Übungsblatt 3

Implementieren Sie einen Suchbaum in *C* oder $C++$, in dem der Einfachheit halber nur ganze Zahlen abgespeichert werden sollen. Verwenden Sie die Struktur (hier in C++ Syntax, unter Linux mit `g++` compilieren, unter Visual Studio als `C++`-Projekt anlegen, in C die Struktur mit typedef erweitern)

```
/* Datenstruktur eines Knotes des Baums */
struct Node {
  int value;
  Node *left, *right;
  Node *parent;  // optional
};
```

Sie können den Wurzelknoten entweder in der `main`-Funktion als Variable definieren oder in eine eigene Struktur für den Suchbaum anlegen:

```
/* Datenstruktur des Suchbaums */
struct SearchTree {
      Node *root;
};
```

Implementieren Sie mindestens die drei Funktionen

- **int insertNode(Node * n, int val)**.

  Die Funktion wird rekursiv ausgeführt: Ist der Wert des Element kleiner (größer) als der Wert des Knotens, führe **insertNode** auf dem linken (rechten) Teilbaum aus, bis der entsprechende Teilbaum leer ist. An dem leeren Teilbaum wird der neu erzeugte Knoten mit dem neuen Wert eingefügt. Ist der Wert bereits vorhanden, füge ihn nicht noch einmal ein. Ist der Baum leer, wird der neue Knoten der Wurzelknoten. Der Rückgabewert soll -1 sein, falls der Wert bereits vorhanden ist, ansonsten 0

  Rufen Sie die Funktion ausgehend von dem Wurzelknoten auf oder über eine optionale Funktion **int insert (SearchTree * baum, int val)**

- **bool containsNode(Node * n, int val)**

  Die Funktion wird ebenfalls rekursiv ausgeführt: Ist der Wert des Elements kleiner (größer) als der Wert des Knotens, suche im linken (rechten) Teilbaum weiter bis das Element gefunden wurde oder der Teilbaum leer ist. Ist der (Teil-)Baum leer, ist das Element nicht vorhanden. Der Rückgabewert soll *true* sein, wenn der Wert gefunden wurde, ansonsten *false*.

  Rufen Sie die Funktion ausgehend von dem Wurzelknoten auf oder über eine optionale Funktion **bool contains (SearchTree * baum, int val)**

- **void inorder (Node * n)**

  Die Funktion gibt die Werte mittels Inorder-Traversierung aus (siehe Vorlesungsfolien). Auch die Funktion ruft sich rekursiv auf, bis alle Werte durchlaufen wurden.

  Rufen Sie die Funktion ebenfalls ausgehend von dem Wurzelknoten auf oder über eine optionale Funktion **void inorder (SearchTree * baum)**

Erstellen Sie einen Testtreiber und Testfälle, um die Korrektheit Ihrer Implementierung nachzuweisen.

Erstellen Sie ferner zwei Messreihen, bei denen jeweils $n = 2^{12}, 2^{13}, \ldots, 2^{17}$ Werte in den Suchbaum eingefügt werden. In der ersten Messreihe werden verschiedene Werte in aufsteigender Reihenfolge eingefügt, in der zweiten Messreihe werden zufällige Werte eingefügt. Es soll jeweils die Zeit ausgegeben werden, die zum Einfügen aller Werte benötigt wird. Wie verhält sich die Laufzeit als Funktion von n?

Achtung, unter Visual Studio kann es leicht zu einen `Stack overflow` kommen,[1] d.h. der Stack-Speicherbereich ist nicht groß genug. Falls das bei Ihnen der Fall ist, wann tritt das Problem auf?

[1] Mit z.B.

```
#include <sys/resource.h>
...
struct rlimit MyLimits;
getrlimit(RLIMIT_STACK,&MyLimits);
MyLimits.rlim_cur = MyLimits.rlim_max / 2;
setrlimit(RLIMIT_STACK,&MyLimits);
```

kann die Stack-Größe auf die halbe maximale Größe gesetzt werden und das Programm sollte wieder laufen.
Alternativ kann über "Projekt" → "Eigenschaften" → "Linker" → "System" → "Stapelreservierungsgröße" die "Stapelreservierungsgröße" (Stackgröße) rauf gesetzt werden (Standard ist nur 1 MB), z.B: "250000000" → ca. 250 MB