**Purpose:** The analyzeNode function traverses the Abstract Syntax Tree (AST) to perform semantic checks, ensuring the source code adheres to the language's semantic rules.

**Parameters:**

* · ASTNode \*node: Pointer to the current node in the AST.
* SymbolTable \*symbolTable: Pointer to the symbol table, which keeps track of declared identifiers and their attributes.

**Functionality:**

* · **Null Check:** If the current node is NULL, the function returns immediately.
* **Switch Statement:** Determines the type of the current AST node and performs corresponding semantic checks:
  + **Declaration:** Attempts to add the variable to the symbol table. If it already exists, an error message is printed indicating a redeclaration.
  + **Assignment:**
    - Checks if the variable has been declared. If not, an error message is printed.
    - Verifies type compatibility between the variable and the assigned value. If there's a mismatch, an error message is printed.
  + **Function Call:** Validates the function's existence and checks if the provided arguments match the function's parameters. If there's an issue, an error message is printed.
  + **Default Case:** Handles other node types as needed.
* **Recursive Analysis:** The function recursively calls itself for each child of the current node, ensuring a comprehensive traversal of the AST.

void analyzeNode(ASTNode \*node, SymbolTable \*symbolTable) {

    if (node == NULL) return;

    switch (node->type) {

        case DECLARATION:

            // Handle variable declaration

            if (!addToSymbolTable(symbolTable, node->identifier, node->dataType)) {

                printf("Error: Redeclaration of variable '%s'\n", node->identifier);

            }

            break;

        case ASSIGNMENT:

            // Ensure variable is declared

            if (!isDeclared(symbolTable, node->identifier)) {

                printf("Error: Undeclared variable '%s'\n", node->identifier);

                break;            }

          // Type checking for assignment

            if (!isTypeCompatible(symbolTable, node->identifier, node->value)) {

                printf("Error: Type mismatch in assignment to '%s'\n", node->identifier);

            }

            break;

      case FUNCTION\_CALL:

            // Check function exists and arguments match

            if (!isFunctionValid(symbolTable, node->identifier, node->arguments)) {

                printf("Error: Invalid function call '%s'\n", node->identifier);

            }

            break;

        // Add other cases like arithmetic expressions, return statements, etc.

        default:

            break;

    }

    // Recursively analyze child nodes

    for (int i = 0; i < node->numChildren; i++) {

        analyzeNode(node->children[i], symbolTable);

    }

}