INTRODUÇÃO

OBJNOTA, uma biblioteca de rotinas em PASCAL orientado por objeto, é o primeiro resultado final do projeto de pesquisa "Informática em Música: Procedimentos e Funções Básicas" ao qual tenho me dedicado desde 1989 com o apoio do CNPq. Esta biblioteca, desenvolvida em 1991-92 contém as operações básicas necessárias para o desenvolvimento de programas que lidam com o parâmetro altura em música. A programação orientada por objeto permitiu utilizar as diversas codificações utilizadas na Teoria da Música com uma simples mudança na declaração do programa, além de permitir facilmente a introdução de outras codificações e operações.

Este trabalho é dividido em seis partes e procura cobrir todos os aspectos importantes de OBJNOTA, desde a definição das operações até a documentação das rotinas, fornecendo ao programador as informações necessárias para o desenvolvimento de aplicativos.

A primeira parte, "Operações Básicas", explica as operações utilizadas em termos musicais e matemáticos. Observe-se contudo, que as explicações contidas nesta parte não prescinde de um entendimento de certa forma especializado sobre a Teoria da Música desenvolvidas nas últimas décadas. A segunda parte, "Codificação", expõe as diversas codificações utilizadas

para a notas e intervalos musicais. A terceira, “Referência das Globais”, lista as constantes e tipos à disposição do programador, enquanto a quarta, “Referência dos Objetos”, lista os métodos pertencentes a cada dos objetos, com pequenos exemplos práticos. A quinta parte, “Exemplos de Utilização”, apresenta três programas como exemplo de utilização de várias das rotinas da biblioteca. A última parte, “Documentação das Rotinas”, corresponde a um manual técnico de OBJNOTA e descreve cada uma das rotinas usadas.

OPERAÇÕES BÁSICAS

Operações Recursivas

O termo Operações Recursivas é aplicado àquelas operações que de alguma forma mantém as características fundamentais de um conjunto de notas, permitindo a recuperação do conjunto original. Por exemplo, para obter-se a melodia original de uma transposição a um intervalo de quarta-justa ascendente basta transpô-la a um intervalo de quarta-justa descendente; para obter-se a melodia original de um retrógrado basta retrograda-lo novamente.

Enarmonia

Denominamos notas enarmônicas aquelas que, embora pertencentes ao mesmo grau da escala cromática do sistema temperado, têm significados tonais diferentes como, por exemplo, Lab e Sol#. Enarmonia pode ser definida então como a operação que modifica o significado tonal de uma nota musical. A operação Enarmonia faz sentido apenas no sistema s*tonal* considerando que no sistema s*temperado* não há diferenciação de significados tonais para as notas.

A implementação de uma função para a enarmonização deve levar em consideração além da própria nota a direção da enarmonização. Por exemplo, a nota Sib enarmonizada na direção descendente produzirá o La#, e na direção ascendente o Dodb. Neste trabalho utilizamos o conceito de direção de enarmonia baseado nos graus da escala tonal com a enarmonização descendente representada por graus negativos e a ascendente por graus positivos. Assim para produzir o La#, o Sib deve ser enarmonizado na direção -1, i.e., um grau tonal descendente. O valor de direção pode evidentemente ser diferente da unidade. Portanto, dada uma nota ***n***, suas equivalentes enarmônicas podem ser obtidas pela fórmula

$$\mathbf{E}d(n) = n + 12d \qquad (1)$$

onde ***d*** representa a direção da enarmonia.

O âmbito útil para direção no sistema *stemperado* é de -3 a +3. Outros valores produzirão enarmonia dupla, conforme podemos verificar na lista das enarmônicas de Fa# (Tabela 1).

Tabela 1: Enarmonia

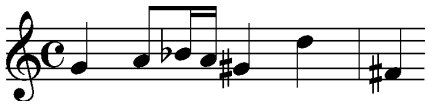
Direção	0	1	2	3	4	5	6	7	8
Enarmônica	Fa#	Solb	Latb	Sipb	Dosb	Dos#	Req#	Mid#	Fa#
Direção	-8	-7	-6	-5	-4	-3	-2	-1	0

Inversão

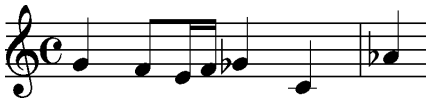
Inversão, no sentido aqui empregado, i.e., inversão melódica estrita ou real, significa a redisposição de uma linha melódica em movimento contrário à

original mantendo as mesmas qualidades intervalares. Por exemplo, a linha melódica do Ex. 1a invertida produz a do Ex. 1b.

Ex. 1a



Ex. 1b



Neste trabalho utilizamos o conceito de inversão em torno de um eixo, ao qual denominamos eixo de inversão. No caso do Ex. 1, a linha melódica está invertida em torno do Sol, ou, em outras palavras a nota Sol funciona como eixo de inversão para a obtenção da segunda linha melódica. Inversão pode então ser definida como a operação que transfere uma determinada nota para uma posição à mesma distância de um eixo mas em direção oposta, ou, dados uma nota e um eixo, a inversão desta nota corresponde a outra nota que esteja a um mesmo intervalo do eixo mas na direção oposta. Desta forma, a inversão de uma nota ***n*** em relação a um eixo ***x*** distante desta nota um intervalo ***i***, corresponderia a:

$$\mathbf{I}x(n) = x - i; \quad (2)$$

considerando entretanto que $i = n - x$, temos que

$$\mathbf{I}x(n) = x - (n - x)$$

$$\mathbf{I}x(n) = 2x - n. \quad (3)$$

O exame da fórmula (3) acima nos leva a redefinir inversão, considerando que esta é na verdade uma soma, com a transposição do negativo da nota, i.e., da inversão da nota em torno do eixo **0**, transposta ao intervalo **2*x***. Assim,

$$\mathbf{I}x(n) = \mathbf{T}z(-n) = z - n, \qquad (4)$$

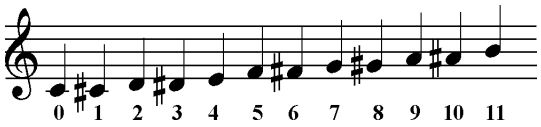
onde $z = 2x$, representando o índice de inversão.

Na prática, trabalhamos com inversões em torno de uma nota ou em torno de um ponto eqüidistante entre duas notas. Conseqüentemente, o valor de ***i*** e de ***x*** nas fórmulas acima pode assumir um valor inteiro ou real com a parte decimal correspondente a meia unidade. Convém lembrar entretanto, que para cada eixo de inversão há um eixo de inversão secundário correspondente a um intervalo de trítono do original.

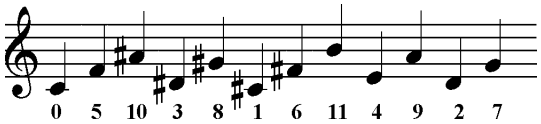
Multiplicação

A operação multiplicação, conforme o nome indica, consiste na multiplicação do código numérico de uma nota pelo código numérico de um intervalo, seguida da operação módulo (**mod**) oitava. Embora tradicionalmente a multiplicação seja realizada apenas no sistema *stemperado*, ela pode também ser realizada no sistema *stonal*. Evidentemente os resultados obtidos nos dois sistemas são enarmonicamente equivalentes. Os multiplicadores tradicionalmente utilizados são os correspondentes aos intervalos de quarta-justa e de quinta-justa, os quais transformam uma escala cromática (Ex. 2a) em respectivamente um ciclo de quartas (Ex. 2b) e de quintas (Ex. 2c), ou vice-versa.

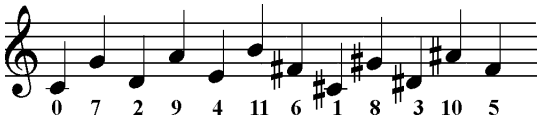
Ex. 2a



Ex. 2b



Ex. 2c



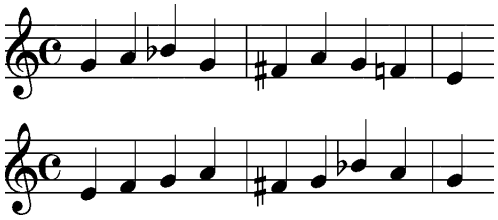
A multiplicação de uma nota ***n*** por um intervalo (multiplicador) ***m*** pode ser obtida pela fórmula

$$\mathbf{M}m(n) = nm \bmod 12 \qquad (5)$$

Retrogradação

Retrógrado (*cancrizans* [Latim], *Krebsgang* [Alemão], e *al rovescio* [Italiano]) consiste na redisposição de uma linha melódica começando com a última nota e concluindo com a primeira (Ex. 3).

Ex. 3

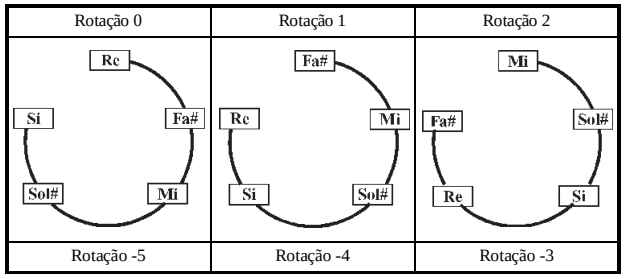


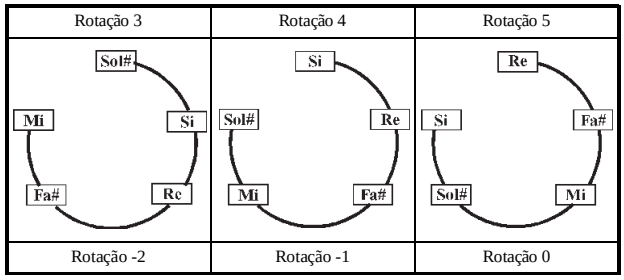
A operação Retrógrado pode ser definida como a redisposição das notas de um conjunto ordenado de tal forma que o conjunto resultante corresponda ao original lido de trás para a frente.

Rotação

Dispondo-se um determinado conjunto de notas em forma circular, i.e., com o último elemento sendo seguido pelo primeiro, pode-se definir Rotação como qualquer conjunto resultante cujo elemento inicial seja qualquer dos elementos do conjunto original. A operação rotação leva em consideração a posição, no conjunto original, da nota que deverá iniciar o conjunto e a direção da rotação, considerando como positiva a rotação no sentido anti-horário.

Tabela 2

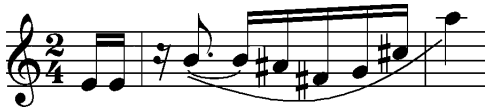




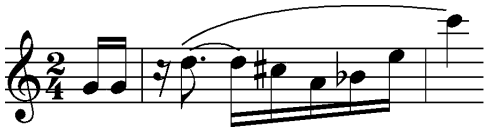
Transposição

Transposição consiste na transferência de uma nota ou um conjunto de notas a um determinado intervalo mantendo-se as mesmas relações intervalares originais. Por exemplo, a linha melódica do Ex. 4a tranposta ascendentemente ao intervalo de terça-menor produz a linha melódica do Ex. 4b.

Ex. 4a



Ex. 4b



Transposição pode ser definida como o resultado da soma de um valor correspondente a uma nota com o valor correspondente a um intervalo. Desta forma, a transposição de uma linha melódica a um determinado intervalo, por exemplo, corresponderia ao resultado da soma do valor de cada das notas que compõe a linha por uma constante correspondente ao intervalo. Dados uma nota ***n*** e um intervalo de transposição ***i***, a transposição de ***n*** ao intervalo ***i*** pode ser representada como:

$$\mathbf{T}i(n) = n + i \qquad (6)$$

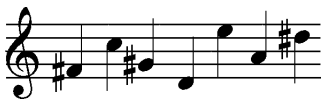
Operações Não-Recursivas e Parcialmente Recursivas

As operações não-recursivas e parcialmente recursivas quando aplicadas a um conjunto de notas modifica algumas de suas características fundamentais tornando impossível a recuperação integral do conjunto original. A principal característica não recuperável consiste na ordem das notas, i.e., uma vez aplicada qualquer destas operações, torna-se impossível ordená-las como no original, a menos que o resultado obtido coincida com o original.

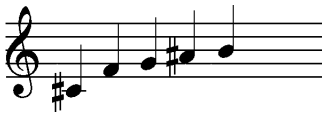
Complementação

A operação complementação produz as notas cromáticas temperadas não presentes no conjunto de notas original. Por exemplo, se aplicada esta operação ao conjunto de notas do Ex. 5a, o conjunto resultante será o do Ex. 5b. O conjunto resultante é ordenado ascendentemente sem considerar as diferenças qualitativas tonais.

Ex. 5a



Ex. 5b

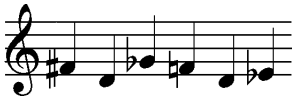


Exclusão de Repetições

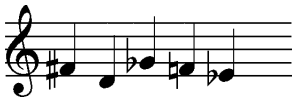
Esta operação exclui as notas repetidas de um conjunto podendo-se eliminar apenas as repetições

tonais, i.e., as notas com a mesma qualidade tonal; ou todas as repetições sem consideração de diferenças enarmônicas. O conjunto de notas do Ex. 6a, após aplicado esta operação com a opção de excluir apenas as repetições tonais resultará no conjunto do Ex. 6b, e com a opção de excluir todas as repetições resultará no conjunto do Ex. 6c.

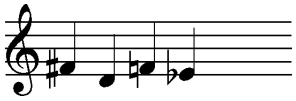
Ex. 6a



Ex. 6b



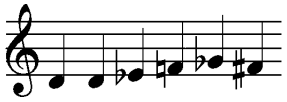
Ex. 6c



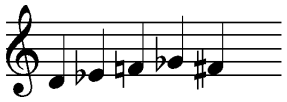
Ordenação

A operação Ordenação redispõe um conjunto de notas em ordem ascendente, considerando-se a freqüência das notas, a começar da mais grave, com ou sem exclusão de notas repetidas. O conjunto de notas do Ex. 6a, após aplicado esta operação com a opção de conservar as repetições resultará no Ex. 7a; com a opção de excluir apenas as repetições tonais resultará no Ex. 7b; e com a opção de excluir todas as repetições no Ex. 7c.

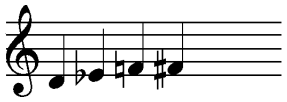
Ex. 7a



Ex. 7b



Ex. 7c



Características de Conjuntos e Notas

As operações constantes desta seção produzem características específicas de notas ou de conjunto de notas. A operação Freqüência opera em notas, Intervalo em duas notas, e todas as restantes em conjuntos de até 12 notas temperadas distintas.

Eixo de Simetria

A operação Eixo de Simetria verifica se o conjunto de notas é simétrico, e se o for, fornece o(s) eixo(s) em torno dos quais o conjunto poderá ser invertido sem perda de seu conteúdo original não considerando as diferenças enarmônicas. Por exemplo, esta operação quando aplicada ao conjunto de notas Do#-Re-Fa#-Sol produzirá os eixos Mi-Sib, significando que se invertido em torno de qualquer destas notas conservará no sistema *stemperado* as mesmas notas originais. A notação utilizada no caso de um eixo coincidente

com uma nota é ***n-t*** e no caso de um eixo entre duas notas é $\boldsymbol{n}_1/\boldsymbol{n}_2$-$\boldsymbol{t}_1/\boldsymbol{t}_2$, onde ***t*** significa um trítono em relação à nota ***n***.

Forma Normal

A Forma Normal ou Ordem Normal consiste na representação de um conjunto de notas de maneira compactada com a finalidade de facilitar a verificação de suas propriedades e de compará-lo com outros conjuntos. A Forma Normal é representada com as notas do conjunto em ordem ascendente e com os menores intervalos próximos ao início. Assim, a Forma Normal para o conjunto 5-10-6-1, aqui representados na codificação *NotaCodigo* no sistema *stemperado*, é 5-6-10-1.

Forma Prima

Assim como a Forma Normal, a Forma Prima consiste na representação de um conjunto de notas de maneira compactada com a finalidade de facilitar a verificação de suas propriedades e de compara-la com outros conjuntos. Existem entretanto duas diferenças fundamentais entre a Forma Prima e a Forma Normal: primeiro, a Forma Prima corresponde à mais compactada entre as Formas Normais de um conjunto ou de sua inversão, e segundo, a Forma Prima é tranposta de modo a iniciar com a classe de notas 0 (Do). Por exemplo, a Forma Prima do conjunto 7-3-9-8-6 é 0-1-2-3-6, considerando que a Forma Normal de sua inversão (3-4-5-6-9) é mais compactada que a sua Forma Normal (3-6-7-8-9).

Freqüência

Esta operação fornece a freqüência em Hz de uma determinada nota. Para o sistema *stemperado* a freqüência é calculada considerando-se o semitono como igual a

$$\sqrt[12]{2} \quad ,$$

e para o sistema *stonal* a freqüência é calculada considerando-se o semitono cromático como igual a (semitono cromático de Pitágoras)

$$\frac{2187}{2048} \quad .$$

Intervalo

Dadas duas notas, o intervalo entre elas pode ser definido como a diferença entre os códigos numéricos correspondentes a estas notas ou, considerando n_1 e n_2 as duas notas dadas o intervalo i entre elas pode ser obtido pela fórmula

$$i = n_1 - n_2. \qquad (7)$$

A operação Intervalo leva em conta os diversos tipos de intervalo, i.e., classe, simples e composto, e a consideração de direcionalidade. Na forma aqui definida, o intervalo pode ser expresso como um código numérico (função Intervalo), ou como um código misto (função IntervaloTonal) resultante da distância diatônica e da qualidade intervalar, além de uma codificação adicional denominada oitava-ponto-intervalo paralela à codificação oitava-ponto-nota.

Vetor Intervalar

O Vetor Intervalar de um conjunto de notas pode ser definido como a listagem do conteúdo de classes intervalares daquele conjunto e indica o número de notas comuns por transposição. O Vetor Intervalar é apresentado como uma literal de 6 elementos, cada um representando o número de ocorrências de uma classe intervalar (ci) começando com a ci 1 e em ordem concluindo com a ci 6. Assim, o Vetor Intervalar do conjunto de notas 10-5-0-8-9-4 é 322431 significando que este conjunto contém 3 ci 1, 2 ci 2, 2 ci 3, 4 ci 4, 3 ci 5, e 1 ci 6 ou, por exemplo, que este conjunto transposto ao intervalo 1 conterá 3 notas em comum com o original.

Vetor Inversão

O Vetor de Inversão ou Vetor de Índices pode ser definido como a listagem das somas das classes de notas de um conjunto e indica o número de notas comuns por inversão. O Vetor de Inversão é apresentado como uma literal de 12 elementos, cada um representando o número de ocorrências dos índices de inversão (o dobro dos eixos de inversão) começando com 0 e em ordem concluindo com 11. Assim, o Vetor de Inversão do conjunto de notas 3-5-2-9-7 é 403032222232, significando que este conjunto contém 4 somas que produzem o índice de inversão 0, 0 que produz o índice 1, etc., ou, por exemplo, que este conjunto invertido em torno do eixo 1 (índice de inversão 2) conterá 3 notas em comum com o original.

CODIFICAÇÃO

Notas

As codificações utilizadas para as notas musicais em OBJNOTA—*NotaCodigo*, *NotaNome*, *NotaLetra*, e *OitavaPontoNota*— tem como referência comum duas codificações numéricas às quais denominamos codificação tonal e codificação temperada.

A codificação tonal, utiliza os valores 0 a 95 como códigos representativos das notas, em um âmbito que inclui desde sétuplo-bemóis a sétuplo-sustenidos, conforme apresentados na Tabela 3. Aqui, como referência, utilizamos os nomes das notas (Do, Re, Mi, Fa, Sol, La, e Si) e, para as alterações, os seguintes símbolos:

hb	sétuplo-bemol	#	sustenido
sb	sêxtuplo-bemol	d#	duplo-sustenido
pb	quíntuplo-bemol	t#	triplo-sustenido
qb	quádruplo-bemol	q#	quádruplo-sustenido
tb	triplo-bemol	p#	quíntuplo-sustenido
db	duplo-bemol	s#	sêxtuplo-sustenido
b	bemol	h#	sétuplo-sustenido
n	natural		

Tabela 3: Códigos de Notas Tonais

	hb	sb	pb	qb	tb	db	b	n	#	d#	t#	q#	p#	s#	h#
Do	-	90	91	92	93	94	95	0	1	2	3	4	5	6	-
Re	7	8	9	10	11	12	13	14	15	16	17	18	19	20	-
Mi	21	22	23	24	25	26	27	28	29	30	31	32	33	34	-
Fa	-	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Sol	-	49	50	51	52	53	54	55	56	57	58	59	60	61	-
La	62	63	64	65	66	67	68	698	70	71	72	73	74	75	-
Si	76	77	78	79	80	81	82	83	84	85	86	87	88	89	-

Para a consideração de registro, adiciona-se ou subtrai-se 96 para cada oitava correspondente à posição da nota. Neste trabalho consideramos a oitava inicial, aquela representada pelos códigos originais, como 0 (zero). Desta forma Sol0 tem como código 55, enquanto Sol1 tem como código 151 (55 + 96) e Sol2 206 (55 + 96 + 96). A anulação do registro é realizada por uma operação módulo (**mod**) 96, um valor que, como veremos, nesta codificação corresponde ao intervalo tonal de oitava justa.

A codificação temperada, utiliza os valores 0 a 11 como códigos representativos das notas, em um âmbito que inclui apenas as doze notas cromáticas sem consideração de diferenças enarmônicas, conforme apresentados na Tabela 4:

Tabela 4: Códigos de Notas Temperadas

	n	#
Do	0	1
Re	2	3
Mi	4	-
Fa	5	6
Sol	7	8
La	9	10
Si	11	-

Para a consideração de registro, adiciona-se ou subtrai-se 12 para cada oitava correspondente à posição da nota. Assim como para a codificação tonal consideramos a oitava inicial, aquela representada pelos códigos originais, como 0 (zero). Desta forma Sol0 tem como código 7, enquanto Sol1 tem como código 19 (7 + 12) e Sol2 31 (7 + 12 + 12). Aqui, a anulação de registro é realizada por uma operação módulo (**mod**) 12 correspondente, nesta codificação, ao intervalo tonal de oitava justa.

Considerando que o valor correspondente à oitava justa da codificação tonal (96) é múltiplo de 12, a operação módulo (**mod**) 12 aplicada à codificação tonal converterá esta para a codificação temperada. Entretanto, uma vez realizada esta operação, torna-se impossível reaver os significados tonais originais.

Além das duas codificações numéricas, utilizamos quatro codificações literais à quais denominamos *NotaCodigo*, *NotaNome*, *NotaLetra* e *OitavaPontoNota*.

A codificação *NotaCodigo* corresponde à codificação numérica — tonal ou temperada — representada como texto. Isto significa que na programação, enquanto as codificações numéricas são tratadas como do tipo inteiro (**integer type**), a *NotaCodigo* é tratada como do tipo literal (**string type**).

A codificação *NotaNome* corresponde aos nomes latinos das notas musicais (Do, Re, Mi, Fa, Sol, La, e Si), utilizando os símbolos para alterações indicados acima. Na codificação tonal, as cinco notas alteradas utilizam apenas o símbolo do sustenido (Do#, Re#, Fa#, Sol#, e La#). A indicação de registro é realizado pela indicação da oitava após o nome da nota ou do símbolo da alteração.

A codificação *NotaLetra* corresponde às letras com que se designam as notas musicais (A, B, C, D, E, F, e G). Assim como em *NotaNome*, os símbolos para alterações são os mesmos indicados acima e as cinco notas alteradas utilizam apenas o símbolo do sustenido (C#, D#, F#, G#, e A#). A indicação de registro é realizado pela indicação da oitava após o nome da nota ou do símbolo da alteração.

A codificação *OitavaPontoNota* corresponde à codificação numérica—tonal ou temperadada—oitava inicial representada como parte decimal e pela indicação da oitava como parte inteira de um número do tipo real (**real type**), representada como texto. Assim Do#1 (C#1) é representado por 1.01 e Sib4 (Bb4) por 4.82 no sistema tonal e 4.10 no sistema temperado. Quando o registro é desconsiderado, a parte inteira, a qual corresponde à oitava, é sempre 0 (zero).

Intervalos

Assim como para as notas musicais, as codificações utilizadas para os intervalos musicais — *Intervalo* e *IntervaloTonal* — e para os tipos de intervalos musicais — *iclasse*, *isimples*, *icomposto*, e *ioitavaponto* — têm como referência comum duas codificações numéricas paralelas às codificações das notas.

A primeira é derivada e compatível com a codificação tonal, tendo 96 como código para o oitava justa, conforme apresentado na Tabela 5. Nesta tabela utilizamos os graus da escala diatônica e, para a qualidade intervalar, os seguintes símbolos:

sd	sêxtuplo-diminuto	M	maior
pd	quíntuplo-diminuto	A	aumentado
qd	quádruplo-diminuto	dA	duplo-aumentado
td	triplo-diminuto	tA	triplo-aumentado
dd	duplo-diminuto	qA	quádruplo-aumentado
d	diminuto	pA	quíntuplo-aumentado
m	menor	sA	sêxtuplo-aumentado
J	justo	hA	sétuplo-aumentado

Tabela 5: Códigos de Intervalos Tonais

	sd	pd	qd	td	dd	d	m	J	M	A	dA	tA	qA	pA	sA	hA
1ª	-	-	-	-	-	-	-	0	-	1	2	3	4	5	6	-
2ª	7	8	9	10	11	12	13	-	14	15	16	17	18	19	20	-
3ª	21	22	23	24	25	26	27	-	28	29	39	31	32	33	34	-
4ª	35	36	37	38	39	40	-	41	-	42	43	44	45	46	47	48
5ª	49	50	51	52	53	54	-	55	-	56	57	58	59	60	61	-
6ª	62	63	64	65	66	67	68	-	69	70	71	72	73	74	75	-
7ª	76	77	78	79	80	81	82	-	83	84	85	86	87	88	89	-
8ª	90	91	92	93	94	95	-	96	-	-	-	-	-	-	-	-

Para os intervalos compostos, adiciona-se ou subtrai-se 96 para cada oitava adicional. Assim, enquanto o intervalo de terça-menor tem como código 27, o intervalo de décima-menor tem como código 123 (27 + 96). A transformação de um intervalo composto em simples é realizada por uma operação módulo (**mod**) 96.

A segunda codificação é derivada e compatível com a codificação temperada, tendo 12 como código para o oitava justa, confome apresentado na Tabela 6.

Tabela 6: Códigos de Intervalos Temperados

	m	J	M	A
1ª	-	0	-	-
2ª	1	-	2	-
3ª	3	-	4	-
4ª	-	5	-	6
5ª	-	7	-	-
6ª	8	-	9	-
7ª	10	-	11	-
8ª	-	12	-	-

Para os intervalos compostos, adiciona-se ou subtrai-se 12 para cada oitava adicional. Assim, enquanto o intervalo de terça-menor tem como código 3, o intervalo de décima-menor tem como código 15 (3 + 12). A transformação de um intervalo composto em simples é realizada por uma operação módulo (**mod**) 12.

As codificações dos intervalos podem ainda indicar a consideração de ordem intervalar. Neste caso, os intervalos descendentes serão representados por um sinal negativo à frente do código.

Assim como nas operações com notas, é possível converter os intervalos do sistema tonal em intervalos do sistema temperado, bastando para isto efetuar a operação módulo (**mod**) 12. Aqui também os significados originais são perdidos, tornando impossível recuperá-los.

A codificação a que denominamos *Intervalo* corresponde às codificações numéricas representada como texto, à semelhança da codificação *NotaCodigo* utilizada para as notas.

A codificação a que denominamos *IntervaloTonal* é composta de duas partes. A primeira correspondente à distância em graus

de escala e a segunda à qualidade intervalar, por exemplo, 3m (terça-menor), 4A (quarta-aumentada), 1dd (prima duplo-diminuta).

Os tipos de intervalos musicais têm os seguintes significados:

iclasse = classes intervalares;
isimples = apenas intervalos simples;
icomposto = intervalos simples e compostos; e
ioitavaponto = notação oitava ponto intervalo.

Por exemplo, o intervalo de décima-segunda aumentada (décima-terceira menor) pode ser representado nas formas indicadas na tabela seguinte:

Tabela 7: Tipos de Intervalos

	Intervalo		*IntervaloTonal*	
	sistema tonal	sistema temperado	sistema tonal	sistema temperado
icomposto	152	20	12A	13m
isimples	56	8	5A	6m
iclasse	40	4	4d	3M
ioitavaponto	1.56	1.08	1.5A	1.6m

REFERÊNCIA DAS GLOBAIS

considerar_direcao constante

Declaração `considerar_direcao = true;`

Função Usada na inicialização do objeto *OIntervalo* para determinar que os intervalos sejam ordenados, isto é, que a direção intervalar seja considerada. A utilização desta constante gera intervalos ascendentes e descendentes, estes últimos representados pelo sinal negativo à frente do código.

considerar_registro constante

Declaração `considerar_registro = true;`

Função Usada na inicialização dos objetos *ONota*, *OConjunto*, *NotaCodigo*, *OitavaPontoNota*, *NotaNome* e *NotaLetra* para determinar que o registro das notas, i.e., a oitava, seja considerado. A utilização desta constante gera notas em todos os registros.

desconsiderar_direcao constante

Declaração `desconsiderar_direcao = false;`

Função Usada na inicialização do objeto *OIntervalo* para determinar que os intervalos não sejam ordenados, isto é, que a direção intervalar não seja considerada. A utilização desta constante gera intervalos independentes de suas direções, ascendentes ou descendentes, tal como acontece em uma estrutura vertical (acordes).

desconsiderar_registro constante

Declaração `desconsiderar_registro = false;`

Função Usada na inicialização dos objetos *ONota*, *OConjunto*, *NotaCodigo*, *OitavaPontoNota*, *NotaNome* e *NotaLetra* para determinar que o registro das notas, i.e., a oitava, não seja considerado. A utilização desta constante gera classes de notas.

iclasse constante

Declaração `iclasse = 0;`

Função Usada na inicialização do objeto *OIntervalo* para determinar a geração de classes intervalares.

icomposto constante

Declaração `icomposto = 2;`

Função Usada na inicialização do objeto *OIntervalo* para determinar que os intervalos sejam utilizados em sua plenitude. A utilização desta constante gera intervalos simples e compostos.

ioitavaponto constante

Declaração `ioitavaponto = 3;`

Função Usada na inicialização do objeto *OIntervalo* para determinar a geração de intervalos na notação oitava-ponto-intervalo.

isimples constante

Declaração `isimples = 1;`

Função Usada na inicialização do objeto *OIntervalo* para determinar a geração de intervalos simples independentemente da distância entre as notas.

NotaCodigo tipo

Declaração
```
NotaCodigo = object (OConjunto)
  constructor Iniciar (ntotal: integer; sistema:
      byte; reg: boolean; var codigo_inicial);
  function NomeParaCodigo (item: texto6):
      integer; virtual;
  function CodigoParaNome (item: integer):
      texto6; virtual;
end;
```

Função Objeto descendente de *OConjunto* que utiliza codificação numérica, em **string,** para indicar as notas musicais.

NotaLetra tipo

Declaração
```
NotaLetra = object (OConjunto)
  constructor Iniciar (ntotal: integer; sistema:
      byte; reg: boolean; var letra_inicial);
  function NomeParaCodigo (item: texto6):
      integer; virtual;
  function CodigoParaNome (item: integer):
      texto6; virtual;
end;
```

Função Objeto descendente de *OConjunto* que utiliza como códigos as letras correspondentes às notas musicais.

NotaNome tipo

Declaração

```
NotaNome = object (OConjunto)
  constructor Iniciar (ntotal: integer; sistema:
      byte; reg: boolean; var nome_inicial);
  function NomeParaCodigo (item: texto6):
      integer; virtual;
  function CodigoParaNome (item: integer):
      texto6; virtual;
end;
```

Função Objeto descendente de *OConjunto* que utiliza como códigos os nomes correspondentes às notas musicais.

OConjunto tipo

Declaração

```
OConjunto = object (ONota)
  total_notas: word;
  codigo: pvetorinteiro;
  constructor Iniciar (ntotal: integer; sistema:
      byte; reg: boolean);
  destructor Finalizar; virtual;
  procedure NovaNota (indice: integer;
      nova_nota: texto6);
  function NotaAtual (indice: integer): texto6;
  procedure NovoConjunto (var novos_codigos);
  procedure ConjuntoAtual (var codigos_atuais);
  function RegistroAtual (indice: integer):
      byte;
  function TotalNotas: word;
  procedure NovoTotal (novo_total: word);
  procedure Transferir (var objeto: OConjunto);
  function Enarmonica (enota: texto6; direcao:
      shortint): texto6;
  function Intervalo (objeto: OIntervalo; nota1,
      nota2: texto6): texto6;
  function IntervaloTonal (objeto: OIntervalo;
      nota1, nota2: texto6): texto6;
  procedure Transpor (tintervalo: texto6);
  procedure Inverter (eixo_inversao: texto12);
  procedure Multiplicar (multiplicador:
      integer);
  procedure Retrogradar;
  procedure Rotar (posicao: integer);
  procedure ExcluirRepeticao (opcao: byte);
  procedure Ordenar (opcao: byte);
  function VetorIntervalar: texto6;
  function VetorInversao: texto12;
```

```
  procedure FormaNormal;
  procedure FormaPrima;
  procedure Complemento;
  function EixoSimetria: string;
  function Frequencia (inota: texto6): real;
  function CodigoParaIntervalo (cod: integer):
      texto6;
  function IntervaloParaCodigo (tintervalo:
      texto6): integer;
  function NomeParaCodigo (item: texto6):
      integer; virtual;
  function CodigoParaNome (item: integer):
      texto6; virtual;
private
  procedure VetorIntervalo (var Ivetor:
      OConjunto);
  procedure OrdemIntervalar (var externo:
      integer);
  function NotaIndice (ele: texto6): integer;
end;
```

Função Objeto básico, descendente de *ONota*, contendo todas as operações básica utilizadas pelos seus objetos descendentes *NotaCodigo*, *OitavaPontoNota*, *NotaNome* e *NotaLetra.*

OIntervalo tipo

Declaração

```
OIntervalo = object
  tipo: 0..3;
  direcao: boolean;
  constructor Iniciar (tipo_de_intervalo: byte;
      direcao_inicial: boolean);
end;
```

Função Objeto básico cuja inicialização determina o tipo de intervalo musical a ser utilizado por outros objetos.

OitavaPontoNota tipo

Declaração

```
OitavaPontoNota = object (OConjunto)
  constructor Iniciar (ntotal: integer; sistema:
      byte; reg: boolean; var opn_inicial);
  function NomeParaCodigo (item: texto6):
      integer; virtual;
  function CodigoParaNome (item: integer):
      texto6; virtual;
end;
```

Função Objeto descendente de *OConjunto* que utiliza codificação numérica, em **string,** para indicar as notas musicais. A parte inteira representa o registro (oitava) e a decimal a nota musical.

ONota tipo

Declaração

```
ONota = object
  oitava: byte;
  classe: boolean;
  constructor Iniciar (sistema: byte; reg:
      boolean);
private
  function Modulo (elemento: integer): integer;
  function Linha (item: integer): byte;
  function Coluna (item: integer): byte;
  function NpC (item: texto6; notas: string):
      integer;
  function CpN (item: integer; notas: string):
      texto6;
end;
```

Função Objeto básico antecessor de *OConjunto*.

PNotaCodigo tipo

Declaração

```
PNotaCodigo = ^NotaCodigo;
```

Função Ponteiro (**pointer**) para o objeto *NotaCodigo*.

PNotaLetra tipo

Declaração

```
PNotaLetra = ^NotaLetra;
```

Função Ponteiro (**pointer**) para o objeto *NotaLetra*.

PNotaNome tipo

Declaração

```
PNotaNome = ^NotaNome;
```

Função Ponteiro (**pointer**) para o objeto *NotaNome*.

POConjunto tipo

Declaração `POConjunto = ^OConjunto;`

Função Ponteiro (**pointer**) para o objeto *OConjunto*.

POIntervalo tipo

Declaração `POIntervalo = ^OIntervalo;`

Função Ponteiro (**pointer**) para o objeto *OIntervalo.*

POitavaPontoNota tipo

Declaração `POitavaPontoNota = ^OitavaPontoNota;`

Função Ponteiro (**pointer**) para o objeto *OitavaPontoNota*.

PONota tipo

Declaração `PONota = ^ONota;`

Função Ponteiro (**pointer**) para o objeto *ONota*.

pvetorinteiro tipo

Declaração `pvetorinteiro = ^vetorinteiro;`

Função Ponteiro (**pointer**) para o tipo *vetorinteiro*.

pvetortexto tipo

Declaração `pvetortexto = ^vetortexto;`

Função Ponteiro (**pointer**) para o tipo *vetortexto*.

stemperado — constante

Declaração `stemperado = 12;`

Função Usada na inicialização dos objetos *ONota*, *OConjunto*, *NotaCodigo*, *OitavaPontoNota*, *NotaNome* e *NotaLetra* para determinar que as notas enarmônicas equivalentes sejam desconsideradas. A utilização desta constante gera notas numeradas de 0 a 11.

stonal — constante

Declaração `stonal = 96;`

Função Usada na inicialização dos objetos *ONota*, *OConjunto*, *NotaCodigo*, *OitavaPontoNota*, *NotaNome* e *NotaLetra* para determinar que as diferenças enarmônicas sejam consideradas. A utilização desta constante gera notas numeradas de 0 a 95, permitindo que as peculiaridades da música tonal resultem corretas.

texto12 — tipo

Declaração `texto12 = string[12];`

Função Tipo (**type**) **string** contendo no máximo 12 caracteres. Usado para variáveis destinadas a armazenar o eixo de inversão no método *Inverter* e o vetor de inversão no método *VetorInversao*.

texto6 — tipo

Declaração `texto6 = string[6];`

Função Tipo (**type**) **string** contendo no máximo 6 caracteres. Usado para variáveis destinadas a armazenar os códigos das notas e dos intervalos em diversos métodos e o vetor intervalar no método *VetorIntervalar*.

vetorinteiro tipo

Declaração `vetorinteiro = array [1..1] of integer;`

Função Tipo (**type**) **array of integer** contendo inicialmente um único elemento. Usado para armazenar os códigos em números correspondentes às notas musicais.

vetortexto tipo

Declaração `vetortexto= array [1..1] of texto6;`

Função Tipo (**type**) **array of** *texto6* contendo inicialmente um único elemento. Usado para armazenar os códigos em texto (**string**) correspondentes às notas musicais.

REFERÊNCIA DOS OBJETOS

NotaCodigo

Antecessor `OConjunto`

Métodos

Iniciar `constructor Iniciar (ntotal: integer; sistema: byte; reg: boolean; var codigo_inicial);`

Cria um conjunto de *ntotal* notas musicais na codificação *NotaCodigo* contidas em *codigo_inicial*, no sistema tonal ou temperado de acordo com *sistema,* e com ou sem consideração de registro de acordo com *reg.* O parâmetro variável sem tipo (**untyped variable parameter**) *codigo_inicial* deve ser definido como um vetor (**array**) de *texto6* com *ntotal* elementos, contendo as notas codificadas de acordo com *sistema* e *reg*.

Exemplo:

```
var
  indice   : byte;
  conjunto : NotaCodigo;
const
  notas : array [1..4] of texto6 = ('4', '30',
      '24', '103');

begin
  with conjunto do
    begin
      Iniciar (4, stemperado,
      desconsiderar_registro, notas);
      for indice := 1 to 4 do
        write (NotaAtual (indice):4);
      writeln;
      Finalizar
    end
end.
```

Resultado:

```
   4   6   0   7
```

Codigo ParaNome

```
function CodigoParaNome (item: integer): texto6;
virtual;
```

Converte o **integer** *item*, correspondente ao código da nota, para uma **string** na codificação *NotaCodigo*.

Exemplo:

```
var
  conjunto : NotaCodigo;
  notas      : texto6;

begin
  with conjunto do
    begin
       Iniciar (1, stonal,
         desconsiderar_registro, notas);
      writeln (CodigoParaNome (5));
      Finalizar;
    end
end.
```

Resultado:

```
5
```

Nome ParaCodigo

```
function NomeParaCodigo (item: texto6): integer;
virtual;
```

Converte a **string** *item* na codificação *NotaCodigo*, para um **integer**, correspondente ao código da nota.

Exemplo:

```
var
  conjunto : NotaCodigo;
  notas    : texto6;

begin
  with conjunto do
    begin
      Iniciar (1, stemperado,
         considerar_registro, notas);
      writeln (NomeParaCodigo ('11'));
      Finalizar;
    end
end.
```

Resultado:

```
11
```

NotaLetra

Antecessor OConjunto

Métodos

Iniciar

```
constructor Iniciar (ntotal: integer; sistema: byte;
reg: boolean; var letra_inicial);
```

Cria um conjunto de *ntotal* notas musicais na codificação *NotaLetra* contidas em *letra_inicial*, no sistema tonal ou temperado de acordo com *sistema*, e com ou sem consideração de registro de acordo com *reg*. O parâmetro variável sem tipo

(**untyped variable parameter**) *letra_inicial* deve ser definido como um vetor (**array**) de *texto6* com *ntotal* elementos, contendo as notas codificadas de acordo com *sistema* e *reg*.

Exemplo:

```
var
  indice   : byte;
  conjunto : NotaLetra;


const
  notas : array [1..4] of texto6 = ('E3', 'Gb2',
      'C2', 'G5');


begin
  conjunto.Iniciar (4, stonal,
      desconsiderar_registro, notas);
  for indice := 1 to 4 do
    write (conjunto.NotaAtual (indice):4);
  writeln;
  conjunto.Finalizar;
end.
```

Resultado:

```
   E   Gb   C   G
```

Codigo ParaNome

```
function CodigoParaNome (item: integer): texto6;
virtual;
```

Converte o **integer** *item*, correspondente ao código da nota, para uma **string** na codificação *NotaLetra*.

Exemplo:

```
var
  conjunto : NotaLetra;
  notas    : texto6;


begin
  with conjunto do
    begin
      Iniciar (1, stonal,
         desconsiderar_registro, notas);
      writeln (CodigoParaNome (16));
      Finalizar;
    end
end.
```

Resultado:

```
Dd#
```

Nome ParaCodigo

```
function NomeParaCodigo (item: texto6): integer;
virtual;
```

Converte a **string** *item* na codificação *NotaLetra*, para um **integer**, correspondente ao código da nota.

Exemplo:

```
var
  conjunto : NotaLetra;
  notas    : texto6;

begin
  with conjunto do
    begin
      Iniciar (1, stemperado,
         considerar_registro, notas);
      writeln (NomeParaCodigo ('Eb2'));
      Finalizar;
    end
end.
```

Resultado:

```
27
```

NotaNome

Antecessor `OConjunto`

Métodos

Iniciar

```
constructor Iniciar (ntotal: integer; sistema: byte;
reg: boolean; var nome_inicial);
```

Cria um conjunto de *ntotal* notas musicais na codificação *NotaNome* contidas em *letra_inicial*, no sistema tonal ou temperado de acordo com *sistema*, e com ou sem consideração de registro de acordo com *reg*. O parâmetro variável sem tipo

(**untyped variable parameter**) *nome_inicial* deve ser definido como um vetor (**array**) de *texto6* com *ntotal* elementos, contendo as notas codificadas de acordo com *sistema* e *reg*.

Exemplo:

```
var
  indice   : byte;
  conjunto : NotaNome;


const
  notas : array [1..4] of texto6 = ('Fab3',
      'Solb2', 'Do2', 'Sol5');


begin
  conjunto.Iniciar (4, stemperado,
      desconsiderar_registro, notas);
  for indice := 1 to 4 do
    write (conjunto.NotaAtual (indice):6);
  writeln;
  conjunto.Finalizar;
end.
```

Resultado:

```
    Mi   Fa#    Do   Sol
```

Codigo ParaNome

```
function CodigoParaNome (item: integer): texto6;
virtual;
```

Converte o **integer** *item*, correspondente ao código da nota, para uma **string** na codificação *NotaNome*.

Exemplo:

```
var
  conjunto : NotaNome;
  notas    : texto6;


begin
  with conjunto do
    begin
      Iniciar (1, stemperado,
         desconsiderar_registro, notas);
      writeln (CodigoParaNome (8));
      Finalizar;
    end
end.
```

Resultado:

```
Sol#
```

Nome ParaCodigo

```
function NomeParaCodigo (item: texto6): integer;
virtual;
```

Converte a **string** *item* na codificação *NotaNome*, para um **integer**, correspondente ao código da nota.

Exemplo:

```
var
  conjunto : NotaNome;
  notas      : texto6;


begin
  with conjunto do
    begin
      Iniciar (1, stemperado,
         considerar_registro, notas);
      writeln (NomeParaCodigo ('Sol#3'));
      Finalizar;
    end
end.
```

Resultado:

```
44
```

OConjunto

Antecessor `ONota`

Descendentes `NotaCodigo`
`OitavaPontoNota`
`NotaNome`
`NotaLetra`

Campos

total_notas

```
total_notas: word;
```

O total de notas do conjunto.

codigo `codigo: pvetorinteiro;`
Ponteiro (**pointer**) para o vetor (**array**) contendo os códigos numéricos das notas musicais.

Métodos

Iniciar `constructor Iniciar (ntotal: integer; sistema: byte; reg: boolean);`
Usado por todos os objetos descendentes para inicializar o total de notas *total_notas* fornecido por *ntotal*, criar o ponteiro (**pointer**) *codigo* e reservar espaço na memória para o conjunto de notas musicais. O sistema (tonal ou temperado) é armazenado em *sistema*, e a consideração de registro em *reg*.

Finalizar `destructor Finalizar; virtual;`
Usado por todos os objetos descendentes para limpar as variáveis armazenadas e liberar a memória reservada para o conjunto de notas musicais.

CodigoPara Intervalo `function CodigoParaIntervalo (cod: integer): texto6;`
Retorna o intervalo musical como uma variável do tipo (**type**) *texto6*, correspondente ao código numérico do tipo (**type**) inteiro (**integer**) *cod*.

Exemplo:

```
var
  conjunto : NotaCodigo;
  notas    : texto6;

begin
  with conjunto do
    begin
      Iniciar (1, stonal, considerar_registro,
         notas);
      writeln (CodigoParaIntervalo (13));
      Finalizar
    end
end.
```

Resultado:

```
2m
```

Codigo ParaNome

```
function CodigoParaNome (item: integer): texto6; virtual;
```

Método abstrato usado por todos os objetos descendentes para converter o código numérico da nota musical para o código apropriado.

Eixo Simetria

```
function EixoSimetria: string;
```

Retorna, como uma variável do tipo (**type**) **string**, o(s) eixo(s) de simetria do conjunto. Caso o conjunto seja assimétrico retornará uma **string** vazia.

Exemplo:

```
var
  conjunto : NotaNome;

const
  notas : array [1..4] of texto6 = ('Re', 'Fa',
      'Lab', 'Si');

begin
  with conjunto do
    begin
      Iniciar (4, stonal,
         desconsiderar_registro, notas);
      writeln (EixoSimetria);
      Finalizar
    end
end.
```

Resultado:

```
 Do/Do#-Fa#/Sol Re-Sol# Re#/Mi-La/La# Fa-Si
```

Enarmonica

```
function Enarmonica (enota: texto6; direcao: shortint):
texto6;
```

Retorna, como uma variável do tipo (**type**) *texto6*, a nota enarmonica correspondente a *enota* na direção indicada por *direcao*. O parâmetro *direcao*

indicará a nota musical, correspondente ao grau de escala acima (positivo) ou abaixo (negativo), com a qual será feita a enarmonização. O parâmetro *enota* poderá ser uma nota musical ou um índice apontando para uma nota do conjunto, caso em que deverá ser antecedido pelo caráter "i".

Exemplo:

```
var
  conjunto : NotaNome;
  indice   : byte;

const
  notas : array [1..2] of texto6 = ('Solb',
      'Re');

begin
  with conjunto do
    begin
      Iniciar (3, stonal,
         desconsiderar_registro, notas);
      writeln (Enarmonica ('Lad#', 1));
      writeln (Enarmonica ('Lad#', 2));
      NovaNota (1, Enarmonica ('i1', -1));
      for indice := 1 to 2 do
        write (NotaAtual (indice):4);
      writeln;
      Finalizar
    end
end.
```

Resultado:

```
Si
Dob
 Fa#  Re
```

Frequencia `function Frequencia (inota: texto6): real;`

Retorna, como uma variável do tipo (**type**) **real**, a frequência em Hertz da nota musical *inota*. O parâmetro *inota* poderá ser uma nota musical ou um índice apontando para uma nota do conjunto, caso em que deverá ser antecedido pelo caráter "i".

Exemplo:

```
var
  conjunto : NotaNome;
  notas    : texto6;

begin
  with conjunto do
    begin
      Iniciar (1, stonal, considerar_registro,
         notas);
      writeln (Frequencia ('La4'):4:2);
      Finalizar
    end
end.
```

Resultado:

```
440.00
```

Intervalo `function Intervalo (Objeto: OIntervalo; nota1, nota2: texto6): texto6;`

Retorna, como uma variável do tipo (**type**) *texto6*, o intervalo entre *nota1* e *nota2* do tipo definido por *Objeto*. O intervalo resultante estará em codificação numérica convertida para **string**. Os intervalos descendentes estarão indicados pelo sinal negativo. Os parâmetros *nota1* e *nota2* poderão ser notas musicais ou índices apontando para notas do conjunto, caso em que deverão ser antecedidos pelo caráter "i".

Exemplo:

```
var
  conjunto : NotaCodigo;
  interv   : OIntervalo;

const
  notas : array [1..4] of texto6 = ('3', '11',
      '8', '6');

begin
  interv.Iniciar (iclasse, desconsiderar_direcao);
  with conjunto do
    begin
      Iniciar (4, stemperado,
         desconsiderar_registro, notas);
      writeln (Intervalo (interv, 'i1', 'i2'));
      Finalizar
    end
end.
```

Resultado:

```
4
```

IntervaloParaCodigo

```
function IntervaloParaCodigo (tintervalo: texto6): integer;
```

Retorna o intervalo musical como uma variável do tipo (**type**) **integer**, correspondente ao código tonal do tipo (**type**) *texto6*.

Exemplo:

```
var
  conjunto : OitavaPontoNota;
  notas    : texto6;

begin
  conjunto.Iniciar (1, stonal,
      desconsiderar_registro, notas);
  writeln (conjunto.IntervaloParaCodigo ('3m'));
  conjunto.Finalizar
end.
```

Resultado:

```
27
```

IntervaloTonal

```
function IntervaloTonal (Objeto: OIntervalo; nota1,
nota2: texto6): texto6;
```

Retorna, como uma variável do tipo (**type**) *texto6*, o intervalo entre *nota1* e *nota2* do tipo definido por *Objeto*. O intervalo resultante estará em codificação tonal. Os intervalos descendentes estarão indicados pelo sinal negativo. Os parâmetros *nota1* e *nota2* poderão ser notas musicais ou índices apontando para notas do conjunto, caso em que deverão ser antecedidos pelo caráter "i".

Exemplo:

```
var
  conjunto : NotaLetra;
  interv   : OIntervalo;
```

```
const
  notas : array [1..2] of texto6 = (‘C#4’, ‘Ab3’);

begin
  interv.Iniciar (icomposto,
      considerar_direcao);
  with conjunto do
    begin
      Iniciar (2, stonal, considerar_registro,
         notas);
      writeln (IntervaloTonal (interv, ‘i1’,
      ‘i2’));
      Finalizar
    end
end.
```

Resultado:

```
-3A
```

NomeParaCodigo

```
function NomeParaCodigo (item: texto6): integer;
virtual;
```

Método abstrato usado por todos os objetos descendentes para converter o código utilizado pelo objeto em código numérico.

NotaAtual

```
function NotaAtual (indice: integer): texto6;
```

Retorna, como uma variável do tipo (**type**) *texto6*, a nota musical indicada por *indice* na codificação definida pelo objeto descendente. O parâmetro *indice* deve ser menor ou igual a *total_notas*, isto é, deve indicar a posição de uma nota existente no conjunto.

Exemplo:

```
var
  conjunto : OitavaPontoNota;

const
  notas : array [1..4] of texto6 = (‘3.04’,
      ‘2.06’, ‘2.00’, ‘5.07’);
```

```
begin
  with conjunto do
    begin
      Iniciar (4, stemperado,
         considerar_registro, notas);
      writeln (NotaAtual (3));
      Finalizar;
    end
end.
```

Resultado:

```
2.00
```

Registro Atual

```
function RegistroAtual (indice: integer): byte;
```

Retorna, como uma variável do tipo (**type**) **byte**, a oitava (registro) correspondente à nota indicada por *indice*. O parâmetro *indice* deve ser menor ou igual a *total_notas*, isto é, deve indicar a posição de uma nota existente no conjunto.

Exemplo:

```
var
  conjunto : NotaCodigo;


const
  notas : array [1..4] of texto6 = ('35', '22',
      '52', '55');


begin
  conjunto.Iniciar (4, stemperado,
      considerar_registro, notas);
  writeln (conjunto.RegistroAtual (4));
  conjunto.Finalizar;
end.
```

Resultado:

```
4
```

TotalNotas

```
function TotalNotas: word;
```

Retorna, como uma variável do tipo (**type**) **word**, o total de notas do conjunto.

Exemplo:

```
var
  conjunto : OitavaPontoNota;

const
  notas : array [1..3] of texto6 = (‘3.03’,
      ‘2.11’, ’2.00’);

begin
  conjunto.Iniciar (3, stemperado,
      considerar_registro, notas);
  writeln (conjunto.TotalNotas);
  conjunto.Finalizar;
end.
```

Resultado:

```
3
```

Vetor Intervalar

```
function VetorIntervalar: texto6;
```

Retorna, como uma variável do tipo (**type**) *texto6*, o vetor intervalar do conjunto.

Exemplo:

```
var
  conjunto : NotaCodigo;

const
  notas : array [1..4] of texto6 = (‘11’, ‘4’,
      ‘7’, ‘6’);

begin
  conjunto.Iniciar (4, stemperado,
      desconsiderar_registro, notas);
  writeln (conjunto.VetorIntervalar);
  conjunto.Finalizar
end.
```

Resultado:

```
111120
```

Vetor Inversao

```
function VetorInversao: texto12;
```

Retorna, como uma variável do tipo (**type**) texto12, o vetor de inversão do conjunto.

Exemplo:

```
var
  conjunto : NotaNome;

const
  notas : array [1..5] of texto6 = (‘Do’, ‘Reb’,
      ‘Lab’, ‘Fa#’, ‘Mi’);

begin
  with conjunto do
    begin
      Iniciar (5, stemperado,
         desconsiderar_registro, notas);
      writeln (VetorInversao);
      Finalizar
    end
end.
```

Resultado:

```
423032223220
```

Complemento

```
procedure Complemento;
```

Substitui o conjunto original por seu complemento ordenado ascendentemente.

Exemplo:

```
var
  conjunto : NotaLetra;
  indice   : byte;

const
  notas : array [1..5] of texto6 = (‘F#’, ‘D’,
      ‘G#’, ‘A’, ‘Eb’);

begin
  with conjunto do
    begin
      Iniciar (5, stonal,
         desconsiderar_registro, notas);
      Complemento;
      for indice := 1 to TotalNotas do
        write (NotaAtual (indice):4);
      writeln;
      Finalizar
    end
end.
```

Resultado:

```
   C  C#   E   F   G  A#   B
```

Conjunto Atual

```
procedure ConjuntoAtual (var codigos_atuais);
```

Retorna em *codigos_atuais* o conjunto atual, na codificação definida pelo objeto descendente.

Exemplo:

```
var
  indice   : byte;
  conjunto : NotaLetra;
  atuais   : array [1..4] of texto6;

const
  notas : array [1..4] of texto6 = ('E3', 'Gb2',
     'C2', 'G5');
begin
  with conjunto do
    begin
      Iniciar (4, stonal, considerar_registro,
         notas);
      ConjuntoAtual (atuais);
      for indice := 1 to 4 do
         write (atuais [indice]:4);
      writeln;
      Finalizar
     end
end.
```

Resultado:

```
  E3 Gb2  C2  G5
```

Excluir Repeticao

```
procedure ExcluirRepeticao (opcao: byte);
```

Substitui o conjunto por outro onde as repetições de notas musicais do original são eliminadas de acordo com *opcao*. O parâmetro *opcao* pode assumir os seguintes valores e significados:

1 = exclui apenas as repetições tonais;

2 = exclui todas as repetições.

Exemplo:

```
var
  conjunto : NotaNome;
  indice   : byte;
```

```
const
  notas : array [1..7] of texto6 = (‘Do#’,
      ‘Reb’, ‘Fa#’, ‘Do#’, ‘Sol’, ‘La#’, ‘Sib’);


begin
  with conjunto do
    begin
      Iniciar (7, stonal,
         desconsiderar_registro, notas);
      ExcluirRepeticao (1);
      for indice := 1 to TotalNotas do
         write (NotaAtual (indice):6);
      writeln;
      ExcluirRepeticao (2);
      for indice := 1 to TotalNotas do
         write (NotaAtual (indice):6);
      writeln;
      Finalizar
    end
end.
```

Resultado:

```
   Do#   Reb   Fa#   Sol   La#   Sib
   Do#   Fa#   Sol   La#
```

Forma Normal

```
procedure FormaNormal;
```

Substitui o conjunto por outro correspondente ao original disposto em forma normal.

Exemplo:

```
var
  conjunto : OitavaPontoNota;
  indice   : byte;


const
  notas : array [1..4] of texto6 = (‘3.27’,
      ‘3.14’, ‘2.41’, ‘2.55’);


begin
  with conjunto do
    begin
      Iniciar (4, stonal, considerar_registro,
         notas);
      FormaNormal;
      for indice := 1 to 4 do
        write (NotaAtual (indice):6);
      writeln;
      Finalizar
    end
end.
```

Resultado:

```
  3.14  3.27  2.41  2.55
```

Forma Prima

```
procedure FormaPrima;
```

Substitui o conjunto por outro correspondente ao original disposto em forma prima.

Exemplo:

```
var
  conjunto : NotaCodigo;
  indice   : byte;


const
  notas : array [1..5] of texto6 = ('7', '10',
      '3', '4', '1');


begin
  with conjunto do
    begin
      Iniciar (5, stemperado,
         desconsiderar_registro, notas);
      FormaPrima;
      for indice := 1 to TotalNotas do
        write (NotaAtual (indice):4);
      writeln;
      Finalizar
    end
end.
```

Resultado:

```
   0   1   3   6   9
```

Inverter

```
procedure Inverter (eixo_inversao: texto12);
```

Substitui o conjunto por outro correspondente ao original invertido em torno do eixo *eixo_inversao*. O parâmetro *eixo_inversao* deve ser compatível com a codificação definida pelo objeto descendente e pode corresponder a uma determinada nota musical ou a um ponto entre duas notas musicais. Neste último caso o eixo é representado pelas notas adjacentes separadas por uma barra "/". Adicionalmente, *eixo_inversao* poderá ser

representado por notas musicais ou por índices apontando para notas do conjunto antecedidos pelo caráter "i".

Exemplo:

```
var
  conjunto : NotaNome;
  indice     : byte;
const
  notas : array [1..4] of texto6 = ('Mib4',
      'Sol4', 'Fa#4', 'Si3');


begin
  with conjunto do
    begin
      Iniciar (4, stonal, considerar_registro,
         notas);
      Inverter ('Re4');
      for indice := 1 to 4 do
        write (NotaAtual (indice):6);
      writeln;
      Finalizar
    end
end.
```

Resultado:

```
  Do#4   La3  Sib3   Fa4
```

Multiplicar `procedure Multiplicar (multiplicador: integer);`

Substitui o conjunto por outro correspondente ao original multiplicado por *multiplicador*.

Exemplo:

```
var
  conjunto : NotaCodigo;
  indice   : byte;


const
  notas : array [1..6] of texto6 = ('3', '5',
      '10', '7', '2', '9');


begin
  with conjunto do
    begin
     Iniciar (6, stemperado,
         desconsiderar_registro, notas);
     Multiplicar (5);
```

```
    for indice := 1 to 6 do
      write (NotaAtual (indice):4);
    writeln;
    Finalizar
  end
end.
```

Resultado:

```
   3   1   2  11  10   9
```

NovaNota `procedure NovaNota (indice: integer; nova_nota: texto6);`
Substitui a nota musical indicada por *indice* pela nota *nova_nota*. O parâmetro *indice* deve ser menor ou igual a *total_notas*, isto é, deve indicar a posição de uma nota existente no conjunto. O parâmetro *nova_nota* deve estar de acordo com a codificação do objeto descendente.

Exemplo:

```
var
  indice   : byte;
  conjunto : NotaCodigo;


const
  notas : array [1..4] of texto6 = ('4', '6',
      '0', '7');


begin
  with conjunto do
    begin
      Iniciar (4, stemperado,
         desconsiderar_registro, notas);
      for indice := 1 to 4 do
        write (NotaAtual (indice):4);
      writeln;
      NovaNota (2, '8');
      for indice := 1 to 4 do
        write (NotaAtual (indice):4);
      writeln;
      Finalizar;
    end
end.
```

Resultado:

```
   4   6   0   7
   4   8   0   7
```

Novo Conjunto

```
procedure NovoConjunto (var novos_codigos);
```

Substitui o conjunto por outro indicado por *novos_codigos*. O parâmetro *novos_codigos* deve estar de acordo com a codificação do objeto descendente.

Exemplo:

```
var
  indice   : byte;
  conjunto : NotaNome;
  novos    : array [1..4] of texto6;

const
  notas : array [1..4] of texto6 = ('Mi3',
      'Fa#2', 'Do2', 'Sol5');

begin
  conjunto.Iniciar (4, stemperado,
      desconsiderar_registro, notas);
  for indice := 1 to 4 do
    write (conjunto.NotaAtual (indice):6);
  writeln;
  novos[1] := 'Do1';
  novos[2] := 'Do#1';
  novos[3] := 'Re1';
  novos[4] := 'Mib1';
  conjunto.NovoConjunto (novos);
  for indice := 1 to 4 do
    write (conjunto.NotaAtual (indice):6);
  writeln;
  conjunto.Finalizar;
end.
```

Resultado:

```
   Mi   Fa#    Do   Sol
   Do   Do#    Re   Re#
```

NovoTotal

```
procedure NovoTotal (novo_total: word);
```

Modifica o total de notas do conjunto para *novo_total*.

Exemplo:

```
var
  conjunto : NotaNome;
```

```
const
  notas : array [1..3] of texto6 = (‘Mib’, ‘Si’,
      ‘Do’);

begin
  with conjunto do
    begin
      Iniciar (3, stonal,
         desconsiderar_registro, notas);
      write (TotalNotas);
      NovoTotal (4);
      writeln (TotalNotas:3);
      Finalizar
    end
end.
```

Resultado:

```
3  4
```

Ordenar `procedure Ordenar (opcao: byte);`

Substitui o conjunto por outro onde as notas musicais do original estão ordenadas ascendentemente com ou sem eliminação de repetições de acordo com *opcao*. O parâmetro *opcao* pode assumir os seguintes valores e significados:

1 = mantém todas as repetições;
2 = exclui apenas as repetições tonais;
3 = exclui todas as repetições.

Exemplo:

```
var
  conjunto : NotaLetra;
  indice   : byte;

const
  notas : array [1..7] of texto6 = (‘C#3’,
      ‘Db4’, ‘F#3’, ‘C#3’, ‘G3’, ‘A#3’, ‘Bb3’);

begin
  with conjunto do
    begin
      Iniciar (7, stonal, considerar_registro,
         notas);
      Ordenar (1);
```

```
      for indice := 1 to TotalNotas do
        write (NotaAtual (indice):6);
      writeln;
      Ordenar (2);
      for indice := 1 to TotalNotas do
        write (NotaAtual (indice):6);
      writeln;
      Ordenar (3);
      for indice := 1 to TotalNotas do
        write (NotaAtual (indice):6);
      writeln;
      Finalizar
    end
end.
```

Resultado:

```
   C#3   C#3    F#3    G3    A#3   Bb3   Db4
   C#3   F#3    G3     A#3   Bb3   Db4
   C#3   F#3    G3     A#3   Db4
```

Retrogradar `procedure Retrogradar;`

Substitui o conjunto por outro correspondente ao original retrogradado.

Exemplo:

```
var
  conjunto : NotaLetra;
  indice   : byte;


const
  notas : array [1..5] of texto6 = ('C', 'C#',
      'Ab', 'Bb', 'F#');


begin
  with conjunto do
    begin
      Iniciar (5, stemperado,
         desconsiderar_registro, notas);
      Retrogradar;
      for indice := 1 to 5 do
        write (NotaAtual (indice):4);
      writeln;
      Finalizar
    end
end.
```

Resultado:

```
  F#  A#  G#  C#   C
```

Rotar

```
procedure Rotar (posicao: integer);
```

Substitui o conjunto por outro correspondente ao original deslocado (rotacionado) *posicao* posições. O parâmetro *posicao* pode indicar uma rotação no sentido horário (positivo) ou anti-horário (negativo).

Exemplo:

```
var
  conjunto : OitavaPontoNota;
  indice   : byte;

const
  notas : array [1..6] of texto6 = ('3.00',
      '3.05', '3.10', '2.11', '3.01', '3.06');

begin
  with conjunto do
    begin
      Iniciar (6, stemperado,
         considerar_registro, notas);
      Rotar (-2);
      for indice := 1 to 6 do
        write (NotaAtual (indice):6);
      writeln;
      Finalizar
    end
end.
```

Resultado:

```
  3.01  3.06  3.00  3.05  3.10  2.11
```

Transferir

```
procedure Transferir (var Objeto: OConjunto);
```

Substitui o conjunto definido por *Objeto* pelo original. As características do conjunto original, exceto a codificação, serão mantidas na transferência.

Exemplo:

```
var
  indice    : byte;
  conjunto1 : NotaNome;
  conjunto2 : NotaLetra;
  notas2    : array [1..2] of texto6;
```

```
const
  notas1 : array [1..2] of texto6 = ('Si3',
      'Do4');

begin
  conjunto1.Iniciar (2, stonal,
      considerar_registro, notas1);
  for indice := 1 to 2 do
    write (conjunto1.NotaAtual (indice):4);
  writeln;
  conjunto2.Iniciar (2, stemperado,
      desconsiderar_registro, notas2);
  conjunto1.Transferir (conjunto2);
  for indice := 1 to 2 do
    write (conjunto2.NotaAtual (indice):4);
  writeln;
  conjunto1.Finalizar;
  conjunto2.Finalizar;
end.
```

Resultado:

```
 Si3 Do4
  B3  C4
```

Transpor `procedure Transpor (tintervalo: texto6);`

Substitui o conjunto por outro correspondente ao original transposto ao intervalo musical *tintervalo*. O parâmetro *tintervalo* pode ser codificado em intervalo tonal ou em código numérico representado como *texto6*.

Exemplo:

```
var
  conjunto : OitavaPontoNota;
  interv   : OIntervalo;
  indice   : byte;

const
  notas : array [1..5] of texto6 = ('5.01',
      '5.10', '5.08', '4.09', '4.10');

begin
  interv.Iniciar (isimples, considerar_direcao);
  with conjunto do
    begin
      Iniciar (5, stemperado,
         considerar_registro, notas);
      Transpor ('3m');
```

```
      for indice := 1 to 5 do
        write (NotaAtual (indice):5);
      writeln;
      Finalizar
    end
end.
```

Resultado:

```
5.04 6.01 5.11 5.00 5.01
```

OIntervalo

Campos

tipo `tipo: 0..3;`

O tipo de intervalo, com os significados seguintes:

0 = classe intervalar;
1 = intervalos simples;
2 = intervalos simples e compostos;
3 = notação oitava ponto intervalo.

direcao `direcao: boolean;`

A consideração de direção intervalar, com os seguintes significados:

true = considerar direção;
false = não considerar direção.

Métodos

Iniciar `constructor Iniciar (tipo_de_intervalo: byte; direcao_inicial: boolean);`

Define o tipo de intervalo a ser usado pelos objetos descendentes de *OConjunto*, inicializando *tipo* com *tipo_de_intervalo* e *direcao* com *direcao_inicial*.

Exemplo:

Ver *OConjunto.Intervalo*, *OConjunto.IntervaloTonal*, e *OConjunto.Tranpor*.

OitavaPontoNota

Antecessor `OConjunto`

Métodos

Iniciar

```
constructor Iniciar (ntotal: integer; sistema: byte;
reg: boolean; var opn_inicial);
```

Cria um conjunto de *ntotal* notas musicais na codificação *OitavaPontoNota* contidas em *opn_inicial*, no sistema tonal ou temperado de acordo com *sistema*, e com ou sem consideração de registro de acordo com *reg*. O parâmetro variável sem tipo (**untyped variable parameter**) *opn_inicial* deve ser definido como um vetor (**array**) de *texto6* com *ntotal* elementos, contendo as notas codificadas de acordo com *sistema* e *reg*.

Exemplo:

```
var
  indice   : byte;
  conjunto : OitavaPontoNota;

const
  notas : array [1..4] of texto6 = ('3.04',
      '0.30', '0.24', '0.67');

begin
  with conjunto do
    begin
      Iniciar (4, stemperado,
         considerar_registro, notas);
      for indice := 1 to 4 do
        write (NotaAtual (indice):6);
      writeln;
      Finalizar
    end
end.
```

Resultado:

```
3.04  2.06  2.00  5.07
```

CodigoParaNome

```
function CodigoParaNome (item: integer): texto6;
virtual;
```

Converte o **integer** *item*, correspondente ao código da nota, para uma **string** na codificação *OitavaPontoNota*.

Exemplo:

```
var
  conjunto : OitavaPontoNota;
  notas    : texto6;

begin
  with conjunto do
    begin
      Iniciar (1, stonal, considerar_registro,
         notas);
      writeln (CodigoParaNome (260));
      Finalizar;
    end
end.
```

Resultado:

```
2.68
```

NomeParaCodigo

```
function NomeParaCodigo (item: texto6): integer;
virtual;
```

Converte a **string** *item* na codificação *OitavaPontoNota*, para um **integer**, correspondente ao código da nota.

Exemplo:

```
var
  conjunto : OitavaPontoNota;
  notas      : texto6;
```

```
begin
  with conjunto do
    begin
      Iniciar (1, stemperado,
         considerar_registro, notas);
      writeln (NomeParaCodigo ('3.04'));
      Finalizar;
    end
end.
```

Resultado:

```
40
```

ONota

Descendente `OConjunto`

Campos

oitava `oitava: byte;`
O sistema das notas, isto é, se temperada ou tonal. O valor de *oitava* corresponde ao código da oitava no sistema correspondente.

classe `classe: boolean;`
Se **true**, classe de notas, isto é, desconsiderar registros. Se **false** considerar registro.

Métodos

Iniciar `constructor Iniciar (sistema: byte; reg: boolean);`
Usado pelos objetos descendentes para inicializar *oitava* com *sistema* e *classe* com *reg*.

EXEMPLOS DE UTILIZAÇÃO

Matriz serial

O programa seguinte demonstra o uso do procedimento *Transpor* para construir a matriz dodecafônica. O intervalo de transposição é calculado pelo procedimento *Intervalo* tendo como parâmetros as notas adjacentes da série, produzindo desta forma a inversão requerida em cada coluna da matriz. O procedimento *LerSerie*, aceita qualquer série dodecafônica como dado de entrada e a transfere para a variável *notas*. O procedimento *EscreverLinha* imprime a matriz na tela, extraindo cada nota do objeto *conjunto*. A mudança na definição de *conjunto* para *NotaLetra*, *NotaCódigo*, ou *OitavaPontoNota* produzirá a matriz na codificação correspondente sem necessidade de qualquer outra modificação.

Programa

```
uses objnota;
type
  serie_dodecafonica = array [1..12] of texto6;

var
  conjunto : NotaNome;
  interv   : OIntervalo;
  indice   : byte;
  notas    : serie_dodecafonica;
```

```
procedure LerSerie (var notas: serie_dodecafonica);

  var
    serie : string;
    i     : byte;

  begin
    writeln ('Entre as notas da série separadas
      por espaço:');
    readln (serie);
    serie := serie + ' ';
    for i := 1 to 12 do
      begin
        notas[i] := Copy (serie, 1, Pos (' ',
         serie) - 1);
        Delete (serie, 1, Pos (' ', serie))
      end
  end;

procedure EscreverLinha;
  var
    i    : byte;
    nota : texto6;

  begin
    for i := 1 to 12 do
      begin
        nota := conjunto.NotaAtual (i);
        write (nota, ' ':(5 - length (nota)));
      end;
    writeln;
  end;

begin
  interv.Iniciar (isimples, considerar_direcao);
  LerSerie (notas);
  conjunto.Iniciar (12, stemperado,
      desconsiderar_registro, notas);
  EscreverLinha;
  for indice := 1 to 11 do
    begin
      with conjunto do
        Transpor (Intervalo(interv,
         NotaAtual(indice + 1), NotaAtual
         (indice)));
     EscreverLinha;
    end;
  conjunto.Finalizar
end.
```

Saída

```
Entre as notas da série separadas por espaço:
Do Mib Lab Mi Re Sib Fa# Sol Do# Si Fa La
Do  Re# Sol# Mi  Re  La# Fa# Sol Do# Si  Fa  La
La  Do  Fa  Do# Si  Sol Re# Mi  La# Sol# Re  Fa#
Mi  Sol Do  Sol# Fa# Re  La# Si  Fa  Re# La  Do#
Sol# Si  Mi  Do  La# Fa# Re  Re# La  Sol Do# Fa
La# Do# Fa# Re  Do  Sol# Mi  Fa  Si  La  Re# Sol
Re  Fa  La# Fa# Mi  Do  Sol# La  Re# Do# Sol Si
Fa# La  Re  La# Sol# Mi  Do  Do# Sol Fa  Si  Re#
Fa  Sol# Do# La  Sol Re# Si  Do  Fa# Mi  La# Re
Si  Re  Sol Re# Do# La  Fa  Fa# Do  La# Mi  Sol#
Do# Mi  La  Fa  Re# Si  Sol Sol# Re  Do  Fa# La#
Sol La# Re# Si  La  Fa  Do# Re  Sol# Fa# Do  Mi
Re# Fa# Si  Sol Fa  Do# La  La# Mi  Re  Sol# Do
```

Estudo de Intervalos

"Estudo de Intervalos" é um exemplo simples de Instrução Programada contendo três níveis de dificuldade. O primeiro lida apenas com intervalos diatônicos no tom de Do Maior, o segundo com intervalos diatônicos em qualquer tom e o terceiro com qualquer intervalo conforme indicado na função *Nivel*. Os procedimentos *NotasDiatonicas* e *QuaisquerNotas* escolhem as duas notas e a função *NovoIntervalo* calcula o intervalo tonal entre elas. O teste é efetuado pelo procedimento *Teste* que permite até três respostas erradas antes de fornecer a correta.

Programa

```
uses objnota, crt;
var
  conjunto : NotaNome;
  interv   : OIntervalo;

const
  notas : array [1..7] of texto6 = ('Do', 'Re',
      'Mi', 'Fa', 'Sol', 'La', 'Si');

function Nivel: char;
  begin
   writeln ('Escolha o nível de dificuldade:');
   writeln ('   1: intervalos diatônicos em Do
      Maior');
   writeln ('   2: intervalos diatônicos em
      qualquer tom');
   writeln ('   3: qualquer intervalo');
   repeat until KeyPressed;
   Nivel := ReadKey;
  end;
```

```
procedure NotasDiatonicas (var nota1, nota2:
      texto6);
  begin
    with conjunto do
      begin
        nota1 := NotaAtual (Random (7) + 1);
        nota2 := NotaAtual (Random (7) + 1);
        while nota2 = nota1 do
          nota2 := NotaAtual (Random (7) + 1);
      end
  end;


procedure QuaisquerNotas (var nota1, nota2:
      texto6);
  begin
    with conjunto do
      begin
        nota1 := CodigoParaNome (Random (96));
        nota2 := CodigoParaNome (Random (96));
        while nota2 = nota1 do
          nota2 := CodigoParaNome (Random (96));
      end
  end;


function NovoIntervalo (var nota1, nota2:
      texto6; var direcao: string): texto6;
  var
    temp : texto6;


  begin
    if (Random (2) + 1) = 1
      then
        begin
          direcao := 'ascendente';
          nota1[length (nota1)] := '0';
          nota2[length (nota2)] := '1'
        end
      else
        begin
          direcao := 'descendente';
          nota1[length (nota1)] := '1';
          nota2[length (nota2)] := '0';
        end;
    with conjunto do
      temp := IntervaloTonal (interv, nota1,
      nota2);
    if temp[1] = '-'
      then Delete (temp, 1, 1);
    Delete (nota1, length (nota1), 1);
    Delete (nota2, length (nota2), 1);
    NovoIntervalo := temp;
  end;
```

```
procedure Teste (intv, nota1, nota2: texto6;
      direcao: string);
  var
    tentativas  : byte;
    resposta   : texto6;

  begin
    writeln ('Qual o intervalo ', direcao, '
      entre as notas ', nota1, ' e ', nota2,
      '?');
    tentativas := 1;
    repeat
      begin
        readln (resposta);
        if resposta = intv
          then
            begin
              writeln ('Correto!');
              tentativas := 4
            end
          else
            begin
              if tentativas < 3
                then writeln ('Tente outra vez.')
                else writeln ('O intervalo é ',
                  intv);
              inc (tentativas)
            end
      end
    until tentativas = 4;
  end;

procedure ExecuteTreino (nota1, nota2: texto6);
  var
    direcao : string;
    intv       : texto6;

  begin
    intv := NovoIntervalo (nota1, nota2, direcao);
    Teste (intv, nota1, nota2, direcao);
  end;

procedure Nivel1;
  var
    testes : byte;
    nota1,
    nota2  : texto6;

  begin
    for testes := 1 to 5 do
      begin
        NotasDiatonicas (nota1, nota2);
        ExecuteTreino (nota1, nota2);
      end
  end;
```

```
procedure Nivel2;
  var
    testes : byte;
    nota1,
    nota2  : texto6;

  begin
    for testes := 1 to 5 do
      begin
        with conjunto do
          begin
            nota2 := NotaAtual (Random (7) + 1);
            Transpor (Intervalo (interv, 'i1',
               nota2));
            if nota2[length (nota2)] = '1'
              then Transpor ('-8J');
            if nota2[length (nota2) - 2] = 'd'
      then
              if nota2[length (nota2) - 1] = 'b'
                then Transpor ('1A')
                else Transpor ('-1A');
         end;
        NotasDiatonicas (nota1, nota2);
        ExecuteTreino (nota1, nota2)
     end
  end;

procedure Nivel3;
  var
    testes : byte;
    nota1,
    nota2  : texto6;

  begin
    for testes := 1 to 5 do
      begin
        QuaisquerNotas (nota1, nota2);
        ExecuteTreino (nota1, nota2)
      end
  end;

begin {ExecuteTreino}
  Randomize;
  interv.Iniciar (isimples, considerar_direcao);
  conjunto.Iniciar (7, stonal,
      considerar_registro, notas);
  case Nivel of
    '1' :Nivel1;
    '2' :Nivel2;
    '3' :Nivel3;
  end {case}
end.
```

Saída

```
Escolha o nível de dificuldade:
   1: intervalos diatônicos em Do Maior
   2: intervalos diatônicos em qualquer tom (!)
   3: qualquer intervalo
Qual o intervalo ascendente entre as notas Re e Fa#?
3M
Correto!
Qual o intervalo descendente entre as notas Fa# e Re#?
6M
Tente outra vez.
3M
Tente outra vez.
4J
O intervalo é 3m
Qual o intervalo descendente entre as notas Re# e Do#?
2M
Correto!
Qual o intervalo descendente entre as notas Sol# e Fa#?
2M
Correto!
Qual o intervalo ascendente entre as notas Fa# e Do#?
5J
Correto!
```

Sub-conjuntos

Sub-conjuntos é um programa mais complexo que os anteriores e imprime todos os subconjunto de *n* notas de um dado conjunto. As rotinas utilizadas aqui, com algumas modificações, são parte de um programa completo para operações com conjunto de notas denominado PCN (Processador de Classes de Notas). Sub-conjuntos exemplifica várias rotinas de OBJNOTA, tais como *Finalizar*, *FormaNormal*, *FormaPrima*, *Iniciar*, *NotaAtual*, e *TotalNotas*.

Programa

```
uses objnota, crt;
const
 maxnotas = 127;
 maxlinha = 78;

type
 vettexto  = array [1..maxnotas] of texto6;
 tprima    = string[30];
 texto2    = string[2];
 pont      = ^pconjunto;
```

```
 pconjunto = record
              prima  : tprima;
              normal : string;
              menor  : pont;
              maior  : pont
             end;
 ponteiros = (inicial, anterior, atual);
 vconjunto = array [ponteiros] of pont;

var
 nnotas,
 tnotas,
 subinicial,
 subfinal   : byte;
 i          : integer;
 entrada    : string;
 tsub       : texto2;
 nota1,
 nota2      : vettexto;
 notas      : NotaCodigo;
 conjunto   : vconjunto;

procedure LerEntrada (var texto: string; var
      total: texto2);
 begin
  write ('Entre as classes de notas separadas
      por espaço: ');
  readln (texto);
  write ('Entre o número de notas do subconjunto
      (0 = todos): ');
  readln (total);
 end;

procedure VetorNotas (texto: string; var notas:
      NotaCodigo; var nnotas: byte);
 var
  espaco :   word;
  n1     :   vettexto;
  nova   :   string[2];


 begin
  nnotas := 0;
  while texto[1] = ' ' do
   Delete (texto, 1, 1);
  if texto[length (texto)] <> ' '
   then texto := texto + ' ';
  espaco := Pos (' ', texto);
  while espaco <> 0 do
   begin
    nova := Copy (texto, 1, espaco - 1);
    if (nnotas = 0) or (n1[nnotas] <> nova) then
     begin
      inc (nnotas);
      n1[nnotas] := nova
     end;
```

```
    Delete (texto, 1, espaco);
    espaco := Pos (' ', texto)
   end;
  notas.Iniciar (nnotas, stemperado,
      desconsiderar_registro, n1)
 end;

function TextoNumero (texto: string): integer;
 var
  code,
  numero : integer;

 begin
  Val (texto, numero, code);
  TextoNumero := numero
 end;

procedure ImprimirConjunto (nota: vettexto; var
      conjunto: vconjunto; n1: integer);
 var
  n2,
  i     : integer;
  fim   : boolean;
  linha : string;

 procedure Escrever (texto: string; identa: byte);
  var
   i : byte;

  begin
   while length (texto) > (maxlinha - identa) do
    begin
     i := (maxlinha - identa);
     while texto[i] <> ' ' do
      dec (i);
     writeln (' ': identa, Copy (texto, 1, i -
               1));
     Delete (texto, 1, i - 1)
    end;
   if (texto <> ' ')
    then writeln (' ':identa, texto);
  end;

 procedure IniciarLinha;
  begin
   while length (linha) < n2 do
    linha := linha + ' '
  end;

 procedure NovaLinha (texto: string);
  begin
   Escrever (linha, 0);
   linha := ' ';
   IniciarLinha;
   linha := linha + texto
  end;
```

```
begin {ImprimirConjunto}
 if conjunto[inicial] = nil
  then exit;
 fim := false;
 repeat
  conjunto[atual] := conjunto[inicial];
  while conjunto[atual]^.menor <> nil do
   begin
    conjunto[anterior] := conjunto[atual];
    conjunto[atual] := conjunto[atual]^.menor
   end;
  with conjunto[atual]^ do
   begin
    if prima[length (prima)] = ')'
     then
      begin
       i := Pos ('(', prima);
       linha := '  [' + Copy (prima, 1, i - 1)
           + '] ' + Copy (prima, i, length
           (prima) - i + 1) + ': ';
       n2 := n1 + 14
      end
     else
      begin
       linha := '  [' + prima + ']: ';
       n2 := n1 + 6
      end;
    IniciarLinha;
    while Pos ('|', normal) <> 0 do
     begin
      i := Pos ('|', normal);
      if (length (linha) + n1) < maxlinha
       then linha := linha + '[' + Copy
           (normal, 1, i - 1) + '] '
       else NovaLinha ('[' + Copy (normal, 1, i
           - 1) + '] ');
      Delete (normal, 1, i)
     end;
    if (length (linha)  + n1) < maxlinha
     then
      begin
       linha := linha + '[' + normal + ']';
       Escrever (linha, 0)
      end
     else
      begin
       NovaLinha ('[' + normal + ']');
       Escrever (linha, 0)
      end
   end;
```

```
  if conjunto[atual] <> conjunto[inicial]
   then conjunto[anterior]^.menor :=
      conjunto[atual]^.maior
   else if conjunto[inicial]^.maior <> nil
    then conjunto[inicial] :=
           conjunto[atual]^.maior
    else fim := true;
  Dispose (conjunto[atual]);
  conjunto[atual] := nil
 until fim
 end;

procedure SubConjunto (var conjunto: vconjunto;
      nota1: vettexto; var nota2: vettexto;
      indice, final, nivel, s: integer);
 var
  i, j     : integer;
  temp     : string;
  snormal,
  sprima   : string;

 function TextoNotas: string;
  var
   i    : byte;
   temp : string;

  begin
   temp := '';
   for i := 1 to notas.TotalNotas do
    temp := temp + ' ' + notas.NotaAtual (i);
   TextoNotas := temp;
   notas.Finalizar
  end;


 function FNormal (entrada: string): string;
  begin
   VetorNotas (entrada, notas, nnotas);
   notas.FormaNormal;
   FNormal := TextoNotas
  end;

 function FPrima(entrada: string): string;
  var
   temp  : string;
   total : byte;

  begin
   VetorNotas (entrada, notas, nnotas);
   notas.FormaPrima;
   temp := TextoNotas;
   total := notas.TotalNotas;
   FPrima := temp
  end;
```

```
procedure TransferirConjunto (var conjunto:
     vconjunto; sprima, snormal: tprima);

 procedure NovoConjunto (var prox: pont);
  var
   novo   : pont;

  begin
   new (novo);
   novo^.prima := sprima;
   novo^.normal := snormal;
   if conjunto[inicial] = nil
    then conjunto[inicial] := novo
    else prox := novo;
   conjunto[atual] := novo;
   conjunto[atual]^.menor := nil;
   conjunto[atual]^.maior := nil
  end;

 begin {TransferirConjunto}
  conjunto[atual] := conjunto[inicial];
  if conjunto[atual] = nil
   then NovoConjunto (conjunto[atual]^.menor)
   else
    begin
     while conjunto[atual] <> nil do
      if sprima = conjunto[atual]^.prima
       then
        begin
       if Pos(snormal,conjunto[atual]^.normal) = 0
          then conjunto[atual]^.normal := conjunto
              [atual]^.normal + '|' + snormal;
         conjunto[atual] := nil
        end
       else if sprima < conjunto[atual]^.prima
        then
         begin
          if conjunto[atual]^.menor <> nil
         then conjunto[atual]:=conjunto [atual]^.menor
           else
            begin
           NovoConjunto (conjunto[atual]^ .menor);
           conjunto[atual] := conjunto [atual]^.maior
            end
         end
        else
         begin
          if conjunto[atual]^.maior <> nil
           then conjunto[atual] := conjunto
              [atual]^.maior
           else
            begin
           NovoConjunto (conjunto[atual]^ .maior);
           conjunto[atual] := conjunto [atual]^.maior
            end
         end
    end
 end;
```

```
function Compactar (texto: string): tprima;
 type
  texto2 =   string[2];

 procedure Dezena (t1: texto2);
  var
   i1, i2 : byte;

  begin
   i1 := Pos (t1, texto);
   i2 := Pos ('(', texto);
   while (i1 <> 0) and ((i1 < i2) or (i2 = 0))
     do
    begin
     Delete (texto, i1, 2);
     if t1 = '10'
      then Insert ('A', texto, i1)
      else Insert ('B', texto, i1);
     i1 := Pos (t1, texto);
     i2 := Pos ('(', texto)
    end
  end;

 begin {Compactar}
  if Pos ('10', texto) <> 0
   then Dezena ('10');
  if Pos ('11', texto) <> 0
   then Dezena ('11');
  while Pos (' ', texto) <> 0 do
   Delete (texto, Pos (' ', texto), 1);
  Compactar := texto
 end;

begin {SubConjunto}
 for i := indice to final do
  begin
   nota2[nivel] := nota1[i];
   if (nivel = s) or (i = final)
    then
     begin
      temp := '';
      for j := 1 to s do
       temp := temp + ' '+ nota2[j];
      snormal := Fnormal (temp);
      snormal := Compactar (snormal);
      sprima := FPrima (temp);
      sprima := Compactar (sprima);
     TransferirConjunto (conjunto, sprima, snormal)
     end
    else SubConjunto (conjunto, nota1, nota2, i
        + 1, final + 1, nivel + 1, s)
  end
end;
```

```
begin {SubN}
 LerEntrada (entrada, tsub);
 VetorNotas (entrada, notas, tnotas);
 notas.ExcluirRepeticao (2);
 tnotas := notas.TotalNotas;
 for i := 1 to tnotas do
  nota1[i] := notas.NotaAtual (i);
 i := TextoNumero (tsub);
 if (tsub = '0') or (i >= tnotas) or (i = 0)
  then
   begin
    subinicial := 2;
    subfinal   := tnotas - 1
   end
  else
   begin
    subinicial := i;
    subfinal := subinicial
   end;
 for i := subinicial to subfinal do
  begin
   conjunto[inicial] := nil;
   SubConjunto(conjunto, nota1, nota2, 1,
      (tnotas + 1) - i, 1, i);
   ImprimirConjunto (nota1, conjunto, i + 2);
   if KeyPressed
    then if ReadKey = #27 then
     begin
      notas.Finalizar;
      Exit
     end
  end;
 notas.Finalizar
end.
```

Saída

```
Entre as classes de notas separadas por espaço: 8
10 4 7 3
Entre o número de notas do subconjunto (0 =
      todos): 3
   [012]: [78A]
   [014]: [478] [347]
   [015]: [348] [378]
   [016]: [A34]
   [026]: [48A]
   [027]: [8A3]
   [036]: [47A]
   [037]: [37A]
```

DOCUMENTAÇÃO DAS ROTINAS

Constantes globais

nomes `( = ‘Do Re Mi Fa SolLa Si ‘)`
Nomes das notas musicais usados na codificação *NotaNome*.

letras `( = ‘C D E F G A B ‘)`
Nomes das notas musicais usados na codificação *NotaLetra*.

acidentes `( = ‘hbsbpbqbtbdbb # d#t#q#p#s#h#’)`
Símbolos para as alterações musicais.

intervalos `(string [47] = ‘sd,pd,qd,td,dd,d ,m ,J ,M ,A ,dA,tA,qA,pA,sA,hA’)`
Símbolos para as qualidades intervalares usados por *IntervaloTonal.*

Rotinas gerais

Reajuste Tonal — função

Descrição Efetua o reajuste necessário em operações nas quais o significado tonal das notas não é levado em consideração. Usado por *Multiplicar*; *FormaPrima*; *Complemento*; *EixoSimetria* e *CodigoParaIntervalo*.

Parâmetros **Valores**

`cod (numérico - integer)`: código numérico correspondente a uma nota musical na codificação *stemperado*.

Declaração **Cabeçalho**

```
function ReajusteTonal (cod: integer): integer;
```

Rotina

```
begin
  if (cod > 4) and Odd (cod)
    then ReajusteTonal := cod + 12 + (12 * (cod
      div 2))
    else ReajusteTonal := cod + (12 * (cod div
      2));
end;
```

Operação O parâmetro *cod* em codificação *stemperado* é convertido para a codificação *stonal* com o mesmo significado. Por exemplo a nota Re (D) que na codificação *stemperado* tem como código o valor 2 adquire o valor 14 correspondente à mesma nota na codificação *stonal*.

Mdc — função

Descrição Retorna o Máximo Divisor Comum entre dois valores. Usado por *Rotar*.

Parâmetros **Valores**

`x (numérico - word)`: primeiro valor numérico.

`y (numérico - word)`: segundo valor numérico.

Declarações **Cabeçalho**

```
function Mdc (x, y: word): word;
```

Variáveis

`temp (numérica - word)`: variável temporária para armazenar o resto da divisão entre os dois valores numéricos.

Rotina

```
begin
  repeat
    temp := x mod y;
    x := y;
    y := temp
  until temp = 0;
  Mdc := x
end;
```

Operação O resto da divisão entre os parâmetros *x* e *y*, *temp*, substitui o divisor enquanto *y*, o divisor anterior, substitui o dividendo em uma divisão sucessiva até que o resto seja igual a 0. Ao final da divisão sucessiva *x*, o dividendo, contém o Máximo Divisor Comum.

TpI — função

Descrição Converte um valor literal correspondente a um número inteiro em sua representação numérica. Usado por *Npc*, *IntervaloTonal*, *IntervaloParaCodigo*, *IpC*, *NotaIndice*, e *NomeParaCodigo* do objeto *NotaCodigo*.

Parâmetros **Valores**

`texto (literal - texto6)`: valor literal correspondente a um número inteiro.

Declarações **Cabeçalho**

```
function TpI (texto: texto6): integer;
```

Variáveis

`temp (numérica - integer)`: variável temporária para

armazenar o valor numérico a ser repassado pela função.

`code` (numérica - `integer`): usada por ***Val*** para indicar erro.

Rotina

```
begin
  Val (texto, temp, code);
  TpI := temp;
end;
```

Operação *TpI* apenas transforma o procedimento (procedure) **Val** em uma função que retorna um valor numérico inteiro (**integer**)..

TpR — função

Descrição Converte um valor literal correspondente a um número real em sua representação numérica. Usada por *IntervaloParaCodigo* e por *NomeParaCodigo* do objeto *OitavaPontoNota*.

Parâmetros

Valores

`texto` (`literal` - `texto6`): valor literal correspondente a um número real.

Declarações

Cabeçalho

```
function TpR (texto: texto6): real;
```

Variáveis

`temp` (numérica - `real`): variável temporária para armazenar o valor numérico a ser repassado pela função.

`code` (numérica - `integer`): usada por ***Val*** para indicar erro.

Rotina

```
begin
  Val (texto, temp, code);
  TpR := temp;
end;
```

Operação *TpR* apenas transforma o procedimento (procedure) **Val** em uma função que retorna um valor numérico real (**real**).

Campos de ONota

oitava `numérico (byte)`
Sistema (tonal ou temperado) utilizado pelo conjunto.

classe `lógico (boolean)`
Consideração de registro.

Métodos de ONota

Iniciar **construtor**

Descrição Inicializa o objeto *ONota*.

Parâmetros **Valores**

`sistema (numérico - byte)`: indica qual o sistema a ser utilizado (tonal ou temperado)

`reg (lógico - boolean)`: indica a consideração ou não do registro.

Declaração **Cabeçalho**

```
constructor ONota.Iniciar (sistema: byte; reg:
      boolean);
```

Rotina

```
begin
  oitava := sistema;
  classe := not reg;
end;
```

Operação *Oitava* adquire o valor de *sistema* e *classe* o de negação de *reg*. *Sistema* é normalmente inicializado com as constantes *stemperado* e *stonal* enquanto

reg é normalmente inicializado com as constantes *considerar_registro* e *desconsiderar_registro*.

Modulo **função**

Descrição Retorna o módulo positivo entre o código de uma nota e o código da oitava em operações com classes de notas. Usada por *NovaNota*; *NovoConjunto*; *Enarmonica*; *Transpor*; *Inverter*; e *Iniciar* dos objetos *NotaCodigo*, *OitavaPontoNota*, *NotaNome*, *NotaLetra*.

Parâmetros **Valores**

`elemento (numérico - integer)`: código correspondente a uma nota musical.

Declaração **Cabeçalho**

```
function ONota.Modulo (elemento: integer):
      integer;
```

Rotina

```
begin
  if classe then
    begin
      elemento := elemento mod oitava;
      if elemento < 0
      then inc (elemento, oitava);
    end;
    Modulo := elemento;
end;
```

Operação Se o registro for desconsiderado obtém-se o módulo entre *elemento* e *oitava*. Caso este módulo seja negativo encontra-se o código correspondente positivo somando-se o resultado ao código da oitava. Se o registro for considerado a função retorna o valor original sem alteração.

NpC **função**

Descrição Converte o nome ou a letra correspondente a uma nota musical para o código numérico equivalente.

Usada por *NomeParaCodigo* dos objetos *NotaNome* e *NotaLetra*.

Parâmetros **Valores**

`item (literal - texto6)`: o nome ou a letra correspondente a uma nota musical.

`notas (literal - string)`: literal contendo os nomes ou as letras correspondentes às notas musicais. Normalmente inicializada com as constantes *nomes* e *letras*.

Declarações **Cabeçalho**

```
function ONota.NpC (item: texto6; notas:
      string): integer;
```

Variáveis

`registro (numérica - byte)`: registro (oitava) da nota.

`tlinha (numérica - byte)`: posição da nota em *notas*, correspondente à linha em que o código da nota é encontrado na tabela de códigos.

`tcoluna (numérica - byte)`: posição da alteração na constante *acidentes*, correspondente à coluna em que o código da nota é encontrado na tabela de códigos.

Constantes

`clinha (vetor numérico - array [1..7] of integer = (89, 7, 21, 34, 48, 62, 76))`: constantes indicando o valor de referência das notas diatônicas na tabela de códigos das notas.

Rotina

```
begin
  registro := 0;
  if item[length (item)] in ['0'..'9'] then
    begin
      registro := TpI (item[length (item)]);
      Delete (item, length (item), 1)
    end;
  if (length (item) > 1) and not (item[2] in
      ['a', 'e', "i", 'o'])
    then Insert (' ', item, 2);
```

```
  if item[3] <> 'l'
    then Insert (' ', item, 3);
  while length (item) < 5 do
    item := item + ' ';
  tlinha := (Pos (Copy (item, 1, 3), notas) - 1)
      div 3 + 1;
  if item[4] = ' '
    then tcoluna := 7
    else tcoluna := (Pos (Copy (item, 4, 2),
      acidentes) - 1) div 2;
  NpC := ((clinha [tlinha] + tcoluna) mod
      oitava) + (oitava * registro);
end;
```

Operação

linha 1: *registro* é inicializado em 0;

linhas 2-6: o último caráter de *item* é testado para verificar a indicação de oitava. Em caso positivo a indicação de oitava é transferida para *registro* e o último caráter é excluído de *item*;

linhas 7-12: *item* é ajustado de modo que contenha 5 caracteres, os três primeiros com o nome ou a letra correspondente à nota e os dois últimos com o símbolo correspondentes à alteração;

linha 13: a posição do nome ou da letra correspondente à nota é encontrada e transferida para *tlinha*;

linhas 14-16: se a nota for natural *tcoluna* assume o valor de 7. Caso contrário *tcoluna* assume o valor indicado pela posição da alteração na constante *acidentes*;

linhas 17-18: o código da nota é calculado e transferido.

Linha **função**

Descrição Retorna a linha da tabela de códigos de notas na qual encontra-se o código da nota. Usada por *CpN* e *Frequencia.*

Parâmetros **Valores**

`item (numérico - integer)`: código correspondente a uma nota musical.

Declarações **Cabeçalho**

```
function Onota.Linha (item: integer): byte;
```

Variáveis

`temp (numérica - byte)`: armazenamento temporário para a função.

Rotina

```
begin
  temp := item mod oitava;
  case oitava of
    12 : begin
           if temp > 4
             then inc (temp);
           Linha := (temp div 2) + 1
         end;
    96 : begin
           if temp > 61
             then inc (temp);
           temp := ((temp + 7) div 14) + 1;
           if temp = 8
             then Linha := 1
             else Linha := temp;
         end;
  end; {case}
end;
```

Operação linha 1: desconsidera o registro;
linha 2: verifica o sistema (temperado ou tonal) usado;
linhas 3-7: sistema temperado - reajusta para as notas acima do Mi (E) e calcula o resultado;
linhas 8-15: sistema tonal - reajusta para as notas acima do Sols# e calcula o resultado. Em seguida, se necessário iguala a oitava linha à primeira.

Coluna — função

Descrição Retorna a coluna da tabela de códigos de notas na qual encontra-se o código da nota. Usada por *CpN* e *Frequencia*.

Parâmetros **Valores**

`item (numérico - integer)`: código correspondente a uma nota musical.

Declarações **Cabeçalho**

```
function ONota.Coluna (item: integer): byte;
```

Variáveis

`temp (numérica - byte)`: armazenamento temporário para a função.

`cod (numérica - byte)`: código da nota na oitava básica.

Rotina

```
begin
  cod := item mod oitava;
  case oitava of
  12 : if ((cod > 4) and Odd(cod)) or ((cod < 5)
      and not Odd(cod))
         then Coluna := 8
         else Coluna := 9;
  96 : begin
         if cod = 48 then
           begin
             Coluna := 15;
             Exit
           end;
         if cod > 34 then
           begin
             inc (cod);
             if cod > 90
               then inc (cod);
           end;
         temp := (cod mod 7) + 1;
         if not Odd (cod div 7)
           then inc (temp, 7);
         Coluna := temp;
       end;
  end; {case}
end;
```

Operação

linha 1: desconsidera o registro;

linha 2: verifica o sistema (temperado ou tonal) usado;

linhas 3-5: sistema temperado - verifica em qual coluna encontra-se o código e transfere para a função;

linhas 6 22: sistema tonal - realiza os reajustamentos necessários, calcula o resultado e transfere para a função.

CpN **função**

Descrição Converte o código numérico correspondente a uma nota musical para o nome ou a letra equivalente. Usada por *CodigoParaNome* dos objetos *NotaNome* e *NotaLetra*.

Parâmetros **Valores**

`item (numérico - integer)`: o código numérico correspondente a uma nota musical.

`notas (literal - string)`: literal contendo os nomes ou as letras correspondentes às notas musicais. Normalmente inicializada com as constantes *nomes* e *letras*.

Declarações **Cabeçalho**

```
function ONota.CpN (item: integer;
      notas:string): texto6;
```

Variáveis

`temp (literal - texto6)`: armazenamento temporário para a função.

`registro (literal - string[1])`: registro (oitava) da nota.

Rotina

```
begin
  Str ((item div oitava), registro);
  temp := Copy (notas, (Linha (item) * 3) - 2,
      3) + Copy (acidentes, (Coluna (item) * 2)
      - 1, 2);
  while Pos (' ', temp) <> 0 do
    Delete (temp, Pos (' ', temp), 1);
    if classe
      then CpN := temp
      else CpN := temp + registro;
end;
```

Operação linha 1: transfere o registro (oitava) da nota para registro;

linhas 2-3: encontra o nome ou a letra correspondente ao código da nota musical por sua posição na tabela de códigos de notas;

linhas 4-5: exclui os espaços extras;
linhas 6-8: verifica se o registro deve ser considerado. Caso positivo adiciona-o ao final do nome ou da letra correspondente à nota.

Campos de OIntervalo

tipo `numérico (0..3)`
Tipo de intervalo utilizado (classe, simples, composto, oitava ponto intervalo).

direcao `lógico` **`(boolean)`**
Consideração de direção.

Métodos de OIntervalo

Iniciar — **construtor**

Descrição Inicializa o objeto *OIntervalo*.

Parâmetros **Valores**

`tipo_de_intervalo (numérico -` **`byte`**`)`: indica qual o tipo de intervalo utilizado (classe, simples, composto, oitava ponto intervalo).

`direcao_inicial (lógico -` **`boolean`**`)`: indica a consideração ou não de direção.

Declaração **Cabeçalho**

```
constructor OIntervalo.Iniciar
      (tipo_de_intervalo: byte; direcao_inicial:
      boolean);
```

Rotina

```
begin
  tipo := tipo_de_intervalo;
  direcao := direcao_inicial;
end;
```

Operação *Tipo* adquire o valor de *tipo_de_intervalo* e *direcao* o de *direcao_inicial*. *Tipo_de_intervalo* é normalmente inicializado com as constantes *iclasse*,

isimples, *icomposto* e *ioitavaponto* enquanto *direcao_inicial* é normalmente inicializado com as constantes *considerar_direcao* e *desconsiderar_direcao.*

Campos de OConjunto

total_notas `numérico` **`(word)`**
Total de notas do conjunto.

codigo `ponteiro (pvetorinteiro)`
Aponta para um local da memória onde estão armazenados os códigos das notas.

Métodos de OConjunto

Iniciar — construtor

Descrição Inicializa o objeto *OConjunto*.

Parâmetros **Valores**

`ntotal (numérico - integer)`: total de notas do conjunto.

`sistema (numérico - byte)`: indica qual o sistema a ser utilizado (tonal ou temperado)

`reg (lógico - boolean)`: indica a consideração ou não do registro.

Declaração **Cabeçalho**

```
constructor OConjunto.Iniciar (ntotal: integer;
      sistema: byte; reg: boolean);
```

Rotina

```
begin
  ONota.Iniciar (sistema, reg);
  total_notas := ntotal;
  GetMem (codigo, SizeOf (vetorinteiro) *
      total_notas);
  if codigo = nil then
    begin
      OConjunto.Finalizar;
      Fail;
    end;
end;
```

Operação Inicializa o objeto *ONota*, inicializa *total_notas* com o total de notas do conjunto e reserva memória suficiente para armazenar todas as notas. Em seguida testa para verificar se a reserva de memória teve sucesso e em caso contrário finaliza o objeto *OConjunto* e aborta.

Finalizar — destruidor

Descrição Finaliza o objeto *OConjunto*.

Declaração **Cabeçalho**

```
destructor OConjunto.Finalizar; virtual;
```

Rotina

```
begin
  if codigo <> nil then
    begin
      FreeMem (codigo, SizeOf (vetorinteiro) *
      total_notas);
      codigo := nil;
    end;
end;
```

Operação Caso o conjunto exista, libera a memória reservada e anula o ponteiro.

NovaNota — procedimento

Descrição Substitui a nota musical indicada por *indice*.

Parâmetros **Valores**

`indice (numérico - integer)`: aponta para a nota do conjunto a ser substituída.

`nova_nota (literal - texto6)`: nota subsitituta.

Declaração **Cabeçalho**

```
procedure OConjunto.NovaNota (indice: integer;
      nova_nota: texto6);
```

Rotina

```
begin
  codigo^[indice] := Modulo (NomeParaCodigo
      (nova_nota));
end;
```

Operação Converte o literal correspondente à nota musical para o código respectivo e substitui no conjunto na localização indicada por *indice*..

NotaAtual — função

Descrição Retorna a nota indicada por *indice*.

Parâmetros **Valores**

`indice (numérico - integer)`: aponta para a nota do conjunto a ser lida.

Declaração **Cabeçalho**

```
function OConjunto.NotaAtual (indice: integer):
      texto6;
```

Rotina

```
begin
  NotaAtual := CodigoParaNome (codigo^[indice]);
end;
```

Operação Converte o código correspondente à nota musical indicada por *indice* para o literal respectivo e transfere para a função.

NovoConjunto — procedimento

Descrição Substitui um conjunto de notas por outro.

Parâmetros **Variáveis**

`novos_codigos (sem tipo definido)`: códigos das notas que devem substituir as contidas no conjunto.

Declarações **Cabeçalho**

```
procedure OConjunto.NovoConjunto (var
      novos_codigos);
```

Variáveis

`i (numérica - word)`: contador para laço e índice para o conjunto de notas.

`s (numérica - word)`: tamanho do vetor que contém os códigos das notas.

`temp (ponteiro - pvetortexto)`: vetor temporário para os códigos das notas.

Rotina

```
begin
  s := SizeOf (vetortexto) * total_notas;
  GetMem (temp, s);
  if temp = nil
    then Exit;
  Move (novos_codigos, temp^, s);
  for i := 1 to total_notas do
    codigo^[i] := Modulo ( NomeParaCodigo
      (temp^[i]));
  FreeMem (temp, s);
  temp := nil;
end;
```

Operação linha 1: calcula o tamanho do vetor contendo os códigos das notas;
linhas 2-4: reserva memória para os códigos das notas e testa se a reserva teve sucesso. Em caso negativo aborta;
linha 5: transfere *novos_codigos* para *temp*;
linhas 6-7: converte e transfere os novos códigos para a localização do conjunto original;
linhas 8-9: libera a memória reservada e anula o ponteiro *temp*.

ConjuntoAtual — procedimento

Descrição Lê todo o conjunto atual transferindo-o para *codigos_atuais*.

Parâmetros **Variáveis**

`codigos_atuais (sem tipo definido)`: códigos das notas do conjunto.

Declarações **Cabeçalho**

```
procedure OConjunto.ConjuntoAtual (var
      codigos_atuais);
```

Variáveis

`i (numérica - word)`: contador para laço e índice para o conjunto de notas.

`s (numérica - word)`: tamanho do vetor que contém os códigos das notas.

`temp (ponteiro - pvetortexto)`: vetor temporário para os códigos das notas.

Rotina

```
begin
  s := SizeOf (vetortexto) * total_notas;
  GetMem (temp, s);
  if temp = nil
    then Exit;
  for i := 1 to total_notas do
    temp^[i] := CodigoParaNome (codigo^[i]);
  Move (temp^, codigos_atuais, s);
  FreeMem (temp, s);
  temp := nil;
end;
```

Operação

linha 1: calcula o tamanho do vetor contendo os códigos das notas;

linhas 2-4: reserva memória para os códigos das notas e testa se a reserva teve sucesso. Em caso negativo aborta;

linhas 5-6: converte e transfere os códigos originais para *temp*;

linha 7: transfere os códigos de *temp* para *codigos_atuais*;

linhas 8-9: libera a memória reservada e anula o ponteiro *temp*.

RegistroAtual — função

Descrição Retorna o registro (oitava) da nota indicada por *indice*.

Parâmetros **Valores**

`indice (numérico - integer)`: aponta para a nota da qual se deseja o registro.

Declaração **Cabeçalho**

```
function OConjunto.RegistroAtual (indice:
      integer): byte;
```

Rotina
```
begin
  RegistroAtual := codigo^[indice] div oitava
end;
```

Operação Calcula o registro e o transfere para a função.

TotalNotas — função

Descrição Retorna o total de notas do conjunto.

Declaração **Cabeçalho**
```
function OConjunto.TotalNotas: word;
```

Rotina
```
begin
  TotalNotas := total_notas;
end;
```

Operação Transfere *total_notas* para a função.

NovoTotal — procedimento

Descrição Modifica o total de notas do conjunto.

Parâmetros **Valores**

`novo_total (numérico - word)`: total de notas desejado.

Declaração **Cabeçalho**
```
procedure OConjunto.NovoTotal (novo_total:
      word);
```

Rotina
```
begin
  total_notas := novo_total;
end;
```

Operação Transfere o total de notas desejado para *total_notas*.

Transferir — procedimento

Descrição Transfere as notas do conjunto original para o conjunto definido por *Objeto*.

Parâmetros **Variáveis**

`Objeto (objeto - OConjunto)`: objeto com o conjunto a ser substituído.

Declarações **Cabeçalho**

```
procedure OConjunto.Transferir (var Objeto:
     OConjunto);
```

Variáveis

`i (numérica - word)`: contador para laço e índice para o conjunto de notas.

Rotina

```
1  begin
2    FreeMem (Objeto.codigo, SizeOf (vetorinteiro)
       * Objeto.total_notas);
3    GetMem (Objeto.codigo, SizeOf (vetorinteiro) *
       total_notas);
4    if Objeto.codigo = nil
       then Exit;
5    Objeto.oitava := oitava;
6    Objeto.classe := classe;
7    Objeto.total_notas := total_notas;
8    for i := 1 to total_notas do
9      Objeto.codigo^[i] := codigo^[i];
   end;
```

Operação

linha 1: libera a memória reservada para o conjunto a ser substituído;

linha 2: reserva memória para o conjunto a ser substituído considerando que o total de notas do novo conjunto pode ser diferente do anterior;

linhas 3-4: testa para verificar se a reserva de memória teve sucesso. Em caso negativo aborta;

linhas 5-7: transfere os campos do conjunto substituto para os campos de *Objeto*;

linhas 8-9: transfere os códigos do conjunto substituto para *Objeto*.

Enarmonica — função

Descrição Retorna uma nota enarmonicamente equivalente a *enota*.

Parâmetros **Valores**

`enota (literal - texto6)`: nota original ou índice para uma nota do conjunto (quando antecedido pelo caráter “i”).

`direcao (numérico - shortint)`: direção da enarmonização.

Declaração **Cabeçalho**

```
function OConjunto.Enarmonica (enota: texto6;
      direcao: shortint): texto6;
```

Rotina

```
begin
  Enarmonica := CodigoParaNome (Modulo
      (NotaIndice (enota) + (direcao * 12)));
end;
```

Operação Calcula o código da nota enarmonizada na direção indicada, converte-o em uma literal e a transfere para a função.

Intervalo — função

Descrição Retorna o código correspondente ao intervalo entre duas notas.

Parâmetros **Valores**

`Objeto (objeto - OIntervalo)`: objeto com a definição do tipo de intervalo desejado.

`nota1 (literal - texto6)`: primeira nota ou índice para uma nota do conjunto (quando antecedido pelo caráter "i").

`nota2 (literal - texto6)`: segunda nota ou índice para uma nota do conjunto (quando antecedido pelo caráter "i").

Declarações **Cabeçalho**

```
function OConjunto.Intervalo (Objeto:
      OIntervalo; nota1, nota2: texto6): texto6;
```

Variáveis

`temp (numérica - integer)`: variável temporária para o código numérico do intervalo.

`tempr (numérica - real)`: variável temporária para a codificação oitava ponto intervalo.

`temps (literal - texto6)`: variável temporária para o literal do intervalo.

Rotina

```
begin
  temp := NotaIndice (nota2) - NotaIndice
     (nota1);
  case Objeto.tipo of
    0 : begin
          temp := temp mod oitava;
          if abs (temp) > (oitava div 2) then
            begin
              if temp > 0
                then dec (temp, oitava)
                else inc (temp, oitava);
            end;
        end;
    1 : temp := temp mod oitava;
    2 : ;
    3 : tempr := (temp div oitava) + ((temp mod
         oitava) / 100);
  end; {case}
  if (temp < 0) and not Objeto.direcao then
    begin
      if (Objeto.tipo > 0) and classe
        then temp := 12 + temp
        else temp := abs (temp);
    end;
  if Objeto.tipo = 3
    then Str (tempr:4:2, temps)
    else Str (temp, temps);
  Intervalo := temps;
end;
```

Operação

linha 1: calcula o intervalo;

linhas 2-15: reajusta o código do intervalo de acordo com o tipo;

linhas 3-11: (classes intervalares);

linha 4: reduz a um intervalo simples;

linhas 5-10: se o intervalo for maior que um trítono encontra o complemento;

linha 12: (intervalos simples) reduz a um intervalo simples;

linha 13: (intervalos compostos);

linha 14: (notação oitava ponto intervalo): põe o registro como parte inteira e o código da nota como parte decimal de um número real;

linhas 16-21: se a direção intervalar não deve ser considerada nos intervalos simples e compostos, reajusta para a representação em inteiro positivo;
linhas 22-25: converte o código do intervalo em literal e transfere para a função.

IntervaloTonal **função**

Descrição Retorna o intervalo tonal entre duas notas.

Parâmetros **Valores**

`Objeto (objeto - OIntervalo)`: objeto com a definição do tipo de intervalo desejado.

`nota1 (literal - texto6)`: primeira nota ou índice para uma nota do conjunto (quando antecedido pelo caráter "i").

`nota2 (literal - texto6)`: segunda nota ou índice para uma nota do conjunto (quando antecedido pelo caráter "i").

Declarações **Cabeçalho**

```
function OConjunto.IntervaloTonal (Objeto:
      OIntervalo; nota1, nota2: texto6): texto6;
```

Variáveis

`ponto (numérica - byte)`: posição do ponto na notação oitava ponto intervalo.

`nome (literal - texto6)`: código literal do intervalo.

Rotina

```
begin
  nome := Intervalo (Objeto, nota1, nota2);
  ponto := Pos ('.', nome);
  if ponto <> 0
    then IntervaloTonal := Copy (nome, 1, ponto
      - 1) + '.' + CodigoParaIntervalo (TpI
      (Copy (nome, ponto + 1, length (nome) -
      ponto)))
    else IntervaloTonal := CodigoParaIntervalo
      (TpI (nome));
end;
```

Operação linha 1: obtém o código do intervalo;
linhas 2-3: obtém a posição do ponto e testa sua existência;
linhas 4-6: o ponto existe, portanto notação oitava ponto nota. Copia o registro, após o ponto adiciona o intervalo convertido e transfere para a função;
linha 7: o ponto não existe. Converte o intervalo e transfere para a função.

Transpor — procedimento

Descrição Transpõe um conjunto de notas a um determinado intervalo.

Parâmetros **Valores**

`tintervalo (literal - texto6)`: intervalo de transposição.

Declarações **Cabeçalho**

```
procedure OConjunto.Transpor (tintervalo:
      texto6);
```

Variáveis

`itrans (numérica - integer)`: código numérico do intervalo de transposição.

`i (numérica - integer)`: contador para laço e índice para o conjunto de notas.

Rotina

```
begin
  itrans := IntervaloParaCodigo (tintervalo);
  for i := 1 to total_notas do
    codigo^[i] := Modulo (codigo^[i] + itrans);
end;
```

Operação Obtém o código numérico do intervalo de transposição e o adiciona a cada nota do conjunto.

Inverter **procedimento**

Descrição Inverte um conjunto de notas em torno de um eixo.

Parâmetros **Valores**

`eixo_inversao (literal - texto12)`: eixo de inversão representado por códigos literais de notas musicais ou por índices apontando para notas do conjunto.

Declarações **Cabeçalho**

```
procedure OConjunto.Inverter (eixo_inversao:
      texto12);
```

Variáveis

`indice (numérica - integer)`: índice de inversão.

`i (numérica - integer)`: contador para laço e índice para o conjunto de notas.

Rotina

```
begin
  i := Pos ('/', eixo_inversao);
  if i = 0
    then indice := NotaIndice (eixo_inversao) *
      2
    else indice := NotaIndice (Copy
      (eixo_inversao, 1, i - 1)) + NotaIndice
      (Copy (eixo_inversao, i + 1, length
      (eixo_inversao) - i));
  for i := 1 to total_notas do
  codigo^[i] := Modulo (indice - codigo^[i]);
end;
```

Operação linhas 1-2: encontra a posição da barra no eixo de inversão e testa para verificar se o eixo corresponde a um ponto entre duas notas;

linha 3: (inversão em torno de uma nota) - calcula o índice de inversão;

linhas 4-6: (inversão em torno de um ponto entre duas notas) - calcula o índice de inversão pela média aritmética entra as duas notas;

linhas 7-8: inverte o conjunto subtraindo os códigos das notas do índice de inversão.

Multiplicar **procedimento**

Descrição Multiplica as notas de um conjunto por um intervalo.

Parâmetros **Valores**

`multiplicador (numérico - integer)`: multiplicador correspondente a um intervalo.

Declarações **Cabeçalho**

```
procedure OConjunto.Multiplicar (multiplicador:
      integer);
```

Variáveis

`reg_atual (numérica - integer)`: registro (oitava) atual.

`i (numérica - integer)`: contador para laço e índice para o conjunto de notas.

Rotina

```
begin
  for i := 1 to total_notas do
    begin
      reg_atual := codigo^[i] div oitava;
      codigo^[i] := (codigo^[i] * multiplicador)
         mod 12;
      if oitava = stonal
        then codigo^[i] := ReajusteTonal
         (codigo^[i]);
      codigo^[i] := codigo^[i] + (oitava *
         reg_atual);
    end;
end;
```

Operação linha 3: separa o registro do código da nota;
linha 4: efetua a multiplicação;
linhas 5-6: se o sistema for tonal efetua o reajuste necessário;
linha 7: adiciona o registro.

Retrogradar **procedimento**

Descrição Dispõe um conjunto de notas de trás para a frente.

Declarações **Cabeçalho**

```
procedure OConjunto.Retrogradar;
```

Variáveis

`temp (numérica - integer)`: código temporário para as trocas.

`elementos (numérica - integer)`: índice para as trocas.

`i (numérica - integer)`: contador para laço e índice para o conjunto de notas.

Rotina

```
begin
  elementos := total_notas;
  for i := 1 to (elementos div 2) do
    begin
      temp := codigo^[elementos];
      codigo^[elementos] := codigo^[i];
      codigo^[i] := temp;
      dec (elementos);
    end;
end;
```

Operação Inicializa elementos com o total de notas do conjunto, divide o conjunto em duas partes e troca os códigos, em ordem crescente, da primeira parte com o códigos, em ordem decrescente, da segunda parte.

Rotar — procedimento

Descrição Efetua a rotação da notas de um conjunto.

Parâmetros **Valores**

`posicao (numérico - integer)`: número de posições a serem rotacionadas.

Declarações **Cabeçalho**

```
procedure OConjunto.Rotar (posicao: integer);
```

Variáveis

`limite (numérica - integer)`: limite do laço interno.

`temp (numérica - integer)`: código temporário para as trocas.

`i1 (numérica - integer)`: contador para o laço externo.

`i2` (`numérica` - **`integer`**): contador para o laço interno.

`fonte` (`numérica` - **`integer`**): índice do primeiro elemento da troca.

`destino` (`numérica` - **`integer`**): índice do segundo elemento da troca.

`passos` (`numérica` - **`integer`**): limite do laço externo.

Rotina

```
begin
  posicao := (total_notas - posicao) mod
      total_notas;
  if posicao = 0
    then exit;
  if posicao < 0
    then inc (posicao, total_notas);
  passos := Mdc (total_notas, posicao);
  limite := total_notas div passos - 1;
  for i1 := 1 to passos do
    begin
      destino := i1;
      fonte := destino;
      temp := codigo^[destino];
      for i2 := 1 to limite do
        begin
          dec (fonte, posicao);
          if fonte <= 0 then
            inc (fonte, total_notas);
          codigo^[destino] := codigo^[fonte];
          destino := fonte
        end;
      codigo^[destino] := temp
    end
end;
```

Operação

linha 1: ajusta *posicao* para não extrapolar o número de notas do conjunto;

linhas 2-3: se *posicao* for igual a zero não é necessário operar;

linhas 4-5: se *posicao* for negativo encontra o correspondente positivo;

linha 6: calcula o limite do laço externo;

linha 7: calcula o limite do laço interno;

linhas 8-22: laço externo;

linhas 10-12: inicializa os índices dos elementos de troca e armazena o código apontado por *destino* em *temp*;

linhas 13-20: laço interno;
linhas 15-17: atualiza *fonte*;
linha 18: transfere o código apontado por *fonte* para o local apontado por *destino*;
linha 19: atualiza *destino*;
linha 21: recupera o código armazenado em *temp*.

ExcluirRepeticao **procedimento**

Descrição Exclui as repetições de um conjunto de notas.

Parâmetros **Valores**

`opcao (numérico - byte)`: opção para a exclusão de notas, podendo assumir os seguintes valores e significados:

1 = excluir repetições tonais
2 = excluir todas repetições

Declarações **Cabeçalho**

```
procedure OConjunto.ExcluirRepeticao (opcao:
      byte);
```

Variáveis

`total (numérica - word)`: total de notas do conjunto resultante.

`i1 (numérica - word)`: contador para o laço externo e índice para as notas do conjunto.

`i2 (numérica - word)`: contador para o laço interno e índice para as notas do conjunto.

`temp (ponteiro - pvetorinteiro)`: localização do conjunto resultante na memória.

Rotinas Internas `Exclua (posicao, total)`

Rotina

```
begin
  i1 := 1;
  i2 := 2;
  total := total_notas;
```

```
  repeat
    repeat
      if ((opcao = 1) and (codigo^[i2] =
         codigo^[i1])) or (((opcao = 2) and
         ((codigo^[i2] mod 12) = (codigo^[i1]
         mod 12))) and (Abs (codigo^[i2] -
         codigo^[i1] <= 48))
        then
          begin
            Exclua (i2, total);
            dec (total);
          end
        else inc (i2);
    until i2 > total;
    inc (i1);
    i2 := i1 + 1;
  until i1 >= total;
  if total < total_notas then
    begin
      GetMem (temp, SizeOf (vetorinteiro) *
      total);
      if temp = nil
        then Exit;
      for i1 := 1 to total do
      temp^[i1] := codigo^[i1];
      FreeMem (codigo, SizeOf (vetorinteiro) *
      total_notas);
      codigo := temp;
      total_notas := total;
    end;
end;
```

Operação linhas 1-3: inicializa os limites iniciais dos laços e do total de notas;

linhas 4-18: laço externo;

linhas 5-15: laço interno - verifica se uma nota é repetida. Caso positivo exclui esta nota do conjunto e atualiza o total de notas;

linhas 16-17: atualiza os limites dos laços;

linhas 19-29: caso alguma nota tenha sido excluída:

linhas 21-23: reserva memória para o novo conjunto e testa se a reserva foi feita com sucesso. Caso negativo, aborta;

linhas 24-25: transfere as notas não repetidas para a localização apontada por *temp*;

linhas 26-28: libera a memória ocupada pelo conjunto original e atualiza os campos do objeto.

Exclua **procedimento (ExcluirRepeticao)**

Descrição Exclui uma nota e compacta o conjunto. Usado por e interno a *ExcluirRepeticao*

Parâmetros **Valores**

`posicao (numérico - integer)`: posição da nota a ser excluída.

`total (numérico - word)`: total atual de notas do conjunto.

Declarações **Cabeçalho**

```
procedure Exclua (posicao: integer; total:
      word);
```

Variáveis

`i (numérica - integer)`: contador de laço e índice para as notas do conjunto.

Rotina

```
begin
  for i := posicao to total do
    codigo^[i] := codigo^[i + 1];
end;
```

Operação Copia todas as notas posteriores à nota a ser excluída sobre a imediatamente anterior.

Ordenar **procedimento**

Descrição Dispõe as notas de um conjunto em ordem ascendente com ou sem eliminação de notas repetidas.

Parâmetros **Valores**

`opcao (numérico - byte)`: opção para a exclusão de notas repetidas, podendo assumir os seguintes valores e significados:

1 = deixar repetições
2 = excluir repetições tonais
3 = excluir todas repetições

Declaração **Cabeçalho**

```
procedure OConjunto.Ordenar (opcao: byte);
```

Rotinas Internas

```
Ordene (min, max)
(Menor (x, y))
```

Rotina

```
begin
  if opcao <> 1
    then ExcluirRepeticao (opcao - 1);
  Ordene (1, total_notas);
end;
```

Operação Exclui as notas repetidas conforme a opção e ordena as notas.

Ordene — procedimento (Ordenar)

Descrição Ordena ascendentemente as notas de um conjunto. Procedimento recursivo usado por e interno a *Ordenar*.

Parâmetros **Valores**

`min (numérico - integer)`: índice da nota inicial a ser ordenada.

`max: (numérico - integer)`: índice da nota final a ser ordenada.

Declarações **Cabeçalho**

```
procedure Ordene (min, max: integer);
```

Variáveis

`i1 (numérica - integer)`: índice para a nota anterior.

`i2 (numérica - integer)`: índice para a nota posterior.

`j1 (numérica - integer)`: código da nota central do conjunto.

`j2 (numérica - integer)`: armazenamento temporário para o código da nota a ser trocada.

Rotinas Internas

```
Menor (x, y)
```

Rotina

```
begin
  i1 := min;
  i2 := max;
  j1 := codigo^[(min + max) div 2];
  repeat
    while Menor (codigo^[i1], j1) do
      inc (i1);
    while Menor (j1, codigo^[i2]) do
      dec (i2);
    if i1 <= i2 then
      begin
        j2 := codigo^[i1];
        codigo^[i1] := codigo^[i2];
        codigo^[i2] := j2;
        inc (i1);
        dec (i2);
      end;
  until i1 > i2;
  if min < i2
    then Ordene (min, i2);
  if i1 < max
    then Ordene (i1, max);
end;
```

Operação

linhas 1-3: inicializa os índices *i1* e *i2* e transfere o código da nota central para *j1*;

linhas 4-17: laço principal;

linhas 5-6: encontra a nota anterior mais aguda que a central;

linhas 7-8: encontra a nota posterior mais grave que a central;

linhas 9-16: caso a nota mais aguda estiver antes da mais grave, efetua a troca e atualiza os índices;

linhas 18-19: caso o parâmetro *min* seja menor que a última nota posterior ordenada, ordene dentro dos novos limites;

linhas 20-21: caso o parâmetro *max* seja maior que a última nota anterior ordenada, ordene dentro dos novos limites.

Menor **função (Ordenar (Ordene))**

Descrição Encontra o menor entre dois códigos, baseado na frequência das notas correspondentes. Usado por e interno a *Ordene*

Parâmetros **Valores**

`x (numérica - integer)`: código da primeira nota.

`y (numérica - integer)`: código da segunda nota.

Declaração **Cabeçalho**

```
function Menor (x, y: integer): boolean;
```

Rotina

```
begin
  if Frequencia (CodigoParaNome (x) < Frequencia
      (CodigoParaNome (y))
    then Menor := true
    else Menor := false;
end;
```

Operação Caso a frequência da primeira nota seja menor que a da segunda, retorna verdadeira, caso contrário retorna falsa.

VetorIntervalar **função**

Descrição Fornece o vetor intervalar de um conjunto de notas.

Declarações **Cabeçalho**

```
function OConjunto.VetorIntervalar: texto6;
```

Variáveis

`vetint (vetor numérico - array [1..6] of byte)`: vetor intervalar numérico.

`i (numérico - byte)`: contador de laço e índice para o vetor intervalar numérico.

`s (literal - string[1])`: item do vetor intervalar.

`temp (literal - texto6)`: armazenamento temporário para o vetor intervalar.

Rotinas Internas

`NotaInicial (inicial)`

`(SegundaNota (inicial))`

Rotina

```
begin
  Ordenar (3);
  for i := 1 to 6 do
    vetint[i] := 0;
  NotaInicial (1);
  temp := '';
  for i := 1 to 6 do
    begin
      case vetint[i] of
        10 : s := 'A';
        11 : s := 'B';
        12 : s := 'C';
        else Str (vetint[i], s);
      end; {case}
      temp := temp + s;
    end;
  VetorIntervalar := temp
end;
```

Operação linha 1: ordena o conjunto eliminando todas as repetições;
linhas 2-3: inicializa *vetint*;
linha 4: chama o procedimento recursivo *NotaInicial*, o qual retorna com o vetor intervalar em *vetint*;
linha 5: inicializa *temp*;
linhas 6-15: transfere os numerais, convertidos para literais, de *vetint* para *temp*;
linha 16: transfere o vetor intervalar para a função.

NotaInicial — procedimento (VetorIntervalar)

Descrição Determina a nota inicial para o cálculo do vetor intervalar. Procedimento recursivo usado por e interno a *VetorIntervalar*.

Parâmetros **Valores**

`inicial (numérico - byte)`: índice da primeira nota.

Declaração **Cabeçalho**

```
procedure NotaInicial (inicial: byte);
```

Rotinas Internas

`SegundaNota (inicial)`

Rotina

```
begin
  if (total_notas - inicial) > 0
    then NotaInicial (inicial + 1);
  SegundaNota (inicial);
end;
```

Operação Enquanto *inicial* for menor que o total de notas do conjunto chama recursivamente o mesmo procedimento com o índice apontando para a próxima nota. Em seguida chama *SegundaNota* para calcular os intervalos e preencher *vetint*.

SegundaNota procedimento (VetorIntervalar (NotaInicial))

Descrição Calcula os intervalos entre as notas e preenche o vetor *vetint*. Usado por e interno a *NotaInicial*

Parâmetros **Valores**

`inicial (numérico - byte)`: índice da primeira nota.

Declarações **Cabeçalho**

```
procedure SegundaNota (inicial: byte);
```

Variáveis

`i (numérica - byte)`: contador de laço e índice para a segunda nota.

`interv (numérica - byte)`: intervalo entre as notas apontadas por *inicial* e *i*.

Rotina

```
begin
  for i := (inicial + 1) to total_notas do
    begin
      interv := abs (codigo^[inicial] -
         codigo^[i]) mod 12;
      if interv > 6
        then interv := 12 - interv;
      inc (vetint[interv]);
    end;
end;
```

Operação linha 1: inicializa o contador do laço com a nota imediatamente seguinte à inicial com o limite igual ao total de notas do conjunto;

linha 3: calcula o intervalo entre as notas apontadas por *inicial* e *i*;
linhas 4-5: transforma o intervalo em classe intervalar;
linha 6: registra a ocorrência da classe intervalar em *interv*.

VetorInversao **função**

Descrição Fornece o vetor de inversão ou vetor de índices de um conjunto de notas.

Declarações **Cabeçalho**

```
function OConjunto.VetorInversao: texto12;
```

Variáveis

`i1 (numérica - byte)`: contador de laço e índice para a primeira nota.

`i2 (numérica - byte)`: contador de laço interno e índice para a segunda nota.

`vetint (vetor numérico - array [0..11] of byte)`: vetor de inversão numérico.

`s (literal - string[1])`: item do vetor de inversão.

`temp (literal - texto12)`: armazenamento temporário do vetor de inversão.

Rotina

```
begin
  Ordenar (3);
  for i1 := 0 to 11 do
    vetint[i1] := 0;
  for i1 := 1 to total_notas do
    for i2 := 1 to total_notas do
      inc (vetint[(codigo^[i1] + codigo^[i2])
         mod 12]);
  temp := '';
  for i1 := 0 to 11 do
    begin
      case vetint[i1] of
        10 : s := 'A';
        11 : s := 'B';
        12 : s := 'C';
        else Str (vetint[i1], s);
      end; {case}
```

```
      temp := temp + s;
    end;
  VetorInversao := temp
end;
```

Operação linha 1: ordena o conjunto eliminando todas as repetições;
linhas 2-3: inicializa *vetint*;
linhas 4-6: preenche *vetint* com a soma dos códigos de todas as notas do conjunto;
linha 7: inicializa *temp*;
linhas 8-17: transfere os numerais, convertidos para literais, de *vetint* para *temp*;
linha 18: transfere o vetor de inversão para a função.

FormaNormal — procedimento

Descrição Dispõe o conjunto de notas em forma normal.

Declarações **Cabeçalho**

```
procedure OConjunto.FormaNormal;
```

Variáveis

`ivetor (objeto - OConjunto)`: armazenagem dos intervalos entre as notas adjacentes do conjunto ordenado.

`externo (numérica - integer)`: número de posições para a rotação do conjunto ordenado.

Rotina

```
begin
  Ordenar (3);
  ivetor.Iniciar (total_notas, stemperado,
      desconsiderar_registro);
  VetorIntervalo (ivetor);
  ivetor.OrdemIntervalar (externo);
  Rotar (externo);
  ivetor.Finalizar;
end;
```

Operação linha 1: ordena o conjunto eliminando todas as repetições;

linha 2: inicializa o objeto *ivetor*;
linha 3: obtém o vetor de intervalos do conjunto e o armazena em *ivetor*;
linha 4: obtém a forma mais compacta do vetor de intervalos e retorna o intervalo externo;
linha 5: efetua a rotação do conjunto até a posição indicada por *externo*;
linha 6: finaliza *ivetor*.

FormaPrima **procedimento**

Descrição Dispõe o conjunto em sua forma prima.

Declarações **Cabeçalho**

```
procedure OConjunto.FormaPrima;
```

Variáveis

`ivetor (objeto - OConjunto)`: armazenagem dos intervalos entre as notas adjacentes do conjunto ordenado.

`inicial (numérica - integer)`: intervalo inicial da forma normal do conjunto;

`externo (numérica - integer)`: número de posições para a rotação do conjunto ordenado.

`soma1 (numérica - integer)`: soma cumulativa dos intervalos correspondentes à forma normal do conjunto.

`soma2 (numérica - integer)`: soma cumulativa dos intervalos correspondentes à forma normal da inversão do conjunto.

`pc (numérica - integer)`: soma dos intervalos.

`i (numérica - integer)`: contador de laço e índice para as notas do conjunto.

Rotina

```
begin
  Ordenar (3);
  ivetor.Iniciar (total_notas, stemperado,
      desconsiderar_registro);
```

```
  VetorIntervalo (ivetor);
  ivetor.OrdemIntervalar (externo);
  soma1 := 0;
  soma2 := 0;
  pc := 0;
  inicial := ivetor.codigo^[1];
  for i := 1 to total_notas do
    begin
      soma1 := soma1 + pc + ivetor.codigo^[i];
      pc := pc + ivetor.codigo^[i]
    end;
  ivetor.Retrogradar;
  ivetor.OrdemIntervalar (externo);
  pc := 0;
  for i := 1 to total_notas do
    begin
      soma2 := soma2 + pc + ivetor.codigo^[i];
      pc := pc + ivetor.codigo^[i]
    end;
  if (soma2 > soma1) or ((soma2 = soma1) and
      (inicial < ivetor.codigo^[1])) then
    begin
      ivetor.Retrogradar;
      ivetor.OrdemIntervalar (externo)
    end;
  codigo^[1] := 0;
  for i := 2 to total_notas do
    codigo^[i] := codigo^[i - 1] +
      ivetor.codigo^[i - 1];
  if oitava = stonal then
    for i := 2 to total_notas do
      codigo^[i] := ReajusteTonal (codigo^[i]);
  ivetor.Finalizar;
end;
```

Operação linha 1: ordena o conjunto eliminando todas as repetições;

linha 2: inicializa o objeto *ivetor*;

linha 3: obtém o vetor de intervalos do conjunto ordenado e o armazena em *ivetor*;

linha 4: obtém a forma mais compacta do vetor de intervalos e retorna o intervalo externo;

linhas 5-7: inicializa *soma1*, *soma2* e *pc*;

linha 8: obtém o intervalo inicial da forma normal do conjunto;

linhas 9-13: obtém a soma cumulativa dos intervalos da forma normal do conjunto;

linha 14: retrograda o vetor de intervalos para corresponder à inversão do conjunto;
linha 15: obtém a forma mais compacta do vetor de intervalos correspondente à inversão e retorna o intervalo externo;
linha 16: reinicializa *pc*;
linhas 17-21: obtém a soma cumulativa dos intervalos da forma normal da inversão do conjunto;
linhas 22-27: compara as somas. Caso a inversão seja menos compacta retorna à ordem original;
linhas 28-30: inicializa a primeira nota e substitui o conjunto pela soma sucessiva dos intervalos;
linhas 31-33: caso o sistema utilizado seja o *stonal*, efetua o reajuste necessário;
linha 34: finaliza *ivetor*.

Complemento **procedimento**

Descrição Obtém o complemento do conjunto, isto é, as notas cromáticas ausentes do conjunto.

Declarações **Cabeçalho**

```
procedure OConjunto.Complemento;
```

Variáveis

`comp (vetor numérico - array [1..12] of integer)`: armazenagem temporária do complemento.

`i1 (numérica - integer)`: contador de laço, índice para as notas do conjunto e classes de notas.

`i2 (numérica - integer)`: índice para as notas do conjunto.

`i3 (numérica - integer)`: índice para *comp*.

Rotina

```
begin
  Ordenar (3);
  for i1 := 1 to total_notas do
    codigo^[i1] := codigo^[i1] mod 12;
```

```
  i1 := 0;
  i3 := 1;
  while i1 < 12 do
    begin
      i2 := 1;
      while (i2 <= total_notas) and (codigo^[i2]
         <> i1) do
        inc (i2);
      if i2 > total_notas then
        begin
          comp[i3] := i1;
          inc(i3)
        end;
      inc (i1)
    end;
  FreeMem (codigo, SizeOf (vetorinteiro) *
      total_notas);
  total_notas := 12 - total_notas;
  GetMem (codigo, SizeOf (vetorinteiro) *
      total_notas);
  if codigo = nil
    then Exit;
  for i1 := 1 to total_notas do
    if oitava = stonal
      then codigo^[i1] := ReajusteTonal
      (comp[i1])
      else codigo^[i1] := comp[i1];
end;
```

Operação linha 1: ordena o conjunto eliminando todas as repetições;

linhas 2-3: elimina o registro e, caso no sistema *stonal*, traduz para o sistema *stemperado*;

linha 4: inicializa *i1* como classe de notas 0;

linha 5: *i3* é inicializado para indicar o primeiro elemento do conjunto temporário;

linhas 6-17: obtém as notas ausentes do conjunto e armazena em *comp*;

linha 8: inicializa *i2* para indicar a primeira nota do conjunto;

linhas 9-10: procura a classe de notas indicada por *i1*;

linhas 11-15: caso a classe de notas indicada por *i1* inexista armazena em comp;

linha 16: próxima classe de notas;

linha 18: libera a memória reservada para o conjunto original;
linha 19: obtém o total de notas do complemento;
linhas 20-22: reserva memória para o complemento e testa se a reserva foi feita com sucesso. Caso negativo, aborta;
linhas 23-26: armazena as notas reajustando para o sistema *stonal*.

EixoSimetria **função**

Descrição Obtém o(s) eixo(s) de simetria de um conjunto de notas.

Declarações

Cabeçalho

```
function OConjunto.EixoSimetria: string;
```

Variáveis

`vetint (literal - texto12)`: vetor de inversão do conjunto.

`total (literal - string[1])`: total de notas do conjunto.

`temp (literal - string)`: armazenagem temporária dos eixos de inversão.

`eixo (lógica - boolean)`: indica a existência ou não de um eixo de inversão.

`eixo1 (vetor numérico - array [1..2] of integer)`: códigos correspondentes ao eixo de inversão ou às notas inferiores no caso de um eixo entre notas. *Eixo1[2]* está a um intervalo de trítono de *eixo1[1]*.

`eixo2 (vetor numérico - array [1..2] of integer)`: códigos correspondentes ao eixo de inversão ou às notas superiores no caso de um eixo entre notas. *Eixo2[2]* está a um intervalo de trítono de *eixo2[1]*.

`i1` (numérico - **`integer`**): contador para o laço externo e indicador de classes de notas.

`i2` (numérico - **`integer`**): contador para o laço interno e índice para os eixos.

Rotina

```
begin
  vetint := VetorInversao;
  temp := '';
  case total_notas of
    10 : total := 'A';
    11 : total := 'B';
    12 : total := 'C';
    else Str (total_notas, total);
  end; {case}
  eixo := false;
  for i1 := 0 to 11 do
    if vetint[i1 + 1] = total then
      begin
        eixo := true;
        eixo1[1] := trunc (i1 / 2);
        eixo1[2] := (eixo1[1] + 6) mod 12;
        eixo2[1] := round ((i1 + 0.4) / 2);
        eixo2[2] := (eixo2[1] + 6) mod 12;
        if oitava = stonal then
          for i2 := 1 to 2 do
            begin
              eixo1[i2] := ReajusteTonal
    (eixo1[i2]);
              eixo2[i2] := ReajusteTonal
    (eixo2[i2]);
            end;
          if eixo1[1] = eixo2[1]
            then temp :=   temp + ' ' +
              CodigoParaNome (eixo1[1]) + '-' +
              CodigoParaNome (eixo1[2])
            else temp :=   temp + ' ' +
              CodigoParaNome (eixo1[1]) + '/' +
              CodigoParaNome (eixo2[1]) + '-' +
              CodigoParaNome (eixo1[2]) + '/' +
              CodigoParaNome (eixo2[2]);
      end;
  if not eixo
    then temp := ' ';
  EixoSimetria := temp;
end;
```

Operação linhas 1-2: obtém o vetor de inversão do conjunto e inicializa *temp*;

linhas 3-8: transforma o numérico *total_notas* no literal *total*;

linha 9: inicializa *eixo*;
linhas 10-33: laço externo;
linha 11: testa a existência de um eixo comparando com o vetor de inversão;
linhas 13-17: assinala a existência de eixo e calcula os códigos numéricos;
linhas 18-23: efetua o reajuste necessário para o sistema *stonal*;:
linhas 24-32: converte os códigos em literais e armazena em *temp*;
linhas 34-35: conjunto assimétrico;
linha 36: transfere para a função.

Frequência **função**

Descrição Obtém a frequência em Hz de uma nota, em acordo com o sistema definido.

Parâmetros **Valores**

`inota (literal - texto6)`: nota ou índice para uma nota do conjunto do qual se deseja obter a frequência.

Declarações **Cabeçalho**

```
function OConjunto.Frequencia (inota: texto6):
      real;
```

Variáveis

`cod (numérica - integer)`: código da nota correspondente ao parâmetro *inota*.

Constantes

```
f0 (vetor numérico - array [1..7] of real =
 (1.62962962962920e+0001,
      1.83333333333430e+0001,
  2.06250000000000e+0001,
      2.17283950617420e+0001,
  2.44444444444380e+0001,
      2.75000000000000e+0001,
  3.09375000000000e+0001)):
```

frequências das notas diatônicas no sistema *stonal*.

`ln2 (numérica = 6.93147180560118e-0001)`: logaritmo natural de 2.

`lnsc (numérica = 6.56670345173325e-0002)`: logaritmo natural do semitono cromático de Pitágoras.

`do0 (numérica = 1.63515978312935e+0001)`: frequência do Do0 no sistema *stemperado*.

Rotina

```
begin
  cod := NotaIndice (inota);
  if oitava = stemperado
    then Frequencia := do0 * exp (cod / 12 *
      ln2)
    else Frequencia := f0[Linha (cod)] * exp
      ((Coluna (cod) - 8) * lnsc) * exp ((cod
      div oitava) * ln2);
end;
```

Operação linha 1: converte a literal *inota* na numérica *cod*; linhas 2-5: calcula a frequência de acordo com o sistema definido.

CodigoParaIntervalo — função

Descrição Converte o código numérico de um intervalo para o literal correspondente ao intervalo tonal.

Parâmetros

Valores

`cod (numérico - integer)`: código do intervalo a ser convertido.

Declarações

Cabeçalho

```
function OConjunto.CodigoParaIntervalo (cod:
      integer): texto6;
```

Variáveis

`grandeza_intervalar (literal - texto6)`: grandeza intervalar medida em graus da escala.

`qualidade_intervalar (literal - texto6)`: qualidade do intervalo.

`registro (numérica - integer)`: número de oitavas para os intervalos compostos.

`grandeza (numérica - integer)`: correspondente numérico à grandeza intervalar.

`qualidade (numérica - integer)`: índice para a qualidade intervalar.

`descendente (lógica - boolean)`: direção do intervalo.

Rotina

```
begin
  registro := 0;
  if cod < 0
    then
      begin
        cod := abs(cod);
        descendente := true;
      end
    else descendente := false;
  while cod >= oitava do
    begin
      dec (cod, oitava);
      inc (registro, 7)
    end; {while}
  if oitava = stemperado then
    begin
      cod := ReajusteTonal (cod);
      if cod in [1, 15, 56, 70]
        then inc (cod, 12);
    end;
  if cod > 61
    then inc (cod);
  grandeza := ((cod + 7) div 14) + 1;
  qualidade := (cod mod 7) * 3 + 1;
  if not Odd (cod div 7)
    then inc (qualidade, 24);
  if (grandeza in [4, 5]) and (qualidade > 18)
    then inc (qualidade, 3);
  if cod = 0
    then dec (qualidade, 3);
  inc (grandeza, registro);
  if descendente
    then grandeza := -grandeza;
  Str (grandeza, grandeza_intervalar);
  if intervalos[qualidade + 1] = ' '
    then qualidade_intervalar :=
      intervalos[qualidade]
    else qualidade_intervalar :=
      intervalos[qualidade] +
      intervalos[qualidade + 1];
  CodigoParaIntervalo := grandeza_intervalar +
      qualidade_intervalar;
end;
```

Operação linha 1: inicializa o número de oitavas;
linhas 2-7: verifica se o intervalo é descendente. Se for o caso, transforma-o em ascendente. Em qualquer caso, registra em *descendente*;
linhas 8-12: transforma um intervalo composto em simples e calcula o número de oitavas;
linhas 13-18: efetua os reajustes necessários para o sistema *stemperado*;
linhas 19-20: efetua reajuste necessário para o sistema *stonal*;
linhas 21-22: calcula a grandeza e o índice para a qualidade intervalar;
linhas 23-28: efetua reajustes necessários no índice da qualidade intervalar;
linha 29: adiciona as oitavas à grandeza intervalar;
linhas 30-31: recupera a direção intervalar;
linha 32: converte o numérico correspondente à grandeza intervalar em literal;
linhas 33-36: obtém a qualidade intervalar na constante *intervalos*;
linha 37: transfere para a função.

IntervaloParaCodigo — função

Descrição Converte um intervalo de qualquer tipo no código numérico correspondente.

Parâmetros **Valores**

`tintervalo (literal - texto6)`: intervalo a ser convertido.

Declarações **Cabeçalho**

```
function OConjunto.IntervaloParaCodigo
     (tintervalo: texto6): integer;
```

Variáveis

`ponto (numérica - byte)`: posição do ponto na notação oitava ponto intervalo.

`ioitava (numérica - integer)`: oitava na notação oitava ponto intervalo.

`itrans (numérica - integer)`: armazenamento temporário do código convertido.

Rotinas Internas

`IpC (tintervalo)`

Rotina

```
begin
  ponto := Pos ('.', tintervalo);
  if tintervalo[length (tintervalo)] in ['d',
      'm', 'J', 'M', 'A']
    then
      begin
        if ponto = 0
          then itrans := IpC (tintervalo)
          else
            begin
              ioitava := TpI (Copy (tintervalo,
                  1, ponto - 1));
              itrans := IpC (Copy (tintervalo,
                  ponto + 1, length (tintervalo)
                  - ponto));
              itrans := (ioitava * oitava) +
    itrans;
            end
      end
    else
      begin
        if ponto = 0
          then itrans := TpI (tintervalo)
          else itrans := round (int (TpR
            (tintervalo)) * oitava + frac (TpR
            (tintervalo)) * 100);
      end;
  IntervaloParaCodigo := itrans;
end;
```

Operação

linha 1: encontra a posição do ponto em *tintervalo*;
linha 2: verifica se o intervalo está na notação tonal;
linhas 3-14: intervalo tonal;
  linhas 5-6: não está na notação oitava ponto intervalo. Converte o literal *tintervalo* no código numérico *itrans*;
  linhas 7-13: notação oitava ponto intervalo;
    linha 9: copia a oitava convertida em numérico para *ioitava*;

linha 10: copia o código da nota convertido em numérico para *itrans*;
linha 11: calcula e adiciona a oitava;
linhas 15-21: intervalo em código;
linhas 17-18: não está na notação oitava ponto intervalo. Converte o literal *tintervalo* no código numérico *itrans*;
linhas 19-20: notação oitava ponto intervalo. calcula código correspondente ao intervalo;
linha 22: transfere para a função.

IpC **função (IntervaloParaCodigo)**

Descrição Converte um intervalo na notação tonal para o código numérico correspondente. Usada por e interna a *IntervaloParaCodigo*

Parâmetros **Valores**

`tintervalo (literal - texto6)`: intervalo a ser convertido.

Declarações **Cabeçalho**

```
function IpC (tintervalo: texto6): integer;
```

Variáveis

`descendente (lógica - boolean)`: direção do intervalo.

`codinterv (numérica - integer)`: armazenamento temporário do código numérico correspondente ao intervalo.

`linha (numérica - byte)`: intervalo em graus da escala e indicador da linha na tabela de intervalos.

`coluna (numérica - byte)`: indicador da coluna na tabela de intervalos.

`registro (numérica - byte)`: indicador de oitavas para os intervalos compostos.

`i (numérica - byte)`: posição do último número indicador da quantidade intervalar (graus da escala).

Constantes

`clinha (vetor numérico - array [1..8] of integer = (-6, 7, 21, 35, 49, 62, 76, 90))`: valores de referência para a tabela de intervalos.

Rotina

```
begin
  descendente := false;
  if tintervalo[1] = '-' then
    begin
      Delete (tintervalo, 1, 1);
      descendente := true
    end;
  i := 1;
  while tintervalo[i] in ['0'..'9'] do
    inc (i);
  dec (i);
  linha := TpI (Copy (tintervalo, 1, i));
  Delete (tintervalo, 1, i);
  if length (tintervalo) = 1
    then tintervalo := tintervalo + ' ';
  registro := 0;
  while linha > 8 do
    begin
      dec (linha, 7);
      inc (registro)
    end;
  coluna := (Pos (tintervalo, intervalos) - 1)
      div 3;
  if (linha in [1, 4, 5, 8]) and (coluna > 8)
    then dec (coluna);
  if coluna > 6
    then dec (coluna);
  codinterv := clinha[linha] + coluna;
  if codinterv = 96
    then inc (registro);
  codinterv := (codinterv mod oitava) + (oitava
      * registro);
  if descendente
    then IpC := -codinterv
    else IpC := codinterv;
end;
```

Operação

linhas 1- 6: verifica e registra se o intervalo é descendente. Se for, transforma-o em ascendente eliminando o sinal negativo;

linhas 7-10: obtém a posição do último digito referente à indicação da quantidade intervalar;

linhas 11-12: obtém a quantidade intervalar em graus da escala, e a exclui de *tintervalo*;

linhas 13-15: força *tintervalo* para conter dois caracteres e inicializa *registro*;
linhas 16-20: obtém em que linha da tabela de intervalos encontra-se o código e o número de oitavas do intervalo composto;
linhas 21-25: calcula a coluna na qual encontra-se o código na tabela de intervalos realizando os ajustes necessários;
linha 26: calcula o código do intervalo simples;
linhas 27-28: reajusta *registro* no caso de uma oitava justa;
linha 29: calcula o código final do intervalo;
linhas 30-32: transfere para a função recuperando a direção intervalar.

NomeParaCodigo — função

Descrição Método abstrato.

Parâmetros **Valores**

```
item (literal - texto6)
```

Declaração **Cabeçalho**

```
function OConjunto.NomeParaCodigo (item:
      texto6): integer; virtual;
```

Rotina

```
begin
end;
```

CodigoParaNome — função

Descrição Método abstrato.

Parâmetros **Valores**

```
item (numérico - integer)
```

Declaração **Cabeçalho**

```
function OConjunto.CodigoParaNome (item:
      integer): texto6; virtual;
```

Rotina

```
begin
end;
```

VetorIntervalo **procedimento**

Descrição Obtém o vetor de intervalos de um conjunto de notas. Usado por *FormaNormal* e *FormaPrima.*

Parâmetros **Variáveis**

`ivetor (objeto - OConjunto)`: intervalos entre as notas adjacentes do conjunto.

Declarações **Cabeçalho**

```
procedure OConjunto.VetorIntervalo (var ivetor:
      OConjunto);
```

Variáveis

`i1 (numérica - integer)`: contador de laço e índice para o código das notas.

`i2 (numérica - integer)`: índice para o código das notas.

Rotina

```
1  begin
2    for i1 := 1 to total_notas do
3      begin
4        if i1 = total_notas
5          then i2 := 1
6          else i2 := i1 + 1;
7        ivetor.codigo^[i1] := (codigo^[i2] -
8           codigo^[i1]) mod 12;
9        if ivetor.codigo^[i1] < 0
10         then inc (ivetor.codigo^[i1], 12);
11     end;
12 end;
```

Operação linhas 1-9: laço com *i1* apontando para cada nota do conjunto;
linhas 3-5: *i2* aponta para a nota seguinte a *i1*;
linha 6: obtém o intervalo entre as notas e armazena em *ivetor*;
linhas 7-8: caso o intervalo seja descendente torna-o ascendente.

OrdemIntervalar **procedimento**

Descrição Obtém o vetor intervalar ordenado na forma mais compacta e retorna o intervalo externo do conjunto de notas. Usado por *FormaNormal* e *FormaPrima*

Parâmetros **Variáveis**

`externo (numérico - integer)`: intervalo externo.

Declarações **Cabeçalho**

```
procedure OConjunto.OrdemIntervalar (var
      externo: integer);
```

Variáveis

`i (numérica - integer)`: contador de laço e índice para intervalos do conjunto.

`i1 (numérica - integer)`: índice para intervalos do conjunto.

`i2 (numérica - integer)`: índice para intervalos do conjunto.

Rotina

```
begin
  externo := total_notas;
  for i := 1 to total_notas do
    if codigo^[i] > codigo^[externo] then
      externo := i;
  for i := 1 to total_notas do
    if (i <> externo) and (codigo^[i] =
      codigo^[externo]) then
      begin
        i1 := i;
        i2 := externo;
        repeat
          if i1 = 1
            then i1 := total_notas
            else dec (i1);
          if i2 = 1
            then i2 := total_notas
            else dec (i2);
        until (codigo^[i1] <> codigo^[i2]) or
            (i1 = i) or (i2 = externo);
        if ((codigo^[i] + codigo^[i1]) >
            (codigo^[externo] + codigo^[i2])) or
            (((codigo^[i] + codigo^[i1]) =
            (codigo^[externo] + codigo^[i2]))
            and ((i < externo) and (externo <>
            total_notas)))
          then externo := i;
      end;
  Rotar (externo)
end;
```

Operação linhas 1-4: encontra o maior intervalo do conjunto;
linhas 5-24: verifica se o maior intervalo do conjunto é único;

linhas 6-24: caso o maior intervalo do conjunto não seja único;
linhas 8-9: *i1* aponta para o primeiro e *i2* aponta para o segundo maior intervalo;
linhas 10-18: verifica os intervalos anteriores aos maiores até encontrar um segundo maior intervalo ou completar o ciclo;
linhas 19-23: decide qual o maior intervalo que deve ser considerado como o intervalo externo;
linha 25: efetua a rotação do conjunto de intervalos colocando o maior intervalo como último.

NotaIndice **função**

Descrição Converte o índice da nota ou o código literal da nota no código numérico correspondente à nota. Usada por *Enarmonica*; *Intervalo*; *Inverter* e *Frequencia*

Parâmetros **Valores**

`ele (literal - texto6)`: índice da nota ou código literal da nota.

Declaração **Cabeçalho**

```
function OConjunto.NotaIndice (ele: texto6):
      integer;
```

Rotina

```
begin
  if UpCase (ele[1]) = "i"
    then
      begin
        Delete (ele, 1, 1);
        NotaIndice := codigo^[TpI (ele)];
      end
    else NotaIndice := NomeParaCodigo (ele);
end;
```

Operação linhas 1-7: verifica se *ele* é um índice ou o código literal de uma nota;

linhas 2-6: índice apontando para uma nota do conjunto;
linha 4: exclui o indicador de índice (caráter i);
linha 5: obtém o código da nota indicada pelo índice e transfere para a função;
linha 7: código literal de uma nota: converte para código e transfere para a função.

Métodos de NotaCodigo

Iniciar **construtor**

Descrição Inicializa o objeto *NotaCodigo*.

Parâmetros **Valores**

`ntotal (numérico - integer)`: total de notas do conjunto.

`sistema (numérico - byte)`: indica qual o sistema a ser utilizado (tonal ou temperado)

`reg (lógico - boolean)`: indica a consideração ou não do registro.

Variáveis

`codigo_inicial (sem tipo definido)`: vetor contendo os códigos literais das notas do conjunto.

Declarações **Cabeçalho**

```
constructor NotaCodigo.Iniciar (ntotal: integer;
      sistema: byte; reg: boolean; var
      codigo_inicial);
```

Variáveis

`i (numérica - word)`: contador de laço.

`s (numérica - word)`: memória, em **bytes**, ocupada pelos códigos literais do conjunto de notas.

`temp (ponteiro - pvetortexto)`: aponta para a armazenagem temporária dos códigos literais do conjunto de notas.

Rotina

```
begin
  if not OConjunto.Iniciar (ntotal, sistema,
      reg)
    then Fail;
  s := SizeOf (vetortexto) * total_notas;
  GetMem (temp, s);
  if temp = nil then
    begin
      OConjunto.Finalizar;
      Fail;
    end;
  Move (codigo_inicial, temp^, s);
  for i := 1 to ntotal do
    codigo^[i] := Modulo ( NomeParaCodigo
      (temp^[i]));
  FreeMem (temp, s);
  temp := nil;
end;
```

Operação linhas 1-2: inicializa os objetos antecessores testando para verificar se obteve sucesso. Caso negativo, aborta;

linhas 3-9: calcula e reserva a memória necessária para armazenagem do conjunto de notas. Em seguida testa para verificar se a reserva teve sucesso e em caso contrário finaliza o objeto *OConjunto* e aborta;

linha 10: transfere os códigos literais para o vetor apontado por *temp*;

linhas 11-12: converte os códigos literais para numéricos armazenando-os em *codigo*;

linhas 13-14: libera a memória e anula *temp*.

NomeParaCodigo — função

Descrição Converte um código literal em um código numérico correspondente a uma nota musical.

Parâmetros **Valores**

`item (literal - texto6)`: código literal a ser convertido.

Declaração **Cabeçalho**

```
function NotaCodigo.NomeParaCodigo (item:
     texto6): integer; virtual;
```

Rotina
```
begin
  NomeParaCodigo := TpI (item);
end;
```

Operação Converte o código da nota de literal para numérico.

CodigoParaNome — função

Descrição Converte um código numérico em um código literal correspondente a uma nota musical.

Parâmetros **Valores**

`item (numérico - integer)`: código numérico a ser convertido.

Declarações **Cabeçalho**
```
function NotaCodigo.CodigoParaNome (item:
      integer): texto6; virtual;
```

Variáveis

`temp (literal - texto6)`: armazenagem temporária do código literal.

Rotina
```
begin
  Str (item, temp);
  CodigoParaNome := temp;
end;
```

Operação Converte o código numérico para literal e o transfere para a função.

Métodos de OitavaPontoNota

Iniciar — construtor

Descrição Inicializa o objeto *OitavaPontoNota*.

Parâmetros **Valores**

`ntotal (numérico - integer)`: total de notas do conjunto.

`sistema (numérico - byte)`: indica qual o sistema a ser utilizado (tonal ou temperado)

`reg (lógico - boolean)`: indica a consideração ou não do registro.

Variáveis

`opn_inicial (sem tipo definido)`: vetor contendo os códigos literais das notas do conjunto.

Declarações **Cabeçalho**

```
constructor OitavaPontoNota.Iniciar (ntotal:
      integer; sistema: byte; reg: boolean; var
      opn_inicial);
```

Variáveis

`i (numérica - word)`: contador de laço.

`s (numérica - word)`: memória, em **bytes**, ocupada pelos códigos literais do conjunto de notas.

`temp (ponteiro - pvetortexto)`: aponta para a armazenagem temporária dos códigos literais do conjunto de notas.

Rotina

```
begin
  if not OConjunto.Iniciar (ntotal, sistema,
      reg)
    then Fail;
  s := SizeOf (vetortexto) * total_notas;
  GetMem (temp, s);
  if temp = nil then
    begin
      Oconjunto.Finalizar;
      Fail;
    end;
  Move (opn_inicial, temp^, s);
  for i := 1 to ntotal do
    codigo^[i] := Modulo (NomeParaCodigo
      (temp^[i]));
  FreeMem (temp, s);
  temp := nil;
end;
```

Operação linhas 1-2: inicializa os objetos antecessores testando para verificar se obteve sucesso. Caso negativo, aborta;

linhas 3-9: calcula e reserva a memória necessária para armazenagem do conjunto de notas. Em seguida testa para verificar se a reserva teve sucesso e em caso contrário finaliza o objeto *OConjunto* e aborta;

linha 10: transfere os códigos literais para o vetor apontado por *temp*;

linhas 11-12: converte os códigos literais para numéricos armazenando-os em *codigo*;

linhas 13-14: libera a memória e anula *temp*.

NomeParaCodigo — função

Descrição Converte um código literal em um código numérico correspondente a uma nota musical.

Parâmetros **Valores**

`item (literal - texto6)`: código literal a ser convertido.

Declaração **Cabeçalho**

```
function OitavaPontoNota.NomeParaCodigo (item:
      texto6): integer; virtual;
```

Rotina

```
begin
  NomeParaCodigo := round (int (TpR (item)) *
      oitava + frac (TpR (item)) * 100)
end;
```

Operação Converte o código da nota de literal para numérico.

CodigoParaNome — função

Descrição Converte um código numérico em um código literal correspondente a uma nota musical.

Parâmetros **Valores**

`item (numérico - integer)`: código numérico a ser convertido.

Declarações **Cabeçalho**

```
function OitavaPontoNota.CodigoParaNome (item:
      integer): texto6; virtual;
```

Variáveis

`temp (literal - texto6)`: armazenagem temporária do código literal.

Rotina

```
begin
  Str ((item div oitava) + ((item mod oitava) /
      100):4:2, temp);
  CodigoParaNome := temp;
end;
```

Operação Converte o código numérico para literal e o transfere para a função.

Métodos de NotaNome

Iniciar — construtor

Descrição Inicializa o objeto *NotaNome*.

Parâmetros **Valores**

`ntotal (numérico - integer)`: total de notas do conjunto.

`sistema (numérico - byte)`: indica qual o sistema a ser utilizado (tonal ou temperado)

`reg (lógico - boolean)`: indica a consideração ou não do registro.

Variáveis

`nome_inicial (sem tipo definido)`: vetor contendo os códigos literais das notas do conjunto.

Declarações **Cabeçalho**

```
constructor NotaNome.Iniciar (ntotal: integer;
      sistema: byte; reg: boolean; var
      nome_inicial);
```

Variáveis

`i (numérica - word)`: contador de laço.

`s (numérica - word)`: memória, em **bytes**, ocupada pelos códigos literais do conjunto de notas.

`temp (ponteiro - pvetortexto)`: aponta para a armazenagem temporária dos códigos literais do conjunto de notas.

Rotina

```
begin
  if not OConjunto.Iniciar (ntotal, sistema,
      reg)
    then Fail;
  s := SizeOf (vetortexto) * total_notas;
  GetMem (temp, s);
  if temp = nil then
    begin
      OConjunto.Finalizar;
      Fail;
    end;
  Move (nome_inicial, temp^, s);
  for i := 1 to ntotal do
    codigo^[i] := Modulo (NomeParaCodigo
      (temp^[i]));
  FreeMem (temp, s);
  temp := nil;
end;
```

Operação

linhas 1-2: inicializa os objetos antecessores testando para verificar se obteve sucesso. Caso negativo, aborta;

linhas 3-9: calcula e reserva a memória necessária para armazenagem do conjunto de notas. Em seguida testa para verificar se a reserva teve sucesso e em caso contrário finaliza o objeto *OConjunto* e aborta;

linha 10: transfere os códigos literais para o vetor apontado por *temp*;

linhas 11-12: converte os códigos literais para numéricos armazenando-os em *codigo*;

linhas 13-14: libera a memória e anula *temp*.

NomeParaCodigo **função**

Descrição Converte um código literal em um código numérico correspondente a uma nota musical.

Parâmetros **Valores**

`item (literal - texto6)`: código literal a ser convertido.

Declaração **Cabeçalho**

```
function NotaNome.NomeParaCodigo (item: texto6):
      integer; virtual;
```

Rotina

```
begin
  NomeParaCodigo := NpC (item, nomes);
end;
```

Operação Converte o código da nota de literal para numérico.

CodigoParaNome **função**

Descrição Converte um código numérico em um código literal correspondente a uma nota musical.

Parâmetros **Valores**

`item (numérico - integer)`: código numérico a ser convertido.

Declaração **Cabeçalho**

```
function NotaNome.CodigoParaNome (item:
      integer): texto6; virtual;
```

Rotina

```
begin
  CodigoParaNome := CpN (item, nomes);
end;
```

Operação Converte o código numérico para literal.

Métodos de NotaLetra

Iniciar **construtor**

Descrição Inicializa o objeto *NotaLetra*.

Parâmetros **Valores**

`ntotal (numérico - integer)`: total de notas do conjunto.

`sistema (numérico - byte)`: indica qual o sistema a ser utilizado (tonal ou temperado)

`reg (lógico - boolean)`: indica a consideração ou não do registro.

Variáveis

`letra_inicial (sem tipo definido)`: vetor contendo os códigos literais das notas do conjunto.

Declarações **Cabeçalho**

```
constructor NotaLetra.Iniciar (ntotal: integer;
      sistema: byte; reg: boolean; var
      letra_inicial);
```

Variáveis

`i (numérica - word)`: contador de laço.

`s (numérica - word)`: memória, em **bytes**, ocupada pelos códigos literais do conjunto de notas.

`temp (ponteiro - pvetortexto)`: aponta para a armazenagem temporária dos códigos literais do conjunto de notas.

Rotina

```
begin
  if not OConjunto.Iniciar (ntotal, sistema,
      reg)
    then Fail;
  s := SizeOf (vetortexto) * total_notas;
  GetMem (temp, s);
  if temp = nil then
    begin
      OConjunto.Finalizar;
      Fail;
    end;
  Move (letra_inicial, temp^, s);
  for i := 1 to ntotal do
    codigo^[i] := Modulo (NomeParaCodigo
      (temp^[i]));
  FreeMem (temp, s);
  temp := nil;
end;
```

Operação linhas 1-2: inicializa os objetos antecessores testando para verificar se obteve sucesso. Caso negativo, aborta;
linhas 3-9: calcula e reserva a memória necessária para armazenagem do conjunto de notas. Em seguida testa para verificar se a reserva teve sucesso e em caso contrário finaliza o objeto *OConjunto* e aborta;
linha 10: transfere os códigos literais para o vetor apontado por *temp*;
linhas 11-12: converte os códigos literais para numéricos armazenando-os em *codigo*;
linhas 13-14: libera a memória e anula *temp*.

NomeParaCodigo — função

Descrição Converte um código literal em um código numérico correspondente a uma nota musical.

Parâmetros **Valores**
`item (literal - texto6)`: código literal a ser convertido.

Declaração **Cabeçalho**

```
function NotaLetra.NomeParaCodigo (item:
      texto6): integer; virtual;
```

Rotina

```
begin
  NomeParaCodigo := NpC (item, letras);
end;
```

Operação Converte o código da nota de literal para numérico.

CodigoParaNome — função

Descrição Converte um código numérico em um código literal correspondente a uma nota musical.

Parâmetros **Valores**

`item (numérico - integer)`: código numérico a ser convertido.

Declaração **Cabeçalho**

```
function NotaLetra.CodigoParaNome (item:
      integer): texto6; virtual;
```

Rotina

```
begin
  CodigoParaNome := CpN (item, letras);
end;
```

Operação Converte o código numérico para litera

ÍNDICE

A

acidentes 79, 85, 86, 89

C

classe. *Vide* ONota
codigo. *Vide* OConjunto
CodigoParaIntervalo 29, 42, 80, 100, 123, 124
CodigoParaNome 27, 28, 29, 36, 38, 40, 43, 63, 68, 89, 93, 95, 98, 111, 121, 129, 135, 137, 138, 140, 142, 143
Coluna 30, 87, 88, 89, 123
Complementação 10
Complemento 29, 50, 80, 118
ConjuntoAtual 28, 51, 94
considerar_direcao 25, 47, 60, 66, 70, 91
considerar_registro 25, 37, 39, 41, 42, 45, 47, 48, 49, 51, 52, 54, 57, 59, 60, 62, 63, 64, 70, 84
CpN 30, 86, 87, 89, 140, 143

D

desconsiderar_direcao 26, 45, 91
desconsiderar_registro 26, 36, 38, 40, 43, 44, 45, 46, 49, 50, 52, 53, 54, 55, 56, 57, 58, 60, 66, 73, 84, 115, 116
direcao. *Vide* Enarmonica; OIntervalo

E

Eixo de Simetria 12
EixoSimetria 29, 43, 80, 120, 121
Enarmonia 3, 4
Enarmonica 28, 43, 44, 84, 97, 98, 132
  direcao 28, 43, 98
Exclua 106, 107, 108
ExcluirRepeticao 28, 51, 52, 78, 106, 108, 109
Exclusão de Repetições 10

F

Finalizar 28, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 66, 71, 75, 78, 91, 92, 115, 117, 134, 136, 139, 141
Forma Normal 13
Forma Prima 13
FormaNormal 29, 52, 71, 75, 115, 130
FormaPrima 29, 53, 71, 75, 80, 116, 130
Frequencia 29, 44, 45, 86, 87, 111, 122, 123, 132
Freqüência 12, 14

I

iclasse 20, 23, 26, 45, 90
icomposto 20, 23, 26, 47, 91
Iniciar 27, 28, 29, 30, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 66, 70, 71, 73, 83, 84, 90, 91, 115, 116, 133, 134, 135, 136, 138, 139, 140, 141
Intervalo 12, 14, 22, 28, 42, 45, 46, 62, 65, 66, 70, 98, 99, 100, 132
IntervaloParaCodigo 29, 46, 81, 82, 101, 125, 126, 127
intervalos 79, 124, 125, 128
IntervaloTonal 14, 20, 22, 28, 46, 47, 62, 68, 79, 81, 100
Inversão 4, 5, 15
Inverter 28, 32, 53, 54, 84, 102, 132
ioitavaponto 20, 23, 27, 91
IpC 81, 126, 127, 128
isimples 20, 23, 27, 60, 66, 70, 91

L

letras 79, 85, 89, 142, 143
Linha 30, 86, 87, 89, 123

M

Mdc 80, 81, 105
Menor 109, 110, 111
Modulo 30, 84, 92, 94, 98, 101, 102, 134, 136, 139, 141
Multiplicação 6
Multiplicar 28, 54, 80, 103

N

NomeParaCodigo 27, 28, 29, 37, 39, 41, 47, 63, 64, 81, 82, 85, 92, 94, 129, 132, 134, 135, 136, 137, 139, 140, 141, 142
nomes 79, 85, 89, 140
NotaAtual 28, 36, 38, 40, 44, 47, 48, 50, 52, 53, 54, 55, 56, 58, 59, 60, 61, 62, 66, 68, 70, 71, 75, 78, 93
NotaCodigo 13, 17, 19, 20, 22, 25, 26, 27, 29, 30, 32, 35, 36, 37, 41, 42, 45, 48, 49, 53, 54, 55, 72, 81, 84, 133, 134, 135
NotaIndice 29, 81, 98, 99, 102, 123, 132
NotaInicial 111, 112, 113

NotaLetra 17, 19, 20, 25, 26, 27, 29, 30, 32, 37, 38, 39, 41, 46, 50, 51, 57, 58, 59, 65, 79, 84, 85, 89, 140, 143
NotaNome 17, 19, 20, 25, 26, 28, 29, 30, 32, 39, 40, 41, 43, 44, 45, 50, 51, 54, 56, 59, 65, 67, 79, 84, 85, 89, 138, 140
NovaNota 28, 44, 55, 84, 92
NovoConjunto 28, 56, 76, 84, 93
NovoTotal 28, 56, 57, 96
NpC 30, 84, 85, 86, 140, 142

O

OConjunto 25, 26, 27, 28, 29, 30, 31, 32, 35, 37, 39, 41, 59, 61, 62, 64, 91, 92, 93, 94, 95, 96, 97, 98, 100, 101, 102, 103, 104, 106, 109, 111, 114, 115, 116, 118, 120, 122, 123, 125, 129, 130, 131, 132, 134, 136, 137, 139, 141, 142
  codigo 28, 42, 91, 92, 93, 94, 95, 96, 97, 101, 102, 103, 104, 105, 107, 108, 110, 113, 114, 117, 118, 119, 130, 131, 132, 134, 136, 137, 139, 141, 142
  total_notas 28, 41, 42, 47, 48, 55, 91, 92, 94, 95, 96, 97, 101, 102, 103, 104, 105, 106, 107, 109, 113, 114, 115, 116, 117, 118, 119, 121, 130, 131, 134, 136, 139, 141
OIntervalo 25, 26, 27, 28, 29, 31, 45, 46, 60, 61, 65, 67, 90, 98, 100
  direcao 29, 61, 90, 99
  tipo 29, 61, 90, 99
oitava. *Vide* ONota
OitavaPontoNota 17, 19, 20, 25, 26, 29, 31, 32, 41, 46, 47, 49, 52, 59, 60, 62, 63, 65, 82, 84, 135, 136, 137, 138
ONota 25, 26, 28, 29, 30, 31, 32, 41, 64, 83, 84, 85, 88, 89, 91, 92
  classe 30, 64, 83, 84, 89, 97, 99
  oitava 30, 64, 83, 84, 86, 87, 88, 89, 96, 97, 99, 103, 117, 119, 121, 123, 124, 126, 128, 137, 138
OrdemIntervalar 29, 115, 117, 130, 131
Ordenação 11
Ordenar 28, 57, 58, 108, 109, 111, 112, 114, 115, 116, 118
Ordene 109, 110, 111

P

PNotaCodigo 30
PNotaLetra 30
PNotaNome 30
POConjunto 31
POIntervalo 31
POitavaPontoNota 31
PONota 31
pvetorinteiro 28, 31, 42, 91, 106
pvetortexto 31, 94, 95, 133, 136, 139, 141

R

ReajusteTonal 80, 103, 117, 119, 121, 124
RegistroAtual 28, 48, 95, 96
Retrogradação 7
Retrogradar 28, 58, 103, 117
Rotação 8
Rotar 28, 59, 80, 104, 115, 131

S

SegundaNota 111, 112, 113
stemperado 3, 4, 6, 12, 13, 14, 32, 36, 37, 39, 40, 41, 45, 48, 49, 50, 53, 54, 55, 56, 58, 59, 60, 62, 64, 66, 73, 80, 83, 115, 116, 119, 123, 124, 125
stonal 3, 6, 14, 32, 36, 38, 42, 43, 44, 45, 46, 47, 50, 51, 52, 54, 57, 60, 63, 70, 80, 83, 103, 117, 118, 119, 120, 121, 122, 125

T

texto12 28, 32, 49, 53, 102, 114, 120
texto6 27, 28, 29, 30, 32, 33, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 62, 63, 65, 66, 67, 68, 69, 70, 71, 81, 82, 85, 89, 92, 93, 97, 98, 99, 100, 101, 111, 122, 123, 125, 127, 129, 132, 134, 135, 137, 138, 140, 142, 143
tipo. *Vide* OIntervalo
total_notas. *Vide* OConjunto
TotalNotas 28, 48, 49, 50, 52, 53, 57, 58, 71, 75, 78, 96
TpI 81, 82, 85, 100, 126, 128, 132, 135
TpR 82, 83, 126, 137
Transferir 28, 59, 60, 96, 97
Transpor 28, 60, 65, 66, 70, 84, 101
Transposição 9

V

Vetor Intervalar 15
Vetor Inversão 15
vetorinteiro 31, 33, 91, 92, 97, 107, 119
VetorIntervalar 28, 32, 49, 111, 112, 113
VetorIntervalo 29, 115, 117, 130
VetorInversao 28, 32, 49, 50, 114, 115, 121
vetortexto 31, 33, 94, 95, 134, 136, 139, 141