FOS-Lang (File Operation Scripting Language) Project Report

1. Introduction

FOS-Lang (File Operation Scripting Language) is a mini educational programming language designed to demonstrate essential file operations such as create, write, append, read, and delete. It is ideal for Compiler Design laboratory courses, particularly those using Flex, Bison, and a C backend.

The purpose of FOS-Lang is to help students understand lexical analysis, syntax parsing, semantic analysis, and interpreter/runtime development within a small but practical language environment.

2. Objectives of the Project

• To design a simple domain-specific language focused on file operations.

• To implement the language using Flex (scanner) and Bison (parser).

• To showcase semantic checking and runtime behavior of file operations.

• To create a clean and easy-to-understand language specification.

• To help students understand the structure and design of modern programming languages.

3. Language Design Overview

FOS-Lang focuses on minimal but meaningful constructs. It supports two primary data types (text and file), and five file manipulation operations (create, write, append, read, delete). The syntax resembles modern scripting languages to make it intuitive and practical.

4. Keyword Set

FOS-Lang reserves the following keywords:

file, text, create, write, append, read, delete, print, into, from, as

These keywords cannot be used as variable names

5. Data Types

FOS-Lang supports two primitive types:

1. file — represents a file variable bound to a filename.

2. text — represents a string container in memory. Variables must be declared before use.

6. Syntax and Example Statements

• Declaration:

file myFile; text content;

• Create file:

create file myFile as "info.txt";

• Write to file:

write "Hello" into myFile;

• Append to file:

append "World" into myFile;

• Read from file:

read from myFile into content;

• Delete file:

delete myFile;

• Print: print content;

7. Example Program

file myFile;

```
text data;
 create file myFile as "output.txt";
write "Name: Alex\n" into myFile;
 append "Age: 22\n" into myFile;
 read from myFile into data;
 print data;
delete myFile;
```

8. Compiler Architecture

The compiler/interpreter for FOS-Lang contains the following components:

1. Lexical Analyzer (Flex)

- Responsible for tokenizing input source code.
- Identifies keywords, identifiers, strings, and symbols.

2. Syntax Analyzer (Bison)

- Parses token stream using CFG grammar rules.
- Builds parse structure and triggers semantic actions

3. Semantic Analyzer

- Ensures variables are declared before use.
- Checks type compatibility (e.g., file vs text).
- Ensures correct usage of file operations.

4. Runtime System

- Implements file operations using C functions such as fopen, fwrite, fread, remove.

The architecture aligns with classical compiler design theory, making it suitable for academic demonstration.

9. Grammar (CFG Specification)

Below is the condensed version of the grammar used in Bison:

program → program stmt | stmt

stmt → decl ';' | create ';' | write ';' | append ';' | read ';' | delete ';' | print ';'

decl → 'file' ID | 'text' ID

create → 'create' 'file' ID 'as' STRING

write → 'write' (STRING | ID) 'into' ID

append → 'append' (STRING | ID) 'into'

ID read → 'read' 'from' ID 'into' ID

delete → 'delete'

ID print → 'print' (STRING | ID)

10. Semantic Rules

• Every variable must be declared before being used.

• File operations must target variables of type 'file'.

• Read operations must store into 'text' variables.

• Assignments must match types.

• Undefined or undeclared identifiers generate semantic errors.

11. Implementation Summary

The implementation is done using:

• lexer.l (Flex): Defines tokens, keywords, string literals, identifiers.

• parser.y (Bison): Contains grammar rules and semantic actions.

• Symbol Table: Tracks variable types, filenames, and text values.

• Runtime C Code: Performs file I/O operations. The project compiles into an executable 'foslang' that interprets FOS-Lang programs.

11. Conclusion

FOS-Lang successfully demonstrates the core concepts of compiler construction within a simple and practical domain. Its focus on real file operations makes it more engaging and realistic compared to purely arithmetic-based toy languages.

This project can be extended with functions, conditions, loops, or error recovery to turn it into a more complete scripting language.