

Ein Programm, das den Anwender auffordert ein Passwort einzugeben. Wird das Passwort dreimal falsch eingegeben, wird das Programm beendet.

The first screenshot shows the initial state: a label 'Passwort eingeben:', an empty text input field, and a 'bestätigen' button.

The second screenshot shows the input field filled with ten asterisks, with the 'bestätigen' button still visible.

The third screenshot shows the input field empty and the text 'falsches Passwort' displayed below it.

The fourth screenshot shows the input field filled with ten asterisks, the text 'falsches Passwort' below it, and the 'bestätigen' button.

The fifth screenshot shows the input field filled with ten asterisks, the text 'wieder falsches Passwort - einen Versuch hast du noch!' below it, and the 'bestätigen' button.

Wird das Passwort richtig eingegeben, erscheint eine Meldung über das korrekte Passwort.

The screenshot shows the input field empty, the text 'Zugang wird gewährt.' displayed in green below the input field, and the 'bestätigen' button.

Programmcode:

```
#!/bin/sh
# uebungsaufgabe4.tcl \
exec vmwish "$0" ${1+"$@"}
```

```
global zaehler
set zaehler 0
```

```
proc labelSetzen {ergebnis} {
  $::label2 configure -text $ergebnis -foreground
  green
}
```

```
}
```

```
proc programmBeenden {ausgabe} {  
# Befehle, die vor dem Programmende ausgeführt  
werden sollen
```

```
    if {$ausgabe eq "Zugang wird gewährt."} {  
        labelSetzen $ausgabe
```

```
    } else {  
        destroy .
```

```
    }
```

```
}
```

```
proc vergleichen {eingabe} {
```

```
    set passwort B0x0fP4nd0r4
```

```
    if {$eingabe ne $passwort && $::zaehler == 0} {  
        set ausgabe "falsches Passwort"
```

```
        focus .en1
```

```
    } elseif {$eingabe ne $passwort && $::zaehler  
        == 1} {
```

```
        set ausgabe "wieder falsches Passwort -  
            einen Versuch hast du noch!"
```

```
        focus .en1
```

```
    } elseif {$eingabe ne $passwort && $::zaehler  
        == 2} {
```

```
        set ausgabe "Passwort zu oft falsch  
            eingegeben"
```

```
    } elseif {$eingabe eq $passwort} {  
        set ::zaehler 3
```

```
        set ausgabe "Zugang wird gewährt."
```

```
    }
```

```
    incr ::zaehler
```

```
    programmBeenden $ausgabe
```

```
}
```

```
set label1 [label .lb1 -text "Passwort eingeben:"]
```

```
grid $label1 -row 0 -column 0 -padx 2 -pady 2 -
sticky w
```

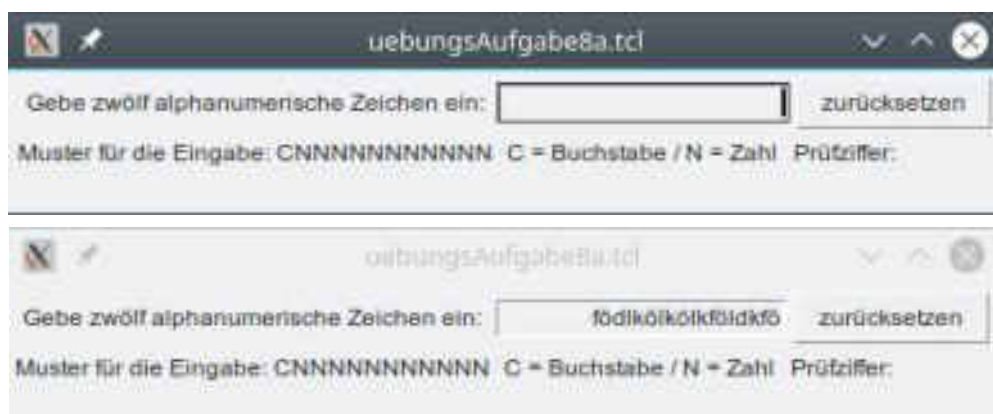
```
set eingabe1 [entry .en1 -textvariable
passwortEingabe -show * -justify right]
grid $eingabe1 -row 0 -column 1 -padx 2 -pady 2
bind $eingabe1 <Return> {focus .bt1}
bind $eingabe1 <KP_Enter> {focus .bt1}
```

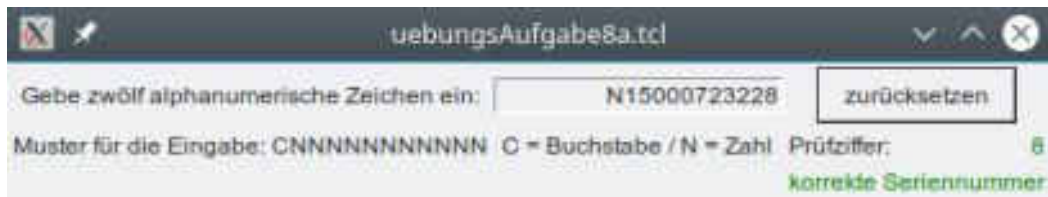
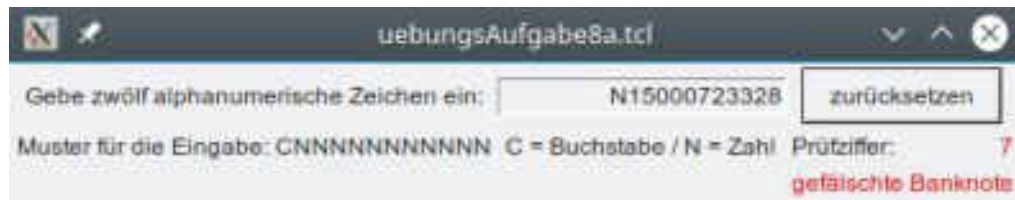
```
set schaltflaeche [button .bt1 -text "bestätigen"
-command {vergleichen $passwortEingabe; .en1
delete 0 end}]
grid $schaltflaeche -row 0 -column 2 -padx 2 -pady
2
bind $schaltflaeche <Return> {vergleichen
$passwortEingabe; .en1 delete 0 end}
bind $schaltflaeche <KP_Enter> {vergleichen
$passwortEingabe; .en1 delete 0 end}
```

```
set label2 [label .lb2 -text ""]
grid $label2 -row 1 -column 0 -padx 2 -pady 2 -
sticky w
```

```
focus $eingabe1
```

Ein Programm, das den Anwender auffordert, eine Seriennummer einzugeben. Die GUI liest eine beliebige Zeichenkette aus zwölf alphanumerischen Zeichen ein. Daraufhin soll geprüft werden, ob die Struktur der eingegebenen Zeichenkette der Vorgabe für eine Seriennummer entspricht. Bei falscher Struktur wird eine Fehlermeldung ausgegeben. Bei richtiger Struktur wird berechnet, ob es sich um eine „korrekte“ oder um eine „gefälschte Banknote“ handelt.





Programmcode:

```
#!/bin/sh
# uebungsaufgabe8a.tcl \
exec vmwish "$0" ${1+"$@"}

proc labelFarbeSetzen {zahl ziffer} {
    if {$zahl == 1} {
        $::label6 configure -text "korrekte
        Seriennummer" -foreground green
        $::label5 configure -text $ziffer -
        foreground green
        update idletasks
    } else {
        $::label6 configure -text "gefälschte
        Banknote" -foreground red
        $::label5 configure -text $ziffer -
        foreground red
        update idletasks
    }
}
```

```

proc zurueckSetzen {} {
    # globaler Zugriff auf Label für die Prüfziffer
    ::label5 configure -text ""
    # globaler Zugriff auf Label für die
    Textmeldung
    ::label6 configure -text ""
}

```

```

proc ermittleLaendercodeFuerBuchstabe {buchstabe}
{
    # ermittle Zahl für Buchstabe dann den Buchstaben
    durch die entsprechende Position ersetzen
    switch $buchstabe {
        A {set laenderCode 1}
        B {set laenderCode 2}
        C {set laenderCode 3}
        D {set laenderCode 4}
        E {set laenderCode 5}
        F {set laenderCode 6}
        G {set laenderCode 7}
        H {set laenderCode 8}
        I {set laenderCode 9}
        J {set laenderCode 10}
        K {set laenderCode 11}
        L {set laenderCode 12}
        M {set laenderCode 13}
        N {set laenderCode 14}
        O {set laenderCode 15}
        P {set laenderCode 16}
        Q {set laenderCode 17}
        R {set laenderCode 18}
        S {set laenderCode 19}
        T {set laenderCode 20}
        U {set laenderCode 21}
        V {set laenderCode 22}
    }
}

```

```

        W {set laenderCode 23}
        X {set laenderCode 24}
        Y {set laenderCode 25}
        Z {set laenderCode 26}
    }
    return $laenderCode
}

```

```

proc pruefen {eingabe} {
    # global berechnetePruefziffer
    # prüfen, ob die Struktur der eingegebenen
    # Zeichenkette der Vorgabe für eine Seriennummer
    # entspricht
    set muster [regexp {[A-Z]{1}[0-9]{11}}
    $eingabe]
    if {$muster == 0} {
        # Ausgabe erzeugen
        set hinweis [tk_messageBox -title "Fehler" -
        message "Die Eingabe entspricht nicht dem
        geforderten Muster." -icon warning -type ok
        -default ok]
        return 0
    }
}

```

```

# Ländercode, bzw. Buchstabe nehmen und mit der
# Position des Buchstabens im Alphabet abgleichen
set buchstabe [string index $eingabe 0]
set laenderCode [ermittleLaendercodeFuerBuchstabe
$buchstabe]

```

```

# den Wert des 1. Index der Eingabe durch den
# neuen Ländercode ersetzen
regsub [string index $eingabe 0] $eingabe

```

```

$laenderCode eingabe

```

Prüfziffer (letzte Zahl der Eingabe) abschneiden
und in eine Variable speichern (für den Vergleich
später)

```
set angegebenePruefziffer [string index $eingabe  
end]
```

die Eingabe ohne das letzte Zeichen speichern

```
set gekuerzteEingabe [string range $eingabe 0 end-  
1]
```

nun die Ziffern 1 bis 12 mit einander addieren
--> ergibt die Quersumme

sprich, die 1. bis letzte Zahl der gekürzten
Eingabe mit einer Schleife durchlaufen und die
einzelnen Positionen miteinander addieren

```
set querSumme 0  
for {set index 0} {$index < [string length  
$gekuerzteEingabe]} {incr index} {  
  set querSumme [expr $querSumme + [string index  
  $gekuerzteEingabe $index]]  
}
```

die Quersumme % 9 --> ergibt den Rest

```
set rest [expr $querSumme % 9]
```

Berechne 8 - Rest

```
set ergebnis [expr 8 - $rest]
```

ist das Ergebnis gleich 0, dann soll die zu
vergleichende Prüfziffer gleich 9 sein

in allen anderen Fällen soll die zu
vergleichende Prüfziffer gleich (8 - Rest) sein

```
if {$ergebnis == 0} {  
  set berechnetePruefziffer 9  
} else {  
  set berechnetePruefziffer $ergebnis  
}
```

```
}
```

```
# ist die zu vergleichende Prüfziffer gleich der  
angegebenen Prüfziffer, soll der Hinweistext  
"korrekte Seriennummer" ausgegeben werde
```

```
# in allen anderen Fällen soll der Hinweistext  
"gefälschte Banknote" ausgegeben werden
```

```
if {  
$berechnetePruefziffer == $angegebenePruefziffer}  
{  
    set ergebnis 1  
} else {  
    set ergebnis 0  
}
```

```
# Werte an Prozedur zur Farbgebung übergeben  
labelFarbeSetzen $ergebnis $berechnetePruefziffer  
}
```

```
# eine beliebige Zeichenkette aus zwölf  
alphanumerischen Zeichen von der Tastatur einlesen  
set label1 [label .lb -text "Gebe zwölf  
alphanumerische Zeichen ein:"]  
grid $label1 -row 0 -column 0
```

```
# Eingabefeld und Übermittlung zur Überprüfung der  
Eingabe
```

```
set eingabe1 [entry .en -textvariable eingabe -  
justify right]  
bind $eingabe1 <Return> {pruefen $eingabe; focus  
$reseten}  
bind $eingabe1 <KP_Enter> {pruefen $eingabe; focus  
$reseten}  
grid $eingabe1 -row 0 -column 1
```

```
# Schaltfläche zum Zurücksetzen der Eingaben
```



```

set reseten [button .bt2 -text "zurücksetzen" -
command {zurueckSetzen; $eingabe1 delete 0 end;
focus $eingabe1}]
bind $reseten <Return> {zurueckSetzen; $eingabe1
delete 0 end; focus $eingabe1}
bind $reseten <KP_Enter> {zurueckSetzen; $eingabe1
delete 0 end; focus $eingabe1}
grid $reseten -row 0 -column 2 -padx 2 -pady 2

# Muster für die Eingabe zeigen
set label2 [label .lb2 -text "Muster für die
Eingabe: CNNNNNNNNNNN"]
grid $label2 -row 1 -column 0

set label3 [label .lb3 -text "C = Buchstabe / N =
Zahl"]
grid $label3 -row 1 -column 1

set label4 [label .lb4 -text "Prüfziffer:"]
grid $label4 -row 1 -column 2 -sticky w

set label5 [label .lb5 -text ""]
grid $label5 -row 1 -column 2 -sticky e

set label6 [label .lb6 -text ""]
grid $label6 -row 2 -column 2

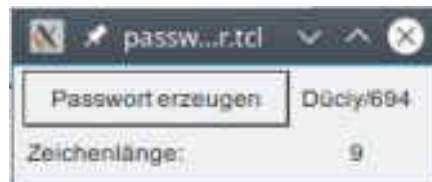
focus $eingabe1

```

Es soll ein Passwortgenerator erstellt werden. Eine zufällige Folge von Buchstaben, Ziffern und Sonderzeichen ist leicht zu programmieren, jedoch schlecht zu merken. Deswegen soll ein Passwortgenerator erstellt werden mit folgenden Vorgaben:

- a) Das Passwort soll insgesamt mindestens sechs, maximal zehn Zeichen enthalten.
- b) Das Passwort besteht aus einem Wort, einem Sonderzeichen und aus einer Zahl.
- c) Das Wort hat zwischen 4 und 6 Buchstaben. Nur der erste Buchstabe ist groß.
- d) Die Zahl hat zwischen 1 und 3 Stellen.
- e) Im Wort wechseln Konsonanten mit Vokalen immer ab.

Beispiele :



Programmcode:

```
#!/bin/sh
# uebungsaufgabe10.tcl \
exec vmwish "$0" ${1+"$@"}
package require vmWidgets

proc labelSetzen {zeichenkette} {
    ::zeichen configure -text $zeichenkette
    set zeichenLaenge [string length $zeichenkette]
    ::laenge configure -text $zeichenLaenge
    update idletasks
}

proc zufallsPassWort {} {
    ## Zahl erzeugen
    # Zahlenlänge zufallsbasiert festlegen
    (zwischen 1 und 3)
    expr srand(clock seconds)
    set zahlenLaenge [expr 1+int(3 * rand())]
```

```

## Zahl zufallsbasiert der Zahlenlänge
    entsprechend erzeugen
switch $zahlenLaenge {
  1 {
    # eine ganze Zahl innerhalb des
    Bereichs 0..9 erzeugen
    set randNum [expr { int(10 * rand()) }]
  }
  2 {
    # eine ganze Zahl innerhalb des Bereichs
    10..99 erzeugen
    set randNum [expr { 10 + int(100 *
      rand()) }]
    # wenn randNum >= 100, dann randNum - 10
    if {$randNum >= 100} {
      set randNum [expr $randNum - 10]
    }
  }
  3 {
    # eine ganze Zahl innerhalb des Bereichs
    100..999 erzeugen
    set randNum [expr { 100 + int(1000 *
      rand()) }]
    # wenn randNum >= 1000, dann randNum -
    100
    if {$randNum >= 1000} {
      set randNum [expr $randNum - 100]
    }
  }
}

```

Sonderzeichen zufallsbasiert erzeugen

Liste mit Sonderzeichen erzeugen

```

set liste [list $ $ % & / ( ) = ? ` * ' _ : \ ; ° ^
, . - # + \ \ / \[ \] | < > ! € μ ~ \" @ \{ \} ß]

```

```

# Listenlänge auf Variable setzen
set anzahl [length $liste]

# zufallsbasierte Zahl anhand von Listenlänge
erzeugen
set zufallsZahl [expr int($anzahl * rand())]

# Index aus Liste mit Zufallszahl auswählen und
den dazugehörigen Wert in eine Variable stecken
set zufallsSonderZeichen [index $liste
$zufallsZahl]

## Wort erzeugen
## Wortlänge zufallsbasiert festlegen (zwischen 4
und 6)
set wortLaenge [expr 4+int(3 * rand())]

## Buchstabenkette mit Wortlänge erzeugen
# Vokalliste erstellen
set vokalListe [list a e i o u ü ö ä]
# Konsonantenliste erstellen
set konsonantenListe [list l g q w r t z p s d f h
j k y x c v b n m]

# eine Zufallszahl aus der Länge jeweils der
Vokal- und Konsonantenliste entnehmen
set vokalListenLaenge [length $vokalListe]
set konsonantenListenLaenge [length
$konsonantenListe]

# die Listen mit einer Schleife durchlaufen und
jeweils zufallsbasiert von den Listen dem Wort
anfügen
set wort ""
set index 0
while {$index < $wortLaenge} {

```

```

set zufallsAlphabetIndex [expr
int($konsonantenListenLaenge * rand())]
set zufallsBuchStabe [lindex $konsonantenListe
$zufallsAlphabetIndex]
append wort $zufallsBuchStabe
set neueWortLaenge [string length $wort]
if {$neueWortLaenge == $wortLaenge} {break}
set zufallsAlphabetIndex [expr
int($vokalListenLaenge * rand())]
set zufallsBuchStabe [lindex $vokalListe
$zufallsAlphabetIndex]
append wort $zufallsBuchStabe
incr index 2
}

```

```

# ersten Buchstabe des neuen Strings in
Großbuchstabe verwandeln

```

```

set wort [string toupper $wort 0]

```

```

## Wort, Sonderzeichen und Zahl verknüpfen
# Passwort ist Wort ( alpha) + Sonderzeichen +
Zahl (integer)

```

```

set ausgabe $wort$zufallsSonderZeichen$randNum

```

```

set passWortLaenge [string length $ausgabe]

```

```

## Passwortlänge prüfen

```

```

# 6 <= Passwortlänge <= 10

```

```

if {6 <= $passWortLaenge <= 10} {

```

```

    # wenn PW korrekt -> Passwort ausgeben

```

```

    labelSetzen $ausgabe

```

```

} else {

```

```

    # wenn PW nicht korrekt -> neu erzeugen

```

```

    zufallsPassWort

```

```

}

```

```

}

```

```

# Prozedur starten
set schaltflaeche [button .bt -text "Passwort
erzeugen" -command {zufallsPassWort}]
grid $schaltflaeche -row 0 -column 0 -padx 2 -pady
2
bind $schaltflaeche <Return> {zufallsPassWort}
bind $schaltflaeche <KP_Enter> {zufallsPassWort}

set zeichen [label .lb -text ""]
grid $zeichen -row 0 -column 1 -padx 2 -pady 2

set beschriftung [label .lb1 -text
"Zeichenlänge:"]
grid $beschriftung -row 1 -column 0 -padx 2 -pady
2 -sticky w

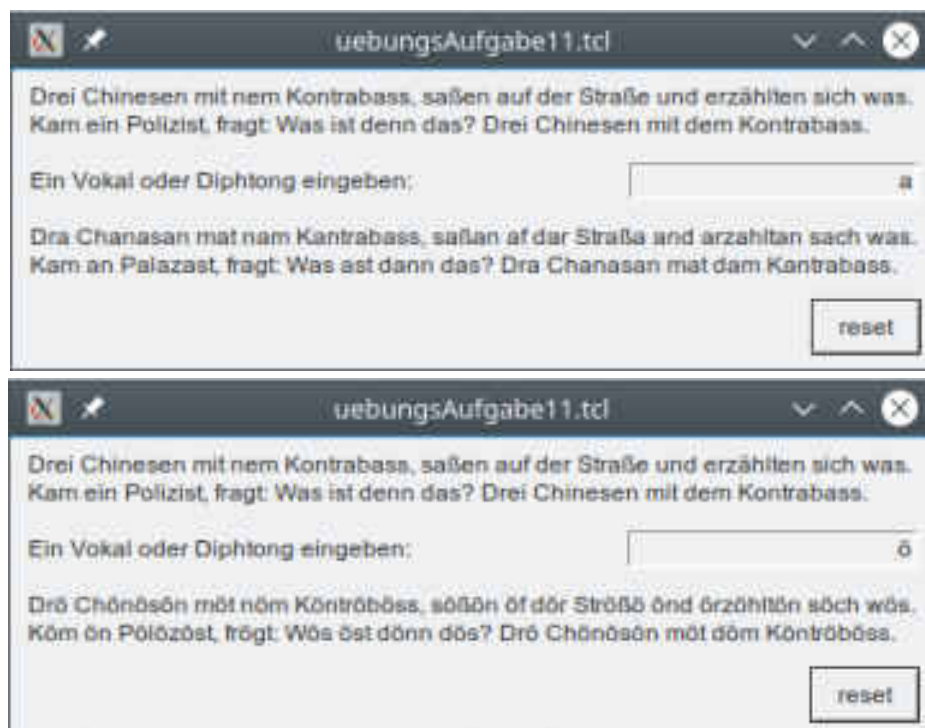
set laenge [label .lb2 -text ""]
grid $laenge -row 1 -column 1 -padx 2 -pady 2

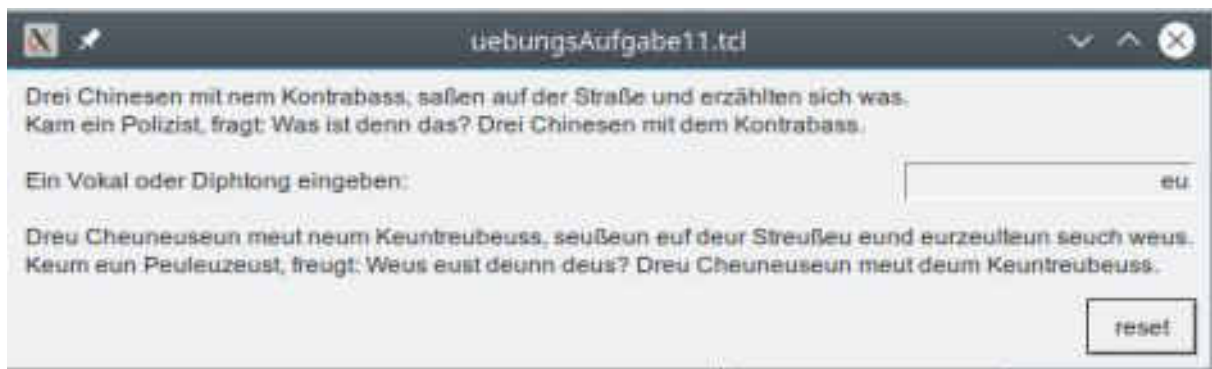
focus $schaltflaeche

```

Es soll eine Routine verfasst werden, die einen Vokal oder Diphtong entgegennimmt, und die entsprechende Strophe mit nur diesem Laut als Strophe ausgibt.
Erlaubte Vokale sind: a, e, i, o, u, ä, ü, ö sowie die Diphtonge: au, ei und eu.
Bei fehlerhafter Eingabe soll eine Fehlermeldung ausgegeben werden.

Beispiele:





Programmcode:

```
#!/bin/sh
# uebungsaufgabe11.tcl \
exec vmwish "$0" ${1+"$@"}
package require vmWidgets

# Eingabe überprüfen
proc pruefen {eingabe} {
# pruefen, ob Eingabe gueltig ist
if {[string match "$eingabe" $::vokaleDiphtongs]
== 0} {
# error "\"$eingabe\" ist keine gültige Eingabe."
vmWidgets::vmDialog .fehler -bd 1 -title "Fehler:"
\
-text "\"$eingabe\" ist keine gültige Eingabe.
Bitte nur Vokale oder Diphtongs eingeben!"\
-image l_error.gif \
-breite 600\
-default 0 \
-buttons [list "tut mir leid"]
return 0
} else {
return 1
}
}

# Prozedur zum zurücksetzen der Eingabe und des
Zieltexts
proc zurueckSetzen {} {
```

```

# globaler Zugriff auf Label für den Zieltext
$::label3 configure -text ""
}

# nur folgende Vokale und Diphtongs sind erlaubt:
au ei eu a e i o u ä ö ü
global vokaleDiphtongs
set vokaleDiphtongs [list au ei eu a e i o u ä ö
ü]

# neuen Text ausgeben lassen
proc zielTextAusgeben {zeichenKette} {
    $::label3 configure -text $zeichenKette
}

proc textErsetzen {eingabe} {
    if {$eingabe eq "ei" || $eingabe eq "au" ||
        $eingabe eq "eu"} {
        set diphtong $eingabe
    }
    set mapListe ""
    foreach element $::vokaleDiphtongs {
        lappend mapListe $element
        lappend mapListe $eingabe
    }
    set neuerText [string map $mapListe [$::label1
cget -text]]
    set laut ""
    if {[string match "*eih*" $neuerText] == 1} {
        set laut "eih"
    } elseif {[string match "*auh*" $neuerText] ==
1} {
        set laut "auh"
    } elseif {[string match "*euh*" $neuerText] ==
1} {
        set laut "euh"
    }
}

```



```

    }
    if {[string length $laut] > 0} {
        set neuerText [string map "$laut $diphtong"
            $neuerText]
    }
# neuen Text einer Prozedur zur Ausgabe übergeben
zielTextAusgeben $neuerText
}

set label1 [label .lb1 -text "Drei Chinesen mit
nem Kontrabass, saßen auf der Straße und erzählten
sich was.\nKam ein Polizist, fragt: Was ist denn
das? Drei Chinesen mit dem Kontrabass\." -justify
left]
grid $label1 -row 0 -column 0 -sticky w -padx 4 -
pady 4

set label2 [label .lb2 -text "Ein Vokal oder
Diphtong eingeben:"]
grid $label2 -row 1 -column 0 -sticky w -padx 4 -
pady 4

set eingabe [entry .en1 -textvariable
vokalDiphtong -validate key -validatecommand
{pruefen %S} -justify right]
grid $eingabe -row 1 -column 0 -sticky e -padx 5 -
pady 5
bind $eingabe <Return> {textErsetzen
$vokalDiphtong; focus $reset}
bind $eingabe <KP_Enter> {textErsetzen
$vokalDiphtong; focus $reset}

set label3 [label .lb3 -text "" -justify left]
grid $label3 -row 2 -column 0 -sticky w -padx 4 -
pady 4

```

```

set reset [button .b -text "reset"]
grid $reset -row 3 -column 0 -sticky e -padx 4 -
pady 4
bind $reset <Return> {zurueckSetzen; $eingabe
delete 0 end; focus $eingabe}
bind $reset <KP_Enter> {zurueckSetzen; $eingabe
delete 0 end; focus $eingabe}

```

focus \$eingabe

Es soll eine Funktion implementiert werden, die als Eingaben zwei Dezimalzahlen Klausurpunkte und Bonuspunkte erhält und zunächst prüft, ob nicht mehr als 100 Klausurpunkte und nicht mehr als 20 Bonuspunkte übergeben wurden. Sollte einer der Tests fehlschlagen, soll eine Fehlermeldung ausgegeben werden z.B. „mehr als 20 Bonuspunkte können nicht angerechnet werden“.

Sollten beide Tests erfüllt sein, so soll die Prozedur die beiden Punktezahlen addieren und die Note wie folgt berechnen:

0 Punkte = Minimalnote

100 Punkte = Maximalnote

Zwischen 0 und 100 Punkten ist die Verteilung linear.

Alles über 100 Punkte = Maximalnote

Beispiele:

uebungsAufgabe12.tcl

Klausurpunkte eingeben:

Bonuspunkte eingeben:

Note berechnen

Note: Maximalnote

reset

uebungsAufgabe12.tcl

Klausurpunkte eingeben:

Bonuspunkte eingeben:

Note berechnen

Note:

reset

Fehler:

 mehr als 100 Punkte können für Klausuren nicht berechnet werden.

tut mir leid

uebungsAufgabe12.tcl

Klausurpunkte eingeben:


Bonuspunkte eingeben:

Note berechnen

Note:

reset

Fehler:

 mehr als 20 Punkte können für den Bonus nicht berechnet werden.

tut mir leid

Programmcode:

```
#!/bin/sh
# uebungsaufgabe12.tcl \
exec vmwish "$0" ${1+"$@"}
package require vmWidgets

# eingegebene Werte für Bonuspunkte überprüfen
proc bonusPunktePruefen {eingabe} {
    if {[string is double $eingabe]} {
        # error "\"$eingabe\" ist keine
        Dezimalzahl."
        vmWidgets::vmDialog .fehler -bd 1 -titel
        "Fehler:" \
        -text "\"$eingabe\" ist keine Dezimalzahl.\" \
            -image l_error.gif \
            -breite 600 \
            -default 0 \
            -buttons [list "tut mir leid"]
        return 0
    } elseif {$eingabe == ""} {
        # error "\"$eingabe\" darf nicht leer sein."
        vmWidgets::vmDialog .fehler -bd 1 -titel
        "Fehler:" \
        -text "Keine Eingabe vorgenommen. Bitte einen
        Dezimalwert eingeben.\" \
            -image l_error.gif \
            -breite 600 \
            -default 0 \
            -buttons [list "tut mir leid"]
        return 0
    } elseif {[string is double $eingabe] &&
    $eingabe > 20} {
        # überprüfen, ob Bonuspunkte über 20 liegen
        # error mehr als "\"$bPunkte\" können nicht
        berechnet werden."
```

```

        vmWidgets::vmDialog .fehler -bd 1 -titel
        "Fehler:" \
            -text "mehr als 20 Punkte können für den
        Bonus nicht berechnet werden."\
            -image l_error.gif \
            -breite 600\
            -default 0 \
            -buttons [list "tut mir leid"]
        return 0
    } else {
        return 1
    }
}

```

eingegebene Werte für Klausurpunkte überprüfen

```

proc klausurPunktePruefen {eingabe} {
    if {[string is double $eingabe]} {
        # error "\"$eingabe\" ist keine
        Dezimalzahl."
        vmWidgets::vmDialog .fehler -bd 1 -titel
        "Fehler:" \
            -text "\"$eingabe\" ist keine
        Dezimalzahl."\
            -image l_error.gif \
            -breite 600\
            -default 0 \
            -buttons [list "tut mir leid"]
        return 0
    } elseif {$eingabe == ""} {
        # error "\"$eingabe\" darf nicht leer sein."
        vmWidgets::vmDialog .fehler -bd 1 -titel
        "Fehler:" \
            -text "Keine Eingabe vorgenommen. Bitte
        einen Dezimalwert eingeben."\
            -image l_error.gif \
            -breite 600\

```

```

        -default 0 \
        -buttons [list "tut mir leid"]
        return 0
    } elseif {[string is double $eingabe] &&
    $eingabe > 100)} {
        # überprüfen, ob Klausurpunkte über 100 liegen
        # error mehr als "\"$kPunkte\" können nicht
        berechnet werden."
        vmWidgets::vmDialog .fehler -bd 1 -titel
        "Fehler:" \
            -text "mehr als 100 Punkte können für
        Klausuren nicht berechnet werden." \
            -image l_error.gif \
            -breite 600\
            -default 0 \
            -buttons [list "tut mir leid"]
        return 0
    } else {
        return 1
    }
}

```

Prozedur zum zurücksetzen der Eingabe und des Zieltexts

```

proc zurueckSetzen {} {
    # globaler Zugriff auf Label für den Zieltext
    $::label4 configure -text ""
}

```

```

proc labelSetzen {ergebnis} {
    $::label4 configure -text $ergebnis
}

```

Note ermitteln

```

proc notenErmittlung {allePunkte} {
    if {$allePunkte == 0} {

```

```

# ist die Gesamtpunktzahl gleich 0, ist die Note
Minimalnote
    set note "Minimalnote"
} elseif {$allePunkte > 0 && $allePunkte < 100}
{
# liegt die Gesamtpunktzahl zwischen 0 und 100,
berechnet sich die Note,
# indem die Punktzahl mit 0.05 multipliziert und 1
dazu addiert wird
    set note [expr {(0.05 * $allePunkte) + 1}]
    if {$note >= 5.50 && $note <= 6.00} {
        set note "sehr gut"
    } elseif {$note >= 4.50 && $note <= 5.49} {
        set note "gut"
    } elseif {$note >= 3.50 && $note <= 4.49} {
        set note "genügend"
    } elseif {$note >= 2.50 && $note <= 3.49} {
        set note "ungenügend"
    } elseif {$note >= 1.50 && $note <= 2.49} {
        set note "schwach"
    } elseif {$note >= 1.00 && $note <= 1.49} {
        set note "schlecht"
    }
}
} elseif {$allePunkte >= 100} {
# ist die Gesamtpunktzahl größer oder gleich 100,
ist die Note Maximalnote
    set note "Maximalnote"
}
# Note an Prozedur zur Ausgabe in ein Label
übergeben
    labelSetzen $note
}

# Gesamtpunkte berechnen
proc punktBerechnung {kPunkte bPunkte} {

```

```

# wenn einer der beiden Punkte doch über dem
Maximalwert sind
    if {$kPunkte > 100} {
        klausurPunktePruefen $kPunkte
        focus $::eingabeKlausurPunkte
    } elseif {$bPunkte > 20} {
        bonusPunktePruefen $bPunkte
        focus $::eingabeBonusPunkte
    } else {
# das Ergebnis in eine Variable Gesamtpunkte
speichern
        set gesamtpunkte [expr {$kPunkte +
            $bPunkte}]
# Gesamtpunkte der Prozedur zur Notenermittlung
übergeben
        notenErmittlung $gesamtpunkte
    }
}

```

```

set label1 [label .lb1 -text "Klausurpunkte
eingeben:" -justify left]
grid $label1 -row 0 -column 0 -sticky w -padx 4 -
pady 4

```

```

set eingabeKlausurPunkte [entry .en1 -textvariable
kPunkte -validate key -validatecommand
{klausurPunktePruefen %S} -justify right]
grid $eingabeKlausurPunkte -row 0 -column 1 -
sticky e -padx 5 -pady 5
bind $eingabeKlausurPunkte <Return>
{klausurPunktePruefen $kPunkte}
bind $eingabeKlausurPunkte <KP_Enter>
{klausurPunktePruefen $kPunkte}

```

```

set label2 [label .lb2 -text "Bonuspunkte
eingeben:" -justify left]

```



```
grid $label2 -row 1 -column 0 -sticky w -padx 4 -  
pady 4
```

```
set eingabeBonusPunkte [entry .en2 -textvariable  
bPunkte -validate key -validatecommand  
{bonusPunktePruefen %S} -justify right]  
grid $eingabeBonusPunkte -row 1 -column 1 -sticky  
e -padx 5 -pady 5  
bind $eingabeBonusPunkte <Return>  
{bonusPunktePruefen $bPunkte}  
bind $eingabeBonusPunkte <KP_Enter>  
{bonusPunktePruefen $bPunkte}
```

```
set noteBerechnen [button .bt -text "Note  
berechnen" -command {punktBerechnung $kPunkte  
$bPunkte}]  
grid $noteBerechnen -row 2 -column 1 -sticky e -  
padx 5 -pady 5  
bind $noteBerechnen <Return> {punktBerechnung  
$kPunkte $bPunkte}  
bind $noteBerechnen <KP_Enter> {punktBerechnung  
$kPunkte $bPunkte}
```

```
set label3 [label .lb3 -text "Note:" -justify  
left]  
grid $label3 -row 3 -column 0 -sticky w -padx 4 -  
pady 4
```

```
set label4 [label .lb4 -text "" -justify right]  
grid $label4 -row 3 -column 1 -sticky e -padx 4 -  
pady 4
```

```
set reset [button .b -text "reset"]  
grid $reset -row 4 -column 1 -sticky e -padx 4 -  
pady 4
```

```

bind $reset <Return> {zurueckSetzen;
$eingabeKlausurPunkte delete 0 end;
$eingabeBonusPunkte delete 0 end; focus
$eingabeKlausurPunkte}
bind $reset <KP_Enter> {zurueckSetzen;
$eingabeKlausurPunkte delete 0 end;
$eingabeBonusPunkte delete 0 end; focus
$eingabeKlausurPunkte}

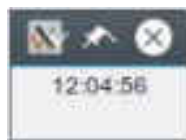
```

focus \$eingabeKlausurPunkte

Eine tickende Uhr ist zu entwickeln:

- a) Variante 1 – eine digitale Uhr: Std:Min:Sec
- b) Variante 2 – eine analoge Uhr: Ziffernblatt, Stunden-, Minuten- und Sekundenzeiger

Variante 1:



Programmcode:

```

#!/bin/sh
# uebungsaufgabe13a.tcl \
exec vmwish "$0" ${1+"$@"}
package require vmWidgets

proc every {ms body} {
    eval $body
    after $ms [list every $ms $body]
}
pack [label .clock -textvar time]
every 1000 {set ::time [clock format [clock sec] -
format %H:%M:%S]}

```

Variante 2:



Programmcode:

```
#!/bin/sh
```

```
# uebungsaufgabe13.tcl \  
exec vmwish "$0" ${1+"$@"}
```

```
package require vmWidgets
```

```
grid [canvas .c -width 200 -height 200]
```

```
set halfpi 1.570796
```

```
set piover6 0.5235987
```

```
set twopi 6.283185
```

```
.c create oval 2 2 198 198 -fill white -outline  
black
```

```
for { set h 1 } { $h <= 12 } { incr h } {  
    set angle [expr { $halfpi - $piover6 * $h }]  
    set x [expr { 100 + 90 * cos($angle) }]  
    set y [expr { 100 - 90 * sin($angle) }]  
    .c create text $x $y -text $h -font {Helvetica  
-12}  
}
```

```
proc hands {} {  
    catch { .c delete withtag hands }  
}
```

```

# die Sekunden seit Mitternacht berechnen
set s [expr { [clock seconds] - [clock scan
00:00:00] }]

# zweiten Zeigers berechnen

set angle [expr { $s * $::twopi / 60. }]
set y [expr { 100 - 90 * cos($angle) }]
set x [expr { 100 + 90 * sin($angle) }]
.c create line 100 100 $x $y -width 1 -tags hands

# Minutenzeigers berechnen

set angle [expr { $s * $::twopi / 60. / 60. }]
set y [expr { 100 - 85 * cos($angle) }]
set x [expr { 100 + 85 * sin($angle) }]
.c create line 100 100 $x $y -width 3 -capstyle
projecting -tags hands

# Stundenzeigers berechnen

set angle [expr { $s * $::twopi / 60. / 60. / 12.
}]
set y [expr { 100 - 60 * cos($angle) }]
set x [expr { 100 + 60 * sin($angle) }]
.c create line 100 100 $x $y -width 7 -capstyle
projecting -tags hands

after 1000 hands
}
hands

```