

# Inf ProgTech Übung: Allgemeine Baum-Funktionen

*Klaus Kusche*

## Binärer Baum

Implementiere einen (*unbalanzierten*) binären Baum, der als Nutzdaten (und zugleich Ordnungskriterium) nur einen (nichtnegativen) **int**-Wert pro Knoten speichert. Der Baum soll folgende Funktionalität (hier als Ausschnitt einer **class**-Deklaration, d.h. als Methoden einer **Baum**-Klasse) bieten:

```
// füge "val" ein
// Returnwert: true wenn ok, false wenn schon vorhanden
bool Insert(int val);
// lösche "val"
// Returnwert: true wenn ok, false wenn nicht vorhanden
bool Delete(int val);
// suche "val"
// Returnwert: true wenn gefunden, false wenn nicht vorhanden
bool Exists(int val) const;

// liefert den kleinsten Wert im Baum oder -1 bei leerem Baum
int getFirst() const;
// liefert den nächsten Wert nach "val" im Baum
// (den kleinsten Wert größer "val", egal, ob es "val" gibt oder nicht)
// oder -1, wenn es kein Element größer "val" mehr gibt
int getNext(int val) const;

// Anzahl der Elemente im Baum
int getSize() const;
// Ist der Baum leer?
bool isEmpty() const;
// Ausgabe von Elementzahl, durchschnittlicher Höhe, maximaler Höhe
void printStats() const;
// Konsistenzprüfung des gesamten Baumes:
// * Haben alle Knoten die richtige Ordnung
//   (linker Sohn < Knoten < rechter Sohn)?
// * Stimmen alle Vater-Pointer?
void checkTree() const;
```

Du kannst dir aussuchen, ob du C oder C++ verwendest (bei C sind die Methoden dann eben durch analoge normale Funktionen mit einem zusätzlichen Parameter für die Wurzel des Bauems zu ersetzen), bzw. ob du mit zwei Klassen (für den Baum als Ganzes und für die einzelnen Baumknoten), mit einer Klasse (für den Baum, die einzelnen Knoten sind dann eine normale **struct**) oder nur mit **strukt**-Daten arbeitest.

Du sollst aber "normale" Pointer verwenden, keine Auto-Pointer oder vergleichbare höhere Pointer-Typen aus der STL oder Boost.

### **Achtung:**

- Achte darauf, dass jeglicher dynamisch angelegte Speicher auch wieder sauber freigegeben wird (bei Verwendung von C++: Im Destruktor!)

- Wenn du C++ verwendest, musst du auch den Copy-Konstruktor und den Assignment-Operator entweder richtig implementieren (als “deep copy”, rekursiv) oder eben deren Verwendung unmöglich machen.

**Tipp:**

- Das Kopieren sowie **getSize**, **printStats** und **checkTree** dürfen und sollen rekursiv gelöst werden.

Bei **getNext** hast du die Wahl: Rekursiv ist der Code merklich kürzer, mit Schleife ist es die bessere Denk- und Programmierübung und etwas schneller (für den produktiven Ernstfall würde ich es mit Schleife programmieren).

Bei den anderen Funktionen sollte man ohne Rekursion auskommen.

- Wenn du **getNext** rekursiv löst, wird dein Baum ohne Vater-Pointer auskommen. Für ein nicht-rekursives **getNext** wirst du in jedem Knoten auch einen Vater-Pointer brauchen. Mit Vater-Pointer ist es jedenfalls die bessere Übung!
- Programmiere vorsichtig: Verwende **assert** reichlich, um unmögliche Programmezustände abzufangen!

Schreibe dazu ein Hauptprogramm. Es soll:

- Viele Inserts und Deletes von Zufallszahlen machen (einige tausend), und danach den Baum auf Konsistenz prüfen und die Statistiken ausgeben.
- Wahlweise (bei kleiner Knotenzahl) den Baum auch komplett sortiert durchlaufen (mit **getFirst** und **getNext**) und alle Werte ausgeben.
- Das Ganze ein paar Mal wiederholen.
- Und schließlich den Baum mit **Delete** komplett entleeren.