Tcl - wstęp

1. Tcl = Tool Command Language; daleki potomek skryptów unix-owych (ksh, csh, bash)
2. jest to język "interpretowany" = nie trzeba kompilować (w rzyczywistości: kompilator "on-the-fly", bytecode, na razie nie ma JIT!)
3. jest to język 100% przenośny: unix/linux, ms windows, palmtop-y(?)
4. wielka łatwość "rozszerzania" Tcl-a! rozszerzenia Tcl można nazywać "komponentami" *(spełniają definicję komponentu w inż. oprogramowania! - są tworzone niezależnie i mogą być bez ograniczeń łączone)*
5. najpopularniejsze rozszerzenie to Tk - służy do tworzenia GUI; istnieje wielka "biblioteka" gotowych rozszerzeń
6. Tcl jest zasadniczo językiem proceduralnym, ale istnieją rozszerzenia wprowadzające OOP: itcl, XOTcl, Snit oraz inne
7. do czego Tcl się dobrze nadaje: budowanie GUI, operacje bazodanowe, generowanie HTML, sklejanie komponentów, unowocześnianie aplikacji spadkowych (z tekstowym interf.), przetwarzanie XML, impl. prot. sieciowych (tclhttpd), ...
8. strona domowa Tcl: www.tcl.tk, wiki.tcl.tk, activestate.com/products/activetcl
9. interpreter tcl: tclsh, wish (ten drugi używa GUI)

Tcl - filozofia

Kod: wszystko jest komendą

```
label .l -text "A ku ku !"; # przykład komendy
```

Dane: wszystko jest stringiem

```
"wlazł kotek na płotek"
{Jan Kowalski {Poznan, ul ...}}
   # string, który równocześnie jest listą
```

1. Tcl zawiera zbiór komend wbudowanych, np "set" (set x 123)
2. także instrukcje sterujące (np "if", "while") SĄ komendami
3. rozszerzenie Tcl dodaje nowe komendy
   (a) *rozszerzenia binarne* (.dll, .so) dodają komendy wbudowane
   (b) *rozszerzena skryptowe* dodają procedury, które zachowują sie w 100% jak komendy; procedury definiuje się komendą "proc"
4. zalecana architektura aplikacji:
   (a) komponenty, np rozszerzenia binarne Tcl pisane w C
   (b) Tcl będący "klejem" łączącym komponenty
   (c) niektóre komponenty warto programować w Tcl
   (d) zazwyczaj GUI tworzy się w TclTk
5. istnieją narzędzia ułatwiające tworzenie rozszerzeń binarnych (SWIG, CriTcl, ...)

6. rozszerzenia Tcl można podzielić na "językowe" i "dziedzinowe" ...
   językowe - np XOTcl (OOP w tcl), Metakit (hierarchiczno/relacyjna lokalna b.d.);
   dziedzinowe - np VTK (grafika 3D, biblioteka 700 klas C++)

Tcl - język

1. komendy (label .l -text 123)
2. zmienne (set x 123), tablice asocjacyjne (set wmi(d123) {Jan Kowalski})
3. substytucja zmiennych ($zmienna, $wmi($i)) i substytucja komend ([kod])
4. stringi, cytowanie ("napis", {napis}, \znak)
5. listy ({1 2 3}, {1 {qqq www} 2 3}), komendy l*, foreach
6. instrukcje sterujące (if, while, foreach, ...)
7. ważniejsze komendy wbudowane (string, l*, expr, info, array, regexp, ...???)
8. procedury; parametry procedur; komendy: proc, global, return, upvar, uplevel

SOP121/ Temat E
http://atos.amu.edu.pl/%7Emhanckow/sop121/sop121e.htm

Tcl - procedury

1. Do definiowania procedur służy komenda "proc"
2. procedury Tcl posiadają parametry (przekazywane przez wartość) i zmienne lokalne; zmienne globalne muszą być deklarowane komendą "global"
3. jest możliwy dostęp do zmiennych dowolnej ramki stosu (upvar); można wykonywać kod w kontekście dowolnej ramki stosu (uplevel)

```
proc mojaProc {a b ccc} {
    # parametry przekazywane "przez wartość"
  set x "$a $b $ccc"; # x jest zmienna lokalną
  return $x
}

# jak przekazywac parametr "przez zmienną/ref" ???
proc zmodyfikujZmianna {zmienna wartosc} {
  upvar $zmienna z
    # z to alias do zmiennej o nazwie $zmienna w niższej ramce stosu
  set z $wartosc
}
# .. i sposób użycia:
set qqq 123
zmodyfikujZmianna qqq 321
set qqq
```

```
## dostęp do zmiennych globalnych (global)

set x 123
set y 111
  # to są zmienne globalne

proc mojaProc {} {
  global x
  set x 321; # dotyczy zmiennej globalnej x
  set ::y 222; # inna metoda dostępu do zmiennej globalnej
  set z 333; # z jest zmienną lokalną
}
```

```
# zastosowania uplevel ...
#

## przypisanie "arytmetyczne"
proc let {zm args} {
  uplevel set $zm \[expr $args\]
}
# sposób użycia:
let x 1+ $x + 3

## wygodna pętla
proc iterate {zm liIter kod} {
  upvar $zm i; for {set i 0} {$i<$liIter} {incr i} { uplevel $kod }
}
# sposób użycia:
iterate i 10 { ...kod... }
```

```
# inny przykład procedury ...

## poszukiwanie na liscie slowa zgodnego ze "wzor"-em
proc znajdz {lista wzor} {
  set w {}
  foreach x $lista { if {[string match $wzor $x]} {lappend w $x} }
  return $w
}
# sposób użycia:
znajdz [info commands] {vtk*Object}
  # dygresja: komenda "info" pozwala na introspekcję/odbicie
```

```
# gdzie jest "dynamizm" w Tcl ???
#

## w zmiennych ...

set ind 123
set wmi(d$ind) {Jan Kowalski}
puts $wmi(d$ind)

set wmi_d$ind {Jan Kowalski}
puts [set wmi_d$ind]; # tu MUSZĘ użyc set a nie ${wmi_d$ind}

## w kodzie ...

set cialo "return $ind"
proc zwrocIndeks {} $cialo

set q {set x 123}
eval $q

## w komendzie expr ...

set x 1+2
expr $x*3; # ile to zwróci ???
expr {$x*3}; # to jest BŁĄD !!!
```

```
set y 123
expr {$y+1}; # to jest ok - i w dodatku działa ZNACZNIE szybciej!!!
   # im mniej dynamizmu tym szybciej kod działa !!!
```

Tcl - struktury danych

1. EIAS (= Everything is a string) zatem nie ma tablic, struktur/C, wsk
   ... zatem jak reprezentować struktury danych ???

2. struktura kodowana w nazwie zmiennych:

```
//C:
   tab[i].a[j].b.c=123;
#Tcl:
   set tab,$i,a,$j,b,c 123
```

3. zagnieżdżone listy:

```
{Jan Kowalski {Poznan ul ???} 123456}
```

4. **dict** czyli słownik; polecenie "dict" jest dostępne w Tcl8.5 lub jako rozszerz. bin.; słownik to lista o parzystej liczbie elementów: klucz1 wartość1 klucz2 wartość2 ...

```
lappend auto_path {E:\TEMP\tcl\dict8.5.1}
package require dict; # ładujemy pakiet dict (powoduje załadowanie odp roz. bin.)
  # dodajemy elementy do słownika eee ...
set eee {}
dict set eee id1000 {}
dict set eee id1001 {}
  #% id1000 {} id1001 {}
dict set eee id1000 imie Michal
dict set eee id1000 nazwisko Hanckowiak
```

```
dict set eee id1001 imie Jan
dict set eee id1001 nazwisko Kowalski
  #% id1000 {imie Michal nazwisko Hanckowiak} id1001 {imie Jan nazwisko Kowalski}
  # zanim sie uzyje zagniezdzonego set, musi istniec element "poziomu 0"
dict set eee id1000 imie "[dict get $eee id1000 imie] Jerzy"
  #% id1000 {imie {Michal Jerzy} nazwisko Hanckowiak} id1001 ...
```

5. **Metakit** - hierarchiczno relacyjna baza danych; baza metakit-owa to tabele z danymi; pole tabeli może być tabelą (zagnieżdżone tabele!); wielka łatwość manipulowania danymi; duża szybkość działania (dane są przechowywane kolumnami a nie wierszami!); jednoplikowe rozszerz. bin.;

```
package require Mk4tcl; # ładujemy pakiet z metakit-em
  #% 2.4.9.5
mk::file open db
  # tworzymy pamieciowa baze danych o nazwie "db"
mk::view layout db.tab1 {imie nazwisko wiek:I}
  # tworzymy tabelę o nazwie tab1 z polami imie, nazwisko, wiek
  # ostatnie pole jest typu I (Int), pozostałe to dowolne stringi
for {set i 0} {$i<100} {incr i} {
  mk::row append db.tab1 imie qqq$i nazwisko www$i wiek [expr $i*10]
}; # napełniamy tabelę treścią ...
```

```
# jak czytać/modyfikować dane w bazie
mk::get db.tab1!15
  #% imie qqq15 nazwisko www15 wiek 150
mk::set db.tab1!15 imie Michal wiek 321
  #% db.tab1!15
mk::get db.tab1!15
  #% imie Michal nazwisko www15 wiek 321
mk::view size db.tab1
  #% 100

# sposoby przebiegania po wierszach
set wiersze [mk::select db.tab1]
foreach w $wiersze {
  _puts [mk::get db.tab1!$w]
}
mk::loop c db.tab1 {
  _puts [mk::get $c]
}

# mozna zadawać jedynie b. proste pytania/ kwerendy:
# np mozna wybrac z tabeli wiersze w ktorych pewne pole ma określoną zawartość ...
foreach x [mk::select db.tab1 -glob nazwisko "Hanc*"] {
  _puts [mk::get db.tab1!$x imie nazwisko]
    # odczytujemy wynik zapytania
}
```

```
# Uwaga:
#   istnieje rozszerzenie Vlerq/ratcl pozwalające zadawać bardziej wyrafinowane
#   zapytania do danych metakit ... (algebra relacji jak w SQL !)
#     patrz też: http://www.equi4.com
```

Tcl - programowanie GUI czyli Tk

Tk to ToolKit do tworzenia GUI (okienek itp) ...

1. widget - "komponent wizualny", kontrolka
2. drzewo widgetów - ścieżka do widgetu .a.b.c; struktura logiczna widgetów (widgety są "kontenerami" tj mogą zawierać inne widgety)
3. umieszczanie widgetów - zarządca geometryczny, np komenda "pack"
4. opcje widgetów i komendy widgetowe - widgety można konfigurować; komenda widgetowa nazywa się tak jak widget, a jej podkomendy cget i configure służą do konfigurowania widgetu

```
label .lab -text "123"
.lab config -text "321"; # uruchamiamy komendę widgetową .lab
```

5. obsługa zdarzeń (ang. events) - opcja "-command" i komenda "bind"

```
bind .a.b.c <Button-1> { ...kod... }
```

SOP121/ Temat E

http://atos.amu.edu.pl/%7Emhanckow/sop121/sop121e.htm

Tcl - obsługa plików/sieci czyli programy sterowane zdarzeniami

Istnieje spór co jest lepszym rozwiązaniem: "aplikacje sterowane zdarzeniami" czy "apliacje wielowątkowe" ...

1. w Tcl początkowo zachęcano do sterowania zdarzeniami (John Ousterhout), obecnie istnieje rozszerzene Threads do tworzenia wątków
2. GUI oczywiście zawsze JEST sterowane zdarzeniami
3. jakie typy zdarzeń występują w Tcl (Tk, fileevent, threads, after); każdy interp Tcl posiada kolejkę zdarzeń; zdarzenia z tej kolejki są obsługiwane gdy ...
4. *model wątków Tcl*: każdy wątek ma własny logiczny interpreter (nie może być 2 wątków w jednym interpie!); wątki są mocno odseparowane, w zasadzie mogą sobie jedynie wysyłać skrypty do wykonania, które są umieszczane w kolejce zdarzeń i sekwencyjnie wykonywane

Operacje na plikach i połączeniach sieciowych:

1. **kanał** (ang. channel) odpowiednik deskryptora pliku/ gniazdka BSD; otrzymujemy go po otwarciu pliku, po uzyskaniu/zaakceptowaniu połączenia Tcl, ...
2. komendy do obslugi plików: open, close, puts, gets, read, fconfigure, eof, ...

```
set f [open moj_plik.txt w]; # tworzymy plik, tworzymy kanał, w jak w fopen() !
```

```
puts $f "A ku ku 1"; # zapis linii zakończonej znakiem końca linni
puts $f "A ku ku 2"
puts $f "A ku ku 3"
close $f

set f [open moj_plik.txt r]; # otwieramy plik
gets $f; # czytamy po jednej linii
  #% A ku ku 3
  #% A ku ku 2
  #% A ku ku 1
close $f

set f [open moj_plik.txt r]
read $f; # odczyt całości pliku za jednym zamachem
  #% A ku ku 1
A ku ku 2
A ku ku 3
close $f
```

3. instalowanie proc obsługi zdarzń plikowych (readable, writeable)

```
fileevent $f readable "procedura_obsługi $f"
  # instalujemy procedure Tcl która zostanie asynchronicznie wywołana
  # gdy pojawi się możliwość (NIEblokującego) czytania z $f
```

4. obsługa połączeń sieciowych TCP:

```
## po stronie klienta ...
set sock [socket adres_IP_serwera port_serwera]; # podłączamy się do serwera
puts $sock "tra la la"; flush $sock

## po stronie serwera ...
socket -server proc_obsługi_klienta port; # instalujemy proc obsługi klientów

proc proc_obsługi_klienta {s addr port} {
  set linia [gets $s]
  puts $s "dostałem od ciebie $linia"
  close $s; # na tym konczymy konwersjację z klientem

    # UWAGA: ta procedura może instalować procedurę obsługi
    # zdarzenia plikowego "readable" dla $s !!!
       # w ten sposób możemy obsługiwać równocześnie wielu klientów
       # bez używania wątków !!!
    #fileevent $s readable "proc_obsługi $s"
}

## !!! pokazać zastosowanie midd_5.tcl !!!
```

Tcl - zastosowanie Tcl w http/html

1. AOLserver, OpenACS, NaviServer - ...
2. **websh** - może działać jako moduł apache, lub jako CGI
3. **tclhttpd** - tclowy serwer www (narzędzia podobne do PHP/JSP, servlet-ów)

```
# przykład skryptu websh:

#load ./libwebsh3.6.0b4_8.4.so

web::cookiecontext qqq

web::command mojaKomenda {
  web::put "<p>A ku ku !</p>"
  web::put "<p><a href=[web::cmdurl ""]>link do mojaKomenda</a></p>"
}

web::command default {
  qqq::init "mojecookie"
  qqq::cset x [expr [qqq::cget x]+1]
  qqq::cset y 123
  qqq::commit

  web::put "<p>x = [qqq::cget x]; y = [qqq::cget y]</p>"
```

```
  web::put "<p>A ku ku !</p>"

  web::put "<p>request = [web::request -names]</p>"
  web::put "<p>param = [web::param -names]</p>"
  web::put "<p>wartosci parametrow:<br><pre>"
  foreach x [web::param -names] {
    web::put "   $x = [web::param $x]\n"
  }
  web::put "</pre></p>"

  web::put "<p><pre>[exec ps -O ppid]</pre></p>"

  web::put "<p><a href=[web::cmdurl mojaKomenda]>link do mojaKomenda</a></p>"
}

web::dispatch
```

Tcl - rozszerzenia binarne w C i C++

1. Tcl API czyli "biblioteka C Tcl-a"; jest to biblioteka libtcl8.4.so/tcl84.dll dostarczana wraz z dystrybucją Tcl-a; funkcje tej biblioteki mają nazwy Tcl_*; biblioteka ta pozwala m.in. dodawać nowe komendy wbudowane (czyli tworzyć rozszerzenia binarne)

2. Dane tcl-owe są przechowywane w strukturach (języka C) **Tcl_Obj**, czyli w tzw "obiektach tcl"

3. Tcl_Obj ma **typ wewnętrzny** oraz dwie reprezentacje: stringową i wewnętrzną; reprezentacja wew. zależy od typu wew. danej; reprezentacje te są uaktualniane w sposób leniwy; używana jest ta reprezentacja, która jest potrzebna

4. w Tcl są następujące typy wew.: string, int, long, float, list, bytecode, cmdName, dict, ...; rozszerz.bin. mogą dodawać nowe typy wew.!

5. zarządzanie pamięcią obiektów tcl: posiadają one licznik odwołań (refcount); refcout powinien być równy liczbie wsk wskazujących na obiekt; gdy zmniejszamy licznik i osiągnie on 0 to obiekt jest usuwany;

```
set x 123; set y $x; # zmienne x i y uzywaje jednego obiektu tcl! (refcount==2)
  # dopiery gdy zmodyfikujemy y tworzony jest drugi obiekt tcl
  # (zasada "copy-on-write")
```

6. nowe **komendy wbudowane** rejestruje się podając nazwę komendy oraz funkcję C obsługującą tą komendę; funkcja ta przypomina funkcję main(int argc, char* argv[]) języka C, w podobny sposób otrzymuje parametry komendy!

Nowa komenda wbudowaną ...

```
// to jest funkcja obslugująca komendę wbudowaną "sumaLiczb"
int SumaLiczbCmd(
  ClientData clientData, Tcl_Interp* interp, int objc, Tcl_Obj* const objv[]
) {
  if (objc != 3) {
    Tcl_SetResult(interp, "musza byc 2 parametry typu int!!!", TCL_STATIC);
    return TCL_ERROR;
  }

  // Tcl -> C++; przekształcamy dane Tcl na dane C/C++
  int arg1, arg2;
  if( Tcl_GetIntFromObj(interp, objv[1], &arg1)==TCL_ERROR ) return TCL_ERROR;
  if( Tcl_GetIntFromObj(interp, objv[2], &arg2)==TCL_ERROR ) return TCL_ERROR;

  int wynik; wynik= arg1+arg2; // wykonujemy operację w języku C

  // C++ -> Tcl
  Tcl_Obj* w= Tcl_NewIntObj(wynik);

  Tcl_SetObjResult(interp, w);
  return TCL_OK;
}
```

```
// funkcja instalujaca nową komendę wbudowaną Tcl-a o nazwie sumaLiczb
// ta funkcja musi sie nazywac: nazwabiblioteki_Init
extern "C" int Rozszerzbin_Init(Tcl_Interp *interp) {
    // tu extern "C" jest niezbedne; inne funkcjie nie wymagaja extern "C"
    // pod Win32 jest tez niezbedne "_export" !!!
  Tcl_CreateObjCommand(interp,"sumaLiczb",SumaLiczbCmd,NULL,NULL);
  return TCL_OK;
}

// Uwaga: powyzsze funkcje (+ #include <tcl.h> na początku)
// powinny się znaleźć w pliku rozszerzbin.cpp
// i powinny zostać skompilowane: g++ -shared rozszerzbin.cpp -o rozszerzbin.so

# sposob użycia tej komendy z poziomu Tcl:
load ./rozrzerzbin.so
sumaLiczb 111 222
#% 333
```

Komenda wbudowana obsługująca klasę Obiekt języka C++ ...

```
/* - komenda "nowyObiekt nazwa" tworzy obiekt C++ i zwiazana z nim komende "nazwa"
   - rodzaje interf. tcl-owego dla C++:
       (I) tworzenie obiektow + wywolywanie metod
      (II) --"-- + wymiana metod polimorf. (???)
*/
class Obiekt {
public:
  Obiekt() { printf("Obiekt %p - konstruktor\n", this); licznik=0; }
  ~Obiekt() { printf("Obiekt %p - destruktor\n", this); }
  long podajLiczbe() {
    return licznik++;
  }
  std::string& podajString() {
    wynik="A ku ku!!"; return wynik;
  }
private:
  std::string wynik;
  long licznik;
};
```

```
int ObiektCmd(
  ClientData clientData, Tcl_Interp* interp, int objc, Tcl_Obj* const objv[]
) {
   if (objc != 2) {
     interp->result = "zla liczba parametrow"; return TCL_ERROR;
       // to przestarzala metoda def resultset !!!
   }
   Obiekt *o= (Obiekt*)clientData;
   char *par1= Tcl_GetString(objv[1]);

   if (strcmp(par1,"podajString")==0) {
     const char *s= o->podajString().c_str();
     Tcl_Obj* string= Tcl_NewStringObj(s, -1);
     Tcl_SetObjResult(interp, string);

   } else if (strcmp(par1,"podajLiczbe")==0) {
     long l= o->podajLiczbe();
     Tcl_Obj* liczba= Tcl_NewLongObj(l);
     Tcl_SetObjResult(interp, liczba);

   } else {
     interp->result = "wywolanie: obiekt metoda"; return TCL_ERROR;
   }
   return TCL_OK;
}
```

```
void ObiektDeleteProc(ClientData clientData)
{
  Obiekt *o= (Obiekt*)clientData; delete o;
}
int NowyObiektCmd(
  ClientData clientData, Tcl_Interp* interp, int objc, Tcl_Obj* const objv[]
) {
   if (objc != 2) {
     interp->result = "musisz podac parametr - nazwe nowego obiektu!";
     return TCL_ERROR; }
   Obiekt *o= new Obiekt();
   Tcl_CreateObjCommand(interp,Tcl_GetString(objv[1]),ObiektCmd,o,ObiektDeleteProc);
   return TCL_OK;
}
extern "C" int Rozszerzbin_Init(Tcl_Interp *interp) {
  Tcl_CreateObjCommand(interp,"nowyObiekt",NowyObiektCmd,NULL,NULL);
  return TCL_OK;
}

# sposob uzycia z poziomu Tcl:
load ./rozszerzbin.so
nowyObiekt o1; # utworzenie obiektu
o1 podajLiczbe
o1 podajString
rename o1 {}; # usunięcie obiektu
```

Najważniejsze funkcje biblioteki C Tcl-a:

1. Tcl_Get*FromObj();
   próba zinterpretowania Tcl_Obj jako obiektu danego typu wew oraz wyciągnięcie tej wew wartości

2. Tcl_New*Obj();
   tworzenie nowych Tcl_Obj z określonym typem wew.

3. Tcl_SetObjResult();
   definiowanie wyniku komendy wbudowanej

4. Tcl_CreateObjCommand();
   definiowanie komendy wbudowanej tj powiązanie nazwy komendy z funkcją C obsługującą tą komendę

pokazać przykład photo !!!

Tcl - bytecode; szybkość działania kodu tcl-owego

Bajtkod Tcl-wy jest językiem opartym na stosie; składa się z rozkazów pobierających dane ze stosu i zostawiających na nim wyniki; elementy stosu to obiekty tcl-owe!

Skąd się bierze różnica szybkości działania alternatywnych (ale równoważnych) skryptów?

```
set x 123; set x [expr $x+2]
0: push1 x
2: push1 123
4: storeScalarStk
5: pop
6: push1 x
8: push1 x
10: loadScalarStk
11: push1 +2
13: concat1 +2
15: exprStk
16: storeScalarStk
17: done
```

exprStk to specjalny rozkaz do uruchomiania fun C obsługującej komendę expr; normalnie robi się to kładąc na stosie nazwę komendy + parametry oraz wywołując invokeStk

```
set x 123; set x [expr {$x+2}]
0: push1 x
2: push1 123
4: storeScalarStk
5: pop
6: push1 x
8: push1 x
10: loadScalarStk
11: push1 2
13: add
14: storeScalarStk
15: done
```

wywołanie komendy "expr" zostało przetłumaczone na rozkazy bajtkodu!!! (podobnie z innymi komendami "tcl core")

```
eval {set x 123}
0: push1 eval
2: push1 {set x 123}
4: invokeStk1 2
6: done
```

to jest najwolniejsza metoda wykonania komendy (nazwę komendy i jej paramtery umieszcza się na stosie + invokeStk)

```
if 1 {set x 123}
0: push1 x
2: push1 123
4: storeScalarStk
5: done
```

komenda "if" została przetłumaczona na rozkazy bajtkodu!!!

```
set y {set x 123}; if 1 $y
0: push1 y
2: push1 {set x 123}
4: storeScalarStk
5: pop
6: push1 if
8: push1 1
10: push1 y
12: loadScalarStk
13: invokeStk1 3
15: done
```

dynamiczna wersja "if" nie może być tak dobrze przetłumaczona na bajtkod!

Kiedy wykonanie kodu Tcl-owego może być BARDZO nieefektywne?

```
# kombinator K
# - pozwala uniknac niepotrzebnego kopiowania danych!!
#   jesli refcount==1 to komendy takie jak lreplace NIE tworza kopii ob. tcl
# - sensowne zastosowanie kombinatora K
#   set x [lreplace $x 10 20 A B C]; # jest tworzona kopia $x
#   set x [lreplace [K2 x] 10 20 A B C]; # nie jest tworzona !!!

proc K {x y} {return $x}
proc K2 x {upvar $x x1; set x2 $x1; set x1 {}; return $x2}
  # [K2 x] <=> [K x [set x {}]]

set x {}; iterate i 10000 {lappend x $i}; # tworzymy bardzo dluga liste

time {
  set x [lreplace $x 10 10 A]
} 1000
  #% 4565.318 microseconds per iteration
time {
  set x [lreplace [K2 x] 10 10 A]
} 1000
  #% 22.527 microseconds per iteration; skąd taka wielka różnica ???
```

Tcl - instalowanie oprogramowania

1. pakiety Tcl - instalowanie/ używanie/ tworzenie pakietów

```
# rozpakowujemy pakiet "mojpakiet" (np zip) w katalogu {C:\qqq}
# pakiet zawiera skrypty i roz.bin. ORAZ plik pkgIndex.tcl
lappend auto_path {C:\qqq}; # dodajemy ścieżkę do pakietu
package require mojpakiet; # ładujemy pakiet
# można zarządać konkretnej wersji pakietu (JEST obsługa wersji pakietów)
package require -exact mojpakiet 1.5

# jak się tworzy pakiety??? najprostsza metoda:
# 1. umieścic pliki w pewnym katalogu
# 2. wszystkie skrypty muszą zawierać komendę "package provide nazwa wersja"
#    roz.bin. muszą zrobić coś podobnego!
# 3. wejśc do katalogu spod konsoli i wykonać komendę "pkg_mkIndex"
```

2. Starkits - aplikacja Tcl w jednym pliku; potrzebny jeszcze dowolny interpreter Tcl z dostępem do pakietu starkit: np "ActiveState tcl" lub "tclkit"; także pakiety mogą być dostarczane jako starkity, ładuje się je poprzez: "source plik.kit; package require nazwa"

3. Starpacks - aplikacja Tcl + interpreter w jednym pliku

4. Starkits/Starpacks - opis patrz: http://www.equi4.com/