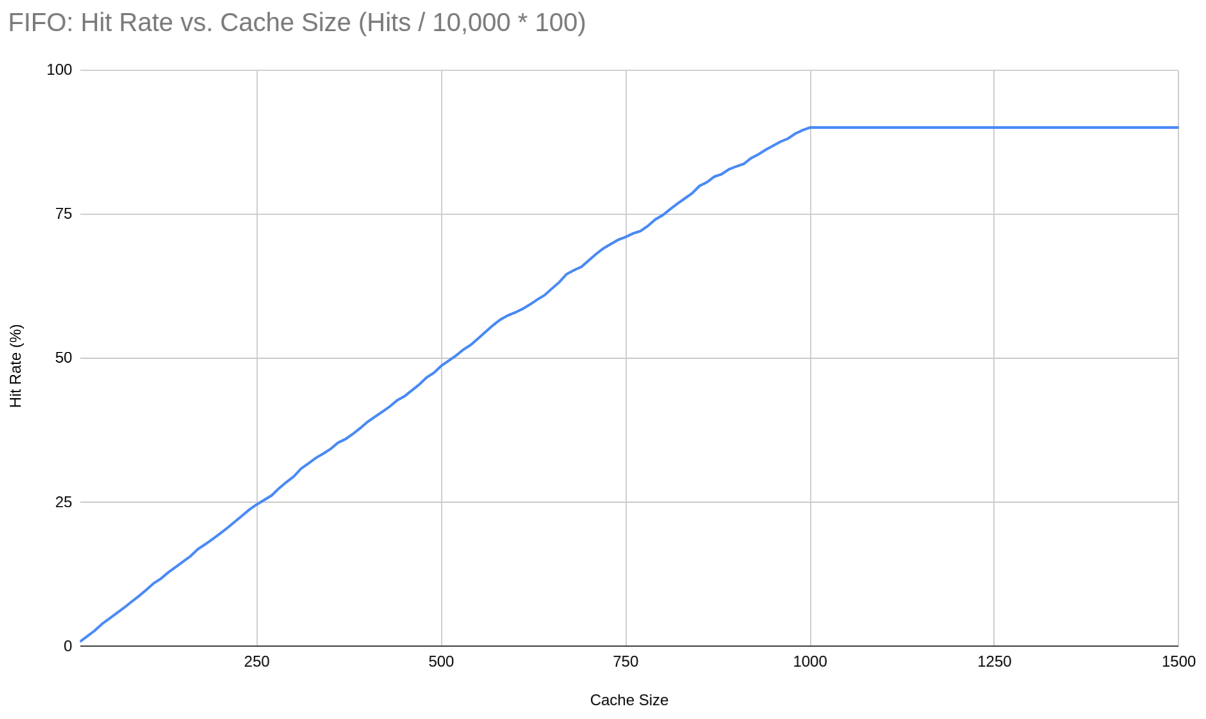
Implementation

1. Check argument to make sure there are:
    a. Only 1 argument passed
    b. That argument is a positive integer
    c. If the above requirements are not satisfied, return 1
2. Use calloc to initialize cache
3. Use while loop on fgets to read into buff variable stopping when we hit a NULL line
4. Store requested page number in request variable
5. Set hit boolean to false
6. Iterate through cache checking each slot for a match to request
    a. If we find a match, set hit boolean to true and break loop
7. After looping through cace, if hit bool is still false we must perform cache shift and add new page number
    a. Start from cache[cacheSize - 2] and move this element up one position to cache[cacheSize -1] which will overwrite the last element of the array
    b. Stop after we have moved element 0 to element 1 position
    c. Add new request page number to element 0
    d. Finally iterate fault counter
8. While loop continues steps 4-7 on each page number passed via stdin (piped from cat addresses.txt)
9. free(cache) to release allocated memory
10. Print values for analysis
    a. Cache size
    b. Number of page faults

Hit Rate



1. Using bash script, executed `cat addresses.txt | ./FIFO $SIZE` beginning with cache size of 10 and incrementing by 10 for each iteration until we reach 1500 cache size
2. As we can see from the graph, hit rate increases pretty linearly with cache size, until cache hits 1000.
3. After cache size is 1000, we are at a 100% hit rate because page numbers in addresses.txt only go up to 1000.
4. Our hit rate flat lines at 90%. The 10% loss is due to the 1000 misses we incur the first time we have to inset each page number.