Hayley Yukihiro

CSIS215 – D01

December 7, 2020

**Programming Assignment 3: LRU Bufferpool**

**Approach**

To start off the approach document, I will outline the direction of the approach to the LRU Bufferpool programming assignment. I reviewed the buffer pool section of the textbook to refresh my understanding to outline my approach. I will be implementing the two ADT files for the buffer block and buffer pool in this order:

Outline Approach

* Create BufferBlock.h to implement BufferBlockADT.h
  + Constructor
  + Destructor
  + getData()
  + setID()
  + getID()
  + getBlocksize()
  + getBlock()
  + setBlock()
* Create LRUBufferPool.h to implement BufferPoolADT.h
  + Constructor
  + Destructor
  + getBytes()
  + printBufferBlockOrder()
  + getLRUBlockID()
* Test main.cpp

The preliminary steps were completed to begin the assignment by compiling all the required .cpp and .h files into a folder that is named per instructions.

**BufferBlock.h**

To begin BufferBlock.h, I created a new header file and inherited the BufferBlockADT.h. I reviewed section 8.3 of the book to understand the expected functionality of a buffer block which to is the block is a size of a buffer since the pool is divided into blocks of a given size. The given size for each block is 4096 bytes as specified in the instructions.

**Constructor:**

* There were two constructors specified in the ADT file so I implemented the default constructor and a constructor with initial conditions
* The second constructor initializes all the blocks by creating a new block with the size of 4096 bytes.
* I added a size variable to the instance variables in the private section to have a variable for getBlocksize() later on. In the constructor, sz will equal size for initialization.
* Since the function getData specifies that the end of the block is determined at sz-1, the for loop initializes the data in each block till it reaches the end of the stack which is sz-1.

**Destructor:**

* Data in the blocks are deleted

**getData(): read the block from pos to pos + sz-1 (or to the end of the block)**

* From the ADT, the getData has the parameters pos for position in the block, sz for size of the block, and pointer to data. To begin the implementation, I created an int block placeholder for the for loop.
* The for loop reads the block from the beginning to the end of the block

**setID(): sets the block ID**

* Assigns the block id to the id variable in the parameter

**getID(): returns the block ID**

* Returns the block ID by referring to thelockID instance variable

**getBlocksize(): returns the size of the block**

* Returns the size of the block by referring to the size instance variable

**getBlock(): returns a block**

* Returns a block by referring to the block instance variable

**setBlock(): sets a block**

* Assigns a block to the blk pointer in the parameter

**LRUBufferPool.h**

To begin LRUBufferPool, I created a header file and inherited BufferBlockADT.h. I also added the constants.h file to the header file because I will be referring to the pool size and block size global variables from constants.h.

**Constructor:**

* Utilized a default constructor and a constructor with initial conditions
* The second constructor gets the file name for the buffer and opens it. Then it loops through the file and read in the text data to a temp object. From the read in, sets the ID and block based on the information from the temp object.
* The file was read in using the example from the assignment instructions

**Destructor:**

* Utilized a default destructor

**getBytes: Copy “sz” bytes from position “pos” of the buffered storage to “space”**

* Required to read data in buffer, LRU read from disk, and last read block to the front via the assignment instructions
* I created two int variables which hold the id from position divided by the block size and the next data assignment which is the modulus of position and block size.
* I declared a block object using bufferblock
* I created two Boolean instance variables after debugging to determine file data read in which I had trouble with later on.
* The first portion of the getBytes implementation is to find the data from the pool and set the block data via searching by the id. The for loop will keep looping through the pool and set blocks if the data is found.
* Initially, I did not create a situation where the data was not found which resulted in incomplete output results. I created an if statement outside of the for loop since the data not included in the loop was not found. First the data is read from the file again which repeats the same process from the constructor. After reading the data in from the file, the block and id are set.
* After successfully setting the data, the LRU read from the disk process begins. Three cases were written: front, middle/end, new. For each case, the block in the pool is rearranged by where the data was found.

**printBufferBlockOrder(): Print the order of the buffer blocks**

* Loops through the pool and prints the buffer blocks by grabbing their ID from the pool

**getLRUBlockID(): Get the block id number of the least recently used buffer block**

* Return the block id number from the pool

**Test main.cpp**

After I completed BufferBlock.h and LRUBufferPool.h, I began the debugging process by using main.cpp and the mydatafile.txt.

* I had the Buffer Pool Output file to compare to which my initial test compilation did not return the same results.
* Only the first block was filed in but the buffer block order was not arranged properly. I realized the getBytes implementation was done wrong because I did not consider utilizing Boolean instance variables in finding data.
* Therefore, the data was not being read in correctly. After I created two implementations of data being found and data not being found, I was able to receive a similar output.
* Simple syntax errors were fixed during debugging as well.
* I also ran into a inconsistency error with the example file and the actual output file with reading in data strings for each block. I was getting different text increments. Somehow, by incrementing i++ for every for loop in the constructor, the data is mixed up. So I incremented i++ before the seekg() implementation to which I was able to receive the correct output results.