

Especificação da Linguagem

Grupo 12, LFA

Junho 2018

Conteúdo

1	Estruturas de dados (e tipos de variáveis)	2
1.1	Sequências	2
1.1.1	Notas musicais	2
1.1.2	Duração duma nota	3
1.1.3	Acordes	3
1.2	Performances	3
1.3	Marcos de Tempo	4
1.3.1	Funções auxiliares	4
1.4	Números	4
1.5	Instrumentos	4
1.6	Arrays	5
2	Geração de áudio	6
2.1	Reprodução	6
2.2	Modos de reprodução	6
2.2.1	Simultâneo	6
2.2.2	Sequencial	7
2.2.3	Usando Marcos de Tempo	7
2.2.4	Repetição	8
2.3	Uso de arrays	8
2.4	Modulações	9
2.4.1	de Tom	9
2.4.2	de Tempo	10
3	Interação com o exterior	10
3.1	Estruturas de dados auxiliares	10
3.1.1	Strings	10
3.2	getInt(string?)	10
3.3	getSequence(string?)	11

4	Controlo de fluxo	11
4.1	Instruções condicionais	11
4.1.1	if	11
4.1.2	with ... choose	11
4.2	Intruções de repetição	11
4.2.1	for	11
5	Configurações (ficheiro auxiliar)	12
5.1	BPM (Beats Per Minute)	13
5.2	Especificação de (novos) instrumentos	13
6	Exemplos (TODO)	13
6.1	Parabéns	13
6.2	Looping machine like thingy?	13
6.3	Mais exemplos	14

1 Estruturas de dados (e tipos de variáveis)

1.1 Sequências

Uma sequência de notas e silêncios pode ser definida, através da palavra chave `sequence`, como:

```
1 sequence melody = [C C G G A A G R F F E E D D C R]; // os espacos
    sao opcionais
```

1.1.1 Notas musicais

A notas são identificadas pela sua letra:

- C para Dó,
- D para Ré,
- E para Mi,
- F para Fá,
- G para Sol,
- A para Lá,
- B para Si.

A letra **R** é usada para silêncios (derivada de *Rests*).

Um cardinal (**#**) a seguir à letra sobe a respetiva nota por meio tom; um b minúsculo (**b**) a seguir à letra desce a respetiva nota por meio tom. Mais que um **#** ou **b** podem ser aplicados a uma nota (por exemplo, **C##**).

Pode, depois do tom, aparecer um número, entre -2 e 8. Este especifica a oitava a usar, sendo -2 a mais grave e 8 a mais aguda. Por omissão, a quarta oitava é usada.

Assim, a sequência já apresentada podia reescrita da seguinte forma:

```
1 sequence melody = [C4 B# G G4 A4 A G R F E#4 Fb4 E D D4 C4 R];
```

1.1.2 Duração duma nota

Por omissão, uma nota demora um tempo¹. A sua duração depende do *tempo*, ou BPM², da música. Esta configuração é explorada na secção 5.

Uma nota pode, no entanto, tocar mais ou menos tempo. Há duas formas de especificar a duração duma dada nota:

1. **Por extensão.** A duração da nota é especificada através de chavetas (**{** e **}**), relativamente à duração unitária. Por exemplo, **C{4}** demora o quádruplo do tempo de **C**, e **C{0.75}** demora três quartos do tempo de **C**.
2. **Notação simplificada.** Utilizam-se apóstrofes para reduzir a duração duma nota em metade, havendo uma correspondência direta com a duração das notas musicais convencionadas³. Por exemplo, **C'** demora metade do tempo de **C**, e **C''** apenas um quarto.

1.1.3 Acordes

O símbolo **|** é usado para tocar várias notas em simultâneo, numa só sequência. Para tocar o acorde de Dó maior (**C**, **E** e **G**), na quarta oitava, podíamos então escrever:

```
1 sequence intro = [C|E|G];
```

1.2 Performances

A associação duma sequência musical com um instrumento representa um terceiro tipo de dados, uma **performance**:

```
1 sequence twinkle = [CC GG AA G{2} FF EE DD C{2}];
2 performance p = twinkle on guitar;
3
4 // ou, alternativamente, definindo a sequencia implicitamente
5 performance p = [CC GG AA G{2} FF EE DD C{2}] on guitar;
```

¹Em termos musicais, uma semínima.

²*Beats Per Minute*

³Semínima (duração de 1), colcheia (duração de 1/2), semicolcheia (duração de 1/4), etc.

1.3 Marcos de Tempo

Um marco de tempo é uma variável que referencia um dado momento entre o início e o fim, inclusivé, da peça musical. **start** é um marco de tempo pré-definido, representando o início da peça.

1.3.1 Funções auxiliares

A função auxiliar **duration**, aceita um parâmetro, do tipo **sequence** ou **performance**, e devolve um **time** igual à sua duração.

```
1 sequence intro = [R{4} C{4} G{4} C5{3.5} E|G|C5{.5} Eb|G|C5{8} C{4}  
    G{4}]; // Strauss – Also Sprach Zarathustra – Intro (https://www.8notes.com/scores/7213.asp)  
2 time endIntro = start + duration(intro);
```

1.4 Números

Números (inteiros ou reais) são também suportados:

```
1 number num = 4;
```

1.5 Instrumentos

Para tocar sequência musical é, naturalmente, necessário especificar que instrumento deve ser utilizado. Assim, para tocar *Twinkle, Twinkle, Little Star* com uma piano, teríamos:

```
1 // definir a sequencia  
2 sequence twinkle = [CC GG AA G{2} FF EE DD C{2}];  
3  
4 // utilizando performances  
5 performance p = twinkle on piano;
```

Vários instrumentos podem ser usados para tocar uma dada sequência. No entanto, nem todos suportam o mesmo registo (por exemplo, um piano suporta uma gama maior de notas que um violino). Se a um instrumento é dada uma sequência que este não suporta, as notas não suportadas são substituídas pela nota mais próxima que é suportada.

Os instrumentos disponíveis são os seguintes:

- piano;
- guitar;
- violin;
- cello;
- bass;
- drums.

A criação de novos instrumentos é suportada, num ficheiro externo de tipo auxiliar, estando detalhada na secção 5. No ficheiro de tipo principal, não é possível definir ou redefinir novos instrumentos, sendo apenas possível usá-los como constantes pré-definidas.

1.6 Arrays

Um array é uma coleção de várias instâncias da mesma estrutura de dados. Suportam-se arrays de sequências, instrumentos e performances.

Um array pode ser definido de duas formas distintas:

```

1 // criar um array com 4 instrumentos – e necessario usar a palavra
  chave instrument para indicar que e um array de instrumentos
2 instrument [] band = [piano , guitar , bass , drums];
3
4 // outra forma de definir o mesmo array
5 instrument [] band = piano and guitar and bass and drums;
6
7 // criar um array com 3 performances
8 performance [] p = [
9     [D{1.5} D{0.5}    E  D G F#{2}] on piano ,
10    [D{1.5} D{0.5}    E  D A G{2}] on bass ,
11    [D{1.5} D{0.5}    D5 B G F# E] on guitar];

```

Pode usar-se um array para criar outro. A palavra chave **and** anexa ao fim do array já existente o elemento ou elementos dados. A palavra chave **except** retira o elemento ou elementos dados do array, se lá estiverem presentes.

```

1 instrument [] band = [piano , guitar , bass , drums];
2
3 // definir arrays a partir de outros arrays
4 instrument [] new_band = band and violin;
5 instrument [] left_over_band = band except bass;

```

Para se aceder a um instrumento, a notação de parênteses retos, começando a contar no 0, é usada. A notação de intervalo, com dois pontos (:) é também suportada.

```

1 performance [] p = [
2     [D{1.5} D{0.5}    E  D G F#{2}] on piano ,
3     [D{1.5} D{0.5}    E  D A G{2}] on bass ,
4     [D{1.5} D{0.5}    D5 B G F# E] on guitar];
5
6 // aceder ao primeiro elemento
7 performance intro = p[0];
8
9 // exemplo da notacao de intervalo
10 performance [] q = p[0:1] and [C5{1.5} C5{0.5} B G A G{3}] on violin
    and p[2:];

```

2 Geração de áudio

2.1 Reprodução

Para reproduzir uma performance, utiliza-se a palavra chave **play**:

```
1 // definir uma performance
2 sequence twinkle = [CC GG AA G{2} FF EE DD C{2}];
3 performance p = twinkle on piano;
4
5 // reproduzir a performance
6 play p;
7
8 // alternativamente, definir a performance implicitamente
9 play twinkle on piano;
10
11 // ou definir a performance e a sequencia implicitamente
12 play [CC GG AA G{2} FF EE DD C{2}] on piano;
```

2.2 Modos de reprodução

No exemplo anterior, não foi especificado quando começar a tocar a sequência. Por omissão, a sequência começa a ser tocada no início da peça (por outras palavras, no tempo 0). Este tempo também pode ser obtido através da palavra chave **start**.

Averiguemos os diferentes modos de reprodução:

2.2.1 Simultâneo

Por omissão, todas as sequências são tocadas começando no tempo 0, ou **start**. Se há mais que uma sequência a ser tocada, todas as sequências são tocadas em paralelo.

```
1 play [CC GG AA G{2} FF EE DD C{2}] on piano;
2
3 // e equivalente, em termos do som produzido no ficheiro final, a
4 play [CC RR RR G{2} FR RE DR C{2}] on piano;
5 play [RR GG AA RR RF ER RD RR] on piano;
```

Um outro exemplo de reprodução simultânea é o seguinte, que separa melodia e harmonia em duas performances diferentes:

```
1 // definir sequencias
2 sequence twinkle = [CC GG AA G{2} FF EE DD C{2}];
3 sequence twinkle_bass = [C|E|G{2} F|A|C{2} C|E|G{4} F|A|C C|E|G G|B
4   |D C|E|G];
5
6 // tocar performances
7 play twinkle on guitar;
8 play twinkle_bass on guitar;
```

2.2.2 Sequencial

A palavra chave **after** indica que uma dada performance deve começar imediatamente após o fim doutra.

```
1 // definir sequencias
2 sequence first_line = [CC GG AA G{2}];
3 sequence second_line = [FF EE DD C{2}];
4
5 // definir performances
6 performance first_line = twinkle on guitar;
7 performance second_line = twinkle_bass on guitar;
8
9 // tocar performances
10 play first_line;
11 after first_line play second_line;
```

Num exemplo mais avançado, onde a sequência de referência (no exemplo acima, `first_line`) passada a `after` é tocada mais que uma vez, pode ser especificado após que performances deve a sequência alvo (no exemplo acima, `second_line`) ser tocada. Por omissão, a sequência alvo é tocada apenas 1 vez, após a primeira reprodução da sequência de referência.

Para obter outros comportamentos, a palavra chave **always** pode ser utilizada.

```
1 // definir sequencias
2 sequence first_line = [CC GG AA G{2}];
3 sequence second_line = [FF EE DD C{2}];
4
5 // definir performances
6 performance first_line = twinkle on guitar;
7 performance second_line = twinkle_bass on guitar;
8
9 // tocar performances
10 play first_line;
11 after first_line always play second_line;
12 after second_line play first_line;
13 // toca first_line (FL), seguido de second_line (SL), e repete uma
    vez, ou seja, FL, SL, FL, SL
```

2.2.3 Usando Marcos de Tempo

A palavra chave **at** especifica um Marco de Tempo específico no qual a performance deve começar, independentemente de haver outras performances a decorrer nesse momento.

```
1 performance verse = [CC E{2} GG B{2} C5C5 GG C{4}] on violin;
2 performance chorus = [GAGA ABBA GEGE EBBE] on violin;
3
4 time chorusStart = start + duration(verse);
5
6 // tocar performances
7 play verse;
8 at chorusStart play chorus;
```

Uma performance pode ser tocada em mais que um momento. Para isso, além da palavra chave **at**, utiliza-se também a palavra chave **and**.

```
1 performance verse = [CC E{2} GG B{2} C5C5 GG C{4}] on violin;
2 performance chorus = [GAGA ABBA GEGE EBBE] on violin;
3
4 time chorusStart = start + 2*duration(verse);
5 time otherTimeChorusStarts = start + 3.14*duration(verse);
6
7 // tocar performances (chorus e tocada 2 vezes)
8 play verse;
9 at chorusStart and otherTimeChorusStarts play chorus;
```

2.2.4 Repetição

Há duas palavras chaves que permitem a repetição: o uso da palavra chave **times** permite repetir uma performance 0 ou mais vezes. **loop** permite repetir uma performance até ao fim da peça ($\text{loop} = \text{play} \infty \text{times}$).

```
1 performance verse = [CC E{2} GG B{2} C5C5 GG C{4}] on violin;
2 performance chorus = [GAGA ABBA GEGE EBBE] on violin;
3 performance bass = [G|B|D{4} A|D|F#{4} G|B|D{4} A|D|F#{4}] on bass;
4
5 time chorusStart = start + 2*duration(verse);
6
7 // tocar performances
8 play verse 2 times on piano;
9 at chorusStart play chorus;
10
11 loop bass; // repete ate ao fim da musica
```

2.3 Uso de arrays

Um array pode ser reproduzido de forma simultânea (forma utilizada por omissão) ou sequencialmente (através do uso da palavra chave **sequentially**).

```
1 // exemplo com sequencias
2 sequence[] melody_lines = [
3     [D{1.5} D{0.5} E D G F#{2}],
4     [D{1.5} D{0.5} E D A G{2}],
5     [D{1.5} D{0.5} D5 B G F# E],
6     [C5{1.5} C5{0.5} B G A G{3}]];
7
8 play melody_lines /*sequentially*/ on piano; // descomentar
9     sequentially para tocar melody_lines de forma sequencial
```

```
1 // exemplo com instrumentos
2 instrument[] band = [piano, guitar, bass, drums];
3
4 play melody_lines[0] /*sequentially*/ on band;
```

```
1 // exemplo com performances
2 performance[] perfors = [
3     [D{1.5} D{0.5} E D G F#{2}] on piano,
```



```

4      [D{1.5} D{0.5}    E D A G{2}] on bass ,
5      [D{1.5} D{0.5}    D5 B G F# E] on guitar];
6
7 play perfors /*sequentially*/;

```

No entanto, não é possível reproduzir um array de seqüências num array de instrumentos. Isto é, por exemplo, não é possível fazer o seguinte:

```

1 // definir array de sequencias
2 sequence[] melody_lines = [
3     [D{1.5} D{0.5}    E D G F#{2}],
4     [D{1.5} D{0.5}    E D A G{2}],
5     [D{1.5} D{0.5}    D5 B G F# E],
6     [C5{1.5} C5{0.5} B G A G{3}]];
7
8 // definir array de instrumentos
9 instrument[] band = [piano, guitar, bass, drums];
10
11 play melody_lines/*sequentially*/ on band; // ilegal!!

```

Para se obter este tipo de resultados, têm que ser utilizadas instruções de repetição (ver secção 4).

2.4 Modulações

Podem obter-se versões modificadas de seqüências ou performances através das funções de modulação. Estas funções devolvem uma nova seqüência ou performance, alterada em algum aspeto (tom ou tempo) em relação a uma dada seqüência ou performance original, respetivamente.

2.4.1 de Tom

Pode mudar-se o tom de uma dada seqüência ou performance de duas formas diferentes: `changeTone()` e `changeOctave()`. `changeTone()` aceita dois parâmetros - a seqüência ou performance a alterar, e quantos meios-tons⁴ deve subir ou descer; `changeOctave()` aceita, de forma semelhante, dois parâmetros - a seqüência ou performance a alterar, e quantas oitavas deve subir ou descer.

```

1 // sequencia original
2 sequence s = [D{1.5} D{0.5} E D G F#{2}];
3
4 sequence s1 = changeOctave(s, -3); // diminuir a oitava por 3
5 // equivalente a dizer:
6 //     sequence s1 = [D1{1.5} D1{0.5} E1 D1 G1 F#1{2}];
7
8 // mudar oitava da sequencia
9 sequence s5 = changeOctave(s, 1); // aumentar a oitava por 1
10 // equivalente a dizer:
11 //     sequence s5 = [D5{1.5} D5{0.5} E5 D5 G5 F#5{2}];
12 // ou a dizer:
13 //     sequence s5 = changeTone(s, 12);

```

⁴Doze meios-tons constituem uma oitava.

```

1 // performance original
2 performance p = [D{1.5} D{0.5} E D G F#{2}] on bass;
3
4 play changeTone(p, 5);
5 // equivalente a:
6 //   play [F#{1.5} F#{0.5} G# F# B A#{2}] on bass;

```

2.4.2 de Tempo

Pode mudar-se o tempo (ou seja, a velocidade) de uma dada sequência ou performance através da função `changeTempo()`. Esta função aceita dois parâmetros - a sequência ou performance a alterar, e um número maior que 0 que representa a relação entre o novo tempo e o tempo original. A nova velocidade é dada por `fator * tempo original`.⁵

```

1 // performance original
2 performance p = [D{1.5} D{0.5} E D G F#{2}] on bass;
3
4 play changeTempo(p, 5); // toca a sequencia 5x mais rapido (cada
   nota dura 1/5 do seu tempo original)
5 // equivalente a:
6 //   play [D{.3} D{0.1} E{.2} D{.2} G{.2} F#{.4}] on bass;

```

```

1 // performance original
2 performance p = [D{1.5} D{0.5} E D G F#{2}] on bass;
3
4 play changeTempo(changeTone(p, 5), 5); // toca a sequencia 5x mais
   rapido, com todas as notas 5 meios-tons mais agudas
5 // equivalente a:
6 //   play [F#{.3} F#{0.1} G#{.2} F#{.2} B{.2} A#{.4}] on bass;

```

3 Interação com o exterior

3.1 Estruturas de dados auxiliares

3.1.1 Strings

Strings são sequências de caracteres, números e símbolos delimitadas por aspas ("). Dentro duma string, aspas podem ser escapadas através de `\`.

Não existe um tipo de dados String explícito, sendo este usado apenas como parâmetro opcional para funções de I/O.

3.2 `getInt(string?)`

`getInt()` permite obter um inteiro através do *Standard In*.

Opcionalmente, pode ser passada uma String, que será impressa antes de aguardar a resposta do utilizador (uma String de *prompt*).

⁵ Assim, um número maior que 1 acelera a velocidade, e um número menor que 1 reduz a velocidade.

3.3 getSequence(string?)

À semelhança de `getInt()`, `getSequence()` permite obter uma sequência através do *Standard In*.

Opcionalmente, pode ser passada uma *String*, que será impressa antes de aguardar a resposta do utilizador (uma *String* de *prompt*).

4 Controlo de fluxo

4.1 Instruções condicionais

4.1.1 if

As palavras chave `if`, `else if` e `else` permitem testar condições. Os operadores suportados numa condição são os de igualdade (`==`), desigualdade (`!=`), menor (`<`), maior (`>`), menor ou igual (`<=`), e maior ou igual (`>=`).

```
1 sequence s = getSequence("Enter a sequence: ");
2
3 if duration(s) > 5:
4     play s on piano;
5 else if duration(s) > 2:
6     play s on cello;
7 else:
8     play s on guitar;
```

4.1.2 with ... choose

As palavras chave `with`, `choose`, `case` e `default` permitem testar a igualdade entre uma dada variável e um conjunto de valores.

```
1 with getInt("Pick a number: ") choose:
2     case 1:
3         play seq on violin;
4     case 2:
5         play seq on guitar;
6     default:
7         play seq on piano;
```

4.2 Instruções de repetição

4.2.1 for

As palavras chave `for` e `in` permitem definir instruções de repetição, ou seja, permitem que um dado código seja executado múltiplas vezes, iterando sobre todos os elementos de um dado array.

```
1 // instrumentos
2 instrument[] band = [piano, guitar, bass, drums];
3 for instrument inst in band:
4     play [ABCDCEBA] on inst;
5
```

```

6 // sequencias
7 sequence [] sequences = [
8     [D{1.5} D{0.5} E D G F#{2}],
9     [D{1.5} D{0.5} E D A G{2}],
10    [D{1.5} D{0.5} D5 B G F# E],
11    [C5{1.5} C5{0.5} B G A G{3}]];
12
13 for sequence seq in sequences:
14     play seq on piano;
15
16 // performances
17 performance [] perfors = [
18     [D{1.5} D{0.5} E D G F#{2}] on piano,
19     [D{1.5} D{0.5} E D A G{2}] on bass,
20     [D{1.5} D{0.5} D5 B G F# E] on guitar];
21
22 time t = start;
23 for performance perfor in perfors:
24     at t play perfor;
25     t = t + duration(perfor);
26
27 // marcos de tempo
28 start_times = [start, 1, 3, 7];
29 for time t in start_times:
30     at t play [C1 E1 G1 E1] on piano;

```

Caso se pretenda iterar sobre algum código um dado número de vezes, sem haver correspondência direta entre esse número e o conteúdo de um dado array, pode usar-se números num dado intervalo, usando a função `range(a,b)`, que devolve um array de inteiros de tipo `[a, a+1, ..., b-2, b-1]`:

```

1 instrument [] band = [piano, guitar, bass, drums];
2 sequence [] sequences = [
3     [D{1.5} D{0.5} E D G F#{2}],
4     [D{1.5} D{0.5} E D A G{2}],
5     [D{1.5} D{0.5} D5 B G F# E],
6     [C5{1.5} C5{0.5} B G A G{3}]];
7
8 time t = start;
9 for number i in range(0:4):
10     at t play sequences[i] on band[i];
11     t = t + duration(sequences[i]);

```

5 Configurações (ficheiro auxiliar)

Um ficheiro do tipo principal (ficheiros com extensão `.principal`) suporta um todas as operações descritas até este ponto. Várias configurações podem ser feitas no ficheiro auxiliar (ficheiros com extensão `.auxiliar`). Nesta secção, vamos abordar as diferentes configurações que podem ser definidas através do ficheiro auxiliar.

5.1 BPM (Beats Per Minute)

BPM é uma palavra reservada⁶ usada para configurar o *tempo* da música. A configuração deve ser feita no ficheiro de configuração, mas é possível reescrevê-la no ficheiro principal (o que define a música a gerar).

```
1 BPM = 160;
```

5.2 Especificação de (novos) instrumentos

O formato *midi* suporta 128 instrumentos diferentes⁷.

É possível criar novos instrumentos associando a uma palavra um número, através da palavra chave `instrument`. O nome dado ao instrumento deixa de poder ser usado como nome de variável.

```
1 // definir dois novos instrumentos
2 instrument strings = 49;
3 instrument synth1 = 81;
```

Os instrumentos já definidos estão associados aos seguintes códigos: 1(*piano*), 25(*guitar*), 41(*violin*), 43(*cello*), 44(*bass*), e 119(*drums*).

6 Exemplos (TODO)

6.1 Parabéns

parabens.auxiliar

```
1 // set default BPM setting
2 BPM = 160;
```

parabens.principal

```
1 // get user input
2 int repeat_times = getInt("Number of repetitions: ");
3
4 // define melody sequences
5 sequence[] melody_lines = [
6     [D{1.5} D{0.5} E D G F#{2}],
7     [D{1.5} D{0.5} E D A G{2}],
8     [D{1.5} D{0.5} D5 B G F# E],
9     [C5{1.5} C5{0.5} B G A G{3}]];
10
11 // perform
12 play melody_lines sequentially repeat_times times on piano;
```

6.2 Looping machine like thingy?

⁶BPM não pode ser usado como nome de uma variável.

⁷Para obter mais informação sobre os diferentes instrumentos disponíveis, ver <http://www.ccarh.org/courses/253/handout/gminstruments/>.

```
1 // tipo , add looping track , add looping track , etc .
```

6.3 Mais exemplos