**Semantic Analysis in Code:**

Unlike a traditional compiler with distinct phases, this code intertwines lexical analysis, syntax analysis, and semantic analysis. However, it does perform semantic checks to ensure that the provided code follows the intended rules.

Semantic analysis primarily involves:

1. **Type Checking (Implicit):** The code checks if identifiers are used appropriately based on whether they are declared types, variables, numbers, or reserved words. It assumes that variables store only integer values.
2. **Declaration Checking:** The code verifies if variables are declared before they are used.
3. **Statement Validation:** It checks the validity of statements such as assignments, and if-statements according to its defined rules.
4. **Expression Validation:** It checks the validity of expressions, by performing checks on operators and operands in assignments and conditions.

**Functions That Perform Semantic Analysis:**

The core semantic analysis is performed by a combination of functions working together, primarily focused on mainAnalyze and its helpers (analyze1a, analyze1b, analyze2a, analyze2b, analyze3a, analyze3b, analyze3Loop).

1. **mainAnalyze(int whichButton)**

* **Purpose:** This function is the entry point for semantic analysis. It drives the overall analysis process.
* **Functionality:**
* Splits the code input by spaces.
* Iterates through the code and calls helper functions to validate identifiers, variables, and statements.
* It sets a flag (f) when an error is found, it prints the errors using printErrors function.

1. **analyze1a(int i, string[] code, int inCondition) and analyze1b(int i, string[] code, int inCondition)**
   * **Purpose:** These functions analyze variable declarations.
   * **Functionality:**
     + Checks for the correct syntax of declaration, making sure there is an identifier followed by variable name and then by assignment or semicolon.
     + It checks for the syntax of declaring and initializing, and declaring and assigning at the same time.
     + Implicitly, the function stores declared variables with null as default value in memoryList in case of declaration.

analyze1b function does the same as analyze1a but with an initial check if the previous character was a semicolon.

1. **analyze2a**(int i, string[] code, int inCondition) and analyze2b(int i, string[] code, int inCondition)

* **Purpose:** These functions analyze assignment statements.
* **Functionality:** Validates assignment syntax, making sure there is a variable followed by =, and expression, ending with semicolon.

analyze2b function does the same as analyze2a but with an initial check if the previous character was a semicolon.

1. **analyze3a(int i, string[] code) and analyze3b(int i, string[] code)**
   * **Purpose:** These functions analyze if statements.
   * **Functionality:**
     + Verifies if statement syntax (if keyword, condition within parentheses, then block within curly braces, with statements inside curly braces).
     + It uses helper function analyze3Loop to process the conditions that have && or || keywords.

analyze3b function does the same as analyze3a but with an initial check if the previous character was a semicolon.

1. **analyze3Loop(int i, string[] code)**

* **Purpose:** Analyzes complex conditions in if statement that contains && or ||.
* **Functionality:** It has same functionality as the condition part of the analyze3a, and keeps looping using recursion to validate those complex conditions have valid syntax.