

CS424-Compiler Construction

Assignment#2

Assignment Report

Muhammad Omer Bin Atique (2020335)

List of Contents

[1 INTRODUCTION 2](#_Toc161137528)

[2 IMPLEMENTATION 2](#_Toc161137529)

[2.1 Parser’s Design 2](#_Toc161137530)

[2.2 Implementation Detail 3](#_Toc161137531)

[2.2.1 The Main Function: 3](#_Toc161137532)

[2.2.2 Explanation of Functions and Corresponding Grammer Rule: 4](#_Toc161137533)

[2.3 Test Cases 6](#_Toc161137534)

# INTRODUCTION

Imagine a small, new programming language called "MiniLang." MiniLang is designed to be simple yet powerful enough to demonstrate key programming concepts. It supports basic arithmetic operations, variable assignments, if-else conditions, and print statements. My task was to design and implement a parser for MiniLang using C++ or python.

This assignment helped me understand the complexities of syntax analysis and prepared me for the subsequent stages of compiler design, moreover, it focuses on constructing a parser that can recognize and validate the syntax of the MiniLang programming language.

# IMPLEMENTATION

## Parser’s Design

The parser is designed to take lexemes generated in the last assignment and perform syntax analysis of that. I had to make a new grammar to formulate the rules of the syntax. Based on that the incoming lexemes are character by character evaluated. Below is the grammar on which this parser is based:

Program-> Declaration Declaration Assignment Condition Update Prin

Declaration -> Datatype Identifier '=' Literal ';'

Datatype -> 'int' | 'Boolean' | 'String'

Literal -> Number | Boolean | String

Assignment -> Identifier '=' Arithmetic';'

Arithmetic -> Identifier '+' Identifier

| Identifier '-' Identifier

| Identifier '\*' Identifier | Identifier '/' Identifier

Condition -> 'if' '(' Arithmetic Relational Literal ')' '{' Print '}'

| 'if' '(' Arithmetic Relational Literal ')' '{' Print '}' 'else' '{' Print '}'

Relational -> '>=' | '>' | '<=' | '<' | '=='

Update -> Identifier '=' Identifier Relational Literal ';'

Print -> 'print' '(' Literal ')' ';'

Identifier -> [a-zA-Z\_] [a-zA-Z0-9\_] \*

Number -> [0-9] +

Boolean -> 'true' | 'false'

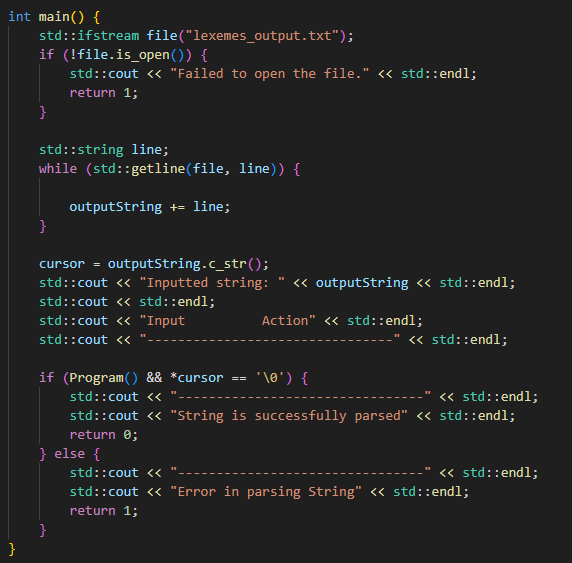
String -> " [a-zA-Z0-9\s] + "

## Implementation Detail

## 2.2.1 The Main Function:

The main function is the entry point of the program. It performs the following tasks:

* Opens a file named "lexemes\_output.txt" to read the input.
* Reads the contents of the file into a string.
* Sets the cursor to point to the string.
* Calls the Program function to parse the input string.
* Prints success or failure messages based on the parsing result.



## 2.2.2 Explanation of Functions and Corresponding Grammer Rule:

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

Here's a breakdown of the functions in the program and their correspondence to the grammar (All functions are of the same structure as above program() code):

* **Program():**
  + Implements the top-level rule of the grammar.
  + Corresponds to the structure of the program where it expects a series of declarations followed by an assignment, condition, update, and print statements.
* **Declaration():**
  + Implements the Declaration grammar rule.
  + Declares a variable with a specified data type, identifier, and literal value.
  + Corresponds to the declaration of variables in the program.
* **Assignment():**
  + Implements the Assignment grammar rule.
  + Assigns a value to a previously declared variable.
  + Corresponds to the assignment of values to variables.
* **Condition():**
  + Implements the Condition grammar rule.
  + Handles conditional statements with if-else constructs.
  + Corresponds to the conditional statements in the program.
* **Update():**
  + Implements the Update grammar rule.
  + Updates the value of a variable based on a relational expression.
  + Corresponds to statements where variables are updated based on conditions.
* **Print():**
  + Implements the Print grammar rule.
  + Prints the value of a literal.
  + Corresponds to print statements in the program.
* **DataType():**
  + Implements the Data grammar rule.
  + Defines the data types allowed for variable declaration.
  + Corresponds to the types of variables allowed in the program.
* **Identifier():**
  + Implements the Identifier grammar rule.
  + Defines the format of valid identifiers.
  + Corresponds to variable names used in the program.
* **Arithmetic():**
  + Implements the Arithmetic grammar rule.
  + Defines arithmetic operations between identifiers.
  + Corresponds to arithmetic operations in the program.
* **Relational():**
  + Implements the Relational grammar rule.
  + Defines relational operators used in conditional statements.
  + Corresponds to relational operators used for comparisons.
* **Boolean():**
  + Implements the Boolean grammar rule.
  + Represents true or false values.
  + Corresponds to boolean literals used in conditional statements.
* **Number():**
  + Implements the Number grammar rule.
  + Defines numeric values.
  + Corresponds to numeric literals used in the program.
* **String():**
  + Implements the String grammar rule.
  + Defines string literals enclosed within double quotes.
  + Corresponds to string literals used in the program.
* **Literal():**
  + Implements the Literal grammar rule.
  + Represents literal values such as numbers, booleans, or strings.
  + Corresponds to literals used throughout the program.

The program's structure follows the grammar rules closely, with each function responsible for handling specific syntactic constructs defined by the grammar. This design allows for clear separation of concerns and facilitates the parsing process.

## Test Cases

* **Simple Case:**

**A screenshot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**\*\* While the code is correct, the grammar rules are problematic. Could not form new and better grammar unfortunately. \*\***