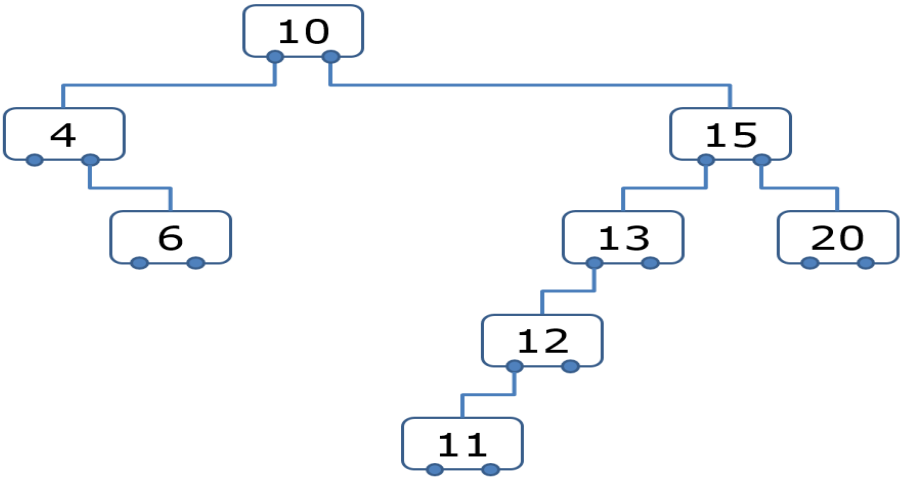
Praktikumstermin Nr. 09, INF
Dynamische Datenstrukturen - Binärer Suchbaum

Abgabe im GIP-INF Praktikum der Woche 5.12.-9.12.2022.

(Pflicht-) Aufgabe INF-09.01: 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 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 (***noch keine Klassen nutzen / definieren!***) eine `struct` Datenstruktur `BaumKnoten` mit einem Attribut `int data` sowie zwei Attributen `BaumKnoten* links` und `BaumKnoten* rechts`. Programmieren Sie ferner zwei Funktionen `einfuegen()` und `ausgeben()`, um einen duplikatfreien Binärbaum über `int` Werten gemäß den Testläufen zu realisieren. Die Funktionen sollen den `anker`, d.h. einen Pointer auf den Wurzelknoten des Baums, als Parameter nehmen. Die Funktion `einfuegen()` soll außerdem einen `int` Wert `wert` als zweiten Parameter nehmen, der dann in den Baum eingefügt werden soll (falls noch nicht dort vorhanden).

Verwenden Sie keine globalen oder `static` Variablen.

Deklarieren Sie die `struct` Datenstruktur `BaumKnoten` sowie die Funktionsprototypen 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,
                               unsigned 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…).

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

Legen Sie im Projekt eine Datei `test_binaerer_suchbaum.cpp` an und kopieren Sie den Inhalt der in Ilias gegebenen gleichnamigen Datei in diese Datei.

Legen Sie im Projekt eine Datei `suchbaum_main.cpp` an und kopieren Sie den Inhalt der in Ilias gegebenen gleichnamigen Datei in diese Datei.

Testläufe (Benutzereingaben sind unterstrichen):

```
Alle Tests erfolgreich (71 REQUIREs in 7 Test Cases)
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 . . .
```

```
Alle Tests erfolgreich (71 REQUIREs in 7 Test Cases)
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 . . .
```

```
Alle Tests erfolgreich (71 REQUIREs in 7 Test Cases)
Leerer Baum.
Naechster Wert (99 beendet): ? 99
Leerer Baum.
Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Aufgabe INF-09.02: Binärer Suchbaum mittels Objekt-Orientierter Programmierung

Legen Sie für diese Aufgabe in Visual Studio Code ein neues Projekt / Verzeichnis an. Ihre Lösung zur vorigen Aufgabe muss erhalten und unverändert bleiben, da Sie diese auch vorzeigen müssen!

Im Rahmen dieser Aufgabe sollen keine `namespaces` verwendet werden.

Programmieren Sie in diesem neuen Projekt in einer Datei `BaumKnoten.h` die Klasse `BaumKnoten`. Jedes `BaumKnoten` Objekt soll in seinem Attribut `data` einen `int` Wert als Nutzdatenwert speichern können. Außerdem sollen in zwei Attributen `BaumKnoten* links` und `BaumKnoten* rechts` jeweils Pointer auf mögliche Kindknoten gespeichert werden (oder der `nullptr` Wert, falls kein entsprechender Kindknoten existiert). Alle diese Attribute sollen von außerhalb der Klasse nicht sichtbar sein.

Programmieren Sie die entsprechenden Getter und Setter für diese Attribute. Benennen Sie die Getter und Setter mit

`get_<attributname>()` und `set_<attributname>()`, also z.B. `get_data()` und `set_data()`. Die Getter und Setter sollen natürlich von außerhalb der Klasse aufrufbar sein.

Programmieren Sie für die Klasse `BaumKnoten` einen Konstruktor, der einen `int` und zwei `BaumKnoten*` Parameter hat und diese drei Parameterwerte mittels Initialisierungsliste in die entsprechenden Attribute des Objekts überträgt.

Die Klasse `BaumKnoten` besitze ferner eine von außerhalb aufrufbare Methode `ausgeben()`, welche einen Parameter `unsigned int tiefe` habe. Im Rahmen der Klassendefinition in der Headerdatei soll der Prototyp dieser Methode definiert werden. Die Methode an sich soll in einer Datei `BaumKnoten.cpp` programmiert werden.

Programmieren Sie dann in einer Datei `BinaererSuchbaum.h` die Klasse `BinaererSuchbaum`.

Objekte dieser Klasse sollen ein von außen nicht sichtbares Attribut `root` vom Typ `BaumKnoten*` haben, welches auf den Wurzelknoten des Baumes verweist bzw. den `nullptr` Wert hat, wenn der Baum leer ist.

Programmieren Sie für das Attribut `root` einen Getter `get_root()`.

Der Konstruktor der Klasse sei parameterlos.

Die Klasse habe zwei von außen aufrufbare Methoden `einfuegen()` und `ausgeben()`. Die Methode `ausgeben()` sei parameterlos, während `einfuegen()` einen `int` Wert als Parameter nimmt, der dann in den Baum eingefügt wird (falls dort nicht schon vorhanden). Programmieren Sie die beiden Methoden in einer Datei `BinaererSuchbaum.cpp` und geben Sie in der Headerdatei (innerhalb der Klassendefinition) nur den Prototypen dieser Methoden an.

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

Legen Sie im Projekt eine Datei `test_binaerer_suchbaum_klasse.cpp` an und kopieren Sie den Inhalt der in Ilias gegebenen gleichnamigen Datei in diese Datei.

Legen Sie im Projekt eine Datei `suchbaum_klasse_main.cpp` an und kopieren Sie den Inhalt der in Ilias gegebenen gleichnamigen Datei in diese Datei.

Testläufe (Benutzereingaben sind unterstrichen):

```
Alle Tests erfolgreich (71 REQUIREs in 7 Test Cases)
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 . . .
```

```
Alle Tests erfolgreich (71 REQUIREs in 7 Test Cases)
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 . . .
```

```
Alle Tests erfolgreich (71 REQUIREs in 7 Test Cases)
Leerer Baum.
Naechster Wert (99 beendet): ? 99
Leerer Baum.
Drücken Sie eine beliebige Taste . . .
```