Tcl/Tk 入門

佐賀大学理工学部知能情報システム学科・只木進一[1]

[1]Shin-ichi Tadaki: Department of Information Science, Saga University, Saga 840(E-mail:tadaki@ai.is.saga-u.ac.jp)

目 次

第1章　Tcl/Tkとは

UNIXの標準的GUI(Graphic User Interface)はX windowシステムと呼ばれるシステムである。X windowシステムは、二つの大きな特徴を持っている。

Server-Client モデル

実際にアプリケーションが実行されるマシンとその結果が表示されるマシンが異なる事が許されている。結果を表示するマシンにX windowのserverプロセスが起動され、アプリケーションがclientとしてそのserverプロセスにアクセスする。

Event-Driven

動作は、マウスのクリックやキーボードからの入力などの出来事(event)によって駆動(drive)される。

X上のアプリケーションは、通常C言語から呼び出すことが出来るライブラリを使ってプログラムされる。

Tcl/Tkは、X上のGUIを容易に構築するための言語パッケージである。Tclは基礎となるスクリプト型のプログラミング言語である。つまり、cshのように、コマンド列をファイルに記述することで、その順序に沿って作業を進めることが出来る。TkはTclの上で動く、X windowシステムのツールキット、つまり様々な部品(widget)の集合である。widgetとは、X windowに表示される、ボタンやラベルなどを指す。

なお、Tcl/TkはMacOSやMicrosoft Windowsでも使うことが出来る。

本テキストのねらいは、TclTkの概要を説明することである。受講者諸君は、演習問題を通じて、実際のTclTkプログラミングに親しんで貰いたい。TclTkには多数の機能が含まれているので、詳しいオプション等はマニュアルページを参照して欲しい。

第2章　Hello World

2.1　準備

emacsが適切にTcl/Tkのコードを扱えるように.emacsに次の行を加える。

Program 2.1.1 .emacs用追加コード

```
(autoload 'tcl-mode "tcl-mode")
(setq auto-mode-alist
        (append (list (cons "\\.tcl$" 'tcl-mode))
        auto-mode-alist))
```

このようにすると、拡張子が.tclであるファイルを開くと、自動的にTcl/Tk用のモードになる。この中では、例えばC-c M-aで編集中のバッファにあるTcl/Tkプログラムを実行することが出来る。他のコマンド類を見るにはC-h mとする (表2.1)。

演習 2.1　上記設定の後、

```
emacs hello.tcl
```

を実行し、正しくtclモード入り、下部に—TCLと表示されることを確認しなさい。また、tcl用コマンド一覧を表示してみなさい。

2.2　簡単な例

例 2.1　次のプログラムをhello.tclとして作成しよう。

Program 2.2.1 hello.tcl

```
#!/usr/local/bin/wish -f
#
# Hello World
label .hello -text "Hello World" -fg "red" -bg "yellow"
button .command -text QUIT -command exit
pack .hello .command -fill x
```

表 2.1: tcl-mode でのキーバインド

キー	関数名	内容
C-c M C-r	tcl-send-tcl-region	指定した領域を Tcl/Tk で実行する
C-c M C-e	tcl-set-tcl-region-end	領域終了指定
C-c M C-s	tcl-set-tcl-region-start	領域開始指定
C-c M]	tcl-end-of-proc	手続き (proc) の終端へ
C-c M [	tcl-beginning-of-proc	手続き (proc) の始点へ
C-c M i	tcl-get-error-info	エラー情報を得る
C-c M h	tcl-hide-process-buffer	tcl/Tk との更新バッファを隠す
C-c M s	tcl-show-process-buffer	tcl/Tk との更新バッファを表示する
C-c M u	tcl-restart-with-whole-file	ファイル全体を再度実行する
C-c M q	tcl-kill-process	実行中のスクリプトの強制停止
C-c M a	tcl-send-buffer	バッファ中のスクリプトを Tcl/Tk で実行する
C-c M w	tcl-send-proc	現在の手続きを Tcl/Tk で実行する
C-c M r	tcl-send-region	現在の領域きを Tcl/Tk で実行する
C-c M e	tcl-send-current-line	現在の行を Tcl/Tk で実行する
C-h m		コマンド一覧表示

これを一旦保存し、実行可能とする。

```
chmod +x hello.tcl
```

ファイル先頭の、#は通常はコメントを示す記号である。しかし、スクリプト型言語の最初にある場合には、#! によって、どのスクリプト型言語を使うかを示すことが出来る。

```
#!/usr/local/bin/wish -f
```

は、Tcl/Tk のスクリプトであることを示している。コマンドラインから

```
/usr/local/bin/wish
```

を実行して、会話的に widget を作りだすことも出来る。そのとき、終了は C-q である。

例で使われている Tcl/Tk コマンドについて順次説明しよう。

```
label .hello -text "Hello World" -fg "red" -bg "yellow"
```

label はラベル widget、つまり文字列や図形などを表示する widget の生成を指示する。widget の種類 (クラスと言う) と実際に生成される widget の名前を区別していることに注意が必要である。.hello はこの widget の名前で

ある。widget は階層的に定義することが出来る。最初の . が、一番上位階層のすぐ下の階層にこの widget を作ることを表している。表示される文字列が`-text` で、文字の色が`-fg` で、背景色が`-bg` で、それぞれ指定される。

次のコマンドは、`button` はマウスでクリック出来るボタンを生成する。

```
button .command -text QUIT -command exit
```

クリックされた時の動作が`-command` で指定される。

widget は生成されただけでは表示されない。表示には、それらを

```
pack .hello .command -fill x
```

のように、`pack` する必要がある。`-fill x` は各 widget を x 方向の幅を揃えて表示することを示している。

それぞれの widget の詳しい使い方は、オンラインで見ることが出来る。たとえば、label widget について知りたい場合には

```
man label
```

とコマンドラインから入力すれば良い。

演習 2.2 表示される、色、文字列を変えてみなさい。

演習 2.3 pack の部分を

```
pack .hello .command -side left -fill y
```

と変更して、動作の違いを見なさい。

2.3 toplevel widget

widget の階層構造の一番上が toplevel と呼ばれる widget である。通常、使う事はないが、一つのアプリケーションで複数のウィンドウを開く必要のある場合に用いる。

```
toplevel パス名
```

標準オプション以外のオプションとして表 2.2 のものを取ることが出来る。

表 2.2: toplevel widget の標準以外の主なオプション

オプション	内容
-background 背景色 -bg 背景色	背景色の指定
-class widget クラス名	クラス名の指定
-height 高さ	高さの指定
-screen スクリーン名	表示スクリーンの指定
-width 幅	幅の指定

2.3.1 screen オプション

他の X ウィンドウなど、明示的にスクリーンを指定するには

`-screen` スクリーン名

を指定する。

例 2.2

Program 2.3.1 toplevel widget の使用例 (hello2.tcl)

```
#!/usr/local/bin/wish -f
#
# Hello World
label .hello -text "Hello World" -fg "red" -bg "yellow"
pack .hello -fill x
toplevel .newtop
button .newtop.command -text QUIT -command exit
pack .newtop.command
```

第3章　ビットマップを表示する

3.1　ビットマップを表示するlabel widget

前章では、label widget を使って文字列を表示したが、label widget は文字列だけでなく、ビットマップと呼ばれる図を表示することが出来る。

例 3.1　ここでは、Tcl/Tk の内部ビットマップ、Tcl/Tk のデモンストレーション用ビットマップ、X ウィンドウが標準で持つビットマップを表示する例を作成する。

Program 3.1.1 bitmap.tcl

```
#!/usr/local/bin/wish -f
#
# bitmap.tcl
set xlib_include "/usr/X11R6.3/include/X11"
label .bitmap1 -bitmap error -fg "red" -bg "yellow" -anchor nw
label .bitmap2 -bitmap @$tk_library/demos/images/flagdown \
-fg "black" -bg "pink" -anchor se -relief raise
label .bitmap3 -bitmap @$xlib_include/bitmaps/xlogo64  \
-fg "green" -bg "white"
button .command -text QUIT -command exit -relief sunken
pack .bitmap1 .bitmap2 .bitmap3 .command -side top -fill x
```

例では、X のビットマップファイルのあるディレクトリ名が `xlib_include` として定義された後、引用時に`$`を着けているように、変数の引用には`$`を用いる。

3.2　label widget

表 3.1: label widget の標準以外の主なオプション

オプション	内容
-height 高さ	高さの指定
-width 幅	幅の指定

例のように、label widget は文字列やビットマップを表示することが出来る。

3.2.1　anchor オプション

表示する文字列やビットマップを、widget のどの位置に置くかを指定するオプションが anchor オプションである。指定は

```
n ne e se s sw w nw center
```

で行う。例えば`-anchor ne` とすると、右上に表示される。

3.2.2　bitmap オプション

次の 2 種類の方法でビットマップを指定することが出来る。

1. 内部ビットマップの指定
   Tcl/Tk は 8 種類のビットマップを内部に持っている。それらは

   ```
   -bitmap error
   ```

   のように、直接ビットマップ名を指定する。使える内部ビットマップの名前は

   ```
   error gray25 gray50 hourglass info questhead question warning
   ```

   である。
2. ビットマップファイルの指定
   直接、ビットマップファイルを指定するには`@`の後ろにファイル名を指定する。Tcl/Tk は、インストール先の変数名`$tk_library` を保持しているので、デモンストレーション用のビットマップを表示する場合に利用出来る。また、X のシステムが持つものを指定するには、パス名

   ```
   /usr/X11R6.3/include/X11/bitmaps
   ```

   の指定が必要になる。なお、X のコマンド `bitmap` でビットマップを作ることも出来る。

演習 3.1　他のビットマップを表示して見なさい。

3.2.3　background オプションと foreground オプション

色は、前景部分と背景部分に分けて指定する。色の名前で指定出来るものは

```
/usr/X11R6.3/lib/X11/rgb.txt
```

に指定されている。途中にスペースを含む場合には、"で囲む必要がある。また、RGB をそれぞれ 16 進 2 桁で表示して`#RRGGBB` と指定する事も出来る。

3.2.4 font オプション

表示する文字列のフォントを指定する。使えるフォント名は

```
xlsfonts
```

で見る事が出来る。

演習 3.2 色やフォントを変更して、表示して見なさい。

3.2.5 height オプションと width オプション

label widget の大きさの指定は height オプションと width オプションで行う。単位は

c	centimeters
m	milimeters
i	inches
p	printer's points (1/72 inch)
	pixels

である。

3.2.6 text オプションと textvariable オプション

表示する文字列の指定には text オプションと textvariable オプションを用いる。定数の場合には `text` を変数の場合には `textvarible` を用いる。

第4章　文字列を入力する

4.1　ビットマップを名前で指定して表示する

例 4.1　前章では、プログラムの内部で表示するビットマップ名を指定していた。今度は、entry widget を用いて、変数として入力する。

Program 4.1.1 bitmap2.tcl

```
#!/usr/local/bin/wish -f
#
# bitmap2.tcl
frame .entryframe
set bitmapdir "/usr/X11R6.3/include/X11/bitmaps/"
set bitmapname "xlogo64"
set bitmappath [format %s%s $bitmapdir $bitmapname]

label .bitmap -bitmap @$bitmappath -fg "green" -bg "white"
button .command -text QUIT -command exit
label .name -textvariable bitmappath
pack .entryframe .bitmap .command .name -side top -fill x

label .entryframe.name -text "bitmap name"
entry .entryframe.entry -bg white -textvariable bitmapname
button .entryframe.ok -text OK -command create_bitmappath
pack .entryframe.name .entryframe.entry .entryframe.ok -side left

proc create_bitmappath {} {
    global bitmapdir bitmapname bitmappath
    set bitmappath [format %s%s $bitmapdir $bitmapname]
    if {[file isfile $bitmappath]} {
        .bitmap config -bitmap @$bitmappath
    } else {
        set bitmappath "no such bitmap"
    }
}
```

ビットマップ名入力用の3つの widget が frame widget `.entryframe` の下位に作成されている。entry widget に文字列を入力し、ボタン `.entryframe.ok` を押すと、ファイルの存在が確認されて、表示される。

proc create_bitmappath は、指定されたビットマップを探して表示するための副手続きである。

```
set bitmappath [format %s%s $bitmapdir $bitmapname]
```

でビットマップファイルへの絶対パスが設定される。

```
.bitmap config -bitmap @$bitmappath
```

によって、既に作成されている widget .bitmap に新しいビットマップが表示される。この例のように、既に作成されている widget の属性を変えるには、

widget パス名 config オプション

と指定する。

4.2 frame widget

frame widget は最も単純な widget で、四角が表示される。また、下位に widget を持たせるのに用いる。

表 4.1: frame widget の標準以外の主なオプション

オプション	内容
-class widget クラス名	クラス名の指定
-colormap path 名	color map の指定
-height 高さ	高さの指定
-relief	3 次元的見え方
-visual	見え方
-width 幅	幅の指定

4.2.1 colormap オプション

X window システムでは、colormap と呼ばれる資源を使って、アプリケーションが指定する色を実際に表示される色に変換する。colormap の指定は new として新たに生成するか、他の widget へのパスを指定する。

4.2.2 visual オプション

```
staticgrey greyscale staticcolor pseudocolor
directcolor truecolor
```

のいずれかを指定し、色の表現力を指定する。

4.2.3 relief オプション

-relief は3次元的な見え方を示し raised、sunken、flat、groove、ridge の 5 種類が使える。

4.3 entry widget

entry widget は 1 行の文字列の入力、編集に用います。

表 4.2: entry widget の主なオプション

オプション	内容
-background 背景色 -bg 背景色	背景色の指定
-borderwidth 幅 -bd 幅	外側枠の幅
-cursor カーソル名	マウスカーソルの表示
-show 文字	入力を指定した文字で表示
-state 状態	入力の許可
-width 幅	幅の指定

4.3.1 show オプション

entry widget からパスワードを入力するなど、入力文字が表示されずに、* などが代わりに表示される必要がある場合があります。その場合には、オプションとして

```
-show *
```

と指定します。

4.3.2 state オプション

-state 状態

状態として disable を指定する事で、内容の変更を禁止します。書き換え禁止を解除するには、状態として normal を指定します。

4.4　副手続き

Tcl は作業の手順をその順序に従って記述するスクリプト型言語である。しかし、主となる作業から分岐した副手続きを記述することが出来る。副手続きは、ボタンをクリックした際の動作などとして記述することが出来る。

副手続きは、キーワード `proc` で始まる

`proc` 手続き名 引数リスト 手続き本体

で記述される。例 4.1.1 における `create_bitmappath` がそれである。例の場合、引数が無い。

副手続きの中で使われる変数は、副手続きの中だけで有効な変数である。このような変数を一般に局所変数 (local variables) と呼ぶ。また、変数の有効な範囲をスコープ (scope) と呼ぶ。

通常は、副手続きの外部の変数は、引数を通じて引用するが、`global` と宣言することで、外部の変数を使うことが出来る。

演習 4.1　ビットマップが別のウィンドウに表示されるように変更しなさい。

演習 4.2　フォント名と色を入力すると、そのフォントと色でアルファベットが表示されるアプリケーションを作成しなさい。

第5章　色見本を作る (その1)

5.1　色見本の作成例

次に、色見本を見るアプリケーションを作りながら、Tcl/Tk プログラミングの方法を見て行く。第 3.2.3 節で述べたように、X window では、赤緑青 (RGB) の三色を混ぜることで一つの色が指定されている。それぞれの配合は 0 から 255 までの数で指定され、16 進数 2 桁で表される。例えば赤は`#ff0000` で、青は`#0000ff` で表される。

例 5.1　まず、簡単な色表示の例を示す。

Program 5.1.1 `Color1.tcl`

```
#!/usr/local/bin/wish -f
#
# color1.tcl
set red 255
set green 0
set blue 255
set color [format #%02x%02x%02x $red $green $blue]
frame .sample -width 20m -height 20m -bg $color
button .command -text QUIT -command exit  -relief raised
pack .sample .command -side left -fill y
```

この例では、色を表す三つの変数 `red`、`green`、`blue` を使っている。値への代入は `set` を用いる。引用する時には変数名の前に`$`が付くことに注意する。

三色の調合した色を変数 `color` に設定し、label widget の背景色として指定している。

5.2　変数の扱い

Tcl/Tk では、変数の型という概念がない。つまり、整数でも実数でも、文字列でも同じように扱うことができる。また、C のようにあらかじめ型宣言をしておく必要もない。

1 次元配列を用いる場合には

```
set arrayname(index) value
```

表 5.1: 変数の扱い

`append` *variable values* 変数 *variable* に値 *values* を後ろから追加する。つまり *variable* は文字列のように扱われる。*values* を複数指定することができる。
`incr` *variable value* 変数 *variable* に値 *value* を加えたものを新しい *variable* の値とする。
`set` *variable value* 変数 *variable* に値 *value* を設定する。*value* を省略すると、現在の値を返す。
`unset` *variable* 変数 *variable* を消去する。

のように () を用いる。2 次元以上の配列も同様に

```
set arrayname(index1,index2) value
```

と扱う。Tcl では、これらはインデックスを含めた一つの変数名として扱われる。従って `a(1,1)` と `a(1, 1)` は異なるものであることに注意する。

式を計算してその値を変数に代入する場合にも `set` を用いる。例えば

```
set a [expr $b + 2]
```

では変数 `b` に 2 を加えた値を `a` に代入する。

5.3　式の表現

第 5.2 節で述べたように、式の表現には `expr` を用いる。式の表現方法を表 5.2 にまとめる。

演習 5.1　RGB の値を entry widget から入力して、色を表示出来るように変更しなさい。

演習 5.2　上記アプリケーションに加えて、RGB の値を反転する機能を追加しなさい。ここで、反転するとは、たとえば r の値に対して $255 - r$ に置き換えることとする。

表 5.2: 式の表現方法。適応する型とは、被演算子の型である。但し $a?b:c$ の場合は、a の型を表す。

表現	内容	適応する型
-a	符号反転	int,real
!a	論理否定	int,real
~a	ビットごとの反転	int
a*b	積	int,real
a/b	商	int,real
a%b	a を b で割った余り	int
a+b	和	int,real
a-b	差	int,real
a<<b	a を b ビット、左へシフト	int
a>>b	a を b ビット、右へシフト	int
a<b	$a < b$ ならば 1、その他は 0	int,real,string
a>b	$a > b$ ならば 1、その他は 0	int,real,string
a<=b	$a \leq b$ ならば 1、その他は 0	int,real,string
a>=b	$a \geq b$ ならば 1、その他は 0	int,real,string
a==b	$a = b$ ならば 1、その他は 0	int,real,string
a!=b	$a \neq b$ ならば 1、その他は 0	int,real,string
a&b	ビットごとの AND	int
a^b	ビットごとの XOR	int
a|b	ビットごとの XOR	int
a&&b	論理積	int,real
a||b	論理和	int,real
a?b:c	選択: a が 0 以外ならば b、それ以外は c	int,real

第6章　色見本を作る (その2)

6.1　色を変化させる

前章の例では、RGB のそれぞれの値をスクリプト内に記述していた。これを画面上から操作することを考える。操作を、マウスだけで行うために scale という widget を使う。

例 6.1　scale widget を使って整数値を取る変数 `IntValue` を 0 から 255 まで変化させ、その結果を label widget に表示する。

Program 6.1.1 `color2.tcl`

```
#!/usr/local/bin/wish -f
#
# scale.tcl
set IntValue 0
scale .scale -label Value -from 0 -to 255 -length 10c \
        -orient horizontal -command SetValue
label .value -textvariable IntValue -fg purple
button .command -text QUIT -command exit  -relief raised
pack .scale .value .command -fill x

proc SetValue value {
    global IntValue
    set IntValue [.scale get]
}
```

scale widget の動作に伴って起動される手続きが `SetValue` である。手続き内の変数は局所変数であるので、全域的変数 `IntValue` を `global` 宣言して使っている。

また、label widget で変数を表示するには

`-textvariable` 変数名

を使う。

6.2　scale widget

scale widget は、一定範囲内の数値を棒状の widget から入力するのに使う事が出来る。

6.2.1　範囲の指定

数値の範囲は

`-from` 開始 `-to` 終了

で指定される。例題では整数が指定されているが、実数を使う事が出来る。有効桁は`-digit` で指定され、数値は切り捨てられる。

演習 6.1　scale widget を使って、RGB 三色の値を操作し、色見本を表示するスクリプトを作成しなさい。ただし、一旦生成された widget の属性を変更して再描画するには

widget 名 `config` 属性

と記述する。例えば`.sample` の前景色を赤に変えるには、以下のように行う。

```
.sample config -fg red
```

第7章　色見本を作る (3)

7.1　カラーデータベースに登録された色を表示する

次は、システムのカラーデータベース/usr/X11R6.3/lib/X11/rgb.txt に登録されている色を表示しよう。

例 7.1　まず、カラーデータベースから色の名前一覧を取り出す副手続き list_color と色を表示する副手続き setcolor を定義する。色データベースの内容は一行ごとに読み込まれ、特別な正規表現で表されるパターンに一致した場合に、色の名前が読み込まれる。

Program 7.1.1 Color3.tcl

```
#!/usr/local/bin/wish -f

# カラーのデータベースを読む
set colordb "/usr/X11R6.3/lib/X11/rgb.txt"
proc list_color {w} {
    global colordb
    set fr [open $colordb r]
    $w delete 0 end
    foreach line [split [read $fr] \n] {
        if {[regexp\
                {( *[0-9]+) ( *[0-9]+) ( *[0-9]+)[^a-zA-Z]*([a-zA-z].*)$}\
                        $line match r g b colorname]} {
            $w insert end $colorname
        }
    }
    close $fr
}

# カラーを設定する
proc setcolor {w y} {
    global colorname
    set i [$w nearest $y]
    set colorname [$w get $i]
    .color config -bg $colorname
}
```

プログラムリスト 7.1.2が各 widget の生成部である。.colortable という frame widget の中に、scrollbar widget と listbox widget が生成される。カ

ラーデータベースから読み込まれた名前は、listbox widget に表示される。色の名前を全て表示することは不可能なので、スクロールバーを設置している。

一旦生成した widget に動作を追加するのが bind である。

```
bind .colortable.color <Button-1> {setcolor %W %y}
```

はクリックで、手続き setcolor が起動するような操作である。引数として、イベントが起こった widget %W とその y 座標%y を副手続きに渡している。これにより、色の名前をクリックすることが label widget の背景色としてその色を表示される。

Program 7.1.2 Color3.tcl の続き

```
#
button .command -text QUIT -command exit  -relief raised
frame .colortable -relief sunken
set colorname none
label .color -textvariable colorname -fg black
pack .colortable  -fill both -expand yes
pack .command .color -fill both -expand no

# スクロールバーの設定
scrollbar .colortable.xscroll \
        -command ".colortable.colorlist xview" -orient horizontal
scrollbar .colortable.yscroll \
        -command ".colortable.colorlist yview" -orient vertical
pack .colortable.xscroll -side top -fill x -expand no
pack .colortable.yscroll -side left -fill y -expand no
# リストボックスの設定
listbox .colortable.colorlist -width 40 -height 20\
        -xscroll ".colortable.xscroll set" \
        -yscroll ".colortable.yscroll set" -selectmode single
pack .colortable.colorlist -side left -fill both -expand yes
# クリック時の動作の定義
bind .colortable.colorlist <Button-1> {setcolor %W %y}
# カラーデータベースを読み表示する
list_color .colortable.colorlist
```

7.2 listbox widget

7.2.1 listbox widget の使途

複数行のテキストを表示する際に使うのが listbox widget である。行の選択、挿入及び削除は可能だが、各行の内部のテキストの編集機能は有していない。つまり、多数の選択肢を示すのに有用な widget である。テキストの編集機能を持つ text widget と使い分ける必要がある。

表 7.1: listbox widget の標準以外の主なオプション

オプション	内容
-height 高さ	高さの指定
-selectmode SelectMode	要素選択方法の指定
-width 幅	幅の指定

7.2.2 selectmode オプション

selectmode オプションは、listbox widget の要素の選択方法を指定するオプションであり、

```
single browse miltiple extended
```

の 4 種類のいずれかを指定する。single が指定されると、マウスをクリックした一つの要素だけの選択が許される。browse が指定されると、マウス左ボタンを押したままドラッグすることが出来るが、選択は一つの要素だけが許される。これが既定値である。multiple を指定すると、各要素をトグルで選択することが出来る。extended はシフトキーを押しながらドラッグすることで、複数要素を選択出来る。

7.2.3 scroll widget との利用

listbox widget は、通常 scroll widget と共に利用する。親となる widget を frame widget や label widget を使って作成し、その子どもとして、listbox widget と scroll widget を定義する。

7.2.4 listbox widget の要素の操作

listbox widget は各要素を指定するインデクスを持っている。インデクスは最初の行の verb—0—から、最後の行の end までである。指定は、数字または表 7.2 にあるキーワードを使う。

新たにリストを表示する場合には、listbox widget の内容を全て消去する。

```
$w delete 0 end
```

ここで$w は widget を表す。この後、文字列 string を順次追加していく。

```
$w insert end $string
```

要素を選択する場合は、マウスクリックなどが起こった場所の座標 y を使って

表 7.2: listbox widget のインデクス

インデクス	内容
0	先頭行
active	現在選択されている行
anchor	選択範囲の基準点
end	最後の行
数字	行
`@x,y`	指定された座標 (x, y) に最も近い行

```
set i [$w nearest $y]
set text [$w get $i]
```

と内容を `text` に得る。

7.3　bind コマンド

bind コマンドは、X イベントに tcl コマンドを結びつける。たとえば、マウスクリックによって引き起こされる動作を決める。

`bind` バインドタグ　イベント名　コマンド

バインドタグは、widget クラス名か具体的に生成された widget へのパス名である。イベント名は、マウス操作やキー操作などの X イベントである。指定されたバインドタグに対して、イベントが送られた時の動作がコマンドで指定される。

7.3.1　イベントの構文

イベントの構文は

<修飾子-型-詳細>

である。一番大事なものが型であり、キーが押されたり、マウスが動いたりするイベントを指す。詳細は、マウスやキーボードののキーを指すのに使われる。修飾子は複数連続させることも可能で、イベントが発生したときにコントロールキーが押されているなどを指定する。

7.3.2　キーボード イベント

たとえば `a` が押された場合のキーボード イベントは以下のように省略することも出来る。

表 7.3: イベントの型

イベント名	内容
ButtonPress, Button	ボタンが押された
ButtonRelease	ボタンが放された
Circulate	ウィンドウの重ね順序が変更された
Colormap	カラーマップが変更された
Configure	ウィンドウの大きさ、位置、境界、重ね順序が変更された
Destroy	ウィンドウが破壊された
Enter	マウスカーソルがウィンドウに入った
Expose	ウィンドウが開かれた
FocusIn	ウィンドウがフォーカスを得た
FocusOut	ウィンドウがフォーカスを失った
Gravity	親ウィンドウの大きさが変更された結果、ウィンドウが移動した
Keymap	ウィンドウのキー割り当てが変更された
KeyPress, Key	キーが押された
KeyRelease	キーが放された
Motion	マウスカーソルがウィンドウ中を移動した
Lease	マウスカーソルがウィンドウを離れた
Map	アイコン状態のウィンドウが開かれた
Property	ウィンドウのプロパティが変更されたか、削除された
Reparent	ウィンドウの親が変わった
Unmap	ウィンドウがアイコンになった
Visibility	ウィンドウの可視属性が変化した

```
<KeyPress-a>
<Key-a>
```

キーボードの詳細情報は**keysym**と呼ばれる。現在のキーボードへのキーマップの割り当ては

```
xmodmap -pke
```

で見る事が出来る。表示文字以外ではたとえば次のような表現を使うことが出来る。

```
Return, Escape, Backspace, Tab, Up, Down, Left, Right,
comma, period, dollar, asciicircum, numbersign, exclam
```

7.3.3　マウスイベント

マウスイベントも省略して表現することが出来る。例えばマウス左ボタンを押すイベントは

```
<ButtonPress-1>
<Button-1>
```

と表すことが出来る。

7.3.4　イベント修飾子

イベント修飾子は、イベント発生時に、Control キーが押されているなど、同時に押されているキーを指定する。修飾子 Mod1 から Mod5 はキーボードごとに異なる。現在の設定を見るには

```
xmodmap
```

を実行する。マウスボタンのダブルクリックやトリプルクリックも修飾子として記述される。

表 7.4: イベント修飾子

イベント修飾子名	内容
Control	Control キー
Shift	Shift キー
Lock	—verb—Lock—キー
Meta, M	Meta_R と Meta_L キー
Alt	Alt_R と Alt_L キー
Mod1, M1	修飾子
Mod2, M2	修飾子
Mod3, M3	修飾子
Mod4, M4	修飾子
Mod5, M5	修飾子
Button1, B1	第 1 マウスボタン (左)
Button2, B2	第 2 マウスボタン (中央)
Button3, B3	第 3 マウスボタン (右)
Button4, B4	第 4 マウスボタン
Button5, B5	第 5 マウスボタン
Double	
Triple	
any	全ての修飾子

7.3.5 イベント列

複数イベントの連続を bind に指定することも出来る。例えば

```
bind $w <Control-x><Control-c> {exit}
```

は<Control-x><Control-c>の連続イベントで終了することを指定する。

演習 7.1 フォント名一覧を示すコマンド xlsfonts の出力を listbox に格納し、フォント名をクリックすることでそのフォント名をそのフォントで表示するアプリケーションを作成しなさい。

フォント名一覧を listbox に表示するには、例えばプログラム 7.3.1 のような副手続きを使うことが出来る。

Program 7.3.1 フォント一覧を listbox に格納するための副手続き例

```
# フォントのデータベースを読む
proc list_font {w} {
    $w delete 0 end
    foreach line [split [exec xlsfonts] \n] {
        $w insert end $line
    }
}
```

第8章　ファイルブラウザを作る

8.1　ファイル一覧

次に、ファイル検索用アプリケーションをつくってみる。ファイル一覧を見るには、UNIX のコマンド `ls` を実行すればよい。Tcl/Tk では、UNIX のコマンドなど外部のコマンドを `exec` で実行することが出来る。

例 8.1

Program 8.1.1 `list1.tcl`

```
#!/usr/local/bin/wish -f
proc list_file {w} {
    set dir [pwd]
    set flist [exec ls -a $dir]
    $w delete 0 end
    foreach line $flist {
        if {[file isdirectory $line]} {
            $w insert end ${line}/
        } else {
            $w insert end ${line}
        }
    }
}

proc fileopen {w y} {
    set i [$w nearest $y]
    set filename [$w get $i]
    if {[file isdirectory $filename]} {
        cd $filename
        list_file $w
    }
}
```

例—8.1.1 はファイルの操作を行う手続きの定義部分である。手続き `file_list` は、現在のディレクトリの一覧を作成する。

```
set flist [exec ls -a $dir]
```

によって、変数 `flist` にファイル一覧が格納される。その後、listbox widget `.filetable.filelist` に、その内容が一行ずつ表示されている。ただし、

Program 8.1.2 list1.tcl のつづき

```
button .command -text QUIT -command exit  -relief raised
label .filetable -relief sunken
pack .filetable .command -fill both -expand yes
scrollbar .filetable.xscroll \
        -command ".filetable.filelist xview" \
        -orient horizontal
scrollbar .filetable.yscroll \
        -command ".filetable.filelist yview" \
        -orient vertical
pack .filetable.xscroll -side top -fill x -expand no
pack .filetable.yscroll -side left -fill y -expand no
listbox .filetable.filelist -width 40 -height 20 \
        -xscroll ".filetable.xscroll set" \
        -yscroll ".filetable.yscroll set"
pack .filetable.filelist -side left -fill both -expand yes
bind .filetable.filelist <Double-Button-1> {fileopen %W %y}

list_file .filetable.filelist
```

```
if{[file isdirectory $line]}
```

によって、そのファイルがディレクトリであるかを判断し、ディレクトリである場合には、最後に/が追加されるようにしている。

手続き fileopen が実際のファイルを操作する部分である。ファイル名がディレクトリである場合には、そのディレクトリへ移動する。通常のファイルである場合には、変数 filename にファイル名が格納されるだけである。

例 8.1.2 は各 widget の生成の部分である。まず、.filetable という label widget の中に、scrollbar widget と listbox widget を生成する。ファイル一覧が長い場合に、スクロールバーの操作で、listbox が動かすことが出来るようにしている。

8.2　UNIX との連係

8.2.1　exec コマンド

Tcl/Tk から外部のコマンドを実行するには、exec コマンドを利用する。プログラム例 8.1.1 では

```
set flist [exec ls -a $dir]
```

によって、UNIX コマンド ls を実行し、その結果を変数 flist に格納している。

8.2.2 file コマンド

file コマンドは、ファイルシステムに関する情報を調べるコマンドである。

表 8.1: file コマンド

コマンド	内容
`file atime` *name*	最後に *name* をアクセスした時間を 10 進数で返す
`file dirname` *name*	*name* の親ディレクトリ名を返す
`file executable` *name*	*name* が実行可能であれば 1 を、それ以外ならば 0 を返す
`file exists` *name*	*name* が存在すれが 1 を、それ以外ならば 0 を返す
`file extension` *name*	*name* 中のドット付き拡張しを返す
`file isdirectory` *name*	*name* がディレクトリならば 1 を、それ以外ならば 0 を返す
`file isfile` *name*	*name* がファイルならば 1 を、それ以外ならば 0 を返す
`file lstat` *name var*	リンク *name* の `stat` 情報を配列 *var* に格納する
`file mtime` *name*	*name* の修正時間を 10 進数で返す
`file owned` *name*	*name* が利用者のファイルならば 1 を、それ以外ならば 0 を返す
`file readable` *name*	*name* が利用者から読みだし可能ならば 1 を、それ以外ならば 0 を返す
`file readlink` *name*	シンボリックリンク *name* の内容を返す
`file rootname` *name*	*name* の拡張子以外の部分を返す

8.2.3 入出力

ファイルへの入出力 (読みだし、書き出し) を行うためには、まずファイルを開ける必要がある。

`open` *filename access permission*

access は読み書きに対する許可を表し、C の `fopen` と互換の形式と POSIX 互換のものを使うことが出来る。*permission* は、新たにファイルが作成される場合のパーミッションで、既定値は `0666` である。

表 8.2: file コマンド (続き)

コマンド	内容
`file size` *name*	*name* 中のバイト数を返す
`file stat` *name var*	*name* の `stat` 情報を配列 *var* に格納する
`file tail` *name*	*name* 中の最後の/以降の部分を返す
`file type` *name*	*name* の種類 (`file directory fifo link socket characterSpecial blockSpecial`) を返す
`file writable` *name*	*name* が利用者から書きだし可能ならば 1 を、それ以外ならば 0 を返す

表 8.3: `open` に対する`fopen` 互換のオプション

オプション	内容
`r`	読みだし。ファイルは存在していなくてはならない
`r+`	読み書き。ファイルは存在していなくてはならない
`w`	書き出し。ファイルが存在していない場合は、新たにファイルが作られる
`w+`	読み書き。ファイルが存在していない場合は、新たにファイルが作られる
`a`	追加書き込み。ファイルは存在していなくてはならない
`a`	読み書き。書き込む場合は追加書き込み。ファイルは存在していなくてはならない

ファイルが正しく開けられると、`stream` 名が返される。この `stream` を使って読み書きを行う。書き出しは

`puts` *stream string*

で行う。また、

`gets` *stream varname*

によって、*stream* から 1 行ずつ読み込まれ、*varname* に格納される。

`read` を使うと、`stream` の内容すべてを一度に読み出すことが出来る。例 7.1.1 では、一旦全てを読み出した後に、`split` で 1 行ずつに区切っている。

表 8.4: open に対する POSIX 互換のオプション

オプション	内容
RDONLY	読みだし専用
WRONLY	書き出し専用
RDWR	読み書き両用
APPEND	追加書き出し
CREAT	ファイルが存在していない場合、新たに作成する
EXCL	CREAT が指定されている場合、そのファイルは存在していてはならない
NOCTTY	端末デバイスを制御端末にすることを禁じる
NONBLOCK	open 手続きは途中で停止せず、必ず終了する
TRUNC	ファイルが存在している場合、先頭から書き込む

8.2.4 ディレクトリ操作

cd コマンドは、Tcl/Tk の中から直接実行することが可能で、作業ディレクトリを変更することが出来る。現在のディレクトリは pwd コマンドで得ることが出来る。

8.2.5 exit と pid

exit コマンドはプロセスを終了させる。exit の実行によって、UNIX のプロセス自身が終了することに注意する。

pid コマンドは、プロセス ID を返す。プロセス ID は、各プロセスで一意なので、作業用一時ファイルの名前等に利用する。

8.2.6 環境変数

UNIX が持つ環境変数は、大域的配列 env に格納されている。例えば、環境変数 PATH は env(PATH) に格納されている。

演習 8.1 設定されている環境変数とその値を listbox に表示するアプリケーションを作成しなさい。

環境変数を listbox に表示するには、例えばプログラム 8.2.1 のような副手続きを使うことが出来る。

Program 8.2.1 環境変数を listbox に格納するための副手続き例

```
proc list_env {w} {
    global env
    $w delete 0 end
    foreach i [array names env] {
        set line [format %s=%s $i $env($i)]
        $w insert end $line
    }
}
```

第9章　ダイアログとメッセージ

9.1　ダイアログとメッセージ

次の例は、ファイル名入力などを別ウィンドウを開いて行うものである。ファイル名が存在しない場合にエラーメッセージが別ウィンドウに表示される機能も付加して置く。

例 9.1　まず次の手続きでファイルを入手する。適当な作業用ディレクトリに移ったあと

```
cp /home/user/httpd/lectures/Is/TclTk/samples/chap9.tar.gz  .
```

を実行する。拡張子.gz は gzip 形式で圧縮されていることを示す。拡張子.tar は tar 形式で複数ファイルがまとめられていることを示している。

```
/usr/local/bin/tar zxvf chap9.tar.gz
```

を実行すると、gzip 圧縮が解除され、tar からファイルが読み出される。この場合、dialog.tcl と message.tcl の二つのファイルが出て来る。

プログラム 9.1.1 は、新しいウィンドウ.newtop を開き、ファイル名入力を得る手続きである。entry widget にファイル名を入れ、ok ボタンをクリックすることで、ファイル名を得ることが出来る。ok ボタンがクリックされると filename という変数が設定される。

以下の部分がファイル名入力部分にフォーカスを得て、それを解放し、ウィンドウを消す部分である。

```
focus $t.entry
grab $t
tkwait variable filename
grab release $t
destroy $t
```

entry widget がフォーカスを得ているので、自動的にカーソルが entry widget に移動する。tkwait コマンドで、filename が設定されるまで作業が待機され、その後、フォーカスを解放して、.newtop が destroy コマンドで破壊される。

Program 9.1.1 dialog.tcl

```
#!/usr/local/bin/wish -f
#
# メッセージ表示副手続きの読み込み
source message.tcl

# ファイル名を選択する副手続き
proc get_filename { } {
    upvar fname filename

# 新しいウィンドウの作成
    toplevel .newtop
    set t .newtop
    label $t.message -text "Enter File Name"
    entry $t.entry -textvariable fdum
    button $t.ok -text OK -command {set filename $fdum;set fdum ""}
    pack $t.message $t.entry $t.ok -side top

# ダイアログウィンドウへのフォーカスの取得と解放
    focus $t.entry
    grab $t
    tkwait variable filename
    grab release $t
    destroy $t
}
```

ファイル名が入力されると、checkfile 手続きで指定したファイルの存在が確認される。ファイルが無い場合には err_message 手続きによって、エラー表示のウィンドウが表示される (プログラム 9.1.2、9.1.3)。この例では、メッセージ表示用副手続きは別ファイル message.tcl に保存され、source コマンドで読み込まれている。

9.2 focus、grab、tkwait

キーボードやマウスからの入力をどの X アプリケーションに渡すかを入力フォーカスと呼ぶ。通常は、一番上に重なったウィンドウやマウスで選択されたウィンドウが保持している。アプリケーションの実行に必要な情報などを取得するためには、そのウィンドウに入力フォーカスを設定する必要がある。

focus パス名

で指定したパス名を持つ widget に入力フォーカスを設定される。

ファイル名入力などは、ファイル名が入力されるまでの間は、同じアプリケーションの他の動作を停止されておく必要がある。このような場合に使うのが grab コマンドである (例 9.1.1)。

Program 9.1.2 dialog.tcl の続き

```
# ファイルが存在するか否かの判定副手続き
proc checkfile {fn} {
    set path [pwd]
    set f [format %s/%s $path $fn]
    if {[file isfile $f]} {
        return $f
    }
    if {[file isdirectory $f]} {
        cd $f
    } else {
        set m "$f というファイルはありません"
        err_message $m
        set f none
    }
        return $f
}

proc open_file { } {
    upvar filename fname
    get_filename
    set fname [checkfile $fname]
}

set filename none
set path [pwd]
frame .command
label .pathname -textvariable path -anchor w
label .filename -textvariable filename  -anchor w
pack .command .pathname .filename  -side top -fill x
button .command.open -text OPEN -command {open_file;set path [pwd]}
button .command.quit -text QUIT -command exit
pack .command.open .command.quit -side left
```

X ウィンドウシステムは、イベントドリブン、つまりキーボード操作やマウス操作を契機にして次の動作に移って行く。従って、ダイアログのように入力を得る場合には特別な注意が必要である。入力待ちなどを明示的に行うのが tkwait コマンドである。例では、ある変数 *variable* が設定されるまで実行を待つ

tkwait variable *variable*

を使っている。たくさんの文字列を表示する必要がある場合には、あるウィンドウ *win* の表示が完了するのを待つ

tkwait visibility *win*

を使う必要がある。その他に、特定のウィンドウ *win* が破壊されるのを待つ

tkwait window *win*

Program 9.1.3 メッセージを表示する副手続き`message.tcl`

```
#!/usr/local/bin/wish -f
# エラーメッセージ表示
proc err_message {m} {
    toplevel .errtop -bg orange
    set t .errtop
    message $t.err -width 200 -text $m\
            -relief sunken -fg red -bg yellow
    button $t.ok -text OK -fg red -bg purple -command {set ok 1}
    pack $t.err $t.ok

    focus $t.ok
    grab $t
    tkwait variable ok
    grab release $t
    destroy $t
}
```

がある。

9.3 destroy コマンド

ダイアログなど、情報取得の為に開いたウィンドウを閉じるのが `destroy` コマンドである。

`destroy widget` 名

と使う。widget 名として”.” を使うと、アプリケーション全体を終了させることが出来る。

演習 9.1 例 9.1.1、9.1.2 及び**??**において、ファイルが存在する場合には、その `stat` 情報を別ウィンドウに表示するようにしなさい。`stat` 情報を表示したウィンドウは `ok` ボタンによって消去出来ること。

なお、ファイル`$f` の `stat` 情報を listbox widget `$w` に表示するには、例えば以下のようにする。

```
file stat $f a
$w delete 0 end
foreach i [array names a] {
    $w insert end "$i = $a($i)"
}
```

第10章　メニュー

10.1　メニュー

実際にアプリケーションを作る場合、動作や様々なオプションの選択などを行うメニューを作る必要がある。この章では、主メニューと副メニュー、およびそれぞれの動作を、例を通じて説明する。

例 10.1　次の例は、ディレクトリを選択し、オプションを指定してファイル一覧を表示するものである。オプションは、表示順序という排他的要素と、表示情報を選択出来るようにしている。プログラムは、5 つのファイルに分割されている。前章と同様に

`/home/user/httpd/lectures/Is/TclTk/samples/chap10.tar.gz`

から、ファイルをコピーして、`/usr/local/bin/tar` を利用して、適当なディレクトリに解凍する。

最初のプログラム (Program 10.1.1) は、全体の主部分に相当している。始めの部分で、アプリケーションのタイトルバーを設定している。

```
wm title . "File Information"
```

その後に、全体の表示の元になる frame を作成し、上部にメニューボタンを配置している。

menubutton widget はメニューの一番上のボタンに相当し、このボタンを押すことで、メニューが表示される。メニューの内容は

メニュー名 `add type` 指定

の形で追加されていく。

Program 10.1.1 Main.tcl

```
#!/usr/local/bin/wish -f
# タイトルバーの設定
wm title . "File Information"

#外部ソースの読み込み
source DefaultColor.tcl
source SelDir.tcl
source Options.tcl
source ListFile.tcl

#ファイル表示のデフォルトの順序
set order "-l"

frame .topframe ; pack .topframe

frame .topframe.fname -relief sunken
set menuframe\
        [label .topframe.menuframe -relief  raised -bg blue -fg white]
pack .topframe.menuframe .topframe.fname -side top -anchor w  -fill x
set flist [label .topframe.fname.filename] ; pack $flist

# メニュー
menubutton $menuframe.dir -text "ディレクトリ"\
        -menu $menuframe.dir.menu
menubutton $menuframe.options -text "オプション"\
        -menu $menuframe.options.menu
pack $menuframe.dir $menuframe.options -side left -fill x

# ディレクトリ表示のメニュー
SelDirMain $menuframe.dir
# 表示オプションの選択メニュー
MkOptionMenu $menuframe.options
# ファイル一覧表示用ウィジェットの定義
ListFile $flist
```

Program 10.1.2 DefaultColor.tcl

```
# デフォルト色の設定
# option add *ウィジェットクラス名.DatabaseName 色
option add *Listbox.background pink
option add *Scrollbar.background orange
option add *Scrollbar.activeBackground red
option add *Menubutton.background blue
option add *Menubutton.activeBackground SkyBlue
option add *Menu.background SkyBlue
option add *Menu.activeBackground yellow
option add *Button.background SkyBlue
option add *Button.foreground white
option add *Button.activeBackground IndianRed
```

widget に何も指定しない場合 (デフォルト) の、色やフォントなどは、X リソースという名で定義されている。X リソースは、リソースファイル、実行時のオプションとして指定する他に、プログラムの中で指定することも出来る。ここでは DefaultColor.tcl (Program 10.1.2) でデフォルトの色の指定を記述している。各 widget クラスごとに、背景色と選択された時の背景色を指定している。

Program 10.1.3 SelDir.tcl

```
#ディレクトリ選択メニュー
# accelerator 付き
proc SelDirMain {w} {
    set md [menu $w.menu -tearoff 0]
    $md add command -label "ファイル一覧表示"\
            -command {Show $flist.filelist} -accelerator Ctrl-s
    bind . <Control-s> {Show  $flist.filelist}
    $md add command -label "ディレクトリを開く"\
            -command SelDir -accelerator Ctrl-o
    bind . <Control-o> SelDir
    $md add command -label "終了"\
            -command exit -accelerator Ctrl-q
    bind . <Control-q> exit
}
##########################################################
# ディレクトリ一覧作成
proc list_dir {w} {
    set dlist [exec ls -a]
    $w delete 0 end
    foreach line $dlist {
        #ディレクトリのみを表示
        if {[file isdirectory $line]} {
            $w insert end ${line}
        }
    }
}
##########################################################
# ファイルを開ける
proc diropen {w y} {
    set i [$w nearest $y]
    set f [$w get $i]
    #ディレクトリの場合だけ、そこに移動
    if {[file isdirectory $f]} {
        cd $f
        list_dir $w
    }
}
```

Program 10.1.4 SelDir.tcl の続き

```
##########################################################
# ディレクトリ選択ウィンドウ
proc SelDir { } {
    set ok 0
    set w [toplevel .seldir]
    button $w.cancel -text CANCEL -relief raised\
            -command {set DirSelected 1}
    button $w.ok -text OK -relief raised\
            -command {set DirSelected 1;set ok 1}
    set wd [label $w.dirtable -relief sunken]
    pack $wd  -fill both -expand yes
    pack $w.cancel $w.ok -fill both -expand no

    #ディレクトリ名一覧
    scrollbar $wd.xscroll -command "$wd.dir xview"\
            -orient horizontal
    scrollbar $wd.yscroll -command "$wd.dirlist yview"\
            -orient vertical
    pack $wd.xscroll -side top -fill x -expand no
    pack $wd.yscroll -side left -fill y -expand no

    listbox $wd.dirlist -width 20 -height 10\
            -xscroll "$wd.xscroll set" \
            -yscroll "$wd.yscroll set" -selectmode single
    pack $wd.dirlist -side left -fill both -expand yes

    # ダブルクリックの登録
    bind $wd.dirlist <Double-Button-1>  {diropen %W %y}

    list_dir $wd.dirlist

    tkwait variable DirSelected
    destroy $w
}
```

ディレクトリ選択の部分は SelDir.tcl (Program 10.1.3 及び 10.1.4) で記述されている。これは、前節以前の例と同様に、ls コマンドの実行結果を listbox widget に格納し、表示している。tkwait コマンドを使って、OK ボタンや CANCEL ボタンが押されるとその widget が破壊されるように作っている。

Program 10.1.5 `ListFile.tcl`

```
# ファイル一覧を表示するための listbox の作成
proc ListFile {w} {
    scrollbar $w.xscroll -command "$w.filelist xview" -orient horizontal
    scrollbar $w.yscroll -command "$w.filelist yview" -orient vertical
    pack $w.xscroll -side top -fill x -expand no
    pack $w.yscroll -side left -fill y -expand no
    listbox $w.filelist -width 70 -height 20\
            -xscroll "$w.xscroll set" \
            -yscroll "$w.yscroll set" -selectmode single
    set wl $w.filelist
    pack $wl -side left -fill both -expand yes
}

# 指定されたオプションでファイル一覧を表示する
proc Show {wl} {
    global order MaxOption mo2v mo2vv

    # オプションの作成
    set optionlist $order
    for {set i 0} {$i < $MaxOption} {incr i 1} {
        if {$mo2v($i)} {
            append optionlist $mo2vv($i)
        }
    }

    # ファイル一覧表示
    set flist [exec ls $optionlist]

    $wl delete 0 end
    foreach line [split $flist \n] {
        $wl insert end $line
    }
}
```

ファイル一覧作成は `ListFile.tcl` (Program 10.1.5) で行われている。`Options.tcl` に記述されているメニューで選択された `ls` コマンドのオプションを指定して `ls` コマンドを実行し、listbox widget に表示している。

Program 10.1.6 Options.tcl

```
#オプションメニュー
proc MkOptionMenu {w} {
    global order MaxOption mo2v mo2vv
    set mo [menu $w.menu  -tearoff 0]

    # 表示順序を指定するメニュー作成
    $mo add cascade -label "表示順序" -menu $mo.order
    set moo [menu $mo.order -tearoff 0]
    $moo add command -label "ファイルの表示順序"
    $moo add separator
    set mool(0) "アルファベット順" ; set moolo(0) "-l"
    set mool(1) "アルファベット逆順" ; set moolo(1) "-lr"
    set mool(2) "変更時間順" ; set moolo(2) "-lt"
    set mool(3) "変更時間逆順" ; set moolo(3) "-ltr"
    set mool(4) "アクセス時間順" ; set moolo(4) "-lu"
    set mool(5) "アクセス時間逆順" ; set moolo(5) " -lur"
    for {set i 0} {$i < 6} {incr i 1} {
        $moo add radiobutton -label $mool($i)\
                -variable order -value $moolo($i)
    }

    # その他のオプションを指定するメニュー作成
    set MaxOption 3
    $mo add cascade -label "表示情報" -menu $mo.options
    set mo2 [menu $mo.options -tearoff 0]

    set mo2l(0) "完全時間情報"
    set mo2l(1) "ファイルシリアル番号"
    set mo2l(2) "ファイルブロック"
    set mo2vv(0) "T"
    set mo2vv(1) "i"
    set mo2vv(2) "s"

    for {set i 0} {$i < $MaxOption} {incr i 1} {
        $mo2 add checkbutton -label $mo2l($i) -variable mo2v($i)
    }
}
```

ls コマンドのオプション選択は Options.tcl (Program 10.1.6) で記述されている。

10.2 menu widget

menu widget はメニューを起動するボタン menubutton と、メニュー項目 menu から構成されている。

10.2.1 menubutton widget

menubutton widget は

```
menubutton パス名 -text ボタン名 -menu メニューパス名
```

という形で利用する。他の標準的オプションを併せて指定することも出来る。ボタン名で指定した文字列の付いたボタンをクリックすると、メニューパスで指定したメニューが表示される。メニューパスは親のパス名の下層に配置する menu widget の名前である。

10.2.2 menu wiget

個別のメニューを表示するには、まず menu widget を生成する。

```
menu メニューパス名 -tearoff 0
```

オプション-tearoff 0 を指定しない場合は、メニューの最初に区切りが表示される。このメニューは、上述のように、menubutton widget を押すことで表示される。

メニューに表示されるメニュー項目は、

```
メニューパス名 add type 指定
```

の形式で追加される。type は

```
command radiobutton checkbutton cascade separator
```

のいずれかである。

メニュー項目のなかで、基本となるのは command の指定である。

```
メニューパス名 add command -label ラベル -command コマンド名
```

普通の button widget と同様に、ボタンが押された場合の動作を指定する。コマンドを指定しない場合には、単にラベルで指定された文字列が表示される。

radiobutton は大域変数を共有して、可能な値のうち、そのいずれかを選択するのに利用する。

```
メニューパス名 add radiobutton -label ラベル\
        -variable 変数名 -value 値
```

Program 10.1.6 では、表示順序 order という排他的な、つまり、いずれかの値の一つを選択する必要のある変数に、アルファベット順や変更時間順などの指定をするのに用いている。

checkbutton はブール変数に値を指定するのに利用する。ブール変数は、0 と 1 の値を取る論理変数のことである。

メニューパス名 add checkbutton -label ラベル -variable 変数名

checkbutton を押すことで、変数の値が 0 と 1 の間で切り替わる。

メニューの中から更に副メニューを表示されるのが cascade である。

メニューパス名 add cascade -label ラベル -menu 副メニューパス名

副メニューの項目の作り方は、通常のメニューと同様である。

メニュー項目の間に線を引いて区切りを作りる場合には

メニューパス名 add separator

で行う。

10.2.3　accelerator

アプリケーションの操作に慣れると、いちいちメニューを開かずに操作するほうが楽になる。このように、メニューと同じ操作をキー操作で行うことをアクセレレイターと呼ぶ。メニュー項目を追加する際に

-accelerator キー操作文字列

を指定すると、メニューにキー操作が表示される。これだけでは、その動作を行うことは出来ないので、bind を使って、別途動作を指定する必要がある。

10.2.4　ポップアップメニュー

ポップアップメニューは、menubutton に結び付けられていないメニューである。menu widget を定義し、pack の代わりに

tk_popup メニュー名 x y

で起動する。x と y はメニューが表示される座標を表す。座標は、root window のものである。マウスがクリックされた場合、その場所にポップアップメニューを表示した場合は, イベントの起こった場所の root window での絶対座標を表す (%X,%Y) を使って

tk_popup メニュー名 %X %Y

とする。

10.3 ウィンドウマネージャーとの交信

X window システムでは、ウィンドウに表示されるタイトルバー、マウスによる移動やサイズ変更などの動作などは、window システムそのものではなく、ウィンドウマネージャーと呼ばれるプロセスが行っている。本学科の教育用システムでは、twm と呼ばれるプログラムを使っている。

Tcl/Tk にはウィンドウマネージャーと交信する wm というコマンドがある。wm は、大きさ、位置、装飾、アイコン、セッション状態、各種情報の取得や設定を行うことが出来る。ここでは、代表的な項目についてだけ説明する。

10.3.1 大きさ、位置、装飾

表 10.1: 大きさ、位置、装飾に対する wm 操作。?で囲まれた部分を設定すると、値の設定を、省略すると現在の値の取得を行うことが出来る。

コマンド	内容
`wm aspect` *win ?a b c d?*	*win* の縦横比を a/b から c/d の間に変更する。
`wm geometry` *win ?geometry?*	*win* の位置や大きさを取得あるいは設定する。
`wm grid` *win ?h h dx dy?*	グリッドの大きさを取得あるいは設定する。
`wm group` *win ?leader?*	*win* で指定したウィンドウの属する group の leader 名を取得あるいは設定する。
`wm maxsize` *win ?width height?*	*win* の最大サイズを取得あるいは設定する。
`wm minsize` *win ?width height?*	*win* の最小サイズを取得あるいは設定する。
`wm positionfrom` *win ?who?*	*win* の位置設定が `program` か `user` かを取得あるいは設定する。
`wm sizefrom` *win ?who?*	*win* のサイズ設定が `program` か `user` かを取得あるいは設定する。
`wm title` *win ?label?*	*win* のタイトルを取得あるいは設定する。

アプリケーションのウィンドウのタイトルを変更するには

```
wm title . "application name"
```

を行う。最後の名前の指定を行わなければ、現在設定されている名前を得ることが出来る。

ウィンドウの大きさは

`wm geometry` パス名 ?wx$h + ox + oy$?

で指定する。サイズは $x \times h$ で、原点を (ox, oy) で指定する。

10.3.2　アイコン

ウィンドウの状態は、通常の `normal`、アイコンになった `iconic`、更に表示されていない `withdraw` がある。それらの状態の制御や、アイコンになった場合の表示を取得または設定する。

表 10.2: アイコン対する wm 操作

コマンド	内容
`wm deiconify` *win*	*win* を開く
`wm iconbitmap` *win* ?*bitmap*?	*win* がアイコンになった時の bitmap を取得あるいは設定する。
`wm iconify` *win*	*win* をアイコンにする。
`wm iconmask` *win* ?*mask*?	*win* のアイコンマスクを取得あるいは設定する。
`wm iconname` *win* ?*mask*?	*win* のアイコン名を取得あるいは設定する。
`wm iconposition` *win* ?*x y*?	*win* のアイコン位置を取得あるいは設定する。
`wm iconwindow` *win* ?*window*?	*win* がアイコンになっている時に表示される *window* を取得あるいは設定する。
`wm state` *win*	*win* の状態 (`normal`、`iconic`、`withdraw`) を得る。
`wm withdraw` *win*	*win* を消去する。

10.3.3　セッション状態

セッションは、起動しているアプリケーションの状態を保持し、ウィンドウシステムが再起動されたときに、この情報でアプリケーションを再起動するものである。

セッションプロトコルを簡単に使うには、アプリケーションを起動したコマンドをそのまま使うのが良い。

```
wm command . "$argv0 $argv"
```

表 10.3: セッション状態対する wm 操作

コマンド	内容
`wm client` *win* ?*name*?	`WM_CLIENT_MACHINE` 属性にホスト名を登録する。
`wm command` *win* ?*command*?	`WM_COMMAND` 属性に起動時のコマンドを登録する。
`wm protocol` *win* ?*name*? ?*command*?	プロトコル要求 *name* を処理する *command* を登録する。

10.3.4 各種情報

表 10.4: 各種情報に対する wm 操作

コマンド	内容
`wm focusmodel` *win* ?*what*?	`active`、`passive` 等のフォーカスモデルを取得あるいは設定する。
`wm frame` *win*	*win* に親ウィンドウがある場合、その ID を取得する。
`wm group` *win* ?*leader*?	*win* の group の leader 名を取得あるいは設定する。
`wm overrideredirect` *win* ?*boolean*?	ウィンドウマネージャーが親ウィンドウを割り当てるのを抑制するフラグを取得あるいは設定する。
`wm transient` *win* ?*leader*?	*win* がその toplevel に対して、一時的に作成されたものであるかを取得あるいは設定する。

演習 10.1 第 8 章の例題のアプリケーションに対して、ファイル名一覧で、ファイル名の場所でマウス右ボタンをクリックすると、編集や `stat` 情報表

示のメニューが現れ、かつそれぞれの動作が行えるアプリケーションを作りなさい。

第11章　テキスト

11.1　簡易テキスト エディター

複数行に渡るテキストを表示し、編集することが出来るのが text widget である。従って text widget は、コマンドの実行結果を表示するのに用いたり、ファイルの編集に使うことが出来る。ここでは、簡単なテキストエディターを例として text widget の使い方を見ていく。

前章と同様に

```
/home/user/httpd/lectures/Is/TclTk/samples/chap11.tar.gz
```

から、ファイルをコピーして、/usr/local/bin/tar を利用して、適当なディレクトリに解凍する。

例 11.1

Program 11.1.1 は、簡易エディターの主部分であり、メニューボタンの定義と、テキストを表示する領域.menutop を定義している。メニューの内容は手続き MkFileMenu で、テキスト表示領域の定義は MkTextArea でそれぞれ行われている。

デフォルトの色の定義は、前章で使った Program 10.1.2 を使って行っている。

Program 11.1.2 では、FILE メニューのメニュー項目を定義すると同時に、OPEN が選択された場合に表示される別ウィンドウを定義している。別ウィンドウには、ディレクトリ一覧、現在選択されているファイル名、OK ボタン、CANCEL ボタンが定義される。ディレクトリ一覧の作成は MkFileList 手続きで行われる。

Program 11.1.3 及び 11.1.4 はファイル一覧を作成する部分である。ファイルが選択された場合は大域変数 filename に名前が設定される。大域変数 filename はファイル一覧表示部の entry widget(.selfiletop.f.entry) でも参照されていて、ファイル名入力部に表示される。このファイル名は、さらに OK ボタン (.selfiletop.cf.ok) を押すことによって、手続き openfile に渡される。

Program 11.1.5 がテキスト操作の主部分である。手続き MkTextArea でテキスト表示用 widget を定義した後、手続き openfile で表示を行っている。

Program 11.1.1 Main.tcl

```
#!/usr/local/bin/wish -f
#
# ファイルをテキストウィジェットに表示する
#
#外部ソースの読み込み
source DefaultColor.tcl
source MkFileMenu.tcl
source MkFileList.tcl
source Text.tcl

#大域変数
set FileSelected 0  ;#ファイルが選択されている
set filename ""   ;#選択されたファイル名
#
wm title . "Simple Editor"

label .menutop
label .texttop
pack .menutop .texttop -fill both -expand yes

menubutton .menutop.menu -text File -menu .menutop.menu.file
pack .menutop.menu -side left
menu .menutop.menu.file  -tearoff 0
MkFileMenu .menutop.menu.file
MkTextArea .texttop
```

ファイルを開けた後、read で内容を読み込み、text widget に表示している。listbox の場合と同様に、insert によって内容を入れているが、最後に余分な一行が入るので、強制的に最後の一行を削除している。

手続き savefile はメニューから SAVE が選ばれた場合に、テキストを保存するものである。このアプリケーションでは、同じファイルに上書きする機能しか用意されていない。ここでも、余分な一行を削除するために、全てではなく、最後の一行以外の部分を保存している。

Program 11.1.2 MkFileMenu.tcl

```
#File メニューの作成
proc MkFileMenu {w} {
    global filename
    $w add command -label OPEN -command SelFile
    $w add command -label "SAVE(overwrite)"\
            -command {savefile $filename}
    $w add command -label QUIT -command exit
}

#ファイル選択ウィンドウ
proc SelFile { } {
    global filename
    set t [toplevel .selfiletop]
    wm title $t "Select File"
    wm sizefrom $t program

    frame $t.list ; pack $t.list
    MkFileList $t.list
    frame $t.f
    frame $t.cf

    pack $t.f $t.cf -side top -expand no
    label $t.f.label -text "file name : "
    entry $t.f.entry -width 20 -textvariable filename
    pack $t.f.label $t.f.entry -side left -expand no

    button $t.cf.ok  -text OK -width 5\
            -command {openfile $filename ; set done 1}
    button $t.cf.quit -text CANCEL -width 5 -command {set done 1}
    pack $t.cf.ok $t.cf.quit -side left

    tkwait variable done
    destroy $t
}
```

11.2 text widget

11.2.1 インデクス

text widget は文章を表示するのに使います。message widget と異なり表示した文字列を編集することが出来ます。基本的なエディターとしての機能も有しています。

文字列の編集をするために、text widget は、文字の位置を行と各行の先頭からの文字数で指定します。行数は 1 から数えますが、各行の文字は 0 から数えます。例えば、1.0 は第 1 行目の先頭の文字を表します。end は単独に使われる場合は、ファイルの終端の文字、1.end は第 1 行目の最後の文字を表します。

Program 11.1.3 MkFileList.tcl

```
#ファイル一覧作成の主部分
proc MkFileList {w} {
    mklistbox $w
    bind $w.listbox <Double-Button-1> {selectfile %W %y}
    list_file $w.listbox
}

#ls コマンドの出力を listbox に格納する
proc list_file {w} {
    set flist [exec ls -a]
    $w delete 0 end
    foreach line $flist {
        if {[file isdirectory $line]} {
            $w insert end ${line}/
        } else {
            $w insert end ${line}
        }
    }
}

#ダブルクリックで選択された場合の動作
proc selectfile {w y} {
    global filename
    set i [$w nearest $y]
    set fname [$w get $i]
    if {[file isdirectory $fname]} {
        cd $fname
        list_file $w
    } else {
        if {[file isfile $fname]} {
            set filename $fname
        }
    }
}
```

11.2.2　テキストの挿入と削除

テキストの追加は insert 操作を使って行う。text widget $t に対して、位置 i に文字列 s を挿入するには

$t insert i s

とする。改行記号は自動では挿入されないので、文字"\n"を使う。

テキストの削除は delete で行う。ある場所 $from$ から to まで削除するには、以下のように指定する。

$t delete $from$ to

Program 11.1.5 の中で新たにファイルを開き表示する場合の操作を例として見て見よう。

Program 11.1.4 MkFileList.tcl の続き

```
#listbox を作る
proc mklistbox {w} {
    scrollbar $w.xscroll -command "$w.listbox xview"\
            -orient horizontal
    scrollbar $w.yscroll -command "$w.listbox yview"\
            -orient vertical
    pack $w.xscroll -side top -fill x -expand yes
    pack $w.yscroll -side left -fill y -expand yes
    listbox $w.listbox -width 30 -height 20\
        -xscroll "$w.xscroll set" -yscroll "$w.yscroll set"
    pack $w.listbox -side left -fill both -expand yes
}
```

表 11.1: text widget のインデクス

表現	内容
line.char	位置による指定
@x, y	座標による指定
current	現在のマウスカーソル下
end	最終文字の直後
insert	挿入カーソルの直後
mark	*mark* の直後
tag.first	*tag* で示される範囲の最初の文字
tag.last	*tag* で示される範囲の最後の文字
window	埋め込み *window* の位置

Program 11.2.1 Text.tcl の一部

```
# ファイル$f を開ける
set fr [open $f r]
set tx .texttop.text
# テキストウィジェットの内容を消去する
$tx delete 1.0 end
# テキストウィジェットにファイルの内容を挿入
$tx insert 1.0 [read $fr]
#delete last character (EOF) : TRICKY!
# 余分につく一行を削除
$tx delete "end -1 chars" end
```

11.2.3 インデクスの演算

Program 11.2.1 の最後で

```
$tx delete "end -1 chars" end
```

という、インデクスの演算"end -1 chars"が行われている。このように二重引用符"を使うことで、インデクスの演算を行うことが出来る。

表 11.2: text widget のインデクスの演算

表現	内容
+*count* chars	インデクスから *count* 文字後ろ
−*count* chars	インデクスから *count* 文字前
+*count* lines	インデクスから *count* 行後ろ (文字位置は保存)
−*count* lines	インデクスから *count* 行前 (文字位置は保存)
linestart	行頭
lineend	行末 (改行文字直前)
wordstart	単語の最初の文字
wordend	単語の最終文字の直後

11.2.4　マーク

text widget では、emacs と同様に、マークを設定して、範囲の選択を開始することが出来る。例えば

```
$t mark set foobar "insert wordstart"
```

は、widget t のカーソルのある単語の先頭に新しいマーク foobar を定義する。更に

```
$t delete foobar insert
```

と実行すると、マーク foobar から、挿入カーソルの現在位置までを削除することが出来る。

```
$t mark unset foobar
```

でマーク foobar を解除することが出来る。

11.2.5　キーバインド

text widget には、emacs のようなキー操作が定義されている。簡単な編集作業を行うには十分である。

演習 11.1　emacs と同様なカットアンドペーストのキーバインドを導入定義しなさい。つまり<Control-x><Control-w>でのカット、<Control-x><Meta-w>でのコピー、及び<Control-y>でのペースト機能を追加しなさい。

なお、<Control-space>は、anchor というマークを設定している。

表 11.3: text widget のキーバインド (マウス関係)

表現	内容
<Any-Key>	通常の文字の挿入する
<Button-1>	挿入位置を設定し、選択範囲を解除し、フォーカスを設定する
<Control-Button-1>	選択範囲を維持したまま挿入位置を設定する
<B1-Motion>	選択範囲を挿入位置から拡張する
<Double-Button-1>	マウスカーソルの下にある単語を選択する
<Triple-Button-1>	マウスカーソルの下にある行を選択する
<Shift-Button-1>	選択範囲の終りをマウスカーソルの近くにする
<Shift-B1-Motion>	選択範囲の調整を継続する
<Button-2>	選択範囲をペーストするか、あるいはスクロール基準点を設定する
<B2-Motion>	ウインドウをスクロールする

```
bind Text <Control-space> {
        %W mark set anchor insert
}
```

また、<Escape><w>に対する動作を定義する場合には、文字 w が表示されないように、その手続きの最後に break が必要になる。

表 11.4: text widget のキーバインド (左右の動作)

表現	内容
<Key-Left>	挿入カーソルを一文字左へ移動し、選択範囲を解除する
<Control-b>	挿入カーソルを一文字左へ移動し、選択範囲を解除する
<Shift-Left>	カーソルを左へ移動し、選択範囲を拡張する
<Control-Left>	カーソルを左へ 1 単語移動し、選択範囲を解除する
<Meta-b>	カーソルを左へ 1 単語移動し、選択範囲を解除する
<Control-Shift-Left>	カーソルを左へ 1 単語移動し、選択範囲を拡張する
<Key-Right>	挿入カーソルを一文字右へ移動し、選択範囲を解除する
<Control-f>	挿入カーソルを一文字右へ移動し、選択範囲を解除する
<Shift-Right>	カーソルを右へ移動し、選択範囲を拡張する
<Control-Right>	カーソルを右へ 1 単語移動し、選択範囲を解除する
<Meta-f>	カーソルを右へ 1 単語移動し、選択範囲を解除する

Program 11.1.5 `Text.tcl`

```
proc MkTextArea {w} {
    scrollbar $w.xscroll -command "$w.text xview"\
            -orient horizontal
    scrollbar $w.yscroll -command "$w.text yview"\
            -orient vertical
    pack $w.xscroll -side top -fill x -expand no
    pack $w.yscroll -side left -fill y -expand no

    text $w.text -width 80 -height 20\
            -xscroll "$w.xscroll set" -yscroll "$w.yscroll set"
    pack $w.text -side left -fill both -expand yes
}

proc openfile {f} {
    global FileSelected 0
    #ファイルの内容を表示する
    if {[file isfile $f]} {
        set fr [open $f r]
        set tx .texttop.text
        $tx delete 1.0 end
        $tx insert 1.0 [read $fr]
        #delete last character (EOF) : TRICKY!
        $tx delete "end -1 chars" end
        set FileSelected 1
    }
}

#表示内容をファイルに上書きする
proc savefile {f} {
    global FileSelected
    if {$FileSelected} {
        set tx .texttop.text
        #delete last character (EOF) : TRICKY!
        set content [$tx get 1.0 "end -1 chars"]
        set fw [open $f w]
        puts $fw $content
    }
}
```

表 11.5: text widget のキーバインド (上下の動作)

表現	内容
<Key-Up>	カーソルを 1 行上に移動し、選択範囲を解除する
<Control-p>	カーソルを 1 行上に移動し、選択範囲を解除する
<Shift-Up>	カーソルを 1 行上に移動し、選択範囲を拡張する
<Control-Up>	カーソルを段落単位で上に移動する
<Control-Shift-Up>	カーソルを段落単位で上に移動し、選択範囲を拡張する
<Key-Down>	カーソルを 1 行下に移動し、選択範囲を解除する
<Control-n>	カーソルを 1 行下に移動し、選択範囲を解除する
<Shift-Down>	カーソルを 1 行下に移動し、選択範囲を拡張する
<Control-Down>	カーソルを段落単位で下に移動する
<Control-Shift-Down>	カーソルを段落単位で下に移動し、選択範囲を拡張する

表 11.6: text widget のキーバインド (大きな移動)

表現	内容
<Next>	次の画面に移動し、選択範囲を解除する
<Shift-Next>	次の画面に移動し、選択範囲を拡張する
<Prior>	前の画面に移動し、選択範囲を解除する
<Shift-Prior>	前の画面に移動し、選択範囲を拡張する
<Home>	カーソルを行頭に移動し、選択範囲を解除する
<Control-a>	カーソルを行頭に移動し、選択範囲を解除する
<Shift-Home>	カーソルを行頭に移動し、選択範囲を拡張する
<End>	カーソルを行末に移動し、選択範囲を解除する
<Control-e>	カーソルを行末に移動し、選択範囲を解除する
<Shift-End>	カーソルを行末に移動し、選択範囲を拡張する
<Control-Home>	カーソルをテキストの最初に移動し、選択範囲を解除する
<Meta-less>	カーソルをテキストの最初に移動し、選択範囲を解除する
<Control-End>	カーソルをテキストの最後に移動し、選択範囲を解除する
<Meta-greater>	カーソルをテキストの最後に移動し、選択範囲を解除する

表 11.7: text widget のキーバインド (その他)

表現	内容
<Select>	カーソル位置を選択範囲の基準点に設定する
<Control-space>	カーソル位置を選択範囲の基準点に設定する
<Shift-Select>	選択範囲をカーソル位置に移動する
<Control-Shift-space>	選択範囲をカーソル位置に移動する
<Control-slash>	テキストウィジェットの全てのテキストを選択する
<Control-backslash>	選択範囲を解除する
<Delete>	選択範囲が存在すればそれを削除する。選択範囲が無ければカーソルの右の文字を削除する
<Backspace>	選択範囲が存在すればそれを削除する。選択範囲が無ければカーソルの左の文字を削除する
<Control-h>	選択範囲が存在すればそれを削除する。選択範囲が無ければカーソルの左の文字を削除する
<Control-d>	カーソルの右の文字を削除する
<Meta-d>	カーソルの右の単語を削除する
<Control-k>	カーソルから行末までを削除する。カーソルが行末にある場合は、改行文字を削除する
<Control-o>	カーソル位置は同じまま、改行文字を挿入する
<Control-w>	カーソルの左の単語を削除する
<Contorl-x>	選択範囲の文字を全て削除する
<COntorl-t>	文字を交換する

第12章　キャンバス

12.1　簡易プロッタ

Program 12.1.1 simple_plotter/Main.tcl

```
#!/usr/local/bin/wish -f
#
#シンプルプロッタ
#簡単な２次元データの描画
#
#Main.tcl

source Canvas.tcl

set bw [frame .buttonframe]
set cw [frame .canvasframe]
pack $bw $cw

#コマンドボタンの設定
button $bw.quit -text QUIT -command exit
button $bw.start -text START -command {start $cw}
pack $bw.quit $bw.start -side left

#キャンバスの作成
createcanvas $cw
```

これまで見て来た例では、部品を集めてアプリケーションを構成してきた。widget を使ったプログラミングの最後に、作図を行う方法を扱う。

作図を行うのに使うことが出来るのが canvas widget である。canvas widget は、線、円弧、楕円、多角形、文字列、ビットマップ、イメージなどを表示することが出来る。ここでは、まずデータを簡単に図として表示する簡易プロッタを例として、canvas widget の使い方見て行く。

前章と同様に

/home/user/httpd/lectures/Is/TclTk/samples/chap12.tar.gz

から、ファイルをコピーして、/usr/local/bin/tar を利用して、適当なディレクトリに解凍する。二つのディレクトリ simple_plotter と simple_objects が生成される。simple_plotter が簡易プロッタである。

Program 12.1.2 simple_plotter/Canvas.tcl

```
#キャンバス
#

#縮尺
set fx 10.
#キャンバスは左上隅が原点のため、y 方向は負の縮尺
set fy -80.
#原点
set yo 100.
set xo 20.

#キャンバスを作る
proc createcanvas {w} {
    scrollbar $w.xscroll\
            -command "$w.canvas xview" -orient horizontal
    scrollbar $w.yscroll\
            -command "$w.canvas yview" -orient vertical
    pack $w.xscroll -side top -fill x -expand no
    pack $w.yscroll -side left -fill y -expand no

    canvas $w.canvas -width 200 -height 200\
            -xscrollcommand "$w.xscroll set"\
            -yscrollcommand "$w.yscroll set"
    pack $w.canvas -side left -fill both -expand yes
}

#描画スタート
proc start {w} {
    global xo yo fx fy

    set fr [open "data" r]
    set points {}
    # 残りのデータ
    foreach line [split [read $fr] \n] {
        # 行から二つのデータを切り出す
        regexp { *([-\.0-9]+) *([-\.0-9]+) *$} $line match x y
        # データ列を追加する
        append points [list [expr $fx*$x+$xo] [expr $fy*$y+$yo]] " "
    }
    eval {$w.canvas create line} $points -joinstyle round
}
```

例 12.1

ディレクトリ simple_plotter の下にある Main.tcl (Program 12.1.1) が、簡易プロッタの主プログラム部分であり、コマンドボタンが定義されている。二つの frame widget が定義され、それぞれにコマンドボタンと作図部分が定義される。

Program 12.1.2 が canvas widget を定義し、その中に作図する主部分である。手続き createcanvas によって、スクロールバーの付いた canvas widget が生成される。スクロールバーの作り方は、今まで使ったものと同様である。

Start ボタンを押すことで、手続き start が実行される。データファイル data からデータが読み込まれ、作図すべき座標に変換された後にリスト points に追加される。

最後に

```
eval {$w.canvas create line} $points -joinstyle round
```

によって、曲線としてデータが描画される。eval を使うことによって、リストに保存された座標列がコマンドの一部として評価される。

ここで注意しなくてはいけないのは、座標系である。通常は左下を原点として、右方向に x の正方向、上方向に y の正方向をとるが、canvas widget では (そして、一般に X ウィンドウでは) 画面の左上が原点であり、下方向に y 座標の正方向が取られることである。例では、原点の座標を (xo, yo) と表し、縮尺を (fx, fy) で表しているが、$fy < 0$ となっている。

12.1.1 様々なオブジェクト

例 12.2 次の例 (ディレクトリ simple_objects) は、canvas widget で使える様々な図形要素を表示し、マウスで移動させるものである。

Program 12.1.3 simple_objects/Main.tcl

```
#!/usr/local/bin/wish -f
#
#簡単なオブジェクト例
#
#Main.tcl

source Canvas.tcl

set bw [frame .buttonframe]
set cw [frame .canvasframe]
pack $bw $cw

#コマンドボタンの設定
button $bw.quit -text QUIT -command exit
pack $bw.quit -side left

#キャンバスの作成
createcanvas $cw
```

Program 12.1.4 simple_objects/Canvas.tcl

```
#キャンバス
#

#キャンバスを作る
proc createcanvas {w} {
    scrollbar $w.xscroll\
            -command "$w.canvas xview" -orient horizontal
    scrollbar $w.yscroll\
            -command "$w.canvas yview" -orient vertical
    pack $w.xscroll -side top -fill x -expand no
    pack $w.yscroll -side left -fill y -expand no

    canvas $w.canvas -width 500 -height 500\
            -xscrollcommand "$w.xscroll set"\
            -yscrollcommand "$w.yscroll set"
    pack $w.canvas -side left -fill both -expand yes
    bind $w.canvas <Button-1> {CanvasMark %x %y %W}
    bind $w.canvas <B1-Motion> {CanvasDrag %x %y %W}

    draw_simple_objects $w.canvas
}

# マウスでオブジェクトを選択
proc CanvasMark {x y w} {
    global canvas
    set canvas($w,obj) [$w find closest $x $y]
    set canvas($w,x) $x
    set canvas($w,y) $y
}

# マウスでオブジェクトを移動
proc CanvasDrag {x y w} {
    global canvas
    set dx [expr $x - $canvas($w,x)]
    set dy [expr $y - $canvas($w,y)]
    $w move $canvas($w,obj) $dx $dy
    set canvas($w,x) $x
    set canvas($w,y) $y
}

source Canvas2.tcl
```

Program 12.1.3 はフレームとボタンを定義する主部分である。

Program 12.1.4 の中の、createcanvas 手続きは、スクロールバーの付いた canvas widget を生成する部分である。更に、マウスクリックとマウスドラッグ時の動作が定義されている。

マウスボタン 1 をクリックすると、手続き CanvasMark が呼び出される。マウスに最も近くにあるオブジェクトが canvas($w,obj) に登録され、それぞれの (x, y) 座標が canvas($w,x) と canvas($w,y) に登録される。

マウスボタン 1 を押したまま移動すると、手続き CanvasDrag が呼び出される。CanvasMark で登録されたオブジェクトがマウスの位置へ移動される。

最後の Program 12.1.5 は、canvas widget が持つ様々な図形要素例の描画の部分である。手続き draw_simple_objects では、線、円弧、楕円、多角形、文字列、ビットマップという基本図形要素が描画されている。更に、drawdata を使ってデータをプロットしている。

Program 12.1.5 simple_objects/Canvas2.tcl

```
#様々なオブジェクトのサンプルを描く
proc draw_simple_objects {w} {
    # 線を引く
    # widget 名 create line 開始座標 終了座標
    $w create line 100 100 400 400
    # widget 名 create arc 左上座標 右下座標
    #                 -start 開始角度 -extent 終了角度
    $w create arc 100 200 200 300 -start 0 -extent 120\
            -style pieslice -fill red
    # widget 名 create 座標 -bitmap ビットマップ名
    $w create bitmap 200 50 -bitmap error -foreground blue
    # widget 名 create oval 左上座標 右下座標
    #                       -fill 内部の色 -width 境界の幅
    $w create oval 350 300 400 450 -fill orange -width 5
    # widget 名 create polygon 座標列 -fill 内部の色
    $w create polygon 200 100 150 200 150 300 200 400 50 450\
            -fill purple
    # widget 名 create text -text 文字列
    $w create text 450 450 -text "これはテスト" -fill green
    drawdata $w
}

# データをプロットする
proc drawdata {w} {
    set fx 10.
    set fy -80.
    set yo 100.
    set xo 20.

    set fr [open "data" r]
    set points {}
    # 残りのデータ
    foreach line [split [read $fr] \n] {
        # 行から二つのデータを切り出す
        regexp { *([-\.0-9]+) *([-\.0-9]+) *$} $line match x y
        # データ列を追加する
        append points [list [expr $fx*$x+$xo] [expr $fy*$y+$yo]] " "
    }
    eval {$w create line} $points -joinstyle round
}
```

索 引