TANGENT

GROUP 4

Aman Panwar - Project Manager
Pranav K Nayak - Language Guru
Taha Adeel Mohammed - System Architect
Vikhyath - System Integrator
Shreya Kumar - Tester
Shambhu Kavir - Tester

Introduction

- Tangent is a statically typed, object oriented programming language intended for creating vector graphics.
- It takes design cues from python's Matplotlib and C++ functionality.
- Using built-in classes on an SVG, Tangent may be used to plot a variety of complicated figures, such as bezier curves and polygons.
- Our language offers the capability to carry out standard programming activities as well as the simplicity of charting and viewing two-dimensional data.

Lexical Analysis

The initial stage of the compiler, commonly referred to as a scanner, is lexical analysis. The High Level Input Program is transformed into a sequence of Tokens. A sequence of characters that may be regarded as a single unit in a programming language's grammar is called a lexical token.

It removes all the comments and white spaces while tokenizing the source code. If a particular sequence of characters do not match any pattern, then we are printing it to another output file.

LEXER

Lexer is a computer program which performs lexical analysis. We have implemented our lexer using flex as it is a standard tool.

Once the parser has also been implemented in the future, the tokens generated by our lexical analyzer will be utilized by the parser to generate an Abstract Syntax Tree (AST).

Example Codes

The following is an example of correct code.

```
1  # Hello world program
2  # lexer-test to check for comments
3
4  driver<: :>
5  (:
6      print("Hello World!\n");
7
8      # var int a := 5;
9      var int b := 2; # this is a valid comment
10
11     print(to_string(b) + " # This is NOT a comment");
12
13     # multi
14     # line
15     # comments
16
17     # comment # still a comment
18 :)
```

Example Codes

The following is an example of incorrect code.

```
# This sample code consists of several unmatched sequences of characters
# The lexical analyzer will throw an error.

driver <: :>
(:
    #incorrect identifier names
    var int 3abc;
    var string abdh$;
    var bool ___;

    # appearance of illegal characters
    print("Hello World"); ^^
    var int a := 5; ``
    $$$ ~

    # open strings
    print("hello);

:)
```

Instructions to compile

Files included

- lexer.l: the lexical analyzer which has been implemented.
- Makefile: to make the process of building the executable easy and also clean up generated files.
- Lexer-Tests: some test files (correct and incorrect) to test the lexical analyzer.

Step 1

Change your directory into lexer folder.

Step 2

Run make all in the terminal. This will generate a target object with the name lexer.

Files Used and Generated

- We use the lexer.l file to generate the lexer.yy.c file.
- We compile the lex.yy.c file to generate our lexical analyzer - “lexer”.
- We use this executable to parse our .tngt files.
- The successfully matched regular expressions are returned as a token stream, to be used by the parser in the future.
- The regular expressions that fail to match any pattern, i.e. any errors we get while parsing are output to “output.txt”

Thank You