# PRACTICAL 1

**a) Write a C Program to remove comment from another C Program and write in another C Program.**

#include<stdio.h> int main(){

FILE \*input = fopen("input.c","r"); FILE \*output = fopen("output.c","w"); char ch=fgetc(input); char ch1; while(ch!=EOF){

if(ch=='/'){

ch=fgetc(input); if(ch=='/'){ ch = fgetc(input); while(ch!='\n'){ ch=fgetc(input);

} } if(ch=='\*'){ ch=fgetc(input); while(ch!='/'){ ch=fgetc(input);

}

}

}else{

// printf("%c",ch); fputc(ch,output);

}

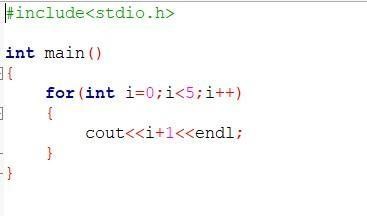
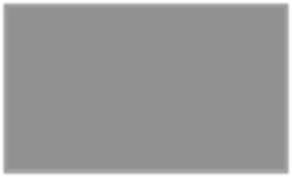
ch = fgetc(input);

} return 0; }

**INPUT:**



**OUTPUT:**



**b) Write a C program to recognize identifiers and numbers.**

#include <stdio.h> int main() { char s[10]; printf("enter identifier\n"); gets(s); if (s[0] == '\_' || (s[0] >= 'a' && s[0] <= 'z') || (s[0] >= 'A' && s[0] <= 'Z')) { for (int i = 1; i < (sizeof(s) / sizeof(s[0])); i++)

{ if (s[i] == '

')

{ printf("invalid\n"); return 0;

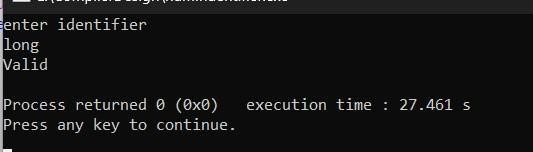
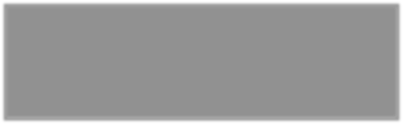
}

}printf("Valid\n");

} else printf("not

Valid\n"); return 0;

}



# PRACTICAL 2

**Write a C program to generate tokens for a C program.**

#include<stdio.h>

#include<string.h>

#include<ctype.h> int main() { int i=1,flag=0; char filename[50],a[50],ch[50],c; printf("\nEnter your Filename:"); scanf("%s",&filename);

FILE \*fi,\*fo,\*fop,\*fk; fi = fopen(filename,"r"); fo = fopen("inter.c","w");

fop = fopen("operator.c","r"); fk = fopen("key.c","r");

c = getc(fi); while(!feof(fi))

{ if(isalpha(c) || isdigit(c) || ( c == '[' || c ==

']') )

{

fputc(c,fo);

} else { if(c ==

'\n')

fprintf(fo,

"\t$\t"); else fprintf(fo,

"\t%c\t",c)

;

} c = getc(fi); } fclose(fi); fclose(fo); fi = fopen("inter.c","r"); printf("\nLEXICAL ANALYSIS");

fscanf(fi,"%s",a); printf("\n Line: %d\n",i++);

while(!feof(fi))

{ if(strcmp(a,"$")

== 0)

{

printf("\n Line: %d\n",i++); fscanf(fi,"%s",a);

}

fscanf(fop,"%s",&ch); while(!feof(fop))

{

if(strcmp(ch,a)==0)

{ printf("\t\t%s\t :

\t%s\t",a,ch); flag = 1;

}

fscanf(fop,"%s",ch);

}

rewind(fop); fscanf(fk,"%s",ch); while(!feof(fk))

{

if(strcmp(ch,a)==0)

{

printf("\t\t%s\t : Keyword\n",a); flag = 1;

}

fscanf(fk,"%s",ch);

} rewind(fk); if(flag == 0)

{ if(isdigit(a[0])) printf("\t\t%s\t : Constant\n",a); else printf("\t\t%s\t :

Identifier\n",a);

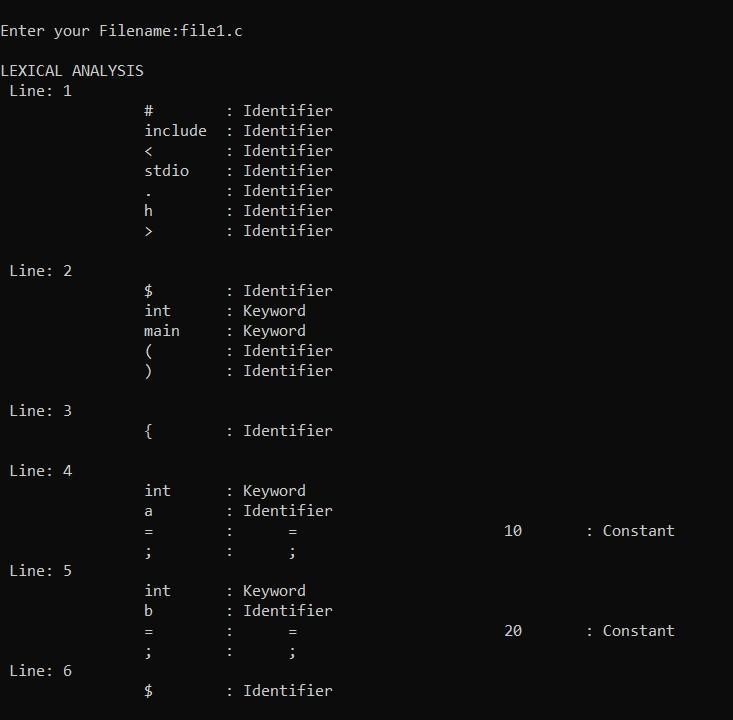
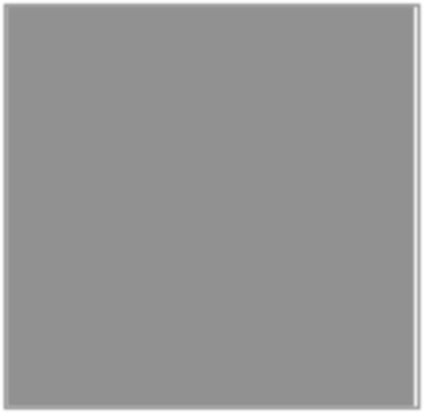
} flag = 0;

fscanf(fi,"%s",a);

}

return 0;

}

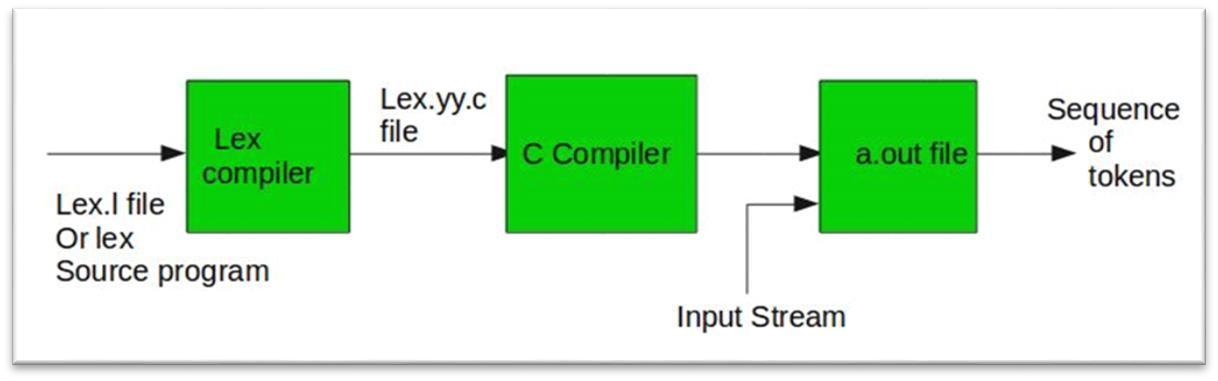


# PRACTICAL 3

**a). To Study about Lexical Analyzer Generator (LEX).**

FLEX (fast lexical analyzer generator) is a tool/computer program for generating lexical analyzers (scanners or lexers) written by Vern Paxson in C around 1987. It is used together with Berkeley Yacc parser generator or GNU Bison parser generator. Flex and Bison both are more flexible than Lex and Yacc and produces faster code. Bison produces parser from the input file provided by the user. The function yylex() is automatically generated by the flex when it is provided with a .l file and this yylex() function is expected by parser to call to retrieve tokens from current/this token stream.

Note: The function yylex() is the main flex function that runs the Rule Section and extension (.l) is the extension used to save the programs.



**Program Structure:**

**Definition Section:** The definition section contains the declaration of variables, regular definitions, manifest constants. In the definition section, text is enclosed in “%{ %}” brackets. Anything written in this brackets is copied directly to the file lex.yy.c

Syntax:

%{

// Definitions

%}

1. **Rules Section:** The rules section contains a series of rules in the form: *pattern action* and pattern must be unintended and action begin on the same line in {} brackets. The rule section is enclosed in “%% %%”.

Syntax:

%%

pattern action

%%

1. **User Code Section:** This section contains C statements and additional functions. We can also compile these functions separately and load with the lexical analyzer.

Basic Program Structure:

%{

// Definitions

%}

%%

Rules

%%

User code section

Examples: Table below shows some of the pattern matches.

|  |  |
| --- | --- |
| Pattern | It can match with |
| [0-9] | all the digits between 0 and 9 |
| [0+9] | either 0, + or 9 |

|  |  |
| --- | --- |
| Pattern | It can match with |
| [0, 9] | either 0, ‘, ‘ or 9 |
| [0 9] | either 0, ‘ ‘ or 9 |
| [-09] | either -, 0 or 9 |
| [-0-9] | either – or all digit between 0 and 9 |
| [0-9]+ | one or more digit between 0 and 9 |

|  |  |
| --- | --- |
| [^a] | all the other characters except a |
| [^A-Z] | all the other characters except the upper case letters |
| a{2, 4} | either aa, aaa or aaaa |
| a{2, } | two or more occurrences of a |
| a{4} | exactly 4 a’s i.e, aaaa |
| . | any character except newline |
| a\* | 0 or more occurrences of a |
| a+ | 1 or more occurrences of a |
| [a-z] | all lower case letters |
| Pattern | It can match with |
| [a-zA-Z] | any alphabetic letter |
| w(x | y)z | wxz or wyz |

**Steps to install Flex:**

Step 1: Download CodeBlocks-minGW setup from here and install it.

Step 2: Download Flex setup program from here and install it.

Step 3: After successfully installation go to CodeBlocks -> minGW -> bin and copy the address of the bin. Then go to environment variables->system variables->path->edit->new and paste the bin address here.

Step 4: Then go to C ->Program Files -> GnuWin32-> bin and again copy the address of the bin. Then go to environment variables->system variables->path->edit->new and paste it here.

Step 5: Make sure you keep the CodeBlocks path over the flex path as shown below:

**How to run the program:**

To run the program, it should be first saved with the extension .l or .lex. Run the below commands on terminal in order to run the program file.

Step 1: flex filename.l or flex filename.lex depending on the extension file is saved with.

Step 2: gcc lex.yy.c

Step 3: a.exe

Step 4: Provide the input to program in case it is required

Note: Press Ctrl+D or use some rule to stop taking inputs from the user. Please see the output images of below programs to clear if in doubt to run the programs.

**b). Create a Lex program to take input from text file and count no** **of characters, no. of lines & no. of words.**

%{ int line = 0; int chars = 0; int words

= 0;

%}

%%

[\n] {line++;words++;}

[\t ' '] {words++;}

[a-zA-Z] {chars++;}

%%

int main() {

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

printf("\nNumber of Lines are: %d",line+1); printf("\nNumber of Words are: %d",words); printf("\nNumber of Characters are: %d",chars);

}

int yywrap()

{ return

1;

}

**INPUT :**

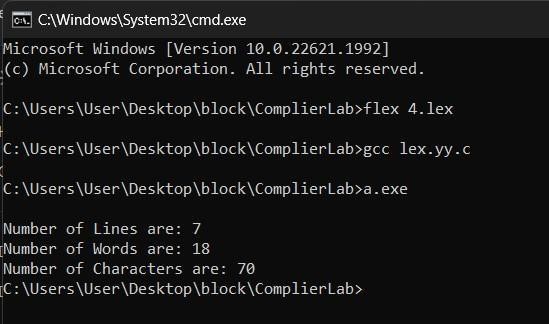
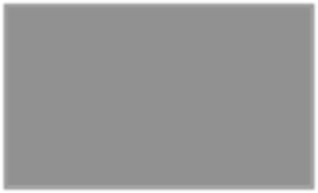
Hello This is a

exceptionally normal

file



**OUTPUT :**



# PRACTICAL 4

**a). WAP to implement yytext method in a LEX program.**

%{

#include <stdio.h>

#include <stdlib.h>

%}

%%

[0-9]+ { printf("Found %d\n", atoi(yytext)); }

. { /\* Ignore other characters \*/ }

%%

int main() { printf("Enter numbers (Ctrl+Z to end):\n"); yylex(); // Start the lexical analysis return 0; } int yywrap() { return 1; }

%}

%option noyywrap %%

"echo" { ECHO; }

"reject" { REJECT; }

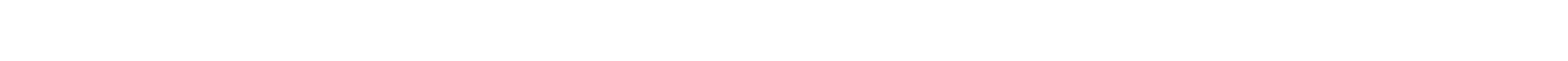
[a-zA-Z]+ { printf("Matched word: %s\n", yytext); }

[0-9]+ { printf("Matched number: %s\n", yytext); }

.|\n { ECHO; }

%{

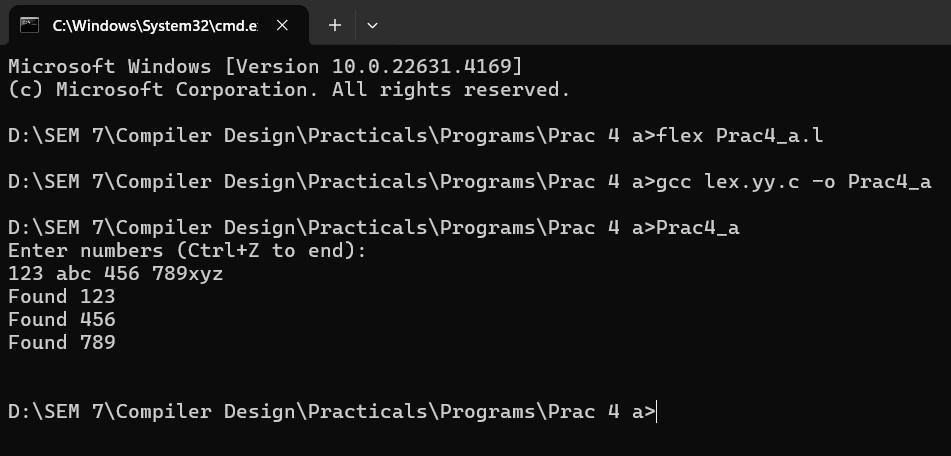
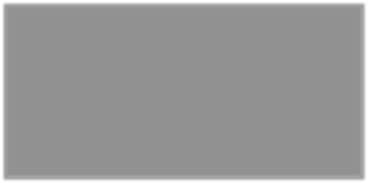
#include <stdio.h>



**b). WAP to implement ECHO, REJECT functions providedinLex.**



**Output:**



%%

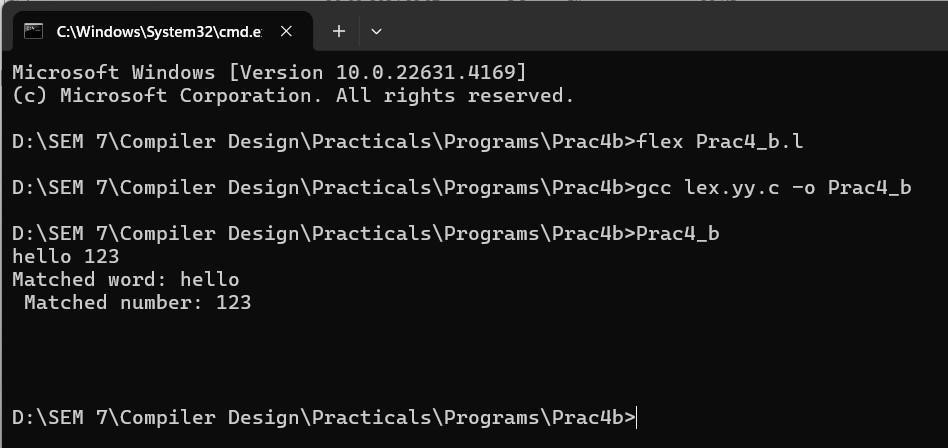
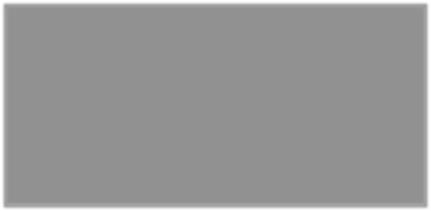
int main() { yylex(); return

0;

}



**Output:**



**c).****WAP to implement BEGIN directive in a LEX program.**

%{

#include <stdio.h>

%}

%option noyywrap

/\* Declare the start conditions \*/

%x WORD\_STATE

%%

[0-9]+ { printf("Number:

%s\n", yytext);

BEGIN WORD\_STATE; /\* Switch to WORD\_STATE after recognizing a number

\*/

}

[a-zA-Z]+ { printf("Word:

%s\n", yytext);

BEGIN INITIAL; /\* Switch back to INITIAL state after recognizing a word \*/

}

<INITIAL>. {

printf("Other (in INITIAL): %s\n", yytext);

}

<WORD\_STATE>. {

printf("Other (in WORD\_STATE): %s\n", yytext);

}

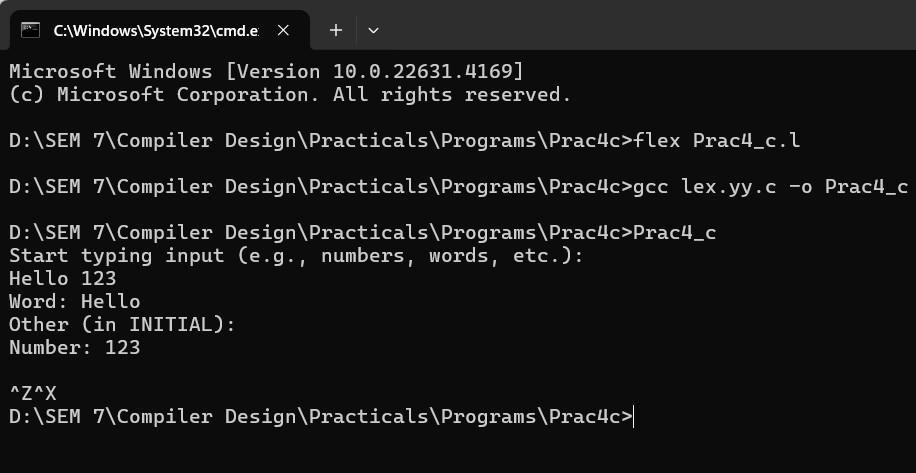
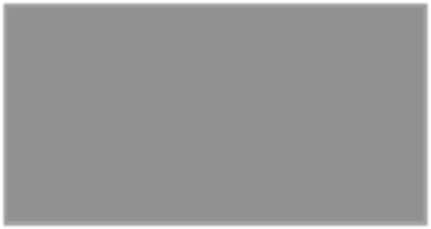
%%

int main() { printf("Start typing input (e.g., numbers, words, etc.):\n"); yylex(); /\* Start lexical analysis \*/ return 0;

}



**Output:**



# PRACTICAL 5

**a). Write a Lex program to count number of vowels and consonants** **in a given input string.**

%{

#include <stdio.h>

int vowel\_count = 0; int consonant\_count = 0;

%}

%%

[aeiouAEIOU] { vowel\_count++; }

[b-df-hj-np-tv-zB-DF-HJ-NP-TV-Z] { consonant\_count++; }

[ \t\n] { /\* Ignore whitespace and newlines \*/ }

. { /\* Ignore all other characters \*/ }

%%

int yywrap() { return 1; // Return 1 to indicate no more input files

}

int main() { yylex(); printf("Total Vowels:

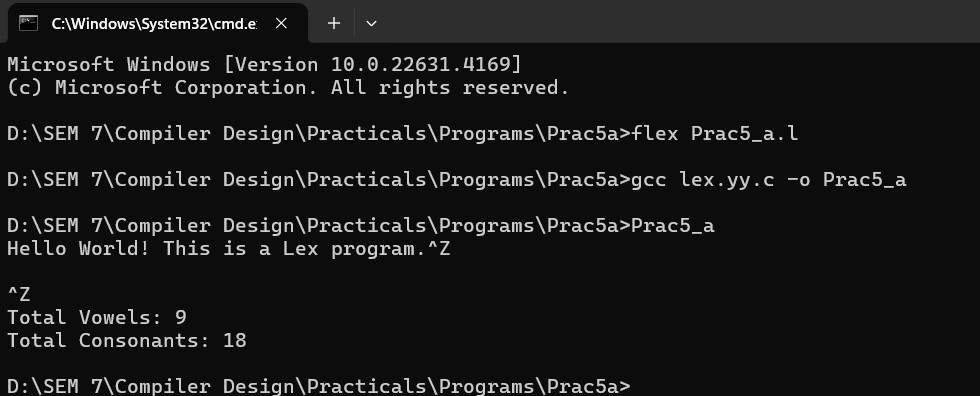
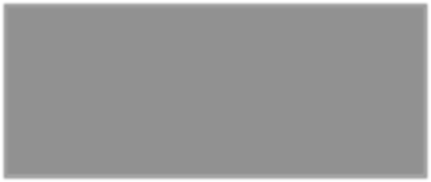
%d\n", vowel\_count); printf("Total Consonants:

%d\n", consonant\_count); return 0;

}



**Output:**



**b). Write a Lex program to print out all numbers from the given**

**file.**

%{

#include <stdio.h>

%}

%%

[0-9]+ { printf("Number: %s\n", yytext); }

.|\n { /\* Ignore other characters \*/ }

%%

int yywrap() { return 1; }

int main(int argc, char \*argv[])

{ if (argc != 2) { fprintf(stderr, "Usage: %s

<filename>\n", argv[0]); return 1;

}

FILE \*file = fopen(argv[1], "r"); if (!file) { perror("Error opening file"); return 1;

}

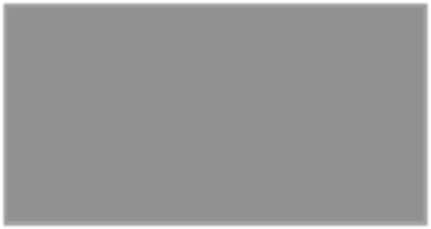
yyin = file; yylex();

fclose(file); return 0; }  **Input.txt**

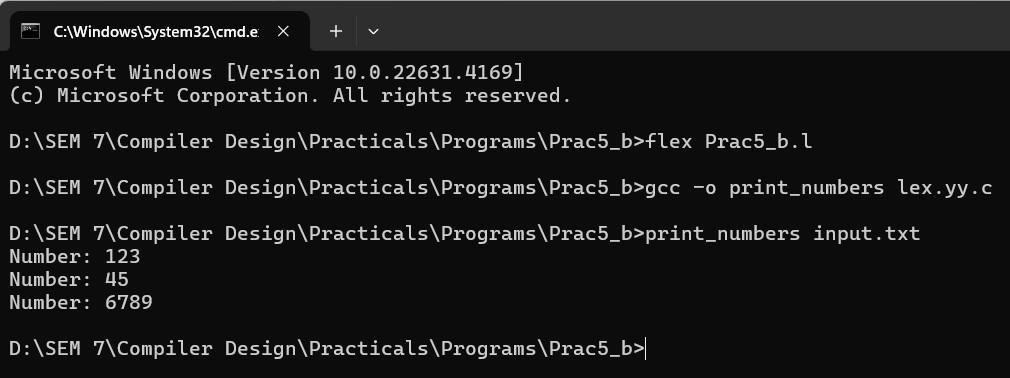
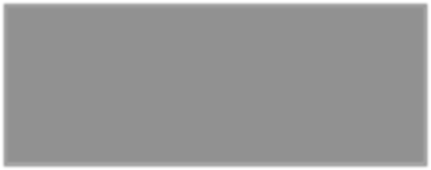
123

Some text 45

More numbers 6789



**Output:**



**c). Write a Lex program to count the number of comment lines in**

**a given C program.**

%{

#include <stdio.h>

int single\_line\_comments = 0; // Count of single-line comments int

multi\_line\_comments = 0; // Count of multi-line comments int in\_multi\_line\_comment = 0; // Flag to track if inside a multi-line comment

%}

%%

"//".\* { single\_line\_comments++; } // Single-line comment

"/\*" { in\_multi\_line\_comment = 1; } // Start of multi-line comment "\*/" { if (in\_multi\_line\_comment) { // End of multi-line comment multi\_line\_comments++; in\_multi\_line\_comment = 0;

}

}

\n { if (in\_multi\_line\_comment) { // Increment line count for multi-line comments multi\_line\_comments++;

}

}

. { /\* Ignore other characters \*/ } // Ignore other characters

%%

int yywrap() { return 1; }

int main(int argc, char \*argv[])

{ if (argc != 2) { fprintf(stderr, "Usage: %s

<filename>\n", argv[0]); return 1;

}

FILE \*file = fopen(argv[1], "r"); if (!file) { perror("Error opening file"); return 1;

}

yyin = file; yylex(); fclose(file);

// Output the count of comments printf("Number of single-line comments: %d\n", single\_line\_comments); printf("Number of multi-line comment lines: %d\n", multi\_line\_comments);

return 0;

}

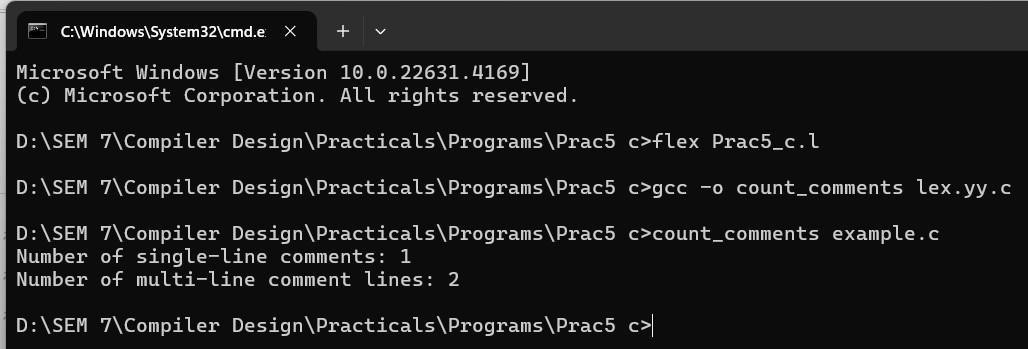
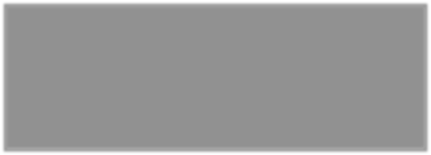
example.c

#include <stdio.h>

// This is a single-line comment int main() { printf("Hello, World!\n"); /\* This is a multi-line comment \*/ return 0;

}

**Output:**



# PRACTICAL 6

**a). WAP to implement unput and input.**

**Unput:**

%{

#define yywrap() 1

%}

%%

"un" { printf("\n the unput character=");ECHO; } [a-z]+ { printf("\n string contains only lower case letters=");ECHO; unput('n'); unput('u'); printf("\n the string after unput="); ECHO;

}

[a-zA-Z]+ { printf("\n string contains both upper and lower case letters=");ECHO;

}

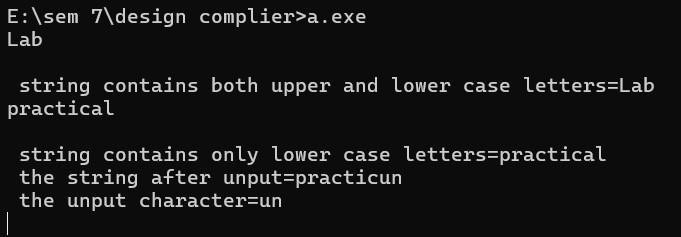
%%

int main() { yylex();return

0;

}

**Output:**



**Input:**

%{

#define yywrap() 1

%}

%%

[a-zA-Z0-9]+ { printf("\n string contains mixed letters=");ECHO;

}

"/\*" { printf("\n the comment begins");int c; while ((c = input()) != '\*') {if (c == EOF) { printf("\n Error:

Comment not closed\n");return 1;

} } if ((c = input()) == '/') { printf("\n the comment ends");

} else { printf("\n Error: Invalid comment format\n");

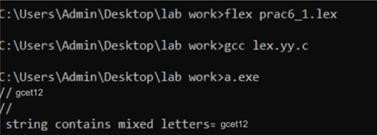
}

}

%% int main() {yylex(); return 0;

}

**Output:**



**b). WAP to implement yy terminate, yy\_flush\_bufferin LEX program.**

%{

#undef yywrap

#define yywrap() 1

%}

%%

[a-z]+ { printf("\nlowercase word = ");ECHO; printf("\nStart of yyteminate()"); yyterminate(); printf("\nEnd of yyterminate");

}

[a-zA-Z]+ { printf("\nmixed word = " );

ECHO;

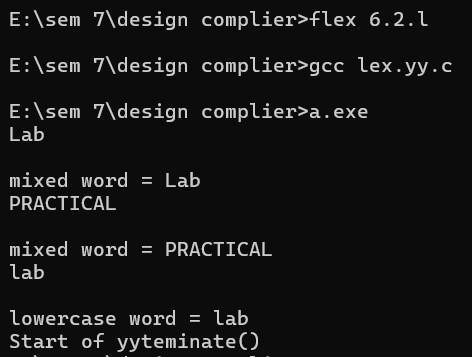
}

%%

int main() { yylex(); return 0;

}

**Output:**



%{

#undef yywrap

#define yywrap() 1

%}

%%

[a-z]+ { printf("\nlowercase word using yytext =

%s", yytext); printf("\nlowercase word using echo = ");

ECHO; printf("\nlength of the word = %d", yyleng); YY\_FLUSH\_BUFFER;

printf("\nlowercase word using yytext after yy\_flush\_buffer = %s", yytext); printf("\nlowercase word using echo after yy\_flush\_buffer = "); ECHO; printf("\nlength of the word after yy\_flush\_buffer = %d", yyleng);

}

[a-zA-Z]+ { printf("\nmixed word = ");

ECHO;

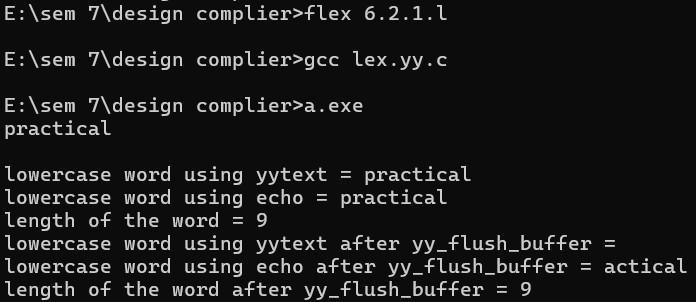
}

%%

int main() { yylex(); return

0; }

**Output:**



**c). WAP to implement yy wrapin LEX program.**

%{

#include <stdio.h>

%}

%%

[a-zA-Z]+ { printf("Word:

%s\n", yytext);

} [0-9]+ { printf("Number:

%s\n", yytext);

} . { printf("Invalid character: %c\n", yytext[0]);

}

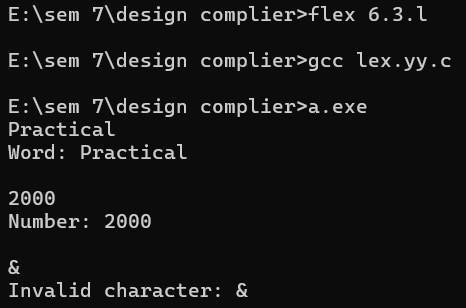
%%

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

0;

}

**Output:**



**d). WAP to implement yy more and yy less in LEX program.**

%{

#define yywrap() 1

%}

%%

[a-z]+ { printf("\n lower case letters= ");

ECHO;

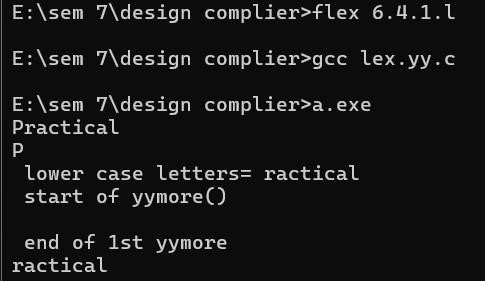
printf("\n start of yymore()\n");yymore(); printf("\n end of 1st yymore\n"); }

%%

int main() { yylex();return 0;

}

**Output:**



%{

#undef yywrap

#define yywrap() 1

%}

%%

[a-z]+ { printf("\nLowercase word = ");ECHO; yyless(3); printf("\nThe word after yyless() = ");ECHO;

}

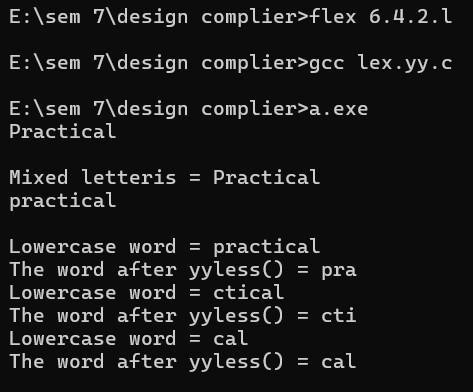
[a-zA-Z]+ { printf("\nMixed letteris = ");ECHO;

}

%%

int main() { yylex(); return 0; }

**Output:**



# PRACTICAL 7

**WAP to Find the “First” set Input: The string consists of grammar symbols. Output: The First set for a given string. Explanation: The student has to assume a typical grammar. The program when run will ask for the string to be entered. The program will find the First set of the given string.**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#define MAX 10

void findFirst(char, int, int); void addToResultSet(char);

int n;

char production[MAX][MAX], first[MAX];

int main() { int i, choice; char c, ch; printf("Enter the number of productions: "); scanf("%d", &n);

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

printf("Enter production %d: ", i+1); scanf("%s", production[i]);

} do { printf("\nEnter the symbol to find First set: "); scanf(" %c", &c);

findFirst(c, 0, 0); printf("First(%c) = { ", c); for(i = 0; i < strlen(first); i++) { printf("%c ", first[i]);

}

printf("}\n");

// Reset the first set for next input first[0]

= '\0';

printf("Do you want to continue? (1 to continue, 0 to exit): "); scanf("%d", &choice); } while(choice == 1); return 0;

}

// Function to find the First set of a given symbol void findFirst(char

c, int q1, int q2) {

int j;

// If the symbol is a terminal, add it to the result set if(!(isupper(c))) { addToResultSet(c); return;

}

// Otherwise, find the production rules for the non-terminal for(j = 0; j < n; j++) { if(production[j][0] == c) { if(production[j][2] == '#') { // '#' is used to represent epsilon (ε) if(production[q1][q2] == '\0') addToResultSet('#'); else if(production[q1][q2] != '\0' &&

(q1 != 0 || q2 != 0)) findFirst(production[q1][q2], q1,

(q2 + 1)); else addToResultSet('#');

} else if(!isupper(production[j][2])) {

// If production is a terminal, add it to the first set

addToResultSet(production[j][2]);

} else {

// Recursively find the first set of the non-terminal findFirst(production[j][2], j, 3);

}

}

}

}

void addToResultSet(char c) {

int i;

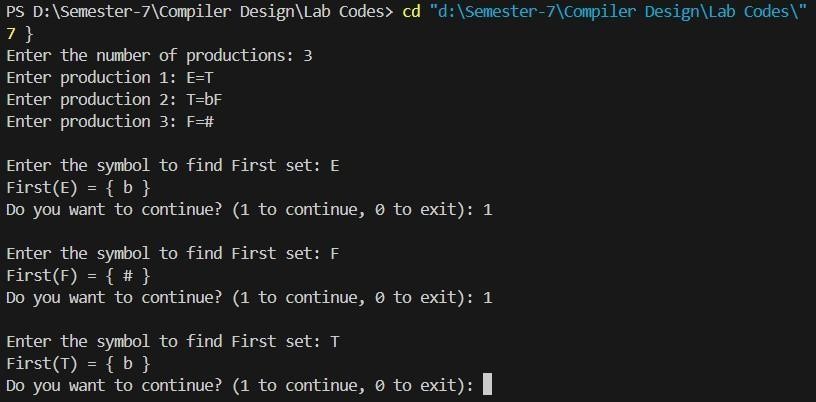
for(i = 0; i < strlen(first); i++) { if(first[i] == c) return;

}

first[strlen(first)] = c; first[strlen(first) + 1] = '\0';

}

**Output:**



# PRACTICAL 8

**WAP to Find the “Follow” set. Input: The string consists of grammar symbols. Output: The Follow set for a given string. Explanation: The student has to assume a typical grammar. The program when run will ask for the string to be entered. The program will find the Follow set of the given string.**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#define MAX 10

void findFirst(char, int, int); void findFollow(char); void addToResultSet(char[],

char); int

n;

char production[MAX][MAX], first[MAX], follow[MAX];

int m = 0, followVisited[MAX] = {0}; int main() { int i, choice; char c; printf("Enter the number of productions: "); scanf("%d",

&n); for (i = 0; i < n; i++) { printf("Enter production %d: ", i

+ 1); scanf("%s", production[i]);

}

do

{

printf("\nEnter the symbol to find Follow set: "); scanf(" %c", &c);

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

followVisited[i] = 0; findFollow(c);

printf("Follow(%c) = { ", c);

for (i = 0; i < strlen(follow); i++)

{ printf("%c ", follow[i]);

}

printf("}\n"); follow[0]

= '\0';

m = 0; printf("Do you want to continue? (1 to continue, 0 to exit): "); scanf("%d", &choice); } while (choice == 1); return 0; } void findFirst(char c, int q1, int q2)

{ int j; if

(!isupper(c))

{

addToResultSet(first, c); return; } for (j =

0; j < n; j++)

{ if (production[j][0] == c)

{ if (production[j][2] == '#')

{

if (production[q1][q2] == '\0')

{

addToResultSet(first, '#'); } else

{ findFirst(production[q1][q2], q1, q2 + 1);

}

}

else if (!isupper(production[j][2]))

{

addToResultSet(first, production[j][2]);

}

else

{ findFirst(production[j][2], j, 3);

}

}

}

} void findFollow(char

c) { int i, j;

if (followVisited[c - 'A']) return;

followVisited[c -

'A'] = 1;

if (production[0][0] == c)

{

addToResultSet(follow, '$');

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

{ for (j = 2; j < strlen(production[i]); j++)

{ if (production[i][j]

== c) { if

(production[i][j + 1] != '\0')

{ findFirst(production[i][j + 1], i,

j + 2);

int k; for (k = 0; k < strlen(first); k++)

{ if

(first[k] != '#')

{

addToResultSet(follow, first[k]); } } if (strchr(first, '#'))

{

findFollow(production[i][0]);

} } else { if (c != production[i][0])

{

findFollow(production[i][0]);

}

}

}

}

}

} void addToResultSet(char result[], char

c)

{ int i; for (i = 0; i < strlen(result); i++)

{ if (result[i] == c)

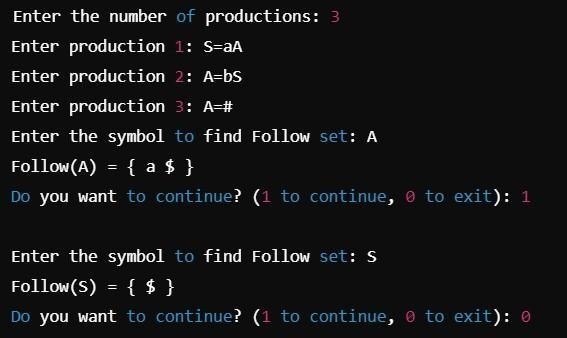
{

return;

} } result[strlen(result)] = c; result[strlen(result) + 1] = '\0';

}

**Output:**



## PRACTICAL 9

**Construct a recursive descent parser for a given grammar.**

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<stdlib.h>

#include<ctype.h>

char ip\_sym[15],ip\_ptr=0,op[50],tmp[50]; void e\_prime(); void e(); void t\_prime(); void t(); void f(); void advance(); int n=0;

void e() {

strcpy(op,"TE'"); printf("E=%-

25s",op); printf("E->TE'\n"); t();

e\_prime();

}

void e\_prime() {

int i,n=0,l;

for(i=0;i<=strlen(op);i++)

if(op[i]!='e')

tmp[n++]=op[i];

strcpy(op,tmp); l=strlen(op); for(n=0;n < l && op[n]!='E';n++) if(ip\_sym[ip\_ptr]=='+')

{

i=n+2;

do {

op[i+2]=op[i]; i++;

}while(i<=l); op[n++]='+';

op[n++]='T'; op[n++]='E'; op[n++]=39;

printf("E=%25s",op);

|  |  |  |
| --- | --- | --- |
| printf("E'>+TE'\n"); |  | advance(); |
| t(); |  | e\_prime(); |

}

else { op[n]='e';

for(i=n+1;i<=strlen(op);i++)

op[i]=op[i+1];

printf("E=%25s",op); printf("E'-

>e");

}

}

void t() {

int i,n=0,l;

for(i=0;i<=strlen(op);i++)

if(op[i]!='e') tmp[n++]=op[i];

strcpy(op,tmp);

l=strlen(op); for(n=0;n < l && op[n]!='T';n++) i=n+1; do {

op[i+2]=op[i];

i++;

}while(i < l);

op[n++]='F';

op[n++]='T'; op[n++]=39; printf("E=%25s",op); printf("T->FT'\n"); f(); t\_prime();

}

void t\_prime() {

int i,n=0,l; for(i=0;i<=strlen(op);i++)

if(op[i]!='e')

tmp[n++]=op[i];

strcpy(op,tmp); l=strlen(op); for(n=0;n < l && op[n]!='T';n++) if(ip\_sym[ip\_ptr]=='\*')

{

i=n+2;

do {

op[i+2]=op[i]; i++; }while(i < l); op[n++]='\*'; op[n++]='F'; op[n++]='T'; op[n++]=39;

printf("E=%-25s",op); printf("T'->\*FT'\n");

advance();

f(); t\_prime();

} else {

op[n]='e';

for(i=n+1;i<=strlen(op);i++)

op[i]=op[i+1];

printf("E=%25s",op);

printf("T'->e\n");

}

}

void f() {

int i,n=0,l;

for(i=0;i<=strlen(op);i++) if(op[i]!='e')

tmp[n++]=op[i];

strcpy(op,tmp); l=strlen(op); for(n=0;n < l && op[n]!='F';n++)

if((ip\_sym[ip\_ptr]=='i')||(ip\_sym[ip\_ptr]=='I')) {

op[n]='i';

printf("E=%25s",op);

printf("F>i\n");

advance();

} else {

if(ip\_sym[ip\_ptr]=='(') { advance();

e();

if(ip\_sym[ip\_ptr]==')') { advance(); i=n+2; do {

op[i+2]=op[i]; i++;

}while(i<=l);

op[n++]='('; op[n++]='E';

op[n++]=')';

printf("E=%-

25s",op);

printf("F-

>(E)\n");

}

} else {

printf("\n\t syntax error");

getch();

exit(1);

}

}

}

void advance() {

ip\_ptr++;

}

void main() { int i;

printf("\nGrammar without left recursion");

printf("\n\t\t E->TE' \n\t\t E'->+TE'|e \n\t\t T->FT' "); printf("\n\t\t T'->\*FT'|e \n\t\t F->(E)|i"); printf("\n Enter the input expression:");

gets(ip\_sym); printf("Expressions");

printf("\t Sequence of production rules\n");

e();

for(i=0;i < strlen(ip\_sym);i++) {

if(ip\_sym[i]!='+'&&ip\_sym[i]!='\*'&&ip\_sym[i]!='('&&ip\_sym[i]!=')'&&ip\_sym[i]!='i'&&ip\_sym[i]!='

I') {

printf("\nSyntax error");

break;

}

for(i=0;i<=strlen(op);i++)

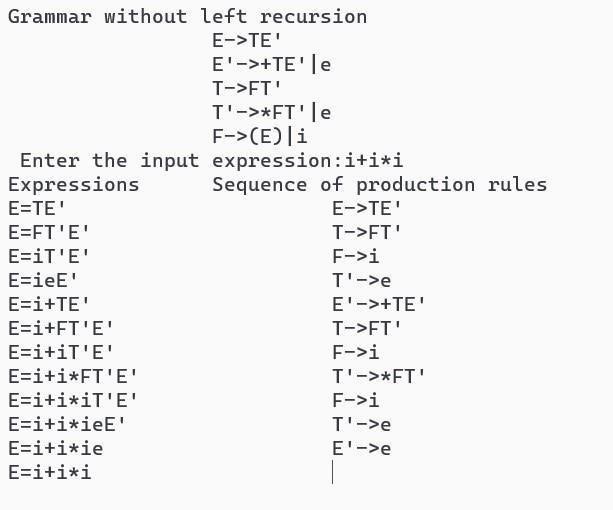
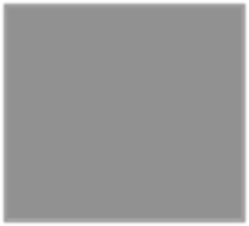
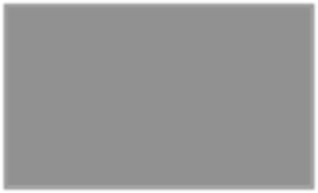
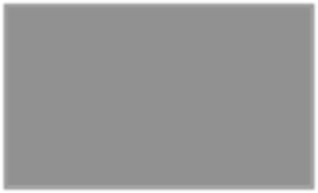
if(op[i]!='e') tmp[n++]=op[i]; strcpy(op,tmp); printf("\nE=%-

25s",op);

} getch();

}

**Output:**



**PRACTICAL 10**

**Write a C program for constructing of LL(1) parsing.**

#include<stdio.h>

#include<string.h>

char \*input; int i = 0; char lasthandle[6], stack[50]; char handles[][5] = {")E(", "E\*E", "E+E", "i", "E^E"}; int top = 0, l; char prec[9][9] = {

/\*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);

}

}

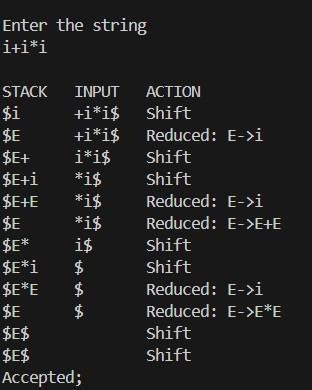
}

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

Accepted;");

}

**Output:**



**PRACTICAL 11**

**Implement a C program to implement operator precedence parsing.**

#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]={

/\*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=topt+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);

}

}

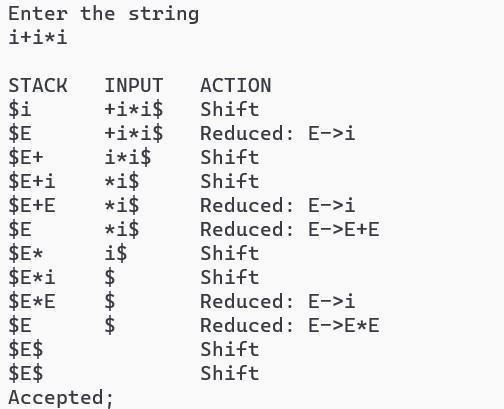
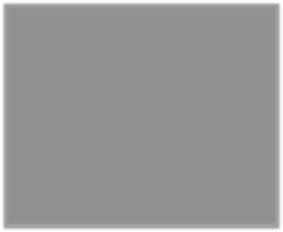
}

if(strcmp(stack,"$E$")==0)

printf("\nAccepted;"); else

printf("\nNot Accepted;"); }

**Output:**



**2**

**Given a parsing table, Parse the given input using Shift Reduce Parser for any unambiguous grammar.**

#include <stdio.h>

#include <string.h>

#define MAX\_LENGTH 100

int stack[MAX\_LENGTH]; char input[MAX\_LENGTH]; int top = -1;

void push(int element) { if (top <

MAX\_LENGTH - 1) {

stack[++top] = element;

} else { printf("Error:

Stack overflow\n");

}

}

int pop() { if (top >=

0) { return

stack[top--];

} else { printf("Error:

Stack underflow\n");

return -1;

}

}

int main() {

printf("Enter the input string: ");

scanf("%s", input);

int length = strlen(input);

int i = 0; while (i < length) {

if (top >= 0 && stack[top] == 'b' && top >= 1 && stack[top - 1] == 'a') {

pop();

pop(); push('S');

printf("Reduce: ab -> S\n");

} else {

push(input[i++]);

printf("Shift: %c\n", stack[top]);

}

}

while (top >= 1 && stack[top] == 'b' && stack[top - 1] == 'a') {

pop(); pop();

push('S');

printf("Reduce: ab -> S\n");

}

if (top == 0 && stack[top] == 'S') {

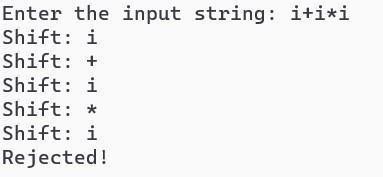
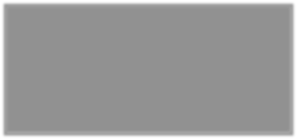
printf("Accepted!\n");

} else {

printf("Rejected!\n");

}

return 0; } **Output:**



**3**

**Introduction to YACC and generate calculator program.**

Lexical Analyzer Code:

%{

#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;

}

Parser Source Code:

%{

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

%}

%token NUMBER

%left '+' '-'

%left '\*' '/' '%'

%left '(' ')'

%%

ArithmeticExpression: 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()

{

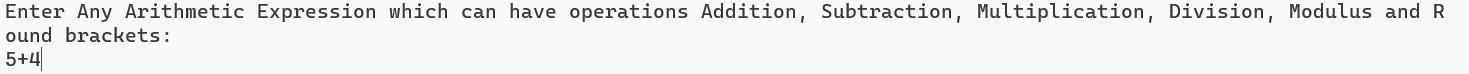
printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Division, Modulus and Round brackets:\n"); yyparse(); if(flag==0) printf("\nEntered arithmetic expression is Valid\n\n"); } void yyerror()

{ printf("\nEntered arithmetic expression is Invalid\n\n");

flag=1;

}

**Output:**



**4**

**Generate 3-tuple intermediate code for given infix expression.**

#include <stdio.h>

#include <string.h>

void pm(); void plus(); void div();

int i, ch, j, l, addr = 100; char ex[10], exp[10], exp1[10], exp2[10], id1[5], op[5], id2[5];

int main() {

while (1) {

printf("\n1.assignment\n2.arithmetic\n3.relational\n4.Exit\nEnter the choice:"); scanf("%d", &ch); switch (ch) { case 1:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | printf("\nEnter the expression with assignment | | | |  |
| operator:"); |  | scanf("%s", exp); |  |  |  | l = |
| strlen(exp); |  | exp2[0] = '\0'; |  |  |  |  |
|  |  | i = 0; |  |  |  |  |
|  |  | while (exp[i] != '=') { |  |  |  |  |
|  |  | i++; |  |  |  |  |
|  |  | } |  |  |  |  |

strncat(exp2, exp, i);

strrev(exp);

exp1[0] = '\0';

strncat(exp1, exp, l - (i + 1));

strrev(exp1);

printf("Three address code:\nt=%s\n%s=t\n", exp1, exp2);

break;

case 2:

printf("\nEnter the expression with arithmetic operator:");

scanf("%s",

ex); strcpy(exp, ex); l = strlen(exp);

exp1[0] = '\0';

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

if (exp[i] == '+' || exp[i] == '-') { if (exp[i

+ 2] == '/' || exp[i + 2] == '\*') {

pm();

break;

} else{ plus(); break;

}

} else if (exp[i] == '/' || exp[i]== '\*') { div(); break; }

}

break;

case 3:

printf("Enter the expression with relational operator:");

scanf("%s%s%s", &id1, &op, &id2);

if (((strcmp(op, "<") == 0) || (strcmp(op, ">") == 0) || (strcmp(op, "<=") == 0) || (strcmp(op, ">=") == 0)|| (strcmp(op, "==") == 0) || (strcmp(op, "!=") == 0)) ==

0)

printf("Expression is error");

else {

printf("\n%d\tif %s%s%s goto %d", addr, id1, op, id2, addr + 3);

addr++;

printf("\n%d\t T:=0", addr);  addr++; printf("\n%d\t goto %d", addr, addr + 2);

addr++; printf("\n%d\t T:=1", addr);

} break; case 4:

return 0;

}

}

return 0;

}

void pm() { strr ev(e xp); j = l- i - 1; strn cat( exp 1,exp, j); strr

ev(exp1); printf("Three address code:\nt=%s\nt1=%c%ct\n", exp1, exp[j + 1],

exp[j]);

}

void div() {

strncat(exp1, exp, i + 2); printf("Three address code:\nt=%s\nt1=t%c%c\n",

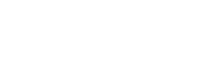
exp1, exp[i + 2], exp[i + 3]);

}

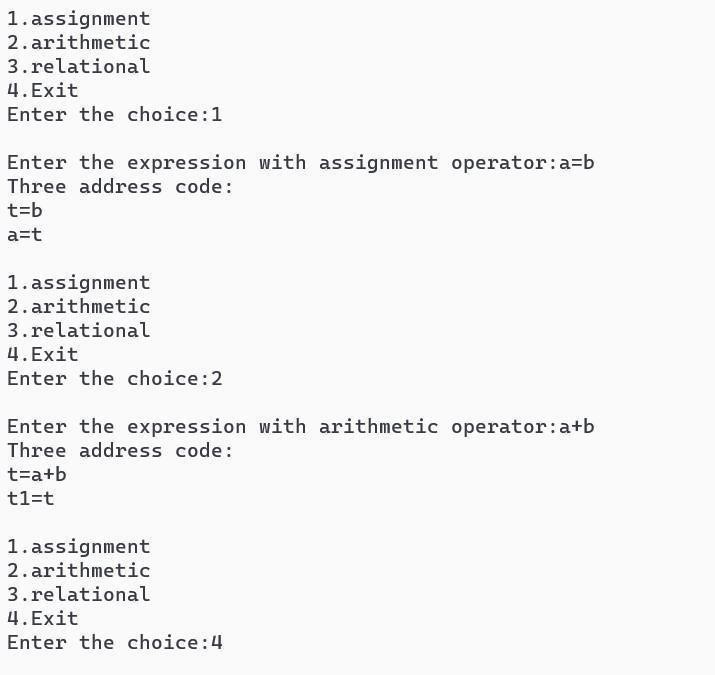
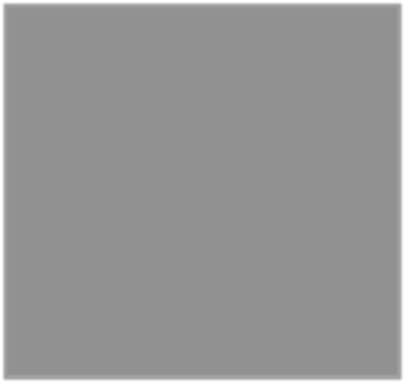
void plus() {

strncat(exp1, exp, i + 2); printf("Three address

code:\nt=%s\nt1=t%c%c\n", exp1, exp[i + 2], exp[i + 3]); }



**Output:**



## PRACTICAL 15

**Extract predecessor and successor from given control flowgraph.**

#include <stdio.h> #include

<stdlib.h>

#include <string.h>

#define MAX\_LABEL\_LENGTH 20

#define MAX\_BLOCKS 100

int main() {

int numBlocks;

printf("Enter the number of basic blocks in the CFG: "); scanf("%d", &numBlocks);

char predecessors[MAX\_BLOCKS][MAX\_BLOCKS][MAX\_LABEL\_LENGTH]; char successors[MAX\_BLOCKS][MAX\_BLOCKS][MAX\_LABEL\_LENGTH]; for

(int i = 0; i < numBlocks; i++) {

int numSuccessors; printf("Enter the number of successors for Basic Block %d: ",

i); scanf("%d", &numSuccessors); printf("Enter successors for Basic Block %d (space-separated labels): ", i);

for (int j = 0; j < numSuccessors; j++) { scanf("%s", successors[i][j]);

}

}

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

= 0; j < numBlocks; j++) { predecessors[i][j][0] = '\0';

// Initialize to empty string

}

}

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

= 0; j < numBlocks; j++) { for (int k = 0; k <

numBlocks; k++) { for (int l = 0; l < numBlocks; l++) {

if (strcmp(successors[i][j], successors[k][l]) == 0) {

sprin tf(pr edec essor s[i][j],"%sBB%d ",pred eces sors[i][j],k);

}

}

}

}

}

printf("\nPredecessors andSuccessors:\n");

for (int i = 0; i < numBlocks; i++) {

printf("Basic Block BB%d:\n", i);

printf("Predecessors: %s\n", predecessors[i][0]); printf("Successors: "); for (int j = 0; j < numBlocks; j++) { printf("%s", successors[i][j]); if (j < numBlocks - 1) {

printf(" ");

}

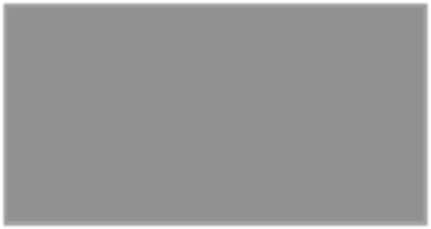
}

printf("\n\n");

}

return 0;

}



**Output:**

