Compiler Construction Labsheet-7

Theory related to Yet Another Compiler Compiler (YACC)

1. The objective of this lab is
    - To understand the working of Yet Another Compiler Compiler (YACC)

The UNIX utility yacc (Yet Another Compiler Compiler) parses a stream of tokens, typicallygenerated by lex, according to a user-specified grammar.

Structure of a yacc file

A yacc file looks much like a lex file:

...definitions...
%%
...rules...
%%
...code...

Definition section: As with lex, all code between %{ and %} is copied to the beginning of the resultingC file. There can also be various definitions.

C code: Any code between %{ and %} is copied to the C file. This is typically used for defining file variables and for prototypes of routines defined in the code segment.

definitions: The definitions section of a lex file was concerned with characters; in yacc this is tokens. These token definitions are written to a .h file when yacc compiles this file.

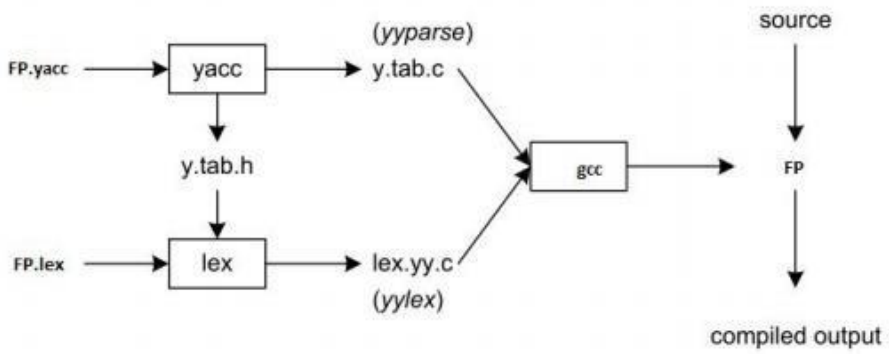
associativity rules: These handle associativity and priority of operators (will be covered in the coming labs).

Rules section: As with lex, several combinations of pattern and action. The patterns are now those ofcontext-free grammar rather than regular expressions as with lex.

User code: This can be very elaborate, but the main ingredient is the call to yyparse, the grammaticalparse.

Lex Yacc interaction

Conceptually, lex parses a file of characters and outputs a stream of tokens; yacc accepts a stream of tokens and parses it, performing actions as appropriate. In practice, they are more tightly coupled.
If your lex program supplies a tokeniser, the yacc program will repeatedly call the yylex routine. The lex rules will probably function by calling return every time they have parsed a token. We will now see how lex returns information in such a way that yacc can use it for parsing.



The shared header file of return codes : If lex is to return tokens that yacc will process, they must agree on what tokens there are.This is done as follows.

Example 1:
First write Yacc program with file extension .y for example "sampleyacc.y"
//Yacc program that recognizes sentences having one or more WORD and end with a NUM linked with ***samplelex.l***

```
%{
# include <stdio.h>
%}
%token NUM WORD
%%
sentence: s{return 1;}
   ;
s:WORDS NUM
   ;
WORDS: WORD WORDS
     | WORD
;
%%

void main()
{
printf ("Enter the sentence:\n");
yyparse();
printf("Valid statement:\n");
exit(0);
}
```

```
void yyerror()
{
printf("Invalid statement:\n");
exit(0);
}
```

// ***samplelex.l*** lex program to generate tokens WORD and NUM defined in ***sampleyacc.y***
Yacc program

```
%{
        #include "y.tab.h"
%}
%%
[a-z]+ return WORD;
[0-9]+ return NUM;
\n return yytext;
%%
```

//Compiling yacc and lex

```
/*
$ yacc -d sample.y              // compile yacc program
$ flex samplelex.l              // compile lex program
$ cc y.tab.c lex.yy.c -ll –ly   // using yacc and lex  to generate .c program
$ ./a.out
*/
```

Other things to note:
• The WORD and NUM tokens that were used in the lex code are defined here in the definitions section; lex knows about it by including the y.tab.h file.

Example 2: Write Yacc and Lex to accept a*b*.

Example 3: Write Yacc and Lex for $a^n b^n$ where $n \geq 0$

Example 4: Consider the following context free grammar
$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$

where id can be generated from [a-z]+.

Write YAAC and LEX programs to check the given arithmetic expression can be generated by the above grammar.

Return values

In the above, very sketchy example, lex only returned the information that there was a number, not the actual number. For this we need a further mechanism. In addition to specifying the return code, the lex parser can return a value put on top of the stack, so that yacc can access it. This symbol is returned in the variable yylval. By default, this is defined as an int so that the lex program would have: extern int yylval;

```
%%
[0-9]+ {yylval=atoi(yytext); return NUMBER;}
```

Example:5

// sample program to print the integer value of NUM token returned by lexer in yacc code using yylval

Homework:

1. **Write a yacc and lex program to perform addition of two numbers.**