CS 300 Fall 2014 – Programming Assignment 6

Assigned: October 22, 2014 **Due: November 5, 2014**

Binary Search Trees: Use a binary search tree to manage data for students who have enrolled in CS 300 for some semester. The list is provided as the file `enrollment_list`. Each student record must be read from the file, a node created for the student (as described below), and the node added to the binary search tree so that students will be alphabetically ordered in the tree by last name and for last names which are the same, further ordered by first-name-followed-by-middle-name. The enrollment website had a bug and didn't check if a student was already enrolled when they tried to enroll. Some students forgot if they had enrolled, so enrolled again. As a result, the `enrollment_list` file contains some duplicate records. The tree must not contain duplicates.

After the data from `enrollment_list` has been read into your tree, read the file `drop_list` which contains records (in the same format as `enrollment_list`) of students who dropped the course before it started and need to be removed from the tree. The resulting class list, found by traversing the tree in order, must be saved in a file called `classlist.txt` in the format shown in the sample run below.

Create files called `bin_search_tree.c` and `bin_search_tree.h` which will implement a binary search tree data structure. Make new files `classlist.c` and `classlist.h` which will contain the main program.

Requirements:

- Create the file `bin_search_tree.c` containing functions which implement a binary search tree, with the prototypes and functionality listed below:
  - `void key_setequal(Key a, Key b);` – Sets the key `a` to the value of the key `b`.
  - `int key_isequal(Key a, Key b);` – Returns true (1) if the two keys are equal, or false (0) otherwise.
  - `int key_lessthan(Key a, Key b);` – Returns true (1) if `a` comes before `b` in the order used for sorting.
  - `void tree_init(Tree *t);` – initialize the tree `*t`.
  - `Node *tree_makenode(Key k, Node *parent);` – makes a new `Node` (with `malloc`), sets its key to `k`, its parent to `parent`, and its left and right children to NULL.
  - `Node *tree_root(Tree t);` – returns the root of `t`. (This is an identity function, included for completeness.)
  - `Node *tree_parent(Node *n);` – Returns the parent of `n`.
  - `Node *tree_leftchild(Node *n);` – Returns the left child of `n`.
  - `Node *tree_rightchild(Node *n);` – Returns the right child of `n`.
  - `Node *tree_search(Key k, Tree t);` – Returns the node in tree `t` whose key is `k` if it exists, or NULL if it doesn't exist.
  - `int tree_num_children(Node *n);` – Returns the number of children (0, 1 or 2) of node `n`.
  - `void tree_insert(Tree *t, Key k);` – Adds a new node whose key is `k` to the tree `t`.
  - `void tree_delete(Tree t, Key k);` – Removes the node whose key is `k` from the tree `t` if it is contained in the tree.
  - `void tree_preorder(Node *n, FILE *fp);` – Writes out the key for each node of tree `n`, one key per line, in preorder, to the file `fp`.
  - `void tree_postorder(Node *n, FILE *fp);` – Writes out the key for each node of tree `n`, one key per line, in postorder, to the file `fp`.
  - `void tree_inorder(Node *n, FILE *fp);` – Writes out the key for each node of tree `n`, one key per line, inorder, to the file `fp`.
  - `int tree_empty(Tree t);` – Returns true (1) if tree `t` is empty and false (0) otherwise.
  - `int tree_height(Tree t);` – Returns the height of the tree `t`.
  - `void tree_makenull(Tree t);` – Deletes all nodes of the tree `t`.

The above prototypes will go in the file `bin_search_tree.h`, together with the type definitions for the tree:

```
#define KEY_LENGTH 100

/* Type definitions */
typedef char Key[KEY_LENGTH];

typedef struct node
{
    Key key;
    struct node *parent;
    struct node *leftchild;
    struct node *rightchild;
} Node;

typedef Node * Tree;
```

- In the file `bin_search_tree.c`, only the functions `key_setequal`, `key_isequal`, and `key_lessthan` may use functions from the string library. All other functions in this file which need to do string comparisons or copies must use one of these `key_*` functions to do so. Functions in `classlist.c` may call string functions directly when creating the keys.
- The `main()` function of your program must go in the file `classlist.c` and must contain a variable of type `Tree` to which the data is added using the function `tree_insert` and removed using `tree_delete`.
- All access to, and changing of, the data in your `Tree` variable done in `classlist.c` must make use of functions from `bin_search_tree.c` and **not** access `Node` data members directly.
- The format of each line of the input files is four fields: the course number "CS 300", first name, middle name, and last name, separated by colon characters. As each line is read, create a `Key` for a `Node` from it by ignoring the course number, and making a string in the format "last, first middle". For example, if "John:X:Doe" is read from the file, the `Key` "Doe, John X" would be created. This facilitates the required sorting of nodes in the tree because it is sorted alphabetically by last name, and for last names which are the same, further sorted by the first name followed by the middle name as a single string.
- At the end of your program, call the function `tree_makenull` for your `Tree`.
- Copy the `makefile` from assignment 5 and update it to work for compiling your program.
- A sample run of your program should look like:

```
The class list was saved in the file classlist.txt.
```

- The above sample run must write a file called `classlist.txt` with the following contents:

```
Class List for CS 300:

Acharya, Vijay Kumar
Addington, Alan Aaron
Akbar, Adnan Mohammad
Bajaj, Ravi Kumar
Brown, Kenneth Wayne
Browning, Kerry Cole
Chai, Tze-Chen X
Crandall, Dale Thomas
Donnelly, Denise Kay
Hansen, Christopher Raymond
Johnson, Chester Joseph
Johnson, Christopher Leonard
Johnson, Ronald Dean
Jones, James Randall
```

```
Keeling, Christine Theresa
Khan, Mohammad Tariq
Kilmer, Joel Fritz
Landingham, Kerry Randolph
Lane, Crystal Gale
Lewis, Donald Cole
Lowe, Xavier Roger
Mack, Ronald David
Morgan, Joseph Duane
Navarro, Raymond Lewis
Nguyen, David
Nguyen, Thuy Thi
Osbourne, Gary Randal
Patterson, Joseph Lewis
Poston, David Joseph
Romero, Roderick Ray
Sampson, Wayne Phillip
Swanson, Kimberly Kylie
Thomas, Carla May
Thompson, William Allen
Turner, Michael Robert
Turner, Robert Harry
Waggoner, Lindsey Kay
White, Frank Thomas
```

Reminder:

- Be sure that your program includes your name and ID and the necessary comments, and follows the style standards as listed in the document posted on the Information page in Blackboard, called "Requirements for Programs Submitted for Assignments". Part of the grade will be for style and quality.
- Carefully test your program.
- You are welcome to write your program at home. If you do, be sure to compile and test it in the lab before submitting it.

How to submit your program:

- Submit your files electronically using **~cs300d/bin/handin 6 classlist.c classlist.h bin_search_tree.c bin_search_tree.h makefile**
  The first parameter to the handin program ("6") is the assignment number, and the remaining parameters are the file names.

Extra credit version of program (optional):

Meeting the requirements given above will allow you to earn a maximum of 100/100 for this assignment. Modifying your program to meet the requirements of this extra credit section will give you the opportunity for a maximum of an additional 25 points for a total of 125/100.

Sets: Update your program by adding the files `set.c` and `set.h` to implement a restricted **set** data structure (also called a *dictionary*):

- Create the file `set.c` containing functions which implement a set, with the prototypes and functionality listed below:
  - `void set_init(Set *s);` – Initialize the set `s`.
  - `void set_insert(Set *s, Element e);` – Adds the element `e` to `s`.
  - `void set_delete(Set *s, Element e);` – Removes the element `e` from `s`.

- `int set_member(Element e, Set *s);` – Returns true (1) if element `e` is a member of `s` and false (0) if it's not.
- `Element *set_min(Set *s);` – Returns a pointer to the first element of `s` (or NULL if `s` is empty).
- `Element *set_max(Set *s);` – Returns a pointer to the last element of `s` (or NULL if `s` is empty).
- `int set_empty(Set *s);` – Returns true (1) if `s` is empty and false (0) if it's not empty.
- `void set_inorder(Set *s, FILE *fp);` – Writes out each element of `s`, one element per line, inorder, to the file `fp`.
- `void set_makenull(Set *s);` – Deletes all elements of `s`.

The above prototypes will go in the file `set.h`, together with the type definitions for the set:

```
/* Type definitions */
typedef Key Element;

typedef struct
{
   Tree tree;
} Set;
```

- In `classlist.c`, change the type of the variable which stores the class data from `Tree` to `Set` and make the necessary updates to the function calls. This file must not directly contain any variables of type `Tree` or any calls to `tree_*` functions. All access to, and changing of, the data in your `Set` done in `classlist.c` must make use of functions from `set.c`.
- Add a comment at the top of `classlist.c` that you are doing the extra credit version of the assignment.
- Update your `makefile` so that it will correctly compile and use both the `bin_search_tree` and `set` modules.
- Your program must produce the same output as shown in the sample run above.
- Submit your files electronically using **~cs300d/bin/handin 6 classlist.c classlist.h bin_search_tree.c bin_search_tree.h set.c set.h makefile**