Programmiersprachen 2 — Übung 10: Bonusaufgabe — WS 2000

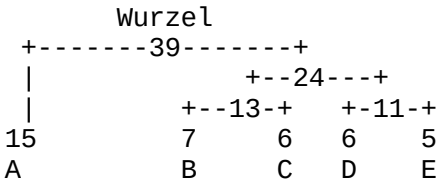
(binärer Baum)

Aufgabenstellung:

Um eine Datei nach dem Huffman-Algorithmus zu komprimieren, muß ein binärer Baum aufgebaut werden. Dies soll heute geübt werden.

Der Huffman-Baum

- Jedes aufgetretene Zeichen wird in einem Knoten gespeichert, der die Häufigkeit als Gewicht enthält.
- Die Knoten werden nach ihrem Gewicht sortiert.
- Die beiden freien Knoten mit dem geringsten Gewicht werden ermittelt.
- Ein Elternknoten wird für diese beiden Knoten erzeugt. Diesem Elternknoten wird ein Gewicht zugewiesen, das der Summe der beiden Gewichte der Kindknoten entspricht.
- Der Elternknoten wird in die Liste der freien Knoten nach Gewicht einsortiert, und die zwei Kindknoten werden aus der Liste entfernt.
- Die oben genannten Schritte werden solange ausgeführt, bis nur noch ein freier Knoten übrig ist. Dieser freie Knoten wird als Wurzel des Baumes bezeichnet.
- Beispiel: In einer Datei treten die folgenden Buchstaben in folgender Häufigkeit auf:
  A 15, B 7, C 6, D 6, E 5
  Der Baum sieht letztendlich also folgendermaßen aus:



Ein Knoten ist folgenden Typs:

```
TKnoten = RECORD
            Zeichen: char;
            Gewicht: word;
            left, right: ^TKnoten
          END;
```

Zeigen left und right auf NIL, so handelt es sich um ein Blatt; das Gewicht gibt dann an, wie häufig dieses Zeichen in der Datei aufgetreten ist.

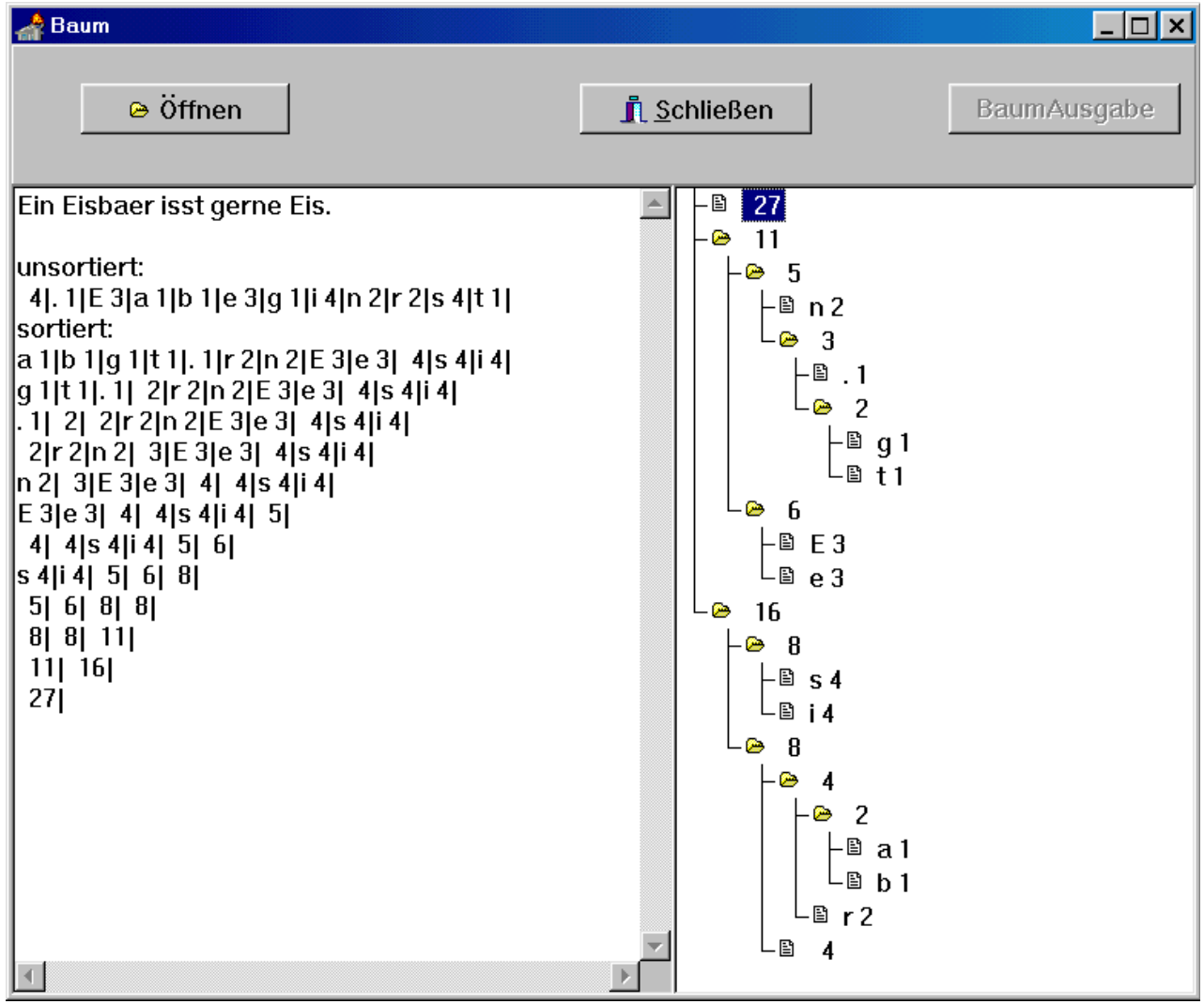
- Erstellt ein Array[0..255] mit Zeigern auf die Baumblätter und initialisiert die Blätter.
- Lest eine Datei zeichenweise ein und inkrementiert dabei das entsprechende Blattgewicht.
- Baut aus dem Array eine TList auf, die nur Blätter mit Gewicht > 0 enthält.
- Sortiert die Liste nach Gewicht.
- Faßt jeweils zwei nebeneinanderliegende Blätter (Knoten) zu einem zusammen (der neue Knoten enthält als Gewicht dann die Summe der beiden Gewichte) und sortiert ihn ein, bis nur noch eine Wurzel übrig ist. Gebt nach jedem Zusammenfassungsdurchlauf einen Zwischenstatus aus (siehe Beispiel auf Rückseite).
- Gebt am Ende den erstellten Baum aus.
- Gebt beim Programmende den Speicher ordnungsgemäß wieder frei!

Zur Ausgabe werden ein Memo und die Komponente TOutline benutzt.

- Gebt nach jeder Änderung in der Liste einen Zwischenstatus im Memo aus, also nach dem Einlesen, dem Sortieren und jeder neuen Listenzusammenfassung.
- Zur Ausgabe des Baumes braucht nur die gegebene Prozedur aufgerufen werden, die den Baum im Outline darstellt.

- **TList**
  - ist ein Zeiger auf eine Liste von Zeigern,
  - muß also mit *Create* erzeugt und mit *Free* wieder entfernt werden,
  - hat u.a. die Methoden *Add*, *Delete*, *Insert*, *Remove*, *Move*, *Exchange*, *First* und *Last*,
  - Elemente lassen sich über *Items* ansprechen und der Index läßt sich mit *IndexOf* ermitteln,
  - die Anzahl der in der Liste enthaltenen Elemente steht in *count*.

Als Hilfe ist das gesamte Programmgerüst schon vorgegeben. Lediglich die mit (*!!!*) kommentierten Stellen sind von Euch zu programmieren.



Was hat das mit Komprimierung zu tun:

Wie Ihr seht, sind die Zeichen, die am häufigsten auftauchen, am nächsten an der Wurzel. Ordnet man jetzt jeder Verzweigung nach links (von der Wurzel aus gesehen) eine Null und jeder nach rechts eine Eins zu, so würden (statt jedes Zeichen mit 8 Bit) die häufig auftretenden Zeichen mit wenig Bits kodiert werden, die seltenen mit viel.

Im Beispiel links wären das folgende Entsprechungen:

- n: 000
- g: 00110
- t: 00111
- E: 010
- i: 101

Compris?