Tcl/Tk
Programação

SUMÁRIO

1 Introdução

A linguagem Tcl/Tk foi desenvolvida pelo Dr. John K. Ousterhout e sua equipe da Universidade da California, tendo sido mantida pela Sun Microsystems e posteriormente pela Scriptics e atualmente pela Ajuba Solutions.

A linguagem Tcl é essencialmente uma linguagem de script, podendo ser compilada para criar um executável em formato binário. Tk é o seu *Tool Kit*, sua biblioteca gráfica.

O uso mais comum para o Tcl é na criação de programa para os sistemas UNIX, Windows e Mac. Entretanto vem crescendo o número de empresas que utilizam a linguagem Tcl em seus servidores de Web, entre elas a America On-Line(AOL).

Tcl é uma linguagem poderosa, aliando um eficiente compilador JIT, a uma sintaxe extremamente simples. O compilador Tcl é tão eficiente, que há pouca diferença em velocidade, entre um programa Tcl interpretado e sua versão compilada.

1.1 O Interpretador Tcl

Os interpretadores Tcl mais usados são o wish e o tclsh. O wish é o interpretador para o X-Window, ou o Windows, enquanto o tclsh é o interpretador para o modo texto, sendo mais usado em scripts CGI.

Para executar um programa em Tcl digite na linha de comando:

```
$ wish nome_do_programa.tcl
```

ou

```
$ tclsh nome_do_programa.tcl
```

ou coloque na primeira linha do programa o comentário:

```
#!/usr/bin/wish
```

ou

```
#!/usr/bin/tclsh
```

e mude a permissão de acesso do programa para executável ou 755:

```
# chmod 755 nome_do_programa.tcl
```

Então, para executar o programa, simplesmente digite o nome do programa na linha de comando do shell.

Note que este procedimento se aplica somente ao UNIX. No Windows, simplesmente dê um duplo clique em um arquivo com a extensão *.TCL que o programa será executado.

De um modo geral o interpretador Tcl no Windows oferece mais recursos que no UNIX, entretanto, a versão UNIX é mais rápida.

1.2 Variáveis

Em Tcl só existe um tipo de dado: strings, ou cadeias de caracteres. Contudo, alguns comandos, especialmente os que realizam cálculos, interpretam seus arqumentos como valores numéricos ou booleanos. Nesse caso os dados estarão nos seguintes formatos:

Tipo	Exemplo
Inteiro	**123**(decimal), **0xFF**(hexadecimal), **0377**(octal)
Ponto flutuante	**1.2**, **2.**, **3e5**, **1.23e+5**
Booleano(lógico)	**true**, **false**, **0**, **1**, **yes**, **no**

Uma variável pode ter qualquer nome.

Para atribuir um valor a uma variável utiliza-se o comando set:

```
set x 0
```

O comando anterior atribui o valor 0(zero) à variável x.

Para ler o valor de uma variável basta colocar o caractere $, antes do nome da variável:

```
puts $x
```

Exibe o valor da variável x no console.

Você não precisa declarar variáveis, elas são criadas à medida em que são referenciadas.

1.3 Substituições

Quando você passa argumentos para um comando ou atribui uma string a uma variável, Tcl primeiro interpreta o conteúdo da string passada e substitui as chamadas de funções por seus valores de retorno. Por exemplo, ao executar o comando abaixo, Tcl primeiro avaliará a string, expandindo-a, e em seguida passará a string resultante para o comando puts:

```
puts "O valor de 2 + 4 é [expr 2 + 4]"
```

Ao ler e expandir a expressão Tcl terá como resultado:

```
puts "O valor de 2 + 4 é 7"
```

e no console será exibido:

```
O valor de 2 + 4 é 7
```

Quando Tcl realiza substituções, algumas sequências de caracteres são interpretadas de forma especial. São elas:

Sequência	Significado
\a	Um beep(0x7)
\b	Backspace(0x8)
\f	Avanço de página(0xC)
\n	Nova linha(0xA)
\r	Retorno de carro(0xD)
\t	Tabulação horizontal(0x9)
\v	Tabulação vertical(0xB)
\space	Espaço
\newline	Espaço
\ddd	Um valor octal(d=0-7)
\xdd	Um valor hexadecimal(d=0-9, a-f, A-F)
\c	o próprio caractere c
\\	A própria barra \

É possível representar cadeias de caracteres entre aspas("), apóstrofos('), ou chaves({}), entretanto, Tcl não fará nenhuma substituição caso a cadeia de caracteres esteja entre chaves. Por exemplo:

```
puts {O valor de 2 + 4 é [expr 2 + 4]}
```

produzirá como saída no console:
O valor de 2 + 4 é [expr 2 + 4]}
Sem realizar nenhuma substituição.

1.4 Blocos de comandos

Pode-se agrupar comandos em blocos, colocando-se todos os comando entre chaves:

```
if {$x == 1}  {
   set y 1
   set z 2
}
```

O comando anterior execurá o bloco {set y 0; set z 2} caso o conteúdo da variável x seja 1.

1.5 Operadores

Os operadores matemáticos e lógicos suportados por Tcl são os seguintes, em órdem de precedência:

Operador	Significado
- ~ !	Negativo, Não bit a bit, Negação lógica
* / %	Multiplicação, divisão, parte inteira da divisão
<< >>	Deslocamento de bits à esquerda, Deslocamento de bits à direita
< > <= >=	Menor, Maior, menor ou igual, maior ou igual
== !=	Igual(comparação lógica), diferente
&	E(AND) bit a bit
^	OU EXCLUSIVO(XOR) bit a bit
|	OU(OR) bit a bit
&&	E lógico(AND)
||	OU lógico(OR)
x ? y : z	Se x então y, caso contrário, z

Tcl também suporta as seguintes funções matemáticas através do comando expr:

Função	Significado
abs	Módulo
acos	Arco-cosseno
asin	Arco-seno
atan	Arco-tangente
atan2	Arco-tangente
ceil	Arredonda para maior
cos	Cosseno
cosh	Cosseno hyperbólico
double	Transforma em um número de dupla precisão
exp	Calcula o número e elevado a x
floor	Arredonda para menor
fmod	Calcula o resto de uma divisão
hypot	Calcula a hypotenusa de um triângulo retângulo
int	Converte em inteiro
log	Calcula o logarítimo natural de x
log10	Calcula o logarítimo na base 10 de x
pow	x elevado a y

rand	Gera um número aleatório
round	Aredonda um número
sin	Seno
sinh	Seno hyperbólico
sqrt	Raiz quadrada
tan	Tangente
tanh	Tangente hyperbólica

Para realizar uma operação matemática basta passar a expressão para o comando expr:
puts [expr 2 + 3 * (1 + sin(32))]
Exibirá no console:
6.65428004373

1.6 Exibindo saída no console

Para exibir saída no console, utiliza-se o comando puts:
puts "Alô Mundo!\n"
Exibira no console:
Alô Mundo!

1.7 Comentários

Comentários em Tcl começam com o caractere sustenido(#). Comentários podem vir em qualquer parte do programa, contudo, se vierem no final de uma linha de código, deve ser precedidos do um caractere ponto-e-vírgula(;):
puts "Esta linha contém um comentário\n"; # Este é um comentário
puts "Esta linha contém um comentário incorreto\n" # Este comentário não foi precedido do sial (;) e gerará um erro
Este comentário é válido
puts "OK!"

1.8 Pontuação

Todos os comandos em Tcl devem ser terminados por um ponto-e-vírgula ou por uma nova linha:
puts "Esta é uma linha\n"; puts "Esta é uma outra linha\n"
é equivalente a:
puts "Esta é uma linha\n"
puts "Esta é uma outra linha\n"

1.9 Exercícios

Escreva um programa em Tcl que calcule a hypotenusa de um triângulo retângulo com catetos 4 e 5.
Escreva um programa que calcule a média entre os números 12 e 46.

1.10 Respostas

puts "A hypotenusa do triângulo retângulo cujos catetos são 4 e 5 é [expr hypot(4,5)]"
puts "A média dos números 12 e 46 é [expr (12 + 46) / 2]"

2 Cadeias de caracteres

Em Tcl existe apenas um tipo de dado: strings. Strings são cadeias de caracteres, que podem ser delimitadas por aspas duplas("), aspas simples(') ou chaves({}). O delimitador da string determina se haverá ou não substituição, como visto na aula de introdução deste curso.

Para atribuir uma string a uma variável, utilizamos o comando set:

```
set x "Esta é uma string"
```

Para ler o conteúdo da variável colocamos um caractere $ antes do nome da variável:

```
puts $x
```

Tcl oferece diversos comandos e funções para tratar strings:

Comando/Função	Descrição
append	Concatena várias strings.
binary format	Retorna uma representação binária de uma string, de acordo com um formato especificado.
binary scan	Estrai valores de uma string binária, para uma variável, de acordo com o formato especificado.
format	Retorna uma string formatada, de modo semelhante ao comando sprintf do ANSI C.
regexp	Retorna 1 se a *expressão regular* corresponde à string.
regsub	Substitue a primeira porção da string que corresponde à *expressão regular*.
scan	Extrai valores de uma string, em uma variável, de modo semelhante ao comando sscanf do ANSI C.
string compare	Compara duas strings.
string first	Encontra a primeira ocorrencia de uma string em outra.
string index	Retorna o caractere na posição especificada.
string is	Verifica se os dados em uma string são de um determinado tipo.
string last	Encontra a última ocorrência de uma string em outra.
string length	Retorna o tamanho da string.
string match	Verifica se uma string corresponde a outra dada.
string range	Retorna um trecho de uma string.
string tolower	Converte uma string em minúsculas.
string toupper	Converte uma string em maiúsculas.
string trim	Remove os espaços em branco à direita e à esquerda de uma string.
string trimleft	Remove os espaços em branco à esquerda de uma string.
string trimright	Remove os espaços em branco à direita de uma string.
string wordend	Retorna o caractere logo após o último caractere na palavra especificada na string.
string wordstart	Retorna o primeiro caractere na palavra especificada na string.
subst	Realiza uma substituição em uma string.

Exemplos:

append
```
set x "1"
append x "234"
puts $x
```
Resulta em:
1234

string first
```
set x "abc def ghi"
set y [string first "e" $x]
puts $y
```
Resulta em:
5

string length
```
set x "abcdefgh"
puts [string length $x]
```
Resulta em:
8

string tolower
```
set x "LETRAS MAIÚSCULAS"
puts [string tolower $x]
```
Resulta em:
letras maiúsculas

string trim
```
set x "    abc    "
puts "[string trim $x] def"
```
Resulta em:
abc def

string trimright
```
set x "    abc    "
puts "[string trimright $x] def"
```
Resulta em:
abc def

format
```
set x [format {%-20s %3.2f %2d} Salário 2.1 3]
puts $x
```
Resulta em:
Salário 2.10 3

string index
```
set x "123 abc 456 def"
puts [string index $x 5]
```
Resulta em:
b

string range
```
set x "abcdefgh"
puts [string range $x 3 6]
```
Resulta em:
defg

string toupper
```
set x "letras minúsculas"
puts [string toupper $x]
```
Resulta em:
LETRAS MINÚSCULAS

`string trimleft`
```
set x "    abc    "
puts "[string trimleft $x] def"
```
Resulta em:
abc def

Para uma descrição detalhada de todas as funções string, consulte a documentação on-line, ou o Tcl/Tk Reference Guide, ou ainda o Tcl/Tk Electronic Reference.

2.1 Expressões regulares

Expressões regulares são um poderoso meio para realizar pesquisas em strings. A tabela a seguir, apresenta os operadores utilizados em Tcl para realizar buscas em strings, vetores e listas:

Operador	Descrição
expreg|expreg	Compara ambas as expressões.
expreg*	Compara zero ou mais ocorrências de expreg.
expreg+	Compara uma ou mais ocorrências de expreg.
expreg?	Compara zero ou uma ocorrência de expreg.
.	Compara um único caractere.
^	Compara o início da string.
$	Compara o fim da string.

\c	Compara o caractere c mesmo que ele seja especial.
[abc]	Compara o conjunto de caracteres.
[^abc]	Compara os caracteres que não estejam no conjunto.
[a-z]	Compara a faixa de caracteres.
[^a-z]	Compara os caracteres que não estejam na faixa.
( )	Agrupa expressões.

Exemplos

```
puts [regexp {[1-5]} "12345678"]
```

Retorna 1, pois um dos números de 1 a 5 está incluído na string.

```
puts [regexp {[1-5]} "6789"]
```

Retorna 0, pois nenhum dos números de 1 a 5 está incluído na string.

```
set x ""
regexp -nocase {[^def]} "abcdef" x
puts $x
```

Retorna abc, pois regexp atribuirá à string todos os caracteres que não correspondam à expressão fornecida.

Expressões regulares são muito usadas em bancos de dados SQL e programas CGI que procuram informações em páginas HTML.

3 Estruturas de dados

Tcl suporta dois tipos de estruturas de dados: *vetores associativos* e *listas*. A seguir estudaremos cada uma dessas estruturas, assim como os recursos que Tcl oferece para manipulá-las.

3.1 Vetores associativos

Vetores são variáveis capazes de armazenar vários valores ao mesmo tempo. Sendo que cada valor é referenciado por um índice, de modo semelhante a uma matriz.

Vetores podem ter uma, duas ou múltiplas dimensões. No entanto, é mais comun trabalhar com vetores com até 5 dimensões. Neste caso, na resolução de sistemas de equações com múltiplas variáveis.

Observe o exemplo de um vetor típico:

Suponhamos que queiramos armazenar os nomes dos dias da semana em um vetor:

```
dia(1) = "domingo"
dia(2) = "segunda-feira"
dia(3) = "terça-feira"
dia(4) = "quarta-feira"
dia(5) = "quinta-feira"
dia(6) = "sexta-feira"
dia(7) = "sábado"
```

Em Tcl faríamos:

```
set dia(1) "domingo"
set dia(2) "segunda-feira"
set dia(3) "terça-feira"
set dia(4) "quarta-feira"
set dia(5) "quinta-feira"
set dia(6) "sexta-feira"
set dia(7) "sábado"
```

E para lermos o nome do dia armazenado na posição 2:

puts $dia(2)

Se você já conhece outras linguagens de programação, deve estar se perguntando se Tcl começa os números dos índices com zero(0) ou um(1). A resposta é: Tcl não começa. De fato, Tcl trabalha com vetores associativos. Dessa forma os índices pode ser números, letras, palavras ou qualquer combinação de caracteres.

Suponha que você queira armazenar os dados de um cliente em um vetor *cliente*. Podemos armazenar o nome do cliente no elemento *nome* do vetor, enquanto o telefone seria armazenado no elemento *telefone*, e o e-mail no elemento *e-mail*:

set cliente(nome) "Roberto Luiz Souza Monteiro"

set cliente(telefone) "+55-71-9121-7576"

set cliente(e-mail) "souzamonteiro@souzamonteiro.com"

Para acessarmos o nome do cliente faríamos:

puts $cliente(nome)

E na verdade, é assim que trabalham os bancos de dados em Tcl: atribuindo o registro corrente a um vetor associativo. Para entender melhor como funcionam vetores associativos em bancos de dados, veja o código fonte da biblioteca TDO - Tcl Database Access Objects, disponível em http://www.souzamonteiro.com/tdo.shtml.

Tcl oferece as seguintes funções para tratamento de vetores:

Função	Descrição
array anymore	Verifica, e retorna 1, se ainda existirem elementos a serem pesquisados no vetor.
array donesearch	Termina uma busca em um vetor.
array exists	Verifica, e retorna 1, se o vetor existir.
array get	Retorna uma lista onde cada elemento ímpar corresponde ao nome do elemento no vetor, e cada elemento par corresponde ao valor do elemento.
array names	Retorna uma lista com os nomes de cada elemento no vetor.
array nextelement	Retorna o nome do próximo elemento no vetor.
array set	Atribui a cada elemento no vetor, o valor correspondente ao existente na lista fornecida.
array size	Retorna o número de elementos no vetor.
array startsearch	Retorna um ponteiro para uma busca sequencial em um vetor.
parray	Exibe, na saída padrão, os nomes e os valores de todos os elementos no vetor, que correspondam ao *padrão* especificado.

Exemplos

Suponha o vetor dos dias da semana:

```
set dia(1) "domingo"
set dia(2) "segunda-feira"
set dia(3) "terça-feira"
set dia(4) "quarta-feira"
set dia(5) "quinta-feira"
set dia(6) "sexta-feira"
set dia(7) "sábado"
```

array get

```
puts [array get dia]
```

Retornará:

4 quarta-feira 5 quinta-feira 1 domingo 6 sexta-feira 2 segunda-feira 7 sábado 3 terça-feira

array names
puts [array names dia]
Retornará:
4 5 1 6 2 7 3

array size
puts [array size dia]
Retornará:
7

parray
puts [parray dia]
Retornará:
dia(1) = domingo
dia(2) = segunda-feira
dia(3) = terça-feira
dia(4) = quarta-feira
dia(5) = quinta-feira
dia(6) = sexta-feira
dia(7) = sábado

Listas

Listas são provavelmente as estruturas de dados mais poderosas. De fato, Tcl oferece muitos recursos para manipulação de listas. Mas o que são listas?

Listas são estruturas capazes de armazenar coleções de dados, de forma estruturada. Uma lista pode conter outras listas aninhadas e é possivel ordenar e pesquisar listas do mesmo modo como se faria em um banco de dados, e com a mesma eficiência.

Veja o seguinte exemplo:

Suponha um banco de dados de clientes, onde cada registro conteria o nome, o telefone e o e-mail de cada cliente. Tcl poderia armazenar o banco de dados em uma lista, onde cada registro sera armazenado em um elemento da lista como uma segunda lista, e nesta segunda lista cada elemento corespoderia a um campo no banco de dados:

```
{
   {nome telefone e-mail}
   {"Roberto Luiz Souza Monteiro" "+55-71-9121-7576" "souzamonteiro@souzamonteiro.com"}
   {"Katia Souza Monteiro" "+55-71-242-8798" "katia@souzamonteiro.com"}
   {"Regina C. S. de Andrade" "+55-71-386-0627" "regina@souzamonteiro.com"}
}
```

Sabendo que o primeiro elemento na lista corresponde à lista de nomes dos campos na tabela, cada elemento seguinte, corresponde a uma linha, ou registro, na tabela. E de fato, no banco de dados TclVSdb, os dados são armazenados dessa forma nas tabelas do banco de dados.

Tcl oferece as seguintes funções para tratamento de listas:

Função	Descrição
concat	Concatena várias lista para formar uma nova lista.
join	Converte uma lista em uma string, separando os elementos da lista com o separador indicado.
lappend	Adiciona um elemento ao final da lista.

lindex	Retorna o valor do elemento indicado na lista.
linsert	Insere um elemento em uma lista. Na verdade retorna uma nova lista.
list	Cria uma lista, a partir dos elementos fornecidos como argumentos.
llength	Retorna o tamanho de uma lista.
lrange	Retorna uma lista contendo os elementos dentro da faixa especificada.
lreplace	Substitui valores em uma lista. Na verdade retorna uma nova lista.
lsearch	Executa uma busca em uma lista. Suporta expressões regulares.
lsort	Ordena uma lista. Suporta várias opções. Na verdade retorna uma nova lista.
split	Converte uma string em uma lista, considerando um caractere especificado na string, como o separador dos elementos.

Exemplos

concat

```
set tabela {}
puts [llength $tabela]
```

Retornará 0.

```
set tabela [concat domindo segunda-feira terca-feira quarta-feira quinta-feira sexta-feira sabado]
puts [llength $tabela]
```

Retornará 7, pois será criada uma nova lista onde cada elemento da lista corresponderá a uma das strings fornecidas.

join

```
set tabela [concat domindo segunda-feira terca-feira quarta-feira quinta-feira sexta-feira sabado]
puts [join $tabela ","]
```

Retornará:
domindo,segunda-feira,terca-feira,quarta-feira,quinta-feira,sexta-feira,sabado

lappend

```
set tabela [concat domindo segunda-feira terca-feira quarta-feira quinta-feira sexta-feira]
puts [llength $tabela]
```

Retornará 6.

```
lappend tabela sabado
puts [llength $tabela]
```

Retornará 7.

lindex

```
set tabela [concat domindo segunda-feira terca-feira quarta-feira quinta-feira sexta-feira sabado]
puts [lindex $tabela 2]
```

Retornará:
terca-feira

llength

```
set tabela [concat domindo segunda-feira terca-feira quarta-feira quinta-feira sexta-feira sabado]
puts [llength $tabela]
```

Retornará 7.

lsearch

```
set tabela [concat domindo segunda-feira terca-feira quarta-feira quinta-feira sexta-feira sabado]
puts [lsearch -exact $tabela quarta-feira]
```

Retornará 3, pois o quarta-feira é o elemento número 3. Domingo é o elemento 0.

lsort

```
set tabela {vermelho alaranjado amarelo verde
azul anil violeta}
puts [lsort -increasing $tabela]
```

Retornará:

```
alaranjado amarelo anil azul verde vermelho
violeta
```

Se estivermos trabalhando com registros, onde:

```
set cliente {
   {1 {Roberto Luiz Souza Monteiro}}
   {2 {Katia Souza Monteiro}}
   {3 {Adriano Lavigne}}
}
puts [lsort -increasing -index 1 $cliente]
```

Retornará:

```
{3 {Adriano Lavigne}} {2 {Katia Souza
Monteiro}} {1 {Roberto Luiz Souza Monteiro}}
```

Observe que se você estiver atribuindo elementos a uma lista de forma estruturada, em um arquivo de script, como foi feito acima, deve terminar cada linha com o caractere \ pois Tcl interpreta o caractere de final de linha \n como o final do comando.

Assim a atribuição da lista de clientes ficaria:

```
set cliente { \
   {1 {Roberto Luiz Souza Monteiro}} \
   {2 {Katia Souza Monteiro}} \
   {3 {Adriano Lavigne}} \
}
```

No shell do interpretador Tcl(wish ou tclsh) isso não é necessário: você pode copiar e colar o fragmento de código anterior, na linha de comandos do wish, que ele funcionará corretamente.

split

Suponha uma função que lê um arquivo de configurações, onde exista a estrutura:
variável=valor

Podemos usar split para quebrar a string lida do arquivo, em uma lista onde o elemento 0 será o nome da variável e o elemento 1 o valor da variável:

```
set linha "showtip=false"
set token [split $linha =]
puts [lindex $token 0]
puts [lindex $token 1]
```

Retornará:

```
showtip
false
```

4 Controle de fluxo

Com frequência, em programação, nossos programas precisam escolher entre certas direções determinadas, de acordo com certas condições pre-estabelecidas. A isso se chama controle de fluxo de execução do programa. Tcl oferece mecanismos avançados para que o programador possa controlar o fluxo de execução de seus programas. Mais adiante estudaremos cada um dos recursos que Tcl oferece para que controle de fluxo de execução de programas. Mas antes vamos entender um pouco mais o que é fluxo de execução.

Analizemos a seguinte situação: Precisamos imprimir todos os números ímpares de 0 a 100. O *algorítimo* do nosso programa ficaria assim:

```
N = 0
enquanto N < 100 faça {
   se resto(N, 2) <> 0 então {
      imprima N
   }
   incremente N
}
```

Na primeira linha do nosso programa, iniciamos a variável N como 0. Em seguida entramos em um *laço de repetição*(uma parte do programa que se repete enquanto uma condição dada permanecer verdadeira) enquanto N for menor que 100.

No interior do laço de repetição, verificamos se o número N é ímpar(um número é impar se o resto da divisão do número por dois for diferente de zero). Se for ímpar, imprimimos o número.

Incrementamos(somamos mais um) ao número e repetimos o laço. Cada repetição do laço se chama *interação*.

Vejamos agora como ficaria cada parte do programa em Tcl:

```
set N 0
while {$N < 100} {
   if {[expr fmod($N,2)]} {
      puts $N
   }
   incr N
}
```

4.1 if... then...elseif...else...

A expressão *if...then...elseif...else...* permite controlar o fluxo de execução do programa de acordo com diferentes condições dadas. A sintaxe é:

if *expr1 corpo1* elseif *expr2 corpo2*... else *corpoN*

Onde *expr1*, *expr2*, *exprN* são diferentes condições a serem testadas e *corpo1*, *corpo2*, *corpoN* são porções de código que devem ser executadas caso a condição correspondente seja verdadeira. A expressão else somente é executada caso todas as outras expressões sejam avaliadas como falsas.

Exemplo:

```
if {$x == 0} {
   puts "X é igual a 0"
} elseif {$x < 0} {
   puts "X é menor que 0"
} else {
  puts "X é maior que 0"
}
```

4.2 while...

A expressão *while...* permite executar um laço de repetição enquanto uma expressão for avaliada como verdadeira. A sintaxe de while é:

while *expr1 corpo*

Onde expr1 é a expressão que deve ser avaliada como verdadeira(*1*, *true* ou *yes*) para que o laço seja repetido. E *corpo* é o fragmento de código que será executado a cada repetição do laço.

Pode-se interromper a execução de um laço a qualquer momento através do comando **break**.

Para pular para a próxima interação de um laço, saltando todas as instruções restantes, podemos utilizar o comando **continue**.

Exemplo:

```
set x 0
while {$x < 10} {
   puts $x
}
```

4.3 for...

A expressão *for...* permite executar um laço de repetição enquanto uma expressão for avaliada como verdadeira. A sintaxe de for é:

for *início teste próximo corpo*

O onde *início* contém o código de *inicialização da variável* usada no *teste*, *teste* contém uma expressão que deve ser avaliada como verdadeira para que o laço se repita, *próximo* contém um fragmento de código que é avaliado a cada repetição, e que normalmente é utilizado para incrementar a variável e *corpo* contém o fragmento de código que se deseja executar a cada *interação* do laço.

Pode-se interromper a execução de um laço a qualquer momento através do comando **break**.

Para pular para a próxima interação de um laço, saltando todas as instruções restantes, podemos utilizar o comando **continue**.

Exemplo:

```
for {set x 1} {$x < 10} {incr x} {
   puts $x
}
```

4.4 foreach...

A expressão *foreach...* oferece um meio de interagir por todos os elementos de uma *lista*, de forma simplificada. *foreach* é muito usada no processamento de informações retornadas por consultas *SQL*.

A sintaxe de foreach é:

foreach *variável lista corpo*

Onde *variável* é o nome de uma variável que irá conter, a cada interação, o valor de um elemento da *lista*. *lista* é a lista que será percorrida e *corpo* é o fragmento de código que será executado a cada *interação*.

Pode-se interromper a execução de um laço a qualquer momento através do comando **break**.

Para pular para a próxima interação de um laço, saltando todas as instruções restantes, podemos utilizar o comando **continue**.

Exemplo:

```
set lista {vermelho alaranjado amarelo verde azul anil violeta}
foreach cor $lista {
   puts $cor
}
```

4.5 switch...

O comando *switch* é provavelmente o meio mais poderoso para se controlar o fluxo de execução de um programa em Tcl. *switch* permite que se compare o valor de uma *string* com diversos valores dados e se execute a porção de código correspondente. *switch* suporta *expressões regulares* e é muito usado em *comandos definidos pelo usuário* para avaliar as *opções* passadas para o comando.

A sintaxe de switch é:

switch [-exact] [-regexp] [-glob] *string* {
   valor1 corpo1
   valor2 corpo2
   valorN corpoN
}
Onde as *opções* entre cochetes [] são opcionais, *string* é a string que será comparada com os *valores* definidos, *valor1*, *valor2*, *valorN* são os valores que serão comparados com a *string* e *corpo1*, *corpo2*, *corpoN* são os fragmentos de código que serão executados caso o valor correspondente coincida com a *string* passada para o comando.

Exemplo:

```
set x "arquivo"
switch $x {
   arquivo {puts "Voce selecionou a opção arquivo"}
   editar {puts "Voce selecionou a opção editar"}
   ajuda {puts "Voce selecionou a opção ajuda"}
}
```

4.6 exit

O comando *exit* encerra a execução do programa e retorna um valor para o *shell*.

A sintaxe de exit é:

exit valor

Onde *valor* é o valor que será retornado ao *shell*.

Exemplo:

```
set lista {vermelho alaranjado amarelo verde azul anil violeta}
foreach cor $lista {
   puts $cor
}
exit 0
```

5 Funções e procedimentos

Procedimento é uma subrotina ou porção de código, que é executado a partir de um nome que o referencia. Uma função é um procedimento que retorna um valor.

Para entender melhor o que é uma função, vejamos o seguinte exemplo:

Desejamos calcular o quadrado de um número dado e imprimi-lo na tela:

```
proc quadrado {numero} {
   return [expr $numero * $numero]
}
puts [quadrado 2]
```

Retornará 4 pois 2 vezes 2 é igual a 4.

5.1 Criando procedimentos e funções

Criamos procedimentos e funções através do comando **proc**. A sintaxe de proc e a seguinte:

proc *nome* {[*argumento1*] [*argumento2*]*...* [*argumentoN*]} {
   [upvar *nível variável_acima variável_local*]
   [global [*variavel1*] [*variavel2*]*...* [*variavelN*]
   instruções...
   [return [*valor*]]

}

Onde *nome* é o nome do *procedimento* ou *função*, *argumento1*, *argumento2*, *argumentoN* são *argumentos* ou valores opcionais passados ao procedimento ou função, *variavel1*, *variavel2*, *variavelN* são variaveis *globais*(ou seja variáveis vistas em todo o programa), *instruções* são as instruções que serão executadas no procedimento ou função e *valor* é um valor que opcionalmente será retornado ao procedimento que chamou a função. O comando *upvar* será estudado mais adiante.

5.2 Argumentos

Argumentos são valores passados para a função ou procedimento. No caso do exemplo visto acima, do cálculo do quadrado de um número, *número* é o argumento passado à função quadrado. Argumentos podem ter valores *default*, ou padrão. Assim, caso nenhum valor seja passado para a função, um valor padrao será usado como argumento. Para definir um valor padrão para um argumento devemos colocar o nome do argumento e o valor padrão entre chaves {}, assim:

```
proc quadrado {{numero 0}} {
   return [expr $numero * $numero]
}
```

Caso existam vários argumentos, cada argumento que possuir um valor padrão deverá vir entre chaves, assim:

proc *nome* {{*argumento1 valor1*} {*argumento2 valor2*}... {*argumentoN valorN*}} { ... }

Em muitos casos é importante fornecer valores padrões para os argumentos de uma função. Isso evitará erros de tempo de execução. Assim, estude bem suas funções e procedimentos.

5.3 Escopo das variáveis

Variáveis podem ser visíveis em todo o programa, ou somente em partes dele. A isso se dá o nome de *escopo*. Quando uma variável não pode ser vista em determinada parte do programa, se diz que ela está fora do escopo da função ou procedimento.

De um modo geral, variáveis criadas no interior de uma função ou procedimento, serão visíveis somente no interior da função ou procedimento onde foi criada. Para tornar uma variável visível em todo o programa, usa-se o comando **global**. Exemplo:

global código, nome, e-mail

Torna as variáveis *código*, *nome* e *e-mail* disponíveis para todo o programa.

Entretanto, para utilizar as variáveis acima dentro de uma outra função, você terá que repetir o mesmo comando acima dentro da função onde deseja utilizar as variáveis.

Muitas vezes entretanto, desejamos utilizar uma variável criada em um nível acima da nossa função, e não uma variável global. Nestes casos podemos utilizar o comando **upvar**, para criar um ponteiro para a nossa variável um ou mais níveis acima. Exemplo:

upvar 1 registro reg

Faz com que a variável *reg* referencie a variável *registro* que está na função chamadora(um nível acima). Caso nível seja precedido pelo sinal #, nível será tratado como um valor absoluto. Assim #0 estará se referindo ao nível mais alto no programa.

5.4 Retornando valores

É possivel retornar valores de funções através do comando return, como visto no exemplo de introdução desta aula. Contudo, isso não é obrigatório.

5.5 Criando bibliotecas de funções

Bibliotecas de funções são arquivos contendo declarações de diversas funções que queremos reaproveitar em nossos programas. Não há nenhum formato especial para arquivos de bibliotecas.

Para carregar uma biblioteca utilizamos o comando **source**. A sintaxe de source é:

source arquivo.tcl

Carregará a biblioteca.

O comando source também pode ser usado para carregar programas inteiros em Tcl. Ao carregar um arquivo, source compilará e executará os comandos no arquivo.

5.6 Criando pacotes

Uma forma elegante de distribuir bibliotecas é colocando-as em pacotes. Um pacote é criado colocando um comando **package provide** no início do arquivo. A sintaxe de *packageprovide* é:

package provide *nome versão*

Onde nome é o nome do pacote(biblioteca) que se está provendo e *versão* é o número da versão do pacote.

Para carregar um pacote coloque no ínicio de seu programa:

package require *nome versão*

Onde nome é o nome do pacote(biblioteca) que se está solicitando e *versão* é o número da versão do pacote.

Para instalar um pacote em seu computador, crie um diretório, dentro do diretório das bibliotecas do Tcl, com o nome do pacote seguido da versão. Em seguida entre no diretório e execute o comando:

pkg_mkIndex .

Isso criará um índice que Tcl utilizará para encontrar o pacote. Examine o diretório do Tcl em seu computador para entender melhor os nomes dos diretórios de pacotes.

O diretório das bibliotecas do Tcl no Windows é:

C:\Arquivos de Programas\Tcl\lib\

E no Linux, normalmente é:

/usr/lib/

5.7 Namespaces

Namespaces são a nova forma de encapsular variáveis e funções em bibliotecas e programas em Tcl. Trata-se de um tema avançado e difícil de entender para o iniciante. Assim nós o abordaremos no final do curso na aula 26.

6 Acesso ao sistema de arquivos

Tcl possui diversos comandos que permitem acessar o sistema de arquivos de um computador. Muitos desses comandos, possuem opções que só estam disponíveis em um determinado sistema operacional.

Para uma maior portabilidade de seu programa, é recomendável utilizar apenas as opções indepentes de sistema operacional.

Comando	Descrição
file atime	Retorna o tempo em segundos, a partir 01/01/1970, desde o último acesso ao arquivo.
file attributes	Retorna ao define os atributos de um arquivo. As opções são diferentes para o UNIX, o Windows e o MacOS.
file copy	Copia um ou mais arquivos para outro.
file delete	Remove um arquivo.
file dirname	Retorna o caminho completo de um arquivo.
file executable	Retorna 1 se o arquivo for um programa executável pelo usuário, 0 caso contrário.
file exists	Retorna 1 se o arquivo existir(e se o usuário puder ler o seu diretório), 0 caso contrário.
file extension	Retorna a extensão de um arquivo.
file isdirectory	Retorna 1 se o arquivo for um diretório, 0 caso contrário.
file isfile	Retorna 1 se o arquivo for um arquivo regular, 0 caso contrário.
file join	Une nome de arquivo e diretórios, usando o separador correto para o sistema operacional.
file lstat	O mesmo que file stat, mas usando a chamada lstat do kernel.
file mkdir	Cria um ou mais diretórios. Pode criar uma árvore inteira.
file mtime	Retorna o tempo em segundos, a partir 01/01/1970, desde a última modificação no arquivo.
file nativename	Retorna o nome específico do arquivo para o sistema operacional corrente.
file owned	Retorna 1 se o arquivo pertencer ao usuário, 0 caso contrário.
file pathtype	Retorna o tipo do caminho: *absolute*, *relative* ou *volumerelative*.
file readable	Retorna 1 se o arquivo puder ser lido pelo usuário, 0 caso contrário.
file readlink	Retorna o valor do link sinbólico.
file rename	Renomeia um arquivo.
file rootname	Retorna o nome do arquivo sem a extensão.
file size	Retorna o tamanho do arquivo.
file split	Retorna o caminho de um arquivo em uma lista. Cada subdiretório como um elemento da lista.
file stat	Retorna os resultados da chamada à função stat do kernel em uma lista, cujos elementos são: *atime*, *ctime*, *dev*, *gid*, *ino*, *mode*, *mtime*, *nlink*, *size*, *type* e *uid*.
file tail	Retorna o nome do arquivo, sem o caminho.
file type	Retorna o tipo do arquivo: *file*, *directory*, *characterSpecial*, *blockSpecial*, *fifo*, *link* ou *socket*.
file volume	Retorna a lista dos drivers locais no Windows, dos drivers locais e de rede no MacOS e apenas "/" no UNIX.
file writable	Retorna 1 se o usuário puder escrever no arquivo, 0 caso contrário.

Exemplos

file copy

file copy c:/autoexec.bat c:/autoexec.bak

Copia c:\autoexec.bat para c:\autoexec.bak.

file delete

file delete c:/autoexec.bak

Apaga o arquivo c:\autoexec.bak

file exists

puts [file exists c:/autoexec.bat]

Retorna 1 pois o arquivo existe no meu computador.

file join

puts [file join c autoexec.bat]

No Windows retorna c:/autoexec.bat

file mkdir	file rename
file mkdir c:/tmp/dir1/subdir1/subsubdir1 Cria o diretório tmp e todos os subdiretorios especificados abaixo dele.	file rename c:/autoexec.bak c:/autoexec.old Renomeia c:\autoexec.bak para c:\autoexec.old.
file size	file volume
puts [file size c:/autoexec.bat] Retorna o tamanho do arquivo	puts [file volume] Em meu computador retornou, no Windows: a:/ c:/ d:/ e:/ f:/ E no Linux apenas: /

No Windows, não utilize "\" para separar diretórios e nomes de arquivos, use "/", pois Tcl interpreta o caractere "\" como um caractere especial. Se desejar poderá usar "\\".

7 Entrada e saída em arquivos e portas

Tcl oferece um conjunto completo de comandos para realização de entrada e saída em arquivos e portas de comunicação. Em geral trabalhar com portas de comunicação é como trabalhar com arquivos, exigindo apenas um pouco mais de atenção, pois as portas podem parar de responder por alguns instantes. Em especial quando trabalhando com impressoras fiscais, conectadas à porta serial, ou scanners à porta paralela. Nestes casos é preciso introduzir rotinas de tratamento de erros e tentar repetidamente conectar à porta de comunicação. Por essa razão, nesta aula estudaremos também o sistema de tratamento de erros em Tcl.

7.1 Comandos de entrada e saída

A seguir são apresentados os comandos de entrada e saída em arquivos em Tcl. Alguns comandos possuem variações, assim, para uma descrição detalhada de todos os comandos disponíveis em Tcl para realização de entrada e saída em arquivos e portas de comunicação.

Comando	Descrição
close	Fecha um canal de comunicação.
eof	Retorna 1 se foi encontrado o final do arquivo, 0 caso contrário.
fblocked	Retorna 1 se a última operação de leitura tiver exaurido todos os dados disponíveis no canal de comunicação, 0 caso contrário.
fconfigure	Configura ou obtém as opções relativas ao canal de entrada e saída. São elas: -blocking; -buffering *full*|*line*|*none*; -buffersize; -translation *auto*, *binary*, *cr*,*crlf* e *lf*; -sockname; -peername; -mode.
fcopy	Copia dados de um canal para outro. Pode operar em background e executar um comando à medida em que os dados são transferidos.
fileevent	Avalia um script à medida que o canal se torna pisponível para leitura ou gravação.
flush	Descarrega o buffer de saída.
gets	Lê a próxima linha no canal de entrada, descartando o caractere de nova linha(\n).
open	Abre um canal de comunicação em um dos modos de acesso: r, r+, w, w+, a, a+.
puts	Envia uma string para o canal de saída. Opcionalmente, omitindo o caractere de nova linha(\n).
read	Lê todos os caracteres no canal de entrada, ou o número especificado de bytes. Opcionalmente, omitindo o caractere de nova linha(\n).
seek	Posiciona o ponteiro de acesso no arquivo. A origem pode ser: start, current ou end.
socket	Abre uma porta TCP, como cliente ou servidor.
tell	Retorna a posição atual no arquivo.

Exemplos

open, read, puts, close
Lê o arquivo de entrada
set entrada [open c:/autoexec.bat r]
set arquivo [read $entrada]
close $entrada
Exibe o arquivo de entrada na tela
puts "Entrada:\n"
puts "$arquivo\n"
Copia o arquivo de entrada c:\autoexec.bat para c:\autoexec.bak
set saida [open c:/autoexec.bak w]
puts $saida $arquivo
close $saida
Lê o arquivo c:\autoexec.bak e o exibe na tela
set entrada [open c:/autoexec.bak r]
set arquivo [read $entrada]
close $entrada
puts "Saida:\n"
puts "$arquivo\n"

Um exemplo avançado

O exemplo a seguir é uma pequena biblioteca para ler e salvar configurações em arquivos de configuração. Arquivos de configuração têm o seguinte formato:

# Comentário
variável1=valor1
variável2=valor2
...
variávelN=valorN

Onde *variável1*, *variável2*, *variávelN* são nomes de variáveis e *valor1*, *valor2*, *valorN* são os valores atribuídos às variáveis.

Arquivos de configuração costuman ter a seguinte nomenclatura:

```
UNIX:    .nome_do_arquivo
Windows: nome_do_arquivo.ini
```

No UNIX, salve o arquivo no diretório do usuário: "~/". No Windows, na pasta da aplicação, ou na pasta "C:\Windows\".

Carregue a bliblioteca usando o camando source:

source rc.tcl

Caso tenha colocado o arquivo no mesmo diretório do seu programa. Caso contrário, indique o caminho completo.

```
# rc.tcl - Suporte a arquivos de recursos
#
# Copyright (c) 2000 by Roberto Luiz Souza Monteiro
#
# Veja o arquivo "licenca.termos" para informacoes sobre uso e
redistribuicao
# deste arquivo, e para NEGACAO DE TODA E QUALQUER GARANTIA.

#
# rc_load rc_array rc_file
#
```

```
# Carrega um arquivo de recursos em um vetor associativo
#
# rc_array: vetor associativo onde cada elemento corresponde
#           a uma variavel no arquivo de recursos
#
# rc_file: arquivo de recursos
#
# Retorno: 1 - arquivo encontrado
#          0 - arquivo nao encontrado
#
# Linhas de comentarios comecam com o caractere sustenido(#)
# Linhas em branco sao ignoradas
# Recursos devem estar no formato: variavel=valor
#
proc rc_load {rc_array rc_file} {
    global $rc_array

    set rc_found 1

    if {[file exists $rc_file]} {
        set handle [open $rc_file r]
        set rc [read $handle]
        close $handle

        set rc_list [split $rc "\n"]

        if {[llength $rc_list] > 0} {
            for {set i 0} {$i < [llength $rc_list]} {incr i} {
                set rc_line [lindex $rc_list $i]

                set rc_line [string trim $rc_line]

                if {[string index $rc_line 0] != "#"} {
                    if {[string trim $rc_line] != ""} {
                        set rc_line_list [split $rc_line "="]
                        if {[llength $rc_line_list] == 2} {
                            set name [string trim [lindex
$rc_line_list 0]]
                            set value [string trim [lindex
$rc_line_list 1]]

                            # Cria os elementos do vetor associativo
                            set ${rc_array}($name) $value
                        }
                    }
                }
            }
        }
    } else {
```

```
        set rc_found 0
    }

    return $rc_found
}

#
# rc_save rc_array rc_file
#
# Salva um vetor associativo em um arquivo de recursos
#
# rc_array: vetor associativo onde cada elemento corresponde
#           a uma variavel no arquivo de recursos
#
# rc_file: arquivo de recursos
#
# Recursos estarao no formato: variavel=valor
#
proc rc_save {rc_array rc_file} {
    global $rc_array

    set rc_list [array get $rc_array]

    set handle [open $rc_file w+]

    for {set i 0} {$i <= [expr "[llength $rc_list] - 2"]} {incr i} {
        set rc_line ""
        append rc_line [lindex $rc_list $i] "=" [lindex $rc_list [expr
"$i + 1"]]

        incr i

        puts $handle $rc_line
    }

    close $handle
}
```

Trantando erros

Tcl oferece um recurso bastante poderoso para tratar erros: o comando catch. A sintaxe de catch é:

catch *script variável*

Onde *script* é o código Tcl cujos erros serão interceptados e *variável* é o nome de uma variável onde será salvo qualquer erro retornado pelo script. Caso ocorra um erro catch retornará 1, caso contrário retornará 0.

Exemplo:

```
set erro ""
set ocorreu_erro [ catch {
                        # Le o arquivo de entrada
```

```
                    set entrada [open c:/autoexec.bat r]
                    set arquivo [read $entrada]
                    close $entrada
                } erro
]
if {$ocorreu_erro == 1} {
    puts "Ocorreu o seguinte erro enquanto tentava abrir o arquivo
c:/autoexec.bat\n"
    puts "Erro: $erro\n"
} else {
    # Exibe o arquivo de entrada na tela
    puts "Entrada:\n"
    puts "$arquivo\n"
}
```

Gerando erros

Você pode retornar erros de suas funções através do comando return:

return [-code *código*] [-erroinfo *informação*] [-errorcode *código_de_erro*]

Onde *código* pode ser ok, error, return, break, continue ou um valor inteiro, *informação* contém uma descrição do erro e *código_de_erro* contém um código para o erro ocorrido.

Você também pode gerar erros através do comando error:

error *mensagem* [*informação*] [*código_de_erro*]

As variáveis globais errorInfo e errorCode receberão os valores *informação* e *código_de_erro* respectivamente.

8 Interagindo com o sistema operacional

Tcl permite que seus programas interajam com os sistema operacional, através de um pequeno conjunto de comandos. Embora sejam poucos, os comandos, você verá que são suficientes para atender a todas as suas necessidades.

Comando	Descrição
cd	Muda o diretório de trabalho.
clock clicks	Retorna a data/hora como um número inteiro de alta resolução, dependente do sistema operacional.
clock format	Converte um número inteiro representando data/hora em uma data e/ou hora compreensivel para um ser humano.
clock scan	Converte uma data/hora em formato humano para um número inteiro.
clock seconds	Retorna a data/hora como um número inteiro, representando os segundos passados desde uma determinada data. A data de início da contagem depende do sistema operacional. No Windows é 22:00:00H de 31/12/1969.
exec	Executa um processo usando cada argumento como uma palavra para um "duto" ou redirecionamento de saída. Em outras palavras executa um comando no shell e retorna o valor enviado para a saída padrão. Suporta "dutos" ou redirecionamentos.
glob	Retorna uma lista de todos os arquivos que correspondam ao padrão indicado. O padrão deve estar no formato suportado pelo shell csh. Opcionalmente suprime erros caso não encontre nenhum arquivo.
pid	Retorna o indentificador de um processo especificado, ou do próprio interpretador Tcl. Muito útil para criar arquivos temporários.
pwd	Retorna o diretório de trabalho.

Exemplos

cd

```
cd "C:/Arquivos de Programas/Tcl/lib/"
```

Muda para o diretório C:\Arquivos de Programas\Tcl\lib\

clock format, clock seconds

```
puts [clock format [clock seconds] -format "%H:%M:%S"]
```

Exibe a hora atual no formato hh:mm:ss.

```
puts [clock format [clock seconds] -format "%d/%m/%Y"]
```

Exibe a data atual no formato dd/mm/aaaa.

```
puts [clock format 0 -format "%H:%M:%S de %d/%m/%Y"]
```

Exibe a hora e a data do ínicio da contagem dos tempos em seu sistema operacional.

O comando **clock format** suporta os seguintes *especificadores* de formato:

Especificador	Descrição
%%	%
%a	Dia da semana abreviado.
%A	Dia da semana completo.
%b	Mês abreviado.
%B	Mês completo.
%c	Data e hora local.
%d	Dia(01 - 31).
%H	Hora(00 - 23).
%h	Hora(00 - 12).
%j	Dia(001 - 366).
%m	Mês(01 - 12).
%M	Minuto(00 - 59).
%p	AM/PM.
%S	Segundos(00 - 59).
%U	Semana(01 - 52).
%w	Dia da semana(0 - 6).
%x	Data local.
%X	Hora local.
%y	Ano(00 - 99).
%Y	Ano completo.
%Z	Fuso horário

exec

```
exec /bin/tar cvzf backup.tgz *
```

Executa o comando tar no UNIX.

```
exec c:/windows/notepad.exe
```

Abre o bloco de notas no Windows.

pwd

```
puts [pwd]
```

Exibe o diretório de trabalho.

9 Introdução ao Tk

TK é o *Tool Kit*, o kit de ferramentas gráficas do Tcl. Com Tk é possível construir rapidamente aplicativos gráficos para UNIX, Windows e MacOS. Para você ter uma idéia de como é rápido e fácil construir aplicativos baseados em janelas, utilizando Tk, veja o exemplo abaixo. Não se preocupe em entender os comandos. Eles serão estudados mais tarde.

9.1 Alô Mundo

Veja uma implementação do programa "Alô Mundo!" em Tk. Alô mundo é o primeiro programa, que todo programador escreve quando está aprendendo uma linguagem.

```
button .alomundo -text "Alô Mundo" -command {exit}
pack .alomundo
```

Pronto! As duas linhas de comando anteriores farão tudo: Criarão um botão em uma janela, configurarão o envento *Click* do botão e posicionarão o botão na janela.

Experimente: Copie e cole o código acima na linha de comando do interpretador Tcl, o wish.

No exemplo anterior, a primeira palavra, **button**, é o comando usado para se criar um botão em Tk. Ela indica a classe a que pertence o *widget*, componente gráfico, que queremos criar. A segunda palavra *.alomundo*, é o nome do objeto que queremos criar. Observe que o nome do objeto começa com um ponto (.). A terceira palavra **-text** é o nome de uma propriedade que queremos configurar. A palavra seguinte *"Alô Mundo"* é o valor da propriedade que estamos configurando. A quinta palavra **-command**, é também uma propriedade, que indica ao Tk a ação que deve ser executada quando o botão for pressionado. Neste caso a ação será sair do programa: *exit*.

Uma vez que um widget seja criado, será criado também um comando, com o mesmo nome do widget, que permitirá acessar as propriedades do widget.

9.2 Determinando os nomes dos widgets

Em Tk os widgets são agrupados em uma hierarquia, como uma árvore de diretórios. Onde cada nível é separado por um ponto (.). O nome do nível mais alto, e que corresponde á primeira janela, criada automaticamente quando o interpretador Tcl é iniciado é ponto (.). Assim um widget inserido na janela principal se chamará *.nome_do_widget*. Na maioria dos casos, teremos os widgets arrumados no interior de molduras, que por sua vez estarão no interior de alguma janela. Assim, o nome de um widget nesta condição seria:*. nome_da_janela.nome_da_moldura.nome_do_widget*.

9.3 Opções

Em Tk as propriedades de um widget são chamadas de *opções*. Para ler uma opção de um widget utilizamos o sub-comando **cget**, do widget cuja opção queremos ler. Assim em nosso exemplo "Alô Mundo", para ler-mos o valor da opção *text* faríamos o seguinte:

```
puts [.alomundo cget -text]
```

O comando acima exibirá no console Tcl o texto "Alô Mundo".

Para configurar uma opção de um widget utilizamos o sub-comando **configure**, do widget cuja opção queremos configurar. Assim em nosso exemplo "Alô Mundo", para configurar-mos o valor da opção *text* faríamos o seguinte:

.alomundo configure -text "OK"

puts [.alomundo cget -text]

O comando acima exibirá no console Tcl o texto "OK".

Observe que o texto no botão, mudou imediatamente de "Alô Mundo" para "OK".

Cada widget Tk possui um número enorme de opções. A seguir são apresentadas as principais opções suportadas pelos widgets Tk.

Observe que muitas delas não são suportadas por todos os widgets, enquanto que outros widgets suportarão outras opções que não são apresentadas aqui, e serão vistas mais adiante.

Opção	Descrição
-activebackground	Cor de fundo quando o widget está ativo.
-activeborderwidth	Largura da borda quando o widget está ativo.
-activeforeground	Cor do primeiro plano quando o widget está ativo.
-anchor	Referência para posicionamento do widget: n, ne, e, se, s, sw, w, nw e center.
-background	Cor de fundo.
-bitmap	Bitmap para exibir no widget: error, gray12, gray25, gray50, gray75, hourglass, info, questhead, question, warning ou *@nome_do_arquivo*.
-borderwidth	Largura da borda.
-command	Comando a ser executado quando o evento *default* do widget ocorrer.
-cursor	Cursor do mouse. O nome do cursor depende do sistema operacional. No UNIX veja o arquivo /usr/include/X11/cursorfont.h. Também pode ser o nome de um arquivo: *@nome_do_arquivocor_de_fundo*. Onde *cor_de_fundo* indica a cor transparente.
-disabledforeground	Cor de fundo quando o widget está desabilitado.
-exportselection	Indica se uma seleção em um widget irá para o clipboard externo ou não. 1 - Exportar, 0 - Não exportar. Tk possui seu próprio clipboard.
-font	Nome e atributos da fonte em uma lista.
-foreground	Cor de primeiro plano.
-height	Altura do widget.
-hilightbackground	Cor do retângulo em volta do widget quando ele não detém o foco.
-hilightcolor	Cor do retângulo em volta do widget quando ele detém o foco.
-hilightthickness	Largura do retângulo em volta do widget quando ele detém o foco.
-image	Imagem a ser exibida no widget. Nome da imagem obtida com o comando image.
-insertbackground	Cor de fundo da área do cursor no modo *inserir*.
-insertborderwidth	Largura da área do cursor no modo *inserir*.
-insertofftime	Tempo em que o cursor deve permanecer desativado enquanto ele pisca no modo *inserir*.
-insertontime	Tempo em que o cursor deve permanecer ativado enquanto ele pisca no modo *inserir*.
-insertwidth	Largura do cursor no modo *inserir*.
-jump	Informa às *scrollbars* e *escalas* conectadas ao widget para atrasarem a atualização até o botão do mouse ser liberado.
-justify	Alinhamento do texto: left, center, right.
-orient	Orientação: horizontal, vertical.
-padx	Espaçamento horizontal.
-pady	Espaçamento vertical.

-relief	Efeito 3D da borda do widget: flat, groove, raised, ridge, sunken.
-repeatdelay	Tempo necessário para iniciar a auto-repetição ao se manter pressionado um botão ou tecla.
-repeatinterval	Tempo entre as auto-repetições.
-selectbackground	Cor de fundo dos ítens selecionados.
-selectborderwidth	Largura da borda ao redor dos ítens selecionados.
-selectforeground	Cor do primeiro plano dos ítens selecionados.
-setgrid	Indica se o widget deve controlar a grade de redimensionamento para a janela onde ele está inserido.
-state	Estado: normal, disabled, active.
-takefocus	Indica se o widget deve sempre ou nunca receber o foco. Se vazio, isso será decido pelo Tk.
-text	Texto a ser exibido no widget.
-textvariable	Variável que contém o texto que deve ser exibido no widget. O widget será atualizado sempre que o valor da variável mudar, e o valor da variável mudará caso o texto no widget mude.
-troughcolor	Cor da área de deslizamento das scrollbars. Não funciona no Windows.
-underline	Posição do caractere sublinhado no texto do widget. Usado em menus e botões para indicar a tecla de atalho. Se -1, Nenhum caractere será sublinhado.
-width	Largura do widget.
-wraplength	Largura da área coberta pela linha de texto para que ocorra uma quebra automática de linha.
-xscrollcommand	Comando usado para comunicação com a scrollbar horizontal.
-yscrollcommand	Comando usado para comunicação com a scrollbar vertical.

10 Criando e gerenciando janelas

10.1 Em Tk janelas são chamadas Toplevels *e são criadas com o comando* toplevel *como mostrado abaixo:*

toplevel *.nome_da_janela* [-opção1 *valor1*] [-opção2 *valor2*]... [-opçãoN *valorN*]

Onde *.nome_da_janela* é o nome do objeto *janela* que desejamos criar, *opção1*, *opção2*, *opçãoN* são opções que desejamos configurar e *valor1*, *valor2*, *valorN* são os valores que desejamos atribuir a cada uma das opções que desejamos configurar.

Exemplo:

```
toplevel .principal
button .principal.ok -text "OK" -command {exit}
pack .principal.ok
```

Criará uma janela com um botão "OK" no centro.

Além de algumas das opções apresentadas na aula anterior os Toplevels suportam as seguintes opções:

Opção	Descrição
-class	Nome da classe da janela para ser usada na base de dados de opções.

-colormap	Mapa de cores a ser usado para a janela. Pode ser new, o caminho de outra janela, ou vazio.
-container	Se 1, faz com que a janela seja um container para outro aplicativo.
-menu	Nome de um widget menu para ser usado na janela.
-use	Identificador de uma janela onde esta janela será embutida. Veja winfo id.
-screen	Tela onde esta janela será apresentada. Esta opção funciona no UNIX.
-visual	Visual a ser usado para a janela.

Exemplos

-container, -use

```
toplevel .a -container 1
```

Cria uma janela para conter outras.

```
toplevel .principal -use [winfo id .a]
button .principal.ok -text "OK" -command {exit}
pack .principal.ok
```

Criará uma janela com um botão "OK" no centro, dentro da janela *.a*.

10.2 Obtendo informações sobre uma janela

Podemos obter informações sobre as janelas através do comando **winfo**. winfo pertence a uma classe de comandos que suportam sub-comando. A seguir são apresentados todos os comando winfo:

Comando	Descrição
winfo allmapped	Retorna 1 se a janela e todos os seus descendentes estiverem mapeados.
winfo atom	Retorna um identificador inteiro para o atom, dado o seu nome na janela.
winfo atomname	Retorna o nome de um atom dado o seu identificador inteiro.
winfo cells	Retorna o número de células no mapa de cores da janela.
winfo children	Retorna uma lista com os nomes de todas as janelas filhas da janela dada.
winfo class	Retorna o nome da classe da janela.
winfo colormapfull	Retorna 1 se o mapa de cores da janela estiver cheio. Caso contrário, retorna 0.
winfo containing	Retorna o nome da janela contendo o ponto X Y no display da janela dada.
winfo depth	Retorna a definição em bits por pixel da janela.
winfo exists	Retorna 1 se a janela existir. Caso contrário 0.
winfo fpixels	Retorna um número em ponto flutuante, correspondente ao número de pixels na janela, que correspondem à distância dada.
winfo geometry	Retorna a geometria da janela em pixels, na forma: larguraxaltura+x+y.
winfo height	Altura da janela.
winfo id	Identificador da janela em formato hexadecimal.
winfo interps	Retorna uma lista de todos os interpretadores Tcl registrados no display da janela.
winfo ismapped	Retorna 1 se a janela estiver mapeada. Caso contrário 0.
winfo manager	Retorna o nome do gerenciador de geometria responsavel pela janela.
winfo name	Retorna o nome da janela, sem o nome da janela mãe.
winfo parent	Retorna o nome da janela mãe da janela dada.
winfo pathname	Retorna o nome da janela, dado o seu identificador hexadecimal.
winfo pointerx	Retorna a coordenada x do ponteiro do mouse.
winfo pointerxy	Retorna as coordenadas x e y do ponteiro do mouse.

winfo pointery	Retorna a coordenada y do ponteiro do mouse.
winfo pixels	Retorna o número de pixels na janela, que correspondem à distância dada arredondada para o inteiro mais próximo.
winfo reqheight	Retorna uma string decimal correspondente a altura requisitada pela janela.
winfo reqwidth	Retorna uma string decimal correspondente a largura requisitada pela janela.
winfo rgb	Retorna uma lista com três valores RGB correspondentes à cor dada, na janela.
winfo rootx	Retorna a coordenada X do canto superior esquerdo da janela, dentro da janela raiz, incluindo a borda da janela.
winfo rooty	Retorna a coordenada Y do canto superior esquerdo da janela, dentro da janela raiz, incluindo a borda da janela.
winfo server	Retorna informações sobre o servidor da janela.
winfo screen	Retorna o nome da tela onde a janela está. Na forma *nome_do_display.índice*.
winfo screencells	Retorna o número de células no mapa de cores padrão na tela da janela.
winfo screendepth	Retorna a definição em bits por pixel da tela da janela.
winfo screenheight	Retorna a altura da tela em pixels.
winfo screenmmheight	Retorna a altura da tela em milímetros.
winfo screenmmwidth	Retorna a largura da tela em milímetros.
winfo screenvisual	Retorna a classe do visual da tela: *directcolor*, *grayscale*, *pseudocolor*, *staticcolor*, *staticgray* ou *truecolor*.
winfo screenwidth	Retorna a largura da tela em pixels.
winfo toplevel	Retorna o nome da janela que contém a janela dada.
winfo visual	Retorna a classe do visual da janela: *directcolor*, *grayscale*, *pseudocolor*, *staticcolor*, *staticgray* ou *truecolor*.
winfo visualsavailable	Retorna uma lista contendo as classes de visuais disponíveis para a janela.
winfo vrootheight	Retorna a altura da tela virtual onde está a janela dada. Disponível no UNIX.
winfo vrootwidth	Retorna a largura da tela virtual onde está a janela dada. Disponível no UNIX.
winfo vrootx	Retorna o deslocamento X da tela virtual onde está a janela dada.
winfo vrooty	Retorna o deslocamento Y da tela virtual onde está a janela dada.
winfo width	Retorna a largura da janela.
winfo x	Retorna a coordenada X do canto superior esquerdo da janela dentro da janela mãe.
winfo y	Retorna a coordenada Y do canto superior esquerdo da janela dentro da janela mãe.

Em geral os resultados variam de um sistema operacional para outro. Assim os exemplos serão vistos logo a seguir, quando tiver-mos estudado o gerenciador de janelas.

10.3 Gerenciando janelas

Podemos gerenciar as janelas criadas com Tk, através do *gerenciador de janelas*. A seguir são apresentados todos os comando do gerenciador de janelas **wm**:

Comando	Descrição
wm aspect	Informa ao gerenciador de janelas o aspecto desejado para a janela.

wm client	Informa ao gerenciador de janelas a máquina cliente onde o aplicativo está rodando.
wm colormapwindows	Informa ao gerenciador de janelas para usar um mapa de cores privado para as janelas internas.
wm command	Informa ao gerenciador de janelas o comando usado para invocar o aplicativo.
wm deiconify	Retorna uma janela, que estava minimizada, ao seu estado original.
wm focusmodel	Especifica o modelo de foco para a janela: active ou passive.
wm frame	Retorna o identificador para a moldura da janela. Caso não haja nenhuma moldura, retorna o identificador da janela.
wm geometry	Muda a geometria da janela. Na forma: larguraxaltura+x+y.
wm grid	Informa que a janela deve ser gerenciada como uma grade, com as relações especificadas entre grades e pixels.
wm group	Determina um lider para o grupo ao qual a janela pertence.
wm iconbitmap	Especifica um ícone para a janela. É um bitmap X11 e não um bitmap Windows.
wm iconify	Minimiza a janela.
wm iconmask	Especifica uma máscara para ser usada com o ícone especificado com wm iconbitmap.
wm iconname	Especifica um nome para o rótulo do ícone.
wm iconposition	Especifica a posição do ícone na janela raíz.
wm iconwindow	Especifica uma janela para ser usada como ícone quando a janela for minimizada.
wm maxsize	Especifica a dimensão máxima da janela.
wm minsize	Especifica a dimensão mínima da janela.
wm overrideredirect	Se 1, exibirá a borda da janela. Se 0, a janela será exibida sem bordas.
wm positionfrom	Indica de quem a posição atual da janela foi requisitada: program ou user.
wm protocol	Especifica um comando Tcl para ser executado, quando ocorrer uma mensagem de protocolo do tipo especificada.
wm resizable	Informa se a janela pode ser redimensionada, e em que direção isso pode ocorrer.
wm sizeable	Informa quem mudou as dimensões da janela: program ou user.
wm state	Retorna o estado atual da janela: *normal*, *icon*, *iconic* ou *withdrawn*.
wm title	Determina o título da janela.
wm transient	Informa que a janela é uma escrava da janela mestre. Isso faz com que somente um ícone seja apresentado para todo o aplicativo, e que ao fechar ou minimizar a janela mestre, a janela escrava seja fechada ou minimizada.
wm withdraw	Esconde a janela.

Exemplos

winfo screenheight, winfo screenwidth, wm geometry, wm protocol, wm title, wm withdraw

```
toplevel .principal
button .principal.ok -text "OK" -command {exit}
pack .principal.ok
```

Cria uma janela com um botão "OK" no centro.

```
set x [expr {([winfo screenwidth .principal] - [winfo width .principal])/2}]
set y [expr {([winfo screenheight .principal] - [winfo height .principal])/2}]
wm geometry .principal [winfo width .principal]x[winfo height .principal]+$x+$y
```

Centraliza a janela na tela.

```
wm title .principal "Curso de Tcl/Tk"
```

Especifica o título da janela.

wm protocol .principal WM_DELETE_WINDOW {exit}

Determina que o programa seja encerrado caso a janela seja fechada, clicando no botão fechar da moldura da janela.

wm withdraw .

Esconde a janela raíz do Tk.

11 Gerenciando a disposição dos widgets na janela

O modo como os widgets são arrumados na janela é chamado geometria. Atualmente, Tk possui três gerenciadores de geometria: **pack**, **place** e **grid**. Cada um correspondendo a um comando de mesmo nome.

11.1 O comando pack

O comando pack arruma os widgets na janela, segundo coordenadas relativas, do tipo "coloque os widgets de cima para baixo, esticando-os para preencher todo o espaço horizontal".

Veja um exemplo:

```
toplevel .t
button .t.b1 -text "Botão 1"
button .t.b2 -text "Botão 2"
button .t.b3 -text "Botão 3"
pack .t.b1 .t.b2 .t.b3
```

Cria uma janela com três botões arrumados de cima para baixo.

O comando pack não apenas dispõe os widgets na janela inicialmente, como mantém os widgets na mesma disposição, independente de o usuário redimensionar a janela, ou não.

De um modo geral, quando não estiver utilizando um ambiente de desenvolvimento visual, como o Visual Tcl, deverá preferir utilizar o comando **pack** ou o comando **grid**. Quando estiver desenhando suas telas com o Visual Tcl, deverá, a maioria das vezes, optar pelo gerenciador **place**.

A seguir são apresentados todos os comando do gerenciador de geometria pack:

Comando	Descrição
pack	Determina como os widgets devem ser dispostos na janela mestre.
pack forget	Faz com que um widget deixe de ser gerenciado pelo pack.
pack info	Retorna a configuração do pack para o widget dado.
pack propagate	Configura a propagação para a janela mestre.
pack slave	Retorna uma lista com os *escravos* dentro da janela *mestre*.

O comando pack suporta diversas opções. As principais são:

Opção	Descrição.
-anchor	Ponto de referência para o pack dispor o widget: n, ne, e, se, s, sw, w, nw e center.

-expand Se 1, o pack expandirá o widget para ocupar todo o espaço disponível. Se 0 o widget não será expandido.

-in Nome da janela onde o pack deve colocar o widget. É opcional, pois o nome do widget já indica o caminho.

-padx Espaço horizontal extra.

-pady Espaço vertical extra.

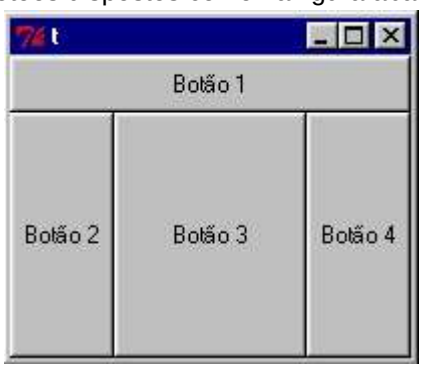
-fill Como o widget deve ser expandido: none, x, y, both.

-side Lado a partir do qual o pack deve colocar o widget: top, botton, left, right.

Exemplos:

```
toplevel .t
button .t.b1 -text "Botão 1"
button .t.b2 -text "Botão 2"
button .t.b3 -text "Botão 3"
button .t.b4 -text "Botão 4"
pack .t.b1 -side top -fill x
pack .t.b2 -side left -fill y
pack .t.b3 -side left -fill both -expand 1
pack .t.b4 -side left -fill y
wm geometry .t 200x150
```

Cria uma janela com quatro botões dispostos como na figura abaixo:



11.2 O comando grid

O comando grid dispõe os widgets em uma grade na janela, onde cada widget ocupa uma ou mais células da grade.

Veja o exemplo:

```
toplevel .t
button .t.b1 -text "Botão 1"
button .t.b2 -text "Botão 2"
button .t.b3 -text "Botão 3"
button .t.b4 -text "Botão 4"
grid .t.b1 -row 0 -column 0
grid .t.b2 -row 0 -column 1
grid .t.b3 -row 1 -column 0
grid .t.b4 -row 1 -column 1
```

Cria uma janela com quatro botões dispostos em uma grade.

O comando **grid** é muito útil para criar tabelas e painéis de botões como em calculadoras.

A seguir são apresentados todos os sub-comando do comando **grid**:

Comando	Descrição
grid [configure]	Determina como os widgets devem ser dispostos na grade da janela mestre.
grid bbox	Retorna as coordenadas do retângulo ocupado pelos widgets nas linhas e colunas determinadas. O formato é *linha1coluna1 linha2 coluna2*.
grid columnconfigure	Retorna ou configura as colunas indicadas na grade mestre. As opções são: -minsize, -pad e -weight.
grid forget	Remove o(s) widget(s) da grade.
grid info	Retorna as configurações do grid para o widget especificado.
grid location	Retorna a coluna e a linha contendo as coordenadas de tela especificadas.
grid propagate	Determina se o widget tentará redimensionar seus ancestrais para caber neles.
grid remove	Remove o(s) widget(s) da grade, mas continua lembrando as suas configurações.
grid rowconfigure	Retorna ou configura as linhas indicadas na grade mestre. As opções são: -minsize, -pad e -weight.
grid size	Retorna o tamanho da grade no formato *colunalinha*.
grid slaves	Sem opções, retorna uma lista dos escravos dispostos no *mestre*. Caso contrário retorna uma lista com os widgets na linha e/ou coluna especificados.

O comando grid suporta diversas opções. As principais são:

Opção	Descrição.
-column	Coluna.
-columnspan	Número de colunas que o widget ocupará.
-in	Nome da janela onde o grid deve colocar o widget. É opcional, pois o nome do widget já indica o caminho.
-padx	Espaço horizontal extra.
-pady	Espaço vertical extra.
-row	Linha.
-rowspan	Número de linhas que o widget ocupará..
-sticky	Determina a direção para o widget se ancorar: n, s, e, w.

Exemplos

```
toplevel .t
button .t.b1 -text "Botão 1" -width 24
button .t.b2 -text "Botão 2"
button .t.b3 -text "Botão 3"
button .t.b4 -text "Botão 4"
button .t.b5 -text "Botão 5" -width 24
grid .t.b1 -row 0 -column 0 -columnspan 3
grid .t.b2 -row 1 -column 0
grid .t.b3 -row 1 -column 1
grid .t.b4 -row 1 -column 2
grid .t.b5 -row 2 -column 0 -columnspan 3
```

Cria uma janela com quatro botões dispostos em uma grade, como na figura abaixo:

11.3 O comando place

O comando place é muito difícil de usar, pois trabalha com coordenadas absolutas. De um modo geral, só é usado pelos ambientes de desenvolvimento visual, como o Visual Tcl, onde o programador clica e arrasta para desenhar os widgets em uma janela. Isso é feito de modo transparente para o programador, de modo que ele não precisa se preocupar com a sintaxe do comando que está sendo utilizado para desenhar a janela de seu programa.

Não abordaremos o comando place aqui, pois você provavelmente nunca o utiliza-rá diretamente.

12 Visão geral dos widgets

Widgets são compondentes gráficos como botões, caixas de texto, imagens e menus, utilizados para permitir que o usuário interaja com o programa.

Tk suporta diversos widgets básicos, comuns a vários sistemas operacionais e pode ser extendido através da criação de mega-widgets(widgets compostos de outros widgets), que serão estudados na aula 21.

Os widgets Tk, em geral, possuem mais recursos e são mais rápidos que os widgets disponibilizados pelo sistema operacional. Entretanto, a Scriptics, atualmente Ajuba Solutions, tornou alguns widgets nativos. O que fez com que algumas opções desaparecessem em algumas plataformas. Um exemplo disso é o widget scrollbar, que no windows não puporta as opções relativas às cores dos sliders.

Existem diferenças de performance de um sistema operacional para outro. Por exemplo: no Linux os widgets são várias vezes mais rápidos que no Windows. Entretanto, Trabalhando com mega-widgets esta relação pode se inverter. É o caso dos mega-widgets BWidget, que no Windows são ligeiramente mais rápidos que no Linux. Isso considerando que os testes foram realizados na mesma máquina e sob as mesmas cirtunstâncias.

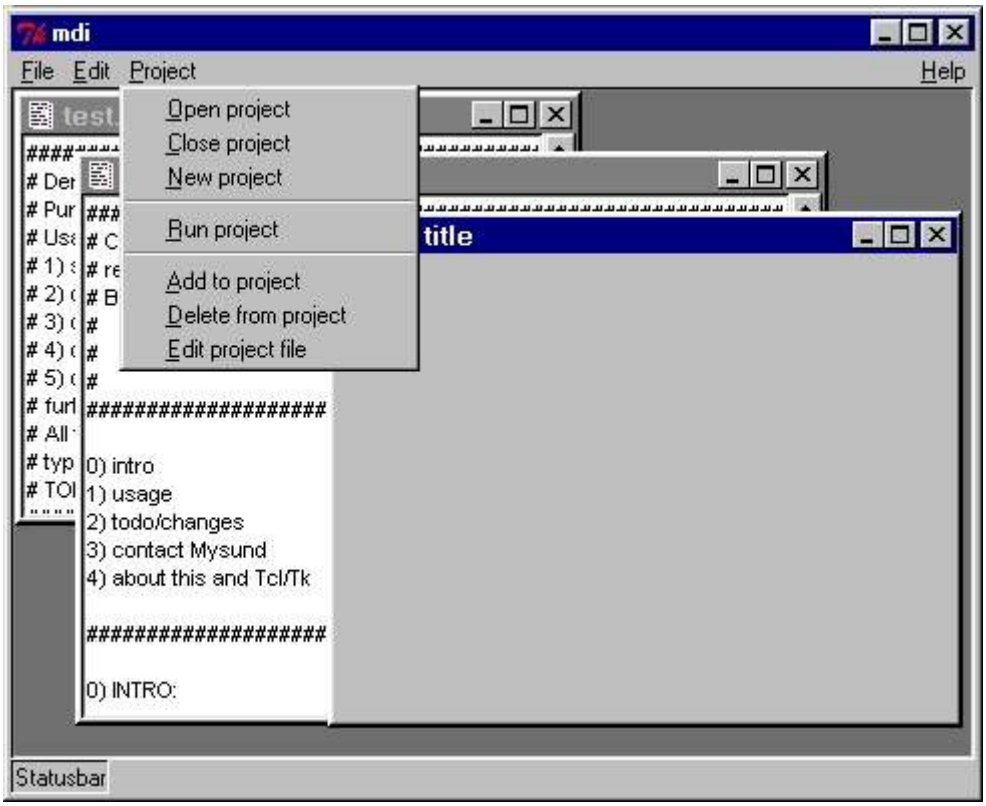
A seguir são apresentados os widgets básicos de Tk 8.3.

12.1 toplevel

Os toplevels, ou janelas, foram discutidos na aula anterior. Assim, para maiores detalhes, veja a aula 11.

Atualmente, Tk não suporta *formulários MDI*, que são interfaces de usuário compostas de múltiplos documentos. Na prática, isso significa que uma janela funciona como um gerenciador de janelas para outras janelas embutidas. Se você prestou atenção na aula anterior, deve ter verificado, que é possível embutir uma janela em outra, usando a opçào **-use** de Tk. Contudo, isso não criará uma janela com moldura e botões de controle dentro da janela mãe. apenas colocará a janela dentro da outra. Formulários MDI são gerenciadores de janelas completos, com opções de maximizar, minimizar etc.

Embora Tk não tenha o suporte MDI embutido, você pode obter esta funcionalidade através da biblioteca **Mysunds_MDI**. Veja um exemplo de um programa escrito em Tcl/Tk, usando a biblioteca Mysunds_MDI:



Observe que apesar da incrível semelhança com os formulários MDI do Windows, trata-se de uma biblioteca escrita em 100% puro Tcl. Na prática, você terá que editar o arquivo para poder utilizá-la no Linux, pois o autor utilizou nomes de cursores de mouse que só existem no Windows. Mas basta uma pequena alteração, em uma única função e tudo funcionará bem. Veja a documentação incluída no pacote da biblioteca para entender melhor o que precisa ser feito.

Você pode fazer o download da biblioteca Mysunds_MDI através do site: http://www.souzamonteiro.com/download_freeware.shtml.

12.2 message

As messages são widgets parecidos com labels e estão em desuso, pois os labels os substituíram totalmente.

12.3 frames

Frames são molduras, onde os widgets podem ser agrupados. Frames são muito usados para tornar as janelas dos programas mais elegantes e organizadas.

As opções mais importantes suportadas pelos frames são:

Opção	Descrição
-borderwidth	Largura da borda.
-relief	Tipo de relevo 3D: flat, groove, raised, ridge, sunken.

Veja um exemplo:

```
toplevel .t
frame .t.f1 -relief ridge -borderwidth 2
```

```
button .t.f1.b1 -text "OK" -command {exit}
pack .t.f1 -padx 5 -pady 5
pack .t.f1.b1 -padx 5 -pady 5
```

Criará a janela abaixo, com um botão no interior de uma moldura:

12.4 canvas

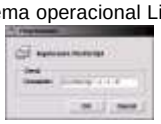
Os canvas são os widgets mais importantes, pois permitem que os programas em Tcl/Tk imprimam documentos PostScript de alta qualidade.

Canvas são áreas para desenho, com suporte a uma grande variedade de objetos vetoriais. De fato, você poderá fazer de tudo com um canvas, até mesmo criar mega-widgets como é o caso da biblioteca Mysunds_MDI e dos megawidgets BWidget.

Um bom exemplo do uso dos canvas para impressão é a biblioteca **TkPrinter**. Veja algumas telas do programa de demonstração da biblioteca TkPrinter e das caixas de diálogo *Visualizar Impressão*, *Imprimir* e *Configurar Impressora*, no Linux e no Windows:

Caixa de diálogo
Visualizar Impressão
no
sistema operacional Linux.

Caixa de diálogo
Imprimir
no
sistema operacional Linux.



Caixa de diálogo
Configurar Impressora
no sistema operacional Linux.

Caixa de diálogo
Imprimir
no sistema operacional Windows.

A biblioteca TkPrinter faz parte do meu conjunto de mega-widgets e você pode fazer o download dela através do meu site: http://www.souzamonteiro.com/download_freeware.shtml. Procure o arquivo widgets-1_0_0.tgz.

Para Linux TkPrinter foi escrito em 100% puro Tcl. Para Windows TkPrinter utiliza uma DLL (tkprint80.dll, tkprint81.dll ou tkprint82.dll), incluída na distribuição. O código fonte das DLLs também está disponível em meu site sob o nome de **tkprint**. As DLLs não foram desenvolvidas por mim, eu desenvolvi apenas o código em Tcl/Tk.

TkPrinter é sensível ao sistema operacional, e selecionará, automaticamente, as bibliotecas necessárias, sem que seu programa precise ser modificado para funcionar no Linux e no Windows.

Os canvas serão estudados em detalhes na próxima aula.

12.5 button

Buttons são botões que podem ser clicados com o mouse. Nós os temos utilizado em todos os nossos exemplos até aqui.

Botões suportam texto e imagens. Mas só texto ou imagem de cada vez, nunca os dois ao mesmo tempo.

As opções mais importantes disponíveis para os botões são:

Opção	Descrição
-command	Comando Tcl para ser executado quando o botão for pressionado.
-image	Imagem para ser exibida no botão.
-text	Texto para ser exibido no botão.
-underline	Índice do caractere que deve ser sublinhado para indicar a tecla de atalho.

Os comando suportados pelos botões são:

Comando	Descrição
flash	Faz o botão piscar, alternando as cores normais e as cores *active*.
invoke	Executa o script -command de um botão.

Veja um exemplo:

```
toplevel .t
frame .t.f1 -relief ridge -borderwidth 2
button .t.f1.b1 -text "OK" -command {exit}
pack .t.f1 -padx 5 -pady 5
pack .t.f1.b1 -padx 5 -pady 5
```

Criará a janela abaixo, com um botão no interior de uma moldura:

Agora execute:

```
.t.f1.b1 configure -activebackground white
.t.f1.b1 flash
```

Fará o botão piscar.

12.6 entry

Entries são caixas de texto para digitação. São muito usadas com bancos de dados e programas que requerem algum tipo de digitação de dados pelo usuário.

Os widgets entry podem ser configurados para verificar a validade dos dados digitados e formatá-los ao final da digitação. Com Tcl/Tk menor que 8.3, utilizamos binds para verificar a validade dos dados. Com Tcl/Tk maior ou igual a 8.3 podemos utilizar as opções especificas de validação disponíveis. Neste curso estudaremos apenas a validação através de binds, pois as opções de Tk 8.3 apenas configuram os binds para nós, e além disso, muitas distribuições do Linux vem com Tcl/Tk 8.0.4, que só permite validação através de binds.

As validações serão estudadas na aula 16 que trata de eventos em Tk.

As principais opções disponíveis para os widgets entry são:

Opção	Descrição
-justify	Determina o alinhamento do texto: left, center, right.
-state	Estado: normal, disabled.
-show	Caractere para ser exibido quando o usuário estiver digitando uma senha.

-textvariable Variável usada para *data-bind*.

Os principais comando disponíveis para os widget entry são:

Comando	Descrição
bbox	Retorna as coordenadas X1 Y1 X2 Y2 do caractere indicado.
delete	Apaga a faixa de caracteres indicados.
get	Retorna a string apresentada no widget entry.
icursor	Exibe o cursor antes do caractere indicado.
index	Retorna o índice numérico correspondente ao índice dado.
insert	Insere a string dada na posição indicada.
scan	Veja os comandos de scroll na aula 29.
selection adjust	Ajusta o final da seleção para o índice fornecido.
selection clear	Remove a seleção.
selection from	Ajusta o ponto âncora para a seleção.
selection present	Retorna 1 se algum caractere estiver selecionado. Caso contrário, 0.
selection range	Seleciona a faixa de caracteres indicados.
selection to	Ajusta o final da seleção.

O widget entry suporta os seguintes índices para os caracteres: *número*, **anchor**, **end**, **insert**, **sel.first**, **sel.last**, *@coordenadaX*.

Exemplos:

```
toplevel .t
entry .t.e -textvariable entrada
label .t.lbl -textvariable entrada -anchor sw -justify left
pack .t.e
pack .t.lbl -expand 1 -fill x
```

Cria uma janela com uma caixa para digitação e um rótulo onde é exibido todo texto digitado pelo usuário.

12.7 label

Labels são rótulos onde podemos exibir texto ou imagem. São widgets muito simples e suas opções são as opções gerais apresentadas na aula 09.

Exemplos:

```
toplevel .t
entry .t.e -textvariable entrada
label .t.lbl -textvariable entrada -anchor sw -justify left
pack .t.e
pack .t.lbl -expand 1 -fill x
```

Cria uma janela com uma caixa para digitação e um rótulo onde é exibido todo texto digitado pelo usuário.

12.8 text

O widget text é um componente extremamente sofisticado, que suporta texto, imagens, hyper-texto e também outras janelas e widgets embutidos.

Com o auxílio da biblioteca HTML Library é possível renderizar HTML 2.0 e algumas tags do HTML 3.0. A biblioteca HTML Library foi escrita em 100% puro Tcl pelos engenheiros da Sun Microsystem e é realmente excelente. Eu a utilizo com uma pequena modificação em meu browser HTML **TkShip**:

Browser HTML TkShip
sendo executado
no sistema operacional Linux

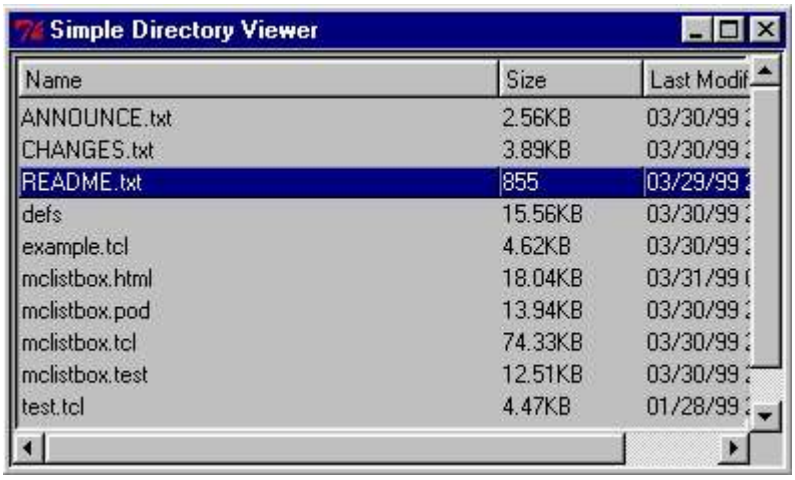
TkShip foi inteiramente escrito em Tcl/Tk e faz parte do meu conjunto de mega-widgets, podendo ser executado como um aplicativo ou como uma biblioteca de ajuda(help). Você pode fazer o download do browser TkShip através do meu site: http://www.souzamonteiro.com/download_freeware.shtml. Procure o arquivo widgets-1_0_0.tgz.

O widget text será estudado em detalhes na aula 14.

12.9 listbox

Listboxes são caixas de listagem, onde são exibidas listas de ítens que podem ser selecionados pelo usuário.

Atualmente as listboxes suportam apenas uma coluna, mas você pode utilizar o mega-widget mclistbox, para exibir tabelas bastante sofisticadas.



Você pode fazer o download do mega-widget mclistbox através do meu site: http://www.souzamonteiro.com/download_freeware.shtml.

As principais opções suportadas pelo widget listbox são:

Opção	Descrição
-xscrollcommand	Comando usado para conectar a scrollbar horizontal.
-yscrollcommand	Comando usado para conectar a scrollbar vertical.
-selectMode	Modo de seleção: single, browse, multiple, extended.

Os comando suportados pelos listboxes são:

Comando	Descrição
activate	Ajusta o elemento ativo para o índice especificado.
bbox	Retorna uma lista {x y largura altura} com as coordenadas do retângulo ao redor do elemento indicado.
curselection	Retorna uma lista com os elementos selecionados.
delete	Remove a faixa de elementos da listbox.
get	Retorna uma lista contendo os elementos na faixa indicada.
index	Retorna um número correspondente ao índice especificado.
insert	Insere um elemento na posição especificada.
nearest	Retorna o índice do elemento próximo à coordenada especificada.
scan	Veja os comandos de scroll na aula 29.
selection anchor	Ajusta a âncora da seleção para o elemento especificado.
selection clear	Remove a seleção na faixa especificada..
selection includes	Retorna 1 se o elemento estiver selecionado. Caso contrário, 0.
selection set	Adiciona todos os elementos na faixa especificada à seleção.
see	Rola a tela para que o elemento especificado se torne visível.
size	Retorna o número de elementos na listbox.
xview	Rola a janela horizontalmente.
yview	Rola a janela verticalmente.

O widget listbox suporta os seguintes índices: *número*(começa com 0), **active**, **anchor**, **end**, *@X,Y*.

Exemplo:

```
toplevel .t
listbox .t.list -selectmode multiple -yscrollcommand {.t.vscr set}
scrollbar .t.vscr -orient vertical -command {.t.list yview}
pack .t.list -side left
pack .t.vscr -side left -expand 1 -fill y
```

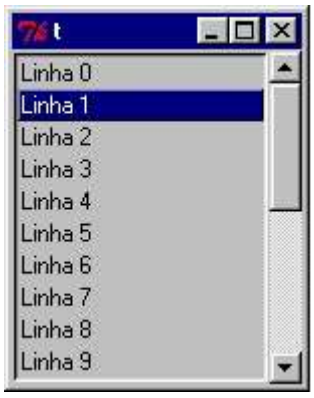
Cria uma janela com uma listbox e uma scrollbar.

```
for {set i 0} {$i <= 20} {incr i} {
   .t.list insert end "Linha $i"
}
```

Preenche a listbox com 21 ítens.

```
bind .t.list <ButtonRelease> {
   bell
   puts "Você selecionou os ítens: [.t.list curselection]"
}
```

Configura a listbox para soar um beep e exibir os ítens selecionados, sempre que o usuário liberar o botão do mouse sobre a listbox.



12.10menu, menubutton

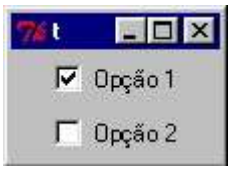
Tk suporta menus bastante sofisticados sendo que seu estudo merece uma aula a parte. Assim esturaremos os menus e botões de menu na aula 15.

12.11checkbutton

Checkbuttons são pequenos botões que podem ser selecionados pelo usuário. Mas fácil que falar de um botão de checagem é mostrá-lo:

```
toplevel .t
checkbutton .t.chk1 -text "Opção 1" -variable chkopcao1 -onvalue 1
checkbutton .t.chk2 -text "Opção 2" -variable chkopcao2 -onvalue 1
pack .t.chk1 .t.chk2
set chkopcao1 1
set chkopcao2 0
```

Exibe uma janela com dois botões de checagem, estando o primeiro botão selecionado.



Checkbuttons podem ser conectados a bancos de dados através da opção **-variable**, que especifica uma variável para coneção.

As principais opções suportadas pelos widgets checkbutton são:

Opção	Descrição
-command	Comando a ser executado quando o checkbutton for clicado.
-indicatoron	Especifica se o indicador de seleção deve ser exibido ou não. Se 1 exibe um indicador. Caso contrário, não exibe o indicador.
-offvalue	Valor que a variável indicada por -variable deve apresentar para que o checkbutton não esteja selecionado.
-onvalue	Valor que a variável indicada por -variable deve apresentar para que o checkbutton esteja selecionado.
-selectcolor	Cor a ser usada no indicador quando ele estiver selecionado.
-selectimage	Imágem a ser exibida no indicador quando ele estiver selecionado.
-variable	Variável conectada ao checkbutton.

Os principais comandos suportados pelo widget checkbutton são:

Comando	Descrição
deselect	Remove a seleção do checkbutton.
flash	Faz o checkbutton piscar, alternando as cores normais e as cores *active*.
invoke	Executa o script -command do checkbutton.
select	Seleciona o checkbutton.
toggle	Troca o estado atual do checkbutton.

12.12radiobutton

Radiobuttons são semelhantes a checkbuttons mas são mutuamente exclusivos. Somente um botão de rádio pode estar selecionado em um grupo de botões. Mas fácil que falar de um botão de rádio é mostrá-lo:

```
toplevel .t
radiobutton .t.rad1 -text "Opção 1" -variable opcao -value 1
radiobutton .t.rad2 -text "Opção 2" -variable opcao -value 0
pack .t.rad1 .t.rad2
set opcao 1
```

Exibe uma janela com dois botões de rádio, estando o primeiro botão selecionado.

Radiobuttons podem ser conectados a bancos de dados através da opção **-variable**, que especifica uma variável para coneção.

As principais opções suportadas pelos widgets radiobutton são:

Opção	Descrição
-command	Comando a ser executado quando o radiobutton for clicado.
-indicatoron	Especifica se o indicador de seleção deve ser exibido ou não. Se 1 exibe um indicador. Caso contrário, não exibe o indicador.
-selectcolor	Cor a ser usada no indicador quando ele estiver selecionado.
-selectimage	Imágem a ser exibida no indicador quando ele estiver selecionado.
-value	Valor que a variável indicada por -variable deve apresentar para que o radiobutton esteja selecionado.
-variable	Variável conectada ao radiobutton.

Os principais comandos suportados pelo widget radiobutton são:

Comando	Descrição
deselect	Remove a seleção do radiobutton.
flash	Faz o radiobutton piscar, alternando as cores normais e as cores *active*.
invoke	Executa o script -command do radiobutton.
select	Seleciona o radiobutton.

12.13scrollbar

Scrollbars são controles que permitem rolar, horizontal ou verticalmente, o conteúdo de outro widget.

O exemplo a seguir mostra uma janela com um widget listbox e uma scrollbar vertical conectada a ele:

```
toplevel .t
listbox .t.list -selectmode multiple -yscrollcommand {.t.vscr set}
scrollbar .t.vscr -orient vertical -command {.t.list yview}
pack .t.list -side left
pack .t.vscr -side left -expand 1 -fill y
```

Cria uma janela com uma listbox e uma scrollbar.

```
for {set i 0} {$i <= 20} {incr i} {
   .t.list insert end "Linha $i"
}
```

Preenche a listbox com 21 ítens.

```
bind .t.list <ButtonRelease> {
   bell
   puts "Você selecionou os ítens: [.t.list curselection]"
}
```

Configura a listbox para soar um beep e exibir os ítens selecionados, sempre que o usuário liberar o botão do mouse sobre a listbox.

As scrollbars não têm vida isolada e só fazem sentido quando conectadas a algum widget.

As scrollbars suportam as seguintes opções:

Opção	Descrição
-activerelief	Relevo usado no elemento ativo.
-command	Comando a ser executado quando o slider for movimentado.
-elementborderwidth	Largura usada nas bordas dos elementos internos..
-width	A dimensão mais estreita da scrollbar.

Os principais comandos suportados pelo widget scrollbar são:

Comando	Descrição
activate	Exibe o elemento com os atributos de ativo. Elementos podem ser: arrow1, trough1, slider, trough2 ou arrow2.
delta	Retorna uma fração da posição do slider para um movimento deltaX deltaY.
fraction	Retorna um número entre 0 e 1 representando o ponto X Y na área de deslizamento da scrollbar.
get	Retorna os ajustes da scrollbar como uma lista contendo: {primeiro último}.
identify	Retorna o nome do elemento nas coordenadas X Y.
set	Move o *slider* para a posição representada pela porcentagem *primeiro último* na área de deslizamento do slider.

12.14scale

Scales são escalas deslizantes. São muito úteis para criar controles como os de tonalidade para scanners. Veja a GUI para o scanner paralelo Primax, escrita em 100% puro Tcl e disponível em http://www.souzamonteiro.com/download_freeware.shtml.

Veja o exemplo:

```
global texto
toplevel .t
scale .t.scl -orient horizontal -from 0 -to 100 -resolution 1 -variable valor -command
escala:moveu
label .t.lbl -textvariable texto -anchor sw -justify left
pack .t.scl -side top -expand 1 -fill x
pack .t.lbl -side top -expand 1 -fill x
set valor 0
proc escala:moveu {posicao}  {
   global texto
   set texto "Você ajustou a escala para $posicao"
}
```

O exemplo criará uma janela com uma escala e um rótulo:

As scales suportam as seguintes opções:

Opção	Descrição
-bigincrement	Valor a ser usado para um *grande incremento* da escala.
-command	Comando a ser executado quando o valor da escala for alterado. Deve ser o nome de um comando que receba como argumento o valor da escala.
-digits	Número de algarismos significativos que devem ser retidos.
-from	Menor valor da escala.
-label	Rótulo a ser exibido sob a escala.
-length	Largura ou altura da escala.
-resolution	Valor do incremento.
-showvalue	Se 1, exibe o valor da escala no rótulo. Se 0, não exibe o valor da escala.
-sliderlength	Largura ou altura do slider.
-sliderrelief	Relevo do slider.
-tickinterval	Intervalo entre os ticks da escala.
-to	Maior valor da escala.
-variable	Variável que conterá ou ajustará o valor da escala.
-width	Largura do elemento mais estreito da escala.

Os principais comandos suportados pelo widget scale são:

Comando	Descrição
coords	Retorna as coordenadas X e Y correspondentes ao valor especificado.
get	Retorna o valor atual da escala, ou caso sejam fornecidos os valores X Y, retorna o valor sob as coordenadas especificadas.

identify Retorna o nome da parte da escala sob as coordenadas especificadas: *slider*, *trough1* ou *trough2*.

set Muda o valor da escala.

13 O widget canvas

O widget canvas provê uma área de desenho vetorial para os programas em Tcl/Tk. Os canvas possuem um número enorme de comandos e opções e serão estudados, em detalhes, nesta aula. A utilização do widget canvas para impressão será estudada na aula 20.

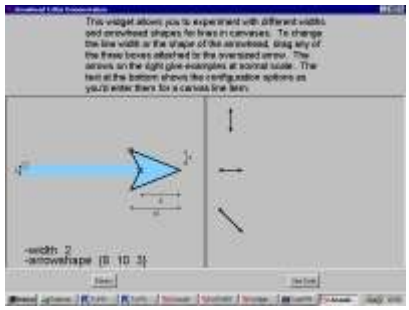
Canvas são criados através do comando **canvas**:

```
toplevel .t
canvas .t.c
pack .t.c -expand 1 -fill both
.t.c create rectangle 5 5 50 50 -fill blue
.t.c create arc 100 100 200 200 -style pieslice -start 45 -extent 225 -fill red
```

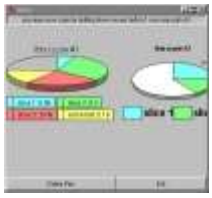
Criará uma janela com um canvas em seu interior e desenhará um retângulo azul e um semi-círculo vermelho dentro do canvas.

Veja como um canvas pode ser poderoso:

Planta baixa
de uma construção em um canvas



Um editor de setas
construído com um canvas



Gráficos de pizza
em um canvas
utilizando a biblioteca
tkpiechart 5.3

A seguir serão apresentadas as principais opções do widget canvas:

Opção	Descrição
-closeenough	Determina o quão próximo deve estar o cursor do mouse para que seja considerado como estando no interior de um ítem.
-confine	Determina se será possível visualizar além da área de scroll. Se 1, permite a visualização. Caso contrário, não permite.
-scrollregion	Determina a área de scroll. No formato: {X1 Y1 X2 Y2}.
-xscrollincrement	Determina o incremento para rolagem horizontal da janela do canvas.
-yscrollincrement	Determina o incremento para rolagem vertical da janela do canvas.

O canvas suporta as seguintes unidades de medida: pixels, polegadas(**i**), centímetros(**c**), milímetros(**m**) e pontos(**p**). Veja os exemplos:

```
Valor em pixels:      10
Valor em polegadas:   10i
Valor em centímetros: 10c
Valor em milímetros:  10m
Valor em pontos:      10p
```

As coordenadas crescem de cima para baixo de da esquerda para a direita.

Os seguintes índices são suportados: *número*, **end**, **insert**, **sel.first**, **sel.last**, *@x,y*.

A seguir serão apresentadas os comandos do widget canvas:

Comando	Descrição
addtag	Adiciona uma tag à lista de tags, associada a cada ítem que satisfaca a especificação dada.
bbox	Retorna uma lista com quatro elementos, correspondentes às coordenadas do retângulo envolvendo os ítens cujas tags sejam dadas.
bind	Associa eventos aos ítens cujas tags sejam dadas.
canvasx	Retorna a coordenada X dentro do canvas, que corresponde à coordenada X da tela.
canvasy	Retorna a coordenada Y dentro do canvas, que corresponde à coordenada Y da tela.
coords	Retorna ou define as coordenadas de um ítem.
create	Cria um novo ítem, do tipo especificado.
dchars	Apaga os caracteres dentro da faixa especificada, nos ítens cuja tag for especificada.
delete	Apaga os ítens cujas tags sejam especificadas.
dtag	Remove a tag especificada.
find	Retorna os ítens que satisfaçam as especificações dadas. Veja as especificações logo a seguir.
focus	Ajusta o foco para o ítem dado.
gettags	Retorna a lista de todos os ítens associados à tag dada.
icursor	Ajusta a posição do cursor, para inserção, dentro dos ítens cuja tag seja dada.
index	Retorna uma string numérica dando a posição do caractere, dentro do ítem, cuja tag seja dada.
insert	Insere texto dentro do ítem cuja tag seja dada.
itemcget	Retorna o valor da opção indicada, no ítem cuja tag seja dada.
itemconfigure	Configura o valor da opção indicada, no ítem cuja tag seja dada.
lower	Coloca o ítem cuja tag seja dada, abaixo do ítem indicado.
move	Move os ítens cuja tag seja dada, para a posição indicada no canvas. A nova posição será um deslocamento da posição atual.
postscript	Gera um arquivo *Encapsulated Postscript*, representando a parte do canvas especificada. Suporta muitas opções que serão tratadas mais adiante.
raise	Coloca o ítem cuja tag seja dada, acima do ítem indicado.
scale	Reescala os ítens indicados pela tag dada.
scan	Veja os comandos de scroll na aula 29.
select adjust	Ajusta o final da seleção no ítem indicado, para uma nova posição.
select clear	Remove a seleção do ítem indicado.
select from	Ajusta a âncora da seleção no ítem indicado.

select item	Retorna a identificação do ítem selecionado.
select to	Ajusta a seleção no ítem especificado.
type	Retorna o tipo do primeiro ítem apontado pela tag dada.
xview	Rola a janela do canvas horizontalmente.
yview	Rola a janela do canvas verticalmente.

Veja as especificações de procura para o comando **find**:

Especificação	Descrição
above	Seleciona o ítem que esteja depois do ítem cuja tag seja dada.
all	Seleciona todos os ítens dentro do canvas.
bellow	Seleciona o ítem que esteja antes do ítem cuja tag seja dada.
closest	Seleciona o ítem que esteja mais próximo das coordenadas dadas depois do ítem dado.
enclosed	Seleciona todos os ítens que estejam dentro da área X1 Y1 X2 Y2 dada.
overlapping	Seleciona todos os ítens que ultrapassam ou estejam dentro da área X1 Y1 X2 Y2 dada.
withtag	Seleciona todos os ítens cuja tag seja dada..

Veja os tipos de ítens suportado pelo widget canvas, e suas opções:

13.1 arc

nome_do_canvas **create arc** *x1 y1 x2 y2* [*opção valor* ...]

Opção	Descrição
-fill	Cor do preechimento.
-outline	Cor da linha.
-stipple	Um bitmap.
-tags	Nome da tag para identificar o objeto.
-width	Largura da linha.
-extent	Ângulo final.
-outlinestipple	Bitmap usado para desenhar a linha.
-start	Ângulo inicial.
-style	Estilo: pieslice(fatia de pizza), chord(corda), ou arc(arco).

13.2 bitmap

nome_do_canvas **create bitmap** *x y* [*opção valor* ...]

Opção	Descrição
-anchor	Posição da âncora.
-background	Cor de fundo.
-bitmap	O bitmap.
-foreground	Cor de primeiro plano.
-tags	Nome da tag para identificar o objeto.

13.3 image

nome_do_canvas **create image** *x y* [*opção valor* ...]

Opção	Descrição
-anchor	Posição da âncora.

-image	A imagem.
-tags	Nome da tag para identificar o objeto.

13.4 line

nome_do_canvas **create line** *x1 y1 x2 y2 ... xN yN* [*opção valor* ...]

Opção	Descrição
-fill	Cor do preechimento.
-smooth	Se 1 a linha será desenhada como uma curva. Caso contrário, não será uma curva.
-stipple	Um bitmap.
-tags	Nome da tag para identificar o objeto.
-width	Largura da linha.
-arrow	Especifica se a linha será terminada por uma seta: none, first, last, both.
-arrowshape	Uma lista com três elementos que descrevem a seta.
-capstyle	Como desenhar os pontos finais da linha: butt, projecting, round.
-joinstyle	Como devem ser as junções dos vértices: bevel, miter, round.
-splinesteps	Ângulo da curva.

13.5 oval

nome_do_canvas **create oval** *x1 y1 x2 y2* [*opção valor* ...]

Opção	Descrição
-fill	Cor do preechimento.
-outline	Cor da linha.
-stipple	Um bitmap.
-tags	Nome da tag para identificar o objeto.
-width	Largura da linha.

13.6 polygon

nome_do_canvas **create polygon** *x1 y1 x2 y2 ... xN yN* [*opção valor* ...]

Opção	Descrição
-fill	Cor do preechimento.
-smooth	Se 1 a linha será desenhada como uma curva. Caso contrário, não será uma curva.
-stipple	Um bitmap.
-tags	Nome da tag para identificar o objeto.
-width	Largura da linha.
-splinesteps	Ângulo da curva.

13.7 rectangle

nome_do_canvas **create rectangle** *x1 y1 x2 y2* [*opção valor* ...]

Opção	Descrição
-fill	Cor do preechimento.
-outline	Cor da linha.
-stipple	Um bitmap.

-tags	Nome da tag para identificar o objeto.
-width	Largura da linha.

13.8 text

nome_do_canvas **create text** *x y* [*opção valor* ...]

Opção	Descrição
-anchor	Âncora.
-fill	Cor do preechimento.
-font	Nome e propriedades da fonte.
-justify	Alinhamento do texto: left, right ou center.
-stipple	Um bitmap.
-tags	Nome da tag para identificar o objeto.
-text	String a ser exibida.
-width	Largura da linha de texto. Se 0, a quebra ocorrerá somente se for encontrado um caractere de quebra de linha "\n".

13.9 window

nome_do_canvas **create window** *x y* [*opção valor* ...]

Opção	**Descrição**
-anchor	Âncora.
-height	Altura da janela.
-width	Largura da janela.
-window	Nome da janela que deve ser associada ao ítem.
-tags	Nome da tag para identificar o objeto.

A seguir estão as opções suportadas pelo comando **postscript**:

Opção	Descrição
-colormap	Especifica uma variável que deve conter um vetor cujos elementos contém o código Postscript para uma cor em particular.
-colormode	Pode ser: color, grey ou mono.
-file	Nome do arquivo a ser gerado. Caso não seja fornecido nenhum nome, o *Postscript* será retornado como resultado da execução do comando.
-fontmap	Especifica o mapa da fonte que deve ser usado. É uma variável contendo um vetor cujos elementos são listas de dois elementos: {*font size*}.
-height	Altura da área do canvas que deve ser impressa. O default é a altura do canvas.
-pageanchor	Âncora da posição da impressão sobre o papel. O default é center.
-pageheight	Altura da impressão no papel.
-pagewidth	Largura da impressão no papel.
-pagex	Ponto de posicionamento X.
-pagey	Ponto de posicionamento Y.
-rotate	Se 1, rotaciona a pagina de 90 graus: "*landscape*". Caso contrário, não rotaciona.
-width	Largura da área do canvas que deve ser impressa. O default é a largura do canvas.
-x	Coordenada X do canto superior esquerdo da área do canvas a ser impressa.
-y	Coordenada Y do canto superior esquerdo da área do canvas a ser impressa.

As opções PostScript serão estudadas na aula 20, que trata de técnicas de impressão.

O Exemplo a seguir mostra como "soltar" os objetos dentro de um canvas, para que eles possam ser posicionados com o mouse:

```
global mover1 mover2
set mover1 0
set mover2 0
toplevel .t
canvas .t.c
pack .t.c -expand 1 -fill both
.t.c create rectangle 5 5 50 50 -fill blue -tags item1
.t.c create arc 100 100 200 200 -style pieslice -start 45 -extent 225 -fill red -tags item2
```

Criará uma janela com um canvas em seu interior e desenhará um retângulo azul e um semi-círculo vermelho dentro do canvas.

```
.t.c bind item1 <ButtonPress> {set mover1 1}
.t.c bind item1 <ButtonRelease> {set mover1 0}
.t.c bind item1 <Motion> {
   if {$mover1 == 1} {
      .t.c move item1 [expr %x - [lindex [.t.c coords item1] 0]] [expr %y - [lindex [.t.c coords item1] 1]]
   }
}
```

Configura o item item1, o retângulo, para que se movimente para a posição do mouse sobre a janela caso o botão do mouse esteja pressionado sobre o ele.

```
.t.c bind item2 <ButtonPress> {set mover2 1}
.t.c bind item2 <ButtonRelease> {set mover2 0}
.t.c bind item2 <Motion> {
   if {$mover2 == 1} {
      .t.c move item2 [expr %x - [lindex [.t.c coords item2] 0]] [expr %y - [lindex [.t.c coords item2] 1]]
   }
}
```

Configura o item item2, o semi-círculo, para que se movimente para a posição do mouse sobre a janela caso o botão do mouse esteja pressionado sobre o ele.

14 O widget text

O widget text oferece uma área para renderização de hyper-texto, que pode ser desde texto baseado nas *tags* Tk até SGML, HTML e XML. Para renderização de texto HTML pode-se usar a biblioteca HTML Library 0.3, já para trabalhar com XML ou SGML é preciso utilizar o parser TclXML 1.2 e implementar as rotinas para renderizar o documento. A biblioteca HTML Library 0.3 suporta apenas HTML 2.0. Pode ser uma boa ocasião para começar-mos a pensar em implementar um *motor* para HTML 4, usando TclXML.

Veja como o widget text é poderoso:

Browser TkShip implementado
utilizando o widget text

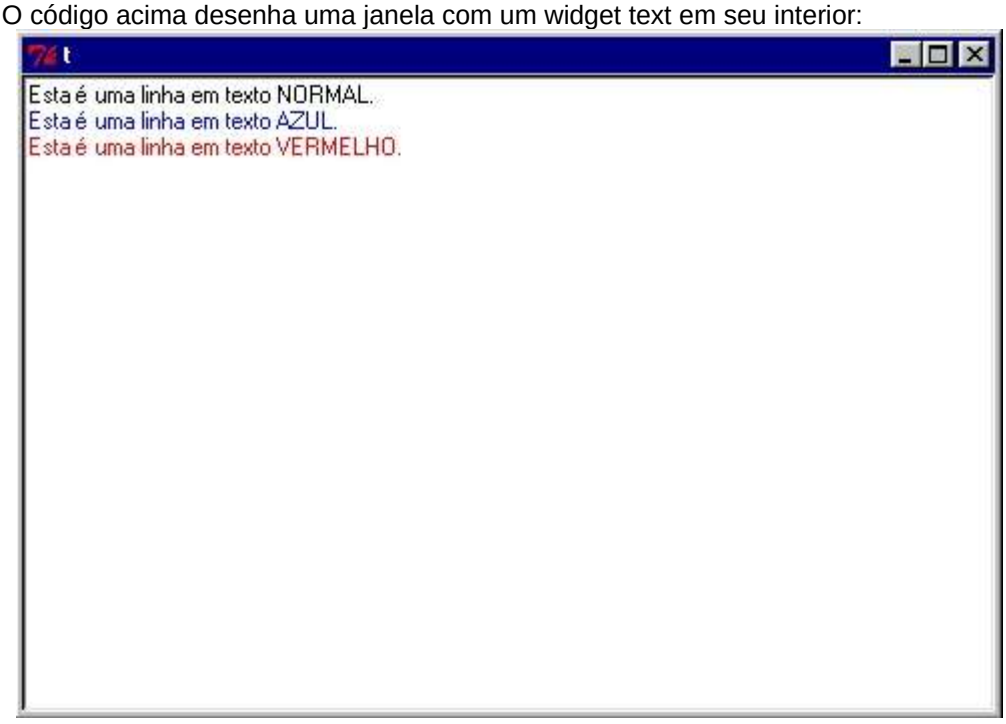
A imagem acima mostra o browser TkShip que eu implementei utilizando o widget text e a biblioteca HTML Library 0.3. Fiz algumas modificações na biblioteca, mas foi apenas para melhorar o aspecto da pagina.

Você pode obter o mesmo efeito, inserindo texto no widget e configurando *tags* Tk para cada parte do documento, mas isso, em geral, não vale a pena. É melhor usar HTML.

Veja um exemplo:

```
toplevel .t
text .t.text
pack .t.text -expand 1 -fill both
.t.text tag configure AZUL -foreground blue
.t.text tag configure VERMELHO -foreground red
.t.text insert end "Esta é uma linha em texto NORMAL.\n"
.t.text insert end "Esta é uma linha em texto AZUL.\n"
.t.text insert end "Esta é uma linha em texto VERMELHO.\n"
.t.text tag add AZUL 2.0 2.31
.t.text tag add VERMELHO 3.0 3.35
```

O código acima desenha uma janela com um widget text em seu interior:



Vejamos agora cada uma das opções suportadas pelo widget text:

Opção	Descrição
-spacing1	Espaço acima do parágrafo. Utiliza unidades de tela.

-spacing2 Espaço entre parágrafos. Utiliza unidades de tela.

-spacing3 Espaço abaixo do parágrafo. Utiliza unidades de tela.

-tabs Lista de Tabulação, dadas como distâncias em unidades de tela. Cada Tabulação pode ser seguida de um dos modificadores: left, right, center ou numeric.

-wrap Especifica a quebra de linha. Pode ser: none, char ou word.

Os comandos suportados pelo widget text são:

Comando	Descrição
bbox	Retorna uma lista com as coordenadas de um retângulo em volta do caractere indicado, na forma: {x y largura altura}.
compare	Compara os índices de acordo com o operador relacional indicado, na forma: *índice1 operador indice2*.
delete	Apaga a faixa de caracteres indicada.
dlineinfo	Retorna uma lista na fora {x y largura altura linha_de_base} descrevendo a área na tela ocupada pela linha indicada.
dump	Retorna informações detalhadas sobre o texto na faixa dada. Suporta as chaves: -all, -mark, -tag, -text e -window, para especificar o tipo de informação desejada. É possível especificar a chave -command para executar um comando para cada tipo de elemento dentro da faixa dada.
get	Retorna a string de caracteres na faixa indicada.
image cget	Retorna o valor de uma opção na imagem indicaca pelo índice dado.
image configure	Configura o valor de uma opção na imagem indicaca pelo índice dado.
image create	Cria uma imagem na posição indicada pelo índice dado.
image names	Retorna os nomes de todas as imagens no widget.
index	Retorna a posição do índice na forma *linha.coluna*.
insert	Insere texto no widget, aplicando as tags indicadas.
mark gravity	Retorna ou configura o caractere adjacente à marca dada. Pode ser configurado para left ou right.
mark names	Rerorna os nomes de todas as marcas configuradas, em uma lista.
mark next | previous	Retorna o nome da próxima/anterior marca após/antes do índice.
mark set	Aplica a marca indicada no índice dado.
mark unset	Remove cada marca indicada.
scan	Veja os comandos de scroll na aula 29.
search	Retorna o índice do primeiro caractere correspondente à expressão especificada na faixa de texto indicada. Suporta as seguintes chaves: -forward, -backward, -exact, -regexp, -count *variável*, -nocase.
see	Ajusta a janela para que o caractere apontado pelo índice dado, seja visível.
tag add	Aplica uma *tag* aos caracteres na faixa dada.
tag bind	Configura um evento para uma *tag* dada.
tag cget	Retorna o valor de uma opção para a *tag* dada.
tag configure	Configura o valor de uma opção para a *tag* dada.
tag delete	Apaga as informações existentes na *tag* dada.
tag lower	Muda a prioridade da *tag* dada para que ela fique abaixo da outra *tag* dada.
tag names	Retorna uma lista com os nomes de todas as *tags* associadas com o caractere sob o índice dado.
tag nextrange	Procura nos caracteres na faixa dada pela primeira região coberta pela tag dada. Retorna a faixa de caracteres da região encontrada.

tag prevrange	Executa a mesma função do comando nextrange, mas procura de traz para a frente..
tag raise	Muda a prioridade da *tag* dada para que ela fique acima da outra *tag* dada.
tag ranges	Retorna todas as faixas de caracteres cobertos pela *tag* dada.
tag remove	Remove a *tag* dada da faixa de caracteres dada.
window cget	Retorna o valor da opção dada da janela embutida dada.
window configure	Configura o valor da opção dada da janela embutida dada.
window create	Cria uma janela embutida no índice dado.
window names	Retorna os nomes de todas as janelas embutidas no widget text.
xview	Rola a janela horizontalmente.
yview	Rola a janela verticalmente.

O widget text suporta os seguintes índices, na forma *base modificador*:

Base: *linha***.***coluna*, *@X*,*Y*, **end**, *marca*, *tag***.first**, *tag***.last**, *caminho_para_a_janela_embutida*, *nome_da_imagem*.

Modificador: **+** ou **-***contador* **chars**, **+** ou **-** *contador* **lines**, **linestart**, **lineend**, **wordstart**, **wordend**.

As *tags* suportam as seguintes opções, além daquelas suportadas pelo widget text:

Opção	Descrição
-bgstipple	Um bitmap padrão para o fundo.
-fgstipple	Um bitmap padrão para o frente.
-lmargin1	Margem esquerda da primeira linha do parágrafo.
-lmargin2	Margem esquerda do corpo do parágrafo.
-offset	Deslocamento da linha de base, em relação à linha de base normal.
-overstrike	Se 1, o texto será riscado. Caso contrário, o texto não será riscado.
-rmargin	Margem direita para todas as linhas sob esta tag.
-tabs	Tabulações na forma de uma lista: {*tab1 tab2 tab3... tabN*}.
-underline	Se 1, o texto será sublinhado. Caso contrário, o texto não será sublinhado.

As janelas embutidas suportam as seguintes opções:

Opção	Descrição
-align	Alinhamento vertical da janela, em relação à linha onde está inserida: top, center, bottom, baseline.
-create	Executa o *script* indicado, que deve retornar o *nome_de_caminho* da janela. Deve ser utilizado, caso a opção -window não seja fornecida.
-padx	Espaço entre as laterais esquerda e direita da janela e os outros ítens no widget text. Em unidades de tela.
-pady	Espaço entre as laterais superior e inferior da janela e os outros ítens no widget text. Em unidades de tela.
-stretch	Se 1, a janela se esticará verticalmente para preencher a linha. Caso contrário, a janela não se esticará.
-window	Nome da janela.

As imagens embutidas suportam as seguintes opções:

Opção	Descrição

-align Alinhamento vertical da imagem, em relação à linha onde está inserida: top, center, bottom, baseline.

-image Uma imagem criada com o comando image.

-nome Nome da imagem.

-padx Espaço entre as laterais esquerda e direita da janela e o outros ítens no widget text. Em unidades de tela.

-pady Espaço entre as laterais superior e inferior da janela e o outros ítens no widget text. Em unidades de tela.

Exemplos

O exemplo a seguir foi adaptado do material que acompanha o Kit de instalação do Tcl/Tk 8.3.1:

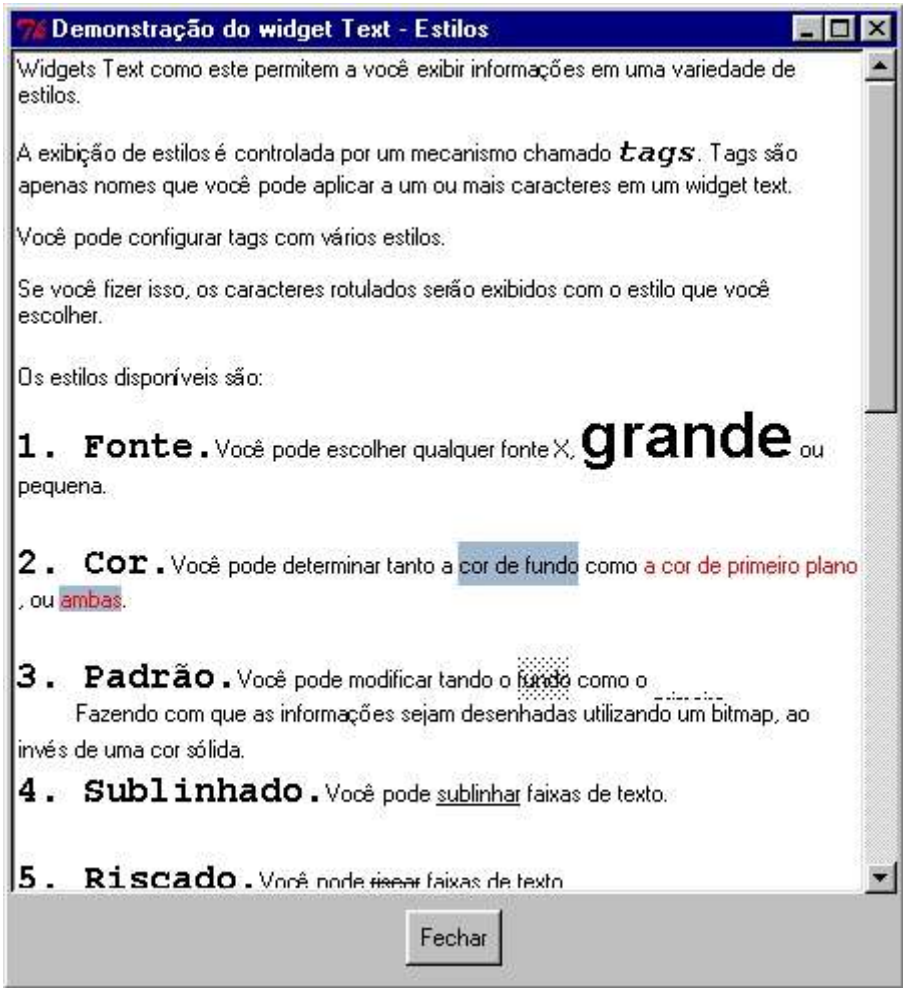
```
#
# Este script cria um widget text que ilustra os vários tipos de estilos que podem
# ser configurados para as tags
#
# Desenha a janela
set w .style
catch {destroy $w}
toplevel $w
wm title $w "Demonstração do widget Text - Estilos"
wm iconname $w "style"
frame $w.buttons
pack $w.buttons -side bottom -fill x -pady 2m
button $w.buttons.dismiss -text Fechar -command "destroy $w"
pack $w.buttons.dismiss -side left -expand 1
text $w.text -yscrollcommand "$w.scroll set" -setgrid true -width 70 -height 32 -wrap word
scrollbar $w.scroll -command "$w.text yview"
pack $w.scroll -side right -fill y
pack $w.text -expand yes -fill both
# Configura os estilos
$w.text tag configure bold -font {Courier 12 bold italic}
$w.text tag configure big -font {Courier 14 bold}
$w.text tag configure verybig -font {Helvetica 24 bold}
if {[winfo depth $w] > 1} {
    $w.text tag configure color1 -background #a0b7ce
    $w.text tag configure color2 -foreground red
    $w.text tag configure raised -relief raised -borderwidth 1
    $w.text tag configure sunken -relief sunken -borderwidth 1
} else {
    $w.text tag configure color1 -background black -foreground white
    $w.text tag configure color2 -background black -foreground white
    $w.text tag configure raised -background white -relief raised -borderwidth 1
    $w.text tag configure sunken -background white -relief sunken -borderwidth 1
}
$w.text tag configure bgstipple -background black -borderwidth 0 -bgstipple gray12
$w.text tag configure fgstipple -fgstipple gray50
$w.text tag configure underline -underline on
$w.text tag configure overstrike -overstrike on
```

```
$w.text tag configure right -justify right
$w.text tag configure center -justify center
$w.text tag configure super -offset 4p -font {Courier 10}
$w.text tag configure sub -offset -2p -font {Courier 10}
$w.text tag configure margins -lmargin1 12m -lmargin2 6m -rmargin 10m
$w.text tag configure spacing -spacing1 10p -spacing2 2p -lmargin1 12m -lmargin2 6m -rmargin 10m
# Insere o texto
$w.text insert end {Widgets Text como este permitem a você exibir informações em uma variedade de estilos.
A exibição de estilos é controlada por um mecanismo chamado }
$w.text insert end tags bold
$w.text insert end {. Tags são apenas nomes que você pode aplicar a um ou mais caracteres em um widget text.
Você pode configurar tags com vários estilos.
Se você fizer isso, os caracteres rotulados serão exibidos com o estilo que você escolher.
Os estilos disponíveis são:}
$w.text insert end "\n1. Fonte." big
$w.text insert end "Você pode escolher qualquer fonte X, "
$w.text insert end grande verybig
$w.text insert end " ou "
$w.text insert end "pequena.\n"
$w.text insert end "\n2. Cor." big
$w.text insert end "Você pode determinar tanto a "
$w.text insert end "cor de fundo" color1
$w.text insert end " como "
$w.text insert end "a cor de primeiro plano" color2
$w.text insert end "\n, ou "
$w.text insert end ambas {color1 color2}
$w.text insert end ".\n"
$w.text insert end "\n3. Padrão." big
$w.text insert end "Você pode modificar tando o "
$w.text insert end fundo bgstipple
$w.text insert end " como o "
$w.text insert end "primeiro plano." fgstipple
$w.text insert end {Fazendo com que as informações sejam desenhadas utilizando um bitmap, ao invés de uma cor sólida.}
$w.text insert end "\n4. Sublinhado." big
$w.text insert end "Você pode "
$w.text insert end sublinhar underline
$w.text insert end " faixas de texto.\n"
$w.text insert end "\n5. Riscado." big
$w.text insert end "Você pode "
$w.text insert end riscar overstrike
$w.text insert end " faixas de texto.\n"
$w.text insert end "\n6. Efeitos 3-D." big
$w.text insert end {Você pode modificar o fundo, para que ele seja desenhado com bordas que farão os caracteres parecerem estar }
$w.text insert end "em alto relevo" raised
```

```
$w.text insert end " ou "
$w.text insert end "em baixo relevo" sunken
$w.text insert end ".\n"
$w.text insert end "\n7. Alinhamento." big
$w.text insert end "Você pode fazer com que as linhas sejam\n"
$w.text insert end "alinhadas pela esquerda,\n"
$w.text insert end "alinhadas pela direita, ou\n" right
$w.text insert end "centralizadas.\n" center
$w.text insert end "\n8. Sobre-escrido e sub-escrito."  big
$w.text insert end "Você pode controlar a "
$w.text insert end "posição vertical do texto para criar sobre-escritos como 10"
$w.text insert end "n" super
$w.text insert end " ou sub-escritos como X"
$w.text insert end "i" sub
$w.text insert end ".\n"
$w.text insert end "\n9. Margens." big
$w.text insert end "Você pode controlar o espaçamento lateral"
$w.text insert end " em\ncada lado do texto:\n"
$w.text insert end "Este parágrafo é um exemplo de uso de " margins
$w.text insert end "margens. Ele consiste de uma simples linha de texto " margins
$w.text insert end "quebrada na tela. Existem duas especificações " margins
$w.text insert end "para margens à esquerda, uma para a primeira linha " margins
$w.text insert end "exibida, associada com a linha de texto, " margins
$w.text insert end "e uma para as linha subsequentes,  " margins
$w.text insert end "que serão quebradas caso necessário. Há também uma " margins
$w.text insert end "especificação separada para a margem direita, " margins
$w.text insert end "a qual é usada para determinar o ponto para a quebra de linha.\n" margins
$w.text insert end "\n10. Espaçamento." big
$w.text insert end "Você pode controlar o espaço entre linhas com três\n"
$w.text insert end "parâmetros separados. \"Spacing1\" determina "
$w.text insert end "o espaço extra deixado\nsobre uma linha, \"spacing3\" "
$w.text insert end "determina o espaço extra deixado abaixo de uma linha,\ne "
$w.text insert end "e se a linha de texto será quebrada, \"spacing2\" determina o "
$w.text insert end "espaço extra deixado\nentre as linhas "
$w.text insert end "que compõem o texto.\n"
$w.text insert end "Este parágrafo endentado, ilustra como o espaçamento " spacing
$w.text insert end "pode ser usado. Cada parágrafo, é uma " spacing
$w.text insert end "única linha no widget text, o qual sofre " spacing
$w.text insert end "uma quebra de linha gerada pelo widget.\n" spacing
$w.text insert end "Spacing1 está configurado para 10 pontos neste texto, " spacing
$w.text insert end "o que resulta em saltos relativamente grandes entre " spacing
$w.text insert end "os parágrafos. Spacing2 está configurado para 2 pontos, " spacing
$w.text insert end "oque resulta em apenas um pequeno espaçamento " spacing
$w.text insert end "no parágrafo. Spacing3 não está sendo usado " spacing
$w.text insert end "neste exemplo.\n" spacing
$w.text insert end "Para ver como o espaçamento funciona, selecione uma faixa de texto "
spacing
$w.text insert end "neste parágrafo. A seleção " spacing
```

$w.text insert end "cobrirá o espaçamento adicional." spacing

É isso. Copie e cole o fragmento de código acima na linha de comando do **wish**. Você obterá a janela abaixo:



15 Criando Menus

Tk suporta todos os tipos de menus existentes no UNIX e no Windows. Entretanto, para utilizar menus que combinam imagens e texto em uma mesma opção, será necessário utilizar o conjunto de mega-widgets **Tix**, que serão vistos na aula 21.

Veja um exemplo a seguir, adaptado do código fonte do meu browser TkShip:

```
toplevel .w1

wm protocol .w1 WM_DELETE_WINDOW {exit}

wm title .w1 "HTML Browser"
wm resizable .w1 1 1
# Centraliza a janela na tela
set x [expr {([winfo screenwidth .w1] - 640)/2}]
set y [expr {([winfo screenheight .w1] - 480)/2}]
```

```
#
# Cria os paineis
#
# Painel principal
frame .w1.fr0 -borderwidth 2 -relief raised
# Painel do menu
frame .w1.fr0.fr1 -borderwidth 2 -relief raised
# Painel da barra de ferramentas
frame .w1.fr0.fr2 -borderwidth 2 -relief raised
# Painel da barra de navegacao
frame .w1.fr0.fr3 -borderwidth 2 -relief raised
# Painel da janela do browser
frame .w1.fr0.fr4 -borderwidth 2
# Painel da barra de status
frame .w1.fr0.fr5 -borderwidth 2 -relief raised
#
# Cria os componentes dos paineis
#
# Cria os menus
menubutton .w1.fr0.fr1.file -text "Arquivo" -underline 0 -menu .w1.fr0.fr1.file.menu
menubutton .w1.fr0.fr1.edit -text "Editar" -underline 0 -menu .w1.fr0.fr1.edit.menu
menubutton .w1.fr0.fr1.options -text "Opcoes" -underline 0 -menu .w1.fr0.fr1.options.menu
menubutton .w1.fr0.fr1.help -text "Ajuda" -underline 1 -menu .w1.fr0.fr1.help.menu
# Menu Arquivo
menu .w1.fr0.fr1.file.menu
.w1.fr0.fr1.file.menu add command -label "Abrir" -accelerator "Ctrl+O" -command {bell}
.w1.fr0.fr1.file.menu add separator
.w1.fr0.fr1.file.menu add command -label "Sair" -accelerator "Ctrl+R" -command {exit}
# Menu Editar
menu .w1.fr0.fr1.edit.menu
.w1.fr0.fr1.edit.menu add command -label "Procurar" -accelerator "Ctrl+F" -command {bell}
# Menu Opcoes
menu .w1.fr0.fr1.options.menu
.w1.fr0.fr1.options.menu add cascade -label "Tamanho da fonte" -menu .
w1.fr0.fr1.options.menu.font
.w1.fr0.fr1.options.menu add cascade -label "Nivel de indentacao" -menu .
w1.fr0.fr1.options.menu.indent
# Menu Ajuda
menu .w1.fr0.fr1.help.menu
.w1.fr0.fr1.help.menu add command -label "Conteudo" -accelerator "F1" -command {bell}
.w1.fr0.fr1.help.menu add separator
.w1.fr0.fr1.help.menu add command -label "Sobre" -command {bell}
menu .w1.fr0.fr1.options.menu.font
.w1.fr0.fr1.options.menu.font add radiobutton -label "Pequena" -value 0 -variable Size
-command {bell}
.w1.fr0.fr1.options.menu.font add radiobutton -label "Media" -value 4 -variable Size -command
{bell}
.w1.fr0.fr1.options.menu.font add radiobutton -label "Grande" -value 12 -variable Size -command
{bell}
```

```
menu .w1.fr0.fr1.options.menu.indent
.w1.fr0.fr1.options.menu.indent add radiobutton -label "Pequena" -value 0.6 -variable Indent
-command {bell}
.w1.fr0.fr1.options.menu.indent add radiobutton -label "Media" -value 1.2 -variable Indent
-command {bell}
.w1.fr0.fr1.options.menu.indent add radiobutton -label "Grande" -value 2.4 -variable Indent
-command {bell}
#
# Instala os componentes
#
# Instala os paineis
pack .w1.fr0 -expand 1 -fill both
pack .w1.fr0.fr1 .w1.fr0.fr2 .w1.fr0.fr3 -fill x
pack .w1.fr0.fr4 -expand 1 -fill both
pack .w1.fr0.fr5 -fill x

# Instala os componentes da barra de menu
pack .w1.fr0.fr1.file .w1.fr0.fr1.edit .w1.fr0.fr1.options -side left
pack .w1.fr0.fr1.help -side right
# Redimensiona a janela
wm geometry .w1 640x480+$x+$y
```

O fragmento de código acima produzirá a janela abaixo:

Além dos menus comuns ao UNIX e o Windows, Tk também suporta um tipo de menu que somente existe no UNIX: os *botões de menu*.

De fato, os menus acima foram criados com o auxílio de quatro *botões de menu*, configurados para parecerem com uma barra de menu.

Botões de menu podem ser posicionados livremente sobre a janela, como na figura abaixo:

Botões de menu não devem ser confundidos com os comboboxes(as caixas de combinação do Windows). Estas são providas por mega-widgets como **Tix** e **BWidget**.

Vejamos agora as principais opções disponíveis para os **menubuttons**:

Opção	Descrição
-direction	Direção para abrir o menu: above, below, left, right e flush.
-indicatoron	Se 1, exibirá um pequeno indicador ao lado do rótulo do menu. Caso contrário, o indicador não será exibido.
-menu	Caminho do menu que deve ser aberto quando o botão de menu for invocado.

Os menus suportam as seguintes opções, além daquelas suportadas pelos outros widgets:

Opção	Descrição
-postcommand	Comando Tcl para ser executado antes que o menu seja exibido.
-selectcolor	Cor do indicador para checkbuttons e radiobuttons.
-tearoff	Se 1, inclui uma linha pontilhada sobre as opções do menu, que quando clicada, solta o menu, criando uma janela movel para ele. Caso contrário, não inclui a linha pontilhada.
-tearoffcommand	Comando para ser invocado quando o menu for destacado. O nome do menu e da janela criada serão passados ao comando.
-title	Título da janela criada para o menu quando ele for destacado.
-type	Tipo de menu: menubar, tearoff ou normal.

Os menus suportam os seguintes comandos:

Comando	Descrição
activate	Torna a entrada indicada, a entrada ativa.
add	Adiciona um novo tipo de entrada ao menu. Os tipos de entradas suportados são: cascade, checkbutton, command, radiobutton e separator.
clone	Cria um clone do menu, na forma *nome_do_novo_menutipo_de_menu*.
delete	Remove todas as entradas do menu, na faixa indicada.
entrycget	Retorna o valor de uma opção no menu.
entryconfigure	Configura o valor de uma opção no menu.
index	Retorna um índice numérico para o índice dado.
insert	Insere um novo tipo de entrada no menu, na posição indicada. Os tipos de entradas suportados são: cascade, checkbutton, command, radiobutton e separator.
invoke	Invoca a entrada indicada do menu.
post	Exibe o menu nas coordenadas de tela indicadas.
postcascade	Coloca o submenu associado com a entrada cascade(cascata) na posição dada.
type	Retorna o tipo de entrada na posição dada.
unpost	Remove a janela de modo que ela não seja mais exibida.
yposition	Retorna a coordenada Y da entrada especificada no menu.

As entradas dos menus suportam as seguintes opções:

Opção	Descrição
-accelerator	Uma string para ser exibida ao lado do texto da entrada do menu.
-command	Comando a ser executado quando o menu for invocado.
-columnbreak	Se 1, a entrada aparecerá no topo de uma nova coluna no menu.
-hidemargin	Se 1, as margens padrão não serão exibidas ao redor no ítem do menu.

-indicatoron	Se 1, exibirá um pequeno indicador ao lado do rótulo da entrada do menu. Caso contrário, o indicador não será exibido. Usado com checkbuttons e radiobuttons.
-label	Rótulo da entrada do menu.
-menu	Nome de caminho do menu cascade onde a entrada deve ser exibida..
-offvalue	Valor que deve ser atribuído à variável associada ao checkbutton quando ele não estiver selecionado.
-onvalue	Valor que deve ser atribuído à variável associada ao checkbutton quando ele estiver selecionado.
-selectcolor	Cor do indicador em um checkbutton ou radiobutton.
-selectimage	Imagem para ser exibida no indicador de um checkbutton ou radiobutton.
-value	Valor que deve ser atribuído à variável associada ao radiobutton quando ele estiver selecionado.
-variable	Nome de uma variável global que conterá o valor de um checkbutton ou radiobutton.

Os menus suportam os seguintes índices: *número*, **active**, **last**, **none**, *@Y*, *string para procura*.

15.1 Criando menus suspensos

Para criar um menu suspenso, daqueles que surgem quando se clica com o botão direito do mouse sobre uma janela, usamos o comando **tk_popup**:

tk_popup *menu x y* [*entrada*]

Onde *menu* é o nome(caminho) do menu que se deseja exibir, *x* e *y* são as coordenadas onde o menu deve ser exibido, e *entrada* é a entrada do menu que deve ser exibida nas coordenadas especificadas.

Veja o exemplo seguinte, baseado no fragmento de código apresentado no início da aula:

tk_popup .w1.fr0.fr1.help.menu 300 300

O menu Ajuda será exibido nas coordenadas 300,300 da tela.

Os eventos(**binds**) serão estudados na próxima aula.

16 Associando eventos

Eventos são ações realizadas pelo usuário e interceptadas pelo programa. Em Tk eventos são chamados **binds**(aderências). Os binds são usados para verificar as teclas pressionadas pelo usuário e os movimentos do ponteiro do mouse.

Também podemos simular eventos, de modo que o programa se comporte como se o usuário tivesse realizado uma determinada ação.

O uso mais frequente de binds em programas comerciais é a verificação dos dados digitados pelo usuário, ou para criar teclas de atalho para botões e menus.

A seguir estudaremos os comandos usados para conectar eventos aos widgets Tk e para simulá-los, quando for necessário.

Comando	Descrição
bind *tag*	Retorna uma lista de todas as *sequências*, para as quais existem *binds* correspondentes à *tag* dada.
bind *tag sequência*	Retorna o *script* correspondente à *sequência* e à *tag* dadas.
bind *tag sequência script*	Adere o *script* à *sequência* para a *tag* dada. Se *script* for uma string vazia, a *bind* é apagada. Se o primeiro caractere do script for +, então o *script* será anexado ao script atualmente associado à *sequência* e à *tag* dadas.

bindtags *janela* [*lista_de_tags*]	Ajusta a ordem de precedência das tags para a *janela* de acordo com a lista dada. Se a lista estiver vazia, a ordem será ajustada para a ordem padrão.
event add <<*virtual*>> *sequência sequência...*	Configura para que o evento virtual <<*virtual*>> seja disparado, se qualquer das sequências dadas ocorrer.
event delete <<*virtual*>> [*sequência...*]	Apaga a sequência dada, ou todas caso nenhuma sequência seja especificada, da lista de sequências que disparam o evento virtual dado.
event generate *janela evento* [-when *quando*] [*opção valor...*]	Provoca um evento, na janela dada, como se ele tivesse sido gerado pelo sistema. Várias opções estão disponíveis e serão listadas a seguir. A opção -when indica quando o evento deve ser disparado: now (imediatamente), tail(coloca no final da fila de eventos), head(coloca no início da fila de eventos) ou mark(como head, mas coloca entre os eventos gerados anteriormente).
event info <<*virtual*>>	Retorna a lista de sequências que disparam o evento virtual. Se o evento não for especificado, retorna a lista de todos os eventos virtuais definidos.

O argumento sequencia é uma lista de um ou mais tipos de evento. Um evento pode ser desde um caractere ASCII a uma string na forma <*modificador-modificador-tipo-detalhe*>, ou <<*nome*>>, para um evento virtual.

Veja a lista de *modificadores* possíveis: **Any**, **Control**, **Shift**, **Lock**, **Double**, **Triple**, **Button1**, **Button2**, **Button3**, **Button4**, **Button5**, **B1**, **B2**, **B3**, **B4**, **B5**, **Meta**, **M**, **Mod1**, **Mod2**, **Mod3**, **Mod4**, **Mod5**, **M1**, **M2**, **M3**, **M4**, **M5**, **Alt**.

Os *tipos* de eventos possíveis são: **Activate**, **ButtonPress**, **Button**, **ButtonRelease**, **Circulate**, **Colormap**, **Configure**, **Deactivate**, **Destroy**, **Enter**, **Expose**, **FocusIn**, **FocusOut**, **Gravity**, **KeyPress**, **Key**, **KeyRelease**, **Motion**, **Leave**, **Map**, **Property**, **Reparent**, **Unmap**, **Visibility**.

Os *detalhes* suportados são: para botões, um *número* de 1 a 5, para teclas, um *símbolo* definido em **/usr/include/X11/keysymdef**.

As *tags* podem ser: uma *janela interna*, e se aplicará somente a esta janela, um *toplevel*, e se aplicará a todas as suas janelas internas, uma *classe de janelas*, e se aplicará a todos os *widgets* desta classe, ou **all** e se aplicará a todas as janelas.

Dentro do script de um evento, você pode fazer referência à tecla pressionada, à janela onde o evento ocorreu, às coordenadas do mouse no momento em que o evento ocorreu etc. A seguir são apresentadas Os códigos, que atuam como se fossem variáveis, definidas por Tk para referenciar cada uma dessas informações:

Variável	Descrição
%%	Um sinal %.
%#	O campo *serial* do último evento.
%a	O campo *acima*.
%b	O número do botão.
%c	O campo *contador*.
%d	O campo *detalhe*.
%f	O campo *focus*.
%h	O campo *altura*.
%k	O campo *código da tecla*.
%m	O campo *modo*.
%o	O campo *override_redirect*.
%p	O campo *place*.
%s	O campo *estado*.

%t	O campo *time*.
%w	O campo *largura*.
%x	O campo x.
%y	O campo y.
%A	O caractere ASCII.
%B	O campo *largura da borda*.
%E	O campo *send_event*.
%K	O símbolo da tecla pressionada como *texto*.
%N	O símbolo da tecla pressionada como um *número decimal*.
%R	A janela principal(root).
%S	Identificador *sub_window*.
%T	O campo *tipo*.
%W	O nome(*caminho*) da janela onde o evento ocorreu.
%X	O campo x_root.
%Y	O campo y_root.

Os eventos suportam as seguintes opções:

Opção	Código	Evento válido
-above *janela*	%a	Configure
-borderwidth *largura*	%B	Configure
-button *número*	%b	ButtonPress, ButtonRelease
-count *número*	%c	Expose
-detail *detalhe*	%d	Enter, Leave, Focus
-focus *booleano*	%f	Enter, Leave
-height *altura*	%h	Configure
-keycode *número*	%k	KeyPress, KeyRelease
-keysym *nome*	%K	KeyPress, KeyRelease
-mode *modo*	%m	Enter, Leave, Focus
-override *booleano*	%o	Map, Reparent, Configure
-place *onde*	%p	Circulate
-root *janela*	%R	KeyPress, KeyRelease, ButtonPress, ButtonRelease, Enter, Leave, Motion
-rootx *coordenada*	%X	KeyPress, KeyRelease, ButtonPress, ButtonRelease, Enter, Leave, Motion
-rooty *coordenada*	%Y	KeyPress, KeyRelease, ButtonPress, ButtonRelease, Enter, Leave, Motion
-sendevent *booleano*	%E	Todos os eventos
-serial *número*	%#	Todos os eventos
-state *estado*	%s	Todos os eventos
-subwindow *janela*	%S	KeyPress, KeyRelease, ButtonPress, ButtonRelease, Enter, Leave, Motion
-time *inteiro*	%t	KeyPress, KeyRelease, ButtonPress, ButtonRelease, Enter, Leave, Motion, Property
-x *coordenada*	%x	KeyPress, KeyRelease, ButtonPress, ButtonRelease, Enter, Leave, Motion, Expose, Configure, Gravity, Reparent

-y *coordenada*	%y	KeyPress, KeyRelease, ButtonPress, ButtonRelease, Enter, Leave, Motion, Expose, Configure, Gravity, Reparent

16.1 Associando teclas de atalho

Normalmente associamos teclas de atalho a botões e menus. Para indicar a tecla de atalho em um botão sublinhamos a tecla correspondente através da opção **-underline**:

```
# Cria uma janela
toplevel .t
# Cria um botao com a primeira letra do rotulo sublinhada
button .t.b -text OK -underline 0 -command {destroy .t}
# Dispoe o botao na janela
pack .t.b
# Configura o evento pressionar as teclas Alt( esquerda ) + O( minusculo ) para que invoque o comando default do botao
bind .t <Key-Alt_L><Key-o> {.t.b invoke}
```

O exemplo acima configura a tecla de atalho ALT+O para o botao OK.

Para indicar a tecla de atalho para um menu devemos utilizar a opção **-accelerator** do menu:

```
# Cria a janela
toplevel .w1

wm protocol .w1 WM_DELETE_WINDOW {exit}

wm title .w1 "HTML Browser"
wm resizable .w1 1 1
# Centraliza a janela na tela
set x [expr {([winfo screenwidth .w1] - 640)/2}]
set y [expr {([winfo screenheight .w1] - 480)/2}]
#
# Cria os paineis
#
# Painel principal
frame .w1.fr0 -borderwidth 2 -relief raised
# Painel do menu
frame .w1.fr0.fr1 -borderwidth 2 -relief raised
# Painel da barra de ferramentas
frame .w1.fr0.fr2 -borderwidth 2 -relief raised
# Painel da barra de navegacao
frame .w1.fr0.fr3 -borderwidth 2 -relief raised
# Painel da janela do browser
frame .w1.fr0.fr4 -borderwidth 2
# Painel da barra de status
frame .w1.fr0.fr5 -borderwidth 2 -relief raised
#
# Cria os componentes dos paineis
#
# Cria os menus
menubutton .w1.fr0.fr1.file -text "Arquivo" -underline 0 -menu .w1.fr0.fr1.file.menu
menubutton .w1.fr0.fr1.edit -text "Editar" -underline 0 -menu .w1.fr0.fr1.edit.menu
```

```
menubutton .w1.fr0.fr1.options -text "Opcoes" -underline 0 -menu .w1.fr0.fr1.options.menu
menubutton .w1.fr0.fr1.help -text "Ajuda" -underline 1 -menu .w1.fr0.fr1.help.menu
# Menu Arquivo
menu .w1.fr0.fr1.file.menu
.w1.fr0.fr1.file.menu add command -label "Abrir" -accelerator "Ctrl+O" -command {bell}
.w1.fr0.fr1.file.menu add separator
.w1.fr0.fr1.file.menu add command -label "Sair" -accelerator "Ctrl+R" -command {exit}
# Menu Editar
menu .w1.fr0.fr1.edit.menu
.w1.fr0.fr1.edit.menu add command -label "Procurar" -accelerator "Ctrl+F" -command {bell}
# Menu Opcoes
menu .w1.fr0.fr1.options.menu
.w1.fr0.fr1.options.menu add cascade -label "Tamanho da fonte" -menu .
w1.fr0.fr1.options.menu.font
.w1.fr0.fr1.options.menu add cascade -label "Nivel de indentacao" -menu .
w1.fr0.fr1.options.menu.indent
# Menu Ajuda
menu .w1.fr0.fr1.help.menu
.w1.fr0.fr1.help.menu add command -label "Conteudo" -accelerator "F1" -command {bell}
.w1.fr0.fr1.help.menu add separator
.w1.fr0.fr1.help.menu add command -label "Sobre" -command {bell}
menu .w1.fr0.fr1.options.menu.font
.w1.fr0.fr1.options.menu.font add radiobutton -label "Pequena" -value 0 -variable Size
-command {bell}
.w1.fr0.fr1.options.menu.font add radiobutton -label "Media" -value 4 -variable Size -command
{bell}
.w1.fr0.fr1.options.menu.font add radiobutton -label "Grande" -value 12 -variable Size -command
{bell}
menu .w1.fr0.fr1.options.menu.indent
.w1.fr0.fr1.options.menu.indent add radiobutton -label "Pequena" -value 0.6 -variable Indent
-command {bell}
.w1.fr0.fr1.options.menu.indent add radiobutton -label "Media" -value 1.2 -variable Indent
-command {bell}
.w1.fr0.fr1.options.menu.indent add radiobutton -label "Grande" -value 2.4 -variable Indent
-command {bell}
#
# Instala os componentes
#
# Instala os paineis
pack .w1.fr0 -expand 1 -fill both
pack .w1.fr0.fr1 .w1.fr0.fr2 .w1.fr0.fr3 -fill x
pack .w1.fr0.fr4 -expand 1 -fill both
pack .w1.fr0.fr5 -fill x

# Instala os componentes da barra de menu
pack .w1.fr0.fr1.file .w1.fr0.fr1.edit .w1.fr0.fr1.options -side left
pack .w1.fr0.fr1.help -side right
# Redimensiona a janela
wm geometry .w1 640x480+$x+$y
```

```
# Configura a tecla de atalho para o menu Arquivo->Sair como sendo Ctrl( esquerdo ) + R
( minusculo )
bind .w1 <Key-Control_L><r> {exit}
```

O fragmento de código acima, adaptado do código fonte do meu browser TkShip, cria uma janela com uma barra de menu e configura a tecla de atalho do menu Arquivo->Sair para CTRL+R.
Copie e cole o código acima na linha de comandos do wish e veja como funciona.

Validando a entrada do usuário

Validar a entrada digitada pelo usuário é uma dos usos mais importantes para os binds. Tk 8.3 ou superior inclui o suporte a validação de dados digitados no widget entry, entretanto, versões mais antigas do Tk, e que acompanham a maioria das distribuições do Linux, só podem validar a entrada digitada através de binds. De fato, tudo o que as opções de validação do Tk 8.3 fazem é configurar binds para o widget entry, como faremos nesta seção.
Mas antes de prosseguir-mos, muitas pessoas me perguntam como trocar o comportamento default do X-Window/Windows de TAB para ENTER, para mudar o foco entre os widgets de uma janela. O fragmento de código a seguir fará exatamente isso:

```
# Configura a tecla ENTER para que realize a mudanca de foco entre os widgets de uma janela
bind Toplevel all <Key-Return> {focus [tk_focusNext %W]}
```

Agora vejamos como configurar os binds de um widget entry para que seja realizada a validação dos dados:

```
# Cria a janela
toplevel .t
entry .t.e -justify right
pack .t.e -side left -expand 1 -fill x
# Configura o tipo de dado que pode ser digitado
# Esta forma nao e elegante, voce deve evita-la
# bind .t.e <Key> {
#     if {((%N < 44) || (%N > 57) || ([string length [%W get]] >= 10)) && ((%N >= 32) && (%N <=
128))} {
#         bell
#         break
#     }
# }
# Esta e a forma elegante de configurar a validacao dos dados
#    A parte ([regexp -nocase {[0-9]} %A] == 0) determina que somente valores numericos
podem ser aceitos
#    A parte ([string length [%W get]] >= 10) determina que somente 10 digitos devem ser aceitos
#    A parte ((%N >= 32) && (%N <= 128)) e necessaria para permitir os caracteres ASCII de
controle como
#        DELETE e BACKSPACE
bind .t.e <Key> {
   if {(([regexp -nocase {[0-9]} %A] == 0) || ([string length [%W get]] >= 10)) && ((%N >= 32) &&
(%N <= 128))} {
      bell
      break
   }
}
```

```
# Faz com que todo o texto dentro do widget seja selecionado, quando ele receber o foco
bind .t.e <FocusIn> {
   .t.e selection range 0 end
}
```

Agora se desejar-mos que os dados sejam formatados após a digitação, deveremos criar uma função que formate o texto para nós. Como a função abaixo, que formata um numero na forma #.##, útil para valores monetários:

```
# float2cur valor
# Formata um valor numerico para a forma #.##
proc float2cur {{valor 0.00}} {
   set vetor {}
   set vetor [split $valor .]
   set p1 ""
   set p2 ""
   set val ""
   if {[llength $vetor] == 2} {
      if {[lindex $vetor 0] != ""} {
         set p1 [lindex $vetor 0]
      } else {
         set p1 "0"
      }
      if {[lindex $vetor 1] != ""} {
         set x1 [format {%1$-2s} [lindex $vetor 1]]
         regsub " " $x1 0 x1
         regsub " " $x1 0 x1
         set p2 $x1
      } else {
         set p2 "00"
      }
   } else {
      if {[lindex $vetor 0] != ""} {
         set p1 [lindex $vetor 0]
      } else {
         set p1 "0"
      }
      set p2 "00"
   }
   if {[string length $p2] > 2} {
      set x2 [string range $p2 0 1]
      if {[string index $p2 2] > 5} {
         if {$x2 < 99} {
            if {[string index $x2 0] == 0} {
               set x2 [string index $x2 1]
               incr x2
               set x2 [format {%1$02u} $x2]
            } else {
               incr x2
```

```
            }
         } else {
            set x2 "00"
            incr p1
         }
      }
      set p2 $x2
   }
   append val $p1 . $p2
   return $val
}
# Cria a janela
toplevel .t
entry .t.e -justify right
pack .t.e -side left -expand 1 -fill x
# Configura o tipo de dado que pode ser digitado
bind .t.e <Key> {
   if {(([regexp -nocase {[0-9]} %A] == 0) || ([string length [%W get]] >= 10)) && ((%N >= 32) &&
(%N <= 128))} {
      bell
      break
   }
}
# Faz com que todo o texto dentro do widget seja selecionado, quando ele receber o foco
bind .t.e <FocusIn> {
   .t.e selection range 0 end
}
# Configura para que a informacao digitada seja formatada quando o widget perder o foco
bind .t.e <FocusOut> {
   set temp [float2cur [%W get]]
   %W delete 0 end
   %W insert 0 $temp
}
```

Experimente o fragmento de código acima copiando-o e colando-o na linha de comandos do wish. Os dados serão formatados quando o widget entry perder o foco.

17 Trabalhando com imagens

Tk, atualmente, suporta três formatos para arquivos de imagens: GIF, PGM e PPM. Outros formatos como JPEG, PNG, TIFF e WMF podem ser acessados através de bibliotecas externas. Entretanto, a Ajuba Solutions, atual mantenedor do Tcl/Tk pretende incluir estas extensões na próxima versão(8.4) que deverá estar disponível, em sua versão final, em dezembro. Ícones no formato do Windows(*.ICO) são suportados através do comando winico, caso você compile o seu programa usando o **freeWrap** para Windows, ou utilize uma biblioteca externa. Ícones do X-Window(pixmaps) são suportados através de uma biblioteca em Tcl puro, cujo código fonte será apresentado no final desta aula. Você pode usar arquivos XPM(pixmaps X11) no Windows.

A seguir são apresentados os comando para leitura, manipulação e gravação de imagens em Tcl/Tk:

Comando	Descrição
image create	Cria uma nova imagem, do tipo e com o nome especificado. Suporta várias opções.
image delete	Apaga a imagem indicada.
image height	Retorna a altura da imagem em pixels.
image names	Retorna os nomes de todas as imagens existentes.
image type	Retorna o tipo da imagem indicada.
image types	Retorna todos os tipos de imagens suportadas.
image width	Retorna a largura da imagem em pixels.

As opções suportadas dependem do tipo da imagem. Os tipos suportados por Tk são: bitmap e photo. *Bitmap* é uma imagem monocromática, que pode ter um fundo transparente. É um bitmap *X-Window* e não um bitmap Windows. Os bitmaps do Windows são suportados através de bibliotecas externas. *Photo* é uma imagem colorida, que pode ser lida e salva em vários formatos. Pode ser transparente.

As opções para as imagens do tipo bitmap são as seguintes:

Opção	Descrição
-background	Cor de fundo.
-data	Conteúdo do bitmap no formato X11.
-file	Nome do arquivo cujo conteúdo define o bitmap X11.
-foreground	Cor de primeiro plano.
-maskdata	Conteúdo da máscara do bitmap no formato X11.
-maskfile	Nome do arquivo cujo conteúdo define a máscara do bitmap X11.

As opções para as imagens do tipo photo são as seguintes:

Opção	Descrição
-data	Conteúdo da imagem no formato especificado.
-format	Formato da imagem. Os formatos nativos são GIF, PGM e PPM.
-file	Nome do arquivo cujo conteúdo define a imagem no formato indicado.
-height	Altura da imagem em pixels.
-palette	Configura a resolução do cubo de cor a ser reservado para a imagem.
-width	Largura da imagem em pixels.

As imagens do tipo **photo** suportam ainda os seguintes comandos:

Comando	Descrição
*nome_da_imagem*blank	Limpa a imagem tornando-a totalmente transparente.
*nome_da_imagem*copy *fonte* [*opção valor*...]	Copia uma região de uma imagem para outra. Suporta várias opções descritas a seguir.
*nome_da_imagem*get *x y*	Retorna a cor do ponto *x y* como uma lista {R G B}.
*nome_da_imagem*put *dado* [-to *x1 y1 x2 y2*]	Configura os pixels na região dada para os valores contidos no vetor *dado*.
*nome_da_imagem*read *arquivo* [*opção valor*...]	Lê a imagem do arquivo. Suporta várias opções descritas a seguir.
*nome_da_imagem*redither	Reditheriza a imagem.
*nome_da_imagem*write *arquivo* [*opção valor*...]	Salva a imagem no arquivo.

O comando copy suporta as seguintes opções:

Opção	Descrição

-from *x1 y1 x2 y2*	Região da imagem *fonte* a ser copiada.
-to *x1 y1 x2 y2*	Região da imagem *destino* a ser copiada.
-shrink	Reduz a imagem para que a região copiada fique ajustada com o canto inferior direito da imagem.
-zoom *x y*	Magnifica a imagem.
-subsample *x y*	Reduz a imagem.

O comando **read** suporta as seguintes opções:

Opção	Descrição
-format *formato*	Formato da imagem.
-from *x1 y1 x2 y2*	Região da imagem *fonte* a ser lida.
-to *x1 y1 x2 y2*	Região da imagem *destino* a ser copiada.
-shrink	Reduz a imagem para que a região copiada fique ajustada com o canto inferior direito da imagem.

O comando **write** suporta as seguintes opções:

Opção	Descrição
-format *formato*	Formato da imagem.
-from *x1 y1 x2 y2*	Região da imagem *fonte* a ser salva.

Veja um exemplo:

```
toplevel .t
label .t.l
pack .t.l
image create photo logo -file [file join $tk_library images logo100.gif]
.t.l configure -image logo
```

Será exibida uma janela com o logo-tipo Tcl/Tk. A variável global tk_library contém o caminho da biblioteca Tk em seu sistema operacional. No Windows será *C:/Arquivos de Programa/Tcl/Tk8.2* e no Linux */usr/lib/tk8.2*, caso você esteja usando a versão 8.2 do Tcl/Tk. No diretorio tk8.x existe um sub-diretório *images* que contém diversos arquivos para promoção do Tcl. Você pode usá-los a vontade.

17.1 Mapas de pixels(arquivos XPM)

Os mapas de pixels, pixmaps X11, são largamente usados como arquivos de ícones devido à facilidade para criação de ícones no próprio código do programa. Veja o conteúdo típico de uma arquivo XPM:

```
/* XPM */
static char * calc_xpm[] = {
"32 32 7 1",
"       c None",
".      c #666666",
"+      c #E5E5E5",
"@      c #7F7F7F",
```

```
"#      c #FFFFFF",
"$      c #000000",
"%      c #CCCCCC",
"                          ",
"     .....................     ",
"     .+++++++++++++++++++@.     ",
"     .+###################@.     ",
"     .+#################$#$#@.     ",
"     .+###################@.     ",
"     .+@@@@@@@@@@@@@@@@@@@@.     ",
"     .%%%%%%%%%%%%%%%%%%%%.     ",
"     .++++$++++$++++$++++$.     ",
"     .+%%%$+%%%$+%%%$+%%%$.     ",
"     .+%$%$+%$%$+%$%$+%$%$.     ",
"     .+%%%$+%%%$+%%%$+%%%$.     ",
"     .+$$$$+$$$$+$$$$+$$$$.     ",
"     .++++$++++$++++$+%%%$.     ",
"     .+%%%$+%%%$+%%%$+%%%$.     ",
"     .+%$%$+%$%$+%$%$+%$%$.     ",
"     .+%%%$+%%%$+%%%$+%%%$.     ",
"     .+$$$$+$$$$+$$$$+$$$$.     ",
"     .++++$++++$++++$++++$.     ",
"     .+%%%$+%%%$+%%%$+%%%$.     ",
"     .+%$%$+%$%$+%$%$+%$%$.     ",
"     .+%%%$+%%%$+%%%$+%%%$.     ",
"     .+$$$$+$$$$+$$$$+$$$$.     ",
"     .++++$++++$++++$++++$.     ",
"     .+%%%$+%%%$+%%%$+%%%$.     ",
"     .+%$%$+%$%$+%$%$+%$%$.     ",
"     .+%%%$+%%%$+%%%$+%%%$.     ",
"     .+$$$$+$$$$+$$$$+$$$$.     ",
"     .%%%%%%%%%%%%%%%%%%%%.     ",
"     .%%%%%%%%%%%%%%%%%%%%.     ",
"     .....................     ",
"                          "};
```

O pixmap acima define o ícone de uma calculadora, da barra de ferramenta do meu programa de controle financeiro. Copie e cole para um arquivo de texto vazio e salve-o como calc.xpm. Experimente abrir com o The Gimp no Linux, ou outro programa que suporte pixmaps.

O código fonte a seguir define o comando xpm-to-image que retorna uma imagem(para ser atribuída à propriedade **-image** de um widget) a partir de um arquivo **XPM**:

```
# ----------------------------------------------------------------------------
#  xpm2image.tcl
# ----------------------------------------------------------------------------
#
#  Copyright 1996 by Roger E. Critchlow Jr., San Francisco, California
#  All rights reserved, fair use permitted, caveat emptor.
#  rec@elf.org
#
```

```
# ----------------------------------------------------------------------------

proc xpm-to-image { file } {
    set f [open $file]
    set string [read $f]
    close $f

    #
    # Analiza a strings no arquivo XPM
    #
    set xpm {}
    foreach line [split $string "\n"] {
        if {[regexp {^"([^\"]*)"} $line all meat]} {
            if {[string first XPMEXT $meat] == 0} {
                break
            }
            lappend xpm $meat
        }
    }

    #
    # Extrai os dados no arquivo
    #
    set sizes  [lindex $xpm 0]
    set nsizes [llength $sizes]
    if { $nsizes == 4 || $nsizes == 6 || $nsizes == 7 } {
        set data(width)   [lindex $sizes 0]
        set data(height)  [lindex $sizes 1]
        set data(ncolors) [lindex $sizes 2]
        set data(chars_per_pixel) [lindex $sizes 3]
        set data(x_hotspot) 0
        set data(y_hotspot) 0
        if {[llength $sizes] >= 6} {
            set data(x_hotspot) [lindex $sizes 4]
            set data(y_hotspot) [lindex $sizes 5]
        }
    } else {
            error "A largura da linha {$sizes} em $file nao computa"
    }

    #
    # Extrai as definicoes de cores
    #
    foreach line [lrange $xpm 1 $data(ncolors)] {
        set colors [split $line \t]
        set cname  [lindex $colors 0]
        lappend data(cnames) $cname
        if { [string length $cname] != $data(chars_per_pixel) } {
```

```
        error "A definição de cor {$line} no arquivo $file esta com um nome incorreto"
    }
    foreach record [lrange $colors 1 end] {
        set key [lindex $record 0]
        set color [string tolower [join [lrange $record 1 end] { }]]
        set data(color-$key-$cname) $color
        if { ![string compare $color "none"] } {
            set data(transparent) $cname
        }
    }
    foreach key {c g g4 m} {
        if {[info exists data(color-$key-$cname)]} {
            set color $data(color-$key-$cname)
            set data(color-$cname) $color
            set data(cname-$color) $cname
            lappend data(colors) $color
            break
        }
    }
    if { ![info exists data(color-$cname)] } {
        error "A definicao de cor {$line} em $file falhou ao definir a cor"
    }
}

#
# Extrai os dados da imagem
#
set image [image create photo -width $data(width) -height $data(height)]
set y 0
foreach line [lrange $xpm [expr 1+$data(ncolors)] [expr 1+$data(ncolors)+$data(height)]] {
    set x 0
    set pixels {}
    while { [string length $line] > 0 } {
        set pixel [string range $line 0 [expr {$data(chars_per_pixel)-1}]]
        set c $data(color-$pixel)
        if { ![string compare $c none] } {
            if { [string length $pixels] } {
                $image put [list $pixels] -to [expr {$x-[llength $pixels]}] $y
                set pixels {}
            }
        } else {
            lappend pixels $c
        }
        set line [string range $line $data(chars_per_pixel) end]
        incr x
    }
    if { [llength $pixels] } {
        $image put [list $pixels] -to [expr {$x-[llength $pixels]}] $y
```

```
        }
        incr y
    }

    #
    # Retorna a imagem
    #
    return $image
}
```

Veja um exemplo:

```
toplevel .t
label .t.l
pack .t.l
.t.l configure -image [xpm-to-image c:/temp/calc.xpm]
```

Para testar o fragmento de código acima, copie e cole a definição do ícone calc.xpm para o arquivo c:\temp\calc.xpm e em seguida copie e cole o fragmento de código acima para a linha de comando do wish.

18 Trabalhando com bancos de dados

Tcl suporta praticamente todos os bancos de dados relacionais existentes, através de interface própria ou ODBC. Todos os bancos de dados **ODBC** são suportados através da interface ODBC.

Os bancos de dados suportados através de interfaces próprias são: **DB2**, **INFORMIX**, **miniSQL** (mSQL), **MySQL**, **PostgreSQL**, **ORACLE** e **Sybase**.

Também existe um banco de dados escrito em 100% puro Tcl: **TclVSdb**. O TclVSdb, é um banco de dados simples e plataforma-independente, podendo ser usado no UNIX e no Windows. Contudo, a performance é bastante inferior à dos RDBMSs mensionados anteriormente.

Nas proximas aulas estudaremos as interfaces Tcl-PostgreSQL e Tcl-ODBC, pois são as mais utilizadas atualmente, e as que recebem atualizações mais frequentes.

Até lá, recomendo que faça o download de todas as interfaces Tcl-RDBMS que lhe interessarem, através do meu site: http://www.souzamonteiro.com/download_freeware.shtml.

Mas antes de proseguir-mos, devemos compreender alguns conceitos relativos a bancos de dados relacionais.

18.1 Tabelas, linhas e colunas

Um banco de dados é composto de **tabelas**, que por sua vez são compostas de **linhas** e estas de **colunas**. Fazendo uma analogia aos bancos de dados do Clipper, que alguns já estão acostumados, tabelas correspondem aos arquivos DBF, linhas aos registros nos arquivos DBF e colunas aos campos nos registros nos arquivos DBF. Veja um exemplo de uma tabela típica:

BAIRROS

Código	Bairro
1	Arenoso
2	Arvoredo
3	Av. Bete
4	Amaralina
5	Aeroporto

6	Barros Reis
7	Barbalho
8	Bonocô
9	Brotas
10	Boca do Rio

18.2 Bancos de dados relacionais

Os bancos de dados relacionais RDBMS(Relational Database Management Systems) permitem estabelecer relacionamentos entre tabelas, de modo que mover o ponteiro de registro sobre uma tabela, provoque um movimento de ponteiro de registro correspondente em outra tabela. Veja um exemplo:

Uma pizzaria possui um catálogo eletrônico de preços por bairro e por produto, de tal modo que as informações sobre os bairros são armazenadas em uma tabela BAIRROS, as informações sobre os produtos em uma tabela PRODUTOS e os preços em uma tabela PREÇOS:

BAIRROS

Código	Bairro
1	Arenoso
2	Arvoredo
3	Av. Bete
4	Amaralina
5	Aeroporto
6	Barros Reis
7	Barbalho
8	Bonocô
9	Brotas
10	Boca do Rio

PRODUTOS

Código	Produto
1	Pizza Média
2	Pizza Grande
3	Pizza Família
4	Lazanha
5	Filé

PREÇOS

Bairro	Produto	Preço
1	1	12.00
1	2	13.90
1	3	17.00
1	4	13.90
1	5	13.90
2	1	12.00
2	2	13.90
2	3	17.00
2	4	13.90
2	5	13.90

Agora, se desejar-mos obter uma lista de preços dos bairros, poderíamos utilizar, em uma linguagem de consulta hipotética, algo assim:

RETORNE BAIRROS.Bairro, PRODUTOS.Produto, PREÇOS.Preço
ONDE PREÇOS.Bairro = BAIRROS.Código **E** PREÇOS.Produto = PRODUTOS.Código
ORDENAR POR BAIRROS.Bairro
Esta linha de comando hipotética retornaria algo assim:

RESULTADO DA CONSULTA

Bairro	Produto	Preço
Arenoso	Pizza Média	12.00
Arenoso	Pizza Grande	13.90
Arenoso	Pizza Família	17.00
Arenoso	Lazanha	13.90
Arenoso	Filé	13.90
Arvoredo	Pizza Média	12.00
Arvoredo	Pizza Grande	13.90
Arvoredo	Pizza Família	17.00
Arvoredo	Lazanha	13.90
Arvoredo	Filé	13.90

18.3 SQL

A **S**tructured **Q**uery **L**anguage(Linguagem de Consulta Estruturada) foi criada com a finalidade de permitir fácil acesso a diferentes bancos de dados, tais como Oracle, XBase, PostgreSQL, mSQL, MySQL, ODBC e outros, sem que fosse preciso aprender uma linguagem diferente para cada sistema de banco de dados. Existem diversos dialetos SQL, contudo, a maioria dos RDBMSs suportam o SQL92 ou um sub-conjunto dele.

De um modo geral você precisará saber programar em SQL para acessar qualquer banco de dados relacional, seja em Tcl, Perl, C/C++ ou mesmo Java. Nesta aula não abordaremos a sintaxe da linguagem, mas você deverá estudá-la, antes que possa proseguir neste curso. Para tanto, eu escreví uma apostila Programando em SQL que está disponível para download em meu site: http://www.souzamonteiro.com/download_freeware.shtml.

SQL é uma linguagem muito fácil e se parece muito com a nossa linguagem de consulta hipotética, apresentada na seção anterior desta aula. Veja como ficaria a nossa linha de comando para acessar a lista de preços de produtos por bairro, em PostgreSQL:

```
SELECT t1.codigo, t1.bairro, t2.codigo, t2.produto, t3.bairro, t3.produto, t3.preco
FROM bairros AS t1, produtos AS t2, precos AS t3
WHERE t3.bairro = t1.codigo AND t3.produto = t2.codigo
ORDER BY t1.bairro;
```

Esta síntaxe pode variar ligeiramente de um RDBMS para outro, mas de um modo geral será bem parecido com o que foi apresentado acima.

Você precisará aprender os comandos **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **CREATE DATABASE**, **CREATE TABLE** e, opcionalmente, ALTER TABLE, CREATE INDEX, DROP DATABASE, DROP TABLE, DROP INDEX, DECLARE, BEGIN WORK, CLOSE, COMMIT, ROLLBACK, FETCH e MOVE. Embora estes comandos opcionais não estejam disponíveis em todos os RDBMSs, eles permitem realizar muitas operações indispensáveis em aplicações avançadas.

18.4 Transações

Transações são modificações que devem ser executadas em grupo, em um banco de dados relacional. Dentro de uma transação, todas as operações executadas devem ocorrer sem erro, ou todas serão canceladas.

Um exemplo típico de transação é quando em um programa de vendas, precisamos realizar a operação de entrada de valor no caixa e a correspondente saída do produto do estoque. Caso ocorra um erro ao tentar salvar as alterações na tabela do caixa, toda a operação deve ser cancelada, ou seria dada baixa no estoque e os dados ficariam inconsistentes.

Na maioria dos casos você deve preferir bancos de dados que suportem transações. No monento, o melhor RDBMS, de código fonte aberto e verdadeiramente gratúito é o PostgreSQL. O MySQL é mais rápido mas não suporta a operação ROLLBACK, que cancela uma operação dentro de uma transação e não pode ser utilizado em um programa comercial sem o pagamento de uma licença. Neste ponto é preciso esclarecer que GPL não quer dizer gratúito. Um produto GPL só pode ser incorporado em outro produto GPL. Para que você possa utilizar um banco de dados em um programa comercial, sem ter que pagar uma licença é preciso que as bibliotecas do RDBMS estejam sob licença **LGPL** ou **BSD**.

Escolha com cuidado o banco de dados que irá utilizar em suas aplicações. E se possível resista à tentação de utilizar ODBC pois trata-se de um produto TOTALMENTE comercial. A interface é livre, mas o Gerenciador ODBC e os drivers não são.

Para uma descrição detalhada de todos os comandos disponíveis em uma interface Tcl-RDBMS, consulte a documentação on-line do seu RDBMS. Para uma aula introdutória à linguagem SQL consulte a apostila **Programando em SQL** disponível para download em http://www.souzamonteiro.com/download_freeware.shtml.

19 Acessando um banco de dados PostgreSQL

A interface PostgreSQL-Tcl oferece diversos comandos que permitem o acesso aos bancos de dados do PostgreSQL através da linguagem SQL. O PostgreSQL suporta um superset do SQL92, oferecendo comandos e tipos de dados adicionais.

Para acrescentar o suporte aos bancos de dados do PostgreSQL em seus programas, você deve incluir o comando a seguir, logo no início de seu programa:

load libpgtcl.so

Se o seu programa for rodar em ambiente UNIX, e:

load libpgtcl.dll

Se o seu programa for rodar no Windows.

Observe que para rodar um programa em Tcl ou qualquer outra linguagem no Windows, você precisará ter um servidor UNIX ou Windows NT rodando o RDBMS do PostgreSQL e aceitando conecções via TCP/IP, pois o PostgreSQL não pode rodar no Windows95/98.

Uma outra alternativa é utilizar um interpretador Tcl com o suporte ao PostgreSQL embutido. O pacote do PostgreSQL inclui os interpretadores **pgtclsh** e **pgtksh**, que permitem executar programas em Tcl para o console e para o X-Window respectivamente.

Neste ponto, você deve instalar e configurar o seu PostgreSQL para que possa prosseguir neste estudo. A seguir estão todas as instruções necessárias para preparar o seu servidor de banco de dados PostgreSQL.

19.1 Instalando o PostgreSQL

Para instalar o PostgreSQL em um sistema compatível com RPMs, coloque o CD-ROM da sua distribuição do Linux em seu drive de CDs e monte-o seguindo os passos abaixo. Mas lembre-se: você precisa ser o usuário **root**.

mount /mnt/cdrom

Caso a sua distribuição seja o CL 5.0, mude para o diretório /mnt/cdrom/conectiva/RPMS:

cd /mnt/cdrom/conectiva/RPMS

Instale os RPMS:

rpm -ivh postgresql-*.rpm

Isso instalará diversos pacotes, incluindo os drivers ODBC e JDBC, interfaces gráficas, documentação etc.

Desmonte o CD-ROM:

cd ~/

umount /mnt/cdrom

Agora, usando o seu editor preferido modifique o arquivo **/etc/rc.d/init.d/postgresql**. Você deve procurar a linha abaixo:

daemon --check postmaster su -l postgres -c \"/usr/bin/postmaster -S -D/var/lib/pgsql\"

E modificá-la para:

daemon --check postmaster su -l postgres -c \"/usr/bin/postmaster -S**-i** -D/var/lib/pgsql\"

Acrescentando a opção **-i** que instrui o servidor postmaster a aceitar conecções via TCP/IP.

Agora edite o arquivo **/var/lib/pgsql/pg_hba.conf**, como o usuário **postgres**(não faça isso como root), e remova o comentário da linha:

host all 0.0.0.0 0.0.0.0 trust

Isso dará permissão de acesso a qualquer usuário, de qualquer máquina para acessar qualquer banco de dados no servidor PostgreSQL.

Se você desejar, pode implementar qualquer outra forma de restrição de acesso, utilizando os modelos que se encontram comentados no arquivo. Por hora, deixe assim, para que outras máquinas da rede possam acessar seus bancos de dados PostgreSQL.

Inicialize o servidor postmaster:

/etc/rc.d/init.d/postgresql start

Agora faça com que o servidor seja inicializado, sempre que o Linux for iniciado. Para isso execute o programa **/usr/sbin/ntsysv**, faça uma marca no serviço postgresql e confirme.

Pronto! O seu servidor está configurado para servir bancos de dados PostgreSQL.

19.2 Criando um usuário

Para acessar um banco de dados PostgreSQL você deve ser um usuário cadastrado na base de dados do PostgreSQL. Para tanto, como o usuário postgres, digite:

$ createuser *nome_de_usuário*

Troque *nome_de_usuário* pelo nome do usuário que você quer criar. Se quiser simplificar as coisas, pode usar o usuário postgres por enquanto. Confirme todas as perguntas que o programa createuser fizer, desta vez.

Você também pode criar um usuário usando a interface **SQL**, através do comando:

CREATE USER *nome_do_usuário***;**

19.3 Criando um banco de dados

Para criar um banco de dados você deve ser um usuário cadastrado na base de dados do PostgreSQL. Para tanto digite:

$ createdb *nome_do_banco_de_dados*

Troque *nome_do_banco_de_dados* pelo nome do banco de dados que você deseja crair.

Você também pode criar um banco de dados usando a interface **SQL**, através do comando:

CREATE DATABASE *nome_do_banco_de_dados***;**

19.4 A interface PostgreSQL-Tcl

Os principais comandos oferecidos pela interface PostgreSQL-Tcl são os seguintes:

Comando	Descrição
pg_connect *nome_do_banco_de_dados* [-host *servidor*] [-port *porta*] [-options*opções*]	Abre uma conecção com o banco de dados. Retorna um *handle* para a conecção.
pg_disconnect *conecção*	Fecha a conecção com o banco de dados. *Conecção* é o handle para a conecção.
pg_exec *conecçãoconsulta_sql*	Executa uma consulta SQL. Retorna um *handle* para o resultado da cunsulta. *Conecção* é o handle para a conecção.
pg_select *conecçãoconsulta_sqlvariável script*	Executa um laço através do resultado de um SELECT, atribuindo ao vetor associativo variável, o conteúdo de cada linha retornada na consulta e executando o script a cada interação. *Conecção* é o handle para a conecção.
pg_result *resultadoopções*	Retorna diversos resultados relacionados com a consulta SQL. Veja as opções na tabela abaixo. *Resultado* é o handle para o resultado da consulta.

O comando pg_result suporta as seguintes opções:

Opção	Descrição
-status	Retorna o *status* da consulta. Verifique sempre.
-error	Retorna o *erro* ocorrido, caso tenha ocorrido um erro. Verifique sempre.
-conn	Retorna a *conecção* que gerou a consulta.
-oid	Retorna o *OID* caso tenha sido efetuado um INSERT, caso contrário, retorna uma string vazia.
-numTuples	Retorna o número de linhas retornadas.
-numAttrs	Retorna o número de colunas retornadas.
-assign *variável*	Atribui o resultado da consulta ao vetor associativo bidimensional *variável*. Onde cada elemento do vetor possui um índice da seguinte forma: *variável* (*número_da_linha*, *nome_da_coluna*).
-assignbyidx *variável* ?*string*?	Atribui o resultado da consulta a um vetor, usando o valor do primeiro atributo e os nomes dos demais atributos como as chaves. Se a string for dada ela será anexada a cada chave: *variável* (*valor_do_primeiro_campo*,*nome_do_campo**string*).
-getTuple *número*	Retorna a linha indicada por *número*, na forma de uma lista, onde cada elemento da lista corresponde a uma coluna.
-tupleArray *númerovariável*	Retorna a linha indicada por *número* no vetor associativo *variável*, onde cada elemento do vetor corresponde a uma coluna e é identificado pelo nome da coluna: *variável*(*nome_da_coluna*).
-attributes	Retorna os nomes das colunas retornadas na consulta.
-lAttributes	Retorna os nomes das colunas retornadas na consulta e as suas propriedades: {{*nome tipo tamanho*}...}.
-clear	Limpa o objeto da consulta. Execute sempre que terminar de usar o objeto, ou ficará sem espaço para trabalhar com o banco de dados e um erro fatal ocorrerá.

Exemplos

Considere as seguintes tabelas:

districts

code	name
1	Arenoso
2	Arvoredo
3	Av. Bete
4	Amaralina
5	Aeroporto
6	Barros Reis
7	Barbalho
8	Bonocô
9	Brotas
10	Boca do Rio

products

code	name
1	Pizza Média
2	Pizza Grande
3	Pizza Família
4	Lazanha
5	Filé

prices

district	product	value
1	1	12.00
1	2	13.90
1	3	17.00
1	4	13.90
1	5	13.90
2	1	12.00
2	2	13.90
2	3	17.00
2	4	13.90
2	5	13.90

Elas são utilizadas em um programa para pizzaria de entrega a domicílio que eu estou terminando e que deverei liberar, sob licença GPL, até o final deste curso.
Abra o interpretador Tcl e digite os comandos a seguir(ou copie e cole se estiver no windows ou usando o kfm e o KTk no KDE):

```
load libpgtcl.so
set con [pg_connect pizza]
set res [pg_resul $con 'SELECT t1.name as district,t2.name as product,t3.value
FROM districts as t1, products as t2, prices as t3
WHERE t3.district = t1.code AND t3.product = t2.code
ORDER BY t1.name;' ]
puts "[format {%-30s} district][format {%-30s} product][format {%10s} value]\n\n"
for {set i 0} {$i < [pg_result $res -numTuples]} {incr i} {
```

```
    pg_result $res -tupleArray $i tuple
    puts "[format {%-30s} $tuple(district)][format {%-30s} $tuple(product)][format {%10s} $tuple
(value)]\n"
}
pg_result $res -clear
pg_disconnect $con
```

O fragmento de código acima imprimirá no console a tabela abaixo:

RESULTADO DA CONSULTA

district	product	price
Arenoso	Pizza Média	12.00
Arenoso	Pizza Grande	13.90
Arenoso	Pizza Família	17.00
Arenoso	Lazanha	13.90
Arenoso	Filé	13.90
Arvoredo	Pizza Média	12.00
Arvoredo	Pizza Grande	13.90
Arvoredo	Pizza Família	17.00
Arvoredo	Lazanha	13.90
Arvoredo	Filé	13.90

Para facilitar o trabalho com bancos de dados do PostgreSQL em Tcl/Tk eu criei uma biblioteca chamada **TDO**(**T**cl **D**atabase Access **O**bjects), que permite o acesso às tabelas de modo muito parecido com o MS-DAO, usado no Visual Basic. Você pode ler um pouco mais sobre a biblioteca TDO e fazer o download dos arquivos através do meu site: http://www.souzamonteiro.com/download_freeware.shtml.

Você também pode estudar o código fonte do meu programa **Cadastro de Clientes e Fornecedores para Linux** - TkCustomer. Faça o download dos arquivos através do meu site: http://www.souzamonteiro.com/download_freeware.shtml.

Para uma descrição detalhada de todos os comandos disponíveis em uma interface Tcl-RDBMS, consulte a documentação on-line do seu RDBMS. Para uma aula introdutória à linguagem SQL consulte a apostila **Programando em SQL** disponível para download em http://www.souzamonteiro.com/download_freeware.shtml.

20 Acessando um banco de dados ODBC

A interface ODBC-Tcl oferece diversos comandos que permitem o acesso aos bancos de dados ODBC.

Para acrescentar o suporte aos bancos de dados ODBC em seus programas, você deve incluir o comando a seguir, logo no início de seu programa:

```
package require tclodbc
```

Eu recomendo que sempre que for possível, você utilize o **ODBC** para acessar bancos de dados no **Windows** e também no **Linux**, pois fazendo assim, você tornará seus programas mais portáveis, devido ao uso de uma API única. De fato, eu acredito que, com o advento do **UnixODBC** e do **TclODBC** para o **MacOS**, o ODBC deverá se tornar a interface oficial do Tcl para acesso a bancos de dados. Na verdade, a versão UNIX do TclODBC, está sendo mantida pela Ajuba Solutions, o mantenedor oficial do Tcl. Assim podemos esperar que nas próximas versões do Tcl, o TclODBC faça parte integrante da distribuição oficial do produto.

20.1 Instalando o TclODBC no Windows

Para instalar o TclODBC siga os seguintes passos:

- Faça o download do arquivo tclodbc22.zip do meu site: http://www.souzamonteiro.com/download_freeware.shtml;
- Descompacte o arquivo em um diretório temporário;
- Mude para o diretório onde você descompactou os arquivos;
- Dê um duplo clique no arquivo setup.tcl;
- Confirme a instalação.

Isso instalará o tclodbc no diretório C:\Arquivos de Programas\Tcl\lib\tclodbc.

20.2 Instalando o TclODBC no Linux

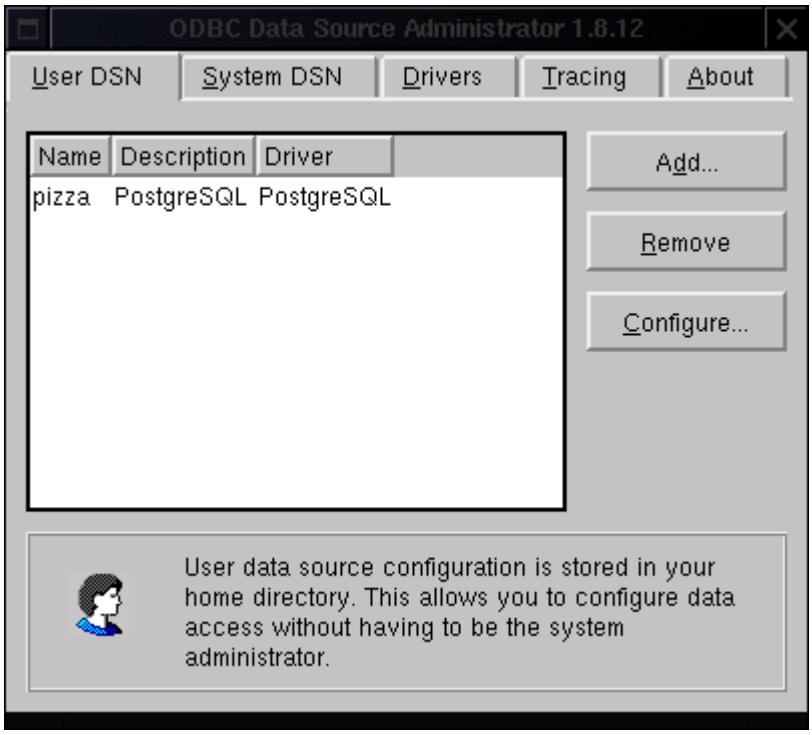
Para instalar o TclODBC siga os seguintes passos(lembre-se que você deve ser o usuário **root** para instalar as bibliotecas):

unixODBC:

- Certifique-se de ter instalado, em seu computador, as bibliotecas **Qt 2.0 Free Edition** ou superior, pois elas são necessárias para compilar os programas ODBCConfig e DataManager. Estas bibliotecas são usadas pelo **KDE**;
- Faça o download do **unixODBC** do meu site: http://www.souzamonteiro.com/download_freeware.shtml ou do site: http://www.unixodbc.org;
- Descompacte o arquivo digitando: **tar xvzf unixODBC-1.8.12.tar.gz**;
- Digite **configure**;
- Digite **make**;
- Digite **make install**;

Após a instalação você deverá instalar os drivers para os bancos de dados que você usará. Digite ODBCConfig na linha de comando do shell.

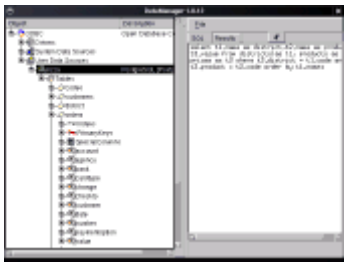
A janela a baixo será exibida:



Clique na guia **Drivers** e selecione a biblioteca do **driver** e a biblioteca do **setup** do driver. O nome da biblioteca do setup termina com **S.so**. Existe uma biblioteca de setup para cada driver.

Cada usuário pode configurar suas próprias entradas no **User DSN**, mas só o usuário root pode instalar drivers novos.

Você pode testar o seu banco de dados e executar comandos SQL através do programa **DataManager**. Veja a tela do programa na figura abaixo:



20.3 TclODBC:

- Instale os **fontes do Tcl/Tk** em seu computador. Se você não tiver, pege do meu site: http://www.souzamonteiro.com/download_freeware.shtml;
- Faça o download do arquivo **tclodbc.tgz** do meu site: http://www.souzamonteiro.com/download_freeware.shtml;
- Descompacte o arquivo digitando: **tar xvzf tclodbc.tgz**;
- Mude para o diretório *tclodbc/*;
- Digite **autoconf** para criar um script **configure**;

- Leia o arquivo README e verifique se há necessidade de passar alguns parâmetros para o script configure. Provavelmente **--prefix**=*/usr/lib* **--exec-prefix**=*/usr/lib* **--with-tcl**=*/usr/src/rpm/BUILD/tcltk-8.0.4/tcl8.0.4*;
- Execute o script **configure**;
- Eu recomendo que você edite o arquivo **Makefile** gerado e remova o parâmetro *USE_TCL_STUBS=1* que é passado para o compilador;
- Digite: **make**;
- Digite: **make install**.

Isso instalará o pacote tclodbc em seu computador.

20.4 Criando um banco de dados

Para criar um banco de dados do Access(MDB), copie e cole o fragmento de código abaixo:

```
# Carrega o TclODBC
package require tclodbc
# Cria o DSN
database configure config_dsn "Microsoft Access Driver (*.mdb)" [list "CREATE_DB=\"pizza.mdb\" General"]
database configure add_dsn "Microsoft Access Driver (*.mdb)" [list "DSN=pizza" "DBQ=pizza.mdb"]
# Conecta o banco de dados e cria as tabelas
database db pizza
db "CREATE TABLE districts (code VARCHAR (10), name VARCHAR (50));"
db "CREATE TABLE products (code VARCHAR (10), name VARCHAR (50));"
db "CREATE TABLE prices (district VARCHAR (10), product VARCHAR (10), price VARCHAR (10));"
# Disconecta o banco de dados
db disconnect
```

É claro que você deve verificar os erros retornados, mas nós ainda não falamos do tratamento de erros em Tcl. Isso será visto na aula 28.

20.5 A interface ODBC-Tcl

Os comandos oferecidos pela interface ODBC-Tcl são os seguintes:

Comando	Descrição
database *identificador fonte usuário senha*	Cria um objeto database, identificado por *identificador* e o conecta à *fonte*(datasource). O comando retorna o *identificador*.
database *identificador string_de_conecção*	Cria um objeto database, identificado por *identificador* e o conecta à *fonte*(datasource). O comando retorna o *identificador*.
database configure *operação driver atributos*	Configura a *fonte de dados* ODBC(datasource). Você pode adicionar uma fonte ao sistema, configurá-la ou remove-la. As *operações* são: add_dsn, config_dsn, remove_dsn, add_sys_dsn, config_sys_dsn, remove_sys_dsn.
database datasources	Retorna uma lista com as *fontes de dados*(datasources) configuradas, na forma: {*nome_da_fonte nome_do_driver*}.
database drivers	Retorna uma lista dos *drivers* configurados, na forma: {*nome_do_driver* {*atributo1=valor1 atributo2=valor2*...}}.

Os comandos oferecidos pelos objetos **database** são:

Método	Descrição
identificador consulta_SQL	Executa uma *consulta SQL* e retorna o resultado em uma lista, onde cada ítem da lista corresponde a uma linha retornada.
identificador disconnect	Disconecta o banco de dados.
identificador set *opção valor*	Configura várias *opções* relativas à conecção. Veja a tabela de opções a seguir.
identificador get *opção*	Retorna o valor de uma *opção* relativa à conecção. Vela a tabela de opções a seguir.
identificador commit	Finaliza um *transação*, salvando os valores.
identificador rollback	Cancela uma *transação*.
identificador tables ?*string*?	Retorna os nomes de todas as tabelas no banco de dados que combinam com a string dada. Os caracteres % e _ são coringas.
identificador columns ?*tabela*?	Retorna os nomes de todas as colunas na tabela dada, ou no banco de dados caso o nome da tabela seja omitido.
identificador indexes *tabela*	Retorna uma lista de todos os índices da tabela especificada.
identificador typeinfo *tipo*	Retorna infomações sobre o tipo de dado especificado.
identificador statement *id condulta_SQL*	Cria um objeto statement. Veja os métodos suportados pelos objetos statement a seguir.
identificador eval *proc consulta_SQL*	Executa uma procedure *proc*, para cada linha retornada na *consulta SQL*. O método passa para a procedure os valores das colunas retornadas. A procedure deve ter o seguinte cabeçalho: proc *procedimento* {*coluna1 coluna2*... colunaN} {*corpo_do_procedimento*...}.
identificador read *vetor consulta_SQL*	Coloca o resultado da consulta no vetor associativo *vetor*. O vetor *vetor* tem a seguinte forma: *vetor*(*valor_da_primeira_coluna_retornada*, *nome_da_coluna*).
identificador read { *vetor1 vetor2*...} *consulta_SQL*	Coloca o resultado da consulta no vetor associativo *vetor*. O vetor *vetor1* tem a seguinte forma: *vetor1*(*valor_da_primeira_coluna_retornada*) e contém os valores de todas as linhas, somente da primeira coluna retornada. O vetor *vetor2* tem a seguinte forma: *vetor2*(*valor_da_segunda_coluna_retornada*) e contém os valores de todas as linhas, somente da segunda coluna retornada e assim por diante.

As opções suportadas pelos métodos **set** e **get** são:

Opção	Valor
autocommit	booleano (0/1, on/off).
concurrency	readonly, lock, values, rowver
maxrows	Valor numérico.
timeout	Valor numérico.
maxlength	Valor numérico.
rowsetsize	Valor numérico.
cursortype	static, dynamic, forwardonly, keysetdriven
encoding	Uma codificação válida.
noscan	booleano (0/1, on/off).

Os métodos suportados pelos objetos **statement** são:

Método	Descrição
id	Executa a declaração e retorna os resultados imediatamente.
id execute *consulta_SQL*	Executa a declaração, mas não retorna os valores. As linhas devem ser lidas uma a uma através do método fetch.
id fetch ?*vetor*? ? *colunas*?	Retorna a próxima linha no resultado da consulta. Se for executado sem parâmetros retorna a linha em uma lista: {*coluna1 coluna2*...}. Vetor é um vetor associativo onde os nomes de cada elemento correspondem aos nomes das colunas retornadas: *vetor* (*nome_da_coluna*).
id rowcount	Retorna o número de linhas afetadas pelo último comando INSERT, UPDATE ou DELETE. Alguns drivers também podem retornar o número de linhas retornadas por uma declaração SELECT.
id columns ? *atributo1atributo2*...?	Retorna os atributos das colunas. Veja os atributos suportados, na tabela abaixo.
id set *opção valor*	Configura várias *opções* relativas ao objeto statement. Vela a tabela de opções a seguir.
id get *opção*	Retorna o valor de uma *opção* relativa ao objeto statement. Vela a tabela de opções a seguir.
id drop	Remove o objeto statement da memória.
id eval *proc consulta_SQL*	Veja os métodos dos objetos database acima.
id read *vetor consulta_SQL*	Veja os métodos dos objetos database acima.
id read {*vetor1 vetor2*...} *consulta_SQL*	Veja os métodos dos objetos database acima.

Os atributos suportados pelo método **columns** são:

Atributo	Descrição
label	Rótulo da coluna.
name	Nome original da coluna.
displaysize	A largura máxima da string de dados da coluna.
type	Tipo em SQL padrão.
typename	Tipo na string específica do banco de dados.
precision	Precisão numérica.
scale	Escala.
nullable	Se a coluna puder conter NULL, será 1.
updatable	Se a coluna puder ser atualizada, será 1.
tablename	Nome da tabela fonte da coluna.
qualifiername	Nome qualificador da tabela.
owner	Dono da tabela.

As opções suportadas pelos métodos **set** e **get** são:

Opção	Valor
concurrency	readonly, lock, values, rowver
maxrows	Valor numérico.
timeout	Valor numérico.
maxlength	Valor numérico.
rowsetsize	Valor numérico.

	static, dynamic, forwardonly, keysetdriven
cursortype	
noscan	booleano (0/1, on/off).

Os tipos de dados SQL são: **CHAR**, **NUMERIC**, **DECIMAL**, **INTEGER**, **SMALLINT**, **FLOAT**, **REAL**, **DOUBLE**, **VARCHAR**.

Os tipos extendidos são: **DATE**, **TIME**, **TIMESTAMP**, **LONGVARCHAR**, **BINARY**, **VARBINARY**, **LONGVARBINARY**, **BIGINT**, **TINYINT**, **BIT**.

20.6 Exemplos

Considere as seguintes tabelas:

districts

code	name
1	Arenoso
2	Arvoredo
3	Av. Bete
4	Amaralina
5	Aeroporto
6	Barros Reis
7	Barbalho
8	Bonocô
9	Brotas
10	Boca do Rio

products

code	name
1	Pizza Média
2	Pizza Grande
3	Pizza Família
4	Lazanha
5	Filé

prices

district	product	price
1	1	12.00
1	2	13.90
1	3	17.00
1	4	13.90
1	5	13.90
2	1	12.00
2	2	13.90
2	3	17.00
2	4	13.90
2	5	13.90

Elas são utilizadas em um programa para pizzaria de entrega a domicílio que eu estou terminando e que deverei liberar, sob licença GPL, até o final deste curso.

20.7 Criando as tabelas

Abra o interpretador Tcl e copie e cole os comandos a seguir(troque o valor da variável *user* pelo seu nome de usuário):

```
# Carrega o TclODBC
package require tclodbc
# Conecta o banco de dados e cria as tabelas
set user rlsm
database connect db pizza $user
db "CREATE TABLE districts (code VARCHAR (10), name VARCHAR (50));"
db "CREATE TABLE products (code VARCHAR (10), name VARCHAR (50));"
db "CREATE TABLE prices (district VARCHAR (10), product VARCHAR (10), price VARCHAR (10));"
# Disconecta o banco de dados
db disconnect
```

20.8 Inserindo dados nas tabelas

```
# Carrega o TclODBC
package require tclodbc
# Conecta o banco de dados e insere os dados
set user rlsm
database connect db pizza $user
db "INSERT INTO districts (code, name) VALUES (\'1\',\'Arenoso\');"
db "INSERT INTO districts (code, name) VALUES (\'2\',\'Arvoredo\');"
db "INSERT INTO products (code, name) VALUES (\'1\',\'Pizza Média\');"
db "INSERT INTO products (code, name) VALUES (\'2\',\'Pizza Grande\');"
db "INSERT INTO products (code, name) VALUES (\'3\',\'Pizza Família\');"
db "INSERT INTO products (code, name) VALUES (\'4\',\'Lazanha\');"
db "INSERT INTO products (code, name) VALUES (\'5\',\'Filé\');"
db "INSERT INTO prices (district, product, price) VALUES (\'1\',\'1\',\'12.00\');"
db "INSERT INTO prices (district, product, price) VALUES (\'1\',\'2\',\'13.90\');"
db "INSERT INTO prices (district, product, price) VALUES (\'1\',\'3\',\'17.00\');"
db "INSERT INTO prices (district, product, price) VALUES (\'1\',\'4\',\'13.90\');"
db "INSERT INTO prices (district, product, price) VALUES (\'1\',\'5\',\'13.90\');"
db "INSERT INTO prices (district, product, price) VALUES (\'2\',\'1\',\'12.00\');"
db "INSERT INTO prices (district, product, price) VALUES (\'2\',\'2\',\'13.90\');"
db "INSERT INTO prices (district, product, price) VALUES (\'2\',\'3\',\'17.00\');"
db "INSERT INTO prices (district, product, price) VALUES (\'2\',\'4\',\'13.90\');"
db "INSERT INTO prices (district, product, price) VALUES (\'2\',\'5\',\'13.90\');"
# Disconecta o banco de dados
db disconnect
```

20.9 Visualizando os valores contidos nas tabelas

Abra o interpretador Tcl e copie e cole os comandos a seguir(troque o valor da variável *user* pelo seu nome de usuário):

Implementação 1:

```
# Carrega o TclODBC
```

```
package require tclodbc
# Conecta o banco de dados
set user rlsm
database connect db pizza $user
# Executa a consulta
set res [db "SELECT t1.name as district,t2.name as product,t3.price
FROM districts as t1, products as t2, prices as t3
WHERE t3.district = t1.code AND t3.product = t2.code
ORDER BY t1.name;"]
# Exibe os resultados
puts "[format {%-30s} district][format {%-30s} product][format {%10s} price]\n\n"
for {set i 0} {$i < [llength $res]} {incr i} {
   set tuple(district) [lindex [lindex $res $i] 0]
   set tuple(product) [lindex [lindex $res $i] 1]
   set tuple(price) [lindex [lindex $res $i] 2]
   puts "[format {%-30s} $tuple(district)][format {%-30s} $tuple(product)][format {%10s} $tuple
(price)]\n"
}
# Disconecta o banco de dados
db disconnect
```

Implementação 2:

```
# Carrega o TclODBC
package require tclodbc
# Conecta o banco de dados
set user rlsm
database connect db pizza $user
# Cria uma procedure para exibir os resultados
proc showresults {district product price} {
   puts "[format {%-30s} $district][format {%-30s} $product][format {%10s} $price]\n"
}
# Exibe um cabecalho
puts "[format {%-30s} district][format {%-30s} product][format {%10s} price]\n\n"
# Executa a consulta e exibe os resultados
db eval showresults "SELECT t1.name as district,t2.name as product,t3.price
FROM districts as t1, products as t2, prices as t3
WHERE t3.district = t1.code AND t3.product = t2.code
ORDER BY t1.name;"
# Disconecta o banco de dados
db disconnect
```

Implementação 3:

```
# Carrega o TclODBC
package require tclodbc
# Conecta o banco de dados
set user rlsm
database connect db pizza $user
# Executa a consulta
db statement stmt  "SELECT t1.name as district,t2.name as product,t3.price
```

```
FROM districts as t1, products as t2, prices as t3
WHERE t3.district = t1.code AND t3.product = t2.code
ORDER BY t1.name;"
stmt execute
# Exibe os resultados
puts "[format {%-30s} district][format {%-30s} product][format {%10s} price]\n\n"
# tuple e o vetor associativo que contera os valores de cada linha retornada a cada execucao do comando fetch
while {[stmt fetch tuple]} {
   puts "[format {%-30s} $tuple(district)][format {%-30s} $tuple(product)][format {%10s} $tuple(price)]\n"
}
# Disconecta o banco de dados
db disconnect
```

Os fragmentos de código acima imprimirão no console a tabela abaixo:

RESULTADO DA CONSULTA

district	product	price
Arenoso	Pizza Média	12.00
Arenoso	Pizza Grande	13.90
Arenoso	Pizza Família	17.00
Arenoso	Lazanha	13.90
Arenoso	Filé	13.90
Arvoredo	Pizza Média	12.00
Arvoredo	Pizza Grande	13.90
Arvoredo	Pizza Família	17.00
Arvoredo	Lazanha	13.90
Arvoredo	Filé	13.90

20.10Alterando dados das tabelas

```
# Carrega o TclODBC
package require tclodbc
# Conecta o banco de dados
set user rlsm
database connect db pizza $user
# Insere os dados na tabela
db "INSERT INTO districts (code, name) VALUES (\'3\',\'Barbalho\');"
db "SELECT * FROM districts WHERE code = \'3\';"
# Atualiza os dados na tabela
db "UPDATE districts SET name = \'Av. Bete\' WHERE code = \'3\';"
db "SELECT * FROM districts WHERE code = \'3\';"
# Disconect o banco de dados
db disconnect
```

20.11Apagando dados das tabelas

```
# Carrega o TclODBC
package require tclodbc
```

```
# Conecta o banco de dados
set user rlsm
database connect db pizza $user
# Visualizando os dados na tabela
db "SELECT * FROM districts;"
# Apagando os dados da tabela
db "DELETE FROM districts WHERE code = \'3\';"
db "SELECT * FROM districts;"
# Disconect o banco de dados
db disconnect
```

20.12Criando índices

```
# Carrega o TclODBC
package require tclodbc
# Conecta o banco de dados
set user rlsm
database connect db pizza $user
# Cria os indices
db "CREATE INDEX districts_idx ON districts(code);"
db "CREATE INDEX products_idx ON products(code);"
db "CREATE INDEX prices_idx ON prices(district, product);"
# Exibe o resultado da consulta
db "SELECT t1.name as district,t2.name as product,t3.price
FROM districts as t1, products as t2, prices as t3
WHERE t3.district = t1.code AND t3.product = t2.code;"
# Disconect o banco de dados
db disconnect
```

Índices tornam as consultas SQL muito mais rápidas, contudo, alguns RDBMSs, não utilizarão os índices, caso você inclua a cláusula ORDER BY na sua declaração SELECT. Por outro lado, nem todos os DBMSs suportam índices. Assim, se você deseja tornar os seus programas realmente portáveis, não utilize índices, se realmente não precisar deles. Por exemplo, se sua tabela possui poucas linhas(menos de 500), não haverá necessidade de se utilizar índices.

21Imprimindo

Até bem pouco tempo só era possível imprimir, a partir de programas em Tcl/Tk, em ambiente UNIX. Isso porque não havia qualquer biblioteca de impressão para Windows ou MacOS.

Recentemente, alguns programadores começaram a desenvolver um sistema de impressão único para UNIX, Windows e MacOS. Esta solução, contudo, não é definitiva, pois a Ajuba Solutions está desenvolvendo seu próprio sistema de impressão, que deverá ser liberado em meados de outubro de 2000.

Nesta aula veremos como imprimir no Linux(UNIX) e Windows. Começaremos com o Windows, pois envolve a utilização de bibliotecas externas. Mas adiante veremos como imprimir em ambiente UNIX.

21.1 Imprimindo em ambiente Windows

Antes que possamos prosseguir, faça o download dos pacotes de impressão para Windows. Caso esteja utilizando a versão 8.2 do Tcl/TK, baixe os seguintes arquivos, do meu site, http://www.souzamonteiro.com/download_freeware.shtml:

- gdi823.zip

- prnt823.zip

Descompacte os arquivos no diretório **C:\Arquivos de Programas\Tcl\lib**.

Para acrescentar a funcionalidade de impressão aos seu programas em Tcl/Tk para Windows, inclua as seguintes linhas no início de seus programas:

```
package require printer
package require gdi
```

A seguir são apresentados os principais comandos suportados pelo pacote **printer.** Os mesmos comandos são válidos para o **MacOS**(arquivo printMac.c incluído nos fontes do pacote prnt823.zip) e o **UNIX**(baixe o pacote printUnix.tar.gz):

Comando	Descrição
printer attr [-hDC*hdc*] [-get {*atributo1 atributo2*...}|-set {{*atributo1valor*} {*atributo2 valor*}...}|-delete {*atributo1atributo2*...}]	Retorna, configura ou remove os atributos e valores especificados. -get retorna a lista de atributos: {{*atributo valor*}...}. -set configura os atributos especificados. -delete remove os atributos especificados.
printer close [-hDC *hdc*]	Fecha a impressora e libera o DC, concluindo os trabalhos(jobs) pendentes.
printer dialog [-hDC *hdc*] [select| page_setup] [-flags *flag*]	Exibe a caixa de diálogo especificada: select, para caixa de seleção de impressora; page_setup, para caixa de configuração de impressora.
printer job [-hDC *hdc*] [start|end]	Inicia ou finaliza um trabalho de impressão.
printer list [-match string] [-verbose]	Retorna uma lista de todas as impressoras que correspondem a *string*, se for fornecida. Se -verbose for utilizado, inclui informações adicionais sobre as impressoras.
printer open [-name nome_da_impressora] [-default] [-attr {{atributo valor}...}]	Abre a impressora e retorna o *hdc*.
printer option [*opção valor*] ...	Obtem ou configura opções.
printer page [-hDC*hdc*] [start|end]	Inicia ou finaliza uma página.
printer send [-postscript|-nopostscript] [-hDC *hdc*] [-printer *nome_da_impressora*] [-file *nome_de_arquivo*|-data *dados*]	Envia um arquivo no formato "raw" para a impressora. -postscript, significa que o arquivo está em formato ASCII; -nopostscript, significa que o arquivo está em formato binário.
printer version	Retorna a versão do pacote.

Em ambiente **UNIX** o pacote printer, simulará um **hdc**.

Os comandos a seguir são exclusivos para ambiente Windows, e constituem a interface **GDI**:

Comando	Descrição
gdi arc *hdc x1 y1 x2 y2* -extent *graus* -fill*cor*-outline cor -outlinestipple *bitmap* -start *graus*-stipple *bitmap*-style [pieslice|chord|arc] -width *largura_da_linha*	Semelhante ao widget canvas: desenha um arco.
gdi bitmap *hdc x y* -anchor [center|n|e|s|w] -background *cor* -bitmap *bitmap* -foreground *cor*	Desenha um bitmap X11.
gdi characters *hdc* [-font *fonte*] [-array vetor]	Cria um vetor com as larguras dos caracteres, no dispositivo ou fonte selecionada.
gdi copybits *hdc* [-window *w*|-screen] [-client] [-source "*a b c d*"] [-destination "*a b c d*"] -scale *número* -calc	Copia uma janela, uma área de uma janela *w* ou a tela inteira para o *hdc* de destino.

gdi image *hdc x y* -anchor [center|n|e|s|w] -image *nome*	Desenha uma *imagem* colorida, que tenha sido criada com o comando image de Tk.
gdi line *hdc x1 y1*... *xn yn* -arrow [first|last|both|none] -arrowshape {*d1 d2 d3*} -capstyle [butt|projecting|round] -fill *cor* -joinstyle [bevel|miter|round] -smooth [true|false] -splinesteps *número* -stipple *bitmap* -width *largura_da_linha*	Semelhante ao widget canvas: desenha uma linha.
gdi map *hdc* [-logical *x*[*y*]] [-physical *x*[*y*]] [-offset {*x y*}] [-default] [-mode *modo*]	Modifica o modo de mapeamento do dispositivo de contexto. Os modos suportados são os mesmos da API do Windows: MM_HIENGLISH(polegadas), MM_LOENGLISH, MM_HIMETRIC(milímetros), MM_LOMETRIC, MM_TWIPS(twips).
gdi oval *hdc x1y1x2y2* -fill *cor* -outline *cor* -stipple *bitmap* -width *largura_da_linha*	Semelhante ao widget canvas: desenha um oval.
gdi photo [-destination "*x y* [*w h*]"] -photo *nome*	Exibe uma imagem colorida nas coordenadas especificadas.
gdi polygon *hdc x1 y1*... *xn yn* -fill *cor* -outline *cor* -smooth [true|false] -splinesteps *número* -stipple *bitmap* -width *largura_da_linha*	Semelhante ao widget canvas: desenha um polígono.
gdi rectangle *hdc x1 y1 x2 y2* -fill *cor* -outline *cor* -stipple *bitmap* -width *largura_da_linha*	Semelhante ao widget canvas: desenha um retângulo.
gdi text *hdc x y* -anchor [center|n|e|s|w] -fill *cor* -font *fonte* -justify [left|right|center] -stipple *bitmap* -text *string* -width *largura_da_linha*	Semelhante ao widget canvas: exibe uma linha de texto.
gdi version	Retorna a versão da biblioteca.

Exemplos

O exemplo a seguir mostra como gerar um relatório utilizando as bibliotecas **gdi** e **printer**:

```
# Abre o job de impressao
set hdc [printer open]
printer job start
gdi map $hdc -mode MM_LOMETRIC

set novapagina 1

# As posicoes, as vezes, sao negativas, na vertical. Se for o seu caso,
# utilize um valor negativo para pos e decremente-o, ao inves de incrementa-lo,
# no loop abaixo. A minha impressora EPSON Stylus COLOR 600, a jato de tinta,
# requer valores negativos, ja a minha impressora XEROX DocuPrint P8e, a laser,
# reguer valores positivos. Multiplique o valor em milimetros por 10.
set pos 200
set numero 1

for {set i 0} {$i < 100} {incr i} {
   if {$novapagina == 1} {
      printer page start

      gdi text $hdc 200 $pos -anchor w -justify left -font {Arial 12 bold} -text "TESTE DE
IMPRESSAO"
      set pos [expr $pos + 50]
      gdi line $hdc 200 $pos 2000 $pos
```

```
        set pos [expr $pos + 50]
        set novapagina 0
        set numero 1
    }

    gdi text $hdc 200 $pos -anchor w -justify left -font {Arial 10} -text "Linha:"
    gdi text $hdc 500 $pos -anchor w -justify left -font {Arial 10} -text $i
    set pos [expr $pos + 50]

    incr numero

    if {$numero > 50} {
        set novapagina 1
        set pos 200
        set numero 1

        printer page end
    }
}

gdi text $hdc 200 $pos -anchor w -justify left -font {Arial 10} -text "Copyright(C) 2000 by Roberto Luiz Souza Monteiro"
set pos [expr $pos + 50]
gdi text $hdc 200 $pos -anchor w -justify left -font {Arial 10} -text "Este codigo pode ser reutilizado nos termos da licenca GNU LGPL"

printer page end

printer job end
printer close
```

21.2 Imprimindo em ambiente UNIX

Nenhuma biblioteca é necessária para imprimir em ambiente **UNIX**, pois o widget **canvas** é capaz de gerar uma saída **PostScript** que pode ser enviada diretamente para o comando **lpr**. Assim, veremos apenas um exemplo de como imprimir usando o widget canvas, sem entrar em detalhes sobre o comando **postscript**, já que ele foi apresentado na aula 13.

Exemplos

O exemplo a seguir utiliza um canvas para imprimir uma pagina de cada vez. Neste exemplo nao ha recurso de visualizacao:

```
# cria uma area de impressao
toplevel .t
wm withdraw .t
canvas .t.c -width 210m -height 297m
pack .t.c

# Configura uma flag para sinalizar a mudanca de pagina
set novapagina 1

# Posicoes em mm.
```

```
set unid m
set pos 20
set numero 1

for {set i 0} {$i < 100} {incr i} {
   if {$novapagina == 1} {
      .t.c create text 20m $pos$unid -anchor w -justify left -font {Arial 12 bold} -text "TESTE DE
IMPRESSAO"
      set pos [expr $pos + 5]
      .t.c create line 20m $pos$unid 2000 $pos$unid
      set pos [expr $pos + 5]
      set novapagina 0
      set numero 1
   }

   .t.c create text 20m $pos$unid -anchor w -justify left -font {Arial 10} -text "Linha:"
   .t.c create text 50m $pos$unid -anchor w -justify left -font {Arial 10} -text $i
   set pos [expr $pos + 5]

   incr numero

   if {$numero > 50} {
      set novapagina 1
      set pos 20
      set numero 1

      .t.c postscript -x 0 -y 0 -width 210m -height 297m -file /tmp/pagina$i.ps
      exec lpr /tmp/pagina$i.ps
      .t.c delete all
   }
}

# Destroi a area de impressao
destroy .t
```

Um exemplo mais avancado de impressao sera apresentado no final deste curso. Eu realizei alguns testes com o **canvasprintdialog** dos Iwidgets, do [incr Tcl/Tk], mais a caixa de dialogo apresentou muita instabilidade. Eu recomendo um estudo do codigo fonte da minha biblioteca de impressao, que faz parte dos meus mega-widgets. Procure na area de download do meu site: http://www.souzamonteiro.com.

Para uma descrição detalhada de todos os comandos disponíveis para impressão em Tcl/Tk, consulte a documentação da sua biblioteca de impressão.

22 Mega-widgets

Mega-widgets são widgets compostos de outros widgets. A distribuição oficial de Tcl/Tk, até a versão 8.3 contém apenas widgets simples. A partir da versão 8.4, serão incluídos também os

principais mega-widgets encontrados nas melhores extensões disponíveis, na Internet, para Tcl/Tk.

A seguir veremos algumas das extensões mais importantes. Nesta aula não ensinaremos a utilizá-las pois isso exigiria uma aula a parte para cada uma delas, pois, em geral, cada conjunto de mega-widgets utiliza uma sintaxe que lhe é própria e totalmente diferente das demais.

22.1 Bwidget

Os mega-widgets BWidget são os melhores para se desenvolver aplicações multi-plataforma, pois foram construídos em 100% puro Tcl.

Trata-se de um trabalho extremamente bem feito, que contém, inclusive, bibliotecas de suporte a idiomas diferentes, assim como classes para auxiliar o desenvolvimento de novos mega-widgets. A documentação é boa e trás um programa exemplo que demonstra todos os mega-widgets suportados.

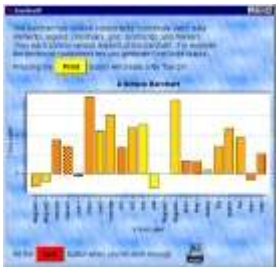
Mega-widgets Toolbar, Combobox, NoteBook, ScrollableFrame, Tree e Status bar dos BWidgets

Recentemente eu inclui o suporte à lingua portuguesa do Brasil. Pegue a distribuição com suporte pt-BR em http://www.souzamonteiro.com/tcl/BWidget-1_2_1_br.tgz.

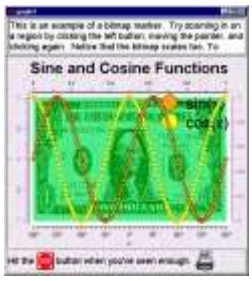
22.2 BLT

Os mega-widgets BLT são os melhores quando se deseja trabalhar com gráficos estatísticos. O Visual Tcl inclui suporte à área de plotagem de gráficos BLT.

A seguir são apresentadas algumas imagens de alguns dos tipos de gráficos suportados pela extensão BLT:



Bráficos de barras



Gráficos de pontos com uma imagem PostScript no fundo

Além dos gráficos estatísticos, BLT também suporta diversos outros mega-widgets incluindo o NoteBook, o Tree e o HyperText.

Pegue a extensão BLT para Windows em meu site: http://www.souzamonteiro.com. A extensão para Linux, já deve estar instalada em seu computador, pois costuma vir em todas as distribuições do Linux.

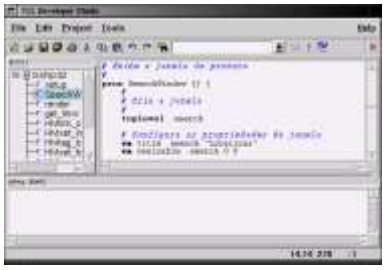
22.3 Tix

Tix é uma das extensões mais antigas para Tcl/Tk. Suporta mais de 40 mega-widgets, além de incluir suporte a pixmaps e, texto e imagem simultaneamente, em botões, lables, notebooks etc.

O único inconveniente de Tix é ter sido criada em C++, o que torna necessário recompilar a extensão para cada sistema operacional onde será utilizada. Até o momento Tix só é suportada na versão 8.0 ou inferior de Tcl/Tk.

O ambiente de programação Tcl Developer Studio foi inteiramente construído utilizando Tix.

A seguir é apresentada uma imagem do Tcl Developer Studio em ação:



Tcl Developer Studio:

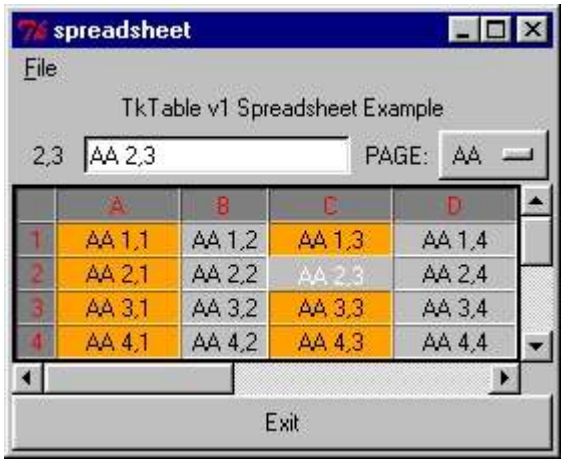
inteiramente construído com
os mega-widgets Tix

Pegue a extensão Tix para Windows em meu site: http://www.souzamonteiro.com. A extensão para Linux, já deve estar instalada em seu computador, pois costuma vir em todas as distribuições do Linux.

22.4 tkTable

tkTable é um mega-widget que oferece uma tabela que pode ser configurada para ser utilizada em planilhas eletrônicas ou para exibição e edição de tabelas em bancos de dados.

Veja a seguir uma planilha simples construída com tkTable:



Uma planilha simples construída com tkTable

Pegue a extensão tkTable para Windows e UNIX em meu site: http://www.souzamonteiro.com.

22.5 tkpiechart

Tkpiechart é um mega-widget para desenho de gráficos em formato pizza construído em 100% puro Tcl/Tk.

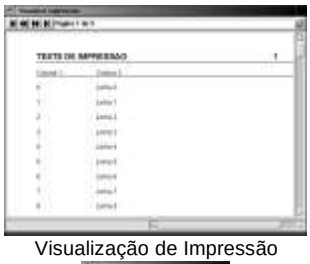
Veja a seguir uma imagem do tkpiechart em ação:

tkpiechart em ação

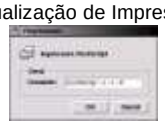
Pegue a extensão tkpiechart em meu site: http://www.souzamonteiro.com.

22.6 Souza Monteiro Widgets

São alguns mega-widgets que eu estou construindo.
A seguir são apresentadas algumas imagens dos meus mega-widgets:



Visualização de Impressão



Caixa de comfiguração
do comando de impressão

Caixa de diálogo
de impressão

Ajuda em HTML:
o browser TkShip pode ser utilizado
como um aplicativo ou como uma
caixa de diálogo de ajuda

Calculadora

Caixa de diálogo sobre

Caixa de diálogo dica do dia

Pegue os meus mega-wigets em meu site: http://www.souzamonteiro.com/tcl/widgets-1_0_0.tgz.

Para uma descrição detalhada de todos os mega-widgets disponíveis para Tcl/Tk, consulte o site http://dev.scriptics.com.

23 O Visual Tcl

O Visual Tcl é o ambiente de desenvolvimento mais utilizado para criação de aplicações gráficas na linguagem Tcl/Tk.

Totalmente escrito em Tcl/Tk, o Visual Tcl pode ser executado, sem qualquer modificação, em todos os sistemas operacionais, nos quais a linguagem Tcl seja suportada.

Alguns recursos disponíveis na versão atual(1.2.0) são:

Todas as ferramentas de formatação estão localizadas na *barra de ferramentas*. Assim é possivel modificar o lay-out do formulário ou as propriedades das fontes de um *Label* ao clique de um botão do mouse.

Os *widgets*(componentes) disponíveis na versão do *interpretador Tcl* que estiver sendo utilizado em uma seção, são todos agrupados em uma *palheta de componentes* semelhante à palheta de controles ActiveX do Visual Basic. Assim, caso se esteja utilizando o interpretador *Tix*, serão exibidos os *mega-widgets Tix* em adição aos widgets *Tk* básicos. Por outro lado, se o interpretador sendo utilizado for o *BLT*, será exibido o componente grafico em adição aos componentes *Tk* padrão.

A configuração dos atributos(propriedades) dos widgets é feita através de uma *palheta de propriedades* semelhante à do Visual Basic. Basta selecionar o widget e clicar na propriedade desejada na palheta de propriedades. Pressione ENTER após alterar uma propriedade.

Todas as funções e procedimentos existentes em um projeto são exibidos na *caixa de funções*. Para alterar a definição de uma função basta dar um duplo clique na função desejada.

Todas as janelas(toplevels) do projeto são exibidas na *caixa de janelas*. Para remover uma janela de um projeto, basta selecionar a janela na lista de janelas existentes e clicar no botão apagar.

A edição das janelas e feita simplesmente selecionando o componente desejado, clicando nas alças de seleção e arastando, exatamente do mesmo modo que é feito no Visual Basic.

A configuração de todos os eventos suportados pelo Tk é feita através da *caixa de eventos*(binds). Para abrir a caixa de configuração de eventos clique no widget cujo evento você deseja configurar e pressione as teclas **ALT+B**. Para configurar a opção **-command** de um widget, basta dar um duplo clique sobre o widget, que uma caixa para digitação dos comandos será aberta. Por exemplo, para configurar o evento PRESSIONAR ENTER basta pressionar ALT+B, clicar na caixa de *teclar*, e pressionar a tecla *Enter*. O Visual Tcl automaticamente criará o evento <Key-Return>. Basta então digitar as ações desejadas na caixa de comandos e clicar em adicionar.

É possível visualizar todos os widgets em uma janela, de forma hierarquisada, através da *caixa da árvore de widgets*. Basta clicar em um widget na árvore e ele será selecionado na janela(formulário).

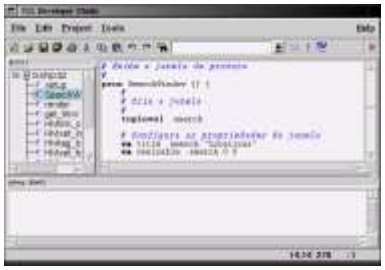
Além deste recursos o Visual Tcl também suporta componentes reaproveitáveis(compounds), que podem ser criados no próprio Visual Tcl. Compounds são semelhantes a componentes ActiveX, sendo escritos em puro Tcl/Tk.

Também é possível criar componentes para a Internet(Tclets). Tclets são parecidos com applets Java, mas são escritos em Tcl/Tk.

Faça o download da versão mais recente do Visual Tcl através do meu site: http://www.souzamonteiro.com/download_freeware.shtml.

24 O Tcl Developer Studio

O Tcl Developer Studio é o melhor ambiente para desenvolvimento de aplicações em Tcl/Tk, quando não se pode, ou não se deseja, criar visualmente uma aplicação. Semelhante ao MS-Developer Studio, inclui todos os recursos que um programador pode desejar, incluindo coloração de sintaxe e visualização dinâmica da estrutura do programa.



Totalmente escrito em Tcl/Tk/Tix, o Tcl Developer Studio pode ser executado, sem qualquer modificação, em todos os sistemas operacionais, nos quais a linguagem Tcl e os mega-widgets Tix sejam suportados.

Alguns recursos disponíveis na versão atual(1.0.0) são:

As principais ferramentas de edição estão localizadas na *barra de ferramentas*. Entretanto, recursos como *comentar bloco*, ou endentar bloco, só podem ser acessados através do *menu editar*, ou através das teclas de atalho.

Para executar a aplicação basta clicar no botão *executar*, na barra de ferramentas.

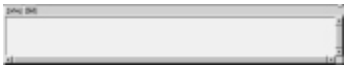
A visualização da estrutura do programa é dinâmica. Ao se criar uma função ela, imediatamente, aparece na *árvore de funções* do programa.

Também é possível trabalhar com vários arquivos ao mesmo tempo, cada um exibindo sua estrutura, de forma hierarquisada, na árvore do programa.

Para navegar até uma função, basta clicar na função na *árvore de funções* e a função será imediatamente exibida na *janela de edição*.

A *janela de edição* de código oferece *coloração de sintaxe* e *modelos* para funções e laços. De fato, ao se digitar as palavras *proc* ou *for* o programa imediatamente criará a estrutura correspondente.

Para se encontrar o início ou o final de um bloco basta posicionar o cursor antes da chave {} ou do cochete [] e o par correspondente será imediatamente destacado em vermelho.



É fácil depurar um programa com o Visual Tcl. Todas as saídas do programa para o console são exibidas na janela do *debuger*. Também é possível selecionar um interpretador ou um debuger externo e passar argumentos para o programa.

Faça o download da versão mais recente do Tcl Developer Studio através do meu site: http://www.souzamonteiro.com/download_freeware.shtml. Não esqueça de baixar também os mega-widgets Tix e instalá-los em seu sistema. Para rodar o Tcl Developer Studio você precisará do Tcl 8.0.5 e do Tix 4.1, pois Tix ainda não foi portado para as novas versões do Tcl. Você pode instalar várias versões do Tcl/Tk em uma mesma máquina, sem nenhum problema.

Se você instalar várias versões do Tcl em um sistema Windows, utilize um atalho para especificar o interpretador Tcl 8.0.5 para executar o Tcl Developer Studio e registre os programas com a extensão *.TCL para serem executados sob a versão do Tcl mais nova que você tiver instalado.

Baixe o Tcl 8.0.5 e o Tix 4.1 através do meu site: http://www.souzamonteiro.com/download_freeware.shtml.

Se você possui um Conectiva Linux 4.0 ou superior, não precisa instalar o Tcl 8.0.5 ou o Tix 4.1, pois o Tcl 8.0.4 e o Tix 4.1 já devem estar instalados em seu computador.

25 Compilando e empacotando programas

Existem diversos compiladores Tcl disponíveis na Internet, contudo, na minha opinião, o melhor deles é o **freeWrap**. Um empacotador que transforma seus scripts Tcl em programas executáveis, extremamente compactos.

O **freeWrap** cria executáveis com aproximadamente 400KB, no Linux, e 500KB no Windows, totalmente auto-contidos: nenhuma biblioteca ou programa adicional será necessário para executá-los.

Para converter um script Tcl em um programa executável utilize a seguinte síntaxe:

freewrap arquivo.tcl

Caso o seu programa Tcl seja formado apenas pelo script arquivo.tcl, será criado um executável arquivo.exe, no Windows, ou arquivo, no Linux.

A sintaxe geral do freeWrap é a seguinte:

freewrap *script_principal.tcl* [**-e**] [**-f** *lista_de_arquivos*] [**-p** *lista_de_pacotes*] [*arquivo*... *arquivo*]

Onde *programa_principal.tcl* é o nome do script principal do seu programa, *lista_de_arquivos* é um arquivo de texto contendo os nomes de todos os arquivos que devem ser embutidos no executável, *lista_de_pacotes* é um arquivo de texto contendo os nomes de todos os arquivos que devem ser empacotados no executável, e *arquivo* é o nome de um ou mais arquivos que devem ser embutidos no executável. A opção **-e** desliga a criptografia do código, que é ligada por padrão.

De um modo geral, qualquer tipo de arquivo pode ser embutido em um executável criado com o freeWrap. Você pode empacotar ícones, bibliotecas em Tcl, DLLs, arquivos HTML, imagens em formatos diversos etc.

Os scripts empacotados poderão ser carregados com o comando **source**:

source *arquivo.tcl*

Imagens poderão ser carregadas com o comando **image**:
image create photo *imagem* **-file** *imagem.gif*

Bibliotecas de carga dinâmica poderão ser carregadas com o comando **load**:
load *biblioteca.so*
load *biblioteca.dll*

Arquivos em formato texto ou binários poderão ser aberto com o comando **open**:
open *arquivo* [*modo*] [*permissões*]

O freeWrap inclui a extensão WINICO, disponível somente para o Windows, que permite atribuir um ícone no formato do Windows, a uma janela. O winico não é capaz de ler um arquivo de ícone embutido: você terá que salvá-lo no disco antes de usá-lo. Para tanto, leia-o de dentro do executável usando os comandos **open** e **read**, configurando o canal para o modo de tradução binário(**-translation binary**) e salve-o no disco rígido usando os comandos **open** e **puts -nonewline**. Em seguida utilize os comandos **winico create** e **winico setwindow** para atribuir o ícone à janela. Veja o exemplo:

```
set icofile smiley.ico

if {[file exists $icofile]} {
   set tmpfile c:/windows/temp/$icofile

   set fin [open $icofile r]
   fconfigure $fin -translation binary

   set fout [open $tmpfile w]
   fconfigure $fout -translation binary

   puts -nonewline $fout [read $fin]

   close $fin
   close $fout

   set ico [winico create $tmpfile]
   winico setwindow . $ico
}
```

Para uma descrição detalhada de todos os recursos do freeWrap, consulte a documentação do programa.
Para maiores informações envie e-mail para info@souzamonteiro.com.

26 Obtendo informações do interpretador Tcl

Tcl oferece diversos comandos que permitem acesso a informações sobre o interpretador, tais como variáveis existentes e procedures. Tcl também oferece o comando **interp**, com dezenas de sub-comandos, que permitem a criação e gerenciamento de múltiplos interpretadores. No entanto, múltiplos interpretadores são pouco usados e não serão abordados aqui.
Nesta aula, estudaremos o comando **info** e seus subcomandos, por ser o mais usado.

Comando	Descrição
info args	Retorna uma lista descrevendo os argumentos da procedure dada, na ordem em que eles aparecem no cabeçalho da procedure.
Info body	Retorna o corpo da procedure dada.
info cmdcount	Retorna o número total de comandos que foram invocados.
info commands	Retorna uma lista de todos os comandos Tcl que correspondem ao padrão fornecido.
info complete	Retorna 1 se a procedure dada for um comando Tcl completo. Caso contrário, retorna 0.
info default	Retorna 1 se a procedure dada possui um argumento *arg*, que tenha um valor default definido e coloca este valor na variável *var* dada.
info exists	Retorna 1 se a variável dada existir, no contexto atual. Caso contrário, retorna 0.
info globals	Retorna uma lista com todas as variáveis globais que correspondam ao padrão fornecido.
info hostname	Retorna o nome do computador a partir do qual o interpretador foi invocado.
info level	Retorna o nível na pilha, da procedure que está sendo invocada.
info library	Retorna o nome do diretório padrão onde os scripts Tcl estão instalados.
info loaded	Retorna uma lista descrevendo os .
info locals	Retorna uma lista com todas as variáveis locais que correspondem ao padrão dado.
info nameofexecutable	Retorna o caminho completo do binário a partir do qual o aplicativo foi invocado.
info patchlevel	Retorna a versão do interpretador Tcl na forma x.x.x.
info procs	Retorna uma lista com todas as procedures, no namespace atual, que correspondam ao padrão dado.
info script	Retorna o nome do script Tcl que está sendo executado.
info sharedlibextension	Retorna a extensão das bibliotecas compartilhadas, na plataforma atual.
info tclversion	Retorna a versão do interpretador Tcl na forma x.x.
info vars	Retorna uma lista com todas as variáveis atualmente visíveis, que correspondem ao padrão dado.

Exemplos

info args

puts [info args tclPkgUnknown]

Retorna os argumentos do comando tclPkgUnknown:

name version exact

info args

puts [info commands]

Retorna todos os comandos e procedures que correspondem ao padrão default (*):

after vwait uplevel continue auto_mkindex_old foreach rename fileevent regexp

tclPkgSetup upvar unset expr tcl_findLibrary load regsub history exit interp

puts incr lindex lsort tclLog string

info globals

puts [info globals]

Retorna todas as variáveis globais que correspondem ao padrão default (*):

tcl_rcFileName tcl_version argv argv0 tcl_interactive tcl_traceCompile auto_oldpath errorCode auto_path errorInfo auto_index env tcl_pkgPath tcl_patchLevel argc tcl_traceExec tcl_library tcl_platform

info nameofexecutable

puts [info nameofexecutable]

Retorna:

/usr/bin/tclsh

info patchlevel

puts [info patchlevel]

Retorna:

8.0.5

info procs

puts [info procs]

No contexto atual retornou:

tclMacPkgSearch auto_load_index unknown auto_import auto_execok pkg_mkIndex auto_mkindex auto_reset auto_qualify tclPkgUnknown auto_load pkg_compareExtension auto_mkindex_old tclPkgSetup tcl_findLibrary history tclLog

info sharedlibextension

puts [info sharedlibextension]

Como este texto está sendo editado no Linux, o comando retornou:

.so

info tclversion

puts [info tclversion]

Retorna:

8.0

info vars

puts [info vars]

No contexto atual retornou:

tcl_rcFileName tcl_version argv argv0 tcl_interactive tcl_traceCompile auto_oldpath errorCode auto_path errorInfo auto_index env tcl_pkgPath tcl_patchLevel argc tcl_traceExec tcl_library tcl_platform

Para uma descrição detalhada de todas as funções para acesso a informações sobre o interpretador Tcl, consulte a documentação on-line, ou o Tcl/Tk Reference Guide, ou ainda o Tcl/Tk Electronic Reference.

27 Trabalhando com Namespaces

Namespaces oferecem uma forma eficiente de encapsular variáveis e procedimentos, especialmente no caso de bibliotecas.

Um **namespace** é um espaço nomeado, dentro do qual podemos definir nossos procedimentos e variáveis sem poluir o espaço **global** com dezenas de definições, muitas vezes conflitantes.

De fato, tudo hoje, em Tcl é definido em termos de namespaces. Variáveis globais estão contidas no **namespace global** e a maior parte das extensões são definidas dentro de um namespace com o mesmo nome da extensão. Veja um exemplo: uma biblioteca matemática chamada Math, poderá definir seus procedimentos em um namespace Math e nos referiríamos a uma procedure quadradro de x, x2, como Math::x2.

Os principais comandos para tratamento de namespaces são:

Comando	Descrição
namespace children	Retorna uma lista de todos os namespaces filhos do *namespace* dado, que correspondem ao *padrão* fornecido.
namespace code	Retorna um novo script, que ao ser avaliado, faz com que o *script* dado seja executado no namespace atual.
namespace current	Retorna o nome completo do namespace dado.
namespace delete	Apaga cada um dos namespaces dados.
namespace eval	Ativa o namespace e avalia a concatenação dos argumentos em seu interior.
namespace export	Adiciona os comandos que correspondem ao *padrão* dado, à lista de comandos exportados. A opção -clear faz com que a lista anterior seja apagada e uma nova seja criada.
namespace forget	Remove os comandos importados, que correspondem ao *padrão* fornecido, do namespace atual.
namespace import	Importa os comandos que correspondem ao *padrão* dado, para o namespace atual. A opção -force substitui os comandos existentes.
namespace inscope	Ativa o namespace dado e avalia, em seu interior, a concatenação dos argumentos dados.
namespace origin	Retorna o caminho completo do *comando* dado.
namespace parent	Retorna o caminho completo do namespace pai do *namespace* dado.
namespace qualifiers	Retorna qualquer qualificador de namespace existente na *string* dada.
namespace tail	Retorna o nome simples, sem o qualificador de namespace a partir da *string* dada.
namespace which	Retorna o caminho completo de um *comando* dado, incluindo o namespace, se a opção -command for utilizada, ou. o caminho completo de uma *variável* dada, se a opção -variable for utilizada.
variable	Cria uma ou mais variáveis, no namespace atual, opcionalmente, atribuindo um *valor* a ela.

Exemplo

```
# Define o pacote
package provide Math 1.0

# Cria o namespace
namespace eval Math {

   # Define os comandos a serem exportados( neste caso, todos )
   namespace export *
}

# Cria uma procedure no interior do namespace( neste caso, x2 )
proc Math::x2 {value} {
   set ret "[expr $value * $value]"
   return $ret
```

}

Neste caso, a procedure x2 poderá ser acessada de duas formas:

```
package require Math
namespace import Math::*
puts [x2 4]
```

Ou

```
package require Math
puts [Math::x2 4]
```

Para uma descrição detalhada de todas as funções para tratamento de namespaces, consulte a documentação on-line, ou o Tcl/Tk Reference Guide, ou ainda o Tcl/Tk Electronic Reference.