**CPL Compiler - Project Documentation**

**Yehudit Yudelevich- 325905628**

**Overview**

This project implements a compiler for a small custom programming language named **CPL**. The compiler performs lexical analysis, parsing, semantic checks, and generates intermediate code in a custom QUAD format. The output is written to a .qud file.

The compiler is implemented in Python using the sly library, which provides the lexer and parser framework. The codebase is structured to be modular and testable, with each phase of the compilation separated into distinct components.

**Code Structure**

**1. cpq.py**

* Entry point of the program.
* Handles argument parsing, file validation, and error handling.
* Calls the compilation pipeline using the Compiler class.

**2. main\_process.py**

* Contains the Compiler class, which coordinates lexing, parsing, and output generation.
* Implements context manager protocol (\_\_enter\_\_, \_\_exit\_\_) for clean usage.

**3. lexser.py**

* Contains the CPLScanner class, which defines the lexical structure of CPL.
* Tokens include keywords, identifiers, numbers, operators, and delimiters.
* Tracks line numbers and reports lexical errors.

**4. parser.py**

* Contains the CPLParser class, implementing parsing rules using sly.Parser.
* Supports declarations, statements, expressions, conditionals, loops, input/output.
* Generates intermediate code using QUAD instructions.
* Includes semantic checks for type consistency and provides meaningful error messages.

**Design Decisions & Implementation Rationale**

**Modularity**: Each compiler phase (lexing, parsing, compiling) is isolated in its own file. This enhances readability, maintainability, and testing capabilities.

**Intermediate Code Generation**: The output format is line-based QUAD code, which is simple and suitable for further processing or interpretation.

**Custom Types & Utilities**:

* QuadResult dataclass represents code fragments.
* IDList assists in managing declarations.
* ErrorQueue simplifies error reporting during parsing.

**Pythonic Design**: Uses type hints, dataclasses, and context managers for cleaner structure and robustness.