Cover Page
CS323 Programming Assignments

1. Names [1. Omar Al Nabulsi]

    [2. Duc Huynh]

    [3. Victoria Naranjo]

2. Assignment Number [1]

3. Due Dates **Softcopy** [11/11], **Hardcopy** [11/13]

4. Turn-In Dates **Softcopy** [11/11], **Hardcopy** [11/13]

5. Executable FileName [Parser.exe]

(**A file that can be executed without compilation by the instructor**

6. LabRoom [CS-101]

(Execute your program in a lab in the CS building before submission)

7. Operating System [Windows 10 (Visual Studio)]

To be filled out by the Instructor:

GRADE:

COMMENTS:

CS323 Documentation

1. Problem Statement:

   To write a syntax analyzer. The syntax analyzer will be created using 29 syntax function rules given from project one. Each will return a token displaying whether it is an identifier, operator, separator or keyword. Each function will also return the corresponding lexeme. The program will remove any left recursions that occur in the text file.

2. How to use the program:
   - Download and extract the .exe file and double click it

     OR
   - Download zip file and extract source code in an editor
   - Run “parser.cpp” file in terminal
     - Skip below steps if running on visual studio
     - If running on linux comment out “system(“pause”);” in source file
     - G++ parser.cpp -o test
     - ./test
     - If you would like to run your own test case you will have to go into the source code and replace the file name on line 50 with your test case file name and then run it with the above steps.

3. Design of the program:
   - Initialized identifier, character and separator tokens
   - Created lexer class and corresponding functions to get every token and determine if it is an identifier, separator, operator or keyword.
   - Created top down parser with 29 function prototypes, change data types to return data statements and print out all production rules
   - Error handling to check for syntax errors and generate error messages
   - Run main until it reaches the last character in the file

4. Limitations:
   - None

5. Shortcomings:
   - Program does not detect right curly brace in parser body and compound functions

Test case 1:

```
! this is comment for this sample code which converts Fahrenheit into Celsius !
function convert$ [fahr:int] {
        return 5 * (fahr -32) / 9;
}
%%
int fahr = 84;
put(convert$(fahr));
```

Output:

```
Token: Identifier       Lexeme:
        <Opt Function Definitions> ::= <Function Definitions> | <Empty>
        <Function Definitions>  ::= <Function> | <Function><Function Definitions>
Token: Keyword  Lexeme: function
        <Function> ::= function <Identifier> ( <Opt Parameter List> ) <Opt Declaration List>
<Body>
Token: Identifier       Lexeme: convert$
Token: Separator        Lexeme: [
        <Opt Parameter List> ::= <Parameter List> | <Empty>
        <Parameter List>  ::=  <Parameter> | <Parameter>,<Parameter List>
        <Parameter> ::= <IDs> <Qualifier>
Token: Identifier       Lexeme: fahr
Token: Separator        Lexeme: :
Token: Keyword  Lexeme: int
        <Qualifier> ::= int
        <Opt Declaration List> ::= <Declaration List> | <Empty>
        <Declaration List>  := <Declaration> ; | <Declaration>;<Declaration List>
        <Declaration> ::= <Qualifier> <IDs>
        <IDs> ::= <Identifier> | <Idefntifier>,<IDs>
        <Body> ::= { <Statement List> }
Token: Keyword  Lexeme: return
        <Statement List> ::= <Statement> | <Statement><Statement List>
        <Statement> ::= <Return>
        <Return> ::= return <Expression>;
Token: Operator Lexeme: *
        <Expression> ::= <Expression> + <Term> | <Expression> - <Term> | <Term>
        <Term> ::= <Term> * <Factor> | <Term> / <Factor> | <Factor>
        <Factor> ::=  - <Primary> | <Primary>
        <Factor> ::= <Primary>
        <Primary> ::= <Identifier> | <Integer> | <Identifier> ( <IDs> ) | ( <Expression> ) | <Real>
| true | false
ERROR*---* No Right Curly Brace found
Press any key to continue . . .
```

Test case 2:

```
int num1, num2, large
if(num1 >= num2) {
        large = num1;
}
Else {
        11.00
        large = num2;
}
```

Output:

```
Token: Identifier       Lexeme:
        <Opt Function Definitions> ::= <Function Definitions> | <Empty>
        <Function Definitions>  ::= <Function> | <Function><Function Definitions>
Press any key to continue . . .
```

Test case 3:

```
%%
int i,x,y,total;
x = 3;
y = 9;
i = 1;
total = x + y - i;
```

Output:

```
Enter file name testcase.txt
Token: Identifier       Lexeme:
        <Opt Function Definitions> ::= <Function Definitions> | <Empty>
        <Function Definitions>  ::= <Function> | <Function><Function Definitions>
Token: Separator        Lexeme: %%
        <Opt Declaration List> ::= <Declaration List> | <Empty>
        <Declaration List>  := <Declaration> ; | <Declaration>;<Declaration List>
      <Declaration> ::= <Qualifier> <IDs>
Token: Keyword  Lexeme: int
        <Qualifier> ::= int
        <IDs> ::= <Identifier> | <Idefntifier>,<IDs>
        <Statement List> ::= <Statement> | <Statement><Statement List>
Press any key to continue . . .
```