bmalloc: Dynamic Memory Allocation in C

In this lab, we will practice using pointers, memory management, and structures in the C programming language. And what better way to understand memory management, than to build your own `malloc` and `free`, let's call them `bmalloc` and `bfree`. (Go Raptors!)

`malloc` and `free` are used to dynamically allocate memory in our C programs. As explained in lecture, a memory footprint of a process can be broken into a variety of segments (e.g. code, data, bss, heap, and stack). The heap is managed dynamically by keeping track of unallocated memory areas using something called a *free list*. Most of this lab will be managing this free list. **Of course, you should not use the built in `malloc` and `free` functions.**

Steps

1. Read and understand the `malloc` manual page (`man malloc`) and Ch. 7 of Kerrisk.
2. Understand the simple binary tree driver program included.
   ```
   $for i in {1..100}; do echo $RANDOM; done | ./simplebtree
   ```
3. Write bmemetest.c to test `bmalloc` and `bfree`. (for example, a linked list program)
4. Understand the block structure used to maintain the free list:
   a. the size of the block
   b. whether the block is free or being used
   c. the pointer to the next block in the list
   d. the pointer to the previous block in the list
5. Create a preliminary `bmalloc` that can allocate one block.
6. Use the utility function `print_freelist` to display the current state of the free list.
7. Modify `bmalloc()` so that it can allocate multiple blocks.
8. Write `bfree`.
9. Make `bfree` merge adjacent free regions.

Testing

1. Try running the simplebtree with your implementation of `bmalloc` and `bfree`.
2. Write your own test program to assure your implementations are correct. How would you measure the performance of your functions?

Deliverables

bmem.c	your implementation of the two functions: `bmalloc()` & `bfree()`
bmemtest.c	a test for your implementation (add this to the Makefile)

Extra

- Is the `free` in the `block_t` necessary? How could you avoid that extra book-keeping?
- Implement `bcalloc()` and `brealloc()`
- Use the `brk()` and `sbrk()` system calls rather than a large `char` buffer.

bmem.h

```
/***
 * Keith O'Hara <kohara@bard.edu>
 * Sep 2013
 * CMSC328: Embedded Operating Systems
 *
 * bmalloc: Dynamic Memory Management
 */

#ifndef _BMEM_H
#define _BMEM_H

#include <stdlib.h>

#define MAX_HEAP_SIZE (2 << 24)

void* bmalloc(size_t size);
void bfree(void* ptr);

#endif /* _BMEM_H */
```

bmem.c (the start of it anyway)

```
#include "bmem.h"
#include <stdio.h>

#define align4(x) (((((x) - 1) >> 2) << 2) + 4)[1]

typedef struct block {
    size_t size;
    struct block* next;
    struct block* prev;
    int free;
}block_t;

static char heap[MAX_HEAP_SIZE];
static block_t* freelist = 0;
```

[1] This macro increases the the size X so that it is divisible by 4.

The Free List – An Example

	address	**type of contents**	**data**
char * buffer -->	4004	size	60
	4008	next	4064
	4012	prev	0
	4016	free	0
	4020	data	
	4024		
	...		
	4064	size	140
		next	4204
		prev	4004
		free	0
		data	xxxx
	...		xxxx
	4204	size	1048376
		next	0
		prev	4064
		free	1
	…		xxxx

```
bmalloc(sizeof(char) * 44)
bmalloc(sizeof(int) * 31)
```