Don Bosco Institute of Technology, Kulra(W)
Department of Computer Engineering

CSL601: System Programming and Compiler Construction Lab2022-23

Name : Ronak Surve
Roll No : 64
Subject : SPCC Lab
Batch : C

Experiment No.:	06
Experiment Title:	Parser generator tool : Yacc. Task: Write a program to recognize valid arithmetic expression that uses operators + , -,* and / using YACC.
Student Name	Ronak Surve
Roll No.	64
Objectives :	1) Parser generator tool : Yacc. Task: Write a program to recognize valid arithmetic expression that uses operators + , -,* and / using YACC.

Theory /Algorithm :	YACC (Yet Another Compiler-Compiler) is a tool used for generating parsers or syntax analyzers for a programming language. It takes input in the form of a grammar specification and produces a parser in C or C++ code. The YACC tool works in two phases: Parsing Phase: YACC reads the grammar specification and builds a parsing table based on it. This parsing table is used to parse the input source code. Parsing Phase: YACC reads the input source code and applies the parsing table to generate a parse tree. The syntax analyzer is the part of the YACC-generated parser that performs the parsing of the input source code. The syntax analyzer reads the input source code and tries to match it with the production rules specified in the grammar. If there is a match, it generates a parse tree. If there is no match, it generates an error. The YACC tool provides several functions, including: Grammar Specification: YACC allows the specification of a grammar for a programming language using BNF (Backus-Naur Form) notation. Parser Generation: YACC generates a parser in C or C++ code based on the grammar specification. Error Recovery: YACC can be configured to recover from certain types of

errors in the input source code, making it more robust.

Symbol Table Management: YACC can generate code to manage a symbol table for the input source code, which can be used for semantic analysis and code generation.

Tree Traversal: YACC-generated parsers can traverse the parse tree to perform semantic analysis and generate code.

In YACC, production rules are defined using BNF notation. A production rule has a left-hand side (LHS) and a right-hand side (RHS). The LHS specifies a non-terminal symbol, while the RHS specifies a sequence of terminal and non-terminal symbols that can be derived from the non-terminal symbol. Here's an example of a production rule:

bash

Copy code

```
expr : expr '+' term

     | expr '-' term

     | term

     ;

term : term '*' factor

     | term '/' factor

     | factor

     ;

factor : '(' expr ')'

       | NUM

       ;
```

In this example, expr, term, and factor are non-terminal symbols, while +, -, *, /, (,), and NUM are terminal symbols. The expr production rule specifies that an expr can be derived from another expr followed by a + or - followed by a term, or just a term. The term production rule specifies that a term can be derived from another term followed by a * or / followed by a factor, or just a factor. The factor production rule specifies that a factor can be derived from a (followed by an expr followed by a), or just a NUM.

With these production rules, YACC can generate a parser that can parse arithmetic expressions and generate a parse tree.

Program Code:

scan.l

```
%{

#include<stdio.h>
#include "y.tab.h"
%}
%%
[a-zA-Z]+ return VARIABLE;
[0-9]+ return NUMBER;
[ \t]       ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

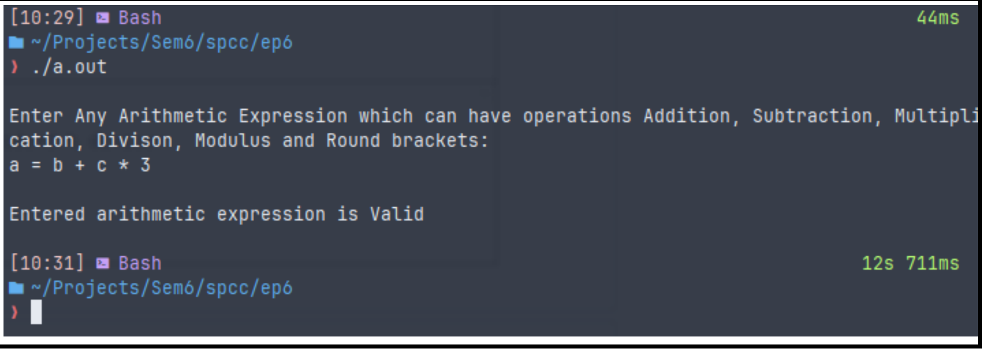
parser.y

```
%{
    #include<stdio.h>
%}
%token NUMBER
%token VARIABLE
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
%%
S: VARIABLE'='E {
       printf("\nEntered arithmetic expression is
Valid\n\n");
       return 0;
     }
E:E'+'E
 |E'-'E
 |E'*'E
 |E'/'E

 |E'%'E

 |'('E')'

 | NUMBER

 | VARIABLE

;

%%

void main()

{

   printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:\n");

   yyparse();

}

int  yyerror()

{

   printf("\nEntered arithmetic expression is Invalid\n\n");



}
```

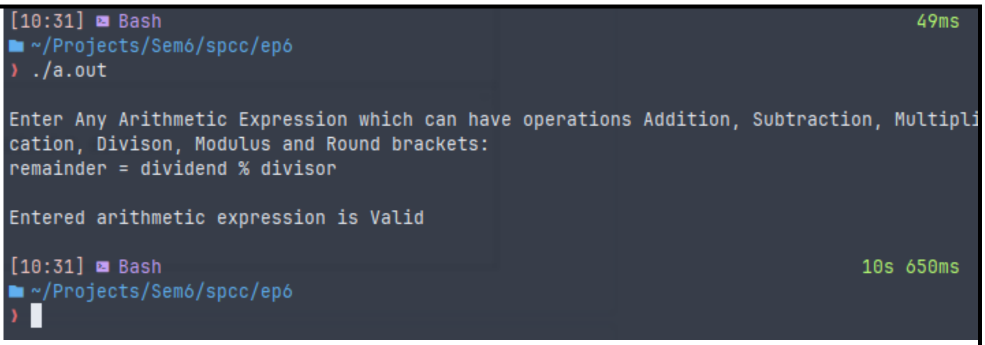
Input to the Program:

1. a = b + c * 3
2. remainder = dividend % divisor
3. p + q = 5
4. sum = 2 +5
5. res = ((A +B) * C - (D + E)) + F

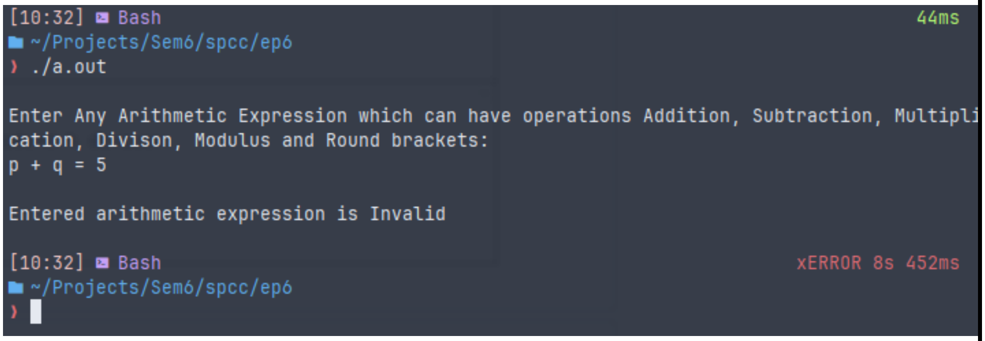
Output of the program:



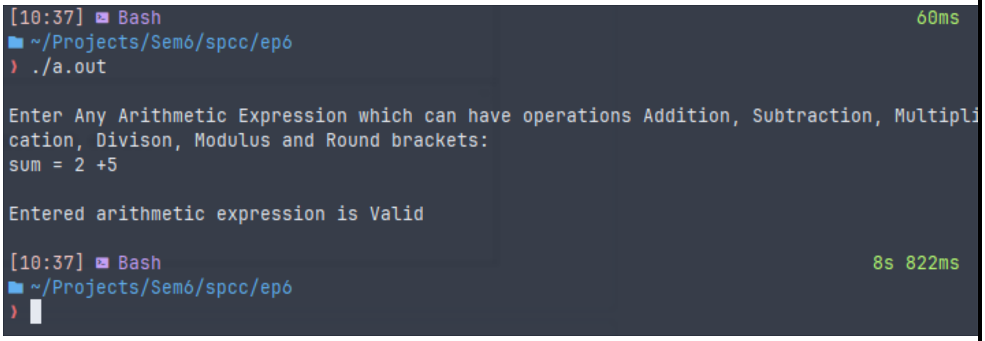
```
[10:29] Bash                                                                   44ms
~/Projects/Sem6/spcc/ep6
› ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multipli
cation, Divison, Modulus and Round brackets:
a = b + c * 3

Entered arithmetic expression is Valid

[10:31] Bash                                                              12s 711ms
~/Projects/Sem6/spcc/ep6
›
```



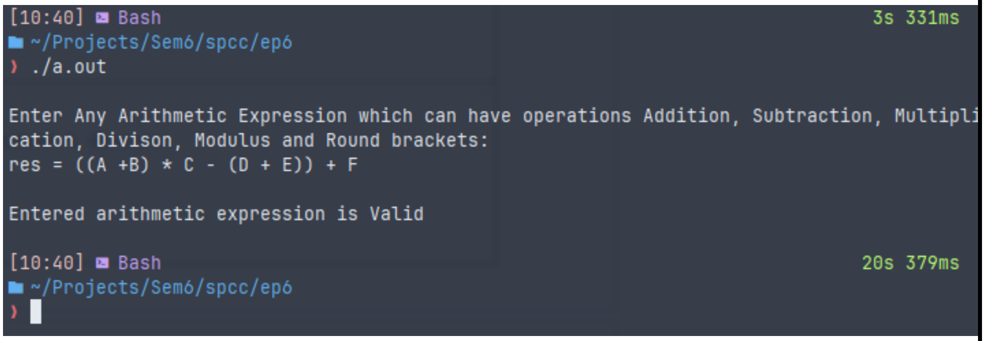
```
[10:31] Bash                                                                     49ms
~/Projects/Sem6/spcc/ep6
❯ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multipli
cation, Divison, Modulus and Round brackets:
remainder = dividend % divisor

Entered arithmetic expression is Valid

[10:31] Bash                                                                 10s 650ms
~/Projects/Sem6/spcc/ep6
❯
```



```
[10:32] Bash                                                                     44ms
~/Projects/Sem6/spcc/ep6
❯ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multipli
cation, Divison, Modulus and Round brackets:
p + q = 5

Entered arithmetic expression is Invalid

[10:32] Bash                                                          xERROR 8s 452ms
~/Projects/Sem6/spcc/ep6
❯
```



```
[10:37] Bash                                                                     60ms
~/Projects/Sem6/spcc/ep6
❯ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multipli
cation, Divison, Modulus and Round brackets:
sum = 2 +5

Entered arithmetic expression is Valid

[10:37] Bash                                                                  8s 822ms
~/Projects/Sem6/spcc/ep6
❯
```



```
[10:40] Bash                                                                  3s 331ms
~/Projects/Sem6/spcc/ep6
❯ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multipli
cation, Divison, Modulus and Round brackets:
res = ((A +B) * C - (D + E)) + F

Entered arithmetic expression is Valid

[10:40] Bash                                                                 20s 379ms
~/Projects/Sem6/spcc/ep6
❯
```

Outcome of the Experiment:	YACC is a tool used for generating parsers or syntax analyzers for a programming language. It is used with the FLex tool to create a simple compiler for validating source code syntax. In this experiment these two tools were used to create a compiler to check the validity of arithmetic expressions with all the basic operators. These expressions should also be valid for any programming language.
Reference:	Class notes https://www.geeksforgeeks.org/