Język skryptowy Tcl i pakiet okienkowy Tk

Witold Paluszyński
witold@ict.pwr.wroc.pl
http://sequoia.ict.pwr.wroc.pl/~witold/

Niniejszy dokument zawiera materiały do wykładu na temat tworzenia prostych aplikacji okienkowych w w języku Tcl/Tk. Jest on udostępniony pod warunkiem wykorzystania wyłącznie do własnych, prywatnych potrzeb i może być kopiowany wyłącznie w całości, razem z niniejszą stroną tytułową.

Elementy języka Tcl

Tcl (czyt. tikl, ang. *Tool Command Language*) jest językiem służącym do pisania skryptów, czyli krótkich, interpretowanych (tzn. wykonywanych w postaci źródłowej, bez kompilacji) programów, będących ciągami poleceń oddzielonych średnikami lub znakami nowej linii

polecenie Tcl jest ciągiem słów, z których pierwsze jest nazwą polecenia a pozostałe argumentmi:

```
puts stdout "Prosty napis"
set x 5
puts stdout "x ma teraz wartosc $x"
```

ze względu na mały rozmiar i interpretacyjne wykonanie, skrypty są często tworzone eksperymentalnie metodą prób i błędów

Elementy języka Tcl (cd.)

wykonanie polecenia Tcl polega na obliczeniu wartości poszczególnych argumentów i, po zastąpieniu argumentów ich wartościami, spowodowanie wykonania polecenia z tak skonstruowanym zestawem argumentów

polecenie Tcl	skutek
set x 5	zmienna x ma wartość "5"
set y x	zmienna y ma wartość "x"
set z x+y	zmienna z ma wartość "x+y"
set z x + y	UWAGA: zła liczba argumentów polecenia set
set z " x + y "	teraz znów dobrze
set z $x*\$y	zmienna z ma wartość "5*$y"

Tcl: obliczanie wartości argumentów

1. zastąpienie wyrażeń typu $zmienna wartościami zmiennych (zmiennych się nie deklaruje, i domyślnym typem danych jest napis znakowy)

2. rekurencyjne wywołanie interpretera `tclsh` dla wykonania poleceń zagnieżdżonych w nawiasach [ ], zastąpienie takich poleceń wynikiem

- nawiasy { } blokują wyliczanie zarówno [ ] jak i $, cudzysłowy " " blokują tylko [ ]

- backslash \ odbiera specjalne znaczenie wszelkim znakom (sobie też)

- obliczanie wartości argumentów przeprowadzane jest tylko jeden raz!!

Tcl: obliczanie wartości wyrażeń — expr

polecenie Tcl	skutek
set u 3	u ma wartość 3
set v "$u"	v ma wartość 3
set w {$u}	w ma wartość $u
set x [expr $u]	x ma wartość 3
set y {$u + 2}	y ma wartość "$u + 2"
set z [expr $y*4]	jaką wartość ma z?
set x {expr $y+$z}	x ma wartość "expr $y+$z"
set y [expr $x+3]	*syntax error in expression*
set y [expr "$z*7"]	y ma wartość "77"
set x {expr $y+$z}	x ma wartość "expr $y+$z"
set z [eval $x]	z ma wartość "88"

wyrażenia obliczane przez `expr`: `< <= == != ...`, również: `sin($x)`

eval powoduje dodatkowe obliczenie wartości swojego argumentu

Tcl: polecenia złożone

polecenia złożone:

```
if { $x > 0 } {
 ...
}
while { $x <= 100 } {
 ...
}
foreach dz {Pon Wto Sro Czw Pia Sob Nie} {
 ...
}
for { set i 0 } { $i < 100 } { set i [expr $i+1] } {
 ...
}
```

polecenia sprawdzające warunki logiczne dokonują ponownego obliczenia wartości wyrażeń, jeden dodatkowy raz (podobnie jak eval)

Tcl: dalsze przykłady obliczania wartości wyrażeń

polecenie Tcl	skutek
set x 10	x ma wartość 10
set y {$x}	y ma wartość $x
if {$x==10} {puts stdout jest}	"jest"
if {$y==10} {puts stdout jest}	nic
if "$y==10" {puts stdout jest}	"jest"

Dygresja: zmiennych Tcl nie deklaruje się, ale można sprawdzić czy dana zmienna istnieje, tzn. została wcześniej użyta:

```
if ![info exists x] {
  set x 0
} else {
  incr x
}
```

Tcl: procedury

procedury stanowią konstrukcje pozwalające związać schematy obliczeń z parametrami formalnymi:

```
proc potega {x n} {
set wynik 1
while {$n > 0} {
  set wynik [expr $wynik * $x]
  set n [expr $n - 1]
  }
return $wynik
}
```

przykłady wywołań:

```
% potega 2 16
65536
% potega 2 31
-2147483648
% potega 2 30
0
% potega 1.5 2
2.25
% potega 2 1.5
4
% potega 2 -1
1
```


Tcl: procedury — przykłady

przykład procedury zdefiniowanej rekurencyjnie:

```
proc fib {n} {
if {$n<=2} {expr 1} else \
{expr [eval fib [expr $n-1]] + [eval fib [expr $n-2]]}}

% fib 40
102334155
```

składnia zapisu poleceń Tcl wymaga by były one zapisane w jednej linijce, jednak znak nowej linii może znaleźć się jako zwykły znak w wyrażeniach (przy pomocy {} lub \), zatem alternatywna wersja zapisu tej procedury:

```
proc fib {n} {
  if {$n<=2} {
    expr 1
  } else {
    expr [eval fib [expr $n-1]] \
       + [eval fib [expr $n-2]]
  }
}
```

następujący zapis nie jest poprawny:

```
proc fib {n} {
  if {$n<=2} {expr 1}
  else {expr [eval fib [expr $n-1]] +
             [eval fib [expr $n-2]]}}
```


Tcl: procedury — zmienne globalne

Polecenie `global` użyte w procedurze deklaruje zmienną/zmienne jako globalne, to znaczy powoduje, że występująca w procedurze zmienna odwołuje się do zmiennej o tej samej nazwie w środowisku globalnym.

Istnieją jeszcze inne formy rozszerzenia zakresu zmiennej poza lokalny zakres bieżącej procedury. Polecenie `variable` wiąże zmienną lokalną z globalną przestrzenią nazw (która musiała być wcześniej utworzona), a polecenie `upvar` wiąże zmienną lokalną z inną zmienną o tej samej nazwie występującą wcześniej na stosie wywołania procedur.

Tcl: operacje na napisach znakowych

polecenie Tcl	zwraca wartość
`string first wzor "To jest wzor"`	8
`string last o "To jest wzor"`	10
`string range "To jest wzor" 3 6`	"jest"
`string compare alicja barbara`	-1
`string compare zuzanna joanna`	1
`string compare "jola" {jola}`	0
`string length "To jest wzor"`	12
`string toupper kasia`	"KASIA"
`string trim "     badziewie     "`	"badziewie"
`string replace "Ala ma kota" 7 10 "kanarka"`	"Ala ma kanarka"
`string match {[a-z]*} Mariola`	0
`string match {[A-Z]*} Lucyna`	1

ostatnie dwa przykłady powyżej ilustrują proste dopasowanie wzorców z wykorzystaniem znaków specjalnych `[]*?` (1 oznacza sukces dopasowania)

Tcl: dopasowanie wzorców z wyrażeniami regularnymi

polecenie `regexp` realizuje dopasowanie wzorców z użyciem pełniejszego języka wyrażeń regularnych, zdefiniowanego następująco:

znak	znaczenie w wyrażeniu regularnym
. (kropka)	oznacza jeden dowolny znak
^ i $	oznaczają odpowiednio początek i koniec napisu
[]	jeden z zawartych w nich znaków, lub przedziałów
?	oznacza opcjonalne wzięcie wyrażenia poprzedzającego
* i +	oznaczają powtórzenia: 0-lub więcej, i 1-lub więcej razy
\	przed dowolnym znakiem odbiera mu znaczenie specjalne
inny znak	oznacza ten sam znak

Przykłady:

polecenie Tcl	skutek
regexp {[+−][0-9]+\.[0-9]*} 10.33	zwraca 0
regexp {[+−][0-9]+\.[0-9]*} +.33	zwraca 0
regexp {[+−][0-9]+\.[0-9]*} −0.33	zwraca 1

Tcl: operacje wejścia/wyjścia

czytanie w pętli zawartości pliku tekstowego i wyświetlanie jej na wyjściu:

```
proc naekran {plik} {
  set plikw [open $plik r]
  for {set x 1} {$x >= 0} { } {
    set x [gets $plikw wiersz]
    puts stdout $wiersz
  }
  close $plikw
}
```

przydatną własnością powyższego mechanizmu otwierania plików jest, że można podać zamiast nazwy rzeczywistego pliku polecenie Unix'a (poprzedzone znakiem |), co spowoduje utworzenie podprocesu wykonującego dane polecenie, i jednokierunkową komunikację z nim

np. można wyświetlić listę plików w katalogu bieżącym wywołując: `naekran "|ls -1"`

Tcl: wywoływanie programów zewnętrznych

Aby wywołać zewnętrzny program Unixa (albo własny użytkownika) ze ścieżki względnej lub bezwzględnej, można użyć polecenia `exec`:

```
set out [exec finger]
puts $out
```

Polecenie `exec` pozwala również na zastosowanie skierowania strumieni wejścia i wyjścia, np.:

```
exec ps -ef > /dev/tty
```

W ogólnym przypadku argumentami polecenia `exec` może być **potok** poleceń, czyli ciąg poleceń połączonych znakiem | (podobnie jak w przypadku „zwykłych" interpreterów poleceń systemu Unix: Bourne shell, C-shell, bash). Na przykład:

```
exec ps -fu $(LOGNAME) | grep sh
```

Co ciekawe, polecenie to działa również pod systemem Windows, choć ma tam mniejszy sens niż pod systemem Unix.

Okienkowy interpreter poleceń pakietu Tk

Pakiet Tk jest nakładką na język Tcl wprowadzający szereg elementów graficznych, zwanych widżetami (ang. *window gadget*). Elementy te są wyświetlane w specjalnym oknie interpretera Tk, który nazywa się `wish`.

Skrypt Tk może utworzyć pewną liczbę widżetów, dokonywać na nich różnych manipulacji, a także wykonywać różne obliczenia na poziomie języka Tcl, jak również korzystając z poleceń systemu operacyjnego i dowolnych programów.

Możliwości pakietu Tk w połączeniu z językiem Tcl są olbrzymie. Tutaj przedstawione zostaną na kilku przykładach możliwości użycia kilku najprzydatniejszych widżetów ilustrujące tworzenie prostych, niewielkich aplikacji okienkowych.

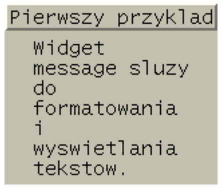
Na uwagę zasługuje również przenośność takich aplikacji. Ponieważ istnieją interpretery Tcl i Tk na wielu systemach komputerowych i okienkowych, skrypty Tcl i Tk można przenosić i uruchamiać na nich bez modyfikacji ani rekompilacji.

Tworzenie i wyświetlanie widżetów

Widżety Tk tworzy się poleceniami stanowiącymi nazwę ogólną danego widżeta (klasy widżetów). Pierwszym argumentem jest nazwa konkretnego tworzonego widżeta, zawsze zaczynająca się od kropki, a dalszymi argumentami jego parametry.

Utworzone widżety można wyświetlić w oknie poleceniem pack.

```
#!/bin/sh
# Copyright (C) 2003 Witold Paluszynski \
exec wish "$0" "$@"

label .naglowek -text "Pierwszy przyklad" -relief raised
message .komunikat -text \
  "Widzet message sluzy do formatowania i wyswietlania tekstow."

pack .naglowek
pack .komunikat
```

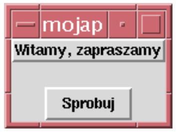


Wykonywanie operacji na widżetach

Stworzenie widżeta powoduje również utworzenie polecenia Tk, będącego nazwą własną widżeta, pozwalającego przesyłać widżetom różne żądania. Na przykład żądanie *configure* (pierwszy argument polecenia widżeta) nakazuje widżetowi zmienić pewne jego parametry.

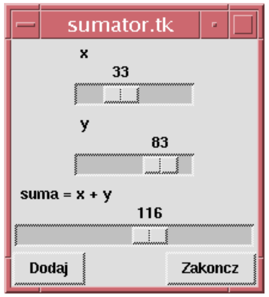
Poniższy przykład ilustruje wysłanie żądania rekonfiguracji napisu po kliknięciu przez użytkownika na obraz widżeta button.

```
#!/bin/sh
# Copyright (C) 2003 Witold Paluszynski \
exec wish "$0" "$@"

label .naglowek -text "Witamy, zapraszamy" -relief raised
label .odpowiedz
button .aktyw -text "Sprobuj" -command \
  {.odpowiedz configure -text "Serdecznie pozdrawiamy"}

pack .naglowek
pack .odpowiedz
pack .aktyw
```



Użycie konstrukcji języka Tcl

```
#!/bin/sh
# Copyright (C) 2003 Witold Paluszynski \
exec wish "$0" "$@"

scale .x -to 100 -length 100 -label "x" -orient horizontal
scale .y -to 100 -length 100 -label "y" -orient horizontal
scale .z -to 200 -length 200 -label "suma = x + y" -orient horizontal
button .suma -text "Dodaj" -command oblicz
button .koniec -text "Zakoncz" -command exit

proc oblicz {} { .z set [expr "[.x get] + [.y get]"] }

pack .x; pack .y; pack .z
pack .suma -side left
pack .koniec -side right
```



Zestaw przydatnych widżetów

label - jednowierszowy komunikat tekstowy
message - wielowierszowy komunikat tekstowy
scale - suwak z wartościami liczbowymi
scrollbar - suwak do przesuwania innych widżetów
entry - jednowierszowe pole do wprowadzania tekstu
listbox - wielowierszowe pole do wyświetlania tekstu
text - pole tekstowe o wielu możliwościach

widżetom wydajemy polecenia dla nich właściwe

```
nazwa-widgeta polecenie [arg] ...
```

np.

```
.napis configure -text "nowa tresc napisu"
```

Pełną dokumentację widżetów, ich działania i parametrów, można uzyskać w systemie komputerowym poleceniem `man`. Opisy widżetów Tk znajdują się w sekcji *n*, a więc przykładowa postać polecenia: `man -s n label`.

Geometria wyświetlania widżetów: `pack`

Najprostszym mechanizmem rozmieszczenia widżetów w oknie interpretera Tk jest polecenie `pack`, które rozmieszcza widżety „po kolei", przydzielając im prostokątne obszary wypełniające okno interpretera:

- określenie strony okna (left/righ/top/bottom):

```
pack .suma -side left
pack .koniec -side right
```

- określenie miejsca w obszarze widżeta (n/e/s/w/ne/nw/se/sw/center):

```
pack .a -side top -anchor w
```

- korzystanie z hierarchii widżetów i konstrukcji `frame`:

```
frame .input
label .input.l -text "Plik zrodlowy:"
entry .input.e -width 60 -textvariable open
button .input.b -text "Wybor" -command ...
pack .input.l .input.e .input.b -side left
pack .input -side top
```

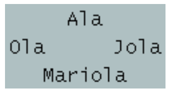

Geometria wyświetlania widżetów: `grid`

Polecenie `grid` służy do rozmieszczania widżetów oknie interpretera układając je w wirtualną tablicę składającą sie z rzędów i kolumn.

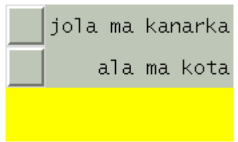
- Zwykła kolejność wystąpienia widżetów w poleceniu `grid` determinuje porządek kolumn w danym rzędzie.
- Kolejne polecenia `grid` wyświetlają kolejne rzędy.
- Polecenie `grid` ma opcję `-sticky` z argumentami `nesw` pozwalającymi przesuwać widżet w ramach danej pozycji w tablicy.
- Można również umieszczać widżety w określonych pozycjach tablicy przy użyciu opcji `row` i `col`.
- Widżet może zajmować więcej niż jeden rząd tablicy przez zastosowanie opcji `-rowspan` i/lub więcej niż jedną kolumnę przez zastosowanie opcji `-columnspan`.
- Polecenie `grid rowconfigure grid columnconfigure` dają możliwość współpracowania z mechanizmem zmiany rozmiaru okna interpretera Tk.
- Polecenie `grid remove` pozwala chwilowo usunąć widżet z ekranu bez kasowania jego ustawień.

Geometria wyświetlania widżetów: przykłady

```
label .l1 -text "Ala"
label .l2 -text "Ola"
label .l3 -text "Jola"
label .l4 -text "Mariola"
pack .l1 -side top
pack .l4 -side bottom
pack .l2 -side left
pack .l3 -side right
```




```
label .l1 -text "ala ma kota"
button .ala
label .l2 -text "jola ma kanarka"
button .jola
grid .ala .l1
grid .jola .l2
frame .f -background yellow
grid .f
grid .l1 -sticky e
grid .f -sticky ew
.f configure -height 40
grid .f -columnspan 2
grid .l1 -row 1 -column 1
grid .l2 -row 0 -column 1
```



Przykład: przeglądarka do plików z widżetami text i tk_getOpenFile

```
#!/bin/sh
# Copyright (C) 2003 Witold Paluszynski \
exec wish "$0" "$@"


# tworzymy zestaw guzikow operacji, kazdy wywoluje jakas procedure
frame .operacje

# guzik .operacje.wybierz wywoluje tk_getOpenFile do wyboru pliku mysza
set types {
    {{pliki tekstowe}   {.txt}    TEXT}
    {{pliki Pascalowe}  {.p .pas} }
    {{obrazki PGM}      {.pgm}    }
    {{dowolne pliki}    *         }}
button .operacje.wybierz -text "Wybierz" -command {
  .operacje.potwierdz configure -text ""
  set nazwapliku [tk_getOpenFile -filetypes $types] }
```



```
# guzik .operacje.wyswietl wywoluje wlasna procedure wyswietlplik
button .operacje.wyswietl -text "Wyswietl" -command {
  .operacje.potwierdz configure -text ""
  wyswietlplik $nazwapliku
  .operacje.potwierdz configure -text " Fertig! " }

proc wyswietlplik {plik} {
  .oknopliku.tekst delete 0.0 end
  set plikw [open $plik r]
  while {[eof $plikw] == 0} {
    gets $plikw wiersz
    .oknopliku.tekst insert end $wiersz
    .oknopliku.tekst insert end "\n"
  }
  close $plikw
}

# guzik .operacje.wyczysc czysci okno pliku i teksty widgetow
button .operacje.wyczysc -text "Wyczysc" -command {
  .operacje.potwierdz configure -text ""
  .oknopliku.tekst delete 0.0 end
  set nazwapliku ""
  .operacje.potwierdz configure -text " Yessir! " }
```

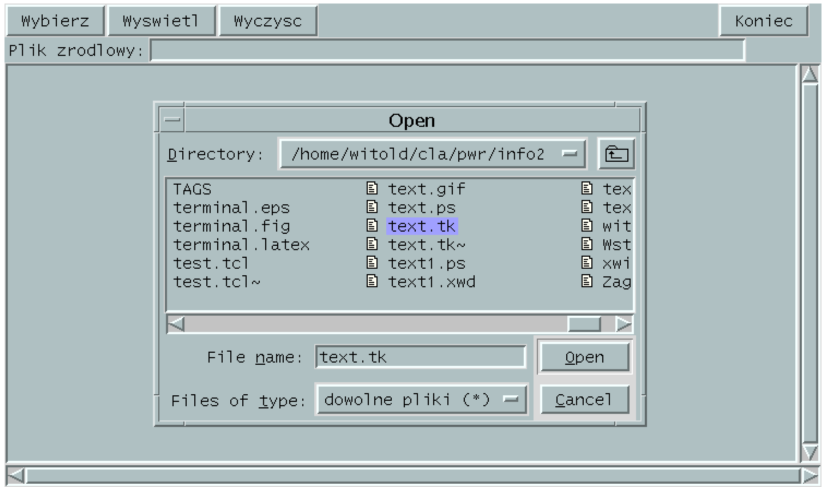


```
# pole tekstowe do potwierdzen i guzik do zakonczenia pracy
label .operacje.potwierdz -width 40
button .operacje.koniec -text "Koniec" -command {exit}


# standardowy komplet do wyboru pliku: etykietka i widget entry
# do wpisywania nazwy pliku (ustawia zmienna nazwapliku)
frame .plik
label .plik.napis -text "Plik zrodlowy:"
entry .plik.nazwa -width 60 -textvariable nazwapliku


# widget text z dwoma suwakami, pionowy suwak stanowi komplet
# z oknem tekstowym dla latwiejszego zarzadzania geometria
frame .oknopliku
text .oknopliku.tekst -width 80 -height 24 -wrap none \
  -yscrollcommand ".oknopliku.ysuwak set" \
  -xscrollcommand ".xsuwak set"
scrollbar .oknopliku.ysuwak -command ".oknopliku.tekst yview"

# poziomy suwak dla dlugich linii tekstu, bedzie na samym dole
scrollbar .xsuwak -orient horiz -command ".oknopliku.tekst xview"
```



```
# pelna specyfikacja geometrii; uzywamy dwupoziomowej hierarchii
# widgetow dla rozmieszczenia widgetow tam gdzie chcemy

# na gorze pasek guzikow operacji
pack .operacje.wybierz -side left
pack .operacje.wyswietl -side left
pack .operacje.wyczysc -side left
pack .operacje.potwierdz -side left
pack .operacje.koniec -side right
pack .operacje -side top -anchor w

# ponizej pole wyboru nazwy pliku
pack .plik.napis .plik.nazwa -side left
pack .plik -side top -anchor w

# dalej okno wyswietlanego tekstu i suwak pionowy
pack .oknopliku.tekst -side left
pack .oknopliku.ysuwak -side right -fill y
pack .oknopliku -side top

# na koncu suwak poziomy
pack .xsuwak -side top -fill x
```

okno po wywołaniu procedury `tk_getOpenFile` przyciskiem Wybierz



okno z wyświetloną zawartością pliku, przesuniętą suwakiem pionowym