Lab #3: Memory Lab

Prof. Jae W. Lee (jaewlee@snu.ac.kr)
Department of Computer Science and Engineering
Seoul National University

TA (snu-arc-sysprog-ta@googlegroups.com)

Contents

- Important Dates
- Goal of This Lab
- Environment Setup
- Overview
- The *memmgr* Specification
- Code and Test
- Grading Policy
- Submission

Before getting started

- **Do not directly fork the repository!**
  - Your code will be visible to others
  - Refer to README on how to clone the repository
  - No pull requests either!

Important Dates

- 9 Apr. - Lab Hand-out Session (Today!)
- 16 Apr. - Live Q&A Session (1)
- 23 Apr. - Live Q&A Session (2)
- 29 Apr. 23:59 - Submission Deadline

- Questions about the lab will be conducted through github issue.
  Feel free to post questions (except your code!)
- All Sessions are totally optional.

Before the Presentation::

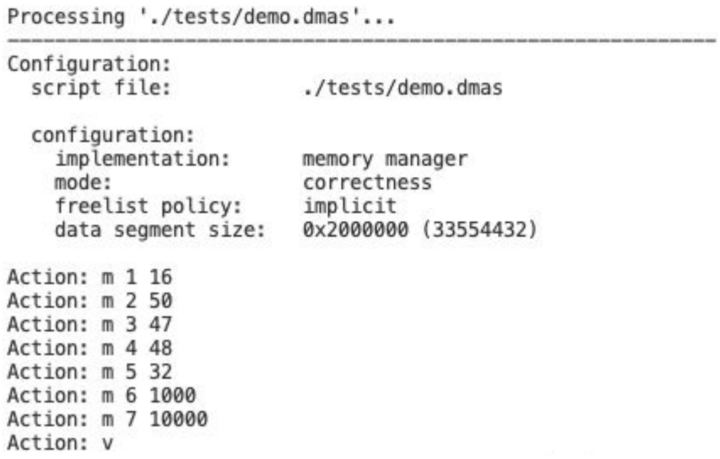
- **All contents on this slide are sourced from the README file.**

- **For detailed information, please refer to README.**
    - https://github.com/SNU-ARC/2024_spring_sysprog_Lab3/blob/main/README.md

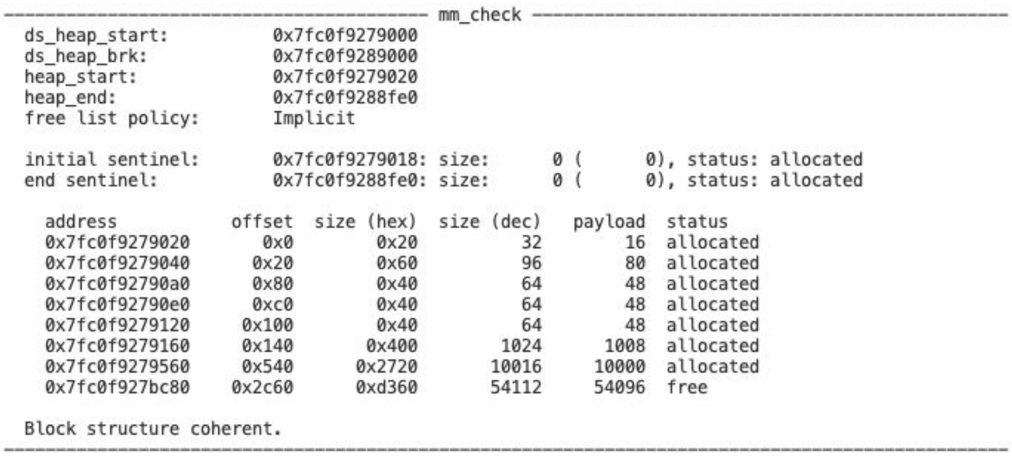
Goal of this Lab (1/2)

- **Implement a dynamic memory manager.**

$ mm_driver ./test/demo.dmas



```
Processing './tests/demo.dmas'...
------------------------------------------------------------
Configuration:
  script file:            ./tests/demo.dmas

  configuration:
    implementation:       memory manager
    mode:                 correctness
    freelist policy:      implicit
    data segment size:    0x2000000 (33554432)

Action: m 1 16
Action: m 2 50
Action: m 3 47
Action: m 4 48
Action: m 5 32
Action: m 6 1000
Action: m 7 10000
Action: v
```



```
------------------------------------- mm_check -------------------------------------
ds_heap_start:          0x7fc0f9279000
ds_heap_brk:            0x7fc0f9289000
heap_start:             0x7fc0f9279020
heap_end:               0x7fc0f9288fe0
free list policy:       Implicit

initial sentinel:       0x7fc0f9279018: size:      0 (      0), status: allocated
end sentinel:           0x7fc0f9288fe0: size:      0 (      0), status: allocated

  address           offset  size (hex)  size (dec)   payload  status
  0x7fc0f9279020       0x0        0x20          32        16  allocated
  0x7fc0f9279040      0x20        0x60          96        80  allocated
  0x7fc0f92790a0      0x80        0x40          64        48  allocated
  0x7fc0f92790e0      0xc0        0x40          64        48  allocated
  0x7fc0f9279120     0x100        0x40          64        48  allocated
  0x7fc0f9279160     0x140       0x400        1024      1008  allocated
  0x7fc0f9279560     0x540      0x2720       10016     10000  allocated
  0x7fc0f927bc80    0x2c60      0xd360       54112     54096  free

Block structure coherent.
------------------------------------------------------------------------------------
```

Goal of this Lab (2/2)

- **You will learn**
  - how a dynamic memory manager works
  - how to implement a dynamic memory manager
  - how to work with macros in C
  - how to work with function pointers in C
  - how to debug code
  - that writing a dynamic memory manager is simple in theory and difficult in practice.

Environment setup (1/3)

- **You can get skeleton code and test bench from git repo**
  - `git clone https://github.com/SNU-ARC/2024_spring_sysprog_Lab3.git`

Environment setup (2/3)<Optional>

- If you want to keep your own repository, keep the lab's visibility to private. Otherwise, others would see your work.

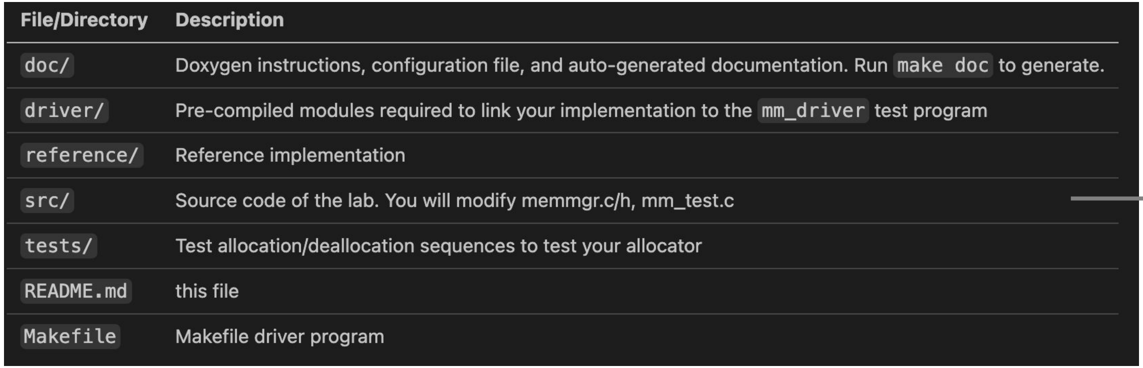
- Changing visibility
  - After cloning the repository, you should change the push remote URL to your own repository.

1. Create an empty repository that you're going to manage (again, keep it private)
2. Copy the url of that repository
3. On your terminal in the cloned directory, type

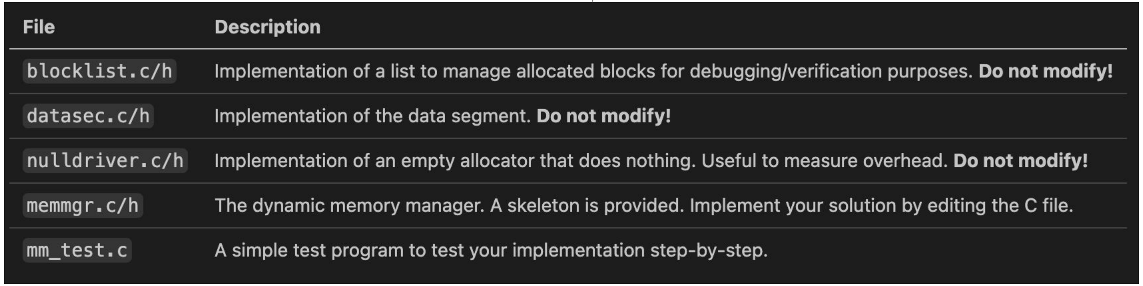
   ```
   git remote set-url --push origin <repo_url>
   ```
4. Check with git remote -v if the push URL has changed to yours while the fetch URL remains the same (this repo)

Environment setup (3/3)

- **The handout contains the following files and directories.**



File/Directory	Description
`doc/`	Doxygen instructions, configuration file, and auto-generated documentation. Run `make doc` to generate.
`driver/`	Pre-compiled modules required to link your implementation to the `mm_driver` test program
`reference/`	Reference implementation
`src/`	Source code of the lab. You will modify memmgr.c/h, mm_test.c
`tests/`	Test allocation/deallocation sequences to test your allocator
`README.md`	this file
`Makefile`	Makefile driver program



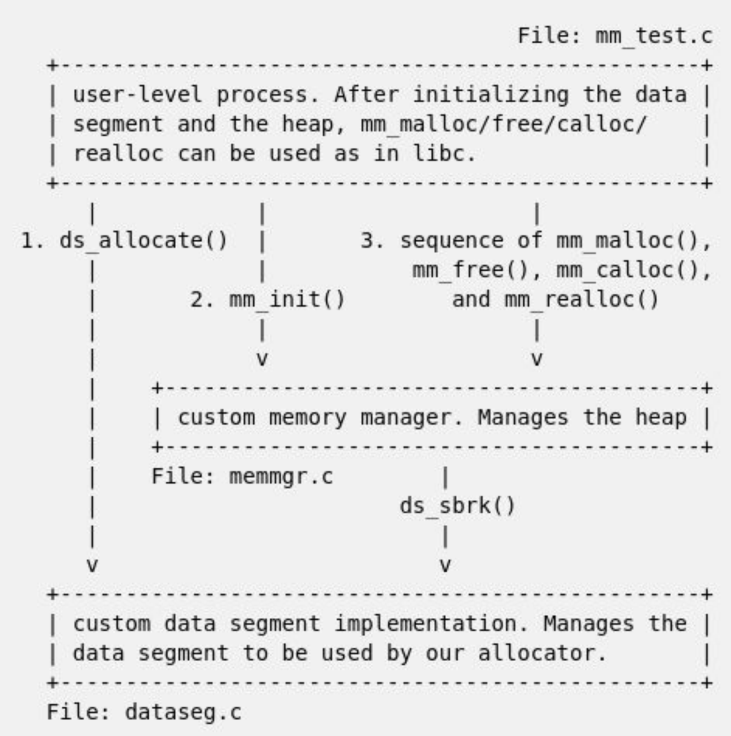
File	Description
`blocklist.c/h`	Implementation of a list to manage allocated blocks for debugging/verification purposes. **Do not modify!**
`datasec.c/h`	Implementation of the data segment. **Do not modify!**
`nulldriver.c/h`	Implementation of an empty allocator that does nothing. Useful to measure overhead. **Do not modify!**
`memmgr.c/h`	The dynamic memory manager. A skeleton is provided. Implement your solution by editing the C file.
`mm_test.c`	A simple test program to test your implementation step-by-step.

Overview

- **Handle memory allocation and management in memmgr.c**
  - Implement dynamic memory allocation and management functions such as *`malloc`*, *`realloc`* and *`free`*.
  - Implement both implicit and explicit free lists to track and allocate free blocks using the best fit policy.
  - For explicit free list, use the LIFO(Last-In-First-Out) policy.

Overview

- **We will use a simulated heap**
  - Real heap is not directly manipulable
  - dataseg.c will give a simulated heap space
  - memmgr.c will manage that heap space given by dataseg.c



```
                                             File: mm_test.c
 +--------------------------------------------------+
 | user-level process. After initializing the data |
 | segment and the heap, mm_malloc/free/calloc/     |
 | realloc can be used as in libc.                  |
 +--------------------------------------------------+
     |                |                  |
1. ds_allocate()      |       3. sequence of mm_malloc(),
     |                |            mm_free(), mm_calloc(),
     |          2. mm_init()           and mm_realloc()
     |                |                  |
     |                v                  v
     |      +-------------------------------------------+
     |      | custom memory manager. Manages the heap  |
     |      +-------------------------------------------+
     |      File: memmgr.c         |
     |                         ds_sbrk()
     |                             |
     v                             v
 +--------------------------------------------------+
 | custom data segment implementation. Manages the |
 | data segment to be used by our allocator.        |
 +--------------------------------------------------+
 File: dataseg.c
```

memmgr specification

- **64-bit operator**
  - One word = 8 bytes
- **Minimal Block size = 32 bytes**
  - All block addresses should be aligned to 32 bytes
- **Each block should have boundary tags**
  - Header & Footer
  - Bit 0 of each boundary tag should indicate the status of the block (1: allocated, 0: free)

memmgr specification

- *mm_malloc(size)*
  - Returns a pointer to an allocated payload block of at least size bytes.
  - The entire allocated block must lie within the heap region and must not overlap with any other block.

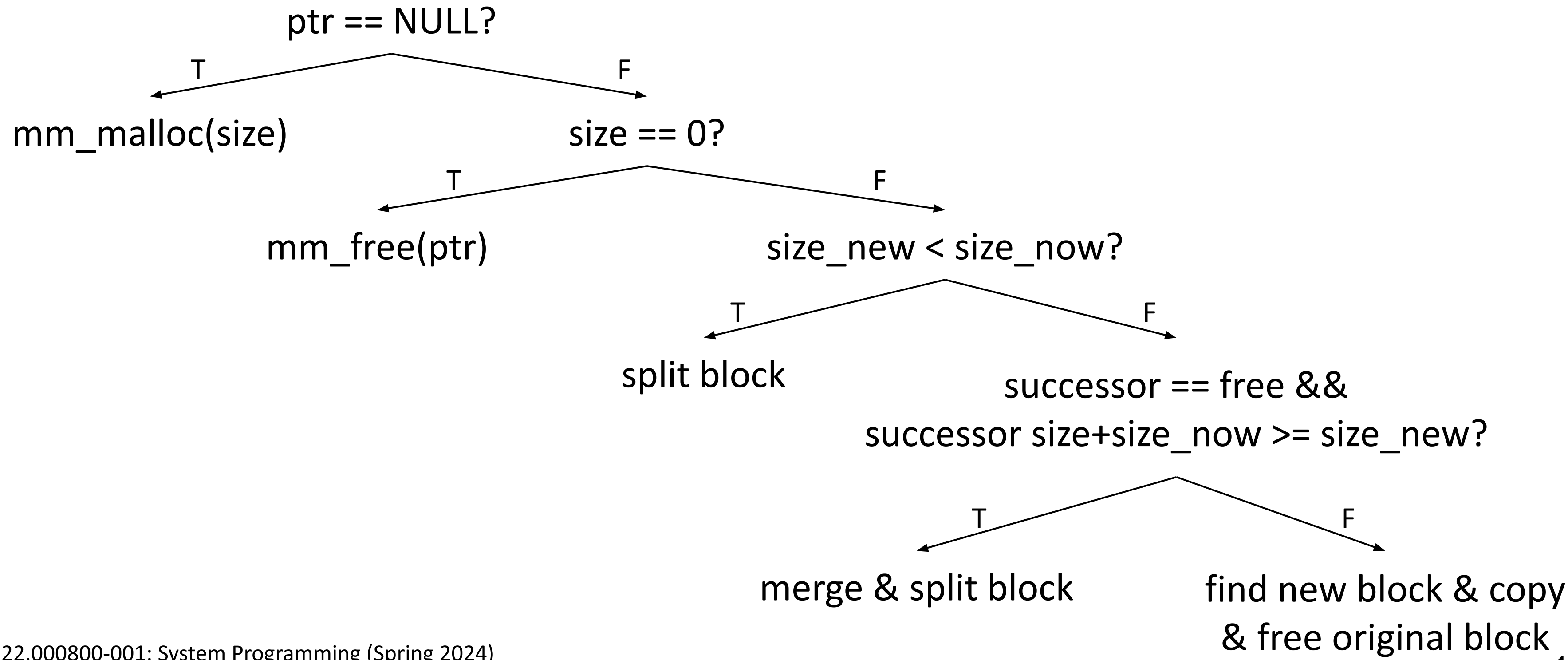
- *mm_free(ptr)*
  - Deallocate the block pointed to by `ptr` that was returned by an earlier call to *mm_malloc(), mm_calloc()*, or *mm_realloc()* and has not yet been freed.
  - When the caller attempts to release a memory block that has already been freed, an error message is generated.

memmgr specification

- *mm_realloc(ptr, size)*
  - If (***ptr*** == NULL), then *mm_malloc(size)*.
  - If (***size*** == 0), then *mm_free(ptr)*.
  - If (***ptr*** != NULL) then it must point to a valid allocated block.
    - if ***size****(new)* **< *size pointed by ptr***_(current)_
      - the block should be split into allocated block and a new free block.
    - if ***size****(new)* **> *size pointed by ptr***_(current)_
      - if successor block is free and large enough when merged, increase the block size and fix the remaining free block.
      - if not, assign a new block by the allocation policy(best-fit) and copy the original contents. Afterwards, the original block must be freed.

memmgr specification

- ***mm_realloc(ptr, size)***



ptr == NULL?

T

mm_malloc(size)

F

size == 0?

T

mm_free(ptr)

F

size_new < size_now?

T

split block

F

successor == free &&
successor size+size_now >= size_new?

T

merge & split block

F

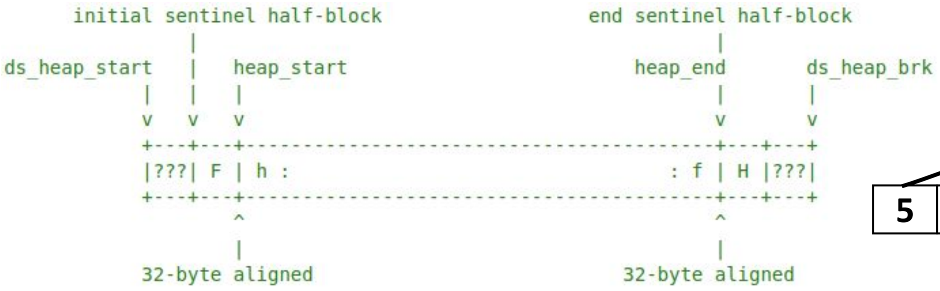
find new block & copy
& free original block

memmgr specification

- **Best-fit policy**
  - Select the smallest available block that is large enough
  - Aims to reduce fragmentation and optimize space utilization

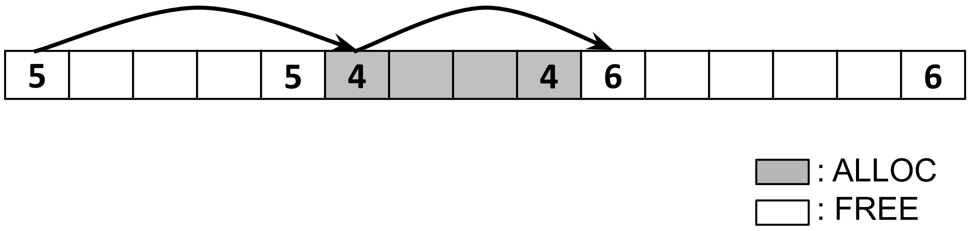
memmgr specification

- **Support different free list managements - Implicit free list**
    - Utilize the spaces within the memory itself to track free blocks



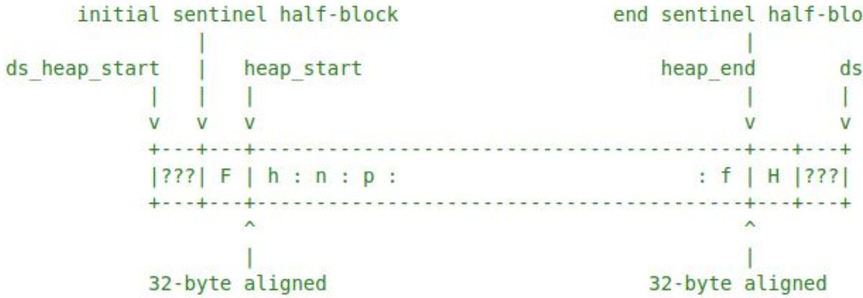
```
        initial sentinel half-block                          end sentinel half-block
                      |                                                 |
ds_heap_start         |    heap_start                              heap_end        ds_heap_brk
            |         |    |                                            |               |
            v         v    v                                            v               v
            +---+---+---------------------------------------------+---+---+
            |???| F | h :                                       : f | H |???|
            +---+---+---------------------------------------------+---+---+
                        ^                                           ^
                        |                                           |
            32-byte aligned                              32-byte aligned
```

- h,f: header/footer of free block
- H,F: header/footer of allocated block



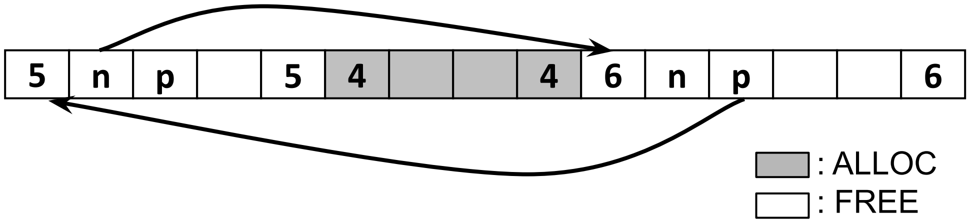
memmgr specification

- **Support different free list managements - Explicit free list**
  - Create a linked list of free blocks
  - Each block points to the next and previous block



```
      initial sentinel half-block                  end sentinel half-block
                |                                             |
ds_heap_start   |   heap_start                         heap_end       ds_heap_brk
            |   |   |                                         |       |
            v   v   v                                         v       v
            +---+---+-----------------------------------------+---+---+
            |???| F | h : n : p :                         : f | H |???|
            +---+---+-----------------------------------------+---+---+
                    ^                                         ^
                    |                                         |
            32-byte aligned                           32-byte aligned
```

- h,f: header/footer of free block
- n,p: next/previous pointer
- H,F: header/footer of allocated block



Code and Test (1/3)

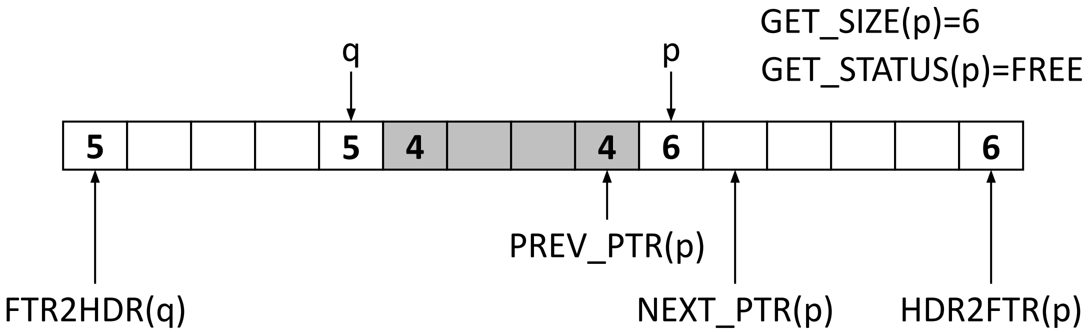
- **The skeleton provides some useful tools such as**
  - *macros* for easy data manipulation
  - *mm_check* for checking and dumping heap status
  - *functions* for logging information

- **You have to implement the following two parts:**
  - in ***mm_init(), mm_malloc(), mm_realloc(), mm_free()***
    - Main functions to allocate and manage blocks.
    - Functions should be compatible with both implicit and explicit free list.

  - in ***bf_get_free_block_implicit(), bf_get_free_block_explicit()***
    - Track and allocate a free block based on the best fit policy within each free list method.
    - For peak utilization, search for free blocks using the above principles.

Code and Test (1/3)

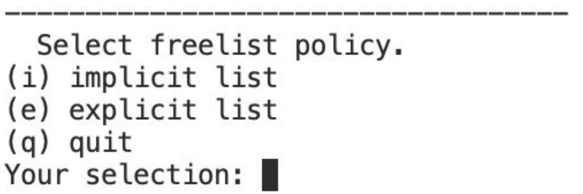
- **Macro examples**



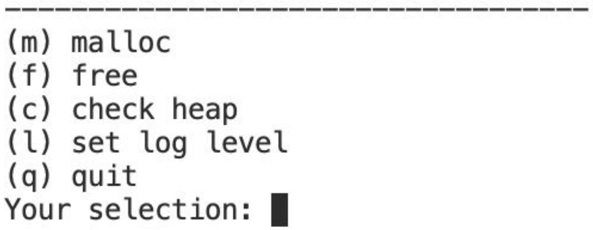
Code and Test (2/3)

- *mm_test*
  - An interactive testing tool to check your implementation
  - Reference *mm_test* is given, but do not take the outputs literally.
  - Build with *$ make mm_test*

$ mm_test

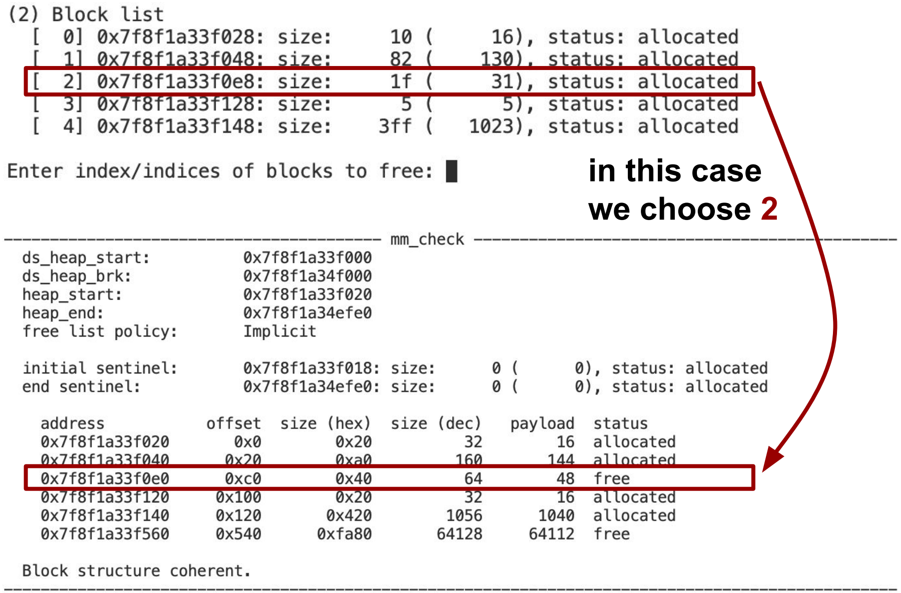


```
-----------------------------------
  Select freelist policy.
(i) implicit list
(e) explicit list
(q) quit
Your selection: █
```



```
-----------------------------------
(m) malloc
(f) free
(c) check heap
(l) set log level
(q) quit
Your selection: █
```

example) m 16 → m 130 → m 31 → m 5 → m 1023 → f



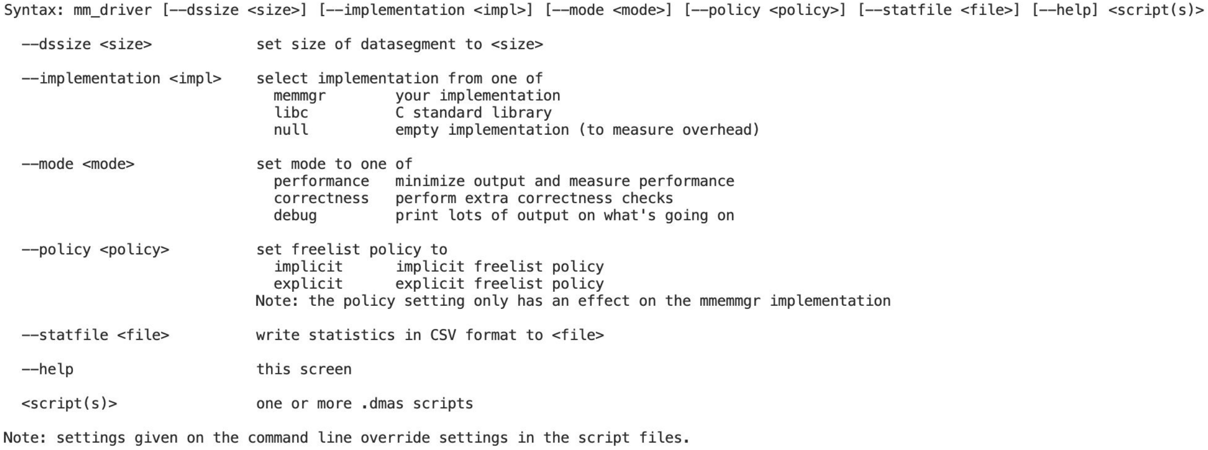
```
(2) Block list
  [  0] 0x7f8f1a33f028: size:     10 (     16), status: allocated
  [  1] 0x7f8f1a33f048: size:     82 (    130), status: allocated
  [  2] 0x7f8f1a33f0e8: size:     1f (     31), status: allocated
  [  3] 0x7f8f1a33f128: size:      5 (      5), status: allocated
  [  4] 0x7f8f1a33f148: size:    3ff (   1023), status: allocated

Enter index/indices of blocks to free: █


---------------------------------------- mm_check ----------------------------------------
  ds_heap_start:          0x7f8f1a33f000
  ds_heap_brk:            0x7f8f1a34f000
  heap_start:             0x7f8f1a33f020
  heap_end:               0x7f8f1a34efe0
  free list policy:       Implicit

  initial sentinel:       0x7f8f1a33f018: size:      0 (      0), status: allocated
  end sentinel:           0x7f8f1a34efe0: size:      0 (      0), status: allocated

    address             offset  size (hex)  size (dec)  payload  status
    0x7f8f1a33f020         0x0        0x20          32       16  allocated
    0x7f8f1a33f040        0x20        0xa0         160      144  allocated
    0x7f8f1a33f0e0        0xc0        0x40          64       48  free
    0x7f8f1a33f120       0x100        0x20          32       16  allocated
    0x7f8f1a33f140       0x120       0x420        1056     1040  allocated
    0x7f8f1a33f560       0x540      0xfa80       64128    64112  free

  Block structure coherent.
------------------------------------------------------------------------------------------
```

in this case we choose 2

Code and Test (3/3)

- *mm_driver*
  - Driver program that issues allocations and free requests from script
  - Can print stats at the end
  - Build with *$ make mm_driver*

$ mm_driver --help

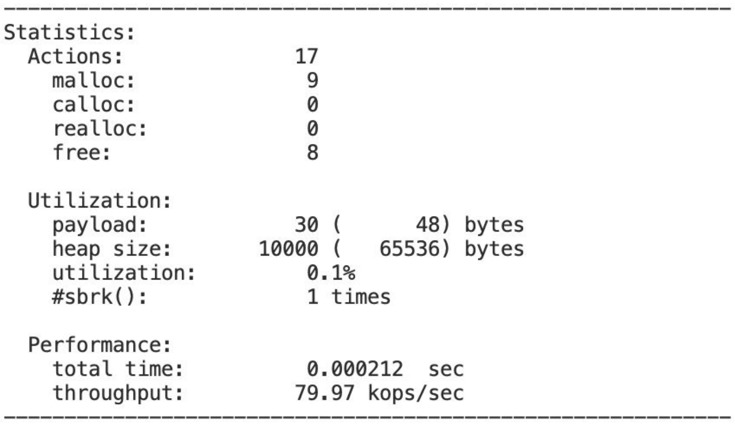


```
Syntax: mm_driver [--dssize <size>] [--implementation <impl>] [--mode <mode>] [--policy <policy>] [--statfile <file>] [--help] <script(s)>

  --dssize <size>            set size of datasegment to <size>

  --implementation <impl>    select implementation from one of
                               memmgr        your implementation
                               libc          C standard library
                               null          empty implementation (to measure overhead)

  --mode <mode>              set mode to one of
                               performance   minimize output and measure performance
                               correctness   perform extra correctness checks
                               debug         print lots of output on what's going on

  --policy <policy>          set freelist policy to
                               implicit      implicit freelist policy
                               explicit      explicit freelist policy
                             Note: the policy setting only has an effect on the mmemmgr implementation

  --statfile <file>          write statistics in CSV format to <file>

  --help                     this screen

  <script(s)>                one or more .dmas scripts

Note: settings given on the command line override settings in the script files.
```

Code and Test (3/3)

- *mm_driver*
  - If you want to own test, you can copy `*.dmas` and change test sequence.
  - You can change the policy by ‘*$ mm_driver --policy <policy>*’ or by changing ‘*heap <policy>*’ in `*.dmas`

$ mm_driver ./test/demo.dmas



```
------------------------------------------------------------
Statistics:
  Actions:                17
    malloc:                9
    calloc:                0
    realloc:               0
    free:                  8

  Utilization:
    payload:              30 (       48) bytes
    heap size:         10000 (    65536) bytes
    utilization:           0.1%
    #sbrk():               1 times

  Performance:
    total time:            0.000212   sec
    throughput:           79.97 kops/sec
------------------------------------------------------------
```

- test sequence in directory ‘test'
  - 100K.dmas
    - 100K allocation for performance comparison of explicit and implicit free list.
  - alloc.dmas
    - 2048 malloc for normal operation without error.
  - demo.dmas
    - Normal allocation for checking performance.
  - ls.dmas
    - Memory sequence of `ls -R`

Grading Policy

- **Test bench : 80 %**
  - Hidden test cases will be used along with the given example tests
- **Report** : **20 %**
  - Briefly explain your code including following information
    - How and when to increase/decrease the heap size
    - How to implement an explicit free list
    - (If any) Explain the added macros
  - Show and analyse the performance difference between implicit and explicit free list
- **For late submission:**
  - A deduction of 20%p per 24 hours

Submission (via eTL)

- **Write-up**
  - Briefly describe your implementation.
  - Filename: [student_id].pdf (example: 2024-12345.pdf)
  - **Please** submit it in **pdf** format. Other formats are not accepted.
- **Compress your source code and write-up into a single file**
  - Compress **memmgr.c** and your report
  - Filename should be [student_id].tar (example: 2024-12345.tar).
  - **Please** submit it in **tar** format. Other formats are not accepted.
  - Refer to README.md for submission instructions.
- **Submission deadline: by 23:59 on April 29, 2024**

Question?