Binärer Suchbaum

Aufgabe 1: conco

Erstellen Sie die Konkordanz (*concordance*) für einen umfangreichen Text.
Eine Konkordanz listet die Wörter eines Textes alphabetisch sortiert mit Angabe ihrer Häufigkeit auf, etwa für diesen Absatz:

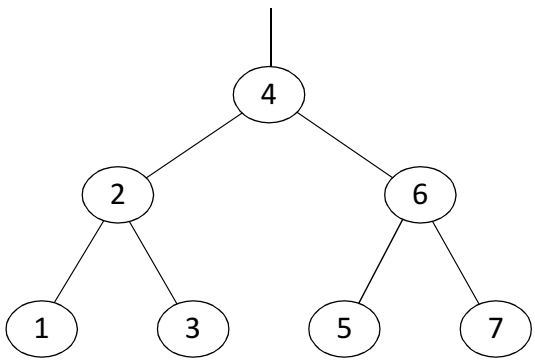
```
Absatz                  1
Angabe                  1
      ...
Häufigkeit              1
Konkordanz              2
      ...
alphabetisch            1
concordance             1
die                     2
      ...
etwa                    1
für                     2
      ...
umfangreichen           1
```

Mit einem binären Suchbaum (*binary search tree, BST*) kann eine Konkordanz leicht erstellt werden. Ein BST ist ein Binärbaum, dessen Knoten einer Ordnung unterliegen, z.B.:

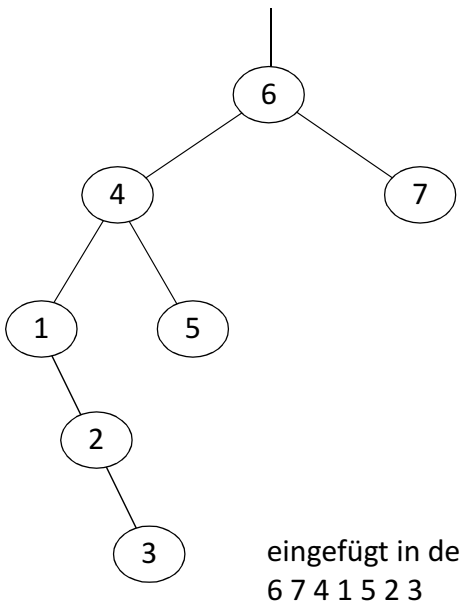
Für jeden Knoten T gilt:

- der Inhalt des *linken* Nachfolgers von T ist *kleiner* als der Inhalt von T
- der Inhalt des *rechten* Nachfolgers von T ist *größer* als der Inhalt von T

Beispiele, Knoten mit ganzzahligem Inhalt:



eingefügt in der Folge:
4 2 6 1 3 5 7

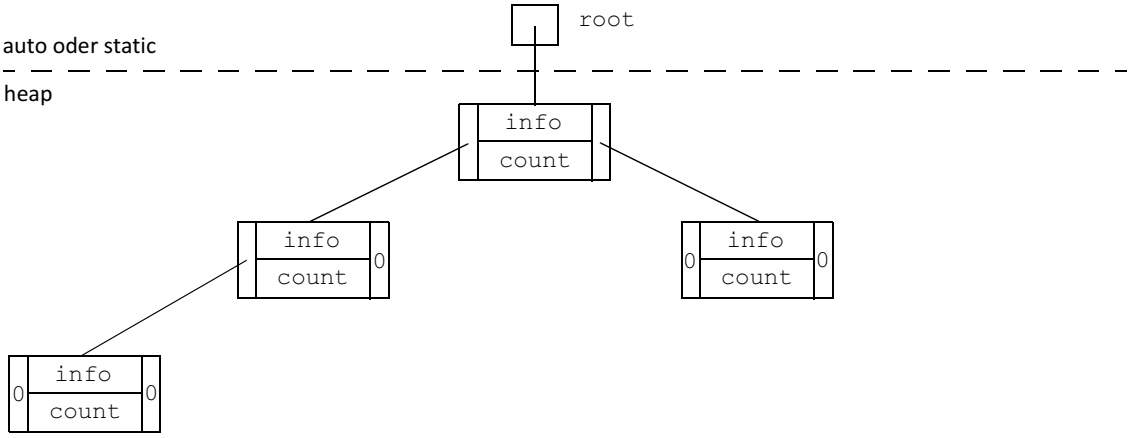


eingefügt in der Folge:
6 7 4 1 5 2 3

Die Struktur des Baums hängt von der Einfügereihenfolge ab.

Um eine Konkordanz zu erstellen, wird der Baum wie folgt konstruiert:

- Das Inhaltselement **info** eines Knotes ist vom Typ **Zeichenkette**.
- Knoten mit identischem Inhalt werden nicht mehrfach gespeichert, vielmehr hat jeder Knoten einen Zähler **count** für die Anzahl dieser Knoteninhalte.
- Der Baum wird auf dem heap allokiert.



Die **Einfügeoperation** kann leicht rekursiv implementiert werden. Dabei werden Knoten ausschließlich als Blätter eingefügt und die Operation beginnt immer bei der Wurzel des Baums.

Wenn T zu Beginn auf die Wurzel zeigt oder NULL ist, dann lautet die rekursive Operation:

insert*(T, Wert):*
wenn T = NULL:
-> Einfügeposition gefunden, Knoten hier erzeugen
sonst:
wenn Wert < Wert dieses Knotens:
-> links rekursiv weitersuchen, ***insert****(T-links, Wert)*
wenn Wert > Wert dieses Knotens:
-> rechts rekursiv weitersuchen, ***insert****(T-rechts, Wert)*
sonst:
-> identischen Knoten gefunden, Zähler inkrementieren

Hierbei ist zu beachten, dass *insert* den Wert von T ändern können muss.

Auch die Ausgabe der Knoteninhalte in sortierter Folge kann rekursiv erfolgen. Die sogenannte in-order **Traversierung** (inorder traversal) *besucht* die Knoten des Baums in sortierter Folge:

traverseInorder*(T):*
wenn T != NULL:
-> ***traverseInorder****(T-links)*
-> ***visit(...)****, besuche diesen Knoten*
-> ***traverseInorder****(T-rechts)*

Hinweise zur Implementation:

- Verwenden Sie mehrere Dateien, um die Anwendung *conco* von dem Container *Bst* zu trennen. Auch sollten Prototypen und Definitionen separiert werden.
- Prototypen stellen die Schnittstelle zum Container dar.
- Beginnen Sie mit der Definition der Datenstruktur.
- Legen Sie fest, in welchem Gültigkeitsbereich sich der root-Zeiger befinden soll. Eine Anwendung sollte nämlich auch mehrere Bäume „instanziieren“ können.
- Die **Funktion zum Vergleichen** der Knoteninhalte soll **als weiterer Parameter** an die insert-Funktion übergeben werden. Der Parameter soll so typisiert werden, dass die Standardfunktion **`strcmp`** verwendet werden kann.
- Die Besuchsfunktion **`visit`** gibt z.B. die Information eines Knotens (`info` und `count`) an `stdout` aus. Um unterschiedliche `visit`-Funktionen verwenden zu können, soll diese Funktion **als Parameter** an die Funktion **`traverseInorder`** übergeben werden.
- Legen Sie fest, in welcher Datei die verschiedenen Vergleichs- und Besuchsfunktionen am besten aufgehoben sind.
- Gegeben ist:
  - eine Funktion **`getLine()`** zum Einlesen einer Zeile von `stdin`
    Achtung: diese Funktion deallokiert nicht. Sollten Sie im rufenden Programm den Speicher freigeben?
  - ein Programm **`genWords.c`** zum Erzeugen von Zufallswörtern
  - eine Datei **`LutherBibelUnixLF.txt`**, die Unix-LFs enthält und als Eingabedatei dienen kann.
- Schreiben Sie eine weitere Vergleichs- und eine weitere Besuchsfunktion Ihrer Wahl und wenden Sie sie an.
  (z.B.: Groß-/Kleinschreibung für Sortierung ignorieren, Strings anhand der Länge vergleichen und eine *Stringlängenkonkordanz* erstellen, ...)
- Erstellen Sie Ihre executables als Release-Versionen.

Mögliche Programmaufrufe:

```
./genWords
./genWords  > zuf.txt
./conco < zuf.txt
./conco < zuf.txt > conco.out

./genWords | ./conco
./genWords | ./conco > conco.out

./conco < LutherBibelUnixLF.txt > conco.out
```

Datei anzeigen:

`cat conco.out` oder
`more conco.out` (seitenweise anzeigen)

Hinweis:
Das Ubuntu-Terminalfenster kann auch Umlaute anzeigen:
Menü -> Terminal -> Zeichenkodierung festlegen -> Westlich (ISO ...)