CSE 1320: Intermediate Programming
University of Texas at Arlington
Spring 2021
Dr. Alex Dillhoff

Assignment 8

This assignment focuses on binary search trees and project organization.

Your code must compile without any warnings or errors and run without segmentation faults to receive credit. Additionally, any allocated memory must be freed. Points will be taken for inconsistent formatting.

1. Create a program which reads in CSV data of Creatures and loads them into a Binary Search Tree. Your program should present a menu to the user allowing them to traverse the tree, search for a Creature, and traverse by type.

   For the traversal option, a submenu should prompt the user to select depth-first search (preorder, inorder, postorder) or breadth-first search.

   For the search option, the name of the creature should be input. If the creature exists, print out the details. Otherwise, report that the creature could not be found.

   When traversing by type, prompt the user to enter the type and then perform an inorder traversal. If the given node matches the requested type, print out the details. Otherwise, simply skip to the next node in the traversal.

   Duplicate Entries

   The data given in the CSV file will be organized such that the tree will be balanced and organized by **armor class**. If there are duplicate entries, you should enter them to the same BST node using either a dynamic array OR a linked list. You will need to augment the BST `struct` to support this.

   Other Requirements

   - Read in a CSV file from the command line when running the program.
   - Convert the raw CSV data into the `Creature` `struct` and add it to a Binary Search Tree.
   - Make sure you properly release all of your allocated memory.
   - Format your code consistently.
   - Save your code for this problem under a directory named `problem1`.

   Creature `struct`

```c
typedef struct {
    char *name;
    char *type;
    int hp;
    int ac;
    int hp;
} Creature;
```

Example Run

```
1. Traverse Tree
2. Search
3. Traverse by Type
4. Exit
> 1
Select Traverse Type:
1. Preorder DFS
2. Inorder DFS
3. Postorder DFS
4. BFS
> 2
NAME: Centaur
HP: 45
AC: 12
SPEED: 50
TYPE: Monstrosity
...

1. Traverse Tree
2. Search
3. Traverse by Type
4. Exit
> 2
Enter name: Terrasque
Unable to find "Terrasque"

1. Traverse Tree
2. Search
3. Traverse by Type
4. Exit
> 3
Enter type: Fiend
NAME: Hell Hound
HP: 45
AC: 15
SPEED: 50
TYPE: Fiend
...
```

2. Take the code you created for the first problem and organize it following a basic file structure. Create a `Makefile` which has build configurations for the release and debug builds. It should also contain a rule for cleaning up the build folder. Your `release` and `debug` targets should build the code to the `build` directory (create it if it doesn't exist).

   File Organization

   Your functions and code should be organized as follows:

   - `creature_list.(c|h)` - The code related specifically to building the creature list program.
   - `bst.(c|h)` - Code related to the Binary Search Tree implementation.
   - `utils.(c|h)` - Code that provides some utility that isn't directly related to the specific program. This could include things like opening a file and testing it, removing a newline, etc.

   Other Requirements

   - Save your files in a directory called `problem2`.
   - The code files should be in a subdirectory called `src`.
   - Any allocated memory should be released properly.
   - Your code should be formatted consistently.

Create a `zip` file using the name template `LASTNAME_ID_A8.zip` which includes the all required code files. Submit the `zip` file through Canvas.