The ALB Programming Language

Alberto Navalón Lillo

April 7, 2019

Contents

Introduction

After several months thinking about it, I decided to create my own programming language. I have always wanted to do something big, and this matches the definition. By now, it is just a bunch of lines of C++ code that follows some orders of other code. But in a future time, I hope on this being a big project and used by other people to do their work.

The ALB language is an interpreted language, such as BASIC, MATLAB, JavaScript, Perl, PHP, Python, R... This has its advantages. For example, platform independence (by now, the cross-platform executing is not too advanced, so we still have some troubles with it), dynamic typing and scoping, and reflection. But, as everything, it has also its disadvantages: probably the main disadvantage would be the slower execution compared to direct native machine code.

This book is supposed to help beginners and experienced programmers to learn and understand the ALB Programming Language, and its simple structure. If you have some experience on programming, you will probably notice some similarities in ALB code with languages such as Python, C++ and Bash, which are the main three influencers of ALB. Also, I would like to highlight the main purpose of the ALB language: *make a powerful language, suitable for beginners and experts*.

ALBERTO NAVALÓN LILLO
April 6th 2019

Chapter 1

The basics

1.1 Hello World!

First of all, as in every single programming book on Earth, we will cover the classic *Hello World!* in the ALB Programming Language.

```
BEGIN
        out {
                :string "Hello World!" NEWL ;
        }
END
```

Now, the explanation of all this stuff. You will notice that the code begins with the keyword **BEGIN** and ends with the keyword **END**. Well, those are delimiters of the code that will be executed. Anything not between these keywords will not give any errors, but it will be ignored by the interpreter. Now you have to remember to write these delimiters in all of your ALB programs, and put your code between them.

Apart from the **BEGIN** and **END** delimiters, we also have an **out** structure. This structure, delimited by curly braces, Note that it is not mandatory to leave a white space between the **out** keyword and the opening brace, so it can also be:

```
out{
        :string "Hello World!" NEWL ;
}
```

The **out** structure and function is part of the standart output, or *stdout*, about which we will be talking later. Inside an **out** structure, composed of several statements, you need to identify the type of the output by typing a colon, followed by the type **not leaving any white space between them**. The types of outputs are the following:

- **string** or **str**: array of characters (texts, words...). Must be between double quotes (“”).
- **int**: short for integer. Numeric. Can hold an integer value between -2147483647 and 2147483647. Discussed Numeric Types section.
- **float**: short for floating-point number. Numeric. Can hold a decimal value between 1.17549e-038 and 3.40282e+038. Discussed in Numeric Types section.
- **char**: an only character. Must be an ASCII character and it must also be between single quotes.