





**Graphic Era**  
**Hill University**  
DEHRADUN • BHIMTAL • HALDWANI

## PROJECT AND TEAM INFORMATION

### Project Title

FastC-Lite: A Minimalist C Compiler

### Student/Team Information

Team Name:	
Team member 1 (Team Lead)	<p>Negi, Ishita – 22022994 <a href="mailto:Ishitanegi016@gmail.com">Ishitanegi016@gmail.com</a></p> 
Team member 2	<p>Saini, Samar– 220112209 <a href="mailto:sainisamar009@gmail.com">sainisamar009@gmail.com</a></p> 

Team member 3	<p>Vashisth, Shivansh – 220212001 <a href="mailto:shivanshvashisthsv08@gmail.com">shivanshvashisthsv08@gmail.com</a></p> 
Team member 4	<p>Dewal, Anurag – 220111378 <a href="mailto:anuragdeval02@gmail.com">anuragdeval02@gmail.com</a></p> 

## 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 Completed	Team Member
<ol style="list-style-type: none"> <li>1. Lexical Analyzer (lexer.l)</li> <li>2. Syntax Parser (parser.y)</li> <li>3. Symbol Table Logic</li> <li>4. Intermediate Code Generation</li> <li>5. Parse Tree In-order Traversal</li> <li>6. File Input via yyin</li> <li>7. Keyword, Constant, Identifier Recognition</li> </ol>	<p>Shivansh Vashisth</p> <p>Shivansh Vashisth</p> <p>Ishita Negi</p> <p>Samar Saini</p> <p>Anurag Dewal</p> <p>Anurag Dewal</p> <p>Ishita Negi and Samar Saini</p>

## Challenges/Roadblocks (7 pts)

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

## Tasks Pending (7 pts)

Task Pending	Team Member (to complete the task)
<ol style="list-style-type: none"> <li>1. Code Optimization (TAC level)</li> <li>2. Error Handling &amp; Reporting</li> <li>3. Documentation + Sample Runs</li> <li>4. Assembly Code Generation</li> <li>5. Test Cases &amp; Validation Script</li> </ol>	<p>Samar Saini</p> <p>Anurag Dewal</p> <p>Ishita Negi</p> <p>Shivansh Vashisth</p> <p>Samar Saini</p>

## Project Outcome/Deliverables (2 pts)

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

## Progress Overview (2 pts)

1. ~80% of the compiler core is implemented and functional.
2. Input/output, symbol table, parse tree, and intermediate code generation are working.
3. Grammar enhancement, optimization, and full error handling are in progress.
4. Optional features (assembly generation, graphical parse tree) are proposed but not critical.

## Codebase Information (2 pts)

1. **Lexer and Parser:** Built using **Lex (lexer.l)** and **Yacc (parser.y)**  
Tokenizes the input source code and parses it using grammar rules to build the parse tree.
2. **Symbol Table & Shared Logic:** Modularized into **symtable.c + symtable.h**  
Handles symbol insertion, lookup, and table display across lexical and syntax stages.
3. **Compiler Executable:** Final binary (compiler) compiled using GCC with -lfl linking Flex runtime.
4. **Repository:** <https://github.com/shivanshvashisth/FastC-Lite>
5. **Key components and commits:**
  - lexer.l — Token recognition (keywords, identifiers, constants, comments)
  - parser.y — Grammar for loops, conditionals, assignments; generates parse tree
  - symtable.c — Functions for inserting/searching symbols
  - symtable.h — Exposes shared declarations and externs
  - main() (inside parser.y) — Controls parsing pipeline and calls analysis phases

## Testing and Validation Status (2 pts)

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

## Deliverables Progress (2 pts)

Deliverable	Status
1. Lexical Analyzer	Completed
2. Syntax Parser	Completed
3. Semantic Analyzer	Completed
4. Intermediate Code Generator	Completed
5. File Input Handling	Completed
6. Error Handling	In Progress
7. Documentation	In Progress

