MUHAMMAD SHAHZAIB SHAHEEN

FA20-BCS-040

Q4: HOW FUNCTIONS WORK?

Lexical Analyzer

- **Input Source Code**
- **Tokenization:** The source code is divided into meaningful units called tokens. (keywords, identifiers, operators, constants, and symbols)
- **Regular Expressions and Patterns:** The Lexical Analyzer uses regular expressions or patterns to define the rules for recognizing different types of tokens.
- **Scanning:** scans the source code character by character.
- **Recognizing Tokens:**
- **Building Tokens:** When a token is recognized, the Lexical Analyzer builds the token by collecting the characters that form it.
- **Ignoring Whitespace and Comments:** The Lexical Analyzer typically ignores whitespace characters.
- **Output Token Stream:** The Lexical Analyzer produces a stream of tokens as its output.
- **Passing Tokens to the Syntax Analyzer:** The generated token stream is passed to the next phase of the compiler, the Syntax Analyzer (Parser).

Semantic Analysis

- **Input:** Semantic analysis takes the output of the lexical and syntax analysis phases as its input.
- **Symbol Table Construction:** The compiler constructs a symbol table, a data structure that stores information about variables, functions, and other program entities. This table keeps track of the names, types, and scopes of these entities.
- **Type Checking:** it checks that you're not trying to add a string to an integer.
- **Scope Analysis:** The compiler checks for variable scope violations.
- **Declaration Checking:** Semantic analysis verifies that variables and functions are declared before they are used.
- **Function Overloading and Signature Matching:** If the programming language supports function overloading, the compiler checks that functions with the same name have different parameter lists.
- **Constant Folding:** This involves evaluating constant expressions at compile-time to optimize the code.
- **Error Reporting:** If semantic errors are detected, such as type mismatches, undeclared variables, or scope violations, the compiler reports these errors to the user.

- **Intermediate Code Generation:** Depending on the compiler design, semantic analysis may also involve generating intermediate code.
- **Output:** The output of the semantic analysis phase is either an error report indicating the presence and nature of semantic errors or, in the absence of errors, an enhanced representation of the program that captures its semantic meaning.