ECE368 Programming Assignment #3

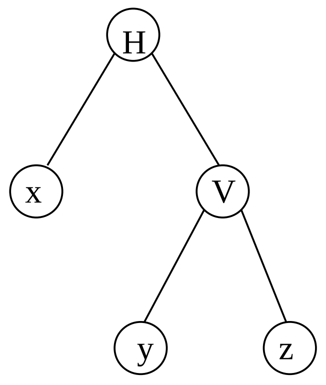
Due Monday, March 5, 2018, 11:59pm

Description

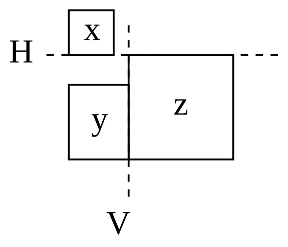
This project is to be completed on your own. You will implement a program involving tree traversal(s) to compute the "2-Dimensional (2D) packing" of rectangles, represented by a binary tree.

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 horizontal 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` that defines the structures you want to use and to declare the functions you need to manipulate the structures. You should define these functions in the source file `packing.c` (and declare their prototypes in `packing.h`). These functions should be called by a `main` function that resides in `packing_main.c`.

Your programs should be compiled with the following command:

```
gcc -std=c99 -pedantic -Wvla -Wall -Wshadow -g packing.c packing_main.c -o
proj3
```

If you supply a makefile, we will use your makefile to compile your program. The executable `proj3` would be invoked as follows:

```
./proj3 input_file output_file1 output_file2 output_file3
```

The executable loads the binary tree from `input_file`, performs packing, and saves the results in three different output files, namely, `output_file1`, `output_file2`, and `output_file3`.

Your packing.c file should contain the following functions (you may also need other functions):

1. Load binary tree from file

The input filename 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 lines, and each line corresponds to a node in the binary tree. If it is a leaf node (which is a real rectangle), it has been printed with the format `"%d(%le,%le)\n"`, where the `int` is the label of the rectangle, the first `double` is the width of the rectangle, and the second `double` is the height of the rectangle.

If there are *n* rectangles in the packing, the labels are from 1 through *n*.

If it is non-leaf node, it is simply a character (followed by a newline character). The character is either `'V'` or `'H'`, representing either a vertical cutline or a horizontal cutline, respectively.

These nodes are printed in a preorder traversal of the binary tree. In this preorder traversal, the left branch is visited before the right branch.

2. Print the binary tree in postorder fashion

This function prints the constructed binary tree into an output file; the filename is provided in the command line (`output_file1`). The format for a leaf node (rectangle) is the same as in the input file. The format for a non-leaf node is also the same as in the input file. The only difference is that the ordering of the nodes in the output file is determined by a postorder traversal of the binary tree. In this postorder traversal, the left branch is visited before the right branch.

3. Perform packing

Perform packing on the binary tree you have loaded in, using data type `double` for your computation of dimensions and coordinates. Note that you do not really pack the rectangles tightly, as shown in Figure (b).

4. Save dimensions to file

This function prints the dimensions of all leaf and non-leaf nodes into an output file; the filename is provided in the command line (`output_file2`). The format for a leaf node (rectangle) is the same as in the input file. The format for a non-leaf node is of the format `"V(%le,%le)\n"` for a node that is a vertical cut and of the format `"H(%le,%le)\n"` for a node that is a horizontal cut. The width and height of a non-leaf node correspond to the dimensions of the smallest rectangular room containing all rectangles "packed" in the sub-tree rooted at the non-leaf node. The ordering of the nodes in this file should be the same as the ordering in the input file. In other words, the ordering is determined by a preorder traversal of the binary tree.

5. Save coordinates to file

The output filename (`output_file3`) is provided in the command line. The file should contain a line for each rectangle. The ordering of the rectangles in the output file should be the same as the ordering of rectangles in the input file. Every line should be printed with the format the format `"%d(%le,%le)\n"`, where the `int` is the label of the rectangle. The two `double`'s are the *x*- and *y*-coordinates of the rectangle.

Your `main` function in the `packing_main.c` file should call the aforementioned five functions in the correct order.

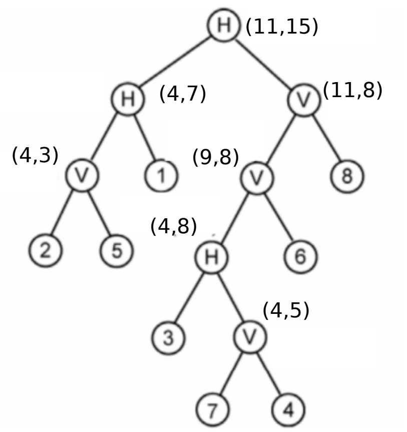
Consider the following input file:

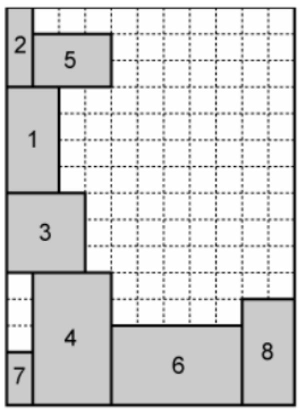
```
H
H
V
2(1.000000e+00,3.000000e+00)
5(3.000000e+00,2.000000e+00)
1(2.000000e+00,4.000000e+00)
V
V
H
3(3.000000e+00,3.000000e+00)
V
7(1.000000e+00,2.000000e+00)
4(3.000000e+00,5.000000e+00)
6(5.000000e+00,3.000000e+00)
8(2.000000e+00,4.000000e+00)
```

The following figure shows the corresponding binary tree and packing. The numbers in parentheses next to each internal node (i.e., non-leaf node) denote the width and height of the smallest room enclosing the rectangles in the subtree of this node.

The first output file `output_file1` should have the following format:

```
2(1.000000e+00,3.000000e+00)
```





```
5(3.000000e+00,2.000000e+00)
V
1(2.000000e+00,4.000000e+00)
H
3(3.000000e+00,3.000000e+00)
7(1.000000e+00,2.000000e+00)
4(3.000000e+00,5.000000e+00)
V
H
6(5.000000e+00,3.000000e+00)
V
8(2.000000e+00,4.000000e+00)
V
H
```

The second output file `output_file2` should have the following format:

```
H(1.100000e+01,1.500000e+01)
H(4.000000e+00,7.000000e+00)
V(4.000000e+00,3.000000e+00)
2(1.000000e+00,3.000000e+00)
5(3.000000e+00,2.000000e+00)
1(2.000000e+00,4.000000e+00)
V(1.100000e+01,8.000000e+00)
V(9.000000e+00,8.000000e+00)
H(4.000000e+00,8.000000e+00)
3(3.000000e+00,3.000000e+00)
V(4.000000e+00,5.000000e+00)
7(1.000000e+00,2.000000e+00)
4(3.000000e+00,5.000000e+00)
```

```
6(5.000000e+00,3.000000e+00)
8(2.000000e+00,4.000000e+00)
```

The third output file `output_file3` should have the following format:

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

Submission:

The project requires the submission (through Blackboard) of the C-code (source and include files), and a report explaining the time complexity of constructing a binary tree from an input file and the time complexity of computing the coordinates of the rectangles for the given binary tree. Your report should not be longer than 1 page and should be in plain text format or pdf. You should create a zip file called proj3.zip that contains all of the above and submit the zip file. If the zip file contains a `makefile`, we will use that file to make your executable.

Grading:

The grade depends on the correctness of your program, the efficiency of your program, the clarity of your program documentation and report.

The report will account for 10% of the entire grade. The entire program will account for 90% of the entire grade. We do not test individual functions because you have the flexibility to design the structures required for this assignment. The correctness of the first output file will account for 25% of the entire grade. The correctness of the second output file will account for 30% of the entire grade. The correctness of the third output file will account for 35% of the entire grade.

It is important all the files that have been opened are closed and all the memory that have been allocated are freed before the program exits. Memory issues will result in 40% penalty.

Getting started:

We provide two sample input files (`r3.pr` and `r8.pr`) for you to evaluate the run-times of your packing program. `r3.pr` is the 3-rectangle example and `r8.pr` is the 8-rectangle example. In the 3-rectangle example, x has label 3, y has label 1, and z has label 2.

We also provide the sample output files for both examples: `r3.po`, `r3.dim`, and `r3.co` are the first, second, and third output files for `r3.pr`. `r8.po`, `r8.dim`, and `r8.co` are the first, second, and third output files for `r8.pr`.

Copy over the files from the Blackboard website. Check out the Blackboard website for any updates to these instructions.

Start packing (because spring is almost here)!