Project Progress Report: Evaluation I

Project Title: Trisynth – Custom Native Compiler

Group ID: 1 **Date:** February 2, 2026

Team Members:

- Chitraksh Vasantati (CS23B054) P. Sathvik (CS23B042) S. Danish Dada (CS23B047)

1. WORK DONE SO FAR (WEEKS 1–2)

We have successfully completed the foundational **Frontend Phase** of the Trisynth compiler.

- **Language Specification:** Finalized the *TrisynthC* grammar using EBNF, supporting strict typing (`uint32`, `int`, `bool`), control flow constructs (`if`, `while`), and function definitions.
- **Lexical Analysis (Week 1):** Implemented a regex-based lexer in Python that tokenizes keywords, operators, and literals, while handling comments, whitespace, and accurate line-number tracking for error reporting.
- **Syntax Analysis (Week 2):** Developed a recursive-descent parser that validates the token stream and constructs a hierarchical Abstract Syntax Tree (AST), including nodes such as `BinaryOp` and `VarDecl`.

2. WORK PLANNED TO DEMONSTRATE

We will demonstrate the complete frontend pipeline during the evaluation:

1. **Live Tokenization:** Conversion of raw source code (e.g., factorial computation) into a verified token stream.
2. **AST Visualization:** Parsing tokens to display the AST structure, validating correct operator precedence and syntactic hierarchy.
3. **Error Handling:** Demonstration of syntax error detection (e.g., missing semicolons) with precise line-number reporting.

3. DEVIATIONS WITH JUSTIFICATION

None. The project is progressing strictly according to the predefined weekly milestone plan. The lexer and parser modules are fully implemented and integrated ahead of the next phase.

4. PLANNED WORK FOR NEXT DEMONSTRATION

By the next evaluation (February 23), we plan to complete the Semantic Analysis phase, IR generation, and initial optimization passes (Weeks 3–7):

Week 3 **Semantic Analysis:** Implementation of symbol tables for scope resolution and strict type checking to prevent semantic errors.

Week 4 **Intermediate Representation (IR):** Lowering the AST into a linear, machine-independent IR.

Week 5 **IR Validation:** Verification of IR correctness and construction of the control-flow graph (CFG).

Week 6 **Optimization Pass I:** Implementation of constant folding to evaluate constant expressions at compile time.

Week 7 **Optimization Pass II:** Implementation of dead code elimination to remove unreachable or redundant instructions.