

Tarea 3.2

Enrique Abraham García Torres
A00818997

06 Sep 2021

1 LittleDuck2020+1



Figure 1: My Little Duck

1.1 ANTLR

1.1.1 Code

```
package main

import (
    "fmt"
    "testing"

    "github.com/antlr/antlr4/runtime/Go/antlr"
    "github.com/kachus22/LittleDuck/ANTLR/parser"
)

type TreeShapeListener struct {
    *parser.BaseLittleDuckListener
}

func NewTreeShapeListener() *TreeShapeListener {
    return new(TreeShapeListener)
}

type SyntaxError struct {
    line, column int
    msg          string
}

func (s *SyntaxError) Error() string {
    return fmt.Sprintf("line %d:%d %s", s.line, s.column, s.msg)
}

type ErrorListener struct {
    *antlr.DefaultErrorListener
    Errors []error
}

func NewErrorListener() *ErrorListener {
    return new(ErrorListener)
}

func (s *ErrorListener) SyntaxError(
    recognizer antlr.Recognizer, offendingSymbol interface{},
    line, column int, msg string, e antlr.RecognitionException) {
    s.Errors = append(s.Errors, &SyntaxError{line, column, msg})
}
```

```

type TI struct {
    src    string
    valid bool
}

var testData = []*TI{
    {"../examples/good_input.txt", true},
    {"../examples/bad_input.txt", true},
}

func TestInputs(t *testing.T) {
    for _, ts := range testData {
        fmt.Printf("RUNNING\t%s\n", ts.src)
        // Setup the input
        input, _ := antlr.NewFileStream(ts.src)

        // Create the Lexer
        lexer := parser.NewLittleDuckLexer(input)

        // Load the Tokens
        stream := antlr.NewCommonTokenStream(
            lexer, antlr.TokenDefaultChannel)

        // Create the Parser
        parser := parser.NewLittleDuckParser(stream)

        errorListener := NewErrorListener()
        parser.RemoveErrorListeners()
        parser.AddErrorListener(errorListener)
        parser.BuildParseTrees = true
        tree := parser.Programa()
        antlr.ParseTreeWalkerDefault.Walk(
            NewTreeShapeListener(), tree)
        if ts.valid && len(errorListener.Errors) > 0 {
            for _, e := range errorListener.Errors {
                fmt.Printf("%s", ts.src)
                t.Error(e)
            }
        } else if ts.valid && len(errorListener.Errors) == 0 {
            fmt.Printf("SUCCESS\t%s\n\n", ts.src)
        }
    }
}

```

1.1.2 Test

```
$ go test -v .
=== RUN    TestInputs
RUNNING ../examples/good_input.txt
SUCCESS ../examples/good_input.txt

RUNNING ../examples/bad_input.txt
../examples/bad_input.txt
example_test.go:77: line 4:9 missing LET_ID at ':'
../examples/bad_input.txt
example_test.go:77: line 7:4 no viable alternative at input '9'
--- FAIL: TestInputs (0.00s)
FAIL
FAIL
```

1.2 Gocc

1.2.1 Code

```
package main

import (
    "fmt"
    "testing"

    "github.com/kachus22/LittleDuck/Gocc/lexer"
    "github.com/kachus22/LittleDuck/Gocc/parser"
)

type TI struct {
    src    string
    valid bool
}

var testData = []*TI{
    {"../examples/good_input.txt", true},
    {"../examples/bad_input.txt", true},
}

func TestInputs(t *testing.T) {
    p := parser.NewParser()
    for _, ts := range testData {
        fmt.Printf("RUNNING\t%s\n", ts.src)
        l, _ := lexer.NewLexerFile(ts.src)
        _, err := p.Parse(l)
        if ts.valid && err != nil {
            t.Error(err)
        } else {
            fmt.Printf("SUCCESS\t%s\n\n", ts.src)
        }
    }
}
```

1.2.2 Test

```
$ go test -v .
=== RUN   TestInputs
RUNNING ../examples/good_input.txt
SUCCESS ../examples/good_input.txt

RUNNING ../examples/bad_input.txt
example_test.go:28:
../examples/bad_input.txt:4:10: error: expected id; got: ":"
--- FAIL: TestInputs (0.00s)
FAIL
FAIL    github.com/kachus22/LittleDuck/Gocc    0.173s
FAIL
```