🦈 Basic

Miss Ylva Llywelyn

2023/10/14

Contents

Chapter 1: Grammer

This is a notation for writing down the grammer of the language. It uses regex syntax, with the components themselves being italicised.

block	{ *statement** }
statement	*expression* ;
expression	
const-expresion	”[a-zA-Z0-9]*”
	[+-]?[0-9]+(\.[0-9]+)?
	true|false

Chapter 2: Code Listing

```
###############
### IMPORTS ###
###############

from __future__ import annotations
import sys, signal
from enum import Enum

#################
### CONSTANTS ###
#################

DIGITS = '1234567890'

##############
### ERRORS ###
##############

class ShorkError(Exception):
    def __init__(self, startPosition:Position, endPosition:Position, errorName:str, details
        self.startPosition = startPosition
        self.endPosition = endPosition
        self.errorName = errorName
        self.details = details

    def __repr__(self) -> str:
        return f"""{self.errorName}: {self.details}
File: {self.startPosition.filename}, Line {self.startPosition.line}"""

class IllegalCharacterError(ShorkError):
    def __init__(self, startPosition:Position, endPosition:Position, details: str) -> None
        super().__init__(startPosition, endPosition, "Illegal␣Character", details)

################
### POSITION ###
################

class Position:
    def __init__(self, index:int, line:int, column:int, filename:str, filetext:str) -> None
        self.index = index
        self.line = line
        self.column = column
        self.filename = filename
        self.filetext = filetext

    def Advance(self, currentChar) -> Position:
        self.index += 1
        self.column += 1

        if currentChar == '\n':
            self.line += 1
            self.column = 0

        return self

    def Copy(self) -> Position:
        return Position(self.index, self.line, self.column, self.filename, self.filetext)
```

```python

##############
### TOKENS ###
##############

class TokenType(Enum):
    INT         = 1
    FLOAT       = 2

    PLUS        = 4
    MINUS       = 8
    MULTIPLY    = 16
    DIVIDE      = 32

    LPAREN      = 64
    RPAREN      = 128

    EOF         = 256

class Token:
    def __init__(self, tokenType:TokenType, value:any = None) -> None:
        self.tokenType = tokenType
        self.value = value

    def __repr__(self) -> str:
        if self.value: return f'{self.tokenType}:{self.value}'
        else: return f'{self.tokenType}'

#############
### LEXER ###
#############

class Lexer:
    def __init__(self, text: str, filename:str) -> None:
        self.text:str = text
        self.position:Position = Position(-1, 0, -1, filename, text)
        self.currentChar:str = None
        self.Advance()

    def Advance(self) -> None:
        self.position.Advance(self.currentChar)
        self.currentChar = self.text[self.position.index] if self.position.index < len(self

    def MakeTokens(self) -> list[Token]:
        tokens = []

        while self.currentChar != None:
            if self.currentChar in '␣\t':
                self.Advance()

            elif self.currentChar in DIGITS:
                tokens.append(self.MakeNumber())

            elif self.currentChar == '+':
                tokens.append(Token(TokenType.PLUS))
                self.Advance()
            elif self.currentChar == '-':
                tokens.append(Token(TokenType.MINUS))
                self.Advance()
            elif self.currentChar == '*':
                tokens.append(Token(TokenType.MULTIPLY))
                self.Advance()
            elif self.currentChar == '/':
                tokens.append(Token(TokenType.DIVIDE))
                self.Advance()
            elif self.currentChar == '(':
                tokens.append(Token(TokenType.LPAREN))
                self.Advance()
            elif self.currentChar == ')':
                tokens.append(Token(TokenType.RPAREN))
                self.Advance()

            else:
                char = self.currentChar
                startPosition = self.position.Copy()
                self.Advance()
                raise IllegalCharacterError(startPosition, self.position, f"'{char}'")

        tokens.append(Token(TokenType.EOF))
        return tokens

    def MakeNumber(self) -> Token:
        numString = ''
        dotCount = 0

        while self.currentChar != None and self.currentChar in DIGITS+'.':
            if self.currentChar == '.':
                if dotCount == 1: break
                dotCount += 1
                numString += '.'
            else:
                numString += self.currentChar
            self.Advance()

        if dotCount == 0:
            return Token(TokenType.INT, int(numString))
        else:
            return Token(TokenType.FLOAT, float(numString))

###########
### RUN ###
###########

def Run(text:str, filename:str) -> None:
    try:
        lexer: Lexer = Lexer(text, filename)
        tokens: list[Token] = lexer.MakeTokens()

        print(tokens)
    except ShorkError as e:
        print(e.__repr__())

def __SignalHandler(sig, frame):
    sys.exit(0)

if __name__ == "__main__":
    signal.signal(signal.SIGINT, __SignalHandler)
    while True:
        text = input(">␣")
        Run(text, "<STDIN>")
```