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. Any-
thing 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.

1.2   Stdout

The standard output (or *stdout* for short), is part of the base of every programming language. The *stdout* shows the user (or whoever is looking at the screen of your computer) some data that ALB is processing during runtime. In ALB, the standard output is basically formed by the **out** structure/function and the **specific outs**, about which we will be talking later.

1.2.1   out

You have probably noticed the usage of **out** in the *Hello World!* example in the 1.1 section. Well, this is a widely used structure/function in ALB, as it can easily show on screen the data selected. In the example, the selected data is the "Hello World!" string, but it can be of some other types:

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 in 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.

- **newline** or **newl**: this is not actually a type of output. It makes the same function as the NEWL keyword, which can only be used in non-numeric outputs (**string/str** and **char**), as it is shown in the *Hello World!* example. Writing it in an **int** or a **float** output would be useless, but it will not imply an exception.

- **var**: ALB does not have support for variables yet. But, to use it, after writing **var**, leave a space and type a dollar symbol inmediately followed by the variable name. This would be an example of usage of the **var** output type in an **out** structure/function:

```
out {
   :var $aux ;
}
```

, which would show the value of the *aux* variable.