1 Lex and Yacc : an Introduction

With the help of Lex and Yacc, we are able to transform a structural input to another output file or stream. With the help of lex we can create tokens, which are identified by using regular expression. Thus in structured input, the input is divided into tokens. In C language the structured input, the tokens are variable names, constants, strings, operators, etc. This process of tokenizing the input is called lexical analysis or *lexing*. This lex lexer is fast as compared to a C code written by hand by a person.
When we create tokens with the help of lex, there is a need to have a relationship between the tokens. This relationship is defined by using grammar with the help of Yacc. In Yacc, we use production rules in BNF(Backus Naur Form). The yacc parser automatically detects whenever a sequence of events matches the one in the grammar defined in the Yacc file. The yacc parser is slow compared to a C code parser written by hand by a person.

2 Lex

2.1 Brief Introduction

Lex itself is a very useful tool, other than using it with Yacc. Lex divides the given input stream into set of indivisible units called tokens. When the matching with the token happens, we give a certain action corresponding to the matched pattern, that action is called. Lex itself does not provides an executable, by using the lex tool the lex file creates a subroutine yylex() which returns the token matched from the lex file. The lex file has filename.l extension.

2.2 Syntax of Lex File

A Lex file is divided into three sections seperated by delimeter %%.

```
___ definitions ___
%%
__ rules ___
%%
___ subroutines ___
```

The definitions section contains initialization code which may have some C library header files or initializing variables. It may also contain variable or identifier assigned to a regular expression. The rules section has two fields, first is the regular expression used to match a pattern for the given stream and corresponding to the stream is the action to be taken with the help of C code embedded in the braces({ C code }).

```
regular expression          action

[0-9]+                      {
                                printf("Number matched");
                            }
```

The subroutine has C functions which helps the lexer for ease of action.

2.3 Lex predefined variabled

There are some keywords used in lex file which may point to the matched string, length of the string, etc.

- yytext
  Pointing to the matched null terminated string.
- yylength
  Length of the matched string
- yylval
  Value associated with the token
- int yylex(void) A function called to invoke lexer, returns the token value
- FILE *yyout
  A file stream pointer to the output stream.
- FILE *yyin
  A file stream pointer to the input stream.
- ECHO
  A macro defined to write the matched string to the output stream.

2.4 Using Lex

Suppose we have written a lex file named filename.l, and now we want to create a lex for the defined lex file. We create C file named lex.yy.c which create C lexer from filename.l file.

```
$$ lex filename.l
```

Thus after generating the C lexer we can create the executable using the gcc compiler

```
$$ gcc lex.yy.c -ll -o output
```

Thus our executable is ready.

3 Yacc

3.1 Breif Introduction

Yacc is abbrevated as yet another compiler compiler. Yacc defines relationship between the tokens and when the relationship is matched corresponding action is taken similar to the Lex file. The relationship between the tokens is termed as grammar and the grammar is defined with the help of Backur Naur Form(BNF). Similar to the lexer, yacc creates a subrouting yyparse() which parses the given input stream. The yacc file has filename.y extension.

3.2 Syntax of Yacc file

Similar to the Lex syntax, yacc is also divided into three sections

```
___ definitions ___
%%
__ rules ___
%%
___ subroutines ___
```

The definition section contains initialization code which may contain C header files or variables. Also tokens are also defined in this section. There are 255 tokens pre-defined. The token defined by user starts after number 255. In yacc, the rules section contains of productions in the form of Backus Naur Form(BNF). For each corresponding production there is a action written in C code.

```
production rules                actions

E :
    E + E                       {printf("addition");}
    | E - E                     {printf("subtraction");}
    | id                        {printf("variable");}
```

3.3 Using Yacc

Suppose we have written a yacc file named filename.y and now we want to create a parser for the defined yacc file. With the help of yacc tool we create a C file named y.tab.c also a header file y.tab.h can be create

```
$$ yacc filename.y

$$ yacc filename.y --defines
```

The C code created can be converted to executable using gcc compiler

```
$$ gcc y.tab.c -o output
```