ASSIGNMENT 5

Name – Masood Ismail Tamboli
Roll no – 223081
GR.No - 22020190
Batch – C3

AIM: Write a program using YACC specifications to implement syntax analysis phase of compiler to recognize simple and compound sentences given in input file.

Problem Statement: Write a program using YACC specifications to implement syntax analysis phase of compiler to recognize simple and compound sentences given in input file.

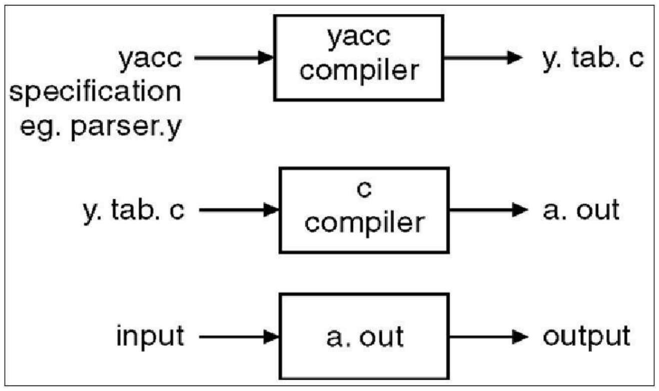
Theory: -

Yacc (**Yet Another Compiler-Compiler**) is a computer program for the Unix operating system developed by Stephen C. Johnson. It is a Look Ahead Left-to-Right (LALR) parser generator, generating a parser, the part of a compiler that tries to make syntactic sense of the source code, specifically a LALR parser, based on an analytic grammar written in a notation similar to Backus– Naur Form (BNF). Yacc is supplied as a standard utility on BSD and AT&T Unix. GNU- based Linux distributions include Bison, a forward-compatible Yacc replacement.

Yacc is one of the automatic tools for generating the parser program. Basically Yacc is a LALR parser generator. The Yacc can report conflicts or ambiguities (if at all) in the form of error messages. LEX and Yacc work together to analyse the program syntactically.

Yacc is officially known as a “parser”. Its job is to analyze the structure of the input stream, and operate of the “big picture”. In the course of it’s normal work, the parser also verifies that the input is syntactically sound.

YACC stands for “**Yet Another Compiler Compiler**” which is a utility available from
Unix.



Structure of a yacc file:

A yacc file looks much like a lex file:

...definitions..

%%

...rules...

%%

...code...

Yacc provides a general tool for imposing structure on the input to a computer program. The Yacc user prepares a specification of the input process; this includes rules describing the input structure, code to be invoked when these rules are recognized, and a low-level routine to do the basic input.

Yacc then generates a function to control the input process. This function, called a parser, calls the user-supplied low-level input routine (the lexical analyzer) to pick up the basic items (called tokens) from the input stream. These tokens are organized according to the input structure rules, called grammar rules; when one of these rules has been recognized, then user code supplied for this rule, an action, is invoked; actions have the ability to return values and make use of the values of other actions.

Basic Specifications:

Names refer to either tokens or non-terminal symbols. Yacc requires token names to be declared as such. In addition, it is often desirable to include the lexical analyzer as part of the specification file; it may be useful to include other programs as well.

Thus, every specification file consists of three sections: the declarations, (grammar) rules, and programs. The sections are separated by double percent ``%%'' marks. (The percent

``%" is generally used in Yacc specifications as an escape character.

In other words, a full specification file looks like declarations

```
%%
rules
%%
programs
```

The declaration section may be empty. Moreover, if the programs section is omitted, the second %% mark may be omitted also;
thus, the smallest legal Yacc specification is

```
%%
rules
```

Blanks, tabs, and newlines are ignored except that they may not appear in names or multi-character reserved symbols. Comments may appear wherever a name is legal; they are enclosed in /* . . . */, as in C and PL/I.

The rules section is made up of one or more grammar rules.

A grammar rule has the form: A : BODY ;
A represents a non-terminal name, and BODY represents a sequence of zero or more names and literals. The colon and the semicolon are Yacc punctuation.

Names may be of arbitrary length, and may be made up of letters, dot ``.", underscore ``_", and non-initial digits. Upper and lower case letters are distinct. The names used in the body of a grammar rule may represent tokens or non-terminal symbols.

Comparing Sentence Types

Sentences give structure to language, and in English, they come in four types: simple, compound, complex and compound-complex. When you use several types together, your writing is more interesting. Combining sentences effectively takes practice, but you'll be happy with the result.

1. The **simple sentence** is an independent clause with one subject and one verb.
   For example: we are the indian.

2. The Compound sentence is two or more independent clause, joined with comma, semicolon & conjuction.

Code: -

sentence.l (LEX File):

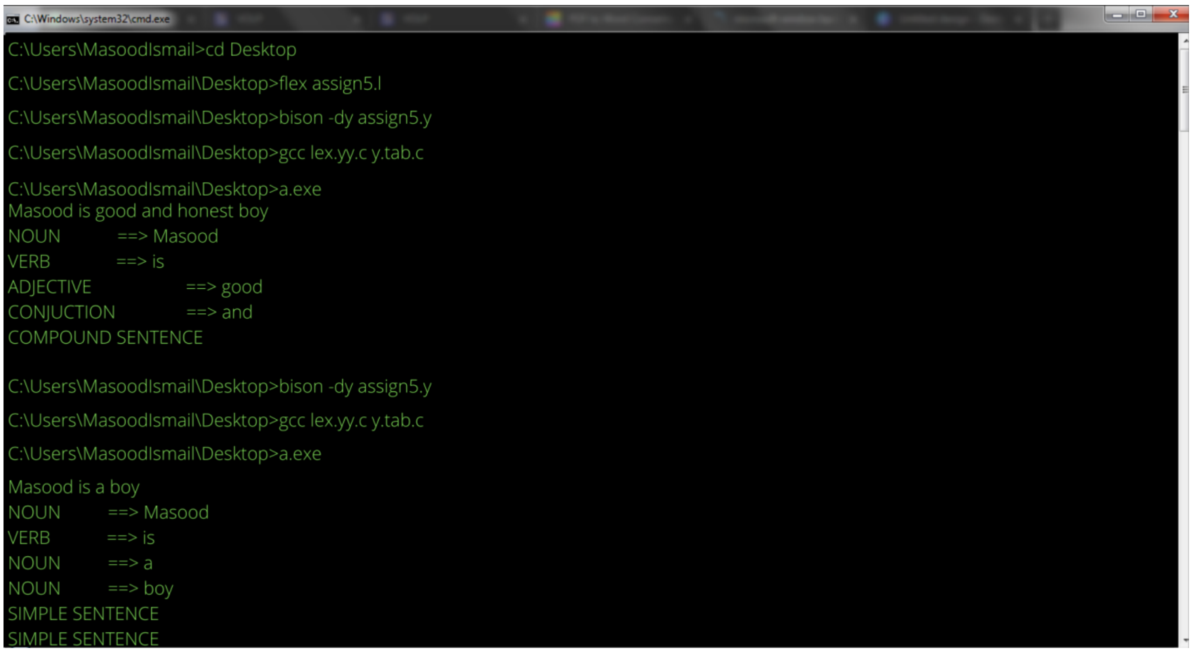
```
%{
#include "y.tab.h"
%}
%%
[\t ] ;    //IGNORE WHITE SPACES
am|is|are|have|has|can|will|shall|eat|sing|go|goes {
printf("VERB\t==>%s\n",yytext);return VERB;}
very|simply|gently { printf("VERB\t==>%s\n",yytext);return(ADVERB); }
and|or|also|so|but|if|then
{printf("CONJUNCTION\t==>%s\n",yytext);return (CONJUNCTION);}
fast|good|honest {printf("ADJECTIVE\t==>%s\n",yytext);return (ADJECTIVE);}
I|he|she|we|they|you|this {printf("PRONOUN\t==>%s\n",yytext);return
(PRONOUN);} in|on|to {printf("PREPOSITION\t==>%s\n",yytext);return
(PREPOSITION);}
[a-zA-Z]+ {printf("NOUN\t==>%s\n",yytext);return (NOUN);}
. ;
%%
int yywrap()
{
return 1;
}
```

sentence.y (YACC File):

```
%{
#include<stdio.h>
void
yyerror(char*); int
yylex();
FILE* yyin;
%}

%token NOUN PRONOUN ADJECTIVE VERB ADVERB CONJUNCTION
PREPOSITION

%%
sentence: compound { printf("COMPOUND SENTENCE\n");}
        |
        simple {printf("SIMPLE SENTENCE\n");}
        ;
simple: subject VERB object;
```

```
compound: subject VERB object CONJUNCTION subject

VERB object; subject: NOUN|PRONOUN;

object: NOUN|ADJECTIVE NOUN|ADVERB NOUN|PREPOSITION NOUN;
%%
void yyerror(char *s)
{
printf("ERROR:%s",s);
}
int main(int argc,char* argv[])
{
yyin=fopen(argv[1],"r"
); yyparse();
fclose(yyin
); return 0;
}
```

Output: -



```
C:\Users\MasoodIsmail>cd Desktop

C:\Users\MasoodIsmail\Desktop>flex assign5.l

C:\Users\MasoodIsmail\Desktop>bison -dy assign5.y

C:\Users\MasoodIsmail\Desktop>gcc lex.yy.c y.tab.c

C:\Users\MasoodIsmail\Desktop>a.exe
Masood is good and honest boy
NOUN          ==> Masood
VERB          ==> is
ADJECTIVE              ==> good
CONJUCTION             ==> and
COMPOUND SENTENCE

C:\Users\MasoodIsmail\Desktop>bison -dy assign5.y

C:\Users\MasoodIsmail\Desktop>gcc lex.yy.c y.tab.c

C:\Users\MasoodIsmail\Desktop>a.exe

Masood is a boy
NOUN        ==> Masood
VERB        ==> is
NOUN        ==> a
NOUN        ==> boy
SIMPLE SENTENCE
SIMPLE SENTENCE
```