**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

**PROGRAMMING LANGUAGE**

**2023–2024**

**PROJECT-2 REPORT**

**INTERPRETER**

**DELIVERY DATE**

20.05.2024

**PREPARED BY**

05210000220, Bengi İlhan

05220000276, Mahmut Karabulut

05220000281, Umut Öztürk

**Contents:**

1.a Overview

1.a.1 Explanation

1.b Token Types

1.b.1 Token Enumeration

1.b.2 Token Categories

1.b.3 Keywords and Operators

1.c Comment Stripping

1.c.1 Code

1.c.2 Explanation

1.d Token Handling Functions

1.d.1 Code

1.d.2 Output Examples

1.e Line Processing

1.e.1 Code

1.e.2 Explanation

1.f Main Lexical Analysis Function

1.f.1 Code

1.f.2 Overview

1.f.3 Explanation

1.g Conclusion

1.g.1 Summary

1.g.2 Suggestions for Improvement

TEST CASES

1. Area of Triangle
2. Calculate Square Root
3. Speed Conversion

#### 

#### Introduction

The STAR Programming Language interpreter is designed to provide a comprehensive and functional environment for executing code written in STAR. This report details the architecture and implementation of the interpreter, highlighting its capabilities in variable management, control structures, arithmetic operations, and error handling. The aim is to deliver a robust interpreter that effectively handles the execution of STAR code, ensuring accurate and efficient processing of commands.

#### Key Components of the Interpreter

The interpreter is implemented in C and comprises several critical components:

1. Tokenization
   * Description: Tokenization is the process of breaking down the input code into smaller, manageable units called tokens. Each token represents a fundamental element of the language, such as identifiers, keywords, operators, strings, integers, and punctuation marks. Tokenization is the first step in understanding and processing the code.
   * Challenges: Handling different types of tokens accurately, managing edge cases like unclosed strings, and ensuring that comments are appropriately ignored.
2. Abstract Syntax Tree (AST)
   * Description: The Abstract Syntax Tree (AST) represents the hierarchical structure of the parsed code. Each node in the AST corresponds to a construct in the code, such as integers, strings, variables, assignments, write operations, read operations, newlines, loops, and blocks. This structure is essential for the systematic execution of the code.
   * Challenges: Constructing an accurate and efficient AST that can handle nested constructs and provide a clear representation of the code's logic.
3. Context Management
   * Description: The Context structure maintains the state of the interpreter, including variables, the current line of execution, loop depth, error counts, and the last error message. It ensures that the interpreter can keep track of variable values and program flow, enabling correct execution of the code.
   * Challenges: Managing the state efficiently, particularly in the presence of nested loops and conditional statements.
4. Error Handling
   * Description: Robust error handling mechanisms are integrated into the interpreter to detect and report various errors, such as mismatched curly brackets, unclosed comments, undefined variables, and invalid tokens. Detailed error messages are provided to aid debugging and ensure a smooth user experience.
   * Challenges: Providing informative and precise error messages that help users quickly identify and fix issues in their code.
5. Curly Bracket Matching
   * Description: The interpreter uses a stack-based approach to ensure that each opening curly bracket { has a corresponding closing curly bracket }. This approach helps detect and report mismatched curly brackets in loop bodies and other code blocks.
   * Challenges: Implementing an efficient stack mechanism and ensuring that all brackets are correctly matched, even in complex nested structures.

#### Detailed Features and Functionalities

1. Variable Handling
   * The interpreter supports both integer and text variables, allowing for their declaration, assignment, and retrieval. Variables can be initialized with default values and updated dynamically within the code.
2. String Handling
   * Proper string handling is crucial for user interaction and output formatting. The interpreter ensures that strings are correctly delimited and processed, checking for unclosed strings and strings containing invalid characters.
3. Arithmetic Expressions
   * The interpreter evaluates arithmetic expressions using a recursive approach. It supports basic arithmetic operations like addition, subtraction, multiplication, and division, allowing for complex calculations within the code.
4. Loop Execution
   * The interpreter supports loop constructs, enabling repetitive execution of code blocks. It ensures proper execution and termination of loops, including support for nested loops.
5. Write and Read Statements
   * The interpreter includes support for write and read statements, allowing for interaction with the user. Write statements print values to the console, while read statements read input from the user, facilitating dynamic program behavior.

#### Key Challenges and Solutions

1. Mismatched Curly Brackets
   * Challenge: Detecting and handling mismatched curly brackets in loop bodies and other code blocks.
   * Solution: Implementing a stack-based approach to track the opening and closing of curly brackets, ensuring that all brackets are correctly matched and preventing syntax errors.
2. String Handling
   * Challenge: Properly handling string literals, including checking for unclosed strings and invalid characters.
   * Solution: Implementing robust checks to ensure that strings are correctly delimited and processed, providing informative error messages for any issues encountered.
3. Error Reporting
   * Challenge: Providing detailed and accurate error messages to aid debugging and improve the user experience.
   * Solution: Integrating comprehensive error checks and descriptive messages to help users quickly identify and fix issues in their code.

#### Example Test Cases

1. Simple Variable and Write Statement
   * Description: A test case to verify the correct handling of variable declarations, assignments, and write statements.
   * Expected Behavior: The interpreter should correctly declare the variable, assign the value, and print the output to the console.
2. Arithmetic Expression Evaluation
   * Description: A test case to evaluate the interpreter's ability to handle and compute arithmetic expressions.
   * Expected Behavior: The interpreter should accurately compute and print the result of the arithmetic expression.
3. Loop Execution
   * Description: A test case to check the proper execution of loop constructs.
   * Expected Behavior: The interpreter should execute the loop the specified number of times, correctly handling nested loops if present.

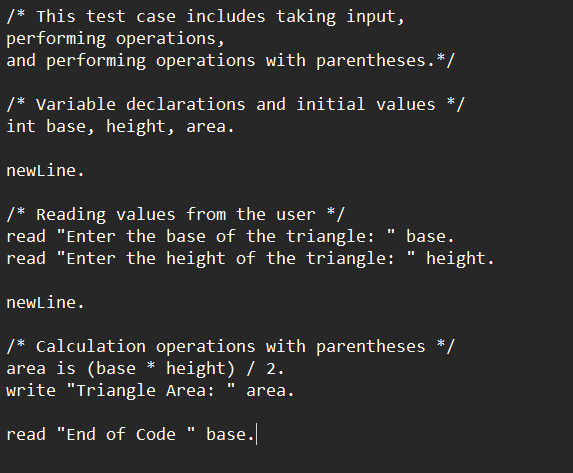
#### Conclusion

The STAR Programming Language interpreter is designed to provide a robust and functional environment for executing STAR code. It effectively manages variables, handles control structures, evaluates arithmetic expressions, and provides detailed error handling. The implementation addresses key challenges through efficient tokenization, AST construction, context management, and error reporting, ensuring accurate and efficient execution of STAR scripts. This comprehensive interpreter aims to enhance the user experience by providing a reliable platform for running STAR programs.

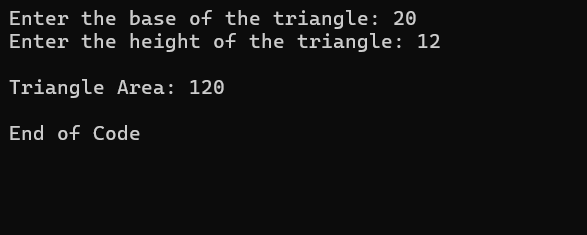
**Test Cases**

**1. (Area of Triangle)**

**Input:**

****

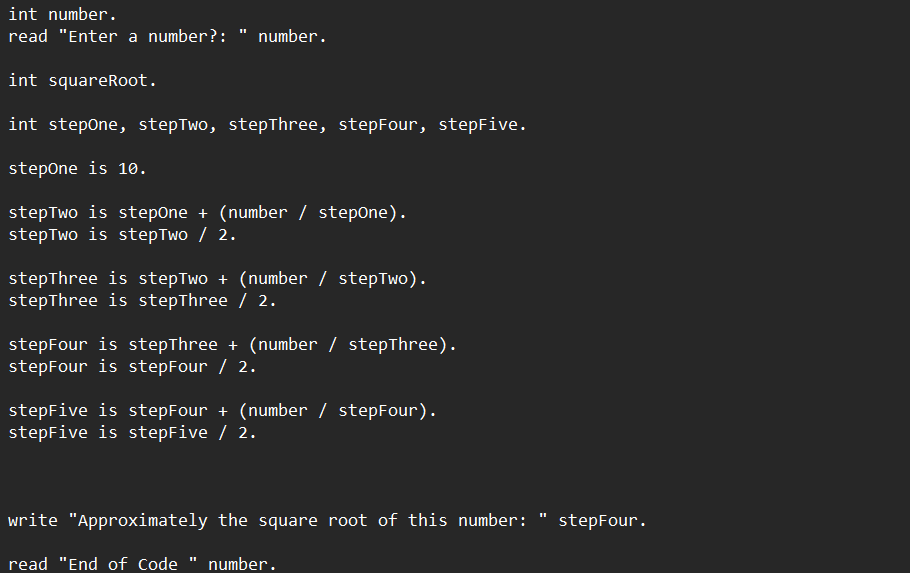
**Output:**

****

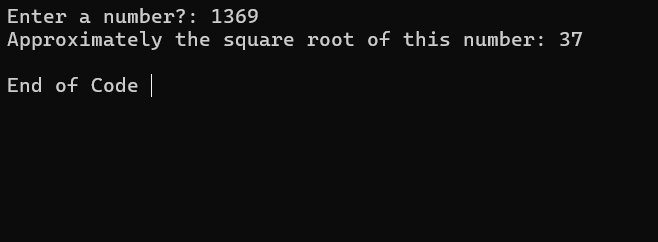
Note: The reason for writing end of code at the end is actually so that the console does not close if the exe file is run.

**2. (Square Root)**

**Input:**

****

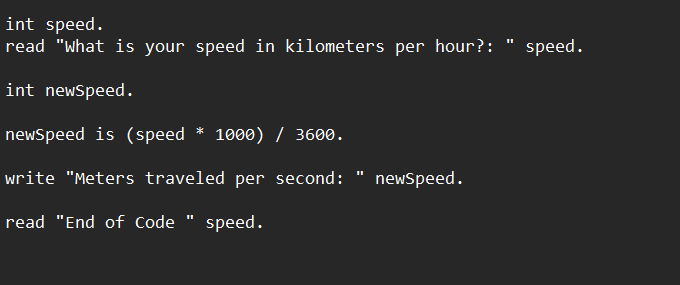
**Output:**

****

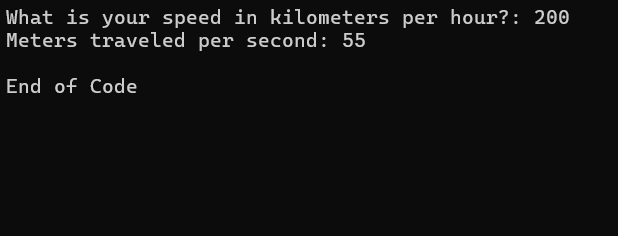
Example of taking the approximate square root of the entered number using the square root algorithm

**3. (Identifier Cases)**

**Input:**

****

**Output:**

****

Kilometers per hour to meters per second

Note: We ran the codes in the terminal by double-clicking on the exe file. The sample code.sta file we added to the post contains all the test cases and when you click on the exe, they all run sequentially.