Fourward Programming Language Specification

Introduction

Fourward is a simple, beginner friendly programming language made for learning programming concepts while keeping things intuitive. It emphasizes readability and straightforward syntax, making it ideal for first-time programmers.

Structure of Statements

Statements in Fourward are terminated by a semicolon (`;`). These statements can include variable declarations, arithmetic operations, conditionals, and loops. Indentation is not required but helps improve readability.

Example:

```
let x = 5;
if (x > 3) {
    print("x is greater than 3");
}
```

Reserved Words

The following keywords are reserved in Fourward and cannot be used as identifiers:

```
let, const, if, else, while, for, function, return, print, input, true,
false, null
```

Data Types

Fourward supports the following data types:

- **Integer (int)**: Whole numbers (`5`, `-3`)
- **Float (float)**: Decimal numbers (`3.14`, `-0.5`)
- **String (str)**: Text enclosed in double quotes (`"Hello"`)
- **Boolean (bool)**: Logical values (`true`, `false`)
- **Null (null)**: Represents the absence of a value

Arithmetic Operations

Fourward supports basic arithmetic operations:

- `+` Addition
- `-` Subtraction
- `*` Multiplication
- `/` Division
- `%` Modulus

Comparative Operators

- `==` Equal to
- `!=` Not equal to
- `>` Greater than
- `<` Less than
- `>=` Greater than or equal to
- `<=` Less than or equal to

Control Flow (Selection Sequences)

Fourward supports conditional execution using `if`, `else if`, and `else`.

Example:

```
if (x > 10) {
    print("x is large");
} else {
    print("x is small");
}
```

Loops (Repetition Sequences)

- **`while` loop**: Repeats as long as the condition is true.
- **`for` loop**: Iterates over a specified range or collection.

Example:

```
for (let i = 0; i < 5; i++) {
    print(i);
}
```

Functions

Functions are declared using the `function` keyword and may return a value using the `return` keyword.

Example:

```
function add(a, b) {
    return a + b;
}
```

Token Identification

Tokens are identified by their **line number** and **position** within the line. The tokenizer scans the input stream and assigns token types and locations. This information is stored in a **symbol table** for future reference.

Example Token Format:

```
Line 1, Col 5: let (keyword)
Line 1, Col 9: x (identifier)
Line 1, Col 11: = (operator)
Line 1, Col 13: 5 (integer)
```
