7th Assignment

Aim: Design lex & Yacc prog to validate type and syntax of var declaration in Java.

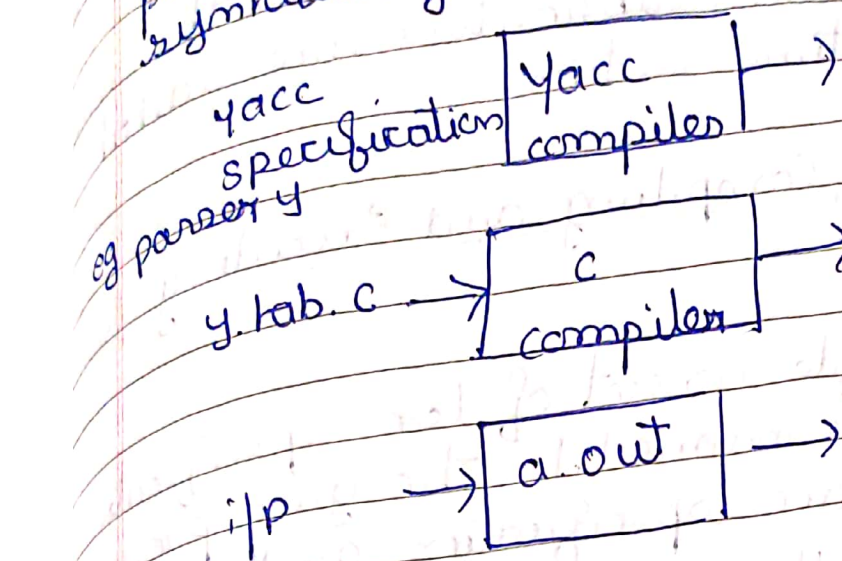
Problem statement: WAP using Yacc ~~statem~~ specifier to implement lexical analysis Phase of compiler to validate type and syntax of var declaration in Java.

Sw Requirement: OS: Ubuntu
Sw: FLEX, YACC

Theory:

① - Yacc (Yet another Compiler-Compiler) is a computer program for the UNIX OS developed by Stephen C. Johnson.
- It is a look Ahead Left to Right (LALR) parser generator generating a parser the part of the compiler that tries to make syntatic sense of the source code specifically of a LALR parser based on a analytic grammer written in the notation similar to Backus Naur form (BNF)
- Yacc is supplied as a std utility based on BSP and AT&T UNIX
- ~~LEX~~ LEX and Yacc work together to analyze the structure of the i/p stream and

operate of "big picture"
- In the course of its normal work the parser also verifies that the i/p is syntatically sound.



Structure of Yacc file

A yacc file looks much like a lex file

```
... definition ...
%%
... rules ...
%%
... code ...
```

- As with lex all the code between %{ and %} is copied to the begning of resulting c file
- Rules as with no of combination of pattern and action
- The patterns are now those of a context free grammer rather than of a regular grammer as was the 3 case with lex code

- Input to yacc is divided in 3 sections
- The definition section consists of declarations and ccode braketed by "%{ and %}"
- The BNF grammes is placed in the rules section and user subroutine are added in subroutine section

③ Translating, Compiling and Executing Yacc Program.

- The lex prog file consist of lex specification and should be named .l. The yacc prog file consists of yacc specification and should be named .y
- following command may be issued generate the parser.

```
Lex <filename>.l
Yacc <filename>.y
cc lex.yy.c y.tab.c -ll
./a.out
```

- Yacc reads the grammes description .y and generates the parser function yyparse in file y.tab.c
- The -d option causes yacc to generate the definition for tokens that are declared in the .y and place them in file y.tab.h
- Lex reads the pattern description in .l includes files y.tab.h and generates a lexical analyzer function yylex in file lex.yy.c

- Finally the lexer and the parser are compiled and linked (-ll) together to form the o/p file a.out

Lexical Analyzer for Yacc

- The parser and lexical analyzer must agree on the token nos in order for communication between them to take place.
- The nos may be chosen by Yacc or chosen by the user.
- In either case the '#define' mechanism of C is used to allow the lexical analyzer to return these nos symbolically

Eg

- Suppose that the token name DIGIT has been defined in the declaration section of Yacc specification file
- The relevant portion of the lexical analyzer might look like

```
yylex() {
extern int yylval;
int c;
...
c = getchar();
... switch (c) {
... case '0':
... case '1':
...
  case '9':
```

```
44| val = c - '0'
  return (DIGIT)
  ...}
  ...
```

- The intent is to return no of digit and a value equal to the numerical value of the digit
- Provided that the lexical analyzer code is placed in the programs section of the specification file
- The identifier DIGIT will be defined as the token no associated with the token digit

③ Comparing sentence types

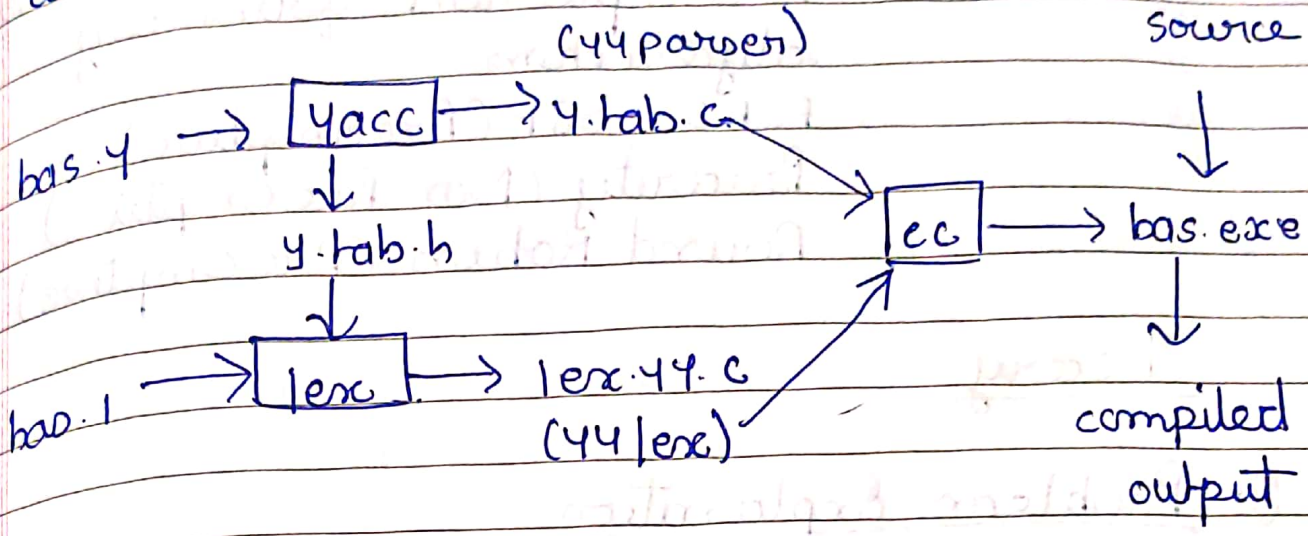
- Sentences give structure to lang and in English they come in 4 types simple compound and complex and complex compound
- When you use several types together your writing is more interesting
- Combining sentence effectively takes practise but you'll be happy with the result

① The simple sentence is independent clause with one subject and one verb.
Eg We are Indian.

② The compound sentence has two or more independent clause joined with ,,;& conjunctions

⑥ Application

- Yacc is used to generate parsers which are integral part of compiler



Conclusion

Thus we have studied lexical analyzer, syntax analysis and implemented Lex & Yacc application for syntax analyzer to validate the given infix expression

NAME: Ankita Sachin Bonde
ROLL:19
TE A
****

ASSIGNMENT 7

Write a program using YACC specifications to implement syntax analysis phase of compiler to validate type and syntax of variable declaration in Java.

****

/*valid.l program*/

```
%{

        #include "y.tab.h"
        #include<string.h>
        #include<stdio.h>
%}

letter [A-Za-z]
digit [0-9]

%%

"int"                   {strcpy(yylval.str,yytext); return INT;}
"char"                  {strcpy(yylval.str,yytext); return CHAR;}
"float"                 {strcpy(yylval.str,yytext); return FLOAT;}
[ ]                             ;
[,]                             {return ',';}
[;]                             {return ';';}

{letter}({letter}|{digit})*     {strcpy(yylval.str,yytext);
                                        return ID;}
\n                              {return '\n';}
.                               {return yytext[0];}

%%
void yyerror(char *msg)
{
        printf("\nError: %s",msg);
}
```

****

/*valid.y program*/

```
%{
        #include<stdio.h>
        #include<string.h>
        void yyerror(char *);
        int yylex(void);
        char q[10][10];
        int qindex=0;
        struct symtab
```

```
	{
		char type[10],symbol[10];
	}sym[10];
	int sindex=0;
%}

%union
{
	char str[10];
};

%type <str> DL T L
%token <str> ID INT CHAR FLOAT
%token <str> '\n'

%%

LINE:DL '\n'		{printf("Valid declaration \n");return 0;}
   |	DL '\n' LINE
	;

DL:T L ';'		{
				for(int i=0;i<qindex;i++)
				{
					strcpy(sym[sindex].type,$1);
					strcpy(sym[sindex].symbol,q[i]);
					sindex++;
				 }
				 qindex=0;
			}
   ;

T:INT
  |CHAR
  |FLOAT

  ;

L:ID			{		strcpy(q[qindex],$1); qindex++;   }
 |L ',' ID		{		strcpy(q[qindex],$3); qindex++;   }
 ;

%%

int main()
{
	printf("Enter the Declarative Statement: \n");
	yyparse();
	printf("DATA TYPE\tVARIABLE\n");
	for(int i=0;i<sindex;i++)
	{
		printf("%s\t\t%s \n",sym[i].type,sym[i].symbol);
	}
```

```
        return 0;
}
```
****

OUTPUT:

```
ankita@ankita-1011PX:~/Desktop/ankita_SPOS/Turn1/A3$  lex valid.l
ankita@ankita-1011PX:~/Desktop/ankita_SPOS/Turn1/A3$  yacc -d valid.y
ankita@ankita-1011PX:~/Desktop/ankita_SPOS/Turn1/A3$ gcc lex.yy.c y.tab.c -ll
ankita@ankita-1011PX:~/Desktop/ankita_SPOS/Turn1/A3$./a.out
Enter the Declarative Statement:
char  a;
int b,c;
float z;
char x;

Valid declaration
DATA TYPE  VARIABLE
char            a
int             b
int             c
float           z
char            x
```

****