

UNIVERSIDADE FEDERAL DE ALAGOAS

Instituto de Computação

Especificações da Linguagem **Valcode**

Luís Antônio da Silva Nascimento, Vinicius Monteiro Pontes

Maceió, AL - 2021

Sumário

1. Introdução	2
2. Estrutura Geral de um Programa	2
3. Conjunto de tipos de dados e nomes	3
3.1 Palavras reservadas	3
3.2 Identificadores	3
3.3 Comentários	3
3.4 Inteiros	3
3.5 Float	4
3.6 Caracteres	4
3.7 Cadeia de caracteres	4
3.8 Booleanos	4
3.9 Arranjo unidimensional	4
3.10 Operações suportadas	4
3.12 Coerção	5
3.13 Expressões Lógicas	5
4. Conjunto de Operações	5
4.1 Aritméticas	5
4.2 Relacionais	6
4.3 Lógicas	7
4.4 Precedência e Associatividade	7
5. Instruções	8
5.1 Atribuição	8
5.2 Estruturas Condicionais	8
5.2.1 If e else	8
5.3 Estruturas de Iteração	9
5.3.1 While	9
5.3.2 For	9
5.4 Input and Output.	9
5.4.1 Input	9
5.4.2 Print	10
6. Exemplos de Algoritmos	10
6.1 Alô mundo	10
6.2 Série de Fibonacci	11
6.3 Shell Sort	12

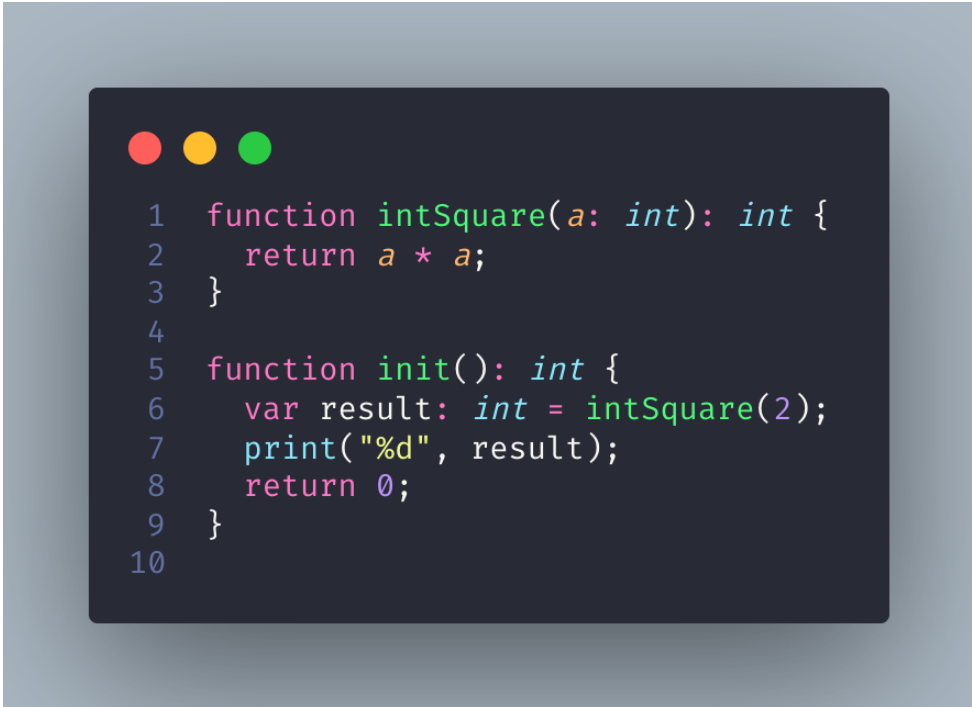
1. Introdução

A linguagem de programação Valcode é uma linguagem fortemente tipada e seu objetivo principal é sua confiabilidade e escritabilidade. Ela não suporta coerção de tipos ou declaração de funções dentro de outras funções.

2. Estrutura Geral de um Programa

Um programa escrito em Valcode segue o seguinte padrão:

- A definição da função principal, que o compilador irá considerar como o ponto inicial de execução e que deve ser identificada pela palavra reservada **init**.
- A abertura de um bloco de escopo é delimitada por uma chave aberta '{' e seu fechamento por uma chave fechada '}'.
- Funções são definidas com a palavra reservada **function**, seu identificador e seus parâmetros (o tipo de cada parâmetro deve ser especificado), que são separados por vírgulas e contidos dentro de uma abertura e fechamento de parênteses. Em seguida deve ser especificado seu tipo de retorno e seu escopo de bloco. O retorno de uma função é definido pela palavra reservada **return** e seu valor.
- Toda instrução deve acabar com um ponto e vírgula, exceto quando ela possuir um escopo de bloco (como escopo de if, escopo de loop, etc).



```
1  function intSquare(a: int): int {  
2      return a * a;  
3  }  
4  
5  function init(): int {  
6      var result: int = intSquare(2);  
7      print("%d", result);  
8      return 0;  
9  }  
10
```

3. Conjunto de tipos de dados e nomes

3.1 Palavras reservadas

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**
21. **ups**

3.2 Identificadores

Identificadores são quaisquer cadeias de caracteres não-vazias que não contêm símbolos especiais (exceto *underscore* '`_`') ou espaços, e não devem iniciar com números ou *underscore*, e são *case-sensitive*.

3.3 Comentários

Valcode só permite comentários de uma linha, que são definidos com o caractere especial '`#`' no início da linha.

3.4 Inteiros

A palavra reservada **int** identifica as variáveis que vão conter um inteiro (tamanho limitado a 4 bytes). A constante literal de um **int** é representada por uma sequência de dígitos.

3.5 Float

A palavra reservada **float** identifica as variáveis que vão conter um número de ponto flutuante (tamanho limitado a 4 bytes). A constante literal de um float é representada por um inteiro seguido por um ponto e uma sequência de dígitos.

3.6 Caracteres

A palavra reservada **char** identifica as variáveis que vão conter um caractere (tamanho limitado a 1 byte). A constante literal de um char é representada por um caractere ASCII.

3.7 Cadeia de caracteres

A palavra reservada **string** identifica as variáveis que vão conter uma cadeia de caracteres (tamanho dinâmico). A constante literal de uma **string** é representada por uma cadeia de caracteres ASCII.

3.8 Booleanos

A palavra reservada **bool** identifica as variáveis que vão conter um booleano (tamanho limitado a 1 byte). A constante literal de um bool são **true** ou **false**.

3.9 Arranjo unidimensional

Um arranjo unidimensional é definido ao adicionar colchetes ao tipo da variável (tamanho é dinâmico). Arranjos unidimensionais só podem armazenar variáveis do mesmo tipo.

3.10 Operações suportadas

As operações suportadas por cada tipo em Valcode são detalhadas abaixo.

Tipo	Operador(es)
------	--------------

int	atribuição, aritméticos e relacionais
float	atribuição, aritméticos* e relacionais
string	atribuição, relacionais e concatenação
char	atribuição, relacionais
bool	atribuição, igualdade, diferença, lógicos

* **floats** não são suportados na operação de módulo

3.11 Valores padrão

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

3.12 Coerção

A linguagem Valcode não suporta coerção de tipos.

3.13 Expressões Lógicas

As expressões lógicas de Valcode podem ser compostas por:

1. Uma constante literal ou variável **bool**
2. Uma expressão com operações lógicas
3. Uma expressão com operações relacionais

4. Conjunto de Operações

4.1 Aritméticas

Operador	Operação
+	Retorna a soma entre dois operandos ou a concatenação (sendo concatenação de um int, float, bool com uma string ou char, resultando em uma string)
-	Retorna a diferença entre dois operandos
*	Retorna o produto entre dois operandos
/	Retorna o quociente entre dois operandos
//	Retorna a divisão inteira entre dois operandos
%	Retorna o módulo entre dois operandos
ung	Retorna o unário negativo de um operando
ups	Retorna o unário positivo de um operando

4.2 Relacionais

Operadores	Operações
==	Retorna true se os dois operandos forem iguais, caso contrário false .
!=	Retorna true se os dois operandos forem falsos, caso contrário false .
>=	Retorna true se o operando da esquerda é maior ou igual ao operador da direita, caso contrário false .
<=	Retorna true se o operando da esquerda é menor ou igual ao operando

	da direita, caso contrário false .
>	Retorna true se o operando da esquerda é maior que o operando da direita, caso contrário false .
<	Retorna true se o operando da esquerda é menor que o operando da direita, caso contrário false .

4.3 Lógicas

Operadores	Operação
not	Retorna o operando negado.
and	Retorna true se ambas expressões da esquerda e direita forem verdadeiras, caso contrário false .
or	Retorna true se a expressão da esquerda ou da direita, ou ambas forem verdadeiras. Caso contrário false .

4.4 Precedência e Associatividade

A tabela seguinte mostra as precedências em ordem decrescente.

Operadores	Operação	Associatividade
- +	Unário negativo e unário positivo	<i>Direita → Esquerda</i>
* /	Multiplicação e divisão	<i>Esquerda → Direita</i>
%	Resto	<i>Esquerda → Direita</i>
+ -	Adição e subtração	<i>Esquerda → Direita</i>
not	Negação	<i>Direita → Esquerda</i>
< > <= >=	Comparativos	<i>Esquerda → Direita</i>

== ?=	Igualdade ou Desigualdade	<i>Esquerda → Direita</i>
and or	Lógicos	<i>Esquerda → Direita</i>

5. Instruções

Em Valcode toda instrução de uma linha deve acabar com o símbolo “;”. Escopos de bloco (funções, condicionais, loops, etc) devem começar com “{” e acabar com “}”.

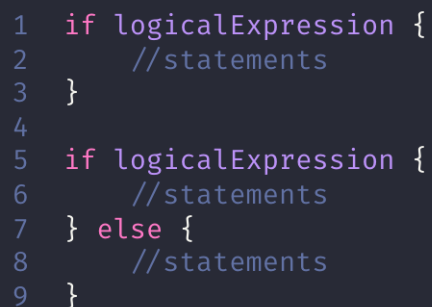
5.1 Atribuição

A atribuição de uma variável em Valcode é feita com o símbolo “=”, com o lado esquerdo da instrução contendo o identificador e o tipo da variável (caso seja a declaração da variável) e o lado direito contendo o valor ou expressão atribuída à variável. Ambos os lados precisam ter o mesmo tipo, já que a linguagem não permite coerção. Pensando em confiabilidade, Valcode enxerga atribuição como uma instrução e não uma operação.

5.2 Estruturas Condicionais

5.2.1 If e else

A instrução condicional **if** é seguida por uma expressão lógica que resultará em um **bool**, e um escopo de bloco com o código que será executado caso a avaliação deste bool resulte em true. Se a expressão seja avaliada como false, o código dentro do escopo de bloco do else seguinte será executado, se ele existir.




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

5.3 Estruturas de Iteração

5.3.1 While

Valcode utiliza a palavra reservada **while** para implementar uma estrutura de iteração com controle lógico. O laço permanecerá sendo executado enquanto a sua condição permanecer verdadeira.



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

5.3.2 For

Na instrução **for**, o número de iterações é definido pelo programador e controlado por um contador. Ela é estruturada da seguinte maneira:



Onde start, step, stop são inteiros. O valor inicial de start é incrementado por step enquanto a variável for menor que stop.

5.4 Input and Output.

Valcode implementa as funções de entrada e saída com as palavras reservadas **input** e **output**.

5.4.1 Input

Na função de **input**, o tipo de entrada que será buscado do usuário deve ser passado como primeiro parâmetro na função (sendo separados por vírgula caso seja mais de uma variável), como segundo parâmetro, devem ser passadas as variáveis que os valores devem ser atribuídos respectivamente.

5.4.2 Print

Na função de print, seu primeiro parâmetro é a string a ser imprimida, que pode conter a formatação de uma ou mais variáveis, que serão listadas respectivamente como os próximos parâmetros.

5.5 Funções


Funções são definidas com a palavra reservada **function**, seguida por seu identificador e parâmetros dentro de parênteses (seus tipos devem ser especificados), separados por vírgulas; o modelo semântico de passagem de parâmetros é o modo de entrada e saída, sendo a sua implementação feita por meio da passagem por valor e referência (sendo a referência sendo usada apenas em arranjos unidimensionais). Em seguida deve ser especificado seu tipo de retorno e seu escopo de bloco.

O retorno de uma função é definido pela palavra reservada **return** e seu valor. Uma função é chamada ao referenciar seu identificador, seguido pelos valores ou variáveis que serão passados como seus parâmetros dentro de um par de parênteses. Se o valor de retorno de uma função não for especificado, ela irá

retornar o valor padrão de seu tipo de retorno. Valcode não permite *overloading* de funções, nem definição de funções dentro de outras funções.


6. Exemplos de Algoritmos

6.1 Alô mundo



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

6.2 Série de Fibonacci



```
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 fib_Storage: int[] = [0, 1];
10    var n: int;
11    input("int", n);
12    var i: int = 0;
13    fibonacci(n, fibStorage);
14    while i < n + 1 {
15        print("%d", fibStorage[i]);
16    }
17    return 0;
18 }
```

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;
25     input("int", n);
26     var i: int = 0;
27
28     for i, 1, n {
29         input("int", array[i]);
30     }
31
32     i = 0;
33     for i, 1, n {
34         print("%d", array[i]);
35     }
36
37     shellShort(array, n);
38
39     i = 0;
40     for i, 1, n {
41         print("%d ", array[i]);
42     }
43     return 0;
44 }
```