HW7: Memory Allocation

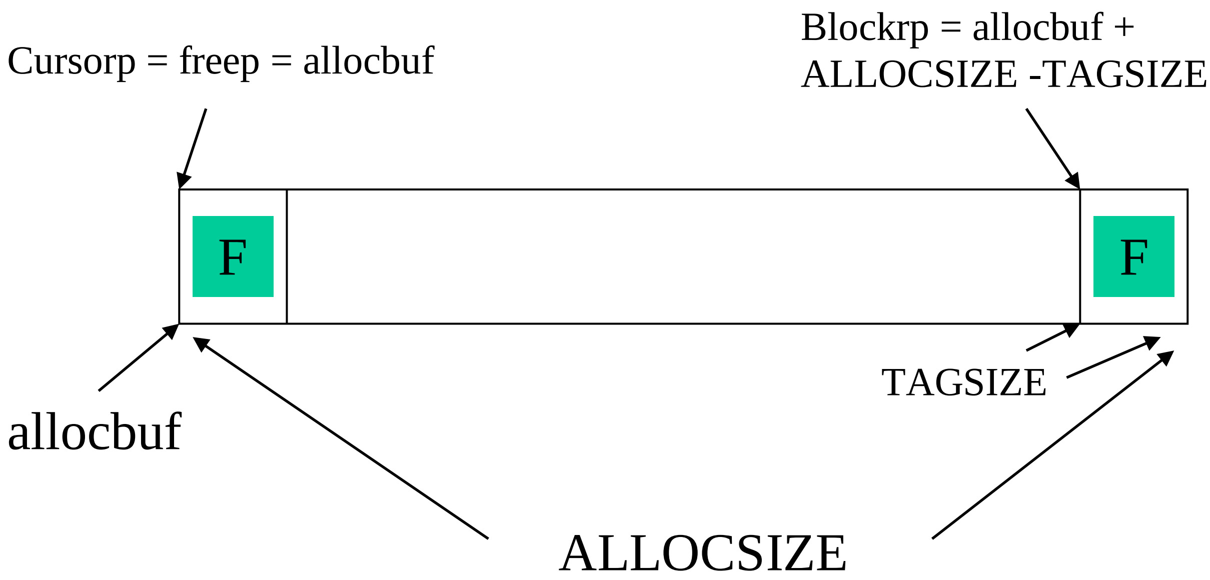
Assigned: 17 July 2018 **Due:** 26 July 2018

In C programming, we use malloc() and free() to allocate and free blocks of memory while our program is running. This assignment shows how the malloc and free library functions could be implemented. The actual implementations are more complicated than what we have set up here, but like the functions in this assignment, they are just C-coded functions. The system data structures can be damaged if a calling program writes beyond the boundaries of a block given out by malloc (or alloc) -- leading to quite mysterious bugs.

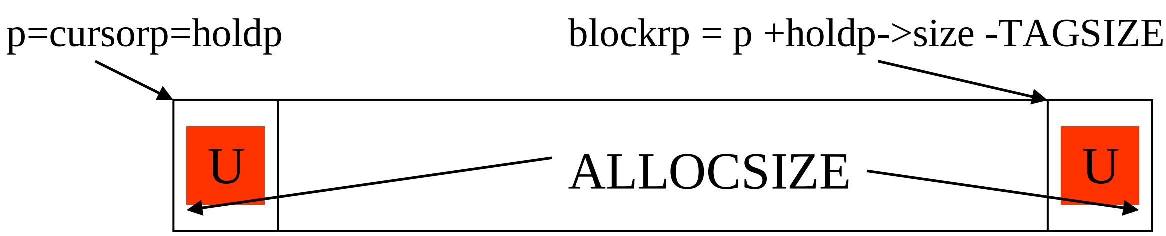
In this assignment, you are to write part of a dynamic storage allocation package. The package provides three function calls: void initalloc(), to initialize the data structures involved; char * alloc(int n), which returns a pointer to a block of n chars when called; and void freef(char * p), which frees the block of n chars earlier given to the data structure so that it can be given out to another. A package somewhat like this is covered in Section 8.7 of Kernighan and Ritchie -- however, be very clear that there are important differences between the two packages. The most important differences are that we do NOT try to allocate new space if we run out, we keep ALL our space to allocate in a single array, and we do not keep our free blocks in order, so we find another way to coalesce blocks when they are being freed (one which is faster than a linear search of the free list).

The algorithm you will be working on takes storage blocks from an array of ALLOCSIZE characters, and returns them to requesting callers. The main feature of this package is that these blocks can then be freed in any order and the small blocks freed will be merged back into longer blocks in the array structure. In order to perform this merge efficiently, a rather complex structure must be placed on the individual blocks passed out to callers. In particular, this means that if a caller wants to alloc(n), we must look for a block of n+k bytes, where the k bytes will contain the overhead.

In this program, the structure of a free block is as follows:



The structure of a used block is as follows:



In separate files in this directory are alloctest.c, alloc.c, and alloc.h. The file alloctest.c is a main() program to drive and test your alloc() and freef() functions. All of the alloc(), initalloc(), and some needed helper functions enchain() and unchain() to place blocks on the free chain of blocks are already provided. NOTE: The functions in alloc.c that can be called from the main program which in a separate source file are not declared with the keyword “static”. Other functions that are only accessed by functions in the same file are declared with the keyword “static”. This means that they are not part of the API to alloc.c and can not be called directly from the main program.

Your job is to write the freef() function. Note that each block which is handed out has information at the left (struct blockl) and at the right (struct blockr), to aid the freef() function in coalescing freed blocks.

In particular, both ends have an 8-bit pattern to let us know if the block is free or used. Then if it is free, the length is immediately available, in particular so one can get back to the left end of the block from the right end. At the left end of the block is the pointer to the next and previous blocks on the freechain (in freef(), we will have to remove adjacent blocks from the chain to coalesce with the block being freed).

alloctest.c provides an interactive test facility. For example, the commands:

a 200

a 100

would call alloc(200) and then alloc(100) and put the returned pointers in holdp[0] and holdp[1].

Then:

f 0

would free the 200-byte block of the 0th alloc (pointer in holdp[0]).

The "d" command dumps the free list, following the nextp pointers starting from freep. Thus it does not warn you about problems in the prevp pointers. You can rewrite it to make it follow these as well if you want more debugging info.

alloctest can also be driven by file input. A file "newtest.in" is provided as an example and for your final run. Use LINUX redirection to make alloctest read newtest.in. When you have correctly written the code for freef, you should get results as shown in the file "example_script" when you run the LINUX command:

./alloctest <newtest.in

Leave alloc.c, the executable alloctest, and a typescript showing all usual information for a run of "./alloctest <newtest.in" in your directory for grading. Other files that should be there are alloc.h, alloctest.c, and newtest.in, but these should not be edited from the provided copies.