Prof. Dr. Ralf Hinze
Markus Heinrich, M.Sc.
Sebastian Schweizer, M.Sc.

TU Kaiserslautern
Fachbereich Informatik
AG Programmiersprachen

Übungsblatt 11: Grundlagen der Programmierung (WS 2019/20)

Ausgabe: 24. Januar 2020
Abgabe: 31. Januar 2020, 15 Uhr

Aufgabe 1 Bucketsort auf Arrays (7 Punkte)

Schreiben Sie Ihre Lösungen in die Datei `Bucketsort.fs` *aus der Vorlage* `Aufgabe-11-1.zip`.

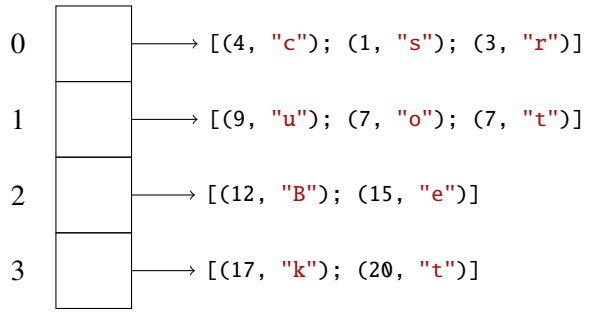
Der Bucketsortalgorithmus eignet sich zur Sortierung von Eingaben, die in einem Intervall (nahezu) gleichverteilt sind. Liegt eine Gleichverteilung der Eingabewerte vor, so ist die Laufzeit linear zur Länge der Eingabe.

Schreiben Sie eine Funktion `bucketsort: (Int * 'T) list -> Int -> Int -> (Int * 'T) list`, die eine Eingabeliste `elems`, bestehend aus Schlüssel/Wert-Paaren, aufsteigend nach dem Schlüssel sortiert. Die Schlüsselwerte müssen dabei im Intervall [0,`upper`) liegen. Bucketsort unterteilt das Intervall in `n` (möglichst) gleich große Teile und ordnet jedes Paar der Eingabe anhand des Schlüsselwertes in einen Bucket ein. Wir repräsentieren die Buckets durch ein Array vom Typ `Array<(Int * 'T) list>` der Länge `n`. Sind alle Elemente in die Buckets einsortiert, werden die einzelnen Listen sortiert (z.B. mit `List.sortBy`) und abschließend konkateniert und als Ergebnis zurückgegeben.

Die folgende Abbildung illustriert anhand eines Beispiels die Funktionsweise des Algorithmus für `upper=21` und `n=4`. Als Eingabeliste verwenden wir:

```
let ex1 = [ (12, "B"); (9, "u"); (4, "c"); (17, "k"); (15, "e")
          ; (20, "t"); (1, "s"); (7, "o"); (3, "r"); (7, "t")]
```

Die Elemente werden ausgehend von ihrem Schlüssel in Listen eines vierelementigen Arrays (“Buckets”) einsortiert:



Im letzten Schritt müssen die im Array enthaltenen Listen sortiert und konkateniert werden. Im Beispiel kommt der Schlüssel 7 doppelt vor. Die ursprüngliche Reihenfolge von Schlüsselduplikaten muss in dieser Aufgabe nicht erhalten bleiben (unsere Implementierung des Sortierverfahrens ist also *nicht stabil*). Wir erhalten als Ergebnis:

```
bucketsort ex1 21 4 =
    [ (1, "s"); (3, "r"); (4, "c"); (7, "t"); (7, "o")
    ; (9, "u"); (12, "B"); (15, "e"); (17, "k"); (20, "t")]
```

Hinweis: Da wir in dieser Aufgabe mit Arrays arbeiten, verwenden wir den Typ `Int` *statt* `Nat`.

Tipp: Das Element `(key, value)` soll in den Bucket mit dem Index `(key * n) / upper` einsortiert werden.

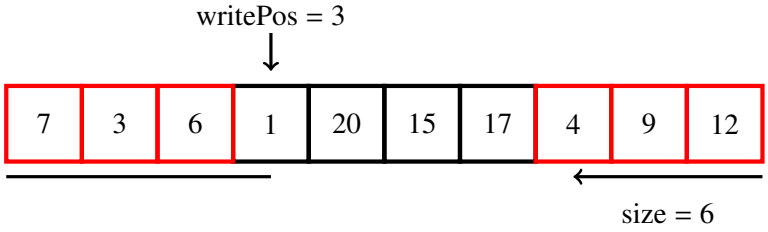
Aufgabe 2 Ringpuffer (11 Punkte)

Schreiben Sie Ihre Lösungen in die Datei `RingBuffer.fs` *aus der Vorlage* `Aufgabe-11-2.zip`.

Wir verwenden ein Array `buffer` und die Ganzzahlen `size` und `writePos`, um einen Ringpuffer zu beschreiben:

```
type RingBuffer<'T> =
    { buffer: Array<'T>
      size: Int ref
      writePos: Int ref}
```

`writePos` bezeichnet den Index innerhalb `buffer`, an dessen Stelle das nächste Element eingefügt wird. `size` gibt die aktuelle Größe des Ringpuffers an, d. h. wie viele Elemente im Ringpuffer vor `writePos` enthalten sind. Wird beim Lesen der Index 0 unterschritten, soll am rechten Ende des `buffer` Arrays weitergelesen werden. Entsprechend soll beim Setzen von `writePos` das Array nicht überschritten werden, sondern auf 0 gesetzt werden. Die folgende Abbildung veranschaulicht die Funktionsweise anhand eines `RingBuffer<Int>` mit Gesamtkapazität 10 und Größe 6, d. h. der Ringpuffer enthält aktuell 6 Elemente (rot markiert, gelesen wird in der Reihenfolge 4, 9, 12, 7, 3, 6).



Wenn ein Element aus dem Ringpuffer gelesen wird, soll `size` um 1 verringert werden, d. h. das gelesene Element kann danach nicht mehr abgerufen werden. Wird in den Ringpuffer geschrieben, so wird der Array-Eintrag an der Stelle `writePos` überschrieben und gleichzeitig `size` um 1 erhöht. Abschließend muss `writePos` auf den Index des nächsten Elements im Ringpuffer gesetzt werden.

Beispiele:

```
let ex1 = { buffer = [|0; 0; 0|]; size = ref 0; writePos = ref 0 }
let ex2 = { buffer = [|1; 2; 3|]; size = ref 1; writePos = ref 1 }
let ex3 = { buffer = [|7; 3; 6; 1; 20; 15; 17; 4; 9; 12|]
            size = ref 6
            writePos = ref 3 }
```

Hinweis: Da wir in dieser Aufgabe mit Arrays arbeiten, verwenden wir den Typ `Int` *statt* `Nat`.

a) Schreiben Sie die Funktion `create: Int -> RingBuffer<'T>`, die einen Ringpuffer mit Gesamtkapazität `capacity` initialisiert. Die anfängliche Schreibposition und Größe des Puffers sollen `0` sein.

   Beispiel: Das Ergebnis von `create<Int> 3` ist `ex1`.

   Hinweis: Verwenden Sie die Funktion `Array.zeroCreate`[1].

b) Schreiben Sie eine Funktion `get: RingBuffer<'T> -> 'T option`, welche das jeweils nächste im Puffer enthaltene Element zurückgibt. Beachten Sie, dass dabei die Anzahl der enthaltenen Elemente `size` um 1 verringert werden soll. Wenn keine Elemente mehr vorhanden sind, soll `None` zurückgegeben werden.

   Hinweis: In F# hat das Ergebnis einer Modulo-Operation das Vorzeichen des Dividenden. Daher ist `-3 % 10 = -3`. *Um ein Ergebnis mit positivem Vorzeichen zu erhalten, können wir* `(10 - 3) % 10 = 7` *rechnen. Für ganze Zahlen n, k mit* $0 \leq k \leq n$ *können wir also statt* `-k % n` *den Ausdruck* `(n - k) % n` *verwenden, um ein positives Ergebnis zu bekommen.*

[1] https://msdn.microsoft.com/visualfsharpdocs/conceptual/array.zerocreate%5b%27t%5d-function-%5bfsharp%5d

c) Schreiben Sie eine Funktion `put: RingBuffer<'T> -> 'T -> unit`, welche ein Element `elem` auf den Ringpuffer schreibt. Das Element soll an der Stelle `writePos` in `buffer` eingefügt werden. Im Anschluss soll `writePos` um 1 erhöht werden, sofern der Index noch im Array liegt, ansonsten soll `writePos` wieder bei 0 beginnen. Entsprechend muss `size` um 1 erhöht werden, sofern die maximale Kapazität des Ringpuffers nicht überschritten wird.

Aufgabe 3 Bibliothek mit Ausnahmen (12 Punkte)

Schreiben Sie Ihre Lösungen in die Datei `Bib.fs` *aus der Vorlage* `Aufgabe-11-3.zip`.

Wir implementieren die Bibliothek von Übungsblatt 10, Aufgabe 2 noch einmal, jedoch verwenden wir Ausnahmen anstelle der Variantentypen für die Rückgaben. Zur Erinnerung: Eine Bibliothek ist definiert als Liste von Büchern. Es werden wieder folgende Typdefinitionen aus `BibTypes.fs` verwendet.

```
type Ausleihstatus =
    | Dauerleihe of String // Name der Arbeitsgruppe, an die das Buch ausgeliehen ist
    | NormaleLeihe of String // Name der Person, an die das Buch ausgeliehen ist
    | Verfuegbar

type Buch =
    { titel: String                      // Eindeutiger Titel
      exemplare: Ausleihstatus ref list // Exemplarliste hat feste Länge
      warteliste: String list ref }     // Liste von Personen
```

Hinweis: Für den Fall, dass Sie Aufgabe 2 von Übungsblatt 10 nicht gelöst haben, befindet sich eine mögliche Lösung in Bib-A10.2.fs. Sie dürfen in Ihrer Lösung der aktuellen Aufgabe die dort enthaltenen Funktionen ***nicht*** *aufrufen.*

a) Schreiben Sie eine Funktion `listFind: ('T -> Bool) -> 'T list -> 'T`, die eine Funktion `predicate` und eine Liste als Argumente erwartet und das erste Element `x` der Liste zurückgibt, für das `predicate x = true` ist. Sollte kein solches Element in der Liste vorhanden sein, soll die Funktion die Ausnahme `NichtGefunden` (definiert in `BibExceptions.fs`) werfen.

b) Verwenden Sie die Funktion `listFind` aus Aufgabenteil a), um eine Funktion `findeBuch: Buch list -> String -> Buch` zu schreiben, die ein Buch anhand seines exakten Titels `titel` in einer Liste von Büchern ("Bibliothek") `bib` sucht.

   Ist das Buch nicht in der Bibliothek vorhanden, soll folgende Ausnahme geworfen werden.

   ```
   exception BuchUnbekannt of String  // Buch nicht in Bibliothek vorhanden
   ```

   Der gesuchte Buchtitel soll der Ausnahme als Argument übergeben werden.

c) Schreiben Sie eine Funktion `leiheBuch: Buch list -> String -> String -> unit`, die eine Liste von Büchern ("Bibliothek") `bib`, einen Buchtitel `titel` und eine Person `person` als Argument erwartet. Als Rückgabetyp wird `unit` verwendet. Die unterschiedlichen Fälle werden durch folgende Ausnahmen abgebildet.

   ```
   exception Warteliste      // Person auf die Warteliste gesetzt
   exception NichtVerfuegbar // Alle Exemplare in Dauerleihe, daher keine Warteliste
   ```

   Die Funktion `leiheBuch` soll sich wie folgt verhalten:

   1. Ist das Buch mit dem Titel `title` nicht in der Bibliothek `bib` vorhanden, soll die Ausnahme `BuchUnbekannt` mit dem gesuchten Buchtitel als Argument geworfen werden.
   2. Ist in `bib` noch ein Buchexemplar verfügbar, so soll dessen Status in `NormaleLeihe` mit dem Namen der ausleihenden Person geändert werden. Die Funktion terminiert erfolgreich.
   3. Sind in der Bibliothek nur Buchexemplare vorhanden, die in Dauerleihe sind, so soll die Funktion die Ausnahme `NichtVerfuegbar` werfen.
   4. Wenn kein Buchexemplar im Status `Verfuegbar` vorhanden ist, es aber mindestens ein Exemplar im Status `NormaleLeihe` gibt, soll die Person ans Ende der Warteliste des Buches angehängt werden. Die Funktion soll die Ausnahme `Warteliste` werfen.

d) Schreiben Sie eine Funktion `rueckgabe: Buch list -> String -> String -> unit`, die eine Liste von Büchern ("Bibliothek") `bib`, einen Buchtitel `titel` und eine Person `person` als Argumente entgegennimmt und `()` zurückgibt, wenn die Person das Buch erfolgreich zurückgeben konnte.

   Die Funktion soll sich wie folgt verhalten:

   1. Wenn das Buch nicht in `bib` ist, kann dieses auch nicht zurückgegeben werden, d. h. die Funktion soll die Ausnahme `BuchUnbekannt` mit dem gesuchten Buchtitel als Argument werfen.
   2. Wird ein Exemplar gefunden, welches von `person` als `NormaleLeihe` ausgeliehen ist, soll der Status des Exemplars auf `Verfuegbar` geändert werden. Falls zusätzlich die Warteliste für das Buch nicht leer ist, soll die erste Person von der Warteliste entfernt und das Buch direkt an diese Person geliehen werden. Die Funktion `rueckgabe` soll schließlich `()` zurückgeben.
   3. Hat `person` kein Exemplar eines Buches ausgeliehen, kann es nicht zurückgeben werden. In diesem Fall soll die Ausnahme `KeinExpemlarVerliehenAn` mit der Person, die das Buch zurückgeben wollte, als Argument geworfen werden.

e) Vergleichen Sie Ihre Lösung mit der ersten Implementierung von Übungsblatt 10. Diskutieren Sie in den Übungen welche Vor- und Nachteile die unterschiedlichen Ansätze jeweils mit sich bringen.