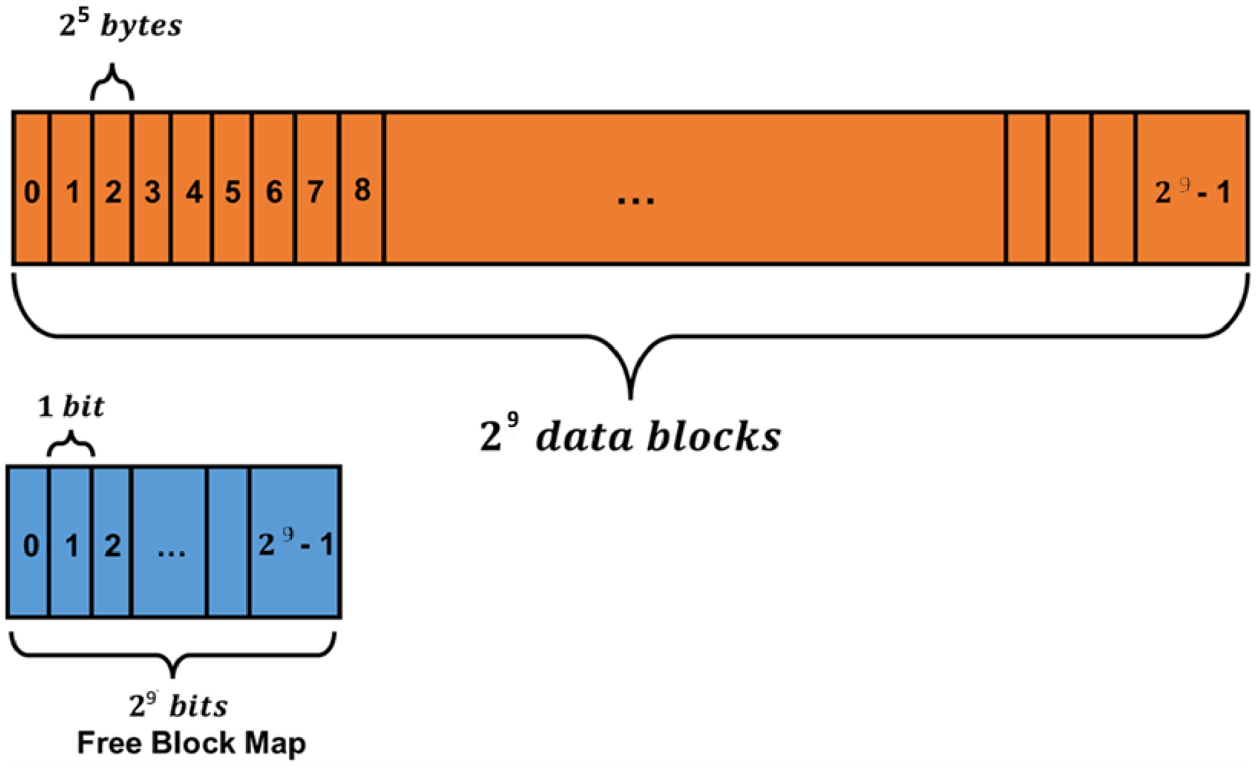
Assignment 4 - Block Store

Objectives:

Become more familiar with the following concepts:
- **Bitmaps, Bytes, and Blocks**
- **Block storage devices**
- **Free block maps**
- **Checking system errors with errno**

You are to implement a storage library capable of storing blocks of data for the user. This data is to be file-backed, i.e. the stored data is written to disk so that it may be used later. Your implementation should create opaque, self-contained objects.

The below diagrams shows an example block store with 512 32-byte blocks. Your assignment is to develop a block store with 2048 64-byte blocks.



This exercise touches on many concepts you may be unfamiliar with, so be sure to read this document completely. Questions and requests for clarification should be directed at the course discussion board within the Canvas LMS.

Modular Programming:

1. Modular Programming and proper organization
2. Documentation of code via thorough comments
3. Parameter testing and error checking

Bitmaps:

A bitmap is a data structure that can represent a large set of boolean data in a compact way. A boolean value can be represented with a single bit, but the smallest addressable unit is an entire byte, which leaves a lot of wasted space. If you need to represent multiple boolean values, you're going to waste even more memory, and this can add up quickly. This leaves less memory for other programs, which is a terrible thing for an OS or service to do.

A bitmap uses binary operations to store multiple boolean variables in a single byte, reducing your memory footprint to ⅛ of what it would be otherwise. In doing this, however, a bitmap must use a more complex addressing system that uses both a bit and byte address in addition to masking.

Please review your bits learning module and the included bitmap
Resource Link: https://en.wikipedia.org/wiki/Bit_array

Block Storage Devices:

A block storage device is exactly what it sounds like: a device that stores blocks of data. Most storage devices (hard disk, optical disc, and flash drives) are block devices with varying block sizes (the number of bytes per block). Reading and writing data in blocks reduces overhead and allows for better data handling. Each block is referenced by an id number starting from 0 to N - 1 number of blocks, and these blocks are contiguous, essentially making a block device a giant physical array.

Please review your array learning module.
Resource Link: https://en.wikipedia.org/wiki/Block_(data_storage)

Block Maps:

A block map is how block storage devices keep track of important metadata. With this block storage library, we will be using the Free Block Map (FBM), **implemented as a bitmap**, to keep track of important device data.

The FBM keeps track of which blocks are currently in use. A bit value of one means the block is in use, and zero means the block is free. When a user requests a block of space, the library scans the FBM, finds a free block, marks that block as in use in the FBM, and returns the block id number for the user to use. When a block is freed, the corresponding bit in the FBM is cleared. Without this, devices would have no way of tracking which blocks are in use, and current data present is easily overwritten.

You must store your FBM starting in block 1022 of the device and continuing to subsequent blocks as many as needed, but not more. The tests in test/tests.cpp are written with the assumption that your FBM is stored starting at this particular block. The choice of this block is well-motivated as many physical devices are more performant in the middle physical blocks, and it is important that FBM access is as fast as possible.

Resource Link: https://en.wikipedia.org/wiki/Block_allocation_map

Assignment Requirements:

1) You must update and complete the given CMakeLists.txt to build a shared object (a dynamic library) called libblock_store.so that uses src/block_store.c and include/block_store.h for source files.
2) Complete all the functions as described in the header to build a library called **libblock_store.so**.
3) You must check all function calls for error conditions. Occasionally, this will mean you will have to check errno, declared in <errno.h>. Please refer to Google for information.
4) You must ensure that your solution is free of memory leaks. You can check your program for leaks with valgrind.

Rubric:

This assignment is out of 100 points. The included tests.cpp file provides points for a variety of tests.

All file operations should be using POSIX interface, you will lose points for using FILE objects. If your code does not compile, you will receive a zero for the assignment.

Insufficient Parameter Validation -20% of rubric score
Insufficient Error Checking -20% of rubric score
Insufficient Block and Inline Comments -20% of rubric score
Submission compiles with warnings (with -Wall -Wextra -Wshadow) -70%
Submission does not compile -100% of rubric score