Computer Science 1 – Program 5
Buddy Book (Binary Search Trees)
Assigned: 3/23/11
Due: 4/6/11 (Wednesday) at 11:55pm (WebCourses time)

Purpose

Learn to implement the functionality of a binary search tree and to combine that functionality with linked lists. Of course, more and more practice with pointers!!!

The Problem

After watching the movie, "The Social Network", you get motivated into thinking you could be the next Mark Zuckerberg. Sure, whatever, maybe you're only in Computer Science I, but you can do ANYTHING after that killer Program 4!

So here's the initial goal: you will build a database manager for a basic social networking site, BuddyBook.com. Expecting that your idea could one day go viral, you don't want too many limitations (planning for the future); so the database should be able store an indefinite number of people (well, as long as they are at least 13 years old…legal mumbo jumbo). Of course, it will need to store a bit of basic information for each user, such as their ID, first and last name, as well as their age. Additionally, each user will need to maintain a list of their buddies.

Just like with any database application, you will need to be able to quickly search through the database to find a person based on their name or their unique ID. And with your vast knowledge of computer science, you know that a simple array of people or a linked list of people would be an EPIC FAIL, as that would result in an O(n) search. So you will need to implement the database as a binary search tree, thereby taking advantage of O(logn) search, insert, delete, etc.

You will store people in Buddy Book based on ID numbers, which also happens to be the same as alphabetical by last and then first name. (This means that you do not need to insert by names – you can merely insert by ID, which is guaranteed to be also in order by name.) So Bob Smith is guaranteed to have a "smaller" ID than Joe Smith. Assume that no two people will have the same first and last names.

You will need to make use of the following structures:

```
typedef struct buddyList {
     int buddy;
     struct buddyList *next;
} buddies;
```

This struct is a LINKED-LIST. Every user must maintain a linked list of their buddies. So you will use this struct as a member of the BSTnode struct.

```
typedef struct tree_node {
     int ID;
     char firstName[21];
     char lastName[21];
     int age;
     buddies *myBuddies;
     struct tree_node *left;
     struct tree_node *right;
} BSTnode;
```

This struct will be used to store each user's information. All members of this struct should be self-explanatory, with the possible exception of: `buddies *myBuddies;`

That member, as mentioned above, represents the head of a linked list of this particular user's buddies.

Your Buddy Book program will read in a series of instructions from a file, such as ADD, DELETE, BUDDY, UNBUDDY, etc., and you will need to perform the instruction and print the appropriate information to the output file.

Input/Output Specifications

The input file will be called "`buddyBook.in`". The first line of the file will be an integer, *k*, indicating the number of commands that will be run on the database. The following *k* lines will each be a single command, **with the relevant data on the same line**, separated by spaces.

The commands you will have to implement are as follows:

- `ADD` – Creates a new account (a new node). The command will be followed by the following information: ID, a non-negative int variable, which you will use to sort the accounts by, a first name and last name, each no longer than 20 characters, and age, which is a non-negative integer. You will have to check if this person is at least 13 years old before creating a new account, as children under the age of 13 cannot create accounts.
- `FINDNAME` – This command will be followed by a name, first then last. You must search the database to find this name, and then print out some information about this person. It should print their name, age, and the number of Buddies they have. If there is not a person by that name in the database, it should print an error message saying so.
- `FINDID` – Similar to `FINDNAME`, this will find a person, based on their ID number instead. The output should be the same. It should print their name, age, and the number of Buddies they have. If there is not a person by that name in the database, it should print an error message saying so.
- `BUDDY` – This will be followed by two names, first and then last for both. When you see this command, this means that the two people are now Buddies, and thus you should add both people's IDs to each others' Buddy-lists. All connections on BuddyBook are always two-way – e.g. if John is in Mary's Buddy-list, then Mary is also in John's. If either person in the `BUDDY` command is not in the database, or if they are already Buddies, an error message should be displayed (see sample output below).
- `UNBUDDY` – This is the opposite of `BUDDY`. Like `BUDDY`, it will be followed by two names. You will have to find both names, and remove their IDs from each others' Buddy-lists. Again, if either person is not in the database, or they are not actually Buddies, an error message should be displayed.
- `DELETE` – This command, followed by a name, means you must delete this user's account from the database. Before you do so, however, you must remove them from any other users' Buddy-lists. Thus, for the user you are deleting, you will need to loop through their own Buddy-list, and find all of their Buddies by searching for their IDs, and run your `UNBUDDY` command to remove them from each others' lists. If the person you are trying to delete does not actually have an account already, an error message should display.
- `PRINTBUDDIES` – This command, followed by a name, first then last, prints the member's name, the number of buddies they have, and then it prints an alphabetical list of their buddies.
- `PRINTTREE` – This command prints all members in alphabetical order (last name, and then first name), along with printing basic information about these members (see output below).

Your program must output to a file, called "**`buddyBook.out`**". **You must follow the program specifications exactly**. You will lose points for formatting errors and spelling.

*****WARNING*****

Your program MUST adhere to this EXACT format of the output shown below (spacing capitalization, use of dollar signs, periods, punctuation, etc). The graders will use very large input files, resulting in very large output files. As such, the graders will use text comparison programs to compare your output to the correct output. If, for example, you have two spaces between in the output when there should be only one space, this will show up as an error even through you may have the program correct. You WILL get points off if this is the case, which is why this is being explained in detail. Minimum deduction will be 10% of the grade, as the graders will be forced to go to text editing of your program in order to give you an accurate grade. So as an example, here is the first format shown above (if the instruction was an addition):

Again, your output MUST ADHERE EXACTLY to the sample output shown below.

Sample Input

```
ADD 56784 Homer Simpson 43
ADD 65324 Marge Simpson 41
ADD 35477 Bart Simpson 10
ADD 22379 Ned Flanders 38
BUDDY Homer Simpson Marge Simpson
BUDDY Homer Simpson Bart Simpson
BUDDY Maggie Simpson Bart Simpson
FINDNAME Bart Simpson
FINDNAME Homer Simpson
FINDID 65324
ADD 12352 Maude Flanders 37
BUDDY Ned Flanders Maude Flanders
BUDDY Ned Flanders Homer Simpson
BUDDY Homer Simpson Marge Simpson
FINDNAME Homer Simpson
FINDNAME Ned Flanders
FINDNAME Maude Flanders
BUDDY Homer Simpson Maude Flanders
FINDNAME Homer Simpson
PRINTBUDDIES Homer Simpson
UNBUDDY Homer Simpson Ned Flanders
PRINTBUDDIES Homer Simpson
PRINTBUDDIES Marge Simpson
UNBUDDY Homer Simpson Marge Simpson
PRINTBUDDIES Marge Simpson
FINDID 56784
FINDID 22379
FINDID 77777
DELETE Maude Flanders
PRINTBUDDIES Ned Flanders
FINDNAME Ned Flanders
FINDNAME Maude Flanders
PRINTTREE
```

Corresponding Output (saved to file called "buddyBook.out")
PAY ATTENTION TO THE EXACT FORMAT (SPACING, PUNCUATION, ETC.)

```
Added Homer Simpson, age 43
Added Marge Simpson, age 41
Bart Simpson rejected – You must be 13 or over to create a profile.
Added Ned Flanders, age 38
```

```
Homer Simpson and Marge Simpson are now Buddies.
Cannot Perform Buddy Command:
   Bart Simpson – This profile does not exist.
Cannot Perform Buddy Command:
   Maggie Simpson – This profile does not exist.
   Bart Simpson – This profile does not exist.
Bart Simpson not found.
Found:  Homer Simpson, age 43, 1 Buddy
Found:  Marge Simpson, age 41, 1 Buddy
Added Maude Flanders, age 37
Ned Flanders and Maude Flanders are now Buddies.
Ned Flanders and Homer Simpson are now Buddies.
Cannot Perform Buddy Command:
   Homer Simpson and Marge Simpson are already Buddies.
Found:  Homer Simpson, age 43, 2 Buddies
Found:  Ned Flanders, age 38, 2 Buddies
Found:  Maude Flanders, age 37, 1 Buddy
Homer Simpson and Maude Flanders are now Buddies.
Found:  Homer Simpson, age 43, 3 Buddies
Homer Simpson had 3 Buddies:
   Maude Flanders
   Ned Flanders
   Marge Simpson
Homer Simpson and Ned Flanders are no longer Buddies.
Homer Simpson has 2 Buddies:
   Maude Flanders
   Marge Simpson
Marge Simpson has 1 Buddy:
   Homer Simpson
Homer Simpson and Marge Simpson are no longer Buddies.
Marge Simpson has no Buddies.
Found:  Homer Simpson, age 43, 1 Buddy
Found:  Ned Flanders, age 38, 1 Buddy
ID 77777 not found.
Deleted Maude Flanders.
Ned Flanders has no Buddies;
Found:  Ned Flanders, age 38, 0 Buddies
Maude Flanders not found.
Members of Buddy Book:
   Ned Flanders, age 38, 0 Buddies
   Homer Simpson, age 43, 1 Buddy
   Marge Simpson, age 41, 0 Buddies
```

Helpful Suggestions / Comments

The original plan was to have all users of Buddy Book indexed alphabetically, based on their last name and then first name. There was no reason to even have a user ID. However, all of the binary tree code and linked list code on the course website uses a basic "int data" as its key value (indexing value). As such, and in an effort to facilitate less crying, we gave each user an int ID that just so happens to "match" up with their alphabetical position based on last name and then first. What is the benefit, or how does this help you? Instead of having to modify the binary search tree code (and linked list code) to search based on string compares of last and first names, you can stick with using the ID's of the users, which just so happens to be INTS! This allows you to essentially recycle the code on the course website, with little need for modification.

Deliverables

Turn in a single file, *buddyBook.c*, over WebCourses that solves the specified problem.

Grading Details

Your program will be graded upon the following criteria:

1) Adhering to the implementation specifications described here.
2) Your algorithmic design.
3) Correctness.
4) The frequency and utility of the comments in the code, as well as the use of white space for easy readability. (We're not kidding here. If your code is poorly commented and spaced and works perfectly, you could earn as low as 80-85% on it.)
5) Compatibility to Dev C++. (If your program does not compile in Dev C++, you will get a sizable deduction from your grade.)
7) **Use of Binary Search Trees with Linked Lists. Since the purpose of this assignment is to teach you to Binary Search Trees, you will not receive credit if you don't use them.**

Restrictions

Name the file you create and turn in *buddyBook.c*. Although you may use other compilers, your program must compile and run using Dev C++. Your program should include a header comment with the following information: your name, course number, section number, assignment title, and date. You should also include comments throughout your code, when appropriate.