**COMPILE CONSTRUCTION LAB FINAL**

**SUBMITTED BY: DUA NADEEM**

**SUBMITTED TO: SIR. BILAL HAIDER**

**REGISTRATION N0: FA21-BCS-016**

**Question#1**

**Explain two core functions of your mini compiler.**

Two core functions of the mini compiler are:

1. **Lexical Analysis (Scanning) –**

**Core Function:** The primary responsibility of the scanner is to read the raw source code text (which is just a string of characters) and transform it into a stream of *tokens*. These tokens are the basic building blocks of the language and represent the smallest meaningful units, like identifiers, keywords, operators, and literals.

* **Process:**
  + The Scanner reads the input source code character by character.
  + It groups characters that logically belong together, based on the rules of the source language. For example:
    - i, n, t, , x would be grouped into the keyword int and an identifier x.
    - + would be identified as the plus operator token.
    - 123 would be identified as the number literal 123.
    - "hello" could potentially be grouped as a string literal (If the mini-language supports them)
  + Whitespace (spaces, tabs, newlines) is generally ignored by the scanner (but might be relevant to track line numbers for error messages), it usually serves to separate tokens.
  + It assigns a type to each token.
  + It also usually saves the actual string form or the interpreted value of that token for later usage by parser and semantic analyzer
  + It passes these tokens to the parser for further processing.
  + It performs minimal checks. It has the ability to know what are the legal combinations of the input character stream for every language defined by the mini-compiler itself.

**2. Syntax Analysis (Parsing) -**

* **Core Function:** The parser receives the stream of tokens from the Scanner and attempts to construct an Abstract Syntax Tree (AST). The AST is a tree-like data structure that represents the hierarchical grammatical structure of the program based on its defined rules in formal grammar by compiler developer. The goal of parser is to ensure if all those combination rules of a specific language have been correctly fulfilled.
* **Process:**
  + The Parser reads tokens from the stream received by Scanner.cs, generally from left to right and following a certain algorithm such as recursively descendent parsing or any variant.
  + It follows rules, specified in the formal grammar.
  + It identifies how different elements belong to an statement and builds internal nodes in tree representation (e.g., assignment statement, if/else, mathematical operation or variable) based on programming languages rule defined for this parser.
  + It ensures token flow conforms to the formal grammar structure defined.
  + The parser creates node objects corresponding to tokens.
  + After completion, an AST will represent how is all these statements connected hierarchically.