Para separar estes conjuntos, deve-se aplicar, mapear, a função de separar *tokens* pelo caractere vírgula em cada elemento, cada *string*, da lista de compassos. A função, em Clean, que mapeia uma função em uma lista de elementos é a função ***map***, conforme mostrado a seguir:

```
8 #compNotasTempo = map (StringTokens isSepVirgula) compassos
```

***Listagem 3.10** – função map*

O Programa modificado para separar os conjuntos de nota_figura é mostrado na listagem 3.11 a seguir:

```
 1 module musicaTexto
 2 import StdEnv,StdArq
 3
 4 Start :: *World -> ([[{#Char}]],*World)
 5 Start comp
 6 #(conteudo,endereco,comp) = AbrirArquivoTextoExpl comp
 7 #compassos = StringTokens isSepBarraN conteudo
 8 #notaFigura = map (StringTokens isSepVirgula) compassos
 9 =(notaFigura,comp)
10
11 isSepBarraN :: Char -> Bool
12 isSepBarraN  x = '\n'== x
13
14 isSepVirgula:: Char -> Bool
15 isSepVirgula x = ',' == x
16
17
18
19
20
21
```

```
press any key to exit
([["do5 sm","sol5 sm","mi5 m"],["re5 m","fa5   m"],["sol5 c","fa5 c","do5 m","la
#5 sm"]],42)
```

Listagem 3.11 – Separando Notas com figuras do compasso

Observe que a declaração de tipo da função **Start** muda novamente, já que, agora, tem-se como resposta uma lista de compassos, onde cada compasso é uma lista de *strings* nota_figura, ou seja:

Start :: *World -> (**[[{#Char}]]**,*World),

Observe que cada elemento do compasso é uma string representando um conjunto nota_figura, ou seja:

notasFiguras = [notaFigura 1, notaFigura 2, notaFigura 3]

- **notaFigura 1** = ["do5 sm", "sol5 sm", "mi5 m"]
- **notaFigura 2** = ["re5 m", "fa5   m"]
- **notaFigura 3** = ["sol5 c", "fa5 c", "do5 m", "la#5 sm"]

Quarto passo: Já se tem separada a música em compassos e os compassos em conjunto de *string* nota_acorde. Necessita-se, agora, separar as notas musicais de suas respectivas figuras. Para tanto, deve-se acessar cada *string* nota_acorde, dentro da lista de

compassos, dentro da lista música. Esta ação, de relativa complexidade, envolvendo *loops* dentro de *loops* em linguagens procedurais, pode ser feita com simplicidade utilizando novamente a notação Zermelo-Fraenkel, a qual permite que se aplique uma função ou regra em cada elemento de um domínio. No caso, o domínio será o de uma lista de compassos de uma música. Assim, se pega um compasso da música e se aplica (mapeia) em cada elemento do compasso a função de separar tokens pelo caractere espaço.

```
16 isCaracEspaco:: Char -> Bool
17 isCaracEspaco x= ' ' == x
```

***Listagem 3.12** – Função de separar tokens*

O elemento nota_figura é um elemento da lista musical no formato nota_figura obtida no passo anterior. Assim, a linha de programa que cria a nova estrutura da música com os conjuntos nota_figura separados fica:

```
9 #notaFigSeparadas = [map (StringTokens isCaracEspaco) x \\ x<-notaFigura]
```

***Listagem 3.13** – Cria nova estrutura da música*

Onde x é um elemento de ***notaFigura***, como, por exemplo: ***"do5 sm"***. Ao aplicar: ***map (StringTokens isCaracEspaco) "do5 sm,*** tem-se a lista ***["do5","sm"]***.

A listagem 3.14 mostra o programa completo que devolve a lista ***notaFigSeparadas***.

```
1  module musicaTexto
2  import StdEnv,StdArq
3
4  Start :: *World -> ([[[{#Char}]]],*World)
5  Start comp
6  #(conteudo,endereco,comp) = AbrirArquivoTextoExpl comp
7  #compassos = StringTokens isSepBarraN conteudo
8  #notaFigura = map (StringTokens isSepVirgula) compassos
9  #notaFigSeparadas = [map (StringTokens isCaracEspaco) x \\ x<-notaFigura]
10 =(notaFigSeparadas,comp)
11
12 isSepBarraN :: Char -> Bool
13 isSepBarraN  x = '\n'== x
14
15 isSepVirgula:: Char -> Bool
16 isSepVirgula x = ',' == x
17
18 isCaracEspaco:: Char -> Bool
19 isCaracEspaco x= ' ' == x
20
21   press any key to exit
22
23   ([[["do5","sm"],["sol5","sm"],["mi5","m"]],[["re5","m"],["fa5","m"]],[["sol5","c
24   "],["fa5","c"],["do5","m"],["la#5","sm"]]],42)
25
```

***Listagem 3.14** – Música com todos os elementos separados em listas*

Observe, que a declaração de tipo da função **Start** muda novamente, já que, agora, tem-se como resposta uma lista de compassos, onde cada compasso é uma lista que contém listas de notas e figuras, ou seja:

Start :: *World -> ([[[**{#Char}**]]],*World),

Observe que cada elemento do compasso é uma *string* representando um conjunto nota_figura, ou seja:

notasFigurasSeparadas = [notaFigSeparadas 1, notaFigSeparadas 2, notaFigSeparadas 3]

- **notaFigSeparadas 1** = [["do5”,”sm"], ["sol5”,”sm"], ["mi5”,”m"]]
- **notaFigSeparadas 2** = [["re5”,”m"], ["fa5”,”m"]]
- **notaFigSeparadas 3** = [["sol5”,”c"], ["fa5”,”c"], ["do5”,”m"], ["la#5”,”sm"]]

Quinto passo: De posse do arquivo texto convertido em uma estrutura de listas, onde cada elemento da música foi identificado e separado corretamente, deseja-se, finalmente, transpor as notas musicais, ou seja, o primeiro elemento da lista de nota_figura. Novamente, esta é uma tarefa ainda mais complexa, já que se deve modificar o primeiro elemento de cada lista de nota_figura, a qual está dentro de uma lista de compasso, a qual está dentro da lista da música. O que demandaria um programa recursivo com vários *loops*, pode ser novamente resolvido com uma notação Zermelo-Fraenkel, apenas se mapeando a função de transposição nos elementos da lista ***notaFigSeparadas*** obtida no passo anterior.

```
10 #musicaTransposta = [map transpoe x \\ x<-notaFigSeparadas]
```

Listagem 3.15 *– Função map e Zermelo Fraenkel*

Para tanto, deve-se implementar a função ***transpoe***, a qual é mostrada a seguir e, explicada passo a passo.

```
26 transpoe :: [{#Char}] -> [{#Char}]
27 transpoe notaFig=:[not,fig] =[notaTransposta,fig]
28 where
29    indiceNota      = (indice not)
30    novoIndice      = indiceNota + numeroDeSemiTons
31    notaTransposta  = notas!!novoIndice
```

Listagem 3.16 *– Função transpoe*

Para facilitar o acesso ao elemento nota da lista de notas_figuras, utilizar-se-á o recurso de declarar a estrutura do argumento de uma função antes de aplicar a regra. Observe na função ***transpoe*** que se tem uma forma diferente de declarar o argumento:

notaFig =:[not,fig]

ao declarar desta forma, a estrutura que vem após os sinais ***=:*** é a estrutura do argumento. No caso, tem-se declarado que o argumento ***notaFig*** é uma lista que contém dois elementos. Na declaração do tipo da função, tem-se que cada um destes elementos é uma *string* ({#Char}), já que o argumento é uma lista de *strings*:

transpoe :: [{#Char}] -> [{#Char}]

Assim, a regra da função é devolver uma nova lista com dois elementos, colocando no primeiro elemento a ***notaTransposta*** e mantendo o segundo elemento (***fig***) sem alteração.

```
transpoe notaFig=:[not,fig] =[notaTransposta,fig]
```

***Listagem 3.17** – Regra da função "transpoe"*

Para completar esta função, basta implementar como obter o elemento ***notaTransposta***.

Para transpor a nota musical, o primeiro elemento da lista de nota_figura, deve-se proceder as seguintes ações:

1- Obter o índice da nota original na lista de notas musicais. Isto é obtido aplicando uma função que, a partir da string contendo o nome da nota musical

   - indiceNota = (indice not)

   A função que devolve o índice da nota é:

```
indice :: {#Char} -> Int
indice nota = hd [ind \\nt<-notas & ind<-(indexList notas) | nt == nota]
```

   ***Listagem 3.18** – Função que devolve o índice da nota*

   Ou seja, pega-se um domínio contendo os índices de todas as notas musicais (**nt**) da lista ***notas***, mas só se coloca no conjunto solução o índice da nota que se está procurando (**nt == nota**). No final, para se pegar o índice da lista, basta pegar seu primeiro elemento (**hd**).

2- Calcular o índice da nota transposta (***novoIndice***) para que se possa pegar a nota equivalente ao índice na lista ***notas***.

   - novoIndice = indiceNota + numeroDeSemiTons

3- Finalmente, pode-se obter a ***notaTransposta***, bastando pegar a nota na lista ***notas*** que possua o ***novoIndice***.

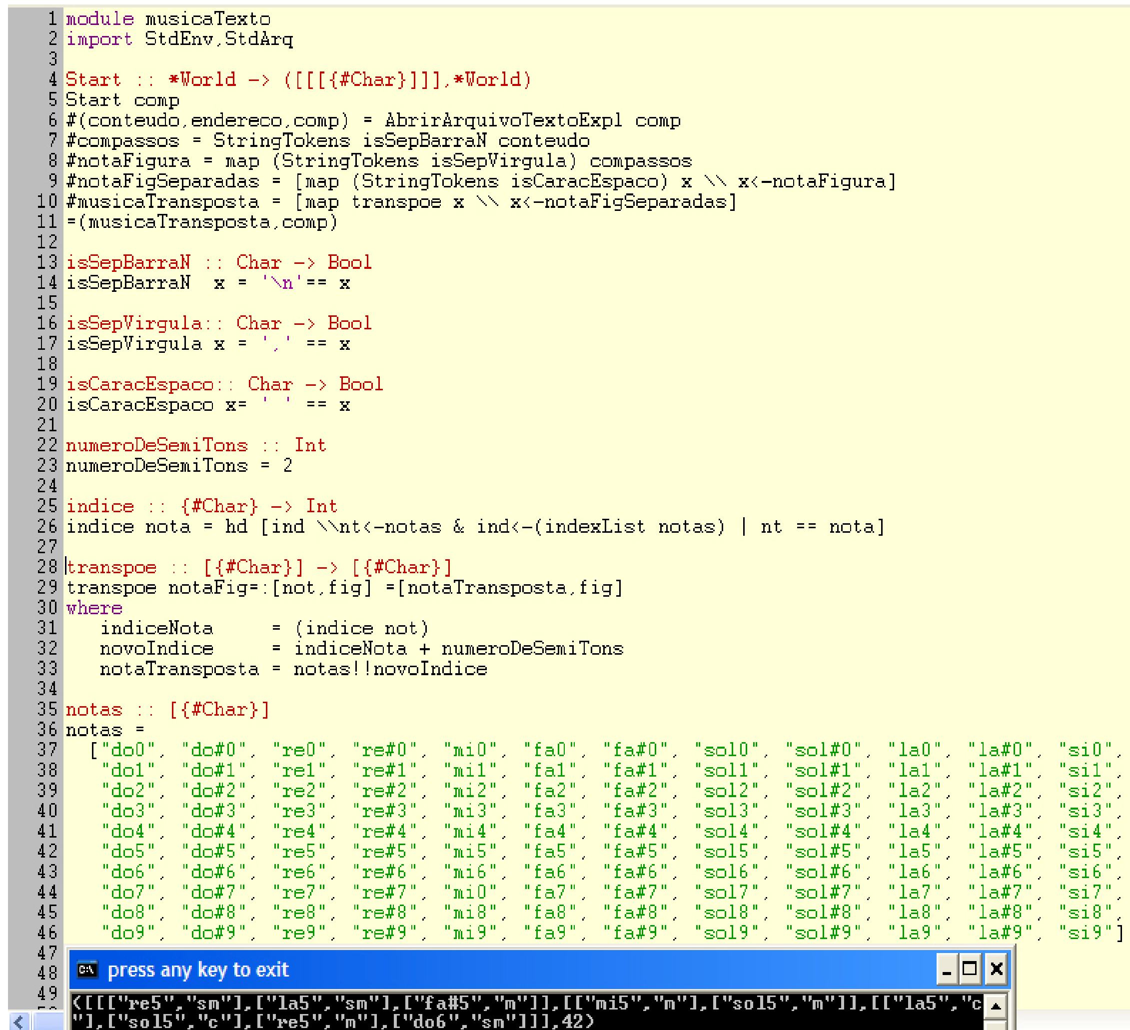
   - **notaTransposta = notas!!novoIndice**

A função ***transpoe*** completa fica, então:

```
26 transpoe :: [{#Char}] -> [{#Char}]
27 transpoe notaFig=:[not,fig] =[notaTransposta,fig]
28 where
29    indiceNota      = (indice not)
30    novoIndice      = indiceNota + numeroDeSemiTons
31    notaTransposta  = notas!!novoIndice
```

***Listagem 3.19** – Função "transpoe" completa*

A listagem 3.20 mostra o programa completo que faz a leitura de uma música em formato texto e a transpõe conforme número de semitons fornecido, no caso, 2 semitons acima.



```
1 module musicaTexto
2 import StdEnv,StdArq
3
4 Start :: *World -> ([[[{#Char}]]],*World)
5 Start comp
6 #(conteudo,endereco,comp) = AbrirArquivoTextoExpl comp
7 #compassos = StringTokens isSepBarraN conteudo
8 #notaFigura = map (StringTokens isSepVirgula) compassos
9 #notaFigSeparadas = [map (StringTokens isCaracEspaco) x \\ x<-notaFigura]
10 #musicaTransposta = [map transpoe x \\ x<-notaFigSeparadas]
11 =(musicaTransposta,comp)
12
13 isSepBarraN :: Char -> Bool
14 isSepBarraN  x = '\n'== x
15
16 isSepVirgula:: Char -> Bool
17 isSepVirgula x = ',' == x
18
19 isCaracEspaco:: Char -> Bool
20 isCaracEspaco x= ' ' == x
21
22 numeroDeSemiTons :: Int
23 numeroDeSemiTons = 2
24
25 indice :: {#Char} -> Int
26 indice nota = hd [ind \\nt<-notas & ind<-(indexList notas) | nt == nota]
27
28 transpoe :: [{#Char}] -> [{#Char}]
29 transpoe notaFig=:[not,fig] =[notaTransposta,fig]
30 where
31    indiceNota     = (indice not)
32    novoIndice     = indiceNota + numeroDeSemiTons
33    notaTransposta = notas!!novoIndice
34
35 notas :: [{#Char}]
36 notas =
37   ["do0", "do#0", "re0", "re#0", "mi0", "fa0", "fa#0", "sol0", "sol#0", "la0", "la#0", "si0",
38    "do1", "do#1", "re1", "re#1", "mi1", "fa1", "fa#1", "sol1", "sol#1", "la1", "la#1", "si1",
39    "do2", "do#2", "re2", "re#2", "mi2", "fa2", "fa#2", "sol2", "sol#2", "la2", "la#2", "si2",
40    "do3", "do#3", "re3", "re#3", "mi3", "fa3", "fa#3", "sol3", "sol#3", "la3", "la#3", "si3",
41    "do4", "do#4", "re4", "re#4", "mi4", "fa4", "fa#4", "sol4", "sol#4", "la4", "la#4", "si4",
42    "do5", "do#5", "re5", "re#5", "mi5", "fa5", "fa#5", "sol5", "sol#5", "la5", "la#5", "si5",
43    "do6", "do#6", "re6", "re#6", "mi6", "fa6", "fa#6", "sol6", "sol#6", "la6", "la#6", "si6",
44    "do7", "do#7", "re7", "re#7", "mi0", "fa7", "fa#7", "sol7", "sol#7", "la7", "la#7", "si7",
45    "do8", "do#8", "re8", "re#8", "mi8", "fa8", "fa#8", "sol8", "sol#8", "la8", "la#8", "si8",
46    "do9", "do#9", "re9", "re#9", "mi9", "fa9", "fa#9", "sol9", "sol#9", "la9", "la#9", "si9"]
47
48
49
```


***Listagem 3.20** – Programa para transpor música em formato texto*

Este exemplo é bem abrangente, apresentando alguns conceitos novos e reforçando os já vistos anteriormente, agrupando todos para a implementação de um aplicativo em música.

Para finalizar o básico necessário para se implementar aplicativos visuais, falta apenas apresentar um exemplo de como fazer o processo inverso do exemplo anterior, ou seja, tendo uma lista de lista de lista de *strings*, formatá-la para salvar em formato texto.

Para tanto, basta implementar uma função básica que converte uma lista de *string* em uma *string* com um determinado separador em lugar da vírgula. Um exemplo é restituir o separador inicial, o espaço, transformando a lista em uma *string*.

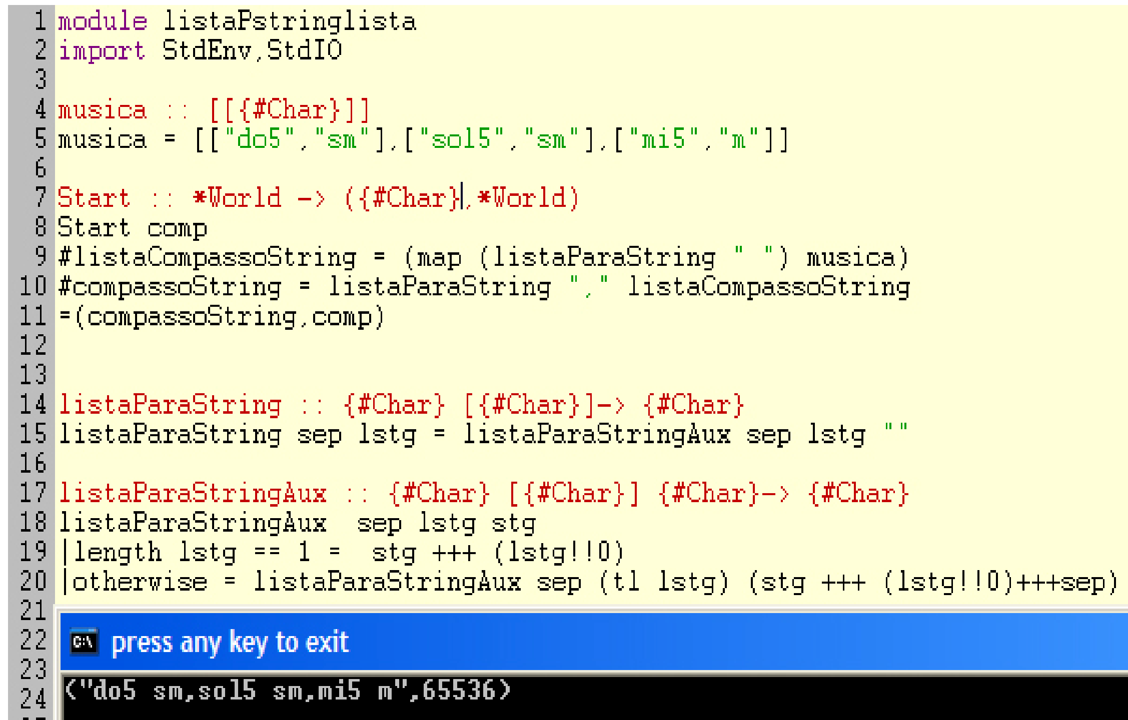
3.4.2 Transformando uma lista de *strings* em uma *string* com um separador entre os elementos da lista pré-especificado.

Para fazer esta transformação, conversão ou *casting* de dados, será implementado, para treinamento de outras estruturas, um programa recursivo tradicional. No mesmo será lida uma lista, pegando cada elemento da lista e acrescentando a uma *string*, inicialmente vazia, este elemento acrescido do separador. Quando a lista estiver vazia se encerra a montagem da *string* e se finda o processo. A listagem 3.21, a seguir, mostra o programa que faz isto.

```
1  module listaPstringlista
2  import StdEnv,StdIO
3
4  musica :: [{#Char}]
5  musica = ["do5","sm"]
6
7  Start :: *World -> ({#Char},*World)
8  Start comp
9  #notaAcordeString = listaParaString " " musica
10 =(notaAcordeString,comp)
11
12
13 listaParaString :: {#Char} [{#Char}]-> {#Char}
14 listaParaString sep lstg = listaParaStringAux sep lstg ""
15
16 listaParaStringAux :: {#Char} [{#Char}] {#Char}-> {#Char}
17 listaParaStringAux  sep lstg stg
18 |length lstg == 1 =  stg +++ (lstg!!0)
19 |otherwise = listaParaStringAux sep (tl lstg) (stg +++ (lstg!!0)+++sep)
20
21 press any key to exit
22
23 ("do5 sm",65536)
24 _
```

***Listagem 3.21** – Convertendo uma lista em uma string e separador*

Observe que a primeira etapa de reversão do processo foi cumprida. Agora, em vez de uma lista nota_figura, adotar-se-á uma lista de compasso, o qual possui várias listas nota_acorde. Assim, o processo se repete, já que se tem, agora, em um compasso, novamente uma lista de *strings* nota_acorde. A listagem 3.22 a seguir, ilustra este processo.



```
1 module listaPstringlista
2 import StdEnv,StdIO
3
4 musica :: [[{#Char}]]
5 musica = [["do5","sm"],["sol5","sm"],["mi5","m"]]
6
7 Start :: *World -> ({#Char},*World)
8 Start comp
9 #listaCompassoString = (map (listaParaString " ") musica)
10 #compassoString = listaParaString "," listaCompassoString
11 =(compassoString,comp)
12
13
14 listaParaString :: {#Char} [{#Char}]-> {#Char}
15 listaParaString sep lstg = listaParaStringAux sep lstg ""
16
17 listaParaStringAux :: {#Char} [{#Char}] {#Char}-> {#Char}
18 listaParaStringAux  sep lstg stg
19 |length lstg == 1 =  stg +++ (lstg!!0)
20 |otherwise = listaParaStringAux sep (tl lstg) (stg +++ (lstg!!0)+++sep)
21
22 press any key to exit
23
24 ("do5 sm,sol5 sm,mi5 m",65536)
```

***Listagem 3.22** – Convertendo uma lista de lista de strings para string*

Observe que o primeiro passo foi converter a lista de lista de *string* para uma lista de *string*, ou seja, transformar o compasso em uma lista de *string*. Isto é feito mapeando a função do exemplo anterior na lista do compasso, ou seja:

```
9 #listaCompassoString = (map (listaParaString " ") musica)
```

***Listagem 3.23** – Função listaCompassoString*

O próximo passo é aplicar na lista de compasso já com elementos *string*, novamente a função ***listaParaString*** agora com o separador vírgula.

```
10 #compassoString = listaParaString "," listaCompassoString
```

***Listagem 3.24** – Função CompassoString*

Para finalizar, deve-se implementar uma função que pegue uma lista de compassos e retorne a *string* original do arquivo “acordes.txt” mostrado na figura 3.1.

Para fazer isto, primeiro deve-se aplicar a função anterior a todos os compassos existentes na música. Assim, a linha de programa que faz isto é dada a seguir:

```
11 #listaCompassoString = [map (listaParaString " ") y\\y<- [x\\x<-musica]]
```

***Listagem 3.25** – Retorna a string original da fig. 3.2*

O programa da listagem 3.26 a seguir mostra esta parte do programa rodando.

```
 1 module listaPstringlista
 2 import StdEnv,StdIO
 3
 4 musica :: [[[{#Char}]]]
 5 musica = [ [["do5","sm"],["sol5","sm"],["mi5","m"]] ,
 6            [["re5","m"],["fa5","m"]],
 7            [["sol5","c"],["fa5","c"],["do5","m"],["la#5","sm"]]  ]
 8
 9 //Start :: *World -> ([{#Char}],*World)
10 Start comp
11 #listaCompassoString = [map (listaParaString " ") y\\y<- [x\\x<-musica]]
12 #listaMusicaCompassoString =  map (listaParaString ",")listaCompassoString
13 =(listaMusicaCompassoString,comp)
14
15
16 listaParaString :: {#Char} [{#Char}]-> {#Char}
17 listaParaString sep lstg = listaParaStringAux sep lstg ""
18
19 listaParaStringAux :: {#Char} [{#Char}] {#Char}-> {#Char}
20 listaParaStringAux  sep lstg stg
21 |length lstg == 1 =  stg +++ (lstg!!0)
22 |otherwise = listaParaStringAux sep (tl lstg) (stg +++ (lstg!!0)+++sep)
23
24  press any key to exit
25
26 (["do5 sm,sol5 sm,mi5 m","re5 m,fa5 m","sol5 c,fa5 c,do5 m,la#5 sm"],65536)
27
```

***Listagem 3.26** – Programa transformando lista de lista de lista de compassos em string*

Observe que, agora, já se tem uma lista de *strings*, e, portanto, para gerar um arquivo texto final, basta gerar uma *string* utilizando a função ***listaParaString*** com o separador de saltar linha. Um detalhe antes é importante. O caractere de saltar linha normalmente é formado por dois caracteres:

- um de saltar linha (caractere 13)
- outro de retorno de carro (caractere 10)

Em arquivos texto do Windows, utiliza-se somente o caractere de saltar linha (13). Assim, no programa a seguir, em vez de ‘\n’, será inserido apenas o caractere 13 (***toChar 13***).

O programa final, acrescentando a função ***GravarArquivoTextoExpl*** para salvar a música em formato texto, é mostrado na listagem 3.27 a seguir.

```
1  module listaPstringlista
2  import StdEnv,StdIO,StdArq
3
4  musica :: [[[{#Char}]]]
5  musica = [ [["do5","sm"],["sol5","sm"],["mi5","m"]] ,
6             [["re5","m"],["fa5","m"]],
7             [["sol5","c"],["fa5","c"],["do5","m"],["la#5","sm"]]  ]
8
9  //Start :: *World -> ([{#Char}],*World)
10 Start comp
11 #listaCompassoString = [map (listaParaString " ") y\\y<- [x\\x<-musica]]
12 #listaMusicaCompassoString =  map (listaParaString ",")listaCompassoString
13 #musicaCompassoString = listaParaString {toChar 10} listaMusicaCompassoString
14 #comp = GravarArquivoTextoExpl musicaCompassoString comp
15 =(musicaCompassoString,comp)
16
17
18 listaParaString :: {#Char} [{#Char}]-> {#Char}
19 listaParaString sep lstg = listaParaStringAux sep lstg ""
20
21 listaParaStringAux :: {#Char} [{#Char}] {#Char}-> {#Char}
22 listaParaStringAux  sep lstg stg
23 |length lstg == 1 =  stg +++ (lstg!!0)
24 |otherwise = listaParaStringAux sep (tl lstg) (stg +++ (lstg!!0)+++sep)
25
26 C:\!!!2007 mes a mes\abril\helcio\___FINALIZANDO A DISSERTAÇÃO 20_04_2007\PROGRAM...
27
28 ("do5 sm,sol5 sm,mi5 m
29 re5 m,fa5 m
30 sol5 c,fa5 c,do5 m,la#5 sm",
```

***Listagem 3.27** – Música convertida para formato texto*

A figura 3.2 a seguir mostra o arquivo "acordes2.txt" salvo pelo programa.

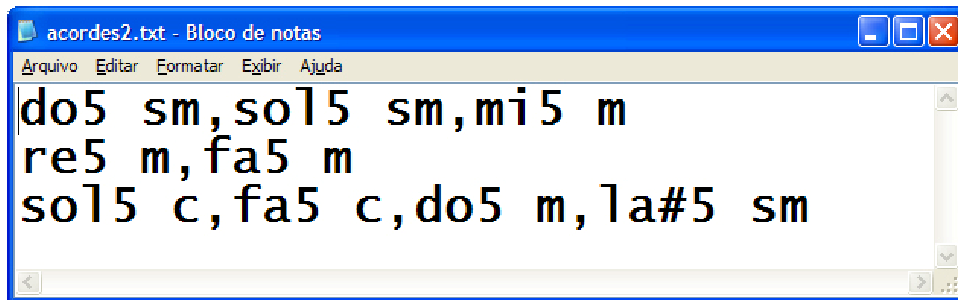


***Figura 3.2** – Arquivo texto equivalente à lista musical*

Que é equivalente ao mostrado na figura 3.1 do arquivo "acordes.txt". mostrada a seguir novamente.

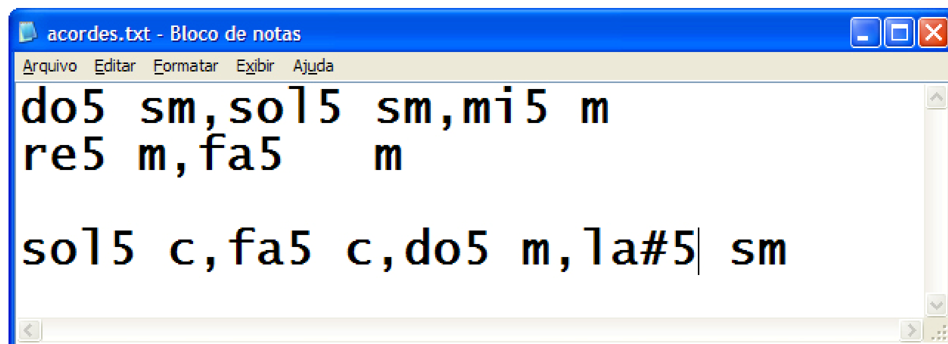


***Figura 3.3** – Mostra o arquivo acordes novamente para comparação*

Observe que os arquivos são os mesmos, a menos do excesso de separadores no compasso 2 e na linha 3, os quais são redundantes.

3.5 Conclusão.

Com os conceitos e aplicativos mostrados neste capítulo, pode-se agora partir para implementações de interfaces visuais com uma base mais sólida, podendo-se focar mais nos detalhes dos elementos da interface do que na programação básica de funções utilizando Tipos Únicos.

Capítulo 4

Bibliotecas para manipulação de arquivos MIDI.

Este capítulo se encarrega de apresentar os conceitos e implementação de bibliotecas destinadas a ler e salvar arquivos MIDI, permitindo, também, manipulá-los e editá-los para implementações de novos aplicativos MIDI. Assim, implementou-se três bibliotecas para esta finalidade:

1. ***separaEventosF0* -** Biblioteca que abre um arquivo MIDI SMF formato 0, separando cada evento em uma lista formatada para facilitar consultas e edição. Esta biblioteca também permite que se toque e salve o arquivo MIDI partindo das listas de eventos.
2. ***geraMIDIF0*** – Biblioteca que permite se criar e salvar arquivos MIDI SMF formato 0 a partir de uma lista de eventos multi-canais, montada na seqüência com que os mesmos foram executados, ou seja, de forma semelhante com que são armazenados por um SMF formato 0. A facilidade que a mesma introduz é se poder entrar eventos com nome de notas e com nomes de figuras musicais e se a nota atual é uma nota solo ou se pertence a um acorde.
3. ***geraMIDIF1*** – Biblioteca que permite se criar e salvar arquivos MIDI SMF formato 1 a partir de uma lista de canais MIDI, onde cada canal possui uma nova lista de eventos similar ao da biblioteca ***geraMIDIF0***.

4.1 Biblioteca *separaEventosF0.*

O fluxograma, a seguir, ilustra a lógica de implementação desta biblioteca.

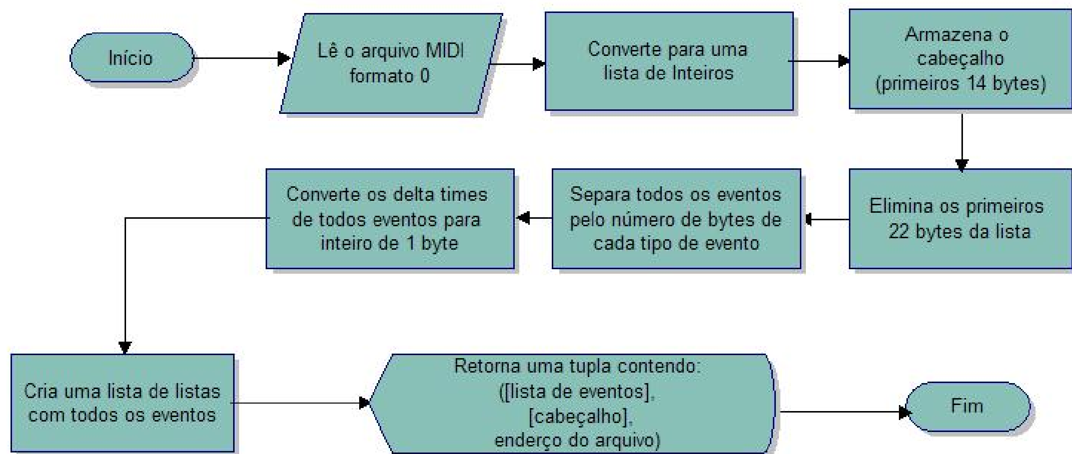


Figura 4.1 *– Fluxograma da biblioteca "separaEventosF0"*

O processo se inicia aplicando a função ***abrirEventosMIDI*** em um arquivo SMF formato 0. Esta função realiza várias ações, mostradas na listagem 4.1, a saber:

- Abre o arquivo (linhas 20 a 25);
- Testa se o mesmo é um arquivo MIDI, para tanto, checa se os quatro primeiros caracteres formam a string "*MThd*" (linha 27);
- Testa se o arquivo é um SMF formato 0. Isto é feito checando o caractere de índice 9 do arquivo aberto (linha 28);
- Converte o mesmo para inteiro (linha 29);
- Retorna uma tupla, onde:
    - O primeiro elemento da mesma é a lista do arquivo aberto e convertido, eliminados os primeiros 22 bytes (cabeçalho *MThd* + MTrk + contagem do *track MTrk* – ver capítulo 2, item 2.11.1 e 2.11.2);
    - O segundo elemento é o cabeçalho geral do arquivo (*MThd*), contendo o formato, número de *tracks* e *ppq*;
    - O terceiro é o endereço do arquivo lido.

```
18 abrirEventosMIDI ::  *World -> ([Int],[Int],{#Char},.World)
19 abrirEventosMIDI proc
20 #  (certo, proc) = selectInputFile proc
21 |  isNothing certo =([],[],"", proc)
22 #  caminho = fromJust certo
23 #(_,file, proc)= fopen caminho FReadData proc
24 #(conteudo1, file)= freads file 1457664
25 #(ok, proc)= fclose file proc
26 #  tm1 = size conteudo1
27 |  (not (conteudo1%(0,3) == "MThd")) = ([1000],[],"",proc)
28 |  (conteudo1.[9]<> toChar 0) = ([2000],[],"",proc)
29 #  listaArqMIDI = [(digitToInt x)+48\\x<-:conteudo1]
30 =  (drop 22 listaArqMIDI,take 14 listaArqMIDI, caminho,proc)
```

Listagem 4.1 *– Função "abrirEventosMIDI"*

Após ler o arquivo MIDI, deve-se chamar a função ***listaEventos*** que se encarregará de montar uma lista com todos os eventos do arquivo separados em listas pré-formatadas da seguinte forma:

[status , tipo , **nBytesOff**, **número de bytes do delta time**, delta-time + evento]

- Na cabeça de cada lista está o código do status do evento da mesma, permitindo-se que se consulte de forma simples de qual evento se trata;
- No segundo elemento da lista vem uma característica do tipo do evento. Se o evento for de ativar nota, o segundo elemento é o código da nota; se o status for de um evento de meta-evento, o segundo elemento é o código, o tipo do meta evento, e assim por diante. Ex: [144,**60**,... significa: ativar nota no canal 0 (144) e que a nota é a **60** (do5);
- O terceiro elemento **indica quantos bytes do arquivo original** foram eliminados no processo de leitura. Este número de bytes algumas vezes não é igual ao número de Bytes da lista de eventos devido ao fato de que a função de criar a lista de eventos incluir o status em eventos com running status. Com este elemento, também se pode, ao salvar novamente o arquivo, modificado ou não, eliminar o byte de status acrescido à lista dos eventos identificados, caso se queira manter esta característica do arquivo original;
- O quarto elemento indica **quantos bytes de delta time** o evento possui. Ex: [144,60, 4, **2**,... significa, que o evento é de ativar nota no canal 0, nota do5, e que o delta time deste evento possui **2** bytes;

- O restante da lista contém a mensagem (o evento) original, com o delta time, status e bytes de dados.

Exemplo: Dados dois eventos consecutivos, do arquivo MIDI aberto pela função ***abrirEventosMIDI***, um evento sem running status e outro com, aplicar a função ***listaEventos*** na mesma.

Trecho do arquivo SMF aberto e convertido para uma lista de inteiros

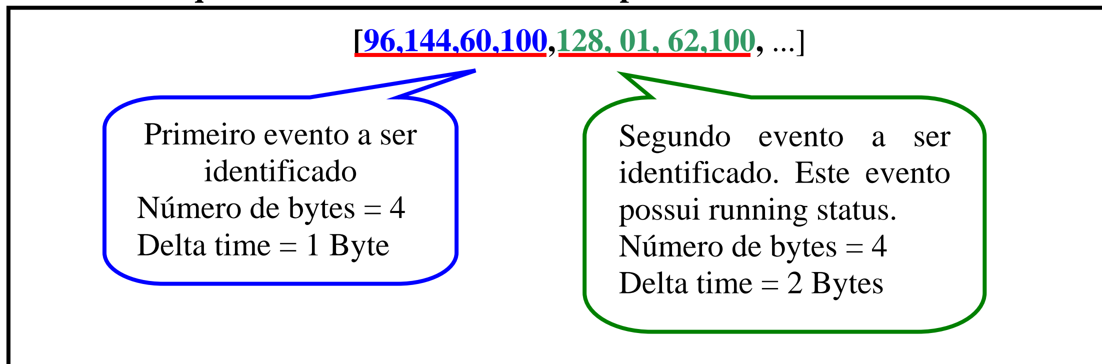


Figura 4.2 *– Trecho do arquivo SMF aberto e convertido para uma lista de inteiros*

Lista de eventos identificada e formatada pela função *listaEventos*

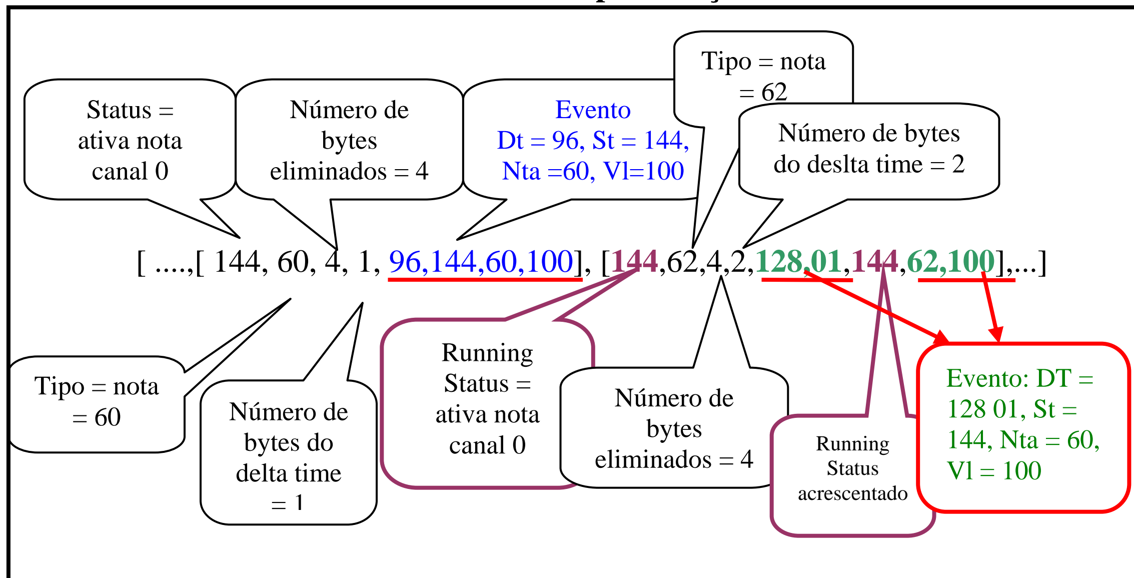


Figura 4.3 *– Lista de eventos identificada e formatada pela função "listaEventos"*

Para identificar cada evento, a função ***listaEventos*** deve reconhecer quantos Bytes de delta time possui o evento, se o mesmo possui ou não running status, qual é o tipo de

evento e quantos Bytes o evento possui. Para tanto deve-se conhecer todos os tipos de eventos MIDI existentes, bem como o número de bytes de cada um.

Alguns eventos possuem número de Bytes fixos (o delta time não está incluído). Os Eventos possíveis possuem um Byte de status com código entre 128 e 255. Os mesmos, quanto ao número de Bytes, são separados como segue:

- **Mensagens com 1 Byte –** Um Byte de Status: Código entre 244 e 254.
- **Mensagens com 2 Bytes –**Um Byte de Status e um Byte de Dado: Código entre 192 a 223, 241 e 243.
- **Mensagem com 3 Bytes -** Um Byte de Status e dois Bytes de Dados: Código entre 128 a 191 , de 224 a 239 e 242.
- **Mensagem com número variável de Bytes –** Dois Bytes de Status, um Byte de número de Bytes e um ou vários Bytes de dados. Tipo Mensagem Exclusiva de Sistema e Meta Eventos. Código 240 ou 255

Ou:

128...191	192...223	224...239	240	241	242	243	244...254	255
3 Bytes	2 Bytes	3 Bytes	Var	2 Bytes	3 Bytes	2 Bytes	1 Bytes	Var

***Tabela 4.1** – Número de Bytes das mensagens MIDI*

A análise de um arquivo MIDI, portanto, deve levar em consideração os seguintes itens:

- o tipo de evento,
- o número de Bytes que o mesmo possui,
- se o mesmo possui ou não running status
- o número de Bytes do delta time

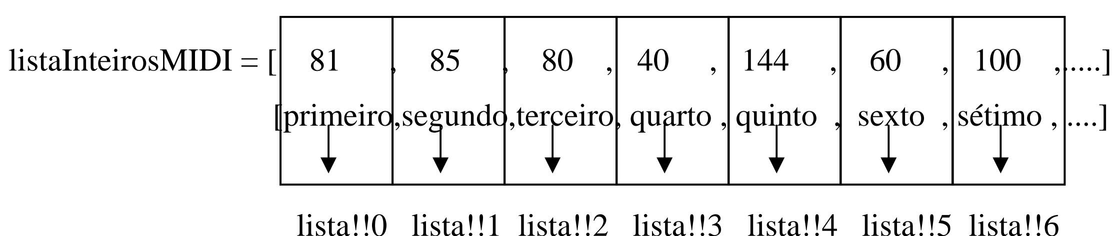
Assim, o menor número de Bytes que um evento da lista poderia ter seria 1 (um), ou seja, um evento de um byte com running status e delta time de um Bytel. Neste caso, ao se identificar o mesmo ter-se-ia de ler apenas um Byte, ou seja, o do delta time, já que o status seria o status corrente, o status do último evento.

O Maior número de Bytes que se deve analisar para identificar um evento seria o caso de não se ter running status, possuir um delta time de 4 Bytes e se ter que analisar pelo menos 3 Bytes do evento para identifica-lo, como no caso de se ativar uma nota ou de um meta evento.

Assim, fica simples separar os eventos da lista aberta. Basta checar todas as possibilidades de eventos, pelo número de Bytes, com e sem running status. Para tornar o programa mais eficiente, checa-se, em primeiro lugar, os eventos mais utilizados, como o de Ativar Nota, Desativar Nota, Meta Evento, Lirismo, Mensagem Exclusiva de Sistema e Troca de instrumento.

Como a análise dos eventos possui a mesma sintaxe, serão mostrados, aqui, exemplos de eventos mais comuns, pelo número de Bytes do delta time do evento, com running status e sem running status.

A lista será tratada pelos elementos iniciais, com o seguinte formato



Seguem alguns exemplos de análise de eventos:

4.1.1 Analisando se o evento é um evento de ativar nota com delta time de um Byte.

- Um evento é de ativar nota, **sem running status** e com delta time de um Byte quando:

- Primeiro Byte é menor que 128 -> delta time de 1 Byte;
- Segundo Byte é maior que 128 -> Byte de status;
- Segundo Byte é maior que 143 e menor que 160 -> ativar nota.

Caso estas três condições ocorram, o evento é de ativar nota, **sem running status** e com delta time de um Byte, e a lista do evento fica da seguinte forma:

[status de ativar nota, Nota, 4, 1, Delta time, status de ativar nota, Nota, Volume], conforme listagem 4.2 a seguir:

```
219 // XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
220 //      EVENTO SEM RUNNING STATUS(segundo > 127) E DELTA TIME DE UM BYTE (primeiro < 128)
221 // XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
222
223       //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
224      // 1- TESTA SE O EVENTO (segundo) É DE ATIVAR NOTA (143 < st < 160)
225     //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
226 |  ( (primeiro < 128)   &&   (segundo > 143) && (segundo < 160) )
227         =[segundo,terceiro,   4  , 1 , primeiro, segundo  ,terceiro,quarto]
228       //=[status ,  nota  ,NBdrop,nBDT,deltatime,ativarNota,  nota  ,volume]
```

***Listagem 4.2** – Reconhecendo ativar nota, delta time 1 Byte e sem running status*

- Um evento é de ativar nota, **com running status** e com delta time de um Byte quando:

- Primeiro Byte é menor que 128 -> delta time de 1 Byte;
- Segundo Byte é menor que 128 -> running status (deve-se utilizar o status do evento anterior. Assim, adota-se o argumento ***st*** da função, o qual guarda sempre o último status reconhecido;
- Checa se o argumento ***st*** é maior que 143 e menor que 160-> ativar nota.

Caso estas três condições ocorram, o evento é de ativar nota, **com running status** e com delta time de um Byte, e a lista do evento fica da seguinte forma:

[running status, Nota, 3, 1, Delta time, running status, Nota, Volume],

conforme listagem 4.3 a seguir:

```
164 // XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
165 //        EVENTO COM RUNNING STATUS(segundo < 128) E DELTA TIME DE UM BYTE (primeiro < 128)
166 // XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
167
168       //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
169      // 1- TESTA SE O RUNNING STATUS(st) É DE ATIVAR NOTA (143 < st < 160)
170     //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
171 |  ( (primeiro < 128)&& (segundo  < 128)    &&    (st > 143) && (st < 160) )
172          =[  st  , segundo,   3  , 1  ,primeiro,     st   ,segundo,terceiro]
173        //=[status,  nota   ,NBdrop,nBDT,deltatime,ativarNota, nota   ,volume  ]
```

Listagem 4.3 *– Reconhecendo ativar nota, delta time 1 Byte e com running status*

4.1.2 Analisando se o evento é um Meta Evento qualquer com delta time de um byte.

Um evento é um Meta Evento com delta time de um Byte quando:

- Primeiro Byte é menor que 128 -> delta time de 1 Byte;
- Segundo Byte é igual a 255 -> Meta Evento.

Neste caso, o terceiro Byte é o tipo do meta evento e o quarto é o número de bytes de dados do meta evento.

Caso estas duas condições ocorram, o evento é um Meta Evento, **sem running status** e com delta time de um Byte, e a lista do Meta Evento fica da seguinte forma:

[255, meta evento, 4+valor do quarto Byte, 1, Delta time, 255, evento, Bytes de dados],

conforme listagem 4.4 a seguir:

```
251       //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
252      // 5- TESTA SE O EVENTO (segundo==255) É UM EVENTO META EVENTO
253     //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
254 |  ( (primeiro < 128)   &&   (segundo == 255) )
255          =[  255 ,terceiro,4+quarto, 1  , primeiro,   255    ,terceiro,quarto]++(take quarto (drop 4 lt))
256        //=[status,Mevento , NBdrop ,nBDT,deltatime,metaEvento, evento ,NBytes]++(número de bytes do quarto byte -4)
257
```

Listagem 4.4 – Reconhecendo Meta evento, delta time 1 Byte e sem running status

OBS. Um **Meta Evento,** por convenção do padrão MIDI, bem como o evento de **Mensagem Exclusiva de sistema**, **não podem utilizar running status**.

Assim, a função ***listaEventos*** checa toda a lista MIDI lida até que não existam mais elementos na mesma, reconhecendo todos os eventos existentes, retornando uma lista de lista de eventos, todos com a estrutura mostrada nestes três exemplos.

4.1.3 Outras funções da biblioteca.

Esta biblioteca também permite salvar uma música, caso ela esteja no mesmo formato definido nesta biblioteca.

1- Função "salvarArquivoMIDI".

A listagem 4.5 a seguir, mostra a função ***salvarArquivoMIDI***. A mesma faz o seguinte:

- Pega a lista de eventos (***mus***) e acrescenta à mesma o cabeçalho(***cab***) dado no argumento da função;

```
42 salvarArquivoMIDI cab mus proc
```

- Acrescenta o *label* do track "MTrk" [77,84,114,107] ;

```
60   lableTrack = [77,84,114,107]
```

- Acrescenta o número de bytes dos eventos;

```
61   nBytesTrack=(transInt4B (length musica))
```

Assim:

```
44 # arqMIDIF0 =cab ++ lableTrack ++ nBytesTrack ++ musica
```

- Acrescenta os eventos, eliminando os quatro primeiros Bytes da lista ***mus*** de cada evento. Estes 4 primeiros Bytes foram acrescentados pela função ***listaEventos***;

```
59   musica =flatten ( map (drop 4) mus)
```

- Converte a lista para um vetor de caracteres e salva o arquivo;

```
46 # vetorfusao ={toChar x\\x<-arqMIDIF0}
```

- Salva um arquivo MIDI temporário no disco C.

```
54 #(_,file2, proc)= fopen "C:\\tocaMIDIF0.mid" FWriteData proc
```

```
32 salvarArquivoMIDI :: [Int] [[Int]] *World -> *World
33 salvarArquivoMIDI cab mus proc
34 # (_, proc) = OSStopMusic proc
35 # arqMIDIF0 =cab ++ lableTrack ++ nBytesTrack ++ musica
36 # vetorfusao ={toChar x\\x<-arqMIDIF0}
37 # (yy, proc) = selectOutputFile "Salvar o arquivo MIDI: escolha o diretorio!" "*.mid" proc
38 | yy==Nothing = proc
39 # endEscolhido = fromJust yy
40 #(_,file2, proc)= fopen endEscolhido FWriteData proc
41 # file2 = fwrites vetorfusao file2
42 #(_, proc)= fclose file2 proc
43 #(_,file2, proc)= fopen "C:\\tocaMIDIF0.mid" FWriteData proc
44 # file2 = fwrites vetorfusao file2
45 #(_, proc)= fclose file2 proc
46 =proc
47 where
48   musica =flatten ( map (drop 4) mus)
49   lableTrack = [77,84,114,107]
50   nBytesTrack=(transInt4B (length musica))
```

Listagem 4.5 *– Função de salvar o arquivo MIDI formato 0*

2 – Função “playMIDISalvo”.

Esta função toca o arquivo temporário criado pela função de salvar o arquivo MIDI. Caso o arquivo não exista, ou seja, não tenha sido salvo ainda, nada acontece: (*| **not OK** = (**True,proc**)*)

```
52 playMIDISalvo :: !*World ->(!Bool,!*World)
53 playMIDISalvo  proc
54 # (ok,path) = GetShortPathName "C:\\tocaMIDIF0.mid"
55 | not ok = (True,proc)
56 = OSPlayMusic path True proc
```

Listagem 4.6 *– Função de tocar o arquivo MIDI formato 0*

3 – Função “playMIDITrabalho”.

Esta função concatena as duas funções anteriores, ou seja, salva um arquivo MIDI temporário e o toca. Apenas não abre o gerenciador de arquivos para que o usuário salve o arquivo onde desejar. Assim, como o salvamento não abre nenhuma janela, dá-se a ilusão que se está tocando diretamente a lista de eventos MIDI, mas, na realidade, está se criando um arquivo e tocando-o.

```
58 playMIDITrabalho :: [.Int] [[.Int]] *World ->(!Bool,!*World)
59 playMIDITrabalho cab mus proc
60 # (_, proc) = OSStopMusic proc
61 # arqMIDIF0 =cab ++ lableTrack ++ nBytesTrack ++ musica
62 # vetorfusao ={toChar x\\x<-arqMIDIF0}
63 #(_,file2, proc)= fopen "C:\\tocaMIDIF0.mid" FWriteData proc
64 # file2 = fwrites vetorfusao file2
65 #(_, proc)= fclose file2 proc
66 #proc = playMIDISalvo proc
67 =proc
68 where
69   musica =flatten ( map (drop 4) mus)
70   lableTrack = [77,84,114,107]
71   nBytesTrack=(transInt4B (length musica))
```

***Listagem 4.7** – Função de tocar o arquivo MIDI formato 0 temporário*

4.2 Lendo eventos de arquivos MIDI SMF formato 1.

Caso se deseja implementar uma função, uma biblioteca para também separar eventos de arquivos SMF formato 1, basta aplicar a mesma lógica da função criada para leitura de formato 0 para formato 1 a cada track dos SMF formato 1.

O fluxograma, a seguir, mostra a lógica desta implementação.

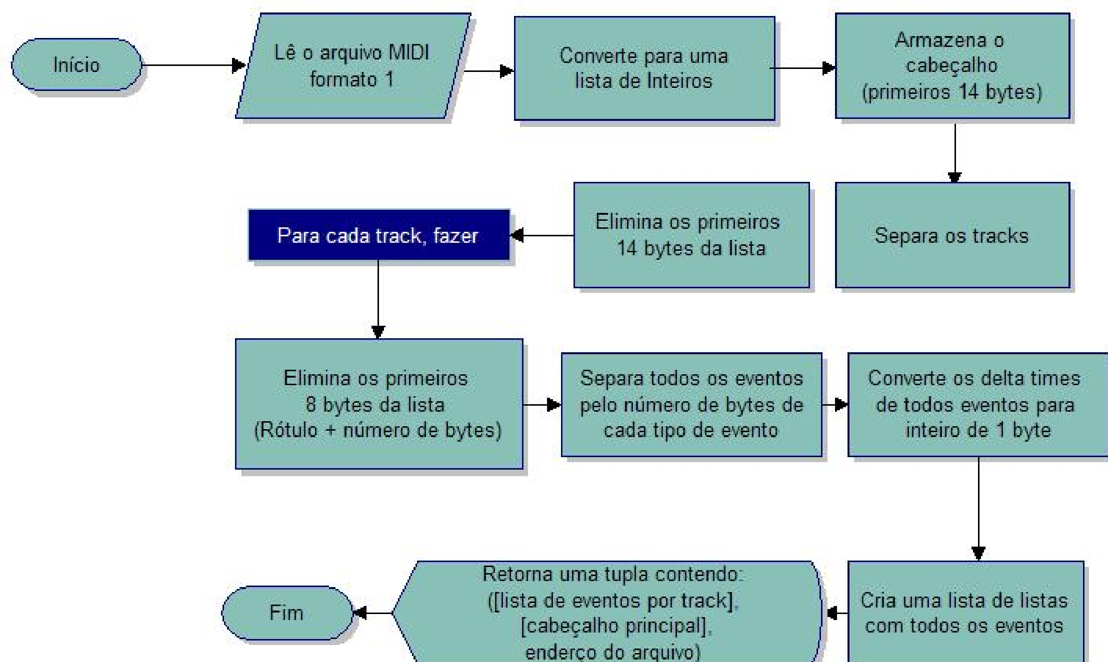


***Figura 4.4** – Fluxograma da biblioteca Separa Eventos MIDI F1*

4.3 Gera MIDI formato 0.

Esta é uma biblioteca para geração de arquivos SMF formato 0 em baixo nível.

O fluxograma, a seguir, ilustra a lógica de implementação desta biblioteca.

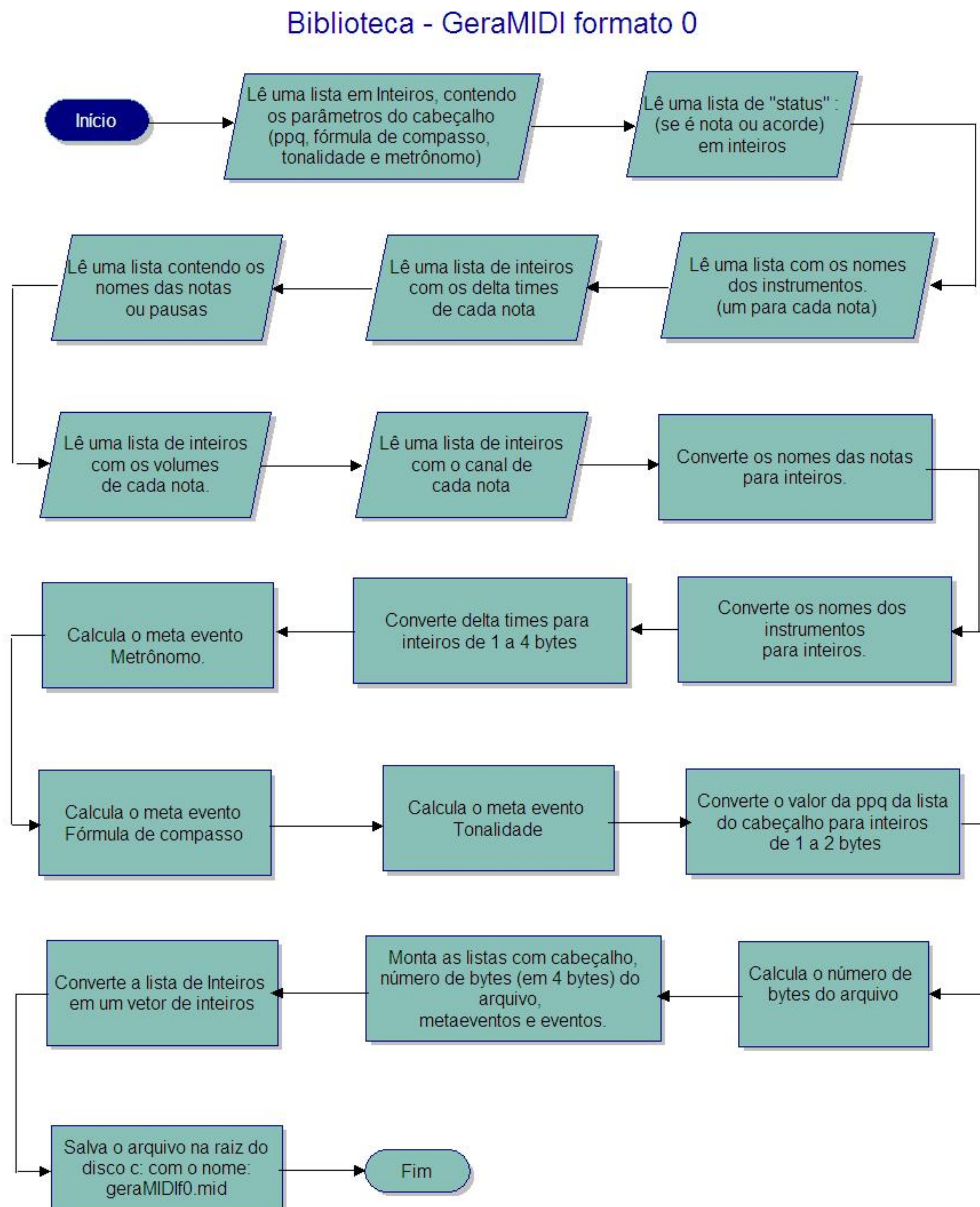


***Figura 4.5** – Fluxograma da biblioteca Separa Eventos MIDI F0*

Ela possui as seguintes potencialidades:

1. Facilita ao usuário entrar com dados de listas, aderentes ao programador para manipulação por uma linguagem funcional;

2. Trata automaticamente notas solos e notas em acordes, colocando delta times de forma a efetivar o estado da nota (solo/acorde);
3. Permite ao mesmo entrar com notas e figuras musicais com nomes;
4. Calcula o meta evento de tempo dado o valor do metrônomo e a *ppq*;
5. Monta uma lista contendo o arquivo MIDI já com o cabeçalho, fim de track e *labels*, inserindo automaticamente, no mesmo, conta e converte a quantidade de Bytes do track para o formato de quatro Bytes, evitando ao usuário ter que contar e converter manualmente.

Antes de se mostrar as funções da biblioteca, é interessante mostrar como utilizá-la, para conhecer o formato dos dados que serão manipulados pela mesma. A listagem 4.8 a seguir, mostra um programa que utiliza a biblioteca ***geraMIDIF0***.

```
1  module testeFormato0
2
3  import StdEnv,StdIO,StdArq
4  import geraMIDIF0
5
6  Start proc
7  # cabecalho = [
8               96, // ppq
9                3, // numerador da formula de compasso
10               4, // denominador da formula de compasso
11               2, // armadura de clave
12             120  // metronomo
13               ]
14
15
16 # ativaDesativa = [   1,        1,      0,       1,       0,       1,       0,       0,       1,      1,       0,        0     ]
17 # programChange = ["flauta","violao","violao","flauta","flauta","flauta","flauta","flauta","violao","violao","violao","violao" ]
18 # deltaTime     = [   0,        0,      96,      0,       48,      0,       48,      0,       0,      0,       96,       0     ]
19 # nota          = ["re5",     "la5",   "la5",  "sol5",  "sol5",   "mi5",   "mi5",   "re5",   "la5",  "do6",   "la5",    "do6"  ]
20 # volume        = [  100,      100,     0,      100,      0,      100,      0,       0,      100,    100,      0,        0     ]
21 # canal         = [   1,        0,      0,       1,       1,       1,       1,       1,       0,      0,       0,        0     ]
22 # listaMIDI = geraMidiF0 cabecalho ativaDesativa programChange deltaTime nota volume canal
23
24
25
26 # arqSalvar = map toChar listaMIDI
27 # vetorSalva = {x\\x<-arqSalvar}
28 # (_,file3, proc)= fopen "C:\\geraMidiF0.mid" FWriteData proc
29 # file3 = fwrites vetorSalva file3
30 # (_, proc)= fclose file3 proc
31 # (_, proc) = OSStopMusic proc
32 # (_, proc) = playMid "C:\\geraMidiF0.mid" proc
33
34 = (listaMIDI,proc)
```

Listagem 4.8 – *Utilizando a biblioteca* ***geraMIDIF0***

A função que cria uma lista com os eventos MIDI em seqüência, formato 0, é a função ***geraMidiF0***. No exemplo de teste, atribui-se à variável ***listaMIDI*** a lista gerada pela chamada da função ***geraMidiF0***.

```
22 # listaMIDI = geraMidiF0 cabecalho ativaDesativa programChange deltaTime nota volume canal
```

A função ***geraMidiF0*** possui 7 (sete) argumentos, cada argumento uma lista onde todas as listas, menos “cabecalho”, **devem** possuir o mesmo número de elementos. Nestas listas (menos “cabecalho”), cada elemento corresponde aos elementos de mesmo índice em cada uma das listas, ou seja:

```
# ativaDesativa  = [    1,       1,
# programChange  = ["flauta", "violao",
# deltaTime      = [    0,       0,
# nota           = ["re5",     "la5",
# volume         = [  100,     100,
# canal          = [    1,       0,
```

Indica: Evento de ativar nota (1), com instrumento “flauta”, delta time = 0, nota “re5”, com volume=100, tocada no canal 1.

Figura 4.6** – Utilizando a biblioteca **geraMIDIF0

- **Primeiro elemento da lista: cabeçalho.**

  “cabeçalho” é uma lista com os seguintes elementos:

  [ppq,
  numerador da fórmula de compasso,
  denominador da fórmula de compasso,
  armadura de clave (30 tipos de armadura),
  metrônomo]

  - **ppq:** Valor máximo = 4095
  - **numerador da fórmula de compasso:** Valores aceitos pela teoria musical
  - **denominador da fórmula de compasso:** Valores aceitos pela teoria musical
  - **armadura de clave:** Um valor entre 0 e 29. A biblioteca disponibiliza uma função, ***mostraTonalidade***, que, a partir de um número identificador da armadura de clave, devolve o Meta Evento de armadura de clave pronto para ser inserido no arquivo MIDI que se está montando.

    Como exemplo, se for executada a função ***mostraTonalidade 2***, a mesma devolveria o meta evento de armadura de clave de sol maior (**[0,255,89, 2,1,0]**) A listagem , a seguir, mostra esta função:

```
156 mostraTonalidade :: Int -> [Int]
157 mostraTonalidade keysign
158 |keysign ==  0 = [0,255,89,2,0,0]// "DoM"
159 |keysign ==  1 = [0,255,89,2,0,1]//"Lam"
160 |keysign ==  2 = [0,255,89,2,1,0]//"SolM"
161 |keysign ==  3 = [0,255,89,2,1,1]//"Mim"
162 |keysign ==  4 = [0,255,89,2,2,0]//"ReM"
163 |keysign ==  5 = [0,255,89,2,2,1]//"Sim"
164 |keysign ==  6 = [0,255,89,2,3,0]//"LaM"
165 |keysign ==  7 = [0,255,89,2,3,1]//"Fa#m"
166 |keysign ==  8 = [0,255,89,2,4,0]//"MiM"
167 |keysign ==  9 = [0,255,89,2,4,1]//"Do#m"
168 |keysign == 10 = [0,255,89,2,5,0]//"SiM"
169 |keysign == 11 = [0,255,89,2,5,1]//"Sol#m"
170 |keysign == 12 = [0,255,89,2,6,0]//"Fa#M"
171 |keysign == 13 = [0,255,89,2,6,1]//"Re#m"
172 |keysign == 14 = [0,255,89,2,7,0]//"Do#M"
173 |keysign == 15 = [0,255,89,2,7,1]//"La#m"
174
175 |keysign == 16 = [0,255,89,2,255,0]//"FaM"
176 |keysign == 17 = [0,255,89,2,255,1]//"Rem"
177 |keysign == 18 = [0,255,89,2,254,0]//"SibM"
178 |keysign == 19 = [0,255,89,2,254,1]//"Solm"
179 |keysign == 20 = [0,255,89,2,253,0]//"MibM"
180 |keysign == 21 = [0,255,89,2,253,1]//"Dom"
181 |keysign == 22 = [0,255,89,2,252,0]//"LabM"
182 |keysign == 23 = [0,255,89,2,252,1]//"Fam"
183 |keysign == 24 = [0,255,89,2,251,0]//"RebM"
184 |keysign == 25 = [0,255,89,2,251,1]//"Sibm"
185 |keysign == 26 = [0,255,89,2,250,0]//"SolbM'
186 |keysign == 27 = [0,255,89,2,250,1]//"Mibm"
187 |keysign == 28 = [0,255,89,2,249,0]//"DobM"
188 |keysign == 29 = [0,255,89,2,249,1]//"Labm"
```

Listagem 4.9 – *Função "mostraTonalidade"*

- **metrônomo:** A partir do valor do metrônomo, a função calcula o tempo da semínima em microssegundos com três bytes (***byteMetronomo***) e monta o meta evento de tempo (aqui denominado por ***metaEventoMetronomo***).

```
16      metaEventoMetronomo = [0,255,81,3] ++ byteMetronomo
```

- ## Segundo elemento da lista: ativaDesativa.

ativaDesativa é uma lista com elementos 0 ou 1, os quais determinam se o evento atual é de ativar ou desativar nota:

- 0 -> evento de desativar nota
- 1 -> evento de ativar nota

- ## Terceiro elemento da lista: programChange.

programChange é uma lista, onde cada elemento corresponde a um instrumento que a nota musical utilizará para soar. A cada nota, da lista

nota, tem-se um instrumento correspondente. A Função de gerar lista se encarrega de transformar o nome do instrumento no código MIDI correspondente. A função que faz isto é ***pegaCodigoInstr***. Os instrumentos colocados na biblioteca para teste são mostrados na listagem 4.10 a seguir, onde a função de conversão nome para código também é mostrada.

```
49 pegaCodigoInstr :: {#.Char} -> .Int
50 pegaCodigoInstr x =: "piano"    =  0
51 pegaCodigoInstr x =: "flauta"   = 75
52 pegaCodigoInstr x =: "violino"  = 40
53 pegaCodigoInstr x =: "saxofone" = 64
54 pegaCodigoInstr x =: "violao"   = 24
55 pegaCodigoInstr x =: "trompete" = 56
```

Listagem 4.10 *– Conversão de nome de instrumento para código MIDI*

A função de gerar a lista MIDI se encarrega de colocar automaticamente antes de cada evento de ativar nota um evento de mudança de instrumento.

- ## Quarto elemento da lista: deltatime.

deltatime é uma lista com valores em inteiro correspondente ao delta time de um evento, no caso, de ativar ou desativar nota. Nesta função não precisa se preocupar se o delta time irá possuir um, dois, três ou quatro Bytes, a mesma fará a conversão necessária. O maior valor de delta time é 4294967167.

- ## Quinto elemento da lista: nota.

nota é uma lista de notas musicais. A função de gerar lista permite que se entre com o nome da nota e a converte para o código MIDI equivalente. O índice da nota na lista é o próprio código MIDI.

A figura 4.7 mostra a lista de onde o índice da nota é consultado. Colocou-se nesta tabela as 128 notas possíveis[59].

```
69 //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
70 notaSB :: [.{#Char}]
71 notaSB =
72   ["do0", "do#0", "re0", "re#0", "mi0", "fa0", "fa#0", "sol0", "sol#0", "la0", "la#0", "si0",
73    "do1", "do#1", "re1", "re#1", "mi1", "fa1", "fa#1", "sol1", "sol#1", "la1", "la#1", "si1",
74    "do2", "do#2", "re2", "re#2", "mi2", "fa2", "fa#2", "sol2", "sol#2", "la2", "la#2", "si2",
75    "do3", "do#3", "re3", "re#3", "mi3", "fa3", "fa#3", "sol3", "sol#3", "la3", "la#3", "si3",
76    "do4", "do#4", "re4", "re#4", "mi4", "fa4", "fa#4", "sol4", "sol#4", "la4", "la#4", "si4",
77    "do5", "do#5", "re5", "re#5", "mi5", "fa5", "fa#5", "sol5", "sol#5", "la5", "la#5", "si5",
78    "do6", "do#6", "re6", "re#6", "mi6", "fa6", "fa#6", "sol6", "sol#6", "la6", "la#6", "si6",
79    "do7", "do#7", "re7", "re#7", "mi0", "fa7", "fa#7", "sol7", "sol#7", "la7", "la#7", "si7",
80    "do8", "do#8", "re8", "re#8", "mi8", "fa8", "fa#8", "sol8", "sol#8", "la8", "la#8", "si8",
81    "do9", "do#9", "re9", "re#9", "mi9", "fa9", "fa#9", "sol9", "sol#9", "la9", "la#9", "si9",
82    "do10  ","do#10","re10","re#10","mi10","fa10","fa#10","sol10"]
83 //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Figura 4.7 *– Lista das 128 notas possíveis*

---

[59] um Byte de dados possui 7 bits-> 128 valores diferentes

- **Sexto elemento da lista: volume.**

  volume é uma de valores de volume, variando de 0 a 127 (128 valores possíveis).

- **Sétimo elemento da lista: canal.**

  canal é uma lista de canais MIDI, com valores de 0 a 15 (dezesseis canais MIDI possíveis –> Um *nible* – 4 Bits).

Estes são os sete parâmetros da função ***geraMidiF0.*** A listagem 4.11 a seguir mostra o trecho de código da função ***geraMidiF0***.

```
10 geraMidiF0 :: .[a] [.Int] [{#.Char}] [.Int] [{#.Char}] [.Int] [.Int] -> [Int] | toInt a
11 geraMidiF0 cab atDt pc dTime nta vol cn
12     =cabe1 ++ ppq ++ cabe2 ++ contagem ++ cabe3++cabe4++cabe5 ++  nltFinal ++ fimArq
13     where
14      fCompasso           = formulaCompasso (toInt(cab!!1)) (toInt(cab!!2))
15      byteMetronomo       = metronomoBytes (toInt(cab!!4))
16      metaEventoMetronomo = [0,255,81,3] ++ byteMetronomo
17      metaEventoKeySign   = mostraTonalidade (toInt(cab!!3))
18      cabe1               = [77,84,104,100,0,0,0,6,0,0,0,1]
19      ppq                 = transInt2B (toInt(hd cab))
20      cabe2               = [77,84,114,107]
21      cabe3               = fCompasso
22      cabe4               = metaEventoKeySign
23      cabe5               = metaEventoMetronomo
24      nbytes              = (length cabe3) + (length cabe4) + (length cabe5) + (length fimArq)
25      fimArq              = [0,255,47,0]
26      nltFinal            = geraMidiF0Aux atDt pc "nada" dTime nta vol cn []
27      contagem            = (transInt4B(nbytes+(length nltFinal)))
```

***Listagem 4.11** – Função que gera a lista MIDI formato 0*

A partir da lista MIDI obtida, pode-se salvar a mesma em um arquivo SMF, para tanto, antes deve-se converter a lista de inteiros para uma lista de caracteres, converter a lista para vetor e salvar. A listagem 4.12 a seguir, mostra este processo. Para salvar, é só utilizar a função ***GravarArquivoDadosExpl*** da biblioteca StdArq para salvar.

```
26 # arqSalvar = map toChar listaMIDI
27 # vetorSalva = {x\\x<-arqSalvar}
```

***Listagem 4.12** – Função que gera a lista MIDI formato 0*

O programa completo está gravado no CD em anexo.

4.4 Gera MIDI formato 1.

Esta é uma biblioteca para geração de arquivos SMF formato 1 em baixo nível. Ela possui as seguintes potencialidades:

1. Facilita ao usuário entrar com dados de listas, aderentes ao programador e à manipulação por uma linguagem funcional, onde cada lista é um track a ser implementado pelo programa (cada lista é uma lista similar à lista manipulada pelo programa gera MIDI formato 0);
2. Trata automaticamente notas solos e notas em acordes, colocando delta times de forma a efetivar o estado da nota (solo/acorde);
3. Trata pausas de figuras musicais;
4. Trata notas solos e acordes de forma mais completa, permitindo partir de um acorde e iniciar um novo acorde, acrescentando mais um identificador na lista de status da nota;
5. Diferente da geraMIDIF0, nesta biblioteca entra-se com as notas com seu código em inteiro, caso o programador prefira entrar com nomes de notas, basta aplicar a função de converter lista de nomes em lista de códigos (***pegaCodigoNotaInt***) antes de montar a lista para geração do arquivo MIDI. Esta função está na biblioteca **geraMIDIF0**, vista no item 4.3. Exemplo de uso:

pegaCodigoNotaInt "do5" -> devolve **60**

Para aplicar esta função em uma lista inteira de notas basta aplicar a função ***pegaCodigoNotaInt*** na lista toda de notas utilizando a função ***map*** do Clean, ou seja:

map pegaCodigoNotaInt ["do5", "mi5","sol5"]-> devolve **[60,64,67]**

6. Calcula o meta evento de tempo dado o valor do metrônomo e a ppq;
7. Monta uma lista contendo o arquivo MIDI já com o cabeçalho, os *tracks* com seu contador de Bytes, fim de *track* e *label*.

O fluxograma, a seguir, ilustra a lógica de implementação da biblioteca ***geraMIDIF1***.

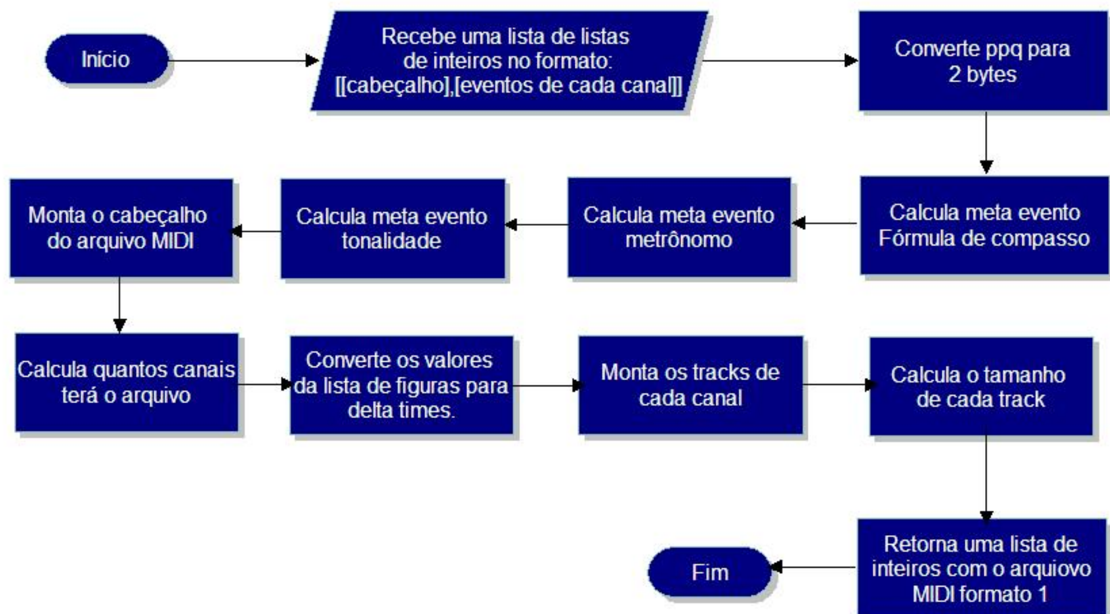


Figura 4.8** – Fluxograma da função **geraMIDIF1

Antes de se mostrar as funções da biblioteca, é interessante mostrar como utilizá-la, para conhecer o formato dos dados que serão manipulados pela mesma. A listagem 4.13 a seguir, mostra um programa que utiliza a biblioteca ***geraMIDIF1***.

```
1  module testeF1
2  import geraMIDIF1
3  import StdEnv,StdIO
4
5  Start proc
6  #musica =[  [[ 96,  // ppq
7                  3,  // numerador da formula de compasso
8                  4,  // denominador da formula de compasso
9                  2,  // armadura de clave
10              120]],// metrônomo
11
12            [[1,1,0,0],         // nota->1, acorde ->0 ou 2
13             [24,24,24,24],     // instrumento [0..127]
14             [4,4,4,4],         // figura [1..]
15             [69,500,69,72],    // nota [0..127], pausa -> 500
16             [100,100,100,100], // volume [0..127]
17             [0,0,0,0]          // canal [0..15]
18            ],
19            [[1,1,1,1],
20             [73,73,73,73],
21             [4,8,8,4],
22             [500,67,64,500],
23             [100,100,100,100],
24             [1,1,1,1]
25            ],
26            [[1,1],
27             [73,73],
28             [2,4],
29             [62,500],
30             [100,100],
31             [1,1]
32            ]]
33
34 # listaMIDI = geraMidiF1 musica
35 # arqSalvar = map toChar listaMIDI
36 # vetorSalva = {x\\x<-arqSalvar}
37 # (_,file3, proc)= fopen "C:\\geraMidiF1.mid" FWriteData proc
38 # file3 = fwrites vetorSalva file3
39 # (_, proc)= fclose file3 proc
40 # (_, proc) = OSStopMusic proc
41 # (_, proc) = playMid "C:\\geraMidiF1.mid" proc
42 = (listaMIDI,proc)
```

***Listagem 4.13** – Utilizando a função geraMIDIF1*

A função que cria uma lista com os eventos MIDI em seqüência, formato 1, é a função ***geraMidiF1***. No exemplo de teste, atribui-se à variável ***listaMIDI*** a lista gerada pela chamada da função ***geraMidiF1***.

```
35 # listaMIDI = geraMidiF1 musica
```

A função ***geraMidiF1*** possui 1 (um) argumento: uma lista de lista de listas. De forma semelhante à função de gerar MIDI em formato 0, a primeira lista contém o cabeçalho, idêntico ao cabeçalho da ***geraMIDIF0***, não sendo necessário detalhá-lo novamente. A única diferença é que se tem um nível a mais de lista, ou seja, coloca-se o cabeçalho dentro de outra lista para compatibilizar seu tipo de dados com o das demais listas.

[cabeçalho] -> exemplo: [[96, 3, 4, 2, 120]]

As demais listas são listas de **tracks**, similares, também, às listas da ***geraMIDIF0***, com a diferença que, ao invés de entrar com cada lista separada, entra-se com todas as listas, todos os *tracks*, dentro de uma lista individual, conforme exemplo a seguir:

```
13          [[1,1,0,0],           // nota->1, acorde ->0 ou 2
14          [24,24,24,24],        // instrumento [0..127]
15          [4,4,4,4],            // figura [1..]
16          [69,500,69,72],       // nota [0..127], pausa -> 500
17          [100,100,100,100],    // volume [0..127]
18          [0,0,0,0]             // canal [0..15]
19         ],
```

Listagem 4.14 *– Entrada de listas*

Têm-se agora três estados de notas: 1= nota solo, 0 = nota em acorde e 2 nota em outro acorde. Neste estado já não se trata mais, como no formato 0, se a o evento é de ativar nota(1) ou desativar nota (0), a própria função se encarrega de gerar os eventos de desativar nota no instante correto. Os estados agora indicam se uma nota é solo, se está em um acorde (1) ou se está iniciando em outro acorde (2). No capítulo 6, estudos de casos, alguns exemplos mais elaborados serão lá apresentados.

Incorporou-se, nesta biblioteca, a pausa musical. Para tanto, utilizou-se um código de nota inexistente no protocolo MIDI (0 a 127), o código 500, para simbolizar a pausa. No capítulo de estudos de casos, Capítulo 6, é implementado um editor de partituras que permite que se entre com a palavra "pausa" em vez de código, bem como instrumentos, volume e todos os parâmetros da música em formato textual.

Assim, como na biblioteca de gerar MIDI em formato 0, esta biblioteca gera uma lista contendo o arquivo MIDI completo, devendo o mesmo ser transformado em uma lista de caracteres para depois ser salvo como um vetor de dados.

A listagem 4.15 a seguir mostra o trecho de código da função ***geraMidiF1.***

```
 9 geraMidiF1 :: [ [ [Int]] ]-> [Int]
10 geraMidiF1 musica = [77,84,104,100,0,0,0,6,0,1,0,length musica]++
11                        ppqCabPrinc ++
12                        trackMetaEvento ++
13                        (processaCanais musica)++
14                        [72,67,74,38,76,86,76]
15 where
16   cab                  = (hd (musica!!0))
17   ppqCabPrinc          = (transInt2B (toInt(hd(hd (musica!!0)))))
18   fCompasso            = formulaCompasso (toInt(cab!!1)) (toInt(cab!!2))
19   byteMetronomo        = metronomoBytes (toInt(cab!!4))
20   metaEventoMetronomo = [0,255,81,3] ++ byteMetronomo
21   metaEventoKeySign    = mostraTonalidade (toInt(cab!!3))
22   cabe2                = [77,84,114,107]
23   cabe3                = fCompasso
24   cabe4                = metaEventoKeySign
25   cabe5                = metaEventoMetronomo
26   nbytes               = (length cabe3) + (length cabe4) + (length cabe5) + (length fimArq)
27   fimArq               = [0,255,47,0]
28   trackMetaEvento      = cabe2 ++ (transInt4B(nbytes)) ++cabe3++cabe4++cabe5++fimArq
29
```

. ***Listagem 4.15*** *– Função geraMidiF1*

4.5 Conclusão.

Este capítulo se encarregou de mostrar como abrir, analisar, gerar e tocar arquivos MIDI SMF formato 0 e formato 1, de várias formas diferentes. No mesmo foram implementadas bibliotecas e funções úteis para projeto e implementação de aplicativos MIDI, sem que se tenha de seguir o protocolo MIDI em baixo nível, com suas estruturas de delta times e eventos.