FORMAL LANGUAGES AND AUTOMATA THEORY
WARM UP PROJECT ASSIGNMENT

REG#: _____ NAME#: _____

COURSE CODE: CS224 INSTRUCTOR: MUHAMMAD SAJID ALI

TOTAL MARKS: 100

Objective

This warm-up project aims to connect your theoretical understanding of Regular Expressions and Finite Automata with a practical application — by building a Lexical Analyzer using Flex for a C++ programming language.

Group Formation

Please make groups of exactly three students each. You are strongly advised to make balanced groups, with an objective to facilitate/provide peer-help to students with relatively weaker programming skills. Class Representative (CR) is kindly requested to make a group-members list on a single paper and make it convenient to submit the consolidated list before the next lecture.

Build Lexical Analyzer

In class we have thoroughly studied Regular Expressions (R.E) to generate different languages. It's time to put the studied concepts into action and get something useful out of it.

In this warmup project assignment, you will build Lexical Analyzer (LA) of basic compiler front-end. This means that your compiler should be able to read an input from source file and generate required tokens. For this task of the frontend, you will use ***flex*** to create a Lexical Analyzer for the C++ Language. Figure out all the tokens your language allows and write them out in a Lex file. Submit the Lex file along with example source files. Make sure examples cover most of the tokens supported by C++ language. Your LA will identify all tokens from a source input file and store them in an output file including:

- Token Type (e.g., keyword, identifier, operator, etc.)
- Token Value
- Line Number

Some sample token types include for your understanding but are not limited to:

- Keywords: `int, float, if, else, return, etc.`
- Identifiers: `variable`, `function` and `class names` etc
- Operators: `+, -, *, /, %, =, ==, etc.`
- Separators/Punctuation: `(, ), {, }, ;, ,, etc.`
- Literals: integer, float, and string constants
- Comments: single-line (`//`) and multi-line `(/* ... */)`
- Whitespace: should be ignored, but line numbers must still be tracked

Introduction to FLEX

Flex is a free and open-source software alternative to lex. It is a computer program that generates lexical analyzers. It is frequently used as the lex implementation together with Berkeley Yacc parser generator on BSD-derived operating systems, or together with GNU bison in *BSD ports and in Linux distributions. For more details and tutorials please visit this website.

Sample Input and Output:

- Sample Input

```
int main() {
    float x = 3.14;
    // This is a comment
    if (x > 0) {
        x = x + 1;
    }
    return 0;
}
```

- Sample Output

```
Line 1: Token = int          → Keyword
Line 1: Token = main         → Identifier
Line 1: Token = (            → Separator
Line 1: Token = )            → Separator
Line 1: Token = {            → Separator
Line 2: Token = float        → Keyword
Line 2: Token = x            → Identifier
Line 2: Token = =            → Operator
Line 2: Token = 3.14         → Float Literal
      ...
```

Warmup Project Report:

Add the following details to your warmup project report.

1. Group Members
2. Your Lexical Analyzer Name
3. Lexical Analyzer code
4. Sample Examples and Outputs for Lexical Analyzer

Submission Format:

Submit your complete project as a single zipped folder named - Warmup_Project_Assignment_<underscore separated list of reg number of all group members>. The zipped folder must include:

1. .l file (your flex code)
2. .cpp test files (used as input)
3. Output files or screenshots demonstrating the analyzer's results

Please submit the report separately, as a pdf file.

Grading:

1. Group Members
2. Lexical Analyzer (55%)
3. Project Report (30%)
4. Viva (15%)
5. Note: marks for the project report and lexical analyzer may be deducted based on performance in the viva