MIST | Technology for Advancement
Military Institute of Science and Technology

Term Paper

Compiler Design

Subject code: CSE-303

Topic – Complex Number

Author:

Iftekharul Islam 202214024

Abstract

This term paper explores the critical aspects of compiler design, focusing on the recognition of tokens and lexical analysis with a specific emphasis on complex numbers. Lexical analysis, a fundamental phase in the compilation process, involves scanning the source code and converting it into a sequence of tokens. This paper delves into the theoretical and practical elements of lexical analysis, illustrating how complex numbers, which consist of real and imaginary parts, can be accurately identified and processed by a lexical analyzer.

Introduction

Compiler design is a foundational discipline in computer science, central to developing and executing programming languages. A key phase is lexical analysis, which transforms raw source code into structured tokens, facilitating syntax and semantic analysis. The lexical analyzer, or scanner, reads input code and partitions it into tokens based on the language's lexical grammar. This ensures the code adheres to syntactic and semantic rules, streamlining the parsing phase.

This term paper focuses on recognizing complex numbers—numerical data with real and imaginary components. For this at first we'll be defining the transition diagram, pattern for numbers, and then use the numbers pattern to further construct the pattern, transition diagram and finally implement the code for identifying the token

Patterns:

digit : [0-9]
number : digit*(.digit+)? (E[+-]?digit+)?
complex-number : number[+-](number)?i — [+-]? (number)? i

Transition Diagrams:

The Transition diagrams were made following the patterns using UML code

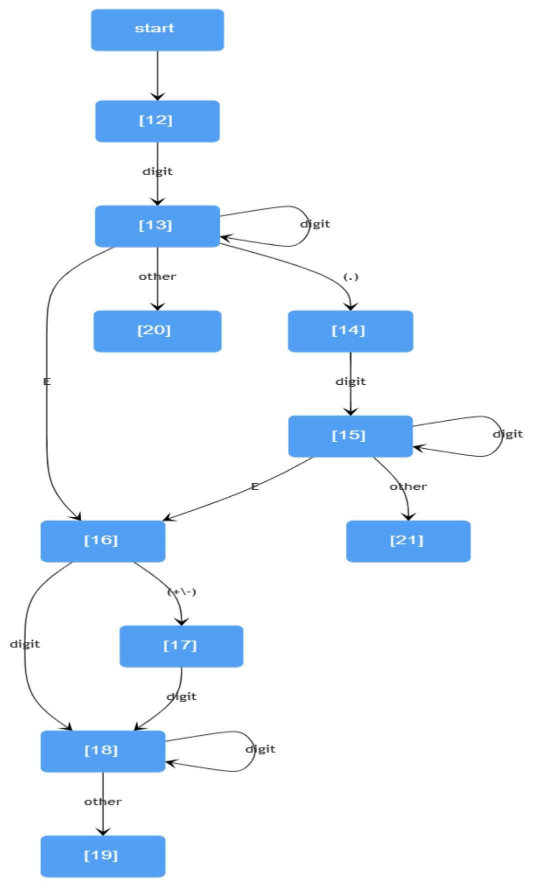

Figure 1: Transition Diagram for Number

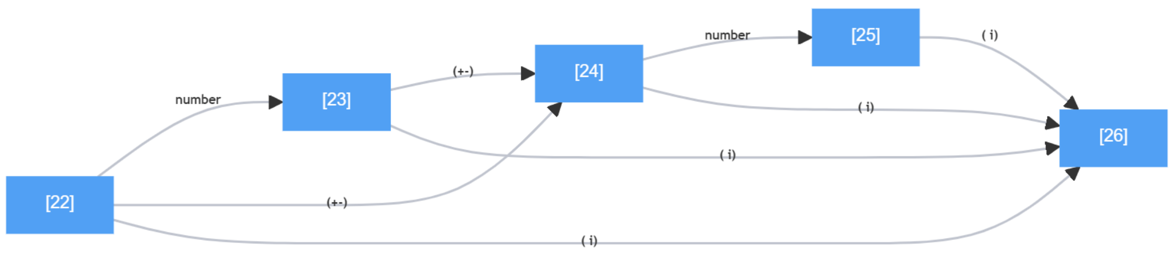

Figure 2: Transition Diagram for Complex Number

Code Explanation:

TYPE: which is an enum, values set to NUM, CN, INVALID for type identification and error detection purpose

TOKEN: a class for the tokens with attribute (string) and type (TYPE) variables

nextChar: gets next character from input string retract: sets the pointer to previous index fail: when the token is not a valid Complex number, the fail is called which ends the program

getNumber

We're using the getNumber function to: at first identify if the number user has inputted is a correct number format or not, if it is correct it returns the number TOKEN with it's value and type, else fail() is called, because if a number is not valid, then the complex number's real/imaginary part will not be valid, so it won't be a complex number.
Now, there's a catch from just straight-forward number identification, if from the **case 12** if we encounter other characters we used to run fail() for normal cases, but for identifying complex numbers, we won't do it, as the letter 'i' is there right after a number, so instead we'll retract and return the number, further execution will be handled by getCN() function. While checking every character we'll add them to tokenValue if acceptable, to get the full number as the Attribute

getCN:

This is the most important part of the code, as it is the primary function to identify if a number is complex or not. For different characters we'll basically follow the transition diagram and transition to next states,
and for numbers we made a temporary variable of class TOKEN (with name tmp) and used the getNumber() function to check if the numbers are valid or not (we compared the numbers by using the getter getType of TOKEN tmp), then if the numbers are valid, then we'll move on to the next states otherwise call the fail() function.
For the attribute, we'll use the string tokenValue and add the characters we found (if they're valid) and for numbers, we'll add the numbers from the tmp TOKEN's attribute (calling the getAttr getter). Now, the main function will take the input and we'll call the getCN() function to identify if the inputted value is a complex number or not. We're using the switch-case for enumerating over the TYPE enum (return type of TOKEN's 'type' variable)

Conclusion

In this term paper, we go through the process of lexical analysis and identification of tokens. Here we analyze the transition diagram and the code I made for the identification of Complex Numbers.