CSCI 1120 (Low-Level Computing), Fall 2015

Homework 7

Credit: 20 points.

1 Reading

Be sure you have read the assigned readings for classes through 11/18.

2 Programming Problems

Do the following programming problems. You will end up with at least one code file per problem. Submit your program source (and any other needed files) by sending mail to `bmassing@cs.trinity.edu`, with each file as an attachment. Please use a subject line that mentions the course number and the assignment (e.g., "csci 1120 homework 7"). You can develop your programs on any system that provides the needed functionality, but I will test them on one of the department's Linux machines, so you should probably make sure they work in that environment before turning them in.

1. (20 points) Your mission for this assignment is to complete a partial implementation in C of a binary search tree (a.k.a. sorted binary tree) of `int`s. (I'm hoping that all of you know about this data structure from CS2. If you don't — the Wikipedia article[1] is a reasonable description (but I recommend that you not read the example code until/unless you try to write your own).

   This partial implementation consists of a number of files:

   - Function declarations for tree: int-bst.h[2].
   - Starter file for function definitions: int-bst.c[3].
   - Test program and supporting files: test-int-bst.c[4], test-helper.c[5], test-helper.h[6].
   - Makefile for compiling (comments in the file tell you how to use it): Makefile.bst[7]. *NOTE* that you should use your browser's "download" or "save" function to obtain this file, rather than copying and pasting the text. This is because copy-and-paste will likely replace the tab characters in the file with spaces, with bad consequences (since tabs are semantically significant in makefiles.)

[1] `http://en.wikipedia.org/wiki/Binary_search_tree`

[2] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2015fall/Homeworks/HW07/Problems/int-bst.h`

[3] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2015fall/Homeworks/HW07/Problems/int-bst.c`

[4] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2015fall/Homeworks/HW07/Problems/test-int-bst.c`

[5] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2015fall/Homeworks/HW07/Problems/test-helper.c`

[6] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2015fall/Homeworks/HW07/Problems/test-helper.h`

[7] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2015fall/Homeworks/HW07/Problems/Makefile.bst`

Your job is to modify the file `int-bst.c` so it includes function definitions for all the functions declared in `int-bst.h`. (You may want to add additional "helper" functions, but if so they should probably go only in `int-bst.c`.) Notice that the function that removes a single element of the tree (`int_bst_remove`) is optional — you can provide an "implementation" that just prints an error message, or for extra credit you can actually implement this operation. You should not need to modify any other files, unless you want to add additional tests to `test-int-bst.c`.

Sample output of the test program:

```
inserting 40 into tree [ ]
result [ 40 ]
inserting 30 into tree [ 40 ]
result [ 30 40 ]
inserting 50 into tree [ 30 40 ]
result [ 30 40 50 ]
inserting 20 into tree [ 30 40 50 ]
result [ 20 30 40 50 ]
inserting 60 into tree [ 20 30 40 50 ]
result [ 20 30 40 50 60 ]
inserting 16 into tree [ 20 30 40 50 60 ]
result [ 16 20 30 40 50 60 ]
inserting 14 into tree [ 16 20 30 40 50 60 ]
result [ 14 16 20 30 40 50 60 ]
inserting 18 into tree [ 14 16 20 30 40 50 60 ]
result [ 14 16 18 20 30 40 50 60 ]
inserting 24 into tree [ 14 16 18 20 30 40 50 60 ]
result [ 14 16 18 20 24 30 40 50 60 ]
inserting 56 into tree [ 14 16 18 20 24 30 40 50 60 ]
result [ 14 16 18 20 24 30 40 50 56 60 ]
inserting 64 into tree [ 14 16 18 20 24 30 40 50 56 60 ]
result [ 14 16 18 20 24 30 40 50 56 60 64 ]
inserting 30 into tree [ 14 16 18 20 24 30 40 50 56 60 64 ]
result [ 14 16 18 20 24 30 40 50 56 60 64 ]
inserting 50 into tree [ 14 16 18 20 24 30 40 50 56 60 64 ]
result [ 14 16 18 20 24 30 40 50 56 60 64 ]
test data in order [ 14 16 18 20 24 30 30 40 50 50 56 60 64 ]
40
  30
      20
          16
              14
                  .
                  .
              18
                  .
                  .
          24
              .
```

```
                .
        .
    50
        .
        60
            56
                .
                .
            64
                .
                .
finding 0 in tree [ 14 16 18 20 24 30 40 50 56 60 64 ]
result false
finding 100 in tree [ 14 16 18 20 24 30 40 50 56 60 64 ]
result false
finding 10 in tree [ 14 16 18 20 24 30 40 50 56 60 64 ]
result false
finding 40 in tree [ 14 16 18 20 24 30 40 50 56 60 64 ]
result true
finding 14 in tree [ 14 16 18 20 24 30 40 50 56 60 64 ]
result true
finding 64 in tree [ 14 16 18 20 24 30 40 50 56 60 64 ]
result true
removing 0 from tree [ 14 16 18 20 24 30 40 50 56 60 64 ]
result [ 14 16 18 20 24 30 40 50 56 60 64 ]
removing 16 from tree [ 14 16 18 20 24 30 40 50 56 60 64 ]
result [ 14 18 20 24 30 40 50 56 60 64 ]
removing 60 from tree [ 14 18 20 24 30 40 50 56 60 64 ]
result [ 14 18 20 24 30 40 50 56 64 ]
removing 30 from tree [ 14 18 20 24 30 40 50 56 64 ]
result [ 14 18 20 24 40 50 56 64 ]
removing 50 from tree [ 14 18 20 24 40 50 56 64 ]
result [ 14 18 20 24 40 56 64 ]
40
  20
        14
            .
            18
                .
                .
        24
            .
            .
  64
        56
            .
            .
        .
```

```
inserting 0 into tree [ 14 18 20 24 40 56 64 ]
result [ 0 14 18 20 24 40 56 64 ]
inserting 100 into tree [ 0 14 18 20 24 40 56 64 ]
result [ 0 14 18 20 24 40 56 64 100 ]
inserting 0 into tree [ 0 14 18 20 24 40 56 64 100 ]
result [ 0 14 18 20 24 40 56 64 100 ]
inserting 100 into tree [ 0 14 18 20 24 40 56 64 100 ]
result [ 0 14 18 20 24 40 56 64 100 ]
after removing all elements [ ]
```

Hint: You may find it helpful to look more closely at the sorted-list example briefly shown in class and available on the course "sample programs" page — it's meant to be a model for one way to implement a linked data structure in C, and the functions you need to write code for are meant to be tree versions of functions in `sorted-int-list.c`. It's up to you whether to use recursion or iteration or both, but I advise that recursion will probably be much easier for the two functions that print the tree.