UNIVERSITY of BRADFORD

School of Electrical Engineering & Computer Science

Coursework Assignment Part 1: Abstract Data Types: Lists, Trees, Heaps

Date Issued Semester 1, 8:00 Monday 3rd October 2016 (week 3)

Date Due Semester 1, 17:00 Monday 9th January 2017

Each student is provided with a (different by content) list of numbers to be used, in sequence, as keys for entries into the Abstract Data Type in the exercise below, as opted by the student at the start of the report.

Your assignment is to show the trace of the operations of insertion and re-structuring as applicable and requested, involved for your list of keys when they are entered into the ADT structure of your choice (list, heap, tree) as detailed below.

Form of Submission

Your submission should be in plain text files uploaded in Blackboard module relevant assessment section.

The content of your submitted files should be as follows:

1. The results file has your UoB in the first line and the ADT choice (from LIST, TREE, HEAP) in the second line, e.g.:

```
14012345

LIST
```

2. The word 'data' and your assigned list of keys, e.g.

```
data 25, 10, 15, ...
```

You will find your keys against your name in the list provided online on Blackboard. It is recommended that you copy and paste the whole line on to your submission document. When you paste the line it will be noticeable if it shows someone else's name. You can then safely delete your name from the copy, knowing you have the right data.

3. One section from:

TPO

A. Singly linked LIST
B. AVL TREE
C. Binary HEAP

as detailed below.

4. Your full running code (Java files) that implements your results for exercise 3 above.

5. Nothing else! For example, although you might well find it convenient to draw the various trees as you construct them, do not include such diagrams in your submission.

The section 3 above will contain the results to one of the following problems you will choose from the following:

3A. Doubly linked LIST

The task here is to show a trace of the operations needed to insert objects with your (list of) keys, one by one, into an initially empty doubly-linked list such as the result is an ascendingly sorted list by the keys values.

Your submission should have the section heading 'LIST trace' followed by the coded trace of operations:

• `Ixx` to **I**nsert key xx at the root of the previously empty list;

• `IxxAyyBzz` to **I**nsert key xx **A**fter node containing key yy and **B**efore node containing key zz;

• `IxxAyy` to **I**nsert key xx **A**fter tail node containing key yy;

• `IxxByy` to **I**nsert key xx **B**efore head node containing key yy;

with the coded operations in sequence on successive lines, e.g.

```
LIST trace I25 I10B25 I15A10B25 I30A25 I40A30 …
```

3B. AVL TREE

The task here is to show a trace of the operations needed to insert objects with your (list of) keys, one by one, into an initially empty AVL tree with restoration of AVL balance (if necessary) after each insertion.

Your submission should have the section heading 'AVL trace' followed by the coded trace of operations:

- `Ixx` **to insert key xx at the root of the previously empty AVL tree;**
- `IxxLyy` **to insert key xx as the left child of the node containing key yy;**
- `IxxRyy` **to insert key xx as the right child of the node containing key yy;**
- `Rxx` **to rotate the node containing key xx with that of its parent (in order to restore AVL balance);**

with the coded operations in sequence on successive lines, e.g.

```
AVL trace I25 I10L25 I15R10 R15 R15 …
```

3C. Binary HEAP

The task here is to show a trace of the operations needed to insert objects with your (list of) keys, one by one, into an initially empty minHeap with restoration of heap order (if necessary) with each insertion.

Your submission should have the section heading 'HEAP trace' followed by the coded trace of operations:

- `Hx` **to create a hole at location x in the heap;**
- `X` **to move the hole up the heap by swapping the hole with its parent;**
- `Ixx` **to insert key xx into the hole;**

with the coded operations in sequence on successive lines, e.g.

```
Heap trace H1 I25 H2 X I10 H3 I15 ...
```