Question # 2:  
 explain core function of this mini compiler ?

Ans: **Lexical Analysis:**

* This function takes the input expression as a string and tokenizes it into smaller units (tokens) like numbers, operators, and parentheses.
* **Example**: "3 + (4 \* 5) \* 7" → Tokens: ["NUMBER: 3", "PLUS: +", "LPAREN: (", "NUMBER: 4", "TIMES: \*", "NUMBER: 5", "RPAREN:)", "TIMES: \*", "NUMBER: 7"]

**Syntax Parsing (Parser):**

* This function parses the tokens into a hierarchical structure called a parse tree. It checks the syntactic structure of the expression and ensures that it follows the rules of arithmetic expressions (e.g., operator precedence and parentheses).
* **Example**: The expression "3 + (4 \* 5) \* 7" would result in a parse tree representing the structure: + -> 3, \* -> 4, 5, 7.

**Semantic Analysis:**

* This function checks the tokens for semantic errors, though in this code, it's just a placeholder. Typically, it would check for errors like division by zero or type mismatches.
* **Example**: For "3 / 0", this function would throw an error indicating division by zero.

**Code Optimization:**

* This function optimizes the tokens by evaluating simple expressions like 3 + 5 into 8 directly, reducing unnecessary computations.
* **Example:** After optimization, "3 + 5" becomes "8".