CSCE 313 Programming Assignment 1

Buddy Allocator

Aldo Leon Marquez UIN: 326004699

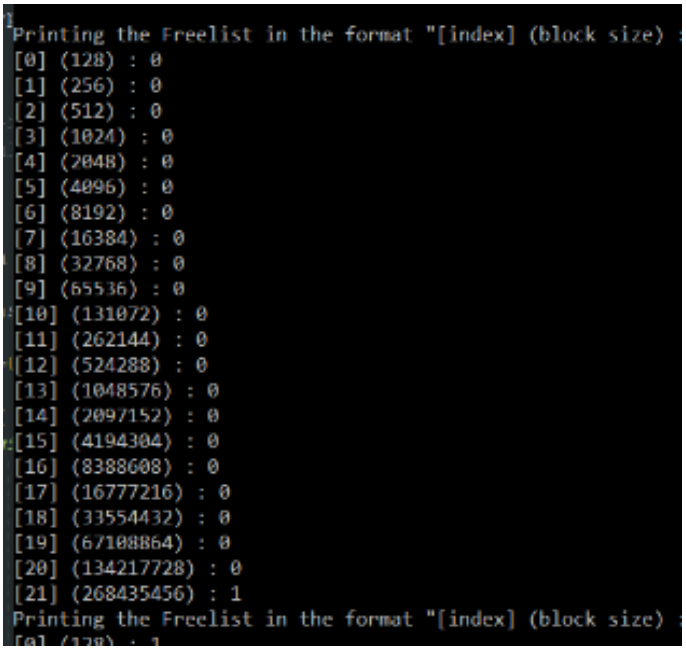
The objective of assignment 1 is to, make simple memory allocator that will help us understand and practice how to manage memory for a process and the concepts surrounding it.

As with any other class we need a constructor, for this project the constructor initializes the reserved memory block, initializes the correct sizes for the memory block and the supporting vector list, after this we will need to implement the two core functions of an allocator: an allocate function and a free function. The Allocate function will return a pointer to the reserved memory position (with an overhead offset) and then will be later restored with a free function.

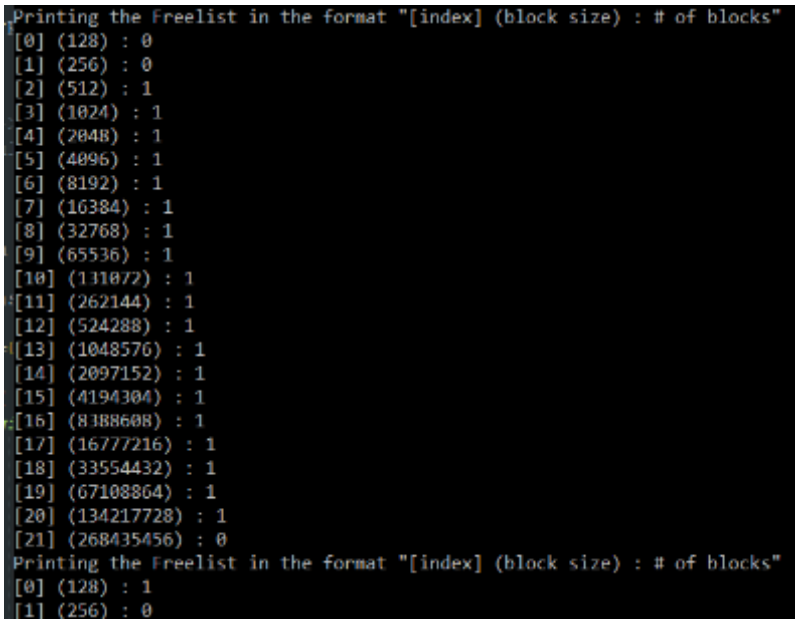
Although it sounds and looks simple (nope) this functions also require helper functions. For the Allocate function we need a helper function that splits the free block into the required block size, the function reorganizes the pointers and block size data. Now for the free function we need two helper functions, A merge function that combines recently freed block (as much as much as possible) and a function to determine whether or not two free blocks should be merged.

The process of filling and freeing the block will look like this

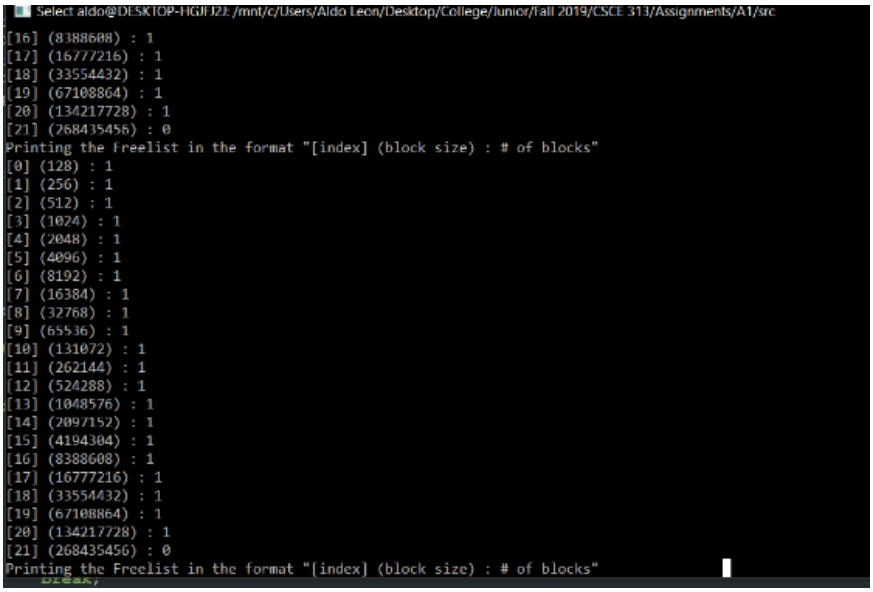
1) A memory block of with no changes 256MB.



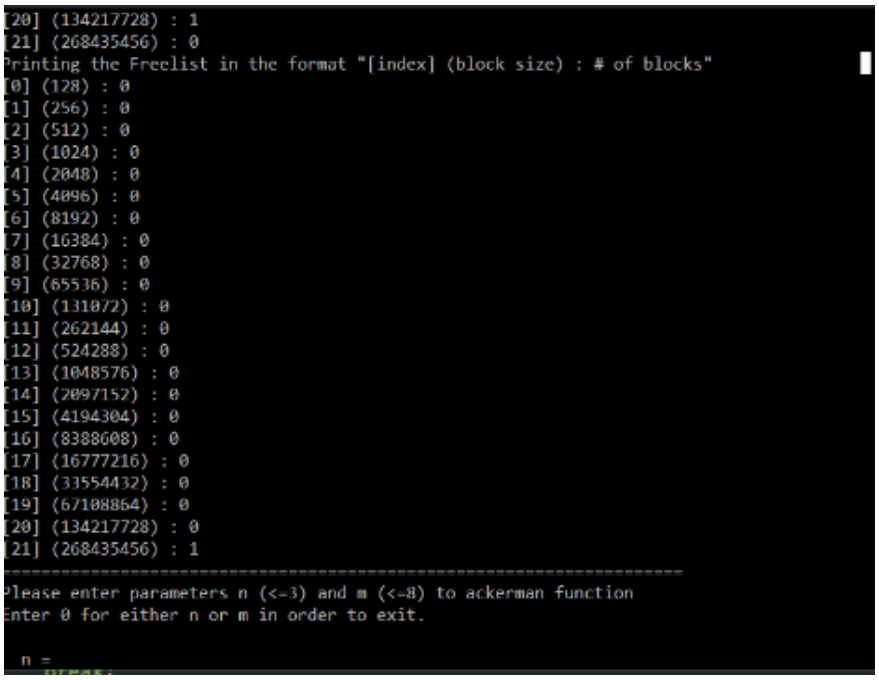
2) The memory Block with some 128 bytes block allocated.



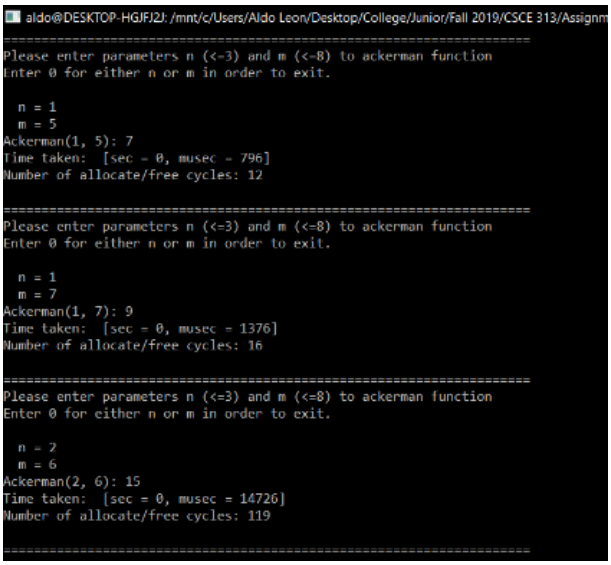
3) The memory block after freeing the blocks , before merging.

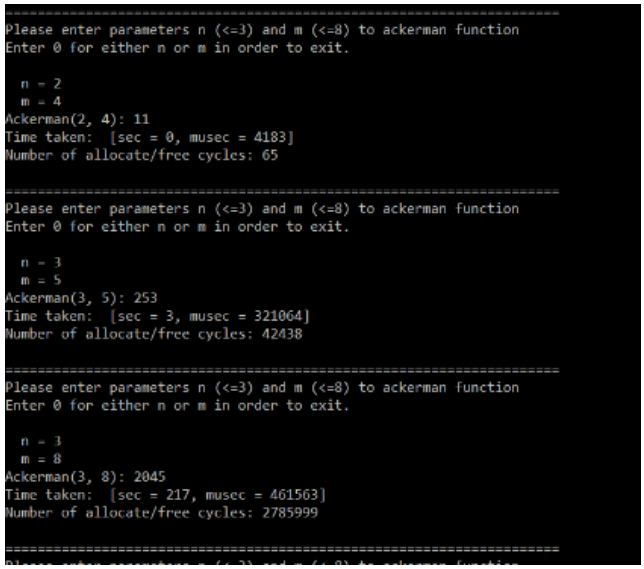


4) Block after merging.



To check correctness and performance of the code, we run the Ackerman Function for n <= 3 and m <= 8. I'm including some examples of the function returning the desired result





To analyze the performance, I'm including some charts and graphs of the usage and runtime of the Ackerman function (Testing several possible combinations). Regarding the performance: The values recorded were form a different run were the only applications open on the computer were the Linux shell and notepad ++, the speed of the function does improve with this condition.

Values of n=1

M	Time	Cycles	musec	Ackerman(n,m)
1	0	4	44	3
2	0	6	230	4
3	0	8	674	5
4	0	10	586	6
5	0	12	2229	7
6	0	14	428	8
7	0	16	2316	9
8	0	18	1770	10

Values for n=2

M	Time	Cycles	musec	Ackerman(n,m)
1	0	14	1151	5
2	0	27	5278	7
3	0	44	3937	9
4	0	65	5220	11
5	0	90	10382	13
6	0	15	8834	15
7	0	152	13044	17
8	0	189	12562	19

Values for n=3

M	Time	Cycles	musec	Ackerman(n,m)
1	0	106	7239	13
2	0	541	44291	29
3	0	2432	170658	61
4	0	10307	774000	125
5	3	42438	101125	253
6	12	172233	479529	509
7	50	696964	576765	1021
8	187	2785999	340573	2045

Chart for Cycles (axis y in log base 10 scale)

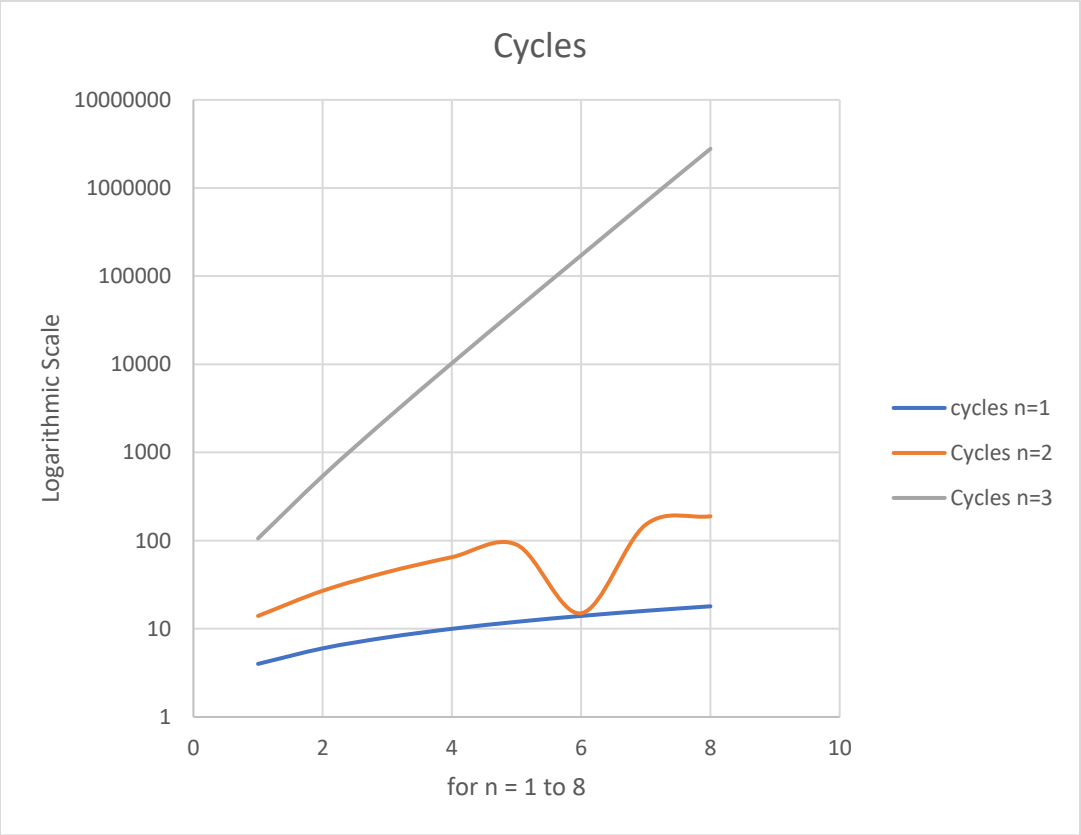
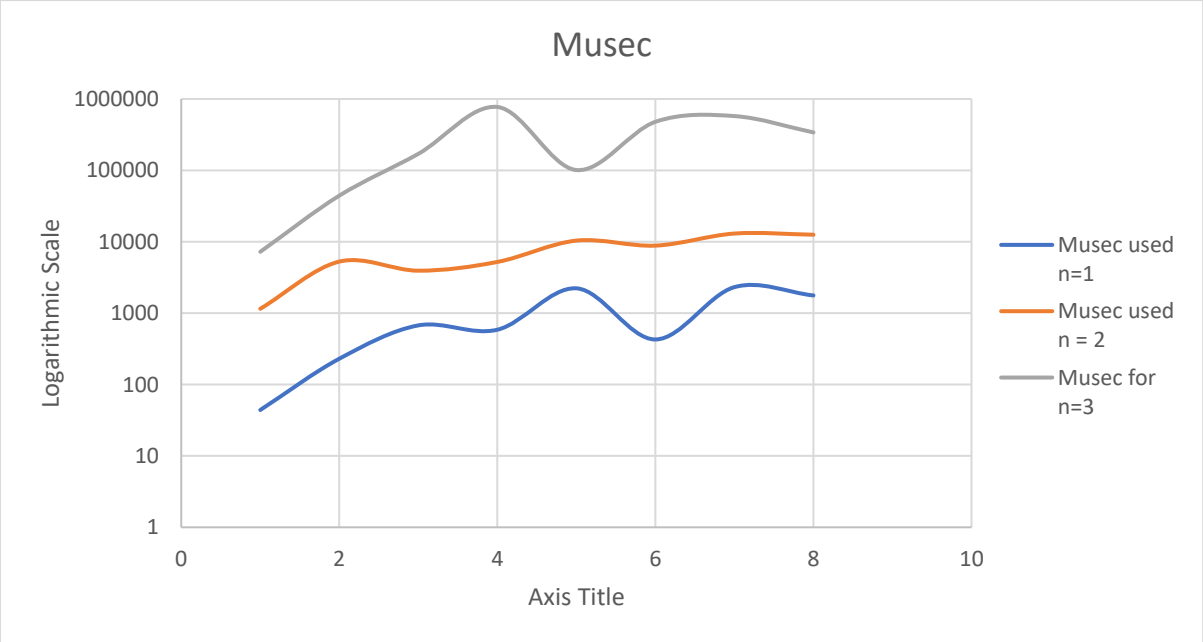


Chart for Musec (axis y in log base 10 scale)



Notes on Performance:

- Before running the final test, while implementing the getopt operation I modified something in the code that ruined my Ackerman (2,8) for 128MB, So I was forced to use a minimum of 256MB to run it.
- I'm definitely sure that speed and usage could be improve, although I didn't use a bool bit to determine if a block is free or not, I believe the Blockheaded size could be reduced even further.
- Also, to improve speed and usage some parts of my code could be trimmed off of refined, because I ended up implementing some redundant operations/check, but I decided not to mess any further after the getopt incident.

Implementation Notes:

- No use of a free bit;
- To determine a buddy, I use the offset from the start to determine if it's a left or left buddy, a left index has an even index and an odd buddy a right one.
- The merge function work recursively