

ANEXO 1 - ASCII

A Tabela ASCII (American Standard Code for Information Interchange) é usada pela maior parte da indústria de computadores para a troca de informações. Cada caractere é representado por um código de 8 bits (um byte). Abaixo mostramos a tabela ASCII de 7 bits. Existe uma tabela estendida para 8 bits que inclui os caracteres acentuados.

Tabela ASCII – 127 códigos (7 BITS)

Caractere	Decimal	Hexadecimal	Binário	Comentário
NUL	00	00	0000 0000	Caracter Nulo
SOH	01	01	0000 0001	Começo de cabeçalho de transmissão
STX	02	02	0000 0010	Começo de texto
ETX	03	03	0000 0011	Fim de texto
EOT	04	04	0000 0100	Fim de transmissão
ENQ	05	05	0000 0101	Interroga
ACK	06	06	0000 0110	Confirmação
BEL	07	07	0000 0111	Sinal sonoro
BS	08	08	0000 0100	Volta um caracter
HT	09	09	0000 1001	Tabulação Horizontal
LF	10	0A	0000 1010	Próxima linha
VT	11	0B	0000 1011	Tabulação Vertical
FF	12	0C	0000 1100	Próxima Página
CR	13	0D	0000 1101	Início da Linha
SO	14	0E	0000 1110	Shift-out
SI	15	0F	0000 1111	Shift-in
DLE	16	10	0001 0000	Data link escape
D1	17	11	0001 0001	Controle de dispositivo
D2	18	12	0001 0010	Controle de dispositivo
D3	19	13	0001 0011	Controle de dispositivo
D4	20	14	0001 0100	Controle de dispositivo
NAK	21	15	0001 0101	Negativa de Confirmação
SYN	22	16	0001 0110	Synchronous idle
ETB	23	17	0001 0111	Fim de transmissão de bloco
CAN	24	18	0001 1000	Cancela
EM	25	19	0001 1001	Fim de meio de transmissão
SUB	26	1A	0001 1010	Substitui

ESC	27	1B	0001 1011	Escape
FS	28	1C	0001 1100	Separador de Arquivo
GS	29	1D	0001 1101	Separador de Grupo
RS	30	1E	0001 1110	Separador de registro
US	31	1F	0001 1111	Separador de Unidade
Espaço	32	20	0010 0000	
!	33	21	0010 0001	
"	34	22	0010 0010	
#	35	23	0010 0011	
\$	36	24	0010 0100	
%	37	25	0010 0101	
&	38	26	0010 0110	
'	39	27	0010 0111	
(40	28	0010 1000	
)	41	29	0010 1001	
*	42	2A	0010 1010	
+	43	2B	0010 1011	
,	44	2C	0010 1100	
-	45	2D	0010 1101	
.	46	2E	0010 1110	
/	47	2F	0010 FFFF	
0	48	30	0011 0000	
1	49	31	0011 0001	
2	50	32	0011 0010	
3	51	33	0011 0011	
4	52	34	0011 0100	
5	53	35	0011 0101	
6	54	36	0011 0110	
7	55	37	0011 0111	
8	56	38	0011 1000	
9	57	39	0011 1001	
:	58	3A	0011 1010	
;	59	3B	0011 1011	
<	60	3C	0011 1100	
=	61	3D	0011 1101	
>	62	3E	0011 1110	
?	63	3F	0011 1111	
@	64	40	0100 0000	
A	65	41	0100 0001	
B	66	42	0100 0010	
C	67	43	0100 0011	
D	68	44	0100 0100	
E	69	45	0100 0101	
F	70	46	0100 0110	

G	71	47	0100 0111
H	72	48	0100 1000
I	73	49	0100 1001
J	74	4A	0100 1010
K	75	4B	0100 1011
L	76	4C	0100 1100
M	77	4D	0100 1101
N	78	4E	0100 1110
O	79	4F	0100 1111
P	80	50	0101 0000
Q	81	51	0101 0001
R	82	52	0101 0010
S	83	53	0101 0011
T	84	54	0101 0100
U	85	55	0101 0101
V	86	56	0101 0110
W	87	57	0101 0111
X	88	58	0101 1000
Y	89	59	0101 1001
Z	90	5A	0101 1010
[91	5B	0101 1011
\	92	5C	0101 1100
]	93	5D	0101 1101
^	94	5E	0101 1110
_	95	5F	0101 1111
`	96	60	0110 0000
a	97	61	0110 0001
b	98	62	0110 0010
c	99	63	0110 0011
d	100	64	0110 0100
e	101	65	0110 0101
f	102	66	0110 0110
g	103	67	0110 0111
h	104	68	0110 1000
i	105	69	0110 1001
j	106	6A	0110 1010
k	107	6B	0110 1011
l	108	6C	0110 1100
m	109	6D	0110 1101
n	110	6E	0110 1110
o	111	6F	0110 1111
p	112	70	0111 0000
q	113	71	0111 0001
r	114	72	0111 0010
s	115	73	0111 0011

t	116	74	0111 0100
u	117	75	0111 0101
v	118	76	0111 0110
w	119	77	0111 0111
x	120	78	0111 1000
y	121	79	0111 1001
z	122	7A	0111 1010
{	123	7B	0111 1011
	124	7C	0111 1100
}	125	7D	0111 1101
~	126	7E	0111 1110
DELETE	127	7F	0111 1111

TABELA ASCII 256 CÓDIGOS (8 BITS)
Código ASCII estendido para IBM PC

De uma forma geral, os caracteres ASCII possuem valores de 0 a 127 (decimal) ou 7F (hexadecimal). Contudo, quando o IBM PC foi desenvolvido a placa de vídeo possuía apenas um byte para definir cada caractere em um monitor de 80x25 caracteres. Com o avanço da tecnologia os monitores passaram a possuir muito mais definição, superando e muito o número máximo de caracteres por tela desde sua concepção. Sendo assim porque não se adicionar mais um bit por caractere? Porque não inventar, representar, outros 128 caracteres novos? Para isto foi criado o **Código ASCII Estendido**, o qual é mostrado conforme utilizado pelo Windows:

[illegible]

Para quem quiser converter para decimal e não sabe, segue uma tabela prática. (você pode utilizar a calculadora do Windows).

Convertendo Hexadecimal para Decimal

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F.
0	000	001	002	003	004	005	006	007	008	009	010	011	012	013	014	015
1	016	017	018	019	020	021	022	023	024	025	026	027	028	029	030	031
2	032	033	034	035	036	037	038	039	040	041	042	043	044	045	046	047
3	048	049	050	051	052	053	054	055	056	057	058	059	060	061	062	063
4	064	065	066	067	068	069	070	071	072	073	074	075	076	077	078	079
5	080	081	082	083	084	085	086	087	088	089	090	091	092	093	094	095
6	096	097	098	099	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

ANEXO 2 –

INSTRUMENTOS DO CANAL 1-9, 11-16

AcousticGrandPiano	0
BrightAcousticPiano	1
ElectricGrandPiano	2
Honky-tonkPiano	3
ElectricPiano1	4
ElectricPiano2	5
Harpsichord	6
Clavi	7
Celesta	8
Glockenspiel	9
MusicBox	10
Vibraphone	11
Marimba	12
Xylophone	13
TubularBells	14
Dulcimer	15
DrawbarOrgan	16
PercussiveOrgan	17
RockOrgan	18
ChurchOrgan	19
ReedOrgan	20
Accordion	21
Harmonica	22
TangoAccordion	23
AcousticGuitar(nylon)	24
AcousticGuitar(steel)	25
ElectricGuitar(jazz)	26
ElectricGuitar(clean)	27
ElectricGuitar(muted)	28
OverdrivenGuitar	29
DistortionGuitar	30
Guitarharmonics	31
AcousticBass	32
ElectricBass(finger)	33
ElectricBass(pick)	34

FretlessBass	35
SlapBass1	36
SlapBass2	37
SynthBass1	38
SynthBass2	39
Violin	40
Viola	41
Cello	42
Contrabass	43
TremoloStrings	44
PizzicatoStrings	45
OrchestralHarp	46
Timpani	47
StringEnsemble1	48
StringEnsemble2	49
SynthStrings1	50
SynthStrings2	51
ChoirAahs	52
VoiceOohs	53
SynthVoice	54
OrchestraHit	55
Trumpet	56
Trombone	57
Tuba	58
MutedTrumpet	59
FrenchHorn	60
BrassSection	61
SynthBrass1	62
SynthBrass	63
SopranoSax	64
AltoSax	65
TenorSax	66
BaritoneSax	67
Oboe	68
EnglishHorn	69
Bassoon	70
Clarinet	71
Piccolo	72
Flute	73
Recorder	74
PanFlute	75
BlownBottle	76
Shakuhachi	77
Whistle	78
Ocarina	79
Lead1(square)	80
Lead2(sawtooth)	81
Lead3(calliope)	82

Lead4(chiff)	83
Lead5(charang)	84
Lead6(voice)	85
Lead7(fifths)	86
Lead8(bass+lead)	87
Pad1(newage)	88
Pad2(warm)	89
Pad3(polysynth)	90
Pad4(choir)	91
Pad5(bowed)	92
Pad6(metallic)	93
Pad7(halo)	94
Pad8(sweep)	95
FX1(rain)	96
FX2(soundtrack)	97
FX3(crystal)	98
FX4(atmosphere)	99
FX5(brightness)	100
FX6(goblins)	101
FX7(echoes)	102
FX8(sci-fi)	103
Sitar	104
Banjo	105
Shamisen	106
Koto	107
Kalimba	108
Bagpipe	109
Fiddle	110
Shanai	111
TinkleBell	112
Agogo	113
SteelDrums	114
Woodblock	115
TaikoDrum	116
MelodicTom	117
SynthDrum	118
ReverseCymbal	119
GuitarFretNoise	120
BreathNoise	121
Seashore	122
BirdTweet	123
TelephoneRing	124
Helicopter	125
Applause	126
Gunshot	127

INSTRUMENTOS DO CANAL 10 ->Percussão e Bateria

Código 0 = Standard Set
Código 8 = Room Set
Código 16 = Power Set
Código 32 = Jazz Set
Código 48 = Orchestral Set
Código 40 = Brush Set
Código 24 = PElectronic Set
Código 56 = SFX Set
Código 25 = TR-808 Set
Código 16 = Power Set

ANEXO 3 -

Tabela de notas, Frequência e Código Midi equivalente dentro da faixa de frequências audíveis pelo ser humano

Notas musicais	Frequências	Código MIDI
re#-1	19.4454 Hz	15
mi-1	20.6017 Hz	16
fa-1	21.8267 Hz	17
fa#-1	23.1246 Hz	18
sol1	24.4997 Hz	19
sol#-1	25.9565 Hz	20
la-1	27.50000 Hz	21
la#-1	29.1352 Hz	22
si-1	30.8677 Hz	23
do0	32,7032 Hz	24
do#0(reb0)	34,6478 Hz	25
re0	36,708 Hz	26
re#0(mib0)	38,8908 Hz	27
mi0	41,2034 Hz	28
fa0	43,6536 Hz	29
fa#0(solb0)	46,2494 Hz	30

sol0	48,9994 Hz	31
sol#0(lab0)	51,913 Hz	32
la0	55 Hz	33
la#0(sib0)	58,2704 Hz	34
si0	61,7354 Hz	35
do1	65,4064 Hz	36
do#1(reb1)	69,2956 Hz	37
re1	73,4162 Hz	38
re#1(mib1)	77,7818 Hz	39
mi1	82,4068 Hz	40
fa1	87,307 Hz	41
fa#1(solb1)	92,4986 Hz	42
sol1	97,9988 Hz	43
sol#1(lab1)	103,8262 Hz	44
la1	110 Hz	45
la#1(sib0)	116,541 Hz	46
si1	123,4708 Hz	47
do2	130,8128 Hz	48
do#2(reb2)	138,5914 Hz	49
re2	146,8324 Hz	50
re#2(mib2)	155,5634 Hz	51
mi2	164,8138 Hz	52
fa2	174,6142 Hz	53
fa#2(solb2)	184,9972 Hz	54
sol2	195,9978 Hz	55
sol#2(lab2)	207,6524 Hz	56
la2	220 Hz	57
la#2(sib0)	233,0818 Hz	58

si2	246,9416 Hz	59
do3 (dó central do piano)	261,6256 Hz	60
do#3(reb3)	277,1826 Hz	61
re3	293,6648 Hz	62
re#3(mib3)	311,127 Hz	63
mi3	329,6276 Hz	64
fa3	349,2282 Hz	65
fa#3(solb3)	369,9944 Hz	66
sol3	391,9954 Hz	67
sol#3(lab3)	415,3046 Hz	68
la3 (diapasão)	440 Hz	69
la#3(sib0)	466,1638 Hz	70
si3	493,8834 Hz	71
do4	523,2512 Hz	72
do#4(reb4)	554,3652 Hz	73
re4	587,3296 Hz	74
re#4(mib4)	622,254 Hz	75
mi4	659,2552 Hz	76
fa4	698,4564 Hz	77
fa#4(solb4)	739,9888 Hz	78
sol4	783,9908 Hz	79
sol#4(lab4)	830,6094 Hz	80
la4	880 Hz	81
la#4(sib0)	932,3276 Hz	82
si4	987,7666 Hz	83
do5	1046,5022 Hz	84
do#5(reb5)	1108,7306 Hz	85
re5	1174,659 Hz	86

re#5(mib5)	1244,508 Hz	87
mi5	1318,5102 Hz	88
fa5	1396,913 Hz	89
fa#5(solb5)	1479,9776 Hz	90
sol5	1567,9818 Hz	91
sol#5(lab5)	1661,2188 Hz	92
la5	1760 Hz	93
la#5(sib0)	1864,655 Hz	94
si5	1975,5332 Hz	95
do6	2093,0046 Hz	96
do#6(reb6)	2217,461 Hz	97
re6	2349,3182 Hz	98
re#6(mib6)	2489,0158 Hz	99
mi6	2637,0204 Hz	100
fa6	2793,8258 Hz	101
fa#6(solb6)	2959,9554 Hz	102
sol6	3135,9634 Hz	103
sol#6(lab6)	3322,4376 Hz	104
la6	3520 Hz	105
la#6(sib0)	3729,31 Hz	106
si6	3951,0664 Hz	107
do7	4186,009 Hz	108
do#7(reb7)	4434,922 Hz	109
re7	4698,6362 Hz	110
re#7(mib7)	4978,0318 Hz	111
mi7	41,2034 Hz	112
fa7	5587,6518 Hz	113
fa#7(solb7)	5919,9108 Hz	114

sol7	6271,927 Hz	115
sol#7(lab7)	6644,8752 Hz	116
la7	7040 Hz	117
la#7(sib0)	7458,6202 Hz	118
si7	7902,1328 Hz	119
do8	8372,018 Hz	120
do#8(reb8)	8869,8442 Hz	121
re8	9397,2726 Hz	122
re#8(mib8)	9956,0634 Hz	123
mi8	10548,0818 Hz	124
fa8	11175,3034 Hz	125
fa#8(solb8)	11839,8216 Hz	126
sol8	12543,854 Hz	127
sol#8(lab8)	13289,7504 Hz	
la8	14080 Hz	
la#8(sib0)	14917,2404 Hz	
si8	15804,2656 Hz	
do9	16744,0362 Hz	
do#9(reb9)	17739,6884 Hz	
re9	18794,5452 Hz	
re#9(mib9)	19912,127 Hz	
mi9	21096,1636 Hz	
fa9	22350,6068 Hz	

ANEXO 4 -

Linguagem funcional CLEAN

Este capítulo visa justificar o paradigma escolhido para o desenvolvimento e implementação deste trabalho de dissertação, fornecedor de um subsidio conceitual e prático da utilização da linguagem de programação escolhida, CLEAN, de forma que este texto atenda não somente a leitores com bases sólidas em programação, mas, também, aos músicos que posteriormente venham buscar neste trabalho uma fonte de informações que os permitam desenvolver aplicativos para manipulação e análise de ações no domínio musical.

Alguns tópicos são apresentados com mais detalhes devido à carência de documentação sobre a linguagem CLEAN, por ser a mesma ainda relativamente nova.

A escolha do paradigma e da linguagem de programação

A escolha correta do melhor paradigma e respectiva linguagem de programação para se implementar uma classe de problemas é fundamental para se atingir os resultados esperados, bem como para facilitar, tornar mais aderente, a modelagem do problema e tratamento dos erros que forem surgindo durante o projeto.

Conforme citado na introdução, a matemática e a música sempre andaram juntas destes os primórdios da humanidade. Isto se deu e se dá devido à música possuir técnicas e raciocínio extremamente matemáticos e lógicos, inclusive pela mesma trabalhar com um sistema de grafia (notação musical) em base binária.

Desta forma, a escolha de um paradigma que trabalhe bem com funções matemáticas e que facilite implementações afins, é o ideal para se trabalhar e gerar ferramentas também no domínio musical.

Além de se escolher um paradigma adequado, deve-se também escolher uma linguagem que seja a mais aderente possível aos fenômenos que se deseja modelar.

A escolha natural, feitas tais análises, se dá pelo paradigma funcional, o qual possui várias vantagens em relação ao paradigma procedimental, descritas com detalhes e simplicidade por RUFINO e LIMA [29].

Até uma década atrás, a linguagem funcional LISP [30] foi a linguagem mais utilizada em computação musical, principalmente na construção de sistemas especialistas e sistemas inteligentes. Atualmente existem linguagens funcionais mais poderosas e mais fiéis ao paradigma matemático, podendo-se citar, entre elas: Haskell e Clean.

Dentre estas duas linguagens, fortemente tipadas, sem problemas de transparência referencial, avaliação destrutiva e efeitos colaterais, optou-se por utilizar a linguagem Clean [31] por possuir, também, o atrativo de se poder implementar interfaces gráficas de uma forma simples e por rodar em diversos sistemas operacionais sem que seja necessário a instalação de dlls.

A seguir, apresenta-se alguns conceitos relevantes para que se possa entender a escolha do referido paradigma, bem como princípios básicos de utilização da linguagem de programação CLEAN, suas potencialidades e aderência ao paradigma matemático.

Por que mais uma linguagem?

Fica sempre a seguinte pergunta no ar:

- Com uma linguagem apenas um bom programador não seria capaz de resolver todos os problemas de modelamento e implementação computacionais?

A resposta é: praticamente sim!

Uma pergunta conseqüente seria:

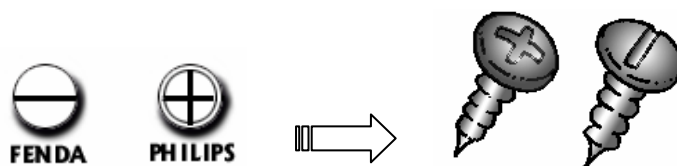
- Para que então tantos tipos de linguagens?

Pode-se responder esta pergunta utilizando uma analogia muito simples e aderente ao conceito. Já que as linguagens de programação, tais como CLEAN, são ferramentas computacionais, associe-as ao conceito e à utilização das ferramentas que normalmente são utilizadas no dia a dia.

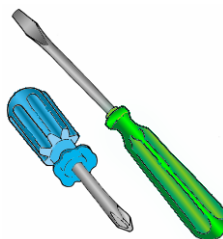
Suponha que tenha que realizar uma tarefa bem simples:

Apertar um parafuso

Suponha que possua um parafuso do tipo Philips, ou seja, em vez de ter um chanfro em linha reta em sua cabeça ele tem um chanfro em X (ou em cruz se preferir).



Ao utilizar tal parafuso, pode-se tentar apertá-lo com uma chave de fenda comum ou pode-se fazer a mesma tarefa com uma chave própria (a chave Philips).



A chave Philips foi desenvolvida para apertar parafusos Philips. Assim, a tarefa será mais simples, mais rápida e adequada aos propósitos propostos quando utiliza-se a ferramenta certa (o parafuso Philips permite um aperto melhor).

Aproveite este exemplo e se questione:

- **Será possível apertar um parafuso Philips utilizando uma chave de fenda comum?**



A resposta é sim, apesar de não conseguir obter um aperto tão firme quanto o que seria efetuado com a chave Philips.



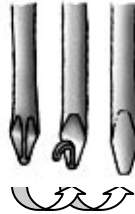
- **E se for apertar um parafuso comum, tipo fenda, será que conseguiria apertá-lo com uma chave tipo Philips?**



A resposta a princípio seria não.



Diz-se a princípio, devido ao fato de que, se fizer algumas modificações na chave Philips, tal como limar a cabeça da mesma até que ela fique chata, tal como uma chave de fenda comum, seria possível realizar a tarefa proposta.



O que ocorre é que tal modificação radical descaracterizará por completa a ferramenta utilizada, chegando a tal ponto de não permitir que a mesma realize adequadamente as tarefas que anteriormente fazia, ou seja, apertar parafusos Philips com mais força, mais torque.

Ao mesmo tempo, a modificação desta ferramenta demandará do usuário uma destreza e um conhecimento especializado que o permita modificar a ferramenta adequadamente.

Assim, apertar um parafuso comum fica mais fácil e adequado utilizando uma chave de fenda comum, sendo uma tarefa difícil apertá-lo com uma chave de fenda Philips.

Desta forma, para cada tarefa do dia a dia tem-se vários tamanhos e tipos de chaves de fenda, alicates, etc. Cada ferramenta atendendo tipos de tarefas específicas, **paradigmas** diferentes.

O mesmo acontece com as linguagens de programação. Existem tarefas computacionais que são mais facilmente resolvidas se forem utilizadas as ferramentas computacionais adequadas, ou seja, com **paradigmas aderentes** à aplicação.

Paradigma

Cada tarefa pode ser implementada e solucionada de várias formas diferentes. A forma com que se soluciona um determinado problema é chamada de **paradigma**.

Como exemplo, antes de Santos Dumont, os relógios eram sempre carregados no bolso e presos a uma correntinha. Esta solução parecia perfeita, ou seja, atendia todas as expectativas dos usuários.

- Então, para que modificá-la ?



Esta pergunta nem chegou a ser questionada, já que era perfeita. Eis que um belo dia, Santos Dumont resolveu inventar sua máquina voadora, a qual exigia que suas mãos estivessem firmemente fixas aos controles delas, já que se as retirassem poderiam ocorrer erros que possivelmente acarretariam a queda das mesmas. Como poderia, portanto ele, o piloto, olhar as horas para, por exemplo, verificar o tempo de combustível, retirando o relógio do bolso, já que o painel ainda não havia sido inventado (um novo paradigma)?



Neste momento surgiu a necessidade de se modificar o paradigma de como utilizar e carregar um relógio. Assim, Santos Dumont implementou uma nova solução, um novo paradigma, que deu origem aos relógios de pulso. Este é um exemplo clássico de mudança de paradigma. A mudança dos relógios movidos à corda e os movidos à pilha (bateria) é um outro exemplo de quebra de paradigma. É totalmente diferente trabalhar, consertar ou apenas utilizar um relógio com corda e um relógio com pilha (bateria). O surgimento dos relógios digitais também foi uma nova mudança de paradigma, a princípio desacreditada pelos Suíços, mas popularizada com sucesso pelos japoneses.



A chave Philips e a chave Allen são, também, exemplos de mudanças de paradigma.



As linguagens de programação também possuem paradigmas de programação diferentes. Assim, pode-se citar quatro paradigmas muito conhecidos, a saber:

- 1- O paradigma imperativo, procedural ou procedimental.
- 2- O paradigma lógico
- 3- O paradigma funcional
- 4- O paradigma de Orientação a Objeto

As linguagens desenvolvidas em cada um destes paradigmas poderão realizar basicamente as mesmas tarefas, só que de formas totalmente diversas. Assim, algumas tarefas, alguns aplicativos, serão mais facilmente implementadas em uma linguagem do que em outra.

Como escolher o paradigma mais adequado a um determinado projeto?

A resposta é simples: - escolha o paradigma mais **aderente** à sua aplicação. Para isto, deve-se conhecer os conceitos, particularidades, potencialidades e restrições de cada um.

Aderência

Um outro ponto relevante, que deve ser destacado, é que nem sempre um paradigma que possua todas as ferramentas necessárias à efetivação da tarefa é aderente ao usuário.

Entende-se por aderência a forma com que se apresenta uma determinada tarefa ou ferramenta ao programador ou a um usuário final do produto.

Observe o seguinte questionamento e exemplo:

- Suponha que seja desejado apresentar o resultado de um programa musical a um músico. Qual seria a melhor maneira de apresentar tais resultados?

Pode-se refazer esta pergunta da seguinte forma:

- Qual seria a forma mais aderente de apresentar uma música gerada por um programa de computador, por um aplicativo, para um músico?
- Por outro lado, qual seria a forma mais aderente para apresentar o resultado a uma pessoa que não domine a sintaxe musical?

A resposta é que para um músico a forma de apresentação mais aderente de uma música ao mesmo seria uma partitura convencional do tipo:



Se um profissional não for um músico, fatalmente não entenderá nada do que este gráfico, este desenho da partitura, está querendo informar. Muito menos poderá aquilatar se existem ou não erros nesta apresentação. Neste caso, pode-se dizer que a partitura musical não é uma forma aderente de apresentar os resultados do programa para uma pessoa que não seja músico.

Desta forma deve-se criar um outro tipo de notação que possa apresentar os resultados de um programa musical de forma mais aderente para um usuário não músico. Por exemplo:

Compasso = [(Ré5,c), (Si5,c), (G#5,Sm), (Dó6, Smp),(Si5, c)]

Um usuário com conhecimentos de computação ou matemática percebe rapidamente que a informação é uma lista com 5 tuplas de dois elementos cada, cujo nome é Compasso. Se for informado ao mesmo que o primeiro elemento de cada tupla é uma nota musical e que o segundo é o tempo, a duração desta nota, a informação será facilmente absorvida por ele. Por outro lado, se tentássemos explicar, ensinar ao mesmo como ler a simbologia da notação musical, esta seria uma tarefa bem mais complicada e complexa.

Assim, aderência é tornar sua ferramenta ou os resultados de seu programa fáceis de serem lidos ou utilizados pelo usuário alvo de seu sistema.

Aderência não tem nada a ver com o grau de complexidade ou de recursos de um sistema e sim, com a forma mais simples de apresentar o que se queira a um determinado usuário, de forma que o mesmo não tenha a mínima dificuldade em utilizar os conhecimentos e ferramentas de seu sistema.

Para melhor fixar o que foi afirmado, veja como uma linguagem com paradigma funcional, no caso CLEAN, é aderente ao paradigma matemático:

Para tanto, observe a definição de um número de Fibonacci:

Fibonacci (n) = n, se $n \leq 1$

Fibonacci (n) = Fibonacci (n-2) + Fibonacci (n-1), se $n > 1$

A implementação dessa função em paradigma funcional, linguagem CLEAN, é apresentada a seguir:

Fibonacci n | $n \leq 1 = n$

Fibonacci n | $n > 1 = \text{Fibonacci (n-2)} + \text{Fibonacci (n-1)}$

Como se pode verificar, o programa escrito em CLEAN é praticamente a própria definição matemática do problema proposto, ou seja, CLEAN é uma linguagem em paradigma funcional, extremamente aderente ao paradigma matemático.

Se fosse utilizado o paradigma procedural ou imperativo, Linguagem C, o mesmo programa ficaria da seguinte forma:

Sequência de Fibonacci escrita em C.

```
#include <stdio.h>
#include <stdlib.h>

int fib(int n){
    int x,lofib,hifib,i;

    if (n <= 1) return(n);
    lofib = 0;
    hifib = 1;
    for (i = 2;i <= n; i++){
        x = lofib;
        lofib = hifib;
        hifib = x + lofib;}
    return (hifib);}

void main(){
    int fibo;
    char string[25];

    fibo = fib(45);
    itoa(fibo, string, 15);
    printf("fib(45)=%d \n",fibo,string);
    return;}
```


Pode-se perceber que a implementação procedural da série de Fibonacci muito se difere da definição matemática da mesma. Se você não for um bom programador em Linguagem C, fatalmente não entenderia o que tal programa faz. Neste caso, diz-se que a linguagem procedural, no caso C, não é aderente ao paradigma matemático. Isto não significa, entretanto, que não se pode implementar funções matemáticas em C, nem que tal paradigma não seja eficiente nestes casos. O que se pode afirmar é que o mesmo não é aderente, ou seja, não possui a clareza desejada na implementação e leitura de programas que realizem tarefas do paradigma matemático.

Erro de tipo

Algumas linguagens podem acarretar efeitos colaterais indesejáveis e imprevisíveis nos programas implementados. As linguagens procedurais são aquelas que mais efeitos colaterais produzem. CLEAN foi criado de tal forma a procurar evitá-los.

A definição dos tipos de dados numa linguagem de programação é algo importante, os mesmos evitam em parte o que denominamos de efeitos colaterais, os quais são comuns em sistemas gerados por linguagens com paradigma procedural ou imperativo. Para que se entenda melhor o conceito e a necessidade da tipagem de dados, veja a seguinte analogia, dada por alguns questionamentos:

-Quanto você acha que é dois mais dois?

A resposta é simples, ou seja: quatro.



-E quanto é um tijolo mais dois tijolos?

A resposta também é simples: três tijolos.



-E quanto é uma banana mais um abacaxi?



Agora o problema ficou um pouco mais complicado de ser solucionado, ou seja:

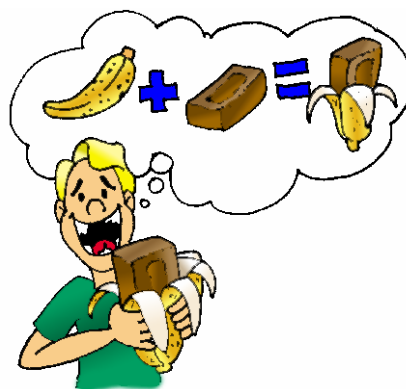
Como somar banana com abacaxi?

Algumas pessoas poderiam tentar uma resposta dizendo que o resultado é uma salada de frutas. Mesmo se fosse, onde esta resposta seria armazenada, já que não se tem inicialmente lugar para guardar o tipo salada de frutas? Apenas se possui um lugar para guardar banana e outro para abacaxi. Assim, o que seria mais correto: guardar a salada de frutas no local das bananas ou no dos abacaxis?



Em qualquer um dos lugares que for armazenado o resultado, ele não teria nenhum significado, estaria fora do contexto.

-Bom, e quanto seria uma banana mais um tijolo?



Agora realmente o problema perde todo o significado, nem a salada de frutas serve como resposta.

Neste momento, sucinta o seguinte questionamento:

- O que somar tijolo com banana tem a ver com linguagens tipadas?

Serve para alertar que não se pode somar tijolo com banana. O correto, quando for solicitado tal absurdo, seria informar à pessoa que pediu que se somasse tijolo com banana que a mesma cometeu um erro de tipo e que infelizmente não se pode efetuar a operação desejada sem causar problemas imprevisíveis em futuras operações.

Misturar bananas com tijolos é semelhante, em computação a se misturar inteiros com reais, operar tipos de dados diferentes.

Infelizmente, as escolas não se preocupam em manipular corretamente estes tipos de dados, e, assim, comete-se o absurdo de realizar a seguinte soma:

$$2 + 3,5 = 5,5$$

Isto pode parecer, a princípio, estar correto. O que ocorre é que esta é uma operação envolvendo elementos de dois conjuntos distintos, onde, no conjunto dos inteiros não existem elementos reais e vice versa. Realizando tal soma estar-se-ia ferindo no mínimo a propriedade de fechamento dos conjuntos, ou seja, em um conjunto, ao somarmos dois de seus elementos sempre teremos como resultado um elemento deste conjunto.

Na computação isto se agrava. Primeiro porque os conjuntos não são infinitos, existe uma quantidade máxima de elementos que se pode armazenar na memória de um computador. Por outro lado, a forma de se armazenar os inteiros é completamente diferente da forma de se armazenar os reais. Desta forma, por absurdo, se fosse possível somar inteiros com reais, como o resultado seria armazenado: como real ou como inteiro?

A linguagem Funcional CLEAN, assim como as linguagens tipadas modernas, não permite que, mesmo por distração, o programador opere com tipos de dados diferentes.

Observando a soma $2 + 3,5$, fica o questionamento: o que será que a pessoa que propôs esta conta queria fazer? E se a mesma tivesse digitado por engano a vírgula e o resultado fosse a quantidade de adrenalina a ser injetada em um paciente com parada cardíaca?

Além de não permitir tais erros de tipo, CLEAN [32] é completamente aderente ao paradigma matemático. Desta forma, o mesmo não permite absurdos cometidos por linguagens procedurais, tal como a avaliação destrutiva de variáveis.

Ex: $x = x + 1$

Entender como tais erros ocorrem exigiria do leitor um conhecimento mais profundo de programação, o que não é o interesse no momento, mas é importante saber que os mesmos existem e tentar evitar que eles ocorram quando utilizarmos linguagens ou aplicativos que utilizam o paradigma procedural, ou, uma solução melhor, escolher uma linguagem, um paradigma, que por si só não permita que tais fatos ocorram, assim como o funcional.

A Linguagem CLEAN

Um dos pontos que torna a Linguagem Funcional CLEAN atrativa a desenvolvimentos no domínio musical está na facilidade de se implementar qualquer sistema computacional com a mesma. CLEAN possui uma ferramenta matemática que permite que se defina o conjunto imagem, o conjunto de todas as possíveis soluções para um problema, de uma forma simples, formal e que apenas exige do programador ter o conhecimento do domínio de onde pertencem seus dados e das regras que devem ser aplicadas aos mesmos para gerar o conjunto solução (Imagem). Esta ferramenta matemática é denominada por Notação Zermelo-Frankel [29] (Compreensão de listas e vetores).

Tipos de dados:

• Caractere

- Um **caractere** é um símbolo de um alfabeto.
- O alfabeto utilizado pelo computador é o ASCII:

ASCII -> American Standard Code for Information Interchange [ANEXO 1]

- No CLEAN, um caractere é representado por um símbolo colocado entre apóstrofos.

Exemplos: 'a' = caractere letra **a** minúscula, '1' = caractere dígito **1**.

- Não se pode confundir o **caractere '2'** com o **inteiro 2**, um é um símbolo e o outro um valor a ser manipulado, e assim por diante.
- Na tabela com este alfabeto, os caracteres que representam os dígitos iniciam com o código 48. As letras maiúsculas começam com o código.

Assim:

- Dígitos => '0' = 48 , '1' = 49, ... '9' = 57.
 - Letras Maiúsculas => 'A' = 65, 'B' = 66, ... 'Z' = 90.
 - Letras Minúsculas => 'a' = 97, 'b' = 98, ... 'z' = 122.
- O tipo de dado CARACTERE é referenciado no CLEAN como: **Char**

• String

- Uma String é uma cadeia de caracteres colocados entre aspas. Exemplo: "Casa", "casa", "1234". Onde "Casa" e "casa" são Strings diferentes devido a ambas começarem com caracteres ASCII diferentes. O caractere 'c' possui um código igual a 99 e o caractere 'C' igual a 67 [ANEXO 1].
- Uma String é uma cadeia (vetor) de caracteres que devem ser "lidos" de uma só vez, e não caractere por caractere.

- O tipo de uma String é : { **#Char** }
- Um vetor de caracteres são elementos de mesmo tipo de dados entre chaves e separados por vírgula. Exemplo: {**1,2,3**} = **vetor de inteiros** ou {**'a','r'**} = **vetor de caracteres**.
- Uma String é um vetor de caracteres que só possui sentido se lido de uma vez só.
- Em CLEAN, a representação de uma cadeia de caracteres pode ser feita de duas formas:

1. **Estrita = String = {#Char}**

2. **Normal = {Char}**

Veja a diferença:

Estrita (String):

Start :: {#Char}

Start = {'a','g','o','r','a'}

Devolve: “agora”

Normal:

Start :: {Char}

Start = {'a','g','o','r','a'}

Devolve: {'a','g','o','r','a'}

Ou seja: Ao declararmos o tipo de dados de saída da função **Start** como String (um vetor de caracteres estrito), ao executarmos o vetor de caracteres, o CLEAN devolveu uma String. No caso de declararmos como um vetor de caracteres, o mesmo devolveu um vetor de caracteres. Pode parecer a princípio ser a mesma coisa, mas, na realidade, são “entidades” diferentes.

Obs. Você pode acrescentar caracteres em uma string, mas, em uma cadeia (vetor) de caracteres não estritos não.

Exemplo:

Obs. +++ é o operador de concatenação de duas cadeias estritas de caracteres (**Strings**).

O mesmo não se aplica a uma cadeia (vetor) de caracteres não estrita.

1- operação com duas cadeias de caracteres não estritas.

funde :: {Char} {Char} -> {Char}

funde x y = x +++ y

Start = funde {'a','g','o','r','a'} {' ','s','o','u','.'}

Devolve: Uma mensagem de erro dizendo não ser possível realizar a operação.

2- operação com duas cadeias de caracteres estritas (String).

funde :: {#Char} {#Char} -> {#Char}

funde x y = x +++ y

Start = funde {'a','g','o','r','a'} {' ','s','o','u','.'}

Devolve: “agora sou.”

ou

funde x y = x +++ y

Start = funde "agora" " sou."

Devolve: “agora sou.”

Obs. Isto mostra que os dois tipos de dados {#Char} e {Char} são diferentes, apesar de representarem uma cadeia de caracteres.

Para normalizar a nomenclatura, diz-se que:

- Uma cadeia de caracteres estrita (String) é um vetor de caracteres eager.
- Uma cadeia de caracteres não estrita é um vetor de caracteres lazy.

Assim:

- Em um vetor eager, todos os elementos do mesmo são acessados, lidos, de uma vez só.
- Em um vetor lazy, cada elemento do mesmo pode ser acessado, lido, separadamente.

- **Vetores**

- Um vetor possui elementos de mesmo tipo de dados delimitados por chaves e separados por vírgula. Exemplo: **{1,2,3}** = **vetor de inteiros** ou **{‘a’,’r’}**= **vetor de caracteres**.
- A declaração de um vetor é feita colocando-se o tipo do dado entre chaves. Por exemplo: **{1,2,3}** é do tipo: **{Int}** , **{ “casa”, “porta” }** é do tipo: **{ {#Char} }**, **{2.4}** é do tipo **{Real}**, **{ {1,2}, {4,5,6}, {3} }** é do tipo: **{ {Int} }**.
- Já foi visto anteriormente um caso particular de vetores: a cadeia de caracteres estrita: **String**.
- Um vetor possui número fixo de elementos, com a exceção da String.

- **Funções de Manipulação de Vetores**

- **Pegando um elemento de um vetor**

Praticamente, para os propósitos iniciais, esta é a função que mais será utilizada, e, portanto, somente ela será apresentada.

Para pegar um elemento do vetor, basta acrescentar no fim do mesmo o sinal de ponto seguido de um valor inteiro entre colchetes. Este valor fará com que o elemento do vetor cujo índice for o inteiro colocado entre colchetes seja retornado.

Exemplo:

Start = {‘a’,’r’,’o’}.[2]

Devolve: ‘o’, o qual é o caractere de índice 2. Lembre-se que o primeiro elemento possui índice 0.

Start = “aro”.[2]

Devolve: ‘o’, o que já era de se esperar, já que uma String é um vetor.

• Matrizes

- Uma matriz é um vetor de vetores com elementos de mesmo tipo de dados.
- Cada elemento de uma matriz é um vetor.
- Uma matriz possui vetores como elementos, separados por vírgula e delimitados por uma chave. Exemplo: `{{1, 34}, {22, 33}, {4}}`. O tipo de dado desta matriz exemplo, é declarado da seguinte forma: `{{Int}}`

• Funções de Manipulação de Matrizes

Novamente só vamos abordar neste documento como pegar um elemento de uma matriz

- Pegando um elemento de uma matriz

O procedimento é semelhante ao utilizado para vetores, com a diferença que entre colchetes ficarão dois inteiros: o primeiro deles indica qual é o vetor da matriz que está sendo acessado, e, o segundo valor inteiro indica qual elemento deste vetor será devolvido.

Exemplo:

```
matriz :: {{Int}}
```

```
matriz = { {1,2},{3},{5,6,7} }
```

```
Start = matriz.[2,0]
```

Devolve: 5, ou seja, o primeiro elemento do vetor de índice 2 (o terceiro vetor)

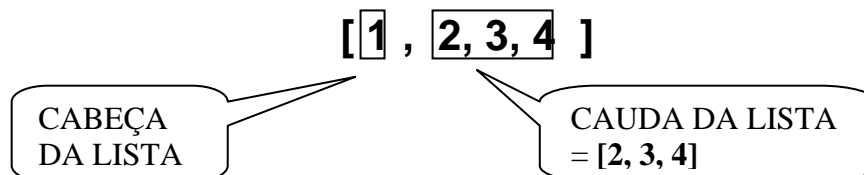
Observe que para manipular uma matriz, deve-se declarar o tipo dela. Para tanto, teve-se de criá-la antes de manipulá-la.

• Listas

- Uma lista possui elementos de mesmo tipo de dados delimitados por colchetes e separados por vírgula. Exemplo: **[1,2,3]** = **lista de inteiros** ou **['a', 'r']** = **lista de caracteres**.
- A declaração de tipo de uma lista é dada pelo tipo de seus elementos colocado entre colchetes. Exemplo:
 1. a lista **[1,2,3]** possui o tipo **[Int]**,
 2. a lista **['a', 'r']** possui o tipo **[Char]**,
 3. a lista **["casa", "eu"]** possui o tipo **[{#Char}]**,
 4. a lista **[[1,2],[3],[44,5]]** possui o tipo **[[Int]]**.
- Uma lista possui um número variado de elementos, ou seja, você pode eliminar um elemento de uma lista ou mesmo acrescentar.
- Uma Lista em CLEAN possui a seguinte estrutura: **[c:r]**, onde **c** é a cabeça da lista (o primeiro elemento dela) e **r** o resto da lista (sua calda, ou seja, a lista sem a cabeça).
- **Lista vazia**->Uma lista vazia é representada por **[]**, ou seja, o abrir e fechar de colchetes com tantos espaços em branco internamente quanto se deseje.

• Exemplos de listas finitas:

- lista de inteiros: **[1, 2, 3, 4]**



Comparando a lista **[1, 2,3,4]** com **[c:r]** temos que: **c = 1** e **r = [2,3,4]**

- lista de reais: **[1.0, 2.0, 3.0, 4.0]** onde **c = 1.0** e **r = [2.0, 3.0, 4.0]**
- lista de strings: **["a","b","c"]** onde **c = "a"** e **r = ["b","c"]**
- lista de caracteres: **['a','b','c']** onde **c = 'a'** e **r = ['b','c']**

- lista de listas: [['a','b'], ['d','e']] onde $c = ['a','b']$ e $r = [['d','e']]$

CABEÇA DA
LISTA = ['a','b']

CAUDA DA LISTA =
[['d','e']]

• Tuplas

Uma tupla é uma coleção de elementos separados por vírgula. Tal coleção é delimitada por parênteses. Alguns exemplos de tuplas incluem:

- ('a',1, "casa") – uma tupla formada por elementos de três diferentes tipos de dados.
- ([1, 2, 3], [4, 5, 6]) – uma tupla formada por duas listas.

Algumas vezes você poderá sentir necessidade de agrupar informações de diferentes tipos. Por exemplo, podemos pensar em abstrair uma pessoa pelo seu nome e sua identificação. O nome pode ser representado por um valor do tipo String enquanto que a identificação pode ser representada por um inteiro. A representação de uma pessoa por uma lista não seria possível pois a lista deveria ter dois elementos de tipos diferentes (nome e identificação) e isto violaria a propriedade de que todos os elementos da lista devem ter um mesmo tipo. O uso de uma tupla formada por dois elementos de tipos diferentes (nome e identificação) pode ser usado neste caso.

As tuplas, como mostram os exemplos descritos anteriormente, podem ter elementos que apresentam diferentes tipos de dados. Podemos ter tuplas formadas por listas assim como listas formadas por tuplas. Os exemplos que seguem mostram listas cujos elementos correspondem a tuplas.

• Exemplos de listas de tuplas:

- **Lista de tuplas de dois elementos inteiros:**

$[(1, 2), (3, 4), (5, 6)]$

Neste caso se $[c:r]$ é a lista $[(1, 2), (3, 4), (5, 6)]$, então:

$c = (1, 2)$ e $r = [(3, 4), (5, 6)]$

- Lista de tuplas de três elementos inteiros:

$[(1, 2, 3), (4, 5, 6)]$

Neste caso se $[c:r]$ é a lista $[(1, 2, 3), (4, 5, 6)]$, então:

$c = (1, 2, 3)$ e $r = [(4, 5, 6)]$

- Lista de tuplas de tuplas de três elementos inteiros:

$[((1,2,3),(4,5,6)),((7,8,9),(10,11,12))]$

Neste caso se $[c:r]$ é a lista $[((1,2,3),(4,5,6)),((7,8,9),(10,11,12))]$ então:

$c = ((1,2,3),(4,5,6))$ e $r = [((7,8,9),(10,11,12))]$

A linguagem CLEAN possui algumas funções básicas que permitem a manipulação deste tipo de dado. Se estas não forem suficientes você poderá criar as suas próprias funções.

• Declaração de tipo

Quando for declarar o tipo de uma função cujos argumentos utilizem o tipo tupla, proceda da seguinte forma:

Exemplo 1: tuplas de inteiros

A seguir tem-se a definição do tipo de uma função f cujo argumento de entrada é uma tupla de dois elementos inteiros e devolve como resposta uma tupla de dois elementos inteiros.

$f :: (\text{Int}, \text{Int}) \rightarrow (\text{Int}, \text{Int})$

Exemplo 2: tuplas de inteiros e reais

A função f cujo tipo é definido a seguir apresenta como argumento de entrada uma tupla de dois elementos, sendo um inteiro e o outro um real. O valor resultante do cálculo desta função é um valor do tipo **Int**.

$f :: (\text{Int}, \text{Real}) \rightarrow \text{Int}$

Exemplo 3: tuplas de inteiros, reais e caracteres

Neste exemplo tem-se a definição do tipo de uma função f cujo argumento de entrada é uma tupla de três elementos, sendo um inteiro, o outro um real e o último um caractere. A saída é uma lista de inteiros.

$f :: (\text{Int}, \text{Real}, \text{Char}) \rightarrow [\text{Int}]$

Pelos exemplos descritos observa-se que os tipos das tuplas são especificados pela enumeração dos tipos dos elementos que a constituem delimitados por parênteses. Assim temos que:

- $(\text{"Carlos"}, 2312) :: (\text{String}, \text{Int})$ (leia como: a tupla $(\text{"Carlos"}, 2312)$ é de tipo $(\text{String}, \text{Int})$).
- $(\text{'a'}, \text{False}, 2) :: (\text{Char}, \text{Bool}, \text{Int})$ (leia como: a tupla $(\text{'a'}, \text{False}, 2)$ é de tipo $(\text{Char}, \text{Bool}, \text{Int})$).

• Notação Zermelo-Frankel

Este trabalho de dissertação trabalha com conjuntos de notas musicais e regras de pertinências das mesmas a estes conjuntos. A este conjunto de notas são aplicadas várias regras de inferência para formar os conjuntos soluções para que se possa transpor uma música para tonalidades, as quais possuem suas regras específicas. Este tipo de problema é simples de ser modelado e implementado utilizando a notação Zermelo-Frankel, disponibilizada na linguagem CLEAN com completa aderência e fidelidade matemática. Desta forma, é interessante que sejam aqui apresentados os detalhes e conceitos de implementação e utilização desta poderosa ferramenta matemática.

Esta notação, proposta por Ernest Zermelo, um matemático alemão, e por Adolf Frankel, um lógico Israelita, é um recurso, uma ferramenta matemática poderosa, veloz e eficiente na arte de computar valores de uma função [29]. A notação Zermelo-Frankel

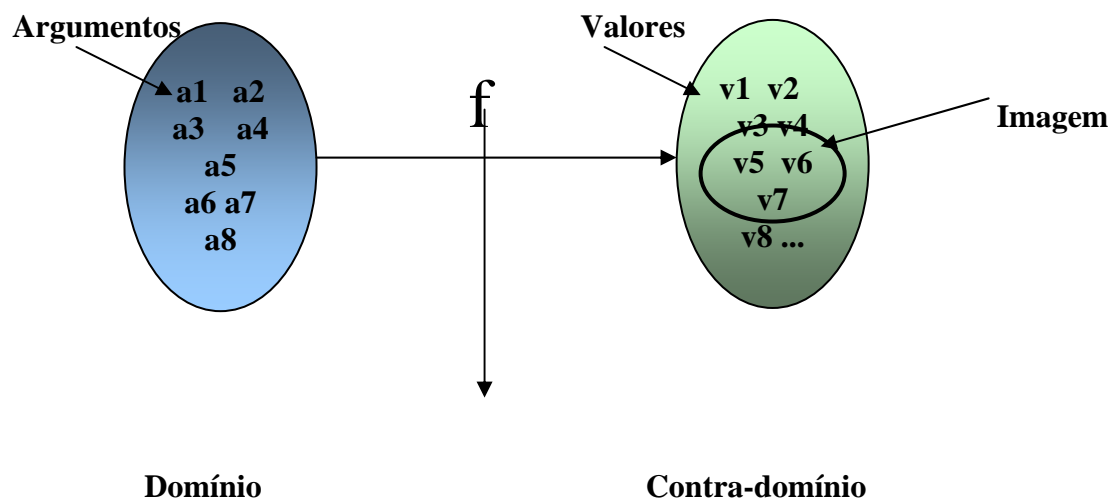
formaliza, explicita a lei de formação de um conjunto imagem, dado o domínio da função e as regras de inferência da mesma. Para entendê-la melhor, deve-se ater a alguns conceitos fundamentais.

Computando o valor de uma função

No começo do século, matemáticos e filósofos se preocupavam com métodos que, a partir do argumento, pudessem calcular o valor de uma função. Assim, dado um elemento do domínio, eles buscavam descobrir, desenvolver um processo mecânico para se avaliar, encontrar o correspondente elemento do contradomínio. Este processo mecânico deveria ser constituído por uma sucessão de aplicações de regras de inferências, as quais determinariam o valor da função.

Os filósofos da matemática deram, a este processo, o nome de **computação do valor da função**. Foi Alonzo Church (Church, 1941) quem propôs o primeiro sistema formal de computação.

Observe a figura, a seguir:



Regras de Inferência

- Onde: argumento é um elemento do domínio e valor um elemento do conjunto imagem.

Conceitos Fundamentais

1. Computar o valor de uma função é aplicar a função e suas regras de inferência a um elemento do domínio (denominado de argumento da função). Esta aplicação determina, infere, qual elemento do contradomínio é o valor (elemento do contradomínio) resultante desta aplicação.
2. Os elementos de um domínio devem possuir a mesma lei de formação, o mesmo tipo de dado.
3. O conjunto imagem é o conjunto de todos os valores computados por uma determinada função quando a mesma é aplicada a todos os elementos (argumentos) do domínio.
4. Os valores do conjunto imagem e do contradomínio devem possuir a mesma lei de formação, o mesmo tipo de dado.
5. O conjunto imagem, portanto, é formado pelos valores do contradomínio identificados pela aplicação da(s) regra(s) de inferência da função em todos os argumentos do domínio.
6. Assim, o conjunto imagem possuirá menos elementos que o contradomínio, ou, no máximo, a mesma quantidade. O conjunto imagem só terá menos elementos que o domínio se existir restrições aos elementos do domínio (veremos a seguir como isto é feito).

Declarando o tipo de uma função (domínio e imagem)

É fundamental para tornar um programa mais legível e modular que se possa explicitar a declaração do tipo de dado de cada função. Matematicamente isto é similar a se declarar o domínio e a imagem da mesma.

Em CLEAN, fazer isto é uma tarefa simples, bastando ao programador digitar o **nome_da_função** seguido dos sinais **::**, feito isto, após tais sinais deve-se colocar o tipo de dado de cada argumento da função (domínio) seguido dos sinais **->** seguido do tipo de dado do valor (resultado) da função (imagem).

Nome_da_função::tipo_de_dados_do_domínio->tipo_de_dados_do_contraDomínio

Observações Gerais da Notação

Zermelo, e depois finalizado por Frankel, definiram, formalizaram o conjunto imagem da seguinte forma:

Domínio: $\{x \in \mathbb{N}\}$

Função : $f(x) = x^3$

Notação Matemática da formalização do conjunto imagem:

$$S = \{ x^3 \mid x \in \mathbb{N} \}$$

Implementação em CLEAN:

$$S = \{ x^3 \parallel x \leftarrow [0..] \}$$

tal que pertence (naturais = intervalo de 0 a infinito)

Obs. O símbolo $^$ é o operador de potenciação utilizado pelo CLEAN.

Para exemplificar esta notação, observe o seguinte exemplo:

- Mostrar o conjunto dos quadrados dos inteiros entre 1 e 5.

Da matemática tem-se : $S = \{ x^2 \mid 1 \leq x \leq 5 \}$

Em CLEAN tem-se : $S = [x^2 \parallel x \leftarrow [1..5]]$

Utilizando a Notação Zermelo-Frankel

Para tornar mais clara a implementação desta notação, segue-se um exemplo comentado:

Exemplo:

Implementar a Notação Zermelo-Frankel em CLEAN de tal forma que a mesma gere o conjunto imagem formado pela lista de todos os números naturais pares de 0 a 9.

Domínio: $\{x \in \mathbf{N} \mid 0 \leq x \leq 9 \wedge x \text{ é par} \}$

ou seja-> **{2,4,6,8}**

Função: $f(x) = x$

ou regra -> o valor de x será igual ao do argumento, sem qualquer modificação.

A pergunta que deveria ser suscitada agora é:

- Como montar corretamente a notação para que tal lista seja gerada?



Observe como fazer isto passo a passo.

- 1- Primeiro, o que se deseja é uma lista, assim, inicia-se colocando os colchetes que delimitam uma lista:

[]

- 2- O próximo passo é colocar o separador entre a função e o domínio utilizados por tal notação:

[\]

Função   **Domínio e restrições**

- 3- Quando a função possuir apenas uma regra, coloca-se a regra do lado esquerdo do separador \.
- Se a mesma possuir mais de uma regra, coloca-se do lado esquerdo do separador a função com seu(s) argumento(s).

- No caso, a função é $f(x)=x$, possuindo apenas uma regra: x , ou seja, o valor inferido é igual ao argumento da função.
- Neste caso, coloca-se uma letra ou palavra qualquer como regra, desde que a mesma comece com uma letra minúscula (exigência da linguagem CLEAN).
- Feito isto, obtém-se:

$$[x \setminus]$$

4- O próximo passo, é colocar o domínio sem suas restrições (mesmo que elas existam).

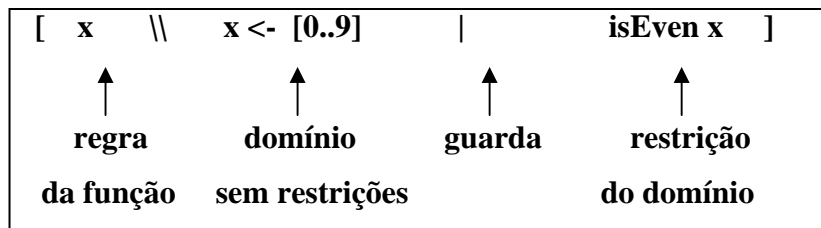
- O domínio é: **x pertence ao conjunto dos números naturais, tal que x é maior ou igual a 0 e menor ou igual a 9 ($\{x \in \mathbb{N}, 0 \leq x \leq 9\}$).**
- Assim, como o teclado do computador não possui o símbolo de pertence (\in) nem o do conjunto dos naturais (\mathbb{N}), os mesmos são substituídos no CLEAN por $<-$ e $[0..9]$ respectivamente, onde $[0..9]$ corresponde à lista com todos os naturais de 0 a 9.
- Esta lista ($[0..9]$) é equivalente a **$[0,1,2,3,4,5,6,7,8,9]$** .
- A notação em CLEAN fica:

$$[x \setminus x <- [0..9]]$$

5- Se o que se deseja fosse apenas isto, a lista produzida seria uma cópia do domínio, ou seja, uma lista de elementos x , tal que cada elemento da imagem é igual a um elemento do domínio.

- No exercício proposto, o domínio possui uma regra de restrição:
 - O elemento do domínio x , argumento da função, tem que **ser par**.
- Para iniciar uma regra no CLEAN, deve-se colocar uma **guarda**, uma barra vertical, após a descrição geral do domínio.
- Logo após esta guarda ($|$) coloca-se a(s) restrição(ões).
- A restrição, neste caso, é que só devem ser considerados pela função os argumentos pares do domínio.
- Uma restrição é reconhecida quando a mesma necessita de aplicação de alguma (ou mais de uma) função aos elementos do domínio.
- Neste presente caso, deve-se aplicar uma função que verifica cada argumento do domínio se o mesmo é **par**.

- Se for, a regra da função é aplicada ao mesmo e o valor é inferido. Se não for, o elemento do domínio é recusado e nele não é aplicada a regra de inferência da função em questão que inferirá os valores do conjunto imagem.
- Em CLEAN, existe uma função que testa se um inteiro é par: **isEven**.
- A notação Zermelo-Frankel para listas fica, finalmente, da seguinte forma:



ou seja:

$$[x \parallel x \leftarrow [0..9] \mid \text{isEven } x]$$

a lista gerada por esta notação é:

[0, 2, 4, 6, 8]

Resumindo:

- Uma compreensão de listas, portanto, consiste de duas partes delimitadas pelo abrir e fechar de colchetes, separadas por \parallel (duas barras invertidas).
- A parte esquerda contém uma função ou uma regra.
- **O domínio** $x \leftarrow xs$, onde xs é uma lista ou um vetor, que aparem no lado direito dos sinais divisórios \parallel . O sinal \leftarrow significa: pertence.
- Uma expressão $x \leftarrow xs$ é denominada de *gerador*.

Sedimentando os Conceitos Fundamentais

Foi visto anteriormente a generalização e exemplos de uso da notação Zermelo-Frankel. Pôde-se perceber ser a mesma uma solução aderente ao paradigma matemático e à forma com que nosso cérebro manipula a mesma informação.

É interessante rever tais conceitos com uma abordagem diferente e com alguns detalhes a mais, utilizando um exemplo já referenciado anteriormente.

O Exemplo

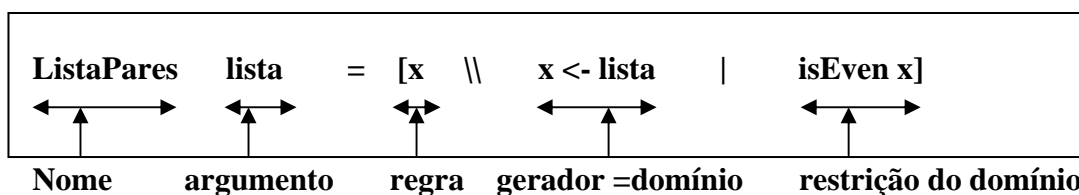
Assim, se deseja-se obter uma lista com os elementos pares inferidos de uma outra lista com uma quantidade qualquer de elementos, basta implementar um programa de forma análoga que seria a sua descrição matemática, ou seja:

ListaPares :: [Int] -> [Int]	Declaração de tipo da função
ListaPares lista = [x \ x <- lista isEven x]	Declaração da função

Revisando conceitos básicos

Função, argumento, domínio e regra:

- Uma função possui um nome, argumento(s) e regra(s). O sinal de igualdade (=) separa a regra da função com seu argumento.

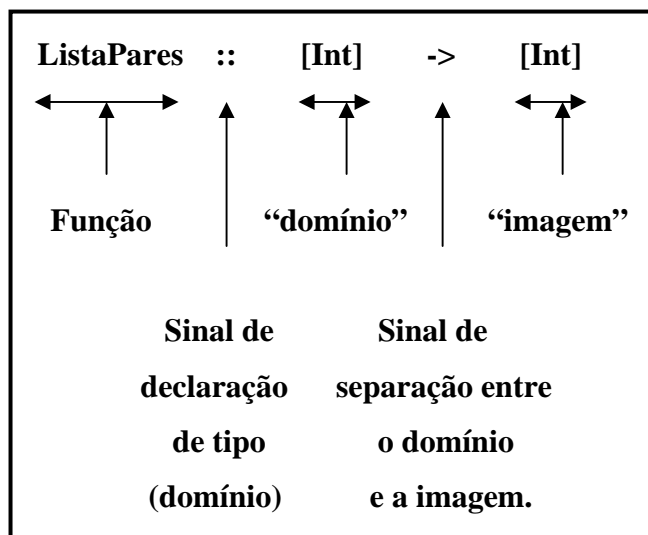


- No exemplo, o nome da função é: **ListaPares** e o seu argumento é **lista**.
- O conjunto de todos os argumentos de **lista** representa o domínio da função, o qual é novamente retratado na notação Zermelo-Frankel no gerador (**x<-lista**) que gera elemento por elemento do domínio para a inferência (determinação) dos elementos do conjunto imagem.
- Dentro da notação Zermelo-Frankel, a regra é o que está antes das duas barras invertidas, ou seja: **x**. A regra deste exemplo, **x**, diz que o elemento da imagem extraído do domínio não sofrerá modificações quando inferido por ela. Assim, se a função fosse:

ListaPares lista = [x \ x <- lista], o resultado seria uma lista idêntica à lista dos argumentos, ou seja, o domínio e a imagem seriam iguais devido a regra discriminar que o valor dos elementos na imagem serão iguais ao do domínio, ou seja, $f(x) = x$, ou, simplesmente: **x**.

- Assim, o domínio é o conjunto de todos os elementos representados pelo argumento da função, o que, pela declaração de tipo da função utilizada como exemplo resulta: o “domínio” é uma lista de inteiros **[Int]**.
- Na notação Zermelo-Frankel, o domínio é descrito pelo gerador da mesma, o que, no exemplo dado é: **x <- lista**.
- A imagem é o conjunto de todos os elementos resultantes da aplicação das regras de inferência da função a todos os elementos do domínio (com respectivas restrições). Pela declaração de tipo da função, a imagem, assim como o domínio, é uma lista de inteiros.

Declaração do tipo de dados do argumento e do valor da função:



- A imagem é, portanto, conforme afirmado, o conjunto formado pela aplicação da(s) regra(s) de inferência da função a todos os elementos do domínio, resguardadas as restrições feitas ao domínio na declaração da função. Neste caso, a restrição imposta é que a função somente deverá ser aplicada aos elementos pares do domínio (da lista de inteiros fornecida), ou seja: **isEven x**.

- Assim, tem-se:
 - **Regra** = devolve uma lista de elementos do tipo **x (Int)** sem alterá-los
 - **Domínio** = Os elementos de **x** pertencem à lista dada: **x <- lista**
 - **Restrição** = Só deverão ser pegos os elementos pares da lista: **isEven x**
 - **Imagem** = uma lista dos elementos pares do domínio: **[x \ x <- lista | isEven x]**

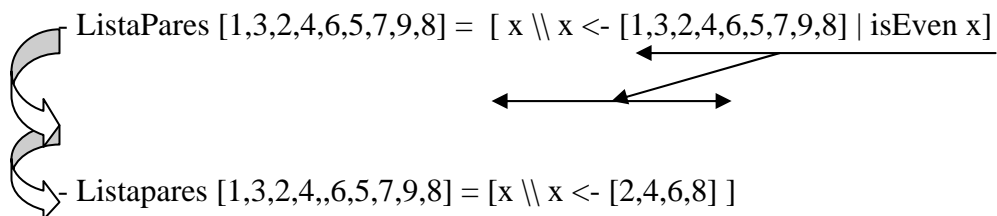
Exemplo:

Start = ListaPares [1,3,2,4,6,5,7,9,8]

Devolve:

[2,4,6,8]

ou seja:



- ListaPares [1,3,2,4,,6,5,7,9,8] = [2,4,6,8]

Obs. Perceba que o domínio que inicialmente era a lista [1,3,2,4,6,5,7,9,8], aplicada a restrição imposta ao mesmo, é como ele se tornasse: [2,4,6,8], sob o qual a regra (x) finalmente seria aplicada.

Funções com uma ou mais regras


Observe, a seguir, como implementar funções matemáticas que possuem uma ou mais regras de inferência, com respectivas condições de aplicabilidade de cada uma delas.

1- Função com uma regra de inferência:

Quando uma função possui apenas uma regra, a mesma vem logo após ao sinal de igualdade (=) .

- **1o. Exemplo**

Formalização Matemática

$$f(x) = x$$


A regra de inferência desta função é:


->dado um argumento do domínio, o valor no contra-domínio é igual ao valor do argumento (elemento), sem qualquer modificação. Neste caso, o conjunto imagem será igual ao conjunto domínio.

Implementação em CLEAN

f x = x

- **2o. Exemplo**

Formalização Matemática

$$f(x) = x^2$$


A regra de inferência desta função é:

->dado um argumento do domínio, o valor no contra-domínio é igual ao valor do argumento (elemento) elevado ao quadrado. Neste caso, novamente a quantidade de elementos da imagem é igual à do domínio.

Implementação em CLEAN

f x = x^2

Função com duas regras de inferência

1º. Exemplo (uma função hipotética qualquer)

Formalização Matemática

$$\begin{cases} f(x) = \text{sen}(x), & \text{se } 0.0 < x < 3,1416 \\ f(x) = \text{cos}(x), & \text{se } 3,1416 \leq x \leq 4,7124 \end{cases}$$

\uparrow \uparrow
regras **condições das regras**

onde: $3,1416 = \Pi(\text{pi})$

e $4,7124 = 3\Pi/2$

Implementação em CLEAN

Funções com mais de uma regra e condições de aplicação de cada uma devem utilizar guadas (|) para iniciar cada regra. Desta forma, após o último argumento da função, deve-se iniciar uma guarda com a primeira condição. A cada condição, a cada regra, uma nova guarda deve ser iniciada. Veja como isto é feito através do seguinte exemplo:

```
f x
| x > 0.0    && x < 3.1416 = sin x
| x >= 3.1416 && x <= 4.7124 = cos x
```

\uparrow \uparrow \uparrow

Guardas condições regras da função

Obs1. Em CLEAN, em vez de vírgula para separar a parte inteira da parte decimal de um número é utilizado o sinal de ponto (.).

Obs2. A regra da condição que obter sucesso é a que será executada

Obs3. Observe que neste exemplo as condições de aplicabilidade das regras não estão completas, ou seja: o que aconteceria se o valor do argumento x fosse instanciado, assumisse um valor entre com um valor entre $3\Pi/2$ e 2Π (ou valores múltiplos deste e dos demais)? O que ocorre é que o programa retornaria uma mensagem de erro dizendo que não encontrou uma regra que satisfizesse. Assim, existe uma função em CLEAN

que permite se aplicar uma outra regra a qualquer condição atendida. Esta função é: **otherwise**, e deve ser colocada depois da última guarda existente, ou seja:

f x

| **x > 0.0 && x < 3.1416 = sin x**

| **x >= 3.1416 && x <= 4.7124 = cos x**

| **otherwise = abort “argumento nao previsto pelas regras da função”**

-
- Veja que nesta guarda, utiliza-se como regra, como resultado, uma informação no formato de uma String.
 - Os sinais **&&** corresponde ao operador lógico **E**.
 - Anterior à mensagem (a string) está a palavra reservada **abort** . A mesma é uma função que faz com que o CLEAN interrompa a execução da função e envie a mensagem, a String, logo após ela, ao usuário.
 - Se entrarmos com o argumento 5.0 a esta função, **f 5.0**, a mensagem descrita **“argumento nao previsto pelas regras da função”** será apresentada e a tentativa de aplicação da função ao argumento **5.0** será abortada pela função **abort**.
-

2º. Exemplo (fatorial de um número)

Formalização Matemática

$$\begin{cases} f(x) = 1, & \text{se } x = 0 \\ f(x) = x \cdot (x-1)!, & \text{se } x > 0 \end{cases} \quad \dots \text{Obs. Por definição, } 0! = 1.$$

\uparrow
regras

\uparrow
condições das regras

Implementação em CLEAN

```
f x
| x == 0 = 1
| x > 0 = x * f(x-1)
```

Guardas condições regras da função

Utilizando a condição **otherwise** e a função regra **abort** para eventuais valores não previsto na função fatorial, tal como: -5, **f** fica:

```
f x
| x == 0 = 1
| x > 0 = x * f(x-1)
| otherwise = abort ("não existe fatorial de " ++ (toString x) )
```

Veja que neste caso a mensagem devolve uma frase que: “**nao existe fatorial de x**”, onde **x** é o valor do argumento. A função **toString** transforma o argumento da função em uma string, para que a função **++** [LIMA e RUFINO] faça a concatenação de duas strings: a mensagem e o argumento convertido em string.

Observe que quando uma função possui mais de uma regra, uma condição deve ser atendida para que a presente regra seja aplicada ao argumento em questão.

- 1- Caso a condição da primeira regra falhe (não seja verdadeira), o CLEAN passa a verificar a próxima condição.
- 2- Se a condição da próxima regra falhar, novamente o CLEAN passa a verificar a próxima condição.
- 3- O item 2 (anterior) é executado até que uma condição seja atendida. Quando isto ocorrer, a regra desta condição é aplicada ao argumento em questão (sob teste da condição) e o valor no contradomínio é inferido.

Mais detalhes sobre a notação Zermelo-Frankel

Uma expressão $x \leftarrow xs$ é denominado de *gerador*.

Tal como na Matemática, CLEAN permite que se utilizem condições para filtrar (eliminar) elementos do contradomínio. Veja o seguinte exemplo ilustrativo:

Exemplo: Gerar uma lista cujos elementos sejam o quadrado dos números pares que estão no intervalo de 1 a 10.

Da matemática temos: $S = \{ x^2 \mid 1 \leq x \leq 10 \wedge x \text{ é par} \}$

Em CLEAN¹ temos: $S = [x*x \mid x \leftarrow [1..10] \mid \text{isEven } x]$

Obs. A função **isEven** é uma primitiva do CLEAN que testa se um valor é ou não par.

Utilizando mais de um *gerador* ao mesmo tempo

Na notação Zermelo-Frankel, após as duas barras invertidas ($\mid\mid$) pode aparecer mais de um *gerador* separados por uma vírgula. Quando isto ocorre temos uma *combinação ortogonal de geradores*. Ao se utilizar uma combinação ortogonal de geradores, a expressão colocada na frente das duas barras invertidas é calculada para toda possível combinação das variáveis correspondentes. O exemplo, a seguir, torna mais claro o que foi dito:

```
compr2.icl - C:\testesCLEAN082001\compr2.icl
1 module compr2
2 import StdEnv
3
4 Start :: [(Int, Int)]
5 Start = [(x,y) \ x <- [1..3], y <- [4..6]]

press any key to exit

[(1,4),(1,5),(1,6),(2,4),(2,5),(2,6),(3,4),(3,5),(3,6)]
```

¹ Observe o quanto aderente, o quão parecido, é a notação em CLEAN à definição matemática.

Nela, a expressão colocada na frente das duas barras invertidas é calculada para toda possível combinação das variáveis correspondentes.

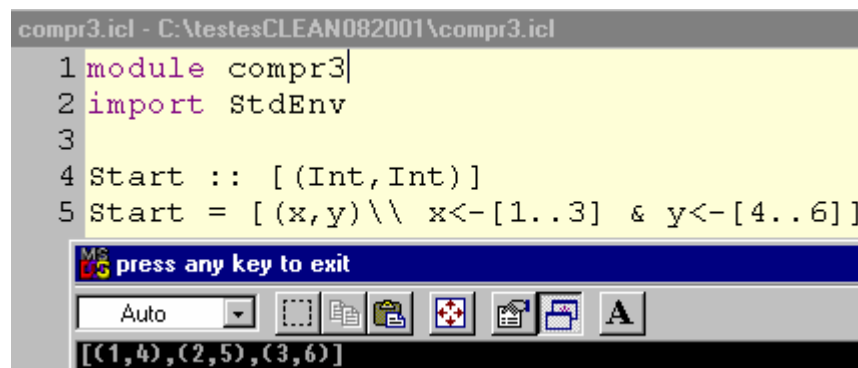
Este programa devolve a lista [(1,4),(1,5),(1,6),(2,4),(2,5),(2,6),(3,4),(3,5),(3,6)], ou seja, ele pega o primeiro elemento do primeiro gerador ($x \leftarrow [1..3]$) e devolve todas as combinações com os elementos do segundo gerador ($y \leftarrow [4..6]$), ou seja: (1,4), (1,5) e (1,6), depois pega o segundo elemento do primeiro gerador e devolve todas as combinações com os elementos do segundo gerador, ou seja: (2,4), (2,5) e (2,6), o processo continua até que o programa tenha obtido todas as combinações possíveis de elementos do primeiro gerador com os elementos do segundo gerador. Finalmente o sistema devolve a lista [(1,4),(1,5),(1,6),(2,4),(2,5),(2,6),(3,4),(3,5),(3,6)], ou seja: todas as combinações de (x,y) para os geradores dados.

Combinação paralela de geradores

Existe ainda uma outra forma de combinar geradores denominada de: *combinação paralela de geradores*. Para diferenciar da forma anterior, nesta combinação os geradores são separados pelo símbolo & em vez de vírgula (,).

A diferença entre as duas formas de utilizar geradores é que a segunda, a combinação paralela de geradores, produza como resultado apenas a combinação de elementos de mesmo índice de cada gerador, ou seja: o primeiro elemento do primeiro gerador como primeiro elemento do segundo gerador, o segundo elemento do primeiro gerador com o segundo elemento do segundo gerador, e assim sucessivamente.

Veja o exemplo a seguir:



```
compr3.icl - C:\testesCLEAN082001\compr3.icl
1 module compr3
2 import StdEnv
3
4 Start :: [(Int,Int)]
5 Start = [(x,y) \ x<-[1..3] & y<-[4..6]]

press any key to exit

Auto
[(1,4),(2,5),(3,6)]
```

os geradores são separados pelo símbolo & em vez de vírgula (,)

O programa acima gera como resultado a lista [(1,4),(2,5),(3,6)] , ou seja, a combinação de elementos de mesmo índice:

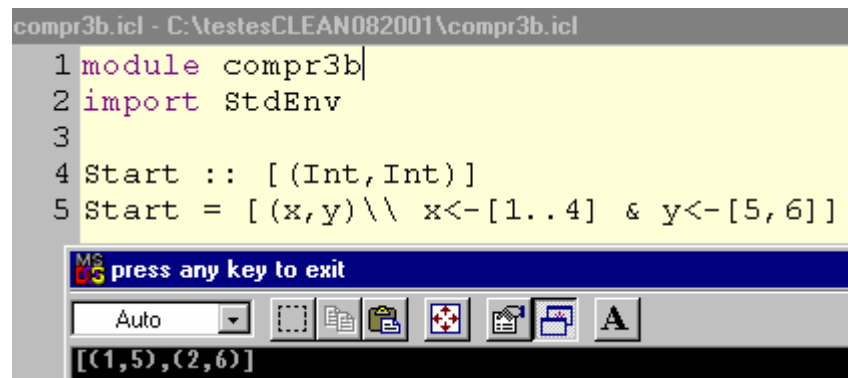
Primeiro gerador – [1 , 2 , 3]

Segundo gerador – [4 , 5 , 6]

Se o primeiro gerador tiver uma quantidade de elementos diferente da do segundo gerador, o programa encerra a geração da lista quando todos os elementos do gerador que possuir a menor lista for percorrido. Veja o exemplo a seguir:

Primeiro gerador – [1 , 2 , 3 , 4]

Segundo gerador – [5 , 6]



The screenshot shows a window titled 'compr3b.icl - C:\testesCLEAN082001\compr3b.icl'. The code in the editor is:

```

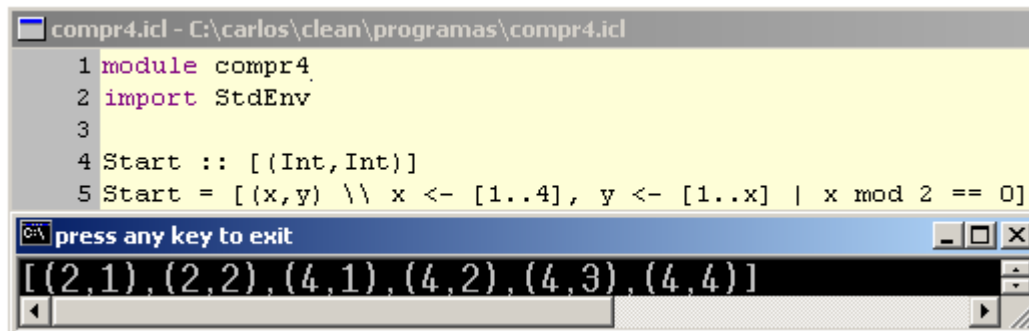
1 module compr3b
2 import StdEnv
3
4 Start :: [(Int,Int)]
5 Start = [ (x,y) \ x<-[1..4] & y<-[5,6] ]

```

Below the code, a status bar displays 'press any key to exit'. At the bottom, the output of the program is shown as '[(1,5),(2,6)]'.

Assim, quando a menor lista é totalmente percorrida, todos os geradores combinados com & param de gerar elementos.

Pode-se estabelecer condições para os valores a serem gerados por uma combinação de geradores. A condição, mais uma vez, deve ser separada dos geradores por uma barra vertical. O programa devolve a lista [(2,1),(2,2),(4,1),(4,2),(4,3),(4,4)]. Observe que a visibilidade da variável *x* (denominada de *escopo* de *x*) não se restringe somente ao lado esquerdo da compreensão. Ela também pode ser manipulada no lado direito do gerador introduzindo *x*. Entretanto *y* não pode ser utilizada nos geradores precedendo-a, isto é, *y* pode somente ser usada em (*x*,*y*) e na condição. O símbolo <- que define a pertinência é especialmente utilizado na compreensão de listas e não constitui um operador.



```

compr4.icl - C:\carlos\clean\programas\compr4.icl
1 module compr4
2 import StdEnv
3
4 Start :: [(Int,Int)]
5 Start = [(x,y) \\ x <- [1..4], y <- [1..x] | x mod 2 == 0]

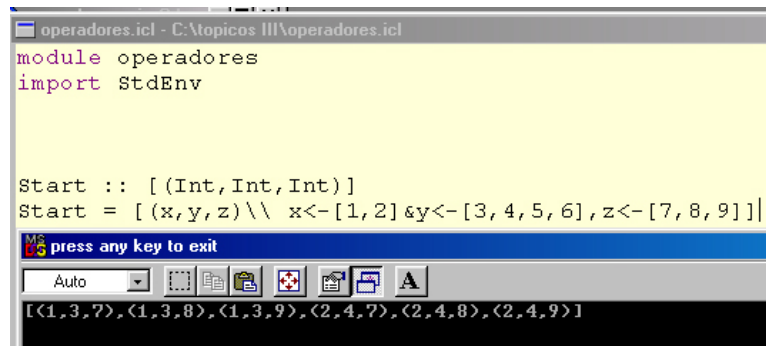
press any key to exit
[(2,1),(2,2),(4,1),(4,2),(4,3),(4,4)]

```

- Prioridade dos operadores dos geradores (& e ,).

O operador “,” possui prioridade em relação ao operador “&”.

Vamos ver alguns exemplos para tornar claro esta afirmativa.



```

operadores.icl - C:\topicos III\operadores.icl
module operadores
import StdEnv

Start :: [(Int,Int,Int)]
Start = [(x,y,z) \\ x <- [1,2] & y <- [3,4,5,6], z <- [7,8,9]]

press any key to exit
Auto
[<1,3,7>,<1,3,8>,<1,3,9>,<2,4,7>,<2,4,8>,<2,4,9>]

```

Interface Visual

Toda interface visual e gráfica necessita da utilização de avaliações destrutivas e de ferir o princípio da transparência referencial matemática para sua viabilização. Fazer isto é matematicamente inaceitável, mas, sem tais permissões, é impossível se implementar tais interfaces. Para tanto, as linguagens funcionais modernas criaram princípios, filosofia, métodos, de como se implementar tais interfaces sem que, com isto, tais absurdos matemáticos venham acarretar efeitos colaterais graves nos produtos gerados por elas.

Assim, a linguagem Haskell [33] [34] criou **Mônadas** e O CLEAN o princípio dos **Tipos únicos**. Este princípio garante que cada variável do sistema, cada dado utilizado, seja referenciado apenas uma vez em um processo, e, como cada processo só possui variáveis e funções locais, os efeitos destrutivos causados nos processos são controlados localmente e não afetam globalmente os demais processos. A seguir, é apresentado progressivamente o projeto de interfaces com campos de texto, botões e outros recursos.

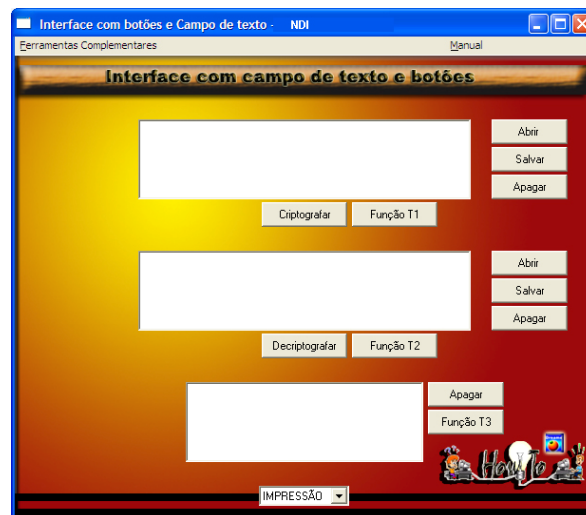
- **Interface com botões e caixa de texto**

Existem 3 tipos básicos de interface: NDI, SDI e MDI.

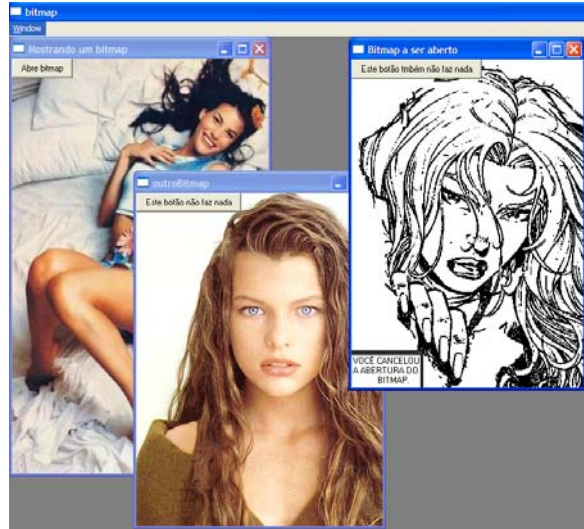
- **NDI - No Document Interface** . Esta é uma interface que possui um arcabouço, campos de textos, botões, pop-ups, mas que não permite inclusão de imagens. É uma interface denominada de diálogo. Exemplo: Janelas de informação de erros do sistema.



- **SDI - Single Document Interface** – é uma interface semelhante à NDI, com a diferença que pode-se colocar imagens de fundo e mudar o layout e conteúdo da janela. Possui a restrição de apenas permitir a visualização de uma janela de cada vez. Exemplo: Paint Brush, Bloco de notas, ...



- **MDI - Multiple Document Interface** – Interface com todos os recursos de imagem, permitindo que se abram várias janelas ao mesmo tempo. Exemplo: Word, Photoshop,



No sistema proposto nesta dissertação, utilizar-se-á estes três tipos de interface. A interface MDI para permitir que vários processos sejam abertos em paralelo, a SDI para cada processo (aplicativo) e a NDI para mensagens de erro e informações gerais e, também, para aplicativos.

A diferença básica de uma NDI para uma SDI e uma MDI está apenas no cabeçalho onde se permitirá abrir imagens de fundo e criar janelas dentro de janelas. Como todos os conceitos de uma interface NDI são válidos para as outras duas, neste capítulo serão abordados detalhes relevantes à construção de NDIs.

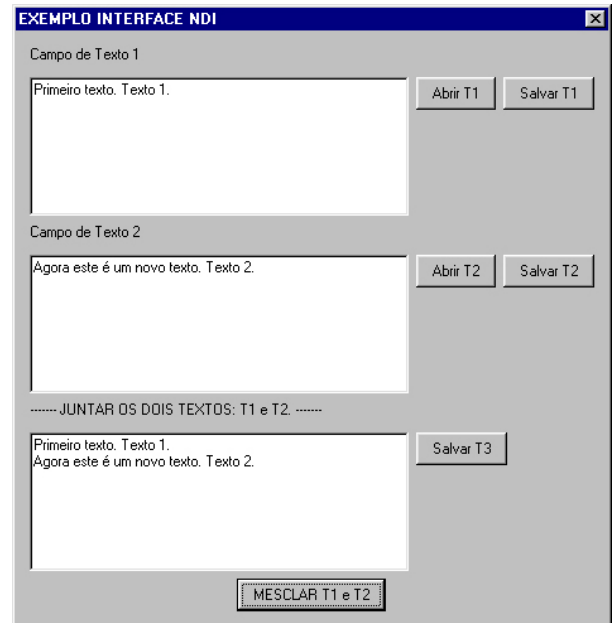
• Construção de interfaces NDIs

Uma janela é delimitada por um arcabouço, ou seja, uma barra mais grossa em azul contendo o nome da janela e linhas laterais e inferior que a delimita.

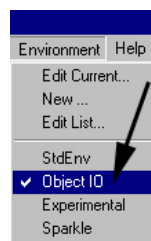
Para exemplificar, serão apresentados os passos de construção de uma janela NDI que posua os seguintes componentes:

- Três campos de texto.
- O primeiro e o segundo para colocar um texto escrito ou lido de um arquivo texto qualquer.
- O terceiro texto pode ser também editado normalmente, mas, se você clicar no botão logo abaixo dele, no mesmo será colocado os textos dos dois primeiros campos.

- Cada um dos dois primeiros textos possui um botão de abrir um arquivo texto de até 300.000 caracteres e um botão de salvar o que estiver no campo.
- No terceiro campo tem um botão de salvar e um de mesclar os dois primeiros textos. Veja na figura ao lado a interface que acabamos de especificar:




Para que se compile um programa com interface e ferramentas de IO (entrada e saída de dados), não se deve, antes, esquecer de compilar (rodar o programa). Você deve configurar o ambiente (Environment) para **Objet IO**, conforme figura a seguir:



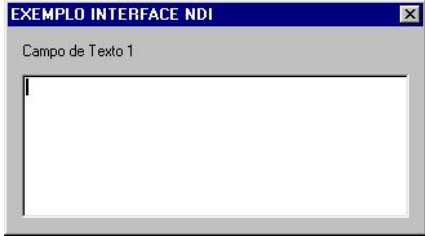
• Implementação do Arcabouço

O programa a seguir possui apenas o cabeçalho principal que será utilizado nos outros exemplos. A interface abre uma moldura (arcabouço) com um label: **MOUDURA**. Código do programa **arcabouço.icl**.

<pre> module arcabouço import StdIO,StdEnv // linha padrão que inicia uma interface NDI Start comp = startIO NDI 0 objeto[] comp // função que abre um processo para criação de // uma janela de diálogo objeto proc # (_,proc)=openDialog "" dialogo proc =proc // função que abre a janela de diálogo, dá // nome ao arcabouço e cria uma linha de texto // através do comando TextControl dialogo = Dialog "EXEMPLO INTERFACE NDI" (TextControl "MOUDURA" []) [WindowClose (noLS closeProcess)] </pre>	
---	---

- Colocando um novo lable e um campo de texto.

Código do programa **um.icl**

<pre> module um import StdIO,StdEnv Start comp = startIO NDI 0 objeto [] comp objeto proc // linha que cria identificadores para os objetos (campo de texto, // botões, ... #(ids,proc)=openIds 2 proc #(_,proc)=openDialog "" (dialogo ids) proc =proc dialogo ids = Dialog "EXEMPLO INTERFACE NDI" // TextControl cria uma linha de texto // ControlPos indic o alinhamento, no caso, à esquerda da janela(Left) (TextControl "Campo de Texto 1" [ControlPos (Left,zero)]) // :+: -> cola de controles, acrescenta mais um controle // na interface :+: //EditControl cria um campo de edição, no caso, com 300 pixels de // largura // e com 8 linhas EditControl "" (PixelFormat 300) 8 [ControlId t1,ControlPos (Left,zero)]) [WindowClose (noLS closeProcess) ,WindowId wd1] // nomeando os identificadores (t1 é o campo de texto, wd1 é o // arcabouço)] where t1 =ids!!0 wd1 =ids!!1 </pre>	
---	---

Observe, também, que na função **objeto** foram criados dois identificadores: um para a janela de diálogo e outro para o campo de texto.

A seguinte linha faz isto:

`#(ids,proc)=openIds 2 proc`, em, em consequência disto, a função dialogo é agora chamada com um argumento, este argumento são os identificadores. A linha que faz isto é: `#(_,proc)=openDialog ""` (**dialogo ids**) `proc`.

A linha:

`EditControl "" (PixelWidth 300) 8 [ControlId t1,ControlPos (Left,zero)]` cria um campo de texto de **300** pixels de largura por **8** linhas colocado no lado esquerdo (**Left**) da janela de diálogo.

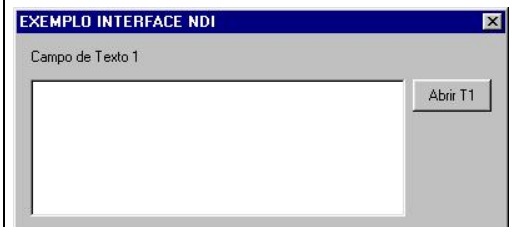
O símbolo `:+:` que significa control glue (cola de controle), adiciona um controle a mais na janela de diálogo.

- **Colocando um lable, um campo de texto e um botão de abrir texto.**

O programa dois.icl

```
module dois
import StdIO,StdEnv

Start comp
= startIO NDI
  0
  objeto
  []
  comp
objeto proc
#(ids,proc)=openIds 2 proc
#(_,proc)=openDialog "" (dialogo ids) proc
=proc
dialogo ids = Dialog "EXEMPLO INTERFACE NDI"
  (TextControl "Campo de Texto 1" [ControlPos
    (Left,zero)]
  :+
  EditControl "" (PixelWidth 300) 8 [ControlId t1,
    ControlPos (Left,zero)]
  :+
  //ButtonControl cria um botão
  //ControlFunction faz com que uma função seja disparada
  //ao se clicar no respectivo botão, no caso, a função ler1
  ButtonControl "Abrir T1" [ControlFunction ler1]
  )
  [WindowClose (noLS closeProcess)
  ,WindowId wd1
  ]
where
  t1 =ids!!0
  wd1 =ids!!1
  //Função para abrir um arquivo de texto através do explorer
  ler1 (el,proc)
  //selectInputFile abre o explorer para seleção de um
  arquivo
  #(maybeFile, proc)= selectInputFile proc
  | isNothing maybeFile= (el,proc)
  # (Just nome)= maybeFile
  //fopen abre o arquivo selecionado
  //FReadText incide que o arquivo aberto é para leitura
  # (ok, file, proc)= fopen nome FReadText proc
  //freads faz a leitura de n bytes do arquivo, no caso:
  // 3.000.000 Bytes
  # (conteudo, file1)= fread file 300000
  // setControlText escreve em um campo de texto,
  // no caso, o conteúdo de até 3.00.000 Bytes do arquivo lido
```



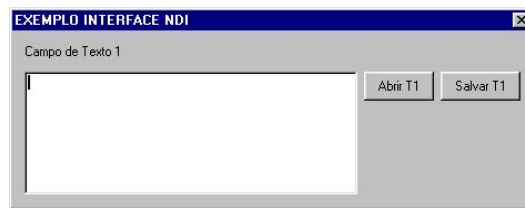
<pre># proc= appPIO (setControlText t1 conteudo) proc //fclose fech o arquivo aberto para liberá-lo para outros programas ou processos utilizarem. # (ok, proc)= fclose file1 proc =(el,proc)</pre>	
---	--

OBS.

Veja que, como um botão foi criado, o mesmo deverá fazer alguma coisa, assim, uma função foi associada ao mesmo: **ler1**

- **Mantendo a interface criada e colocando um botão para salvar o que estiver no campo de texto**

Daqui para frente, como os programas aumentam sensivelmente, primeiro será apresentada a interface gerada, e, logo após o código do programa. O programa da próxima interface é **tres.icl**



```
module tres
import StdIO,StdEnv

Start comp
= startIO NDI
  0
  objeto
  []
  comp
objeto proc
#(ids,proc)=openIds 2 proc
#(,proc)=openDialog "" (dialogo ids) proc
=proc
dialogo ids = Dialog "EXEMPLO INTERFACE NDI"
  (TextControl "Campo de Texto 1" [ControlPos (Left,zero)])
  :+
  EditControl "" (PixelWidth 300) 8 [ControlId t1,ControlPos (Left,zero)]
  :+
  ButtonControl "Abrir T1" [ControlFunction ler1]
  :+
  ButtonControl "Salvar T1" [ControlFunction salvar1]
  )
[WindowClose (noLS closeProcess)
,WindowId wd1
]

where
  t1 =ids!!0
  wd1 =ids!!1
//ABRIR ARQUIVO ATRAVES DO EXPLORER NO CAMPO DE TEXTO t1
ler1 (el,proc)
#      (maybeFile, proc)= selectInputFile proc
|      isNothing maybeFile= (el,proc)
#      (Just nome)= maybeFile
# (ok, file, proc)= fopen nome FReadText proc
# (conteudo, file1)= fread file 300000
# proc= appPIO (setControlText t1 conteudo) proc
# (ok, proc)= fclose file1 proc
```

```

=(el,proc)
//SALVAR ARQUIVO texto1 ATRAVES DO EXPLORER
salvar1(el,proc)
//getWindow lê os identificadores da janela
# (Just dial,proc) = accPIO(getWindow wd1) proc
# (se,Just x) = getControlText t1 dial
// selectOutputFile seleciona um arquivo para ser gravado
# (talvez, proc) = selectOutputFile "Salvar o resultado: escolha o diretorio!" "*.txt" proc
| isNothing talvez= (el,proc)
//FWriteText prepara o endereço para gravação
# (ok,file2, proc)= fopen (fromJust talvez) FWriteText proc
| not ok = (el,proc)
// fwrites grava o conteúdo desejado no arquivo aberto
# file2 = fwrites x file2
# (ok, proc)= fclose file2 proc
# proc= appPIO (setControlText t1 ("(* Conteudo t1 gravado com sucesso *)")+++
{toChar 13,toChar 10}+++ {toChar 13,toChar 10}+++ x))proc
=(el,proc)
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

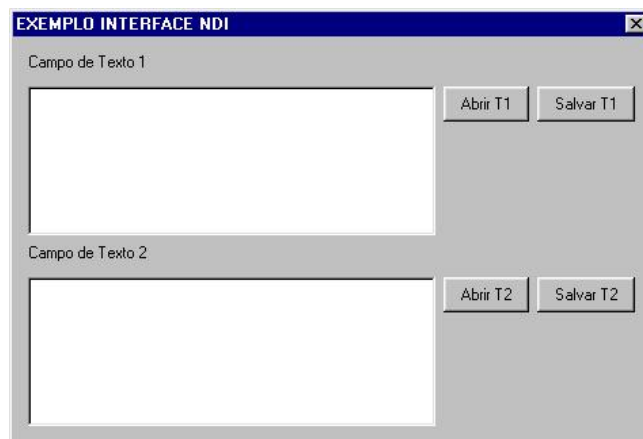
Observe que daqui pra frente o que se faz é apenas repetir ações e comandos já conhecidos das interfaces anteriores.

- **Acrescentando mais um campo de texto com respectivos botões. Observe que “a receita” se repete.**

O programa está é quatro.icl.

Observe que o número de identificadores de janela foi aumentado para 3, ou seja:

#(ids,proc)=openIds 3 proc



```

module quatro
import StdIO,StdEnv

Start comp
= startIO NDI
  0
  objeto
  []
  comp
objeto proc
#(ids,proc)=openIds 3 proc
#(_,proc)=openDialog "" (dialogo ids) proc
=proc
dialogo ids = Dialog "EXEMPLO INTERFACE NDI"

```

```

(TextControl "Campo de Texto 1" [ControlPos (Left,zero)]
:+:
EditControl "" (PixelWidth 300) 8
      [ControllId t1,ControlPos (Left,zero)]
:+:
ButtonControl "Abrir T1" [ControlFunction ler1]
:+:
ButtonControl "Salvar T1" [ControlFunction salvar1]
:+:
TextControl "Campo de Texto 2" [ControlPos (Left,zero)]
:+:
EditControl "" (PixelWidth 300) 8
      [ControllId t2,ControlPos (Left,zero)]
:+:
ButtonControl "Abrir T2" [ControlFunction ler2]
:+:
ButtonControl "Salvar T2" [ControlFunction salvar2]
)
[WindowClose (noLS closeProcess)
,WindowId wd1
]
where
t1 =ids!!0
t2 =ids!!1
wd1 =ids!!2
//ABRIR ARQUIVO ATRAVES DO EXPLORER NO CAMPO DE TEXTO t1
ler1 (el,proc)
# (maybeFile, proc)= selectInputFile proc
| isNothing maybeFile= (el,proc)
# (Just nome)= maybeFile
# (ok, file, proc)= fopen nome FReadText proc
# (conteudo, file1)= fread file 300000
# proc= appPIO (setControlText t1 conteudo) proc
# (ok, proc)= fclose file1 proc
=(el,proc)
//SALVAR ARQUIVO texto1 ATRAVES DO EXPLORER
salvar1 (el,proc)
# (Just dial,proc) = accPIO(getWindow wd1) proc
# (se,Just x) = getControlText t1 dial
# (talvez, proc) = selectOutputFile "Salvar o resultado: escolha o diretorio!" "*.txt" proc
| isNothing talvez= (el,proc)
# (ok,file2, proc)= fopen (fromJust talvez) FWriteText proc
| not ok = (el,proc)
# file2 = fwrites x file2
# (ok, proc)= fclose file2 proc
# proc= appPIO (setControlText t1
  ("(* Conteudo t1 gravado com sucesso *)"+++
   {toChar 13,toChar 10}+++ {toChar 13,toChar 10}
   +++ x))proc
=(el,proc)
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//ABRIR ARQUIVO ATRAVES DO EXPLORER NO CAMPO DE TEXTO t2
ler2 (el,proc)
# (maybeFile, proc)= selectInputFile proc
| isNothing maybeFile= (el,proc)
# (Just nome)= maybeFile
# (ok, file, proc)= fopen nome FReadText proc
# (conteudo, file1)= fread file 300000
# proc= appPIO (setControlText t2 conteudo) proc
# (ok, proc)= fclose file1 proc
=(el,proc)
//SALVAR ARQUIVO texto2 ATRAVES DO EXPLORER
salvar2 (el,proc)
# (Just dial,proc) = accPIO(getWindow wd1) proc
# (se,Just x) = getControlText t2 dial
# (talvez, proc) = selectOutputFile "Salvar o resultado: escolha o diretorio!" "*.txt" proc
| isNothing talvez= (el,proc)
# (ok,file2, proc)= fopen (fromJust talvez) FWriteText proc
| not ok = (el,proc)
# file2 = fwrites x file2
# (ok, proc)= fclose file2 proc
# proc= appPIO (setControlText t2
  ("(* Conteudo t2 gravado com sucesso *)"+++

```

```

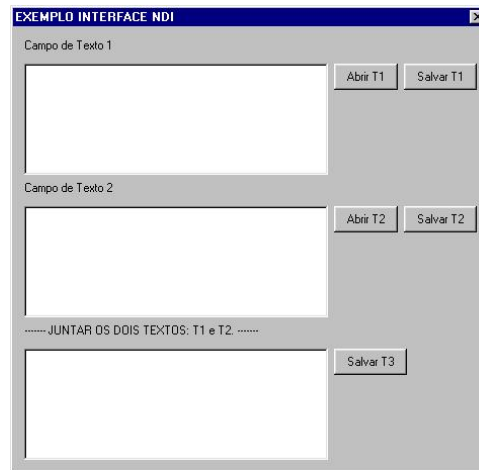
{toChar 13,toChar 10}+++ {toChar 13,toChar 10}
+++ x))proc
=(el,proc)
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

- **Acrescentando mais um campo de texto com um botão de salvar e um label antes, predizendo o que se deseja fazer neste campo de texto**

O programa é cinco.icl.

Observe que o número de identificadores de janela foi aumentado para 3, ou seja: **#(ids,proc)=openIds 4 proc**



```

module cinco
import StdIO,StdEnv

Start comp
= startIO NDI
  0
  objeto
  []
  comp
objeto proc
#(ids,proc)=openIds 4 proc
#(_,proc)=openDialog "" (dialogo ids) proc
=proc
dialogo ids = Dialog "EXEMPLO INTERFACE NDI"
  (TextControl "Campo de Texto 1" [ControlPos (Left,zero)])
  :+
  EditControl "" (PixelWidth 300) 8
    [ControllId t1,ControlPos (Left,zero)]
  :+
  ButtonControl "Abrir T1" [ControlFunction ler1]
  :+
  ButtonControl "Salvar T1" [ControlFunction salvar1]
  :+
  TextControl "Campo de Texto 2" [ControlPos (Left,zero)]
  :+
  EditControl "" (PixelWidth 300) 8
    [ControllId t2,ControlPos (Left,zero)]
  :+
  ButtonControl "Abrir T2" [ControlFunction ler2]
  :+
  ButtonControl "Salvar T2" [ControlFunction salvar2]
  :+

```



```

        TextControl "----- JUNTAR OS DOIS TEXTOS: T1 e T2. -----" [ControlPos (Left,zero)]
        :+
        EditControl "" (PixelWidth 300) 8 [ControlId t3,ControlPos (Left,zero)]
        :+
        ButtonControl "Salvar T3" [ControlFunction salvar3]
        )
        [WindowClose (noLS closeProcess)
        ,WindowId wd1
        ]
where
    t1 =ids!!0
    t2 =ids!!1
    t3 = ids!!2
    wd1 = ids!!3
//ABRIR ARQUIVO ATRAVES DO EXPLORER NO CAMPO DE TEXTO t1
    ler1 (el,proc)
    # (maybeFile, proc)= selectInputFile proc
    | isNothing maybeFile= (el,proc)
    # (Just nome)= maybeFile
    # (ok, file, proc)= fopen nome FReadText proc
    # (conteudo, file1)= fread file 300000
    # proc= appPIO (setControlText t1 conteudo) proc
    # (ok, proc)= fclose file1 proc
    =(el,proc)
//SALVAR ARQUIVO texto1 ATRAVES DO EXPLORER
    salvar1(el,proc)
    # (Just dial,proc) = accPIO(getWindow wd1) proc
    # (se,Just x) = getControlText t1 dial
    # (talvez, proc) = selectOutputFile "Salvar o resultado: escolha o diretorio!" "*.txt" proc
    | isNothing talvez= (el,proc)
    # (ok,file2, proc)= fopen (fromJust talvez) FWriteText proc
    | not ok = (el,proc)
    # file2 = fwrites x file2
    # (ok, proc)= fclose file2 proc
    # proc= appPIO (setControlText t1
    ("(* Conteudo t1 gravado com sucesso *)"+++
    {toChar 13,toChar 10}+++ {toChar 13,toChar 10}
    +++ x))proc
    =(el,proc)
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//ABRIR ARQUIVO ATRAVES DO EXPLORER NO CAMPO DE TEXTO t2
    ler2 (el,proc)
    # (maybeFile, proc)= selectInputFile proc
    | isNothing maybeFile= (el,proc)
    # (Just nome)= maybeFile
    # (ok, file, proc)= fopen nome FReadText proc
    # (conteudo, file1)= fread file 300000
    # proc= appPIO (setControlText t2 conteudo) proc
    # (ok, proc)= fclose file1 proc
    =(el,proc)
//SALVAR ARQUIVO texto2 ATRAVES DO EXPLORER
    salvar2(el,proc)
    # (Just dial,proc) = accPIO(getWindow wd1) proc
    # (se,Just x) = getControlText t2 dial
    # (talvez, proc) = selectOutputFile "Salvar o resultado: escolha o diretorio!" "*.txt" proc
    | isNothing talvez= (el,proc)
    # (ok,file2, proc)= fopen (fromJust talvez) FWriteText proc
    | not ok = (el,proc)
    # file2 = fwrites x file2
    # (ok, proc)= fclose file2 proc
    # proc= appPIO (setControlText t2
    ("(* Conteudo t2 gravado com sucesso *)"+++
    {toChar 13,toChar 10}+++ {toChar 13,toChar 10}
    +++ x))proc
    =(el,proc)
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//SALVAR ARQUIVO texto3 ATRAVES DO EXPLORER
    salvar3(el,proc)
    # (Just dial,proc) = accPIO(getWindow wd1) proc
    # (se,Just x) = getControlText t3 dial
    # (talvez, proc) = selectOutputFile "Salvar o resultado: escolha o diretorio!" "*.txt" proc
    | isNothing talvez= (el,proc)
    # (ok,file2, proc)= fopen (fromJust talvez) FWriteText proc
    | not ok = (el,proc)
    # file2 = fwrites x file2
    # (ok, proc)= fclose file2 proc

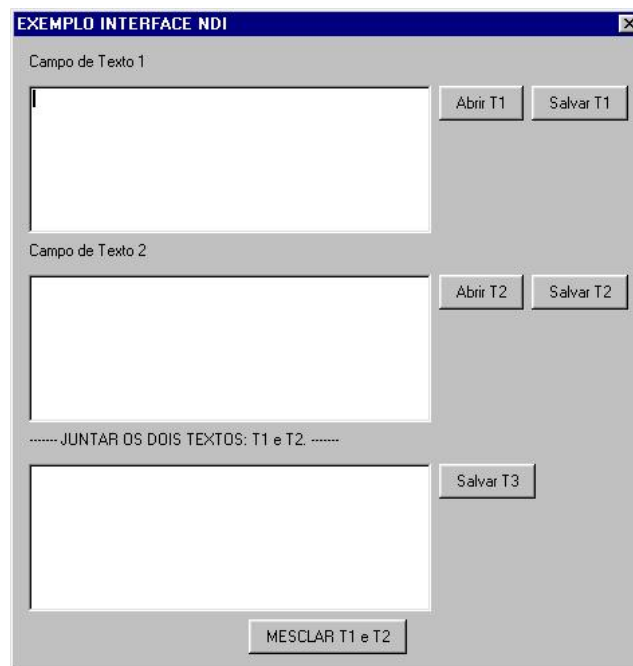
```

```
# proc= appPIO (setControlText t3
  ("(* Conteudo t2 gravado com sucesso *)"+++
   {toChar 13,toChar 10}+++ {toChar 13,toChar 10}
   +++ x))proc
=(el,proc)
```

- **Acrescentando um botão que ao ser acionado faça um merge dos dois textos, ou seja, t1 com t2, e coloca o resultado no campo de texto t3.**

O programa está em **ultimo.icl**.

Foi criada uma função externa à interface (você pode verificar que a endentação não é a mesma, ou seja, está iniciando no alinhamento esquerdo do texto. Esta função serve para saltar uma linha. A mesma já estava sendo utilizada antes mas sem ser uma função {toChar 13, toChar 10}.



```
module cinco
import StdIO,StdEnv

Start comp
= startIO NDI
  0
  objeto
  []
  comp
objeto proc
#(ids,proc)=openIds 4 proc
#(.,proc)=openDialog "" (dialogo ids) proc
=proc
dialogo ids = Dialog "EXEMPLO INTERFACE NDI"
  (TextControl "Campo de Texto 1" [ControlPos (Left,zero)]
   :+
   EditControl "" (PixelFormat 300) 8
```

```

[ControllId t1,ControlPos (Left,zero)]
:+:
ButtonControl "Abrir T1" [ControlFunction ler1]
:+:
ButtonControl "Salvar T1" [ControlFunction salvar1]
:+:
TextControl "Campo de Texto 2" [ControlPos (Left,zero)]
:+:
EditControl "" (PixelWidth 300) 8
[ControllId t2,ControlPos (Left,zero)]
:+:
ButtonControl "Abrir T2" [ControlFunction ler2]
:+:
ButtonControl "Salvar T2" [ControlFunction salvar2]
:+:
TextControl "----- JUNTAR OS DOIS TEXTOS: T1 e T2. -----" [ControlPos (Left,zero)]
:+:
EditControl "" (PixelWidth 300) 8 [ControllId t3,ControlPos (Left,zero)]
:+:
ButtonControl "Salvar T3" [ControlFunction salvar3]
:+:
ButtonControl "MESCLAR T1 e T2" [ControlFunction mesclar,ControlPos (Center,zero)]
)
[WindowClose (noLS closeProcess)
,WindowId wd1
]
where
t1 = ids!!0
t2 = ids!!1
t3 = ids!!2
wd1 = ids!!3
//ABRIR ARQUIVO ATRAVES DO EXPLORER NO CAMPO DE TEXTO t1
ler1 (el,proc)
# (maybeFile, proc)= selectInputFile proc
| isNothing maybeFile= (el,proc)
# (Just nome)= maybeFile
# (ok, file, proc)= fopen nome FReadText proc
# (conteudo, file1)= fread file 300000
# proc= appPIO (setControlText t1 conteudo) proc
# (ok, proc)= fclose file1 proc
=(el,proc)
//SALVAR ARQUIVO texto1 ATRAVES DO EXPLORER
salvar1(el,proc)
# (Just dial,proc) = accPIO(getWindow wd1) proc
# (se,Just x) = getControlText t1 dial
# (talvez, proc) = selectOutputFile "Salvar o resultado: escolha o diretorio!" "*.txt" proc
| isNothing talvez= (el,proc)
# (ok,file2, proc)= fopen (fromJust talvez) FWriteText proc
| not ok = (el,proc)
# file2 = fwrites x file2
# (ok, proc)= fclose file2 proc
# proc= appPIO (setControlText t1
("(* Conteudo t1 gravado com sucesso *)"+++
{toChar 13,toChar 10}+++ {toChar 13,toChar 10}
+++ x))proc
=(el,proc)
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//ABRIR ARQUIVO ATRAVES DO EXPLORER NO CAMPO DE TEXTO t2
ler2 (el,proc)
# (maybeFile, proc)= selectInputFile proc
| isNothing maybeFile= (el,proc)
# (Just nome)= maybeFile
# (ok, file, proc)= fopen nome FReadText proc
# (conteudo, file1)= fread file 300000
# proc= appPIO (setControlText t2 conteudo) proc
# (ok, proc)= fclose file1 proc
=(el,proc)
//SALVAR ARQUIVO texto2 ATRAVES DO EXPLORER
salvar2(el,proc)
# (Just dial,proc) = accPIO(getWindow wd1) proc
# (se,Just x) = getControlText t2 dial
# (talvez, proc) = selectOutputFile "Salvar o resultado: escolha o diretorio!" "*.txt" proc
| isNothing talvez= (el,proc)
# (ok,file2, proc)= fopen (fromJust talvez) FWriteText proc
| not ok = (el,proc)
# file2 = fwrites x file2

```

```

# (ok, proc)= fclose file2 proc
# proc= appPIO (setControlText t2
  ("(* Conteudo t2 gravado com sucesso *)"+++
    {toChar 13,toChar 10}+++ {toChar 13,toChar 10}
    +++ x))proc
=(el,proc)
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//SALVAR ARQUIVO texto3 ATRAVES DO EXPLORER
  salvar3(el,proc)
# (Just dial,proc) = accPIO(getWindow wd1) proc
# (se,Just x) = getControlText t3 dial
# (talvez, proc) = selectOutputFile "Salvar o resultado: escolha o diretorio!" "*.txt" proc
| isNothing talvez= (el,proc)
# (ok,file2, proc)= fopen (fromJust talvez) FWriteText proc
| not ok = (el,proc)
# file2 = fwrites x file2
# (ok, proc)= fclose file2 proc
# proc= appPIO (setControlText t3
  ("(* Conteudo t2 gravado com sucesso *)"+++
    {toChar 13,toChar 10}+++ {toChar 13,toChar 10}
    +++ x))proc
=(el,proc)
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//mesclar t1 e t2
  mesclar(el,proc)
# (Just dial,proc) = accPIO(getWindow wd1) proc
# (_,Just x) = getControlText t1 dial
# (_,Just y) = getControlText t2 dial
# proc= appPIO (setControlText t3 (x+++saltaLinha+++y))proc
=(el,proc)

//Função externa para saltar linha (\ = toChar 13) e (n = toChar 10)
saltaLinha = {toChar 13,toChar 10}

```

A partir dos conceitos e ferramentas apresentadas neste capítulo, fica mais fácil o entendimento do código gerado no aplicativo desta tese, a aderência da linguagem ao problema proposto, bem como facilita a um usuário leigo nesta linguagem iniciar seus primeiros trabalhos, com interface visual ou não, implementando alguns aplicativos em diversas áreas do conhecimento. Para maiores detalhes, indica-se o banco de teses e dissertações da FEELT – UFU, a qual possui informações e trabalhos relevantes utilizando este paradigma, o funcional, e a linguagem CLEAN.

APÊNDICE 1-

FONOLOGIA E FONÉTICA

Extraído do endereço:

<http://images.google.com.br/imgres?imgurl=http://criarmundos.do.sapo.pt/Linguistica/images/aparelho-fonador.gif&imgrefurl=http://criarmundos.do.sapo.pt/Linguistica/pesquisalinguistica02.html&h=358&w=400&sz=11&tbnid=Lgoeo4WFI76VkM:&tbnh=107&tbnw=120&hl=pt-BR&start=1&prev=/images%3Fq%3D%2522aparelho%2Bfonador%2522%26svnum%3D10%26hl%3Dpt-BR%26lr%3D%26sa%3DG>

e registrado aqui para consulta, caso o site saia do ar ou mude de endereço.

Fonologia e Fonética

Na construção de uma língua é preciso, em primeiro lugar, pensar em fonologia e fonética, ou seja, saber o que são e como tratar os sons. Então e qual é a diferença entre fonologia e fonética? Bom, a fonologia estuda o comportamento dos sons e dos fonemas numa língua, enquanto a fonética estuda os sons e os fonemas (incluindo a sua evolução).

Claro que, antes de estudarmos os sons e os seus comportamentos, é preciso saber como são os sons produzidos. Afinal, quem quiser inventar uma língua extraterrestre tem de pensar no modo como os seus extraterrestres produzem sons.

O Aparelho Fonador e o seu Funcionamento

Para que se produzam os sons que caracterizam a fala humana são necessárias três condições:

corrente de ar;

obstáculo à corrente de ar;

caixa de ressonância;

o que se traduz no **aparelho fonador** humano:

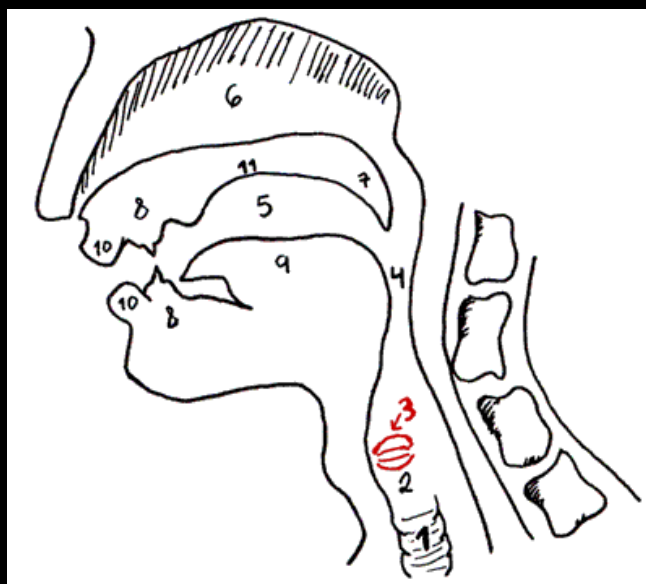
Os pulmões, brônquios e traqueia - São os órgãos respiratórios que permitem a corrente de ar, sem a qual não existiriam sons. A maioria dos sons que conhecemos são produzidos na expiração, servindo a inspiração como um momento de pausa; no entanto, há línguas que produzem sons na inspiração, como o zulo e o boximane - são os chamados cliques.

A laringe, onde ficam as cordas vocais - Determinam a sonoridade (a vibração das cordas vocais) dos sons.

A faringe, boca (e língua) e as fossas nasais - Formam a caixa de ressonância responsável por grande parte da variedade de sons.

Olhemos por um momento para o esquema do aparelho fonador antes de seguir o percurso do ar na produção de sons.

Esquema do Aparelho Fonador



1 - Traqueia

2 - Laringe

3 - Glote (Cordas vocais)

4 - Faringe

5 - Cavidade bucal

6 - Cavidade nasal

7 - Véu palatino ou Palato mole

8 - Maxilares (dentes)

9 - Língua

	10 - Lábios
	11 - Palato duro (céu da boca)

Ao expirar, os pulmões libertam ar que passa pelos brônquios para entrar na **traqueia(1)** e chegar à **laringe(2)**. Na laringe o ar encontra o seu primeiro obstáculo: a **glote(3)** (mais ao menos ao nível da maçã-de-adão, chamada de *gogó* no Brasil), mais conhecida como cordas vocais. Semelhantes a duas pregas musculares, as cordas vocais podem estar fechadas ou abertas: se estiverem abertas, o ar passa sem real obstáculo, dando origem a um **som surdo**; se estiverem fechada, o ar força a passagem fazendo as pregas muscular vibrar, o que dá origem a um **som sonoro**.

Para se perceber melhor a diferença, experimente-se dizer "k" e "g" (não "kê" ou "kapa", nem "gê" ou "jê"; só os sons "k" e "g") mantendo os dedos na maçã-de-adão. No primeiro caso não se sentirá vibração, mas com o "g" sentir-se-á uma ligeira vibração - cuidado apenas para não se dizerem vogais, pois são todas sonoras.

Depois de sair da **laringe(2)**, o ar entra na **faringe(4)** onde encontra uma encruzilhada: primeiro a entrada para a **boca(5)** e depois a para as **fossas nasais(6)**. No meio está o **véu palatino(7)** que permite que o ar passe livremente pelas duas cavidades, originando um **som nasal**; ou que impede a passagem pela cavidade nasal, obrigando o ar a passar apenas pela cavidade bucal - resultando num **som oral**.

A diferença é óbvia: compare-se o primeiro "a" em "Ana" com o de "manta". A primeira vogal é oral e a segunda é nasal.

Por fim, o ar está na cavidade bucal (a boca) que funciona como uma caixa de ressonância onde, usando os **maxilares(8)**, as bochechas e, especialmente, a **língua(9)** e os **lábios(10)**, podem modular-se uma infinidade de sons.

A título de curiosidade, gostaria apenas de recordar um pouco a história do Homem. Discute-se que a linguagem humana pode ter surgido há cerca de 100 mil anos, mas pensemos numa época mais recente - há cerca de 40 mil anos. Nesta altura, e devido a reconstruções tendo por base o registo arqueológico, sabe-se que o aparelho fonador dos Neandertais tinha algumas diferenças marcantes do Homem moderno, nomeadamente, a laringe encontrava-se mais elevada. Isto significa que a língua tinha uma mobilidade menor, limitando a possibilidade da produção de sons.

Som e Fonema - Transcrições

Bom, até aqui já vimos como são os sons produzidos de um modo básico. Mas muitas questões estão ainda por resolver: por exemplo, qual a diferença entre um "p" e um "k"? Onde e como são estes sons produzidos? A resposta, no entanto, tem de ser um pouco

adiada. Primeiro é preciso estabelecer algumas noções relativas aos sons e à sua transcrição para que uns não falem de "alhos" e outros entendam "bugalhos"!

Para começar é preciso distinguir som de fonema. Se todos sabemos o que é um som (ainda agora mesmo vimos como se produzem!), então o que é um fonema? Um fonema é um elemento de significado, o mais pequeno que existe numa palavra - e que quase se pode confundir com um som! Repare-se nas seguintes palavras:

saco

taco

Se não fosse pelo "s" e "t" iniciais, as palavras não se distinguiriam. Assim, tratam-se de duas unidades - representadas fisicamente pelo som (tornam-se audíveis) - que representam uma idéia. E como se distinguem sons de fonemas? Porque o som é representado entre [parêntesis rectos] e o fonema entre /barras/, enquanto as letras são representadas entre "aspas". Concluindo: nas palavras "saco" e "taco" os sons [s] e [t], representados pelas letras "s" e "t", correspondem aos fonemas /s/ e /t/. No entanto, o fonema /s/ pode também ser escrito com "ss" ("assado"), com "ç" ("aço"), com "c" ("cerca"), ou com "x" ("próximo"); podendo ser realizado quer com o som [s], no português normal, quer com o som [s̺], em certas regiões do Norte de Portugal e da Galiza.

Agora vem um outro problema: como é que se sabe que som é qual quando se escreve [a]? Será o [a] de "àrvore" ou de "cana"? Sabe-se que é o [a] de "àrvore" porque existe um **alfabeto fonético internacional**, que convencionou os símbolos que representam cada som e fonema. (Apesar de poder haver algumas interpretações ligeiramente diferentes dos símbolos de língua para língua.)

E, como não há memória que consiga reter tão grande lista, apresentamo-la numa página à parte de modo que possa ficar sempre à mão à medida que for lendo este capítulo:

Alfabeto Fonético Internacional

Nota 1: Este quadro apresenta os sons característicos da língua portuguesa, apresentando ainda alguns outros sons usados (dialectais e não só).







*Nota 2: Cada som é dado com exemplos, sendo que o som correspondente será marcado nos exemplos a **cor**.*

Vogais

[a]	Como em "águas"; produzido com a língua numa posição de repouso e sem
-----	---

	elevação do seu dorso, sem arredondamento dos lábios e boca ligeiramente aberta.
[ɑ]	Como em "cana"; produzido com a língua numa posição de repouso e com o seu dorso um pouco elevado, em comparação ao [a], sem arredondamento dos lábios e boca ligeiramente fechada.
[ɛ]	Como em "pé"; produzido com a língua elevada em direcção ao palato duro (céu da boca) e com o seu dorso ligeiramente elevado, sem arredondamento dos lábios e boca ligeiramente aberta.
[e]	Como em "medo"; produzido com a língua elevada em direcção ao palato duro (céu da boca) e com o seu dorso um pouco elevado, sem arredondamento dos lábios e boca ligeiramente fechada.
[ɐ]	Como em "sede"; produzido com a língua numa posição de repouso e com o seu dorso elevado, sem arredondamento dos lábios e boca quase fechada.
[ɔ]	Como em "cola"; produzido com a língua elevada em direcção ao véu palatino e com o seu dorso ligeiramente elevado, com arredondamento dos lábios e boca ligeiramente aberta.
[o]	Como em "bolo"; produzido com a língua elevada em direcção ao véu palatino e com o seu dorso um pouco elevado, com arredondamento dos lábios e boca ligeiramente fechada.
[i]	Como em "pilha"; produzido com a língua elevada em direcção ao palato duro (céu da boca) e com o seu dorso elevado, sem arredondamento dos lábios e boca ligeiramente fechada.
[u]	Como em "sul"; produzido com a língua elevada em direcção ao véu palatino e com o seu dorso elevado, com arredondamento dos lábios e boca quase fechada.
Semivogais	
[j]	Como em "praia"
[w]	Como em "pau"
Consoantes	
[b]	Como em "ambos"; os lábios tocam-se obstruindo a passagem do ar, as cordas vocais vibram.

[b]	Como em "boi"; os lábios tocam-se apenas muito ligeiramente obstruindo a passagem do ar, as cordas vocais vibram.
[d]	Como em "andar"; a parte imediatamente anterior à ponta da língua toca a parte interior dos dentes incisivos do maxilar superior obstruindo a passagem do ar, as cordas vocais vibram.
[ð]	Como em "espada"; a parte imediatamente anterior à ponta da língua mal toca a parte interior dos dentes incisivos do maxilar superior obstruindo a passagem do ar, as cordas vocais vibram.
[g]	Como em "frango"; a parte posterior da língua toca o palato mole, ou véu palatino, obstruindo a passagem do ar, as cordas vocais vibram.
[ɣ]	Como em "agrado"; a parte posterior da língua mal toca o palato mole, ou véu palatino, obstruindo a passagem do ar, as cordas vocais vibram.
[p]	Como em "pata"; os lábios tocam-se obstruindo a passagem do ar, as cordas vocais não vibram.
[t]	Como em "atado"; a parte imediatamente anterior à ponta da língua toca a parte interior dos dentes incisivos do maxilar superior obstruindo a passagem do ar, as cordas vocais não vibram.
[tʃ]	Como em "chave" nalgumas zonas do Norte de Portugal, e como em "tchau"; a parte imediatamente anterior à ponta da língua toca a parte interior dos dentes incisivos do maxilar superior obstruindo a passagem do ar, e depois a língua desliza depressa para trás, forçando o ar a passar por uma fenda estreita entre o dorso da língua e o palato duro, ou céu da boca, as cordas vocais não vibram.
[k]	Como em "porco"; a parte posterior da língua toca o palato mole, ou véu palatino, obstruindo a passagem do ar, as cordas vocais não vibram.
[m]	Como em "arma"; os lábios tocam-se obstruindo a passagem do ar, as cordas vocais vibram e é uma consoante nasal.
[n]	Como em "cano"; a ponta da língua toca os alvéolos no maxilar superior obstruindo a passagem do ar, as cordas vocais vibram e é uma consoante nasal.
[ɲ]	Como em "vinha"; o dorso da língua toca o palato duro, ou céu da boca, obstruindo a passagem do ar, as cordas vocais vibram e é uma consoante nasal.
[l]	Como em "calo"; a ponta da língua toca os alvéolos no maxilar superior permitindo a passagem do ar lateralmente, as cordas vocais vibram.
[ɫ]	Como em "mel"; a ponta da língua mal toca os alvéolos no maxilar superior

	permitindo a passagem do ar lateralmente, as cordas vocais vibram.
	Como em "alho"; o dorso da língua toca o palato duro, ou céu da boca, forçando o ar a passar lateralmente, as cordas vocais vibram.
[r]	Como em "caro"; a ponta da língua toca, vibrando, os alvéolos no maxilar superior, as cordas vocais vibram.
	Como em "caro" nalgumas zonas de Portugal; a ponta da língua toca, vibrando, os alvéolos no maxilar superior interrompendo por diversas vezes a passagem do ar, as cordas vocais vibram.
[R]	Como em "carro"; a parte posterior da língua toca, vibrando, no palato mole, ou véu palatino, interrompendo por diversas vezes a passagem do ar, as cordas vocais vibram.
[f]	Como em "faca"; o ar é forçado a passar por entre os dentes incisivos do maxilar superior e o lábio inferior, as cordas vocais não vibram.
[v]	Como em "vaca"; o ar é forçado a passar por entre os dentes incisivos do maxilar superior e o lábio inferior, as cordas vocais vibram.
[s]	Como em "posso"; a parte imediatamente anterior à ponta da língua aproxima-se da parte interior dos dentes incisivos do maxilar superior formando uma passagem estreita (como uma fenda) à passagem do ar, as cordas vocais não vibram.
	Como em "posso" nalgumas zonas Norte de Portugal; a ponta da língua aproxima-se da parte interior dos dentes incisivos do maxilar superior formando uma passagem estreita (como uma fenda) à passagem do ar, as cordas vocais não vibram.
[z]	Como em "casa"; a parte imediatamente anterior à ponta da língua aproxima-se da parte interior dos dentes incisivos do maxilar superior formando uma passagem estreita (como uma fenda) à passagem do ar, as cordas vocais vibram.
	Como em "casa" nalgumas zonas Norte de Portugal; a ponta da língua aproxima-se da parte interior dos dentes incisivos do maxilar superior formando uma passagem estreita (como uma fenda) à passagem do ar, as cordas vocais vibram.
	Como em "acho"; o ar é forçado a passar por uma fenda estreita entre o dorso da língua e o palato duro, ou céu da boca, as cordas vocais não vibram.
	Como em "genro"; o ar é forçado a passar por uma fenda estreita entre o dorso da língua e o palato duro, ou céu da boca, as cordas vocais vibram.

A Classificação dos Sons Linguísticos

Para a classificação dos sons é preciso ter em mente três questões importantes:

- Como é que os sons são produzidos?
- Como são transmitidos?
- Como são entendidos?

Tradicionalmente, devido à complexidade óbvia na classificação segundo a transmissão e a compreensão, a classificação dos sons baseia-se essencialmente no modo como os sons são produzidos, ou seja, na sua articulação. No entanto, em alguns pontos classificatórios também se baseia no modo como são transmitidos, ou seja, na acústica. Como este capítulo não pretende ser exaustivo, mas ajudar quem não tem conhecimentos neste campo, tentarei ser o mais simples e clara que for possível (mesmo que, para isso, simplifique demais a gramática).

Os sons classificam-se segundo três categorias:

Vogais: os sons produzidos sem obstáculo à passagem do ar na cavidade bucal (apenas varia a abertura à passagem do ar causada pelos maxilares, língua e lábios), e com vibração das cordas vocais.

Consoantes: os sons produzidos com obstáculo à passagem do ar na cavidade bucal.

Semivogais: dois sons, [j] e [w], que formam uma sílaba com uma vogal – ditongos e tritongos. Pode-se dizer que são quase "formas fracas" de [i] e [u], estando a meio-caminho entre vogais e consoantes.

Classificação das Vogais

As vogais da língua portuguesa podem ser classificadas quanto:

- à região de articulação (ver o esquema do aparelho fonador)
- palatais ou anteriores (língua elevada na zona do **palato duro(11)**)
- centrais ou médias (língua na posição de descanso)
- velares ou posteriores (língua elevada na zona do **véu palatino(7)**)
- ao grau de abertura (elevação do dorso da língua em direcção ao palato)
- abertas (o maior grau de abertura à passagem do ar)
- semi-abertas
- semi-fechadas
- fechadas (o menor grau de abertura à passagem do ar)
- ao arredondamento ou não dos lábios
- arredondadas
- não-arredondadas
- ao papel das cavidades bucal e nasal
- orais
- nasais

De acordo com esta caracterização pode-se preencher o quadro abaixo (em que a nasalidade é marcada pelo til como, por exemplo, em [ã])

	Palatais	Médias	Posteriores
Fechadas	[i] [ĩ]	[e]	[u] [ũ]
Semi-fechadas	[e] [ẽ]	[ɐ] [ã]	[õ] [o]
Semi-abertas	[ɛ]		[ɔ]
Abertas		[a]	
	Não-arredondadas	Não-arredondadas	Arredondadas

Classificação das Consoantes

As dezanove consoantes da língua portuguesa podem ser classificadas quanto:

ao modo de articulação (o ar encontra sempre obstáculo à sua passagem - ver o esquema da cavidade bucal)

oclusivas (passagem do ar interrompida momentaneamente)

constritivas (passagem do ar parcialmente obstruída)

fricativas (passagem do ar por uma fenda estreita no meio da via bucal; som que lembra o de fricção)

laterais (passagem do ar pelos dois lados da cavidade bucal, pois o meio encontra-se obstruído de algum modo)

vibrantes (caracterizadas pelo movimento vibratório rápido da língua ou do véu palatino)

ao ponto ou zona de articulação (o local onde é feita a obstrução à passagem do ar)

bilabiais (contacto dos lábios superior e inferior)

labiodentais (contacto dos dentes do maxilar superior com o lábio inferior)

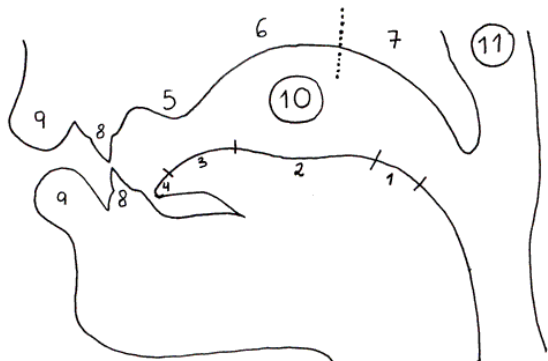
linguodentais (aproximação ou contacto da zona anterior à ponta da língua com a face interior dos dentes do maxilar superior)

alveolares (contacto da ponta da língua com os alvéolos no maxilar superior)

palatais (contacto do dorso da língua com o palato duro, ou céu da boca)

velares (contacto da parte posterior da língua com o palato mole, ou véu palatino)

ao papel das cordas vocais**surdas** (ausência de vibração das cordas vocais)**sonoras** (vibração das cordas vocais)**ao papel das cavidades bucal e nasal****orais** (passagem do ar apenas pela cavidade bucal)**nasais** (passagem do ar pelas cavidades bucal e nasal)

Esquema da Cavidade Bucal	
	1 - Parte posterior da língua 2 - Dorso da língua 3 - Pré-dorso da língua 4 - Ápice ou ponta da língua 5 - Alvéolos 6 - Palato duro (céu da boca) 7 - Véu palatino ou Palato mole 8 - Dentes 9 - Lábios 10 - Cavidade bucal 11 - Passagem para a cavidade nasal

Através desta classificação pode-se preencher o seguinte quadro das 19 consoantes portuguesas:

Papel das cavidades bucal e nasal	Orais						Nasais
	Oclusivas		Fricativas		Laterais	Vibrantes	Oclusivas
Papel das Cordas Vocais	Surd	Son	Surd	Son	Son	Son	Son

Bilabiais	[p]	[b]					[m]
Labiodentais			[f]	[v]			
Linguodentais	[t]	[d]	[s]	[z]			
Alveolares					[l]	[r]	[n]
Palatais			[ɲ]	[ʃ]	[ʒ]		[ɲ]
Velares	[k]	[g]				[R]	

Gostaria ainda de fazer uma nota quanto ao número de 19 consoantes que foi acima referido, pois este número não contempla certas variantes (como o [tʃ] ou o [dʒ]), nem as limitações que a língua impõe. Neste último caso, como em todas as línguas, existem algumas proibições quanto à posição de certas consoantes no início ou final de palavra, assim como em seguimento de certas palavras. Por exemplo, [r] nunca pode surgir em início de palavra.

Encontros vocálicos - Ditongos e Tritongos
Encontros vocálicos é o mesmo que dizer **ditongo** ou **tritongo**, ou seja, um conjunto de uma vogal e uma ou duas semivogais - que é a única altura em que surgem semivogais no português. Não devem, portanto, ser confundidos com **hiatos**: o encontro de duas vogais.

Os ditongos podem ser crescentes (pouco comum, pois são instáveis) ou decrescentes, consoante a vogal esteja no final ou no início do ditongo:

[kwal] - "qual"

[paj] - "pai"

E podem ser orais ou nasais:

[paj] - "pai"

[maʊ] - "mau"

[boj] - "boi"

[põj] - "põe"

[mãj] - "mãe"

[mãw] - "mão"

Estes exemplos foram todos escolhidos para ajudar a exemplificar a diferença entre ditongo e hiato. Se se reparar, todos estes ditongos correspondem a uma única sílaba, mas os hiatos formam duas sílabas. Observe-se os dois exemplos em comparação:

Ditongo
[paj] - "pais"

Hiato
[pa'is] - "pa-ís"

Mas uma língua é um organismo vivo, e as pessoas dizem as coisas de modos diferentes consoante a situação em que se encontram - são estes pormenores que fazem uma língua evoluir e modificar-se mais depressa. Assim, um hiato pode passar a ditongo, se dito

muito depressa; e um ditongo pode passar a hiato se for dito pausadamente de modo a salientar bem todos os sons:

[lu ' ar] - "lu-ar" -->
[saw ' da ' dɐ] - "sau-da-de" -->

[lwar] - "luar"
[sa ' u ' da ' dɐ] - "sa-u-da-de"

Por fim, os tritongos são formados por uma semivogal, uma vogal e outra semivogal, podendo ser orais ou nasais:

[urugwaj] - "Uruguai"
[ẽɪ agw αj] - "enxaguei"

[sagw ẽ w] - "saguão"
[dɐl ẽ kw ẽ j] - "delinquem"

Encontros

consonânticos

É o nome que se dá a um agrupamento de consoantes. Os agrupamentos mais comuns são aqueles em que a segunda consoante é "l" ou "r", embora em alguns casos não surjam no início da palavra:

bloco

dragão

abluir
atlas
vidro
palavra

Outros agrupamentos são mais raros, como os que se seguem:

Gnomo
pneu
apto

mnemónico
psicológico
digno

Nestes agrupamentos, as consoantes pertencem sempre a uma única sílaba. No entanto, quando se encontram no meio da palavra podem pertencer a duas sílabas. Por outro lado, por vezes a língua ao evoluir começa a "considerar" estes agrupamentos como "incómodos" e introduz uma vogal. Veja-se os exemplos abaixo:

a - **pto**
ap - to
a - **pe** - to

di - **gno**
díg - no
di - **gui** - no

Por fim, é preciso um pouco de atenção para não confundir consoantes com letras; evitando, assim, confundir os encontros consonantais com **dígrafos**. Ou seja, um encontro consonantal é um grupo de dois sons consonânticos - [pn] e [kl], por exemplo - enquanto um dígrafo é um grupo de duas letras que representam um som - "rr" representa o [R], por exemplo.

O mais importante a ter em mente relativamente aos encontros vocálicos e consonantais, é que a língua estabelece regras que impedem o "encontro" entre certos sons e em certas posições dentro de uma palavra.

As Sílabas

A sílaba é um som ou conjunto de sons que podem ser ditos numa só expiração. Ou seja, se se disser uma palavra devagar, ninguém dirá:

a - l - u - n - o

Afinal, isso seria soletrar. Qualquer pessoa dirá:

a - l u - n o

Portanto, vamos aos factos:

Uma sílaba forma-se tendo por base uma vogal ou ditongo (ou tritongo), podendo ser ou não rodeada de consoantes. Vejamos alguns exemplos:

i - na - cen - tu - a - dos;

trans - por;

As sílabas podem ser **abertas** ou **fechadas**:

a - ca - ma - do, as quatro sílabas são abertas pois acabam em vogais;

trans - por - tar, as três sílabas são fechadas pois acabam em consoantes;

Por fim, as palavras são classificadas quanto ao número de sílabas que as constituem:

as palavras **monossílabas** possuem apenas uma sílaba, como em **mão**

as palavras **dissílabas** possuem duas sílabas, como em **trans - por**

as palavras **trissílabas** possuem três sílabas, como em **trans - por - tar**

as palavras **polissílabas** possuem mais de três sílabas, como em **a - ca - ma - do**, **i - na - cen - tu - a - dos** ou em **o - to - rri - no - la - rin - go - lo - gis - ta**

A noção de sílaba pode parecer pouco importante, mas é uma base essencial para se compreender muitas noções de que falaremos mais à frente, desde a acentuação das palavras à sintaxe e morfologia.

Acento Tónico e Outros Tipos de Acento

As palavras são, portanto, constituídas por sílabas, mas estas não têm todas o mesmo valor. O que quero dizer? Tomemos a palavra "árvore" como exemplo, dizemos:

ár - vo - re

e não:

ar - **vó** - re ou ar - vo - **ré**

Ou seja, **acentuamos** a primeira sílaba enquanto as restantes não são acentuadas. esta acentuação faz-se por meio de uma maior ou menor utilização de certas características do nosso aparelho fonador:

Intensidade - a força com que uma sílaba é expirada

Os sons podem ser **fortes** (ou seja, tónica)

Os sons podem ser **fracos** (ou seja, átona)

Tom - (também chamado de altura musical) a frequência com que as cordas vocais vibram na altura em que se produz uma sílaba

Os sons podem ser **agudos** (ou seja, altos)

Os sons podem ser **graves** (ou seja, baixos)

Timbre - (também chamado de metal da voz) o tom principal, dos sons que são produzidos, ressoa nas cavidades (bucal e nasal) por onde passa o ar expirado (a posição da língua e a abertura da boca são factores importantes) produzindo tons secundários - é ao conjunto destes tons (o principal e os secundários) que se dá o nome de timbre

Os sons podem ser **abertos**

Os sons podem ser **fechados**

Quantidade - a duração com que os sons são emitidos

Os sons podem ser **longos**

Os sons podem ser **breves**

Esta classificação, claro, está incompleta: uma língua pode fazer a distinção de até 4 (Mandarim) ou 8 tons (Cantonês), por exemplo. Do mesmo modo, a quantidade longa ou breve pode não ter valor distintivo, como acontece nas vogais portuguesas em que a quantidade surge apenas no âmbito de acentuações de insistência ou ênfase.

Tom (exemplo do mandarim)	
fan (nível elevado) -	"vela"
fan (a subir) -	"embaraço"

Quantidade (exemplo do português)	
sim (breve) -	afirmação positiva

fan (a descer) -	"virar"	sim (longo) -	afirmação positiva (feita com enfado)
fan (queda) -	"arroz"		

Mas, pode agora alguém dizer, o que foi dito anteriormente sobre uma sílaba acentuada numa palavra não está correcto, pois algumas palavras apresentam mais do que uma sílaba acentuada. Claro que sim, do mesmo modo que essa acentuação pode mudar na mesma palavra consoante a frase em que está inserida, mas vamos com calma.

Como regra, todas as palavras possuem um **acento tónico**, ou seja, a sílaba é pronunciada de um modo forte; e **acentos átonos**, sílabas pronunciadas de modo fraco. Sendo as palavras classificadas consoante a sílaba em que recai o acento tónico:

palavras agudas ou oxítonas - quando o acento tónico se encontra na última sílaba (p. ex.: **café**; **funil**)

palavras graves ou paroxítonas - quando o acento tónico se encontra na penúltima sílaba (p. ex.: **escola**; **ditongo**)

palavras esdrúxulas ou proparoxítonas - quando o acento tónico se encontra na antepenúltima sílaba (p. ex.: **lâmina**; **quilómetro**)

palavras bisesdrúxulas - quando uma palavra é combinada com certos monossílabos átonos o acento pode recuar (p. ex.: **estudámo-lo**; **faça-se-lhe**)

monossílabos átonos - a única sílaba é pronunciada tão fracamente que se une a uma outra palavra, utilizando o acento tónico dessa palavra como apoio (p. ex.: **diga-me**; **o carro**)

monossílabos tónicos - a única sílaba é pronunciada de um modo forte (p. ex.: **flor**; **sim**)

Em alguns casos, a variação do acento tónico pode inclusivé marcar a diferença entre uma palavra e outra, um significado e outro, por exemplo: "**dú**vida" e "**du**vida", em que no primeiro caso temos o substantivo e no segundo temos o verbo duvidar (tanto pode ser a forma imperativa como o presente do Indicativo). Isto, no entanto, é mais comum na variante do português falado no Brasil pois, em Portugal, as vogais átonas tendem a sofrer um enfraquecimento do timbre, note-se (a sílaba tónica é identificada por um apóstrofo imediatamente antes - p.ex.: **diálogo** = [di'alugu]):

Português do Brasil		Português de Portugal	
correram	[ko'Rer ã w]	correram	[ku'Rer ã w]
correrão	[koRe'r ã w]	correrão	[kuR ə 'r ã w]

Como se referiu acima, as palavras não são compostas apenas por uma sílaba tónica rodeada das restantes átonas - especialmente as palavras longas, com atenção para as palavras derivadas, possuem um ou mais acentos que se encontram entre os "sons

fracos" e os "sons fortes". São as **sílabas subtónicas**, demasiado fortes para serem átonas, e não o suficiente para serem tónicas. Assim, repare-se:

decidida

Esta palavra possui um acento tónico na penúltima sílaba, que é, regra geral, a norma da acentuação tónica no português.

decididamente

Aqui, a palavra mantém o acento tónico na penúltima sílaba, mas permanece uma reminescência da acentuação da palavra original num acento subtónico (em itálico).

Para além destes três acentos - tónico, subtónico, átono - existem ainda os **acentos de insistência**, cuja função é a de realçar certa palavra de acordo com o contexto. São os acentos afectivo e intelectual.

O **acento afectivo** tem um carácter emocional, pois o acento é utilizado quando pretendemos demonstrar a nossa relação afectiva com uma dada situação:

Ele é um *miserável*! (sentimento de cólera ou desprezo)

Isto é *abominável*! (sentimento de cólera ou repulsa)

Olha que *amor*! (sentimento de afecto)

A palavra passa a ter duas sílabas acentuadas (em palavras pequenas o acento afectivo coincide com o acento tónico), sendo que o acento afectivo é quase tão forte como o acento tónico, podendo mesmo ultrapassá-lo. O mesmo efeito acontece com o **acento intelectual**, mas o recurso a este acento tem como função realçar uma noção ou caracterização:

Eu quero razões *objectivas*!

Isto não é *imoral*, é *amoral*!

Não quero razões *subjectivas*!

Então, em termos sonoros, qual a diferença entre o acento afectivo e o acento intelectual?

Acento Intelectual

O acento intelectual coincide sempre com a primeira sílaba, seja a palavra iniciada por consoante ou vogal.

Acento Afectivo

O acento afectivo coincide com a primeira sílaba se a palavra for iniciada por consoante.

Coincide com a segunda sílaba se a palavra for iniciada por vogal.

Coincide com o acento tónico se for

O acento intelectual aumenta a vogal em duração, altura e sobretudo em **intensidade**.

uma palavra pequena.

O acento afectivo aumenta a vogal em intensidade, mas sobretudo em **duração e altura**.

No entanto, como também já foi dito, a acentuação nas palavras pode ser modificada quando estas estão integradas numa frase. Afinal, ao dizermos uma frase estamos a articular e, por vezes, a fundir as palavras umas nas outras, sendo que a frase pode ser dividida em **grupos acentuais** ou de **intensidade**, cada qual apoiado num acento tónico. Vejamos um exemplo:

/ **Dias** / e **noites** / os **horizontes** / se **repetem**. /

Esta frase, quando dita pausadamente, é composta por quatro grupos acentuais, quase todos unindo uma palavra a um monossílabo átono. Mas, se dissermos os dois primeiros grupos depressa, estes fundem-se num só.

/ **Dias** e **noites** /

Aqui, o primeiro grupo acentual vê o seu centro tónico enfraquecer para um acento secundário, um subtónico. Assim, este tem de se integrar no grupo seguinte.

A estas situações de dependência chama-se **ênclise** (quando a primeira palavra mantém o acento tónico em detrimento da palavra seguinte, p. ex.: diga-**me**) e **próclise** (quando a primeira palavra perde o acento tónico tornando-se dependente da palavra seguinte, p. ex.: **os** horizontes). Esta perda de independência pode, inclusivé, resultar em alterações às palavras dependentes, que se vêem reduzidas. Observe-se alguns exemplos tão conhecidos da nossa oralidade:

Ele foi de férias **para as** Maldivas. - Ele foi de férias **prás** Maldivas.

Olha! É o **senhor António!** - Olha! É o **seu António**.

Esta é uma forma da língua evoluir em termos de vocabulário. Em português, são muitas as palavras que surgiram por próclise:

cento	-	cem
grande	-	grão
quanto	-	quão
santo	-	são
tanto - tão		