PRG1x & ADE1x	**Einf. i. d. Programmierung (int. LVA)** **Üb. zu Element. Alg. u. Datenstrukt.**	**WS 13/14, Übung 9**

Abgabetermin: Sa, 11.01.2014

☐ **Gr. 1,** DI (FH) G. Horn, MSc

☐ **Gr. 2,** J.-P. Haslinger, MSc **Name** _____ **Aufwand in h** _____

Punkte _____ **Kurzzeichen Tutor / Übungsleiter** _____ / _____

1. Arbeitszeitaufzeichnungen und Bäume (18 Punkte)

In Übung 7 haben Sie eine Datenstruktur zur Verwaltung Arbeitszeitaufzeichnungen entwickelt. Implementieren Sie diese nun ein weiteres Mal erneut, allerdings nun unter Verwendung eines binären Suchbaums anstatt einer doppelt verketten zyklischen Liste zur Speicherung der Personen. Die Personen sollen alphabetisch sortiert nach ihren Namen im Baum gespeichert werden. Die Zeiterfassungseinträge pro Person bleiben wie gehabt in einfach verketteten linearen Listen, die nun an den einzelnen Baum-Knoten hängen. Versuchen Sie aber nun, diese Listen beim Aufbau nach einem sinnvollen Kriterium (z. B. Name der Aufgabe) zu ordnen, und beschreiben Sie, ob sich dadurch ein Vorteil für die Implementierung ergibt.

Implementieren Sie mindestens die folgenden Prozeduren und Funktionen:

```
PROCEDURE AddWorkEntry(…);
PROCEDURE GetTotalWorkTimeForPerson(…);
PROCEDURE GetAverageWorkTimeForTask(…);
PROCEDURE PrintPersonsForTask(…);
PROCEDURE PrintWorkSummaryForPerson(…);
PROCEDURE Reset;
FUNCTION BusiestPerson: STRING;
FUNCTION TotalWorkEntryCount: INTEGER;
```

Eine Operation zum Löschen einer Person mit allen ihren Einträgen ist diesmal nicht zwingend erforderlich.

Geben Sie, wenn möglich bzw. sinnvoll, iterative Implementierungen für Ihre Prozeduren und Funktionen an oder begründen Sie von Fall zu Fall, warum dies nicht (einfach) möglich ist.

2. Ausbalancieren von Binärbäumen (6 Punkte)

Operationen auf einem Binärbaum (Einfügen, Suchen etc.) sind nur dann wirklich effizient, wenn dieser ausbalanciert ist. In der Praxis lässt sich nun aber selten beeinflussen, in welcher Reihenfolge die in den Baum einzufügenden Daten auftreten, entstehen meist sehr schlecht ausbalancierte Binärbäume, wodurch die Performance des Programms stark leidet.

Um dem entgegenzuwirken, macht es Sinn, den Binärbaum bereits während(!) dem Aufbauen in regelmäßigen Abständen immer wieder auszubalancieren, damit so immer mit einem möglichst ausbalancierten Baum gearbeitet wird.

Ein Algorithmus zu Ausbalancieren könnte dabei so aussehen:

1. Lege ein dynamisches Feld (siehe `dynarray.pas` im Moodle) von Zeigern auf Knoten mit genau so vielen Elementen wie Knoten im Baum an.
2. Laufe über alle Knoten im Baum und übertrage sie als Zeiger sortiert in das Feld.
3. Baue den Baum neu auf: Das mittlere Element des Felds wird dabei zum Wurzelknoten, das mittlere Element der linken Hälfte der linke Sohn des Wurzelknotens, das mittlere Element der rechten Hälfte der rechte Sohn usw.
4. Gib nach Aufbau des neuen Baumes das dynamisch allokierte Feld frei.

Implementieren Sie basierend auf dieser Idee eine entsprechende Prozedur `BalanceTree` und bauen Sie das Ausbalancieren des Suchbaumes sinnvoll(!) in ihre Lösung des ersten Beispiels ein. Testen Sie mit einer entsprechend großen Datenmenge und dokumentieren Sie Ihre Erkenntnisse.