Universidade Federal de Alagoas
Instituto de Computação
Compiladores

Onicla - Especificação mínima da linguagem

Ramon Basto Callado Lima Lopes
José Rubens da Silva Brito

Maceió
2020.1

Sumário

1. Introdução

A linguagem de programação Onicla tem seu desenvolvimento tendo sua base a linguagem de programação C, Pascal, tem como objetivo ser uma linguagem de implementação instrutiva e busca gerar um bom desenvolvimento educacional.

Onicla é indubitavelmente uma linguagem não orientada a objetos. Tem o propósito de ter as funcionalidades mais básicas possíveis. É uma linguagem simplesmente estruturada, não é formalmente *estruturada em blocos* pois não permite a criação de funções dentro de outras funções. Não admite a coerção implícita de tipos gerando assim uma maior confiabilidade, e obviamente possui suas palavras reservadas que, ao serem lidas pelo usuário, são facilmente identificadas a que parte do código pertencem, gerando assim uma melhor legibilidade da linguagem. Os blocos são delimitados pelas palavras reservadas *Begin* e *End*.

2. Estrutura Geral do Programa

Um programa em Onicla é composto obrigatoriamente da seguinte forma:

- Os delimitadores dos blocos de funções são definidos pelas palavras reservadas ***Begin*** (abertura) e ***End*** (fechamento).
- As funções devem conter obrigatoriamente a palavra reservada ***Function***, em seguida o tipo de retorno, identificador e delimitadores '()'.
- A função principal (***Main***) é delimitada pelas palavras reservadas *Begin* e *End*, e o retorno sendo feito pela palavra reservada ***Refound***.
- A função *Main* é estruturada (declarada) ao fim do programa, onde anteriormente todas as outras funções que compõem o código já foram definidas.
- O retorno do programa (***Refound***) é um valor do tipo especificado pelo usuário na declaração da função. Caso não haja declaração de retorno ao fim da função, pelo usuário e a função não seja do tipo *Void* resultará em erro.
- Na declaração de uma função, após seu nome deve vir o bloco de parâmetros delimitado por parênteses com a declaração do tipo e o nome da variável, separados por vírgula das outras variáveis.

Ex.:

```
Function Integer soma(Integer valor1, Integer valor2) Begin
    Integer resultado;
    resultado = valor1 + valor2;
    Refound resultado;
End

Function Integer Main ( ) Begin
    Integer a, b, c;
    Input(a);
    Input(b);
    c = soma(a, b);
    Print(c);
```

Refound 0;

End

3. Conjuntos de Tipo de Dados e Nomes

A linguagem de programação Onicla é *case-sensitive* e a atribuição de valores é uma instrução.

3.1. Palavras Reservadas

Null, Print, Printnl, Printl, Input, True, False, If, Else, Bool, Float, Integer, Characterarray, While, Repeat, Function, Main, Begin, End, Void, Or, And, Character, Refound, ToString.

3.2. Identificador

Os identificadores da linguagem Onicla seguem seguintes restrições:

- Iniciam-se, **OBRIGATORIAMENTE**, com uma letra minúscula.
- As demais sequência de caracteres, podem ser letras maiúsculas, minúsculas ou números.
- O tamanho máximo do identificador é de 16 caracteres
- Não é possível usar palavras reservadas e espaços em brancos para nomear identificadores.
- Uma declaração de variável tem, por exemplo, o formato: ***Integer*** **numero;**
- É permitido a declaração de múltiplas variáveis do mesmo **TIPO** na mesma linha com os identificadores separados por "**,**" (vírgula)

  Ex.: ***Integer*** a, b, c;
- É **OBRIGATÓRIO** uma variável ser declarada antes de ser utilizada em um bloco de instruções.
- As variáveis na linguagem Onicla possuem apenas escopo **LOCAL.**

3.3. Comentário

Os comentários poderão ser efetuados por linha onde podem ser feito através do caractere "**#**".

3.4. Inteiro

Integer identifica variáveis do tipo inteiro com **tamanho** de 32 bits. Onde seus literais são expressos em uma sequência de dígitos inteiros, podendo ser negativo, somente quando colocado um sinal "**-**", na ocorrência em que seja dado um valor excedente, resultará em uma mensagem indicando erro de execução.

Ex.: ***Integer*** *value;*

value = 10;

3.5. Ponto Flutuante

Float identifica variáveis do tipo ponto flutuante no padrão IEEE 754 com **tamanho** de 32 bits. Onde seus literais são expressos em uma sequência de dígitos, seguido por somente um ponto e por fim uma sequência de até 6 dígitos para a parte fracionária. Em caso de exceção do valor máximo, o tratamento é igual ao tópico anterior (3.4).

Ex.: ***Float*** *average*;

average = ***3.1415***;

3.6. Caracteres

Character identifica variáveis do tipo caracter com número de 8 bits. Seus literais são apenas um caractere, utilizando a formatação *ASCII*, é possível declarar um caracter vazio (valor default null), na situação em que seja dado mais de um caractere à variável, a resposta ao caso será um erro.

Pode-se atribuir ao tipo ***Character*** apenas um caractere, sendo necessário o uso de apóstrofo (‘) para marcar o início e o fim do caractere, como podemos ver no exemplo abaixo.

Ex.: ***Character*** *letter*;

letter = ‘a’;

3.7. Cadeia de caracteres

Characterarray identifica variáveis do tipo caracter com número de 8 bits, combinado com o tamanho da cadeia. Seus literais são uma cadeia de caracteres com tamanho mínimo 0 e tamanho máximo ilimitado. O tamanho da cadeia de

caracteres será de forma dinâmica, onde é possível declarar uma cadeia vazia ou com ***n*** caracteres, onde todos os caracteres devem seguir o padrão *ASCII.*

Uma cadeia de caracteres é gerada pela palavra reservada *Characterarray* em seguida o campo com nome da variável. Para realizar uma atribuição a uma cadeia de caracteres, é necessário usar apóstrofo (‘’) para delimitar o início e o fim da cadeia de caracteres. Sendo possível utilizar underline e espaços em branco para definir a constante literal.

Ex.: ***Characterarray*** *word*;

word = ‘Hello Word’;

3.8. *Boolean*

Bool identifica variáveis do tipo booleano, sendo possível atribuir apenas dois valores: *True* ou *False*.

Ex.: ***Bool*** *flag*;

3.9. Arranjo unidimensionais

Um vetor na linguagem Onicla é definido de forma análoga à linguagem de programação C. Onde é possível definir ***<Tipo> Identificador[Tamanho];***, onde o tipo deve ser somente numérico, não admitindo cadeia de caracteres. Caso seja dado um arranjo menor que o tamanho declarado, os espaços não ocupados serão atribuídos com *Null* e para um **acesso fora do limite**, o resultado é um erro.

O tamanho do arranjo tem como sua primeira posição o zero (0) e seu final pelo (tamanho máximo do arranjo - 1).

Ex.: ***Integer*** *notes*[10];

Float *averages*[10];

Um arranjo pode ser preenchido em um laço de controle lógico/contador, ou ser preenchido manualmente em cada posição. Caso escreva em uma posição já ocupada, será mantido o último valor lido, e o anterior será descartado.

Para passar um arranjo como parâmetro para uma função, é preciso que o arranjo seja enviado **OBRIGATORIAMENTE** com o seu tamanho ou o inteiro que representa o tamanho após uma vírgula. É realizada uma comparação entre o tamanho do arranjo e a posição de atribuição do valor. Caso o valor esteja dentro

dos limites do arranjo, a operação é realizada, caso contrário, é gerado um erro de **acesso fora do limite.**

Ex.:

Integer *notas[5];*

notas[0] = 10;

notas[1] = 5;

funcao(notas, <tamanho>);

Envia todo o arranjo e

3.10. Operações Suportadas

As operações que a linguagem Onicla suporta se encontram abaixo.

Tipo	Operações
Integer	Atribuição, aritméticos, relacionais, concatenação
Float	Atribuição, aritméticos, relacionais, concatenação
Characterarray	Atribuição, relacionais, concatenação
Character	Atribuição, relacionais, concatenação
Bool	Relacionais

O tipo **Float** não suporta a operação aritmética de resto de divisão de dois operandos.

3.11. Valores Default

Os valores default atribuídos a cada variável declarada são:

Tipo	Valores Default
Integer	0
Float	0.0

Characterarray	*Null*
Character	*Null*
Bool	*False*

3.12. Coerção

A linguagem Onicla não aceita coerção implícita entre variáveis de tipos diferentes. Toda a verificação de compatibilidade de tipo será efetuada estaticamente. Através disso, pretende-se aumentar a confiabilidade da linguagem e diminuir os erros de tipo.

4. Conjunto de Operadores

4.1. Aritméticos

Operadores	Operações
+	Soma de dois operandos
^	Concatenação de dois Characterarray
-	Subtração de dois operandos
*****	Multiplicação de dois operandos
/	Divisão de dois operandos
%	Resto da divisão de dois operandos
~	Unário negativo: realizará a inversão de sinal tipos Integer ou Float

4.2. Relacionais

Operadores	Operação
<	Menor que
>	Maior que
<=	Menor ou igual que

>=	Maior ou igual que
==	Igualdade entre dois operandos
=/=	Desigualdade entre dois operandos

4.3. Booleano

Operadores	Operação
!	Negação lógica
And	Conjunção
Or	Disjunção

4.4. Concatenação de cadeia de Caracteres

É representado pelo carácter "**^**", suporta dados do tipo ***Characterarray, Character, Integer ou Float, Bool.*** A concatenação de dois tipos de dados (entre os citados acima) gera uma cadeia de caracteres. Como a concatenação sempre gera uma cadeia de caracteres é necessário **OBRIGATORIAMENTE** utilizar primeiro a instrução ***ToString()**** para converter todos os tipos que sejam diferentes de ***Characterarray.***

Após realizar a conversão de tipos explícita do valor através da instrução ***ToString()****(Especificado no tópico 5.6)*, pode-se utilizar o carácter "^" para concatenar os dois tipos de cadeia de caracteres e atribuir a uma nova cadeia de caracteres. A cadeia de caracteres que receberá o resultado da concatenação tem que ser criada antes de ser utilizada, seguindo assim os padrões de criação de variáveis da linguagem de programação Onicla.

Ex.:

```
Function Integer Main ( ) Begin
        Integer a;
        a = 10;
        Characterarray b;
        b = 'Mundo'
        Characterarray new;
```

```
        ToString(a);
        new = a ^ b;
        Print(new);

        Refound 0;
End
```

4.5. Precedência e Associatividade

A precedência se dá por ordem descrente, onde os operadores na mesma linha possuem a mesma precedência.

Precedência	Operadores	Associatividade
~	Menos unário	Direita → esquerda
*** /**	Multiplicativos	Esquerda → direita
%	Resto	Esquerda → direita
+ -	Aditivos	Esquerda → direita
!	Negação lógica	Direita → esquerda
< > <= >=	Comparativos	Esquerda → direita
== =/=	Relacional	Esquerda → direita
And Or	Conjunção e Disjunção	Esquerda → direita

4.5.1. Operadores multiplicativos e aditivos

Ao realizar uma operação em variáveis do mesmo tipo utilizando esses operadores, os valores produzidos por tais operações devem ser atribuídos a variáveis do mesmo tipo.

No caso em que esses operadores sejam empregados em variáveis de tipos distintos, por exemplo, operações entre ***Integer*** e ***Float***, o compilador retornará um erro de tipo.

4.5.2. Operadores comparativos e igualdade

Essas operações geram como resultado um valor do tipo verdadeiro ou falso e não são associativos. Em operações de tipos diferentes não é permitida a operação de comparação ou igualdade entre *Integer* e *Float*.

4.5.3. Operadores de negação lógica, conjunção e disjunção

Após determinação da resultante dessas operações, a resposta obtida será também do tipo *Bool*.

5. Instruções

Cada linha de instrução é encerrada somente se com a presença de "**;**". Sendo os blocos de instrução delimitados por *Begin* para iniciar o bloco e *End* para fechar.

5.1. Atribuição

A atribuição na linguagem Onicla é definida pelo uso de "=", onde o lado esquerdo da igualdade diz respeito ao identificador da variável e o lado direito o valor ou expressão a ser atribuído. Os dois lados devem ser do mesmo tipo, pois a linguagem Onicla não permite coerção.

A linguagem Onicla trata a atribuição como sendo uma instrução, diferente da linguagem C que é uma operação. Visando sempre aumentar a confiabilidade da linguagem.

Ex.:

```
Integer i;
i = 10;
```

5.2. Estrutura condicional de uma ou duas vias

5.2.1. *If* e *Else*

A estrutura de condição ***If*** terá sempre como condição de entrada uma expressão lógica, seguido do seu bloco de instruções a serem executadas, delimitadas por *Begin*e *End*.

O algoritmo executará as instruções contidas no bloco *Begin, End* se e somente se, a sua condição lógica for verdadeira, caso contrário, se o condicional for falso, executará as instruções do bloco ***Else*** se houver.

Ex.:

```
# uma via
If (<condição lógica>) Begin
        <instruções>
End

# duas vias
If (<condição lógica>) Begin
        <instruções>
End
Else Begin
        <instruções>
End
```

5.3. Estrutura iterativa

5.3.1. Controle lógico - *While*

A estrutura iterativa com controle lógica que a linguagem Onicla implementa é a estrutura *While*. Onde a repetição irá ocorrer enquanto a condição for satisfeita. O laço é finalizado quando a condição for falsa. As condições deste bloco são do tipo lógica ou variável booleana.

Ex.:

```
While(<condição lógica>) Begin
        <instruções do bloco>
End
```

5.3.2. Controle por contador - Repeat

Na estrutura ***Repeat*** o número de interações é definido pelo usuário e o controle é feito através do contador, diferente do ***While*** descrito acima. Por se tratar de um bloco, é definido por *Begin* e *End*.

Sua estrutura consiste em 3 parâmetros (**valor inicial, passo e valor final**). O **variável de controle de laço** pode ser declarado na própria estrutura ou fora, porém tem que está dentro de um mesmo escopo do controlador.

A variável de controle de laço é incrementada internamente pelo **passo** ao final de cada ciclo. O **valor final** deve ser sempre maior ou igual ao inicial para que o ***Repeat*** seja executado. O número de interações é definido por:

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

Os valores iniciais e finais dessa estrutura, são valores fechados. Ou seja, o algoritmo é executado tanto no valor inicial, quanto no final.

A variável de controle da repetição não pode ser alterada dentro do ***Repeat***, pois já existe o passo, para determinar a variação do valor inicial. Esse valor deve ser iniciado dentro da instrução, com o valor desejável, sendo ele menor que o valor final de parada.

Ex.:

```
Function Integer contador (Integer count) Begin
        Repeat (Integer i = 0, 1, 10) Begin
                count = i + 1;
        End
        Refound count;
End


Function Integer Main ( ) Begin
        Integer out = contador(0);
        Print(out);
End
```

Nesse caso, o número de acréscimo é dado da seguinte forma:
Nº iterações = ((10 - 0)+1) / 1.
Logo, o valor retornado por *Refound* e dado de saída ao usuário será 11.

5.4. Entrada e saída

As funções de entrada e saída da linguagem Onicla são definidas por dois identificadores ***Input*** para entrada e ***Print*** para saída.

É possível realizar a entrada de múltiplas variáveis em um mesmo ***Input***, sendo cada variável separada por vírgula, tendo a quebra de linha depois da última entrada. De maneira análoga, ocorre com o ***Print***.

5.4.1. Entrada

Irá atribuir um valor de entrada fornecido pelo usuário a uma ou várias variáveis, na qual está utilizando o método ***Input***. Não admite a entrada de diferentes dados em uma mesma linha. E a entrada de dados é encerrada com *enter*. Passando assim, para a próxima entrada ou instrução.

Ex.:

Input(a);

Input(b);

5.4.2. Saída

Irá mostrar na tela o valor atribuído à variável. Como também pode mostrar na tela um *Characterarray* sem ter sido obrigatoriamente definido antes, sendo necessário apenas colocar entre apóstrofo (caso contrário, resultará em **erro**) e o que for escrito será impresso na tela. Ou seja, pode-se mostrar na tela, textos escritos que não foram armazenados em uma variável.

Todas as declarações dentro de um só *Print* serão mostradas na mesma linha e enquanto que as declarações feitas no *Printnl* imprime as variáveis entre parênteses e quebra a linha.

O *Printl* é possível realizar a saída formatada, onde é possível definir o tamanho da saída. Para valores numéricos do tipo *Float* caso não seja especificado o tamanho, por default a saída será apenas com 2 casas fracionárias. Sendo formado pelos parâmetros ***(<quantidade de caracteres>,<valor/variavel>).*** Enquanto que os valores default dos outros tipos (*Integer, Character, Characterarray* e *Bool*) será o tamanho da variável.

Ex.:

Characterarray a, b;

a = ‘Hello’;

b = ‘Word’;

Mesma linha:

Print(a, b);

Linha distintas:

Printnl(a);

Printnl(b);

Concatenação de uma variável do tipo *Characterarray* com *Characterarray*

Print(a ^ 'Ola');

Saída formatada

Float f = 12.344324;

Characterarray r;

r = 'Hello';

Printl(f);

Caso atribuída apenas a variável *Float* sem tamanho, a saída terá apenas 2 casas fracionárias (12.34)

Print(3, f);

Neste caso a saída será (12.3)

Printl(3, r);

A saída gerada será ('Hel')

Printl(r)

A saída gerada será ('Hello')

5.5. Funções

De forma geral, as funções são muito semelhantes à linguagem de programação C. Onde de início ocorre a declaração da palavra reservada ***Function***, para dizer ao compilador que ali irá se iniciar uma função, em seguida o tipo de retorno da função, podendo ser ***Integer***, ***Float***, ***Characterarray, Char*** ou ***Bool***. Em seguida é necessário definir um identificador para a função, onde obrigatoriamente tem que se iniciar com letra minúscula. Em seguida, abertura de parênteses, parâmetros (se houver) e fechamento de parênteses. Por fim, ***Begin*** para iniciar a escrita do bloco de instruções a serem executadas e ao fim ***End*** para informar o encerramento do bloco.

A linguagem Onicla não aceita sobrecarga de funções, ou seja, não é definida outra função com o mesmo identificador. Antes do ***End*** é obrigatório ter o

Refound, palavra reservada essa que efetua o retorno da função, caso não seja atribuído valor a ele, o valor default devolvido por ele é zero.

Para chamar uma função, deve ser utilizado o seu identificador, e dentro dos parênteses, os valores que serão utilizados pela função.

*Function <**Tipo**> <**nome**> (<**Tipo**> <**identificador**>) Begin*
    *<**declarações de Tipos e variáveis**>*
    *<**instruções**>*

    *Refound <**valor a ser retornado**>;*
End

5.6 *ToString()*

A instrução *ToString()* deve ser usada com uma única variável passada como parâmetro, sendo do tipo *Integer*, *Float, Bool* ou *Character*, onde seu retorno é do tipo *Characterarray* para realizar a concatenação com outra cadeia de caracteres.

6. Programas Exemplos

6.1 Alô mundo

```
Function Integer Main ( ) Begin
    Print('Alo mundo!');

    Refound 0;
End
```

6.2 Série de Fibonacci

```
Function Integer fibonacci(Integer n) Begin
    If(n < 2) Begin
        Refound n;
    End
    Else Begin
        Refound fibonacci(n - 1) + fibonacci(n - 2);
    End
End
```

```
Function Integer Main ( ) Begin
        Integer n, total;
        Print('Digite o tamanho da sequencia:');
        Input(n);

        Repeat(Integer i = 0, 1, n) Begin
                total = fibonacci(i);
                Print(total);
        End

        Refound;
End
```

6.3 Shell sort

```
Function Void shellsort(Integer array[ ], Integer n) Begin
        Integer h = 1, c, j;

        While (h < n) Begin
                h = h * 3 + 1;
        End

        h = h / 3;

        While(h > 0) Begin
                Repeat (Integer i = h, 1, n) Begin
        c = array[i];
        j = i;
        While (j >= h And array[j - h] > c) Begin
                array[j] = array[j - h];
                j = j - h;
        End
        array[j] = c;
        End
        h = h / 2;
        End
        Refound;
End

Function Integer Main ( ) Begin
```

```
        Integer n, v;
        Print('Digite o tamanho do array a ser ordenado: ');
        Input(n);
        Integer array[n];

        Print('Digite aleatoriamente os numero para serem ordenados: ');
        Repeat (Integer i = 0, 1, n) Begin
                Input(array[i]);
        End
        Print('Valores adicionados: ');
        Repeat (Integer i = 0, 1, n) Begin
                v = array[i];
                Printnl(v);
        End
        shellsort(array[n], n);

        Print('Valores ordenados: ');
        Repeat (Integer i = 0, 1, n) Begin
                v = array[i];
                Printnl(v);
        End

        Refound;
End
```