Wind: A Flow based Programming Language

Joshua Weinstein

2018

Contents

0.1 Preface

This book is intended as an overview of the Wind programming language. It discusses the paradigm of flow-based programming, and the principles of it. The book discusses the advantages and disadvantages of flow-based programming, while then proceeding onto the syntax and usage of the Wind language. The book also, in detail, describes the C implementation of the language. The state management, the instructions, the translation of source code, and execution are all covered.

The Wind language is not a fully fledged, general purpose programming language. It is a language with a fundamental set of computation tools and components. Wind was developed with the following goals in mind.

Goals

1. An extremely light-weight language that is highly portable.
2. A programming language which does not use dynamic memory allocation.
3. A fluid, highly dynamically typed runtime.
4. A system that allows efficient transfers of immutable data.

The most unique element of Wind is that it is a "bare bones" language. It has no abstract syntax tree, no tokenizer, parser, or standard library. It reads and executes instructions directly from source code, which transition and alter several internal buffers. From a high level overview, these buffers pass around data to one another allowing the state of the data in the program to change. This choice of design permits the language to focus on manipulating and changing data, instead of managing and allocating resources.

Overall, the hope is for this book to serve as an inspiration or reference for computing and programming in a flow-based manner.

0.2 Dedication

This book is dedicated to my mother, Vida, who has guided me and raised me to be an incredible human being I am today. I would not be who I am today without her support.

0.3 About the Author

Joshua Weinstein Grew up in the sunny, green fields of Los Angeles, California. He went to the University of California Berkeley, where he studied Computer Science and Fine Art. He now works as a software engineer, specializing in distributed cloud systems and high performance search. In his spare time, he is an active contributor to the open source community.

Chapter 1

Flow-based Programming

Flow-based programming is a paradigm of programming that deals with the flow of data over one or more destinations. These destinations could in theory be any sort of structure. For this book, and the implementation of *Wind*, they are only considered buffers; fixed sized arrays controlled with pointers and indices.

This chapter will focus on an abstract perspective of flow-based programming. It will describe the fundamental concepts, such as nodes, data, and flows. It will also discuss the arrangements of how data flows can be constructed, and give some respective examples.

This chapter serves as an introduction to flow-based programming, and a preparation for the actual Wind language.

1.1 Data

In flow-based programming, data is considered any of the values or information that are computed or processed. Numbers, integers, strings, characters, lists, are all examples of data. Nearly all programming languages type the data that they contain and compute. At the raw level, data to a computer is simply sequences of 1's and 0's. Programming languages provide much more human readable forms of data. Integers or floats to hold numbers, strings to hold texts, and numerous data structures like a *list* to hold an ordered collection of data.

An important distinction in how data is typically assigned types is whether a type is mutable or immutable. An *immutable* type is one that cannot be altered or changed after it is created. In order to alter or change it, you must create a new copy of that immutable type from existing instances of the type. For example, an integer, or any number, is an immutable type. Numbers are created from operations performed on other numbers.

$$a = 1$$
$$b = a + 1$$

In the above example, a is a variable with the value of 1, while b is a variable with the value of the sum of a and 1. Neither of them can be changed. They might be able to be recreated under the same variable name, but this is not altering the existing value. Next, let's look at an example in C that deals with mutable and immutable data.

```
1 int a = 1;
2 int* b = malloc(sizeof(int));
3 *b = a;
```

Here, b is an integer allocated dynamically on the program's heap, while a is a stack allocated variable. The main difference between them is that, b is a *mutable* data value that can be changed freely after it has been created. Yet a is immutable. The last statement, in the third, line sets the value of b to a. This means b can be considered a mutable type.

However, a and b are not both integers. b is a *pointer* to an integer, specifically a block of heap memory the size of one integer. In comparison to dynamically typed languages, b can be thought of as a one element list. In flow-based programming, the same data types have both mutable and immutable characteristics. Which characteristics of data determine it's behavior is determined by the location and the *movement* of the data.

1.1.1 Data Movement

Generally, moving data in a program means the ownership of values and data are being passed around. Some languages define specific criteria for the movement of objects and memory, like C++. *Move semantics* usually refers to the process of moving dynamically allocated objects from one owner to another. Here is a simple example.

```
1 int* a = malloc(sizeof(int) * 2);
2 a[0] = 1;
3 a[1] = 2;
4 int* b = a;
```

The memory containing enough space for two integers, pointed to by a, is passed to b. After line four, both a and b point to or "hold" the same value. Accessing through either one yields the same behavior. Changing one of the integers of a changes it in b as well.

```
printf("a is %d %d\n", a[0], a[1]);
printf("b is %d %d\n", b[0], b[1]);
b[0] = 5;
printf("a is %d %d\n", a[0], a[1]);
```

Compiling this code will display the following in the terminal:

```
1 2
1 2
5 2
```