Lab 5 Grading Guidelines last modified by Aaron Bloomfield on August 13, 2013 11:27:15 AM EDT

Lab 5 grading guidelines

Pre-lab

Out of 10 points. This lab part **is** expected to compile.

Note that execution of the their pre-lab code requires that the pizza.zip file from the tutorial.

- 4 points: TreeCalc code modifications
  - Note that you can see the skeleton code on the Collab site, and then see what the students modified from there.
  - 4 points: Their code is implemented properly (this can be told by looking at the output for the results, and then checking the style of the code - see the bottom bullet point in this section)
  - 3 points: They made all of the modifications, and have the right idea, but a few things here and there prevented it from working correctly.
  - 2 points: They attempted to modify all the appropriate methods, although they don't work very well. This is for an honest attempt at the pre-lab, even if it doesn't work.
  - 1 point: They made some modifications (some others were not done), but they didn't get anything working.
  - 0 points: They submitted the skeleton code
  - Don't take points off if it does not compile - that will be taken off later. Assume that one were to fix the compiler errors, and grade based on that.
  - -1 point if their code has poor indentation, bad style, etc.
- 2 points: TreeCalc destructor works
  - The destructor will have to either delete all the TreeNodes itself, or by calling clearTree()
  - 0 points if their destructor does not function properly
  - 0.5 points for only deleting the root node and not recursively deleting its children.
  - 1 point if they didn't implement the destructor, but implemented clearTree correctly (meaning that just by calling clearTree in the destructor their code would have been fine).
  - 1 point if they have the right idea, but do something wrong (such as deleting the current node before deleting the children node - you have to delete the children (recursively) first)
  - 2 points if their destructor works properly
- 3 points: Outputs correct results
  - See the output below
  - The first result is 42 (and is the same input as provided in the lab); the other three results are all 216
  - Note that the first and fourth output runs use negative numbers - if their code cannot handle negative numbers, then they will have different results for these execution runs (-1 for not considering negative numbers)
  - The points off below will most likely be for all (or most of) the trees, not just one of them.
  - -2 points if they perform the prefix transversal wrong
  - -2 points if they perform the infix transversal wrong (-1 for incorrect parantheses)
  - -2 points if they perform the postfix transversal wrong
  - Minimum score of 0 on this part
- 1 point: Makefile-pizza
  - From the tutorial; it should contain the following parts (from the last page of the Make tutorial: comments, CXX and CXXFLAGS macros, OFILES, a suffix rule, the rule to compile the program, a 'clean' rule, and the dependency lines
  - We won't worry about the comments for the grading
  - For the remaining 7 parts listed above, they can miss one without a grade penalty
  - Each additional missed part receives 0.5 points off
  - Thus, if they miss 3 or more parts (again, not counting comments), they receive a zero for this part
  - Regardless of these rules, if the file was named incorrectly or had a file extension (.cpp, .txt, etc) then 0.5 points were taken off.

Input:

Four test cases were run. The result of the first is 42, the remaining three yield 216. Each input included a space and a hash character ('#') after the final operator to indicate end-of-input.

- 34 6 + -8 4 / -
- 6 4 + 5 / 3 * 6 6 * *

- 0 3 4 + 7 / 4 - 3 * 2 2 2 * * * 3 * -
- 0 -216 -

Output:

```

Enter elements one by one in postfix notation
Any non-numeric or non-operator character, e.g. #, will terminate input
Enter first element: Enter next element: Enter next element: Enter next element: Enter next
                     Enter next element: Enter next element: Expression tree in postfix exp
Expression tree in infix expression: ((34 + 6) - (-8 / 4))
Expression tree in prefix expression: - + 34 6 / -8 4
The result of the expression tree is 42

======================================================================
Enter elements one by one in postfix notation
Any non-numeric or non-operator character, e.g. #, will terminate input
Enter first element: Enter next element: Enter next element: Enter next element: Enter next
                     Enter next element: Enter next element: Enter next element: Enter next
                     Enter next element: Expression tree in postfix expression: 6 4 + 5 / 3
Expression tree in infix expression: ((((6 + 4) / 5) * 3) * (6 * 6))
Expression tree in prefix expression: * * / + 6 4 5 3 * 6 6
The result of the expression tree is 216

======================================================================
Enter elements one by one in postfix notation
Any non-numeric or non-operator character, e.g. #, will terminate input
Enter first element: Enter next element: Enter next element: Enter next element: Enter next
                     Enter next element: Enter next element: Enter next element: Enter next
                     Enter next element: Enter next element: Enter next element: Enter next
                     Enter next element: Enter next element: Enter next element: Enter next
                     Expression tree in postfix expression: 0 3 4 + 7 / 4 - 3 * 2 2 2 * *
Expression tree in infix expression: (0 - ((((((3 + 4) / 7) - 4) * 3) * (2 * (2 * 2))) * 3
Expression tree in prefix expression: - 0 * * * - / + 3 4 7 4 3 * 2 * 2 2 3
The result of the expression tree is 216

======================================================================
Enter elements one by one in postfix notation
Any non-numeric or non-operator character, e.g. #, will terminate input
Enter first element: Enter next element: Enter next element: Enter next element: Expressio
Expression tree in infix expression: (0 - -216)
Expression tree in prefix expression: - 0 -216
The result of the expression tree is 216

```

The fifth and last execution compiled the pizza executable from the Make tutorial. Note that the executable was not run; it was just compiled, and the output should show a successful execution.

In-lab

Out of 10 points. This lab part **is** expected to compile. Compilation is done through 'make'.

- 7 points: Modified in-lab code (this is similar to the rubric as below)
  - 7 points: If it's working and matches the solutions (see below)
  - 5 points: It generally works, but a few of the test cases are off
  - 3 points: They have the right idea, but it's not working very well
  - 2 points: It looks like they have a vague idea of what's going on, but the code doesn't really work
  - 1 point: For having a pulse (i.e. they submitted something)
  - 0 points: If they didn't submit anything
  - Feel free to interpolate between these values
- 3 points: Test case file testfile4.txt
  - Download and run their test case (testfile4.txt) with the a correct implementation of the binary tree and AVL tree (from the solutions section of the Collab site). The test case should be written such that there is a significant difference between the average node depth for the two tree types (the binary tree should be at least 2.0 times the depth of the

AVL tree)

Input:

Three test cases were run, one for each of the supplied input files (testfile1.txt, testfile2.txt, and testfile3.txt). The words searched for in each of these files were (respectively): hardly, mauve, and unknown.

Output:

```


Please enter the name of a file of words:  this class is so much fun that we can hardly sta
16 words in this text



BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 14
Avg. node depth = 2.78571

AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 14
Single Rotations = 1
Double Rotations = 3
Avg. node depth = 2.28571

        Enter word to lookup >  Word was found: hardly
BST:
Left links followed = 2
Right links followed = 2
Total number of nodes = 14
Avg. node depth = 2.78571

AVL Tree:
Left links followed = 1
Right links followed = 2
Total number of nodes = 14
Single Rotations = 1
Double Rotations = 3
Avg. node depth = 2.28571


========================================================================
Please enter the name of a file of words:  a bee caught dung everywhere flying greatly high
16 words in this text



BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 16
Avg. node depth = 6.0625

AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 16
Single Rotations = 9
Double Rotations = 0
Avg. node depth = 2.5

        Enter word to lookup >  Word was found: mauve
BST:
```

```
Left links followed = 0
Right links followed = 9
Total number of nodes = 16
Avg. node depth = 6.0625

AVL Tree:
Left links followed = 0
Right links followed = 1
Total number of nodes = 16
Single Rotations = 9
Double Rotations = 0
Avg. node depth = 2.5


===============================================================================
Please enter the name of a file of words:  zany cobwebs littered the clockwork orange lands
13 words in this text


BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 13
Avg. node depth = 3.23077

AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 13
Single Rotations = 1
Double Rotations = 2
Avg. node depth = 2.23077

        Enter word to lookup > unknown was not found in /home/submissions/cs216-073/execinp
BST:
Left links followed = 2
Right links followed = 3
Total number of nodes = 13
Avg. node depth = 3.23077

AVL Tree:
Left links followed = 2
Right links followed = 2
Total number of nodes = 13
Single Rotations = 1
Double Rotations = 2
Avg. node depth = 2.23077
```

Post-lab

Out of 10 points. This lab part is **not** expected to compile.

- 10 points: Submitted report in analysis.pdf
  - If the file format is something other than a PDF format, they get zero points. This includes flat text files.
  - There are three parts, as specified in the lab document, that they need to include: numerical results, characterization when AVLs are better than BSTs, and a discussion about the cost of AVL implementation
  - For each of these these parts, we can characterize their report as "good", "okay", "bad", and "non-existant"
    - "Good" means that they manage to hit the relevant points. We aren't looking for length here - just that they understand the concept involved. They get no points off in this case.
    - "Okay" means that they hit some of the relevant points, but either don't understand them fully or did not put in much time to explain them properly (or if no numerical results were provided for testfile4). They get 2 points off (per part) in this case.
    - "Bad" means they did not hit the relevant points. They get 3 points off (per part) in this case

        - "Non-existent" is pretty self-explanatory. 4 points off. Obviously, if they have a few non-existent parts, they can't get below a zero.

To be specific:

- 1 point for **reasons to use testfile4.txt**

- 3 points for **Numerical Results**
    - 3 points for **Good**
    - 2 points for **Okey**
    - 1 points for **Bad**
    - 0 points for **Non-existent**

- 3 points for **Characterization**
    - 3 points for **Good**
    - 2 points for **Okey**
    - 1 points for **Bad**
    - 0 points for **Non-existent**

- 3 points for **Discussion**
    - 4 points for **Good**
    - 2 points for **Okey**
    - 1 points for **Bad**
    - 0 points for **Non-existent**

Untitled note

Find a notebook

Add tag

Add comments

Version 6.0.6: c128369/1.0.1.250

Web Clipper tutorial

Options

To:

Saving clip...

Clip

Share

Article

Simplified Article

Save

Selection

PDF