

**Lab Terminal**

**Name : Muhammad Wasiq**

**Reg. no . : fa21-bcs-039**

**Course : Compiler Construction**

**Q.3:  
  
  
  
1. Combined Regular Expressions**

* Before: Separate regex matching for each token type in multiple loops.
* After: A single combined regex handles all token types, reducing the number of passes over the code.
* Benefit: Improves performance by minimizing redundant matching operations.

std::string combinedRegex;

for (const auto &spec : tokenSpecs) {

if (!combinedRegex.empty())

combinedRegex += "|";

combinedRegex += spec.second;

}

std::regex tokenRegex(combinedRegex);

### **2. Efficient Token Skipping**

* Before: All tokens, including whitespace, were processed and added to the token list.
* After: Whitespace tokens are skipped immediately during token generation.
* Benefit: Reduces memory usage and speeds up further processing.

if (spec.first != "WHITESPACE") {

tokens.push\_back({spec.first, matchValue});

}

### **3. Reusable Token Matcher in Syntax Analysis**

* Before: Each token type required a separate if-check for matching.
* After: A reusable match lambda function simplifies and centralizes token validation.
* Benefit: Reduces code duplication and improves readability.

auto match = [&](const std::string &expectedType) -> Token {

if (index < tokens.size() && tokens[index].type == expectedType) {

return tokens[index++];

}

throw std::runtime\_error("Syntax Error: Unexpected token " + tokens[index].value);

};

### **4. Semantic Type Validation Using Templates**

* Before: Manual checks for data type validity.
* After: Used C++'s type traits (std::is\_integral) to validate integer constraints generically.
* Benefit: Makes the semantic analysis phase more robust and extensible.

if (node.type == "return" && !std::is\_integral<decltype(node.value)>::value) {

throw std::runtime\_error("Semantic Error: Return value must be an integer.");

}

### **Summary of Benefits:**

* Performance: Reduced processing time by combining operations and eliminating unnecessary steps.
* Memory Efficiency: Avoids storing irrelevant tokens.
* Maintainability: Cleaner and modular code structure for future modifications.
* Robustness: Enhanced error handling and type safety during analysis phases.