<=x; count++)
    {
        if(array[count==ch)
        {
            return;
        }
    }
    array[x++]=ch;
}
```

Output:-

Enter Total no. of Productions: 4

Value of Production No. [1]: S=AaAb

Value of Production No. [2]: S=BbBa

Value of Production No. [3]: A=$

Value of Production No. [4]: B=$

Enter production Value to find FOLLOW: S

FOLLOW value of S: { $ }

To Continue, Press Y: y

Result: A C program to FOLLOW (1) -predictive parser for a given grammar is developed.

⑨ Implement a C program to eliminate left recursion from a given CFG.

Aim:- To implant a C program to eliminate left recursion from a given CFG.

Algorithm:-

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

S → (L)/a

L → L,S/S

Program:-

```
#include <stdio.h>
#include<string.h>
#define SIZE 10
int main () {
char non_terminal;
  int num;
  char beta, alpha;
  char production [10][SIZE];
  int index=3;
  printf("Enter the grammar as E →E-A : \n");
  for (int i=0 ; i<num; i++) {
      scanf("%s", production [i]);
  }
```

```
for(int i=0; i<num; i++) {
    printf("\n GRAMMAR ::: %s", production[i]);
    non_terminal = production[i][0];
if(non_terminal == production[i][index]) {
    alpha = production[i][index+1];
    printf(" is left recursion\n");
    while(production[i][index] != 0 && production[i][index] != '|')
        index++;
    if(production[i][index] != 0) {
        beta = production[i][index+1];
        printf("Grammar w/o left recursion:\n");
        printf("%c -> %c %c'\n", non_terminal);
        printf("\n%c' -> %c%c' |E\n", non_terminal);
    }
else
    printf(" can't be reduced\n");
}
else
    printf(" is not left recursive.\n");
    index = 3;
}
}
```

Output :-

Enter No. of Production : 2

Enter the grammar as E → E-A :

S → (L) | a

S → L, S | S

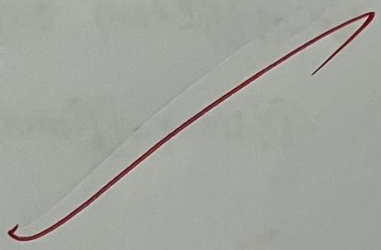
GRAMMAR ::: S → (L) | a is not left recursive.

GRAMMAR ::: L → L, S | S | S is left recursive.

Grammar w/o left recursion.

L → SL'

L' → , L' | E



Result :- To eliminate left recursion from a given CFG using C program is developed.

(10) Implement a C program to eliminate left factoring from CFG.

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

Algorithm:-

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

S → iEtS | iEtSeS | a

E → b

Program:-

```
#include <stdio.h>
#include <string.h>
int main()
{
  char
  gram[20], part1[20], part2[20], modifiedGram[20], newGram[20]
   tempGram[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];
   part1[j] = '\0';
```

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

for (i = 0; i < strlen(part1) || i < strlen(part2); i++)
{
  if (part1[i] == part2[i])
  {
     modifiedGram[k] = part1[i];
     k++;
     pos = i+1;
  }
}

for (i = pos, j = 0; part1[i] != '\0' ; i++, j++){
     newGram[j] = part1[i];
}

modifiedGram[k] = 'X';
modifiedGram[++k] = '\0';
newGram[j] = '\0';
printf ("\n S -> %s", modifiedGram);
printf ("\n X -> %s \n", newGram);

}
```

Output:

Enter Production : S → iEtS|iEtSeS|a

S → iEtSX

X → eS|a

(11) Implement a C program to perform symbol table operations

Aim: To implement a C program to perform symbol table operations.

Algorithm:

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

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int cnt=0;
struct symtab
{
   char label[20];
   int addr;
}
sy[50];
void insert();
int search(char *);
void display();
void modify();
int main()
{
   int ch,val;
```

```
char lab[10];
do
{
  printf("\n1.insert \n2.display \n3 search \n4.modify \n5.exit\n");
  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("label is found");
        else
          printf("label is not found");
        break;
    Case 4:
        modify()
        break;
    Case 5
        exit(0);
}       break;
```

Output

1. insert
2. displa
3. searc
4. modify
5. exit

1.
enter
enter t

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

Result :

symbol

```
void display()
{
  int i;
  for (i=0; i<cnt; i++)
  printf("%s\t%d\n", sy[i].label, sy[i].addr);
}
```

Output:

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

1.
enter the label a
enter the address 100

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

Result: A C program is implemented to perform symbol table operations.

(12) Write a C program to construct recursive descent parsing

Aim: To construct C program recursive descent parsing.

Algorithm:

* Start the code by importing the packages
* Declare the variables
* Assign the variables.
* Save and Run the code.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char input[100];
int i,l;
void main()
{
 printf("\n Recursive descent parsing for the following grammar");
 printf("\n E->TE'\nE'->TE'|@\n F->(E)");
 printf("\n Enter the string to be checked:\n");
  gets(input);
  if(E())
  {
    if(input[i+1]=='\0')
    printf("\n String is accepted");
    else
  }    printf("\n String is not accepted");
```

```
else
    printf("\n String not accepted");
    getch();
E()
{
    if(T())
    {
        if(EP())
            return(1);
        else
            return(0);
    else
        return(0);
    }
EP()
{
    if(input[i]=='+')
    {
        i++;
        if(T())
        {
            if(EP())
                return(1);
            else
                return(0)
            }
        else
            return(1);
        }
```

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

Output :-

Recursive descent parsing for the following grammar.

$E \rightarrow TE'$

$E' \rightarrow +TE'/@$

$T \rightarrow FT'$

$T' \rightarrow *FT'/@$

$F \rightarrow (E)/ID$

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.

Result :- A C program to construct recursive descent parsing is developed.

(13) Write a C program to implement either Top Down parsing technique or Bottom Up parsing technique to check whether the give i/p string is grammar

Aim:- To write C program to implement either Top Down parsing technique or Bottom Up parsing technique to check whether the given i/p string is grammar.

Algorithm:-

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

Program:-

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

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

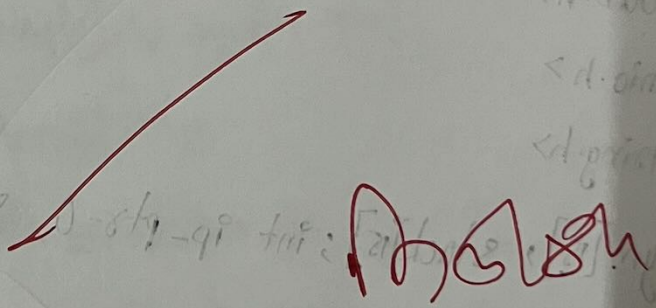
Output:

The grammar is: S→aS, S→Sb, S→ab

Enter the string to be checked:

abb

String accepted



Result: Developed a C program to implement either Top Dow
parsing technique or Bottom Up parsing technique to che
whether the given i/p string is grammar.

(14) Implement the concept of Shift reduce parsing in C program

Aim:- To implement the concept of Shift reduce parsing in C programming.

Algorithm :-

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

Program :-

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.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\t\t SHIFT REDUCE PARSER \n");
   printf("\n GRAMMAR\n");
```

```
printf("\n E→E+E \n E→E/E");
printf("\nE→E*E\n E→a/b");
printf("\n Enter the i/p symbol: \t");
gets(ip_sym);
printf("\n\t Stack implementation table");
printf("\n stack\t\t input symbol\t\t action");
printf("\n \t\t $\t\t \n");
printf("\n $\t\t %s$\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 rejected", stack, ip_sym);
exit(0);
}
return;
}
```

Output:

SHIFT RDUCE 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$	--
$a	+b$	shift a
$E+	b$	shift +
$E+b	$	shift b
$E+E	$	E → b
$E	$	E → E+E
		(Accept)

Result:

Implemented Shift Reduce Parser in C program.

(15) Write a C program to implement the operator precedence

Aim:- To write a C program to implement the operator precedence parsing.

Algorithm:-

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

Program:-

```
#include <string.h>
#include <stdio.h>
char *input;
int i=0;
char lasthandle[6], stack[50]
int top=0,1;
char prec[9][9] = {

int getindex(char c)
{
  switch(c)
  {
    case '+': return 0;
```

```
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++)
else
printf("\n Not Accepted");
}
```

Output :

Enter the

i*(i+i)*i

STACK

$i

$E

$E*

$i*

$E*

$E

$E

Output :-

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
$E*(E	+i)*i$	shift
$E*(E+	i)*i$	shift
$E*(E+i	)*i$	Reduced: E→E+E
$E*(E	)*i$	shift
$E*(E)	*i$	Reduced: E→(E)
$E*	i$	shift
$E*i	$	Reduced: E→i
$E*E	$	Reduced: E→E*E
$E	$	Accept

Result :- A C program to implement the operator procedure is developed.

Write a LEX specification file to take input C program from a.c. file 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", &num1, &num2);
  sum = num1 + num2;
  printf("%d + %d = %d", num1, num2, sum);
  return 0;
}
```

Program:

```
%{
int nchar, nword, nline;
%}
%%
\n {nline++; nchar++;}
[^\t\n]+ {nword++, nchar+=yyleng;}
. {nchar++;}
%%
int yywrap(void) {
 return 1;
}
```

Output:-

Here count.

```
gcc lex.yy
a.exe sample
```

No. of char

No. of word

No. of lines

Result: A

characters

```
}
int main (int argc, char* argv [1]) {
  yyin = fopen (argv [1], "r");
  yylex ();
  printf ("No. of characters = %d \n", nchar);
  printf ("No. of words = %d \n", nword);
  printf ("No. of lines = %d \n", nline);
  fclose (yyin);
}
```

Output:-

```
flex count_line.l
gcc lex.yy.c
a.exe sample.c
No. of characters = 233
No. of words = 33
No. of lines = 10
```

Result:- A Lex Program for counting number of characters is developed.

(17)

Write a LEX program to print all the constants in a given C source program file.

Input source Program:

```
#define P 314
#include <stdio.h>
#include <conio.h>
void main()
{
  int a,b,c = 30;
  printf("hello");
}
```

Program:

```
digit [0-9]
%{
int cons=0;
%}
%%
{digit}+ {cons++; printf("%s is a constant\n", yytext);}
.|\n {}
%%
int yywrap(void) {
return 1; }
int main(void)
{
  FILE *f;
  char file[10];
  printf("Enter file Name: ");
```

scanf("%s",
f=fopen(fi
yyin=f;
yylex();
printf("No.of
fclose(yyin);
}

Output:

flex countconst
gcc lex.yy.c
a.exe
Enter file Nam
314 is a constan
30 is a consta
Number of

Result:-

from the

```
scanf("%s", file);
f = fopen(file, "r");
yyin = f;
yylex();
printf("No. of Constants : %d\n", cons);
fclose(yyin);
}
```

Output:

```
flex countconstants.l
gcc lex.yy.c
a.exe
Enter file Name: sample.c
314 is a constant
30 is a constant
Number of Constants: 2
```

Result :- A LEX program to print the all constants from the source file.

(18)

Write a LEX program to count the number of macros defined and header files included in the C program.

Input Source Program:-

```
#define PI 3.14
#include<stdio.h>
#include<conio.h>
void main()
{
  int a,b,c=30;
  printf("hellow");
}
```

Program:-

```
%{
int nmacro, nheader;
%}
%%
^# define {nmacro++;}
^# include {nheader++;}
.|\n{}
%%
int yywrap(void){
return 1;
}
int main(int argc, char *argv[]){
  yyin = fopen(argv[1],'r');
```

yylex();
printf("No.
printf("No.
fclose(yyin)
}

Output:-

flex count_ma
gcc lex.yy.
a.exe
No. of macro
No. of header

Result:-

A LEX p
and heade

```
    yylex();
    printf("No. of macros defined = %d\n", nmacro);
    printf("No. of header files included = %d\n", nheader);
    fclose(yyin);
}
```

Output:-

```
flex count_maro.l
gcc lex.yy.c
a.exe
No. of macros defined = 1
No. of header files included = 2
```

Result:-

A LEX program to count the no. of macros defined and header files included.

(A)

Write a LEX program to print all HTML tags in the input file.

Input Source Programs:

```
<html>
<body>
<h1>My First Heading</h1>
<p>My first paragraph</p>
</body>
</html>
```

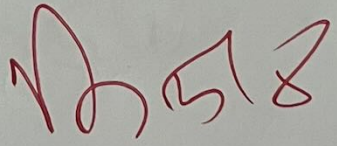
Program:

```
%{
int tags;
%}
%%
"<"[^>]*> {tags++;printf("%s\n", yytext);}
.|\n {}
%%
int yywrap(void){
return 1;}
int main(void)
{
  FILE *f;
char file[10];
printf("Enter File Name: ");
scanf("%s", file);
f= fopen(file, "r");
```

```
    yyin = f;
    yylex();
    printf("\n No. of html tags: %d", tags);
    fclose(yyin);
}
```

Output:

```
flex html.l
gcc lex.yy.c
a.exe
Enter file Name: sample.html
<html>
<body>
<h1>
</h1>
<p>
</body>
</html>
No. of html tags: 8
```



Result:

A LEX program to print all HTML tags in a input file is developed.

(20)

Write a LEX program which reads adds line numbers to the given C program file and display the same in the standard output.

Input Source Program:

```
#define PI 3.14
#include <stdio.h>
#include<conio.h>
void main()
{
  int a,b,c=30;
  printf("hello");
}
```

Program:

```
%{
int yylineno;
%}
%%
int yywrap(void){
return 1;
}
int main(int argc, char *argv[]){
yyin = fopen(argv[1], "r");
yylex();
fclose(yyin);
}
```

Output:

flex addlines.nos.l

gcc lex.yy.c

a.exe

#define PI 3.14

2. #include <stdio.h>

3. #include <conio.h>

4. void main()

{

int a,b,c=30;

printf("hello");

}

Result: A LEX program which adds line nos to the given C program file & display the same in the standard op.

Write a LEX program to identify the capital words from the given input.

Program:

```
%%
[A-Z]+[ \t\n] {printf("%s is a capital words\n",yytext);}
.;
int main()
{
   print("Enter String: \n");
   yylex();
}
int yywrap()
{
   return 1;
}
```

Output :-

```
CAPTIAL of INDIA is DELHI
CAPITAL is a capital word
INDIA is a capital word
DELHI is a capital word
```

Result :-

A LEX program to identify the capital words from i/p is developed.

29

Write a LEX program to check the email address is valid or not.

Program :-

```
%{
int flag=0;
%}
%%
[a-z],[0-9]+@[a-z]+".com"|".in" {flag=1;}
%%
int main()
{
  yylex();
  if(flag==1)
  printf("Accepted");
  else
  printf("Not Accepted");
}
int yywrap()
```

Output :-

sse123@gmail.com

Accepted

Result :-

A LEX program to check email address is valid or not.

Implement a LEX program to check whether the mobile number is valid or not.

Program:-

```
%%
[1-9][0-9]{9} {printf("\n Mobile Number Valid\n");}
+{printf("\n Mobile No. Invalid\n");}
%%
int main()
{
    printf("\n Enter Mobile Number: ");
    yylex();
    printf("\n");
    return 0;
}
```

Output:-

Enter Mobile No. :- 7858453489

Mobile No. Valid

Result:-

A LEX program to check whether mobile no. is valid or not.