

Universidade Federal de Alagoas
Instituto de Computação
Curso de Ciência da Computação

Jvlal
Especificação da Linguagem

João Victor Ribeiro Ferro
Lucas Albuquerque Lisboa

Maceió
2021

Introdução	3
Estrutura geral do programa	3
Ponto de início de execução	3
Definições de procedimentos / funções	3
Definições de instruções	4
Conjunto de tipo de dados e nomes	4
Identificador	4
Palavras reservadas	4
Comentário	5
Definição de variáveis	5
Definição de constantes	5
Tipos de Dados	5
Booleano	5
Inteiro	5
Ponto Flutuante	6
Caractere	6
Cadeia de Caracteres	6
Arranjos Unidimensionais	6
Coerção	7
Concatenação de cadeia de caracteres	7
Valores padrão	7
Conjunto de Operadores	7
Aritméticos	7
Relacionais	8
Lógicos	8
Concatenação	8
Atribuição	8
Operações suportadas	8
Precedência e Associatividade	9
Instruções	9
Atribuição	9
Estrutura condicional	9
Estrutura iterativa com controle lógico	10
Estrutura iterativa controlada por contador	10
Constantes literais e suas expressões	11
Desvios Incondicionais	11
Entrada e Saída	11
ToString()	12
Programas Exemplos	12

Introdução

A linguagem de programação Jvlal foi criada com inspiração nas linguagens C e Python, tendo como meta de utilização o ensino instrutivo da programação.

Jvlal não é orientada a objeto e é case-sensitive (diferencia letras maiúsculas e minúsculas). Já na legibilidade, Jvlal não faz coerção (não admite conversões implícitas de tipo) e possui palavras reservadas, às quais estão em inglês e foram escolhidas para que fique o mais claro possível para mostrar o que o código está fazendo.

Estrutura geral do programa

Ponto de início de execução

O ponto inicial de execução do programa será identificado por uma construção específica da linguagem, sempre será a função `main()` com o tipo de retorno obrigatório `int`.

Ex.:

```
fun int main(){  
    .  
    .  
    .  
    return 0;  
}
```

Definições de procedimentos / funções

Estas poderão existir se forem declaradas fora do corpo de uma outra função ou procedimento. Estas podem ser acessadas se tiverem sua implementação ou assinatura declaradas antes da função chamada. Jvlal não aceita passagem de subprogramas como parâmetro.

A declaração de de função é sempre iniciado pela palavra reservada `fun`, seguido pelo seu tipo de retorno obrigatório (**`int`, `float`, `bool`, `string`, `char`**), seu identificador único e uma lista de parâmetros (os parâmetros acompanharão de seu tipo e seu identificador) delimitada por abre e fecha parênteses (`)`). A abertura e fechamento do bloco é feita por abre e fecha chaves { }.

A declaração de um procedimento seguirá o mesmo padrão, com exceção de que terá o tipo de retorno e será sempre iniciada, não terá o tipo de retorno e será sempre iniciada com a palavra reservada `proc`.

A passagem de parâmetro para os procedimentos e funções se dá por meio do modo de entrada e saída, para todos os tipos de estrutura.

Ex.:

```
fun tipoDoRetorno id ( listaDeParametros){  
    ...  
    return x;  
}  
  
proc id (listaDeParametros) {  
    ...  
}
```

Definições de instruções

As instruções podem ser declaradas somente dentro do bloco de funções ou procedimentos, não podem ser escritas instruções do corpo dos parâmetros de uma função, tal como estruturas de iteração e estruturas de seleção.

Conjunto de tipo de dados e nomes

Identificador

Possuirão sensibilidade à caixa, limite de tamanho de 31 caracteres e seu formato será definido a partir da seguinte expressão regular:

(*'letter'* | (*'_'*)(*'letter'*)) (*'digit'* | *'letter'* | (*'_'*)(*'digit'*) | (*'_'*)(*'letter'*))*

Exemplos de nomes aceitos pela linguagem: *_var1*, *var2*, *var* , *_a*;

Exemplos de nomes não aceitos pela linguagem: *.abc*, *ab@c*, *1ac*.

Palavras reservadas

As palavras reservadas são sempre escritas em inglês, buscando objetividade e simplicidade são listadas a seguir:

const, char, int, float, string, bool, proc, read, print, if, else, false, true, for, while, return, lenght, toString.

Comentário

Os comentários são indicados por *"//"*. Desta forma, o que se escrever na linha após será descartado. Os comentários podem ser feitos apenas por linha.

Definição de variáveis

Jv1al possui escopo global, logo as variáveis podem ser definidas fora do escopo e acessadas globalmente ou dentro do escopo como variáveis locais. São declaradas iniciando com o **tipoDaVariável (int, float, boll, string, char)** e seguidas pelo seu **identificador único**. Caso haja múltiplas variáveis em uma declaração de tipo único, as variáveis serão separadas por **vírgula**. A declaração termina com um **ponto e vírgula**.

Ex.:

```
int a = 2;  
float b = 3.4;  
char c;  
bool d = true;  
string e = "hello";
```

Definição de constantes

Constantes possuem o mesmo padrão de declaração e definição de variáveis, com exceção que iniciam sempre com a palavra reservada **const** e que a atribuição de valor é obrigatório na declaração .

Ex.:

```
const int i = 0;  
const bool d = true;
```

Tipos de Dados

Todos os tipos e estruturas possuirão compatibilidade por nome.

Booleano

É a identificação da variável como sendo do tipo booleana, identificando pela palavra reservada **bool**. Possui dois valores possíveis: **true,false**

Ex.: bool a;

Inteiro

É a identificação da variável como sendo do tipo inteiro de 32 bits, identificado pela palavra reservada **int**. Seus literais são expressos como uma sequência de números decimais.

Ex.: int b;

Ponto Flutuante

É a identificação da variável como sendo do tipo ponto flutuante de 64 bits, identificado pela palavra reservada **float**. Seus literais são expressos como uma sequência de números decimais, seguido de um ponto e demais dígitos.

Ex.: float c;

Caractere

É a identificação da variável como sendo do tipo caractere 8 bits, identificado pela palavra reservada **char**. Guarda um código referente ao seu símbolo da tabela ASCII. A constante literal do caractere é delimitada por **apóstrofo**.

Ex.: char d;

Cadeia de Caracteres

É a identificação da variável como sendo do tipo cadeia de caracteres, identificado pela palavra reservada **string**. Seus literais são expressos como um conjunto, mínimos de 0 caracteres e de tamanho máximo ilimitado. A constante literal é delimitada por aspas.

Ex: string e;

Arranjos Unidimensionais

Os arranjos unidimensionais são compostos pelos tipos determinados acima, não admitindo cadeia de caracteres. Sua declaração iniciará como o tipo seguido do identificador único e, delimitado por abre e fecha parênteses (), o tamanho do arranjo. Caso aconteça um arranjo menor que o tamanho declarado, os espaços serão ocupados com o valor Null e quando passar do limite pré-definido é dado o resultado como zero.

Há uma função que dirá o tamanho do arranjo, a função retorna um inteiro.

Ex.: int s = length(arr)

Ex.:

int arr(9);

float arr(1024);

bool arr(2);

Coerção

Jv1al é estaticamente tipada, não aceitando coerção entre variáveis de tipo diferentes. Dessa forma, todas as verificações de compatibilidade de tipo serão feitas estaticamente.

Concatenação de string

É constituído pelo sinal “#”, na qual suporta os seguintes tipos de dados como string, character, int, float, bool, a sua concatenação gera uma cadeia de caracteres (string). Com isso para realizar concatenação com os outros tipos é necessário utilizar o toString()* com o intuito de converter para **string**.

Depois da conversão de tipos explícita do valor por meio do **toString()**, é utilizado o “#” para concatenar as string atribuindo a uma nova string, a string receberá o resultado da concatenação tem que ser criada antes de ser utilizada, para manter os padrões de criação de variáveis da linguagem de java.

Ex.:

```
fun int main (){  
    int b = 100;  
    string c;  
    c = “dijkstra”;  
    string new_string;  
    toString(b);  
    new_string = b#c;  
  
    return 0;  
}
```

Valores padrão

Tipo	Valor de inicialização
int	0
float	0.0
char	‘ ’(character vazio)
string	“ ”(string vazia)
bool	false

Conjunto de Operadores

Aritméticos

- **Adição:** + ;
- **Subtração:** - ;
- **Multiplicação:** * ;

- **Divisão:** / ;
- **Resto:** %;
- **Unitário Positivo:** ++ ;
- **Unitário Negativo:** -- ;

Relacionais

- **Igualdade:** == ;
- **Desigualdade:** != ;
- **Maior que:** > ;
- **Maior igual:** >= ;
- **Menor que:** < ;
- **Menor igual:** <= ;

Lógicos

- **Negação unária:** ! ;
- **Conjunção:** &;
- **Disjunção:** '|';

Concatenação

- O operador de concatenação '#', pode ser aplicado a string e caracteres, resultando sempre em uma string;

Atribuição

- O operador de atribuição '=';

Operações suportadas

Operador	Tipos que realizam a operação
'!'	bool
'^' '*' '/' '+' '-' '++'	int, float
'<' '<=' '>' '>='	int, float, char
'==' '!=' '='	int, float, bool, char string
'&' ' '	bool
'#'	string,char

Precedência e Associatividade

Operadores	Associatividade
'!' (Not)	Direita para esquerda
'--' (unário negativo) ; '++' (unário positivo)	Direita para esquerda
'*' (multiplicação); '/' (divisão) ; '%' (resto)	Esquerda para direita
'+' (adição) ; '-' (subtração)	Esquerda para direita
'<' ; '>' ; '<=' ; '>='	Não associativo
'==' ; '!='	Não associativo
'&'	Esquerda para direita
' '	Esquerda para direita
'#' (concatenação)	Esquerda para direita
'=' (atribuição)	Direita para esquerda

Instruções

Uma linha de instrução acaba com o ";" no final. O bloco de instruções tem um escopo que inicia-se com abre chaves e termina com fecha chaves.

Atribuição

É definido pelo símbolo '=', sendo o lado esquerdo o identificador único e o lado direito o valor ou expressão. Todos os operandos devem possuir o mesmo tipo de variável alvo.

Na atribuição de um *char*, o caracter a ser atribuído deve vir entre aspas simples ('). Na atribuição de uma *string*, o texto a ser atribuído deve vir entre aspas duplas ("). Na atribuição de tipos numéricos (int e float), não utilizam aspas (caso sejam utilizadas, será acusado erro)

Estrutura condicional

Na estrutura condicional do “if” está relacionada a uma expressão lógica ou a variável booleana dentro de parênteses e como é um bloco de instruções seu escopo será definido por chaves.

O algoritmo irá executar o código contido no “if”. Caso não aceite a expressão lógica entrará no “else”.

Um “else” não pode, em nenhum caso, ser inserido sem estar associado a um “if” colocado anteriormente. O uso do “else” é opcional.

É possível inserir uma série de “if”s em sequência, além de “if” dentro de outro “if”.

Ex.:

```
if (expressão Lógica) {  
    .  
    .  
    .  
}else {  
    .  
    .  
    .  
}
```

Estrutura iterativa com controle lógico

O loop ocorrerá quando a condição for verdadeira, finalizando quando não satisfazer mais a condicional (expressão lógica) e como é um bloco de instrução definido por chaves.

Ex.:

```
while ( expressaoLogica) {  
    .  
    .  
    .  
}
```

Estrutura iterativa controlada por contador

No loop os três valores (a,b,c) representam expressões aritméticas. O valor de a será o valor inicial do contador, b será o valor final e c será o valor de incremento. Esses três valores devem ser inteiros.

O contador “i” deve ser declarado dentro dos parênteses e deve ser um inteiro, a variável só é reconhecida no escopo da função **for**, além de ter o abre e fecha chaves depois da declaração dos parênteses, delimitando o bloco.

Dentro do **for** o valor inicial da repetição não pode ser alterado, já que tem o passo que faz essa mudança.

Ex.:

```
for (int i : a,b,c) {  
    .  
    .  
    .  
}
```

Constantes literais e suas expressões

As expressões regulares das constantes literais são denotadas da seguintes maneira:

1. Constante de inteiro: ((‘digit’)+)
2. Constante de float: ((‘digit’)+) (‘\.’) ((‘digit’)+)
3. Constante de char: (‘\ ’) (‘letter’ | ‘symbol’ | ‘digit’) (‘\’)
4. Constante de bool (‘true’ | ‘false’)
5. Constantes de String (‘\’’) ((‘letter’ | ‘symbol’ | ‘digit’)*) (‘\’ ’)

Desvios Incondicionais

Jv1al não aceita nenhum tipo de desvio incondicional.

Entrada e Saída

- Entrada é realizada por meio da função read()
 - A função read (identificador) atribuíra o valor lido da variável do identificador. Como o identificador já foi declarado, sabemos qual será o seu tipo. Caso tente ser atribuído um valor a uma posição inexistente do array, será acusado erro.
- Saída: É realizada por meio da função print()

- A função print (identificador): irá imprimir o valor na tela. O identificador nesta função também pode ser visto como uma string não instanciada, ou seja, apenas com abertura e fechamento de aspas pode-se escrever algo (entre as aspas) e isto será impresso na tela, não necessariamente tendo que ser uma variável.
- A saída poderá ser formatada, os identificadores dos tipos de formatação são: %d inteiro, %c caractere, %f float, %s string, %b bool. Caso a saída encontra formatada, no código deverá ser especificado quais são as variáveis específicas a cada um dos identificadores de formatação;

Ex .:

```
print ("%s%d%c%b%f", str, int, ch, boolean, float);
```

- A impressão de cada tipo de dado ocupa um determinado espaço na tela, tal que:
 - Um caracter ocupa uma posição;
 - Um booleano ocupa quatro ou cinco posições, a depender de seu valor (true ou false);
 - Um inteiro pode representar a faixa de números entre -2.147.483.648 a 2.147.483.647, logo sua impressão pode ocupar até 11 posições;
 - Um float pode representar a faixa de números entre $\pm 10^{-308}$ a $\pm 10^{308}$, logo sua impressão pode ocupar até 312 posições;

ToString()

A função ToString (identificador) retorna o valor da variável dentro dos parênteses como uma string. Pode ser utilizada em qualquer tipo de dados, com exceção de array (neste caso, é preciso passar cada valor individualmente).

Programas Exemplos

- Alô Mundo:

```
fun int main ( ) {  
    print("Alo Mundo!");  
    return 0;  
}
```

- Série de Fibonacci

- Shell Sort