CS 352 (Fall 16): System Programming and UNIX

Project #11

Binary Trees

due at 9pm, Wed 7 Dec 2016 - **Last Day of Classes!**

1 Overview

In this project, you will implement a Binary Search Tree in C. Your tree will contain integers; you can use the C type `int`, and you'll convert strings to integers with the `%d` format specifier of `sscanf`.

For testcases, we'll go back to the early part of this semester - we'll be reading from `stdin`, so each testcase will be a simple file. Each file will be a series of lines; each line will be a single character (giving a command to perform), sometimes followed by an integer. There will be commands to insert (obviously!), search, find min and max, and delete. There will also be commands to print the tree in two forms: a pre-order traversal, and an in-order traversal. Using these commands, we'll be able to track how you modify the data structure over time, and ensure that you update it properly.

1.1 Extra Credit

All of the commands listed below are required, and will be tested with testcases, with one small exception. Delete in a BST has three cases: the node to delete may have zero, one, or two children. You are required to implement the zero and one child cases. However, the two child case will be extra credit; if you successfully implement it, you will receive a small amount of bonus points. (The amount is not yet determined, but it will not be more than 5 points.)

1.2 Code Style Recommendations

You are allowed to implement this data structure however you wish. However, you are encouraged to use the

```
x = change(x);
```

style. In this style, the return value from the `insert()` and `delete()` functions is a pointer to the new tree. This makes insertion easy, even when the tree was formerly empty - and makes deletion easy, even when you are deleting the last node.

For example, the following code inserts 123 into the current tree:

```
rootNode = bst_insert(rootNode, 123);
```

2 Required Commands

You must implement the following commands:

- `i <int>` (insert)

  Insert the value into the tree. If the value already exists, then print a message to `stderr` but do not modify the tree.

- `d <int>` (delete)

  Delete the value from the tree. If the value does not exist, then print a message to `stderr` but do not modify the tree.

  Extra Credit Note:

  As mentioned above, you are required to implement delete for the cases where the node has zero or one children. Supporting the two-child case is extra credit.

  If you choose to implement this, always swap the node-to-delete with is **successor**. (If you don't, you won't match the example executable.)

- `s <int>` (search)

  Search for a given value in the tree. Print out the line

  ```
  <int> found
  ```

  if it is found; print out the line

  ```
  <int> NOT found
  ```

  if not. Both messages should be printed to `stdout` - that is, it is **not** an error if the value is not found.

- `m` (minimum)

  Print out the minimum value in the tree, followed by a newline. If the tree is empty, print out an error message to `stderr`, but continue running.

- `M` (maximum)

  Print out the maximum value in the tree, followed by a newline. If the tree is empty, print out an error message to `stderr`, but continue running.

- `p` (pre-order traversal)

  Print out a pre-order traversal of the tree, all on one line, with one space before every value (including the first). Do not print any trailing whitespace, except for a newline. If the tree is empty, print out a single blank line.

- `P` (in-order traversal)

  This is the same as the `p` command - except that it is an in-order traversal.

2.1 Input Format

`stdin` will contain one or more lines; the last line may or may not have a newline at the end. Blank lines (including lines that contain only whitespace) should be ignored silently.

Multiple commands can be placed on the same line, separated by semicolons. Break the line into smaller pieces, and then treat them like ordinary lines (meaning that "blank commands" are perfectly legal).

Whitespace should be ignored, except that whitespace (one or more characters) are required in any command that uses a parameter (`i,d,s`). You should check for whitespace with `isspace()` or equivalent.

If you encounter any command which has a letter you don't recognize - or if you encounter a command with more parameters than are required - you must print out a message to `stderr`, skip the command, and then continue running.

2.2 Examples

Instead of placing input examples in this spec, I have provided several testcases to get you started. Pay special attention to the "semicolons" testcase, as it shows you the wide variety of possible ways that semicolons can be used to break a line into smaller pieces.

3 `valgrind` / `gcov`

As in previous projects, we will use the testcases you provide to run `gcov` tests on your program. You must cover all code, except for `malloc()`-failure handling code.

As in previous projects, we will run your code under `valgrind` for all of the official testcases. To get full credit, your code must run with no errors, and must also free all memory that you have used.

4 Makefile

A Makefile is not required in this Project.

5 What You Must Turn In

Turn in the following directory structure using the assignment name `cs352_f16_proj11`:

```
bst352/
    bst352.c
    test_bst352_*
```

NOTE: In this project, you are not required to submit a Makefile. For simplicity, please only submit a single C file - named `bst352.c`

However, as in previous projects, we will be doing `gcov` testing - so make sure that you include enough testcases to cover all of your code!

5.1 Grading Scheme

This project has some grade items which apply to the entire project (not to individual testcases), so our grading scheme has gotten more complex:

- 10% of your grade will come from `gcov` coverage
- 90% of your grade will come from testing your code, comparing it to the example program.

Of course, just as before, we also have a set of penalties which may be applied - these subtract from your overall score:

- -10% - Poor style, indentation, variable names
- -10% - No file header, or missing header comments for any function (other than `main()`)
- -10% - Missing checks of returns codes from standard library functions (such as, but not limited to, `malloc()`)
- -20% - Any use of `scanf()` family `%s` specifier without limiting the number of characters read.

5.2 Testcase Grading

We have decided to subdivide the score from each testcase:

- You must exactly match `stdout` to get **ANY** credit for a testcase.
- You will lose one-quarter of the credit for the testcase if the exit status or `stderr` do not match the example executable.
- You will lose one-quarter of the credit for the testcase if `valgrind` reports any errors. (This includes any memory leaks.)

As always, you lose half of your testcase points if your code does not compile cleanly (that is, without warnings using `-Wall`).

5.3 Testcase Selection

As in Project 7, we will use the testcases that you submit to perform `gcov` testing on your program - and a selection of testcases from the students (plus a few from the instructors) for checking to see if your program works **correctly.**

6 Standard Requirements

- Your C code should adhere to the coding standards for this class: `http://www.cs.arizona.edu/classes/cs352/fall16/DOCS/coding-standards.html`
- Your programs should indicate whether or not they executed without any problems via their **exit status** - that is, the value returned by `main()`. If you return 0, that means "Normal, no problems." Any other value means that an error occurred.

  In this class, we will always use the value 1 to indicate error; however, outside this class, many programs use many different exit status values - to indicate differnt types of errors.

  In `bash`, you can check the exit status of any command (including your programs) by typing `echo $?` immediately after the program runs (don't run **any** commands in-between).

7 Grade Preview

48 hours before the project is due, we will run the automated grading script on whatever code has been turned in at that time; we'll email you the result. This will give you a chance to see if there are any issues that you've overlooked so far, which will cost you points.

Of course, this grading script will not include the full set of student testcases, which we will use in the grading at the end, but it will give you a reasonable idea of what sort of score you might earn later on.

We will only do this once for this project. If you want to take advantage of this opportunity, then make sure that you have working code (which compiles!) turned in by 48 hours before the due date.