Expt. No. 3 Page No. 7

Expt. Name. Use LEX and YACC to implement parser on unambiguous grammar Date :

Aim.

To write the program using LEX and YACC to implement parser on umambiguous grammar.

Algorithm:

File.l:

step 1: start

step 2: Include the necessary header files and declare the necessary variables

step 3: initialize the digits, operators, parenthesis and return the value else print syntax error.

step 4: Call the function and return 1

step 5: stop

File.y

step 1: start

step 2: Include the necessary header files and declare the necessary variables.

step 3: Substitute the values and calculate respectively for Addition, Subtraction, Multiplication and division and return the result.

step 4: Call the main function and print the result.

step 5: stop

Expt. No. ____________ Page No. 8

Expt. Name. ____________ Date : ____________

Program: → File.l

```
% option noyywrap
%{
        #include <stdio.h>
        #include "y.tab.h"
        void yyerror (char *s);
        extern int yylval;
 %}
 %%
[0-9]+    {yylval = atoi (yytext); return NUM; }
[a-z]     {yylval = toascii (*yytext)-97 ; return ID;}
[A-Z]     {yylval = toascii (*yytext)-97; return ID;}
[-+*\n]    {return *yytext;}
"("        {return *yytext;}
")"        {return *yytext;}
[ \t];
.         {yyerror("Syntax Error");}
%%
int yywrap()
 {
  return 1;
 }
```

File.y

```
%{
        #include <stdio.h>
        extern int yylex (void);
        void yyerror(char *);
        int x=0;
```

Expt. No. ____________ Page No.

Expt. Name. ____________ Date : ____________

```
        int val [26];
 %}
 % token NUM ID
 %%
 S:
 S expr '\n'    {x =$2; print ("%d\n", $2);}
| S ID '=' expr '\n'     {val [$2]=$4;}
#
;
expr:
 expr '+' T          {$$ = $1+$3;}
| expr '-' T         {$$ = $1-$3;}
| T                  {$$ = $1;}
| '+' T              {$$ = x+$2;}
| '-' T              {$$ = x-$2;}
;
T:
 F                   {$$ = $1;}
| T '*' F            {$$ = $1 * $3;}
| T '/' F            {$$ = $1 / $3;}
| '*' F              {$$ = x * $2;}
| '/' F              {$$ = x / $2;}
;
F:
NUM                  {$$ = $1;}
| ID                 {$$ =val [$1];}
| '(' expr ')'       {$$ = $2;}
;
%%
```

Expt. No.

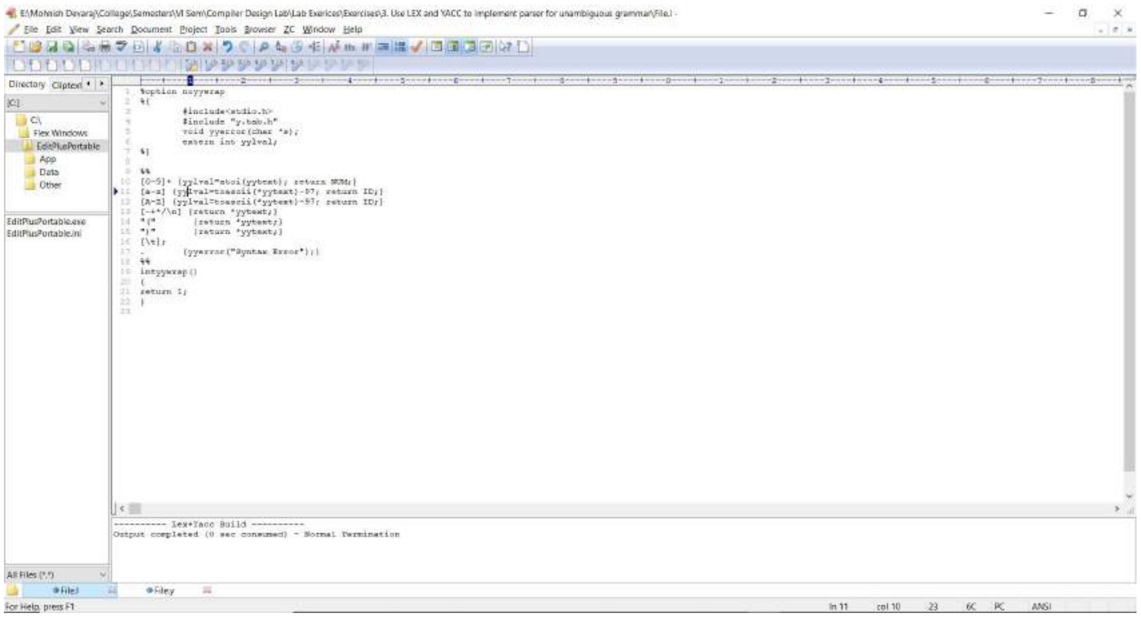
Expt. Name.

Date :

```
void yyerror (char *s)
{
  printf("%s", s);
 }

int main()
{
yyparse();
return 0;
}
```

Result:

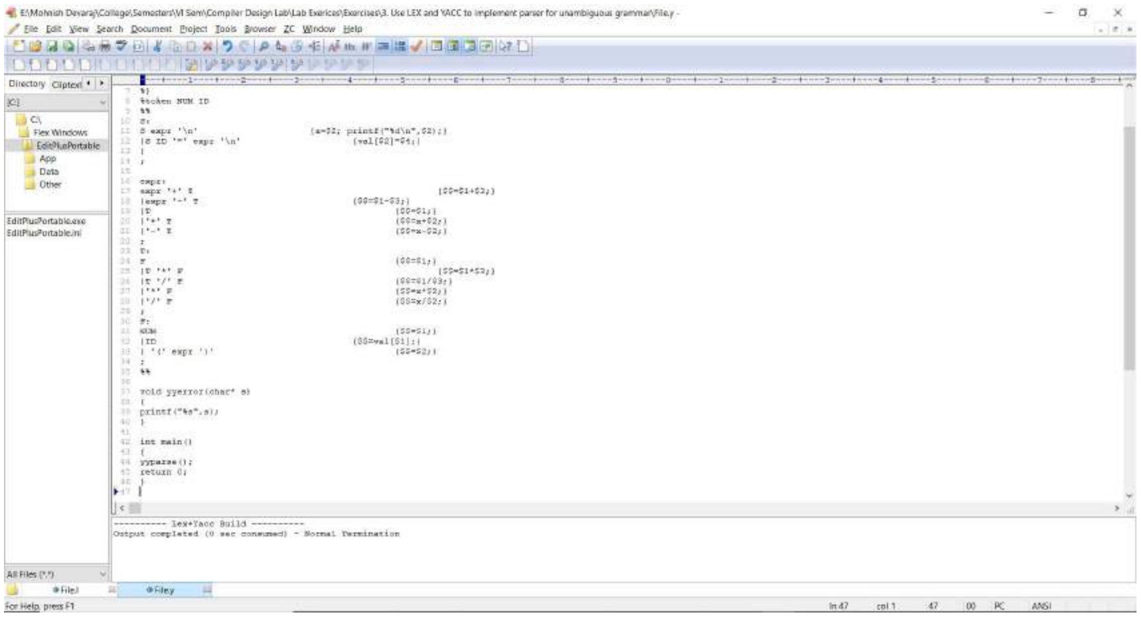
Use LEX and YACC to implement parser for unambiguous is executed successfully.


E:\Mohnish Devaraj\College\Semesters\VI Sem\Compiler Design Lab\Lab Exerices\Exercises\3. Use LEX and YACC to implement parser for unambiguous grammar\File.l -

```
%option noyywrap
%{
        #include<stdio.h>
        #include "y.tab.h"
        void yyerror(char *s);
        extern int yylval;
%}

%%
[0-9]+ {yylval=atoi(yytext); return NUM;}
[a-z] {yylval=toascii(*yytext)-97; return ID;}
[A-Z] {yylval=toascii(*yytext)-97; return ID;}
[-+*/\n] {return *yytext;}
"("        {return *yytext;}
")"        {return *yytext;}
[\t];
.        {yyerror("Syntax Error");}
%%
intyywrap()
{
return 1;
}
```

```
---------- Lex+Yacc Build ----------
Output completed (0 sec consumed) - Normal Termination
```

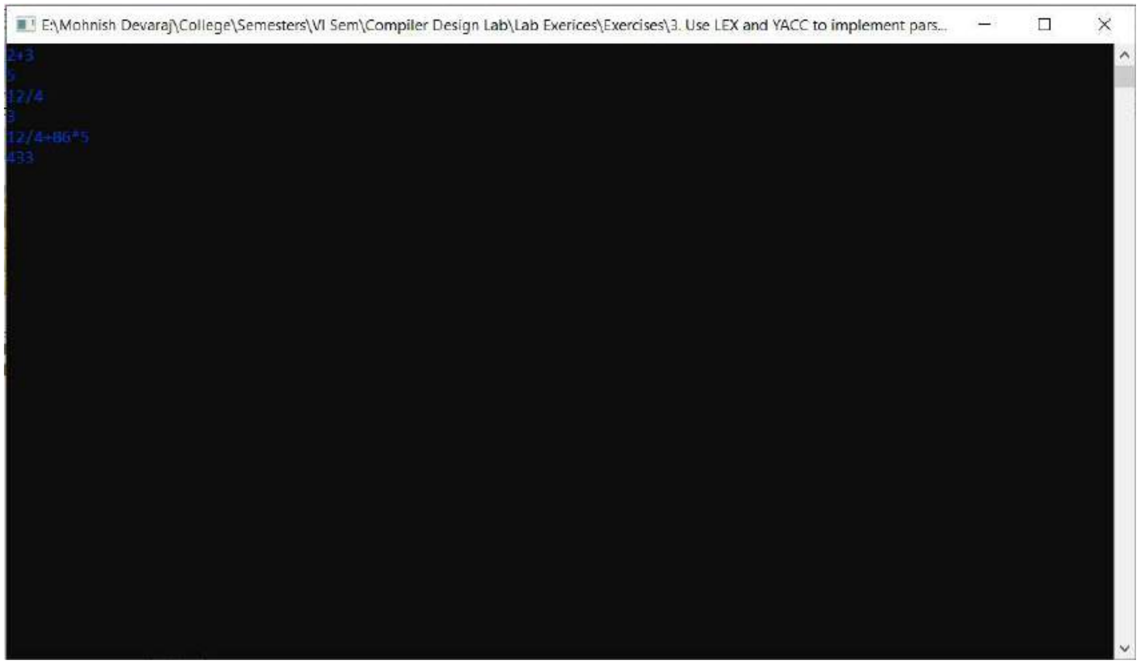
In 11 col 10 23 6C PC ANSI


E:\Mohnish Devaraj\College\Semesters\VI Sem\Compiler Design Lab\Lab Exerices\Exercises\3. Use LEX and YACC to implement parser for unambiguous grammar\File.y -

```
%}
%token NUM ID
%%
S:
S expr '\n'                    {x=$2; printf("%d\n",$2);}
|S ID '=' expr '\n'                    {val[$2]=$4;}
|
;

expr:
expr '+' T                                  {$$=$1+$3;}
|expr '-' T                    {$$=$1-$3;}
|T                                  {$$=$1;}
|'+' T                              {$$=x+$2;}
|'-' T                              {$$=x-$2;}
;
T:
F                                   {$$=$1;}
|T '*' F                                  {$$=$1*$3;}
|T '/' F                            {$$=$1/$3;}
|'*' F                              {$$=x*$2;}
|'/' F                              {$$=x/$2;}
;
F:
NUM                                 {$$=$1;}
|ID                            {$$=val[$1];}
| '(' expr ')'                      {$$=$2;}
;
%%

void yyerror(char* s)
{
printf("%s",s);
}

int main()
{
yyparse();
return 0;
}
```

```
---------- Lex+Yacc Build ----------
Output completed (0 sec consumed) - Normal Termination
```

In 47 col 1 47 00 PC ANSI



E:\Mohnish Devaraj\College\Semesters\VI Sem\Compiler Design Lab\Lab Exerices\Exercises\3. Use LEX and YACC to implement pars...

```
2+3
5
12/4
3
12/4+86*5
433
```