C++ Program: Enforcing Mutable vs Immutable Variables

What this program does

- Maintains a **symbol table**
- Tracks whether a variable is **mutable or immutable**
- Detects **illegal reassignment/modification** of immutable variables
- Reports **semantic errors**

C++ Source Code

```
#include <iostream>
#include <unordered_map>
using namespace std;

// Symbol table entry
struct Symbol {
    string type;
    bool isMutable;
    bool isInitialized;
};

// Symbol table
unordered_map<string, Symbol> symbolTable;

// Declare a variable
void declareVariable(string name, string type, bool isMutable) {
    if (symbolTable.find(name) != symbolTable.end()) {
        cout << "Semantic Error: Variable '" << name << "' already
declared.\n";
        return;
    }
    symbolTable[name] = {type, isMutable, false};
    cout << "Declared variable '" << name << "'\n";
}

// Assign a value to a variable
void assignVariable(string name) {
    if (symbolTable.find(name) == symbolTable.end()) {
        cout << "Semantic Error: Variable '" << name << "' not
declared.\n";
        return;
    }

    Symbol &sym = symbolTable[name];

    // Check immutability
    if (!sym.isMutable && sym.isInitialized) {
        cout << "Semantic Error: Cannot modify immutable variable '"
<< name << "'\n";
        return;
    }

    sym.isInitialized = true;
    cout << "Assigned value to '" << name << "'\n";
}

int main() {

    // Simulating source code
    declareVariable("x", "int", false); // immutable
    assignVariable("x");                // allowed
    assignVariable("x");                // error

    cout << endl;

    declareVariable("y", "int", true);  // mutable
    assignVariable("y");                // allowed
    assignVariable("y");                // allowed
    return 0;
}
```

Sample Output

```
Declared variable 'x'
Assigned value to 'x'
Semantic Error: Cannot modify immutable variable 'x'

Declared variable 'y'
Assigned value to 'y'
Assigned value to 'y'
```

Explanation

- `isMutable = false` → behaves like `const` / `final`
- Immutable variables can be:
  - Assigned **only once**
  - **Never reassigned**
- Semantic checking happens **before code generation**
- This simulates **AST traversal + symbol table lookup**

Compiler Design Relevance

- Demonstrates **semantic rule enforcement**
- Uses **symbol table attributes**
- Detects **illegal rebinding**
- Foundation for **type checking & optimization**