Hinweis: Die Lösungen sind in Form von Kommentaren
(Zeilen beginnend mit #) erklärt.

Aufgabe 1

(a) Analysieren Sie das Programm `color.sh` auf seine Funktionsweise.

(b) Ändern Sie das Programm `color.sh` derart ab, dass jede zweite Zeile blinkt.

Teil (a) und (b)

```
#!/bin/bash

# Spaltenbeschriftung
echo -e "\n Code    40       41       42       43       44
     45       46       47"
# FG = foreground = Vordergrundfarben durchlaufen
for FG in "39" "30" "31" "32" "33" "37"
do
    # Zeilenbeschriftung
    echo -en " $FG "

    if [ -z $blinken ]
    then
      blinken="5;"
    else
      blinken=""
    fi


    # BG = background = Hintergrundfarben durchlaufen
    for BG in "40m" "41m" "42m" "43m" "44m" "45m" "46m" "47m"
    do
          # "\e[Farbcode" = Folgenden Prompt faerben
          # Farbcode in entsprechender ausgeben
          # Vorder- und Hintergrundfarbe
          # "\e[0m" = auf Standard zurücksetzen
          echo -en "\e[$blinken$FG;${BG}e[$FG;$BG\e[0m "
    done
    echo
done
echo
exit
```

Aufgabe 2

Ändern Sie das in der Vorlesung vorgestellte tree-Skript (`tree.sh`) derart ab, dass die

- Verzeichnisse grün
- Links rot
- ausführbaren Dateien blau

dargestellt werden.

Siehe auch "`tree_loesungsbeispiel.sh`"

```
#!/bin/bash
# Aufrufsyntax:
# tree [-d] [-r "pattern"] [directory]

# Ausgabe:
# dateiname*    einfache Datei und ausführbar
# dateiname/    Directory
# dateiname/-   leeres Directory
# dateiname/-r  nicht lesbares Directory (Inhalt nicht auflistbar)
# dateiname/-x  nicht ausführbares Directory (kein cd möglich)

# usage - gibt die Hilfe aus mit dem korrekten Aufrufsyntax
usage() {
  echo "$0 [-d] [-r \"pattern\"] [directory]" >&2
  echo "     -d  nur Verzeichnisse auflisten" >&2
  echo "     -r  nur Verzeichnisse und Dateien, die durch "pattern"
               abgedeckt sind" >&2
}

# coloring - fuer die faerbung des folgenden Textes
coloring() {
   case $1 in
      */) color='\e[0;32m' ;; # Wenn es auf ein / endet ist es ein Ordner
     *\*) color='\e[0;34m' ;; # Wenn es auf ein * endet ist es ausführbar
      *@) color='\e[0;31m' ;; # Wenn es auf ein @ endet  ist es ein Link
   esac
}

# baum - fuer die graphische Baum-Ausgabe verantwortlich
baum()
{
    # Wenn man nicht in das Verzeichnis wechseln darf
    # -e bei echo = Escape-Sequenzen werden interpretiert --> \c
                    Zeilenvorschub unterdruecken
    if [ ! -x "$1" ]; then echo -e "-x\c";  exit; fi
```

```
# In aktuellen Ordner wechseln
cd $1

# set -f = keine Interpretation von wildcars wie * (no globbing)
set -f

# Setzen der positionalen Parameter mit allen Dateien des Directories
# Erst "", damit set folgende Parameter als Text interpretiert
# || = bedingte Ausführung: falls ls nicht klappt Ordner mit "-r"
                            ausgeben
set "" $(ls -A 2>/dev/null) || { echo -e "-r\c"; exit; }



# Jetzt überflüssiges "" ($1) entfernen
shift

# Wenn Order leer, dann mit "-"  ausgeben
if [ $# -eq 0 ]; then  echo -e "-\c"; exit; fi

# args = alle Dateien und Ordner
args="$*"

# temp_args = gueltige Dateien und Ordner (noch leer)
temp_args=""

# Abarbeiten der angegebenen Optionen wenn $optionEN != null
if [ ! -z "$option" ]
then
    # Falls in $option mindestens ein "d" vorkommt
    case $option in
        *d*) for i in $args
            do
                # Wenn aktueller Eintrag ein Verzeichnis ist, dann
                # in die Liste der gueltigen Argumente aufnehmen
                [ -d "$i" ] && temp_args="$temp_args $i"
            done

            # Gueltige Argumente übernehmen
            args="$temp_args"
            temp_args="";;
    esac

    # Falls $option mindestens ein "r" enthaelt
    case $option in
        *r*) for i in $args
            do
                # Wenn aktueller Eintrag kein Ordner ist, dann
                # prüfen, ob er auf den Text $ra passt. Wenn ja,
                # Eintrag an die gültigen Argumente anhängen.
                if [ -d "$i" ] ||   case $i in
                                        $ra) true;;
                                        *) false;;
                                    esac
                then
```

```
                        temp_args="$temp_args $i"
                    fi
                done

                # Gueltige Argumente übernehmen
                args="$temp_args"
                temp_args="";;
        esac
    fi

    # Setzen der Positionsparameter mit den verbliebenen, gueltigen
      Dateien
    if [ ! -z "$args" ]
    then
        # -d = Nur Verzeichniss-Einträge anzeigen.
        # -F = Typisierungszeichen anhängen (z.B. * für ausführbar)
        set "" $(ls -AdF $args 2>/dev/null)
    else
        set ""
    fi
    # Um ersten leeren Positionsparameter zu beseitigen
    shift

    # Graphische Ausgabe des Baumteils
    # Solange noch mehr als ein Eintrag vorhanden
    while [ ! -z "$2" ]
    do
       # Farbige Ausgabe des Eintrags
       coloring $1
       echo -e "\n${tiefe}|-- ${color}$1\e[0m\c"
       color='\e[0m'

       # Wenn aktueller Eintrag Ordner ist dann Teilbaum für den
       # Ordner erstellen.
       # ( ) Sehr wichtig. Nur in der Subshell darf die Variable tiefe
       # verlaengert werden.
       if [ -d "$1" ]; then (tiefe="${tiefe}|   "; baum $1);  fi
       shift
    done

    # Letzter Teil eines Teilbaums
    # Besondere Behandlung, da er mit "`" abgerundet dargestellt wird


    if [ ! -z $1 ] # wenn $1 kein Leerstring ist
    then
     coloring $1
     echo -e "\n${tiefe}\`-- ${color}$1\e[0m\c"
     color='\e[0m'
     if [ -d "$1" ]; then (tiefe="${tiefe}    "; baum $1) fi
    fi
}

# Abbruchssignale wie z.B. <strg>+<c> abfangen und Skript sauber beednen
```

```
trap 'echo; exit' 0 1 2 3 15

# Abarbeiten der angegebenen Optionen
# Betrachte Optionen h, ?, r mit Pattern dahinter (deswegen ":") und d
while getopts hH\?r:d opt
do
    case $opt in

        # Vom Benutzer gesetzte Optionen merken
        d) option="${option}d";;
        r) option="${option}r";ra="$OPTARG";;

        # Hilfe ausgeben und Beenden
        \?) usage; exit 1;;
        [hH]) usage; exit 1;;
    esac
done

# Alle Optionen entfernen und nur noch die weiteren positionale Parameter
behalten
shift $(expr $OPTIND - 1)

# Wurzel des Baums aus positionaler Benutzereingabe ($1) abgreifen
# Falls Wurzel nicht gegeben, aktuelles Arbeisverzeichnis nutzen
dir=${1:-$(pwd)}

# Die Wurzel muss ein Verzeichnis sein
if [ ! -d "$dir" ]
then
    # Fehler, Wurzel war kein Ordner
    echo "$0: $dir ist kein Directory" >&2

    # Hilfe anzeigen
    usage
    exit 1
fi

# Wurzel farbig anzeigen
echo -e "\e[0;32m$dir\e[0m\c"

# Baum durchlaufen (mit allen Teilbäumen, die rekursiv erzeugt werden)
baum $dir
```