CS 240: Programming in C
Spring 2019
Homework 11

Prof. Turkstra

Due April 10th, 2019 9:00 PM

1 Goals

The purpose of this homework is to give you experience in using binary trees. No I/O this time. We'll just focus on the data structures.

Recursion

Part of the purpose of this homework is to gain experience with recursive functions. `insert_node()`, `search_tree()`, `tree_to_list()`, and `delete_tree()` must be implemented recursively to receive any credit!

2 The Big Idea

You've heard that you can easily insert a new element into a binary tree. You've also heard that you can easily find an element that's been inserted into a binary tree. Now you get to prove to yourself that it works. You'll implement the following operations:

- Create a singular tree node;
- Insert a node into a tree;
- Find an element in a tree;
- Convert the tree into a list;
- Delete an entire tree

We will utilize the following structures:

```
typedef struct car {
  char        *make;
  char        *model;
  unsigned int year;
} car;
```

```
typedef struct tree_node {
  struct tree *left_ptr;
  struct tree *right_ptr;
  car         *info;
} tree_node;

typedef struct list_node {
  struct list_node *next_ptr;
  car              *info;
} list_node;
```

Examine hw11.h carefully. It contains a declaration for the tree type that you'll use. It also contains prototypes for the functions that you will write.

3 The Assignment

Study the lecture examples that deal with manipulation of trees. Then, take a look at hw11_main.c to see what it does. Pay particularly close attention to the `tree_print()` function. You'll be looking at generic tree traversal next week.

Functions you will write

You will write the following functions:

`tree_node *create_node(char *, char *, unsigned int);`

Dynamically allocate memory for a tree structure. You will need to allocate space for the `info` field as well as for appropriate fields in the `car` structure.

The first argument is the car's `make`. The second its `model`, and the third its `year`. Remember that they should be *copied*. The `left_ptr` and `right_ptr` should also be initialized to NULL.

This function should return a pointer to the newly allocated node.

Add appropriate `assert()`ion checks.

`void insert_node(tree_node **, tree_node *);`

The first argument is a pointer to the pointer to the root tree element. The second argument is a pointer to the new tree element to be inserted. The key for insertion should be the `model` field. Basically, this function will always be called this way:

```
insert_node(&root, new_item);
```

This means that you can call this function initially when root is NULL and it should set the root variable to point to the newly inserted element.

NOTE: We haven't talked about passing the root of a tree by pointer to a function like this. If this seems hard, feel free to create another function called `my_insert_node(tree_node *, tree_node *)` that looks like the one you saw in lecture. Then `insert_node()` can just check to see if the root is NULL and then call `my_insert_node()` if it's not. Do realize that passing the root pointer by pointer allows you to make a very compact optimization for this routine. Think about it.

No matter what you do, make sure you write your insertion function in a recursive manner.

You should assert that neither the first nor the second argument is NULL. You should assert that the second argument's `left_ptr` and `right_ptr` fields are NULL.

```
tree_node *search_tree(tree_node *, char *);
```

This is a **recursive** function that will search for a tree node whose `model` field matches the second argument. If a matching node is found, the function should return a pointer to the node. If no match is found, the function should return NULL.

The function should assert that the second argument is not NULL. It is okay for the first argument to be NULL.

```
list_node *tree_to_list(tree_node *, enum traversal);
```

This function takes the root of a tree as an argument and "flattens" it according to the specified traversal enum. By "flatten," we mean that you should create a singly-linked list ordered in the specified manner:

1. PREFIX (N-L-R);
2. POSTFIX (L-R-N);
3. FORWARD (L-N-R);
4. REVERSE: (R-N-L).

Do not duplicate the data for the list nodes.

It is not an error for the first argument to be NULL. You should assert that the traversal argument is a valid value.

You are welcome to create helper function(s) as needed.

```
void delete_tree(tree_node **);
```

Delete the entire tree and its associated data recursively. After the function returns, the root of the tree that was passed by pointer should be set to NULL. E.g., if the function is called this way:

```
delete_tree(&root);
```

...then root should be set to NULL after the function is finished.

The function should assert that the first argument is not NULL. However, keep in mind that it is okay for the first argument to point to a NULL pointer.

3.1 Input Files

There are no input files for this assignment.

3.2 Header Files

We provide a header file, hw11.h, for you. It contains prototypes for each of the functions that you will write as well as #definitions for the constants. You should not alter this file. We will replace it with the original when grading.

Note that since you're not using any File I/O operations, you don't need to include `stdio.h`. For this assignment, the only other header files you'll need are `malloc.h` (or `malloc_debug.h`), `assert.h`, and `string.h`.

3.3 Error Codes

There are no error codes for this assignment.

These Standard Rules Apply

- You may add any `#include`s you need to the top of your hw11.c file;
- You may not create any global variables other than those that are provided for you. Creation of additional global variables will impact your style grade;
- Do not look at anyone else's source code. Do not work with any other students.

Submission

To submit your program for grading, type:

```
$ make submit
```

In your hw11 directory. You can do this as often as you wish. We encourage you to submit your code as often as possible. Only your final submission will be graded.

4 Grading

The operation of your functions will be graded out of 100 points. The point breakdown will be determined by the test program.

The test program will be run many times when grading. It is your job to do the same. The lowest score will be your final grade.

This homework will also have a style grade based on 20 points, with 1 point deducted for each code standard violation found.

Your code must compile successfully using `-Wall -Werror -std=c99` to receive any credit. Code that does not compile will be assigned an automatic score of 0.

If your program crashes (e.g., segmentation fault) at any point during the testing process, your score will be reduced by 25 points.