

UNIVERSIDADE FEDERAL DE ALAGOAS

Instituto de Computação

Compiladores

Especificações da Linguagem **Valcode**

Luís Antônio da Silva Nascimento, Vinicius Monteiro Pontes

Maceió, AL - 2020.1

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

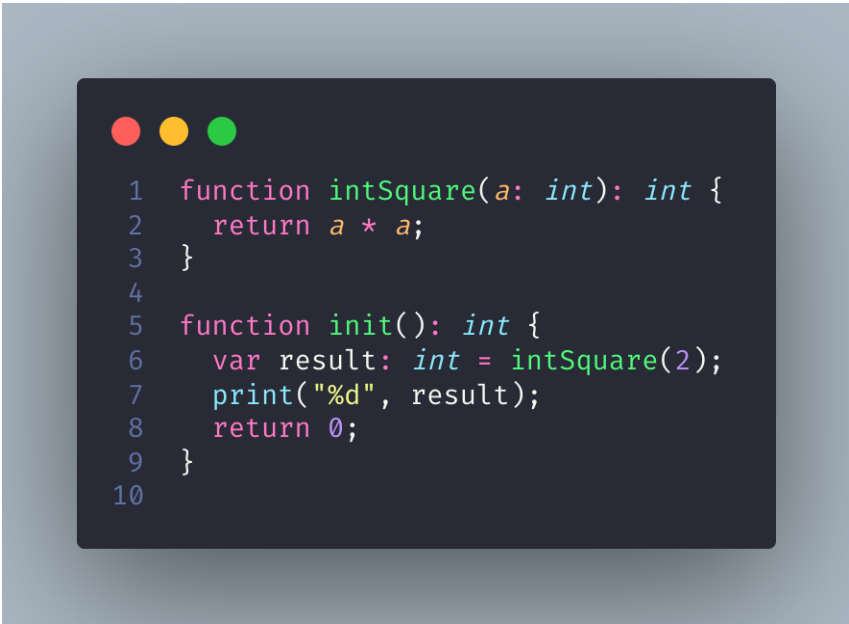
1. Introdução

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 função **init** deve ser a última função declarada, pois Valcode não faz *hoisting* para usar funções que não foram declaradas antes de sua chamada.
- A abertura de um bloco de escopo é delimitada por uma abre chaves '{' e seu fechamento por uma fecha chaves '}'.
- 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. 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 (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 que começam com uma letra e contém apenas caracteres alfanuméricos (símbolos não inclusos) ou *underscore*.

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 número 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 e deve estar contido entre " (apóstrofes) .

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 e deve estar contido entre "" (aspas).

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 pode assumir os valores **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, concatenação
float	atribuição, aritméticos* e relacionais, concatenação
string	atribuição, relacionais e concatenação
char	atribuição, relacionais e concatenação
bool	atribuição, igualdade, diferença, lógicos, concatenação

* **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 um int caso ambos operandos sejam int , e retorna um float caso um dos operandos sejam float .
%	Retorna o resto entre dois operandos
ung	Retorna o inverso de um operando
ups	Retorna a identidade do operando.

4.2 Relacionais

Operadores relacionais **devem** ser formatados corretamente com espaços entre seus operandos.

Operadores	Operações
==	Retorna true se os dois operandos forem iguais, caso contrário false .
!=	Retorna true se os dois operandos forem diferentes, caso contrário false .
>=	Retorna true se o operando da esquerda é maior ou igual ao operando 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
ung ups	Unário negativo e unário positivo	<i>Direita → Esquerda</i>
* /	Multiplicação, 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	Esquerda → Direita
--------	---------	--------------------

5. Instruções

Em Valcode toda instrução de uma linha deve acabar com o símbolo “;”. Escopos (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.

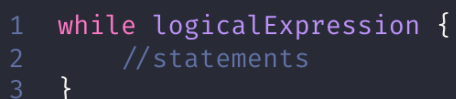
```

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

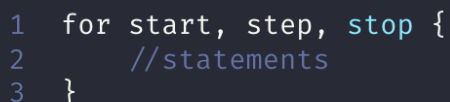
Valcode utiliza a palavra reservada **while** para implementar uma estrutura de iteração com controle lógico. O laço permanecerá sendo executado enquanto a sua condição permanecer verdadeira.



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

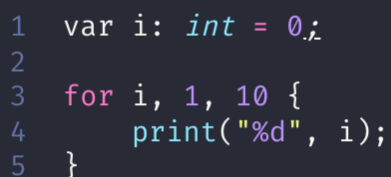
5.3.2 For

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



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

Onde start, step, stop são variáveis ou constantes literais inteiras. O valor inicial de start será incrementado pelo valor de step ao fim de cada iteração, enquanto o valor de start for menor que stop.



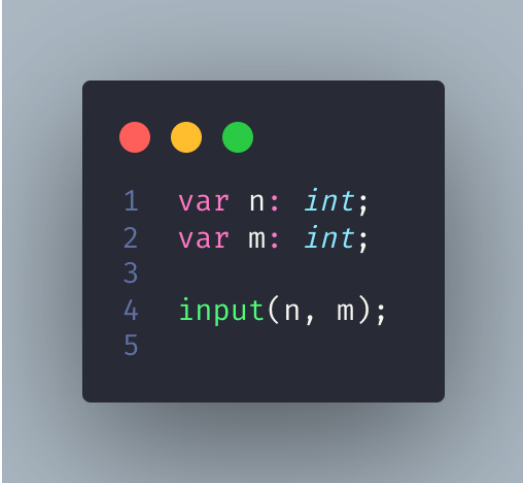
```
1 var i: int = 0;  
2  
3 for i, 1, 10 {  
4     print("%d", i);  
5 }
```

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**, como parâmetros, devem ser passadas as variáveis que os valores serão atribuídos.

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

```
1 var n: int;  
2 var m: int;  
3  
4 input(n, m);  
5
```

```
1  var n: int;  
2  var m: int;  
3  
4  input(n, m);  
5
```

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.

A formatação opcional se dá pelo seguinte formato:

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

```
1 print("%d", a);
```

```
1  print("%d", a);
```

Onde o dígito após o escape “%” diz a quantidade de números que serão preenchidos à esquerda com zeros (caso necessário) e o “d” indica que é um **int**.

A formatação de um float tem por default duas casas decimais:



Onde o ponto após o escape “%” indica a quantidade de números que serão formatados após o ponto do **float**.

Quebra de linhas são utilizadas por padrão ao final da linha a ser imprimida pela função de print.

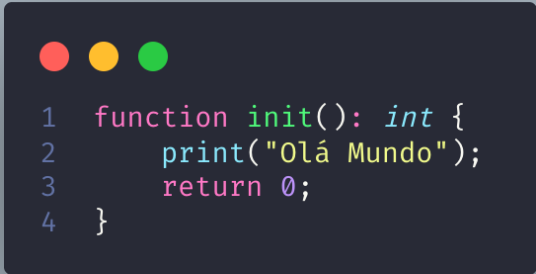
5.5 Funções

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.

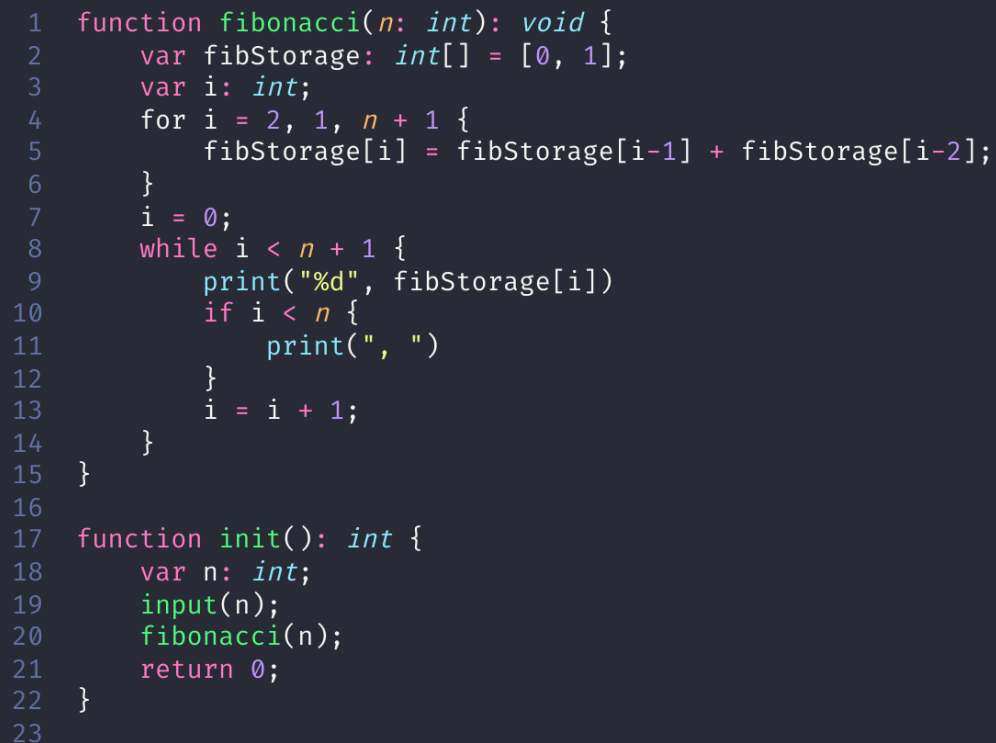
6. Exemplos de Algoritmos

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): void {
2     var fibStorage: int[] = [0, 1];
3     var i: int;
4     for i = 2, 1, n + 1 {
5         fibStorage[i] = fibStorage[i-1] + fibStorage[i-2];
6     }
7     i = 0;
8     while i < n + 1 {
9         print("%d", fibStorage[i])
10        if i < n {
11            print(", ")
12        }
13        i = i + 1;
14    }
15 }
16
17 function init(): int {
18     var n: int;
19     input(n);
20     fibonacci(n);
21     return 0;
22 }
23
```

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