EXPERIMENT 2
IMPLEMENTATION OF LEXICAL ANALYZER USING LEX TOOL

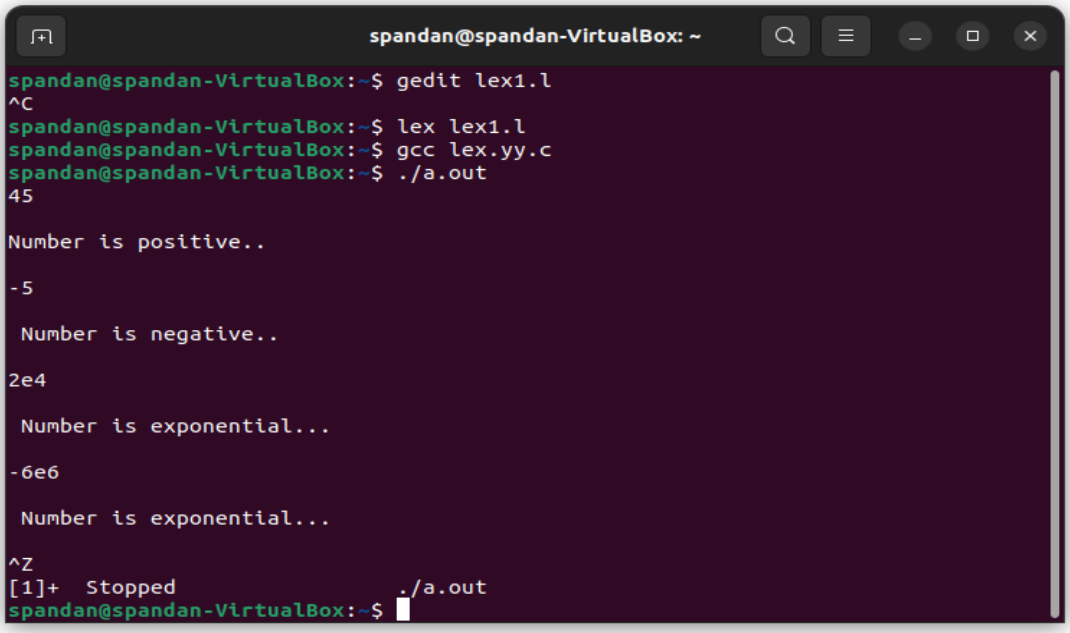
NAME: SPANDAN MUKHERJEE
SUBJECT: COMPILER DESIGN

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

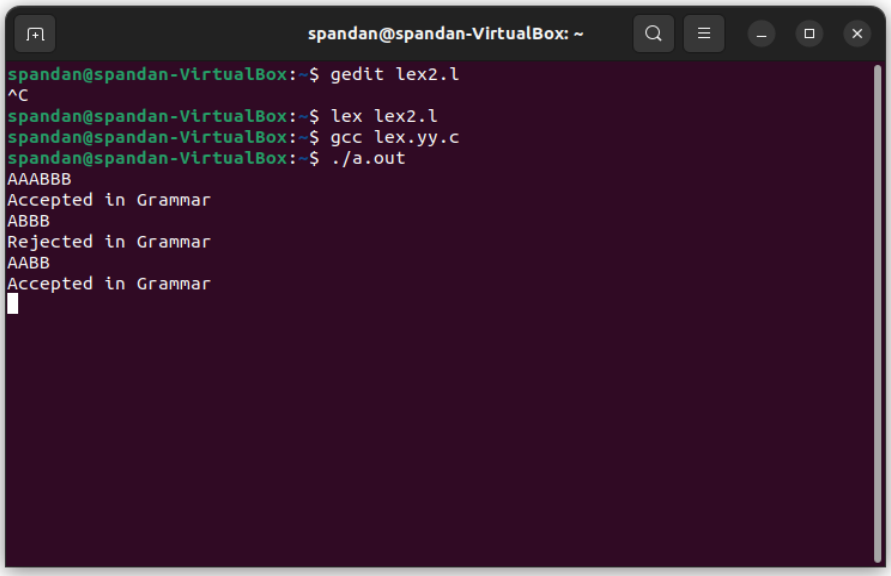


2) Program to recognize the Grammar A^nB^n

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

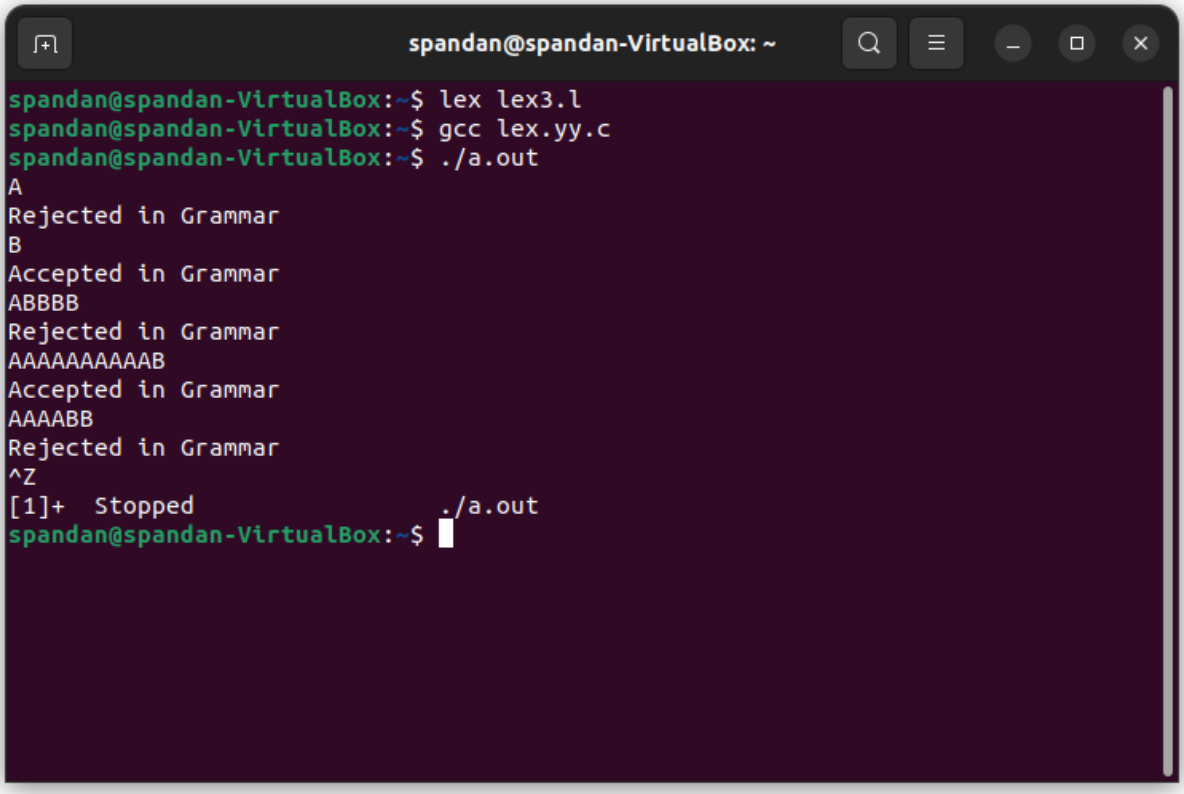


3) Program to recognize the Grammar A^nB

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:~$

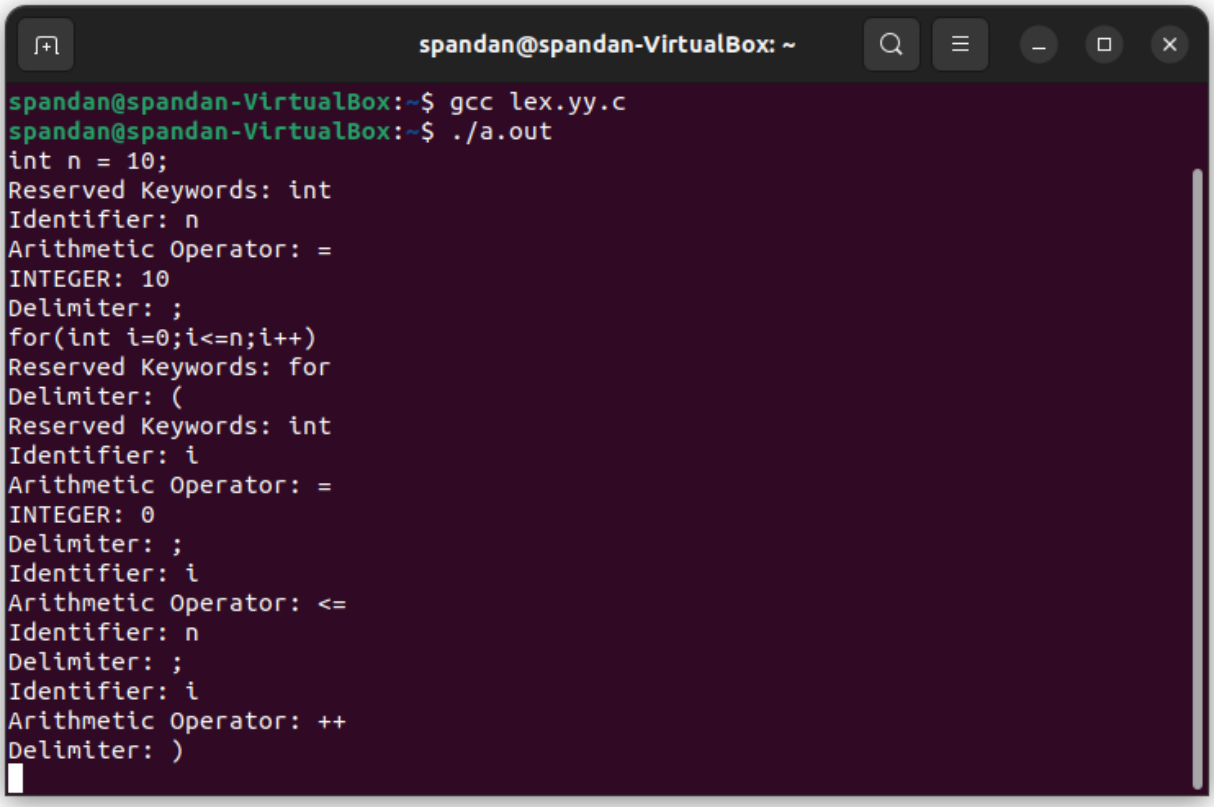
4) C program to implement lexical analyzer using lex tool for a simple statement

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



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