COVER PAGE

PROJECT 2

CPSC 323:

Group 9

- **Joshua Ungheanu**
- **Derek Dorr**
- **Adam Weesner**

Assignment Number	**[2]**

Due Dates:

Softcopy	11/8 in class by 4:00
Hardcopy	11/8 titanium by 11:55pm

Executable FileName [CPSC323_Parser_Project2*.exe*]
(A file that can be executed without compilation by the instructor)

Operating System [*Windows 10*]

GRADE:

COMMENTS:

1. Problem Statement

The second assignment is to write a parser using the foundation built from our assignment 1 project. This parser will analyze the syntax written by an additional file, which is specified by the user. As the parser runs through the file, it will print the tokens, lexemes, and the production rules used. Furthermore, our program can detect faulty syntax errors and output those errors to the console in detail.

2. How to use your program

Can be done using a terminal from either; a Mac OSX, Linux, or titan server through Putty. Note that for method 2, In order to use the program, you should have your terminal setup to run an executable file. Look for the directory that contains the files to be tested (NOTE: using the terminal requires more steps). Once you have accessed to your directory, type the following command: cd [filepath] and hit enter, then type ["CPSC323_Parser_Project2".exe <file.txt>] which contains our syntax analyzer code. Our program will take an input of a .txt path, which will be used to analyze. Note that the .txt must be from the directory which contains our 3 test cases. In order to test more test cases, it is recommended to add those extra ".txt" files into your directory. Once the .txt path have been selected, hit enter and our program will then read the file and write the results to the terminal. The executable file should be working fine on windows OS only and was provided to satisfy the requirements of the assignment.

3. Design of your program

Our program uses the lexer built from the first project as the foundation for our syntax analyzer. For our analyzer, we used a top-down parser. Additionally, we styled our architecture using left factorization. We rewrote the grammar to remove left recursion and backtracking. As the parser scans the input file, it begins separating tokens and lexemes, outputting them to the console. Then we printed the production rules for each token. Error handling was accomplished using a method. If an error was found, it will print the message to the console, then move on to the next code segment. Once all segments are completed, the program will wait for the user to press a key to end.

4. Any Limitation

Our program was limited to less than 60 lines of source code. Any source code with more than 60 lines of code was not tested in this program.

5. Any shortcomings

Having to rewrite our Lexer with the set of grammar rules for our top down parser.

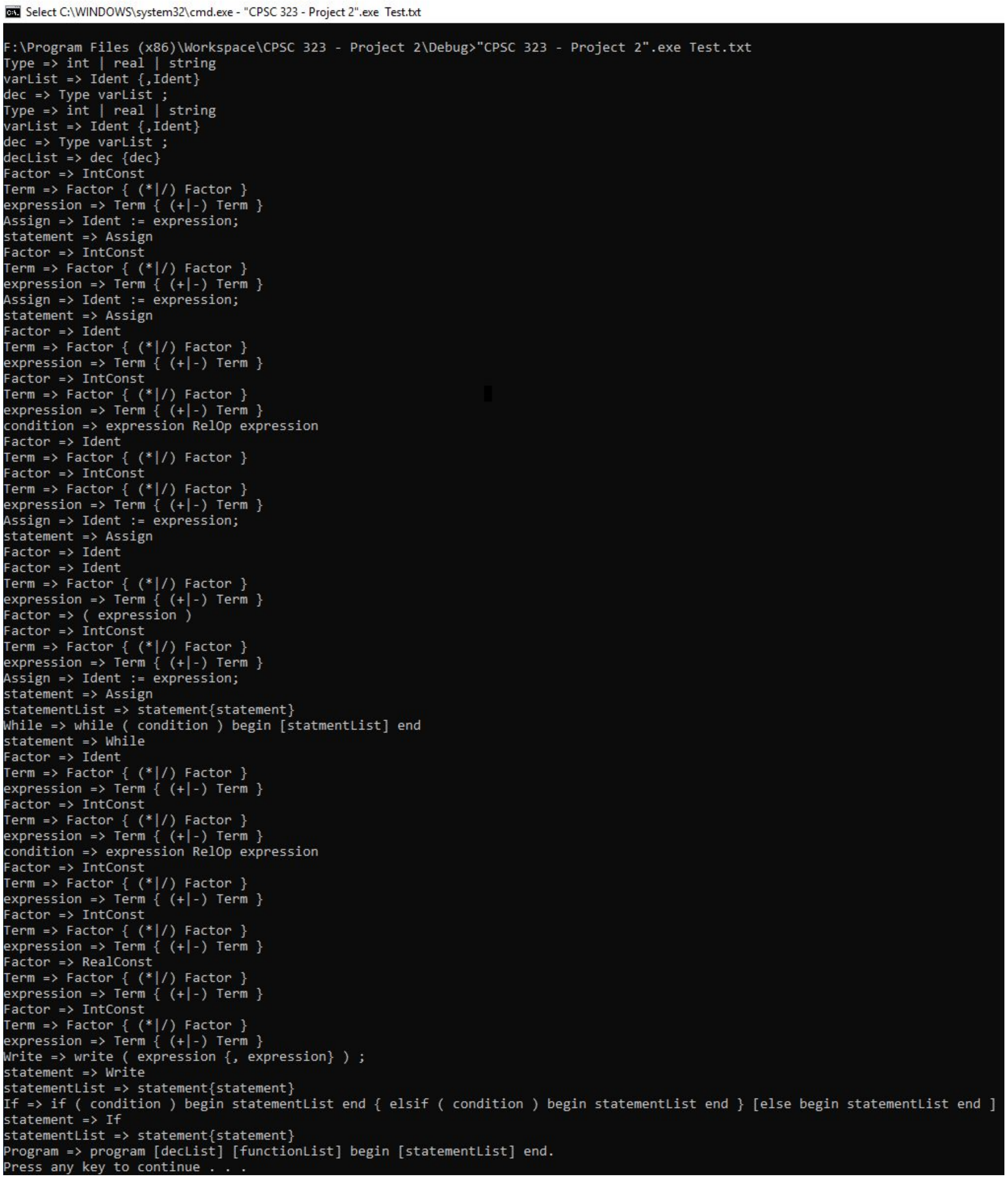
TEST CASES

Case - 1

<u>Input:</u>

```
program
   int myinteger;
   real foo , bar;
  begin
   foo := 1 ;
   bar := 2 ;
   while ( foo <> 10 )
     begin
          foo := foo + 1 ;
          bar := ( bar * foo ) / 2 ;
     end
   if ( bar > 100 )
     begin
          write( 1 , 3 , 3.5 , 7 ) ;
     end
  End.
```

Output:



Select C:\WINDOWS\system32\cmd.exe - "CPSC 323 - Project 2".exe Test.txt

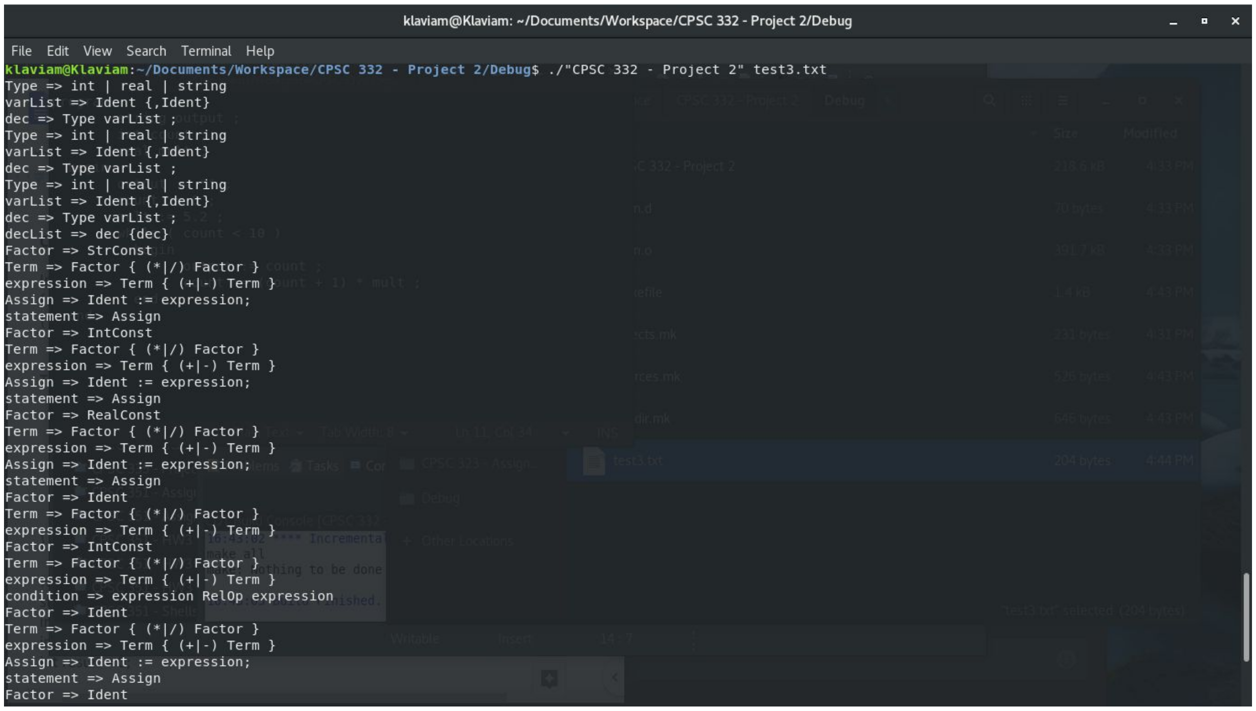
```
F:\Program Files (x86)\Workspace\CPSC 323 - Project 2\Debug>"CPSC 323 - Project 2".exe Test.txt
Type => int | real | string
varList => Ident {,Ident}
dec => Type varList ;
Type => int | real | string
varList => Ident {,Ident}
dec => Type varList ;
decList => dec {dec}
Factor => IntConst
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Assign => Ident := expression;
statement => Assign
Factor => IntConst
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Assign => Ident := expression;
statement => Assign
Factor => Ident
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Factor => IntConst
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
condition => expression RelOp expression
Factor => Ident
Term => Factor { (*|/) Factor }
Factor => IntConst
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Assign => Ident := expression;
statement => Assign
Factor => Ident
Factor => Ident
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Factor => ( expression )
Factor => IntConst
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Assign => Ident := expression;
statement => Assign
statementList => statement{statement}
While => while ( condition ) begin [statmentList] end
statement => While
Factor => Ident
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Factor => IntConst
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
condition => expression RelOp expression
Factor => IntConst
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Factor => IntConst
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Factor => RealConst
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Factor => IntConst
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Write => write ( expression {, expression} ) ;
statement => Write
statementList => statement{statement}
If => if ( condition ) begin statementList end { elsif ( condition ) begin statementList end } [else begin statementList end ]
statement => If
statementList => statement{statement}
Program => program [decList] [functionList] begin [statementList] end.
Press any key to continue . . .
```

Case - 2

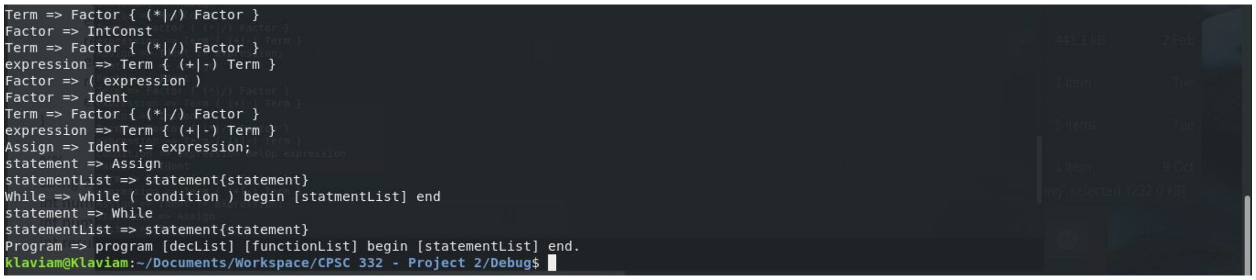
Input:

```
program
        string output ;
        int count ;
        real mult ;
  begin
        output := "" ;
        count := 0 ;
        mult := 5.2 ;
        while ( count < 10 )
          begin
                output := count ;
                count := (count + 1) * mult ;
          end
  end.
```

Output:


klaviam@Klaviam: ~/Documents/Workspace/CPSC 332 - Project 2/Debug
File Edit View Search Terminal Help

```
klaviam@Klaviam:~/Documents/Workspace/CPSC 332 - Project 2/Debug$ ./"CPSC 332 - Project 2" test3.txt
Type => int | real | string
varList => Ident {,Ident}
dec => Type varList ;
Type => int | real | string
varList => Ident {,Ident}
dec => Type varList ;
Type => int | real | string
varList => Ident {,Ident}
dec => Type varList ;
decList => dec {dec}
Factor => StrConst
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Assign => Ident := expression;
statement => Assign
Factor => IntConst
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Assign => Ident := expression;
statement => Assign
Factor => RealConst
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Assign => Ident := expression;
statement => Assign
Factor => Ident
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Factor => IntConst
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
condition => expression RelOp expression
Factor => Ident
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Assign => Ident := expression;
statement => Assign
Factor => Ident
```



```
Term => Factor { (*|/) Factor }
Factor => IntConst
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Factor => ( expression )
Factor => Ident
Term => Factor { (*|/) Factor }
expression => Term { (+|-) Term }
Assign => Ident := expression;
statement => Assign
statementList => statement{statement}
While => while ( condition ) begin [statmentList] end
statement => While
statementList => statement{statement}
Program => program [decList] [functionList] begin [statementList] end.
klaviam@Klaviam:~/Documents/Workspace/CPSC 332 - Project 2/Debug$
```

Case - 3

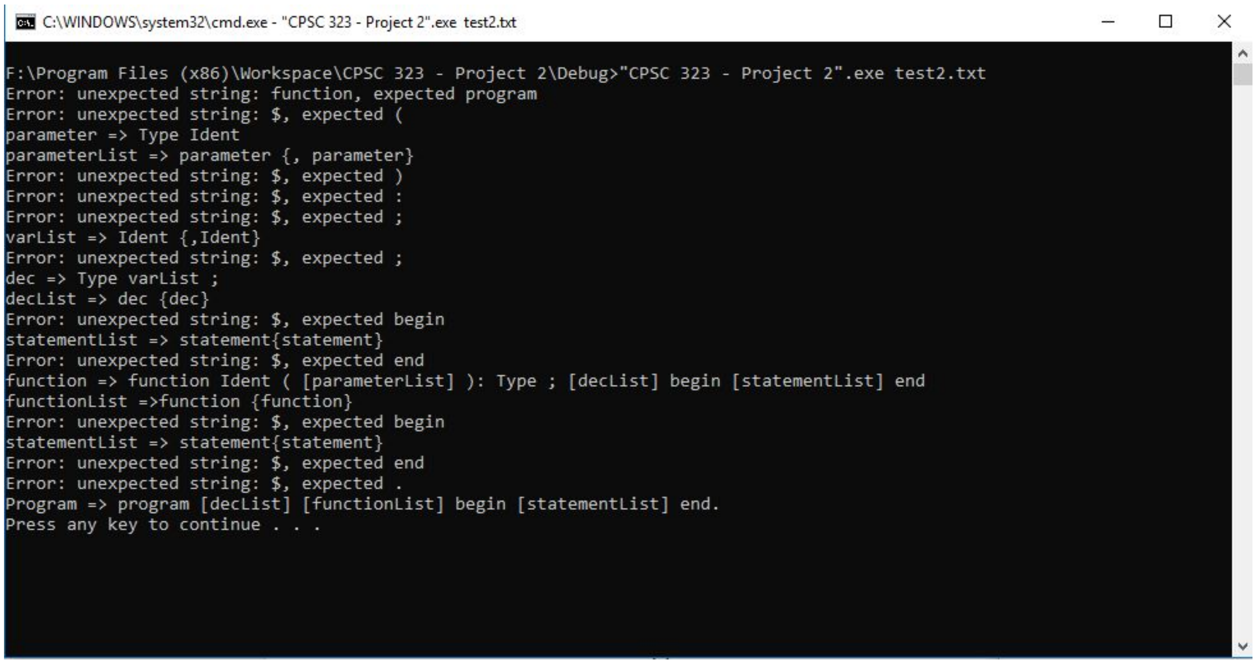
<u>Input:</u> **<u>(testing error cases)</u>**

```
function calculator$ [num1:int, num2:int, op:string]
{
        if (op == "+") {
        return num1 + num2; }
THIS IS AN ERROR
        else (op == "-") {
        return num1 - num2; }
        else (op == "*") {
        return num1 * num2; }
        else (op == "/") {
        return num1 / num2; }
        endif
}

%%
string op = "-"; ! Declarations !
int num1 = 5;
int num2 = 10;
int result = 0;

result = put calculator$ (num1, num2, op));
put (result);
```

Output:


C:\WINDOWS\system32\cmd.exe - "CPSC 323 - Project 2".exe test2.txt

```
F:\Program Files (x86)\Workspace\CPSC 323 - Project 2\Debug>"CPSC 323 - Project 2".exe test2.txt
Error: unexpected string: function, expected program
Error: unexpected string: $, expected (
parameter => Type Ident
parameterList => parameter {, parameter}
Error: unexpected string: $, expected )
Error: unexpected string: $, expected :
Error: unexpected string: $, expected ;
varList => Ident {,Ident}
Error: unexpected string: $, expected ;
dec => Type varList ;
decList => dec {dec}
Error: unexpected string: $, expected begin
statementList => statement{statement}
Error: unexpected string: $, expected end
function => function Ident ( [parameterList] ): Type ; [decList] begin [statementList] end
functionList =>function {function}
Error: unexpected string: $, expected begin
statementList => statement{statement}
Error: unexpected string: $, expected end
Error: unexpected string: $, expected .
Program => program [decList] [functionList] begin [statementList] end.
Press any key to continue . . .
```