Lexical Analysis মানে, তুমি যে C code
compiler এর মধ্যে দিবা, সেখান থেকে প্রয়োজনীয়
Information কে বের করে নিবে।

```
%{#include <stdio.h>
#include "main.cpp"
```
← আগেরটা offline এ symbol table বানানোর জন্য

```
FILE *logout;
FILE *tokenout;
```
} txt file

```
extern "C"
{
   extern int yylex(void);
}
```
} C++ code এ C function use করার জন্য।

```
int line_count = 1;
Symbol_table table;
```
→ object

```
void insert(const char* token, char* yytext) {
    //printf("%s: %s ", token, yytext);
    /*printf("%s ", token);*/
    table.insert(token, yytext, hash_buckets);
}
%}
```
↳ এটা আগেরটা offline এর জন্য।
সেখানে Manually করা। এবার
এখানে lex থেকে Directly যাবে।

```
newLine         \n
delim           [ \t\a]
digit           [0-9]          1
unsigned        {digit}+       12
exponent        [eE][+-]?{unsigned}   e+10
number          ({unsigned}|{unsigned}\.{unsigned}?
|{unsigned}?\.{unsigned}){exponent}?   10.12
letter          [A-Za-z]-{delim}      a
keyword
    program|if|not|end|begin|else|then|do|while|function|
```
→ এখানে এর হচ্ছে।
Or এর Regex.

```
letter          [A-Za-z]-{delim}
keyword
    program|if|not|end|begin|else|then|do|while|function|
Procedure|integer|real|var|oh|array|write|include|int|char|f
loat|string|printf
prnt_stmt    [printf](\.|\n)*{semicolon}
header          {id}\.h
sin_comment     \/\/.*
mul_comment_err "/**"[^"**/"]+
mul_comment     "/*"(([^*]|(("*"+)[^*/]))*)("*"+)"/"
string          \"((\\.|[^"]))+\"
err_str_nl_2    ^\"(\\.|[^"])*[\n]+
err_str_nl      \"(\\.|[^"])*\"
tab_char        \'\t\'
empty_char      \'\'
err_sp_char     \'[a-zA-Z]+[ ]+[a-zA-Z]+\'
char            \'{letter}\'
err_multi_char  \'((\\.|[^'\n\t\a])-[ ]+)+\'
err_char_nl     ^\'(\\.|[^'])*\n
id              (_|{letter})({letter}|{digit}|_)*
addop           [\+-]|or
mulop           [\*\/]|div|mod|and
relop           =|<>|<|<=|>=|>
assignop        :=
dotdot          \.\.
brace           [\[\]\(\)]
other           [,:]
semicolon       ;
hash            #

err_dotdot      {number}\.{unsigned}{exponent}?
err_id          {digit}+{id}

/*
 * Note: printf is being evaluated as a keyword
 * So no need for prnt_stmt
 *
 * Also I am allowing \n in multi line char
 * To disable:
 * string          \"((\\.|[^\n"]))+\"
 */
```

Or এর Regex.

2-2 এর toc এর সমস্যা Regular expression দেওয়া ছিল।

মনে কর, তুমি IP Address চিনতে করবা,

[0-255]\.[0-255]\.[0-255]\.[0-255]

192.168.0.1 ✓

256.2.A.. X

* backslash \ দিয়ে reserved character (. ' [ ] ( )) এগুলা দেখাতে হয়।

তো এখন, তুমি যে token গুলা পাবা সেগুলা নিয়ে কি করবা? সাধারনভাবে তুমি bison এর কাছে scanner parser-generator তৈরী করতে দাও।

```
%%
{newLine} {
    line_count++;
   table.print(logout);
}

{string} {
    insert("STR",yytext);
    printf("str: %s\n", yytext);
    printf("line no: %d\n",line_count);
    fprintf(tokenout,"\n<STRING, %s>\n",yytext);
    fprintf(logout,"\nLine no %d:Token <STRING> lexeme <%s>
    found\n",line_count,yytext);
}


{err_str_nl} {
    fprintf(tokenout,"\n<STRING, %s>\n",yytext);
    fprintf(logout,"\n ERROR string with newline found\nLine
    no %d:Token <STRING> lexeme <%s> found
    \n",line_count,yytext);
}

{tab_char} {
    /*printf("str: %s\n", yytext);*/
    /*printf("line no: %d\n",line_count);*/
    fprintf(logout,"\nERROR: tab in char found\nLine no %d:
    Token <CHAR> lexeme <%s> found\n",line_count,yytext);
}

{char} {
    insert("CONST_CHAR",yytext);
    /*printf("str: %s\n", yytext);*/
    /*printf("line no: %d\n",line_count);*/
    fprintf(tokenout,"\n<CONST_CHAR, %s>\n",yytext);
    fprintf(logout,"\nLine no %d: Token <CONST_CHAR> lexeme
    <%s> found\n",line_count,yytext);
}

{err_sp_char} {
    insert("CHAR",yytext);
    /*printf("str: %s\n", yytext);*/
    /*printf("line no: %d\n",line_count);*/
    fprintf(logout,"\nWARNING: Space inside character
    constant found\nLine no %d: Token <CHAR> lexeme <%s>
    found\n",line_count,yytext);
    fprintf(tokenout,"\n<CHAR, %s>\n",yytext);
}
```

Line count

insert function declare

Error

```
{err_multi_char} {
    insert("CHAR",yytext);
    /*printf("str: %s\n", yytext);*/
    /*printf("line no: %d\n",line_count);*/
    fprintf(logout,"\nWARNING: Multi character constant found
    \nLine no %d: Token <CHAR> lexeme <%s> found
    \n",line_count,yytext);
    fprintf(tokenout,"\n<CHAR, %s>\n",yytext);
}

{err_char_nl} {
    /*printf("str: %s\n", yytext);*/
    /*printf("line no: %d\n",line_count);*/
    fprintf(logout,"\nERROR: Character with newline found
    \nLine no %d: Token <CHAR> lexeme <%s> found
    \n",line_count,yytext);
}

{empty_char} {
    /*printf("str: %s\n", yytext);*/
    /*printf("line no: %d\n",line_count);*/
    fprintf(logout,"\nERROR: Empty character found\nLine no %
    d: Token <CHAR> lexeme <%s> found\n",line_count,yytext);
}

{keyword} {
    insert("KEYWORD",yytext);
    //printf("keyword");
    fprintf(tokenout,"\n<KEYWORD, %s>\n",yytext);
    fprintf(logout,"\nLine no %d: Token <KEYWORD> lexeme
    <%s> found\n",line_count,yytext);

}

{header} {
    insert("header",yytext);
    fprintf(tokenout,"\n<HEADER %s>\n",yytext);
    fprintf(logout,"\nLine no %d: Token <HEADER> lexeme <%
    s> found\n",line_count,yytext);
}


{id} {
    insert("ID",yytext);
    /*printf("id: %s\n", yytext); */
    fprintf(tokenout,"<ID, %s>\n",yytext);
    fprintf(logout,"\nLine no %d: Token <ID> lexeme <%s>
    found\n",line_count,yytext);
}

{sin_comment} {
```

```
    /*printf("\nSingleline comment found: %s\n",yytext);*/
    fprintf(logout,"\nLine no %d: SINGLELINECOMMENT <%s>
    found\n",line_count,yytext);
}

{mul_comment} {
    /*printf("\nMultiline comment found: %s\n",yytext);*/
    fprintf(logout,"\nLine no %d: MULTILINECOMMENT <%s>
    found\n",line_count,yytext);
}

{mul_comment_err} {
    /*printf("\nMultiline comment found: %s\n",yytext);*/
    fprintf(logout,"\nLine no %d: Unterminated
    MULTILINECOMMENT <%s> found\n",line_count,yytext);
}

{unsigned} {
    insert("CONST_INT",yytext);
    /*printf("number:%s\n",yytext);*/
    /*printf("line no: %d\n",line_count);*/
    fprintf(tokenout,"<CONST_INT, %s>\n",yytext);
    fprintf(logout,"\nLine no %d: Token <CONST_INT> lexeme
    <%s> found\n",line_count,yytext);

}

{number} {
    insert("CONST_FLOAT",yytext);
    /*printf("number:%s\n",yytext);*/
    /*printf("line no: %d\n",line_count);*/
    fprintf(tokenout,"<CONST_FLOAT, %s>\n",yytext);
    fprintf(logout,"\nLine no %d: Token <CONST_FLOAT>
    lexeme <%s> found\n",line_count,yytext);

}

{addop} {
    /*insert("ADDOP", yytext);*/
    /*printf("line no: %d\n",line_count);*/
    fprintf(tokenout,"\n<ADDOP, %s>\n",yytext);
    fprintf(logout,"\nLine no %d: Token <ADDOP> lexeme <%s>
    found\n",line_count,yytext);
}

{mulop} {
    /*insert("MULOP", yytext);*/
    /*printf("line no: %d\n",line_count);*/
    fprintf(tokenout,"\n<MULOP, %s>\n",yytext);
    fprintf(logout,"\nLine no %d: Token <MULOP> lexeme <%s>
    found\n",line_count,yytext);
```

```
}

{relop} {
    /*insert("RELOP", yytext);*/
    /*printf("line no: %d\n",line_count);*/
    fprintf(tokenout,"\n<RELOP, %s>\n",yytext);
    fprintf(logout,"\nLine no %d: Token <RELOP> lexeme <%s>
    found\n",line_count,yytext);
}

{assignop} {
    /*insert("ASSIGNOP", yytext);*/
    /*printf("line no: %d\n",line_count);*/
    fprintf(tokenout,"\n<ASSIGNOP, %s>\n",yytext);
    fprintf(logout,"\nLine no %d: Token <ASSIGNOP> lexeme
    <%s> found\n",line_count,yytext);
}


{dotdot} {
    insert("DOTDOT", yytext);
    /*printf("line no: %d\n",line_count);*/
}
{hash}  {
    insert("#",yytext);
}

{delim}+ {}

{err_dotdot} {
    /*printf("Illegal usage of decimal\n");*/
    fprintf(logout,"\nLine no %d: Illegal usage of decimal <%s>
    found\n",line_count,yytext);
}

{err_id} {
    /*printf("Illegal id\n");*/
    fprintf(logout,"\nLine no %d: Illegal usage of id <%s> found
    \n",line_count,yytext);
}

{prnt_stmt} {
    /*printf("Illegal id\n");*/
    fprintf(logout,"\nLine no %d: Printf statement<%s> found
    \n",line_count,yytext);
}

%%

int main(int argc,char *argv[]){
    logout= fopen("log.txt","w");
    tokenout= fopen("token.txt","w");
```

```
int main(int argc,char *argv[]){
    logout= fopen("log.txt","w");
    tokenout= fopen("token.txt","w");

    yylex();

    fprintf(logout, "\nThe final table: \n");
    table.print(logout);

    fclose(tokenout);
    fclose(logout);

    printf("\nTotal line Count: %d\n", line_count);

    return 0;
}
```

কম্পাইল Run করার :

```
#!/bin/bash

lex -o lex.yy.cpp lex.l
g++ -c  main.cpp lex.yy.cpp
g++ -o lex.out lex.yy.o -lfl
chmod u+x lex.out
./lex.out < input1.txt
```

এটার sh file এ হিসেবে save করতে হবে।

chmod +x দিয়ে executable করতে হবে।

ubuntu /vm এ flex / g++ / gcc এগুলো apt-get দিয়ে install করতে হবে।

./ script.sh দিয়ে Run করতে হবে।