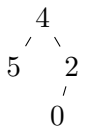
Algorithmen und Datenstrukturen

Aufgabe 1 (3.2.36 b, 3.2.34 b, 3.2.46 c, AGS 3.2.36 a)

Gegeben sind Typdefinitionen für Listen und Binärbäume:

```
1  typedef struct element *list;
2  struct element { int value;  list next; };
3
4  typedef struct node *tree;
5  struct node { int key; tree left, right; };
```

(a) Geben Sie eine Funktion `tree createNode(int n, tree l, tree r)` an, welche einen Binärbaum erzeugt, sodass dessen Wurzel den Schlüsselwert `n` hat und die Nachfolger der Wurzel `l` und `r` sind.

Geben Sie Aufrufe der Funktion an, um den Baum  zu erzeugen.

(b) Geben Sie eine Funktion `void insertl(tree *tp, int n)` an, welche einen Knoten mit dem Schlüsselwert `n` als am weitesten linksstehenden Knoten in den Baum `*tp` einfügt. Gehen Sie davon aus, dass `tp != NULL` ist.

(c) Ein *Blatt* ist ein Knoten, dessen linker und rechter Teilbaum jeweils leer ist. Implementieren Sie eine Funktion `int leafprod(tree t)`, welche für einen Eingabebaum `t` das Produkt der Knotenbeschriftungen der *Blätter* in `t` berechnet!

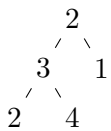
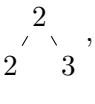
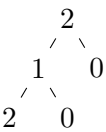
(d) Schreiben Sie eine Funktion `defol(tree *tp)`, welche alle *Blätter* des Baumes `*tp` entfernt. Gehen Sie davon aus, dass `tp != NULL` ist.

(e) Schreiben Sie eine Funktion `treeToList`, welche aus einem beliebigen Binärbaum t des oben genannten Typs eine Liste l der geraden Schlüsselwerte (`key`) von t generiert. Dabei soll l die Schlüsselwerte in folgender Reihenfolge enthalten:

- zunächst die geraden Schlüsselwerte im linken Teilbaum von t,
- dann gegebenenfalls den Schlüsselwert am Wurzelknoten, und
- zuletzt die geraden Schlüsselwerte im rechten Teilbaum von t.

Geben Sie alle dazu erforderlichen Variablendeklarationen und einen Aufruf Ihrer Funktion an. *Hinweis*: Nutzen Sie die Funktion `append` aus der vorherigen Übung.

(f) Schreiben Sie eine Funktion `tree baz(tree s, tree t)`, so dass der Baum `baz(s, t)` aus `s` hervorgeht, indem jede Knotenbeschriftung n in `s` durch die *Anzahl* der Knoten mit Beschriftung n in `t` ersetzt wird. Die Bäume `s` und `t` sollen dabei nicht verändert werden!

Beispiel: Wenn `s =`  und `t =`  , dann ist `baz(s,t) =`  .

Zusatzaufgabe 1 (AGS 3.2.48 ⋆)

(a) Wir programmieren in C ein Spiel für einen Spieler, der ein Wort (z.B. `geheimnis`) erraten muss. Dazu gibt er nacheinander Buchstaben ein. Nach jedem Buchstaben enthüllt das Programm, an welchen Positionen des Wortes dieser vorkommt. Falls der Buchstabe nicht im Wort vorkommt, protokolliert das Programm einen Fehlversuch. Der Spieler gewinnt, wenn er das Wort mit weniger als 5 Fehlversuchen errät; andernfalls hat er das Spiel verloren

Ein unvollständiger Ausschnitt des Programms ist unten gegeben. Vervollständigen Sie ihn unter Beachtung der Kommentare, sodass folgendes gilt:

- Das zu erratende Wort ist im Array `wort` und der Ratefortschritt des Spielers im Array `fortschritt` gespeichert, wobei ein '`-`' für einen noch nicht erratenen Buchstaben steht. Die Variablen `laenge` und `fehlversuche` speichern die Anzahl der Positionen von `wort` bzw. die Anzahl der bisherigen Fehlversuche.
- Vor jedem Versuch gibt das Programm `fortschritt` und `fehlversuche` aus. Anschließend gibt der Spieler ein Zeichen `eingabe` ein.
- Das Programm trägt an den Positionen von `fortschritt`, die den Vorkommnissen von `eingabe` in `wort` entsprechen, das Zeichen `eingabe` ein und erhöht ggf. `fehlversuche`.
- Nach Spielende teilt das Programm dem Spieler mit, ob er gewonnen oder verloren hat.

Sie dürfen eine Hilfsfunktion `int fertig(char fortschritt[], unsigned int laenge)` verwenden, die 1 zurückgibt, wenn das Wort komplett erraten wurde, und andernfalls 0.

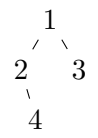
```
char wort[] = "geheimnis";
char fortschritt[] = "---------";
unsigned int fehlversuche = 0;
unsigned int laenge = 9;
while (                                                    ) {
  char eingabe;
  // Bekanntgabe von `fortschritt` und `fehlversuche`

  printf("Nächster Versuch: ");
  // Eingabe einlesen

  // Eingabe prüfen; `fortschritt` bzw. `fehlversuche` anpassen

}
// Bekanntgabe des Gewinners
```

(b) In Aufgabe 1 ist eine Typdefinition für Listen gegeben.
Geben Sie eine Funktion `void delete(list *lp, unsigned int index)` an, die das Element von `*lp`, das sich an der Position `index` befindet (beginnend bei 0), löscht und seinen Speicherbereich freigibt. Falls `index` größer oder gleich der Länge der Liste ist, soll `*lp` unverändert bleiben. *Hinweis:* Sie können davon ausgehen, dass `lp != NULL` gilt.

Zusatzaufgabe 2 (AGS 3.2.49 b,c ⋆)

In Aufgabe 1 ist eine Typdefinition für Binärbäume gegeben.

(a) Schreiben Sie eine Funktion `height`, welche für einen gegebenen Binärbaum die Höhe berechnet und zurückgibt. Die Höhe eines Binärbaumes ist die größte Anzahl von Knoten auf dem Pfad von der Wurzel bis zu einem Blatt; der leere Baum soll die Höhe 0 haben. Der rechtstehende Baum hat die Höhe 3.



(b) Schreiben Sie eine C-Funktion `bfs`, welche für einen gegebenen Binärbaum t einen Binärbaum s zurückgibt, sodass s aus t hervorgeht, indem der Balancefaktor jedes Knotens dem Schlüsselwert dieses Knotens zugewiesen wird. Der Baum t soll dabei nicht verändert werden.

Der Balancefaktor eines Knotens ist die Differenz der Höhe des rechten Teilbaumes minus der Höhe des linken Teilbaumes unter dem Knoten. Die Bäume rechts zeigen ein Beispiel des zurückzugebenden Baumes s (unten) für eine Eingabe t (oben).

Hinweis: Nutzen Sie die Funktion `height` aus dem Aufgabenteil (a).