

## EXPERIMENT 2

### IMPLEMENTATION OF LEXICAL ANALYZER USING LEX TOOL

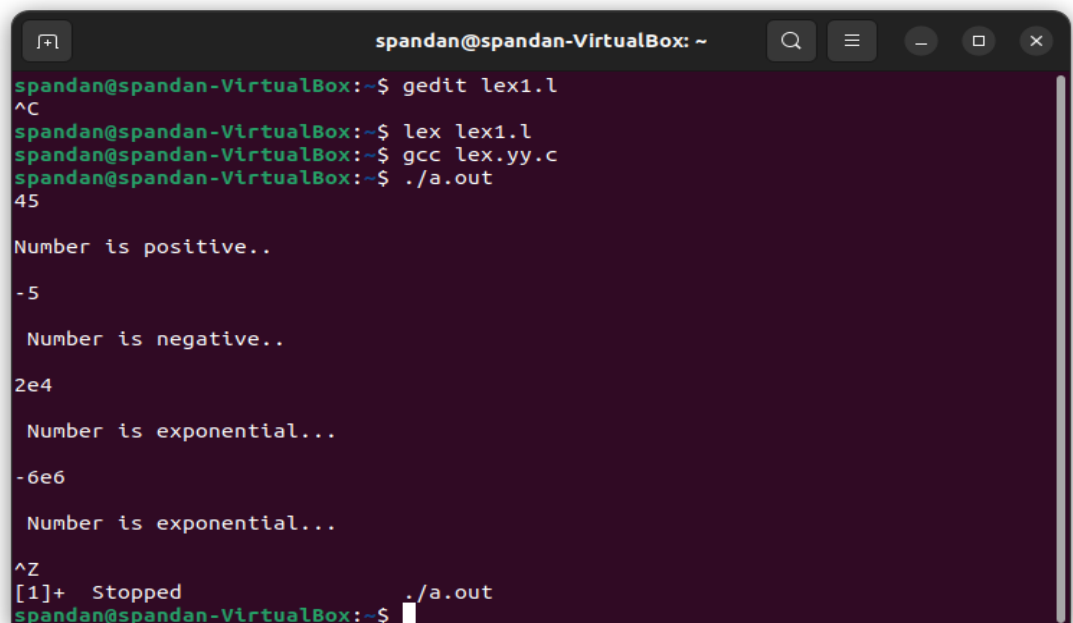
NAME: SPANDAN MUKHERJEE  
SUBJECT: COMPILER DESIGN

#### 1) PROGRAM TO RECOGNIZE INTEGER, REAL AND EXPONENTIAL

CODE:

```
% {  
    #include<stdio.h>  
% }  
  
sign[+-]?  
digit[0-9]+  
exp([eE]){sign}{digit})  
%%  
@printf("\n Enter the number:");  
\+?{digit} printf("\nNumber is positive..\n");  
\- {digit} printf("\n Number is negative..\n");  
{sign}{digit}?\.{digit}?printf("\n Number is real...\n");  
{sign}{digit}(\.{digit}?)?{exp} printf("\n Number is exponential..\n");  
%%  
int yywrap()  
{  
    return 1;  
}  
int main()  
{  
    yylex();  
}
```

OUTPUT



```
spandan@spandan-VirtualBox: ~  
spandan@spandan-VirtualBox:~$ gedit lex1.l  
^C  
spandan@spandan-VirtualBox:~$ lex lex1.l  
spandan@spandan-VirtualBox:~$ gcc lex.yy.c  
spandan@spandan-VirtualBox:~$ ./a.out  
45  
  
Number is positive..  
  
-5  
  
Number is negative..  
  
2e4  
  
Number is exponential...  
  
-6e6  
  
Number is exponential...  
  
^Z  
[1]+  Stopped                  ./a.out  
spandan@spandan-VirtualBox:~$
```

## 2) Program to recognize the Grammar $A^nB^n$


### CODE:

```
% {
#include <stdio.h>
int count_a = 0, count_b = 0;
% }

%%
A { count_a++; }
B { count_b++; }
\n {
    if (count_a == count_b) {
        printf("Accepted in Grammar\n");
    } else {
        printf("Rejected in Grammar\n");
    }
    count_a = count_b = 0;
}

%%
int yywrap()
{
    return 1;
}
int main(void) {
    yylex();
    return 0;
}
```

### OUTPUT



```
spandan@spandan-VirtualBox: ~
spandan@spandan-VirtualBox:~$ gedit lex2.l
^C
spandan@spandan-VirtualBox:~$ lex lex2.l
spandan@spandan-VirtualBox:~$ gcc lex.yy.c
spandan@spandan-VirtualBox:~$ ./a.out
AAABBB
Accepted in Grammar
ABBB
Rejected in Grammar
AABB
Accepted in Grammar
█
```

### 3) Program to recognize the Grammar $A^nB$

CODE:

```
%{
#include <stdio.h>
int count_a = 0, count_b = 0;
%}

%%
A { count_a++;}
B { count_b++; }
\n {
    if (count_b < count_a && count_b == 1) {
        printf("Accepted in Grammar\n");
    }
    else if (count_b == 1 && count_b == count_a) {
        printf("Accepted in Grammar\n");
    }
    else if (count_b == 1) {
        printf("Accepted in Grammar\n");
    }

    else {
        printf("Rejected in Grammar\n");
    }
    count_a = count_b = 0;
}

%%
int yywrap()
{
    return 1;
}
int main(void) {
    yylex();
    return 0;
}
```

OUTPUT

```
spandan@spandan-VirtualBox: ~  
spandan@spandan-VirtualBox:~$ lex lex3.l  
spandan@spandan-VirtualBox:~$ gcc lex.yy.c  
spandan@spandan-VirtualBox:~$ ./a.out  
A  
Rejected in Grammar  
B  
Accepted in Grammar  
ABBBB  
Rejected in Grammar  
AAAAAAAAAAB  
Accepted in Grammar  
AAAABB  
Rejected in Grammar  
^Z  
[1]+  Stopped                  ./a.out  
spandan@spandan-VirtualBox:~$
```

#### 4) C program to implement lexical analyzer using lex tool for a simple statement

CODE:

```
% {
#include<stdio.h>
% }

%%

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

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

"int"|"float"|"char"|"double"|"PI"|"define"|"if"|"else"|"else if"|"for"|"while"|"include" {
    printf("Reserved Keywords: %s\n", yytext);
}

[a-zA-Z_][a-zA-Z0-9_]* {
    printf("Identifier: %s\n", yytext);
}

"+"|"="|"-"|" "/"|"*"|"++"|"--"|"<=" {
    printf("Arithmetic Operator: %s\n", yytext);
}

"#|" "("|")"|" ">"|" "<"|" ";" {
    printf("Delimiter: %s\n", yytext);
}

[ \t\n]+ /* ignore white space */;

. {
    printf("INVALID CHARACTER: %s\n", yytext);
}

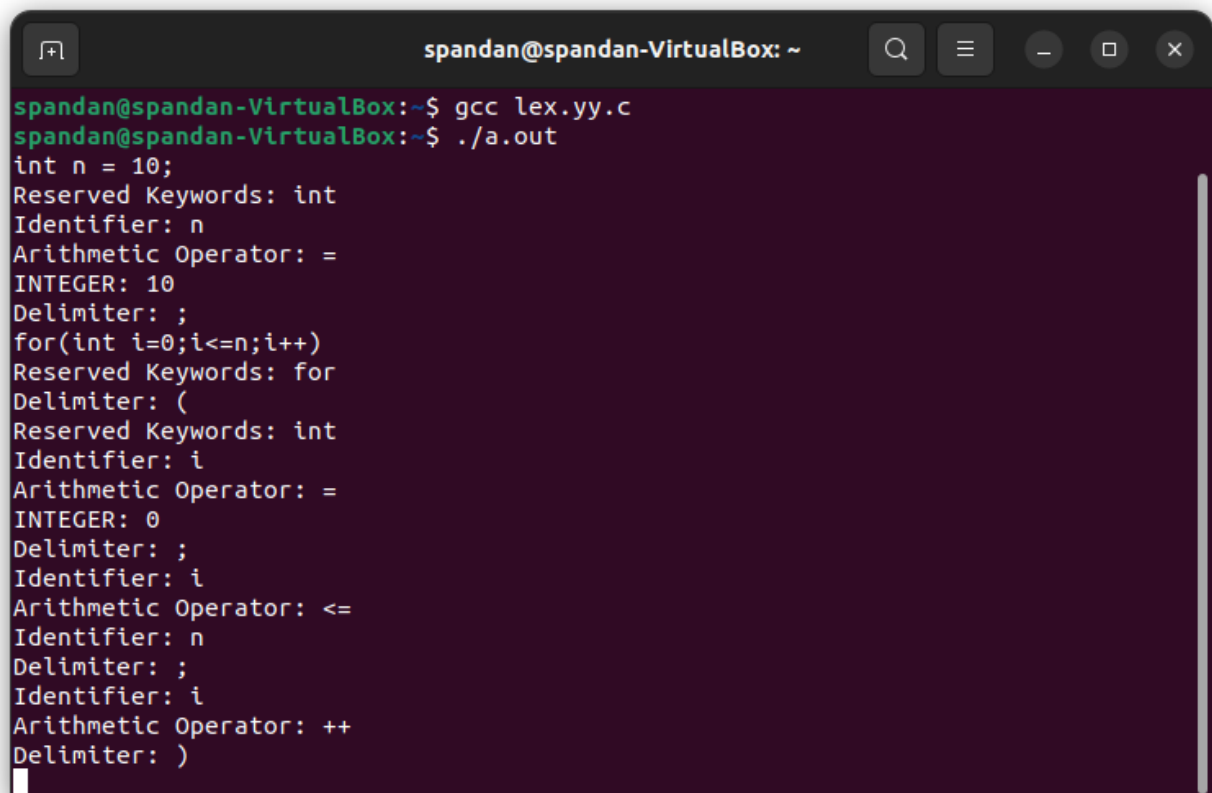
%%

int yywrap(){
    return 1;
}

int main(void) {
```

```
yylex();  
return 0;  
}
```

## OUTPUT



```
spandan@spandan-VirtualBox: ~  
spandan@spandan-VirtualBox:~$ gcc lex.yy.c  
spandan@spandan-VirtualBox:~$ ./a.out  
int n = 10;  
Reserved Keywords: int  
Identifier: n  
Arithmetic Operator: =  
INTEGER: 10  
Delimiter: ;  
for(int i=0;i<=n;i++)  
Reserved Keywords: for  
Delimiter: (  
Reserved Keywords: int  
Identifier: i  
Arithmetic Operator: =  
INTEGER: 0  
Delimiter: ;  
Identifier: i  
Arithmetic Operator: <=  
Identifier: n  
Delimiter: ;  
Identifier: i  
Arithmetic Operator: ++  
Delimiter: )
```

5) Design a compiler to do lexical analysis in c, c ++

CODE:

```
%{
#include<stdio.h>
%}

%%

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

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

"int"|"float"|"char"|"double"|"PI"|"define"|"if"|"else"|"else if"|"for"|"while"|"include" {
    printf("Reserved Keywords: %s\n", yytext);
}

[a-zA-Z_][a-zA-Z0-9_]* {
    printf("Identifier: %s\n", yytext);
}

"stdio.h"|"stdlib.h"|"conio.h" {
    printf("Header files: %s\n", yytext);
}

"#"|"("|")"|">"|"<" {
    printf("Delimiter: %s\n", yytext);
}

"+"|"="|"-"|"/"|"*"|"++"|"--" {
    printf("Arithmetic Operator: %s\n", yytext);
}

[ \t\n]+ /* ignore white space */;

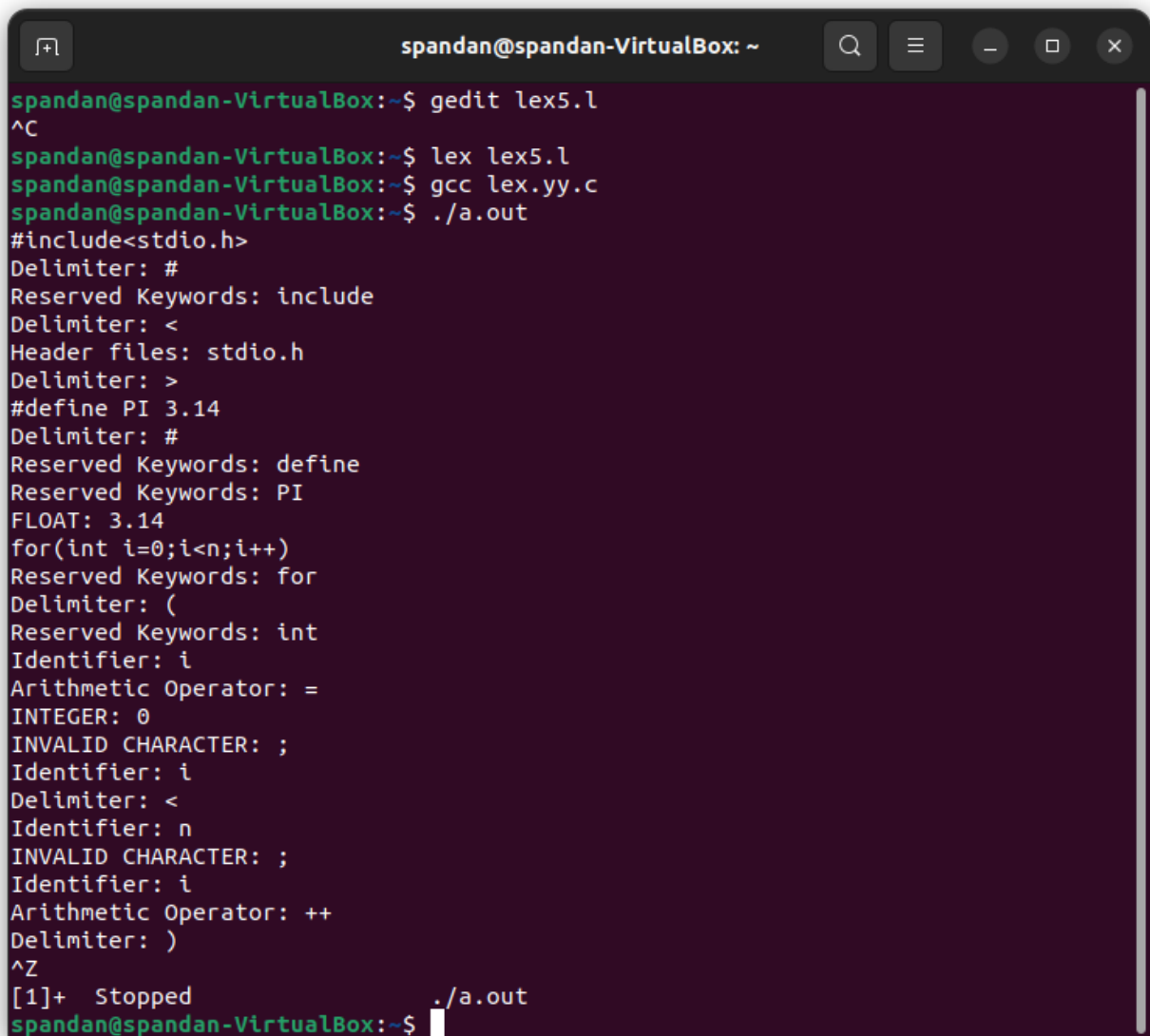
. {
    printf("INVALID CHARACTER: %s\n", yytext);
}

%%

int yywrap(){
    return 1;
}
```

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

## OUTPUT



```
spandan@spandan-VirtualBox: ~  
spandan@spandan-VirtualBox:~$ gedit lex5.l  
^C  
spandan@spandan-VirtualBox:~$ lex lex5.l  
spandan@spandan-VirtualBox:~$ gcc lex.yy.c  
spandan@spandan-VirtualBox:~$ ./a.out  
#include<stdio.h>  
Delimiter: #  
Reserved Keywords: include  
Delimiter: <  
Header files: stdio.h  
Delimiter: >  
#define PI 3.14  
Delimiter: #  
Reserved Keywords: define  
Reserved Keywords: PI  
FLOAT: 3.14  
for(int i=0;i<n;i++)  
Reserved Keywords: for  
Delimiter: (  
Reserved Keywords: int  
Identifier: i  
Arithmetic Operator: =  
INTEGER: 0  
INVALID CHARACTER: ;  
Identifier: i  
Delimiter: <  
Identifier: n  
INVALID CHARACTER: ;  
Identifier: i  
Arithmetic Operator: ++  
Delimiter: )  
^Z  
[1]+  Stopped  
spandan@spandan-VirtualBox:~$ ./a.out
```