Assignment Report

Tokenizer for C Programs

Course: Compiler Construction

Date: August 21, 2025

Objective

The objective of this assignment is to implement a **tokenizer for C programs**. The tokenizer scans a given C source file, separates it into distinct tokens (keywords, datatypes, identifiers, constants, operators, and delimiters), and builds a simple symbol table recording identifiers with their type and line number.

Introduction

In the process of compilation, the first step is **lexical analysis**. It transforms the sequence of characters in the source program into a sequence of tokens.

- **Token**: The smallest meaningful unit (keywords, identifiers, literals, operators, delimiters).
- **Tokenizer vs Lexical Analyzer**: The tokenizer only separates and classifies tokens, while a lexical analyzer also uses formal definitions (regular expressions, DFAs) and handles lexical errors.
- **Symbol Table**: A data structure that stores information about identifiers such as name, type, and line of occurrence.

Algorithm / Approach

The step-by-step procedure for the tokenizer is:

1. Read the file character by character.
2. Accumulate characters into a buffer until a delimiter or operator is reached.
3. Classify the buffer as datatype, keyword, constant, or identifier.
4. Store tokens only if they are distinct (no duplicates).
5. For identifiers, add an entry into the symbol table with type and line number.
6. For operators, check if it forms a two-character operator (`==`, `<=`, etc.).
7. Store delimiters separately.
8. Print all tokens category-wise and display the symbol table.

Implementation

The program is implemented in C language. The following data structures are used:

- Arrays to store distinct tokens (identifiers, keywords, etc.)
- Structure `Symbol` for the symbol table

Key functions:

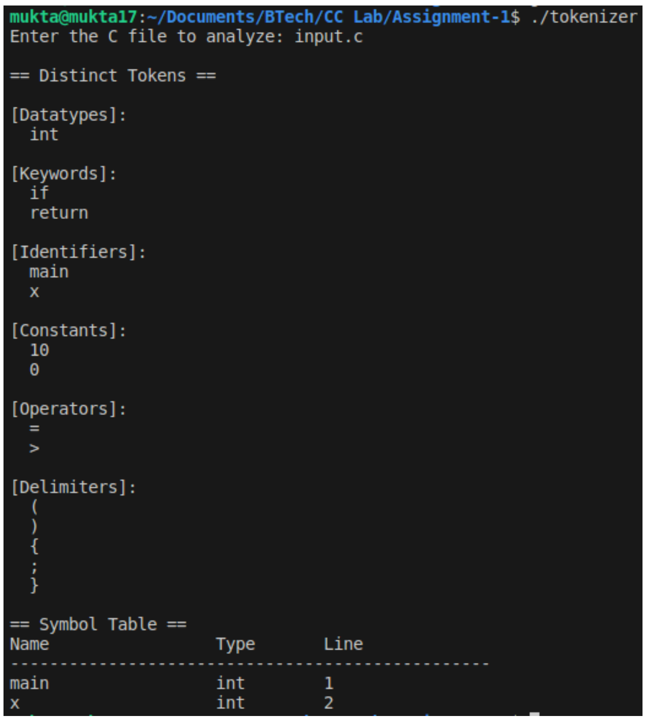
- `isKeyword`, `isDatatype`, `isDelimiter`, `isOperator` for classification
- `addToSymbolTable` for inserting identifiers
- `tokenizeDistinct` for main tokenization
- `printTokens`, `printSymbolTable` for displaying output

Sample Output

Input C File (input.c):

```c
int main() {
    int x = 10;
    if (x > 0) {
        return x;
    }
}
```

Program Output:



```
mukta@mukta17:~/Documents/BTech/CC Lab/Assignment-1$ ./tokenizer
Enter the C file to analyze: input.c

== Distinct Tokens ==

[Datatypes]:
  int

[Keywords]:
  if
  return

[Identifiers]:
  main
  x

[Constants]:
  10
  0

[Operators]:
  =
  >

[Delimiters]:
  (
  )
  {
  ;
  }

== Symbol Table ==
Name                 Type       Line
--------------------------------------------------
main                 int        1
x                    int        2
```

Figure 1: Screenshot of program output

Limitations

- Does not handle comments (`//`, `/* ... */`).
- String and character literals not recognized.
- No support for floating-point constants or hex/oct numbers.
- No scope handling in the symbol table.
- Limited set of operators.

Conclusion

The tokenizer successfully identifies and categorizes distinct tokens from C source code and builds a basic symbol table. While limited compared to a full lexical analyzer, it demonstrates the core principle of tokenization and forms the foundation for further stages of compilation.