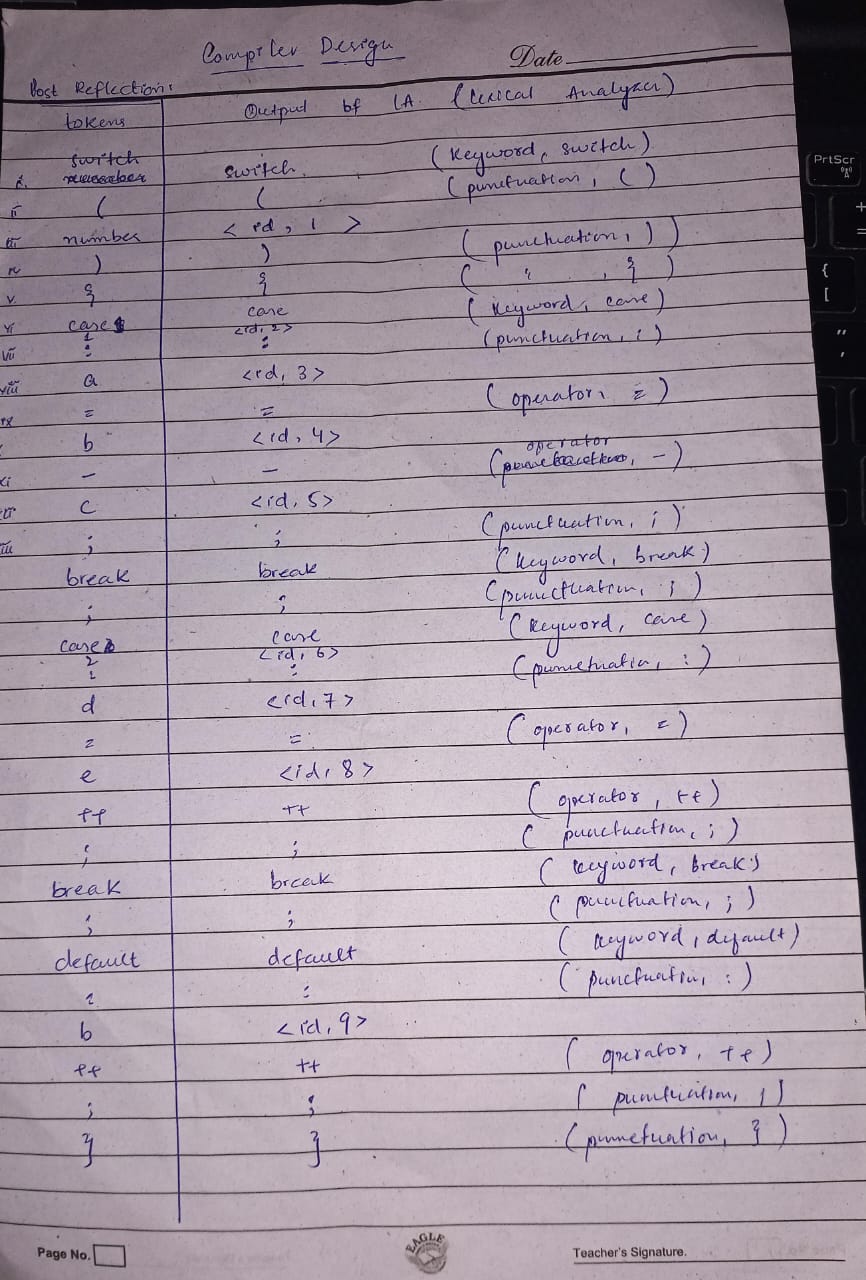
**Q1. Count the number of tokens in the following code; Also, generate the  
output of lexical analyzer**

****

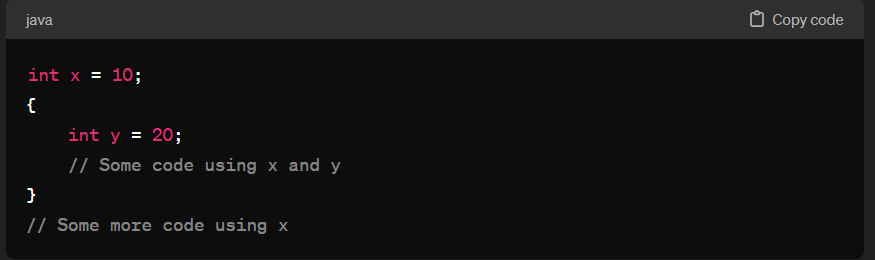
**Q2. Explain the use of symbol table with two example cases.**

A symbol table is a data structure used by compilers, interpreters, and other language processing tools to manage information about the symbols (such as variables, functions, classes, etc.) defined in a program. It associates each symbol with information like its name, type, scope, memory location, and other relevant attributes. Symbol tables play a crucial role in various language processing tasks, including parsing, semantic analysis, optimization, and code generation.

Here are two examples illustrating the use of symbol tables:

1. **Compilation and Semantic Analysis**:

Consider a compiler translating a high-level programming language like C or Java into machine code. During the compilation process, the compiler needs to perform semantic analysis to ensure that the program follows the language's rules and constraints. This includes checking variable declarations, resolving identifiers, and enforcing scoping rules.

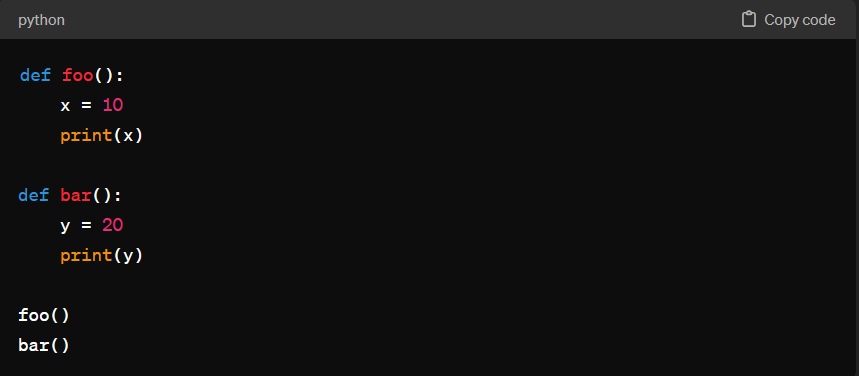


In this example, the compiler needs to keep track of the variables **x** and **y**, their types (**int**), and their scopes (global scope for **x** and block scope for **y**). The symbol table will store this information along with the memory addresses allocated for these variables.

1. **Interpreter or Runtime Environment**:

Symbol tables are also used during interpretation or execution of a program. Interpreters and runtime environments need to resolve symbols dynamically and manage memory allocation for variables and objects.

**Example**:



In this Python script, the interpreter maintains a symbol table to keep track of the functions **foo** and **bar**, their local variables (**x** and **y**), and their respective scopes. When executing each function, the interpreter uses the symbol table to resolve variable references and manage memory.