

**CSE360: Computer Architecture**

**[Summer 2022]**

**Section: 01**

**Submitted to:**

**Md. Tanvir Alam**

**Lecturer,**

**Department of Computer Science & Engineering,**

**East West University**

**Submitted by:**

|  |  |
| --- | --- |
| **Student ID** | **Student Name** |
| **2019-1-60-031** | **Md. Shajibul Islam** |
| **2019-1-60-046** | **Md. Rakibul Haque Ontor** |
| **2020-1-60-152** | **Najnin Sultana Nishu** |
| **2020-1-60-123** | **Meherun Hossain Orpa** |
| **2020-1-60-125** | **Nabila Shaeed** |
| **2020-1-60-193** | **Raka Kamara** |
| **2020-1-60-102** | **Sanjida Akter** |

**Algorithms:  FIFO, LRU, LFU**

**Objectives:**

1. Create 3 Text file of 1000 random numbers (1-10). (Example: 1.txt, 2.txt, 3.txt)
2. Implement the function of FIFO (First in First out), LFU (Least Frequently used) and LRU (Least Recently used). (Block size in Cache = 3)
3. Run the previously Created text files for each of the algorithms and find out the number of hits and misses.
4. Find the average hit of each algorithm.
5. Generate a graph that hits X block in cache for the 3 algorithms.
6. Create a text file of 10000 random numbers (1-10)
7. Run the functions for first 1000,2000,3000…10000 numbers from the text file created in step 5 and find the number of hits.
8. Generate a graph of the number of inputs X hits for each of the algorithms.
9. Find out which algorithm is efficient on the basis of the graphs.

**System requirements:**

Operating System: Windows/Ubuntu/MacOS

Processor: Intel/AMD

Software: Code Blocks

Source code Link:

Drive: <https://drive.google.com/file/d/1o37H4fQRbC4wuZuSmE22JLeIucPPixEQ/view?usp=sharing>

Github:<https://github.com/ShajibEwuCse19/CSE360_LRU_LFU_FIFO_Implementation/tree/master>

IDE: Codeblocks

Language: C++ 17

**Explanation:**

**What is Page Replacement in Operating Systems?**

Operating systems that use virtual memory and Demand Paging require page replacement. Only a certain set of pages for a process are loaded into memory when using demand paging. This is done so that multiple processes can run concurrently in memory.

The operating system must determine which page will be replaced when a process requests the execution of a page that is present in virtual memory. This procedure, called page replacement, is a crucial part of managing virtual memory.

**Why Need Page Replacement Algorithms?**

We must first understand page faults in order to recognize why page replacement methods are necessary. Let's look at what a page fault is.

A page fault happens when a CPU-running application tries to access a page that is in its address space but is not currently loaded into the system's RAM. Page errors happen when the virtual memory is substantially larger than the available RAM. As a result, whenever a page fault happens, the operating system must replace an existing RAM page with the newly requested page. Page replacement algorithms enable the Operating System in determining which page to replace in this case. To reduce the amount of page defects is the main goal of all page replacement methods.

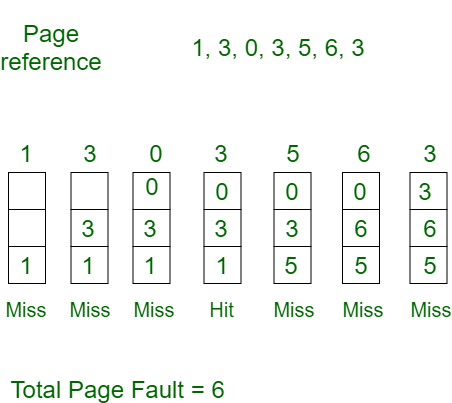
**Page Replacement Algorithms in Operating Systems**

1. **First In First Out (FIFO)**

FIFO algorithm is an algorithm relatively easy to achieve. The idea is first in first out (FIFO, queue), this is the simplest and fairest idea that if a data is the first to enter, then it can be considered in the future the possibility of being accessed is small. Space is full, the first to enter the oldest data will be replaced (eliminated) away.

Of all the page replacement techniques, the FIFO algorithm is the most straightforward. We keep a queue of all the pages that are now in the memory in this. The most recent page is at the back or rear-end of the line, and the oldest page in the memory is at the head of the queue.

The operating system checks the front of the queue whenever a page fault occurs to determine which page should be replaced by the newly requested page. Additionally, it pushes the oldest page to the back of the queue while adding the newly requested page to the front.  
**Example 1:** Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3 page frames.Find the number of page faults.



Initially, all slots are empty, so when 1, 3, 0 come they are allocated to the empty slots —> 3 Page Faults.

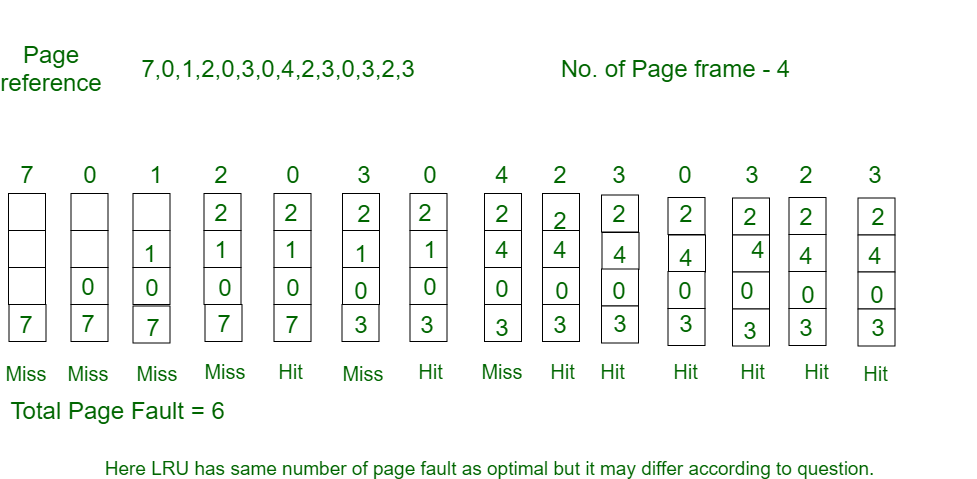
when 3 comes, it is already in memory so —> 0 Page Faults. Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>1 Page Fault. 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>1 Page Fault. Finally, when 3 comes it is not available so it replaces 0 1-page faults.

1. **Least Recently Used (LRU):**

A page replacement method for managing memory is called the Least Recently Used (LRU) algorithm. This approach replaces the page that has not been accessed in a while. Any page in memory that has been idle for a longer time than the others is therefore replaced.

In this algorithm, a page will be replaced which is least recently used.

**Example-2**: Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4-page frames. Find number of page faults.



Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> 4 Page faults

0 is already there so —> 0 Page fault. when 3 came it will take the place of 7 because it is least recently used —>1 Page fault

0 is already in memory so —> 0 Page fault.

4 will takes place of 1 —> 1 Page Fault

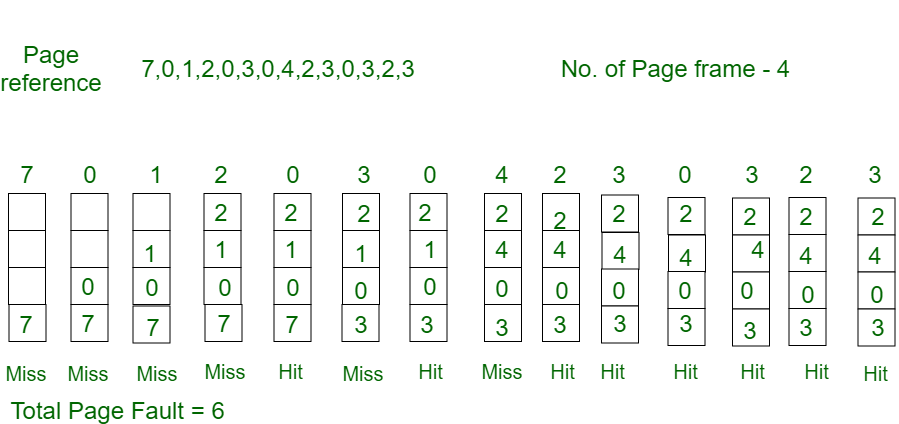
Now for the further page reference string —> 0 Page fault because they are already available in the memory.

**3.Least frequently used (LFU):**

The Least Frequently Used page replacement algorithm is referred to as LFU. The LFU page replacement method eliminates the page that has had the fewest visits in a specific amount of time. The pages that are not utilized too often are replaced. The first page gets replaced first if the frequency of pages is constant.

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

**Example-3:** Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4-page frame. Find number of page faults.



Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> 4 Page faults

0 is already there so —> 0 Page fault. when 3 comes it will take the place of 7 because it is not used for the longest duration of time in the future. —>1 Page fault. 0 is already there so —> 0 Page fault. 4 will take place of 1 —> 1 Page Fault.

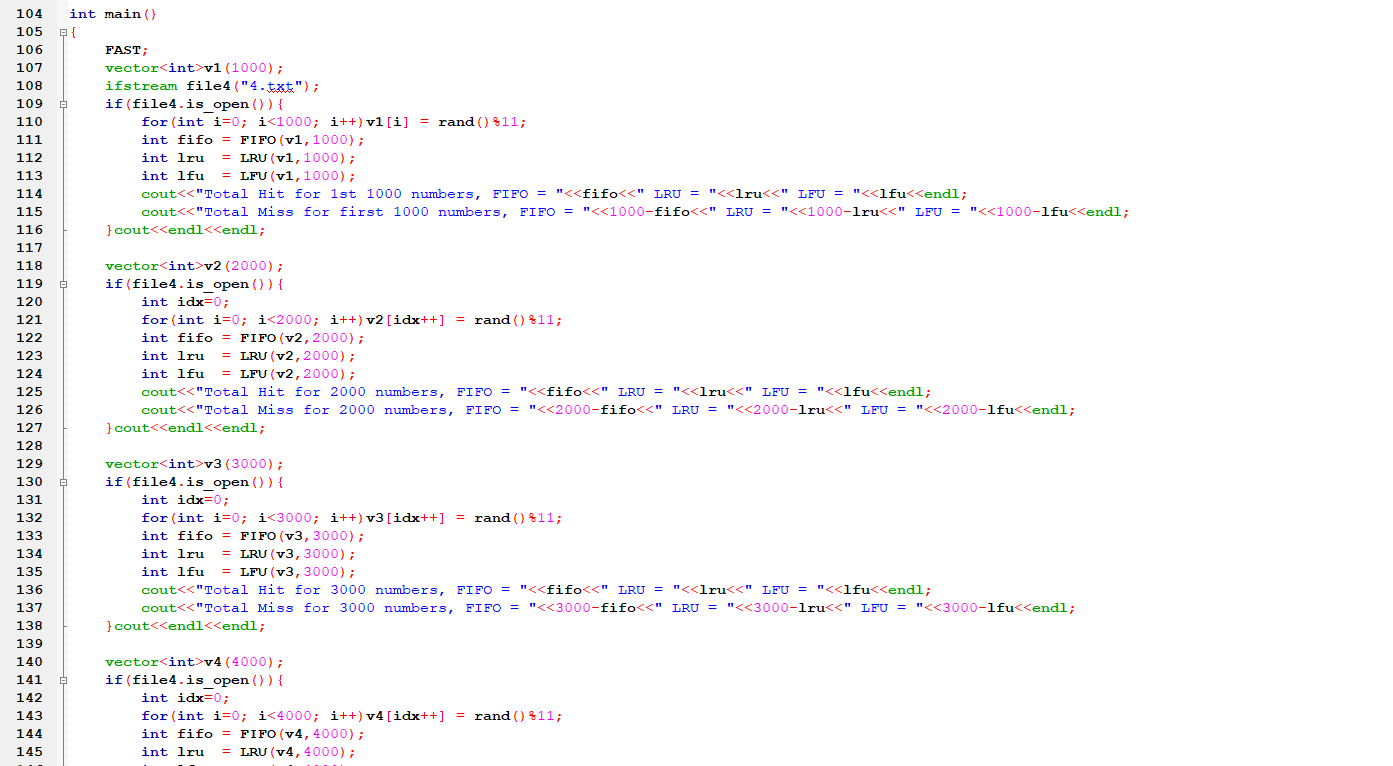
Now for the further page reference string —> 0 Page fault because they are already available in the memory.

Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

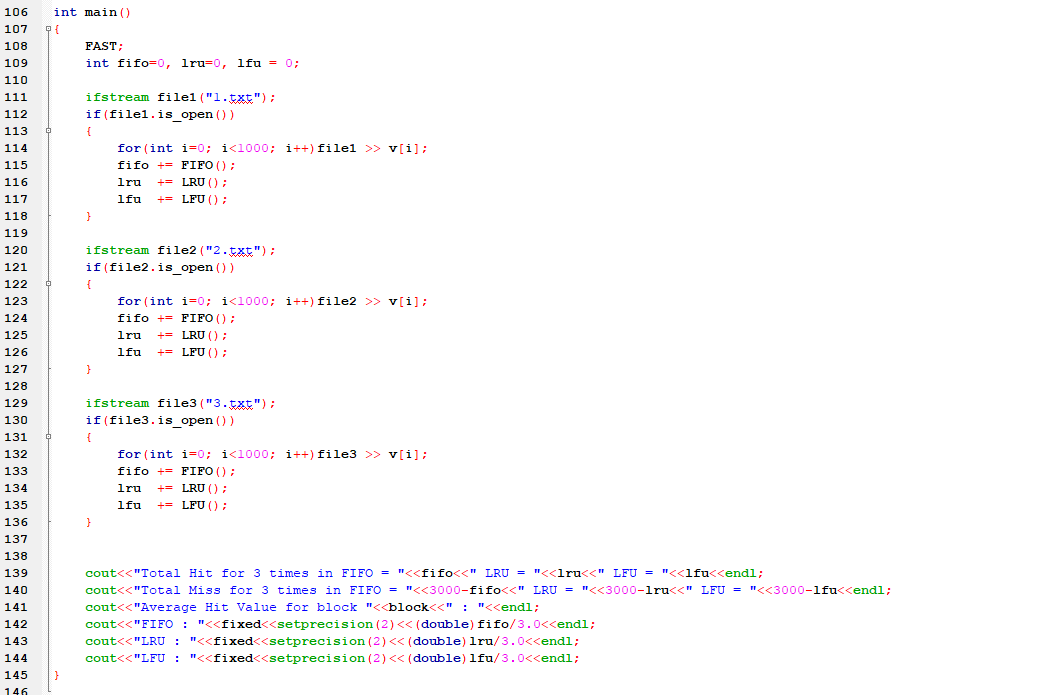
**Function Implementation:**

1. **Main Function**

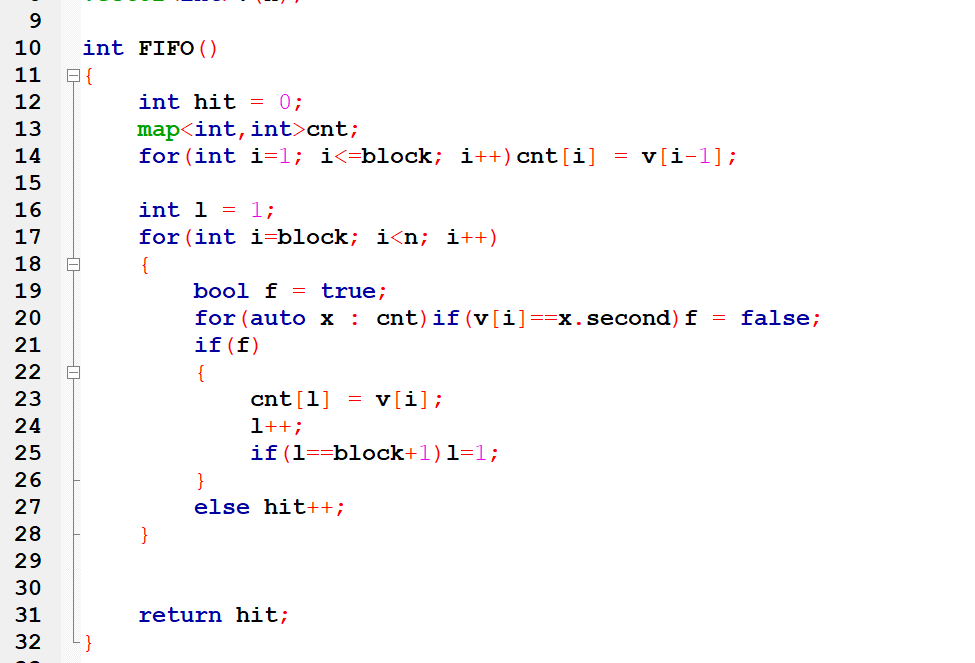
**For problem B:**

****

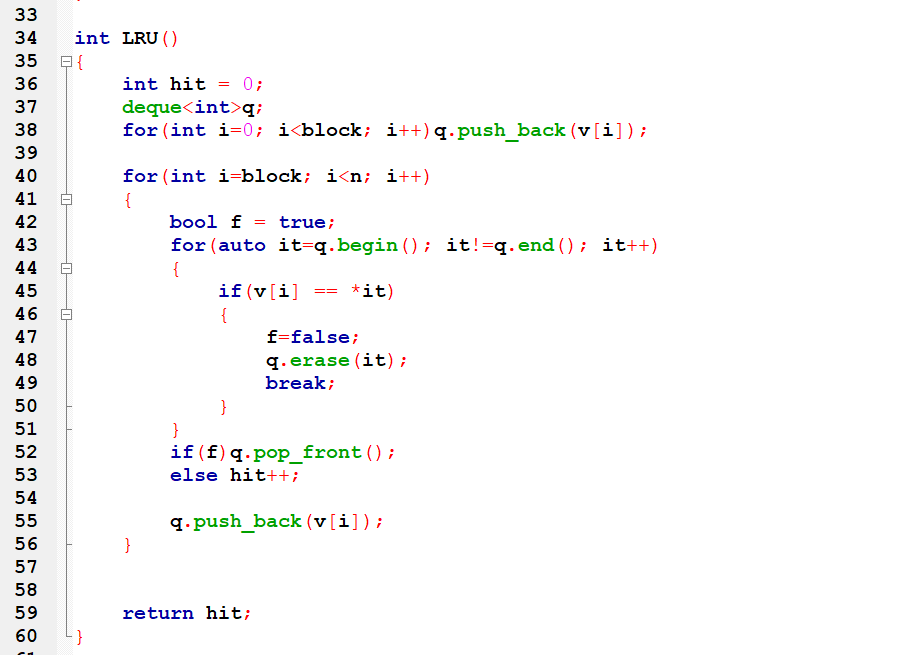
**For Problem A:**

****

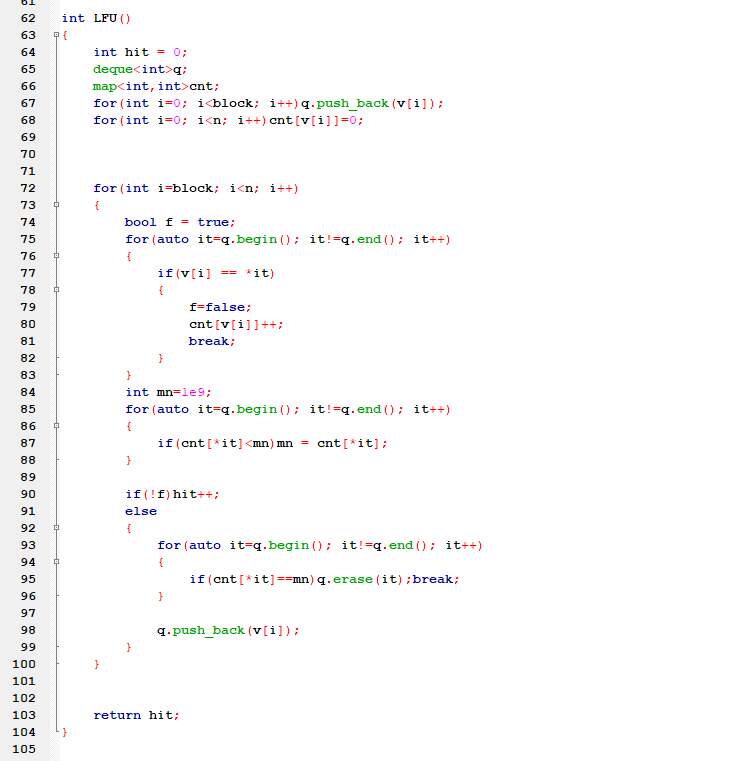
1. **FIFO**

****

1. **LRU**

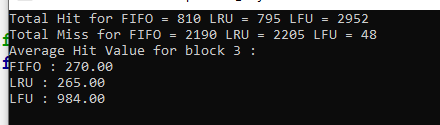
****

1. **LFU**

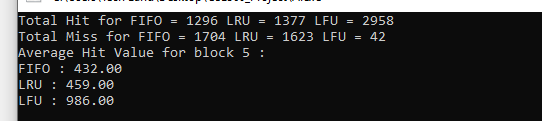
****

**Output:**

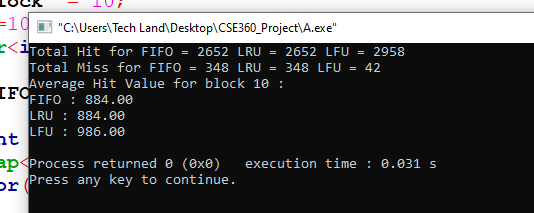
1. **1st Problem For block 3**

****

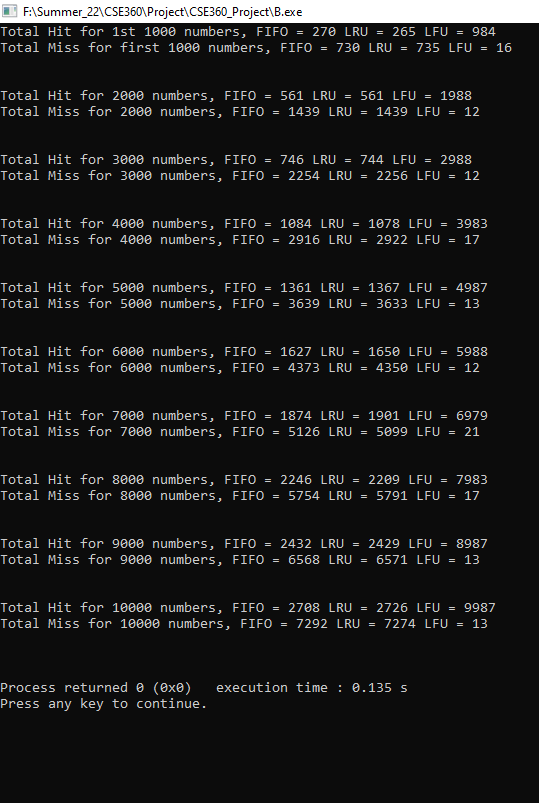
1. **1st Problem For block 5**

****

1. **1st Problem For block 10**

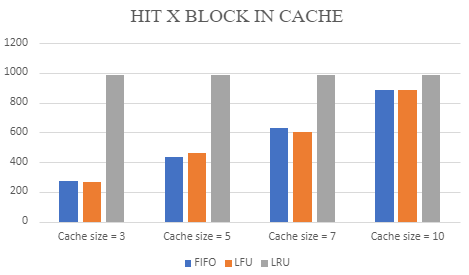
****

**2nd Problem block 3 fixed**

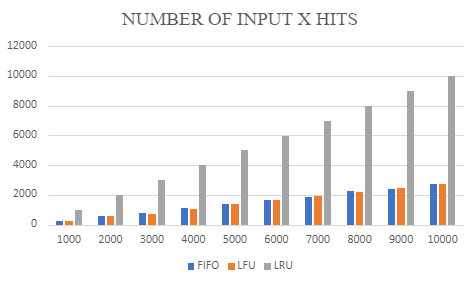
****

**Graph:**

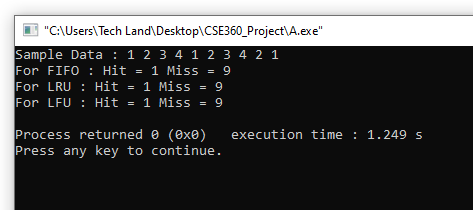
1. **Problem A**

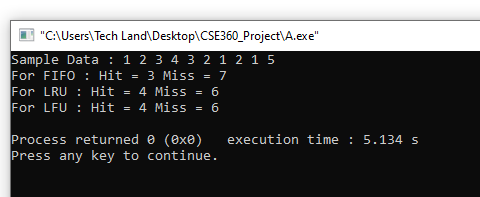
****

1. **Problem B**

****

**Output from random problems which has been solved in the class:**

****

****

We can say from this output, our solution is giving right output.