# **2023 Fall CSED 211 Lab Report**

Lab: Malloc lab

Student number: 20220302

Name: 김지현

1. Introduction

* In this lab, I have implemented a malloc/free function regarding memory efficiency. We learned three types of listing allocated and free blocks in class: implicit, explicit, and segregated. In the below code, I have implemented a segregated list.

1. System Design

* Global variables/functions :
  + These basic constants and macros are referenced from the textbook (figure 9.43).

A screenshot of a computer program

Description automatically generated

* + These are added to implement segregated list from the simple malloc function. First, INITCHUNKSIZE is 64 and used in mm\_init function to extend the heap. Next, LISTLIMIT is a limiting value of the class of segregated list.
  + SET\_PTR macro stores the value of ptr at the pointer location p. NEXT\_PTR macro type-casts ptr to a char pointer. PREV\_PTR macro type-casts ptr to a char pointer and returns the address of the pointer plus +WSIZE
  + SEG\_NEXT\_PTR macro type-casts ptr to a char double pointer and returns the value at that address. SEG\_PREV\_PTR macro type-casts PREV\_PTR(ptr) to a char double pointer and returns the value at that address.
  + Heap pointer point to the heap memory. Seg\_free\_list array pointer manages multiple free blocks within a certain capacity limit of LISTLIMIT.
  + I’ll explain the user defined function later.

A screenshot of a computer program

Description automatically generated

* extend\_heap function – extends the heap with a new free block
  + referenced from the textbook (figure 9.45)
  + First, align to even number of words and requests heap spaces.
  + And initialize the free block header/footer/epilogue header.
  + Then, added this new freed block to the segregated free lists.
  + Then, check for the possible coalescing.

A computer screen shot of a black screen

Description automatically generated

* insert\_free\_block function adds the new freed block to the segregated free lists.
* First, we must determine which segregated free list class the free list has to be in. In other words, the index must be determined.
* I have managed the segregated lists by dividing the index by the power of 2.
* Let the next\_ptr hold the first block of the previously determined class of segregated free lists.
* Then, consider four cases from the textbook Figure 9.40, where different allocation or freed cases of previous and next blocks are considered, and reallocate appropriate pointers.

A screen shot of a computer screen

Description automatically generated

* Delete\_free\_block function deletes a block from the segregated free lists when that block is allocated or coalesced.
  + First, find the index from segregated free list for the certain ptr to be deleted.
  + Next, consider the four cases from figure 9.40 to reallocate pointers.

A computer screen shot of a program code

Description automatically generated

* Coalesce function combines freed function if they are adjacent.
  + Referenced from the textbook (figure 9.46)
  + Again, four cases are considered and delete\_free\_block and insert\_free\_block function is newly added from the reference.
  + Case 1: If both the previous and next blocks are allocated, the function simply returns the pointer to the current block (`bp`). No coalescing is performed in this case because there are no adjacent free blocks.
  + Case 2: If the previous block is allocated and the next block is free, the function deletes the current block and the next block from the free list. The size of the current block is then increased by the size of the next block, and the header and footer of the current block are updated with this new size. The pointer to the current block is then returned.
  + Case 3: If the previous block is free and the next block is allocated, the function deletes the current block and the previous block from the free list. The size of the current block is then increased by the size of the previous block, and the footer of the current block and the header of the previous block are updated with this new size. The pointer to the previous block is then returned.
  + Case 4: If both the previous and next blocks are free, the function deletes the current block, the previous block, and the next block from the free list. The size of the current block is then increased by the size of both the previous and next blocks, and the header of the previous block and the footer of the next block are updated with this new size. The pointer to the previous block is then returned.

A screen shot of a computer program

Description automatically generated

* Find\_fit function finds a suitable block of memory in a segregated list that can fit the requested size (which is asize)
  + First, loops to search for a free block in the segregated lists.
  + Returns the block that can be allocated if found.
  + Otherwise, return null.

A computer screen with text and images

Description automatically generated

* Place function places a requested block size into a free memory block
  + Referenced from the textbook (Practice Problem 9.9)
  + First, delete a certain block from the segregated free lists of free block.
  + If the current block is significantly larger than the requested size, the function splits the block. The requested portion is marked as allocated and the remainder is kept as a new free block and added to the free list.
  + If the current block is not significantly larger than the requested size, the function allocates the entire block.

A computer screen shot of a program code

Description automatically generated

* mm\_init function initiliaze the malloc package.
  + This function is referenced from the textbook (figure 9.44).
  + First, initialize all segregated free lists to NULL.
  + Then, create initially empty heap using PUT function.
    - Alignment padding, prologue header/footer, and epilogue header is added.
  + Then, extend the empty hep with a free block of CHUNKSIZE bytes.
  + In all cases, return -1 if any of these tasks are erroneous.

A computer screen shot of a program

Description automatically generated

* Mm\_malloc function allocate a block by incrementing the brk pointer.
  + Referenced from the textbook (figure 9.47)
  + If the requested size is 0, the function simply returns NULL and ends.
  + If the requested size is less than or equal to `DSIZE`, the block size (`asize`) is set to `2\*DSIZE`.
  + Otherwise, the block size is adjusted to include overhead and alignment requirements.
  + The function then searches the free list for a suitable block using `find\_fit(asize)`, which returns the pointer to the first suitable block, or NULL if none is found. If a suitable block is found, the function allocates it using `place(bp, asize)`, then returns the pointer to the allocated block. If no suitable block is found, the function extends the heap to gain more memory.
  + The size of the extension (`extendsize`) is set to the larger value between the requested block size and `CHUNKSIZE`. The `extend\_heap` function returns the pointer to the first block of the extended heap, or NULL if the heap extension fails.
  + Finally, the block is allocated in the newly extended heap, and the pointer to the allocated block is returned.

A screenshot of a computer program

Description automatically generated

* Mm\_free function frees the blocks that does nothing.
  + Referenced from the textbook (figure 9.46)
  + It first retrieves the size of the block using the `GET\_SIZE` macro and the header of the block.
  + Then, it updates the header and footer of the block to indicate that it is no longer allocated.
  + After that, it inserts the block into the list of free blocks with the `insert\_free\_block` function.
  + Lastly, it calls the `coalesce` function to merge this block with any adjacent free blocks, thus minimizing fragmentation.

A computer screen shot of text

Description automatically generated

* mm\_realloc function reallocate a block of memory with a new size.
  + Firstly, it saves the old pointer and declares a new pointer and a variable to hold the size for copying.
  + Then, it allocates memory of the new size for the new pointer using `mm\_malloc`. If this allocation fails, the function returns `NULL`. It then gets the size of the old block and adjusts the `copySize` to the smaller of the old size and the requested new size. This is to ensure that no data is lost in the case where the new size is smaller than the old size.
  + After this, it copies the contents from the old block to the new block using `memcpy`. Then, it frees the old block using `mm\_free`.
  + Finally, it returns the new pointer, which now points to a block of the requested size.

A computer screen shot of a program code

Description automatically generated

* The result of the code turns out to be this:

A screenshot of a computer screen

Description automatically generated