CS323 Documentation

1. Problem Statement

Using our previous assignment, the lexicographical analyzer, we are now implementing a syntax analyzer. To accomplish this task we are using a top down approach with a recursive descent parser method. While parsing through the file's contents we will be using the lexicographical analyzer to tokenize the source code to then produce the parse tree after being stored.

2. How to use your program

Preconditions:

1. Python 3 is installed in the system and is the version of python being used
2. In order to run the program, it is necessary to keep the following files within the same directory.
    - lexer.py - Python source code for lexical analysis
    - parser.py - Python source code for syntax analysis
3. Must have read permissions on input files, and permission to write to output file.
4. Filenames may not have commas or trailing whitespace

Usage:

i. Synopsis:

Windows: **`python parser.py <FILE> [<FILE> …] [-o <OUTFILE>]`**

Linux: **`python3 parser.py <FILE> [<FILE> …] [-o <OUTFILE>]`**

or **`./parser.py <FILE> [<FILE> …] [-o <OUTFILE>]`**

ii. Arguments:

`<FILE>` - Filename or path to input file of source code to be parsed. If one is not supplied, the user will be prompted for a file. A list of files may be inputted.

-o **`<OUTFILE>`** - Name or path to the output file. Defaults to standard output.

iii. Description:

To execute this program, run the "*parser.py*" script using the terminal. Make sure to run the script using **Python 3**.

From the command line, a file name / path may be inputted in the form "python3 parser.py <file>". A list of input files may be inputted separated by spaces in the form "python3 parser.py <file1> <file2> … <fileN>". If no file is inputted, the program will prompt the user, in which the user may input a single file, or a list of files separated by commas. Note that when entering file paths after being prompted, the file path may not have commas, neither leading nor trailing whitespace. Make sure that there are sufficient read permissions on the input file(s).

The output path may also be specified from the command line by using the -o flag followed by the output file path. The output file will be overwritten with the tokens, corresponding production rules, and the parse tree for every input file entered. If no output file is inputted, then the output defaults to the standard output or console.

3. Design of your program

The program is written in Python 3. The lexical analyzer used is the same as in Project 1, that is, regular expressions are used to parse through and tokenize the source code.

Syntax Analysis (Grammar)

The production rules or grammar used by the syntax analyzer is as follows:

```
<StatementList> -> <Statement> <StatementList> | <empty>
```

Assignment

```
<Statement> -> <Assign>
<Assign> -> <ID> = <Expression>
```

Expressions

```
<Expression> -> <Term> | <Term> + <Expression> | <Term> - <Expression>
<Term> -> <Term> * <Factor> | <Term> / <Factor> | <Factor>
<Factor> -> '(' <Expression> ')' | <ID> | 'True' | 'False' |
      ('+' | '-')?(<FLOAT> | ('.')?<INT>) | //Accept all float forms
```

Declarations

```
<Statement> -> <Declarative>
<Declarative> -> <Type> <ID> <MoreIds>; | <empty>
<MoreIds> -> , <ID> <MoreIds> | <empty>
```

Flow Control Structures

```
<Statement> -> if <Conditional> then <StatementList> endif | if
   <Conditional> then <StatementList> else <StatementList> endif
<Statement> -> while <Conditional> do <StatementList> whileend
<Statement> -> begin <StatementList> end
<Conditional> -> <Expression> <Relop> <Expression> | <Expression>
```

The syntax analyzer implemented in this project uses a top down approach, specifically, using the Recursive Descent Parser (RDP) method. That is, for all non-terminals in the production rules, a function has been created when appropriate. Each function in turn calls other functions of non-terminals, based on the current token and the production rules recognized.

The syntax analyzer treats the source code as a StatementList. For each token, the token, location, and lexeme is printed to output followed by the matched production rules. Note that tokens already matched to a previous rule might not print rules afterwards (I.E. = in assignment)

Parse Tree

At the end of each input file's syntax analysis, the resultant parse tree is printed in the format:

```
["<Non Terminal Symbol 1>"    height: X, Children: {
 ["<Non Terminal Symbol of Child>"        height: X+1, Children: {
  ['<TOKEN>', '<LEXEME>", (<LINENUM>, <COLNUM>)]
 } End of (<Child>, X+1)]
} End of (<Symbol 1>, X)]
```

Each new layer of children gains a space of padding prepended. Non terminals are printed with the non terminal's symbol, the tree node's height, followed by a listing of its children. The children may be more non terminal nodes, or terminal nodes, in which case, the token is printed, with the token type, lexeme, and location.

The non terminal symbols are as follows:

```
SL = StatementList
S  = Statement
   D  = Declarative
      MI = MoreIDs
   A  = Assign
      E  = Expression
         T  = Term
         F  = Factor
   C  = Conditional
```

Throwing Errors

For any errors encountered during syntax analysis, a meaningful error message is printed, both in the output file and in the console. The error message contains the filename, the line number and column number of the token being processed at the time of the error, followed by a message describing the error, such as what occurred that was unexpected and what the analyzer was expecting.

After throwing an error message, the syntax analyzer terminates fully and stops processing the file further.

4. Any Limitation

Error Detection: The program can only detect a single error and will terminate immediately after throwing a meaningful error message.

Inputting File Path when Prompted: If prompted for file paths after giving none as a command line argument, then file paths entered may not have commas, neither leading nor trailing whitespace.

Comments during Syntax Analysis: Comments are completely passed over by the lexical analyzer, as it should. As such, the syntax analyzer never sees the comments, and thus, will not print anything to denote that comments have been encountered.

5. Any shortcomings for each iterations

None for now.