Programming Assignment 4

Binary Search Trees

PROGRAMMING

For this assignment, you will implement a binary search tree that can store any arbitrary struct in its nodes. You will start by completing the *recursive implementation* of the binary search tree (BST) in Worksheet 29. You will then modify it to so it can store arbitrary structures at each node, provided you have a an implementation of a **compare** and **print_type** function for that structure.

We are providing you with the following files:

- bst.c - This is the file in which you'll finish implementing the unfinished BSTree implementation. There is a main function in this file that you should modify to exercise your BST. The file contains several test cases such as **testAddNode**, **testContainsBSTree**, **testLeftMost**, **testRemoveLeftMost**, and **testRemoveNode**. Your implementation must pass all these test cases, and you are strongly encouraged to add your own tests as well.
- bst.h - This file should not be changed.
- structs.h - This file can be extended to test your code with different data types.
- compare.c- Put your compare and print functions in here.
- makefile.txt - Remember to rename this file to **makefile** (without the .txt extension).

Worksheet 29 will get you started on your implementation. However, there is one function not mentioned in the worksheet. You will be using the **compare** function to test two values of a node to determine if one is less than, greater than, or equal to the other. This function is similar to the **compareTo** function in the **Comparable** interface in Java. Rather than embedding it into the data structure, as you would do in Java, we will declare it and assume that the user has provided an implementation of **compare** in the file **compare.c**. That way, the user can substitute an appropriate compare function for any data type that they plan to store in the tree.

For example, if you want to store doubles in your tree, you might define the following struct to store at each node.

```
struct data {
    double num;
}
```

And then define your **compare** function to simply compare the two structs based on the **num** field. However, a user of your data structure could also do the following:

```
struct pizza {
  double cost;
  int numToppings;
  char *name;
}
```

And define a **compare** function that compares pizzas based on their name, cost, or number of toppings.

In this assignment, the **TYPE** macro is set to **void***. This means that the type of value stored in a node is a *void pointer*, which means it can be a "pointer to anything". Whenever you dereference a void pointer, you *must* cast it to a specific type. For example, you can cast a void pointer to **struct data*** (see the definition in **structs.h**), then dereference it to get the **struct data** value the pointer points to. It is the programmer's responsibility to ensure that they cast the void pointer to the same type of value that was actually stored at that location. You should read up on void pointers in your C reference or on the internet.

The **compare** function is needed because we need some way to compare the values stored in the tree nodes. Note that we can't just compare the pointers with the **>**, **<**, or **==** operations since this would just compare the memory addresses the pointers point to. Instead, we want to compare some field of the struct that the pointer points to (e.g. **val->number < otherVal->number**). The **compare** function will make changing this function for different structs much easier.

Finally, we strongly recommend that you add to the **main** function to exercise your binary search tree by adding, removing, and testing for elements.

QUESTIONS

In addition to the programming portion of the assignment you will also be answering some questions about binary search trees. Some of the questions will require you to write your answers on the tree in empty_graph.pdf. We will assume that any empty node boxes are non-existent nodes. For each of these problems, print out a copy of the blank tree, fill in the answers for the question, and please make sure to **WRITE THE QUESTION NUMBER** and your name on the sheet. If you know how to annotate PDF files directly, you can also do that instead of printing off the trees (this may be easier since you will have to submit a digital version of your solutions).

Question 1

Show the binary search tree built by adding numbers in this specific order, assuming the graph is empty to start with: 50, 20, 100, 10, 130, 30, 21.

Question 2

The trouble with binary search trees is that they can become unbalanced depending on the order that you insert values. Give an order for inserting the values 1 through 7 such that the resulting tree is a full binary search tree. This problem does not require you to fill in a tree, just write down the order in which you would insert the values. (Hint: it might be helpful to first draw the full tree to figure out how the values must be arranged, then you can determine the order to add them!)

Question 3

- Part A: Given the following tree, question3.pdf, show the tree after removing the value 16.
- Part B: Using the tree produced by Part A, show the tree after removing the value 17.

Question 4

The computer has built the following decision tree for the Guess the Animal Game, question4.pdf. The player has an animal in mind and will answer the questions shown in the tree. Each of the players responses is used to determine the next question to ask. For example, if the player is thinking of a sea turtle, she would answer Yes to the first (top) question, "does it live in the water?", which leads to the second question "is it a mammal?", to which she would answer No.

Show the decision tree that the computer should build after adding a Zergling and a question to differentiate it, "Does it eat space marines?", to the tree. The question and the animal should be added below existing questions in the tree. Note that Zerglings *do* eat space marines , *do not* live in the water, *do not* climb trees, and *are not* mammals (just in case you didn't know :-))

EXTRA CREDIT

In question 4 above, we use decision trees to store a database of questions and animals for playing the "Guess the Animal" game. The game proceeds by asking users a list of YES/NO questions to determine the animal users are thinking about. A sample session of the game is shown below:

```
Beginning animal game...
Please think of an animal!
> Does it live in the water?
Yes
> Is it a mammal?
No
> Is it a sea turtle?
Yes
That was great! You were thinking of a sea turtle.
```

For this extra credit question, you are asked to implement your own "Guess the Animal" program. Note that we intentionally leave out several details to give you the freedom of creatively building your program. Still, following are some hints for you to begin with.

- Your program should make use of the decision tree structure as in question 4 above. How do you represent your decision tree? What kind of node would be a question? What kind of node would be an animal?
- Use a file to store your list of questions and animals. Read your file into the decision tree every time your program begins.
- What would you do if your program does not know the animal the user is thinking of? Should it be able to learn new animals? If yes, then you should be able to update your database file.
- You can go to the website http://www.animalgame.com/ to try out an implementation of this game.

Submit a file **animal.zip** containing all of the files in your project, including a makefile for compiling it on flip.

GRADING RUBRIC

Programming

- Compile/Style = 15
- struct Node *_addNode(struct Node *cur, TYPE val) = 15
- int containsBSTree(struct BSTree *tree, TYPE val) = 10
- TYPE _leftMost(struct Node *cur) = 10
- struct Node *_removeLeftMost(struct Node *cur) = 10
- struct Node *_removeNode(struct Node *cur, TYPE val) = 15
- compare.c = 5

Questions

- Question 1 = 5
- Question 2 = 5
- Question 3 = 5
- Question 4 = 5

Extra Credit

- Guess the Animal program = up to 20 points depending on how cool it is :-)

What to turn in via TEACH AND CANVAS -

- bst.c
- compare.c
- struct.h (if you have changed it)
- answers.pdf (or .zip is OK too) (if you printed off the trees, please scan your answers)
-