CD Project – Report 1

Team members:

15CO239 – R. RuhiTaj

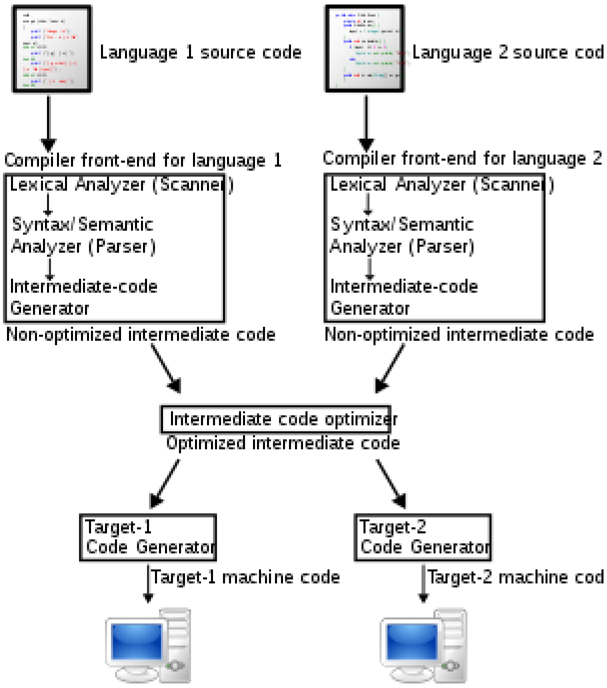
15CO253—Y. M. Greeshma

Language used: C language.

-What is a **COMPILER?**

A compiler is a **COMPUTER SOFTWARE PROGRAM.**

They are a type of TRANSLATOR that supports digital devices, primarily computers.



Language 1 source code
Language 2 source cod
Compiler front-end for language 1
Lexical Analyzer (Scanner)
Syntax/Semantic Analyzer (Parser)
Intermediate-code Generator
Non-optimized intermediate code
Compiler front-end for language 2
Lexical Analyzer (Scanner)
Syntax/Semantic Analyzer (Parser)
Intermediate-code Generator
Non-optimized intermediate code
Intermediate code optimizer
Optimized intermediate code
Target-1 Code Generator
Target-1 machine code
Target-2 Code Generator
Target-2 machine cod

A diagram of the operation of a typical multi language, multi target compiler

-What are **COMPILERS** *used for?*

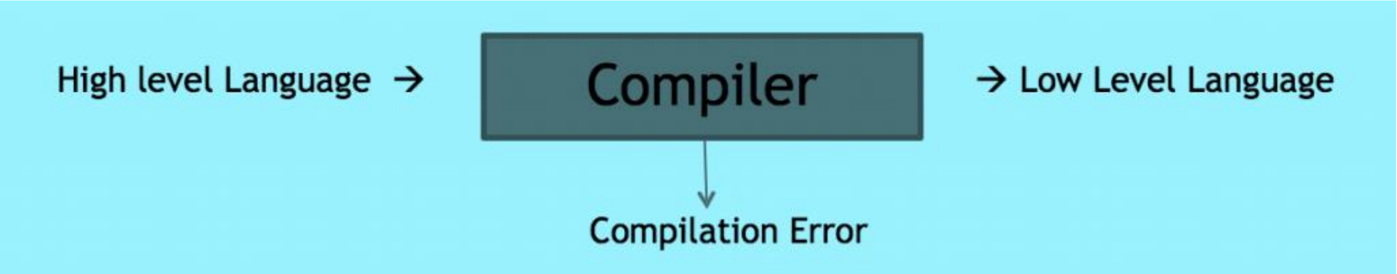
A Compiler transforms code written in one programming language (**Source Code)** to another language (**Target Code).**

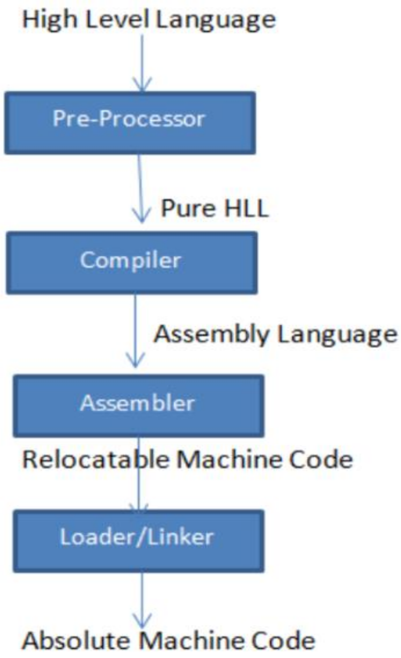
The term - Compiler is primarily used for programs that translate source code from a high-level programming language to a lower level language (e.g., assembly language, object code, or machine code) to create an executable program.

-***What does a* COMPILER *do?***

A compiler performs the following operations: pre-processing, lexical analysis , parsing, semantic analysis (syntax-directed translation) and conversion of input programs to an intermediate representation, code optimization and code generation.

-General Description of a COMPILER:





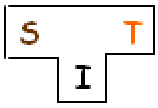
We know a computer is a logical assembly of Software and Hardware. The hardware knows a language that is hard for us to grasp, consequently we tend to write programs in high-level language that is much less complicated for us to comprehend and maintain in thoughts. Now these programs go through a series of transformation so that they can readily be used machines. This is where language procedure systems come handy.

Here are three kinds of compilers you should learn to identify:

- Source language = Host language ⇒ **Self compiler**
- Target language = Host language ⇒ **Self-resident compiler**
- Target language ≠ Host language ⇒ **Cross compiler**

Cross Compiler that runs on a machine ‘A’ and produces a code for another machine ‘B’. It is capable of creating code for a platform other than the one on which the compiler is running. It is characterized by three languages: the source language(S), the target language(T) and the implementation language(I).

- This is represented by a T-diagram as:



- In textual form this can be represented as

$S_{I}T$

Source-to-source Compiler or Transcompiler or transpiler is a compiler that translates source code written in one programming language into source code of another programming language.

-*Phases of a* COMPILER:

Lexical Analysis:

The first phase of scanner works as a text scanner. This phase scans the source code as a stream of characters and converts it into meaningful lexemes.

It converts the input program into a sequence of **Tokens**.

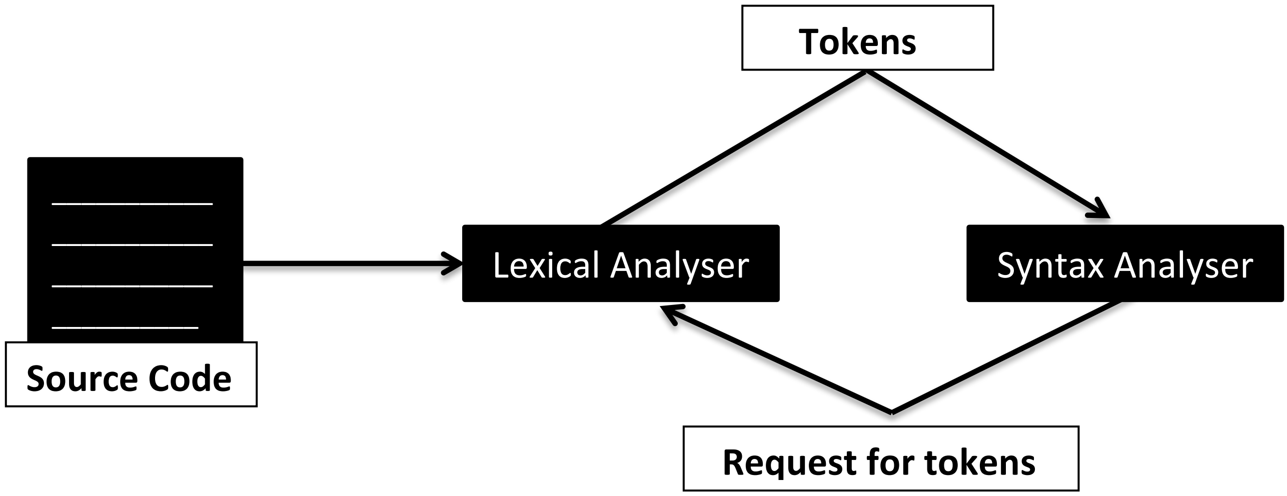
Example of tokens:

- Type token (id, number, real, . . .)
- Punctuation tokens (IF, void, return, . . .)
- Alphabetic tokens (keywords)

Function of Lexical Analyser:

1. Dividing the program into tokens (*Tokenization).*
2. Remove white space characters.
3. Remove comments.
4. Provide error message by providing row number and column number.

The lexical analyser identifies the error with the help of automation machine and the grammar of the given language on which it is based like C, C++.



Need of Lexical Analyser:

1) *Simplicity of design of compiler*

The removal of white spaces and comments enables the syntax analyzer for efficient syntactic constructs.

2) *Compiler efficiency is improved*

Specialized buffering techniques for reading characters speed up the compiler process.

3) *Compiler portability is enhanced*.

Issues in Lexical Analysis:

Lexical analysis is the process of producing tokens from the source program. It has the following issues:

- Look ahead
- Ambiguities

Syntax Analysis:

Syntax Analyser takes the token produced by lexical analysis as input and generates a parse tree.

Semantic Analysis:

Semantic analysis checks whether the parse tree constructed follows the rules of language.

Intermediate Code Generator:

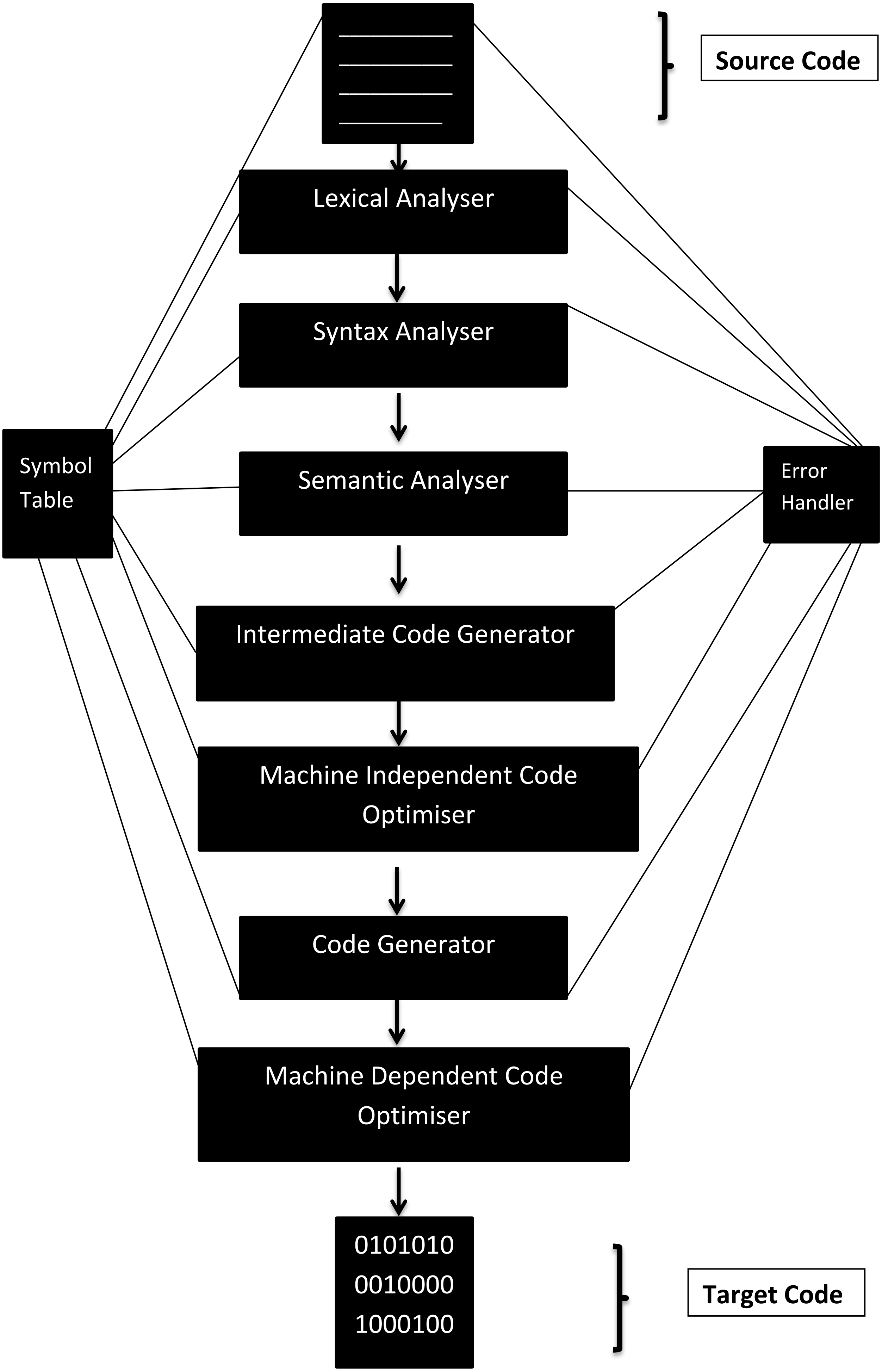
The compiler generates an intermediate code of the source code for the target machine.

Machine Independent Code Optimiser:

Removal of unnecessary code lines, and arranges the sequence of statements in order to speed up.

Code Generator:

Code generator takes the optimized representation of the intermediate code and maps it to the target machine language.

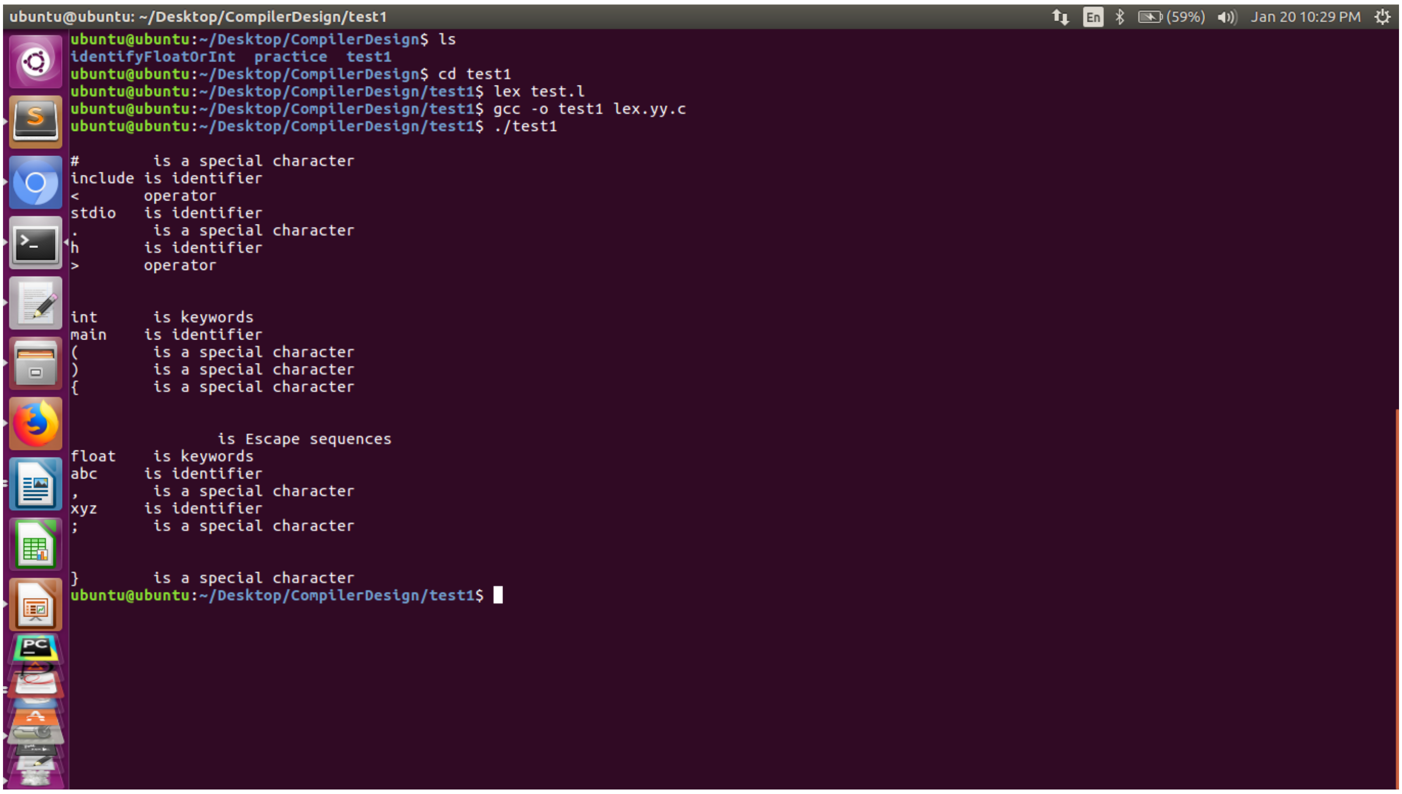

Source Code
Lexical Analyser
Syntax Analyser
Semantic Analyser
Intermediate Code Generator
Machine Independent Code Optimiser
Code Generator
Machine Dependent Code Optimiser
0101010
0010000
1000100
Target Code
Symbol Table
Error Handler

Test Case 1:

```
# include <stdio.h>

int main(){

 float abc,xyz;}
```

Lex Program:

```
%{
        %}
        letter[a-zA-Z]
        digit[0-9]
        %%
        {digit}+("E"("+"|"-")?{digit}+)? printf("\n%s\tis real number",yytext);
        {digit}+"."{digit}+("E"("+"|"-")?{digit}+)? printf("\n%s\t is floating pt no ",yytext);
        "if"|"else"|"int"|"char"|"scanf"|"printf"|"switch"|"return"|"struct"|"do"|"while"|"void"|"
        for"|"float" printf("\n%s\t is keywords",yytext);
        "\a"|"\\n"|"\\b"|"\t"|"\\t"|"\b"|"\\a" printf("\n%s\tis Escape sequences",yytext);
        {letter}({letter}|{digit})* printf("\n%s\tis identifier",yytext);
        "&&"|"<"|">"|"<="|">="|"="|"+"|"-"|"?"|"*"|"/"|"%"|"&"|"||" printf("\n%s\toperator
        ",yytext);
        "{"|"}"|"["|"]"|"("|")"|"#"|"'"|"."|"\""|"\\"|";"|"," printf("\n%s\t is a special
        character",yytext);
        "%d"|"%s"|"%c"|"%f"|"%e" printf("\n%s\tis a format specifier",yytext);
        %%

        int yywrap()
         {
         return 1;
         }

        int main()
        {
        yyin=fopen("cdTestCase1.c","r");
        yylex();
        fclose(yyin);
        return 0;
        }
```



```
ubuntu@ubuntu:~/Desktop/CompilerDesign$ ls
identifyFloatOrInt  practice  test1
ubuntu@ubuntu:~/Desktop/CompilerDesign$ cd test1
ubuntu@ubuntu:~/Desktop/CompilerDesign/test1$ lex test.l
ubuntu@ubuntu:~/Desktop/CompilerDesign/test1$ gcc -o test1 lex.yy.c
ubuntu@ubuntu:~/Desktop/CompilerDesign/test1$ ./test1

#        is a special character
include is identifier
<       operator
stdio   is identifier
.        is a special character
h       is identifier
>       operator


int      is keywords
main    is identifier
(        is a special character
)        is a special character
{        is a special character


                is Escape sequences
float    is keywords
abc     is identifier
,        is a special character
xyz     is identifier
;        is a special character


}        is a special character
ubuntu@ubuntu:~/Desktop/CompilerDesign/test1$ 
```

Test Case 2:

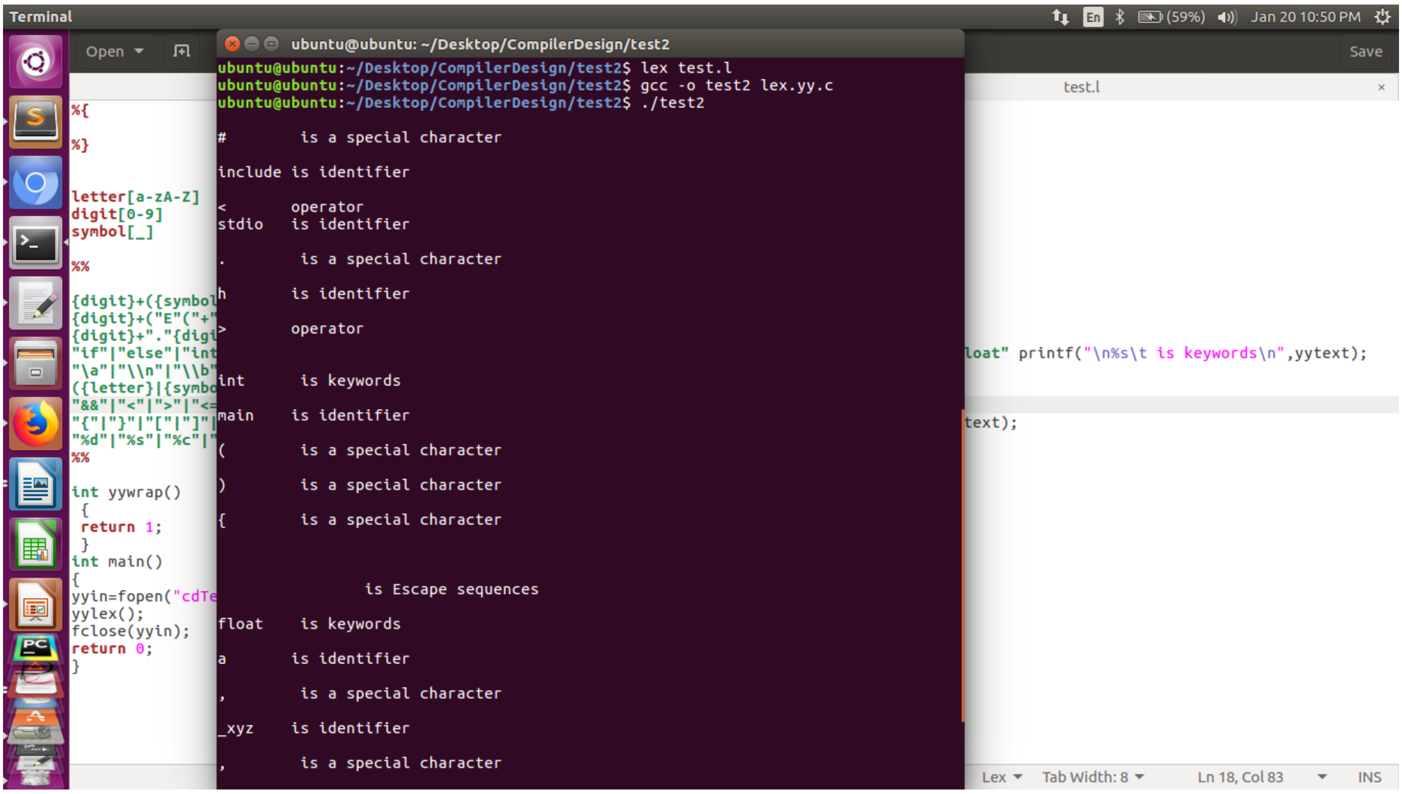
# include <stdio.h>

int main(){float a, _xyz, 8_abc;}

Lex Program:

```
%{

%}



letter[a-zA-Z]

digit[0-9]

symbol[_]



%%

{digit}+({symbol}|{letter}|{digit})+ printf("\n%s\tis not an identifier\n",yytext);



{digit}+("E"("+"|"-")?{digit}+)? printf("\n%s\tis real number\n",yytext);
```

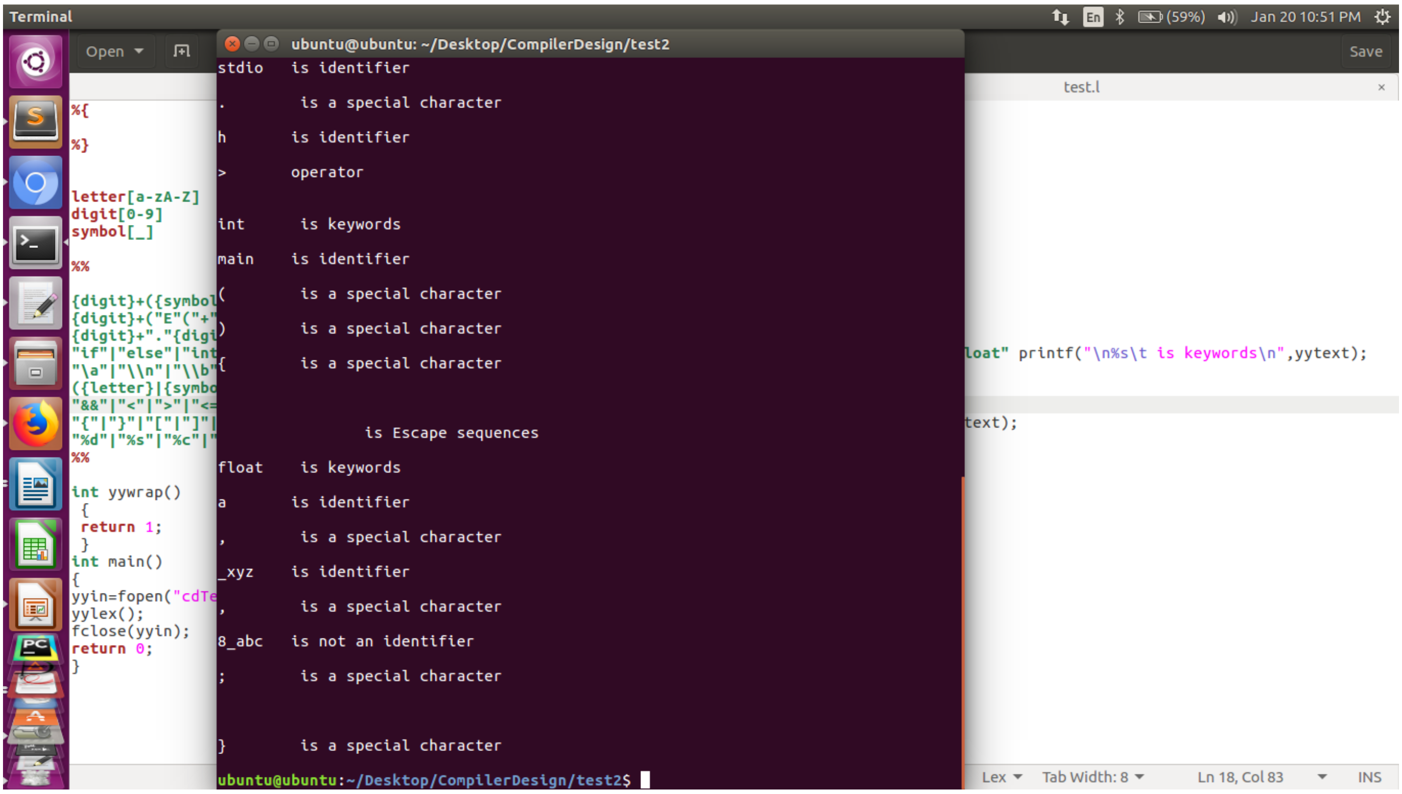
```
{digit}+"."{digit}+("E"("+"|"-")?{digit}+)? printf("\n%s\t is floating pt no\n",yytext);


"if"|"else"|"int"|"char"|"scanf"|"printf"|"switch"|"return"|"struct"|"do"|"while"|"void"|"for"|"float" printf("\n%s\t is keywords\n",yytext);


"\a"|"\\n"|"\\b"|"\t"|"\\t"|"\b"|"\\a" printf("\n%s\tis Escape sequences\n",yytext);

({letter}|{symbol})({letter}|{digit})* printf("\n%s\tis identifier\n",yytext);


"&&"|"<"|">"|"<="|">="|"="|"+"|"-"|"?"|"*"|"/"|"%"|"&"|"||" printf("\n%s\toperator",yytext);

"{"|"}"|"["|"]"|"("|")"|"#"|"'"|"."|"\""|"\\"|";"|"," printf("\n%s\t is a special character\n",yytext);

"%d"|"%s"|"%c"|"%f"|"%e" printf("\n%s\tis a format specifier\n",yytext);

%%


int yywrap()

 {

 return 1;

 }


int main()

{

yyin=fopen("cdTestCase2.c","r");

yylex();

fclose(yyin);

return 0;

}
```



```
%{

%}

letter[a-zA-Z]
digit[0-9]
symbol[_]

%%

{digit}+({symbol
{digit}+("E"("+"
{digit}+"."{digi
"if"|"else"|"int
"\a"|"\\n"|"\\b"
({letter}|{symbo
"&&"|"<"|">"|"<=
"{"|"}"|"["|"]"|
"%d"|"%s"|"%c"|"
%%

int yywrap()
 {
 return 1;
 }
int main()
{
yyin=fopen("cdTe
yylex();
fclose(yyin);
return 0;
}
```

```
loat" printf("\n%s\t is keywords\n",yytext);
text);
```

```
ubuntu@ubuntu: ~/Desktop/CompilerDesign/test2
ubuntu@ubuntu:~/Desktop/CompilerDesign/test2$ lex test.l
ubuntu@ubuntu:~/Desktop/CompilerDesign/test2$ gcc -o test2 lex.yy.c
ubuntu@ubuntu:~/Desktop/CompilerDesign/test2$ ./test2

#        is a special character

include is identifier

<       operator
stdio   is identifier

.        is a special character

h       is identifier

>       operator

int      is keywords

main    is identifier

(        is a special character

)        is a special character

{        is a special character


                is Escape sequences

float    is keywords

a       is identifier

,        is a special character

_xyz    is identifier

,        is a special character
```



```
ubuntu@ubuntu: ~/Desktop/CompilerDesign/test2
stdio   is identifier

.        is a special character

h       is identifier

>       operator

int      is keywords

main    is identifier

(        is a special character

)        is a special character

{        is a special character


                is Escape sequences

float    is keywords

a       is identifier

,        is a special character

_xyz    is identifier

,        is a special character

8_abc   is not an identifier

;        is a special character


}        is a special character
ubuntu@ubuntu:~/Desktop/CompilerDesign/test2$ 
```

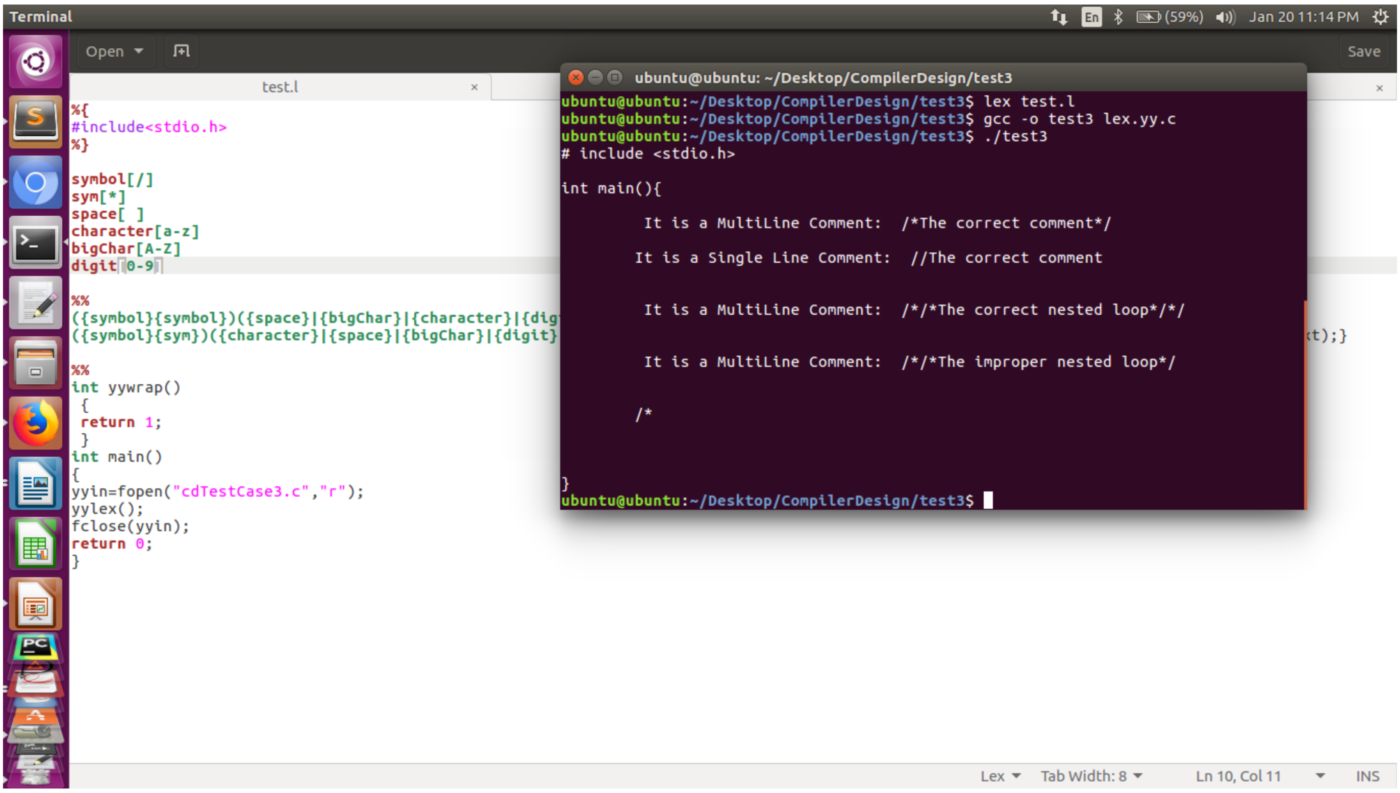
Test Case 3:

```
# include <stdio.h>
int main(){
/*The correct comment*/
//The correct comment
/*/*The correct nested loop*/*/
/*/*The improper nested loop*/
/*
}
```

Lex Program:

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

symbol[/]
sym[*]
space[ ]
character[a-z]
bigChar[A-Z]
digit[0-9]

%%
({symbol}{symbol})({space}|{bigChar}|{character}|{digit})* {printf("It is a Single Line Comment: %s\n",yytext);}
({symbol}{sym})({character}|{space}|{bigChar}|{digit}|{symbol}|{sym})*({sym}{symbol}) {printf(" It is a MultiLine Comment:  %s\n",yytext);}
```

```
%%
int yywrap()
 {
 return 1;
 }
int main()
{
yyin=fopen("cdTestCase3.c","r");
yylex();
fclose(yyin);
return 0;
}
```



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

symbol[/]
sym[*]
space[ ]
character[a-z]
bigChar[A-Z]
digit[0-9]

%%
({symbol}{symbol})({space}|{bigChar}|{character}|{dig
({symbol}{sym})({character}|{space}|{bigChar}|{digit}
t);}

%%
int yywrap()
 {
 return 1;
 }
int main()
{
yyin=fopen("cdTestCase3.c","r");
yylex();
fclose(yyin);
return 0;
}
```

ubuntu@ubuntu: ~/Desktop/CompilerDesign/test3

```
ubuntu@ubuntu:~/Desktop/CompilerDesign/test3$ lex test.l
ubuntu@ubuntu:~/Desktop/CompilerDesign/test3$ gcc -o test3 lex.yy.c
ubuntu@ubuntu:~/Desktop/CompilerDesign/test3$ ./test3
# include <stdio.h>

int main(){

         It is a MultiLine Comment:  /*The correct comment*/

        It is a Single Line Comment:  //The correct comment


         It is a MultiLine Comment:  /*/*The correct nested loop*/*/


         It is a MultiLine Comment:  /*/*The improper nested loop*/


        /*



}
ubuntu@ubuntu:~/Desktop/CompilerDesign/test3$ 
```

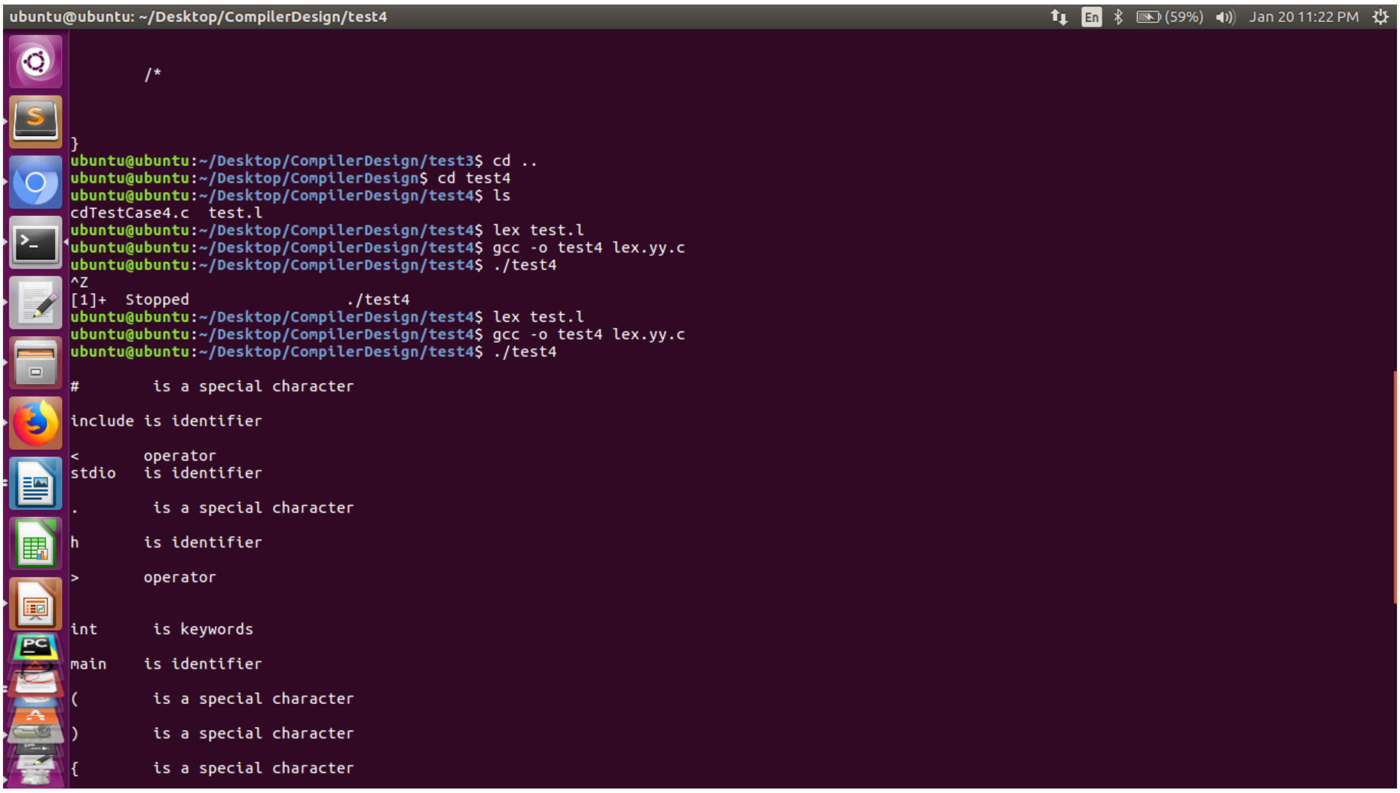
Test Case 4:

```
# include <stdio.h>

int main(){

float abc;

int xyz;

abc = xyz + 4;

}
```

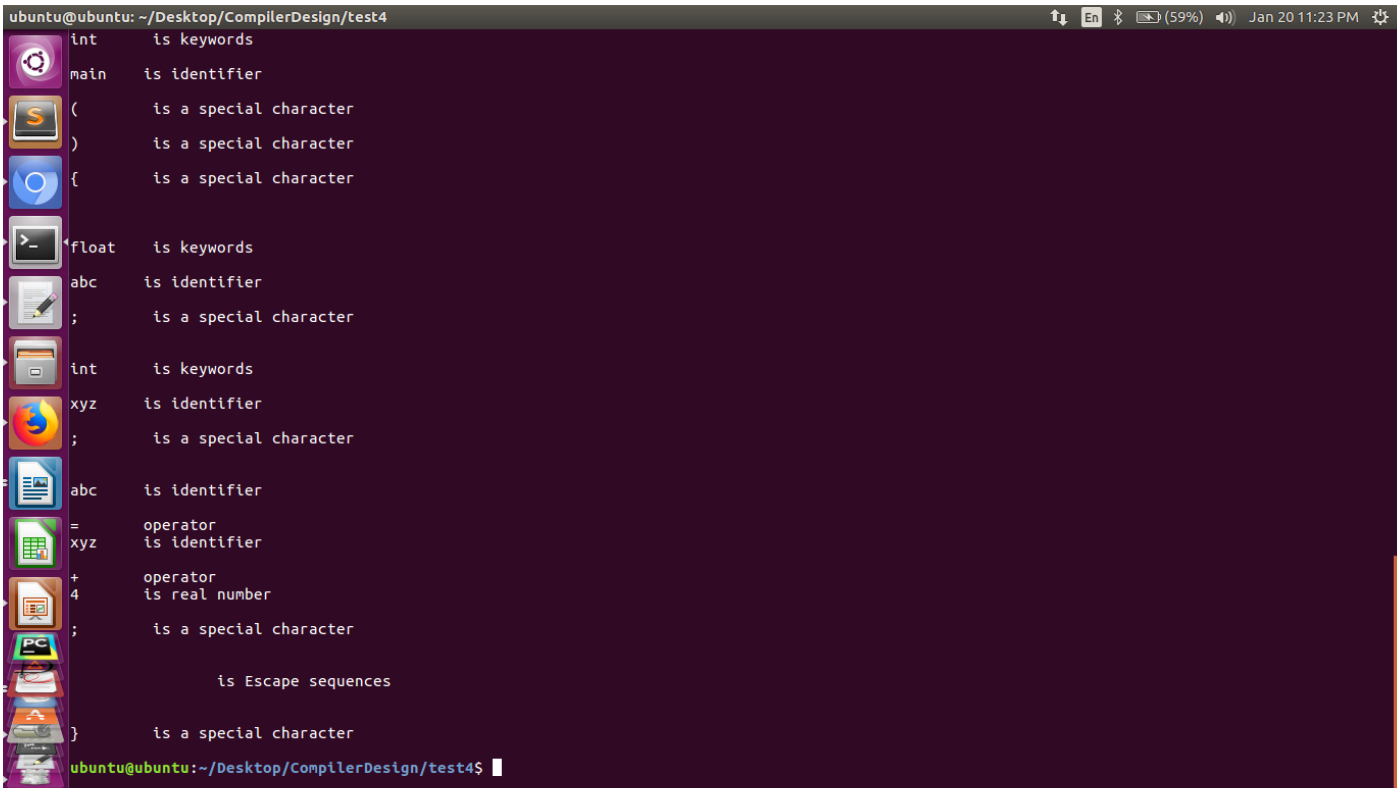
Lex Program:

```
%{


%}

letter[a-zA-Z]

digit[0-9]

symbol[_]

%%


{digit}+({symbol}|{letter}|{digit})+ printf("\n%s\tis not an identifier\n",yytext);

{digit}+("E"("+"|"-")?{digit}+)? printf("\n%s\tis real number\n",yytext);

{digit}+"."{digit}+("E"("+"|"-")?{digit}+)? printf("\n%s\t is floating pt no\n",yytext);

"if"|"else"|"int"|"char"|"scanf"|"printf"|"switch"|"return"|"struct"|"do"|"while"|"void"|"for"|"float" printf("\n%s\t is keywords\n",yytext);

"\a"|"\\n"|"\\b"|"\t"|"\\t"|"\b"|"\\a" printf("\n%s\tis Escape sequences\n",yytext);

({letter}|{symbol})({letter}|{digit})* printf("\n%s\tis identifier\n",yytext);

"&&"|"<"|">"|"<="|">="|"="|"+"|"-"|"?"|"*"|"/"|"%"|"&"|"||" printf("\n%s\toperator",yytext);

"{"|"}"|"["|"]"|"("|")"|"#"|"'"|"."|"\""|"\\"|";"|"," printf("\n%s\t is a special character\n",yytext);

"%d"|"%s"|"%c"|"%f"|"%e" printf("\n%s\tis a format specifier\n",yytext);

%%
```

```
int yywrap()

 {

 return 1;

 }

int main()

{

yyin=fopen("cdTestCase4.c","r");

yylex();

fclose(yyin);

return 0;

}
```



```
        /*



}
ubuntu@ubuntu:~/Desktop/CompilerDesign/test3$ cd ..
ubuntu@ubuntu:~/Desktop/CompilerDesign$ cd test4
ubuntu@ubuntu:~/Desktop/CompilerDesign/test4$ ls
cdTestCase4.c  test.l
ubuntu@ubuntu:~/Desktop/CompilerDesign/test4$ lex test.l
ubuntu@ubuntu:~/Desktop/CompilerDesign/test4$ gcc -o test4 lex.yy.c
ubuntu@ubuntu:~/Desktop/CompilerDesign/test4$ ./test4
^Z
[1]+  Stopped                 ./test4
ubuntu@ubuntu:~/Desktop/CompilerDesign/test4$ lex test.l
ubuntu@ubuntu:~/Desktop/CompilerDesign/test4$ gcc -o test4 lex.yy.c
ubuntu@ubuntu:~/Desktop/CompilerDesign/test4$ ./test4

#        is a special character

include is identifier

<       operator
stdio   is identifier

.        is a special character

h       is identifier

>       operator


int      is keywords

main    is identifier

(        is a special character

)        is a special character

{        is a special character
```



```
int      is keywords
main    is identifier
(        is a special character
)        is a special character
{        is a special character

float    is keywords
abc     is identifier
;        is a special character
int      is keywords
xyz     is identifier
;        is a special character
abc     is identifier
=       operator
xyz     is identifier
+       operator
4       is real number
;        is a special character
                is Escape sequences
}        is a special character
ubuntu@ubuntu:~/Desktop/CompilerDesign/test4$ 
```

Test Case 5:

# include <stdio.h>

int main(){

float x = 0.34;

printf("Value : %f\n",x);

}

Lex Program:

%{

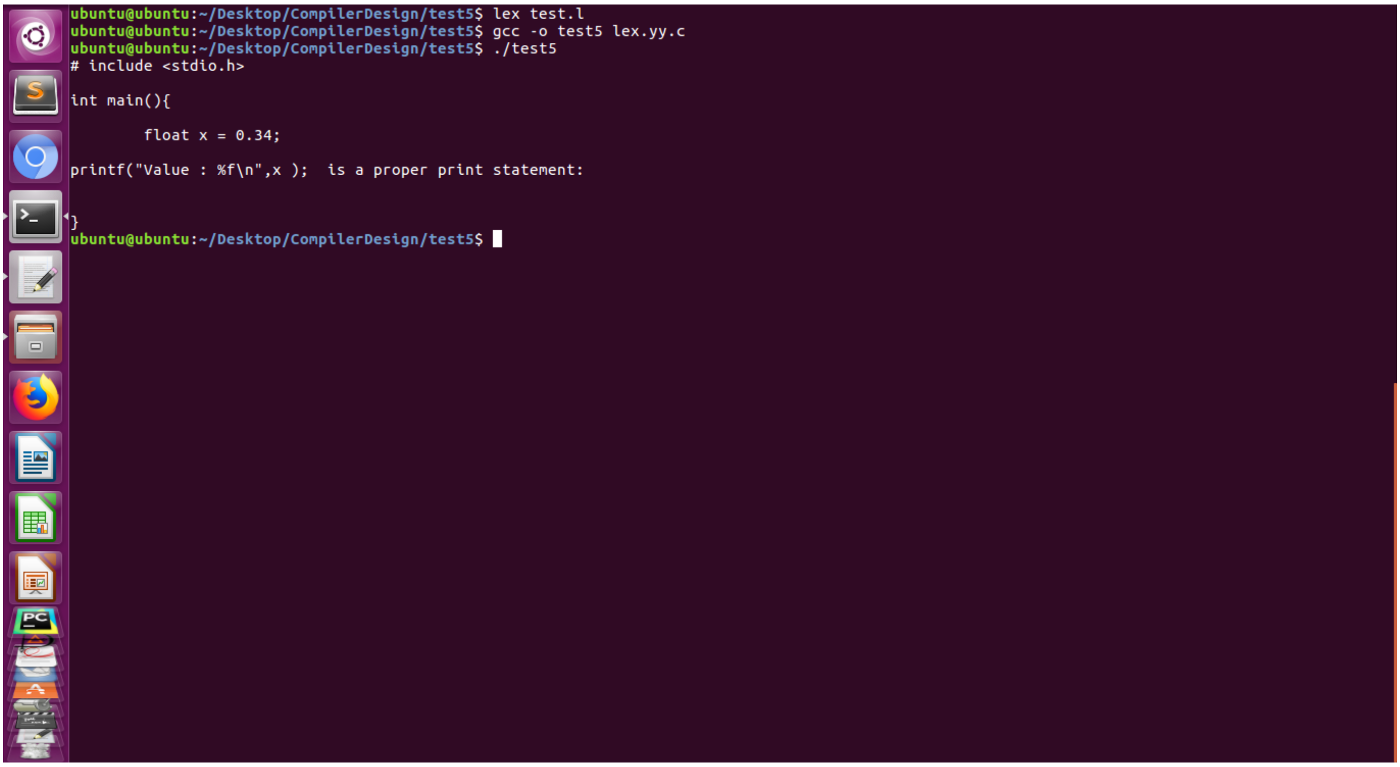
 # include <stdio.h>

 # include <stdlib.h>

%}

letter[a-zA-Z]

```
digit[0-9]
symbol[_]
qoute["]
openBracket[(]
comma[,]
closeBracket[)]
end[;]
perc[%]
dotted[:]
space[ ]
nextLine [\n]
%%
"printf"{openBracket}{qoute}({space}|{letter}|{digit}|{perc}|{dotted}|\\)*{qoute}({space}*{comma}{space}*({digit}*|{letter}*){space}*)?{closeBracket}{end}  {printf("\n%s  is a proper print statement:  \n",yytext);}
%%
int yywrap()
 {
 return 1;
 }
int main()
{
yyin=fopen("cdTestCase5.c","r");
yylex();
fclose(yyin);
return 0;
}
```



```
ubuntu@ubuntu:~/Desktop/CompilerDesign/test5$ lex test.l
ubuntu@ubuntu:~/Desktop/CompilerDesign/test5$ gcc -o test5 lex.yy.c
ubuntu@ubuntu:~/Desktop/CompilerDesign/test5$ ./test5
# include <stdio.h>

int main(){

        float x = 0.34;

printf("Value : %f\n",x );  is a proper print statement:


}
ubuntu@ubuntu:~/Desktop/CompilerDesign/test5$ 
```