# Overflow — Mini Scripting Language Compiler

# User Manual / Project Report

## Introduction

Overflow is a lightweight scripting language compiler developed as part of the Compiler Design Laboratory course. This project demonstrates the complete journey of translating human-written code into executable actions using:

• Lexical Analysis (Flex)

• Syntax Analysis (Bison)

• Intermediate Representation (IR) / Execution

• Basic Semantic Handling (Symbol Table)

Overflow supports core programming constructs such as variable declarations, arithmetic expressions, printing, conditions, and loops. The purpose of this project is to give students a hands-on understanding of how real compilers work internally while keeping the language simple and easy to experiment with.

## System Overview

The Overflow compiler operates through three major phases:

### 1. Lexical Analysis

The Flex lexer scans the raw input program and breaks it into tokens such as:

LET, ID, NUMBER, PRINT, IF, ELSE, WHILE, operators, and symbols.

### 2. Syntax Analysis

The Bison parser checks whether the program follows the rules of the Overflow language. Valid statements get translated into a simple Intermediate Representation (IR).

3. IR Execution

A C-based module reads and executes the IR by evaluating expressions, storing variables, running loops, and printing results.

## 2. Supported Syntax

Overflow supports:

• Variable Declaration

• Assignments

• Print Statements

• If–Else Conditions

• While Loops

## 3. Writing a Program

Rules:

• Statements end with semicolons

• Blocks use {}

• Conditions use ()

• Expressions support + - * /

Sample Program:

```
let x = 5;

let y = x + 3;

print(y);

if (y > 6) {

   print(y);

} else {

   x = x + 1;
```

```
}
while (x < 10) {
    x = x + 1;
}
```

## 4. Building and Running the System

Step 1: Generate Lexer and Parser

```
flex lexer.l
bison -d parser.y
gcc -o overflow parser.tab.c lex.yy.c -lfl
```

Step 2: Generate IR

```
./overflow < sample.of > program.ir
```

Step 3: Execute IR

```
./overflow program.of
```

## 5. Troubleshooting

- Syntax errors: missing semicolons or braces
- Unknown identifiers: variable not declared
- Unexpected tokens: unsupported characters
- Infinite loops: incorrect while condition

## 6. Conclusion

Overflow demonstrates compiler construction fundamentals including tokenization, parsing, IR generation, and execution. The project is modular, simple, and extendable for academic learning.