Lab 3. Dynamic Memory Manager Module

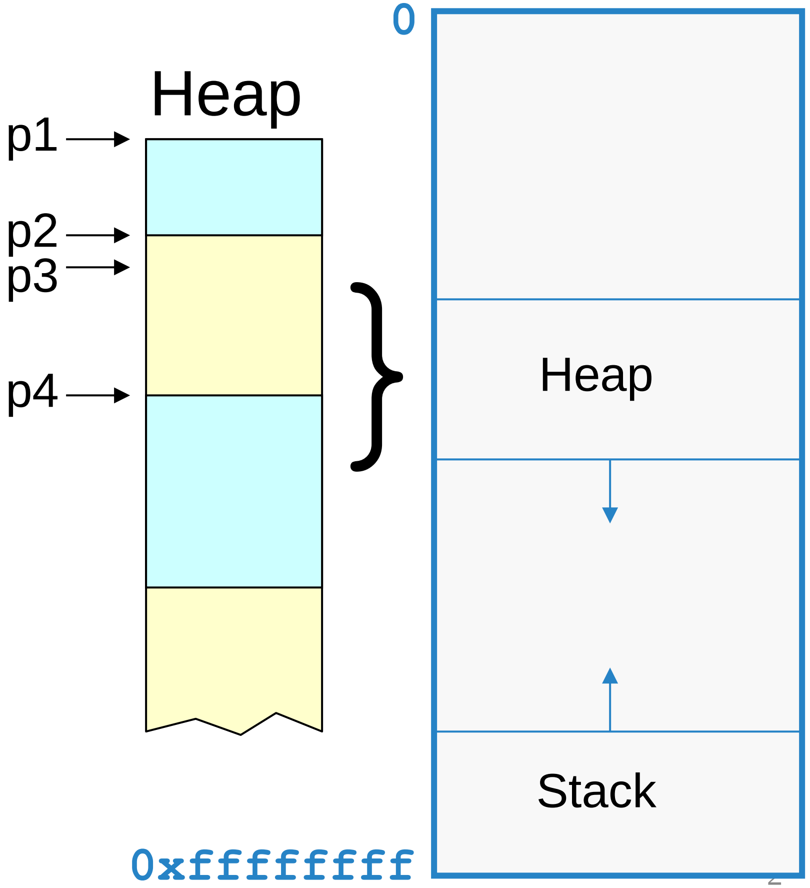
System Programming

Juyoung Park
SNU TNET Lab.

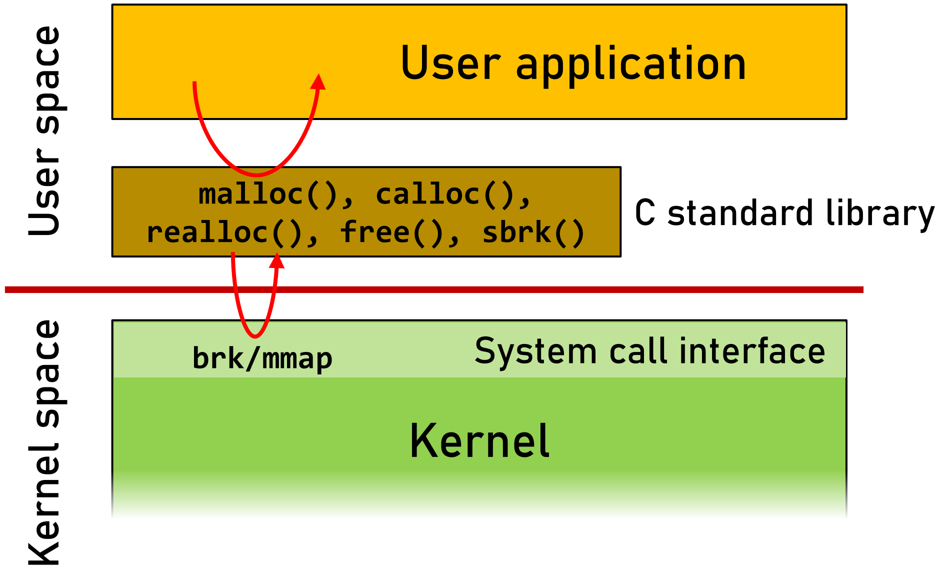
What Have You Learned

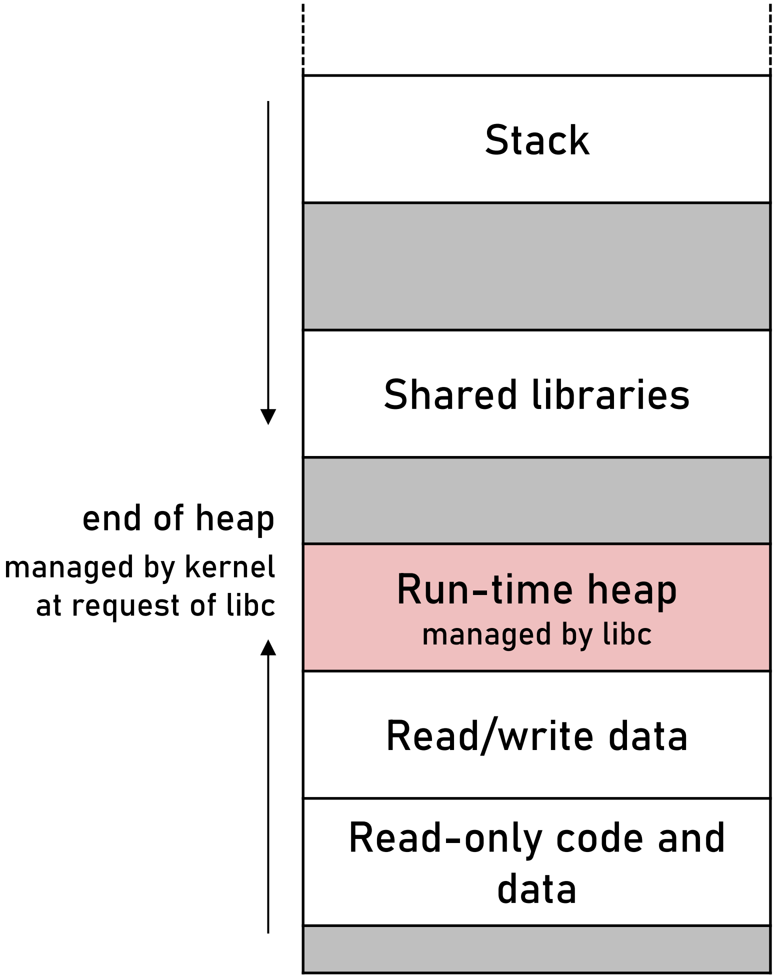
```
#include <stdlib.h>
void *malloc(size_t size);
void free(void *ptr);
```

```
char *p1 = malloc(3);
char *p2 = malloc(1);
char *p3 = malloc(4);
free(p2);
char *p4 = malloc(6);
free(p3);
char *p5 = malloc(2);
free(p1);
free(p4);
free(p5);
```



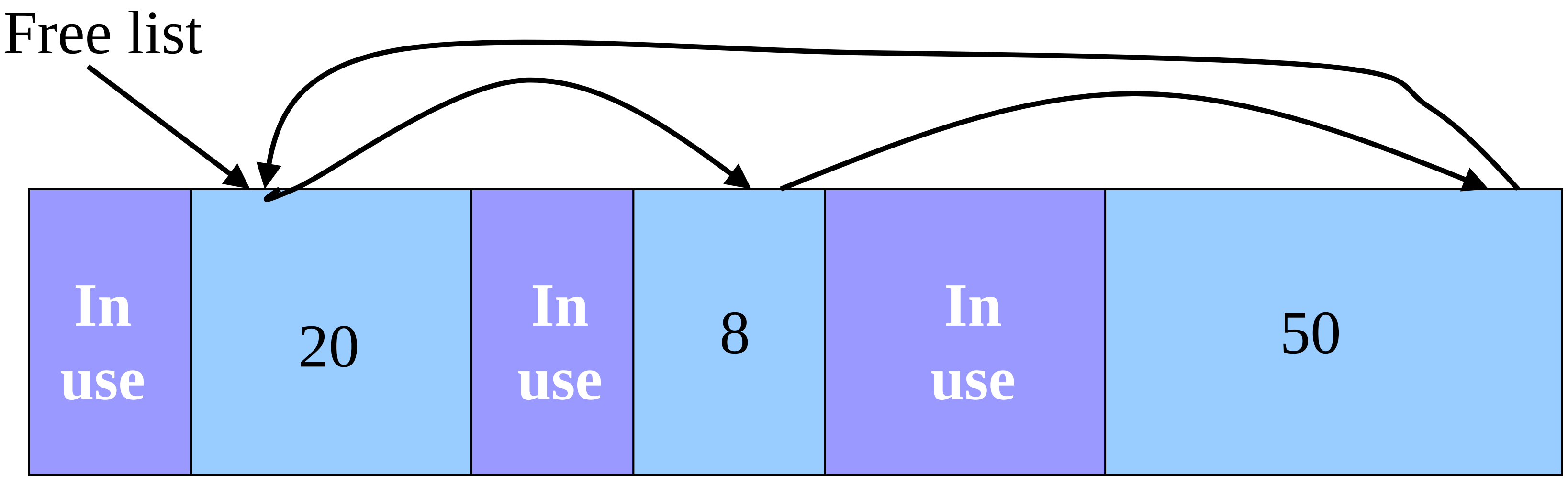
What Have You Learned





What Have You Learned

- K&R Heap Manager



Dynamic Memory Manager Module

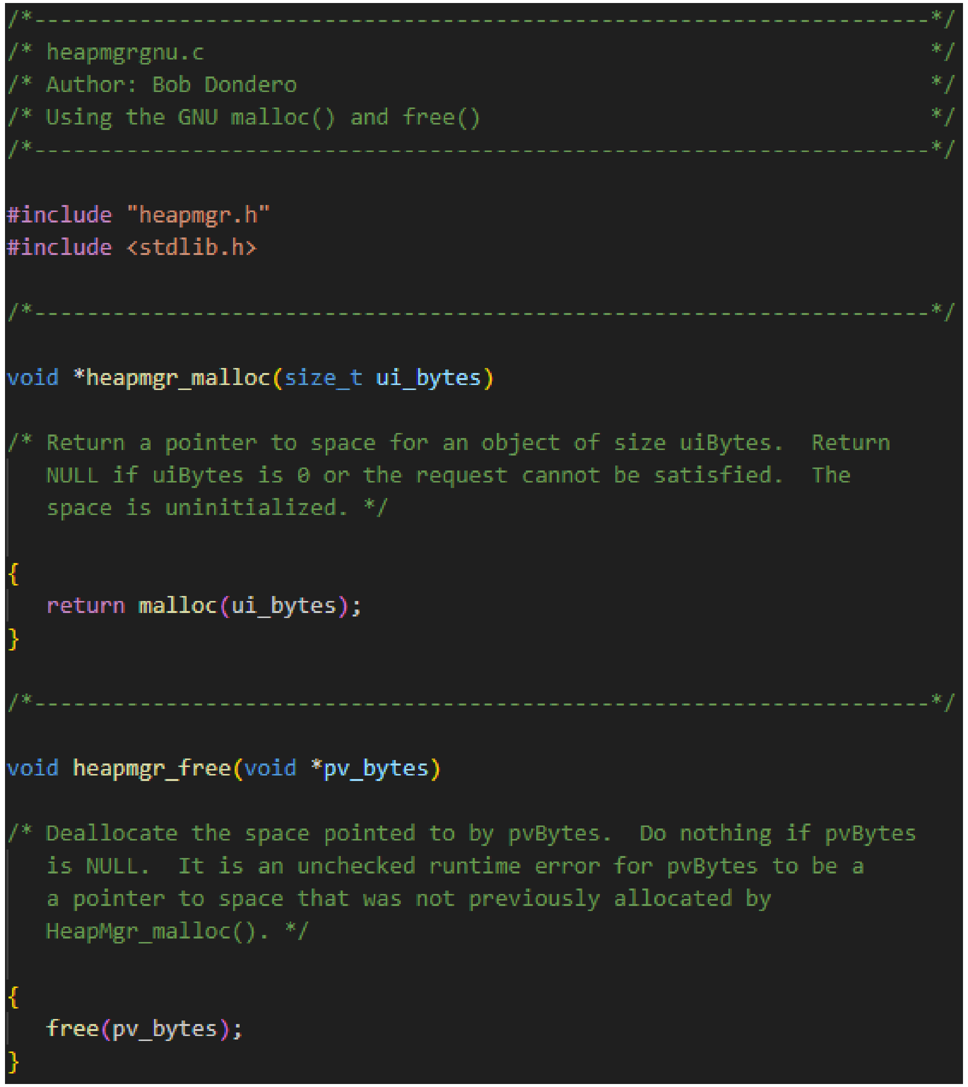
- Build a library that implements malloc() and free()
  - Without using GNU malloc(), free(), calloc(), or realloc()

- The code for your reference(heapmgrgnu.c, heapmgrkr.c) and baseline code (heapmgrbase.c) will be given

- Guidance on the assignment can also be found on README.md

Given Code - heapmgrgnu.c

- Implementation that simply calls the GNU malloc() and free()



```c
/*--------------------------------------------------------------------*/
/* heapmgrgnu.c                                                       */
/* Author: Bob Dondero                                                */
/* Using the GNU malloc() and free()                                  */
/*--------------------------------------------------------------------*/

#include "heapmgr.h"
#include <stdlib.h>

/*--------------------------------------------------------------------*/

void *heapmgr_malloc(size_t ui_bytes)

/* Return a pointer to space for an object of size uiBytes.  Return
   NULL if uiBytes is 0 or the request cannot be satisfied.  The
   space is uninitialized. */

{
   return malloc(ui_bytes);
}

/*--------------------------------------------------------------------*/

void heapmgr_free(void *pv_bytes)

/* Deallocate the space pointed to by pvBytes.  Do nothing if pvBytes
   is NULL.  It is an unchecked runtime error for pvBytes to be a
   a pointer to space that was not previously allocated by
   HeapMgr_malloc(). */

{
   free(pv_bytes);
}
```

Given Code - heapmgrkr.c

- Kernighan and Ritchie (K&R) implementation
  - With small modification for the sake of simplicity
  - `void *heapmgr_malloc(size_t nbytes)`
  - `void heapmgr_free(void *ap)`
  - `Header *morecore(unsigned int nu)`
- A circular, singly-linked list

Given Code - heapmgrbase.c

- Implements baseline code
  - You can start the task with this code

```
void *heapmgr_malloc(size_t size)
void heapmgr_free(void *m)
int check_heap_validity(void)
```

  - Validity check for entire data structures for chunks

  - Other functions for implementation

- A non-circular, singly-linked list

Chunk Structure

```
/* chunkbase.h */
typedef struct Chunk *Chunk_T;


/* chunkbase.c */
struct Chunk {
   /* Pointer to the next chunk in the free chunk list */
   Chunk_T next;
   /* Capacity of a chunk (chunk units) */
   int units;
   /* CHUNK_FREE or CHUNK_IN_USE */
   int status;
};
```

Chunk and Block

- Chunk: base unit for allocate memory
- Block: set of contiguous chunks that store same data

- Free blocks are connected in a linked list
  - which is called a free list

Given Code - heapmgrbase.c

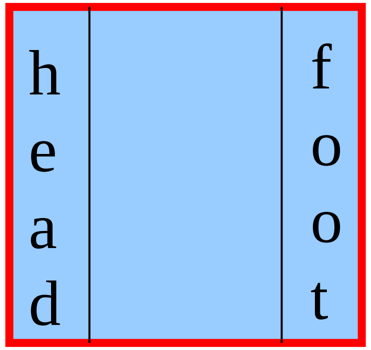
- int check_heap_validity(void)
  - Checks the validity of chunk data structures (chunk: a base unit for allocation)
  - Returns 1 on success or 0 (zero) on failure

- assert(condition);
  - If “condition” evaluates to false, the program will print an error message and terminate

- heapmgrbase.c calls assert(check_heap_validity())
  - At leading and trailing edges of heapmgr_malloc() and heapmgr_free()
  - Checks the integrity of the heap
  - If this assert() fails, it implies that something’s wrong

Your to-do: Make free() faster

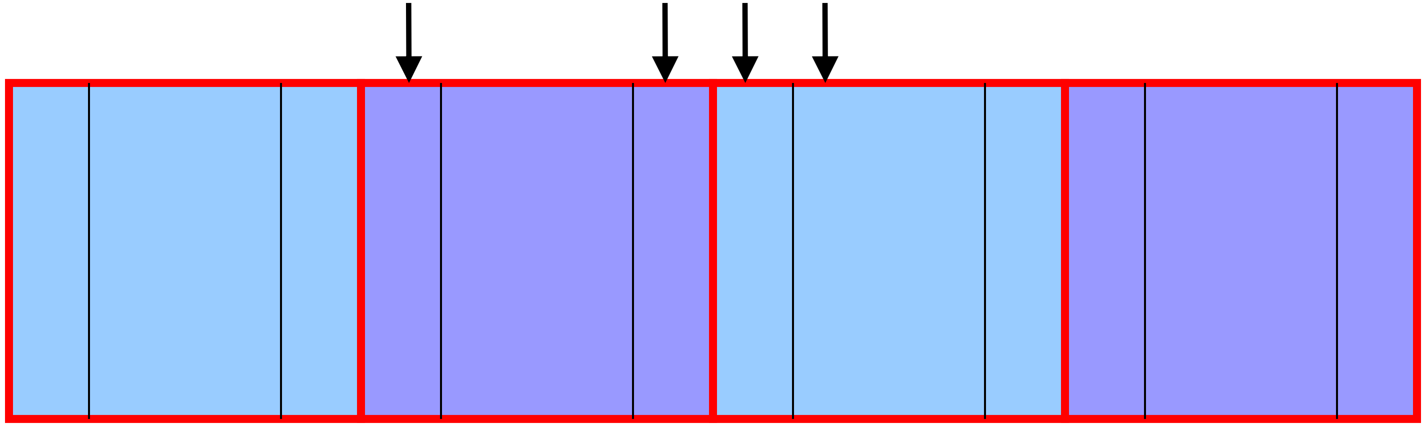
- Base requirement for assignment 3 (write code in heapmgr1.c)
- Implement a doubly-linked free list with the chunk data structure
  - Each chunk now contains a header and a footer (as described in lectures)
  - Chunk is a base unit (e.g., allocate memory in multiples of this unit)



- heapmgr1.c (with footer) should make free() faster
  - Without needing to find/maintain the “previous” node for inserting a freed memory into the free list (K&R)

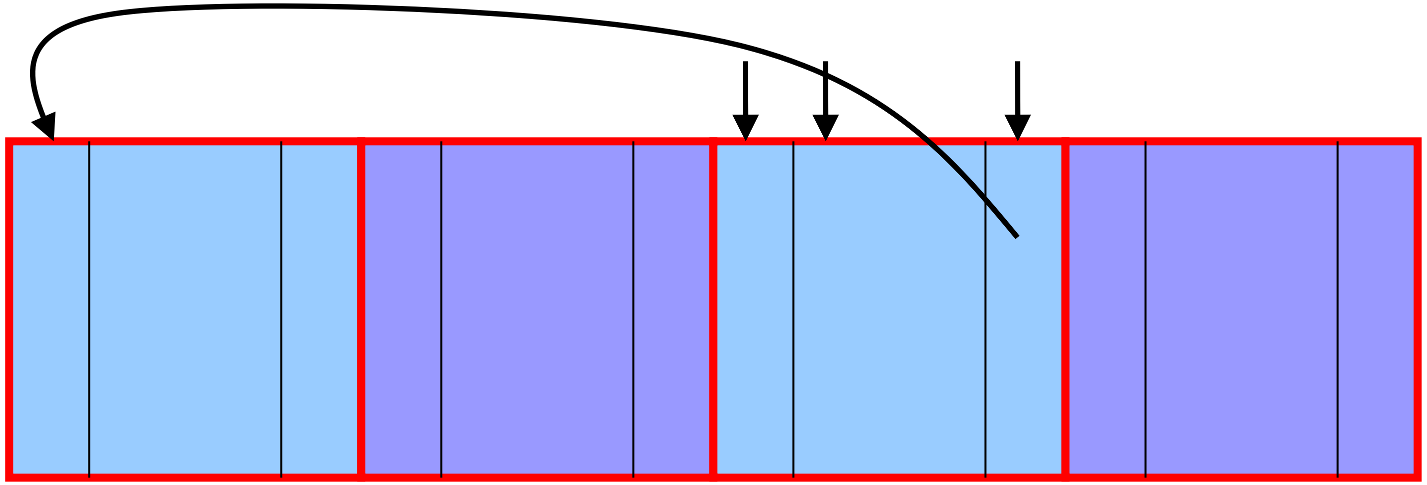
Navigating Previous/Next Contiguous Block

- Start with the user’s data portion of the block
- Go backwards to the head of the block
  - Easy, since you know the size of the header
- Go backwards to the footer of the previous block
  - Easy, since you know the size of the footer
- Go backwards to the header of the previous block
  - Easy, since you know the size from the footer



Navigating Previous/Next Free Block

- Start with the user’s data portion of the block
- Go backwards to the head of the block
  - Easy, since you know the size of the header
- Go forwards to the footer of the block
  - Easy, since you know the block size from the header
- Go backwards to the previous free block
  - Easy, since you have the previous free pointer



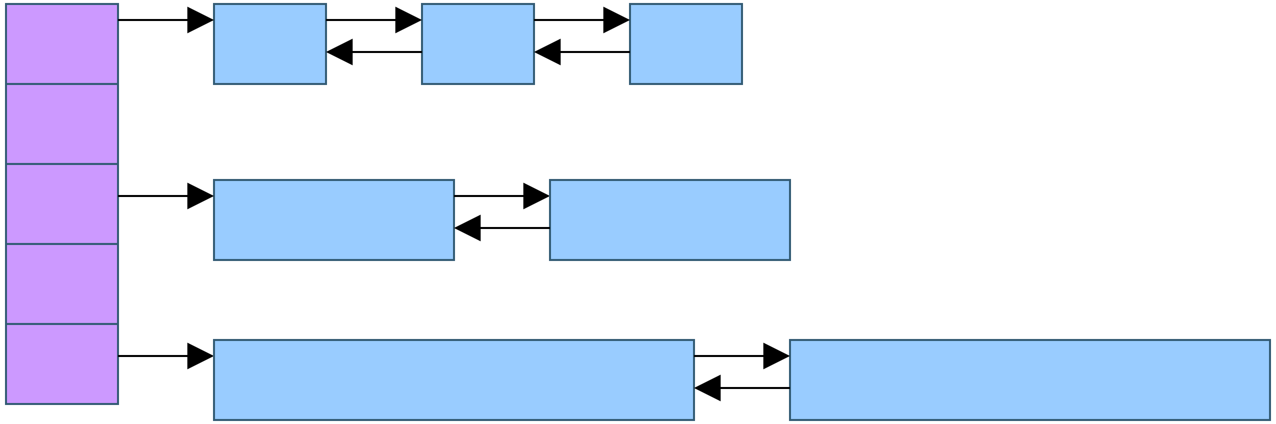
Strategy for Writing heapmgr1.c

- You can start with heapmgrbase.c and enhance it
  - Or you can ignore heapmgrbase.c and do your own way

- You can implement either circular or non-circular list

Extra-credit: Make malloc() faster

- heapmgr1.c shows poor worst-case behavior, so try to enhance it
  - Finding the free block will traverse the list in heapmgr1.c
- Use multiple doubly-linked lists, alias bins (as described in lectures)



Strategy for Binning

- How to set the range of sizes covered by bin?
  - Fixed: 1-10, 11-20, 21-30, …
  - Exponential: 1-2, 3-4, 5-8, 9-16, …

- How to handle the newly allocated large memory?
  - Split into small chunks in advance
  - Allocate in large bin and wait for split in malloc()

- How do you coalesce when free() from binning?
  - Check after coalesce and move to proper bin
  - Move to final bin first and coalesce

chunk.c and chunk.h

- heapmgrbase.c uses chunkbase.c and chunkbase.h to use ‘struct Chunk’ and functions

- If you use heapmgrbase.c as skeleton code, you can use chunkbase.c and chunkbase.h for chunk.c and chunk.h

- If you do not use heapmgrbase.c, you have two choices:
  - 1. Define useful structures and functions in chunk.c and chunk.h and use them
  - 2. Just leave chunk.c and chunk.h as is and not use them

Useful Functions in chunkbase.c & chunkbase.h

- chunk_get_status(Chunk_T c)
  - Returns a chunk's status
- chunk_set_status(Chunk_T c, int status)
  - Set the status of the chunk
- chunk_get_units(Chunk_T c)
  - Returns the size of a chunk
- chunk_set_units(Chunk_T c, int units)
  - Sets the current size in 'units' of 'c'

Useful Functions in chunkbase.c & chunkbase.h

- chunk_get_next_free_chunk(Chunk_T c)
  - Returns the next free chunk in free chunk list
- chunk_set_next_free_chunk(Chunk_T c, Chunk_T next)
  - Sets the next free chunk of 'c' to 'next'
- chunk_get_next_adjacent(Chunk_T c, void *start, void *end)
  - Returns the next adjacent chunk to 'c' in memory space
- chunk_is_valid(Chunk_T c, void *start, void *end);
  - Checks the validity of a chunk

Memory Utilization

- Implement the strategies for good memory utilization
  - All techniques learned in class

  - Check blocks in free list before allocate new memory
  - Divide the free block if free block is bigger then requested
  - Check lower/upper neighbor and coalesce

- If you ignore memory utilization, you won't get points no matter how fast your implementation is

How to Test Your Code

- To test your heapmgr implementations:

```
$ gcc800 -std=gnu99 testheapmgr.c heapmgr1.c chunk.c -o testheapmgr1
$ gcc800 -std=gnu99 testheapmgr.c heapmgr2.c chunk.c -o testheapmgr2
```

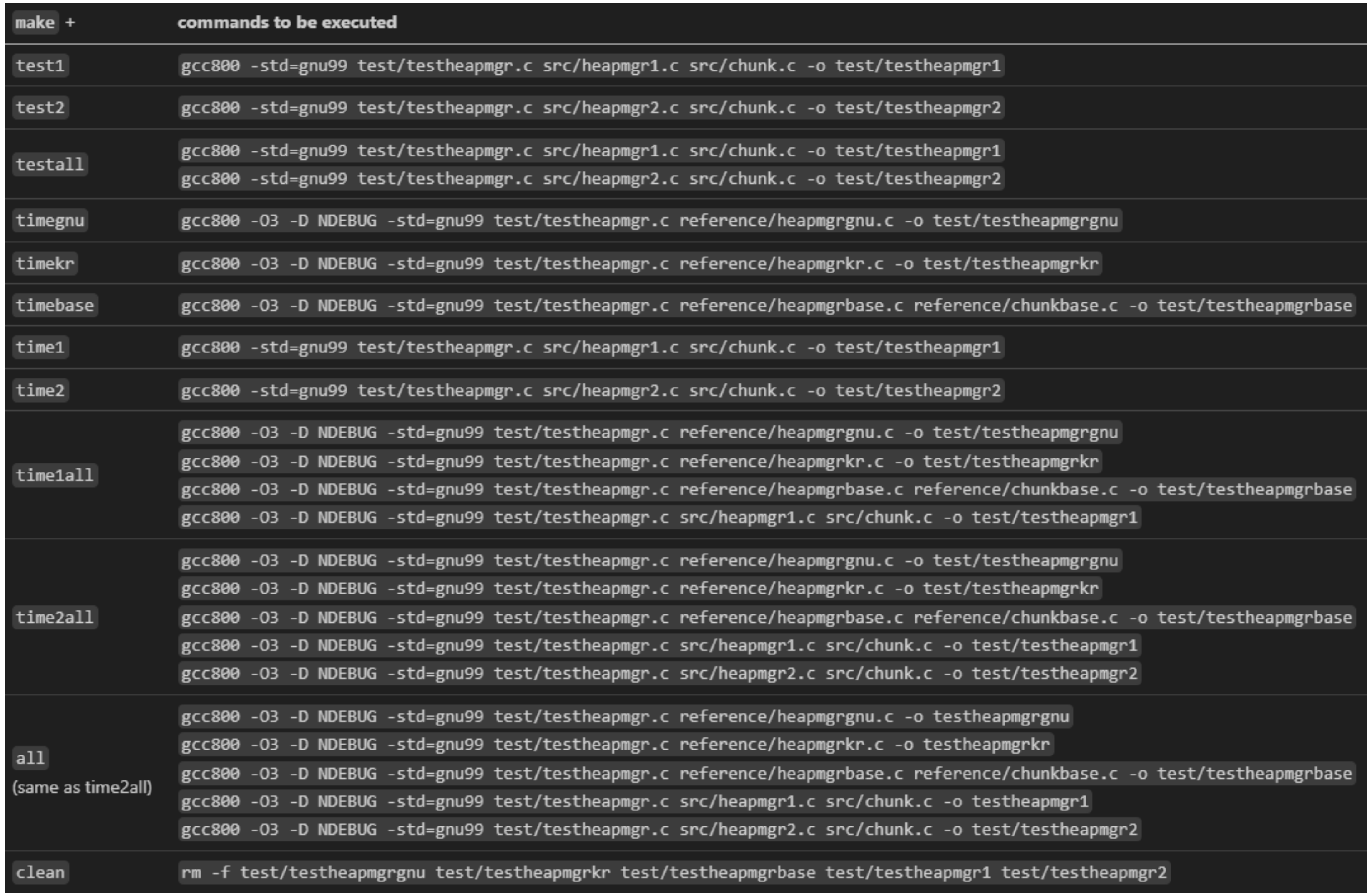
- To collect timing statistics:

```
$ gcc800 -O3 -D NDEBUG -std=gnu99 testheapmgr.c heapmgrgnu.c -o testheapmgrgnu
$ gcc800 -O3 -D NDEBUG -std=gnu99 testheapmgr.c heapmgrkr.c -o testheapmgrkr
$ gcc800 -O3 -D NDEBUG -std=gnu99 testheapmgr.c heapmgrbase.c chunkbase.c -o testheapmgrbase
$ gcc800 -O3 -D NDEBUG -std=gnu99 testheapmgr.c heapmgr1.c chunk.c -o testheapmgr1
$ gcc800 -O3 -D NDEBUG -std=gnu99 testheapmgr.c heapmgr2.c chunk.c -o testheapmgr2
```

- Don't forget -std=gnu99 ; otherwise you'll get error while compiling

How to Test Your Code

- You can also use Makefile to build executable files



make +	commands to be executed
test1	gcc800 -std=gnu99 test/testheapmgr.c src/heapmgr1.c src/chunk.c -o test/testheapmgr1
test2	gcc800 -std=gnu99 test/testheapmgr.c src/heapmgr2.c src/chunk.c -o test/testheapmgr2
testall	gcc800 -std=gnu99 test/testheapmgr.c src/heapmgr1.c src/chunk.c -o test/testheapmgr1 gcc800 -std=gnu99 test/testheapmgr.c src/heapmgr2.c src/chunk.c -o test/testheapmgr2
timegnu	gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c reference/heapmgrgnu.c -o test/testheapmgrgnu
timekr	gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c reference/heapmgrkr.c -o test/testheapmgrkr
timebase	gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c reference/heapmgrbase.c reference/chunkbase.c -o test/testheapmgrbase
time1	gcc800 -std=gnu99 test/testheapmgr.c src/heapmgr1.c src/chunk.c -o test/testheapmgr1
time2	gcc800 -std=gnu99 test/testheapmgr.c src/heapmgr2.c src/chunk.c -o test/testheapmgr2
time1all	gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c reference/heapmgrgnu.c -o test/testheapmgrgnu gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c reference/heapmgrkr.c -o test/testheapmgrkr gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c reference/heapmgrbase.c reference/chunkbase.c -o test/testheapmgrbase gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c src/heapmgr1.c src/chunk.c -o test/testheapmgr1
time2all	gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c reference/heapmgrgnu.c -o test/testheapmgrgnu gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c reference/heapmgrkr.c -o test/testheapmgrkr gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c reference/heapmgrbase.c reference/chunkbase.c -o test/testheapmgrbase gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c src/heapmgr1.c src/chunk.c -o test/testheapmgr1 gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c src/heapmgr2.c src/chunk.c -o test/testheapmgr2
all (same as time2all)	gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c reference/heapmgrgnu.c -o testheapmgrgnu gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c reference/heapmgrkr.c -o testheapmgrkr gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c reference/heapmgrbase.c reference/chunkbase.c -o test/testheapmgrbase gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c src/heapmgr1.c src/chunk.c -o testheapmgr1 gcc800 -O3 -D NDEBUG -std=gnu99 test/testheapmgr.c src/heapmgr2.c src/chunk.c -o test/testheapmgr2
clean	rm -f test/testheapmgrgnu test/testheapmgrkr test/testheapmgrbase test/testheapmgr1 test/testheapmgr2

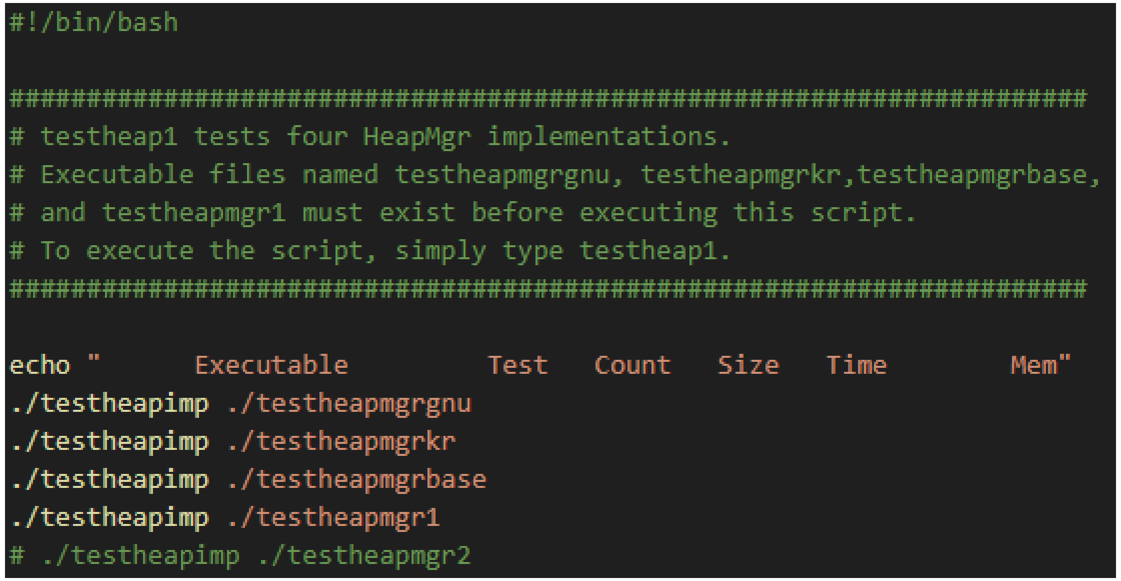
How to Test Your Code

- If you want to use Makefile, move the codes you implemented (chunk.c, chunk.h, heapmgr1.c, hepmgr2.c) in src folder
- You don’t need to move the existing codes
- Executable files will be created in the test folder

```
assignment3
 |- reference
 |   |- chunkbase.c / chunkbase.h
 |   |- heapmgr.h
 |   |- heapmgrgnu.c / heapmgrkr.c
 |   `- heapmgrbase.c
 |- src
 |   |- chunk.c / chunk.h
 |   `- heapmgr1.c / heapmgr2.c
 |- test
 |   |- heapmgr.h
 |   `- testheapmgr.c
 `-Makefile
```

How to Test Your Code

- Bash shell scripts (testheap1 & testheap2) will be provided
- testheap1 runs testheapmgr.c to test four cases (heapmgrgnu.c, heapmgrkr.c, heapmgrbase.c, heapmgr1.c) and reports timing and memory usage statistics



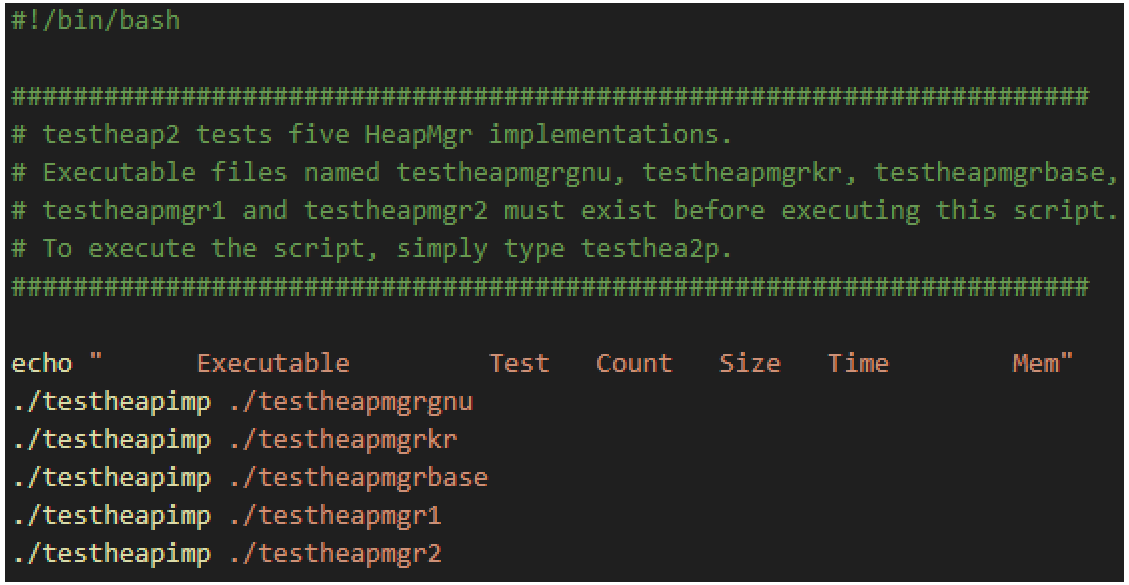
```bash
#!/bin/bash

######################################################################
# testheap1 tests four HeapMgr implementations.
# Executable files named testheapmgrgnu, testheapmgrkr,testheapmgrbase,
# and testheapmgr1 must exist before executing this script.
# To execute the script, simply type testheap1.
######################################################################

echo "      Executable         Test   Count   Size   Time        Mem"
./testheapimp ./testheapmgrgnu
./testheapimp ./testheapmgrkr
./testheapimp ./testheapmgrbase
./testheapimp ./testheapmgr1
# ./testheapimp ./testheapmgr2
```

How to Test Your Code

- testheap2 runs testheapmgr.c to test five cases (heapmgrgnu.c, heapmgrkr.c, heapmgrbase.c, heapmgr1.c, and heapmgr2.c) and reports timing and memory usage statistics

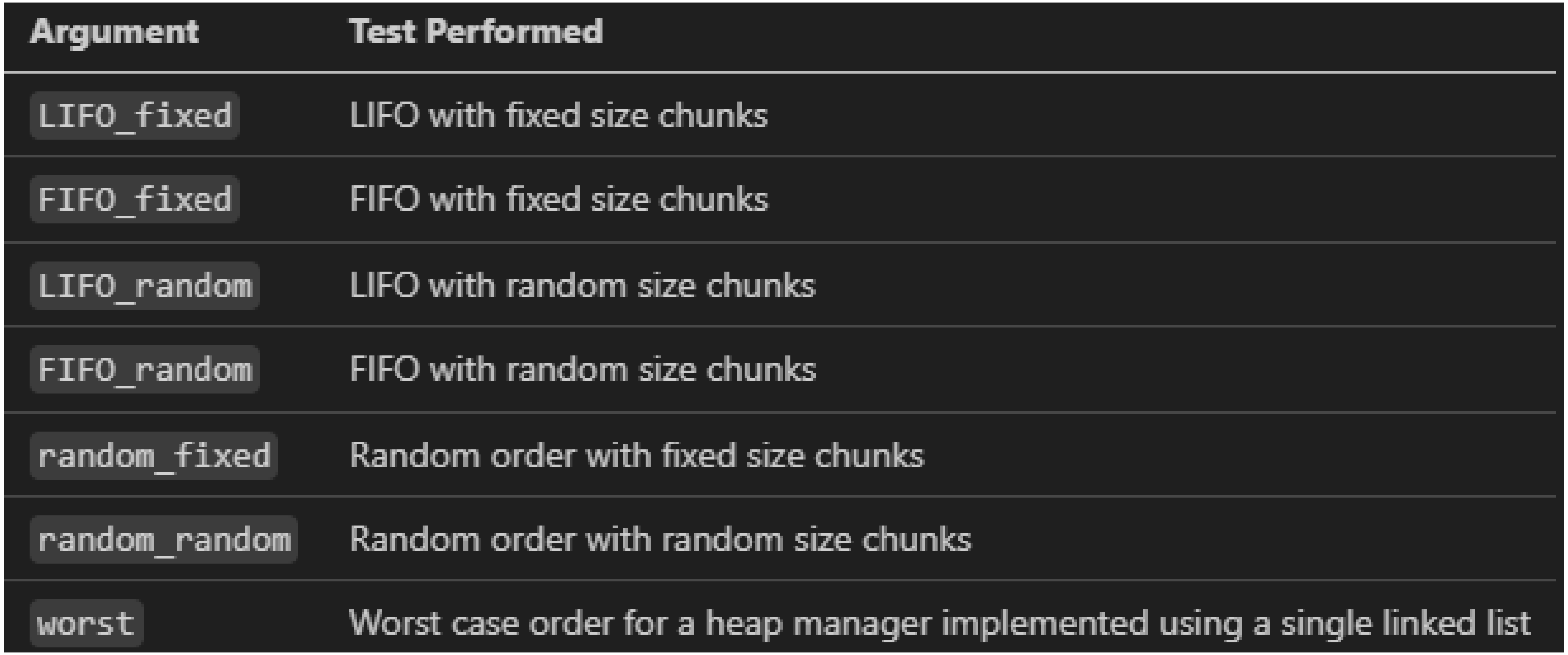


```
#!/bin/bash

#######################################################################
# testheap2 tests five HeapMgr implementations.
# Executable files named testheapmgrgnu, testheapmgrkr, testheapmgrbase,
# testheapmgr1 and testheapmgr2 must exist before executing this script.
# To execute the script, simply type testhea2p.
#######################################################################

echo "      Executable          Test   Count   Size   Time        Mem"
./testheapimp ./testheapmgrgnu
./testheapimp ./testheapmgrkr
./testheapimp ./testheapmgrbase
./testheapimp ./testheapmgr1
./testheapimp ./testheapmgr2
```

How to Test Your Code



Argument	Test Performed
`LIFO_fixed`	LIFO with fixed size chunks
`FIFO_fixed`	FIFO with fixed size chunks
`LIFO_random`	LIFO with random size chunks
`FIFO_random`	FIFO with random size chunks
`random_fixed`	Random order with fixed size chunks
`random_random`	Random order with random size chunks
`worst`	Worst case order for a heap manager implemented using a single linked list

How to Test Your Code

- $ ./testheap1 (or ./testheap2)
  - Provides timing and memory usage statistics for all codes

- $ ./testheapmgr1 LIFO_fixed 100 1000
  - Perform a LIFO_fixed test with testheapmgr1
  - Run heapmgr_malloc() and heapmgr_free() 100 times
  - The (maximum) size of each memory chunk is 1000 bytes

- testheap1 and testheap2 is in test folder

How to Test Your Code

- Set the product of the number of calls and size in bytes to less than 10^9
  - $ ./testheapmgr1 LIFO_fixed 100000 100000 (X)

- In all tests evaluating the implementation on the Bacchus machine, (number of cells) x (size in bytes) $\leq 10^9$ is guaranteed

- Only tests for final check should be performed on the Bacchus machine
  - Otherwise, test in your local machine

Content of readme file

- Your name and student ID
- Result of ./testheap1 or ./testheap2 (paste the output of the testheap1 or testheap2 script)

- (Optionally) An indication of how much time you spent doing the assignment
- (Optionally) Your assessment of the assignment
- (Optionally) Any information that will help us to grade your work in the most favorable light

How to Submit?

- Make a directory

  -
    ```
    $ mkdir 202400000_assign3
    ```

- Move your code and readme file there

  -
    ```
    $ mv heapmgr1.c (heapmgr2.c) chunk.c chunk.h readme 202400000_assign3
    ```

- Make a gzipped tar file for submission

  -
    ```
    $ tar zcf 202400000_assign3.tar.gz 202400000_assign3
    ```

- You **must** submit chunk.c and chunk.h even if you did not use them in your implementation
  - In this case, submit chunk.c and chunk.h given in the assignment without modification

How to Submit?

Structure of directory:

```
YourID_assign3 (don’t use dash)
 |-heapmgr1.c
 |-heapmgr2.c (optional)
 |-chunk.c
 |-chunk.h
 `-readme (don’t use extension such as .txt, .md, …)
```

Example:

```
202400000_assign3
 |-heapmgr1.c
 |-chunk.c
 |-chunk.h
 `-readme
```

- Please set **files and directory’s names** to match the examples above
  - Don’t use any extension for readme file
  - Don’t use dash for submit file
- Structure files and directories as shown above, then proceed with **compression**
- **Deadline: ~11.1(Fri) 21:00**
  - **0 points if deadline is missed**

Grading - heapmgr1.c

- Submit format (12 / 100)
  - readme, files with proper names

- Evaluation from the user’ viewpoint: function correctness (78 / 100)
  - heapmgr_malloc(), heapmgr_free() are well-designed (include validity check)
  - time consumption: faster than heapmgrkr.c, heapmgrbase.c (except worst)

- Evaluation from the programmer’s viewpoint (10 / 100)
  - Clarity (names, comments, line lengths, indentation, etc.)
  - Parameter validation using assert()

Grading - heapmgr2.c (Extra Credit)

- Evaluation from the user' viewpoint: function correctness (+30%)
  - heapmgr_malloc(), heapmgr_free() are well-designed (using bins) (include validity check)
  - time consumption: faster than heapmgrkr.c, heapmgrbase.c and heapmgr1.c

- Extra credit
  - Up to 30% of the scores you earn for implementing heapmgr1.c

- You will not get the 30% extra credit just by submitting heapmgr2
  - If you do not get a perfect score on heapmgr2, extra credit you receive will be reduced