Experiment 1: Lexical analysis using lex tool

1.1) Aim: Write a lex program whose output is same as input.

Program:

```
%%
.ECHO;
%%
int yywarp(void)
{
return 1;
}
int main(void)
{
yylex();
return 0;
}
```

Output:

```
[22A95A0513@Linux ~]$ vi cd2b.l
[22A95A0513@Linux ~]$ vi cd1a.l
[22A95A0513@Linux ~]$ lex cd1a.l
[22A95A0513@Linux ~]$ gcc lex.yy.c -ll
[22A95A0513@Linux ~]$ ./a.out
Sravanthi
Sravanthi
cse
cse
22a95a0513
22a95a0513
Compiler Design
Compiler Design
Parser
Parser
```

1.2) <u>Aim:</u> Write a lex program which removes white spaces from its input file.

Program:

```
%%
[ ] {};
.ECHO;
%%
int yywrap(void){
return 1;
}
int main(void){
yylex();
return 0;
}
```

Output:

```
[22A95A0513@Linux ~]$ vi cd1b.l
[22A95A0513@Linux ~]$ lex cd1b.l
[22A95A0513@Linux ~]$ gcc lex.yy.c -ll
[22A95A0513@Linux ~]$ ./a.out
h e l l o
hello
he ll o
hello
Good M o rn in g ( “  “ )
GoodMorning(“”)
He l l    o   W orl d   “
HelloWorld”
“ - -        -- ”
“----“
```

Experiment 2: Lexical analysis using lex tool

2.1) Aim: To write a Lex program to identify the patterns in the input file.

Program:

```
%{
#include<stdio.h>
%}
%%
["int""char""for""if""while""then""return""do"] {printf("keyword: %s\n");}
[*%\+-] {printf("operator: %s\n", yytext);}
[(){};] {printf("special character: %s\n", yytext);}
[0-9]+ {printf("constant: %s\n", yytext);}
[a-zA-Z_][a-zA-Z0-9_]* {printf("valid identifier is : %s\n", yytext);}
^[^a-zA-Z_] {printf("invalid identifier \n");}
%%
```

Output:

```
[22A95A0513@Linux ~]$ vi cd2a.l
[22A95A0513@Linux ~]$ lex cd2a.l
[22A95A0513@Linux ~]$ gcc lex.yy.c -ll
[22A95A0513@Linux ~]$ ./a.out<cd1b.l
operator: %operator: %
invalid identifier
 ] special char: {
special char: }
special char: ;

invalid identifier
valid identifier is:ECHO
special char: ;

operator: %operator: %
valid identifier  is:int
valid identifier is:yywrap
special char: (
valid identifier is:void
special char: )
special char: {

valid identifier is:return
 constant: 1
special char: ;

special char: }

valid identifier is:int
valid identifier is:main
special char: (
valid identifier is:void
```

```
special char: )
special char: {

valid identifier is:yylex
special char: (
special char: )
special char: ;

valid identifier is:return
 constant: 0
special char: ;

special char: }
```

2.2) **Aim:** To Design a lexical analyzer for given language and the lexical analyzer should ignore redundant spaces, tabs and new lines.

Program:

```
%{
#include<stdio.h>
int i=0, id=0;
%}
%%
[#].*[<].*[>]\n {}
[ \t\n]+ {}
\/\/.*\n {}
\/\*(.*\n)*.*\*\/ {}
auto|break|case|char|onst|continue|default|do|double|else|enum|extern|float|for|goto|if|int|long|registe
r|return|short|signed|sizeof|static|struct|switch|typedef|union|unsigned|void|volatile|while
{printf("token: %d <keyword , %s >\n",++i,yytext);}
[+\-\*\/%<>] {printf("token: %d <operator , %s >\n",++i,yytext);}
[();{}] {printf("token: %d <special char , %s > \n",++i,yytext);}
[0-9]+ {printf("token: %d < constant , %s >\n",++i,yytext);}
[a-zA-Z_][a-zA-Z0-9_]* {printf("token: %d <Id%d ,%s >\n",++i,++id,yytext);}
^[a^a-zA-Z_] {printf("Error invalid token %s\n",yytext);}
%%
```

Output:

```
[22A95A0513@Linux ~]$ vi cd2b.l
[22A95A0513@Linux ~]$ lex cd2b.l
[22A95A0513@Linux ~]$ gcc lex.yy.c -ll
[22A95A0513@Linux ~]$ ./a.out<hello.c
token: 1 <keyword, void >
token: 2 <Idl,main >
token: 3 <special char
token: 4 <special char
token: 5 <special char,
token: 6 <Id2,printf >
token: 7 <special char, (>
"token: 8 <Id3, GOOD >
token: 9 <Id4,MORNING >
"token: 10 <special char, ) >
token: 11 <special char, ; >
token: 12 <special char, } >
```