Logo, company name

Description automatically generated

**COMSATS University Islamabad (CUI)**

**Lab Terminal**

**Submitted to: Sir Bilal Bukhari**

**Submitted By: Sohaib Safeer**

**Reg No: FA21-BCS-054**

**Course: Topics in Computer Science**

**Date: 3 jan, 2025**

Question 2 : give two core funtions of mini compiler ???

Certainly, let's delve deeper into the next() and expression() functions:

**1. next() Function:**

* **Purpose:**
  + The primary role of the next() function is to extract the next meaningful unit (token) from the source code.
  + It acts as the bridge between the raw source code and the rest of the compiler.
* **Key Operations:**
  + **Character-by-Character Reading:** It reads the source code character by character.
  + **Token Identification:**
    - It identifies different token types based on the current character and the following characters:
      * Identifiers (e.g., variables, function names)
      * Keywords (e.g., if, else, while)
      * Literals (e.g., numbers, strings)
      * Operators (e.g., +, -, \*, /, =)
      * Punctuation (e.g., (, ), {, })
  + **Token Classification:**
    - Once a token is identified, it is classified and stored in a suitable data structure.
  + **Source Code Advancement:**
    - The function advances the "read head" (pointer) to the next character in the source code after each token is extracted.

**2. expression() Function:**

* **Purpose:**
  + This function is crucial for parsing and evaluating expressions within the source code.
  + It ensures that expressions are evaluated correctly according to the rules of operator precedence and associativity.
* **Key Operations:**
  + **Recursive Descent:** The expression() function often employs a recursive descent approach to handle complex expressions.
  + **Operator Precedence:** It correctly handles the order of operations (e.g., multiplication and division before addition and subtraction).
  + **Associativity:** It handles left-to-right or right-to-left associativity for operators (e.g., a - b - c is evaluated as (a - b) - c).
  + **Intermediate Code Generation:** As it parses the expression, the expression() function generates a sequence of intermediate code instructions (e.g., load, store, arithmetic operations) that represent the computation.

**In essence:**

* next() provides the fundamental building blocks for the parser by breaking down the source code into individual tokens.
* expression() takes these tokens and constructs the meaning of the expressions, ensuring correct evaluation and generating the necessary instructions for the subsequent stages of the compilation process.