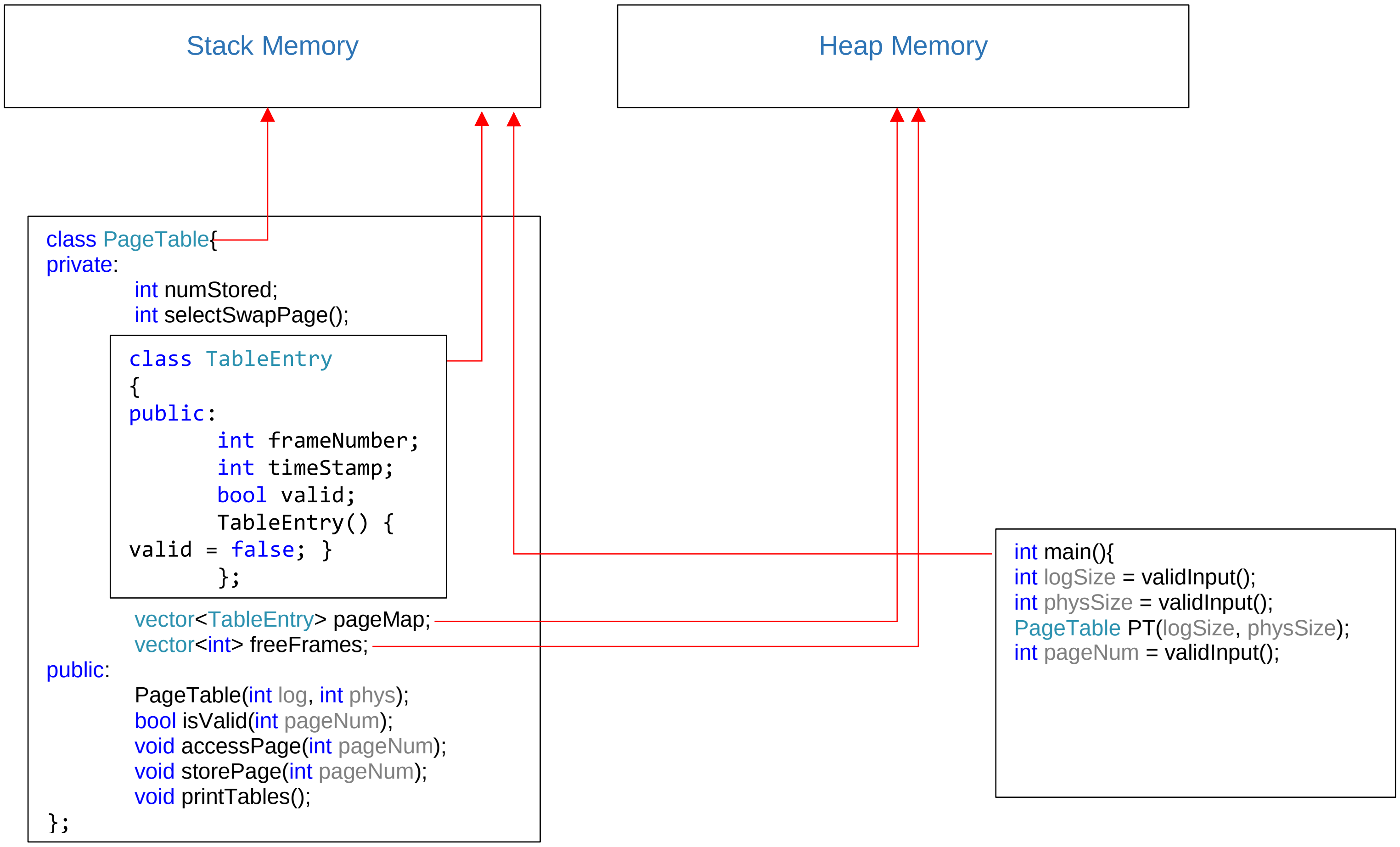
Paging Simulation Project

Nathan Foster

1) What questions you are trying to answer with this project?
   a) How can paging be implemented into a C++ program?
      i) What data types to use?
      ii) What data structures to use?
   b) What are efficient methods of implementation?
      i) How will you find a page fault?
      ii) How will you swap pages when there is no more room in the physical memory?
      iii) How will you track the LFU (Least Frequently Used) frames in physical memory?
2) What is your approach to find the answers?
   a) I first tried to create three separate arrays that would represent the logical memory, page table, and physical memory. The problem with this initial approach was that arrays are not dynamic and are fixed in size; they are contiguously allocated in stack memory.
   b) In order for me to simulate paging I needed to allocate and access memory dynamically in the heap. Therefore I utilized vectors. I then was able to add, resize, delete, the vectors and know if they were empty.
   c) I then realized that declaring 3 separate vectors was still not accurate because Frank told us that the page table was a data structure. So then I decided to place the vectors inside a page table data structure.
   d) I also struggled with the most efficient way to use time stamps for each page in the page table. I needed the page table to hold multiple data types.
   e) I first tried inheritance (class TableEntry : public PageTable) but realized it was unnecessary because I didn't want to create another class that would be allocated outside my parent class (PageTable) in the stack.
   f) So my solution was to make the TableEntry class a private member of the PageTable class. Merging these two data structures allowed me to use two vectors rather than three because now the logical memory and the page table are represented by defining a vector<TableEntry> which is a vector of the data structure TableEntry. One of its members is an integer that acts as a time stamp.
   g) I first tried using a clock method, where the integer would store the amount of time in milliseconds. But I noticed that if you made the vector REALLY large it would DRAMATICALLY increase the runtime of operations. So I had to research more efficient ways to measure the LRU (Least Recently Used) frame in the physical memory. I found that the solution was easier than I anticipated. All you have to do is increment the time stamp by one each time a page is stored.
   h) This led me to find solutions for page faults and page swaps. (PageTable.cpp - storePage)
3) What are the conclusions?
   a) Using vector data types proved to behave more like an actual page table because of its dynamic memory access in the heap.
   b) Placing the vectors inside a class data structure bound the vectors inside a structure that was contiguously allocated in the stack but also allowed dynamic memory access in the heap.
   c) I believe that making the TableEntry class a private member of the PageTable class allowed my program to more accurately simulate paging because I have only of one data structure solution rather than having both a parent and child data structure.
   d) I believe that the time stamp method I researched and implemented is one of the most optimal ways to track LFU.
   e) My method for catching page faults and swapping pages utilizes the time stamp which accurately simulates an operating system using a page table data structure in virtual memory.



```
class PageTable{
private:
        int numStored;
        int selectSwapPage();

        class TableEntry
        {
        public:
                int frameNumber;
                int timeStamp;
                bool valid;
                TableEntry() {
        valid = false; }
                };

        vector<TableEntry> pageMap;
        vector<int> freeFrames;
public:
        PageTable(int log, int phys);
        bool isValid(int pageNum);
        void accessPage(int pageNum);
        void storePage(int pageNum);
        void printTables();
};
```

```
int main(){
int logSize = validInput();
int physSize = validInput();
PageTable PT(logSize, physSize);
int pageNum = validInput();
```

pageMap
pageNum
pageNum
pageNum
pageNum
pageNum
pageNum
pageNum
pageNum
pageNum
pageNum

```
pageMap.resize(logSize);
```

PageTable::
storePage(pageNum)

$\longrightarrow$

TableEntry	valid	timeStamp
		pageMap[pageNum].timeStamp = numStored

PageTable::
freeFrames.pop_back();

$\longrightarrow$

if (freeFrames.empty()){

selectSwapPage();

else{

frameNum =
freeFrames.back();

freeFrames.pop_back();

freeFrames
frame
frame
frame
frame
frame
frame
frame
frame
frame
frame

```
for (int i = physSize - 1; i >= 0; i--)
        freeFrames.push_back(i);
```