Haseeb Ahmed (FA21-BCS-042)

Compiler Construction Lab Terminal Q3

**Optimization Example: Constant Folding**

**Input Code (Source File):**

int x = 3 + 5 \* 2;

**Without Optimization:**

1. The compiler tokenizes the input:
   * int, x, =, 3, +, 5, \*, 2, ;
2. During code generation, it generates instructions to calculate 3 + 5 \* 2 at runtime.

**With Optimization (Constant Folding):**

1. During semantic analysis, the compiler recognizes that 3, 5, and 2 are constants and the operators (+ and \*) form a constant expression.
2. It evaluates the expression:
   * 5 \* 2 = 10
   * 3 + 10 = 13
3. It replaces the expression with the result:
4. int x = 13;

**Generated Code:**

Instead of calculating 3 + 5 \* 2 during program execution, the generated code directly initializes x to 13.

**Benefits:**

1. **Reduces Runtime Overhead:** The computation is done at compile time, saving runtime resources.
2. **Improves Execution Speed:** Directly uses pre-computed values.
3. **Simplifies Generated Code:** Eliminates unnecessary intermediate instructions.

**Applicability in the Mini Compiler:**

If the semantic analysis phase (e.g., Semantique class) has rules for detecting and evaluating constant expressions, this optimization would be straightforward to implement. It can be extended to:

* Arithmetic expressions.
* Boolean expressions (e.g., replacing true && false with false).
* String concatenation of constants (e.g., "Hello" + " World" → "Hello World").

This showcases the potential for optimizing source code even in simpler compilers.