CS 3432 – Computer Organization

Lab 2 Author: Robert Alvarez

**Instructions:**

Assignments must be submitted through GitHub Classroom. main.c file must contain your name, last modification date, due date, course, and instructor name. Only assistance from the instructor, TA, or IA will be permitted.

**Part 1 – Familiarization with an array of pointers and characters.**

This part of the lab would focus on implementing your own “string.h” library and a tokenize function using some of the functions.

my\_string: (Note: You are not allowed to index the pointers☺) [5 points each]

1. Implement strlen.
   1. size\_t str\_len(const char \*s);
2. Implement strcmp and strncmp.
   1. int str\_cmp(const char \*s1, const char \*s2);
   2. int str\_n\_cmp(const char \*s1, const char \*s2, size\_t n);
3. Implement memcpy.
   1. void \*mem\_cpy(void \*restrict dst, const void \*restrict src, size\_t n);
4. Implement strchr.
   1. char \*str\_chr(const char \*s, int c);
5. Implement strpbrk.
   1. char \*str\_p\_brk(const char \*s, const char \*charset);
6. Implement strsep.
   1. char \*str\_sep(char \*\*stringp, const char \*delim);
7. Implement strcat.
   1. char \*str\_cat(char \*s1, const char \*s2);

Tokenizer: [10 points]

1. Make a function defined as “char \*\*tokenize(char \*str, const char \*delims)”.
   1. This function takes two pointers to an array of null-terminated characters.
      1. str: Array of characters split into tokens based on delims characters.
      2. delims: Array of characters used to split “str.”
   2. This function makes a null-terminated array of character pointers with k, not null pointers where k is defined as the number of times when str[i] == delims[j] where 0 <= i < length of str and p <= j <= length of delims.
      1. The pointers in the return value most point to memory places in str. Therefore, you must terminate each substring null and add a pointer to the beginning of the substring in str.

Example:

If str starts at 0x100, in memory, with the value of “Dodge,Challenger.Charger/Durango” and delims = “,./”. Then, tokenizer(str,delims) must return an array with pointers pointing to 0x100, 0x106, 0x111 and 0x119.

File/Libraries:

1. my\_string.h and my\_string.c.
2. tokenizer.h and tokenizer.c.
3. Makefile.
   1. It would be provided to you to compile and link main.c with the required files.

Permitted standard libraries:

* stdlib.h - size\_t

No standard C library can be included if you want to use any other library or other variables/functions from stdlib.h, contact the AI or TA, and we can help you to figure out how to do the same thing without that library or function.

Deadline (GitHub Classroom): February 7th, 2023, by 11:59 pm.

1. Source code (Only .c and .h files, and 1 Makefile)

Grading: (Total of 45 points)

Each function has the weight grade value of its implementation. We would run each function 20 times with randomly generated values and compare your output with ours. Based on the match percentage, it is how much you would get for it. For example, if your str\_len() output is only correct for 15 out of 20 runs, you would get only (15/20)\*5 points = 3.75 out of the 5 maximum possible.

***NOTE:***

Once the deadline is passed, the answer for part 1 will be given so everyone can start part 2 on time. This means that every submission after the deadline would not be considered and if that was your only submission then you get a zero.

**Part 2 – Make balance BST from sorted File lines.**

A k lines file where each line has the structure “BRAND,MODEL1,MODEL2,…,MODELn\n” would be given to the main executable as follows: “./main Inventory.txt”. Each line in the file is ordered in lexicographic order using “BRAND.” Also, “BRAND” and “MODEL” may be repeated between lines but not within the line itself. The “int main(int argc, char \*\*argv)“ function in “./main” must do the following:

1. Use “bst\_t \*create\_bst(FILE \*file)” function declared in BST.h to process the file as follows:
   1. Read lines from a file and make a balance BST where each node has “BRAND” used as a “key” and a list of char \* call models with all “MODEL” s.
      1. Do not implement a balanced BST algorithm like “AVL Trees” or “Red-Black Trees.” Instead, by iterating “in order,” a sorted array would also make a balanced BST.
   2. DO NOT hardcode the number of lines in the file.
      1. Code would be tested with a different file with different brands, models, number of lines, etc.
2. Once the balance BST has been created, your program must offer a menu by calling “void print\_menu(void)”, where we can select any of the following options:
   1. Save BST. [void write\_bst(const bst\_t \*bst, FILE \*file)]
      1. Ask for a string that represent the file name starting at the current directory and open it.
      2. Then, save the current bst with the same format of “BRAND,MODEL1…\n”.
      3. DO NOT free the bst.
   2. Insert. [int add\_line\_to\_bst(bst\_t \*bst, char \*line)]
      1. A string with the format “BRAND,MODEL1,…MODELn” would be given, and it must be added to the appropriate BST node or created a new one if needed.
      2. BST doesn’t have to continue to be balanced by the addition of this new information.
      3. Remember, if a model is repeated on the brand, you must only include it once.
   3. Print BST. [void print\_bst(const bst\_t \*bst)]
      1. This option would print the BST nodes in ascending order by the module name using the format “BRAND: MODEL1, MODEL2, …, MODELn”.
   4. Get BST height. [int depth(const bst\_t \*bst)]
      1. It must return an integer with the tree height.
   5. Get the number of nodes in BST. [size\_t number\_of\_nodes(const bst\_t \*bst)]
      1. It must return an integer with the number of not NULL nodes in the BST.
   6. Exit. [void free\_bst(bst\_t \*bst)]
      1. Free everything in the heap memory before terminating the program.
      2. Return 0 upon success or 1 if anything fails while freeing the heap memory.

REMEMBER: If anything goes wrong at any point in the program, EVERYTHING MUST BE FREE FROM HEAP MEMORY. Failing to do so at EVERY POSSIBLE FAILING POINT would deduct you 15 points from the final grade.

**File/Libraries:**

1. Answers from Lab 2 would be given and needed for this lab.
2. BST.h and the structure of BST.c have been provided with the function that would be tested to grade your lab. Therefore, DO NOT modify function names, or you will get a zero from the automatic grading tool.
3. Create a Makefile with a target call “main” that makes an executable for “main.c” and links all needed files.
4. You can make as many .c/.h files and static functions in any file as needed.

Permitted standard libraries:

* stdio.h – FILE, ssize\_t, getline(), printf() and fprintf()
* stdlib.h – size\_t, free(), malloc(), calloc() and realloc().

No other standard C library can be included. If you want to use any other library, contact the AI or TA and we can help you to figure out how to do the same thing without that library or function.

Deadline (GitHub Classroom): February 21st, 2023, by 11:59 pm.

* + - 1. Source code (Only .c and .h files, and 1 Makefile)

Grading: (Total of 55 points)

*5 Points*: Compile and generate the main executable by calling “make main” and “make all.” Able to run the executable created by typing “./main FILENAME” in the terminal.

*30 Points*: Generate a balance BST when loading the file.

*10 Points*: New lines are inserted correctly in BST (b in the menu).

*5 Points*: Free everything from heap memory when using “f” in the menu.

*5 Points*: Save and print BST with the same brand model format (“a” and “c” in the menu).

*5 Points*: Correct number of nodes and height (“d” and “e” in the menu).

This would be tested multiple times while testing the other menu options.

*5 Points*: Perform every binary search with two parameters instead of three. Only use node and high or node and length instead of node, low, and height parameters.

**Extra credit:** (25 points)

*5 Points*: Makefile is not hardcoded, meaning it uses $@, $<, $^, and ${OBJ} as needed.

*5 points*: Retrieve a model using binary search (only use two parameters).

This would require the models to be always sorted.

*5 points*: Use efficient array size.

Request memory for only 4 pointers; when more is needed, use realloc with twice the size. Suppose the array has more than one fourth of the space free, free half the array.

*10 Points*: Able to compile every .c file without warnings when the following flags are used:

-Wall -Wextra -Werror=format-security -Werror=implicit-function-declaration

-Wshadow -Wpointer-arith -Wcast-align -Wstrict-prototypes -Wwrite-strings

-Waggregate-return -Wcast-qual -Wunreachable-code

(1 point would be deducted out of the 10 possible for each warning; if more than 10 warnings are generated, then no points for this section are given).