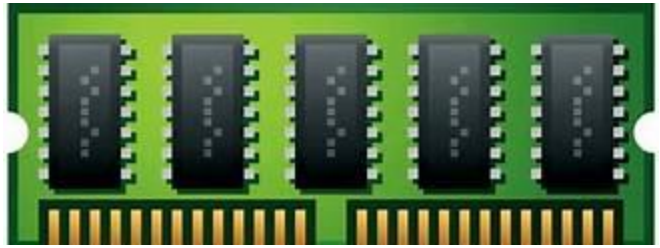
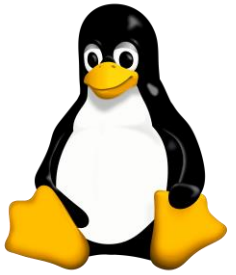
life.augmented

Heap Memory Manger



Advanced system programming under Linux training
@STMicroelectronics
Under supervision: Eng. Reda Maher

Omar Elshamy



Outline

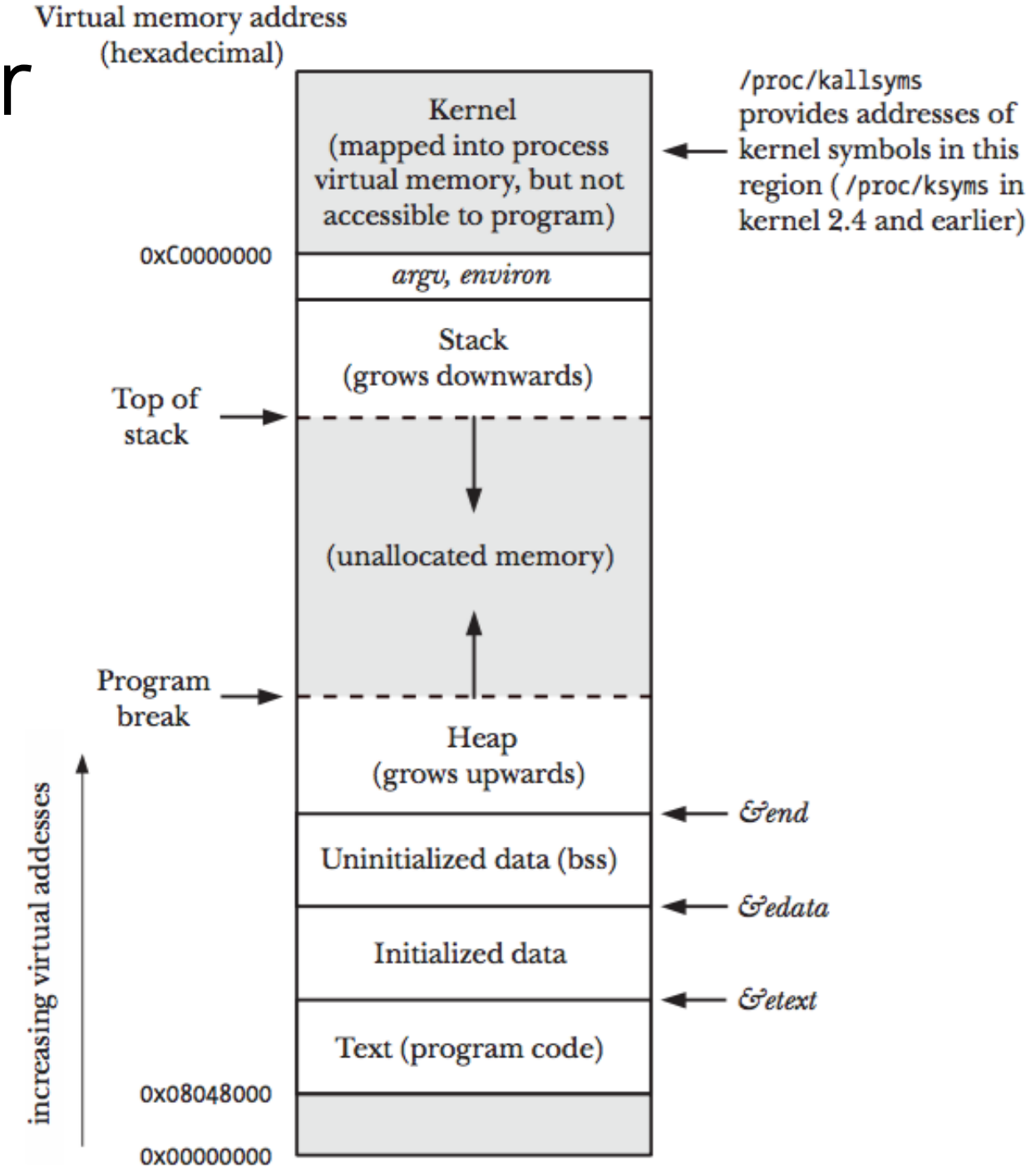
1. Intro to Heap Memory Manger
2. Design of HMM
3. Flowchart & Implementation
4. Testing
5. Make file

1- Heap Memory Manger

Heap: are memory area allocated to each program. Memory allocated to heap can be dynamically allocated, unlike memory allocated to stacks.

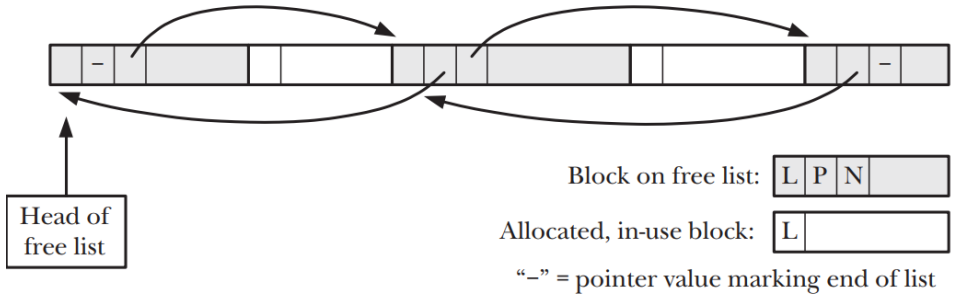
Memory layout consists of:

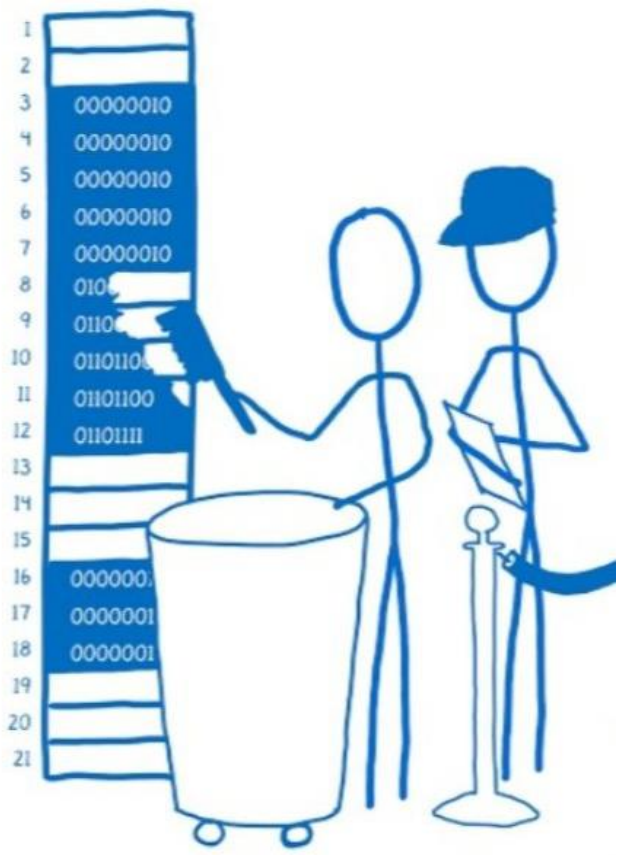
- Text (read-only, sharable).
- Data.
- BSS (Block Started by Symbol).
- Stack.
- Heap (Program Break)



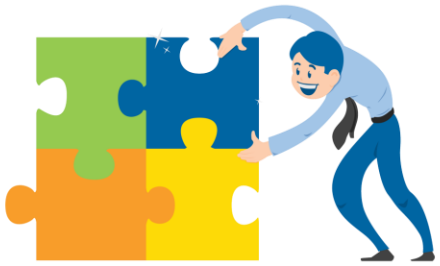
Heap Memory Manger:

- A heap memory manager is a component of a computer's operating system or programming language runtime that is responsible for dynamically allocating and de-allocating memory on the heap.
- When a program requests memory from the heap, the memory manager finds a suitable block of memory and reserves it for the program's use. The memory manager also ensures that different memory allocations do not overlap and cause conflicts by linking all data allocated and save an information about this data into table to can mange it.





HMM Vision



Program Break

Sbrk function.

List

Doubly liked list.

It is a challenge that executes a sequence of operations in according concepts of memory to perform in most of scenarios of allocating.

So we can say our task is :-

1- Understanding how the heap memory manger work in Linux operating system

2- How to design a software can do this task

3- How to avoid memory leaks

2- Design

HMM layers

HMM Abstraction layers consists of:

- HMM
- Doubly list

HMM

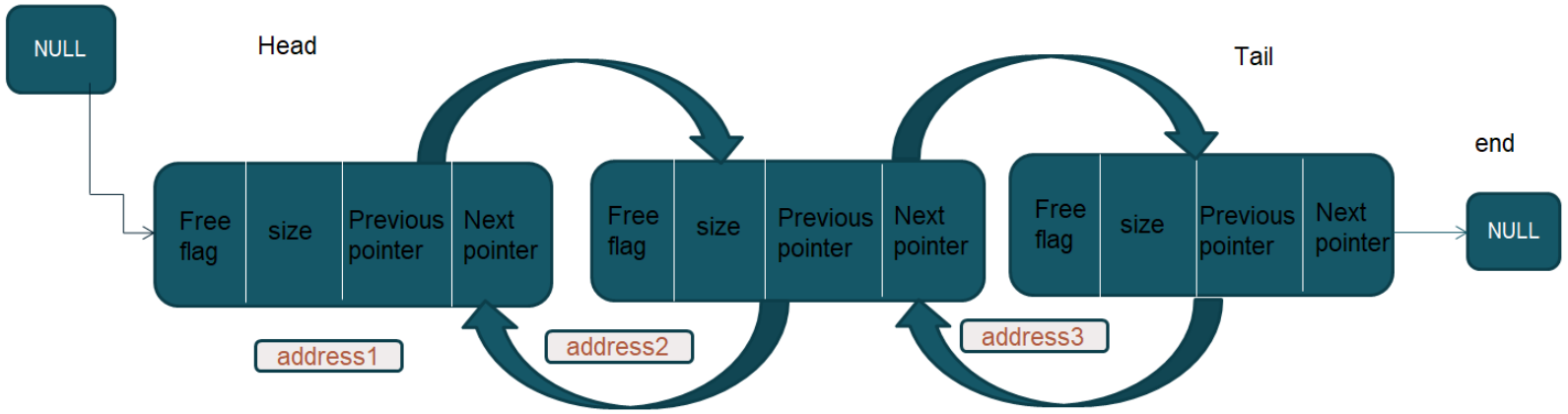
Doubly list

- This list have free and allocated nodes , each node has meta data about data allocated.

- The different between allocated node and free node is the free flag.

Node of list consists of:

- Free flag: show if this node free or allocated
- Size: size of data
- Previous pointer: save address of previous node
- Next pointer: save address of next node



Allocating mechanism:

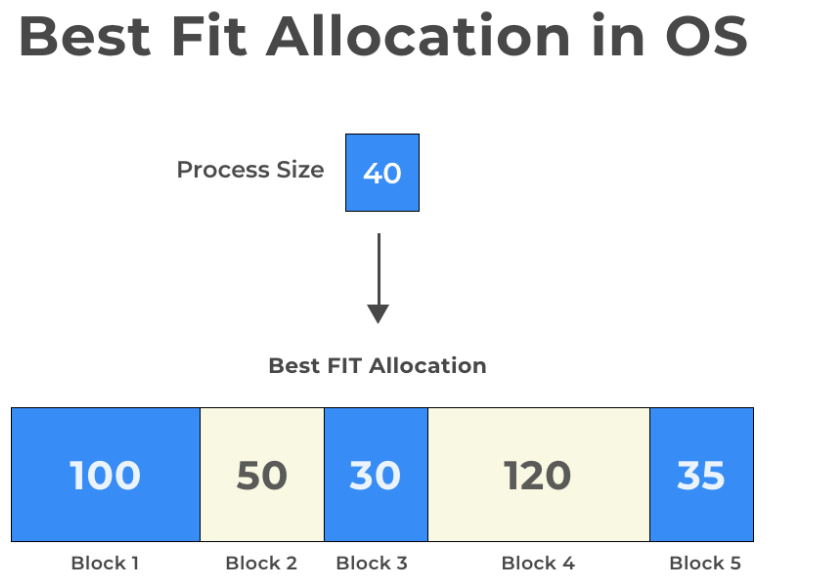
- Alignment size to nearest multiple of bytes (8-bits).
- Move the program break up with a size above the need size to allocate.
- Search on best fit free node of list and split the remainder size of this node.
- If there aren't free node insert new node in the tail of list.

Freeing mechanism:

- Change free flag status to free.
- Check if there are free node after or before free node to merge them to on free node with large size.
- Move the program break down with a size less than the last free node's size.

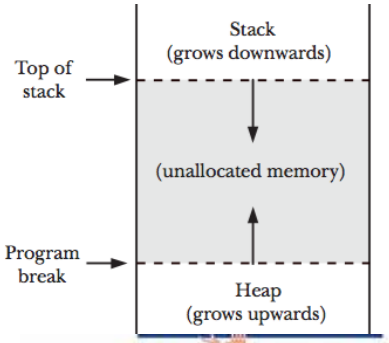
Reallocating mechanism:

- Verification of the address given It's a address of node of the list.
- Check of the relation between new size and old size of this node.
- If new size is bigger than old one so check on the ability to merge with next or previous node.
- Last case allocate new node.



	Size	Can Occupy?	Memory Wastage After Process Occupies	
Block 1	100	Yes	100 - 40 = 60	
Block 2	50	Yes	50 - 40 = 10	Best
Block 3	30	No	-	
Block 4	120	Yes	120 - 40 = 80	
Block 5	35	No	-	

Memory allocating & moving program break



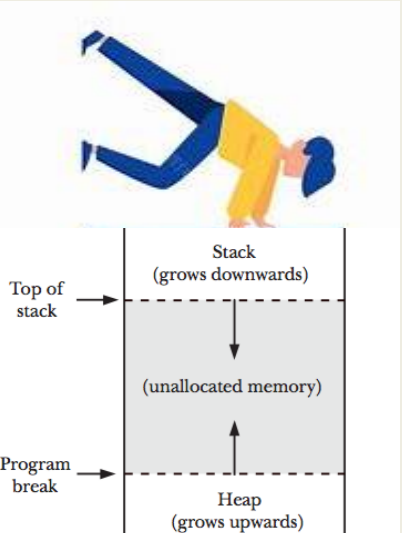
When call Malloc

Move program break up

When call free

Move program break down

OUR CYCLE

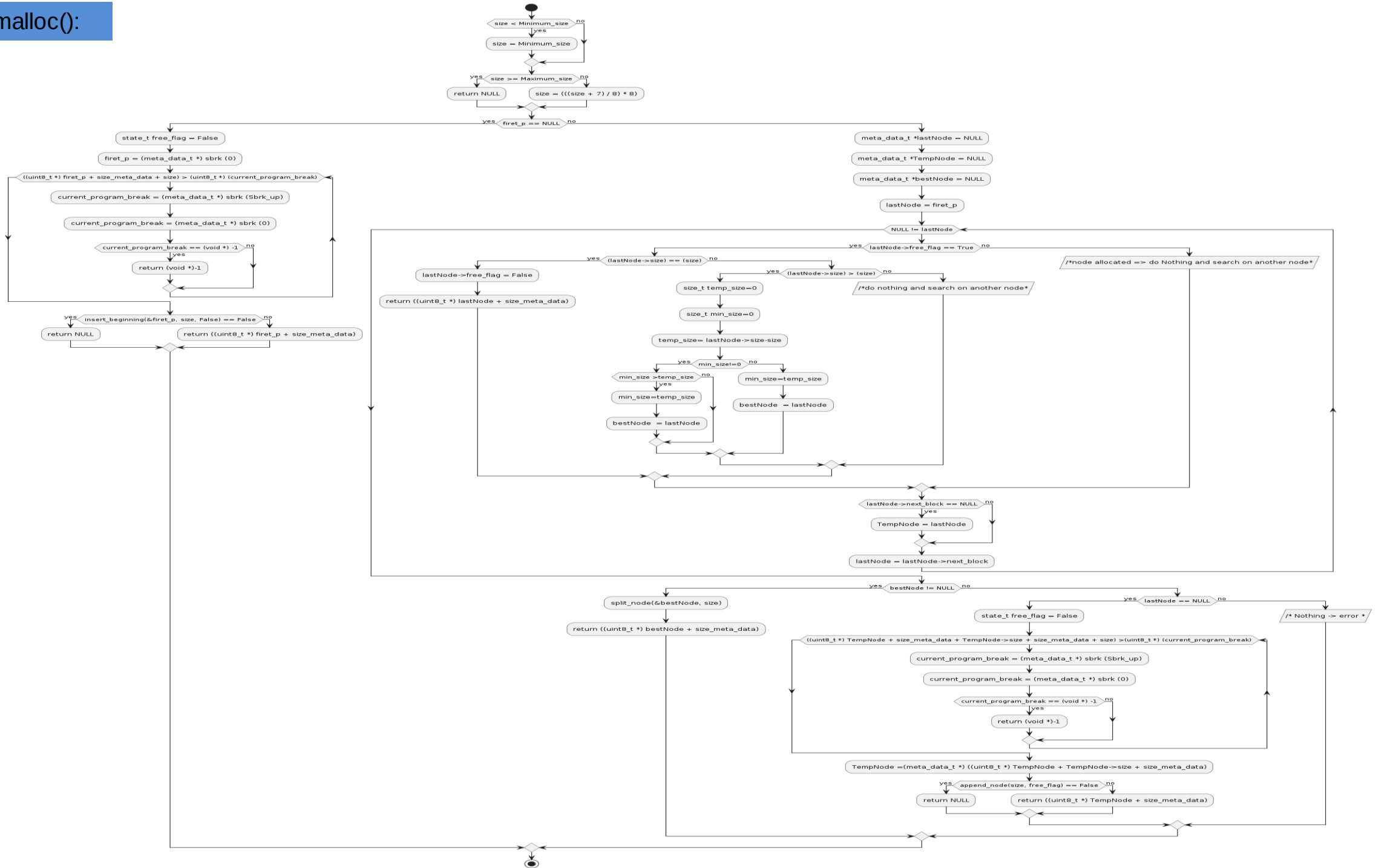


3- Flowchart & Implementation

HMM consists of four function:

- My malloc().
- My free().
- My calloc().
- My realloc().

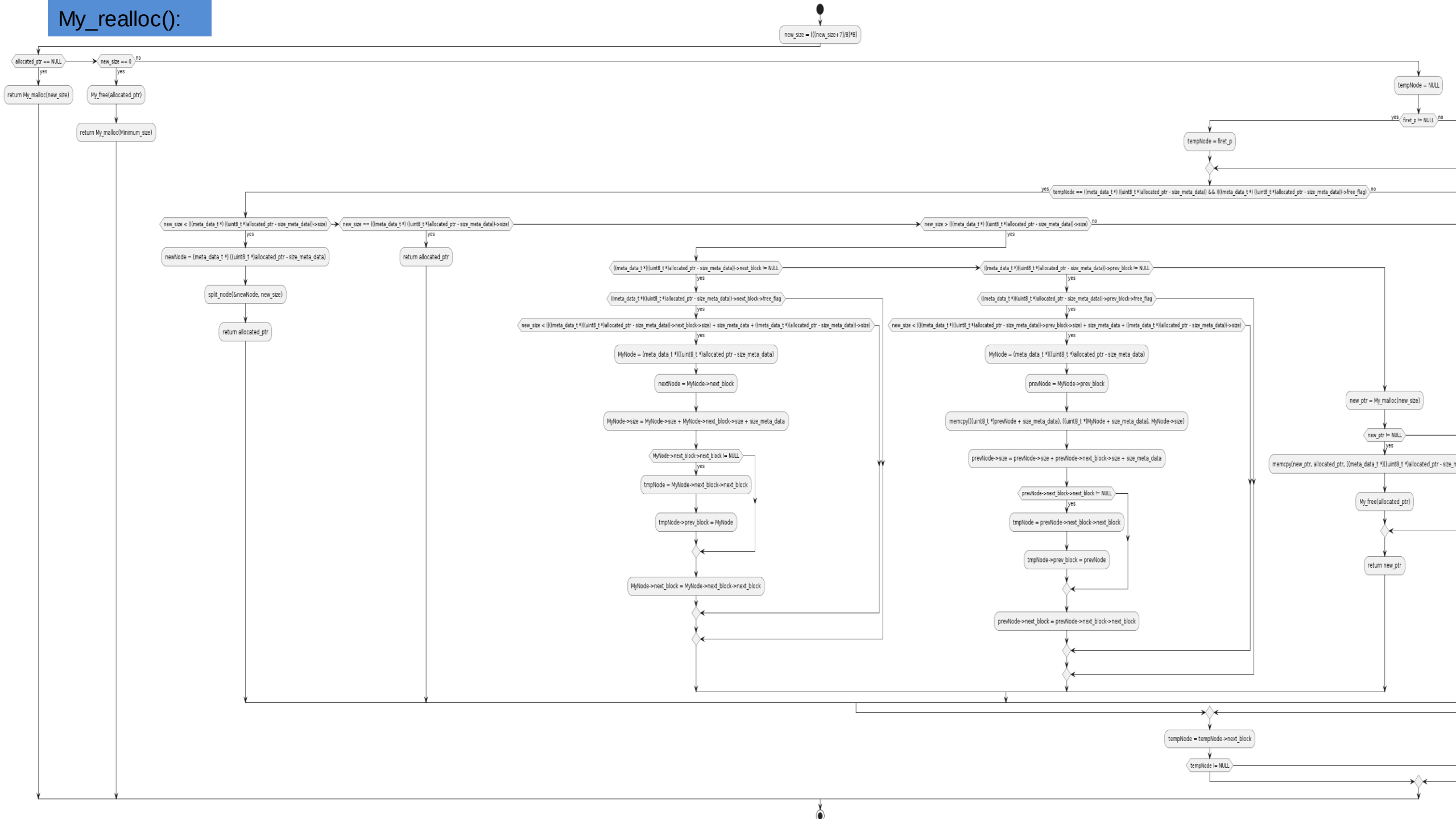
My_malloc():


size < Minimum_size
no
yes
size = Minimum_size
yes
size >= Maximum_size
no
return NULL
size = (((size + 7) / 8) * 8)
yes
firet_p == NULL
no
state_t free_flag = False
firet_p = (meta_data_t *) sbrk (0)
((uint8_t *) firet_p + size_meta_data + size) > (uint8_t *) (current_program_break)
current_program_break = (meta_data_t *) sbrk (Sbrk_up)
current_program_break = (meta_data_t *) sbrk (0)
current_program_break == (void *) -1
no
yes
return (void *)-1
yes
insert_beginning(&firet_p, size, False) == False
no
return NULL
return ((uint8_t *) firet_p + size_meta_data)
meta_data_t *lastNode = NULL
meta_data_t *TempNode = NULL
meta_data_t *bestNode = NULL
lastNode = firet_p
NULL != lastNode
yes
lastNode->free_flag == True
no
/*node allocated => do Nothing and search on another node*
yes
(lastNode->size) == (size)
no
lastNode->free_flag = False
return ((uint8_t *) lastNode + size_meta_data)
yes
(lastNode->size) > (size)
no
size_t temp_size=0
/*do nothing and search on another node*
size_t min_size=0
temp_size= lastNode->size-size
yes
min_size!=0
no
min_size >temp_size
no
yes
min_size=temp_size
bestNode = lastNode
min_size=temp_size
bestNode = lastNode
lastNode->next_block == NULL
no
yes
TempNode = lastNode
lastNode = lastNode->next_block
yes
bestNode != NULL
no
split_node(&bestNode, size)
return ((uint8_t *) bestNode + size_meta_data)
yes
lastNode == NULL
no
/* Nothing -> error *
state_t free_flag = False
((uint8_t *) TempNode + size_meta_data + TempNode->size + size_meta_data + size) >(uint8_t *) (current_program_break)
current_program_break = (meta_data_t *) sbrk (Sbrk_up)
current_program_break = (meta_data_t *) sbrk (0)
current_program_break == (void *) -1
no
yes
return (void *)-1
TempNode =(meta_data_t *) ((uint8_t *) TempNode + TempNode->size + size_meta_data)
yes
append_node(size, free_flag) == False
no
return NULL
return ((uint8_t *) TempNode + size_meta_data)

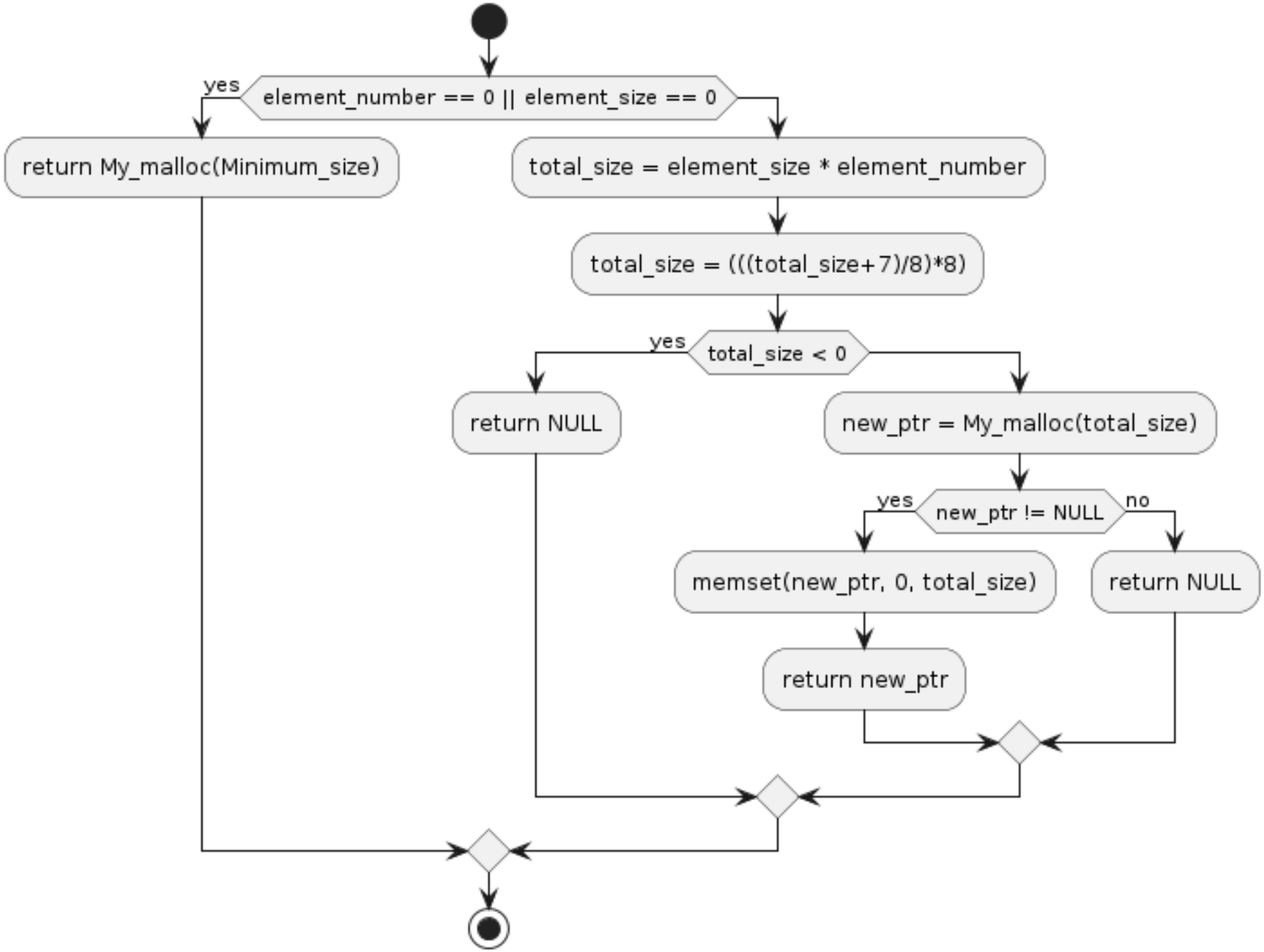
My_free():


NULL != firet_p
yes
no
lastNode = firet_p
/* All nodes are free *
NULL != lastNode
((uint8_t *)free_ptr - size_meta_data) == (uint8_t *)lastNode
yes
no
lastNode->free_flag == False
yes
no
/* Not this node, search on next node *
lastNode->free_flag = True
merge_nodes((meta_data_t *)(free_ptr - size_meta_data))
lastNode = lastNode->next_block
uint32_t N_down = sdbrk_down(firet_p)
N_down > 0
yes
no
int i = 0
/* Do nothing *
i < N_down
current_program_break = (meta_data_t *) sbrk(-Sbrk_down)
current_program_break = (meta_data_t *) sbrk(0)
i = i + 1

My_realloc():

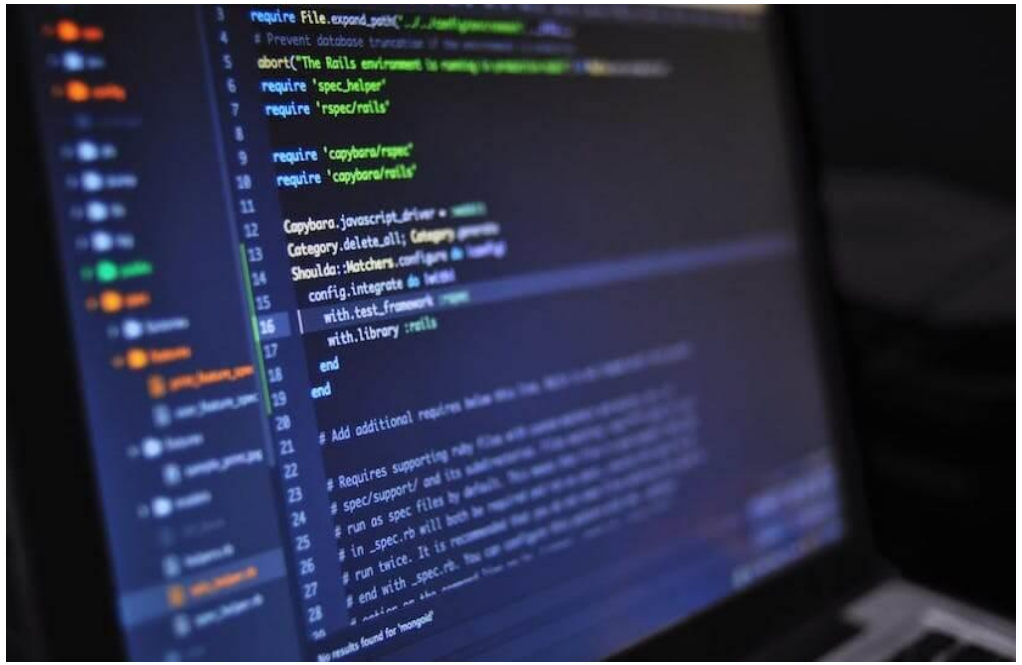

new_size = (((new_size+7)/8)*8)
allocated_ptr == NULL
yes
return My_malloc(new_size)
new_size == 0
yes
My_free(allocated_ptr)
return My_malloc(Minimum_size)
no
tempNode = NULL
firet_p != NULL
yes
no
tempNode = firet_p
tempNode == ((meta_data_t *) ((uint8_t *)allocated_ptr - size_meta_data)) && !((meta_data_t *) ((uint8_t *)allocated_ptr - size_meta_data))->free_flag)
yes
no
new_size < (((meta_data_t *) ((uint8_t *)allocated_ptr - size_meta_data))->size)
yes
newNode = (meta_data_t *) ((uint8_t *)allocated_ptr - size_meta_data)
split_node(&newNode, new_size)
return allocated_ptr
new_size == (((meta_data_t *) ((uint8_t *)allocated_ptr - size_meta_data))->size)
yes
return allocated_ptr
new_size > (((meta_data_t *) ((uint8_t *)allocated_ptr - size_meta_data))->size)
no
yes
((meta_data_t *)((uint8_t *)allocated_ptr - size_meta_data))->next_block != NULL
yes
((meta_data_t *)((uint8_t *)allocated_ptr - size_meta_data))->next_block->free_flag
yes
new_size < ((((meta_data_t *)((uint8_t *)allocated_ptr - size_meta_data))->next_block->size) + size_meta_data + ((meta_data_t *)(allocated_ptr - size_meta_data))->size)
yes
MyNode = (meta_data_t *)((uint8_t *)allocated_ptr - size_meta_data)
nextNode = MyNode->next_block
MyNode->size = MyNode->size + MyNode->next_block->size + size_meta_data
MyNode->next_block->next_block != NULL
yes
tmpNode = MyNode->next_block->next_block
tmpNode->prev_block = MyNode
MyNode->next_block = MyNode->next_block->next_block
((meta_data_t *)((uint8_t *)allocated_ptr - size_meta_data))->prev_block != NULL
yes
((meta_data_t *)((uint8_t *)allocated_ptr - size_meta_data))->prev_block->free_flag
yes
new_size < ((((meta_data_t *)((uint8_t *)allocated_ptr - size_meta_data))->prev_block->size) + size_meta_data + ((meta_data_t *)(allocated_ptr - size_meta_data))->size)
yes
MyNode = (meta_data_t *)((uint8_t *)allocated_ptr - size_meta_data)
prevNode = MyNode->prev_block
memcpy(((uint8_t *)prevNode + size_meta_data), ((uint8_t *)MyNode + size_meta_data), MyNode->size)
prevNode->size = prevNode->size + prevNode->next_block->size + size_meta_data
prevNode->next_block->next_block != NULL
yes
tmpNode = prevNode->next_block->next_block
tmpNode->prev_block = prevNode
prevNode->next_block = prevNode->next_block->next_block
new_ptr = My_malloc(new_size)
new_ptr != NULL
yes
memcpy(new_ptr, allocated_ptr, ((meta_data_t *)((uint8_t *)allocated_ptr - size_m
My_free(allocated_ptr)
return new_ptr
tempNode = tempNode->next_block
tempNode != NULL

My_calloc():

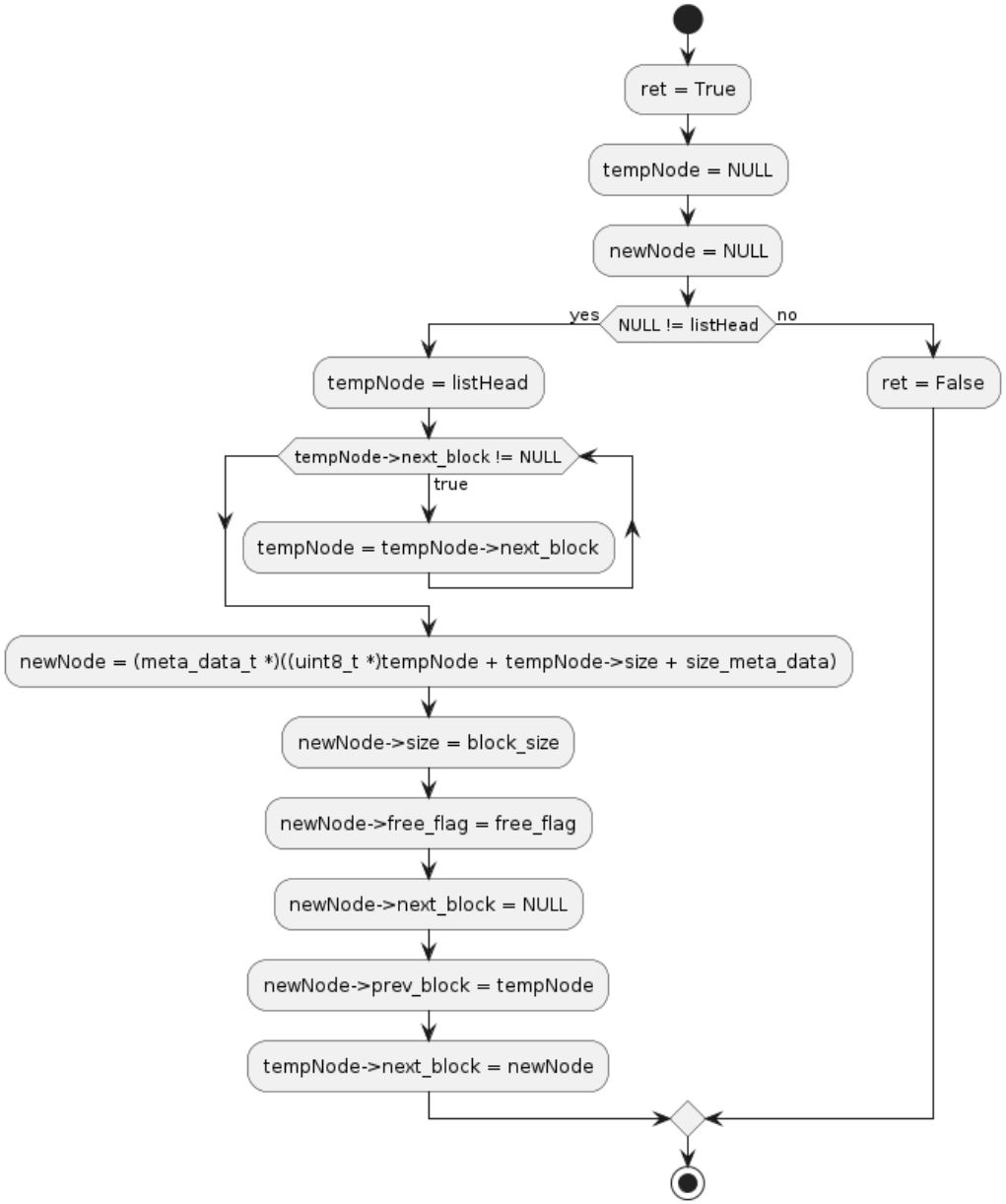

element_number == 0 || element_size == 0
yes
return My_malloc(Minimum_size)
total_size = element_size * element_number
total_size = (((total_size+7)/8)*8)
total_size < 0
yes
return NULL
new_ptr = My_malloc(total_size)
new_ptr != NULL
yes
no
memset(new_ptr, 0, total_size)
return NULL
return new_ptr

Doubly liked list consists of four function:

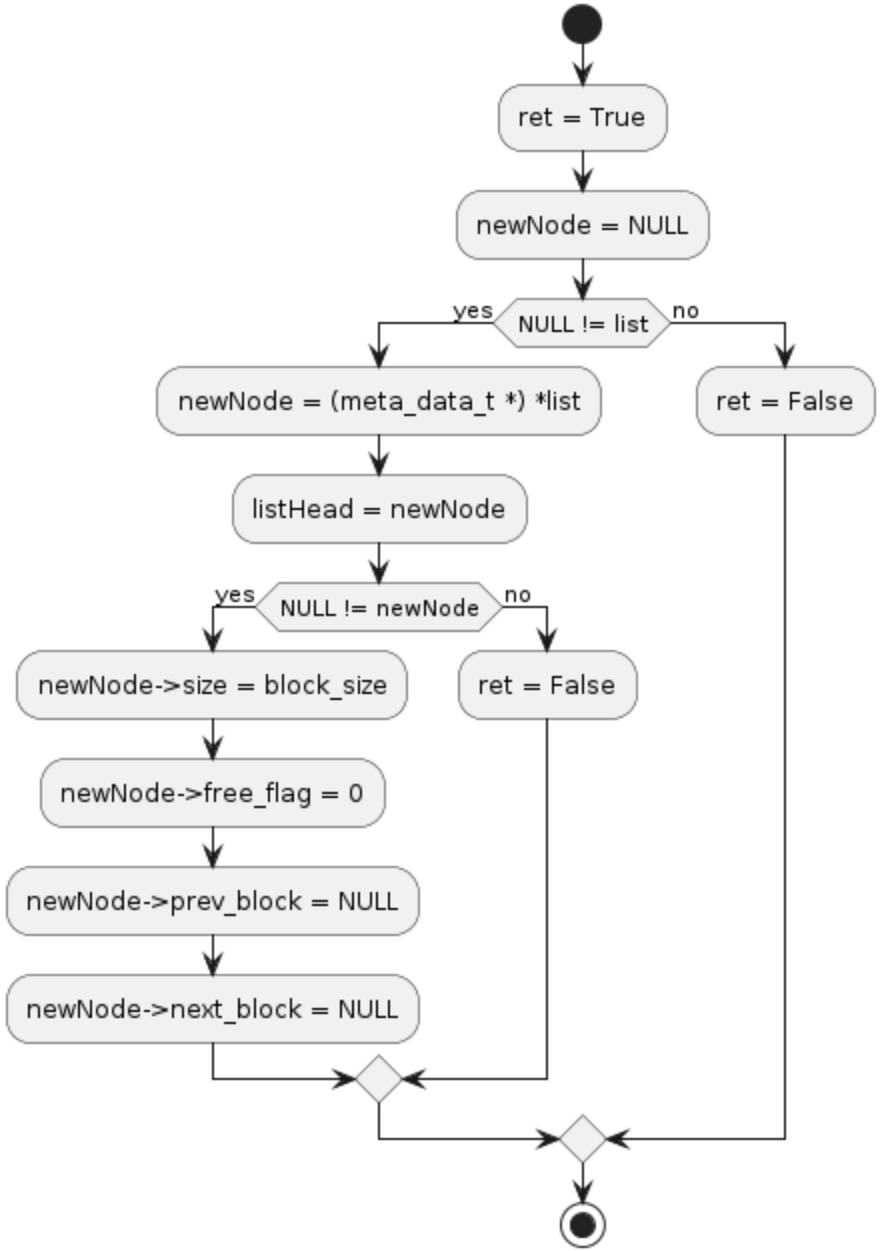
- Insert node().
- Split node().
- Merge nodes().
- Sbrk down().



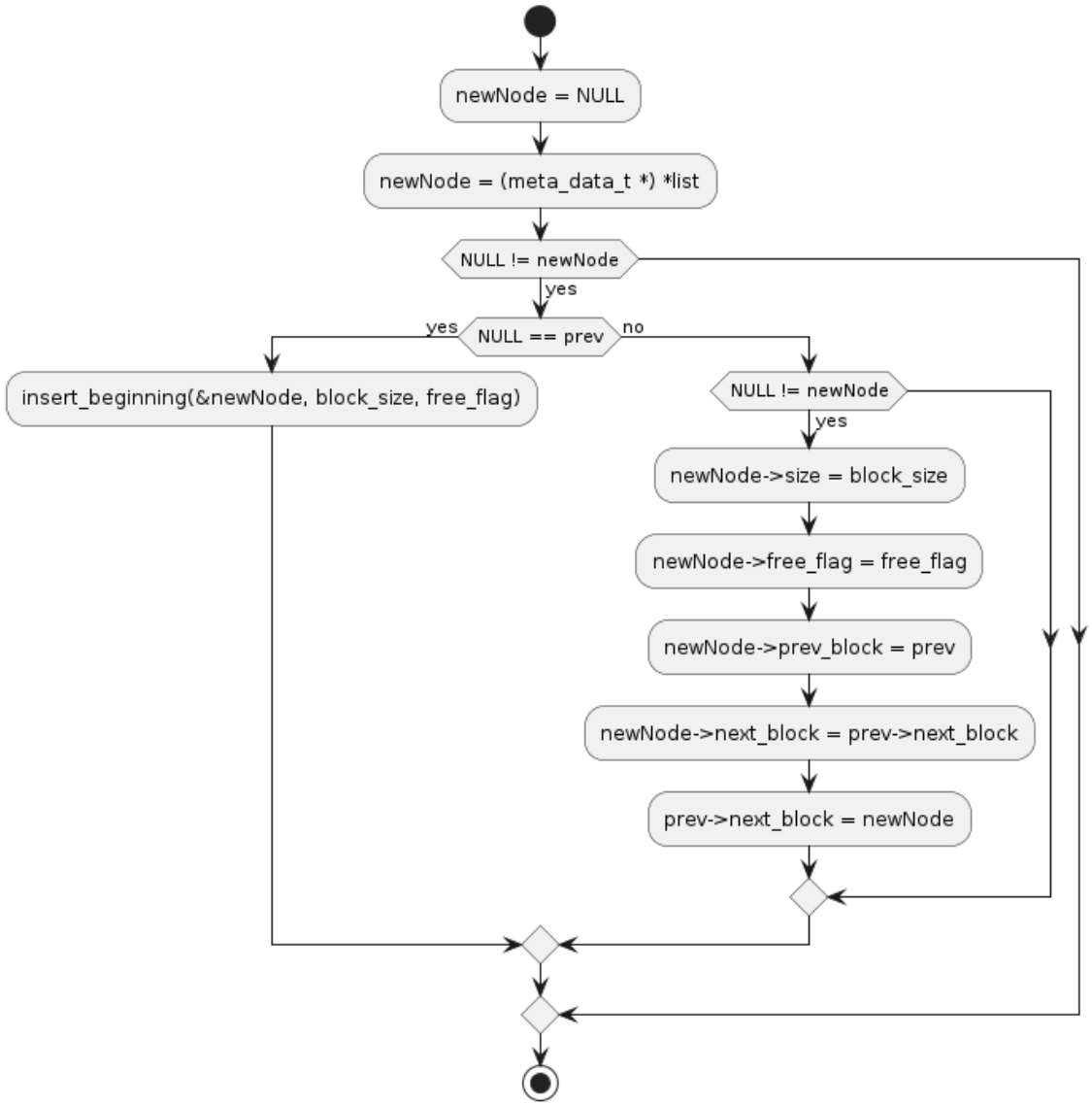
append node():



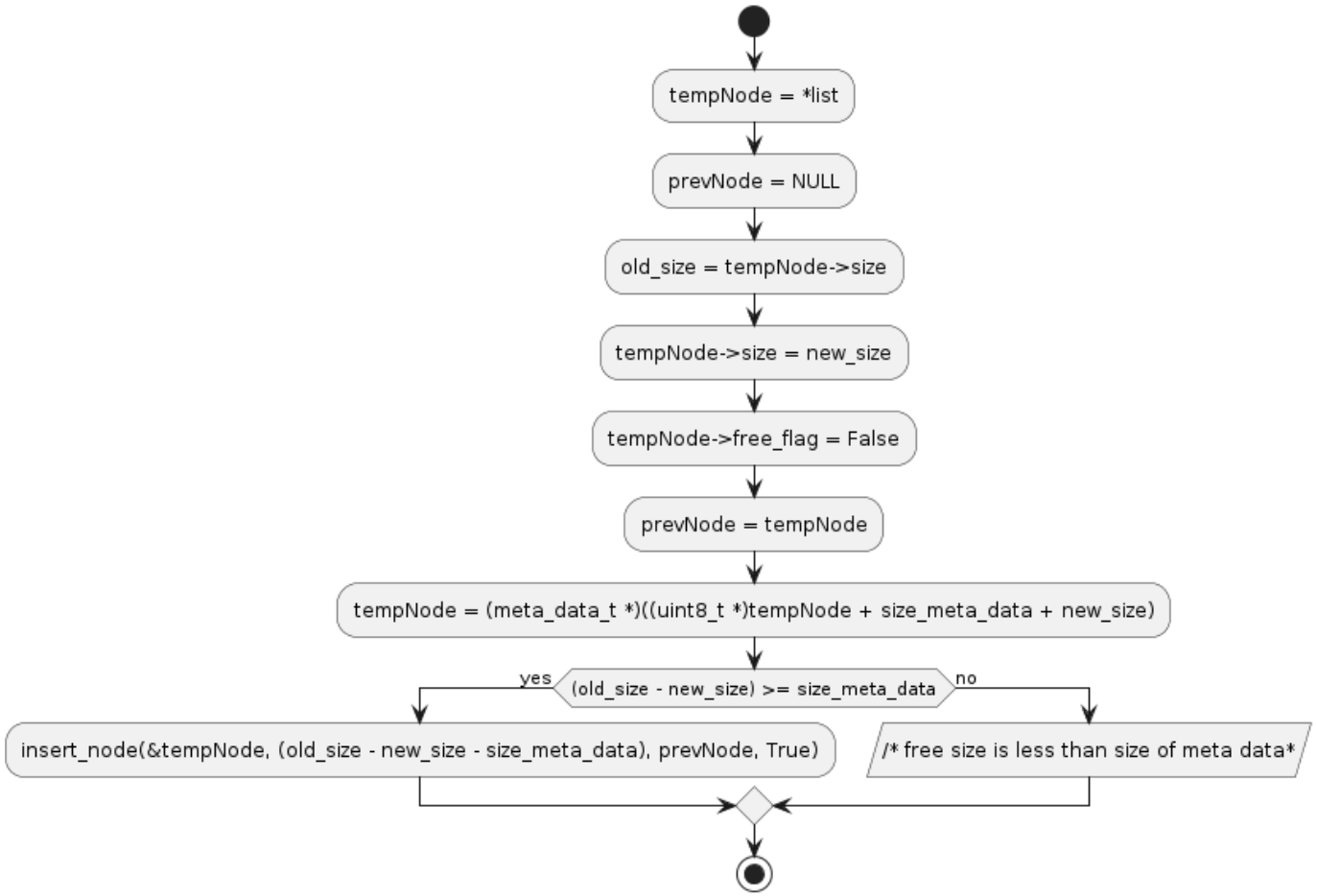
Insert beginning ():



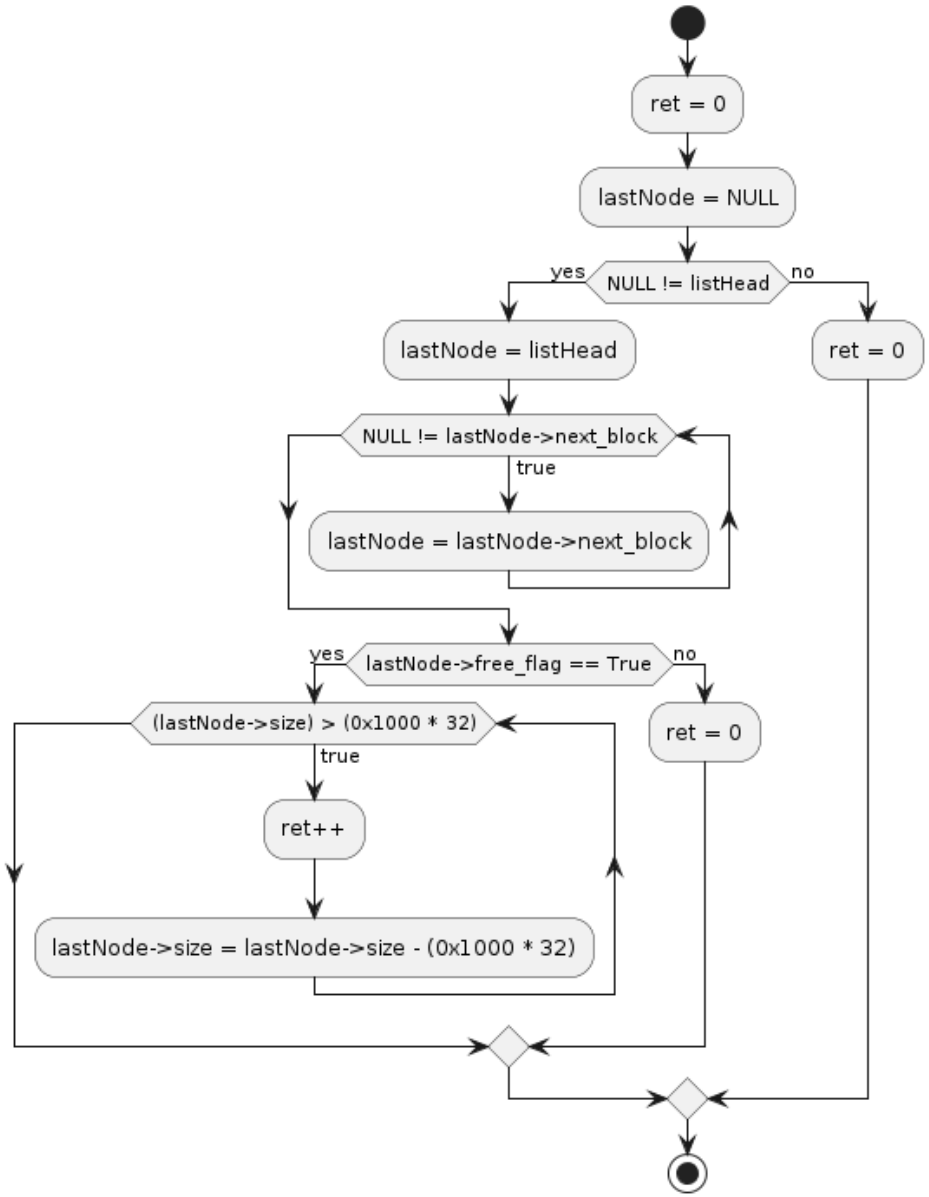
Insert node():



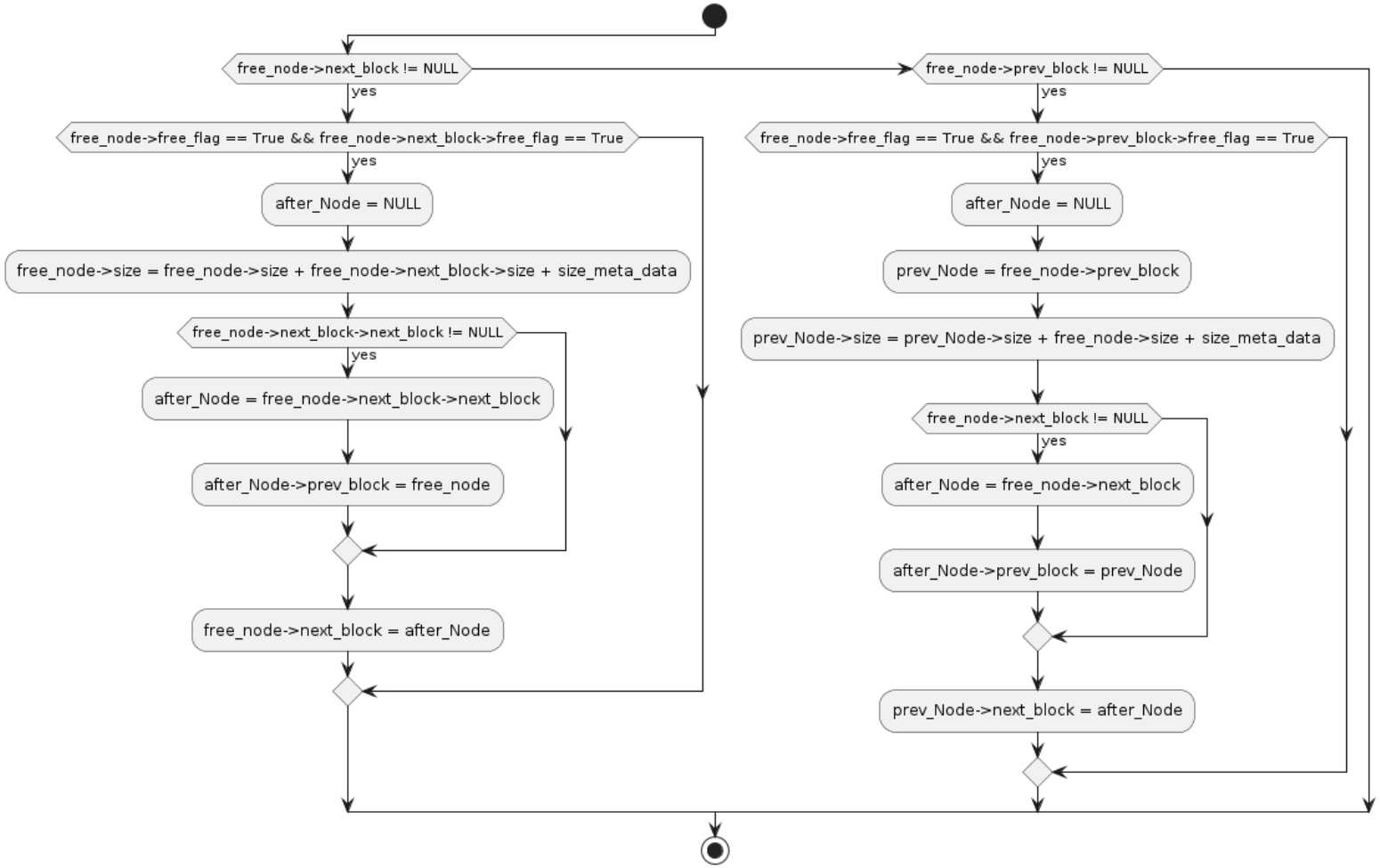
Split node():



Sbrk down():



Merge nodes():



For source code:

4- Testing

First:

- Simple test to verifying allocating and freeing memory
- Print current program break to verify that it moves correctly.

HMM.c | Doubly_Free_list.c | main.c | Doubly_free_list.h | HMM.h | Makefile

main.c > MAX_SIZE

```c
extern char end, edata, etext;

int
main(int argc, char *argv[])
{
    char *ptr[MAX_ALLOCS];
    int freeStep, freeMin, freeMax, blockSize, numAllocs, j;
    printf("etext = %p, edata=%p, end=%p, initial program break=%p\n", &etext, &edata, &end, (char*)sbrk(0));

    if (argc < 3 || strcmp(argv[1], "--help") == 0) {
        printf("%s num-allocs block-size [step [min [max]]]\n", argv[0]);
        exit(1);
    }

    numAllocs = getInt(argv[1], GN_GT_0, "num-allocs");
    if (numAllocs > MAX_ALLOCS) {
        printf("num-allocs > %d\n", MAX_ALLOCS);
        exit(1);
    }

    blockSize = getInt(argv[2], GN_GT_0 | GN_ANY_BASE, "block-size");

    freeStep = (argc > 3) ? getInt(argv[3], GN_GT_0, "step") : 1;
    freeMin =  (argc > 4) ? getInt(argv[4], GN_GT_0, "min") : 1;
    freeMax =  (argc > 5) ? getInt(argv[5], GN_GT_0, "max") : numAllocs;

    if (freeMax > numAllocs) {
        printf("free-max > num-allocs\n");
        exit(1);
    }

    printf("Initial program break:          %10p\n", sbrk(0));

    printf("Allocating %d*%d bytes\n", numAllocs, blockSize);
    for (j = 0; j < numAllocs; j++) {
        ptr[j] = malloc(blockSize);
        if (ptr[j] == NULL) {
            printf("malloc returned null\n");
            exit(1);
        }
    }

    printf("Program break is now:           %10p\n", sbrk(0));

    printf("Freeing blocks from %d to %d in steps of %d\n",
                freeMin, freeMax, freeStep);
    for (j = freeMin - 1; j < freeMax; j += freeStep)
        free(ptr[j]);

    printf("After free(), program break is: %10p\n", sbrk(0));
    while(1);

    exit(EXIT_SUCCESS);
}
```

PROBLEMS | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS

```
omar@omar-VirtualBox:~/Advanced/testing_code$ ./main.exe 1000 10240 1 500 1000
etext = 0x5ade063091d1, edata=0x5ade0630c010, end=0x5ade0630c038, initial program break=0x5ade0825e000
Initial program break:          0x5ade0827f000
Allocating 1000*10240 bytes
Program break is now:           0x5ade08c2a000
Freeing blocks from 500 to 1000 in steps of 1
After free(), program break is: 0x5ade0874a000
```

Second:

- Strong test to verifying all project (tow function: malloc, free).
- This test generate randomized variables of values to of size to allocate and free.
- This script do this operation for 1000 000 times of allocating and max size 10000 bytes (can be changed), this test success.

```
53    }
54
55    void *calloc(size_t nmemb, size_t size)
56    {
57      return My_calloc(nmemb , size);
58    }
59    void *realloc(void *ptr, size_t size)
60    {
61      return My_relloc(ptr , size);
62    }
63    #endif
64    /***************************** Macros section ************************************
65    #define NUM_ALLOCS 1000000
66    #define MAX_SIZE 102400
67    #define MAX_ITERATIONS 1000000 /*milion random variables**/
68
69    /*
70     * Performs a random allocation and deallocation test using the custom memory management f
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Freeing remaining memory at address 0x55555ca1ef30
Freeing remaining memory at address 0x5555562b3b48
Freeing remaining memory at address 0x55555e6adea8
Freeing remaining memory at address 0x55555820ec28
Freeing remaining memory at address 0x5555629b1730
Freeing remaining memory at address 0x55556053ba10
Freeing remaining memory at address 0x55555a5ec0a0
Freeing remaining memory at address 0x55555aff8b40
Freeing remaining memory at address 0x55555772dee8
Freeing remaining memory at address 0x55555cbf3408
Freeing remaining memory at address 0x55555d5bf920
Freeing remaining memory at address 0x555562a64e08
Freeing remaining memory at address 0x55555dcaafc8
Freeing remaining memory at address 0x555561223938
Freeing remaining memory at address 0x55555fbd9a38
Freeing remaining memory at address 0x55555a881440
Freeing remaining memory at address 0x555558063478
Freeing remaining memory at address 0x555561581738
Freeing remaining memory at address 0x5555556a0f500
Freeing remaining memory at address 0x55555e44ea98
Freeing remaining memory at address 0x55555fcb1050
Freeing remaining memory at address 0x55555a68baa0
Freeing remaining memory at address 0x5555610d6960
Freeing remaining memory at address 0x55555ae7b6d8
Freeing remaining memory at address 0x55555b7e9f28
Freeing remaining memory at address 0x5555571de1b0
Freeing remaining memory at address 0x555555bd59f8
Freeing remaining memory at address 0x5555579e0c08
Freeing remaining memory at address 0x555556d17d88
Freeing remaining memory at address 0x55555dbb4ca0
Freeing remaining memory at address 0x55555e534b60
Freeing remaining memory at address 0x555556b5a680
Freeing remaining memory at address 0x555559c468c0
Freeing remaining memory at address 0x555560c98dc8
Freeing remaining memory at address 0x555562cd61a8
Freeing remaining memory at address 0x555559daaaf8
Freeing remaining memory at address 0x55555be1c250
Freeing remaining memory at address 0x555556ae9900
Freeing remaining memory at address 0x55555753fd38
Freeing remaining memory at address 0x5555628a5b28
Freeing remaining memory at address 0x5555561ce350
Freeing remaining memory at address 0x555560d37008
Freeing remaining memory at address 0x55555a4900d0
Freeing remaining memory at address 0x555555b14230
Freeing remaining memory at address 0x55555f374b38
Freeing remaining memory at address 0x5555615c12f0
Freeing remaining memory at address 0x5555579a5ba8
Freeing remaining memory at address 0x55555e05d3e8
Test complete.
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Micros
```

Third:

- Test strong to verifying all project (four function: malloc, free, calloc, realloc).
- This test generate randomized variables of values to of size to allocate, reallocate and free.
- This script do this operation for 100 000 times of allocating and max size 10000 bytes (can be changed), this test success.

```c
/*****************************************************

/****************************** Macros section ******
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

#define NUM_OPERATIONS 100000

#define MAX_SIZE (10000)

enum Operation
{
    MALLOC,
    CALLOC,
    REALLOC,
    FREE
};

void *allocated_blocks[NUM_OPERATIONS];

void perform_random_operations()
{

    srand(time(NULL));

    for (int i = 0; i < NUM_OPERATIONS; i++)
    {
        enum Operation op = rand() % 4;
```

```
Allocated block 0x654507c1ae80 with size 4566
Allocated block 0x654507c1c060 with size 9171
Freed block (nil)
Allocated block 0x6544fc5bf7d0 with size 1813
Freed block (nil)
Reallocated block 0x6544fb0b9330 with size 9110
Allocated block 0x654500801c20 with 4034 elements of size 216
Reallocated block 0x654501bc8e60 with size 2384
Freed block (nil)
Allocated block 0x6544f58b76d0 with size 1657
Allocated block 0x654507c1e440 with 1752 elements of size 2181
Freed block (nil)
Allocated block 0x654507c20680 with size 4303
Freed block (nil)
Allocated block 0x654507c21760 with size 9869
Freed block (nil)
Reallocated block 0x65450615fd80 with size 2084
Reallocated block 0x654507c23e00 with size 7754
Allocated block 0x654502297e80 with size 3567
Freed block (nil)
Allocated block 0x654507c25c60 with 614 elements of size 3082
Allocated block 0x654501a0f1a0 with size 862
Allocated block 0x6544f9659430 with size 5625
Allocated block 0x654507c282d0 with 5258 elements of size 7376
Allocated block 0x654507c29770 with 5561 elements of size 713
Allocated block 0x654507c2ad40 with 6216 elements of size 9097
Reallocated block 0x654507c2c590 with size 8965
Allocated block 0x654507c2e8a0 with 1707 elements of size 2895
Allocated block 0x6544f90962d0 with size 142
Allocated block 0x6545013dc180 with size 1066
Freed block (nil)
Freed block (nil)
Allocated block 0x654507c30a00 with 1117 elements of size 9560
Reallocated block 0x6544fd996d00 with size 6773
Freed block (nil)
Allocated block 0x654507c32cf0 with 305 elements of size 2985
Reallocated block 0x6544fbb24090 with size 49
Allocated block 0x654507c35320 with size 8433
Allocated block 0x654507c37420 with size 8167
Freed block (nil)
Allocated block 0x6544fd0bddf0 with size 4278
Freed block (nil)
Reallocated block 0x6544f965aa40 with size 3879
Allocated block 0x654507c39410 with 7029 elements of size 4862
DONE
```

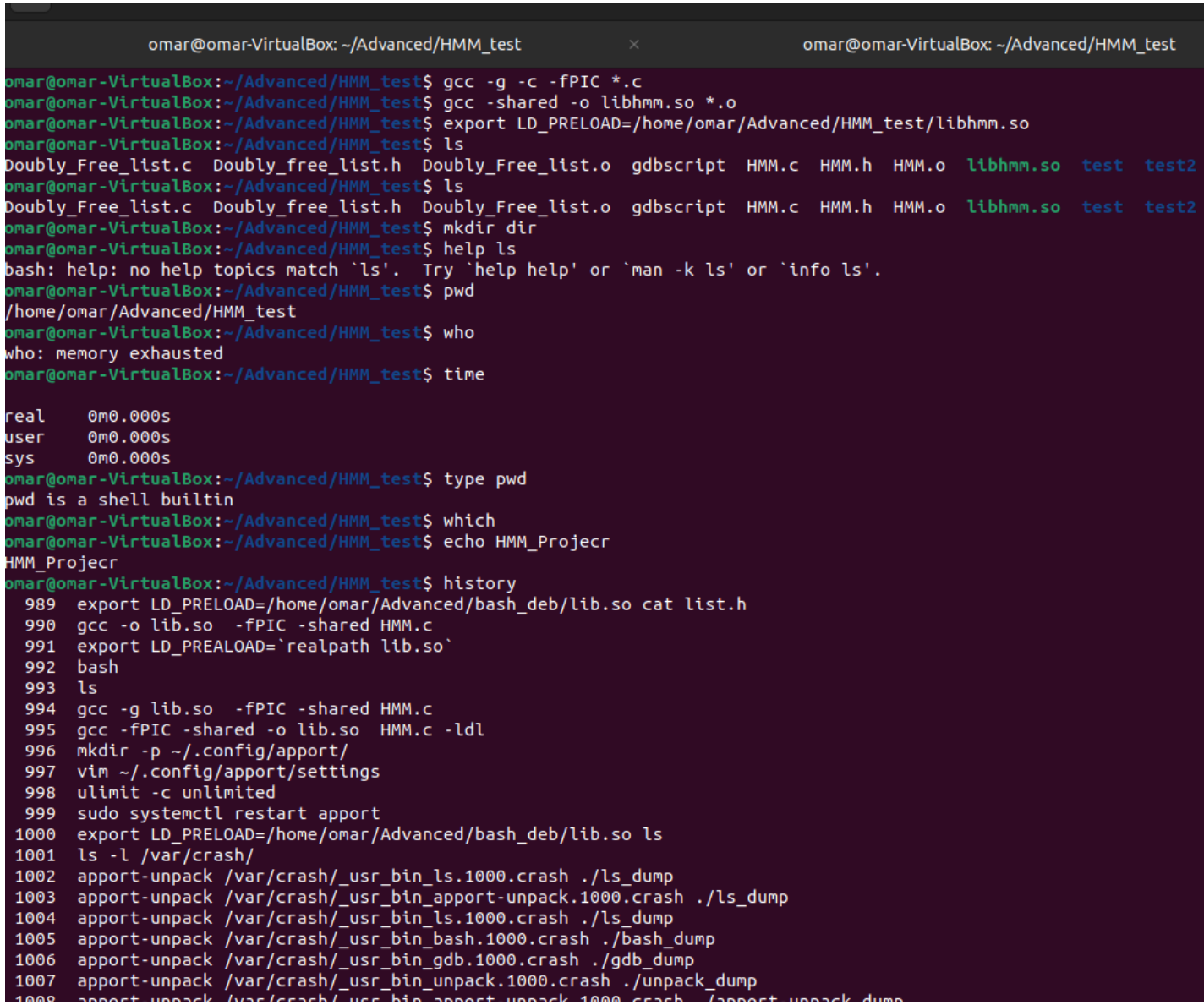
Fourth:

- This test to verifying allocating and freeing memory not causes leak of memory by using Valgired program
- Print if there are memory leak.

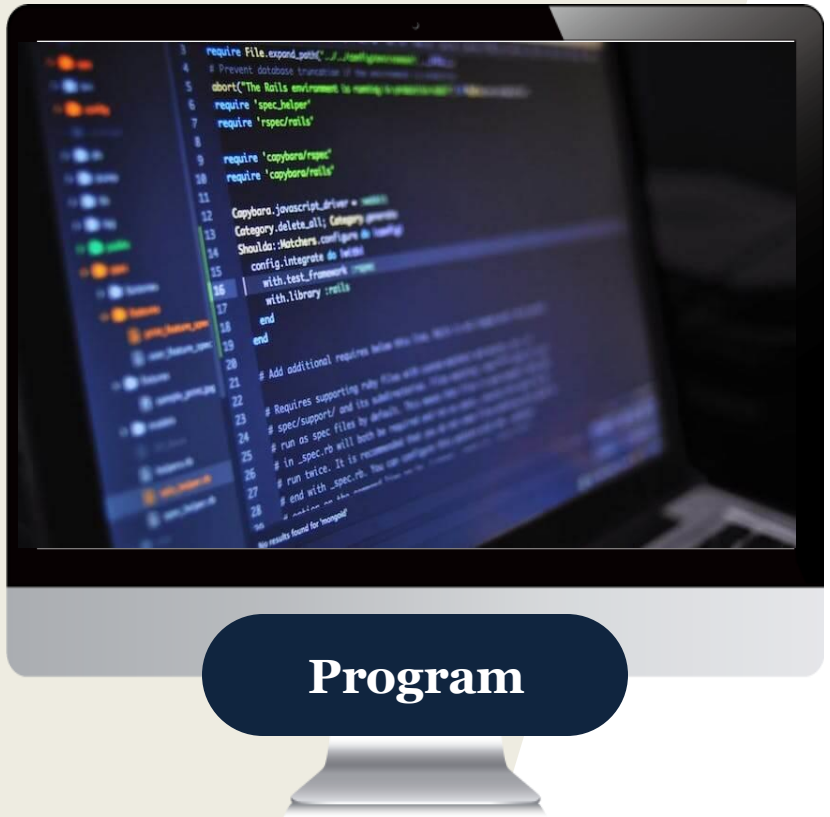
```
Allocated block 0x6502ed0 with size 879
Allocated block 0x15998230 with 6339 elements of size 5504
Allocated block 0x184989c0 with size 9938
Allocated block 0x76fdde0 with size 4966
Freed block (nil)
Allocated block 0x1849b0e0 with 9419 elements of size 5661
Allocated block 0x1849d5f0 with 7949 elements of size 7539
Freed block (nil)
Allocated block 0x10c92500 with size 5701
Reallocated block 0x5bae0b0 with size 1285
Reallocated block 0x151e6490 with size 7134
Allocated block 0x968f650 with size 4718
Allocated block 0xc387fe0 with size 2326
Allocated block 0x1849f540 with 3207 elements of size 5791
Allocated block 0xb2ac5f0 with size 5063
Freed block (nil)
Reallocated block 0x184a1b20 with size 6245
Allocated block 0x184a33d0 with size 9234
Reallocated block 0x54e1f80 with size 7247
Reallocated block 0x184a5830 with size 6094
Reallocated block 0x1461a500 with size 8147
Allocated block 0x184a7040 with size 9591
Reallocated block 0x184a9600 with size 8706
Freed block (nil)
Freed block (nil)
Freed block (nil)
Reallocated block 0xd3e1640 with size 5069
Allocated block 0x721bc10 with size 3246
Reallocated block 0x57676c0 with size 2402
Reallocated block 0x184ab850 with size 7988
Allocated block 0x184ad7d0 with 6056 elements of size 1586
Freed block (nil)
Freed block (nil)
Allocated block 0x184aefc0 with 2178 elements of size 8007
Freed block (nil)
Allocated block 0x184b1210 with 7274 elements of size 7228
Freed block (nil)
Reallocated block 0x71c2720 with size 829
Allocated block 0x15711f10 with size 3154
Allocated block 0x184b2ec0 with 8897 elements of size 6399
Freed block (nil)
Freed block (nil)
Allocated block 0x184b51d0 with 3139 elements of size 99
Allocated block 0x13189e30 with size 1077
Reallocated block 0xd236aa0 with size 5108
DONE
==23586==
==23586== HEAP SUMMARY:
==23586==     in use at exit: 302,354,194 bytes in 49,903 blocks
==23586==   total heap usage: 74,913 allocs, 25,010 frees, 454,532,370 bytes allocated
==23586==
==23586== LEAK SUMMARY:
==23586==    definitely lost: 0 bytes in 0 blocks
==23586==    indirectly lost: 0 bytes in 0 blocks
==23586==      possibly lost: 0 bytes in 0 blocks
==23586==    still reachable: 302,354,194 bytes in 49,903 blocks
==23586==         suppressed: 0 bytes in 0 blocks
==23586== Rerun with --leak-check=full to see details of leaked memory
==23586==
==23586== For lists of detected and suppressed errors, rerun with: -s
==23586== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
omar@omar-VirtualBox:~/Advanced/HMM_project$
```

Fifth:

- This test is replacing my library with glibc and run some commands based on my library such as: ls, echo.



5- Make file



build

Compile this library normally.

Static library

Make my library compiled as .static library using this make file by run: make static .

Shared library

Make my library compiled as .shared library using this make file by run: make dynamic.

Makefile

```
CC := gcc
CFLAGS:= -g -Wall
DYN:= -fPIC
SH:= -shared
OPSH:= -ldl
ST:= ar
OPS:= -rs

SRC:= $(shell ls *.c)
OBJ:= $(SRC:.c=.o)
INC:= $(shell ls *.h)
TARGET:= a

.PHONY: all clean build run

all: clean build run
dynamic: clean DYNAMIC run
static: clean STATIC run

STATIC:
    @$(CC) -c  $(SRC)
    @$(ST) $(OPS) libhmm.so *.o
    @$(CC) -c  main.c  -I./mylib
    @$(CC) -o $(TARGET).exe main.o -L./mylib
    @echo "[Makefile][build] : Compiled successfully."

DYNAMIC:
    @$(CC) -c $(DYN) $(SRC)
    @$(CC) $(SH) -o lib.so *.o $(OPSH)
    @$(CC) -o $(TARGET).exe main.c
    @LD_LIBRARY_PATH=`realpath lib.so`
    @echo "[Makefile][build] : Compiled successfully."

build: $(OBJ)
    @$(CC) $(CFLAGS) $(OBJ) -o $(TARGET).exe
    @echo "[Makefile][build] : Compiled successfully."

$(OBJ): %.o: %.c $(INC)
    @$(CC) $(CFLAGS) -c $< -o $@

run:
    ./$(TARGET).exe
clean:
    @rm -f *.o
    @rm -f *.exe
    @echo "[Makefile][clean] : Cleaned successfully."
```

Reference: The Linux Programming Interface
By MICHEL KERRISK

THANK YOU

to contact me

Click here