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

Jvlal
Especificação da Linguagem

João Victor Ribeiro Ferro
Lucas Albuquerque Lisboa

Maceió
2020.1

Introdução

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

Jvlal é estruturada, é *case-sensitive* (diferencia letras maiúsculas e minúsculas), é estaticamente tipada. 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 ser chamada. Jvlal não aceita passagem de subprogramas como parâmetro.

A declaração de de função é sempre iniciada pela palavra reservada fun, seguido pelo pelo seu tipo de retorno obrigatório (***int, float, bool, string, char***), seu identificador único e uma lista de parâmetros (os parâmetros acompanhados de seu tipo e seu identificador, separados por vírgula) 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 não terá o tipo de retorno e será sempre iniciada com a palavra reservada ***proc***, não retornando nada ao final de sua execução.

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

Ex.:

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

proc id (listaDeParametros){
     …
}
```

Definições de instruções

As instruções somente são declaradas dentro do bloco de funções ou de procedimentos, sendo encerrada com **';'** , sendo esse bloco delimitado por **'{'** e **'}'**.

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')('digit' | 'letter)*

Exemplos de nomes aceitos pela linguagem: count, c1;

Exemplos de nomes não aceitos pela linguagem: .abc, ab@c, 1ac,_1,_.

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, toString, length.

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

Há duas maneiras de declarar uma variável:

1. Pode ser definida fora do escopo de uma função e acessada globalmente, a qual irá compor o escopo global, sendo visível em todo o programa a partir do ponto da declaração;
2. Pode ser definida dentro do escopo do bloco da função como variável local, sendo visível na função a partir do ponto da declaração;

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**. Também é permitido a inicialização como: **tipodaDaVariável** e seu **identificador único** seguido pelo operador de atribuição **=** e o valor a ser atribuído.

Ex.:

```
int a ;
float b = 10.4, f;
char c = 'c';
bool d = True;
string e = "fff";
```

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 inicialização de valor é obrigatória 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 dígitos 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 dígitos decimais, seguido de um ponto e demais dígitos.

Ex.: `float c = 3.15;`

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 **apóstrofos**.

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 (*bool, int, float, char* e *string*). 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 dos limites definidos, é indicado erro.

Sua indexação ocorre mediante deslocamento, ou seja, a posição inicial é *zero* e a última equivale a *tamanho - 1*, similar à linguagem C. Caso tente ser atribuído um valor a uma posição inexistente do array, será acusado erro. Para saber o tamanho do array, usa-se a função ***length(),*** a qual recebe como argumento o identificador do arranjo (ou *string*) e retorna um valor inteiro não negativo.

Ex.:

```
int arr[9];
```

```
float arr[1024];
bool arr[2];
```

Coerção

Jvlal não aceita coerção entre variáveis de tipo diferentes. Dessa forma, todas as verificações de compatibilidade de tipo serão feitas estaticamente. Com exceção com a verificação de limites de array, que, caso seja definido por constante, também será feito estaticamente, mas, caso seja definido por variável, será feito dinamicamente.

Concatenação de string

O operador de concatenação **“#”** gera uma cadeia de caracteres (string). Com isso para realizar concatenação com os outros tipos é necessário utilizar o operador toString()* com o intuito de converter para **string**. Para realizar a concatenação com os tipos int, float, bool e char é preciso realizar a conversão para string.

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

Ex.:

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

        return 0;
}
```

Valores padrão

Tipo	Valor de inicialização
int	0
float	0.0
char	‘ ‘(caracter vazio)

string	“ ”(string vazia)
bool	false

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’)*) (‘\” ’)

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**: | ;

Atribuição

- O operador de atribuição ‘=’;

Operações suportadas

Operador	Tipos que realizam a operação
‘!’	**bool**
‘^’ ‘*’ ‘/’ ‘+’ ‘-’	**int, float**
‘<’ ‘<=’ ‘>’ ‘>=’	**int, float, char, string**
‘==’ ‘!=’ ‘=’	**int, float, bool, char string**
‘&’ ‘|’	**bool**
‘#’	**string, outros tipos somente com operador toString()**

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

Atribuição

É definido pelo símbolo ‘=’, sendo o lado esquerdo o identificador único e o lado direito o valor ou expressão. A expressão do lado direito do operador precisa que todos os seus operandos sejam do mesmo tipo.

Estrutura condicional

A condição da instrução “***if***” deve estar 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. Dessa forma, caso a condição seja verificada ***true***, será executado o conteúdo de instruções presente dentro do par de chaves. Caso seja verificada como ***false***, não entrará no “***if***”.

É possível inserir a o complemento “***else***” que, caso a condição do “***if***”, não se verifique verdadeira, o conjunto de instrução dentro das chaves do “else” será executado. Um “***else***” não pode, em nenhum caso, ser inserido sem estar associado a um “***if***” colocado anteriormente. A inserção do complemento “***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 interativa com controle lógico

A estrutura do “while” deve receber entre parênteses uma expressão lógica e conter um bloco de instruções delimitado por chaves. O loop será executado enquanto a expressão for verdadeira.

Ex.:

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

Estrutura interativa controlada por contador

A estrutura do ***for*** deve possuir variável de controle inteira, que deve ser declarada entre parênteses, a qual só é reconhecida no escopo do ***for***, possuindo um bloco de instruções delimitado por chaves.

Dentro do ***for*** o valor variável de controle não pode ser alterado, já que tem o passo que faz essa mudança.

No loop, os três valores **(a,b,c)** devem ser inteiros. O valor de **a** será o valor inicial do contador, **b** será o valor final e **c** será o valor de incremento, que realiza o passo a cada final de ciclo. Os três valores são pré-avaliados. Além disso, o valor final deve ser sempre maior do que o inicial para que o ***for*** seja executado e o número de interações é definido por:

Nº iterações = (valor final - valor inicial + tamanho do passo) / tamanho do passo

Ex.:

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

Desvios Incondicionais

Jvlal 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. É possível colocar mais de um parâmetro como entrada para a função read, separando os identificadores por vírgula.
- Saída: É realizada por meio da função ***print()***

- A função print pode imprimir o valor na tela, como variáveis e string. O identificador nesta função também pode ser visto como uma string não instanciado, 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 se encontre formatada, no código deverá ser especificado quais são as variáveis de cada um dos identificadores de formatação;
- A impressão de números são alinhados à direita, enquanto os demais são alinhados à esquerda da tela;
- 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 10^{-308} a 10^{308}. No entanto, se for omitido a especificação de casas decimais são utilizadas, por padrão, duas posições após o '.'.
- Há uma variação dessa função, chamada ***println()***, a qual mantém as mesmas propriedades, adicionando apenas uma quebra de linha ao final da impressão.
- Caso aconteça algum erro de formatação o ***print()***, ***println()*** e o ***toString()*** que terão a responsabilidade de tratar o erro de formatação. Ex .:

```
print ("@s@d@c@b@2f", str, int, ch, boolean, float);
```

Programas Exemplos

Alô Mundo

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

Série de Fibonacci

```
proc fib(int n) {
     int n1 = 0, n2 = 1, n3;
     if (n == 0) {
          println("@d", n);
     }
     if (n == 1) {
          println("0, @d", n);
     } else {
          string separator = ",";
          print("0, 1, ");
          while (true) {
               n3 = n1 + n2;
               print("@s@d", separator, n3);
               if (n3 >= n) {
                    return;
               }
               n1 = n2;
               n2 = n3;
          }
     }
}
fun int main() {
     int n;
     read(n);
     fib(n);
     return 0;
}
```

Shell Sort

```
proc shellSort (int array [ ] , int n) {
     int h = 1, c , j;

     while( h < n ) {
          h = h * 3 + 1;
     }
     h = h / 3;
     while ( h > 0 ){
          for (int i : h , 1 , n){
               c = array(i);
               j = i;
               while ( j >= h & array [ j - h ] > c ) {
                    array [ j ] = array [ j - h ];
```

```
                    j = j - h ;
                }
                    array [ j ]  = c;
            }
                h = h / 2;
        }
    }

    fun int main() {
        int n;
        println("Tamanho do array");
        read(n);
        int array [ n ];
        println("Digite os numeros que serao ordenados");
        for( int i : 0,1,n) {
            read( array [ i ] );
        }

        println("lista dos valores digitados");
        for (int i : 0, 1, n){
            int a = array [ i ];
            println (a);
        }

        shellSort (array,n);

        println("valores ordenados");
        for(int i : 0 , 1 , n) {
            int b = array [ i ];
            println(b);
        }
        return 0;
    }
```