# ICS 53, Winter 2020

# Assignment 4: A Memory Allocator

You will write a program which maintains a heap. Your program will allow a user to allocate memory, free memory, and see the current state of the heap. Your program will accept user commands and execute them.

## Assumptions about the heap

The heap is 127 bytes long and memory is byte-addressable. The first address of the heap is address 0, so the last address of the heap is address 126. When we refer to a "pointer" in this assignment we mean an address in memory. All pointers should therefore be values between 0 and 126.

The heap will be organized as an implicit free list. The heap is initially completely unallocated, so it should contain a single free block which is as big as the entire heap. Memory is initialized so that all addresses (other than the header of the initial free block) contain 0. Each block should have a header which is a single byte and the header byte should be contained in memory, just before the payload of the block. The most-significant 7 bits of the header should indicate the size of the block, including the header itself. The least significant bit of the header should indicate the allocation of the block: 0 for free, 1 for allocated. The header for the first block (the initial single free block) must be placed at address 0 in memory.

## Operations

Your program should provide a prompt to the user (">") and accept the following commands.

**malloc** - This operation allows the user to allocate a block of memory from your heap. This operation should take one argument, the number of bytes which the user wants in the payload of the allocated block. This operation should print out a pointer which is the first address of the payload of the allocated block.

Example:

```
>malloc 10

1

>malloc 5

12

>malloc 2

18
```

**free** - This operation allows the user to free a block of memory. This operation takes one argument, the pointer to the payload of a previously allocated block of memory. You can assume that the argument is a correct pointer to the payload of an allocated block.

Example:

```
>malloc 10

1

>malloc 5

12

>free 12

>free 1
```

**blocklist** - This operation prints out information about all of the blocks in your heap. The information about blocks should be printed in the order that the blocks are contained in the heap. The following information should be printed about each block: pointer to the payload, block size, and the allocation status (allocated of free). All three items of information about a single block should be printed on a single line and should be separated by commas.

Example:

```
>malloc 10

1

>malloc 5

12

>blocklist

1, 10, allocated

12, 5, allocated

18, 109, free
```

**writemem –** This operation writes alpha-numeric characters into memory. The operation takes two arguments and there should be no empty spaces or null character at the end of the input. The first argument is a pointer to the location in memory and the second argument is a sequence of alpha-numeric characters which will be written into memory, starting at the address indicated by the pointer. The first character will be written into the address indicated by the pointer, and each character thereafter will be written into the neighboring addresses

sequentially. For example, the operation "`writemem 3 abc`" will write an 'a' into address 3, a 'b' into address '4', and a 'c' into address 5.

You can assume that the pointer argument will always be an address in the heap, assume that all of the characters will be written into addresses in the heap.

**printmem** – This operation prints out a segment of memory in hexadecimal. The operation takes two arguments. The first argument is a pointer to the first location in memory to print, and the second argument is an integer indicating how many addresses to print. The contents of all addresses will be printed on a single line and separated by a single space.

Example:

```
>writemem 5 ABC

>printmem 5  3

41 42 43
```

Notice that the values 41, 42, and 43 are the hexadecimal representations of the ASCII values of the characters 'A', 'B', and 'C'.

**quit** – This quits your program.

## Requirements about allocation and freeing of memory

When a block is requested which is smaller than any existing block in the heap, then your code must perform splitting to create a new block of the appropriate size.

When a block is freed, it must be coalesced with the next block if the next block is free. When a block is freed, it DOES NOT need to be coalesced with the preceding block.

When searching for a block to allocate, be sure to use the best-fit allocation strategy.

## Developing Testcases

Testing is an important part of the programming process, so it is expected that you will develop your own test cases. It is up to you to consider all of the possibilities that can occur, within the limits of the specification, and make your test cases based on that. We will not grade your test cases, so you should not submit them as part of the assignment. However, you will need to test your code before submitting it in order to be satisfied that it meets the specification. If you have a question about how the program is expected to perform, come to office hours and ask the professor, or go to the lab and ask a TA.

## Submission Instructions:

Your source code must be a single c file. Be sure that your program must

compile on openlab.ics.uci.edu using gcc version 4.8.5.

Submissions will be done through Gradescope. The first line of your submitted file should be a comment which includes the name and ID number of you and your partner (if you are working with a partner).