What is Lex?

- Lex is a tool used to create lexical analyzers (tokenizers).
- It reads your source code (like a C or Java program) and breaks it into tokens — like keywords, identifiers, numbers, etc.

Example:If your code has: int a = 10;
Lex will break it into tokens like:

- int → keyword
- a → identifier
- = → assignment operator
- 10 → number
- ; → symbol

What is Yacc?

- Yacc stands for Yet Another Compiler Compiler.
- It is used to create the parser (syntax analyzer)
- It takes the tokens from Lex and checks whether they follow the grammar rules of the language.
- If they do, it builds a parse tree or takes appropriate action.

1. Lex reads your source code → breaks it into tokens.
2. Yacc reads those tokens → checks grammar → builds structure or does processing.

Think of Lex as a scanner and Yacc as a grammar checker.

Simple Flow

1. Lex matches patterns like digits, operators, identifiers.
2. Sends them as tokens to Yacc.
3. Yacc checks if tokens form valid expressions/statements.
4. Takes action like printing, evaluating, or building a syntax tree.

1. Structure of a Lex Program

A Lex file has 3 parts, separated by %%:

Definitions
%%
Rules
%%
C Code (optional)

Example:

```
%{
#include "y.tab.h"  // include tokens from Yacc
%}

%%
[0-9]+      { yylval = atoi(yytext); return NUMBER; }
"+"         { return PLUS; }
"-"         { return MINUS; }
[ \t\n]     ;     // ignore whitespace
.           { return yytext[0]; }
%%

int yywrap() {
   return 1;  // tells the scanner there's no more input
}
```

2. Structure of a Yacc Program

Yacc file also has 3 sections:

Definitions
%%
Grammar Rules

%%
C Code (main function etc.)

Example:

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

%token NUMBER PLUS MINUS

%%
expr : expr PLUS expr   { printf("add\n"); }
     | expr MINUS expr  { printf("sub\n"); }
     | NUMBER           { printf("number\n"); }
     ;
%%

int main() {
   return yyparse();
}

void yyerror(const char *msg) {
   fprintf(stderr, "Syntax Error: %s\n", msg);
}
```

3. Key Functions

yyparse()

- This function is auto-generated by Yacc.
- It drives the parser by calling yylex() repeatedly to get tokens.

- It applies grammar rules based on the tokens.

yylex()

- Generated by Lex.
- It reads input, matches patterns, and returns tokens to yyparse().

yywrap()

- Called when Lex reaches end of input.
- Must return 1 if input is done, 0 if more input is to be scanned.
- Usually written as:

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

yyerror(const char *msg)

- Called by yyparse() when it detects a syntax error.
- You define this function to print error messages or take other actions.
- Example:

```
void yyerror(const char *msg) {
   printf("Syntax error: %s\n", msg);
}
```

yylval

- yylval is a special variable used to pass values from Lex to Yacc (e.g., actual numbers).
- In Yacc:
  %token <val> NUMBER

%union { int val;}

- In Lex: [0-9]+ { yylval.val = atoi(yytext); return NUMBER; }

The typical flow:

1. main() calls yyparse()
2. yyparse() needs a token → it calls yylex()
3. yylex() (from Lex) reads input and returns a token
4. yyparse() applies grammar rules using the token
5. If no rule matches, it calls yyerror()
6. When Lex reaches end of input, it calls yywrap()

File Naming and Compilation

- **Lex file: file.l**
- **Yacc file: file.y**

To compile:

```
lex file.l
yacc -d file.y
gcc lex.yy.c y.tab.c -o parser
./parser
```