# Lexical Analyzer Project Report

CPSC 323-11: Group 4

## Group Members

- Anar Otgonbaatar
- Andres Ugalde
- Daniel Felix
- Gary Samuel

## Code Functionality

This Ruby-based lexical analyzer reads a source code file (.txt), removes all whitespace and comments (including both single-line // and multi-line /* */ comments), and tokenizes the input. Each lexeme is classified into one of the following token types:

- keyword
- operator
- separator
- identifier
- literal (integer or string)

The output is displayed in the format:
**"<lexeme>" = <token>**

The program uses regular expressions to detect lexemes, such as identifiers, numbers, strings, and symbols. It maintains lists of known keywords, operators, and separators for accurate classification.

## Tokenization Methodology

The analyzer first removes comments using regular expressions:

- Single-line comments: **`//.*$`**
- Multi-line comments: **`/\*.*?\*/`** (with multiline flag)

After cleaning, the script uses a regular expression to extract lexemes:
**`/\b[a-zA-Z_][a-zA-Z0-9_]*\b|[0-9]+|\S/`**

This regular expression captures:

- keywords
- identifiers

- integer literals
- operators
- separators

Each lexeme is matched against defined categories in the following order:

- Keywords **(e.g., `int`, `return`, `if`, `else`)**
- Operators **(e.g., `+`, `-`, `>=`, `!=`, `=`)**
- Separators **(e.g., `{`, `}`, `(`, `)`, `;`)**
- Integer literals **(e.g., `42`)**
- String literals **(e.g., `"Hello"`)**
- Identifiers **(e.g., `main`, `a`, `b`)**

# Time and Space Complexity Analysis

Time Complexity:

- Reading the file: O(n)
- Removing comments and whitespace: O(n)
- Tokenizing using regex and classification: O(m), where m is the number of lexemes
- Overall Time Complexity: O(n + m)

Space Complexity:

- Storage of file content: O(n)
- Storage of token list: O(m)
- Overall Space Complexity: O(n + m)