**AIM:**

To design an online lexical analyzer simulator for C programming language that detects token and classifies them into keywords, identifiers, special characters and operators while ignoring tab, new line and redundant spaces.

**PROGRAMMING LANGUAGES USED:**

* C Programming Language
* HTML
* CSS
* JavaScript

**THEORY:**

Given line of code as user input, representing a program snippet, the task is to detect tokens in a C program called the lexical analysis of a compiler. Hence, this simulator is referred to as the lexical analyzer. The lexical analyzer is a part of the compiler that detects tokens of a program and sends it to the syntax analyzer.

Token is the smallest logical unit of a program and can be of the following types:-

* Keywords/ Function Names
* Identifiers
* String Constants
* Special Characters
* Operators

**SIMULATOR LOGIC:**

* Traverse the input program snippet character by character.
* Tokenization i.e., dividing the program into valid tokens.
* Remove tab and white space characters.
* Remove comments.
* Remove the rest parts of the program that are meant for the understanding of the user and are in no way needed during compilation.
* Place the tokens under each of the below sub-categories: -
  + Keywords for the function names.
  + Identifiers for the variables.
  + String constants for constants with fixed sequence of characters.
  + 192 Special characters among 256 defined characters by ASCII.
  + Operators for the mathematical operations.

**INPUT/OUTPUT EXAMPLE:**

***Input:***

*The program is: float x = a + 1b*

***Output:***

*All tokens are:-*

*Valid Keyword: float*

*Valid Identifier: x*

*Valid Operator: =*

*Valid Identifier: a*

*Valid Operator: +*

*Invalid Identifier: 1b*

**COMPLEXITY OF SIMULATOR PROGRAM:**

* ***Time Complexity:***
* ***Auxiliary Space:***

**DISCUSSION:**

We wrote the online simulator using C programming language and integrated it with the website designed using HTML, CSS, JavaScript. The lexical analyzer simulator is designed for C language that detects token and classifies them into keywords, identifiers, string constants, special characters and operators while ignoring tab, new line and redundant spaces.

**SOURCES OF ERROR:**

There can be two possible sources of error for a string in the program snippet to be considered as an invalid identifier:

* **Having an Illegal Character in String:**

It can be of two types-

* If the start of the string is neither a letter (in upper or lower case) nor an underscore symbol (\_).
* If there is(are) illegal special characters like $, #, @ symbols present within the string.
* **String exceeding 31 Characters:**

Sometimes the user input string can be a long one, particularly exceeding 31characters in length. In those cases, the string, although having all required syntax, will be considered as an invalid identifier.  
Example: RohanD\_DebangshuK\_SohamM\_AbhirupSG

**ERROR RECOVERY METHODS:**

Although there are some of error recovery methods that are automatically performed by the compiler, it is not recommended as it may lead to problems in other parts of code if those are not changed accordingly.

These are four methods by which an invalid identifier is turned into a valid one by the compiler-

* **Delete:**

This method deletes extra illegal characters from the string.

* **Insert:**

This method is used only when there is digit(s) present at the start of the input string. Inserting character (namely L/\_) at the of the string can make it a valid identifier.

* **Transpose:**

This method works only when there is a single digit present at the start of the string followed by a letter or underscore. Transposing or interchanging two adjacent characters at the first and second places make the string a valid identifier.

* **Replace:**

This method works by changing the illegal characters to a letter or digit or underscore making the string a valid identifier.