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

Especificações da Linguagem Linkin Park

Hellena Almeida Canuto
João Vitor Santos Tavares

Maceió - AL, 17/12/2021

Sumário

1. Introdução

A Linkin Park 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**;
- A estrutura para declaração das funções é a seguinte: 'Function' <Tipo da função> <Identificador> (<Parâmetros separados por vírgula>) open <Bloco> Close;
  Exemplo:

```
Function Bool Teste(Float x, Bool c) OPEN
    If (x > 10.5) OPEN
        Back c;
    CLOSE
    Back False;
CLOSE
```

- O tipo do retorno de uma função deve ser, necessariamente, o mesmo tipo especificado na declaração da função, isto é, uma função Int, por exemplo, deve apresentar um Int como retorno;
- O retorno de funções é sinalizado com a palavra reservada **Back** (de volta);
- A função principal é declarada ao fim do programa, após as declarações de todas as outras funções adicionais.

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 baixa;
- 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 entre aspas 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 caracter e com limite dinâmico. Seus literais são uma cadeia de caracteres entre aspas simples, cujo mínimo tamanho suportado é 0.

3.8. Booleanos

Bool identifica variáveis do tipo booleano, cujo valor só pode ser representado pelas palavras reservadas **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.

Exemplo de passagem:

```
Empty exemplo(Int algo[5]) OPEN
    …
```

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: Somente o tipo Int suporta a operação de resto.

3.11. Valores Padrão

Enquanto nenhum valor for explicitamente atribuído a uma variável declarada, ela receberá um valor padrão de acordo com seu tipo, de acordo com o listado abaixo:

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

3.12. Coerção

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

A tabela abaixo descreve a precedência e associatividade dos operadores em ordem decrescente de precedência.

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

+ -	**Aditivos**	**Esquerda->Direita**
!	**Negação**	**Direita->Esquerda**
< > <= >=	**Comparativos**	**Sem associatividade**
== !=	**Igualdade**	**Sem associatividade**
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. Como já citado no tópico **coerções**, tais operações só oferecem suporte a operandos de tipos diferentes caso os tipos em questão sejam Int e Float.

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, disjunção 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. O bloco Else não é obrigatório. Em caso de ausência de tal bloco, apenas será verificada a condição relativa ao bloco **If**. Do contrário, caso exista um bloco **Else**, as instruções desse bloco serão executadas.

5.3 Estruturas Interativas.

5.3.1 Controle Lógico - While

Em Linkin Park implementamos a estrutura While para que sirva 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 lógico 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.

5.3.2 Controle por Contador - For

Em Linkin Park, a estrutura **For**, a quantidade de iterações 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(abetura)** e **Close(fechamento)**.

Em uma estrutura iterativa 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 iteração. 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, os comandos 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, de entrada atribui a uma variável determinada de tipo correspondente por meio do comando **Scan**.

O comando Scan possui a seguinte estrutura: Scan(<Variável>). O valor de entrada fornecido pelo usuário será atribuído a variável passada para o Scan, tal valor deve, necessariamente, ser do mesmo tipo que a variável. Isto significa que se a variável é do tipo Int, por exemplo, o valor fornecido pelo usuário deve ser uma sequência de dígitos.

5.4.2 Saída

Os resultados referentes a um programa em especifico são exibidos na tela pelo comando **Print**. Cadeias de caracteres podem ser passadas para o comando **Print** sem, necessariamente, serem declaradas com antecedência, ou seja, na forma de constantes literais.

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

A estrutura da declaração de uma função em LinkinPark foi explicada e exemplificada no tópico 2. 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 Void limited_fibonacci(Int x) Open
     i = 1
     While (fibonacci(i) < x) Open
          Print(fibonacci(i));
          i = i + 1;
     Close

     Back;
Close

Function Int Main ( ) Open
     Int n, total;
     Print('Insira um valor limite: ');
     Scan(n);

     limited_fibonacci(n);

     Back 0;
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 h;

    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 0;
Close
```