**1. Lexical Analyzer in C:**

**Lexeme :** It is a sequence of characters in the source code that are matched by given predefined language rules for every lexeme to be specified as a valid token.

#include<stdio.h>

int isKeyword(char \*str)

{

char k[32][10]={"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"};

int i;

for(i=0;i<32;i++)

if(strcmp(k[i],str)==0)

return 1;

return 0;

}

int isFunction(char \*str)

{

if(strcmp(str,"main")==0||strcmp(str,"printf")==0)

return 1;

return 0;

}

main()

{

int kc,lno=1,sno=0;

char fn[20],c,buf[30];

FILE \*fp;

printf("\nEnter the file name:");

scanf("%s",fn);

printf("\n\nS.No Token Lexeme Line No");

fp=fopen(fn,"r");

while((c=fgetc(fp))!=EOF)

{

if(isalpha(c))

{

buf[kc=0]=c;

while(isalnum(c=fgetc(fp)))

{

buf[++kc]=c;

}

buf[++kc]='\0';

if(isKeyword(buf))

printf("\n%4d keyword %20s %7d",++sno,buf,lno);

else if(isFunction(buf))

printf("\n%4d function %20s %7d",++sno,buf,lno);

else

printf("\n%4d identifier %20s %7d",++sno,buf,lno);

}

else if(isdigit(c))

{

buf[kc=0]=c;

while(isdigit(c=fgetc(fp)))

buf[++kc]=c;

buf[++kc]='\0';

printf("\n%4d number %20s %7d",++sno,buf,lno);

}

if(c=='('||c==')')

printf("\n%4d parenthesis %6c %7d",++sno,c,lno);

else if(c=='{'||c=='}')

printf("\n%4d brace %6c %7d",++sno,c,lno);

else if(c=='['||c==']')

printf("\n%4d array index %6c %7d",++sno,c,lno);

else if(c==','||c==';')

printf("\n%4d punctuation %6c %7d",++sno,c,lno);

else if(c=='"')

{

kc=-1;

while((c=fgetc(fp))!='"')

buf[++kc]=c;

buf[++kc]='\0';

printf("\n%4d string %20s %7d",++sno,buf,lno);

}

else if(c==' ')

c=fgetc(fp);

else if(c=='\n')

++lno;

else

printf("\n%4d operator %6c %7d",++sno,c,lno);

}

fclose(fp);

}

2. **LEX program to implement RE: a(a+b)\***

**Description:-**The given Language which accepts the String which contains the alphabets a , b and the String should start with a.

**Alphabet** = **{ a , b } Language = { a, aa,aaa,ab,abb,abbb ,………..}**

%{

#include<stdio.h>

int result = 0;

%}

pattern a[a|b]\*[\n]

%%

{pattern} {printf("String is valid \n "); }

. { printf("String is not valid \n"); }

%%

main()

{

printf("Enter the String to Automata: ");

yylex();

}

**(b)Write LEX program to implement RE’s: (a+b)\*a**

**(c) Write LEX program to implement RE’s: a(a+b)\*b**

**Write LEX program to implement RE’s: (a+b)\*abb(a+b)\***

**Aim :**To write a LEX program to implement regular expression **(a+b)\*abb(a+b)\***

**Description :**The given Language which accepts the String which contains the alphabets a,b and starts with a and ends with b, for the given Regular Expression **(a+b)\*abb(a+b)\***.

**Alphabet** = **{ a , b }**

**Language = { abb,abba,aabb,aabb,abbabb,abbb,………..}**

**Source Code :**

%{

#include<stdio.h>

int result = 0;

%}

pattern [a|b]\*abb[a|b]\*[\n]

%%

{pattern} { result=1;return; }

. { result=0;return; }

%%

main()

{

printf("Enter the String to Automata: ");

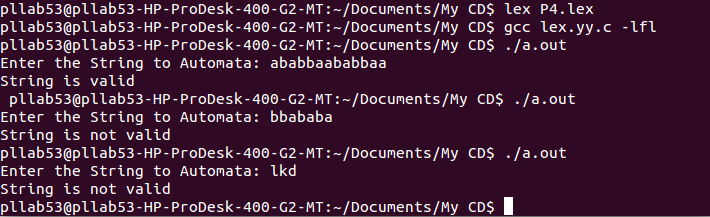
yylex();

result==1?printf("String is valid \n "):

printf("String is not valid \n");

}

**Output:**

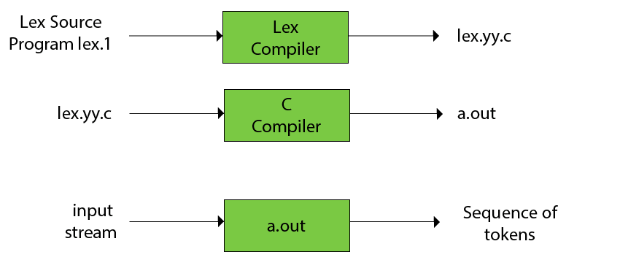


**LEX**

* Lex is a program that generates lexical analyzer. It is used with YACC parser generator.
* The lexical analyzer is a program that transforms an input stream into a sequence of tokens.
* It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program.

**The function of Lex is as follows:**

* Firstly lexical analyzer creates a program lex.1 in the Lex language. Then Lex compiler runs the lex.1 program and produces a C program lex.yy.c.
* Finally C compiler runs the lex.yy.c program and produces an object program a.out.
* a.out is lexical analyzer that transforms an input stream into a sequence of tokens.



**Lex file format**

A Lex program is separated into three sections by %% delimiters. The formal of Lex source is as follows:

%{

Definitions

% }

%%

{ rules }

%%

{ user subroutines }

* Definitions include declarations of constant, variable and regular definitions.
* Rules define the statement of form p1 {action1} p2 {action2}....pn {action}.

Where pi describes the regular expression and action1 describes the actions what action the lexical analyzer should take when pattern pi matches a lexeme.

* User subroutines are auxiliary procedures needed by the actions. The subroutine can be loaded with the lexical analyzer and compiled separately.

**For Ubuntu**

sudo apt -get update

sudo apt -get install flex

sudo apt -get install byacc

sudo apt -get install bison

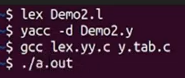
sudo apt -get install bison++

sudo apt -get install byacc-j

**LEX:**

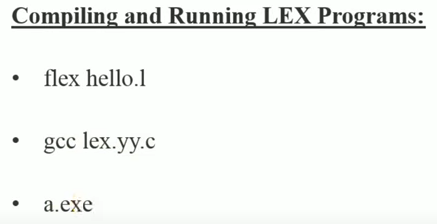
****

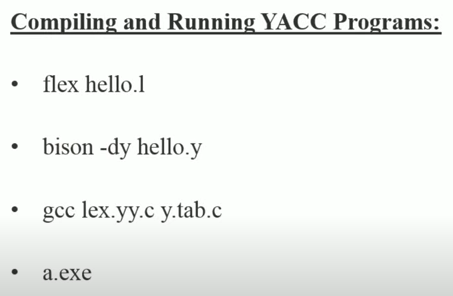
**YACC**



**Flex is a newer and faster version of Lex.** **Jlex is a Java version of Lex**. It generates a scanner coded in Java, though its regular expression definitions are very close to those used by Lex and Flex. Lex, Flex and JLex are largely non- procedural.

***In Windows***





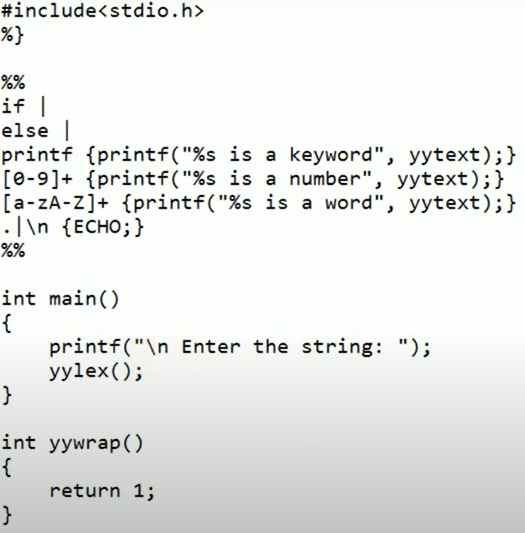
Hello.l -------🡪 flex--------🡪lex.yy.c

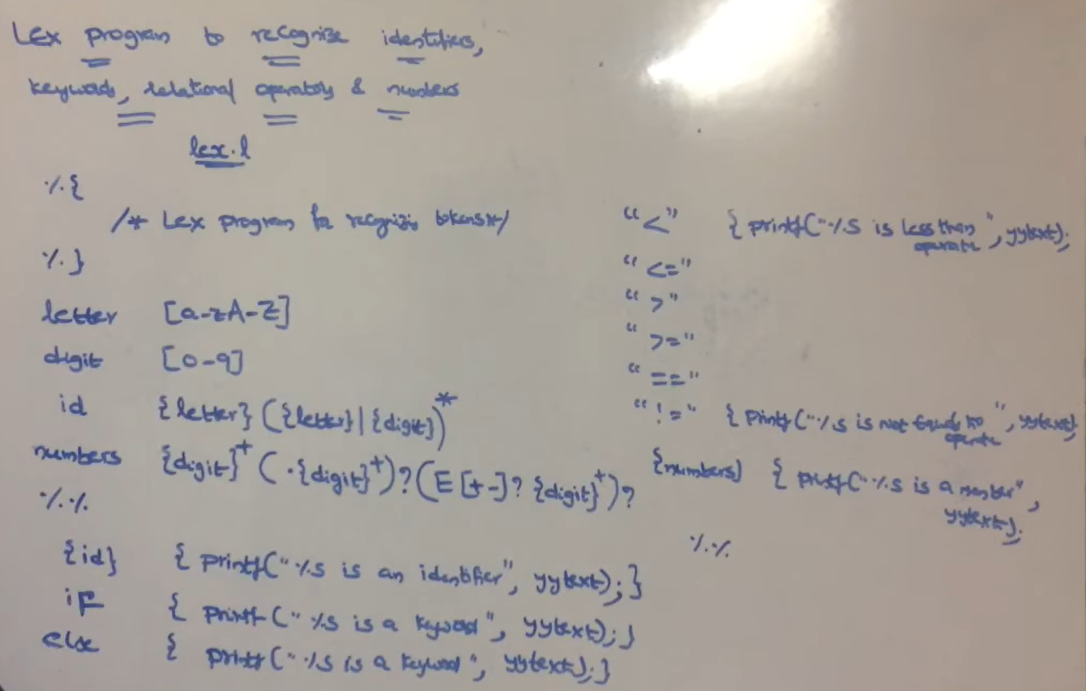
Hello.y ------🡪 bison------🡪y.tab.c

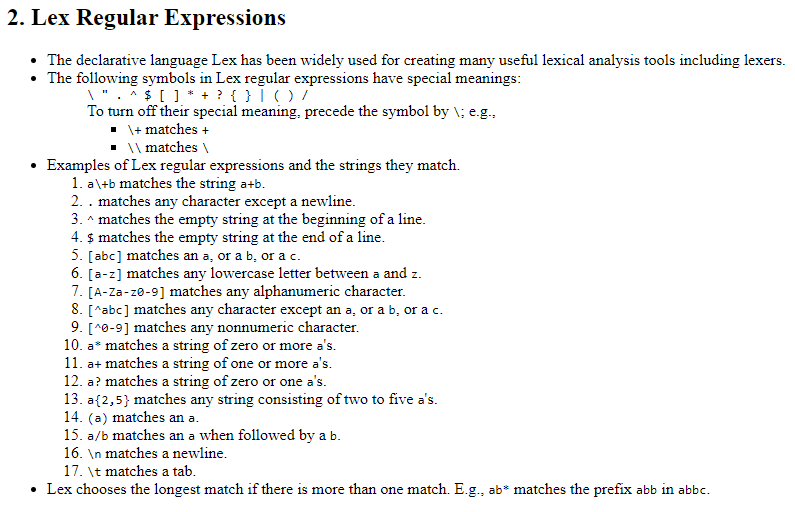
Lex.yy.c y.tab.c -----🡪gcc-----🡪a.exe

Input-----🡪a.exe-----🡪Output

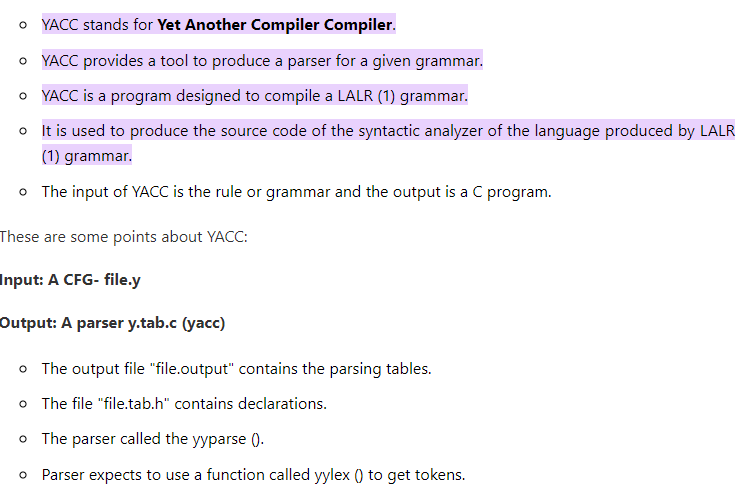
**Lex Program to recognize Keywords, Numbers and Words in a given string**

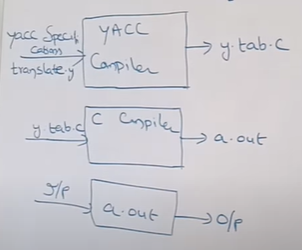


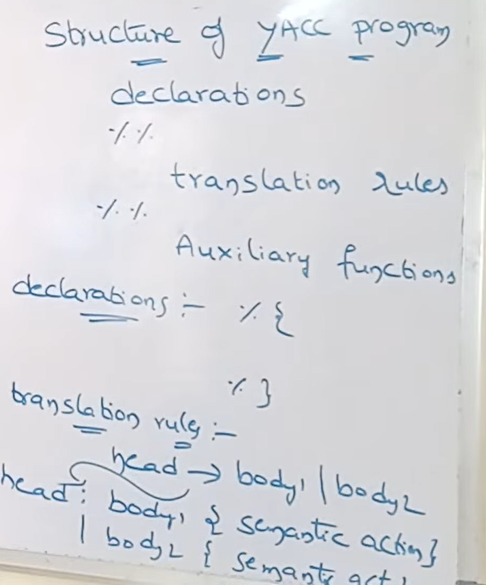


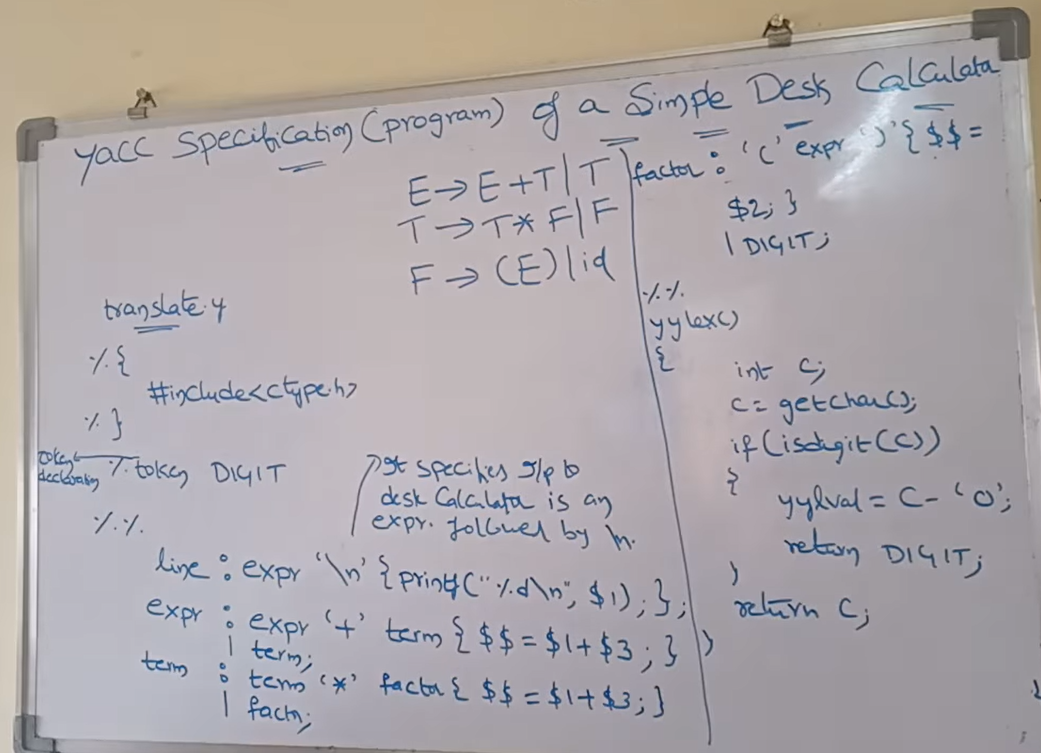
****

**YACC**

****

****

****

****