CS-4031: Compiler Construction Project - Reflection

Group Members:

Student ID	Name
22K-0500	Anas Saleem
22K-4487	Muhammad Usman Sohail
22K-4546	Fiza Farooq
22K-4602	Emanay Arshad
22K-4658	Syed Hussamuddin

Reflection

This project required a complete implementation of a compiler pipeline for a custom language of our choice. We opted to work on a small data-flow language built around table operations; the language supports loading datasets, filtering rows, mapping new fields, computing aggregates, printing results, and iterating over rows. Its block-based structure and expression grammar made it expressive enough to demonstrate real compiler behavior while remaining manageable for implementation.

This project has widened our understanding of the different phases of the compiler pipeline and how they interact. Starting from the lexical phase, we learned how careful token definitions can simplify later stages, and the deterministic token handling that was implemented reinforced how early design decisions affect clarity downstream. As our grammar was quite lightweight, we chose to implement it using a recursive-descent parser which implemented operator precedence, block structures, and statement specialization to build the AST representation of the source code that was needed for semantic analysis and intermediate code generation. The semantic analysis phase introduced symbol tables, scopes, and type constraints and emphasized the importance of orderly, well-defined scopes. Generating an intermediate representation showed how program behavior can be modeled independently of the original source syntax. Even basic optimizations such as constant folding and dead-code elimination demonstrated how IR design affects the ease of transformation. Implementing a small execution backend further emphasized the value of a clean and predictable IR.

The implemented system is quite lightweight but scalable for more advanced data analysis. Our grammar could be extended to handle more built-in functions such as min( ) and max( ). Another key grammar improvement is adding the ability to export CSVs; something such as “save high_salary to “high_salary.csv” improves reusability of the system instead of building thetable from the ground up. Comments are also not included in the grammar, but are an essential improvement to be made in order for better usability and readability. At the current stage, one error in parsing can cause the whole program to stop. To improve error recovery, synchronization tokens and multiple error reporting can be implemented.

Overall, this project provided a comprehensive but focused understanding of compiler construction and showed how each phase contributes to a reliable end-to-end system.