Practical – 1

Aim: Analyze the language processing activities performed by below given language processors for the given code.

(1) Pre-processor

(2) Compiler

(3) Assembler

(4) Linker

Software Requirement: Linux OS, gcc Compiler

Steps:

- Install package (if not available)

gcc, cpp, as

e.g.

```
>> sudo apt install gcc
```

- Make .c file

```
>> vi <filename>.c
```

- Get expanded c program (o/p of preprocessor)

```
>> cpp <filename>.c > <filename>.i
```

- Get assembly code (o/p of compiler)

```
>> gcc -S <filename>.i
```

- Get object code (o/p of assembler)

```
>> as -o <filename>.o <filename>.s
```

- Get executable code

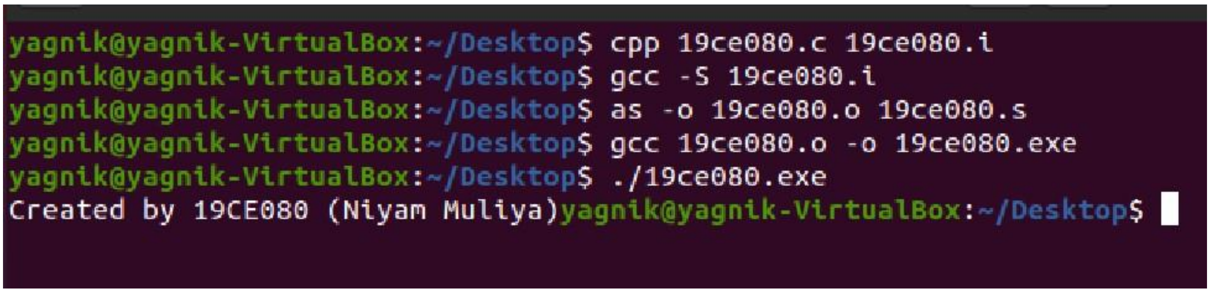
```
>> gcc <filename>.o -o <filename>.exe
```

Code:

```
#include <stdio.h>

//Prepared by Niyam

int main() {

        int id=80;

        printf("Created by 19CE0%d (Niyam Muliya)",id);

  return 0;

}
```

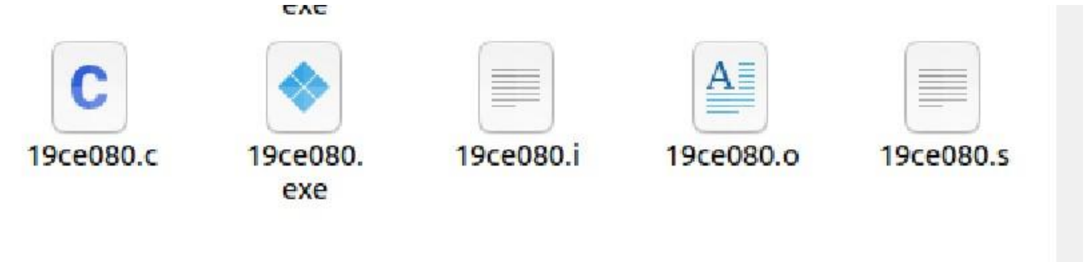
Screenshots

Code execution



```
yagnik@yagnik-VirtualBox:~/Desktop$ cpp 19ce080.c 19ce080.i
yagnik@yagnik-VirtualBox:~/Desktop$ gcc -S 19ce080.i
yagnik@yagnik-VirtualBox:~/Desktop$ as -o 19ce080.o 19ce080.s
yagnik@yagnik-VirtualBox:~/Desktop$ gcc 19ce080.o -o 19ce080.exe
yagnik@yagnik-VirtualBox:~/Desktop$ ./19ce080.exe
Created by 19CE080 (Niyam Muliya)yagnik@yagnik-VirtualBox:~/Desktop$ 
```

Files



Conclusion:

- By performing this practical, we learned the process of c/cpp file execution and also learned to create .i, .s, .exe and .o files steps by step to understand the process of compiler.

Practical – 2

Aim: Develop a program to validate the input string for given regular expression.

- **a*(b|c)db***

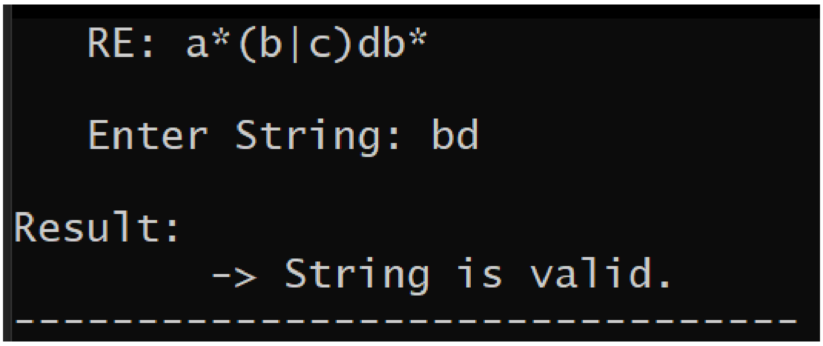
Software Requirement: C++ compiler (Dev C++)

Code:

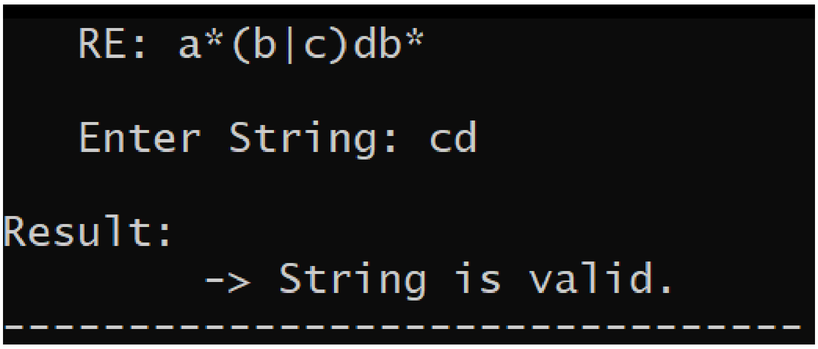
```
#include<bits/stdc++.h>
using namespace std;

int main() {
	cout<<"   RE: a*(b|c)db* \n\n";
	string str;
	cout<<"   Enter String: ";
	cin >> str;

	//RE a*(b|c)db*

	bool flag1 = 0, flag2 = 0;
	int len = str.size();
	int i = 0;
	//check for a*
	while(i < len && str[i] == 'a')
		i++;
	//check for (b|c)
	if(i < len && (str[i] == 'b' || str[i] == 'c')) {
		i++;
		flag1 = 1;
	}
	//check for d
```

```
        if(i < len && str[i] == 'd') {
                i++;
                flag2 = 1;
        }
        //check for b*
        while(i < len && str[i] == 'b') {
                i++;
        }

        //print the final result
        cout<<"\nResult: \n\t";
        if(i == len && flag1 && flag2) {
                cout<<"-> String is valid.";
        }
        else {
                cout<<"-> Invalid String.";
        }
}
```

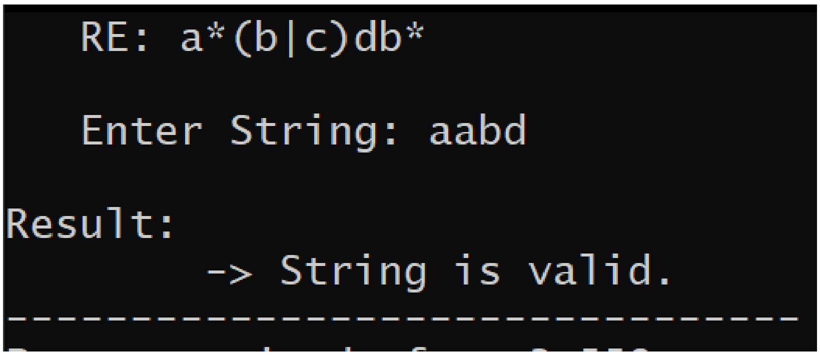
Output:



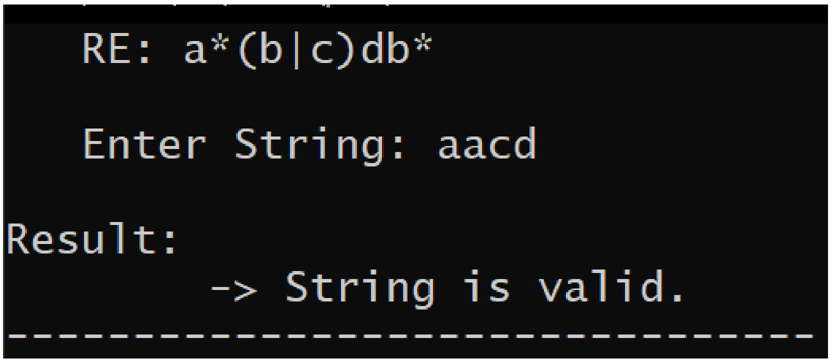
```
   RE: a*(b|c)db*

   Enter String: bd

Result:
        -> String is valid.
--------------------------------
```



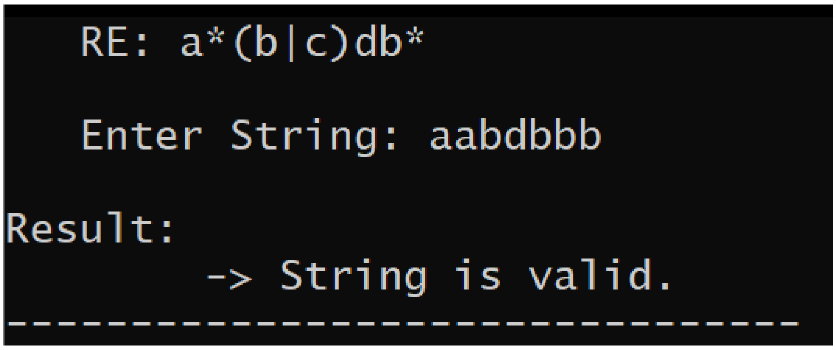
```
   RE: a*(b|c)db*

   Enter String: cd

Result:
        -> String is valid.
--------------------------------
```



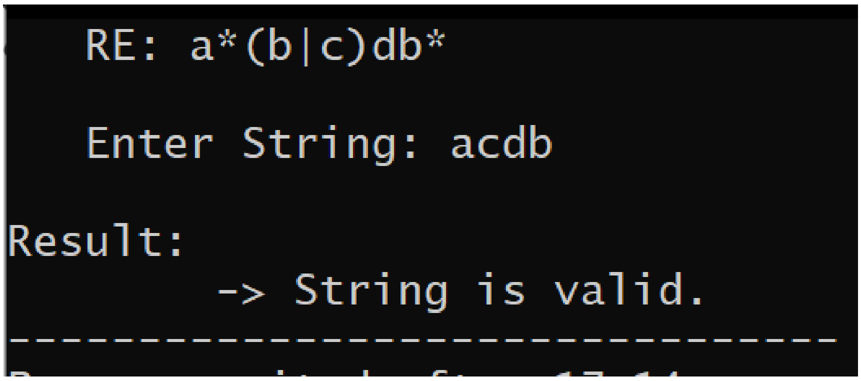
```
   RE: a*(b|c)db*

   Enter String: aabd

Result:
        -> String is valid.
--------------------------------
```



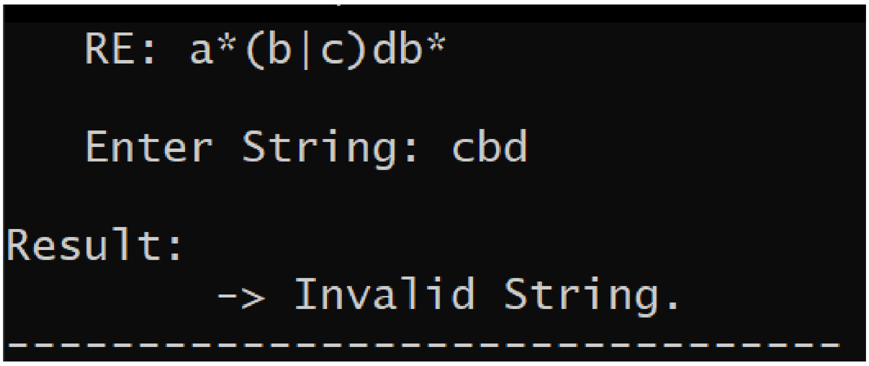
```
   RE: a*(b|c)db*

   Enter String: aacd

Result:
        -> String is valid.
--------------------------------
```



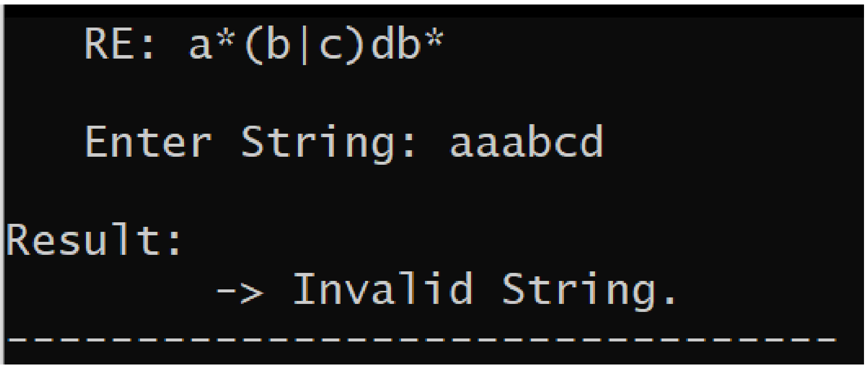
```
   RE: a*(b|c)db*

   Enter String: aabdbbb

Result:
        -> String is valid.
--------------------------------
```



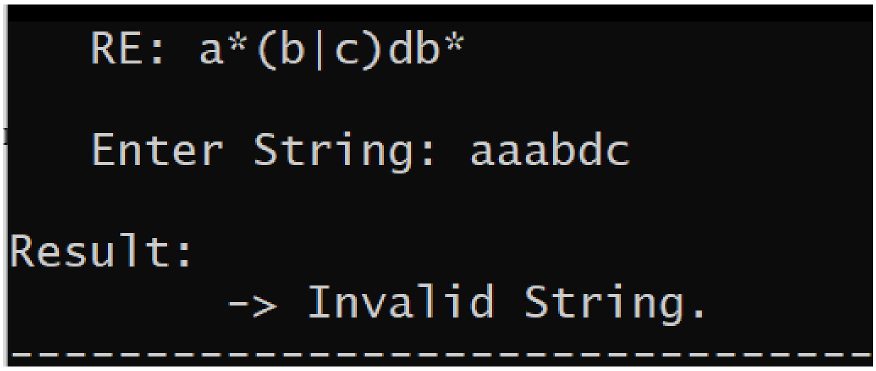
```
   RE: a*(b|c)db*

   Enter String: acdb

Result:
        -> String is valid.
--------------------------------
```



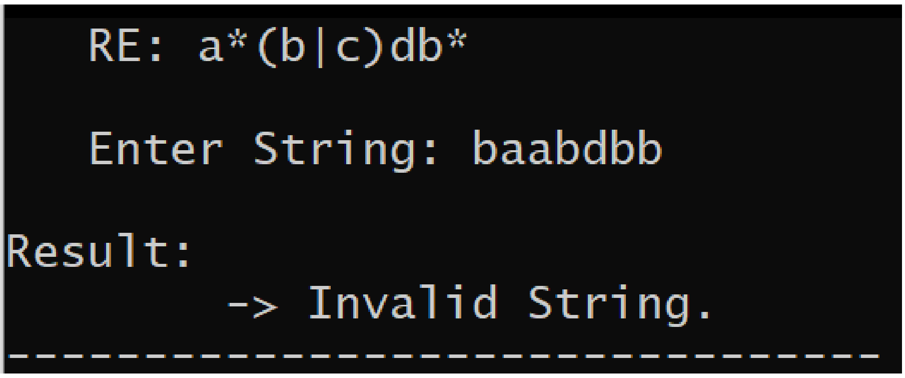
```
   RE: a*(b|c)db*

   Enter String: cbd

Result:
        -> Invalid String.
----------------------------------
```



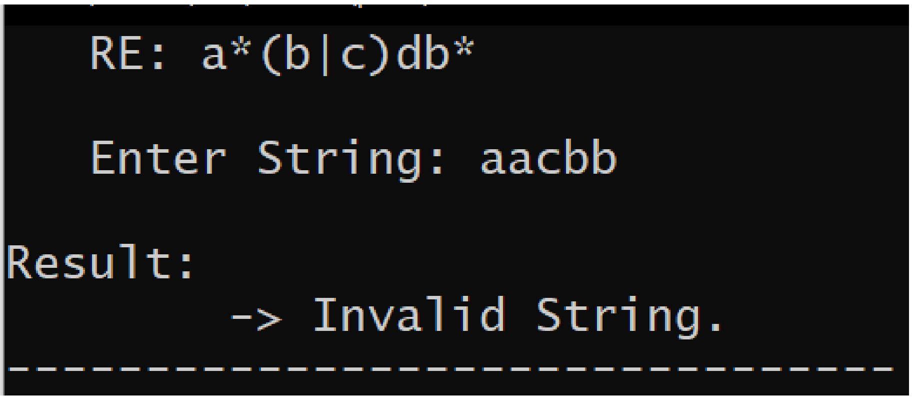
```
   RE: a*(b|c)db*

   Enter String: aaabcd

Result:
        -> Invalid String.
----------------------------------
```



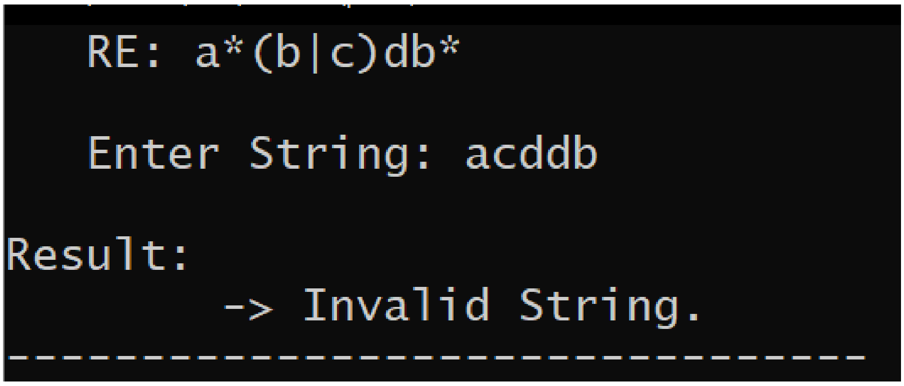
```
   RE: a*(b|c)db*

   Enter String: aaabdc

Result:
        -> Invalid String.
----------------------------------
```



```
   RE: a*(b|c)db*

   Enter String: baabdbb

Result:
        -> Invalid String.
----------------------------------
```



```
    RE: a*(b|c)db*

    Enter String: aacbb

Result:
          -> Invalid String.
----------------------------------------
```



```
    RE: a*(b|c)db*

    Enter String: acddb

Result:
          -> Invalid String.
---------------------------------------
```

Conclusion:

- By performing this practical, we learned to make a C++ program to validate the string for given regular expression.

Practical – 3

Aim: Implement a lex programs for email id and password validation. Rules for email id and password are as below.

- **Email rules**

  (1) It can be yahoo or gmail id.

  (2)yahoo is has two possible extension .com or .co.in

  (3)gmail is only one extension .com

  (4) can be alphanumeric

  (5) length can be maximum 10 letters

  (6) can support special symbols (a) _ (b) . (c) –

- **Password rules**

  (1) length can be 9 to 15 characters

  (2) include

  a. minimum one lower case letter

  b. minimum one upper case letter

  c. minim one number

  d. minimum one symbol from given set (*, ; # $ @)

Software Requirement: lex programming language

Code:

Check Email code

```
%option noyywrap


%{

#include<stdio.h>

%}


%%

[a-zA-Z0-9_.-]{1,10}(@+(gmail.com|yahoo.com|yahoo.co.in))+ {printf("-> Email is Valid.");}
```

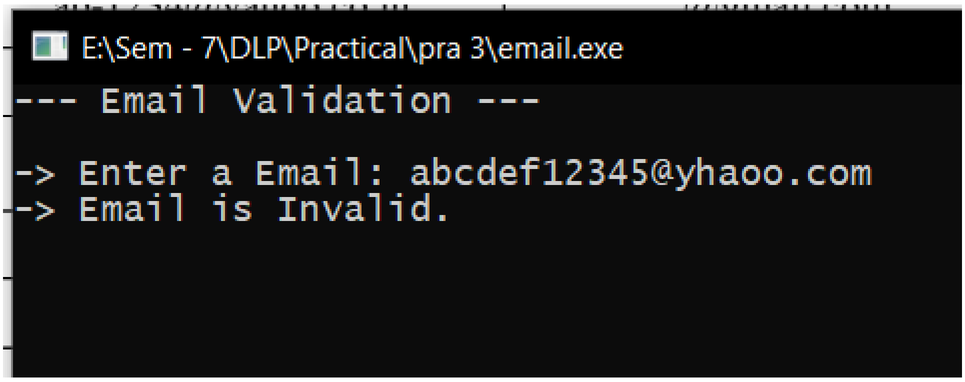
```
.* {printf("-> Email is Invalid.");}
%%

int main(){
   printf("--- Email Validation ---\n\n");
   printf("-> Enter a Email: ");
   yylex();
   printf("\n Prepared By: 19CE080");
}
```

Check Password code

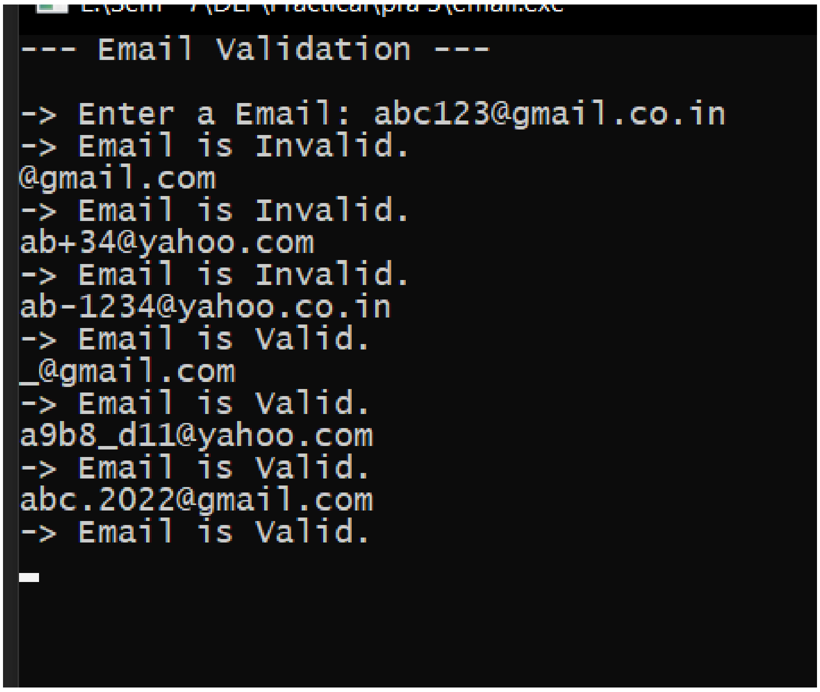
```
%option noyywrap
%{
   #include<stdio.h>
   int size=0;
   int ll = 0, ul = 0, n = 0, s = 0;
%}

%%
[0-9] {n = 1; size++;}
[A-Z] {ul = 1; size++;}
[a-z] {ll = 1; size++;}
[(*,;#$@)] {s = 1; size++;}
\n
{
   if (size >= 8 && size <= 16 && ll == 1 && ul == 1 && s == 1 && n == 1){
      printf("-> Password is Valid.\n");
   }
   else {
      printf("-> Password is Invalid.\n");
   }
}
. ;
%%

int main(){
   printf("--- Password Validation ---\n\n");
   printf("-> Enter a Password: ");
   yylex();
   printf("\n Prepared By: 19CE080");
   return 0;
}
```

Output:

Email

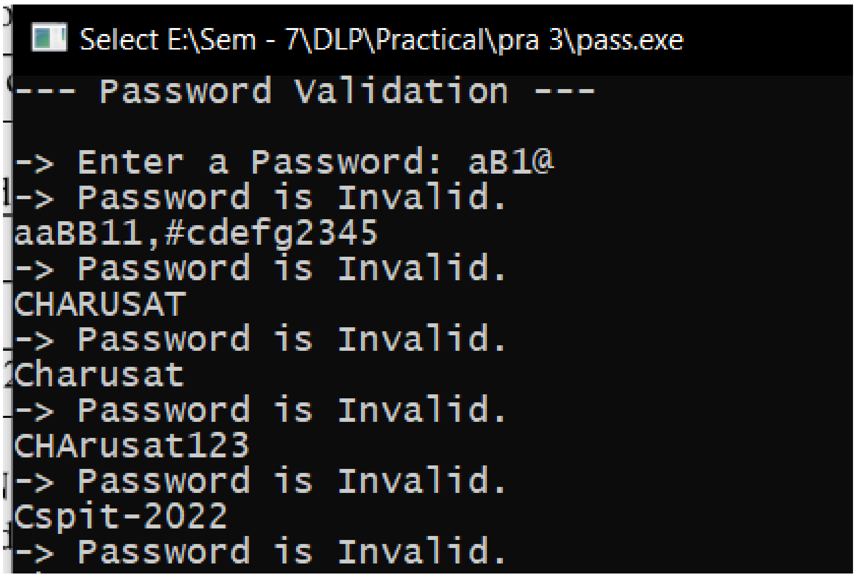


```
--- Email Validation ---

-> Enter a Email: abcdef12345@yhaoo.com
-> Email is Invalid.
```

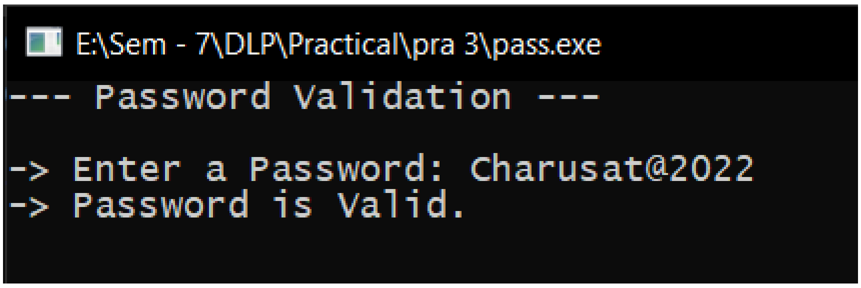


```
E:\Sem - 7\DLP\Practical\pra 3\email.exe
--- Email Validation ---

-> Enter a Email: abc123@gmail.co.in
-> Email is Invalid.
@gmail.com
-> Email is Invalid.
ab+34@yahoo.com
-> Email is Invalid.
ab-1234@yahoo.co.in
-> Email is Valid.
_@gmail.com
-> Email is Valid.
a9b8_d11@yahoo.com
-> Email is Valid.
abc.2022@gmail.com
-> Email is Valid.
```

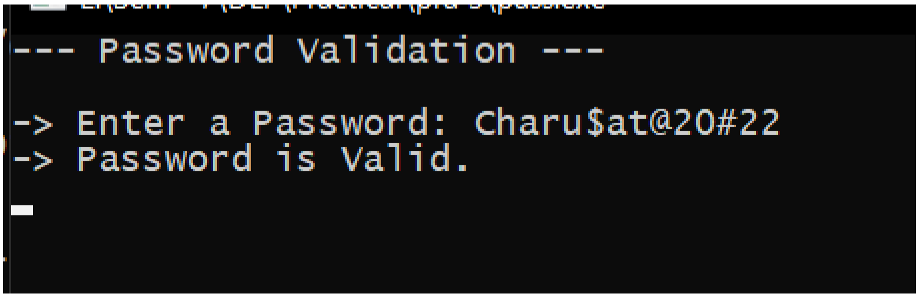
Password



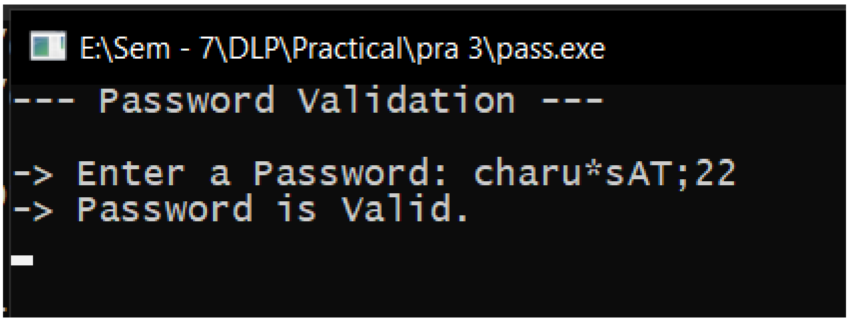
```
Select E:\Sem - 7\DLP\Practical\pra 3\pass.exe
--- Password Validation ---

-> Enter a Password: aB1@
-> Password is Invalid.
aaBB11,#cdefg2345
-> Password is Invalid.
CHARUSAT
-> Password is Invalid.
Charusat
-> Password is Invalid.
CHArusat123
-> Password is Invalid.
Cspit-2022
-> Password is Invalid.
```



```
E:\Sem - 7\DLP\Practical\pra 3\pass.exe
--- Password Validation ---

-> Enter a Password: Charusat@2022
-> Password is Valid.
```



```
--- Password Validation ---

-> Enter a Password: Charu$at@20#22
-> Password is Valid.
```



E:\Sem - 7\DLP\Practical\pra 3\pass.exe

```
--- Password Validation ---

-> Enter a Password: charu*sAT;22
-> Password is Valid.
```

Conclusion:

- By performing this practical, we learned to validate the email and password using lex programming.

Practical – 4

Aim: Design a lexical analyzer of C compiler using lex.

Software Requirement: lex programming language

Code:

```
%{
#include<stdio.h>
int SINGLE_COMMENT=0, MULTI_COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* {  if(!SINGLE_COMMENT && !MULTI_COMMENT) printf("\n%s\tPreprocessor directive",yytext);}
([ ])+ |
\t {}
\n {SINGLE_COMMENT=0;}
int |
float |
char |
const |
double |
long |
while |
for |
struct |
union |
typedef |
sizeof |
static |
singed |
unsigned |
extern |
enum |
do |
if |
break |
continue |
void |
switch |
default |
return |
else |
case |
goto {printf("\n%s\tKeyword",yytext);}
"//"  {SINGLE_COMMENT=1;}
"/*" {MULTI_COMMENT=1;}
"*/" {MULTI_COMMENT=0;}
[0-9$@]{identifier} {if(!SINGLE_COMMENT && !MULTI_COMMENT) printf("\n%s\tLAXICAL ERROR",yytext);}
{identifier}(\[[0-9]*\])? {if(!SINGLE_COMMENT && !MULTI_COMMENT) printf("\n%s\tIdentifier",yytext);}
\"[ ]*.*[ ]*\" {if(!SINGLE_COMMENT && !MULTI_COMMENT)printf("\n%s\tString",yytext);}
[0-9]+ |
[0-9]+.[0-9]+ {if(!SINGLE_COMMENT && !MULTI_COMMENT) printf("\n%s\tNumber",yytext);}
[(){}.,;"$#] {if(!SINGLE_COMMENT && !MULTI_COMMENT) printf("\n%s\tSymbol",yytext);}
= |
\<= |
\>= |
\< |
== |
\>  {if(!SINGLE_COMMENT && !MULTI_COMMENT) printf("\n%s\tOperator",yytext);}
[+*/%&^-] {if(!SINGLE_COMMENT && !MULTI_COMMENT) printf("\n%s\tOperator",yytext);}
%%

int main(int argc, char **argv)
{
	printf("----- Lexical Analyser -----\n");
	printf("=> Input is read from input.c file\n");
	FILE *file;
	file=fopen("input.c","r");
	if(!file)
	{
		printf("could not open the file");
		exit(0);
	}
	yyin=file;
	yylex();
	printf("\n\nPrepared By:- 19CE080 Niyam Muliya\n");
	return(0);
}
int yywrap()
{
	return(1);
}
```

Input files:

Sample file

```
void main()
```

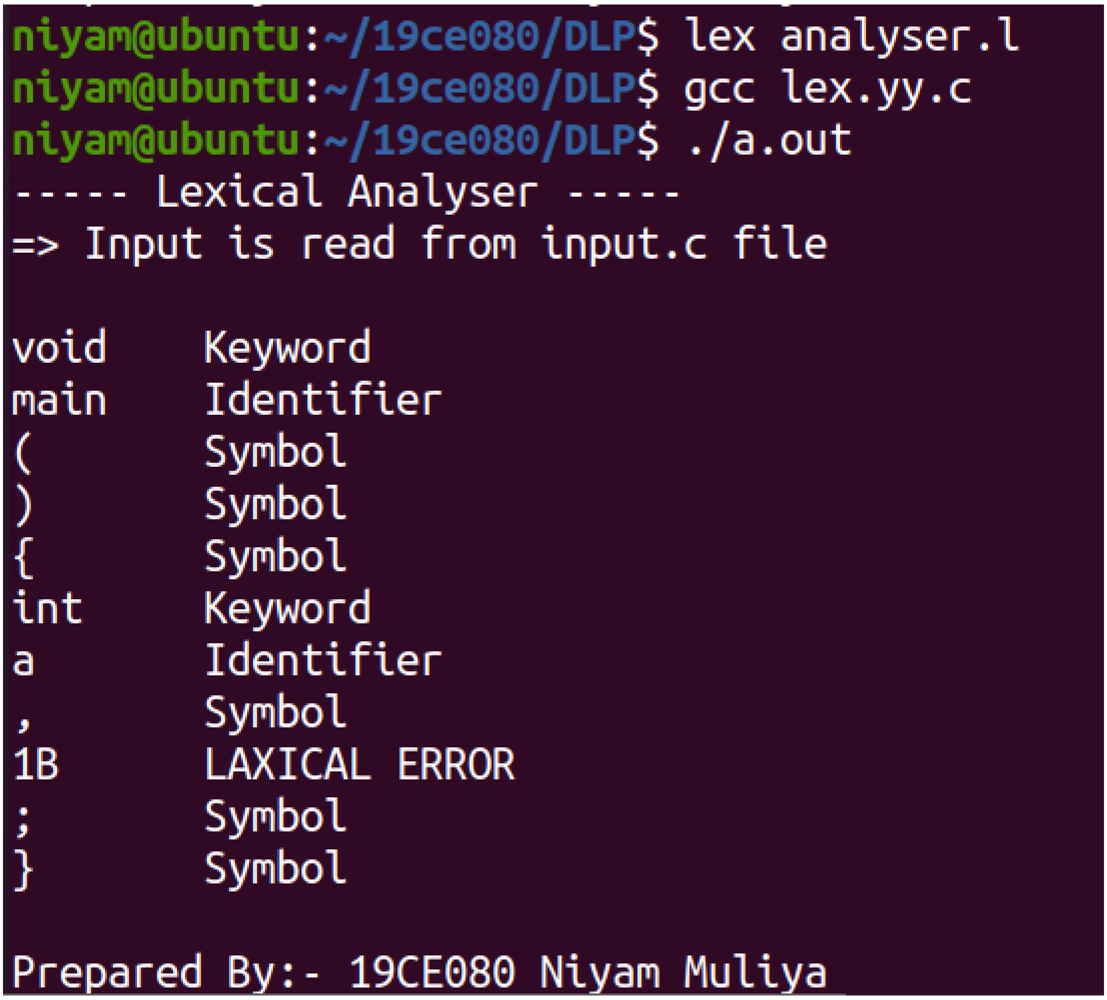
```
{
int a,1B;
}
```

Test case

```
//this is my test program-1
#include
int main()
{
int a;
a = 10 + 2.3; // it is a comment
printf("output = %d",$a);
}
/*end
of
Program
*/
```

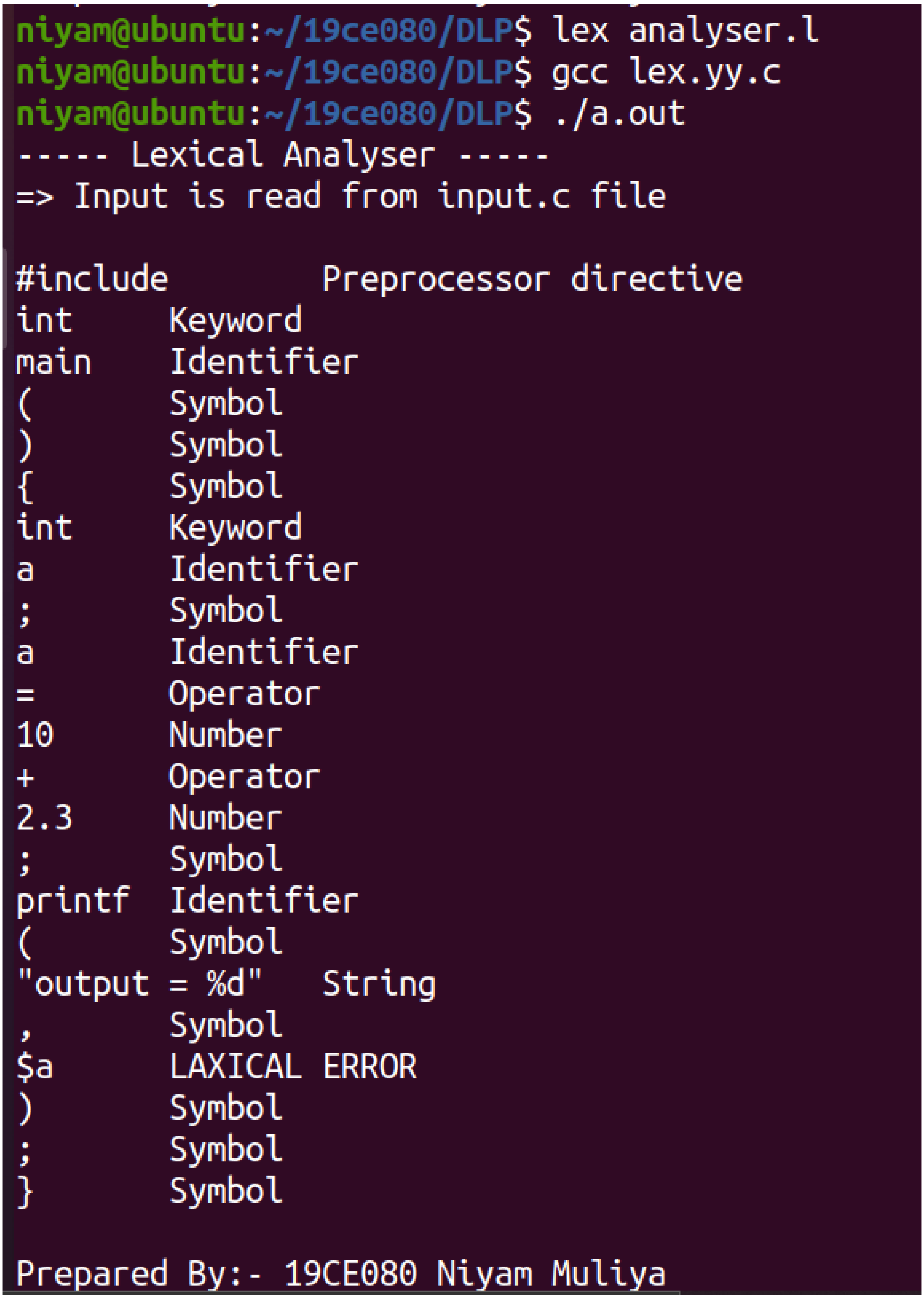
Output:

Sample input



```
niyam@ubuntu:~/19ce080/DLP$ lex analyser.l
niyam@ubuntu:~/19ce080/DLP$ gcc lex.yy.c
niyam@ubuntu:~/19ce080/DLP$ ./a.out
----- Lexical Analyser -----
=> Input is read from input.c file

void    Keyword
main    Identifier
(       Symbol
)       Symbol
{       Symbol
int     Keyword
a       Identifier
,       Symbol
1B      LAXICAL ERROR
;       Symbol
}       Symbol

Prepared By:- 19CE080 Niyam Muliya
```

Test case



```
niyam@ubuntu:~/19ce080/DLP$ lex analyser.l
niyam@ubuntu:~/19ce080/DLP$ gcc lex.yy.c
niyam@ubuntu:~/19ce080/DLP$ ./a.out
----- Lexical Analyser -----
=> Input is read from input.c file

#include        Preprocessor directive
int     Keyword
main    Identifier
(       Symbol
)       Symbol
{       Symbol
int     Keyword
a       Identifier
;       Symbol
a       Identifier
=       Operator
10      Number
+       Operator
2.3     Number
;       Symbol
printf  Identifier
(       Symbol
"output = %d"   String
,       Symbol
$a      LAXICAL ERROR
)       Symbol
;       Symbol
}       Symbol

Prepared By:- 19CE080 Niyam Muliya
```

Conclusion:

- By performing this practical, we created a lexical analyser and identified all tokens.

PRACTICAL 5

AIM: Implement a RDP for the below given grammar.

S → (L) | a

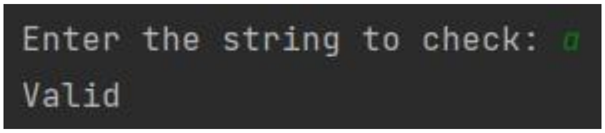
L → S L’ L’
→ ,S L’ | ϵ

Code:

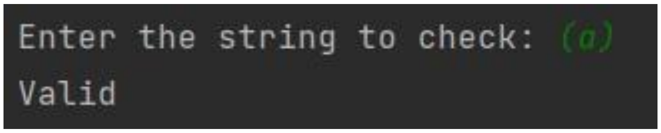
```
global s
s = list(input("Enter the string to check: "))

global i i = 0 def
match(a):     global s
global i     if
i>=len(s):         return
False     elif s[i]==a:
i+=1         return True
else:
        return False

def S():     if match("("):         if
L():             if match(")"):
return True             else:
                return False         else:
            return False     elif
match("a"):
        return True
else:         return
False

def L():     if S():         if
Lx():             return True
else:
```

```
            return False
else:        return False

def  Lx():            if  match(","):
if  S():                        if  Lx():
return  True                         else:
return False            return False
else:        return True

if __name__ == '__main__':
    if S():        if i==len(s):
print('Valid')        else:
            print('Invalid')     else:
        print('Invalid')
```

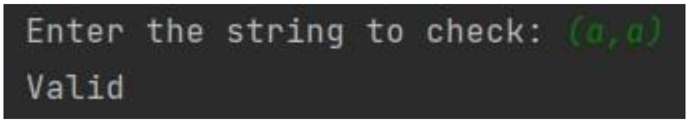
Output:



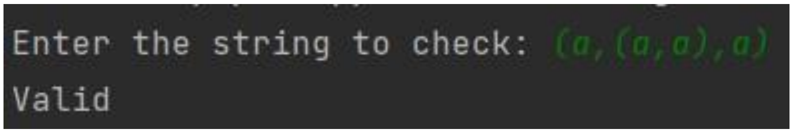
```
Enter the string to check: a
Valid
```



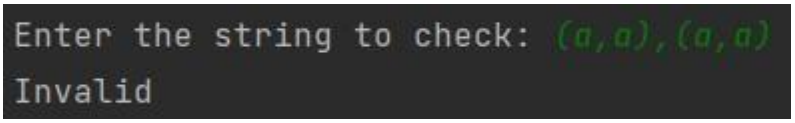
```
Enter the string to check: (a)
Valid
```



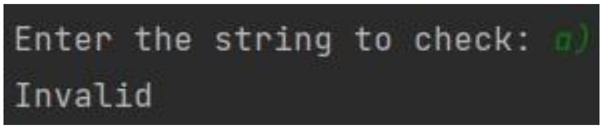
```
Enter the string to check: (a,a)
Valid
```



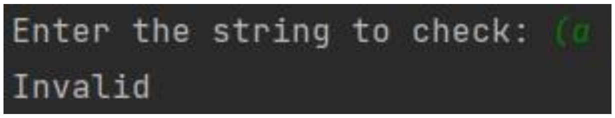
```
Enter the string to check: (a,(a,a),a)
Valid
```



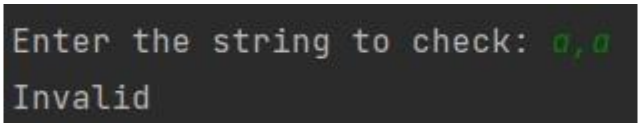
```
Enter the string to check: (a,a),(a,a)
Invalid
```



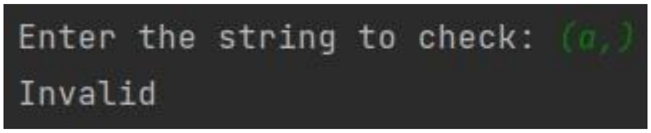
```
Enter the string to check: a)
Invalid
```



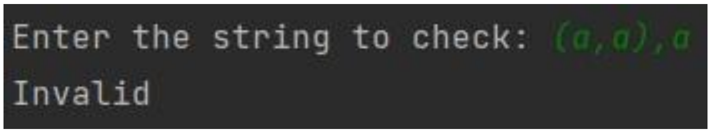
```
Enter the string to check: (a
Invalid
```



```
Enter the string to check: a,a
Invalid
```



```
Enter the string to check: (a,)
Invalid
```



```
Enter the string to check: (a,a),a
Invalid
```

PRACTICAL 6

Aim : Implement a program to validate that given grammar is LL(1) or not.

Code :

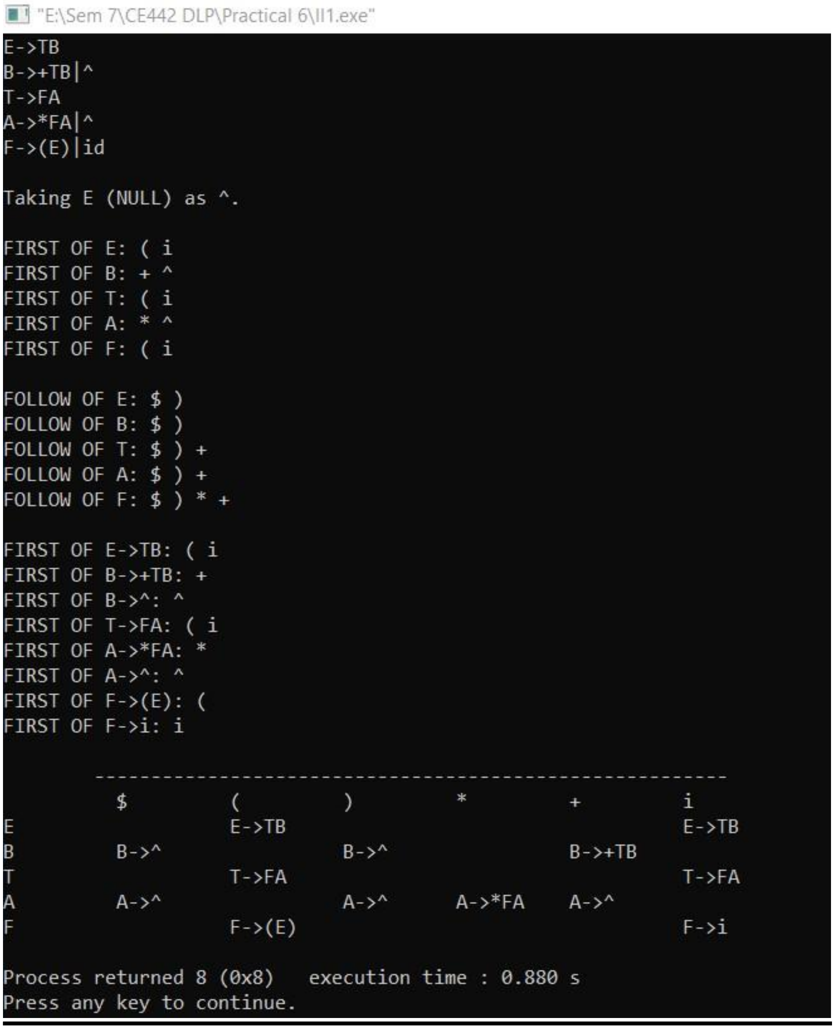
```
#include<stdio.h>
#include<string.h>
#define TSIZE 128

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("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;
    }
}
```

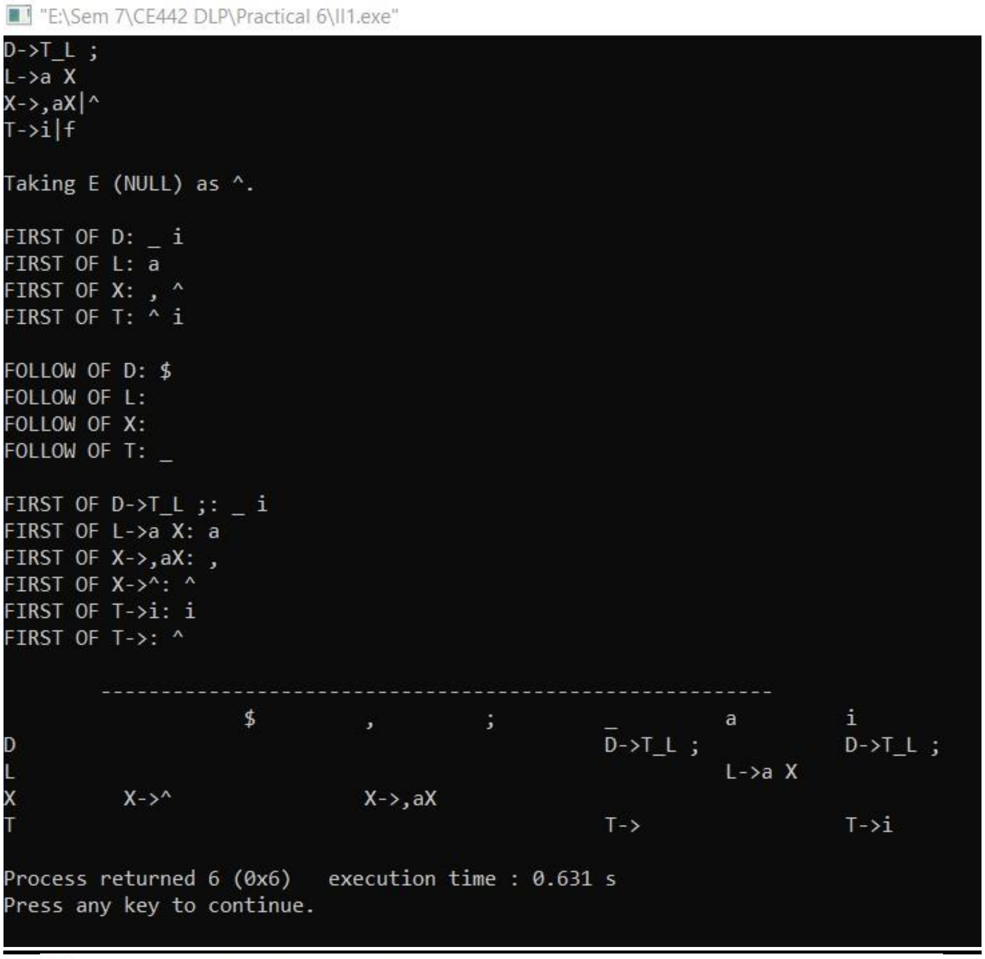
```
void add_FIRST_A_to_FIRST_RHS__B(char A, int B) {
int i;
    for (i = 0; i < TSIZE; ++i) {
        if (i != '^')
            first_rhs[B][i] = first[A - 'A'][i] || first_rhs[B][i];
    }
}

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();
printf("\n\nTaking E (NULL) as ^.");
follow[pro[0].str[0] - 'A']['$'] = 1;
    printf("\n");
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\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 :

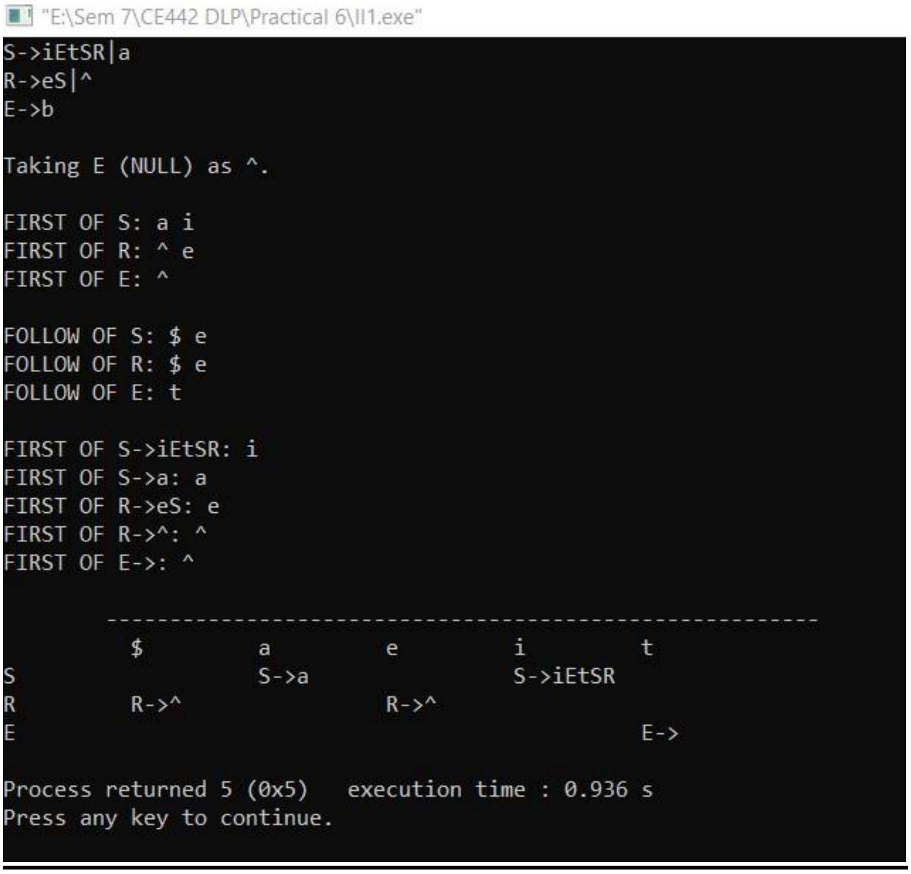


```
E->TB
B->+TB|^
T->FA
A->*FA|^
F->(E)|id

Taking E (NULL) as ^.

FIRST OF E: ( i
FIRST OF B: + ^
FIRST OF T: ( i
FIRST OF A: * ^
FIRST OF F: ( i

FOLLOW OF E: $ )
FOLLOW OF B: $ )
FOLLOW OF T: $ ) +
FOLLOW OF A: $ ) +
FOLLOW OF F: $ ) * +

FIRST OF E->TB: ( i
FIRST OF B->+TB: +
FIRST OF B->^: ^
FIRST OF T->FA: ( i
FIRST OF A->*FA: *
FIRST OF A->^: ^
FIRST OF F->(E): (
FIRST OF F->i: i

        --------------------------------------------------------
          $         (         )         *         +         i
E                   E->TB                                   E->TB
B         B->^                B->^                B->+TB
T                   T->FA                                   T->FA
A         A->^                A->^      A->*FA    A->^
F                   F->(E)                                  F->i

Process returned 8 (0x8)   execution time : 0.880 s
Press any key to continue.
```



"E:\Sem 7\CE442 DLP\Practical 6\ll1.exe"

```
D->T_L ;
L->a X
X->,aX|^
T->i|f

Taking E (NULL) as ^.

FIRST OF D: _ i
FIRST OF L: a
FIRST OF X: , ^
FIRST OF T: ^ i

FOLLOW OF D: $
FOLLOW OF L:
FOLLOW OF X:
FOLLOW OF T: _

FIRST OF D->T_L ;: _ i
FIRST OF L->a X: a
FIRST OF X->,aX: ,
FIRST OF X->^: ^
FIRST OF T->i: i
FIRST OF T->: ^

        --------------------------------------------------------
                    $         ,         ;         _         a         i
D                                                 D->T_L ;            D->T_L ;
L                                                           L->a X
X         X->^                X->,aX
T                                                 T->                 T->i

Process returned 6 (0x6)   execution time : 0.631 s
Press any key to continue.
```



"E:\Sem 7\CE442 DLP\Practical 6\ll1.exe"

```
S->iEtSR|a
R->eS|^
E->b

Taking E (NULL) as ^.

FIRST OF S: a i
FIRST OF R: ^ e
FIRST OF E: ^

FOLLOW OF S: $ e
FOLLOW OF R: $ e
FOLLOW OF E: t

FIRST OF S->iEtSR: i
FIRST OF S->a: a
FIRST OF R->eS: e
FIRST OF R->^: ^
FIRST OF E->: ^

        --------------------------------------------------------
          $         a         e         i         t
S                   S->a                S->iEtSR
R         R->^                R->^
E                                                 E->

Process returned 5 (0x5)   execution time : 0.936 s
Press any key to continue.
```

Practical – 7

Aim: Implement a program to do syntax check of input string using passed table generated in above practical.

Code:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <set>
#include <map>
#include <stack>
 using namespace
std;
 void find_first(vector< pair<char, string> >
gram,      map< char, set<char> > &firsts,
char non_term);
 void find_follow(vector< pair<char, string> >
gram,      map< char, set<char> > &follows,
map< char, set<char> > firsts,      char non_term);


int main(int argc, char const *argv[])
{     if(argc != 3) {         cout<<"Arguments should be <grammar
file> <input string>\n";         return 1;
    }
    // Arguments check
    // cout<<argv[1]<<argv[2];


    // Parsing the grammar file     fstream
grammar_file;     grammar_file.open(argv[1],
ios::in);     if(grammar_file.fail()) {
cout<<"Error in opening grammar file\n";
return 2;
    }      cout<<"Grammar parsed from grammar
file: \n";     vector< pair<char, string> > gram;
int count = 0;     while(!grammar_file.eof()) {
char buffer[20];
        grammar_file.getline(buffer, 19);

        char lhs = buffer[0]; string
        rhs = buffer+3;
        pair <char, string> prod (lhs, rhs);
        gram.push_back(prod);
cout<<count++<<".  "<<gram.back().first<<" ->
"<<gram.back().second<<"\n";
    }
cout<<"\n";
```

```
    // Gather all non terminals     set<char>
non_terms;     for(auto i = gram.begin(); i !=
gram.end(); ++i) {         non_terms.insert(i-
>first);
    }     cout<<"The non terminals in the grammar are: ";
for(auto i = non_terms.begin(); i != non_terms.end(); ++i) {
cout<<*i<<" ";
    }
cout<<"\n";
    // Gather all terminals     set<char> terms;     for(auto i =
gram.begin(); i != gram.end(); ++i) {         for(auto ch = i-
>second.begin(); ch != i->second.end(); ++ch) {
if(!isupper(*ch)) {                 terms.insert(*ch);
            }
        }
    }
    // Remove epsilon and add end character $
terms.erase('^');     terms.insert('$');     cout<<"The
terminals in the grammar are: ";     for(auto i =
terms.begin(); i != terms.end(); ++i) {
cout<<*i<<" ";
    }
    cout<<"\n\n";

    // Start symbol is first non terminal production in grammar
char start_sym = gram.begin()->first;


    map< char, set<char> > firsts;     for(auto non_term =
non_terms.begin(); non_term != non_terms.end();
++non_term) { if(firsts[*non_term].empty()){
        find_first(gram, firsts, *non_term);
        } }

    cout<<"Firsts list: \n";
    for(auto it = firsts.begin(); it != firsts.end(); ++it) {
cout<<it->first<<" : ";         for(auto firsts_it = it-
>second.begin(); firsts_it != it-
>second.end(); ++firsts_it) {
cout<<*firsts_it<<" ";
        }
cout<<"\n";
    }
cout<<"\n";


    map< char, set<char> > follows;     //
Find follow of start variable first
```

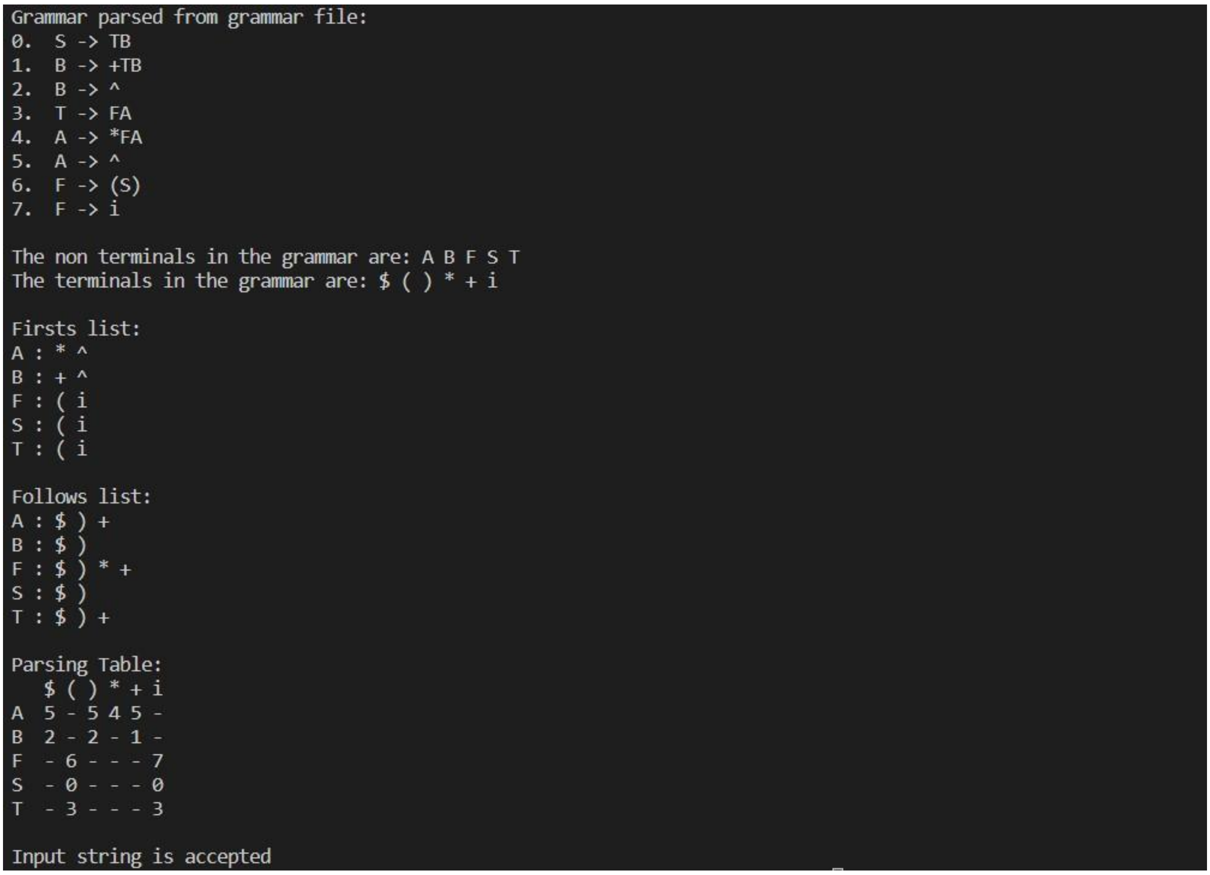
```
char start_var = gram.begin()->first;
follows[start_var].insert('$');
    find_follow(gram, follows, firsts, start_var);     // Find
follows for rest of variables     for(auto it = non_terms.begin();
it != non_terms.end(); ++it) {         if(follows[*it].empty()) {
find_follow(gram, follows, firsts, *it);
        }
    }      cout<<"Follows list: \n";     for(auto it =
follows.begin(); it != follows.end(); ++it) {         cout<<it-
>first<<" : ";         for(auto follows_it = it->second.begin();
follows_it != it-
>second.end(); ++follows_it) {
cout<<*follows_it<<" ";
        }
cout<<"\n";
    }
cout<<"\n";


    int parse_table[non_terms.size()][terms.size()];
fill(&parse_table[0][0], &parse_table[0][0] +
sizeof(parse_table)/sizeof(parse_table[0][0]), -1);
for(auto prod = gram.begin(); prod != gram.end(); ++prod) {
string rhs = prod->second;
         set<char> next_list;         bool finished =
false; for(auto ch = rhs.begin(); ch != rhs.end(); ++ch) {
if(!isupper(*ch)) {     if(*ch != '^') {
                    next_list.insert(*ch);
        finished = true;
                    break;
}                 continue;
            }
             set<char> firsts_copy(firsts[*ch].begin(),
firsts[*ch].end());             if(firsts_copy.find('^') ==
firsts_copy.end()) {
next_list.insert(firsts_copy.begin(), firsts_copy.end());
finished = true;                break;            }
firsts_copy.erase('^');             next_list.insert(firsts_copy.begin(),
firsts_copy.end());
        }
        // If the whole rhs can be skipped through epsilon or reaching the end
        // Add follow to next list         if(!finished) {
next_list.insert(follows[prod->first].begin(), follows[prod-
>first].end());
        }

        for(auto ch = next_list.begin(); ch != next_list.end(); ++ch) {
int row = distance(non_terms.begin(), non_terms.find(prod-
>first));             int col = distance(terms.begin(), terms.find(*ch));
int prod_num = distance(gram.begin(), prod);
if(parse_table[row][col] != -1) {                 cout<<"Collision at
["<<row<<"]["<<col<<"] for production
"<<prod_num<<"\n";
cout<<"Grammer is not LL(1)\n";
continue;
            }
parse_table[row][col] = prod_num;
        }

    }
    // Print parse table     cout<<"Parsing Table: \n";
cout<<"   ";     for(auto i = terms.begin(); i !=
terms.end(); ++i) {         cout<<*i<<" ";
    }     cout<<"\n"; for(auto row = non_terms.begin(); row !=
non_terms.end(); ++row) { cout<<*row<<"  "; for(int col = 0; col <
terms.size(); ++col) {     int row_num = distance(non_terms.begin(),
row);     if(parse_table[row_num][col] == -1) {
                cout<<"- ";
                continue;
            }
cout<<parse_table[row_num][col]<<" ";
        }
cout<<"\n";
    }
cout<<"\n";
     string
input_string(argv[2]);
input_string.push_back('$');
stack<char> st;
st.push('$');     st.push('S');

    // Check if input string is valid     for(auto ch =
input_string.begin(); ch != input_string.end(); ++ch) {
if(terms.find(*ch) == terms.end()) {             cout<<"Input string is
invalid\n";             return 2;
        }
    }

    // cout<<"Processing input string\n";
bool accepted = true;     while(!st.empty() &&
!input_string.empty()) {
        // If stack top same as input string char remove it
         if(input_string[0] == st.top())
{             st.pop();
input_string.erase(0, 1);
        }         else if(!isupper(st.top())) {             cout<<"Unmatched
terminal found\n";             accepted = false;             break;
```

```
}         else {             char stack_top = st.top();             int row
= distance(non_terms.begin(), non_terms.find(stack_top));             int
col = distance(terms.begin(), terms.find(input_string[0]));             int
prod_num = parse_table[row][col];
             if(prod_num == -1)
{     cout<<"No production found
in parse table\n";     accepted
= false;
                break;
        }

            st.pop();             string rhs =
gram[prod_num].second;             if(rhs[0] ==
'^') {                 continue;
            }             for(auto ch = rhs.rbegin(); ch !=
rhs.rend(); ++ch) {                 st.push(*ch);
            }
        }
    }      if(accepted) {
cout<<"Input string is accepted\n";
    }     else {         cout<<"Input
string is rejected\n";
    }

    return 0;
}  void find_first(vector< pair<char, string> >
gram,      map< char, set<char> > &firsts,
char non_term) {

    // cout<<"Finding firsts of "<<non_term<<"\n";
     for(auto it = gram.begin(); it != gram.end(); ++it)
{         // Find productions of the non terminal
if(it->first != non_term) {             continue;
        }

        // cout<<"Processing production "<<it->first<<"->"<<it->second<<"\n";
         string rhs = it-
>second;
        // Loop till a non terminal or no epsilon variable found
for(auto ch = rhs.begin(); ch != rhs.end(); ++ch) {
            // If first char in production a non term, add it to firsts list
if(!isupper(*ch)) {                 firsts[non_term].insert(*ch);
break;
            }
            else {
                // If char in prod is non terminal and whose firsts has no yet
been found out
```

```
                //   Find   first   for   that   non
terminal                  if(firsts[*ch].empty()) {
find_first(gram, firsts, *ch);
                }
                // If variable doesn't have epsilon, stop loop
if(firsts[*ch].find('^') == firsts[*ch].end()) {
firsts[non_term].insert(firsts[*ch].begin(), firsts[*ch].end());
break;
                }
                 set<char> firsts_copy(firsts[*ch].begin(),
firsts[*ch].end());
                // Remove epsilon from firsts if not the last variable
if(ch + 1 != rhs.end()) {                    firsts_copy.erase('^');
                }

                // Append firsts of that variable
firsts[non_term].insert(firsts_copy.begin(), firsts_copy.end());
            }
        }

    }
}  void find_follow(vector< pair<char, string> >
gram,      map< char, set<char> > &follows,
map< char, set<char> > firsts,      char non_term)
{

    // cout<<"Finding follow of "<<non_term<<"\n";
     for(auto it = gram.begin(); it != gram.end(); ++it)
{

        // finished is true when finding follow from this production is
complete         bool finished = true;         auto ch = it-
>second.begin();

        // Skip variables till reqd non terminal
for(;ch != it->second.end() ; ++ch) {
if(*ch == non_term) {                 finished =
false;
                break;
            }
     }
     ++ch;
         for(;ch != it->second.end() && !finished; ++ch)
{             // If non terminal, just append to follow
if(!isupper(*ch)) {
follows[non_term].insert(*ch);                 finished
= true;                 break;
            }
```

```
             set<char>
firsts_copy(firsts[*ch]);
            // If char's firsts doesnt have epsilon follow search is over
if(firsts_copy.find('^') == firsts_copy.end()) {
follows[non_term].insert(firsts_copy.begin(), firsts_copy.end());
finished = true;                 break;
            }
            // Else next char has to be checked after appending firsts to
follow             firsts_copy.erase('^');
follows[non_term].insert(firsts_copy.begin(), firsts_copy.end());
        }


        // If end of production, follow same as follow of variable
if(ch == it->second.end() && !finished) {             // Find
follow if it doesn't have             if(follows[it-
>first].empty()) {                 find_follow(gram, follows,
firsts, it->first);
            }             follows[non_term].insert(follows[it-
>first].begin(), follows[it-
>first].end());
        }

    }

}
```

Output:



```
Grammar parsed from grammar file:
0.  S -> TB
1.  B -> +TB
2.  B -> ^
3.  T -> FA
4.  A -> *FA
5.  A -> ^
6.  F -> (S)
7.  F -> i

The non terminals in the grammar are: A B F S T
The terminals in the grammar are: $ ( ) * + i

Firsts list:
A : * ^
B : + ^
F : ( i
S : ( i
T : ( i

Follows list:
A : $ ) +
B : $ )
F : $ ) * +
S : $ )
T : $ ) +

Parsing Table:
   $ ( ) * + i
A  5 - 5 4 5 -
B  2 - 2 - 1 -
F  - 6 - - - 7
S  - 0 - - - 0
T  - 3 - - - 3

Input string is accepted
```

Practical - 8

Aim:

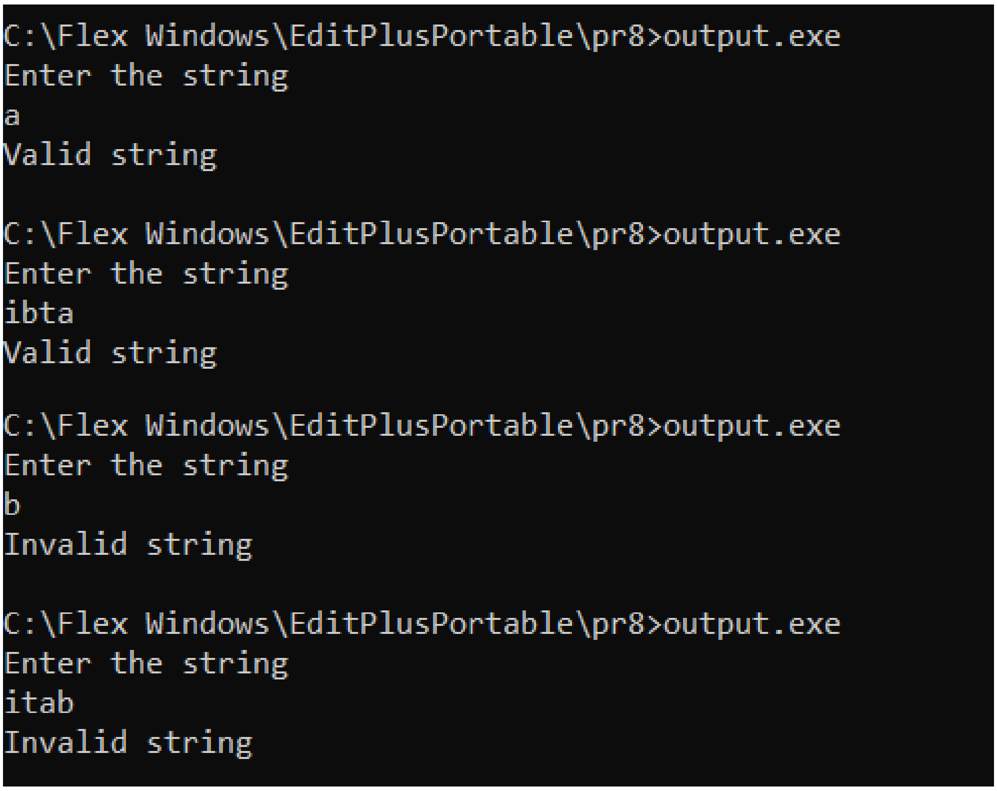
Develop a Yacc program to validate the string for the below given grammar.
S → i E t S S’ | a
S’ → e S | ε
E → b

Code:

```
%option noyywrap

%{

/* Definition section */ #include "pr8.tab.h"

%}

/* Rule Section */

%%

[i] {return i;}

[t] {return t;}

[a] {return a;}

[b] {return b;}

[e] {return e;}

\n {return NL;}

. {return yytext[0];}

%%

int yywrap1()

{

return 1;

}


%{

/* Definition section */ #include<stdio.h>

#include<stdlib.h>

%}
```

```
%token a b i t e NL
/* Rule Section */
%%
stmt: S NL {printf("Valid string\n");exit(0);}
;
S: i E t S Sdash
|
a
;
Sdash: e S
|
;
E: b
;
%%
int yyerror(char *msg)
{
printf("Invalid string\n"); exit(0);
}
//driver code
int main()
{
printf("Enter the string\n"); yyparse();
return 0;
}
```

Output:



```
C:\Flex Windows\EditPlusPortable\pr8>output.exe
Enter the string
a
Valid string

C:\Flex Windows\EditPlusPortable\pr8>output.exe
Enter the string
ibta
Valid string

C:\Flex Windows\EditPlusPortable\pr8>output.exe
Enter the string
b
Invalid string

C:\Flex Windows\EditPlusPortable\pr8>output.exe
Enter the string
itab
Invalid string
```

Practical - 9

Aim:

Implement a program to generate symbol table using appropriate data structure and enter the identifiers tokenized by 4th program.

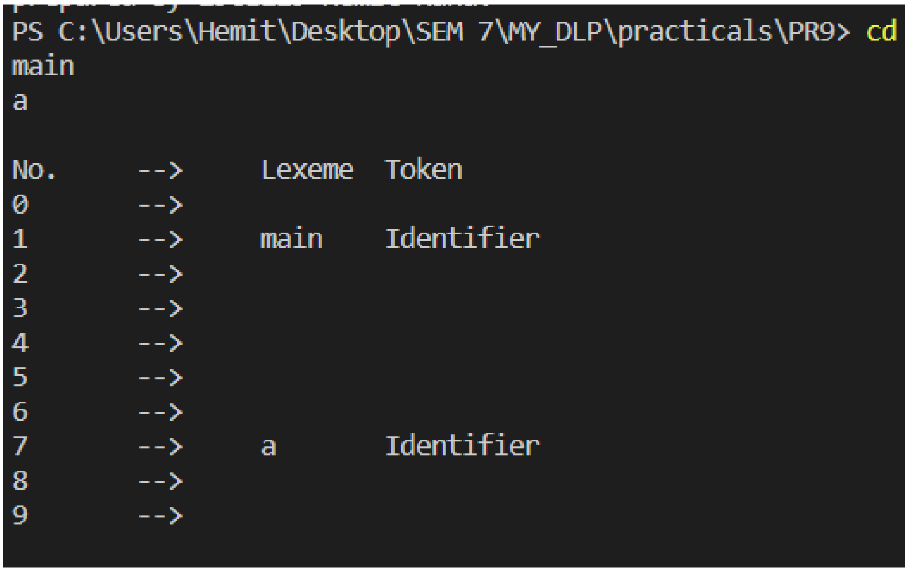
Code:

```
#include<stdio.h>

#include<iostream>

#include<stdlib.h>

#include<string.h>

#include <fstream>

using namespace std;


string hash1[10]= {};

string token[10]= {};


void hash2(string in)

{

  int a;


  for(int i=0; i<in.length(); i++){

    a=a+in[i];

  }


    int mod;

    mod = a % 10;



    if(hash1[mod]=="")

    {

      hash1[mod] = in;
```

```
            token[mod] = "Identifier";

        }
        else
        {
            mod++;
            while(hash1[mod]!="")
            {
                mod++;
                if(mod>9)
                {
                    mod=0;
                }
            }
            hash1[mod] = in;

                token[mod] = "identifier";
        }
}

int main()
{
    string input;
    fstream newfile;

   newfile.open("C:/Flex Windows/EditPlusPortable/pr4/pr9/output.txt",ios::in); //open a file to perform read operation using file object
   //newfile.open("C:/Flex Windows/EditPlusPortable/pr4/pr9/output2.txt",ios::in); //open a file to perform read operation using file object
   if (newfile.is_open()){   //checking whether the file is open
      //string tp;
      while(getline(newfile, input)){  //read data from file object and put it into string.
         cout << input << "\n";   //print the data of the string
        hash2(input);
      }
      newfile.close();   //close the file object.
    }

    cout<<"\nNo.\t-->\tLexeme\tToken";
    for(int j=0 ; j<10; j++)
    {
       cout<<endl<<j<<"\t-->\t"<<hash1[j]<<"\t"<<token[j];
    }
    return 0;
}
```

Output:

```
void main()
{
int a,1B;
}
```



```
PS C:\Users\Hemit\Desktop\SEM 7\MY_DLP\practicals\PR9> cd
main
a

No.     -->     Lexeme  Token
0       -->
1       -->     main    Identifier
2       -->
3       -->
4       -->
5       -->
6       -->
7       -->     a       Identifier
8       -->
9       -->
```

Practical - 10

Aim:

Create a program that takes an infix string as input, convert it to postfix string and generate quadruple table.

Code:

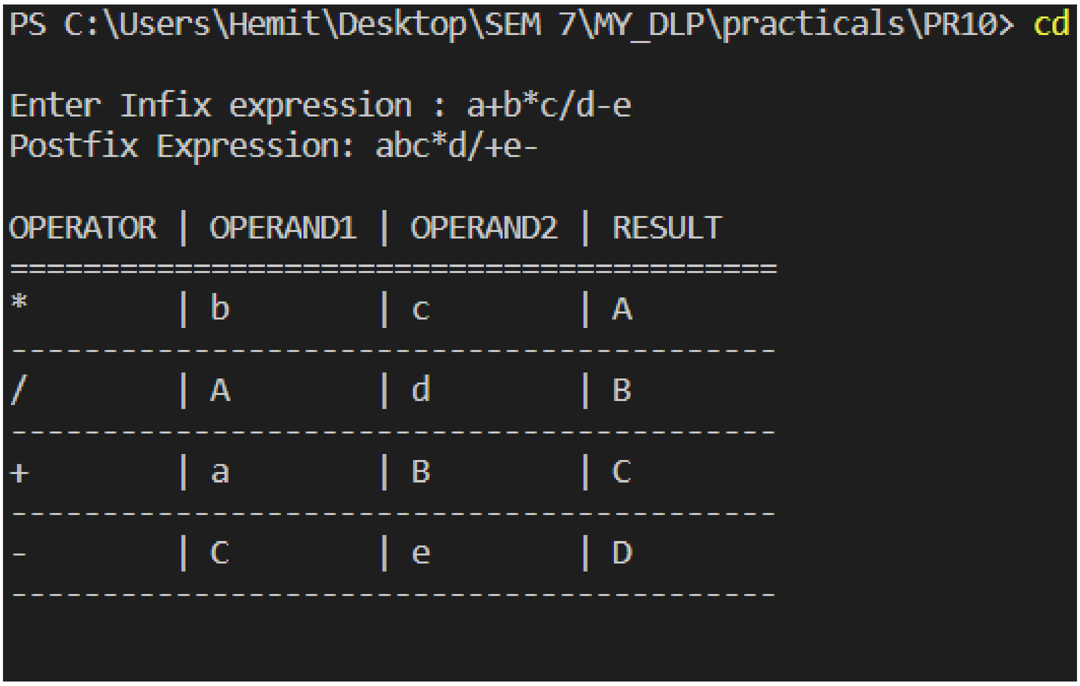
```
#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>


#define SIZE 100


/* declared here as global variable because stack[]

* is used by more than one fucntions */

char stack[SIZE];

int top = -1;


/* define push operation */


void push(char item)

{

   if(top >= SIZE-1)

   {

      printf("\nStack Overflow.");

   }

   else

   {

      top = top+1;

      stack[top] = item;

   }

}
```

```
/* define pop operation */
char pop()
{
  char item ;

  if(top <0)
  {
    printf("stack under flow: invalid infix expression");
    getchar();
    /* underflow may occur for invalid expression */
    /* where ( and ) are not matched */
    exit(1);
  }
  else
  {
    item = stack[top];
    top = top-1;
    return(item);
  }
}

/* define function that is used to determine whether any symbol is operator or not
(that is symbol is operand)
* this fucntion returns 1 if symbol is opreator else return 0 */

int is_operator(char symbol)
{
  if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol =='-')
  {
    return 1;
  }
  else
  {
    return 0;
    }
}

/* define fucntion that is used to assign precendence to operator.
* Here ^ denotes exponent operator.
* In this fucntion we assume that higher integer value
* means higher precendence */

int precedence(char symbol)
{
    if(symbol == '^')/* exponent operator, highest precedence*/
    {
        return(3);
    }
    else if(symbol == '*' || symbol == '/')
    {
        return(2);
    }
    else if(symbol == '+' || symbol == '-')          /* lowest precedence */
    {
        return(1);
    }
    else
    {
        return(0);
    }
}

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
    int i, j;
    char item;
    char x;

    push('(');                          /* push '(' onto stack */
    strcat(infix_exp,")");               /* add ')' to infix expression */

    i=0;
    j=0;
    item=infix_exp[i];        /* initialize before loop*/

    while(item != '\0')        /* run loop till end of infix expression */
    {
        if(item == '(')
        {
           push(item);
        }
        else if( isdigit(item) || isalpha(item))
        {
           postfix_exp[j] = item;            /* add operand symbol to postfix expr */
           j++;
        }
        else if(is_operator(item) == 1)       /* means symbol is operator */
        {
           x=pop();
           while(is_operator(x) == 1 && precedence(x)>= precedence(item))
           {
              postfix_exp[j] = x;               /* so pop all higher precendence operator and */
              j++;
              x = pop();                   /* add them to postfix expresion */
           }
           push(x);
           /* because just above while loop will terminate we have
           oppped one extra item
           for which condition fails and loop terminates, so that one*/
           push(item);              /* push current oprerator symbol onto stack */
        }
        else if(item == ')')        /* if current symbol is ')' then */
        {
           x = pop();               /* pop and keep popping until */
           while(x != '(')            /* '(' encounterd */
           {
              postfix_exp[j] = x;
              j++;
              x = pop();
           }
        }
        else
        { /* if current symbol is neither operand not '(' nor ')' and nor
           operator */
           printf("\nInvalid infix Expression.\n");       /* the it is illegeal  symbol */
           getchar();
           exit(1);
        }
        i++;

        item = infix_exp[i]; /* go to next symbol of infix expression */
    } /* while loop ends here */
    if(top>0)
    {
        printf("\nInvalid infix Expression.\n");       /* the it is illegeal  symbol */
        getchar();
        exit(1);
    }
    if(top>0)
    {
        printf("\nInvalid infix Expression.\n");       /* the it is illegeal  symbol */
```

```
        getchar();
        exit(1);
    }

    postfix_exp[j] = '\0'; /* add sentinel else puts() fucntion */
    /* will print entire postfix[] array upto SIZE */

}

/* main function begins */
int main()
{
    char infix[SIZE], postfix[SIZE];        /* declare infix string and postfix string */

    /* why we asked the user to enter infix expression
    * in parentheses ( )
    * What changes are required in porgram to
    * get rid of this restriction since it is not
    * in algorithm
    * */
    //printf("ASSUMPTION: The infix expression contains single letter variables and single digit
constants only.\n");
    printf("\nEnter Infix expression : ");
    gets(infix);

    InfixToPostfix(infix,postfix);                /* call to convert */
    printf("Postfix Expression: ");
    puts(postfix);                  /* print postfix expression */

    char quar[20][4];
    char str[SIZE];
    int i=0,q_i=0,j,temp=65;
    // printf("Enter the string:");
    // scanf("%s",&str[i]);
    //str=postfix;
    while(postfix[i]!='\0'){
        if(postfix[i]=='+'||postfix[i]=='-'||postfix[i]=='*'||postfix[i]=='/')
        {
            quar[q_i][0]=postfix[i];
            quar[q_i][1]=postfix[i-2];
            quar[q_i][2]=postfix[i-1];
            quar[q_i][3]=temp;

            postfix[i-2]=temp;
            temp++;
            q_i++;
            j=i; i=0; j++;
            while(postfix[j]!='\0'){
                postfix[j-2]=postfix[j]; j++;
            }
            postfix[j-2]='\0';
        }
        else
            i++;
    }
    printf("\nOPERATOR | OPERAND1 | OPERAND2 | RESULT \n");
    printf("==========================================\n");
    for(i=0;i<q_i;i++)
        {printf("%c       | %c       | %c       | %c \n",quar[i][0],quar[i][1],quar[i][2],quar[i][3]);
        printf("------------------------------------------\n");}
    return 0;
}
```

Output:

a+b*c/d-e



```
PS C:\Users\Hemit\Desktop\SEM 7\MY_DLP\practicals\PR10> cd

Enter Infix expression : a+b*c/d-e
Postfix Expression: abc*d/+e-

OPERATOR | OPERAND1 | OPERAND2 | RESULT
==========================================
*        | b        | c        | A
------------------------------------------
/        | A        | d        | B
------------------------------------------
+        | a        | B        | C
------------------------------------------
-        | C        | e        | D
------------------------------------------
```