Giovanni Boscan 10/23/2023

Lab 2 Report

What is FlowScript?

The new language created, FlowScript, serves as an extension of the DOT language, which is used to visualize how processes flow, design workflows, and control flows visually. This extension will allow the user to integrate the capability to script conditions, loops, and function calls by compiling the script and using the Job System Interface to give meaning to those processes.

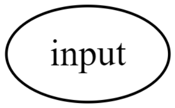
Programming Language Paradigms of FlowScript

FlowScript will be a compiled programming language, that will heavily rely on semantics since we are using the Dot language to give meaning to it. Therefore, it will give the user an interface to translate from code to an actual system of processes. With this level of abstraction, the programmer can handle any complex structure of Jobs. Since the language will be compilated, the compiler will generate an intermediate code that will use the Job System library to control the flow of execution of each script. This will result in a faster execution and will have the ability to optimize the script during compile time as well as handling errors. However, this process can be time-consuming. The first step that the compiler is going to follow is a lexical analysis, which will break down the script into tokens that can either be keywords, identifiers, values, or operators. After that step is completed, it will perform a syntax analysis, which will organize all the tokens into the FlowScript syntax by creating a hierarchical structure that represents the grammatical structure of the script. This can be achieved by creating an Abstract Syntax Tree that uses the Backus-Nour Form to put the tokens in the right nodes and parse the whole tree. This set of rules can be broken down into terminal symbols, which are the atoms of the language defined by the grammar, and non-terminal symbols which are the placeholders that can be replaced by other terminal or non-terminal symbols. By using all these sets of rules, FlowScript can be successfully parsed into C++ code that will execute the processes specified by the programmer in the script.

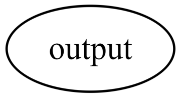
FlowScript Features, Sample Scripts and Functionality

The functional elements of the programming language are the input, output, and the Job, which will represent the process that the Job System will execute. Input and output will be used as keywords in FlowScript that serve as variables to store the initial and final data respectively, therefore, the user cannot have any Jobs with this name. Jobs, also called processes, will only accept one input and will produce one output, both in JSON format to further expand compatibility with other programming languages. The following image and code will represent a single input string:

```
digraph {
    input
}
```

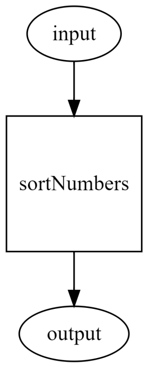


```
digraph {
    output
}
```



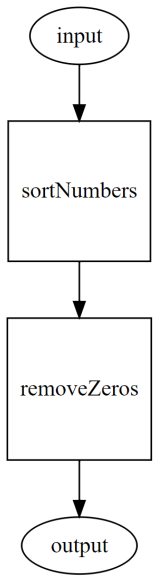
In the same way, a Job is represented as a single unit of process, specifically a pointer to a C++-written function that has exactly one input and one output. This unit is needed in order to execute a single Job. They will be denoted as a square shape and can have any name except for a FlowScript-specific keyword. The following is an example of a Job called "sortNumbers" that takes one input and returns one output:

```
digraph {
    input
    sortNumbers [shape="square"]
    output
    input -> sortNumbers
    sortNumbers -> output
}
```



Also, multiple jobs can be concatenated, and they will be executed in the order shown and the input of the next process will be the output of the previous. The following example shows how the process "sortNumbers" takes an input and then "removeZeros" will take that input and produce an output:

```
digraph {
    input
    output
    sortNumbers [shape="square"]
    removeZeros [shape="square"]
    input -> sortNumbers
    sortNumbers -> removeZeros
    removeZeros -> output
}
```



FlowScript will be able to handle boolean statements, which will be indicated by a "label". The possible operators available are:

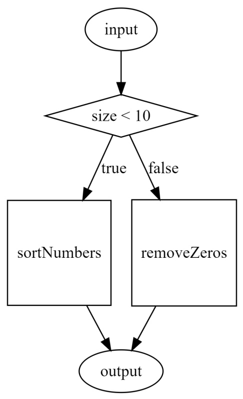
- == (equal to)
- != (not equal to)
- < (less than)
- <= (less than or equal to)
- > (greater than)
- >= (greater than or equal to)

The structure of the conditional has to start with an attribute, followed by any of the boolean operators above, and a target value. The equal to (==) and the not equal to (!=) operators will treat both sides as a string, while the rest of the operators will treat the two sides as numbers. Finally, conditionals can be associated with parenthesis and the operators available are the and operator, denoted as "and", and the or operator denoted as "or". For example:

```
size > 0
country == US
(number <= 10) or (animal != Dog)
(id >= 256) and (bit == 1)
```

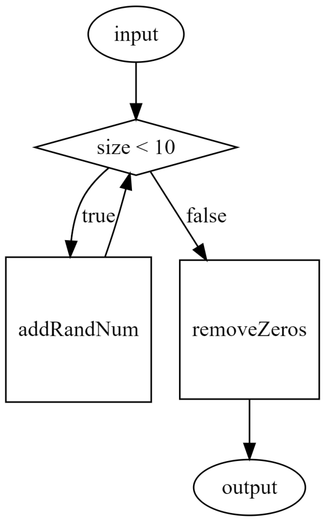
The language will also be capable of handling conditional executions to add even more meaning to the language and process flow. Therefore, an if statement will be visually denoted as a diamond. This block can have any name except for a keyword and a “label” is required in order to distinguish it from the other processes, which will contain the boolean condition itself. Also, they require two output arrows with labels either “true” or “false” that indicate if the condition is either true or false respectively. The following example will take an input, and depending if the size is less than 10 will either sort the array or remove the zeros and will produce an output that will be different given the condition:

```
digraph {
    input
    output

    condition1 [shape="diamond" label="size < 10"]

    sortNumbers [shape="square"]
    removeZeros [shape="square"]

    input -> condition1
    condition1 -> sortNumbers [label="true"]
    sortNumbers -> output

    condition1 -> removeZeros [label="false"]
    removeZeros -> output
}
```



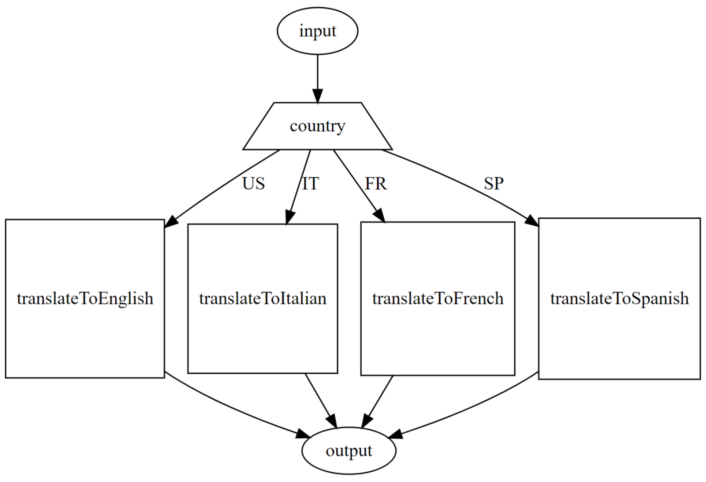
Since FlowScript already has a conditional block, it is also capable of performing loops over one or more processes given a condition. For example,

```
digraph {
    input
    output

    condition1 [shape="diamond" label="size < 10"]

    addRandNum [shape="square"]
    removeZeros [shape="square"]

    input -> condition1
    condition1 -> addRandNum [label="true"]
    addRandNum -> condition1

    condition1 -> removeZeros [label="false"]
    removeZeros -> output
}
```



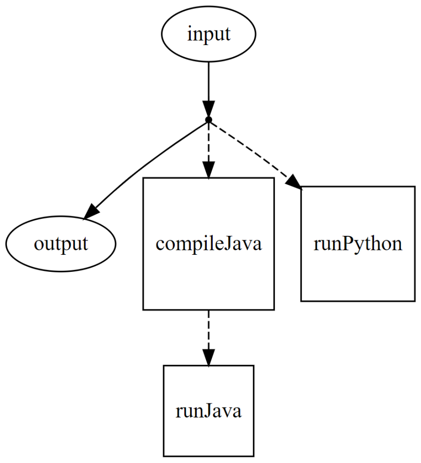
Now, in order to facilitate process branching, the programming language is able to create a switch statement. It will be visually denoted by a trapezium. The switch condition must have a label that indicates the attribute to compare, and its children will have the actual value to compare in a “label” too. For example, the following input will get translated depending on the country attribute present in the input data and will produce a translated output:

```
digraph {
    input
    output

    condition1 [shape="trapezium" label="country"]

    translateToEnglish [shape="square"]
    translateToItalian [shape="square"]
    translateToFrench [shape="square"]
    translateToSpanish [shape="square"]

    input -> condition1

    condition1 -> translateToEnglish [label="US"]
    translateToEnglish -> output

    condition1 -> translateToItalian [label="IT"]
    translateToItalian -> output

    condition1 -> translateToFrench [label="FR"]
    translateToFrench -> output

    condition1 -> translateToSpanish [label="SP"]
    translateToSpanish -> output
}
```



Parallel processing is also possible with FlowScript. Thanks to the features of the Job System library, multithreading can be easily achieved in the programming language if the programmer desires to perform multiple jobs at the same time. The keyword split will indicate that a branch of jobs will be executed at once and the output will be a merge JSON of all the outputs from those jobs once they are finished. It will be visually denoted as a dot. The Jobs that follow the dot are required to have the “dashed” style to indicate that they follow from a split.

```
digraph {
    input
    output

    compileJava [shape="square"]
    runPython [shape="square"]

    runJava [shape="square"]

    split [shape="point" label="split"]

    input -> split
    split -> compileJava [style = "dashed"]
    split -> runPython [style = "dashed"]
    compileJava -> runJava [style = "dashed"]
    split -> output
}
```



Final Overview

The programming language elements of FlowScript include variables, which are represented by the keywords “input” and “output” and have an oval shape. They are used to hold the values of the initial data and the output (both in JSON format) data once all the jobs are finished. Jobs or processes are represented as squares and they represent the execution of a function that takes a single input and produces a single output. FlowScript can also interpret conditional statements in order to be used by if statements or loops. The if statement is a boolean and is denoted by a diamond shape that can branch the process flow depending on the condition result is either true or false. Also, the switch statement is provided as a cleaner solution for condition-matching cases that require more than two branches at once and is represented by a trapezium. Additionally, the loop structure can be achieved by using conditional statements, which allow the repetitive execution of a specific process. Lastly, multithreading is implemented to allow the programmer to execute multiple jobs at the same time and merge the outputs in a JSON format.