Flow — NovoFlow

Flow Programming Language Low-Level Specification

AlexDev404/NovoFlow

1. Lexical Structure

1.1. Comments

- Single line comments begin with `#` and continue to the end of the line.
- Multi-line comments begin with `#'''` and end with `'''#`.

1.2. Identifiers

- An identifier is a sequence of letters, digits, and underscores that starts with a letter or an underscore.
- Identifiers are case sensitive.

1.3. Keywords

- The following keywords are reserved and cannot be used as identifiers:
  - `flow`, `subflow`, `let`, `output`, `if`, `else`, `for`, `in`, `while`, `true`, `false`, `input`, and `print`.

1.4. Literals

- Integer literals are a sequence of digits.
- Floating-point literals are a sequence of digits with a decimal point.
- Boolean literals are `true` or `false`.
- String literals are enclosed in double quotes.

1.5. Operators

- The following operators are supported:
  - Arithmetic operators: `+`, `-`, `*`, `/`, `%`
  - Comparison operators: `==`, `!=`, `<`, `>`, `<=`, `>=`
  - Logical operators: `&&`, `||`, `!`

1.6. Punctuation

- The following punctuation is used in the Flow language:
  - `(`, `)`: Reserved for arithmetic operations.
  - `{`, `}`: Used to enclose code blocks.
  - `[`, `]`: Used to create lists.
  - `,`: Used to separate elements in a list.
  - `;`: Used to separate statements.

2. Syntax

2.1. Variables and Constants

- Variables are defined using the `let` keyword, followed by an identifier, an optional type annotation, and an optional initialization expression.
- Constants are defined using the `let` keyword, followed by an identifier, a mandatory type annotation, and an initialization expression.

2.2. Functions

- Functions are defined using the `flow` keyword, followed by an identifier, a list of input parameters enclosed in braces, and a block of code enclosed in curly braces.
- Subfunctions are defined using the `subflow` keyword, followed by an identifier, a list of input parameters enclosed in braces, and a block of code enclosed in curly braces.
- Functions can return a value using the `output` keyword, followed by an expression or a variable name.

2.3. Control Structures

- Conditional statements are defined using the `if` keyword, followed by an expression in parentheses, and a block of code enclosed in curly braces.
- Optional `else` clauses can be added to `if` statements to handle alternate cases.
- `for` loops are defined using the `for` keyword, followed by an identifier, the `in` keyword, and an expression in parentheses or a range expression, and a block of code enclosed in curly braces.
- `while` loops are defined using the `while` keyword, followed by an expression in parentheses, and a block of code enclosed in curly braces.
- The `break` and `continue` keywords can be used to exit or skip a loop.

2.4. Input and Output

- The `input` keyword can be used to read a string from the standard input.

- The `print` keyword can be used to write text to the standard output, followed by any number of expressions or variables separated by commas.

3. Types

3.1. Flow is a dynamically typed language, meaning that variable types are inferred at runtime.

4. Variables

Variables in Flow are declared using the `let` keyword. Variables can be initialized at declaration or left uninitialized.

```
let x = 5
let y
```

5. Operators

Flow supports all standard arithmetic operators (`+`, `-`, `*`, `/`, `%`) as well as comparison operators (`<`, `>`, `<=`, `>=`, `==`, `!=`).

6. Conditionals

Flow has the standard `if/else` conditional statement. The condition must evaluate to a boolean value.

```
let x = 5
if x > 3 {
   output "x is greater than 3"
} else {
   output("x is less than or equal to 3")
}
```

7. Loops

Flow supports the `while` loop and the `for` loop. The `for` loop can be used to iterate over arrays or ranges of numbers.

```
let i = 0
while i < 5 {
   output i
   i = i + 1
}
```

```
let arr = [1, 2, 3, 4, 5]
for let i in arr {
   output(arr[i])
}
```

8. Functions

Functions in Flow are defined using the `flow` keyword, followed by the function name, and the parameter list enclosed in braces. The function body is enclosed in braces as well.

```
flow Fibonacci n {
   if n <= 1 {
        output n
   } else {
        output Fibonacci(n - 1) + Fibonacci(n - 2)
   }
}
```

9. Libraries

Flow has a standard library that provides basic functionality for common tasks such as input/output, string manipulation, and mathematical operations. To use a function from the standard library, the function name is prefixed with the `lib` keyword.

```
let x = input
let y = lib.sqrt x

output y
```

10. Comments

Flow supports both single-line and multi-line comments. Single-line comments start with `#`, while multi-line comments are enclosed in `#''' '''#`.

```
# This is a single-line comment

#'''
This is a multiline comment

'''#
```

11. Reserved Words

The following words are reserved in Flow and cannot be used as identifiers:

```
flow, output, input, lib, let, if, else, while, for, in
```

12. Built-in Functions

- The `math` library provides built-in mathematical functions like `abs`, `sin`, `cos`, etc.
- The `print` function is used for outputting text to the console.
- The `map` function is used for applying a function to each item in a collection and returning a new collection.

13. Examples

Here are some examples of Flow code:

```
// Calculate the factorial of a number using recursion
flow factorial(n) {
if n <= 1 {
     output(1)
    }
    else {
        output(n * factorial(n - 1))
    }
}
// Calculate the sum of an array of numbers
let arr = [1, 2, 3, 4, 5]
let sum = 0
for let i in arr {
   sum = sum + arr[i]
}
output(sum)
```

```
# This is a program that generates a sequence of Fibonacci numbers using Flow
functions

flow fibonacci_sequence {
    input n: int

    # define a subflow to calculate Fibonacci numbers
    subflow fibonacci(n) {
        if n < 2 {
            output n
        }
```

```
        output fibonacci(n-1) + fibonacci(n-2)
    }

    # generate the Fibonacci sequence
    output sequence: list[int] = map range(n), fibonacci
}

# create an instance of the flow with an input value of 10
fibonacci_flow = fibonacci_sequence { n=10 }

# print the resulting sequence without braces
print "Fibonacci sequence:", fibonacci_flow.sequence[::-1]
```