**Core Functions of the mini compiler:**

**1. Lexer.cpp - std::**vector<Token> lex(std::string input) (Lexical Analysis)

**Explanation:**

* **Includes:** Necessary headers are included for string manipulation, I/O, and token handling.
* **Function Signature:** The function lex takes a std::string (the source code) as input and returns a std::vector<Token> (the tokens generated).
* **Iteration:** The function iterates through the input character by character using a while loop and index i.
* **Whitespace Handling:** If the character is whitespace, it is skipped, and the loop continues to the next character.
* **Digit Handling:** If the character is a digit, it starts collecting digits to form a number, until a non-digit character is encountered, then generates a NUMBER token.
* **Identifier Handling:** If the character is a letter, it collects alphanumeric characters to form an identifier. It checks if identifier is a keyword (int, if, while) and creates a keyword token, or else creates an IDENTIFIER token.
* **Operator Handling:** For the given operators like =, +, -, \*, /, (, ), {, }, ;, >, <, it creates the respective tokens.
* **Error Handling:** If an invalid character is encountered, it prints an error message and returns an empty vector. This is a simple error handling scheme.
* **Return Value:** The function returns a std::vector containing all generated Token objects.

**2. CodeGenerator.cpp - std::**string generateCode(std::shared\_ptr<Node> astRoot) (Code Generation)

**Explanation:**

* **Includes:** Includes headers for code generation, symbol table, and AST node manipulation.
* **generateCode function:**
  + Takes the root of the AST (astRoot) as input.
  + Creates an empty code string that will store the generated assembly-like code.
  + Creates a SymbolTable to track the declared variables.
  + Calls the generateAssemblyCode recursively to generate assembly from the root node, and then returns this code.
* **generateAssemblyCode function:**
  + Takes the current node of AST, the current code, and the symbol table as inputs
  + It uses a switch statement based on the node type.
  + **PROGRAM:** The function iterates over the children of program node.
  + **DECLARATION:** It adds a comment to indicate declaration of the variable to the code, and registers variable to symbol table.
  + **ASSIGNMENT:** It generates code to load the value of the expression in eax, then it moves this value to the assigned variable in memory.
  + **IF\_STATEMENT:** It generates code for the if conditional statement, generating labels for else block and end of if statements. The code does a comparison and jumps to else if it is 0 and jumps to endif after execution.
  + **WHILE\_STATEMENT:** It generates code for the while loop, creating a start label for the loop, and generating comparison and end label and jumps if the condition is not true, otherwise executes the body and jumps back to the start of the loop.
  + **BINARY\_OP:** For binary operation node, it generates assembly to evaluate the left and right side expression and then uses eax and ebx register to compute the value according to the operator.
  + **NUMBER:** It generates assembly code to load the value of number into eax.
  + **IDENTIFIER:** It generates assembly code to load the value of identifier (variable) from memory into eax.
  + **Default:** If node type is not recognized a comment is inserted indicating unknown node type.