ECE368 Project #3

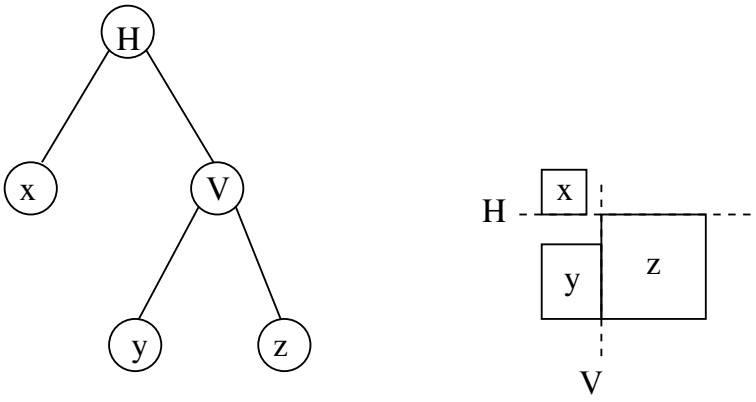
Due Tuesday, March 25, 2014, 11:59pm

Description

This project is to be completed on your own. You will implement a program involving tree traversal(s) to compute the packing of rectangles, represented by a binary tree.

In this binary tree, each leaf node represents a rectangle. Each internal node of the binary tree represents a partitioning of two groups of rectangles by a horizontal cutline or a vertical cutline. Let xHy (xVy) denote a (sub)tree, whose root node is a horizonal cut H (a vertical cut V). The left and right subtrees of H (V) are x and y, respectively. Assume that xHy means x is on top of and y is below the horizontal cut, and xVy means x is to the left and y is to the right of the vertical cut.

In the following figure, we show an example of a "packing" of three rectangles based on a given binary tree representation. Here, each subtree is enclosed by a smallest rectangular room. Assume that the dimensions (width, height) of the three rectangles x, y, and z are $(3,3)$, $(4,5)$, and $(7,7)$. The smallest room containing the subtree yVz is of dimensions $(11,7)$.



(a) A binary tree (b) The corresponding packing

The smallest room containing the subtree $xH(yVz)$ is of dimensions $(11,10)$. This room is partitioned in to two smaller rooms: The top room is of dimensions $(11,3)$ and it contains rectangle x, whereas the bottom room is of dimensions $(11,7)$ and it contains rectangles y and z. We place the lower left corner of each rectangle to the lower left corner of its room.

Assuming that the lower left corner of the smallest room containing all the blocks is at coordinates $(0,0)$, the coordinates of lower left corners of the three rectangles x, y, and z are respectively $(0,7)$, $(0,0)$ and $(4,0)$. Note that even though there is space above y to accommodate x, x has to stay above the horizontal cutline in the packing, as shown in the figure.

Given a binary tree representation, the smallest rectangular room to enclose all rectangles and the coordinates of these rectangles in the corresponding packing can be computed in $O(n)$ time complexity using appropriate tree-traversal algorithm(s).

Deliverables:

In this project, you are to develop your own include file `packing.h` and source file `packing.c`, which can be compiled with the following command:

```
gcc -Werror -Wbad-function-cast -Wall -Wshadow -O3 packing.c -o proj3
```

All declarations and definition of data structures and functions must reside in the include file or source file. The executable `proj3` would be invoked as follows:

```
proj3 input_file output_file
```

The executable loads the binary tree from `input_file`, performs packing, and saves the packing into `output_file`.

Your source code should contain the following three functions (you may also need other functions):

1. Load binary tree from file

The input filename (including path) is provided in the command line. The binary tree contained in the file should be read in, parsed, and stored in the tree data structures defined by you. The input file is divided into two sections. The first section contains information that should help you to construct your binary tree:

- `Number of blocks` (int), which specifies the number of rectangles.
- `Number of nodes` (int), which specifies the number of nodes in the binary tree.

The second section specifies the topology of the binary tree. Every line contains seven fields:

- `thisnode` (integer), which specifies the node number of current node.
- `parnode` (integer), which specifies the node number of the parent of current node. A node number of "-1" indicates the absence of a parent node, i.e., the current node is the root node of the tree.
- `lcnode` (integer), which specifies the node number of the left child node. A node number of "-1" indicates the absence of a left child node.
- `rcnode` (integer), which specifies the node number of the right child node. A node number of "-1" indicates the absence of a right child node.
- `cutline` (char), which specifies the orientation of the cutline. For a leaf node, the cutline orientation is specified with '-'. For an internal node, we use 'H' to denote a horizontal cutline and 'V' to denote a vertical cutline.
- `width` (double), the width of the rectangle if the node is a leaf node. We use '-' for an internal node.

- `height` (double), the height of the rectangle if the node is a leaf node. We use '-' for an internal node.

Note that the list of nodes starts with all the sink (leaf) nodes.

2. Perform packing

Perform packing on the binary tree you have loaded in, using data type double for your computation of coordinates. At the end of the analysis, the program should report the following:

```
Preorder:

Inorder:

Postorder:

Width:
Height:

X-coordinate:
Y-coordinate:

Elapsed time:
```

Here, `Preorder`, `Inorder`, and `Postorder` should print the node numbers as the tree is traversed in preorder, inorder, and postorder fashion. The elapsed time refers to the user time used by the packing function (see project 1). Please do not include the time it takes to load and save. For `Width` and `Height`, you should report the width and height of the smallest room that encloses the packing specified by the binary tree. For `X-coordinate` and `Y-coordinate`, you should report the coordinates of the rectangle with the largest node number.

Note that you do not really pack the rectangles tightly, as shown in Figure (b).

3. Save packing to file

The output filename (including path) is provided in the command line. The first line should specify the number of rectangles.

- `Number of blocks` (int), which specifies the number of rectangles.

Subsequently, the file should contain a line for each rectangle, starting from the lowest indexed rectanlge: Every line contains five fields:

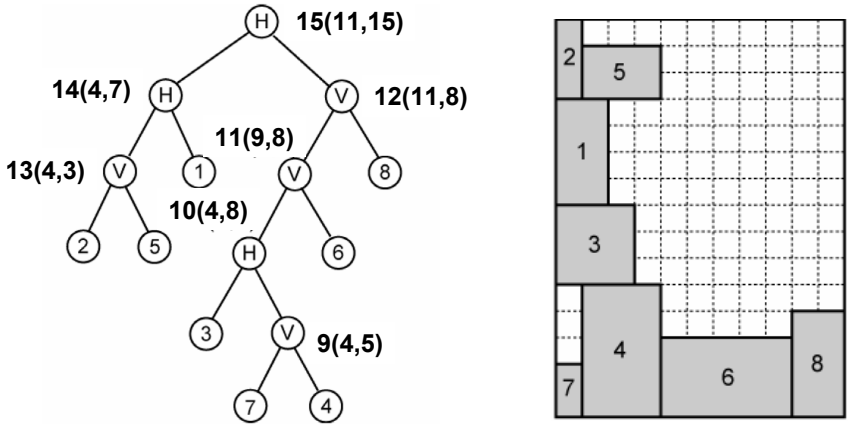
- `thisnode` (integer), which specifies the node number of current node.
- `width` (double), the width of the rectangle.
- `height` (double), the height of the rectangle.

- `xcoord` (double), the x-coordinate of the rectangle.
- `ycoord` (double), the y-coordinate of the rectangle.

Consider the following input file:

```
8
15
1 14 -1 -1 - 2 4
2 13 -1 -1 - 1 3
3 10 -1 -1 - 3 3
4 9 -1 -1 - 3 5
5 13 -1 -1 - 3 2
6 11 -1 -1 - 5 3
7 9 -1 -1 - 1 2
8 12 -1 -1 - 2 4
9 10 7 4 V - -
10 11 3 9 H - -
11 12 10 6 V - -
12 15 11 8 V - -
13 14 2 5 V - -
14 15 13 1 H - -
15 -1 14 12 H - -
```

The following figure shows the corresponding binary tree and packing. The numbers in parentheses besides each internal node denote the width and height of the smallest room enclosing the rectangles in the subtrees of this node. Therefore, the width and the height of the smallest room enclosing all rectangles are 11 and 15. The coordinates of the largest indexed node are $(9,0)$.



The output file should have the following format:

```
8
1 2.000000e+00 4.000000e+00 0.000000e+00 8.000000e+00
2 1.000000e+00 3.000000e+00 0.000000e+00 1.200000e+01
3 3.000000e+00 3.000000e+00 0.000000e+00 5.000000e+00
4 3.000000e+00 5.000000e+00 1.000000e+00 0.000000e+00
5 3.000000e+00 2.000000e+00 1.000000e+00 1.200000e+01
6 5.000000e+00 3.000000e+00 4.000000e+00 0.000000e+00
7 1.000000e+00 2.000000e+00 0.000000e+00 0.000000e+00
8 2.000000e+00 4.000000e+00 9.000000e+00 0.000000e+00
```

The screen dump should show the following:

```
Preorder:  15 14 13 2 5 1 12 11 10 3 9 7 4 6 8

Inorder:  2 13 5 14 1 15 3 10 7 9 4 11 6 12 8

Postorder:  2 5 13 1 14 3 7 4 9 10 6 11 8 12 15

Width:  1.100000e+01
Height:  1.500000e+01

X-coordinate:  9.000000e+00
Y-coordinate:  0.000000e+00

Elapsed Time:  0.000000e+00
```

Grading:

The project requires the submission (through Blackboard) of the C-code (source and include files) and a report detailing the results. There should a table listing the run-times, and the outputs of the packing (width, height, and x- and y-coordinates of the largest indexed rectangles) obtained from running your code on some sample input files. In your report, also comment on the run-time and space complexity of your algorithm. Your report should not be longer than 1 page and should be in plain text format or pdf.

Getting started:

We provide sample input files for you to evaluate the run-times of your packing program. We also provide the sample output and the screen output for the 8-rectangle example: `r6.pck` and `r6.log`, respectively. The input of this example is `r6_flr.txt`. Copy over the files from the Blackboard website. Check out the Blackboard website for any updates to these instructions. *Start packing!*