

Due Feb 29 by 11:59pm **Points** 0 **Submitting** a file upload

Assignment 2: Creating an image database with C structs and pointers

Introduction

In this assignment you will practice C structs and pointers to implement a database for storing images. The database will allow users to query its contents to find all images that have particular attributes. More precisely, the database will keep track of three different attributes for each image: its predominant colour, shape and texture. The database will support three different functions:

- The user can add new images to the database by specifying the file name and the values for the three attributes.
- The user can query the database by specifying the values for the three attributes and the database will return the names of all image files that match these attributes.
- The user can ask to have the database contents printed to the screen (i.e. the names of all files stored in the database together with their attributes).

A simple image database

Your job is to write a program called `image_database` that will read commands from the keyboard and execute them. The program will support the following three commands:

1. The user can insert a new image by typing the following command to specify the name of the file to be added and the value for each of the three attributes:

```
i att1_value att2_value att3_value filename
```

The `i` stands for insert. So for example to insert an image file `tiny.ppm` that is best described by the color/shape/texture attributes yellow, square, and shiny, the user would type:

```
i yellow square shiny tiny.ppm
```

2. The user can ask the database to output the names of all files that match the attributes specified in the `q` command (`q` for query) as follows:

```
q att1_value att2_value att3_value
```

The database will output the names of all image files in the database with matching attributes on a single line. E.g. if the database contains two images named `i1.ppm` and `i2.ppm` that have attributes yellow, square, and shiny and the user types:

```
q yellow square shiny
```

then the database will output:

```
i1.ppm i2.ppm
```

If there are no images in the database with matching attributes, the database will print

```
(NULL)
```

3. The user can ask the database to print all its contents organized by attribute values by simply typing p:

```
p
```

The database will produce one line of output for each image in the database. For example, if the only database contents are two images named i1.ppm and i2.ppm that have attributes yellow, square, and shiny and one image named i3.ppm with attributes blue, round and shiny, the output would be:

```
blue round shiny i3.ppm
```

```
yellow square shiny i1.ppm
```

```
yellow square shiny i2.ppm
```

If there are no images in the database, i.e. the tree is empty, the database will print

```
(NULL)
```

The details of how these commands work will be explained below in the description of Tasks 1-4 that you will implement.

Task 1: Processing user commands

The first task is to complete the main function in image_database.c, which is currently empty. You want to change it such that it keeps reading user commands from the keyboard (stdin) line by line using fgets until EOF is reached. Each line is being tokenized by the function tokenize, which you will also need to complete, to make it easier to determine which command the user is executing (i, q or p) and what the arguments to the function are (e.g. the three attribute values in the case of a q command). You may assume that no input line will be more than BUFFER_SIZE characters long. For parsing, you will find the strtok function useful. Make sure to read and understand its man page.

The main function then checks whether the command is valid (i.e. it's one of i, q or p) and whether the number of arguments for this command is correct. You only need to check for the number of arguments and not whether the individual arguments make sense (e.g. whether a provided filename actually corresponds to an existing file or whether an attribute value is meaningful). If a command is invalid or the number of arguments is incorrect, the program prints an error message to stderr:

```
Invalid command.
```

Otherwise the appropriate function to execute the command is being called (i.e. either tree_insert, or tree_search, or tree_print) with the corresponding arguments.

After handling a command or printing an error message, the program should continue reading commands from the keyboard.

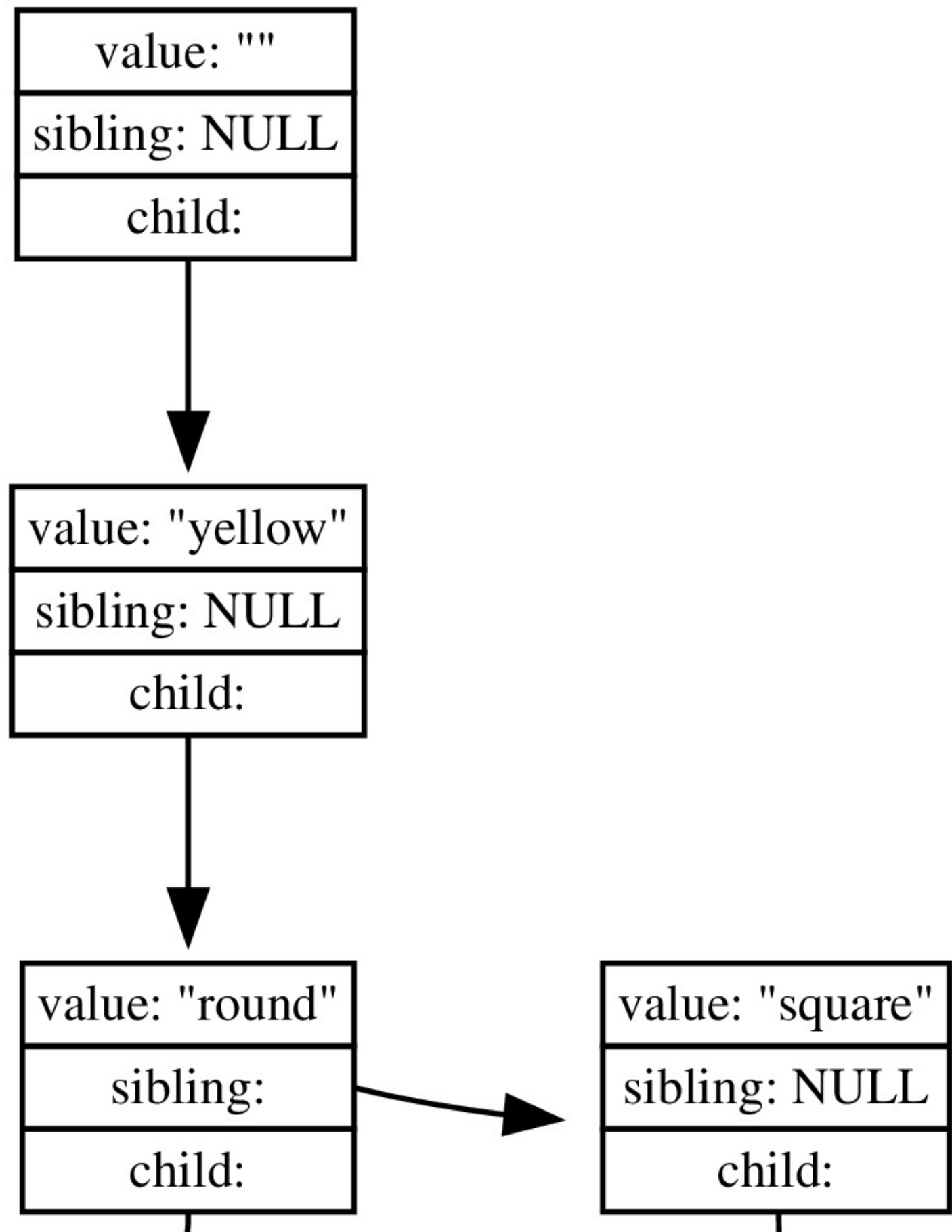
Task 2: Inserting an image into the database

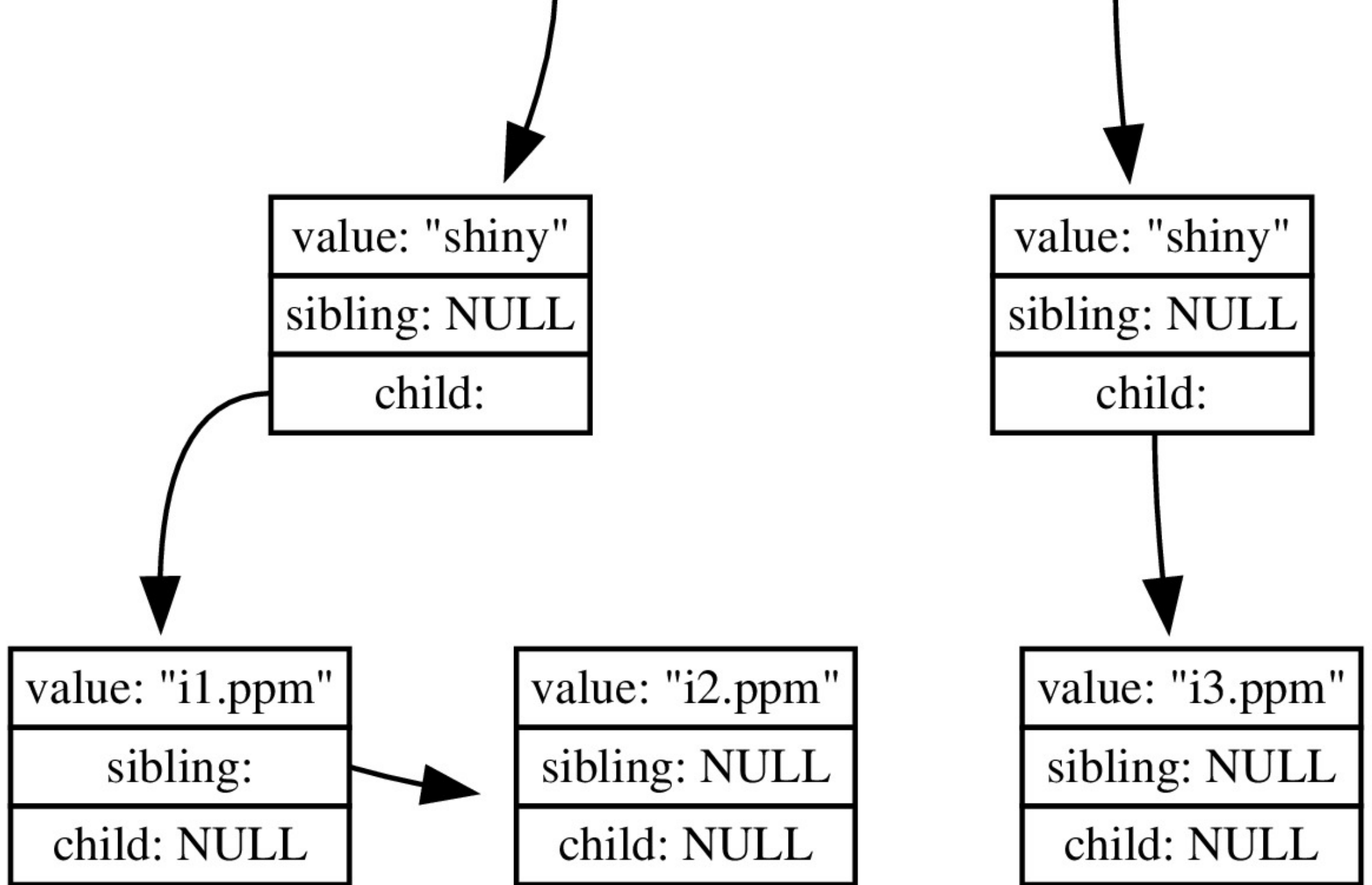
Your next task is to write a function `tree_insert`, which will insert an image with a given set of attributes into the database. If you take a look at the starter code, you will see that we have already defined a struct `struct TreeNode` that will be used to represent a node in the tree:

```
struct TreeNode {  
    char *value;  
    struct TreeNode *sibling;  
    struct TreeNode *child;  
};
```

The database will maintain information about the files in the form of a tree and the struct represents one node in the tree. Each level of the tree corresponds to one of the three attributes (color/shape/texture). The nodes in the bottom level, i.e. the leaf nodes, correspond to actual files. The value field of these nodes will be the filename. The child pointer will be NULL. The value field of the intermediate nodes is used to hold the value of the corresponding attribute.

For example, the tree for a database that contains two files named `i1.ppm` and `i2.ppm` with attributes yellow, round, shiny and one file named `i3.ppm` with attributes yellow, square, shiny would look as follows:





An example of the tree database.

The node at the top of the tree with the empty string as value is the root node, which the main function in the starter code is already creating for you. All other nodes will be created by `tree_insert` when the users inserts images into the database.

When inserting nodes in the lists, insertion should be done in sorted order based on string comparison with `strcmp`. You do not have to handle the case where an image already exists in the database, i.e. you can assume that the image to be inserted does not already exist in the database.

Task 3: Querying the database

The next task is to complete the function `tree_search`, which will query the database for images with the specified attributes. The output should be a list of all matching filenames printed to `stdout` in a single line. The filenames should be printed in sorted order, which should be straight-forward, since in Task 2 we asked you to insert any elements into lists in sorted order.

Task 4: Printing the database contents






The final task is to complete the function `tree_print`, which will print the entire contents of the database. For each file in the database there will be one line in the output that has the following format:

```
att1_value att2_value att3_value filename
```

where `att1_value`, `att2_value` and `att3_value` are the values of the 3 attributes and the last field is the name of the file.

Starter Code

We are providing a Makefile and five source code files as starter code, which you are expected to modify:

- [Makefile](#)
- [image_database.c](#) 
- [tree.h](#) 
- [utils.h](#) 
- [tree.c](#) 
- [utils.c](#) 

Some important hints

Here are a few things to remember as you implement and test the functions:

1. As in A1, it might be cumbersome for testing to enter all the commands at the keyboard. So instead you could create a file that has a series of commands (e.g. to add images to the database) and then use input/output redirection to have your program read commands from the file instead of the keyboard.
2. If you provide input by typing at the keyboard, you provide EOF by typing ctrl-D in a new (empty) line.
3. Make sure to carefully look over the starter code and any comments we provide in the code to help you understand what you need to implement.
4. You may want to add your own helper functions to tree.c.
5. Test thoroughly before submitting a final version. We are using automated grading tools, so your program must compile and run according to the specifications. In particular, your program must produce output in the specified format.
6. Man pages are the best source of information for the system and library calls you need. Start by reading them carefully. For example, you may want to start with the man pages for strcmp and strtok. Also remember that for the midterm and final exam we will provide you with the man pages for any commands you need to use. So this is good practice for the exams as well.
7. Build up functionality one small piece at a time and commit to svn often.
8. Check for errors on every system or library call that is not a print. Send any potential error messages to stderr.
9. Be very careful to initialize pointers to NULL and to make sure that you copy any strings that you use.

Submission and Marking

Your program must compile on a CS lab machine, so please test it there before submission. We will be using gcc to compile program with the flags -Wall and -std=gnu99, as demonstrated in the Makefile. Your program should not produce any error or warning messages when compiled. As in assignment 1, programs that do not compile will receive a 0. Programs that produce warning messages will be penalized.