OPERATING SYSTEM(SE) PROJECT PROPOSAL

# PROJECT AND TEAM INFORMATION

Project Title  
  
Fast-C-Lite: A Minimalistic C Compiler

Student/Team Information

Team Name:  
  
MinimalComp

Team member 1 (Team Lead)  
  
Vashisth, Shivansh – 220212001 – shivanshvashisthsv08@gmail.com

Team member 2  
  
Negi, Ishita – 22022994 – ishitanegi016@gmail.com

Team member 3  
  
Saini, Samar – 220112209 – sainisamar009@gmail.com

Team member 4  
  
Dewal, Anurag – 220111378 – anuragdeval02@gmail.com

# PROJECT PROGRESS DESCRIPTION (35 pts)

## Project Abstract (2 pts)

Fast-C-Lite is a lightweight C language compiler developed using Lex and Yacc, designed to demonstrate the essential phases of compilation. It performs lexical analysis, syntax parsing, semantic analysis, and intermediate code generation. The system builds a symbol table, constructs a parse tree, and outputs three-address code (TAC). It supports conditionals, loops, and variable assignments. Designed with modularity, it allows students to understand the core functionality of compilers through clean terminal-based outputs.

## Updated Project Approach and Architecture (2 pts)

Compiler Architecture:  
  
- Lexical Analysis: Implemented in lexer.l using Lex, scans input for tokens, identifiers, constants, etc.  
- Syntax Parsing: Implemented using Yacc in parser.y, builds the grammar for expressions, conditionals, and loops.  
- Semantic Analysis: Builds symbol table and ensures type correctness.  
- Intermediate Code Generation: Produces TAC with labels and temporary variables.  
- Input Handling: Accepts code via stdin or file redirection using yyin.  
- Output Phases: Clearly prints each phase: Lexical Table → Parse Tree → Intermediate Code.

## Tasks Completed (7 pts)

Task Team Member  
Lexical Analyzer (lexer.l) Shivansh Vashisth  
Syntax Parser (parser.y) Shivansh Vashisth  
Symbol Table Logic Ishita Negi  
Intermediate Code Generation Samar Saini  
Parse Tree In-order Traversal Anurag Dewal  
File Input via yyin Shivansh Vashisth  
Keyword, Constant, Identifier Recognition Ishita Negi

## Challenges/Roadblocks (7 pts)

- Symbol redefinition conflicts in lex.yy.c and y.tab.c resolved by modularizing shared logic into symtable.c.  
- Initial segmentation faults due to unhandled syntax errors; improved Yacc error rules.  
- #include lines caused syntax errors — resolved by skipping preprocessor lines in lexer.  
- Multiple shift/reduce conflicts managed through grammar refinement.  
- File input and stdin handling conflicted — resolved using conditional file opening and yyin.  
- Grammar complexity increased with nested if-else and for loops — resolved by separating condition rules.  
- Intermediate code generation debugging for label jumps and temporary assignments was time-consuming.

## Tasks Pending (7 pts)

Task Team Member  
Code Optimization (TAC level) Samar Saini  
Error Handling & Reporting Anurag Dewal  
Grammar for switch-case Shivansh Vashisth  
Documentation + Sample Runs Ishita Negi  
Assembly Code Generation (Bonus) Shivansh Vashisth  
Graphical Parse Tree Output (Bonus) Anurag Dewal  
Test Cases & Validation Script Ishita Negi

## Project Outcome/Deliverables (2 pts)

- Lexical analyzer, syntax parser, and semantic analyzer fully working.  
- Intermediate code generation output with conditional and loop handling.  
- Symbol table generation with identifiers, constants, and line numbers.  
- CLI interface that compiles .c files via standard input or file redirection.  
- Structured output phases: Lexical → Syntax → Semantic → Intermediate Code.

## Progress Overview (2 pts)

- ~80% of the compiler core is implemented and functional.  
- Input/output, symbol table, parse tree, and intermediate code generation are working.  
- Grammar enhancement, optimization, and full error handling are in progress.  
- Optional features (assembly generation, graphical parse tree) are proposed but not critical.

## Codebase Information (2 pts)

Files Used:  
- lexer.l — Token scanning via Lex  
- parser.y — Grammar and parsing via Yacc  
- symtable.c — Shared function definitions  
- symtable.h — Function declarations + global variable externs  
- compiler — Final GCC-compiled executable  
  
Key Code Components:  
- create\_table(), insert() — Symbol table logic  
- yyparse() — Parser entry  
- Label and temporary variable generation for TAC  
- Error recovery and grammar conflict handling

## Testing and Validation Status (2 pts)

Test Case Status Notes  
Lexical Tokenization Pass All tokens and identifiers correctly parsed  
Parse Tree Generation Pass In-order traversal output  
Symbol Table Output Pass Variable, constant, and keyword tracking  
TAC Generation Pass Handles loops, conditionals  
File Input (< filename.c) Pass Uses yyin to read from file  
Syntax Error Handling Partial Improved, but some constructs still crash

## Deliverables Progress (2 pts)

Deliverable Status  
Lexical Analyzer Completed  
Syntax Parser Completed  
Semantic Analyzer Completed  
Intermediate Code Generator Completed  
File Input Handling Completed  
Error Handling In Progress  
Documentation In Progress  
Optimization / Assembly Output Pending