01.11-2021 COMPILER DESIGN SACHIN RAGHUL T

OBSERVATION - 08 2019103573

1. Write a yacc program to implement arithmetic operations.

```
%{
    #include <ctype.h>
    #include <stdio.h>
    int yylex();
    void yyerror(const char *c);
%}
%token NUBER
% left '+''-'
% left '*''/'
% right UMINUS
%%
lines : expr '\n'
lines expr '\n'
|error '\n' { yyerror ("Enter again : "); yerror; }
;
expr : expr '+' term { $$ = $1 + $3 ; printf("%d\n", $$); }
|expr '-' term { $$ = $1 - $3 } printf ("%d\n", $$); }
|term
;
term : term '*' fac { $$ = $1 * $ 3 ; printf("%d\n", $$ ); }
|term '/' fac { $$ = $1 / $3 ; printf("%d\n", $$) ; }
| fac
;
fac : '(' expr ')' { $$ = $2 ; }
|NUMBER
;
%%
yylex(){
    int c ;
    c = getchar();
    if (isdigit(c)){
        yyval = c-'0';
    return NUBBER;
    }
    return c;
}
```

```
void yyerror (char const *s) {
    fprintf (stderr, "%s\n", s);
}
int main (void) {
    do {
      yyparse ();
    } while (1);
    return 0;
}
```

OUTPUT :-

```
14+52
66
2*3
6
9-2
7
16/2
8
```

2. Write a yacc program to implement logical operations.

CODE :-

```
%{
  #include <ctype.h>
  #include <stdio.h>
  int yylex();
  void yyerror (const char *s);
%}
% token DIGIT
% left '&' '|'
% right '!'
%%
lines : expr '\n'
lines : expr '\n'
|error '\n' {yyerror ("Enter again : "); yyerror ; }
;
expr : expr '&' fac { $$ = $1 & $3; printf("%d\n", $$); }
|expr '|' fac { $$ = $1 | $3 ; printf ("%d\n", $$ ); }
|'!' expr { $$ = ! $2 ; printf ("%d\n", $$ ); }
| fac
;
```

```
| ac : '(' expr ')' { $$ = $2 ; }
| DIGIT
;
%%
yylex () {
  int c ;
  c = getchar();
  if(isdigit(c)) {
      yylval = c - '0';
      return DIGIT;
    }
    return c ;
 }
void yyerror(char const *s){
  fprintf( stderr, "%s\n", s);
}
int main(void) {
  do {
   yyparse();
  } while(1);
  return 0;
}
```

OUTPUT :

```
0 & 1
0
1 & 1
1
0 | 0
0
1 | 0
1
! 1
0
! 0
1
```

3. To write a yacc problem to check the syntax of for expression.

CODE :-

```
%{
 #include <stdio.h>
 #include <y.tab.h>
%}
alpha [A-Z a-z]
digit [0-9]
%%
[\t \n]
for return FOR;
{digit}+ return NUM;
{alpha}({alpha}|{digit})* return ID;
"<=" return LE;
">=" return GE;
"==" return EQ;
"!=" return NE;
"||" return OR;
"&&" return AND;
.    return yytext[0];
%%
%{
 #include <stdio.h>
 #include <stdlib.h>
%}
$token ID NUM for LE GE EQ NE OR AND
$right "-"
$left OR AND
$left '>' '<' LE GE EQ NE
$left '+' '-'
$left 'x' '/'
$right UMINUS
$left '!'
```

```
%%
S : ST { printf("Input accepted\n"); exit(0); }
    ST : FOR '(' E ';' E2 ';' E ')' DEF ;
    DEF : '{' BODY '}'
|';'
|ST
|
;
E : ID '-' E
|E '+' E
|E '-' E
|E '*' E
|E '\' E
|E '<' E
|E '>' E
|E LE E
|E GE E
|E EQ E
|E NE E
|E OR E
|E AND E
|E '+' '+'
|E '-' '-'
| ID
| NUM
;
EQ : E '<' E
|E : E '>' E
|E LE E
|E GE E
|E EQ E
|E NE E
|E OR E
|E AND E
;
%%
```

```
void main() {
    printf("Enter the expression :\n");
    yyparse();
}
int yywrap() { }
void yyerror() {
    printf("\nInvalid syntax \n\n");
}
```

OUTPUT :

```
Enter the expression :
for(i=0 ;i<10 ;i++)
Input accepted
```