# Assignment1 (Lexical Analyser)

Aayush Nirwan
1601001

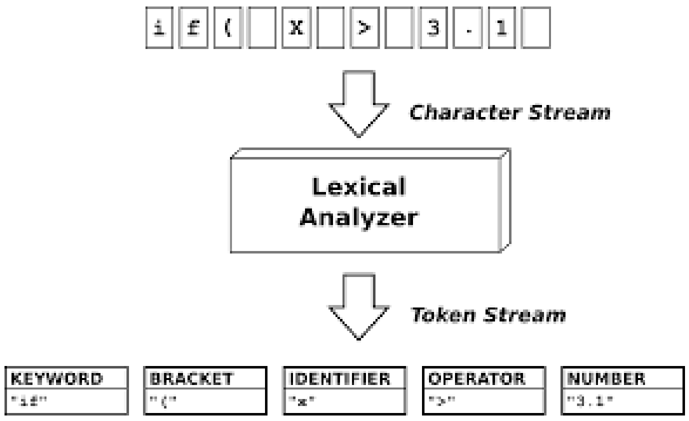
Manan Sharma
1601030

This Assignment will give the flavour, that how a Lexical Analyser works in traditional compilation process.

## Introduction

Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code.

If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works closely with the syntax analyzer. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyzer when it demands.



## Algorithm Description

### Implementation of DFA

```
Input : Sample File
While{till end of file}
      Scan Words
      REMOVE COMMENTS FROM FILE
   int i=0
IF{word[i]==i}
   i++
ELSEIF{Word[i]==n}
i++
ELSEIF{Word[i]==t}
i++
if(word[i]=='\0')
{
printf("KEYWORD INT DETECTED");
}
else{
i--;
printf(" VARIABLE DETECTED");
->Similarily check for other keywords
Output: Write to file
```

The program will ask the user to input a file name to be lexically analyzed . while reading the file it will remove the comments section in the file and then the program will tokenize all the input file based on whitespace . These tokens are then analyzed further to determine the type (keyword, identifier, integer, floating number , operator, literals). The program then prints out the token and the type of that token. DFA has been constructed for the same using nested (if , else) condition which identifies all the tokens present in the file .

In the the above pseudo code we will extract each word from a file and then pass that word into if else condition which checks whether it belongs to some particular type or not if yes than print corresponding type . After reading whole file word by word in the similar way we will insert all tokens in a queue and finally write it into file .

### Implementation of Symbol Table

```
  SAMPLE FILE
  CREATE LINKED LIST
   Insert function()
//Every time a variable is encountered ,insert
   function is called from the main file.
   Display function()
// To display the symbol table
```

Table implementation will keep record of all the variables/identifiers present in the file , corresponding to that variable which type it belongs too and then finally the size of that variable.

How the results has been stored in the table is depicted in Table 1.

Table 1
*Result table.*

| Variable | Type | Size |
|---|---|---|
| a | int | 2bytes |
| ch | char | 1 bytes |
| b | float | 4 bytes |
| e | long | 4 bytes |

## Code Analysis

Code Analysis can be done by measuring some of the points in terms of optimization ,robustness, correctness and asymptotic complexity (both space time) .

### Robustness

Robustness describes how reliable our program is, especially under extreme conditions such as extreme workload and bad or unpredictable user inputs. This program would never crash no matter what the user inputs. Even if the user enters invalid data, program will handle it properly. The worse thing it does is it terminate the program gracefully. In this code file handling is handled properly because if file is completely out of accordance with that format, if the file is improperly formatted, code outputs an error message and terminate the program or if user enters an empty file it will give an error message . While writing program all these things are taken seriously . The user can append his/her own lines to the source C file to check for various possibilities. The code supports most of the corner cases and will most probably generate correct results.

### Correctness

correctness of an algorithm is asserted when it is said that the algorithm is correct with respect to a specification. correctness refers to the input-output behaviour of the algorithm (i.e., for each input it produces the expected output). This program might renders some of test cases but it will work properly in most of test cases . for example while reading a file if this 24.34.34 as a token come . Then ouput will be like this:

$$(24.34.34) - - - - - - - - -is\ \ not\ \ floating\ \ number$$

So, this an example there are many different cases which can be checked .The code is scalable i.e. new dfa can be included very easily and produce correct results. However, if the user try some unknown keywords, then wrong output will be encountered.

### Asymptotic Complexity

**Time Complexity.** The time complexity of the lexical Analyser, can be calculated as:
Suppose if there are n lines of code in source C file and each line contains atmost 30 characters.

$$(WorstTimeComplexity) = O(n * 30)$$

$$(AverageTimeComplexity) = O(n)$$

**Space Complexity.** The Space Complexity of the lexical analyser, can be calculated as:
The whole code i.e ($n\times 20$) characters are stored in one string arrays.

$$(SpaceComplexity) = O(3 * n * 30)$$

### Listings

In this assignment total 3 files are created , one is the source file other one is test file and last one is header file.
Source file ->(MainFile.c) contains the code written in c.
Test file ->(Input.c) contains the Sample code written in c . This file is passed into Source code .
Header File-> (rmcomment.h) and (SymbolTable.h) This Header file is included in the source code , basicaly it contains function for creating symbol table refer to Table1 and for removing comment refer to Table1.
Output File-> (Output.txt) it contains the result.

### Prerequisites

Make sure that C compiler must be installed in your machine whether it is gcc if using Linux or other one.