**Ex. No. 6 GENERATION OF ABSTRACT SYNTAX TREE**

**Date:**

**Aim:**

To generate YACC specification to recognize a valid arithmetic expression and convert it to an abstract syntax tree.

**Algorithm:**

1. Start the program.
2. Specify RE’s to recognize lexemes and classify the lexemes into tokens and return to syntax analyzer.
3. In syntax analysis, get the token and also specify the precedence of the operators.
4. Specify the production rules for the validation of Arithmetic expression.
5. Print the result of the expression.
6. Stop the program.

**Program:**

**exno6.l**

%{

#include<stdio.h>

#include"y.tab.h"

extern yylval;

%}

%%

[\_]?[a-zA-Z\_][a-zA-z0-9\_]\* {yylval=yytext[0];return ID;}

[0-9] {yylval=atoi(yytext);return NUM;}

[+-] {return \*yytext;}

[\*/] {return \*yytext;}

[\t]+;

[()] {return \*yytext;}

[\n] {return 0;}

. {return yytext[0];}

%%

int yywrap()

{

return 1;

}

**exno61.y**

%{

#include<stdio.h>

int i=0;

%}

%token ID

%token NUM

%left '\*'

%left '/'

%left '+'

%left '-'

%%

start: ID {$$=$1;printf("Valid id %d",$1);}

| expr {printf("Result of expression is: %d\n",$$);}

;

expr: expr '+' term {$$=$1+$3;i=i+1;printf("Next tree node: %d %d %c %d Value:%d\n",i,$1,'+',$3,$$);}

| expr '-' term {$$=$1-$3;i=i+1;printf("Next tree node: %d %d %c %d Value:%d\n",i,$1,'-',$3,$$);}

| term {$$=$1;}

;

term: term '\*' factor {$$=$1\*$3;i=i+1;printf("Next tree node: %d %d %c %d value:%d\n",i,$1,'\*',$3,$$);}

| term '/' factor {$$=$1/$3;i=i+1;printf("Next tree node: %d %d %c %d value:%d\n",i,$1,'/',$3,$$);}

| factor {$$=$1;}

;

factor: NUM {$$=$1;printf("Leaf node: %d\n",$1);}

| '(' expr ')' {$$=$2;}

;

%%

void main()

{

printf("\nEnter Expression :\n");

yyparse();

}

yyerror()

{

printf("Invalid Expression\n");

}

**Output:**

welcome@welcome-HP-Notebook:~/Desktop/kavin$ lex exno6.l

welcome@welcome-HP-Notebook:~/Desktop/kavin$ yacc -d exno61.y

welcome@welcome-HP-Notebook:~/Desktop/kavin$ gcc lex.yy.c y.tab.c

welcome@welcome-HP-Notebook:~/Desktop/kavin$ ./a.out

Enter Expression :

(3+1)+(5\*5)/(7-2)

Leaf node: 3

Leaf node: 1

Next tree node: 1 3 + 1 Value:4

Leaf node: 5

Leaf node: 5

Next tree node: 2 5 \* 5 value:25

Leaf node: 7

Leaf node: 2

Next tree node: 3 7 - 2 Value:5

Next tree node: 4 25 / 5 value:5

Next tree node: 5 4 + 5 Value:9

Result of expression is: 9

**Result:**

Thus, the yacc specification to generate the abstract syntax tree for validating the arithmetic expression was implemented, executed and output was verified successfully.