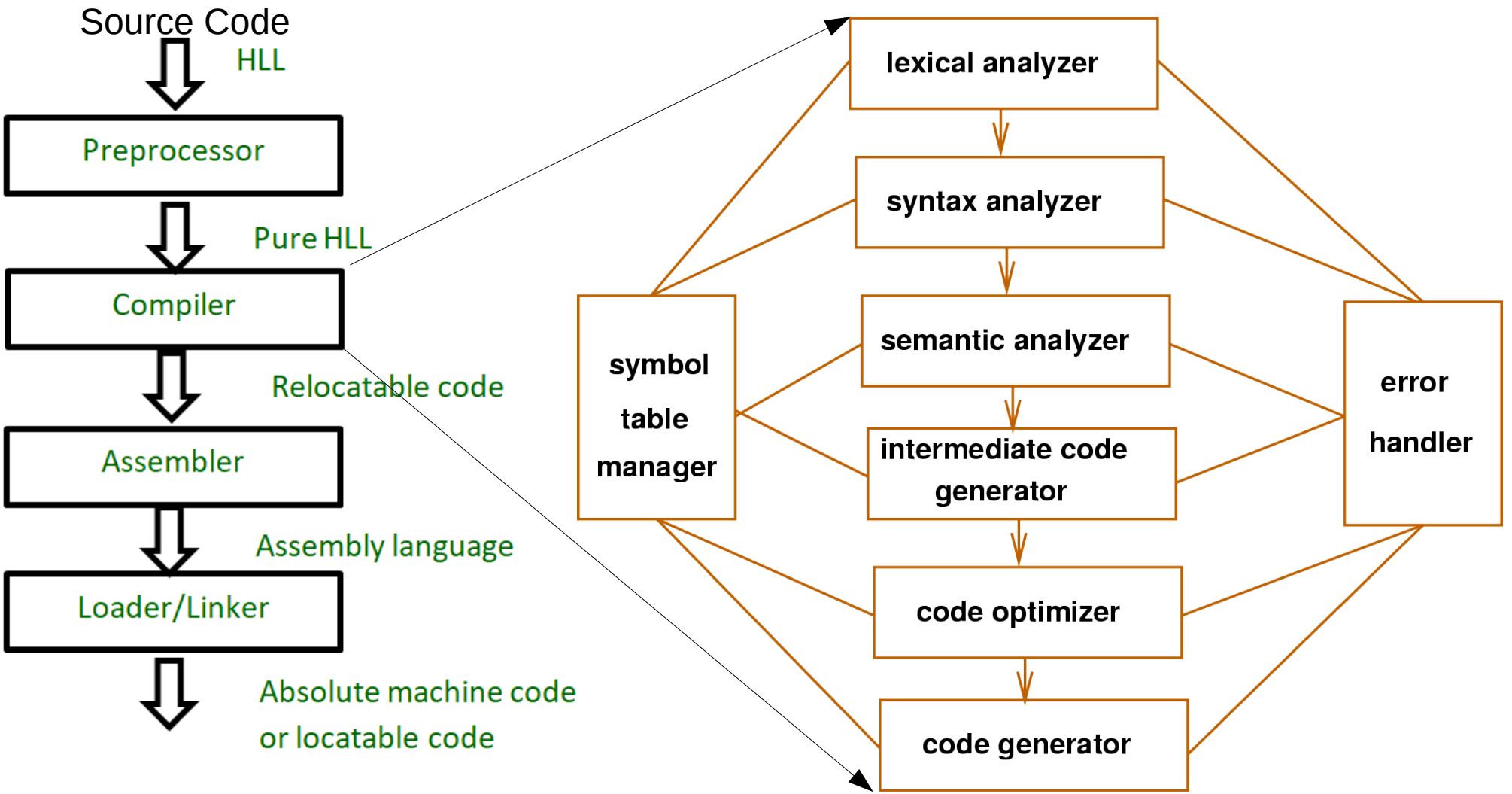
Lex & YACC

Complier :



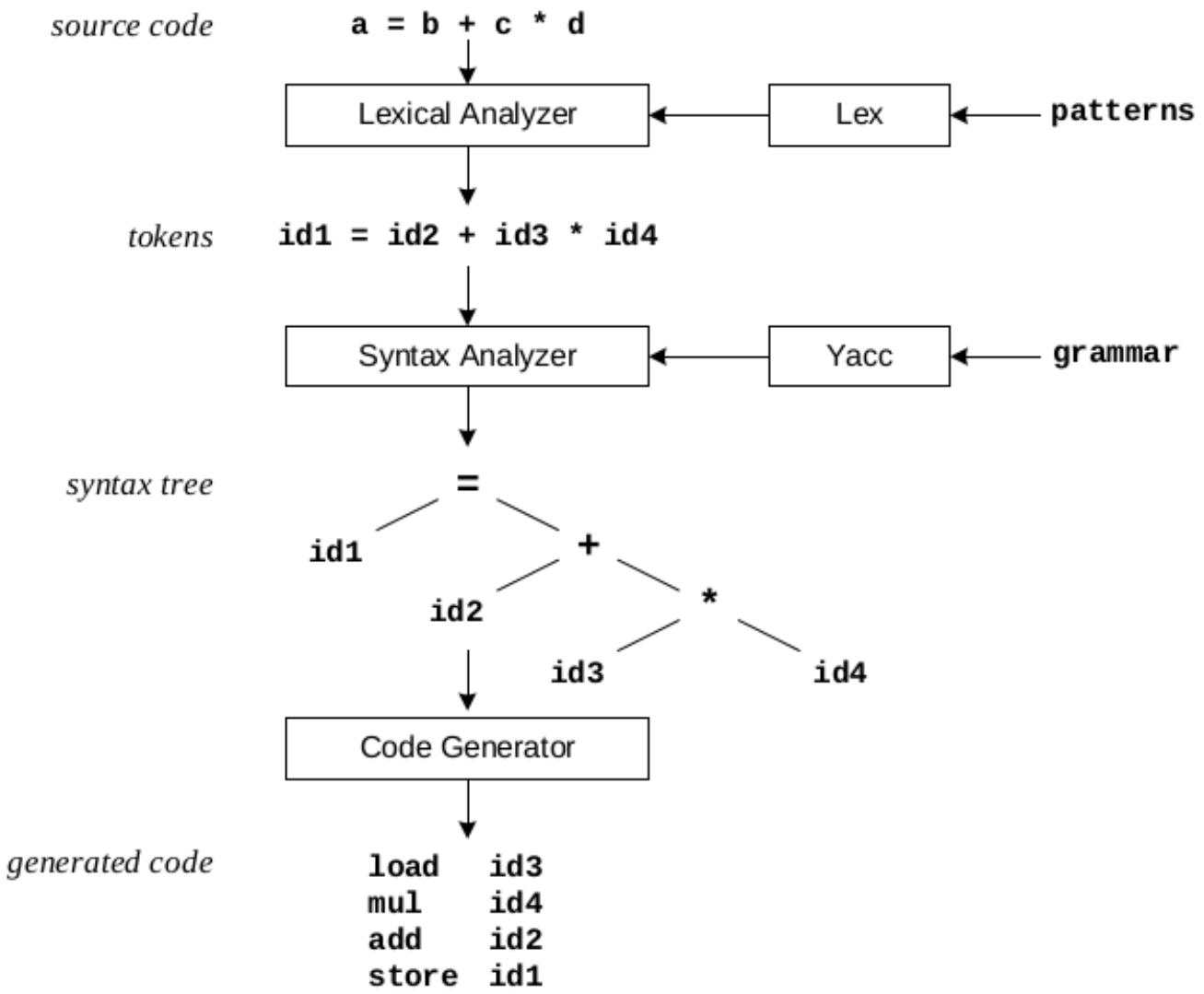
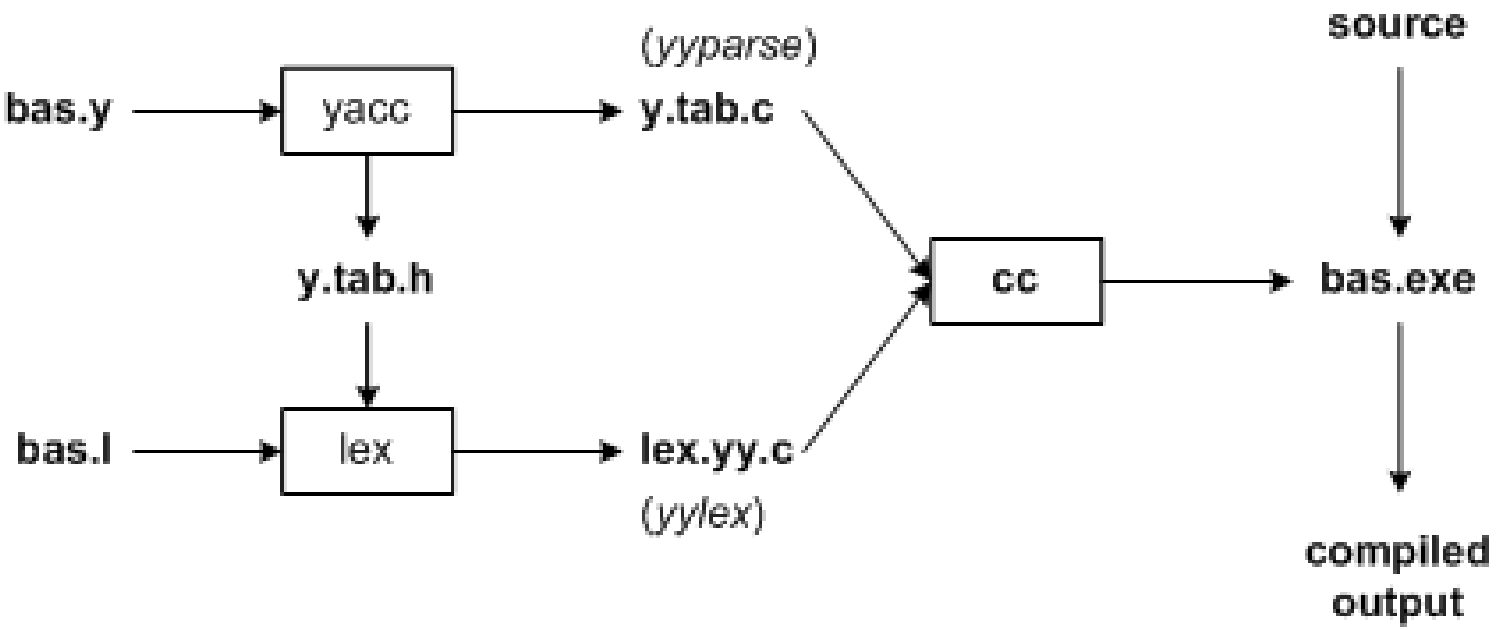
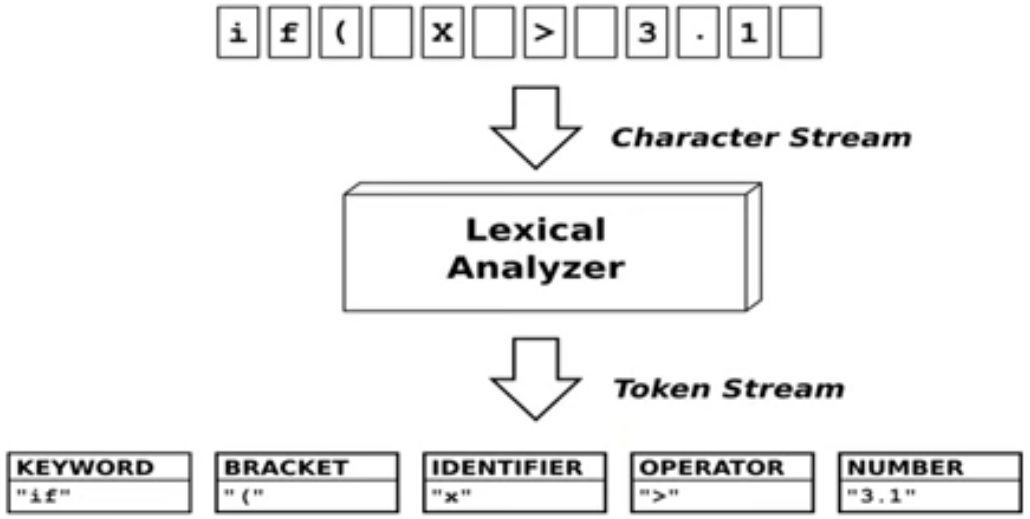


Figure 1: Compilation Sequence


(yyparse)
source
bas.y
yacc
y.tab.c
y.tab.h
cc
bas.exe
bas.l
lex
lex.yy.c
(yylex)
compiled
output

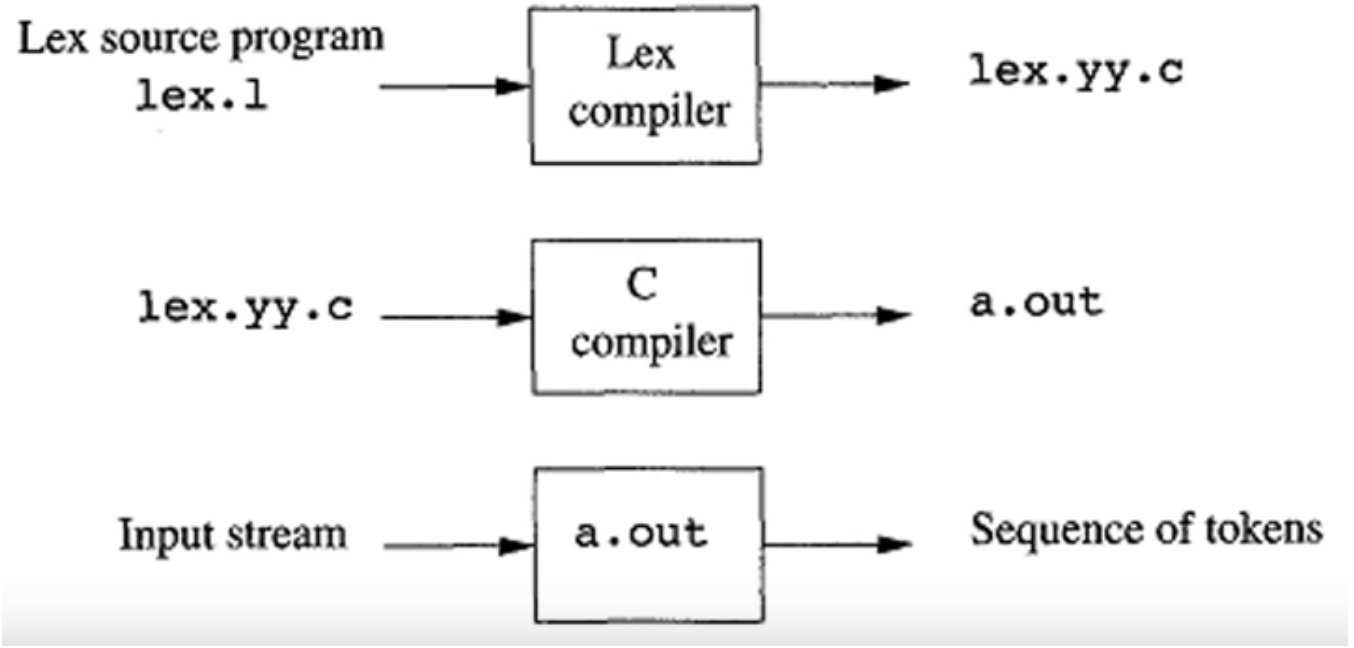
LEX

- It is tool which Generate Lexical analyser
- Lexical analyser is first phase of compiler which take input as source code and generate output as tokens



Lex: Lex will read your **patterns** and generate C code(lex.yy.c) for a lexical analyzer or scanner. The lexical analyzer matches strings in the input, based on your patterns, and converts the strings to tokens.

* Lex is called scanner generator
*Input: Set of Regular Expression(Pattern) and assocuiate action
*Output: lex.yy.c



Working

- An input file, which we call lex.l, is written in the Lex language and describes the lexical analyser to be generated.
- The Lex compiler transforms lex.l to a C program, in a file that is always named lex.yy.c.
- The later file is compiled by the C compiler into a file called a.out, as always.
- The C-compiler output is a working lexical analyser that can take a stream of input characters and produce a stream of tokens

Structure of Lex Programs:

A Lex program has the following form:

{declarations}

%%

{translation rules}

%%

{auxiliary functions}

Decalre variables and c program code that we want to copy Into final program
Use %{--------- %}

Pattern {Action}

Any legal c code lex copies it to the c file after The end of lex generated code

- The declarations section includes declarations of variables
- The translation rules have the form:

  Pattern {Action}
- The third section holds whatever auxiliary functions are used in the actions. Alternatively, these functions can be compiled separately and loaded with the lexical analyser

Pattern	Mathes
.	matches any character except newline character
\.	matches the . character
\n	matches the newline character
\t	matches the tab character
\\	matches the \ character
[\t]	matches either a space or tab character
[^a-d]	matches any character other than a,b,c and d
a	matches a
abc	matches abc
[abc]	matches a, b or c
[a-f]	matches a, b, c, d, e, or f
[0-9]	matches any digit
X+	mathces one or more of X
X*	mathces zero or more of X

Pattern	Mathes
[0-9]+	matches any integer
(...)	grouping an expression into a single unit
|	alternation (or)
(a|b|c)*	is equivalent to [a-c]*
X?	X is optional (0 or 1 occurrence)
[A-Za-z]	matches any alphabetical character

Lex First Program

/Desktop/Unix_shell/Lex_ex/count_vw) - gedit

Open

```
%{ /*this is comment */
#include<stdio.h>
%}
DIGIT [0-9]
%%
"hello world"  {printf("GOod Bye \n");}

.                   ;

%%
```

adhoc@adhoc: ~/Desktop/Unix_shell/Lex_ex/count_vw

```
adhoc@adhoc:~/Desktop/Unix_shell/Lex_ex/count_vw$ lex ex3.l
adhoc@adhoc:~/Desktop/Unix_shell/Lex_ex/count_vw$ ls
count1.l  ex3.l  lex.yy.c
adhoc@adhoc:~/Desktop/Unix_shell/Lex_ex/count_vw$ gcc lex.yy.c -ll
adhoc@adhoc:~/Desktop/Unix_shell/Lex_ex/count_vw$ ls
a.out  count1.l  ex3.l  lex.yy.c
adhoc@adhoc:~/Desktop/Unix_shell/Lex_ex/count_vw$ ./a.out
"hello world"
GOod Bye
```

How to compile/run lex program

$ lex filename.l [It will create lex.yy.c file]

$ gcc lex.yy.c -ll [compile the scanner & main from the lex library(-ll) and create a.out file]

$./a.out ↵

Hell World

show output as----

Good Bye

Variables and functions used in lex

- The routine yylex() provides information by way of global variables and the value it returns, but also by way of macros, functions, and options.

`yyin, yyout`	These are `FILE*`s to the input and output file respectively. Both are accessible and can be assigned to a value other than stdin and stdout
`input()`	Read next character from `yyin`. This is the function invoked by `yylex()` to read the input.
`output()`	Write `yytext` to `yyout`. This is the function invoked by `yylex()` to do an `ECHO`. This function is not supported when generating C++ scanners (see man pages).
`yyless(int n)`	Keeps the *n*th first characters of `yytext` and puts the rest back to `yyin`. For example: `abc/de` is equivalent to `abcde yyless(3);`.
`yymore()`	Keep this token's lexeme in `yytext` when another pattern is matched.
`yywrap()`	Used for multiple input files. Specifies what to do when the `EOF` token is recognised. *flex*'s option `%option noyywrap` instructs the lexer to scan only one file (generates default `yywrap()`), otherwise the funcion must be specified by user. See also `yyrestart()` in documentation.
`yytext`	Array of or pointer to (see man) char where *lex* places the current token's lexeme. The string is automatically null-terminated. It can be modified but not lengthened.
`yyleng`	Integer that holds `strlen(yytext)`.
`yylineno`	Integer that keeps count of lines read. It is maintained automatically by `yylex()` for each `\n` encountered. In *flex*, it is an optional feature that must be specified using `%option yylineno`, for there is a loss of performance.

Lex program to recognize Verb

(~/Desktop/Unix_shell/Lex_ex/verb_ex) - gedit

Open

```
%%

[\t]+                                   /*ignore spaces */  ;
is|am|are|was|were|do|does|did  {printf("%s :is verb \n",yytext);}
[a-zA-Z]+                       {printf("%s : is not verb \n",yytext);}
.|\n                            {ECHO;/*default anyway*/}
%%

main()
{
yylex();
}
```

```
adhoc@adhoc:~/Desktop/Unix_shell/Lex_ex/verb_ex$ gedit verb1.l
adhoc@adhoc:~/Desktop/Unix_shell/Lex_ex/verb_ex$ lex verb1.l
adhoc@adhoc:~/Desktop/Unix_shell/Lex_ex/verb_ex$ gcc lex.yy.c -ll
verb1.l:18:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 {
 ^
adhoc@adhoc:~/Desktop/Unix_shell/Lex_ex/verb_ex$ ./a.out
this is lex program and we do this is in lab also
this : is not verb
 is :is verb
 lex : is not verb
 program : is not verb
 and : is not verb
 we : is not verb
 do :is verb
 this : is not verb
 is :is verb
 in : is not verb
 lab : is not verb
 also : is not verb
```

To count no. Of Vowels & Consonent

l (~/Desktop/Unix_shell/Lex_ex/count_vw) - gedit

Open

```
%{
   #include<stdio.h>
   int vc=0;
   int cc=0;
   int ln=0,nsp=0;
//[b-df-hj-np-tv-z]       {cc++;}
%}

%%
 /* //a|e|i|o|u                    {vc++;}
[aeiouAEIOU]          {vc++;}
[a-zA-Z]              {cc++;}
\n                    {ln++;}
[\t' ']               {nsp++;}
%%

main()
{
printf("enter string for count vowels and consonent");
yylex();
printf("the no. of vowels: %d \n",vc);
printf("the no. of consonent : %d \n",cc);
printf("the no. of line : %d \n",ln);
printf("the no. of spaces : %d \n",nsp);
return 0;
}
```

```
a.out  count1.l  ex3.l  lex.yy.c
adhoc@adhoc:~/Desktop/Unix_shell/Lex_ex/count_vw$ lex count1.l
adhoc@adhoc:~/Desktop/Unix_shell/Lex_ex/count_vw$ gcc lex.yy.c -ll
count1.l:18:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 {
 ^
adhoc@adhoc:~/Desktop/Unix_shell/Lex_ex/count_vw$ ./a.out
enter string for count vowels and consonent
india will win aisa cup
the no. of vowels: 9
the no. of consonent : 10
the no. of line : 2
the no. of spaces : 4
adhoc@adhoc:~/Desktop/Unix_shell/Lex_ex/count_vw$
```

Question on Lex

To count no of vowels and consonants.

To count the length of string

To count the type of numbers-- +integer, -ve integer, +ve fraction, -ve fraction

To count the no of words, character, and lines.

To find if a character apart from alphabets occurs in a string.

To identify set of strings having 3 to 5 alphabets.

To count number of small letter, capital letter, special symbol in string.

Note: You have to practice various question of Lex in LAB