COMPILER DESIGN: PHASE 1 REPORT

Abstract:

The following will be handled by the compiler to be designed by us:

- Char data types
- Functions
- For and while loops
- Nested while loops
- If-else constructs

Compiler

A **compiler** is computer software that transforms computer code written in one programming language (the source language) into another programming language (the target language). Compilers are a type of translator that support digital devices, primarily computers. The name *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.

Splitting a compiler up into small programs is a technique used by researchers interested in producing provably correct compilers. Proving the correctness of a set of small programs often requires less effort than proving the correctness of a larger, single, equivalent program.

We basically have two phases of compilers, namely Analysis phase and Synthesis phase. Analysis phase creates an intermediate representation from the given source code. Synthesis phase creates an equivalent target program from the intermediate representation.

The compiler has two modules namely front end and back end. Front-end constitutes of the Lexical analyzer, semantic analyzer, syntax analyzer and intermediate code generator. And the rest are assembled to form the back end.

1. **Lexical Analyzer –** It reads the program and converts it into tokens. It converts a stream of lexemes into a stream of tokens. Tokens are defined by regular expressions which are understood by the lexical analyzer. It also removes white-spaces and comments.

2. **Syntax Analyzer –** It is sometimes called as parser. It constructs the parse tree. It takes all the tokens one by one and uses Context Free Grammar to construct the parse tree.
   The rules of programming can be entirely represented in some few productions. Using these productions we can represent what the program actually is. The input has to be checked whether it is in the desired format or not.
   Syntax error can be detected at this level if the input is not in accordance with the grammar.

3. **Semantic Analyzer –** It verifies the parse tree, whether it's meaningful or not. It furthermore produces a verified parse tree.

4. **Intermediate Code Generator –** It generates intermediate code that is a form which can be readily executed by machine. We have many popular intermediate codes. Example – Three address code etc. Intermediate code is converted to machine language using the last two phases which are platform dependent.
   Till intermediate code, it is same for every compiler out there, but after that, it depends on the platform. To build a new compiler we don't need to build it from scratch. We can take the intermediate code from the already existing compiler and build the last two parts.

5. **Code Optimizer –** It transforms the code so that it consumes fewer resources and produces more speed. The meaning of the code being transformed is not altered. Optimization can be categorized into two types: machine dependent and machine independent.

6. **Target Code Generator –** The main purpose of Target Code generator is to write a code that the machine can understand. The output is dependent on the type of assembler. This is the final stage of compilation.

Common terminologies encountered

Symbol Table – It is a data structure being used and maintained by the compiler, consists all the identifier's name along with their types. It helps the compiler to function smoothly by finding the identifiers quickly.

Lexical Analysis

Lexical analysis or **tokenization** is the process of converting a sequence of characters into a sequence of tokens (strings with an assigned and thus identified meaning). A program that performs lexical analysis may be termed a *lexer*, *tokenizer,* or *scanner*. A lexer is generally combined with a parser, which together analyze the syntax of programming languages.

Lexeme – A *lexeme* is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token.

Token – A *lexical token* or simply *token* is a pair consisting of a *token name* and an optional *token value*. The token name is a category of lexical unit. Common token names are

- Identifiers: names the programmer chooses;
- Keywords: names already in the programming language;
- Separators (also known as punctuators): punctuation characters and paired-delimiters;
- Operators: symbols that operate on arguments and produce results;
- Literals: numeric, logical, textual, reference literals;
- Comments: line, block.

All tokens recognized are then stored in the **symbol table** except the literals. The literals are stored separately in **constant table**. Constant tables is similar to symbol table in terms of its structure and use.

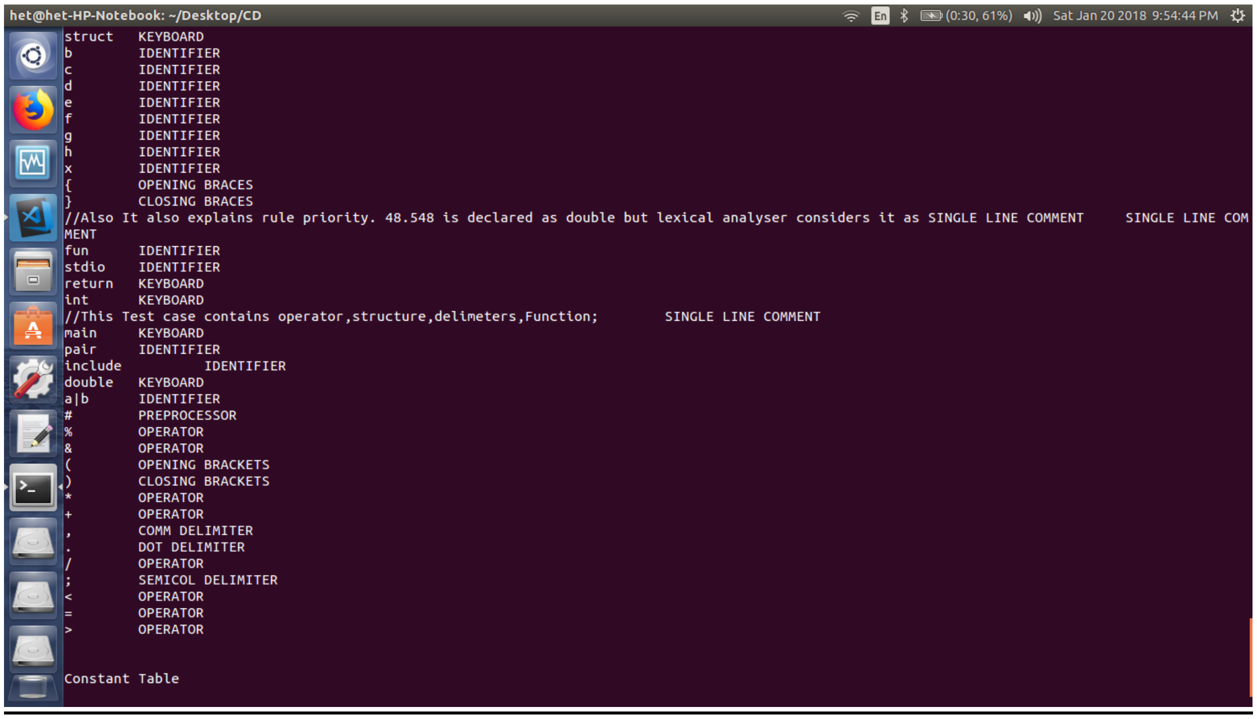
Identifying looping contraints(for and while), if contraints, functions, arrays and structures will be handled in phase 2 of compiler design, i.e. by syntax analyzer (parser).

Code

```
%{

    #include<stdio.h>
    #include<string.h>

    struct hashtable{
        char name[105];
        char type[105];
        int len;
    }table[1005];

    struct consttable{
        char name[105];
        char type[105];
        int len;
    }ctable[1005];


    int Hash(char *s){
        int mod=1001;
        int l=strlen(s),val=0,i;
        for (i=0;i<l;i++){
            val=val*10+(s[i]-'A');
            val=val%mod;
            while(val<0){
                val+=mod;
            }
        }
        return val;
    }

    void insert(char *arg1,char *arg2){

        int l1=strlen(arg1);
        int l2=strlen(arg2);
        int v=Hash(arg1);
        if(table[v].len==0){
            strcpy(table[v].name,arg1);
            strcpy(table[v].type,arg2);

            table[v].len=strlen(arg1);
            return ;
        }

        if(strcmp(table[v].name,arg1)==0)
        return ;

        int i,pos=0;

        for (i=0;i<1001;i++){
            if(table[i].len==0){
                pos=i;
                break;
            }
        }

        strcpy(table[pos].name,arg1);
        strcpy(table[pos].type,arg2);
        table[pos].len=strlen(arg1);


    }

    void insert2(char *arg1,char *arg2){

        int l1=strlen(arg1);
        int l2=strlen(arg2);
        int v=Hash(arg1);
        if(ctable[v].len==0){
            strcpy(ctable[v].name,arg1);
            strcpy(ctable[v].type,arg2);

            ctable[v].len=strlen(arg1);
            return ;
        }

        if(strcmp(ctable[v].name,arg1)==0)
        return ;

        int i,pos=0;

        for (i=0;i<1001;i++){
            if(ctable[i].len==0){
                pos=i;
                break;
            }
        }

        strcpy(ctable[pos].name,arg1);
        strcpy(ctable[pos].type,arg2);
        ctable[pos].len=strlen(arg1);
    }

    void print(){
        int i;

        for ( i=0;i<1001;i++){
            if(table[i].len==0){
                continue;
            }

            printf("%s \t %s\n",table[i].name,table[i].type);
        }
    }

    void print2(){
        int i;

        for ( i=0;i<1001;i++){
            if(ctable[i].len==0){
                continue;
            }

            printf("%s \t %s\n",ctable[i].name,ctable[i].type);
        }
    }

%}

LEQ <=
GEQ >=
EQ =
LES <
GRE >
PLUS \+
MINUS \-
MULT \*
DIV \/
REM %
AND &
OR \|

%%
[ \n\t] ;
\".*\"|\'.*\' {printf("%s \t- STRING CONSTANT\n", yytext);
insert2(yytext,"STRING CONSTANT");}
; {printf("%s \t- SEMICOL DELIMITER\n", yytext); insert(yytext, "SEMICOL
DELIMITER");}
, {printf("%s \t- COMM DELIMITER\n", yytext); insert(yytext, "COMM
DELIMITER");}
\{ {printf("%s \t- OPENING BRACES\n", yytext); insert(yytext, "OPENING
BRACES");}
\} {printf("%s \t- CLOSING BRACES\n", yytext); insert(yytext, "CLOSING
BRACES");}
\( {printf("%s \t- OPENING BRACKETS\n", yytext); insert(yytext, "OPENING
BRACKETS");}
\) {printf("%s \t- CLOSING BRACKETS\n", yytext); insert(yytext, "CLOSING
BRACKETS");}
# {printf("%s \t- PREPROCESSOR\n", yytext); insert(yytext, "PREPROCESSOR");}
printf {printf("%s \t- PRINT\n", yytext); insert(yytext, "PRINT");}
\" {printf("%s \t- DQOUTE\n", yytext); insert(yytext, "DQOUTE");}
\' {printf("%s \t- SQOUTE\n", yytext); insert(yytext, "SQOUTE");}
\\ {printf("%s \t- FSLASH\n", yytext); insert(yytext, "FSLASH");}
\. {printf("%s \t- DOT DELIMITER\n", yytext); insert(yytext, "DOT
DELIMITER");}
\/\/.* {printf("%s \t- SINGLE LINE COMMENT\n", yytext); insert(yytext,
"SINGLE LINE COMMENT");}

"/*"([^*]|\*+[^*/])*\*+"/" {printf("%s \t- MULTI LINE COMMENT\n", yytext);
insert(yytext, "MULTI LINE COMMENT");}

auto|break|case|char|const|continue|default|do|double|else|enum|extern|float|
for|goto|if|int|long|register|return|short|signed|sizeof|static|struct|switch
|typedef|union|unsigned|void|volatile|while|main {printf("%s \t- KEYWORD\n",
yytext); insert(yytext, "KEYBOARD");}
```

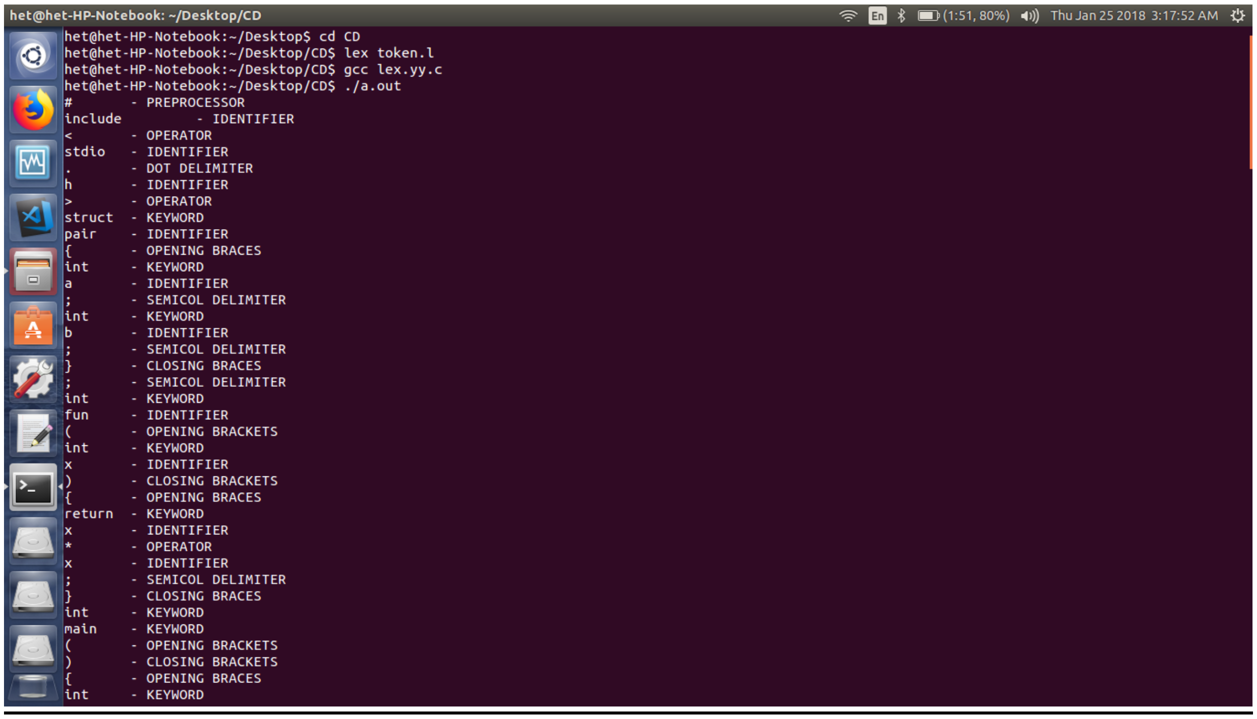
```
[a-z|A-Z]([a-z|A-Z]|[0-9])* {printf("%s \t- IDENTIFIER\n", yytext);
insert(yytext, "IDENTIFIER");}

[1-9][0-9]*|0 {printf("%s \t- NUMBER CONSTANT\n", yytext); insert2(yytext,
"NUMBER CONSTANT");}

{PLUS}|{MINUS}|{MULT}|{DIV}|{EQ}|{LEQ}|{GEQ}|{LES}|{GRE}|{REM}|{AND}|{OR}
{printf("%s \t- OPERATOR\n", yytext); insert(yytext, "OPERATOR");}

(.?) {printf("%s \t- INVALID\n", yytext); insert(yytext, "INVALID");}
%%

int main(){

    int i;
    for (i=0;i<1001;i++){
        table[i].len=0;
    }
    yyin=fopen("test-1.c","r");
    yylex();
    printf("\n\nSYMBOL TABLE\n\n");

    print();

    printf("\n\nCONSTANT TABLE\n\n");
    print2();
}

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

Screenshots



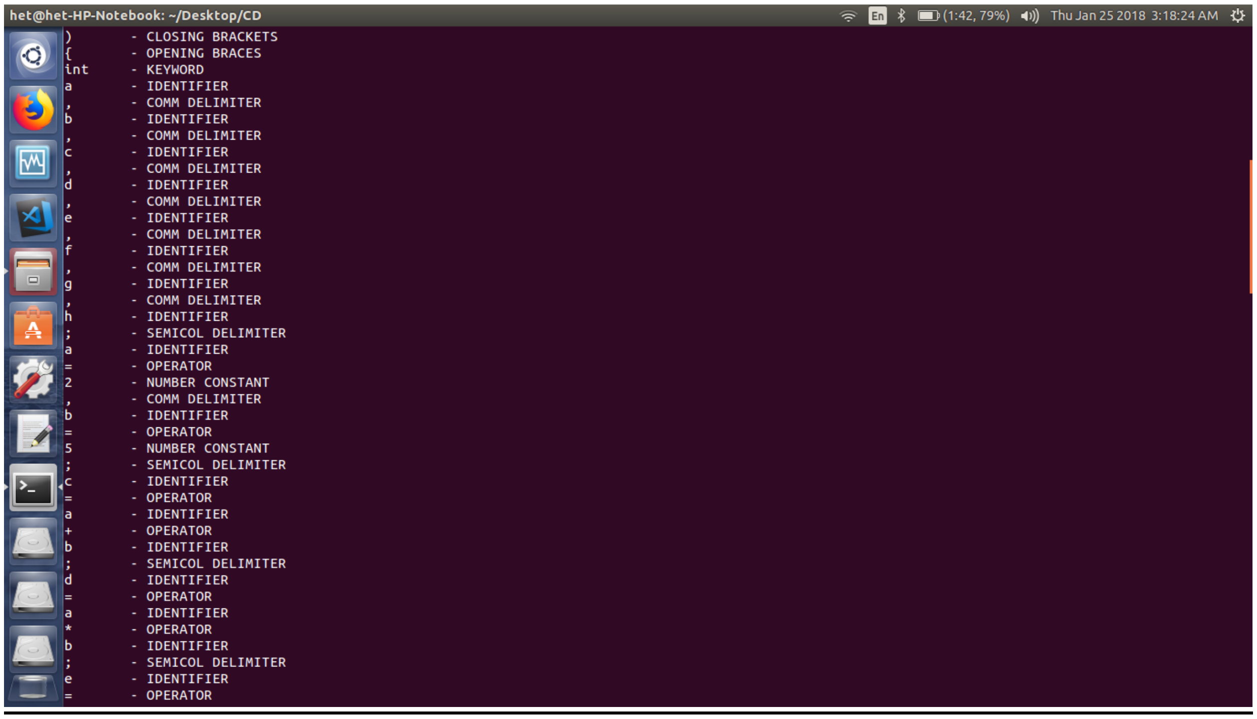
het@het-HP-Notebook: ~/Desktop/CD   En (0:30, 61%) Sat Jan 20 2018 9:54:44 PM

```
struct   KEYBOARD
b        IDENTIFIER
c        IDENTIFIER
d        IDENTIFIER
e        IDENTIFIER
f        IDENTIFIER
g        IDENTIFIER
h        IDENTIFIER
x        IDENTIFIER
{        OPENING BRACES
}        CLOSING BRACES
//Also It also explains rule priority. 48.548 is declared as double but lexical analyser considers it as SINGLE LINE COMMENT     SINGLE LINE COM
MENT
fun      IDENTIFIER
stdio    IDENTIFIER
return   KEYBOARD
int      KEYBOARD
//This Test case contains operator,structure,delimeters,Function;        SINGLE LINE COMMENT
main     KEYBOARD
pair     IDENTIFIER
include          IDENTIFIER
double   KEYBOARD
a|b      IDENTIFIER
#        PREPROCESSOR
%        OPERATOR
&        OPERATOR
(        OPENING BRACKETS
)        CLOSING BRACKETS
*        OPERATOR
+        OPERATOR
,        COMM DELIMITER
.        DOT DELIMITER
/        OPERATOR
;        SEMICOL DELIMITER
<        OPERATOR
=        OPERATOR
>        OPERATOR


Constant Table
```

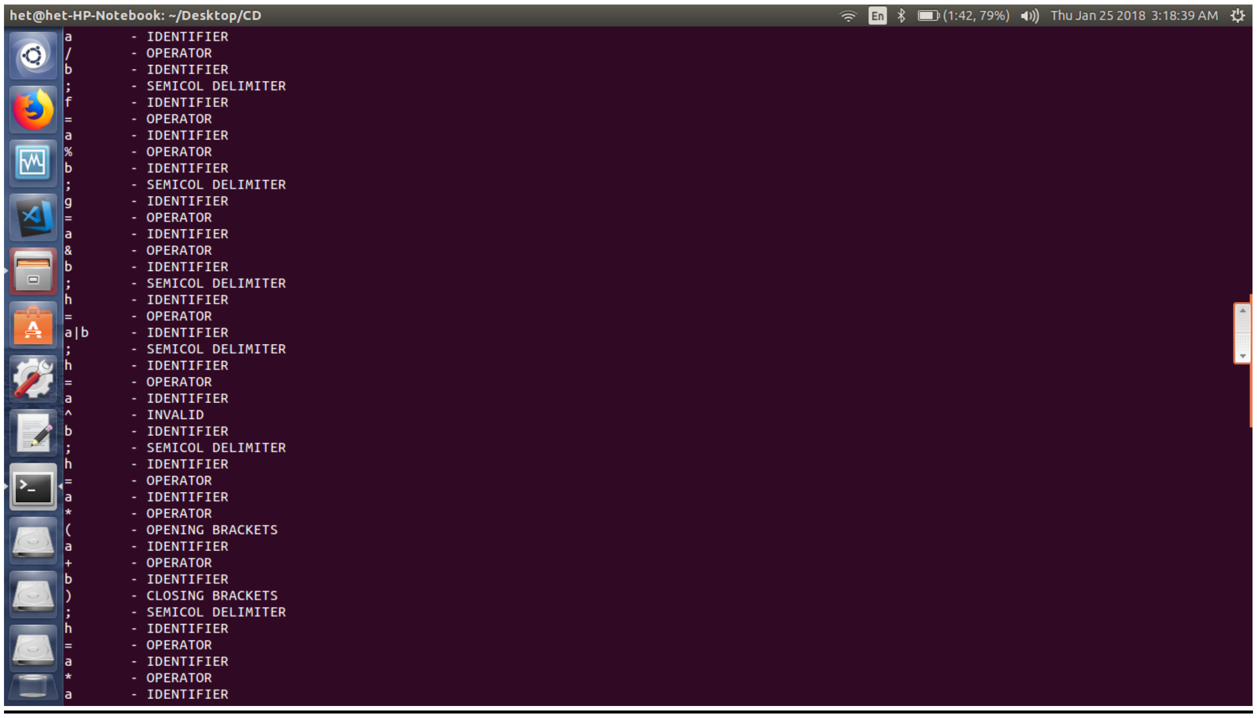


het@het-HP-Notebook: ~/Desktop/CD   En (1:51, 80%) Thu Jan 25 2018 3:17:52 AM

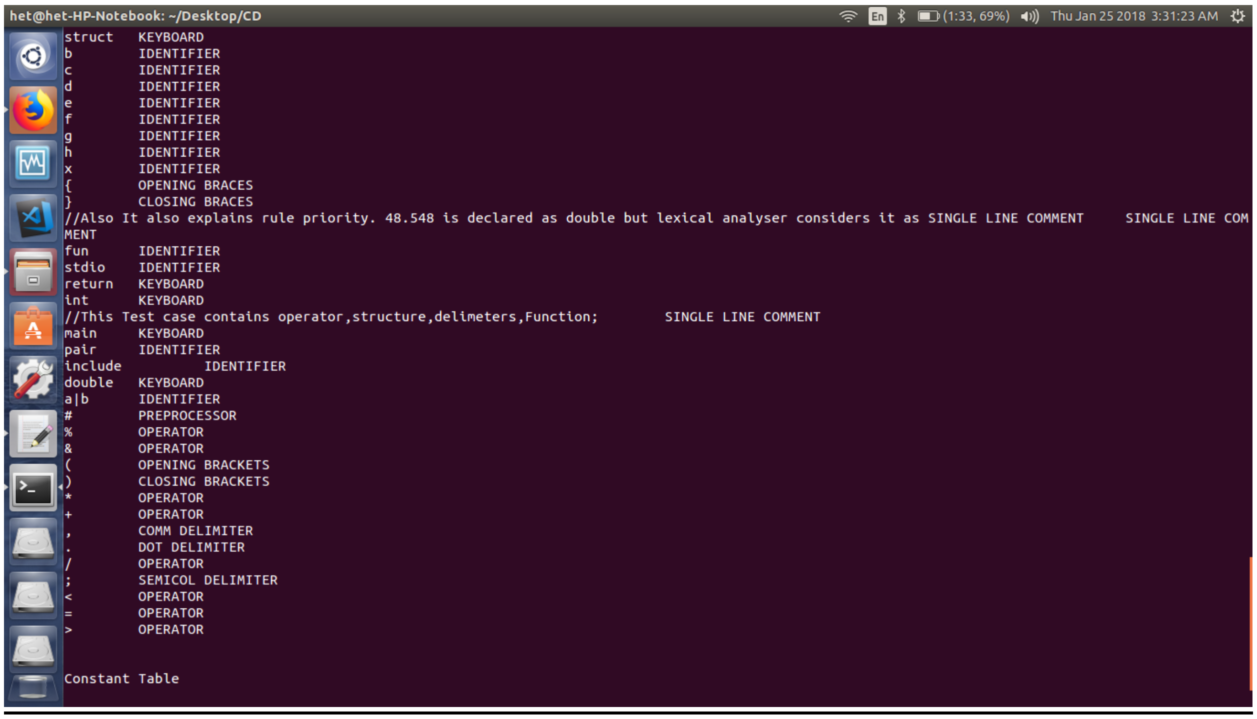
```
het@het-HP-Notebook:~/Desktop$ cd CD
het@het-HP-Notebook:~/Desktop/CD$ lex token.l
het@het-HP-Notebook:~/Desktop/CD$ gcc lex.yy.c
het@het-HP-Notebook:~/Desktop/CD$ ./a.out
#        - PREPROCESSOR
include          - IDENTIFIER
<        - OPERATOR
stdio    - IDENTIFIER
.        - DOT DELIMITER
h        - IDENTIFIER
>        - OPERATOR
struct   - KEYWORD
pair     - IDENTIFIER
{        - OPENING BRACES
int      - KEYWORD
a        - IDENTIFIER
;        - SEMICOL DELIMITER
int      - KEYWORD
b        - IDENTIFIER
;        - SEMICOL DELIMITER
}        - CLOSING BRACES
;        - SEMICOL DELIMITER
int      - KEYWORD
fun      - IDENTIFIER
(        - OPENING BRACKETS
int      - KEYWORD
x        - IDENTIFIER
)        - CLOSING BRACKETS
{        - OPENING BRACES
return   - KEYWORD
x        - IDENTIFIER
*        - OPERATOR
x        - IDENTIFIER
;        - SEMICOL DELIMITER
}        - CLOSING BRACES
int      - KEYWORD
main     - KEYWORD
(        - OPENING BRACKETS
)        - CLOSING BRACKETS
{        - OPENING BRACES
int      - KEYWORD
```


het@het-HP-Notebook: ~/Desktop/CD

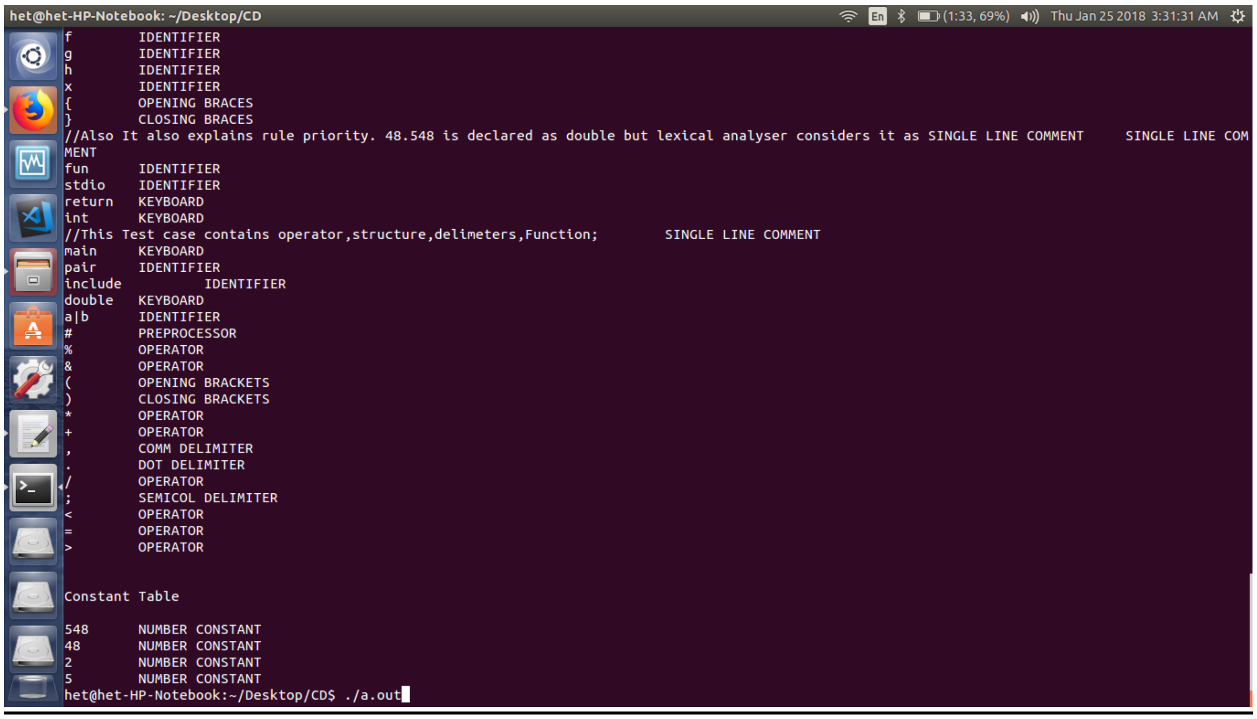
```
)        - CLOSING BRACKETS
{        - OPENING BRACES
int      - KEYWORD
a        - IDENTIFIER
,        - COMM DELIMITER
b        - IDENTIFIER
,        - COMM DELIMITER
c        - IDENTIFIER
,        - COMM DELIMITER
d        - IDENTIFIER
,        - COMM DELIMITER
e        - IDENTIFIER
,        - COMM DELIMITER
f        - IDENTIFIER
,        - COMM DELIMITER
g        - IDENTIFIER
,        - COMM DELIMITER
h        - IDENTIFIER
;        - SEMICOL DELIMITER
a        - IDENTIFIER
=        - OPERATOR
2        - NUMBER CONSTANT
,        - COMM DELIMITER
b        - IDENTIFIER
=        - OPERATOR
5        - NUMBER CONSTANT
;        - SEMICOL DELIMITER
c        - IDENTIFIER
=        - OPERATOR
a        - IDENTIFIER
+        - OPERATOR
b        - IDENTIFIER
;        - SEMICOL DELIMITER
d        - IDENTIFIER
=        - OPERATOR
a        - IDENTIFIER
*        - OPERATOR
b        - IDENTIFIER
;        - SEMICOL DELIMITER
e        - IDENTIFIER
=        - OPERATOR
```


het@het-HP-Notebook: ~/Desktop/CD

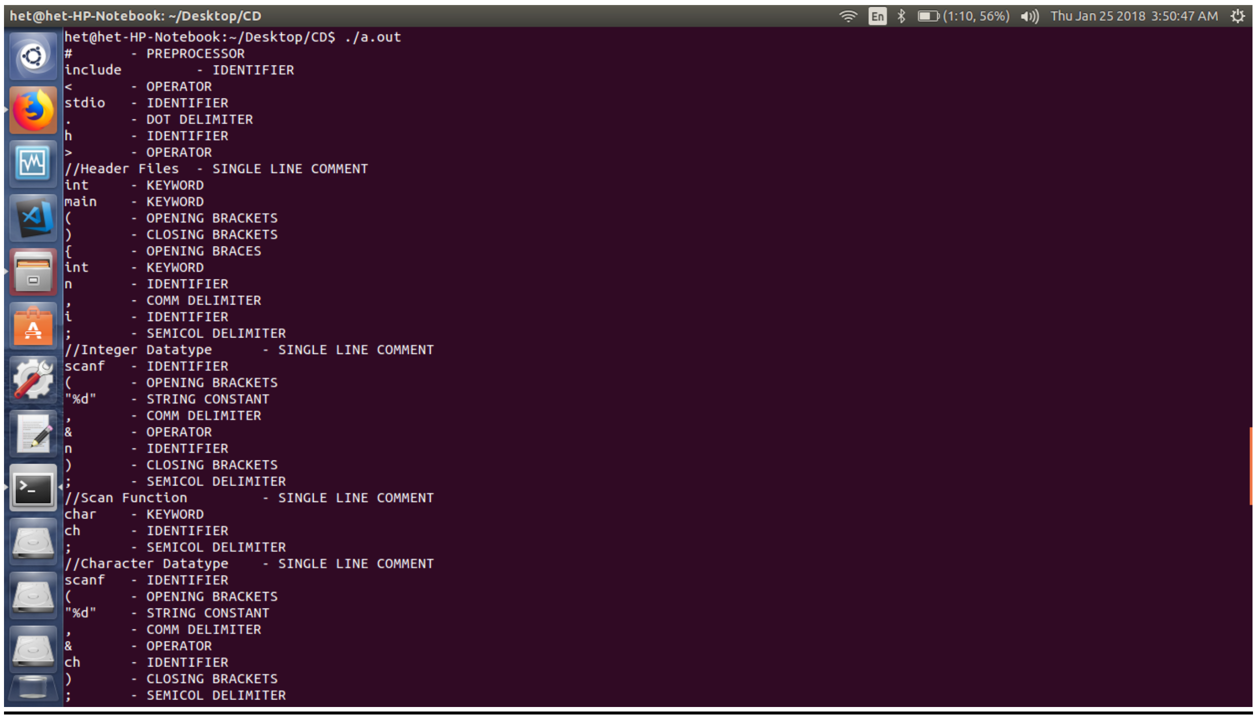
```
a        - IDENTIFIER
/        - OPERATOR
b        - IDENTIFIER
;        - SEMICOL DELIMITER
f        - IDENTIFIER
=        - OPERATOR
a        - IDENTIFIER
%        - OPERATOR
b        - IDENTIFIER
;        - SEMICOL DELIMITER
g        - IDENTIFIER
=        - OPERATOR
a        - IDENTIFIER
&        - OPERATOR
b        - IDENTIFIER
;        - SEMICOL DELIMITER
h        - IDENTIFIER
=        - OPERATOR
a|b      - IDENTIFIER
;        - SEMICOL DELIMITER
h        - IDENTIFIER
=        - OPERATOR
a        - IDENTIFIER
^        - INVALID
b        - IDENTIFIER
;        - SEMICOL DELIMITER
h        - IDENTIFIER
=        - OPERATOR
a        - IDENTIFIER
*        - OPERATOR
(        - OPENING BRACKETS
a        - IDENTIFIER
+        - OPERATOR
b        - IDENTIFIER
)        - CLOSING BRACKETS
;        - SEMICOL DELIMITER
h        - IDENTIFIER
=        - OPERATOR
a        - IDENTIFIER
*        - OPERATOR
a        - IDENTIFIER
```


het@het-HP-Notebook: ~/Desktop/CD   (1:33, 69%) Thu Jan 25 2018 3:31:23 AM

```
struct   KEYBOARD
b        IDENTIFIER
c        IDENTIFIER
d        IDENTIFIER
e        IDENTIFIER
f        IDENTIFIER
g        IDENTIFIER
h        IDENTIFIER
x        IDENTIFIER
{        OPENING BRACES
}        CLOSING BRACES
//Also It also explains rule priority. 48.548 is declared as double but lexical analyser considers it as SINGLE LINE COMMENT     SINGLE LINE COM
MENT
fun      IDENTIFIER
stdio    IDENTIFIER
return   KEYBOARD
int      KEYBOARD
//This Test case contains operator,structure,delimeters,Function;        SINGLE LINE COMMENT
main     KEYBOARD
pair     IDENTIFIER
include          IDENTIFIER
double   KEYBOARD
a|b      IDENTIFIER
#        PREPROCESSOR
%        OPERATOR
&        OPERATOR
(        OPENING BRACKETS
)        CLOSING BRACKETS
*        OPERATOR
+        OPERATOR
,        COMM DELIMITER
.        DOT DELIMITER
/        OPERATOR
;        SEMICOL DELIMITER
<        OPERATOR
=        OPERATOR
>        OPERATOR


Constant Table
```


het@het-HP-Notebook: ~/Desktop/CD   (1:33, 69%) Thu Jan 25 2018 3:31:31 AM

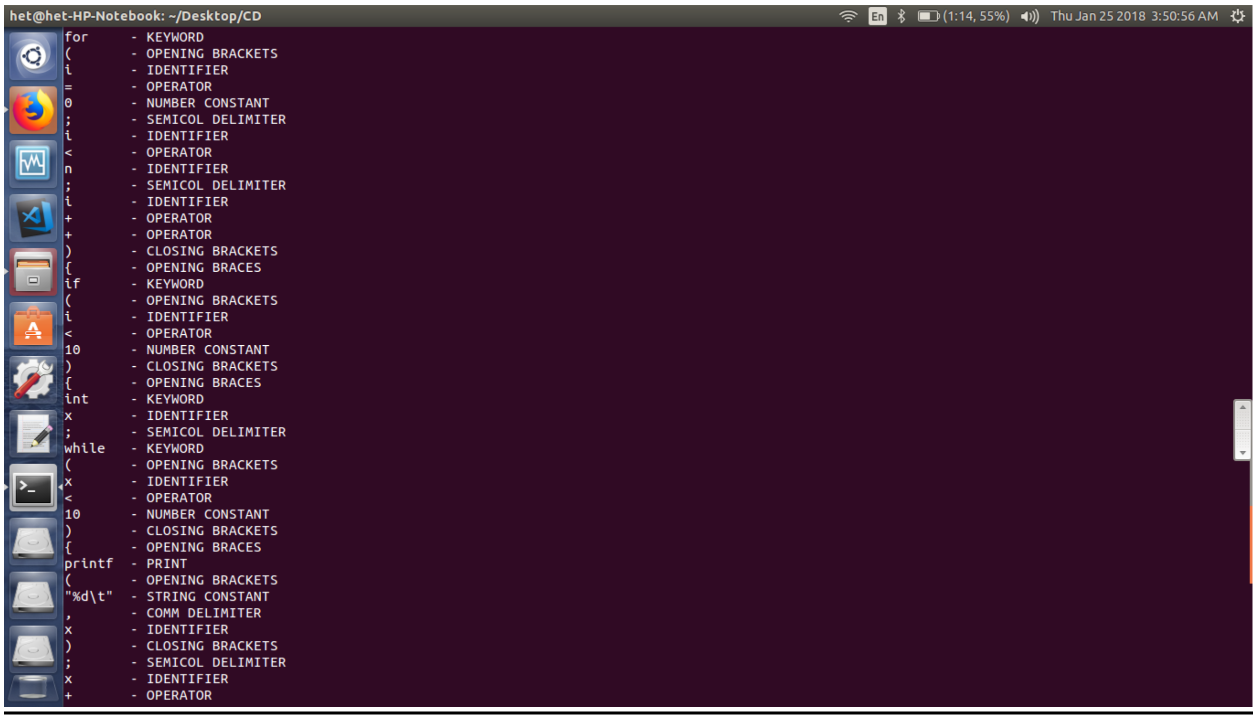
```
f        IDENTIFIER
g        IDENTIFIER
h        IDENTIFIER
x        IDENTIFIER
{        OPENING BRACES
}        CLOSING BRACES
//Also It also explains rule priority. 48.548 is declared as double but lexical analyser considers it as SINGLE LINE COMMENT     SINGLE LINE COM
MENT
fun      IDENTIFIER
stdio    IDENTIFIER
return   KEYBOARD
int      KEYBOARD
//This Test case contains operator,structure,delimeters,Function;        SINGLE LINE COMMENT
main     KEYBOARD
pair     IDENTIFIER
include          IDENTIFIER
double   KEYBOARD
a|b      IDENTIFIER
#        PREPROCESSOR
%        OPERATOR
&        OPERATOR
(        OPENING BRACKETS
)        CLOSING BRACKETS
*        OPERATOR
+        OPERATOR
,        COMM DELIMITER
.        DOT DELIMITER
/        OPERATOR
;        SEMICOL DELIMITER
<        OPERATOR
=        OPERATOR
>        OPERATOR


Constant Table

548      NUMBER CONSTANT
48       NUMBER CONSTANT
2        NUMBER CONSTANT
5        NUMBER CONSTANT
het@het-HP-Notebook:~/Desktop/CD$ ./a.out
```



het@het-HP-Notebook: ~/Desktop/CD

(1:10, 56%) Thu Jan 25 2018 3:50:47 AM

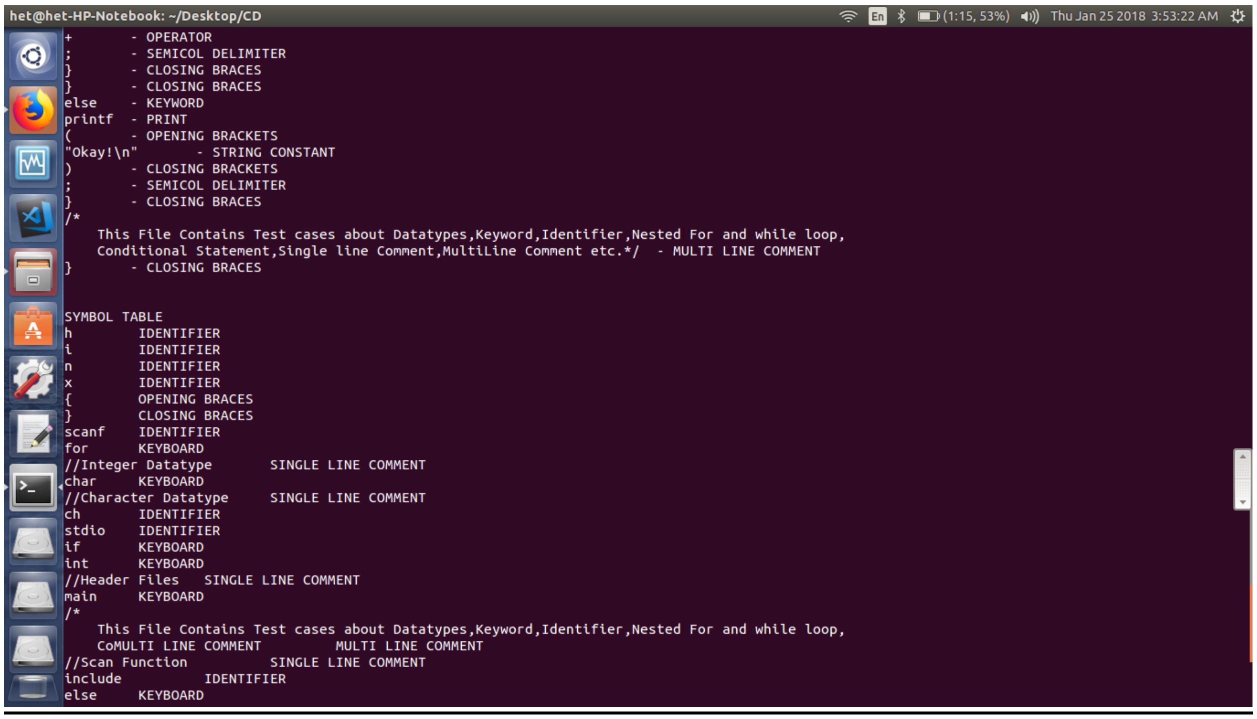
```
het@het-HP-Notebook:~/Desktop/CD$ ./a.out
#        - PREPROCESSOR
include         - IDENTIFIER
<        - OPERATOR
stdio    - IDENTIFIER
.        - DOT DELIMITER
h        - IDENTIFIER
>        - OPERATOR
//Header Files  - SINGLE LINE COMMENT
int      - KEYWORD
main     - KEYWORD
(        - OPENING BRACKETS
)        - CLOSING BRACKETS
{        - OPENING BRACES
int      - KEYWORD
n        - IDENTIFIER
,        - COMM DELIMITER
i        - IDENTIFIER
;        - SEMICOL DELIMITER
//Integer Datatype       - SINGLE LINE COMMENT
scanf    - IDENTIFIER
(        - OPENING BRACKETS
"%d"     - STRING CONSTANT
,        - COMM DELIMITER
&        - OPERATOR
n        - IDENTIFIER
)        - CLOSING BRACKETS
;        - SEMICOL DELIMITER
//Scan Function          - SINGLE LINE COMMENT
char     - KEYWORD
ch       - IDENTIFIER
;        - SEMICOL DELIMITER
//Character Datatype     - SINGLE LINE COMMENT
scanf    - IDENTIFIER
(        - OPENING BRACKETS
"%d"     - STRING CONSTANT
,        - COMM DELIMITER
&        - OPERATOR
ch       - IDENTIFIER
)        - CLOSING BRACKETS
;        - SEMICOL DELIMITER
```



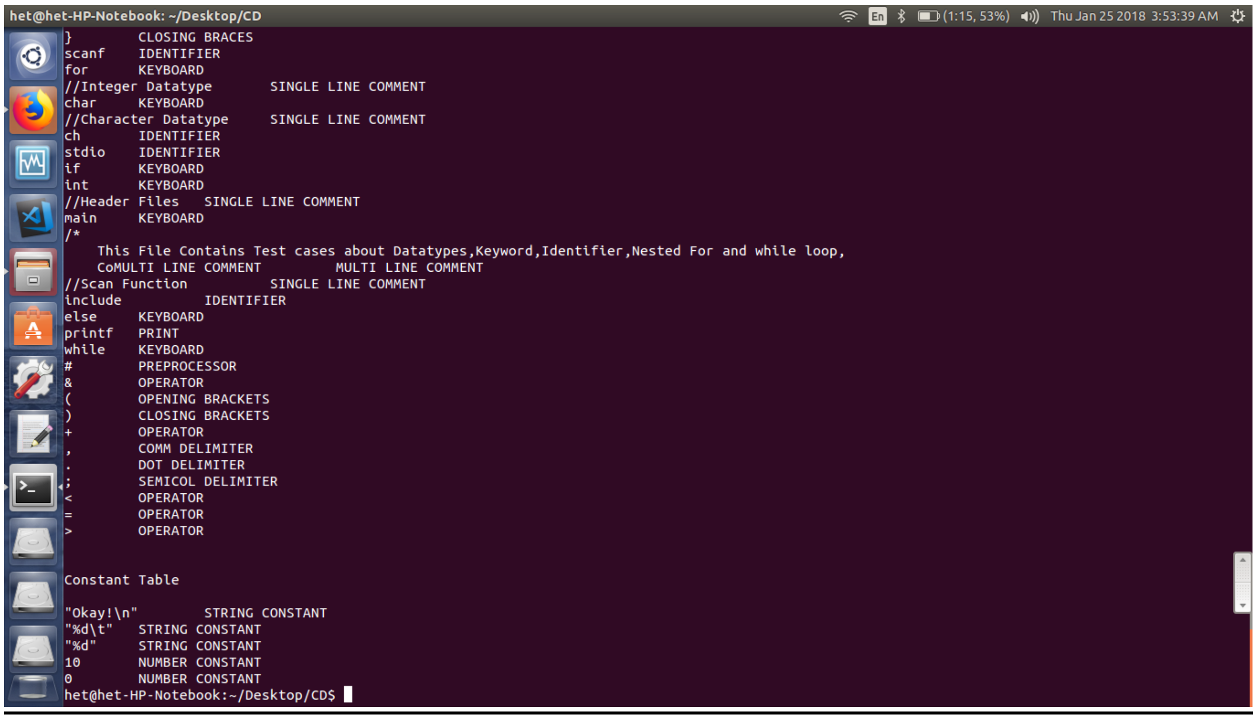
het@het-HP-Notebook: ~/Desktop/CD

(1:14, 55%) Thu Jan 25 2018 3:50:56 AM

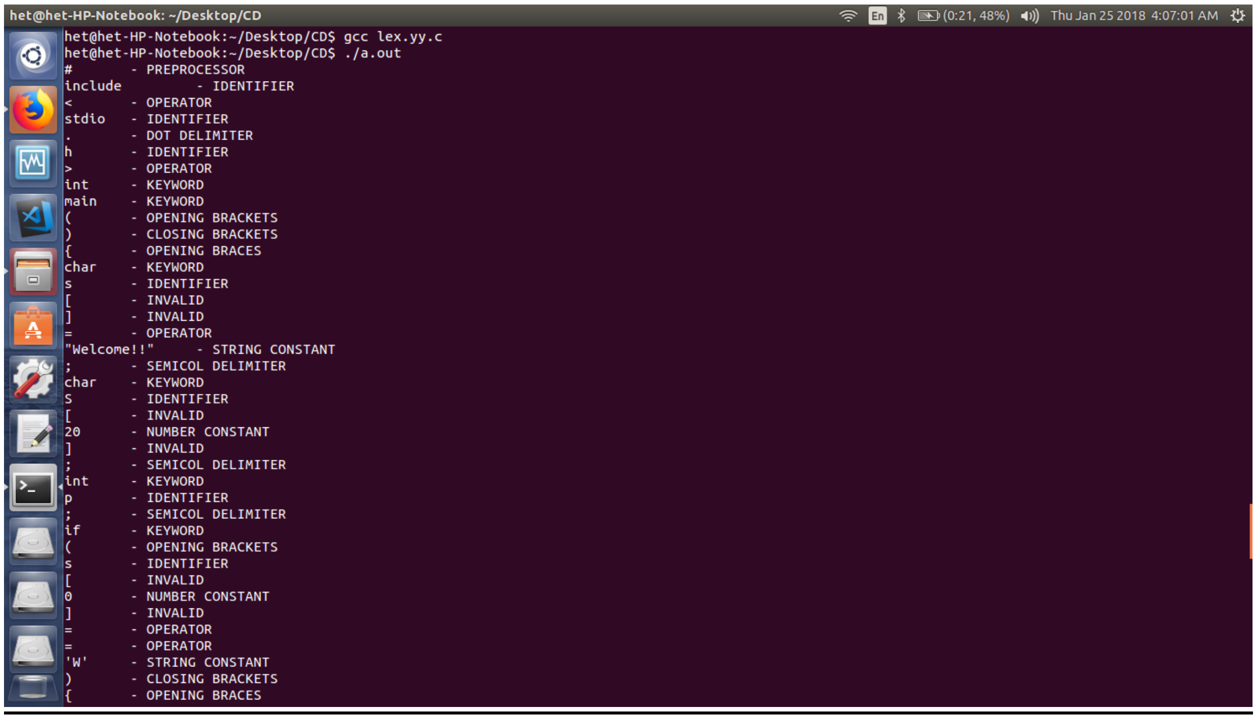
```
for      - KEYWORD
(        - OPENING BRACKETS
i        - IDENTIFIER
=        - OPERATOR
0        - NUMBER CONSTANT
;        - SEMICOL DELIMITER
i        - IDENTIFIER
<        - OPERATOR
n        - IDENTIFIER
;        - SEMICOL DELIMITER
i        - IDENTIFIER
+        - OPERATOR
+        - OPERATOR
)        - CLOSING BRACKETS
{        - OPENING BRACES
if       - KEYWORD
(        - OPENING BRACKETS
i        - IDENTIFIER
<        - OPERATOR
10       - NUMBER CONSTANT
)        - CLOSING BRACKETS
{        - OPENING BRACES
int      - KEYWORD
x        - IDENTIFIER
;        - SEMICOL DELIMITER
while    - KEYWORD
(        - OPENING BRACKETS
x        - IDENTIFIER
<        - OPERATOR
10       - NUMBER CONSTANT
)        - CLOSING BRACKETS
{        - OPENING BRACES
printf   - PRINT
(        - OPENING BRACKETS
"%d\t"   - STRING CONSTANT
,        - COMM DELIMITER
x        - IDENTIFIER
)        - CLOSING BRACKETS
;        - SEMICOL DELIMITER
x        - IDENTIFIER
+        - OPERATOR
```



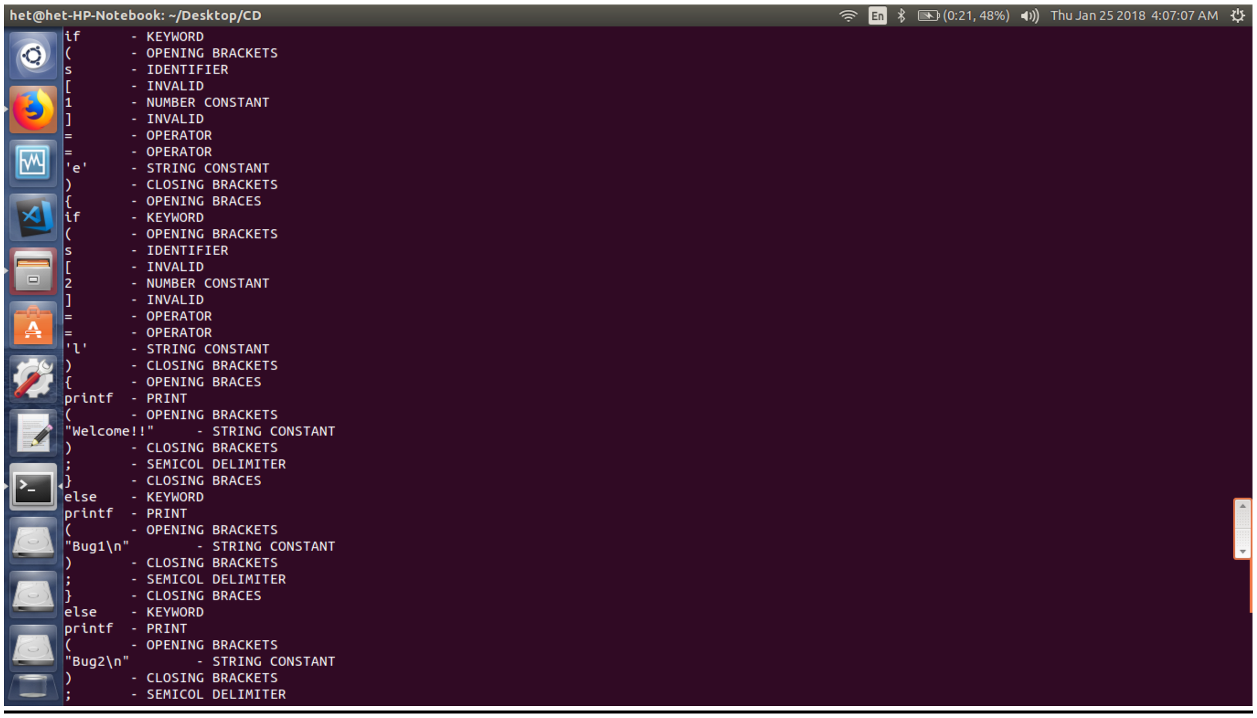
```
+        - OPERATOR
;        - SEMICOL DELIMITER
}        - CLOSING BRACES
}        - CLOSING BRACES
else     - KEYWORD
printf   - PRINT
(        - OPENING BRACKETS
"Okay!\n"        - STRING CONSTANT
)        - CLOSING BRACKETS
;        - SEMICOL DELIMITER
}        - CLOSING BRACES
/*
    This File Contains Test cases about Datatypes,Keyword,Identifier,Nested For and while loop,
    Conditional Statement,Single line Comment,MultiLine Comment etc.*/  - MULTI LINE COMMENT
}        - CLOSING BRACES


SYMBOL TABLE
h        IDENTIFIER
i        IDENTIFIER
n        IDENTIFIER
x        IDENTIFIER
{        OPENING BRACES
}        CLOSING BRACES
scanf    IDENTIFIER
for      KEYBOARD
//Integer Datatype       SINGLE LINE COMMENT
char     KEYBOARD
//Character Datatype     SINGLE LINE COMMENT
ch       IDENTIFIER
stdio    IDENTIFIER
if       KEYBOARD
int      KEYBOARD
//Header Files   SINGLE LINE COMMENT
main     KEYBOARD
/*
    This File Contains Test cases about Datatypes,Keyword,Identifier,Nested For and while loop,
    CoMULTI LINE COMMENT         MULTI LINE COMMENT
//Scan Function          SINGLE LINE COMMENT
include          IDENTIFIER
else     KEYBOARD
```



```
}        CLOSING BRACES
scanf    IDENTIFIER
for      KEYBOARD
//Integer Datatype       SINGLE LINE COMMENT
char     KEYBOARD
//Character Datatype     SINGLE LINE COMMENT
ch       IDENTIFIER
stdio    IDENTIFIER
if       KEYBOARD
int      KEYBOARD
//Header Files   SINGLE LINE COMMENT
main     KEYBOARD
/*
    This File Contains Test cases about Datatypes,Keyword,Identifier,Nested For and while loop,
    CoMULTI LINE COMMENT         MULTI LINE COMMENT
//Scan Function          SINGLE LINE COMMENT
include          IDENTIFIER
else     KEYBOARD
printf   PRINT
while    KEYBOARD
#        PREPROCESSOR
&        OPERATOR
(        OPENING BRACKETS
)        CLOSING BRACKETS
+        OPERATOR
,        COMM DELIMITER
.        DOT DELIMITER
;        SEMICOL DELIMITER
<        OPERATOR
=        OPERATOR
>        OPERATOR


Constant Table

"Okay!\n"        STRING CONSTANT
"%d\t"   STRING CONSTANT
"%d"     STRING CONSTANT
10       NUMBER CONSTANT
0        NUMBER CONSTANT
het@het-HP-Notebook:~/Desktop/CD$
```



```
het@het-HP-Notebook:~/Desktop/CD$ gcc lex.yy.c
het@het-HP-Notebook:~/Desktop/CD$ ./a.out
#       - PREPROCESSOR
include         - IDENTIFIER
<       - OPERATOR
stdio   - IDENTIFIER
.       - DOT DELIMITER
h       - IDENTIFIER
>       - OPERATOR
int     - KEYWORD
main    - KEYWORD
(       - OPENING BRACKETS
)       - CLOSING BRACKETS
{       - OPENING BRACES
char    - KEYWORD
s       - IDENTIFIER
[       - INVALID
]       - INVALID
=       - OPERATOR
"Welcome!!"     - STRING CONSTANT
;       - SEMICOL DELIMITER
char    - KEYWORD
S       - IDENTIFIER
[       - INVALID
20      - NUMBER CONSTANT
]       - INVALID
;       - SEMICOL DELIMITER
int     - KEYWORD
p       - IDENTIFIER
;       - SEMICOL DELIMITER
if      - KEYWORD
(       - OPENING BRACKETS
s       - IDENTIFIER
[       - INVALID
0       - NUMBER CONSTANT
]       - INVALID
=       - OPERATOR
=       - OPERATOR
'W'     - STRING CONSTANT
)       - CLOSING BRACKETS
{       - OPENING BRACES
```



```
if      - KEYWORD
(       - OPENING BRACKETS
s       - IDENTIFIER
[       - INVALID
1       - NUMBER CONSTANT
]       - INVALID
=       - OPERATOR
=       - OPERATOR
'e'     - STRING CONSTANT
)       - CLOSING BRACKETS
{       - OPENING BRACES
if      - KEYWORD
(       - OPENING BRACKETS
s       - IDENTIFIER
[       - INVALID
2       - NUMBER CONSTANT
]       - INVALID
=       - OPERATOR
=       - OPERATOR
'l'     - STRING CONSTANT
)       - CLOSING BRACKETS
{       - OPENING BRACES
printf  - PRINT
(       - OPENING BRACKETS
"Welcome!!"     - STRING CONSTANT
)       - CLOSING BRACKETS
;       - SEMICOL DELIMITER
}       - CLOSING BRACES
else    - KEYWORD
printf  - PRINT
(       - OPENING BRACKETS
"Bug1\n"        - STRING CONSTANT
)       - CLOSING BRACKETS
;       - SEMICOL DELIMITER
}       - CLOSING BRACES
else    - KEYWORD
printf  - PRINT
(       - OPENING BRACKETS
"Bug2\n"        - STRING CONSTANT
)       - CLOSING BRACKETS
;       - SEMICOL DELIMITER
```

Test Case 1:

```
#include<stdio.h>//Header Files

int main(){
    int n,i;//Integer Datatype
    scanf("%d",&n);//Scan Function
    char ch;//Character Datatype
    scanf("%d",&ch);

    for (i=0;i<n;i++){
        if(i<10){
            int x;
            while(x<10){
                printf("%d\t",x);
                x++;
            }
        }

        else printf("Okay!\n");
    }
    /*
    This File Contains Test cases about Datatypes,Keyword,Identifier,Nested
For and while loop,
    Conditional Statement,Single line Comment,MultiLine Comment etc.*/

}
```

Test Case 2:

```
#include<stdio.h>

struct pair{
    int a;
    int b;
};

int fun(int x){
    return x*x;
}

int main(){
    int a,b,c,d,e,f,g,h;
    a=2,b=5;

    c=a+b;
    d=a*b;
    e=a/b;
    f=a%b;
    g=a&b;
    h=a|b;
    h=a^b;
    h=a*(a+b);
    h=a*a+b*b;
    h=fun(b);

    //This Test case contains operator,structure,delimeters,Function;
}
```

Test Case 3:

```
#include<stdio.h>

int main(){
    char s[]="Welcome!!";
    char S[20];

    int p;
    if(s[0]=='W'){
        if(s[1]=='e'){
            if(s[2]=='l'){
                printf("Welcome!!");
            }

            else printf("Bug1\n");
        }
        else printf("Bug2\n");
    }

    else printf("Bug3\n");

    int @<-_-= 2;

    //This test case contains nested conditional statement,Array and print
statement
    //Also there is an error in declaring integer variable which does not
match any regular expression.
}
```