Comment Generator for Code — User Manual

Introduction

The Comment Generator for Code is an educational tool designed for students studying Compiler Design.

This project demonstrates how raw source code is transformed into meaningful English comments using:

• Lexical Analysis (Flex)

• Syntax Analysis (Bison)

• Intermediate Representation (IR)

• Rule-based Natural Language Processing (Python)

The system supports a small set of language constructs such as assignments, printing, conditions, and loops.

The goal is to help learners understand compiler pipelines and apply NLP techniques to generate descriptive documentation automatically.

1. System Overview

The complete workflow of the Comment Generator consists of three phases:

1. Lexical Analysis

The Flex lexer scans the input program and converts text into tokens such as ID, NUMBER, PRINT, IF, WHILE, operators, and symbols.

2. Syntax Analysis

The Bison parser validates code structure and converts each statement into a simple Intermediate Representation (IR).

Example IR:

```
ASSIGN  x   5

PRINT   y
```

```
IF     y>5    BEGIN
```

3. NLP-Based Comment Generation

A Python script reads the IR and generates readable English comments using rule-based templates.

This modular approach makes the project easy to understand and extend.

2. Supported Syntax

The user can write programs using the following features:

• Assignments:

```
x = 5;

y = x + 3;
```

• Print Statements:

```
print(x);

print(x + 5);
```

• Conditional Statements:

```
if (x > 0) {

print(x);

} else {

print(0);

}
```

• While Loops:

```
while (x < 10) {
```

```
x = x + 1;

}
```

The syntax is intentionally simple to support educational parsing and comment generation.

3. Writing a Program

Input programs must follow these rules:

- Each statement ends with a semicolon
- Curly braces define blocks `{ }`
- Conditions are enclosed in parentheses
- Expressions may contain +, -, *, /
- Variables must begin with a letter

Example input program:

```
x = 5;

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 parser parser.tab.c lex.yy.c -lfl
```

Step 2: Generate IR from source code

```
./parser < sample.code > program.ir
```

Step 3: Convert IR to comments

```
python3 comment_gen.py program.ir > comments.txt
```

Sample IR output:

```
ASSIGN   x   5

ASSIGN   y   (x+3)

PRINT    y

IF       y>6   BEGIN
```

Sample comment output:

```
// Assign value of 5 to variable 'x'.

// Assign value of x+3 to variable 'y'.

// Print the value of y.

// If y>6 then:
```

5. Troubleshooting

Common issues include:

• Parse Error

Causes: Missing semicolon, unbalanced parentheses or braces.

• Unknown Token

Ensure only supported operators and identifiers are used.

• Empty Output

The input file may contain no valid statements or parser failed early.

If errors persist, re-check the code format and ensure the lexer and parser have been compiled correctly.

6. Conclusion

The Comment Generator for Code project successfully combines compiler construction.

By generating meaningful comments automatically, it demonstrates how syntax trees and IR can be interpreted through rule-based natural language templates.

This tool is ideal for academic learning, showcasing:

• Compiler pipeline understanding

• Practical Flex/Bison usage

• Basic NLP text generation

The project is compact, modular, and easy to extend with additional language features.