Homework 2

Data Structures and Algorithms in C++
Shlomo Hershkop
Department of Computer Science
Columbia University
Summer 2006

Out: July 13, 2006
Due Tuesday July 18, 11pm (electronically).

Theory (50 points)

1. Describe resulting stack and the output of the following series of stack operations: push(5), push(8), pop(), push(3), push(8), pop(), pop(), push(9), push(1), pop(), push(7), push(4), pop(), pop(), push(4), pop(). Assume that each popped value is printed.

2. When we have a list of items, one way of saving time when deleting an item, is to mark the item as deleted. That is you need to add another field variable saying if the list item is alive. This is known as "Lazy Deletion". In addition the list needs to keep track of how many elements are alive and how many are not. When the number of dead items reaches some point, you can run through the list and actually delete them.
    1. list the advantages and disadvantages of lazy deletion
        1. for array implementation of lists
        2. for linked list implementation
    2. how would this affect binary trees.

3. Define "degree" of a node as the number of its non-empty children. Prove by induction that the number of degree 2 nodes in any binary tree is one less than the number of leaves.

4. Let T be a general ordered tree with more than one node. Is it possible that the preorder traversal of T visits the nodes in the same order as the postorder traversal of T? If so, give an example; otherwise, argue why this cannot occur. Likewise, is it possible that the preorder traversal of T visits the nodes in the reverse order of the postorder traversal of T? If so, give an example; otherwise, argue why this cannot occur.

5. A binary search tree is created by inserting, in order, the following numbers: 18, 3, 25, 0, 2, 11, 12, 34, 30, 40. Draw the tree. Then give the pre-order, post-order and in-order traversals of the tree.

6. Give an O(n) algorithm for computing the depth of all the nodes of a tree T, where n is the number of nodes of T.

7. Describe in pseudo-code a non-recursive method for performing an in-order traversal of a binary tree in linear time. (Hint: Use a stack.)

8. Describe routines to implement two stacks using a single array. Your stack routines should not declare an overflow unless every slot in the array is used.

9. Show the result of adding 4, 14, 3, 2, 15, 18, 17, 39 into an initially empty binary search tree. Then show the result of deleting the root.

10. If the in-order and pre-order printout of the tree is the same, are they exactly the same shape ?? Can you prove this ??

Programming (50 points)

For this project you will compare implementing more or less the same header file with 2 different .cpp files. The idea is to see how changing the implementation will affect the running time of your program.

Say you create a Dictionary.h file which will hold your definition of a dictionary.

```
template<class obj>
Class Dictionary {
public:
        Dictionary();
        ~Dictionary();
        void add(obj);
        bool isPresent(obj);
        int getSize();
        void print();
        void saveToFile(char *);
        void loadFromFile(char *);
private:
        //anything you need here
};
```

1. **Create a simple dictionary program.** Write a simple program interface that allows a user to query a dictionary and find out whether a query word is spelled correctly. Use a "dictionary>" prompt, and reply with "correct" or "incorrect" for each given word. The dictionary should be case-insensitive, and it should raise an error if the user inputs multiple words on a single line. The command "quit" should end the session. Here is a sample session:

```
dictionary> hello
correct
dictionary> world
correct
dictionary> sarcophogus
incorrect
dictionary> 098351
invalid input: non-alphabetic characters
dictionary> hello world
invalid input: multiple words
dictionary> quit
```

   The dictionary should be constructed from /usr/share/dict/words . For step 1 of the assignment, you will implement the dictionary inefficiently, using for example the linked list ADT. You can either use the books list code or written your own.

2. **Make the dictionary efficient.** Now improve the dictionary representation so that a search for a word is completed quickly. You must use a binary search tree, and you should be sure your implementation is efficient even when the input is in sorted order. Using this improved representation, add to the above interface the ability to add words to the dictionary, as well as print the dictionary in alphabetical order.

```
dictionary> intron
incorrect
dictionary> add intron
adding intron
dictionary> intron
correct
dictionary> print dictionary
a
aaa
aaas
aarhus
aaron
aau
aba
...
```

3. **Timing:**
    a. Add a timing class to above both mains. The idea of a timing class is that during construction, you take the current time, and have functions to be able to get time differences. Now time both step 1 and 2 to compare differences in load time and lookup times.

    Example of use:

    ```
    Timing T;
    cout<<"Loading words"<<endl;
    Dictionary words("c:\words.txt");
    cout <<"total time was "<< T.timediff() <<endl;

    …

    T.reset();
    //lookup word
    cout <<"total time was "<< T.timediff() <<endl;
    ```