

UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação
Bacharelado em Ciência da Computação

Especificações da Linguagem Linkin Park

Hellena Almeida Canuto
João Vitor Santos Tavares
Wagner Anthony de Medeiros Silva

Maceió - AL, 17/12/2021

Sumário

Sumário	1
1. Introdução	2
2. Estrutura Geral do Programa	2
3. Tipos de dados e Nomes	2
3.1. Palavras Reservadas	2
3.2. Identificadores	3
3.3. Comentários	3
3.4. Inteiro	3
3.5. Ponto Flutuante	3
3.6. Caracteres	3
3.7. Cadeia de Caracteres	3
3.8. Booleanos	4
3.9. Vetores Unilaterais	4
3.10. Operações Suportadas	4
3.11. Valores Padrão	4
3.12. Coerção	4
4. Conjuntos de Operações	5
4.1. Aritméticos	5
4.2. Relacionais	5
4.3. Lógicos	5
4.4. Concatenação de Cadeia de Caracteres	5
4.5. Precedência e associatividade	5
4.5.1 Operadores multiplicativos e aditivos	6
4.5.2 Operadores Comparativos e Igualdade	6
4.5.3 Operadores de negação e conjunção	6
5. Instruções	6
5.2 Estrutura condicional de uma ou duas vias	7
5.2.1 If e Else	7
5.3 Estruturas Iterativas.	7
5.3.1 Controle Lógico - While	7
5.3.2 Controle por Contador - For	7
5.4 Entrada e Saída	8
5.4.1 Entrada	8
5.4.2 Saída	8
5.5. Funções	8
6. Programas exemplo	9
6.1. Hello World	9
6.2. Sequência de Fibonacci	9
6.3. Shell Sort	10

1. Introdução

A LinkinPark é uma linguagem de programação estaticamente tipada, não orientada a objetos, baseada em Python. O objetivo desta linguagem é servir ao aprendizado e projetos simples/scripts.

2. Estrutura Geral do Programa

O procedimento de escrita de um programa em LinkinPark se dá da seguinte forma:

- Funções e blocos de comandos são iniciadas/os e encerrados através das palavras reservadas **Open** e **Close**, respectivamente;
- A declaração da função principal é feita pelo uso da palavra reservada **main**, juntamente com o uso das já citadas **Open** e **Close**;
- Ao declarar uma função, seu tipo (inteiro, ponto flutuante, void, etc) deve, obrigatoriamente, ser especificado, seguido dos parâmetros da função, entre parênteses, e separados por vírgulas;
- O retorno da função é dado em concordância ao tipo especificado em sua declaração (tal regra também é válida para a função **Main**);
- O retorno de funções é feito utilizando a palavra reservada **Back** (de volta);
- A função principal é declarada ao fim do programa, após as especificações das funções adicionais que o mesmo venha a possuir.

3. Tipos de dados e Nomes

Linkin Park é uma linguagem de programação case-sensitive, ou seja, a mesma reconhece, por exemplo, “Jogo” e “JOGO” como variáveis diferentes e a atribuição de valores funciona como uma instrução.

3.1. Palavras Reservadas

Bool, Char, Str, Back, Open, Close, And, Or, Main, While, For, Float, Function, Print, Scan, Int, True, False, Null, Empty, If, Else.

3.2. Identificadores

Identificadores em Linkin Park seguem os padrões descritos a seguir:

- Devem, obrigatoriamente, ter seu início em caixa alta;
- Com exceção do primeiro caractere, os demais componentes de um identificador podem ser letras, dígitos e/ou underlines;
- Nenhum identificador deve ser maior do que o limite estabelecido de 16 caracteres;
- Palavras reservadas e/ou espaços em branco não podem ser utilizados na criação de um novo identificador;
- Variáveis devem ser previamente declaradas antes de serem utilizadas em qualquer tipo de operação ou bloco de instruções.

3.3. Comentários

São permitidos apenas comentários inline, sinalizados pela utilização do símbolo `//`.

3.4. Inteiro

O tipo inteiro com 32 bits é identificado por **Int**. Os literais são expressos como uma sequência de dígitos inteiros.

3.5. Ponto Flutuante

O tipo ponto flutuante (decimal), que também possui o número de bits limitado a 32, é identificado por **Float**. Seus literais são expressos a partir de um número inteiro seguido de um ponto que é sucedido por uma sequência de números que representam a parte decimal do referido número.

3.6. Caracteres

Caracteres, com limitação de 8 bits, são identificados por **Char**. Seus literais são representados por um caractere simples.

3.7. Cadeia de Caracteres

Em LinkinPark, **Str** é utilizado para identificar variáveis do tipo cadeia de caracteres com tamanho de 8 bits por caractere e com limite dinâmico. Seus literais são uma cadeia de caracteres cujo mínimo tamanho suportado é 0. Pode ser declarada sem atribuição de valor na mesma linha de declaração da variável.

3.8. Booleanos

Bool identifica variáveis do tipo booleano, cujo valor só pode ser representado por **True** ou **False**.

3.9. Vetores Unilaterais

Um vetor é definido seguindo o seguinte padrão `<tipo> identificador[<tamanho do vetor>]`. Em funções, a passagem por parâmetro é feita passando o tamanho do vetor para uma referência na função de destino.

3.10. Operações Suportadas

As seguintes operações são suportadas pela linguagem:

- *Inteiro*: Atribuição, concatenação, aritméticos e relacionais
- *Ponto flutuante*: Atribuição, concatenação, aritméticos e relacionais.
- *Cadeia de caracteres*: Atribuição, concatenação e relacionais.
- *Caracteres*: Atribuição, concatenação e relacionais.
- *Booleanos*: Atribuição, lógica, relacional de igualdade e desigualdade, concatenação.

Observação: O tipo **Float** não suporta a operação de **resto** entre dois operandos apenas o tipo **Int**.

3.11. Valores Padrão

Os valores padrão da linguagem são especificados para definir valores para que não possuem valores atribuídos a elas. Assim, os valores padrão atribuídos para cada variável declarada são:

- *Inteiro*: 0
- *Ponto flutuante*: 0.0
- *Cadeia de caracteres*: Null
- *Caracteres*: Null
- *Booleanos*: False

3.12. Coerção

Essa linguagem é estaticamente tipada, e portanto não aceita coerções entre variáveis que possuem tipos diferentes. As verificações de compatibilidade por tipo devem ser efetuadas de forma estática. Aumentando assim, a confiabilidade da linguagem.

4. Conjuntos de Operações

4.1. Aritméticos

- “+”: Soma entre dois operandos
- “-”: Subtração de dois operandos
- “*”: Multiplicação de dois operandos
- “/”: Divisão de dois operandos
- “%”: Resto da divisão entre operandos
- “_”: Nega variáveis do tipo inteiro e flutuante
- “&”: Concatena duas cadeias de caracteres

4.2. Relacionais

- “==”: Igualdade entre dois operandos
- “!=”: Desigualdade entre dois operandos
- “>=”: Maior ou igual que
- “<=”: Menor ou igual que
- “>”: Maior que
- “<”: Menor que

4.3. Lógicos

- “Not”: Negação
- “And”: Conjunção
- “Or”: Disjunção

4.4. Concatenação de Cadeia de Caracteres

A operação de concatenação na linguagem LinkinPark é representada pelo caractere “&” e suporta apenas o tipo **Str** e dados que possuem associatividade da esquerda para a direita., além de outros tipos de saída de dados.

4.5. Precedência e associatividade

Precedência	Operadores	Associatividade
~	Menos Unário	Direita->Esquerda
*/	Multiplicativos	Esquerda->Direita
%	Resto	Esquerda->Direita
+ -	Aditivos	Esquerda->Direita

!	Negação	Direita->Esquerda
<> <= >=	Comparativos	Sem associatividade
== !=	Igualdade	Esquerda->Direita
And Or	Conjunção	Esquerda->Direita

4.5.1 Operadores multiplicativos e aditivos

Para operações entre variáveis do mesmo tipo, as operações de multiplicação e adição produzem saídas deste mesmo tipo. O único caso tratado pela linguagem de operações multiplicativas/aditivas entre variáveis de tipos diferentes é o caso entre variáveis inteiras e flutuantes, onde prevalece o tipo Float.

4.5.2 Operadores Comparativos e Igualdade

As operações comparativas e de igualdade geram um valor de tipo Booleano (verdadeiro ou falso) e não são associativas. Nas operações com tipos diferentes só é permitida a operação de comparação ou igualdade entre Inteiro e Flutuante.

Em Linkin Park operações comparativas e de igualdade retornam valores do tipo **Bool(True ou False)** que não possuem associatividade. Em operações entre diferentes tipos somente são permitidas comparações de igualdade entre tipos **Int** e **Float**.

4.5.3 Operadores de negação e conjunção

Em Linkin Park, o resultado dessas operações gera uma resposta que também possui o tipo **Bool**.

5. Instruções

Em Linkin Park as instruções em uma determinada linha são encerradas a partir da utilização do símbolo “;”. Já os blocos (funções, condicionais, repetição,etc) são iniciados e terminados pelo uso das palavras reservadas **Open(abertura)** e **Close (fechamento)**.

5.2 Estrutura condicional de uma ou duas vias

5.2.1 If e Else

Blocos da estrutura condicional se devem, necessariamente, possuir uma condição, avaliada por uma expressão lógica ou uma variável tipo **Bool**. Seguidas de um bloco de instruções a executar, delimitado pelas palavras **Open** e **Close**.

Caso a condição seja verdadeira, as instruções contidas no bloco associado a condicional **If** são executadas. Do contrário e caso exista um bloco **Else**, as instruções desse bloco serão executadas. Caso não haja um **Else**, o programa apenas verifica o bloco associado ao **If**.

5.3 Estruturas Iterativas.

5.3.1 Controle Lógico - While

Em Linkin Park implementamos a estrutura **While** para que servirá como estrutura de interação com controle lógico para a linguagem. A repetição nessa estrutura ocorre enquanto a condição de controle for verdadeira. Dessa forma temos que o laço de repetição é executado enquanto a condição de controle lógica da nossa estrutura for **True** (verdade). Assim, o laço vai ser encerrado à partir do momento que a condição de controle se tornar **False** (falso). Portanto, temos que as condições presentes neste bloco são do tipo lógicas, ou ainda variáveis do tipo **Bool**.

5.3.2 Controle por Contador - For

Em Linkin Park, a estrutura **For**, a quantidade de interações é definida pelo programador e seu controle é feito a partir de um contador, sendo uma alternativa ao uso do **While**, anteriormente descrito na seção 5.3.1. Todavia, também se trata de um bloco, necessitando, assim, do uso das palavras reservadas **Open(abertura)** e **Close(fechamento)**.

Em uma estrutura interativa controlada por um contador, são obrigatórias as existências de valor inicial, passos a serem realizados e um valor final para que a interação seja finalizada. O valor inicial é incrementado internamente pelo passo no final de cada interação. O valor final deve ser sempre maior ou igual ao inicial para que o **For** seja executado.

5.4 Entrada e Saída

Em Linkin Park, as funções de entrada e saída são implementadas por meio de palavras chaves, sendo **Scan** reservada para entrada de dados e **Print** para saída.

5.4.1 Entrada

Uma vez fornecida uma entrada pelo usuário, a função de entrada atribui a uma variável determinada de tipo correspondente por meio do método **Scan**.

5.4.2 Saída

Os resultados referentes a um programa em específico são exibidos na tela pelo método **Print**. **Strings** podem ser passadas para o método **Print** sem, necessariamente, serem declaradas com antecedência.

Todas as declarações em um **Print** serão exibidas na mesma linha. Para exibir tais declarações em linhas diferentes, é necessário utilizar mais de um **Print**.

5.5. Funções

Em Linkin Park, temos que as funções presentes na nossa linguagem são em geral incrivelmente semelhantes às funções encontradas na lendária linguagem de programação C. Portanto, na nossa declaração devemos utilizar a palavra reservada **Function** para informar ao compilador que começamos a função. Logo após devemos informar o tipo de retorno da função (**Int**, **Bool**, **Str**, **Char**, **Float**). Logo em seguida é preciso definir um identificador para a nova função, sabendo que tal identificador deve obrigatoriamente começar com uma letra minúscula. Em seguida devemos abrir os parênteses, inserir os parâmetros caso necessário, e fechar os parênteses. Para terminar devemos escrever as palavras reservadas **Open** e **Close** para iniciar a escrita do bloco de instruções.

Nossa linguagem não aceita a sobrecarga de funções, portanto não é permitido funções que possuam o mesmo identificador. Antes do fim da função é necessário utilizar a palavra reservada **Back** para retornar o valor desejado ou o valor padrão de acordo com o tipo da função.

Quando queremos chamar a função, devemos usar o identificador que foi assinalado para a função, e entre os parênteses os valores que serão utilizados nessa função.

6. Programas exemplo

6.1. Hello World

```
# Programa Olá, mundo

Function Int Main() Open
    Print('Hello, world :)');

    Back;
Close
```

6.2. Sequência de Fibonacci

```
Function Int fibonacci(Int n) Open
    If(n < 2) Open
        Back n;
    Close
    Else Open
        Back fibonacci(n - 1) + fibonacci(n - 2);
    Close
Close

Function Int Main ( ) Open
    Int n, total;
    Print('Tamanho da sequencia: ');
    Scan(n);

    For (Int i = 0, 1, n) Open
        total = fibonacci(i);
        Print(total);
    Close

    Back;
Close
```

6.3. Shell Sort

```

Function Empty shellsort(Int array[ ], Int n) Open
    Int c, j;
    Int h = 1;

    While (h < n) Open
        h = h * 3 + 1;
    Close

    h = h / 3;

    While(h > 0) Open

        For (Int i = h, 1, n) Open

            c = array[i];
            j = i;

            While (j >= h And array[j - h] > c) Open

                array[j] = array[j - h];
                j = j - h;

            Close

            array[j] = c;
        Close

        h = h / 2;

    Close

    Back;

Close

Function Int Main ( ) Open
    Int n, current;
    Print('Tamanho do array: ');
    Scan(n);

```

```

    Int array[n];

    Print('Quantos elementos o array possui: ');

    For (Int i = 0, 1, n) Open
        Scan(array[i]);
    Close

    Print('Valores adicionados: ');

    For (Int i = 0, 1, n) Open
        current = array[i];
        Printnl(current);
    Close

    shellsort(array[n], n);

    Print('Valores ordenados: ');

    For (Int i = 0, 1, n) Open
        current = array[i];
        Printnl(current);
    Close

    Back;
Close

```