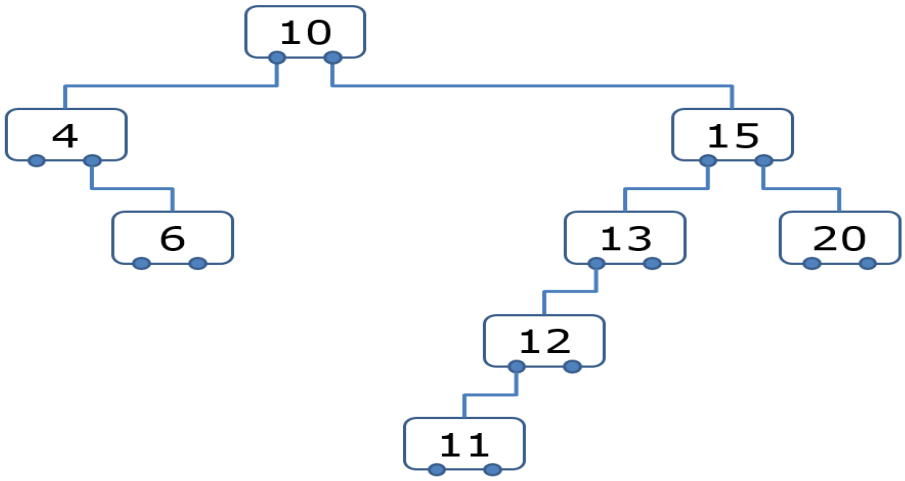
# Praktikumstermin Nr. 10, INF
# Dynamische Datenstrukturen - Binärer Suchbaum, Zeichenkette suchen mit C-Strings

***2019-12-11: Pflicht zum Hochladen der zweiten Aufgabe nach Jenkins hinzugefügt.***

## Aufgabe INF-10.01: Dynamische Datenstruktur: Binärer Suchbaum (duplikatfrei) über `int` Werten

Ein *Binärer Suchbaum (ohne Duplikate) über* `int` *Werten* ist eine Datenstruktur, in der `int` Werte in den Knoten der Datenstruktur nach den im folgenden beschriebenen Regeln gespeichert werden.




Jeder Knoten der Datenstruktur speichert genau einen `int` Wert und besitzt genau einen *Elternknoten* (Ausnahme: *Wurzelknoten* des Baums, der keinen Elternknoten besitzt) und höchstens zwei *Kindknoten*.

Der *erste* in den Baum einzufügende `int` Wert wird im neu zu erzeugenden Wurzelknoten des Baums abgelegt.

Jeder weitere einzufügende `int` Wert wird nach folgendem Prinzip in den Baum eingefügt: Ausgehend vom Wurzelknoten wird der neue Wert mit dem im jeweiligen Knoten gespeicherten Wert verglichen.

1. Ist der neue Wert *gleich* dem Wert im Knoten, so wird der neue Wert nicht erneut in den Baum eingefügt (*duplikatfreier* Baum).

2. Ist der neue Wert *kleiner* dem Wert im Knoten und besitzt der Knoten *keinen* linken Kindknoten, so wird der neue Wert in einen neu zu erzeugenden linken Kindknoten eingefügt.

3. Ist der neue Wert *kleiner* dem Wert im Knoten und besitzt der Knoten einen linken Kindknoten, so wird die Prüfung ab Fall 1. für den linken Kindknoten erneut vorgenommen.

   *Fälle 4. und 5. sind analog zu 2. und 3.:*

4. Ist der neue Wert *größer* dem Wert im Knoten und besitzt der Knoten *keinen* rechten Kindknoten, so wird der neue Wert in einen neu zu erzeugenden rechten Kindknoten eingefügt.

5. Ist der neue Wert *größer* dem Wert im Knoten und besitzt der Knoten einen rechten Kindknoten, so wird die Prüfung ab Fall 1. für den rechten Kindknoten erneut vorgenommen.

Programmieren Sie eine geeignete Datenstruktur `BaumKnoten` sowie Funktionen `einfuegen()` und `ausgeben()`, um einen duplikatfreien Binärbaum über `int` Werten gemäß den Testläufen zu realisieren.

Welche Komponenten die Datenstruktur `BaumKnoten` enthalten muss, sollen Sie anhand der Notwendigkeiten der Funktionen `einfuegen()` und `ausgeben()` entscheiden.

Verwenden Sie keine globalen oder `static` Variablen.

Deklarieren Sie die Datenstruktur `BaumKnoten` sowie die Funktions-prototypen für `einfuegen()` und `ausgeben()` in einer Headerdatei `binaerer_suchbaum.h`, und innerhalb eines Namespaces `suchbaum`.

Implementieren Sie die Funktionen `einfuegen()` und `ausgeben()` in einer Datei `binaerer_suchbaum.cpp`. Sie können gerne zusätzliche Hilfsfunktionen definieren, dann aber innerhalb des Namespaces `suchbaum`.
Die Ausgabefunktion `ausgeben()` rückt die Knotenwerte entsprechend ihrer Tiefe im Baum (d.h. Abstand vom Wurzelknoten) ein, mit zwei

Pluszeichen pro Tiefenstufe. Der Baum ist bei der textuellen Ausgabe „um 90 Grad gegen den Uhrzeigersinn gedreht" im Vergleich zur Diagrammdarstellung.
D.h. zu einem Baumknoten wird erst der rechte Teilbaum ausgegeben, dann der Wert des Knotens selbst, dann der linke Teilbaum.
Wegen der Selbstähnlichkeit (Teilbaum sieht von der Struktur aus wie der gesamte Baum): Realisieren Sie die Ausgabe über eine rekursive Funktion

```
void suchbaum::knoten_ausgeben(BaumKnoten* knoten, int tiefe);
```

… die aus der Funktion `ausgeben()` aufgerufen wird und genau das obige Ausgabeprinzip umsetzt (lassen Sie sich von der „Türme von Hanoi" Funktion inspirieren, falls nötig…).

Implementieren Sie die `main()` Funktion in einer Datei `suchbaum_main.cpp`.

## Testläufe (Benutzereingaben sind unterstrichen):

```
Leerer Baum.
Naechster Wert (99 beendet): ? 10
Naechster Wert (99 beendet): ? 4
Naechster Wert (99 beendet): ? 6
Naechster Wert (99 beendet): ? 15
Naechster Wert (99 beendet): ? 13
Naechster Wert (99 beendet): ? 12
Naechster Wert (99 beendet): ? 15
Naechster Wert (99 beendet): ? 20
Naechster Wert (99 beendet): ? 11
Naechster Wert (99 beendet): ? 15
Naechster Wert (99 beendet): ? 99
++++20
++15
++++13
++++++12
++++++++11
10
++++6
++4
Drücken Sie eine beliebige Taste . . .
```

```
Leerer Baum.
Naechster Wert (99 beendet): ? 3
Naechster Wert (99 beendet): ? 3
```

```
Naechster Wert (99 beendet): ? 3
Naechster Wert (99 beendet): ? 2
Naechster Wert (99 beendet): ? 99
3
++2
Drücken Sie eine beliebige Taste . . .
```

---

```
Leerer Baum.
Naechster Wert (99 beendet): ? 99
Leerer Baum.
Drücken Sie eine beliebige Taste . . .
```

---

## Aufgabe INF-10.02: Zeichenkette suchen, mit C-Strings

Schreiben Sie eine Headerdatei `suchen.h` sowie eine `.cpp` Datei `suchen.cpp` für eine C++ Funktion

```
int zeichenkette_suchen(const char* text, const char* zkette);
```

welche ermittelt, ob die Zeichenkette `zkette` in dem einzeiligen Text `text` (ggfs. mit Leerzeichen und/oder Satzzeichen) vorkommt.
Die maximale Textlänge von `text` und `zkette` betrage jeweils `20` vom Benutzer eingegebene Zeichen.
Der zu suchende Text sei nicht leer, d.h. enthalte außer dem Nullterminator noch mindestens ein Zeichen.
Der Text darf aber leer sein.
Der Benutzer mache nur korrekte Eingaben, d.h. ihr Programm kann davon ausgehen, dass z.B. die eingegebenen Zeichenketten nicht länger als 20 eingegebene Zeichen sind.

Sollte die Zeichenkette nicht in dem Text vorkommen, so soll der Wert `-1` zurückgegeben werden. Anderenfalls wir die Startposition zurückgegeben, ab der das erste (linkeste) Vorkommen von `zkette` in `text` beginnt. Die Zählung der Positionen im Text beginne bei `0`.

Die Funktion darf keinerlei C-String Funktionen aus Libraries wie der `cstring` Library oder C++ `string` benutzen, sondern ausschließlich über den Array-Indexoperator auf die einzelnen Zeichen der beiden C-Strings zugreifen. Sollten Sie für ihre Programmierung die Länge eines C-Strings benötigen, so müssen sie auch dafür eine eigene Hilfsfunktion programmieren, die nur den Array-Indexoperator für den Zugriff auf die einzelnen Zeichen des C-Strings benutzt. Beachten Sie, dass C-Strings

immer mit dem Null-Terminator enden und dass durch Vergleich mit `'\0'` das Ende eines C-Strings erkannt werden kann.

Legen Sie im Projekt eine leere Headerdatei `catch.h` an und kopieren Sie den Inhalt der in Ilias liegenden gleichnamigen Datei in diese Headerdatei.

Legen Sie im Projekt eine Datei `unit_tests.cpp` an mit folgendem Inhalt:

```
// Datei: unit_tests.cpp

#include "catch.h"
#include "suchen.h"

TEST_CASE("Zeichenkette suchen, Text mit Laenge groesser 1, Zeichenkette mit Laenge
groesser 1") {
    REQUIRE(zeichenkette_suchen("abcdabcde", "cda") == 2);
    REQUIRE(zeichenkette_suchen("abcdabcde", "de") == 7);
    REQUIRE(zeichenkette_suchen("abcdabcde", "dex") == -1);
    REQUIRE(zeichenkette_suchen("abcdabcde", "xyz") == -1);
    REQUIRE(zeichenkette_suchen("abcdabcde", "abcdabcd") == 0);
    REQUIRE(zeichenkette_suchen("abcdabcde", "abcdabcdx") == -1);
}

TEST_CASE("Zeichenkette suchen, Text mit Laenge groesser 1, Zeichenkette mit Laenge
1") {
    REQUIRE(zeichenkette_suchen("abcdabcde", "a") == 0);
    REQUIRE(zeichenkette_suchen("abcdabcde", "c") == 2);
    REQUIRE(zeichenkette_suchen("abcdabcde", "e") == 8);
    REQUIRE(zeichenkette_suchen("abcdabcde", "x") == -1);
}

TEST_CASE("Zeichenkette suchen, Text mit Laenge 1") {
    REQUIRE(zeichenkette_suchen("a", "a") == 0);
    REQUIRE(zeichenkette_suchen("a", "c") == -1);
    REQUIRE(zeichenkette_suchen("a", "xy") == -1);
    REQUIRE(zeichenkette_suchen("a", "aa") == -1);
}

TEST_CASE("Zeichenkette suchen, leerer Text") {
    REQUIRE(zeichenkette_suchen("", "") == -1);
    REQUIRE(zeichenkette_suchen("", "a") == -1);
    REQUIRE(zeichenkette_suchen("", "abc") == -1);
}

TEST_CASE("Zeichenkette suchen, Text mit Laenge 20 Zeichen") {
    REQUIRE(zeichenkette_suchen("abcdefghij1234567890", "90") == 18);
    REQUIRE(zeichenkette_suchen("12345678901234567890", "90") == 8);
    REQUIRE(zeichenkette_suchen("abcdefghij1234567890", "9012") == -1);
}

// Vorgegebene Testläufe müssen selbst als Testcases programmiert werden ...
TEST_CASE("Vorgegebene Testlaeufe") {
```

```
}
```

Ergänzen Sie die Unit Tests in dieser Datei um die Testfälle, die in den unten angegebenen Testläufen vorgegeben sind.

Schreiben Sie ferner ein C++ Hauptprogramm, welches die Eingaben entgegennimmt, die Funktion aufruft und das Ergebnis ausgibt (siehe Testläufe). Ergänzen Sie dabei den folgenden Programmrahmen:

```
// Datei: main.cpp

#define CATCH_CONFIG_RUNNER
#include "catch.h"

#include <iostream>
using namespace std;

#include "suchen.h"

int main()
{
    if ( Catch::Session().run() ) {
        system("PAUSE"); return 1;
    }

    // Ihr Code ab hier ...


    system("PAUSE");
    return 0;
}
```

*Laden Sie ihre Lösung auch nach Jenkins hoch und stellen Sie sicher, dass ihr Programm dort auch alle Tests besteht. Das Hochladen nach Jenkins ist **Pflicht** und das Jenkins Ergebnis muss im Praktikum vorgezeigt werden! Es werden die gleichen Testfälle getestet, die auch hier im Aufgabenblatt angegeben sind, inklusive der Testläufe. Der C++ Compiler, der im Jenkins verwendet wird, generiert aber wesentlich „genaueren" Code: Wenn Sie dort bei C-Strings oder bei Arrays über den Rand hinausgreifen, führt dieser Programmierfehler mit dem C++ Compiler des Jenkins Systems unweigerlich zu einem Programmfehler, während einige solche Fehler beim Visual Studio Compiler gar nicht auffallen, weil der generierte Maschinencode viel „ungenauer" ist. Dies ist dann aber kein Fehler des Jenkins-Compilers, sondern deckt nur Programmierfehler ihrerseits viel gnadenloser auf …*
*Wenn ihr Programm also unter Visual Studio richtig zu funktionieren scheint, bedeutet dies noch nicht unbedingt, dass das Programm auch korrekt ist. Erst wenn das Jenkins System ihre Lösung auch akzeptiert, ist soweit alles o.k. …*

Testläufe: *(Benutzereingaben sind zur Verdeutlichung unterstrichen)*

```
================================================================================
All tests passed (27 assertions in 6 test cases)

Bitte geben Sie den Text ein: abcdefg
Bitte geben Sie die zu suchende Zeichenkette ein: bcd99
Die Zeichenkette 'bcd99' ist NICHT in dem Text 'abcdefg' enthalten.
Drücken Sie eine beliebige Taste . . .
```

```
================================================================================
All tests passed (27 assertions in 6 test cases)

Bitte geben Sie den Text ein: abcdefg
Bitte geben Sie die zu suchende Zeichenkette ein: efg
Die Zeichenkette 'efg' ist in dem Text 'abcdefg' enthalten.
Sie startet ab Zeichen 4 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

```
================================================================================
All tests passed (27 assertions in 6 test cases)

Bitte geben Sie den Text ein: abc
Bitte geben Sie die zu suchende Zeichenkette ein: abcde
Die Zeichenkette 'abcde' ist NICHT in dem Text 'abc' enthalten.
Drücken Sie eine beliebige Taste . . .
```

```
================================================================================
All tests passed (27 assertions in 6 test cases)

Bitte geben Sie den Text ein: 012 abc abc 89
Bitte geben Sie die zu suchende Zeichenkette ein: abc
Die Zeichenkette 'abc' ist in dem Text '012 abc abc 89' enthalten.
Sie startet ab Zeichen 4 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

```
================================================================================
All tests passed (27 assertions in 6 test cases)

Bitte geben Sie den Text ein: xy abc abcdefgh
Bitte geben Sie die zu suchende Zeichenkette ein: abcde
Die Zeichenkette 'abcde' ist in dem Text 'xy abc abcdefgh'
enthalten.
Sie startet ab Zeichen 7 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

```
================================================================================
All tests passed (27 assertions in 6 test cases)

Bitte geben Sie den Text ein: xyz 123 456 abc
Bitte geben Sie die zu suchende Zeichenkette ein: 123 4
Die Zeichenkette '123 4' ist in dem Text 'xyz 123 456 abc'
enthalten.
Sie startet ab Zeichen 4 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

```
================================================================================
All tests passed (27 assertions in 6 test cases)

Bitte geben Sie den Text ein: abc defg
Bitte geben Sie die zu suchende Zeichenkette ein: abc d
Die Zeichenkette 'abc d' ist in dem Text 'abc defg' enthalten.
Sie startet ab Zeichen 0 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```