

FACULTY OF ENGINEERING AND TECHNOLOGY BACHELOR OF TECHNOLOGY

Compiler Design

(CD) (203105351)

VI SEMESTER

Computer Science & Engineering Department





**CERTIFICATE**

*This is to certify that*

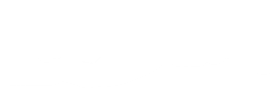
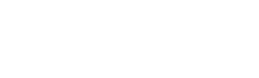
*Mr.*  **LUCKY PATHAN** *with Enrollment No.* **210303105790** has *successfully completed his laboratory experiments in the subject (with Code)* **Compiler Design (203105351)** *from the department of* **Computer Science and Engineering** *during the academic year* ***2022-2023.***



**Date of Submission …..…………… Staff In charge …..……………**

**Head of Department …..……………**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **S**  **r. N**  **o** | **Experiment Title** | **Page No** | | **Date of Start** | **Date of Completing** | **Sign** | **Marks**  **(Out of 10)** |
| **From** | **To** |
| 1 | Program to implement Lexical Analyzer. | 1 | 4 |  |  |  |  |
| 2 | Program to count digits, vowels and symbols in C. | 5 | 6 |  |  |  |  |
| 3 | Program to check validation of User Name and Password in C. | 7 | 8 |  |  |  |  |
| 4 | Program to implement Predictive Parsing LL (1) in C. | 9 | 14 |  |  |  |  |
| 5 | Program to implement Recursive Descent Parsing in C.. | 15 | 17 |  |  |  |  |
| 6 | Program to implement Operator Precedence Parsing in C. | 18 | 21 |  |  |  |  |
| 7 | Program to implement LALR Parsing in C. | 22 | 35 |  |  |  |  |
| 8 | To Study about Lexical Analyzer Generator (LEX) and Flex (Fast Lexical Analyzer) | 36 | 39 |  |  |  |  |
| 9 | Implement following programs using Lex.  a. Create a Lexer to take input from text file and count no of characters, no. of lines & no. of words.  b. Write a Lex program to count number of vowels and consonants in a given input string. | 40 | 42 |  |  |  |  |
| 10 | Implement following programs using Lex.  a. Write a Lex program to print out all numbers from the given file.  b. Write a Lex program to printout all HTML tags in file.  c. Write a Lex program which adds line numbers to the given file and display the same onto the standard output. | 43 | 45 |  |  |  |  |



# Practical 1

**Aim** : Program to implement Lexical Analyzer.

**CODE:-**

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

#include<string.h>

#include<stdlib.h>

void keyw(char \*p);

int i=0,id=0,kw=0,num=0,op=0,sp =0,ar=0,count=1,new\_lune=0;

char keys[32][10]={"auto","break","case","char","const","continue","default","do","dobule","else","enum","extern","float","for","goto","if","int","long","register","return","short","signed","sizeof","struct","switch","typedef","union","unsigned","void","volatiel","while"};

void main()

{

char ch,str[25],seps[20]="\t\n,;(){}[]#\"<>",oper[]="!%^&\*-+=~|.<>/?";

int j;

char fname[50];

FILE \*f1;

f1 = fopen("Laxcode.txt","r");

if (f1 == NULL)

{

printf("File not found");

exit(0);

}

while((ch=fgetc(f1))!=EOF)

{

for ( j = 0; j <=14; j++)

{

if(ch == oper[j])

{

printf("%c is an operator\n",ch);

op++;

count++;

str[i]= '\0';

keyw(str);

}

}

if (ch == '\n')

{

new\_lune++;

}

for ( j = 0; j <= 14; j++)

{

if(i == -1)

break;

if(ch == seps[j])

{

if(ch == '#')

{

while(ch != '>')

{

printf("%c",ch);

ch = fgetc(f1);

}

printf("%c is a header file\n",ch);

i = -1;

break;

}

if (ch =='"')

{

do

{

ch = fgetc(f1);

printf("%c",ch);

} while (ch != '"');

i = -1;

ar++;

count++;

break;

}

if(ch == ',' || ch == ';' || ch == '(' || ch == ')' || ch == '{' || ch == '}' || ch == '[' || ch == ']')

{

printf("%c is an Serpator",ch);

sp++;

count++;

}

str[i]='\0';

keyw(str);

}

}

if(i!=-1)

{

str[i]=ch;

i++;

}

else

i = 0;

}

printf("\n Keywords : %d \n Identifiers : %d \n Operators : %d \n Numbers : %d \n Seprator : %d \n Arfument : %d",kw,id,op,num,sp,ar);

printf("\n Total number of token : %d",count);

printf(" \n Number of lines : %d",new\_lune);

getch();

}

void keyw(char \*p)

{

int k, flag = 0;

for (k = 0; k <=31; k++)

{

if(strcmp(keys[k],p) == 0)

{

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

kw++;

count++;

flag + 1;

break;

}

}

if(flag == 0)

{

if(isdigit(p[0]))

{

printf("%s us a number\n",p);

num ++;

count++;

}

else

{

if(p[0]!='\0')

{

printf("%s is a identifier\n",p);

id++;

count++;

}

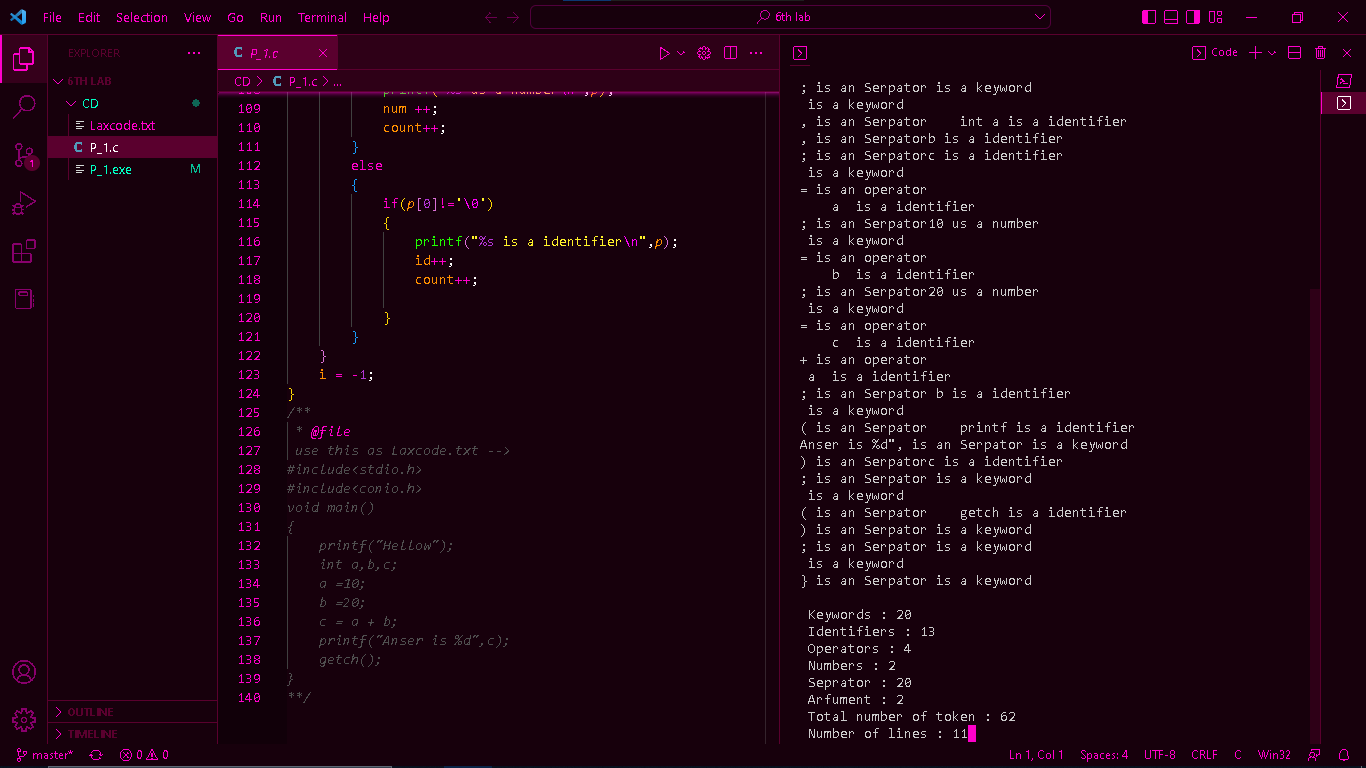
}

}

i = -1;

}

**OUTPUT:-**

****

# Practical 2

**Aim** : Program to count digits, vowels and symbols in C.

**CODE:-**

#include<stdio.h>

#include<string.h>

#include<conio.h>

#include<stdlib.h>

int main()

{

char str[100];

int i =0;

int vovels = 0 , consonant = 0 , digits = 0 , symbols = 0 , space = 0 ;

FILE \*fp;

char ch;

fp = fopen("P2.txt","r");

if(fp == NULL)

{

printf("File not opened.");

exit(1);

}

ch = fgetc(fp);

printf("your string is :\n");

while (!feof(fp))

{

str[i++]=ch;

ch = fgetc(fp);

}

str[i] = '\0';

printf(" %s",str);

fclose(fp);

for(i = 0; str[i]!=0;i++)

{

if(str[i] == 'a' || str[i] == 'A' || str[i] == 'e' ||str[i] == 'E' ||str[i] == 'i' ||str[i] == 'I' ||str[i] == 'o' ||str[i] == 'O' ||str[i] == 'u' || str[i] == 'U')

{

vovels++;

}

else if((str[i]>='a'&& str[i]<='z') || (str[i]>='Z'&& str[i]<='Z'))

{

consonant++;

}

else if ((str[i]>='0'&& str[i]<='9'))

{

digits++;

}

else if ((str[i]>=' '))

{

space++;

}

else

{

symbols++;

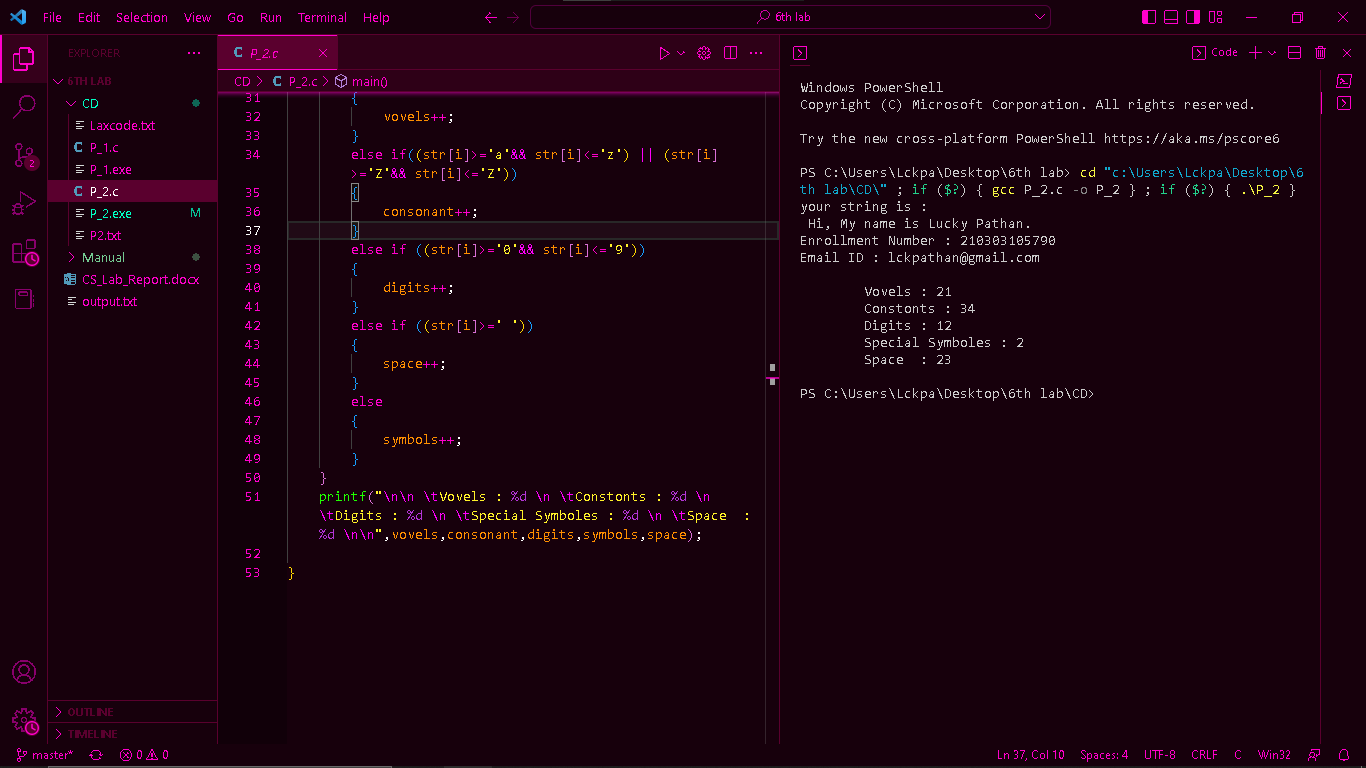
}

}

printf("\n\n \tVovels : %d \n \tConstonts : %d \n \tDigits : %d \n \tSpecial Symboles : %d \n \tSpace : %d \n\n",vovels,consonant,digits,symbols,space);

}

**OUTPUT:-**



# Practical 3

**Aim** : Program to check validation of User Name and Password in C.

**CODE:-**

#include<stdio.h>

#include<string.h>

#include<conio.h>

#include<stdlib.h>

int Cheak\_Password(char Password[])

{

int len = strlen(Password);

int Flag\_1 = 0 , Flag\_2 = 0 , Flag\_3 = 0 , Flag\_4 = 0;

for(int i = 0 ; i <= len ; i++)

{

if(Password[i]>= 65 && Password[i]<= 90) //Cheak upper case

{

Flag\_1 = 1;

}

if(Password[i]>= 97 && Password[i] <= 122) //Cheak Lower cass

{

Flag\_2 = 1;

}

if(Password[i] >= 0 && Password[i] <= 9) //Cheak 0 to 9

{

Flag\_3 = 1;

}

if (Password[i] == '!' || Password[i] == '@' || Password[i] == '#' || Password[i] == '%' || Password[i] == '\_')

{

Flag\_4 = 1;

}

}

if(Flag\_1 == 1 && Flag\_2 == 1 && Flag\_3 ==1 && Flag\_4 ==1)

{

printf("Sucess");

}

else

{

printf("Fail");

if(Flag\_1 != 1)

printf("\nUppercase Missing.");

if(Flag\_2 != 1)

printf("\nLowercase Missing.");

if(Flag\_3 != 1)

printf("\nNumber Missing.");

if(Flag\_4 != 1)

printf("\nSpecial Charater Missing.");

}

}

int main()

{

char user[50];

char passwd[15];

printf("Enter Username : ");

scanf("%s",&user);

printf("Enter Password : ");

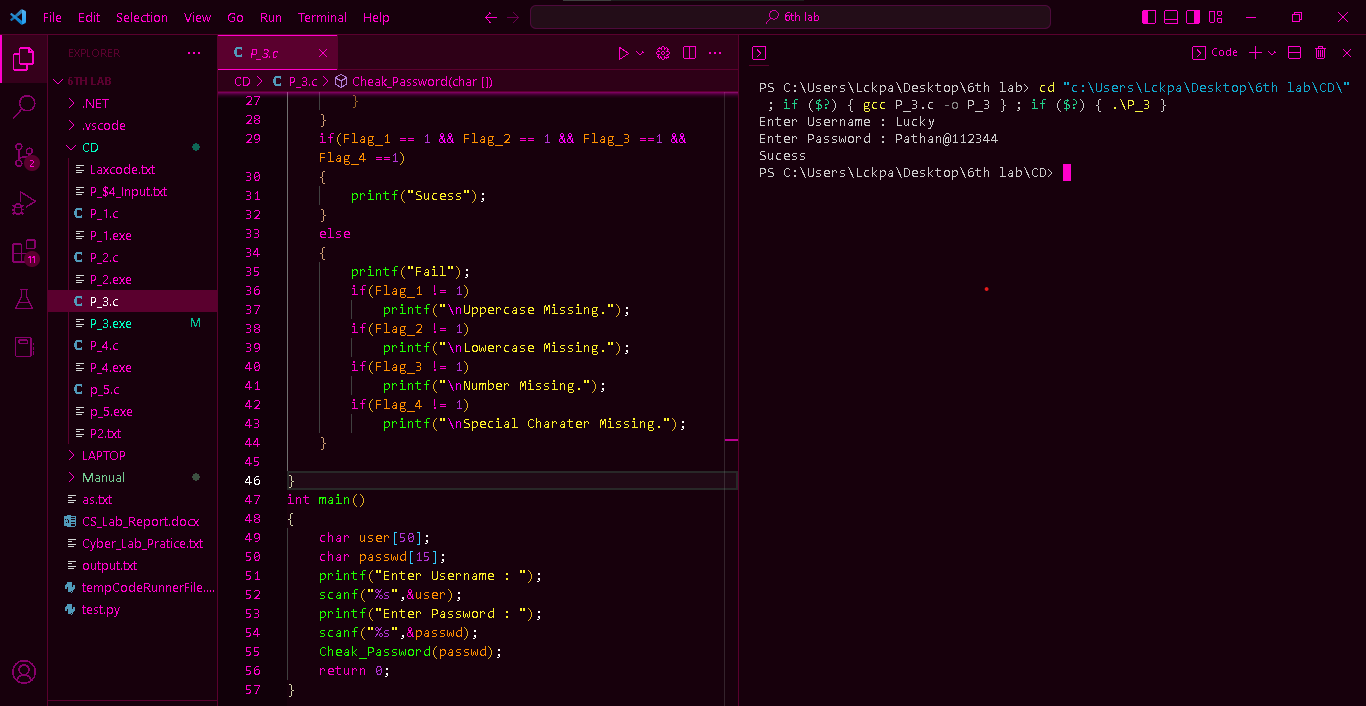
scanf("%s",&passwd);

Cheak\_Password(passwd);

return 0;

}

**OUTPUT:-**



# Practical 4

**Aim** : Program to implement Predictive Parsing LL (1) in C..

**CODE:-**

#include<stdio.h>

#include<string.h>

#define TSIZE 128

int table[100][TSIZE];

char terminal[TSIZE];

char nonterminal[26];

struct product {

char str[100];

int len;

}pro[20];

int no\_pro;

char first[26][TSIZE];

char follow[26][TSIZE];

char first\_rhs[100][TSIZE];

int isNT(char c) {

return c >= 'A' && c <= 'Z';

}

void readFromFile() {

FILE\* fptr;

fptr = fopen("P\_$4\_Input.txt", "r");

char buffer[255];

int i;

int j;

while (fgets(buffer, sizeof(buffer), fptr)) {

printf("%s", buffer);

j = 0;

nonterminal[buffer[0] - 'A'] = 1;

for (i = 0; i < strlen(buffer) - 1; ++i) {

if (buffer[i] == '|') {

++no\_pro;

pro[no\_pro - 1].str[j] = '\0';

pro[no\_pro - 1].len = j;

pro[no\_pro].str[0] = pro[no\_pro - 1].str[0];

pro[no\_pro].str[1] = pro[no\_pro - 1].str[1];

pro[no\_pro].str[2] = pro[no\_pro - 1].str[2];

j = 3;

}

else {

pro[no\_pro].str[j] = buffer[i];

++j;

if (!isNT(buffer[i]) && buffer[i] != '-' && buffer[i] != '>') {

terminal[buffer[i]] = 1;

}

}

}

pro[no\_pro].len = j;

++no\_pro;

}

}

void add\_FIRST\_A\_to\_FOLLOW\_B(char A, char B) {

int i;

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

if (i != '^')

follow[B - 'A'][i] = follow[B - 'A'][i] || first[A - 'A'][i];

}

}

void add\_FOLLOW\_A\_to\_FOLLOW\_B(char A, char B) {

int i;

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

if (i != '^')

follow[B - 'A'][i] = follow[B - 'A'][i] || follow[A - 'A'][i];

}

}

void FOLLOW() {

int t = 0;

int i, j, k, x;

while (t++ < no\_pro) {

for (k = 0; k < 26; ++k) {

if (!nonterminal[k]) continue;

char nt = k + 'A';

for (i = 0; i < no\_pro; ++i) {

for (j = 3; j < pro[i].len; ++j) {

if (nt == pro[i].str[j]) {

for (x = j + 1; x < pro[i].len; ++x) {

char sc = pro[i].str[x];

if (isNT(sc)) {

add\_FIRST\_A\_to\_FOLLOW\_B(sc, nt);

if (first[sc - 'A']['^'])

continue;

}

else {

follow[nt - 'A'][sc] = 1;

}

break;

}

if (x == pro[i].len)

add\_FOLLOW\_A\_to\_FOLLOW\_B(pro[i].str[0], nt);

}

}

}

}

}

}

void add\_FIRST\_A\_to\_FIRST\_B(char A, char B) {

int i;

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

if (i != '^') {

first[B - 'A'][i] = first[A - 'A'][i] || first[B - 'A'][i];

}

}

}

void FIRST() {

int i, j;

int t = 0;

while (t < no\_pro) {

for (i = 0; i < no\_pro; ++i) {

for (j = 3; j < pro[i].len; ++j) {

char sc = pro[i].str[j];

if (isNT(sc)) {

add\_FIRST\_A\_to\_FIRST\_B(sc, pro[i].str[0]);

if (first[sc - 'A']['^'])

continue;

}

else {

first[pro[i].str[0] - 'A'][sc] = 1;

}

break;

}

if (j == pro[i].len)

first[pro[i].str[0] - 'A']['^'] = 1;

}

++t;

}

}

void add\_FIRST\_A\_to\_FIRST\_RHS\_\_B(char A, int B) {

int i;

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

if (i != '^')

first\_rhs[B][i] = first[A - 'A'][i] || first\_rhs[B][i];

}

}

void FIRST\_RHS() {

int i, j;

int t = 0;

while (t < no\_pro) {

for (i = 0; i < no\_pro; ++i) {

for (j = 3; j < pro[i].len; ++j) {

char sc = pro[i].str[j];

if (isNT(sc)) {

add\_FIRST\_A\_to\_FIRST\_RHS\_\_B(sc, i);

if (first[sc - 'A']['^'])

continue;

}

else {

first\_rhs[i][sc] = 1;

}

break;

}

if (j == pro[i].len)

first\_rhs[i]['^'] = 1;

}

++t;

}

}

int main() {

readFromFile();

follow[pro[0].str[0] - 'A']['$'] = 1;

FIRST();

FOLLOW();

FIRST\_RHS();

int i, j, k;

printf("\n");

for (i = 0; i < no\_pro; ++i) {

if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {

char c = pro[i].str[0];

printf("FIRST OF %c: ", c);

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

if (first[c - 'A'][j]) {

printf("%c ", j);

}

}

printf("\n");

}

}

printf("\n");

for (i = 0; i < no\_pro; ++i) {

if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {

char c = pro[i].str[0];

printf("FOLLOW OF %c: ", c);

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

if (follow[c - 'A'][j]) {

printf("%c ", j);

}

}

printf("\n");

}

}

printf("\n");

for (i = 0; i < no\_pro; ++i) {

printf("FIRST OF %s: ", pro[i].str);

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

if (first\_rhs[i][j]) {

printf("%c ", j);

}

}

printf("\n");

}

terminal['$'] = 1;

terminal['^'] = 0;

printf("\n");

printf("\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LL(1) PARSING TABLE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

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

printf("%-10s", "");

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

if (terminal[i]) printf("%-10c", i);

}

printf("\n");

int p = 0;

for (i = 0; i < no\_pro; ++i) {

if (i != 0 && (pro[i].str[0] != pro[i - 1].str[0]))

p = p + 1;

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

if (first\_rhs[i][j] && j != '^') {

table[p][j] = i + 1;

}

else if (first\_rhs[i]['^']) {

for (k = 0; k < TSIZE; ++k) {

if (follow[pro[i].str[0] - 'A'][k]) {

table[p][k] = i + 1;

}

}

}

}

}

k = 0;

for (i = 0; i < no\_pro; ++i) {

if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {

printf("%-10c", pro[i].str[0]);

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

if (table[k][j]) {

printf("%-10s", pro[table[k][j] - 1].str);

}

else if (terminal[j]) {

printf("%-10s", "");

}

}

++k;

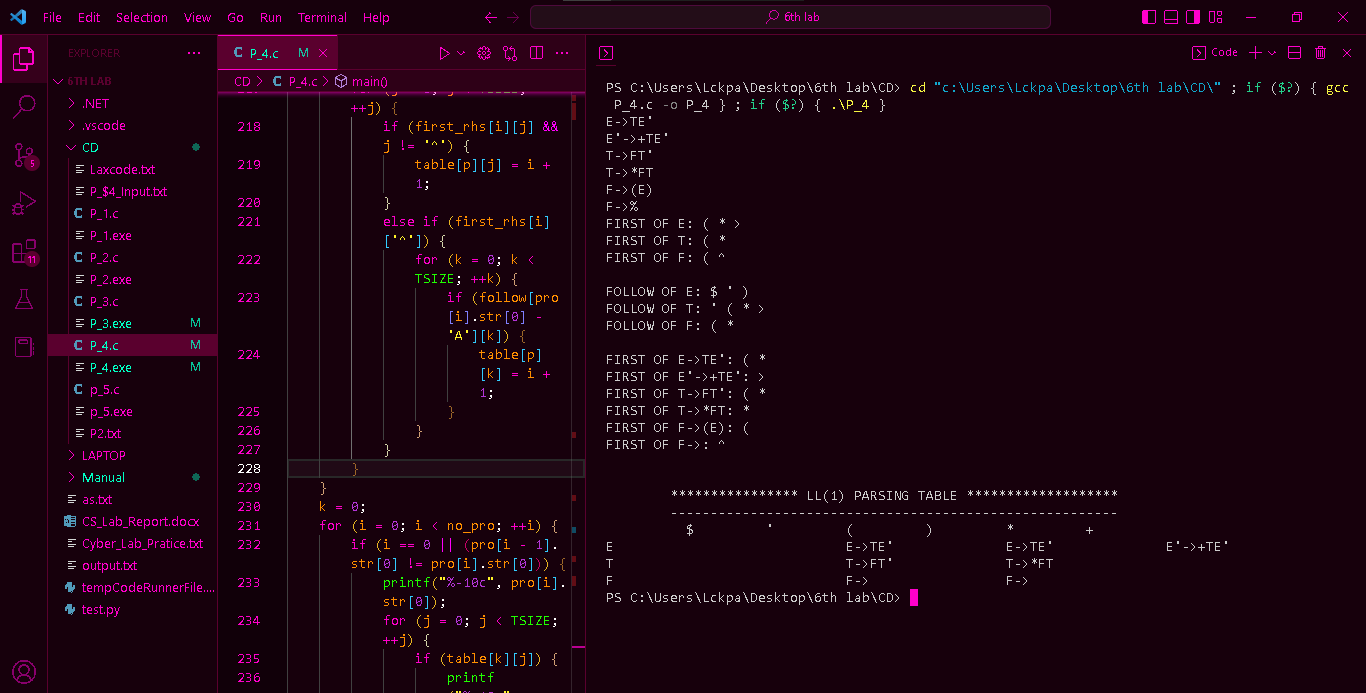
printf("\n");

}

}

}

**OUTPUT:-**



# Practical 5

**Aim** : Program to implement Recursive Descent Parsing in C.

**CODE:-**

#include<stdio.h>

#include<string.h>

#include<ctype.h>

#include<conio.h>

char input[10];

int i, erroe;

void T();

void Tprime();

void E();

void Eprime();

void F();

void main()

{

printf("Enter the algo expression :");

scanf("%s",input);

E();

if (strlen(input) == i && erroe == 0)

{

printf("\n---------------Accpted---------------");

}

else

{

printf("\n---------------Rejected---------------");

}

getch();

}

void E(){

T();

Eprime();

}

void Eprime()

{

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

{

i++;

T();

Eprime();

}

}

void T()

{

F();

Tprime();

}

void Tprime()

{

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

{

i++;

F();

Tprime();

}

}

void F()

{

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

{

i++;

}

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

{

i++;

E();

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

{

i++;

}else

{

erroe = 1;

}

}

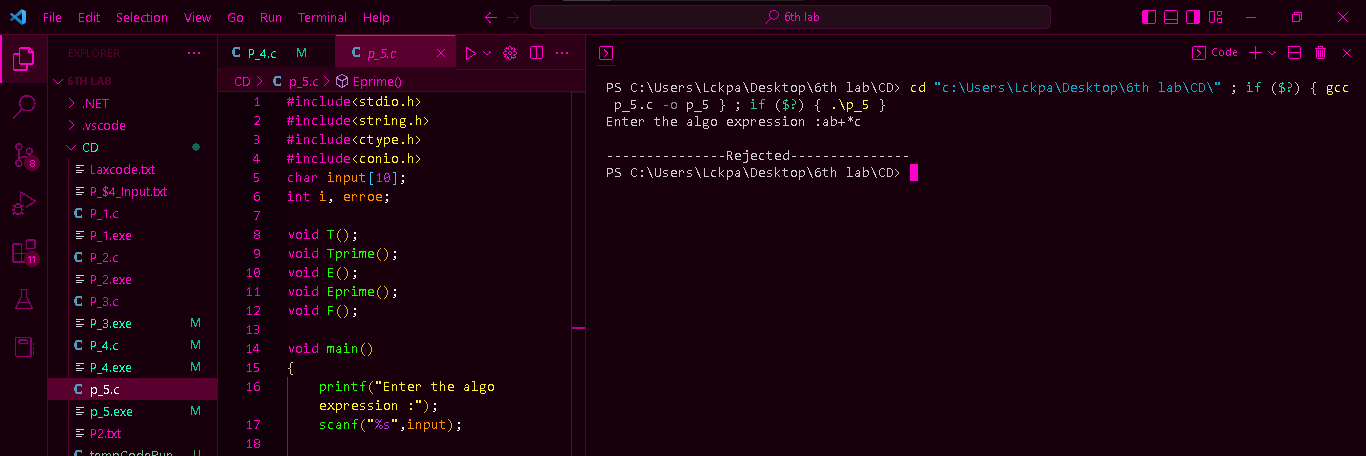
else{

erroe = 1;

}

}

**OUTPUT:-**



# Practical 6

**Aim** : Program to implement Operator Precedence Parsing in C.

**CODE:-**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

char \*input;

int i=0;

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

//(E) becomes )E( when pushed to stack

int top=0,l;

char prec[9][9]={

/\*input\*/

/\*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++)//selecting handles

{

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;//successful reduction

}

}

}

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)

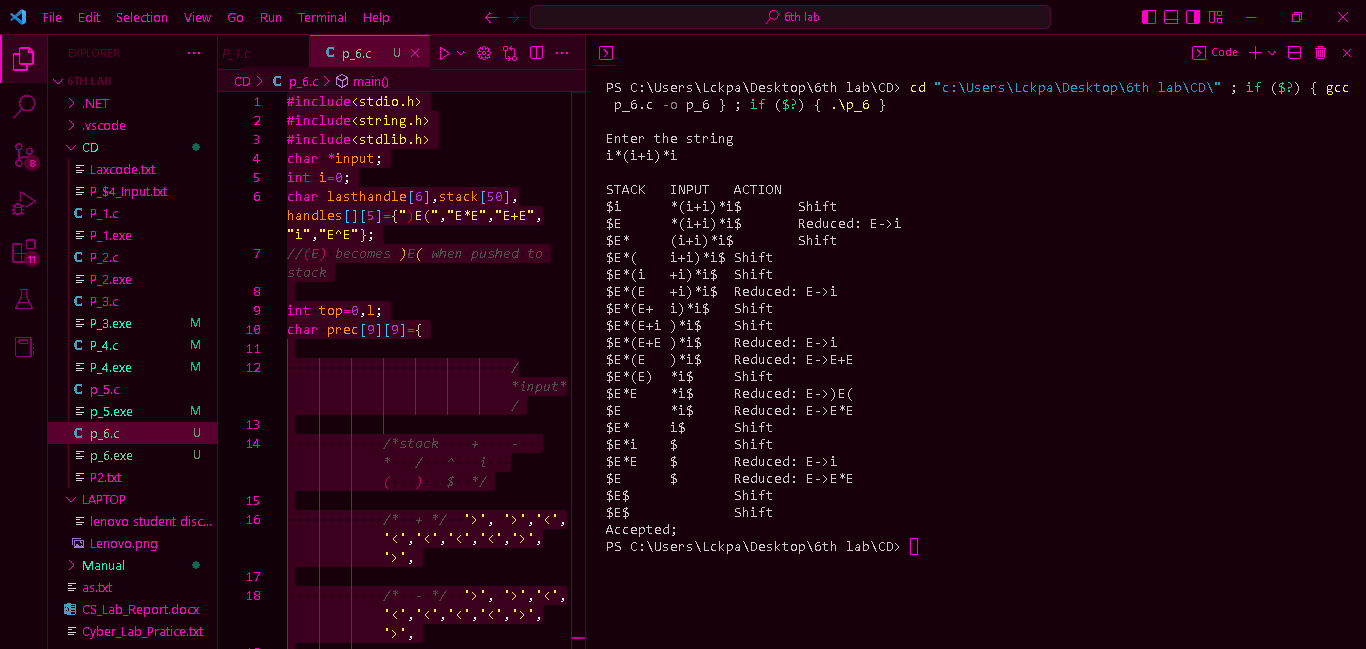
printf("\nAccepted;");

else

printf("\nNot Accepted;");

}

**OUTPUT:-**



# Practical 7

**Aim** : Program to implement LALR Parsing in C.

**CODE:-**

#include<stdio.h>

#include<string.h>

#include<conio.h>

#include<stdlib.h>

void push(char \* , int \* , char);

char stacktop(char \* );

void isproduct(char, char);

int ister(char);

int isnter(char);

int isstate(char);

void error();

void isreduce(char, char);

char pop(char \* , int \* );

void printt(char \* , int \* , char[], int);

void rep(char[], int);

struct action {

char row[6][5];

};

const struct action A[12] = {

{

"sf",

"emp",

"emp",

"se",

"emp",

"emp"

},

{

"emp",

"sg",

"emp",

"emp",

"emp",

"acc"

},

{

"emp",

"rc",

"sh",

"emp",

"rc",

"rc"

},

{

"emp",

"re",

"re",

"emp",

"re",

"re"

},

{

"sf",

"emp",

"emp",

"se",

"emp",

"emp"

},

{

"emp",

"rg",

"rg",

"emp",

"rg",

"rg"

},

{

"sf",

"emp",

"emp",

"se",

"emp",

"emp"

},

{

"sf",

"emp",

"emp",

"se",

"emp",

"emp"

},

{

"emp",

"sg",

"emp",

"emp",

"sl",

"emp"

},

{

"emp",

"rb",

"sh",

"emp",

"rb",

"rb"

},

{

"emp",

"rb",

"rd",

"emp",

"rd",

"rd"

},

{

"emp",

"rf",

"rf",

"emp",

"rf",

"rf"

}

};

struct gotol

{

char r[3][4];

};

const struct gotol G[12] = {

{

"b",

"c",

"d"

},

{

"emp",

"emp",

"emp"

},

{

"emp",

"emp",

"emp"

},

{

"emp",

"emp",

"emp"

},

{

"i",

"c",

"d"

},

{

"emp",

"emp",

"emp"

},

{

"emp",

"j",

"d"

},

{

"emp",

"emp",

"k"

},

{

"emp",

"emp",

"emp"

},

{

"emp",

"emp",

"emp"

},

};

char ter[6] = {

'i',

'+',

'\*',

')',

'(',

'$'

};

char nter[3] = {

'E',

'T',

'F'

};

char states[12] = {

'a',

'b',

'c',

'd',

'e',

'f',

'g',

'h',

'm',

'j',

'k',

'l'

};

char stack[100];

int top = -1;

char temp[10];

struct grammar

{

char left;

char right[5];

};

const struct grammar rl[6] = {

{

'E',

"e+T"

},

{

'E',

"T"

},

{

'T',

"T\*F"

},

{

'T',

"F"

},

{

'F',

"(E)"

},

{

'F',

"i"

},

};

void main()

{

char inp[80], x, p, dl[80], y, bl = 'a';

int i = 0, j, k, l, n, m, c, len;

printf(" Enter the input :");

scanf("%s", inp);

len = strlen(inp);

inp[len] = '$';

inp[len + 1] = '\0';

push(stack, & top, bl);

printf("\n stack \t\t\t input");

printt(stack, & top, inp, i);

do

{

x = inp[i];

p = stacktop(stack);

isproduct(x, p);

if (strcmp(temp, "emp") == 0)

error();

if (strcmp(temp, "acc") == 0)

break;

else

{

if (temp[0] == 's')

{

push(stack, & top, inp[i]);

push(stack, & top, temp[1]);

i++;

} else

{

if (temp[0] == 'r')

{

j = isstate(temp[1]);

strcpy(temp, rl[j - 2].right);

dl[0] = rl[j - 2].left;

dl[1] = '\0';

n = strlen(temp);

for (k = 0; k < 2 \* n; k++)

pop(stack, & top);

for (m = 0; dl[m] != '\0'; m++)

push(stack, & top, dl[m]);

l = top;

y = stack[l - 1];

isreduce(y, dl[0]);

for (m = 0; temp[m] != '\0'; m++)

push(stack, & top, temp[m]);

}

}

}

printt(stack, & top, inp, i);

} while (inp[i] != '\0');

if (strcmp(temp, "acc") == 0)

printf(" \n accept the input ");

else

printf(" \n do not accept the input ");

getch();

}

void push(char \* s, int \* sp, char item)

{

if ( \* sp == 100)

printf(" stack is full ");

else

{

\* sp = \* sp + 1;

s[ \* sp] = item;

}

}

char stacktop(char \* s)

{

char i;

i = s[top];

return i;

}

void isproduct(char x, char p)

{

int k, l;

k = ister(x);

l = isstate(p);

strcpy(temp, A[l - 1].row[k - 1]);

}

int ister(char x)

{

int i;

for (i = 0; i < 6; i++)

if (x == ter[i])

return i + 1;

return 0;

}

int isnter(char x)

{

int i;

for (i = 0; i < 3; i++)

if (x == nter[i])

return i + 1;

return 0;

}

int isstate(char p)

{

int i;

for (i = 0; i < 12; i++)

if (p == states[i])

return i + 1;

return 0;

}

void error()

{

printf(" error in the input ");

exit(0);

}

void isreduce(char x, char p)

{

int k, l;

k = isstate(x);

l = isnter(p);

strcpy(temp, G[k - 1].r[l - 1]);

}

char pop(char \* s, int \* sp)

{

char item;

if ( \* sp == -1)

printf(" stack is empty ");

else

{

item = s[ \* sp];

\* sp = \* sp - 1;

}

return item;

}

void printt(char \* t, int \* p, char inp[], int i)

{

int r;

printf("\n");

for (r = 0; r <= \* p; r++)

rep(t, r);

printf("\t\t\t");

for (r = i; inp[r] != '\0'; r++)

printf("%c", inp[r]);

}

void rep(char t[], int r)

{

char c;

c = t[r];

switch (c)

{

case 'a':

printf("0");

break;

case 'b':

printf("1");

break;

case 'c':

printf("2");

break;

case 'd':

printf("3");

break;

case 'e':

printf("4");

break;

case 'f':

printf("5");

break;

case 'g':

printf("6");

break;

case 'h':

printf("7");

break;

case 'm':

printf("8");

break;

case 'j':

printf("9");

break;

case 'k':

printf("10");

break;

case 'l':

printf("11");

break;

default:

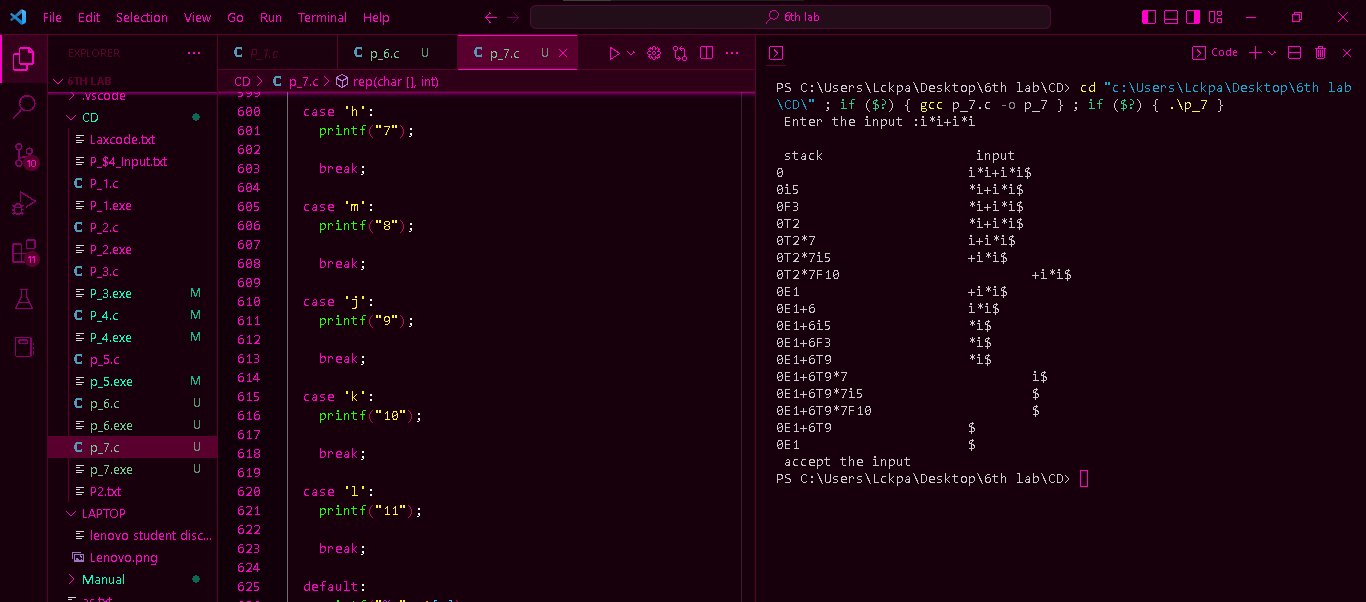
printf("%c", t[r]);

break;

}

}

**OUTPUT:-**



# Practical 8

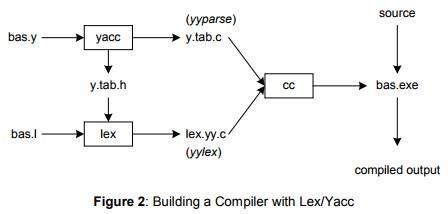
**Aim** : To Study about Lexical Analyzer Generator (LEX) and Flex (Fast Lexical Analyzer).

**CODE:-**

Lex - A Lexical Analyzer Generator:-

Lex is a program generator designed for lexical processing of character input streams. It accepts a high-level, problem oriented specification for character string matching, and produces a program in a general purpose language which recognizes regular expressions. The regular expressions are specified by the user in the source specifications given to Lex. The Lex written code recognizes these expressions in an input stream and partitions the input stream into strings matching the expressions.

The grammar in the above diagram is a text file you create with a text edtior. Yacc will read your grammar and generate C code for a syntax analyzer or parser. The syntax analyzer uses grammar rules that allow it to analyze tokens from the lexical analyzer and create a syntax tree. The syntax tree imposes a hierarchical structure the tokens. For example, operator precedence and associativity are apparent in the syntax tree. The next step, code generation, does a depth-first Lexical Analyzer Syntax Analyzer a = b + c \* d id1 = id2 + id3 \* id4 = + \* id1 source code tokens syntax tree id2 id3 id4 load id3 mul id4 add id2 store id1 Code Generator generated code Lex Yacc patterns grammar 5 walk of the syntax tree to generate code. Some compilers produce machine code, while others, as shown above, output assembly language.



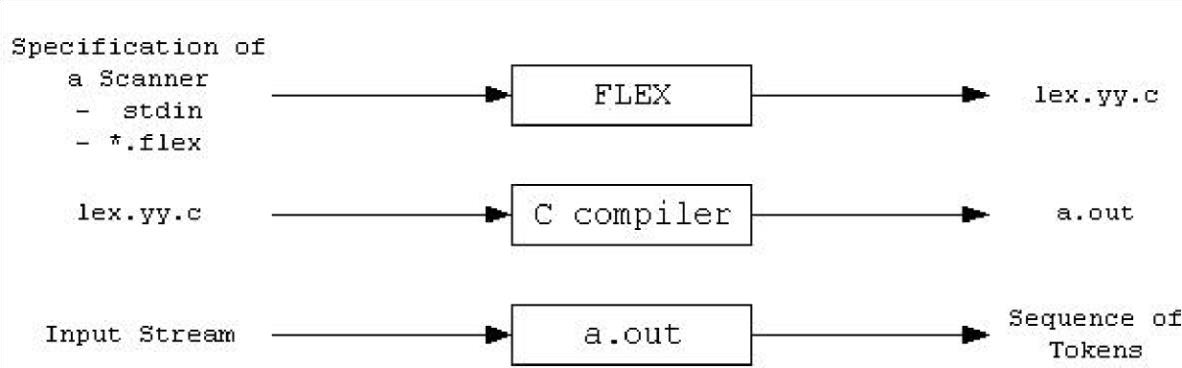
What is Flex?

* Flex is a powerful, open source application framework which allows to build traditional

applications for browser, mobile and desktop using the same programming model, tool, and codebase.

* Flex provides FLEX SDK consisting of the Flex class library (ActionScript classes), the Flex compilers, the debugger, the MXML and ActionScript programming languages, and other utilities to build expressive and interactive rich internet applications (RIA)
* Flex takes care of the user interface (UI) or the client-side functionality of a web application. Server-side functionality dependent on server-side components written in a traditional scripting language

How to use FLEX?

FLEX (Fast LEXical analyzer generator) is a tool for generating scanners. In stead of writing a scanner from scratch, you only need to identify the vocabulary of a certain language (e.g. Simple), write a specification of patterns using regular expressions (e.g. DIGIT [0-9]), and FLEX will construct a scanner for you. FLEX is generally used in the manner depicted here:

First, FLEX reads a specification of a scanner either from an input file \*.lex, or from standard input, and it generates as output a C source file lex.yy.c. Then, lex.yy.c is compiled and linked with the "-lfl" library to produce an executable a.out. Finally, a.out analyzes its input stream and transforms it into a sequence of tokens.

\*.lex is in the form of pairs of regular expressions and C code. (sample1.lex, sample2.lex) lex.yy.c defines a routine yylex() that uses the specification to recognize tokens. a.out is actually the scanner!

How to Compile & Run LEX / YACC Programs on Windows ?

If you are installing Ubuntu (or any Linux based OS) on your system either through Virtual Box or by making your system Multi-Bootable, just to execute your Lex & Yacc programs; then you might be wasting your HDD space & your valuable time. You can easily skip this annoying process and run your programs in Windows OS without any hassles.

Here's how you can do it:

Installing Softwares:

1. Download Flex 2.5.4a
2. Download Bison 2.4.1
3. Download DevC++
4. Install Flex at "C:\GnuWin32"
5. Install Bison at "C:\GnuWin32"
6. Install DevC++ at "C:\Dev-Cpp" 7. Open Environment Variables.
7. Add "C:\GnuWin32\bin;C:\Dev-Cpp\bin;" to path.

Compilation & Execution of your Program:

1. Open Command prompt and switch to your working directory where you have stored your lex file (".l") and yacc file (".y")
2. Let your lex and yacc files be "hello.l" and "hello.y". Now, follow the preceding steps to compile and run your program.
   1. For Compiling Lex file only:
      1. Flex hello.l
      2. gcc lex.yy.c
   2. For Compiling Lex & Yacc file both:
      1. flex hello.l
      2. bison -dy hello.y
      3. gcc lex.yy.c y.tab.c
   3. For Executing the Program
      1. a.exe

EXAMPLE:- HELLO.L FILE

%{

#include "y.tab.h"

int yyerror(char \*errormsg);

%}

%%

("hi"|"oi")"\n" { return HI; }

("tchau"|"bye")"\n" { return BYE; }

. { yyerror("Unknown char"); }

%%

int main(void)

{

yyparse(); return 0;

}

int yywrap(void)

{

return 0;

}

int yyerror(char \*errormsg)

{

fprintf(stderr, "%s\n", errormsg); exit(1);

}

HELLO.Y FILE

%{

#include <stdio.h> #include <stdlib.h> int yylex(void); int yyerror(const char \*s);

%}

%token HI BYE

%%

program: hi bye

;

hi:

HI { printf("Hello World\n"); }

; bye:

BYE { printf("Bye World\n"); exit(0); }

;

# Practical 9

**Aim** : Implement following programs using Lex.

1. Create a Lexer to take input from text file and count no of characters, no. of lines & no. of words.

**CODE:-**

%{

#include<stdio.h> int lines=0, words=0,s\_letters=0,c\_letters=0, num=0, spl\_char=0,total=0; %}

%%

\n { lines++; words++;}

[\t ' '] words++;

[A-Z] c\_letters++;

[a-z] s\_letters++;

[0-9] num++;

. spl\_char++;

%%

main(void)

{

yyin= fopen("practical9.txt","r"); yylex();

total=s\_letters+c\_letters+num+spl\_char;

printf(" This File contains ..."); printf("\n\t%d lines", lines); printf("\n\t%d words",words); printf("\n\t%d small letters", s\_letters); printf("\n\t%d capital letters",c\_letters); printf("\n\t%d digits", num);

printf("\n\t%d special characters",spl\_char);

printf("\n\tIn total %d characters.\n",total);

}

int yywrap()

{

return(1);

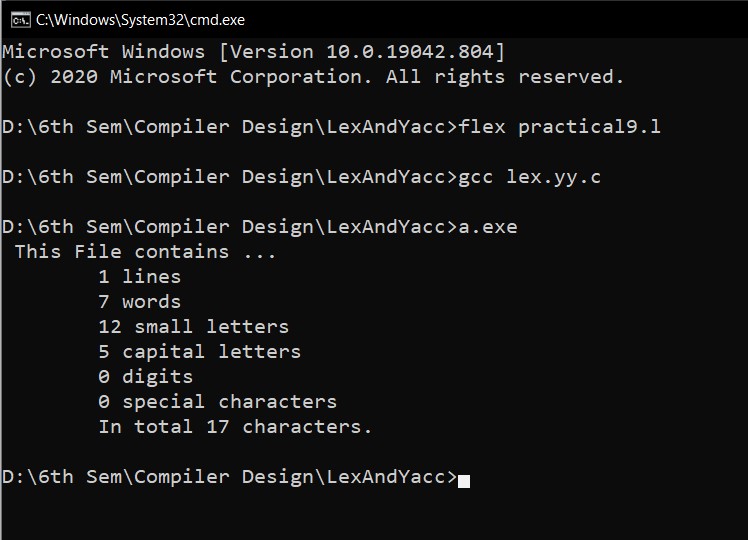
}

Practical9.txt FILE :

Hii

I'm Dijay From Cse

}input string.

**OUTPUT:-**

b. Write a Lex program to count number of vowels and consonants in a given

**CODE:-**

%{ int vow\_count=0;

int const\_count =0;

%}

%%

[aeiouAEIOU] {vow\_count++;}

[a-zA-Z] {const\_count++;}

%%

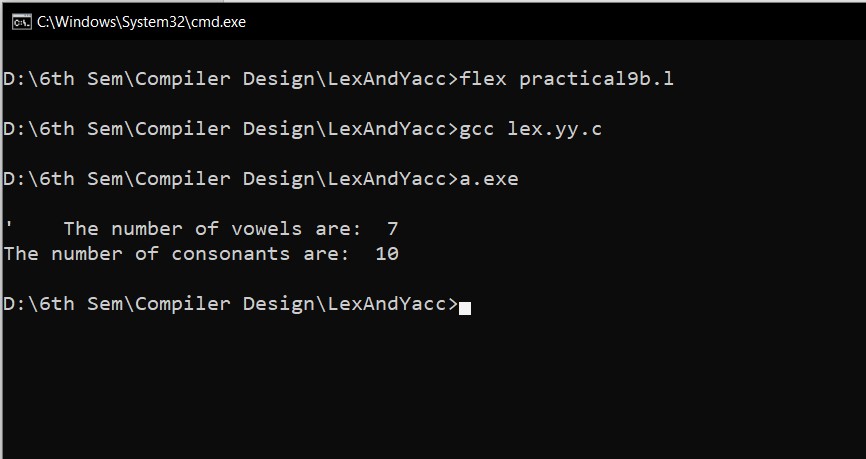
main() {

yyin= fopen("practical8.txt","r"); yylex();

printf("The number of vowels are: %d\n",vow\_count);

printf("The number of consonants are: %d\n",const\_count); return 0; } yywrap() { return 1;

}

**OUTPUT:-**

# Practical 10

**Aim** : Implement following programs using Lex.

1. Write a Lex program to print out all numbers from the given file.

**CODE:-**

%{

#include<stdio.h>

int num=0;

%}

%%

[0-9] num++; ECHO;

%%

main(void)

{

yyin= fopen("practical10.txt","r"); yylex();

printf("\n\t%d digits", num);

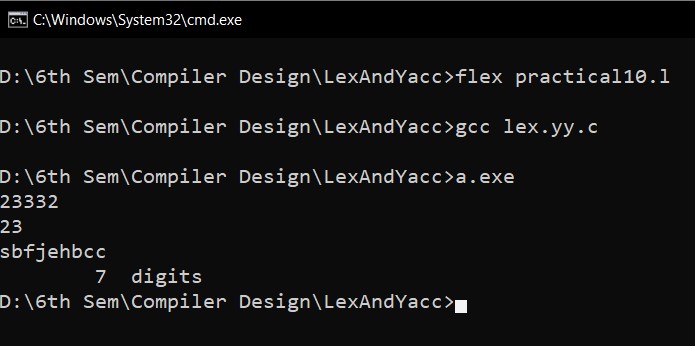
}

int yywrap()

{ return(1);

}

**OUTPUT:-**



1. Write a Lex program to printout all HTML tags in file the same onto the standard output.

**CODE:-**

%{

#include<stdio.h>

%}

%%

\<[^>]\*\> printf("%s\n",yytext);

.|\n;

%%

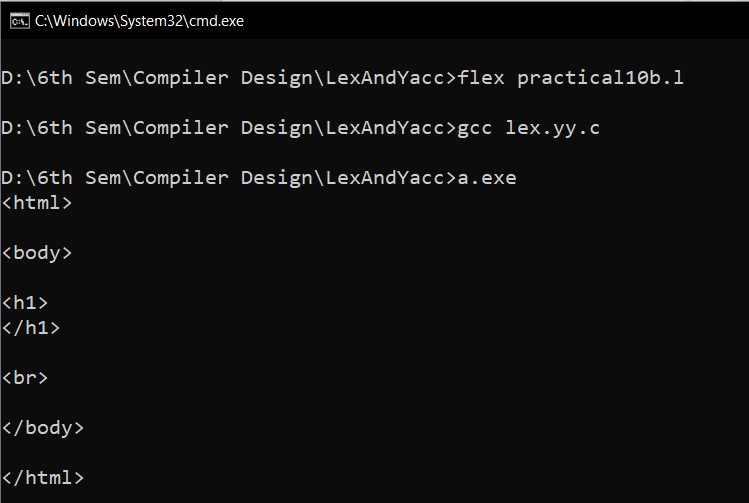
int yywrap()

{ return 1; }

int main() {

yyin=fopen("practical10b.txt","r"); yylex(); return 0; }

**OUTPUT:-**



1. Write a Lex program which adds line numbers to the given file and display

**CODE:-**

%{

#include<stdio.h>

int line\_number = 1;

%}

line .\*\n

%%

{line} { printf("%d %s", line\_number++, yytext); }

%%

int yywrap(){} int main()

{ extern FILE \*yyin;

yyin=fopen("add.txt","r"); if(yyin==NULL){ printf("File Not Found");

} yylex(); return 0;

}

**OUTPUT:-**

