/* Aim: Study Practical on Lex, Yacc Compiler ***/**

Introduction to LEX

Lex (Lexical analyzer generator) is a tool used to generate lexical analyses. Lexical analysis is a process of converting input stream into tokens.It takes a set of regular expression given as input in **.l** file and translates it to c implementation which is stored in **lex.yy.c** file.

Structure of LEX Program

```
/* Declarations */
%%
/* Rules */
%%
/* Auxiliary functions */
```

Declarations

Declarations consist of two-part: auxiliary declarations and regular definitions.

Auxiliary declarations section contains optional global declaration (like imports, function prototype, and global variables)will be copied directly to y.tab.c. And are written in c and enclosed within %{ and %}.

Example:

```
%{
   #include <stdio.h>     /* imports */
   int yyerror(char*);    /* function prototype */
   int num;               /* global variable */
%}
```

Rules

Rules contain two parts: pattern to be matched and corresponding actions.

```
/* Declarations */
%%
{numbers}          { return atoi(yytext);}
[a-zA-Z]+          { printf("Hello, %s\n", yytext); return 0;}
.+                 { return yyerror(yytext);}
%%
```

In the above code,

When a number is sent, it will return the number detected in input characters.

When a text is sent, suppose World it will perform an action for printing Hello, World!.

For the rest of the inputs, it will call yyerror function.

Auxiliary Functions

Lex generates C code for rules and places it in function. We can also add our own code to the lex file.

```
/* Declarations */
%%
/* Rules */
%%
int main() { yylex(); return 0;}
int yywrap(void) { return 1; }
```

Functions and variables starting with yy are Predefined functions and variables in Lex:

yylval is a global variable that is used to pass schematic values associated with a token from lexer to the parser.

yytext is a string(pointer to a list of null-terminated characters) that holds the text matched to the current token. if 12 is as input then yytext for number is 12.

yylex() is the entry point for lex . It reads the input stream and generates token. It is defined in lex.yy.c file but it needs to be manually called by the user.

yywrap() is called by yylex() when the input is exhausted. If 0 is seen yylex() will keep reading from pointer yyin. If 1 is returned then the scanning process is terminated. It is mandatory to define in the Lex file.

yyparse() It calls yylex() for lexical analysis. It returns 0 is the input parses according to rules else 0.

Converting the LEX file to c definition.

Introduction to YACC

YACC (Yet Another Compiler Compiler) is a tool used to generate parser. A parser is a program that checks whether a given input meets a grammatical specification. YACC translates a given Context-free grammar in .y file and converts it to c implementation by creating **y.tab.c** and **y.tab.h** file.

Structure of YACC program

Similar to LEX files, the YACC files contains three sections: Declarations, Rules and Auxiliary functions

```
/* Declarations */
%%
/* Rules */
%%
/* Auxiliary functions */
```

Declarations

YACC declaration contains two parts: auxiliary declarations and YACC declaration. Auxiliary declaration is already defined under LEX structure so we are not going to beat around the bush.

Let's just declare it.

```
%{
   #include <stdio.h>          /* imports */
   #include <stdlib.h>
   #include <ctype.h>
   int yylex(void);           /* function prototype */
   int yyerror(const char*);
%}
```

YACC declaration consists of tokens. We can declare the tokens using %tokens <token_name>

```
%token PLUS MINUS        /* declaring token */
%token BYE
%left  '+'  '-'                  /* defining precedence*/
%left   '*'  '/'
```

Rules

YACC rules use context-free grammar described by Brackus Naur Form (BNF). We will be only discussing enough as we need. Feel free to learn more here. Rules consist of two-part **production** part and **action** part. **Production** consists of **head** and **body**.

For example:
expr : expr '+' expr

LHS side of : i.e expr is the production head and right side expr '+' expr is called production body. Head is always a non-terminal, meaning they are replaced by a group of terminal symbols according to production rules. Terminal symbols are tokens.

Action it is a set of c statements that are executed when input matches the body of production. For example:

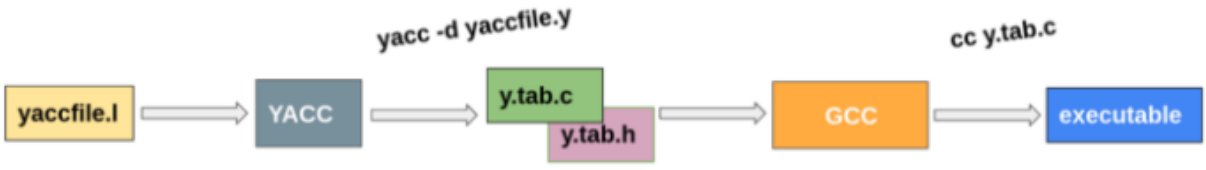
expr: PRINT{ printf("printed");}

When the input matches the production PRINT. It is reduced to the action of printing.

Auxiliary functions

Auxiliary functions are similar to LEX auxiliary functions. What we define here will be added to y.tab.c. Compulsory definitions contain main(), yylex() and yyerror().

```
int main() {
  while(1) yyparse();
}
```



Converting the YACC file to c definition.

Conclusion: Thus we have studied LEX and Yacc Compiler