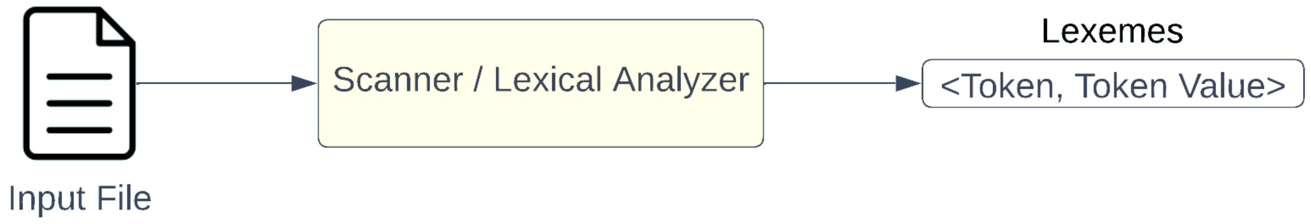
CS424 Compiler Construction

Assignment # 1
Report

Name: Muhammad Hamza Azeem
Reg No: 2020296

Scanner Design:



The input for the scanner is a source file containing the information we intend to process.

The scanner, also referred to as the lexical analyzer, receives the file's content and proceeds to analyze it by breaking it down into lexemes. Each lexeme comprises the identified token and its corresponding value.

Our tokens are categorized into seven different types, each serving a distinct purpose:

1. **Datatype:** Reserved keywords indicating the type of a declared variable, such as int, bool, float, or char.

2. **Keywords:** Other reserved keywords representing specific actions, including print, true, false, if, and else.

3. **Identifiers:** Names given to variables, where, for instance, in the line "int age = 3," "age" is the identifier.

4. **Values:** Representing permissible data that can be assigned to an identifier; in our program, we accept integers, decimals, and strings.

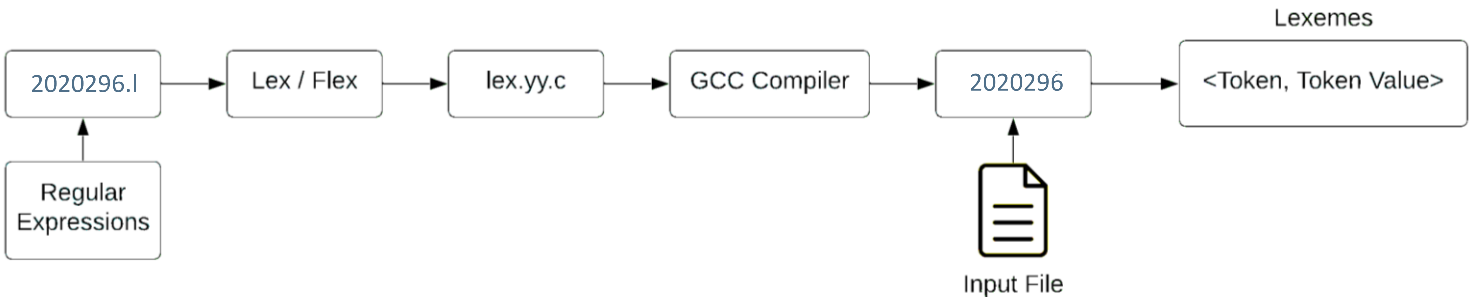
5. **Operators:** Tokens signifying various operators within our program, used for performing mathematical operations.

6. **Parenthesis:** Tokens encompassing parentheses, utilized to delineate the scope of a function or statement.

7. **Others:** Including two additional tokens—one for the assignment operator and another for the delimiter, employed to signify the end of a code segment.

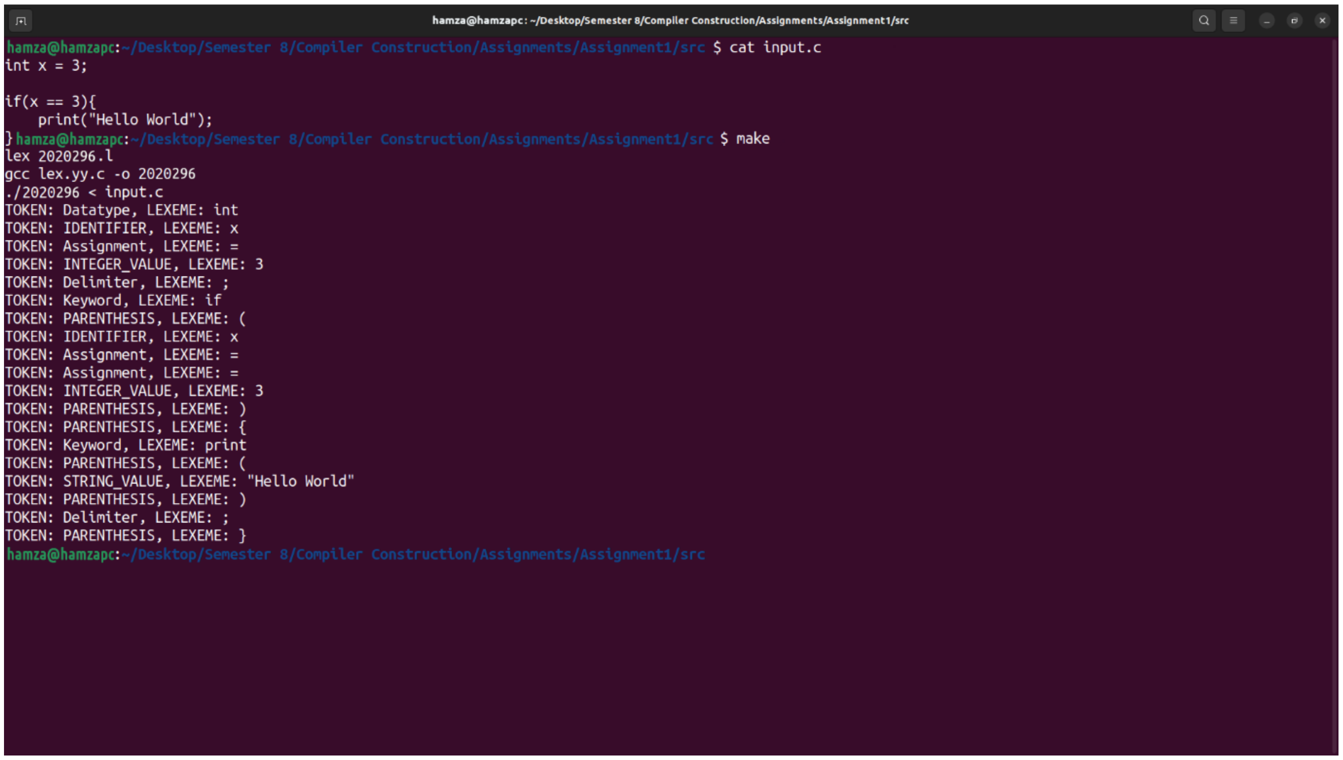
Implementation Details:

In the implementation, we have devised two files: one employing the Lex program and the other not utilizing it. In the non-Lex program, we open the input file in read-only mode using the Linux system call **open()**, read the file's contents, and store them in a character array. Subsequently, we tokenize the input stream by employing a series of **if-else** statements to compare the stream against reserved keywords using the **strcmp()** function.
In the Lex-based program, we initiate by creating a Lex file with the ".l" extension, encompassing rules for tokens expressed as regular expressions in the middle section of the file. The **yylex()** function is then utilized to execute lexical analysis on the input stream, matching it against tokens defined by the regular expressions. The process is graphically represented below.



Testcase:

Attached is a sample test case for the MiniLang language, where an integer variable is created, and its value's correctness is verified through an if-else statement block



```
hamza@hamzapc: ~/Desktop/Semester 8/Compiler Construction/Assignments/Assignment1/src
hamza@hamzapc:~/Desktop/Semester 8/Compiler Construction/Assignments/Assignment1/src $ cat input.c
int x = 3;

if(x == 3){
    print("Hello World");
}hamza@hamzapc:~/Desktop/Semester 8/Compiler Construction/Assignments/Assignment1/src $ make
lex 2020296.l
gcc lex.yy.c -o 2020296
./2020296 < input.c
TOKEN: Datatype, LEXEME: int
TOKEN: IDENTIFIER, LEXEME: x
TOKEN: Assignment, LEXEME: =
TOKEN: INTEGER_VALUE, LEXEME: 3
TOKEN: Delimiter, LEXEME: ;
TOKEN: Keyword, LEXEME: if
TOKEN: PARENTHESIS, LEXEME: (
TOKEN: IDENTIFIER, LEXEME: x
TOKEN: Assignment, LEXEME: =
TOKEN: Assignment, LEXEME: =
TOKEN: INTEGER_VALUE, LEXEME: 3
TOKEN: PARENTHESIS, LEXEME: )
TOKEN: PARENTHESIS, LEXEME: {
TOKEN: Keyword, LEXEME: print
TOKEN: PARENTHESIS, LEXEME: (
TOKEN: STRING_VALUE, LEXEME: "Hello World"
TOKEN: PARENTHESIS, LEXEME: )
TOKEN: Delimiter, LEXEME: ;
TOKEN: PARENTHESIS, LEXEME: }
hamza@hamzapc:~/Desktop/Semester 8/Compiler Construction/Assignments/Assignment1/src
```