

**Lab Terminal**

**Submitted By:**

Ramsha Kokab

FA21-BCS-010

**Course Instructor:**

Syed Bilal Haider Bukhari

**Course:**

Compiler Construction [Lab]

**Date:**

3rd January, 2024

**DEPARTMENT OF COMPUTER SCIENCE**

**COMSATS UNIVERSITY ISLAMABAD, ATTOCK CAMPUS**

### Q4: Explain input/output of your mini compiler.

* ***Lexical Analysis***
* *Input:*

The structure of the lex program consists of three sections:

{definition section}

%%

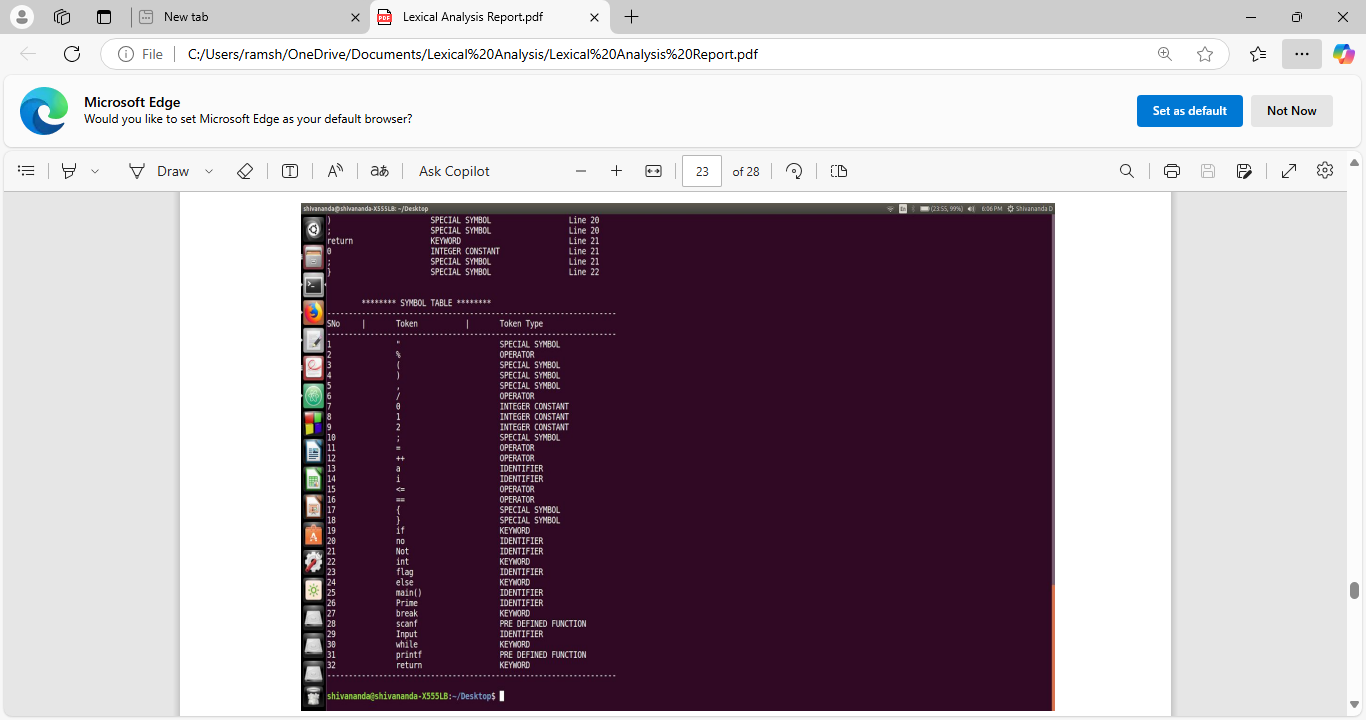
{rules section}

%%

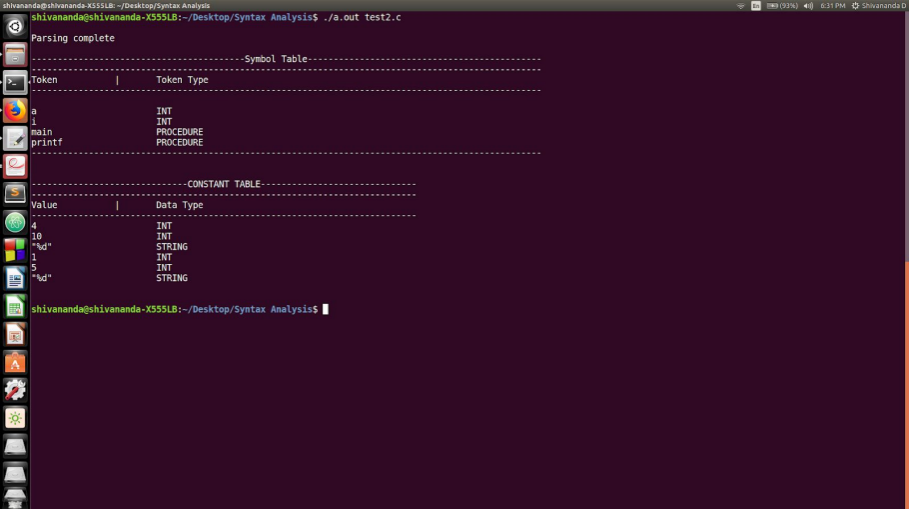
{C code section}

* The definition section defines macros and imports header files written in C. It is also possible to write any C code here, which will be copied verbatim into the generated source file.
* The rules section associates regular expression patterns with C statements. When the lexer sees text in the input matching a given pattern, it will execute the associated C code.
* The C code section contains C statements and functions that are copied verbatim to generated source file. These statements presumably contain code called by the rules in the rules section.
* *Output:*

The lexical analyzer outputs **tokens**, which are the smallest meaningful units of the code (e.g., keywords, identifiers, operators, literals).



* ***Syntax Analysis***
* *Input:* **Tokens** generated by Lexical Aanlyzer.
* *Output:* A **parse** tree or **syntax** tree.



* ***Semantic Analysis***
* *Input:*

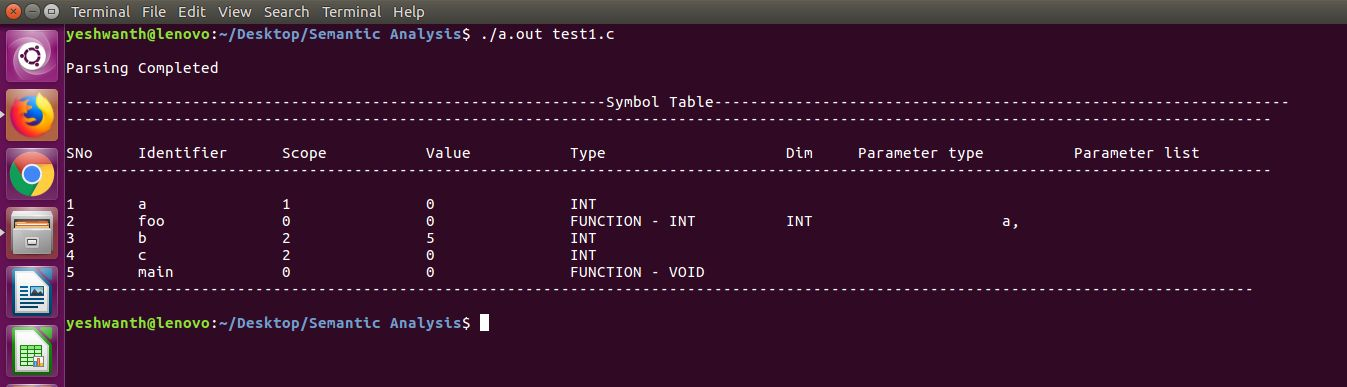
**Syntax tree** generated in Syntax Aanlysis phase.

It also requires information about variables, types, and scopes to ensure semantic correctness.

* *Output:*

The semantic analyzer outputs:

* An updated symbol table with type information, scope, and variable usage.
* Error messages for semantic issues such as type mismatches, undeclared variables, or scope violations.
* A successful validation message if no semantic errors are found.



* ***Intermediate Code Generation***
* *Input:*

The **annotated syntax tree** from the Semantic Analysis phase serves as the primary input.

The **symbol table** with information about variable names, types, and scopes is also used.

* *Output:* A **three-address code (TAC).**

