

UNIVERSIDADE FEDERAL DE ALAGOAS

Instituto de Computação

Especificações da Linguagem **Valcode**

Luís Antônio da Silva Nascimento, Vinicius Monteiro Pontes

Maceió, AL - 2021

Sumário

1. Introduction	2
2. Program General Structure	2
3. Set of data types and names	3
3.1 Reserved Words.	3
3.2 Identifiers.	3
3.3 Comments.	3
3.4 Integers.	3
3.5 Float.	4
3.6 Characters.	4
3.7 Strings.	4
3.8 Booleans.	4
3.9 One-dimensional array.	4
3.10 Supported Operations.	4
3.11 Default Values.	5
3.12 Coercion	5
3.13 Logical expression	5
4. Operation set.	5
4.1 Arithmetic	5
4.2 Relational	6
4.3 Logical	6
4.4 Precedence and Associativity	7
5. Instructions	7
5.1 Assignment	7
5.2 Conditional structures	8
5.2.1 If and else	8
5.3 Iteratives structures.	8
5.3.1 While	8
5.3.2 For	9
5.4 Input and Output.	9
5.4.1 Input	9
5.4.2 Print	9
5.5 Functions	9
6. Algorithms examples	10
6.1 Hello World.	10
6.2 Fibonacci Series	10
6.3 Shell Sort	11

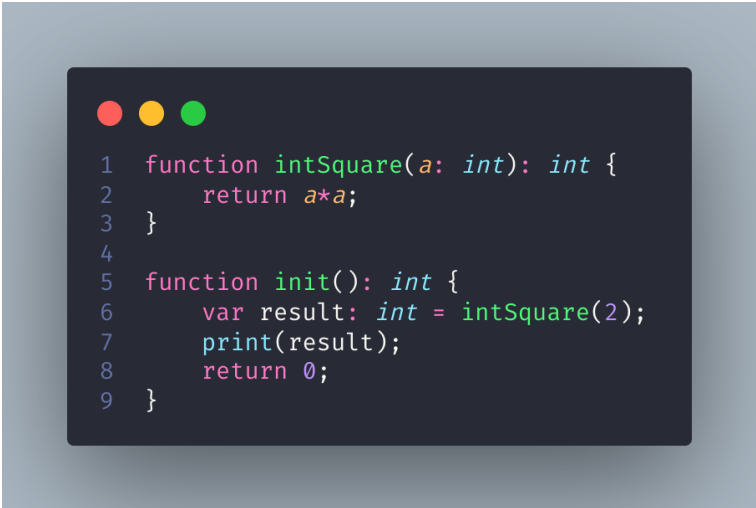
1. Introduction

The Valcode programming language is a strongly typed language and its main goal is its reliability and writability. It doesn't support type coercion and also doesn't allow function declaration inside another function.

2. Program General Structure

A program written in Valcode follows the template:

- The definition of the main function, which the compiler will consider as the start point and must be identified with the reserved word **init**.
- A block scope is contained within a pair of curly brackets **{ }**.
- Functions are defined with the reserved word **function**, its name identifier and its parameters (must specify the parameter type), which are separated by commas, contained within a pair of parenthesis, followed by its return type and its block scope. The return of a function is defined by the reserved word **return** and its value.
- Every instruction must end with a semicolon except when it has a block scope (e.g if scopes, loops scopes). //VERIFY



```
1  function intSquare(a: int): int {  
2      return a*a;  
3  }  
4  
5  function init(): int {  
6      var result: int = intSquare(2);  
7      print(result);  
8      return 0;  
9  }
```

3. Set of data types and names

3.1 Reserved Words.

1. **function**
2. **init**
3. **return**
4. **int**
5. **float**
6. **bool**
7. **char**
8. **string**
9. **true**
10. **false**
11. **if**
12. **else**
13. **while**
14. **for**
15. **void**
16. **null**
17. **and**
18. **or**
19. **not**
20. **ung**

3.2 Identifiers.

Identifiers are any non-empty chain of characters that does not contain any special symbols or spaces, except underscores (`_`), and must not begin with numbers or underscores and also are case-sensitive.

3.3 Comments.

Valcode only allows single line comments which are defined with the special character `#` at the beginning of the line.

3.4 Integers.

The reserved word **int** identifies the variables that will contain an integer (size limited to 4 bytes). A literal constant of an **int** is represented by a sequence of digits.

3.5 Float.

The reserved word **float** identifies the variables that will contain a float number (size limited to 4 bytes). A literal constant of a **float** is represented by an integer followed by a dot and a sequence of digits.

3.6 Characters.

The reserved word **char** identifies the variables that will contain a character (size limited to 1 byte). A literal constant of a **char** is represented by an ASCII character.

3.7 Strings.

The reserved word **string** identifies the variables that will contain a string. A literal constant of a **string** is represented by a set of ASCII characters.

3.8 Booleans.

The reserved word **bool** identifies the variables that will contain a boolean value. The literal constants of a boolean are only: **true**, **false**.

3.9 One-dimensional array.

An array is defined by adding brackets to the type of the variable (size is dynamic).

3.10 Supported Operations.

The supported operations by each type in Valcode are detailed below.

Type	Operation
int	assignment, arithmetic and relational
float	assignment, arithmetic* and relational
string	assignment, relational and concatenation
char	assignment, relational

bool	assignment, logic
------	-------------------

* **float** does not support module operation between two operands. //VERIFY

3.11 Default Values.

Tipo	Valores Default
int	0
float	0.0
string	null
char	null
bool	false

3.12 Coercion

Valcode programming language does not support type coercions.

3.13 Logical expression

Valcode logical expressions can be composed by:

1. A **bool** constant literal or variable
2. A **logical** expression
3. A **relational** expression

4. Operation set.

4.1 Arithmetic

Operators	Operations
+	Returns the sum between two operands or the concatenation of strings
-	Returns the difference between two operands

*	Returns the product between two operands
/	Returns the quotient between two operands
//	Returns the floor division between two operands
%	Returns the modulus between two operands
ung	Returns the unary negation of the operand

4.2 Relational

Operators	Operation
==	Returns true if the two operands are equals, otherwise false.
!=	Returns true if the two operands are different, otherwise false.
>=	Returns true if the left operand is greater or equal than the right operand, otherwise false.
<=	Returns true if the left operand is lesser or equal than the right operand, otherwise false.
>	Returns true if the left operand is greater than the right operand, otherwise false.
<	Returns true if the left operand is lesser than the right operand, otherwise false.

4.3 Logical

Operadores	Operação
not	Returns the negated operand

and	Returns true if the left and right statements are true, otherwise false.
or	Returns true if the left or the right, or both operands are true, otherwise false.

4.4 Precedence and Associativity

The following table shows the precedence order in descending sorting (first line has the highest precedence).

Operators	Operation	Associativity
-	Unary negation	<i>Right → Left</i>
* /	Multiply and Division	<i>Left → Right</i>
%	Modulus	<i>Left → Right</i>
+ -	Addition and Subtraction	<i>Left → Right</i>
not	Negation	<i>Right → Left</i>
< > <= >=	Comparatives	<i>Left → Right</i>
== !=	Equality	<i>Left → Right</i>
and or	Logical	<i>Left → Right</i>

5. Instructions

In Valcode every one line instruction must end with the symbol “;”. Scope blocks (functions, conditionals, loops, etc) must start with “{“ and end with “}”.

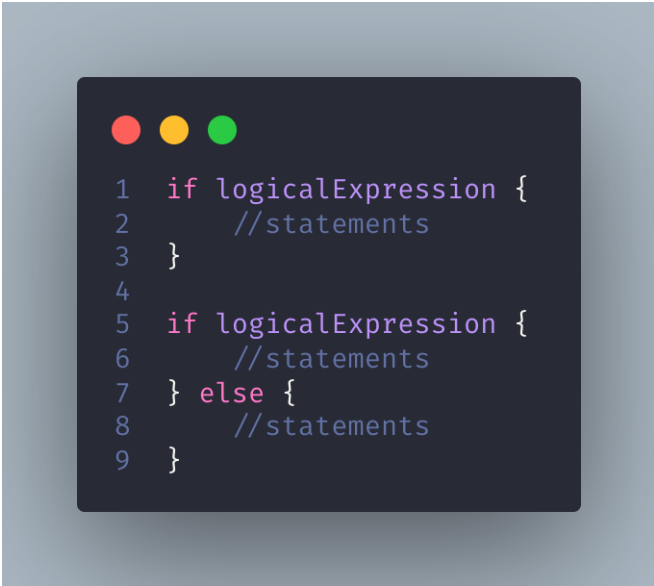
5.1 Assignment

The assignment of a variable in Valcode is made with the symbol “=”, with the left side of the instruction containing its type and identifier, and the right side the value or expression assigned to it. Both sides need to have the same type, since the language doesn’t allow for coercion. With reliability in mind Valcode sees assignment as an instruction and not as an operation.

5.2 Conditional structures

5.2.1 If and else

The conditional statement **if** is followed by a logical expression which it will be evaluated to a **bool**, and a scope block with the code that will be executed if the evaluation is true. If the expression evaluates to false, the code within the following **else**'s scope block will be executed, if there is one.


A code editor window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains two code snippets. The first snippet is an 'if' statement with line numbers 1 to 3. The second snippet is an 'if-else' statement with line numbers 5 to 9.

```
1  if logicalExpression {  
2      //statements  
3  }  
4  
5  if logicalExpression {  
6      //statements  
7  } else {  
8      //statements  
9  }
```

5.3 Iteratives structures.

5.3.1 While

Valcode uses a **while** statement to implement an iterative structure with logical control. The loop will remain being executed as long as its conditional remains truthful.

A code editor window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains a 'while' loop statement with line numbers 1 to 3.

```
1  while logicalExpression {  
2      //statements  
3  }
```

5.3.2 For

In the **for** statement, the number of iterations is defined by the programmer and controlled by a counter. It is structured in the following way:

A code editor window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains the following code:

```
1  for start, step, stop {  
2      //statements  
3  }
```

Where start, step, stop are integers. The initial value of start is incremented by step as long as the variable is less than stop.

5.4 Input and Output.

Valcode implements input and output functions using the reserved words **input** and **print**.

5.4.1 Input

In the **input** function, the type of the input fetched from the user must be provided as a string in the following manner: `input("int")` or `input("string")` //VERIFY

5.4.2 Print

In the print function, it receives as parameters one or more values (constant literals or variables) to send to the output.

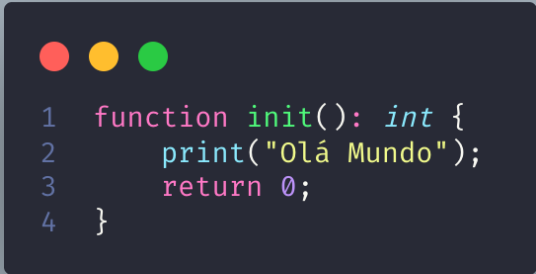
5.5 Functions

Functions are defined with the reserved word **function**, followed by its name identifier and its parameters inside parenthesis (the parameters types must be specified), separated by commas, followed by its return type and its block scope. The return of a function is defined by the reserved word **return** and its value. A function is called by referencing its identifier, followed by the values or variables that are to be passed as its parameters.

If the return value of a function is not specified, it will return the default value of its return type. Valcode does not allow function overloading.


6. Algorithms examples

6.1 Hello World.



```
1 function init(): int {
2     print("Olá Mundo");
3     return 0;
4 }
```

6.2 Fibonacci Series



```
1 function fibonacci(n: int, fibStorage: int[]): void {
2     var i: int = 2;
3     for i, 1, n + 1 {
4         fibStorage[i] = fibStorage[i-1] + fibStorage[i-2];
5     }
6 }
7
8 function init(): int {
9     var fibStorage: int[] = [0, 1];
10    var n: int = input("int");
11    var i: int = 0;
12    fibonacci(n, fibStorage);
13    for i, 1, n + 1 {
14        print(fibStorage[i]);
15    }
16    return 1;
17 }
```

6.3 Shell Sort

```
1 function shellShort(array: int[], n: int): void {
2     var gap: int = n // 2;
3
4     while gap > 0 {
5         var i: int = gap;
6
7         for i, 1, n {
8             var temp: int = array[i];
9             var j: int = i;
10
11             while j ≥ gap and arr[j-gap] > temp {
12                 array[j] = array[j-gap];
13                 j = j - gap;
14             }
15             array[j] = temp;
16         }
17
18         gap = gap // 2;
19     }
20 }
21
22 function init(): int {
23     var array: int[];
24     var n: int = input("int");
25     var i: int = 0;
26
27     for i, 1, n {
28         array[i] = input("int");
29     }
30
31     i = 0;
32     for i, 1, n {
33         print(array[i]);
34     }
35
36     shellShort(array, n);
37
38     i = 0;
39     for i, 1, n {
40         print(array[i] + " ");
41     }
42     return 0;
43 }
```