Experiment - 9

Aim: Convert the BNF rules into YACC form and write code to generate abstract syntax tree.

Program:

bnf.l:

```
%{
   #include"y.tab.h"
   #include<stdio.h>
   #include<string.h>
   int LineNo=1;
%}
identifier [a-zA-Z][_a-zA-Z0-9]*
number [0-9]+|([0-9]*\.[0-9]+)
%%
main\(\) return MAIN;
if return IF;
else return ELSE;
while return WHILE;
int | char | float return TYPE;
{identifier} {strcpy(yylval.var,yytext); return VAR;}
{number} {strcpy(yylval.var,yytext); return NUM;}
'<' | '>' | ">=" | "<=" | "==" {strcpy(yylval.var,yytext); return RELOP;}
[ \t] ;
\n LineNo++;
. return yytext[0];
%%
```

bnf.y:

```
%{
   #include<string.h>
   #include<stdio.h>
```

```
    struct quad{
        char op[5]; char arg1[10]; char arg2[10]; char result[10];
    }QUAD[30];
    struct stack{
        int items[100]; int top;
    }stk;
    int Index=0,tIndex=0,StNo,Ind,tInd;
    extern int LineNo;
%}

%union{
    char var[10];
}
%token <var> NUM VAR RELOP
%token MAIN IF ELSE WHILE TYPE
%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP
%left '-' '+'
%left '*' '/'
%%
PROGRAM : MAIN BLOCK ;
BLOCK: '{' CODE '}' ;
CODE: BLOCK | STATEMENT CODE | STATEMENT ;
STATEMENT: DESCT ';' | ASSIGNMENT ';' | CONDST | WHILEST ;
DESCT: TYPE VARLIST ;
VARLIST: VAR ',' VARLIST | VAR ;
ASSIGNMENT: VAR '=' EXPR{
    strcpy(QUAD[Index].op,"=");
    strcpy(QUAD[Index].arg1,$3);
    strcpy(QUAD[Index].arg2,"");
    strcpy(QUAD[Index].result,$1);
```

```
    strcpy($$,QUAD[Index++].result);
} ;

EXPR: EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);}
| EXPR '-' EXPR {AddQuadruple("-",$1,$3,$$);}
| EXPR '' EXPR {AddQuadruple("",$1,$3,$$);}
| EXPR '/' EXPR {AddQuadruple("/",$1,$3,$$);}
| '-' EXPR {AddQuadruple("UMIN",$2,"",$$);}
| '(' EXPR ')' {strcpy($$,$2);}
| VAR | NUM ;

CONDST: IFST{
    Ind=pop();
    sprintf(QUAD[Ind].result,"%d",Index);
    Ind=pop();
    sprintf(QUAD[Ind].result,"%d",Index);
}
| IFST ELSEST ;
IFST: IF '(' CONDITION ')' {
    strcpy(QUAD[Index].op,"==");
    strcpy(QUAD[Index].arg1,$3);
    strcpy(QUAD[Index].arg2,"FALSE");
    strcpy(QUAD[Index].result,"-1");
    push(Index);
    Index++;
}
BLOCK {
    strcpy(QUAD[Index].op,"GOTO");
    strcpy(QUAD[Index].arg1,"");
    strcpy(QUAD[Index].arg2,"");
```

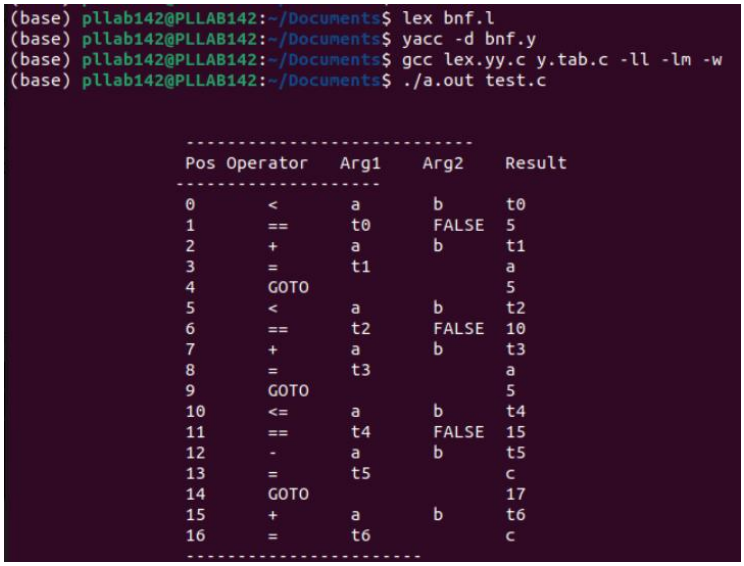
```
    strcpy(QUAD[Index].result,"-1");
    push(Index);
    Index++;
};
ELSEST: ELSE{
    tInd=pop();
    Ind=pop();
    push(tInd);
    sprintf(QUAD[Ind].result,"%d",Index);
}
BLOCK{
    Ind=pop();
    sprintf(QUAD[Ind].result,"%d",Index);
};
CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$); StNo=Index-1; }
| VAR | NUM ;
WHILEST: WHILELOOP{
    Ind=pop();
    sprintf(QUAD[Ind].result,"%d",StNo);
    Ind=pop();
    sprintf(QUAD[Ind].result,"%d",Index);
}
;
WHILELOOP: WHILE '(' CONDITION ')' {
    strcpy(QUAD[Index].op,"==");
    strcpy(QUAD[Index].arg1,$3);
    strcpy(QUAD[Index].arg2,"FALSE");
    strcpy(QUAD[Index].result,"-1");
    push(Index);
    Index++;
```

```
}
BLOCK {
    strcpy(QUAD[Index].op,"GOTO");
    strcpy(QUAD[Index].arg1,"");
    strcpy(QUAD[Index].arg2,"");
    strcpy(QUAD[Index].result,"-1");
    push(Index);
    Index++;
} ;
%%
extern FILE *yyin;
int main(int argc,char *argv[]) {
    FILE *fp;
    int i;
    if(argc>1) {
        fp=fopen(argv[1],"r");
        if(!fp) {
            printf("\n File not found");
            exit(0);
        }
        yyin=fp;
    }
    yyparse();
    printf("\n\n\t\t ----------------------------""\n\t\t Pos Operator Arg1 Arg2
Result" "\n\t\t --------------------");
    for(i=0;i<Index;i++) {
        printf("\n\t\t %d\t %s\t %s\t %s\t
%s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result);
    }
    printf("\n\t\t ------------------------"); printf("\n\n");
    return 0;
```

```
}

void push(int data){
    stk.top++;
    if(stk.top==100)  {
        printf("\n Stack overflow\n");
        exit(0);
    }
    stk.items[stk.top]=data;
}
int pop() {
    int data;
    if(stk.top==-1) {
        printf("\n Stack underflow\n");
        exit(0);
    }
    data=stk.items[stk.top--];
    return data;
}
void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10]) {
    strcpy(QUAD[Index].op,op);
    strcpy(QUAD[Index].arg1,arg1);
    strcpy(QUAD[Index].arg2,arg2);
    sprintf(QUAD[Index].result,"t%d",tIndex++);
    strcpy(result,QUAD[Index++].result);
}
yyerror() { printf("\n Error on line no:%d",LineNo); }
```

Output:


(base) pllab142@PLLAB142:~/Documents$ lex bnf.l
(base) pllab142@PLLAB142:~/Documents$ yacc -d bnf.y
(base) pllab142@PLLAB142:~/Documents$ gcc lex.yy.c y.tab.c -ll -lm -w
(base) pllab142@PLLAB142:~/Documents$ ./a.out test.c
Pos Operator Arg1 Arg2 Result
0 < a b t0
1 == t0 FALSE 5
2 + a b t1
3 = t1 a
4 GOTO 5
5 < a b t2
6 == t2 FALSE 10
7 + a b t3
8 = t3 a
9 GOTO 5
10 <= a b t4
11 == t4 FALSE 15
12 - a b t5
13 = t5 c
14 GOTO 17
15 + a b t6
16 = t6 c