# Programming Assignment 5

Due Thursday, November 5 before midnight

## NAME

Dynamic memory allocation routines:

```
#include <sys/types.h>
void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t num_of_elts, size_t elt_size);
void *realloc(void *pointer, size_t size);
```

## USAGE

Link with any C/C++ program.

## DESCRIPTION

This is an individual assignment. You must write this in C (use the `gcc` compiler), not in C++.

The goal of this assignment is to replace the memory allocation routines that are provided by the C/C++ standard library. The `malloc` function is called by programmers to request a contiguous block of memory. The `free` function frees a block of memory previously allocated, so it can be reused. The `calloc` and `realloc` functions are minor variants of `malloc`.

Read the man page for these functions for more details. In particular, `size_t` is defined in the include file `<sys/types.h>` and you can treat it like any other integer type.

You are to create a file which contains code implementing the C library functions `malloc`, `free`, `realloc`, `calloc`. This is not a complete program, so you'll need to write test programs as well.

`malloc` should maintain a list of free memory blocks and fill incoming requests for memory from the list. When searching the free list for a block of sufficient size, use the first-fit method. If no large enough block is found, create a new free block by calling `sbrk()`. The new free block should be a multiple of the system page size, and large enough to fill the request. If `sbrk()` fails, set `errno` to `ENOMEM` and return `NULL`, as described in the `malloc` man page.

If the requested block is smaller than the found free block, you'll need to split the free block into two pieces: one that stays on the free list, and one that `malloc` can return.

`malloc` should always return values that are divisible by 8 (long word aligned).

The `free` function frees memory allocated by `malloc`. It should simply add the block back onto the free list. If `free` is called with a `NULL` pointer, it returns. If `free` is called with a pointer not allocated by `malloc`, the results are undefined. It would be nice if `free` combined adjacent free blocks, but that is not required.

## DELIVERABLE

Your github repo should contain a file `malloc.c` which contains definitions for `malloc`, `free`, `realloc`, and `calloc` but has no `main()` and with all output (cout, printf) removed.

## HINTS

Start by writing (or gathering) programs that call `malloc`, `free`, etc. Make sure you understand what they do, and test them with the built in malloc, etc.

Call your functions `mymalloc`, `myfree`, `myrealloc`, and `mycalloc` until you're really, really sure everything works. Once you rename them `malloc`, `free`, etc., they get called by constructors, by stream I/O, and in all sorts of places that will cause programs to fail utterly if the memory functions aren't working. This is the reason you are required to use C rather than C++ for this assignment.

Write `mymalloc` first and *test it thoroughly* before attempting the other three functions. Test these functions by writing a separate program which has a `main()` and makes calls to `mymalloc`, `myfree`, etc. In your test program, youll need to declare `mymalloc`, `myfree`, ... as `extern` functions.

To write `malloc`, use a global variable to keep track of the first block on the linked list of free blocks. At the beginning of each block of memory, leave extra space for a structure that contains the size of the block and a pointer to the next block in linked list. (Use a doubly linked list if you prefer.)

Use `sysconf(_SC_PAGESIZE)` to find the systems page size when you need to request new memory from the system.

Since blocks need to be long word aligned, youll need to be able to round a number up to the nearest multiple of 8. This is surprisingly tricky. Write a function to do this and test it separately.

Once youre really sure `malloc` is working, write `free`. Test some more, and then finally write `realloc` and `calloc`. The `calloc` function should call `malloc` and then call `memset` to clear the newly allocated block. The `realloc` function should call `malloc` to make a new block, `memcpy` to move the old information into the new block, and then `free` to release the old block. Beware that in `realloc`, the block can be resized larger or smaller, and the number of bytes you copy is the smaller of the old and new sizes.

When you're sure everything is working, rename your functions `malloc`, `free`, `realloc`, `calloc`. If you now compile any program along with your `malloc.c`, they will get called instead of the standard C/C++ memory allocators.

I use the programs `churn.c` and `churn2.c` in the `os/demo_mem` directory to

heavily test your memory allocators. Feel free to test your functions with these programs as well.

## USEFUL MAN PAGES

`malloc`(3)

`realloc`(3)

`calloc`(3)

`free`(3)

`sbrk`(3)

`sysconf`(3)

`memset`(3)

`memcpy`(3)