Overview
In this assignment you will implement a C++ program that simulates a paging system using the aging algorithm. You will read a sequence of reference requests from a file and load and evict pages as necessary. For this assignment, you may work individually or in a group with one other student.

Virtual memory and paging are explained in Section 3.3 of your textbook. Professor Mike Goss's video on virtual memory also covers these topics. The aging algorithm is described in Section 3.4.7 of your textbook. The LRU Page Replacement Algorithm video also explains the aging algorithm.

Setup
In CodeLite on your virtual machine:

- Create a new C++ project named Project8.
- Add three classes to the project, one named PageTableEntry, one named Counter, and one named PagingSimulator. You may optionally add a class named Process. (See below for details.)

Submitting Your Assignment
Submit the source code (.cpp and .h files) from your project in a zip file. Name the zip file YourLastNameYourFirstNameProject8.zip. Upload the zip file to Canvas before the due date. Remember that late assignments are not accepted in this course. See the syllabus for details.

Important submission notes for students working in a group:
If you worked with another student, name the zip file using the last names of both students: FirstPersonsLastNameSecondPersonsLastNameProject8.zip. One person must upload the zip files to Canvas. The other person must upload a single document stating who their team member was.

Command Line
Your program must accept one argument on the command line, the input file name. An error message must be printed if the number of command line arguments is incorrect.

Input File Format
All fields on a line are separated by whitespace.

The first line of the input file contains three integers: the number of virtual pages, the number of physical page frames, and the size of a page. In the example file below, there are 16 virtual pages, 4 physical page frames, and the size of the page is 32 bytes.

The second line is the number of processes that will be running for the simulation. In the sample input file, one process will be running.

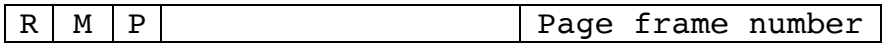
The remaining lines contain a virtual memory reference you need to simulate. The first number is the number of the process requesting the reference. The second field, either "r" or "w", indicates if the reference is a read or a write. The last number is the virtual memory location being requested.

Sample Input File

```
16 4 32
1
0 r 20
0 w 40
0 r 232
0 w 64
0 w 100
0 w 68
0 r 228
0 r 32
0 r 20
0 w 100
```

Assignment

After reading in the first two lines of the file, your simulation should set up page tables and counters for each process. Since, in the operating system, these page tables need to be partially managed by the hardware, each page table entry must be stored in a uint32_t and must have the following format:



In other words, each page table entry is a 32-bit integer where the most significant bit (left-most bit), R, is the referenced bit. The next bit, M, is the modified bit. P is the present/absent bit. The appropriate number of least significant bits are used to store the frame number. For this simulation we will not use protection or caching disabled bits. Note that in the Virtual Memory video, the page frame number is stored in the most significant bits. In this simulation, we will store it in the least significant bits. (See pages 199 - 201 of your textbook for more information about page table entries.)

Each process will need its own page table with one page table entry for each virtual page. When a simulated reference such as 0 w 40 occurs, your program will need to figure out which virtual page is being referenced. It will then check the page table to see if that page is already loaded into a page frame. If the page is loaded, your code must calculate the physical address needed to access virtual address 40. The modified bit must be set when a memory address on the page is modified (write).

If the page is not loaded into a page frame, a page fault occurs. Your code must use the aging algorithm to determine which page will be evicted. We will use the global version of the aging algorithm described in Section 3.4.7. The global version examines all pages of all processes to determine which page to evict. (Section 3.5.1 explains the difference between global and local allocation policies.) To implement this algorithm, each process will have its own counters, one for each page table entry in the page table. The counters will be 8-bit unsigned integers. We will simulate the clock (timer) interrupt by aging the counters every two instructions. When a modified page is evicted, you must indicate that the page is being written back to disk.

When the simulation starts, all page frames will be empty. Thus, you will have several page faults at the beginning of the simulation.

Implementation notes

PageTableEntry Class

- You must represent a page table entry using an unsigned 32-bit integer. This will allow us to simulate what goes on in the hardware to some extent.
- The PageTableEntry class constructor must take one argument: the number of bits needed for the page frame number. For example, if you have four page frames, the number of bits needed for the page frame number is 2.
- Remember that all member variables must be private.
- You may add additional member variables and member functions as needed.

Counter Class

- You must represent a counter using an unsigned 8-bit integer.
- The Counter class constructor must be a no-argument constructor.
- The initial value of the counter must be 0.
- Remember that all member variables must be private.
- You may add additional member variables and member functions as needed.
- Hint: Even though your counter must be represented as an unsigned 8-bit integer, you can use the bitset class (https://en.cppreference.com/w/cpp/utility/bitset) to output the value in binary.

PagingSimulator Class

- The PagingSimulator class constructor must take one argument, the name of the input file (a std::string).
- The class must have one public member function named **run**. This function must have no parameters and must have a void return type.
- You should use private member functions to decompose your code into manageable pieces.
- It is your choice as to which data structures you use to implement the paging simulation.

Process Class (optional)

- You may find it useful to implement a Process class. In this simulation, each process will have a page table and a set of counters.
- If you choose to implement this class (I did!) I suggest that the constructor have two parameters, both of type uint32_t. One of the parameters will be the number of pages and the other will be the number of page frames.
- The constructor can use this information to create the page table (which contains page table entries) and the set of counters used for aging.

Since the PagingSimulator class needs to access the page tables and counters, you may make the PagingSimulator class a friend of the Process class. See http://www.cplusplus.com/doc/tutorial/inheritance/ for more information on friend classes.

Your main method should:

- Check to see if the correct number of command line arguments is given.
- Create an object of type PagingSimulator.
- Call the PagingSimulator's run method to start the simulation.

Output Format

All output must be written to standard output (the console window). Since the output for this assignment is lengthy, I have included sample input and output files with this assignment. Your output does not have to look exactly like mine. However, it should include all information that you see in the sample output files.

Grading Criteria (35 points possible)

Note: Your program must compile to receive credit for this assignment!

Points	Criteria
0-2 points	**Program Structure:** Does the program contain the classes described above? Are function prototypes and member variables declared in the header (.h) file? Are all member variables private? Are all member functions implemented in the .cpp file of the corresponding class?
0-4 points	**PageTableEntry Class:** Does the PageTableEntry class constructor have one parameter as described above? Are table entries stored as 32-bit unsigned integers using the format described in the assignment? Are member functions to access and update page table entries implemented correctly?
0-4 points	**Counter Class:** Does the Counter class have a no-argument constructor? Are counter values stored in 8-bit unsigned integers? Are member functions to access and update counters implemented correctly?
0-2 points	**PagingSimulator Class:** Does the PagingSimulator class constructor have one parameter as described above? Does it contain one public member function named "run"? Are private member functions used to divide up the code into logical segments?
0-15 point	**PagingSimulator Correctness:** Does the simulation work correctly? Are all values calculated correctly? Are pages loaded into the correct page frame? Does the aging algorithm evict the correct page? Does the simulation correctly calculate page numbers and physical addresses? Is a message indicating that the page is begin written back to disk printed to the console when a modified page is evicted?
0-4 points	**PagingSimulator Output:** Are all output values correct? Is all information shown in the sample output file included in the output? Is the output formatted in a way that is easy to read?
0-2 points	**Main Function:** Is the main function implemented as described in the assignment? Does it accept the one command line argument described in the assignment? Is a "usage" error message printed if the number of command lines arguments is not correct? Does it create an instance of the PagingSimulator and then call the run method?
0-2 points	**Style:** Is the code easy to read? Does it follow class style guidelines (See Program Style page in Canvas.)