The purpose of this project is to simulate four memory management algorithms (FIFO, LRU, LFU, & Optimal) and to compare their results for a fixed order of page accesses and number of frames. Here is the general layout of our code:

Main():
I. Variable Declarations
II. Prompt user for input string length (SIZE) and number of frames (PAGESIZE)
III. Prompt user for probabilities
IV. Prompt user for number of available pages
V. Create input string
VI. Set all frames to ‘9999’ or empty
VII. FIFO - Cycle through input string
   a. Cycle down frame list
      i. If not first page access
         1. Copy current state into next frame state
      ii. If empty frame
         1. Page fault
         2. Copy in current page access
         3. Exit Inner Loop
      iii. Else if not a page hit
         1. Move next frame to top
         2. Page fault
      iv. Else
         1. Exit loop
VIII. Output FIFO page faults
IX. Set all frames back to ‘9999’
X. LRU – Cycle through input string
   a. Cycle down frame list
      i. If not first page access
         1. Copy current state into next frame state
      ii. If empty frame
         1. Cycle down frame list
            a. If frame match
               i. Exit loop
               ii. Exit next loop
               iii. Set Match flag
         2. If Match flag not set
            a. Page Fault
            b. Copy in current page access
c. Exit loop
   iii. Else if not a page hit
       1. Move next frame to top
       2. Page fault
   iv. Else
       1. Put matched frame at bottom
       2. Exit loop

XI. Output LRU page faults
XII. Set all frames back to ‘9999’
XIII. Optimal – Cycle through input string
   a. Cycle down frame list
      i. Set all indices to SIZE + 1
   b. If not first page access
      i. Copy current state into next frame state
   c. Cycle down frame list
      i. If page hit
         1. Exit loop
         2. Set Match flag
      ii. If empty frame
         1. Page fault
         2. Copy in current page
         3. Set Match flag
         4. Exit loop
      iii. Else
         1. Cycle down frame list
         2. If page hit
            a. Copy index into frame buffer
            b. Set index to size
   d. If not Match flag
      i. Page fault
      ii. Cycle down frame list
         1. If frame [index] matches SIZE + 1
            a. Set replace to index
            b. Exit loop
         2. else
            a. If temp less than frame[index]
               i. Set temp to frame[index]
               ii. Set replace to index
         iii. Set replace frame to current page
   e. Set Match flag to false
   f. Reset temp to zero

XIV. Output optimal page faults
XV. Set all frames back to ‘9999’
XVI. Cycle through freq array
   a. Set all cells to 1
XVII. LFU – Cycle through input string
a. Cycle down frame list
   i. If not first page access
      1. Copy previous state to current state
   ii. If empty frame
      1. Page fault
      2. Copy in current page
      3. Set LFU flag
   iii. Else if page hit
      1. Increment freq at current index
      2. Copy previous state to current state
      3. Set LFU flag
      4. Exit loop
   iv. Else
      1. If freqtemp is 0
         a. Set freqtemp to freq[index]
         b. Set freqtemp2 to index
      2. Else
         a. If freqtemp greater or equal to freq[index]
            i. Set freqtemp to freq[index]
            ii. Set freqtemp2 to index
         b. If LFUflag is false
            i. Page fault
            ii. Reset freq[freqtemp2] to 1
            iii. Put in new page to freqtemp2 index
            iv. Reset temp flags
      c. Else
         i. Reset temp flags
         ii. Reset LFUflag to false

XVIII. Output LFU page faults
XIX. Output simulation conditions

Wandom(int limit):
   I. Declare a = 0
   II. Set a to rand() mod limit, the number of possible pages
   III. Return a

Analysis:
There are four separate parts of this program. They do not interact with each other. Each independent memory management scheme processes the same randomly generated list of page accesses.

The FIFO scheme replaces the page that has occupied its frame the longest on each page fault. The LFU scheme replaces the page that has been accessed the fewest number of times since it was put into its frame. The LRU scheme replaces the least recently accessed page. The Optimum scheme looks into the future, replacing the page whose reoccurrence will be the farthest in the future.
After completing the project, we compared answers for the different simulations with other groups. Each simulation of LRU, FIFO, and Optimal showed the same results as the other groups’, but LFU consistently showed a few more page faults than other groups. When we examined the code, this inconsistency was caused by our implementation of the LFU algorithm. In our simulation, when a page fault occurs and two or more frames have been accessed with the same frequency, our code replaces the page occupying the highest indexed frame. All other groups that I compared to replaced the lowest indexed frame. Just by coincidence, this caused a few more page faults because of the order of the pages accessed.