Die Bearbeitung dieser Übung geht nicht in die Bewertung ein.
Die zitierten Übungsblätter beziehen sich auf die ALP 2 aus dem SS 14.

1. Es geht um die Funktion `apply_if` aus der 1. Aufgabe des 2. Übungsblattes.
a) Portieren Sie die Funktion nach Go.
b) Konstruieren Sie eine geeignete Testumgebung.

2. Wir betrachten abgeschlossene Intervalle in $\mathbb{R}$ (siehe 1. Aufgabe des 7. Übungsblattes).
a) Implementieren Sie das folgende Interface:

```
package intervall

type Intervall interface { // Mit "x" ist immer das aufrufende Objekt gemeint.

// Liefert das Intervall [a, b], falls a <= b, andernfalls nil.
// New (a, b float64) Intervall

// Liefert genau dann true, wenn r in x enthalten ist.
  Enthält (r float64) bool

// Liefert den Durchschnitt von x und y, falls er nicht leer ist,
// und in diesem Fall nil als Fehler.
  Durchschnitt (Y Intervall) (Intervall, error)

// Liefert genau dann true, wenn y vollständig in x enthalten ist.
  Beinhaltet (Y Intervall) bool

// Liefert die Länge des Intervalls.
  Länge() float64

// Liefert "[a,b]" für das Intervall [a,b].
  String() string
}
```

b) Konstruieren Sie eine geeignete Testumgebung.

3. Wir betrachten Folgen von beliebigen Elementen, gegeben durch das *interface*:

```
package seq

type (
  Any interface{}
  Func func (a Any) Any
  Pred func (a Any) bool
  Rel func (a, b Any) bool
)
type
  Sequence interface {

// Liefert eine leere Folge, wobei eq
// die Gleichheitsrelation auf ihren Elementen ist.
// New(eq Rel) Sequence

// x enthält keine Elemente.
  Clr()
```

```
// Liefert genau dann true, wenn x keine Elemente enthält.
  Empty() bool

// Liefert genau dann true,
// wenn x vom gleichen Typ wie s ist und mit s übereinstimmt.
  Eq (s Sequence) bool

// Liefert eine Kopie von x (Konsequenz: x.Eq (x.Clone()) == true.)
  Clone() Sequence

// Liefert die Anzahl der Elemente in x.
  Num() uint

// a ist als letztes Element an x angehängt.
  Append (a Any)

// a ist als n-tes Element in x eingefügt, falls n < x.Num();
// andernfalls ist nichts verändert.
  Ins (a Any, n uint)

// Liefert genau dann einen Wert < x.Num(), wenn a in x enthalten ist,
// wobei dieser Wert die erste Stelle angibt, an der a in x vorkommt;
// liefert andernfalls x.Num().
  Ex (a Any) uint

// Liefert das n-te Element von x, falls n < x.Num(); andernfalls nil.
  Get (n uint) Any

// Wenn n < x.Num(), ist das n-te Element von x aus x entfernt;
// andernfalls ist nichts verändert.
  Del (n uint) Any

// Alle Elemente von x sind durch ihre jeweiligen Werte unter f ersetzt.
  Trav (f Func)

// Liefert die Folge, die aus allen denjenigen Elementen von x besteht,
// auf die p zutrifft (in unveränderter Reihenfolge).
  Filter (p Pred) Sequence
}
```

a) Implementieren Sie das Interface, wobei Sie Folgen als *slices* repräsentieren.
b) Konstruieren Sie eine geeignete Testumgebung.

4. Heapsort

a) Konstruieren Sie ein Interface für einen ADT „*beschränkte Halde*“ von Elementen eines beliebigen Typ. Übergeben Sie dem Konstruktor eine strikte Ordnungsrelation `less` vom Typ `Rel` (Typen s. Aufg. 2).
b) Implementieren Sie dieses Interface.
c) Benutzen Sie diesen ADT zur Konstruktion einer Funktion *Heapsort* und testen Sie die.

5. Von einem ADT geordneter Mengen seien folgende Funktionen verlangt:
Konstruktion einer leeren Menge, Leeren einer Menge, Prüfung auf Leersein, Angabe der Anzahl ihrer Elemente, Prüfung auf Enthaltensein eines Elements, Einordnen eines gegebenen Elements, Herausgabe ihres kleinsten und größten Elements, Entfernung eines gegebenen Elements sowie Durchlaufen der Menge mit einer gegebenen Bearbeitungsfunktion für die Elemente.
Übergeben Sie dem Konstruktor eine strikte Ordnungsrelation `less` vom Typ `Rel` (Typen s. Aufg. 2).

a) Konstruieren Sie das Interface.
b) Implementieren Sie es mittels Binärbäumen.
c) Konstruieren Sie eine geeignete Testumgebung.