CS 3432 – Computer Organization

Lab 2.1 – Arrays Author: Robert Alvarez

**Instructions:**

Assignments must be submitted with the same person as Lab 1, unless the other person drops the course, through Blackboard. Note that only assistance from your teammates, instructor, TA, or IA will be permitted.

**Part 1 – Familiarization with manipulating an array of characters.**

This part of the lab would focus on implementing your own “string.h” library. For the following instructions, you are not allowed to use indexation of an array. For an explanation of how these functions work (assumptions, input/output examples) refer to the level 3 manual documentation for each function.

my\_string:

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

Note: The function strcat() is not deprecated, but closed. So, when testing your function, test it with strings of at least size 2.

Points distribution:

* There would be no points given to any function that uses array indexing.
* For example: if “str” is pointing to a character in a sequence of character then doing str[i] or \*(str + i) to access the i offset character from \*str is not allowed.
* All the above function must be declared in a header file name “my\_string.h” and define in its respective c file.

**Part 2 – Familiarization with an array of pointers.**

[15 points] Tokenizer:

1. Make a function defined as “char \*\*tokenize(char \*str, const char \*delims)”.
   1. This function takes two pointers where each array of characters is null terminated.
      1. str: Array of characters to be split into tokens based on delims characters.
      2. delims: Array of characters used to split “str.”
   2. This function must return a null-terminated array of pointers of size k+1 where k is the number of times that a character from “delims” was found in “str”.
      1. The pointers in the return value most point to memory places in “str”. Therefore, you must null terminate each substring in “str” and point to the beginning of the substring.

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, 0x117 and 0x125.

[-5 points]: The “malloc” function is called more than once in any possible scenario.

**Part 3 – Familiarization with structs and pointers to structs.**

This part of the lab consists of implementing a general array list “class” that support basic array manipulations. The following functions must be implemented with the given time complexity.

General documentation:

* ArrayList\_t is a struct that contains at least an array of pointers where each pointer points to an element of the data structure.
  + - 1. [MANDATORY] [2 points] Initialization of the struct.

1. Function declaration: “ArrayList\_t \*AL\_init(void);”
2. Task: Initialize an array list that can store 0 elements.
3. Return value: Pointer to an ArrayList\_t variable if successful, NULL otherwise.
4. Run time: O(1).
   * + 1. [MANDATORY] [2 points] Delete the struct.
5. Function declaration: “int AL\_free(ArrayList\_t \*AL, int (\*delete\_data)(void \*));”
6. Task: Delete all AL elements store in heap including AL itself.
7. Return value: Return 0 if everything was free successfully, 1 otherwise.
8. Run time: O(n).
   * + 1. [MANDATORY] [2 points] Print all elements in struct.
9. Function declaration: “void AL\_print(ArrayList\_t \*AL, void (\*print\_data)(void \*));”
10. Task: Print all AL data elements in AL.
11. Return value: N/A.
12. Run time: O(n).
    * + 1. [3 points] Retrieve an element anywhere in the data structure.
13. Function declaration: “void \*AL\_get\_at(ArrayList\_t \*AL, size\_t i);”
14. Task: Return the pointer that points to the element in AL at index I.
15. Return value: Pointer to the element if retrieve successfully, NULL otherwise.
16. Run time: O(1).
    * + 1. [3 points] Overwrite what is contain at an index.
17. Function declaration: “int AL\_set\_at(ArrayList\_t \*AL, size\_t i, void \*elem, void \*(\*copy\_data)(void \*), int (\*delete\_data)(void \*));”
18. Task: Overwrite the element pointed to at index i in AL.
19. Return value: Return 0 if overwrite successfully, 1 otherwise.
20. Run time: O(1).
    * + 1. [3 points] Pre-append.
21. Function declaration: “int AL\_insert\_first(ArrayList\_t \*AL, void \*elem, void \*(\*copy\_data)(void \*));”
22. Task: Add elem as the first element in AL.
23. Return value: Return 0 if pre-append successfully, 1 otherwise.
24. Run time: O(1) amortize.
    * + 1. [3 points] Delete first element.
25. Function declaration: “int AL\_delete\_first(ArrayList\_t \*AL, int (\*delete\_data)(void \*));”
26. Task: Delete first element in AL.
27. Return value: Return 0 if first element was deleted successfully, 1 otherwise.
28. Run time: O(1) amortize.
    * + 1. [3 points] Append.
29. Function declaration: “int AL\_insert\_last(ArrayList\_t \*AL, void \*elem, void \*(\*copy\_data)(void \*));”
30. Task: Add an element as the last element of AL.
31. Return value: Return 0 if appended successfully, 1 otherwise.
32. Run time: O(1) amortize.
    * + 1. [3 points] Delete the last element.
33. Function declaration: “int AL\_delete\_last(ArrayList\_t \*AL, int (\*delete\_data)(void \*));”
34. Task: Delete the last element in AL.
35. Return value: Return 0 if deleted successfully, 1 otherwise.
36. Run time: O(1) amortize.
    * + 1. [3 points] Insert an element at a given index.
37. Function declaration: “int AL\_insert\_at(ArrayList\_t \*AL, size\_t i, void \*elem, void \*(\*copy\_data)(void \*));”
38. Task: Insert an element at index i in AL.
39. Return value: Return 0 if the element was added successfully, 1 otherwise.
40. Run time: O(n).

Note:

* If insertion location is greater than the store number of elements in AL, add elem at the end of the list.
  + - 1. [3 points] Delete an element at a given index.

1. Function declaration: “int AL\_delete\_at(ArrayList\_t \*AL, size\_t i, int (\*delete\_data)(void \*));”
2. Task: Delete an element at index i in AL.
3. Return value: Return 0 if the element was added successfully, 1 otherwise.
4. Run time: O(n).

Note:

* If the deletion location is greater than the store number of elements in AL, return failure case.

Notes:

* [Amortization analysis](https://ocw.mit.edu/courses/6-046j-design-and-analysis-of-algorithms-spring-2015/resources/mit6_046js15_lec05/).
* If AL is not initialized, return failure case for the function.
* If trying to delete from an empty list, return success case.

File/Libraries:

1. my\_string.h and my\_string.c.
2. tokenizer.h and tokenizer.c.
3. ArrayList.h and ArrayList.c.
4. Makefile.

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 figure out how to do the same thing without that library or function.

Deadline (Blackboard): February 20th, 2023, by 11:59 pm.

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

Grading: (Total of 73 points)

Each function has the weight grade value of its implementation. We would run each function multiple times with randomly generated values and compare your output with ours. Based on the match percentage, is how much you would get for it.

[-5 points] If a header file is processed for every file which “include”s it.

* Hint: Look at the pre-processor keywords used in C.

[-5 points] No Makefile or a Makefile that does not generate all the object files requested (must generate no executable) when running “make” in the terminal.

[-10 points] A non-authorized standard library was included in any file or anything else besides size\_t is used from stdlib.h.

***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, you get a zero.