Sia Programming Language — User Manual

1. Introduction

Sia (Syntax Interpreter & Analyzer) is an educational, Python-inspired programming language designed to demonstrate the fundamental phases of compilation using **Flex**, **Bison**, and a **C backend**.

Sia focuses on clarity, simplicity, and semantic rigor. Its hallmark feature is **automatic type inference**, supported by a clean imperative syntax and foundational control-flow constructs. It is a case-sensitive language where statements end with semicolons, blocks use curly braces, and whitespace is ignored except inside strings.

Sia source files use the extension: `.sia`

To compile a program:

```
./sia_compiler program.sia
```

2. Keywords

The following identifiers are reserved:

```
var, INT, FLOAT, STR,
IF, ELSE, WHILE,
SHOW, GET,
true, false
```

These can't be used as variable names.

3. Variables and Types

Sia supports three primitive types:

- **INT** — integer values
- **FLOAT** — floating-point values
- **STR** — string values

The language provides **two variable declaration styles**:

- Explicit Typed Declaration

- Type-Inferential Declaration

3.1 Explicit Typed Declaration

The programmer specifies the type clearly.

```
var INT age = 21;
var FLOAT pi = 3.14;
var STR name = "Sia";
```

3.2 Type-Inferential Declaration

Sia automatically determines the type from the initializer.

```
var x = 10;          // inferred INT
var y = 3.2;         // inferred FLOAT
var msg = "Hello"; // inferred STR
```

The semantic analyzer derives the type from:

- literal forms
- arithmetic expressions
- relational results

4 Assignments

Variables may be reassigned after declaration.

```
age = age + 1;
msg = "Welcome";
```

5. Literals

Sia supports the following literal forms:

- **Integer:** `10`, `0`, `-3`
- **Float:** `2.5`, `0.0`, `3.14`
- **String:** delimited by double quotes, e.g., `"Hello"`
- **Boolean:** `true`, `false`

6. Input and Output

Sia provides simple I/O functions using **GET** and **SHOW.**

6.1 Input — GET

Reads user input into a variable.

```
GET age;
```

6.2 Output — SHOW

Prints one or more expressions.

```
SHOW "Result:", value;
```

Arguments are printed sequentially.

7. Expressions

Sia supports arithmetic and relational expressions following standard precedence.

7.1 Arithmetic Operators

```
+   -   *   /
```

Example:

```
var total = a + b * 3;
```

7.2 Relational Operators

```
==   !=   <   <=   >   >=
```

Example:

```
age >= 18
```

7.3 Parentheses

Parentheses control evaluation order:

```
( a + b ) * c
```

8. Control Flow

8.1 IF / ELSE

Syntax:

```
IF (condition) {
    statement_list
} ELSE {
    statement_list
}
```

Example:

```
IF (age >= 18) {
    SHOW "Adult";
} ELSE {
    SHOW "Minor";
}
```

8.2 WHILE Loop

Syntax:

```
WHILE (condition) {
    statement_list
}
```

Example:

```
var INT i = 3;
WHILE (i > 0) {
    SHOW "Count:", i;
    i = i - 1;
}
```

9. Comments

Sia supports single-line comments:

```
// This is a comment
```

Multi-line comments are not part of the core language.

10. Example Program

```
var name = "User";
var INT n = 3;

SHOW "Enter your name:";
GET name;

SHOW "How many times?";
GET n;

WHILE (n > 0) {
    SHOW "Hello", name;
    n = n - 1;
}
```

11. Future Enhancements

- Multi-line comments (`/* ... */`)
- Logical operators (`and`, `or`, `not`)
- String concatenation using `+`
- Modulo operator (`%`)
- Nested block scoping
- User-defined functions (`func`, `return`)
- Advanced string operations
- Arrays and collections

These extensions are planned for future releases to transform Sia into a more expressive and practical language.

12. Conclusion

The Sia programming language offers a clean, instructive environment for exploring compiler construction. Its simplified design maintains academic rigor while keeping implementation accessible to beginning compiler developers.

By integrating **type inference**, structured **control flow**, and **TAC generation**, Sia serves as both a learning tool and a foundation for future language expansion.