# PROJECT AND TEAM INFORMATION

## Project Title

|  |
| --- |
| **Design and Implementation of a C Subset Interpreter** |

## Student / Team Information

|  |  |
| --- | --- |
| Team Name: | Bug-Smashers |
| **Team member 1 (Team Lead)**  Khushi Chauhan  Student ID: 220221623  Email: chauhankhushikc03@gmail.com |  |
| **Team member 2**  Saumya Mishra  Student ID : 220221344  Email: msaumyaa08@gmail.com |  |
| **Team member 3**  Dev Shukla  Student ID : 220211545  Email: devu020asa@gmail.com |  |
| **Team member 4**  Naval Kishore Singh  Student ID : 210211071  Email: kishornaval002004@gmail.com |  |

# PROPOSAL DESCRIPTION:

## Motivation

|  |
| --- |
| Understanding how compilers and interpreters work at a deeper level is crucial for systems and software engineers. However, many learners find it abstract.  This project aims to simplify that learning curve by creating an interpreter for a subset of the C programming language. By handling code directly and interpreting it line-by-line, the interpreter will allow students and developers to grasp the inner workings of parsing, syntax analysis, semantic checking, and runtime execution.  This tool could also be used as an educational resource in compiler design courses or as a testing utility for small C code snippets, fostering better programming practices and deeper insight into language translation mechanisms. |

## State of the Art / Current solution

|  |
| --- |
| Currently, interpreters exist for languages like Python, JavaScript, and Lua.  The C language, however, is primarily compiled using GCC, Clang, or similar compilers, which produce binary executables. While tools like C-INT (C Interpreter) exist, they are large and not beginner friendly. For educational purposes, there is a lack of lightweight, easy-to-understand interpreters specifically targeting the basic constructs of C. Our project aims to fill that gap by focusing on a clean, minimal interpreter that can serve as both a learning tool and a foundational framework for further development. |

## 

## Project Goals and Milestones

|  |
| --- |
| **Goals:**   * Build an interpreter that supports a simplified subset of the C language. * Support basic data types, variable declarations, arithmetic operations, loops, conditionals, and print statements. * Provide accurate parsing, interpretation, and error handling.   **Milestones:**   * **Week 1:** Finalize subset grammar and design lexical rules. * **Week 2:** Implement tokenizer and parser. * **Week 3:** Build abstract syntax tree and interpreter engine. * **Week 4:** Add support for control flow and symbol table. * **Week 5:** Add error handling, testing, and documentation. |

## Project Approach

|  |
| --- |
| The interpreter will be built using **C++** due to its compatibility with the C language and efficient memory handling. The project will follow the classical compiler design flow:   1. **Lexical Analysis** using custom tokenizer 2. **Syntax Analysis** via recursive descent parsing or Bison/Flex 3. **AST Construction** for intermediate representation 4. **Semantic Analysis** for variable scope and type checks 5. **Evaluation Engine** for direct execution of AST nodes   We’ll use **VS Code** for development and **Git** for version control. The system will include a REPL (Read-Eval-Print Loop) to allow users to input and execute C-like code directly. |

## System Architecture:

|  |
| --- |
|  |

## Project Outcome / Deliverables

|  |
| --- |
| * A fully functional C Subset Interpreter executable. * Support for input code via REPL or file. * Syntax and runtime error reporting. * Documentation with supported syntax, grammar rules, and example programs. * Demo showcasing interpretation of basic programs involving variables, control flow, and expressions. |

# Assumptions

|  |
| --- |
| * The user will only use the supported subset of C. * Only integer data types are used for simplicity. * No function declarations or pointers will be handled in this version. |

## References

|  |
| --- |
| * Aho, Lam, Sethi, Ullman – Compilers: Principles, Techniques and Tools * https://www.geeksforgeeks.org/compilers-set-1-introduction * https://flex.sourceforge.net/ – Lexical analyser generator * https://www.gnu.org/software/bison/ – Parser generator * <https://github.com/lotabout/write-a-C-interpreter> |