Madan Bhandari Memorial College

Lab No. 1 **Date: 2025/08/12**

Title: Write a program to calculate the number of page fault for user input references string and frame size using FIFO page replacement algorithm.

In operating systems, memory management plays a crucial role in ensuring efficient execution of processes. When a process is executed, it requires pages to be loaded into main memory. However, the number of available frames in main memory is limited. If a page required by a process is not present in memory, a page fault occurs, and the operating system must bring the page from secondary storage into main memory.

Page Replacement Algorithms decide which page in memory should be replaced when a new page needs to be loaded and the memory is already full.

The FIFO (First-In-First-Out) Page Replacement Algorithm is one of the simplest page replacement strategies. It replaces the oldest page in memory the one that was loaded first regardless of how frequently or recently it has been accessed.

Frame Size: Frame size is the fixed size of a block of physical memory where pages are loaded, typically matching the page size to avoid fragmentation.

Page Fault: A page fault occurs when a program attempts to access a page not currently in physical memory, requiring the operating system to load the page from secondary storage into RAM.

Algorithm:

```
1) Initialize an empty queue with a fixed size.
2) For each page request.
        • If page is in memory, continue
        • If page is not in memory i.e page fault occurs:
        → If there is space in memory, insert the new page into the queue.
        → If memory is full, remove the page at the front of the queue (oldest) and insert the new page
          at the rear and increment the page fault.
3) Output the total number of page faults.
```

Programming Language: C

IDE: Dev C++

Source Code:

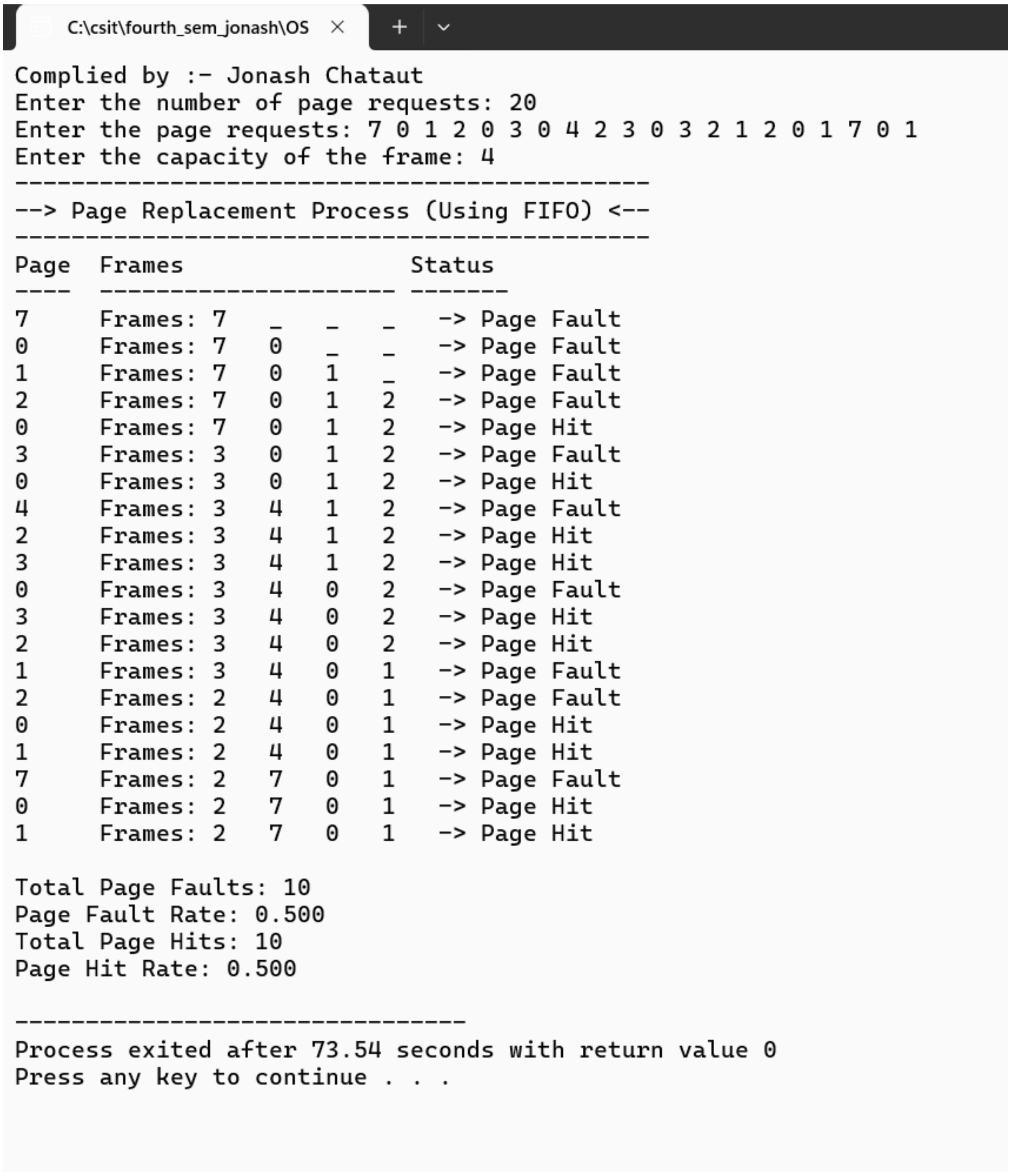
```
#include <iostream>
#include <iomanip>
using namespace std;
// Seach the page
int search(int frames[], int n, int page)
{
   for (int i = 0; i < n; i++)
   {
      if (frames[i] == page)
      {
         return i; // Page found in frame
      }
   }
   return -1; // Page not found in frame
}
void printFrames(int frames[], int frame_size, bool page_hit)
{
   cout << "Frames: ";
   for (int i = 0; i < frame_size; i++)
      if (frames[i] == -1)
         // cout << " _ ";
         cout << setw(4) << "_";
      else
         cout << setw(4) << frames[i];
   cout << setw(22) << (page_hit ? "-> Page Hit" : "-> Page Fault") << "\n";
}
void fifo(int pages[], int n, int capacity)
{
   int frames[capacity];
   int faults = 0;
   int nextFrame = 0; // Index of the next frame to be replaced

   // Initialize frames with -1 to indicate an empty frame
   for (int i = 0; i < capacity; i++)
   {
      frames[i] = -1;
   }
   cout << "---------------------------------------------\n";
   cout << "--> Page Replacement Process (Using FIFO) <--\n";
   cout << "---------------------------------------------\n";
   cout << setw(6) << left << "Page" << setw(22) << "Frames" << "Status\n";
   cout << setw(6) << left << "----" << setw(22) << "---------------------" << "-------\n";
   for (int i = 0; i < n; i++)
   {
      int page = pages[i];
      cout << setw(6) << left << page;
      bool page_hit;
      if (search(frames, capacity, page) == -1)
      {
            // Page fault
            frames[nextFrame] = page;
            nextFrame = (nextFrame + 1) % capacity;
            faults++;
            // cout << "Page Fault!\n";
            page_hit = false;
        }
        else
        {
            page_hit = true;
        }
        printFrames(frames, capacity, page_hit);
    }
    cout << "\nTotal Page Faults: " << faults << "\n";
    cout << "Page Fault Rate: " << fixed << setprecision(3) << (float)faults / n << "\n";
    cout << "Total Page Hits: " << n - faults << "\n";
    cout << "Page Hit Rate: " << fixed << setprecision(3) << (float)(n - faults) / n << "\n";
}

int main()
{
    cout << "Complied by :- Jonash Chataut\n";
    int np;     // Number of page requests
    int capacity; // Capacity of the frame
    cout << "Enter the number of page requests: ";
    cin >> np;
    if (np <= 0)
    {
        cout << "Error: Invalid number of page requests.\n";
        return 0;
    }
    int pages[np];
    cout << "Enter the page requests: ";
    for (int i = 0; i < np; i++)
    {
        cin >> pages[i];
    }
    cout << "Enter the capacity of the frame: ";
    cin >> capacity;
    if (capacity <= 0)
    {
        cout << "Error: Invalid capacity.\n";
        return 0;
    }
    fifo(pages, np, capacity);

    return 0;
}
```

Output:



```
Complied by :- Jonash Chataut
Enter the number of page requests: 20
Enter the page requests: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter the capacity of the frame: 4
---------------------------------------------
--> Page Replacement Process (Using FIFO) <--
---------------------------------------------
Page  Frames                Status
----  --------------------- -------
7     Frames: 7   _   _   _   -> Page Fault
0     Frames: 7   0   _   _   -> Page Fault
1     Frames: 7   0   1   _   -> Page Fault
2     Frames: 7   0   1   2   -> Page Fault
0     Frames: 7   0   1   2   -> Page Hit
3     Frames: 3   0   1   2   -> Page Fault
0     Frames: 3   0   1   2   -> Page Hit
4     Frames: 3   4   1   2   -> Page Fault
2     Frames: 3   4   1   2   -> Page Hit
3     Frames: 3   4   1   2   -> Page Hit
0     Frames: 3   4   0   2   -> Page Fault
3     Frames: 3   4   0   2   -> Page Hit
2     Frames: 3   4   0   2   -> Page Hit
1     Frames: 3   4   0   1   -> Page Fault
2     Frames: 2   4   0   1   -> Page Fault
0     Frames: 2   4   0   1   -> Page Hit
1     Frames: 2   4   0   1   -> Page Hit
7     Frames: 2   7   0   1   -> Page Fault
0     Frames: 2   7   0   1   -> Page Hit
1     Frames: 2   7   0   1   -> Page Hit

Total Page Faults: 10
Page Fault Rate: 0.500
Total Page Hits: 10
Page Hit Rate: 0.500

--------------------------------
Process exited after 73.54 seconds with return value 0
Press any key to continue . . .
```

Lab No. 2 **Date: 2025/08/12**

Title: Write a program to calculate the number of page fault for user input references string and frame size using OPR page replacement algorithm.

The Optimal Page Replacement (OPR) Algorithm is a theoretical page replacement strategy that achieves the minimum possible number of page faults. It works by replacing the page that will not be used for the longest period of time in the future. Since it requires future knowledge of the page reference string, OPR cannot be implemented in a real operating system but is used for comparison and analysis purposes.

In OPR, when a page fault occurs and the memory is full, the algorithm scans the upcoming page references to determine which page in memory will be used farthest in the future (or not at all). That page is then replaced with the new page.

Algorithm:

1. For each page request:

- If the page is in memory, continue.

- If the page is not in memory:

• If memory is full, determine which page will be used the furthest in the

future (or not at all) and remove it.

• Add the new page to memory.

• Increment the page faults.

2. Output the total number of page faults.

Programming Language: C

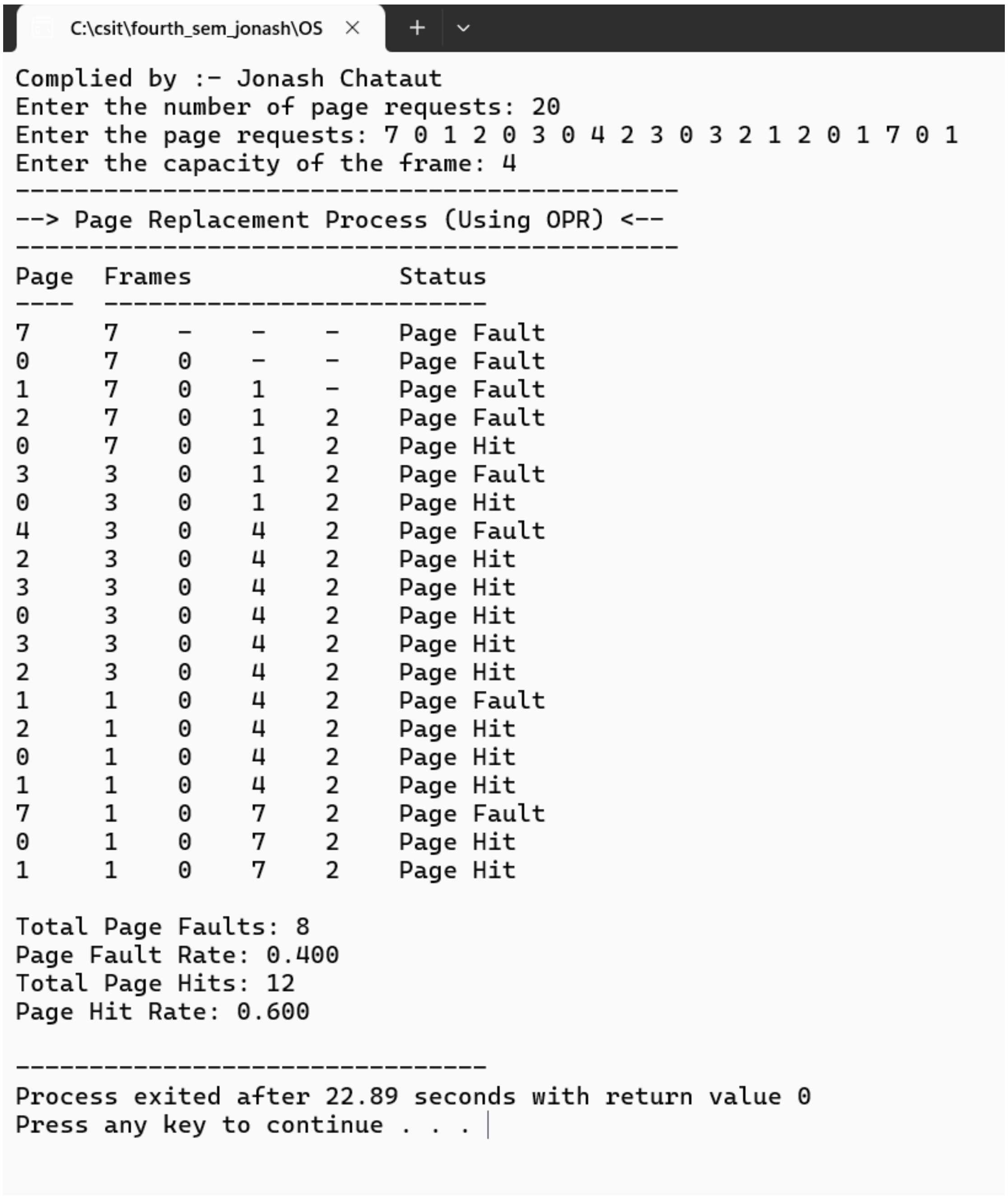
IDE: Dev C++

Source Code:

```
#include <iostream>
#include <iomanip>
using namespace std;
bool search(int frames[], int n, int page)
{
   for (int i = 0; i < n; i++)
   {
      if (frames[i] == page)
         return true;
   }
   return false;
}
int predict(int ref_str[], int frames[], int refStrLen, int index, int frame_occupied)
{
   int result = -1;
   int farthest = index;
   for (int i = 0; i < frame_occupied; i++)
   {
      int j;
      for (j = index; j < refStrLen; j++)
      {
         if (frames[i] == ref_str[j])
         {
            if (j > farthest)
            {
               farthest = j;
               result = i;
            }
            break;
         }
      }
      if (j == refStrLen)
         return i;
   }
   return (result == -1) ? 0 : result;
}
void printFrames(int frames[], int frame_size, int frame_occupied, bool page_hit)
{
   for (int i = 0; i < frame_size; i++)
   {
      if (i < frame_occupied)
         cout << left << setw(5) << frames[i];
      else
         cout << left << setw(5) << "-";
   }
   cout << left << setw(15) << (page_hit ? "Page Hit" : "Page Fault") << "\n";
}
void optimalPage(int pages[], int np, int capacity)
{
   int frames[capacity];
```

```
    int frame_occupied = 0;
    int faults = 0;
    cout << "-----------------------------------------------\n";
    cout << "--> Page Replacement Process (Using OPR) <--\n";
    cout << "-----------------------------------------------\n";
    cout << setw(6) << left << "Page" << setw(5 * capacity) << left << "Frames" << "Status\n";
    cout << setw(6) << left << "----";
    for (int i = 0; i < capacity; i++)
        cout << setw(5) << "-----";
    cout << "------\n";
    for (int i = 0; i < np; i++)
    {
        int page = pages[i];
        cout << setw(6) << left << page;
        bool page_hit = false;
        if (search(frames, frame_occupied, page))
        {
            page_hit = true;
        }
        else
        {
            // Page fault
            faults++;
            if (frame_occupied < capacity)
            {
                frames[frame_occupied++] = page;
            }
            else
            {
                int pos = predict(pages, frames, np, i + 1, frame_occupied);
                frames[pos] = page;
            }
        }
        printFrames(frames, capacity, frame_occupied, page_hit);
    }
    cout << "\nTotal Page Faults: " << faults << "\n";
    cout << "Page Fault Rate: " << fixed << setprecision(3) << (float)faults / np << "\n";
    cout << "Total Page Hits: " << np - faults << "\n";
    cout << "Page Hit Rate: " << fixed << setprecision(3) << (float)(np - faults) / np << "\n";
}
int main()
{
    cout << "Complied by :- Jonash Chataut\n";
    int np, capacity;
    cout << "Enter the number of page requests: ";
    cin >> np;
    if (np <= 0)
    {
        cout << "Error: Invalid number of page requests.\n";
        return 0;
    }
    int pages[np];
    cout << "Enter the page requests: ";
    for (int i = 0; i < np; i++)
    {
      cin >> pages[i];
    }
    cout << "Enter the capacity of the frame: ";
    cin >> capacity;
    if (capacity <= 0)
    {
      cout << "Error: Invalid capacity.\n";
      return 0;
    }
    optimalPage(pages, np, capacity);
    return 0;
  }
```

Output:



```
Complied by :- Jonash Chataut
Enter the number of page requests: 20
Enter the page requests: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter the capacity of the frame: 4
---------------------------------------------
--> Page Replacement Process (Using OPR) <--
---------------------------------------------
Page  Frames                  Status
----  --------------------------
7     7    -    -    -    Page Fault
0     7    0    -    -    Page Fault
1     7    0    1    -    Page Fault
2     7    0    1    2    Page Fault
0     7    0    1    2    Page Hit
3     3    0    1    2    Page Fault
0     3    0    1    2    Page Hit
4     3    0    4    2    Page Fault
2     3    0    4    2    Page Hit
3     3    0    4    2    Page Hit
0     3    0    4    2    Page Hit
3     3    0    4    2    Page Hit
2     3    0    4    2    Page Hit
1     1    0    4    2    Page Fault
2     1    0    4    2    Page Hit
0     1    0    4    2    Page Hit
1     1    0    4    2    Page Hit
7     1    0    7    2    Page Fault
0     1    0    7    2    Page Hit
1     1    0    7    2    Page Hit

Total Page Faults: 8
Page Fault Rate: 0.400
Total Page Hits: 12
Page Hit Rate: 0.600

--------------------------------
Process exited after 22.89 seconds with return value 0
Press any key to continue . . .
```

Madan Bhandari Memorial College

Lab No. 3 **Date: 2025/08/12**

Title: Write a program to calculate the number of page fault for user input references string and frame size using LRU page replacement algorithm.

The Least Recently Used (LRU) Page Replacement Algorithm is based on the principle of replacing the page that has not been used for the longest period of time in the past. The assumption is that pages used recently are more likely to be used again soon, while pages not used for a long time are less likely to be needed.

When a page fault occurs and the memory is full, LRU scans the usage history of the pages currently in memory and selects the one with the longest idle time (least recently used) for replacement.

Algorithm:

```
1. Initialize an empty list to keep track of the pages in memory and their last used time.
2. For each page request:
    - If the page is in memory, update its last used time.
    - If the page is not in memory:
        - If memory is full, remove the page that has not been used for the longest time.
        - Add the new page to the memory.
        - Increment the page faults.
3. Output the total number of page faults.
```

Programming Language: C

IDE: Dev C++

Source Code:

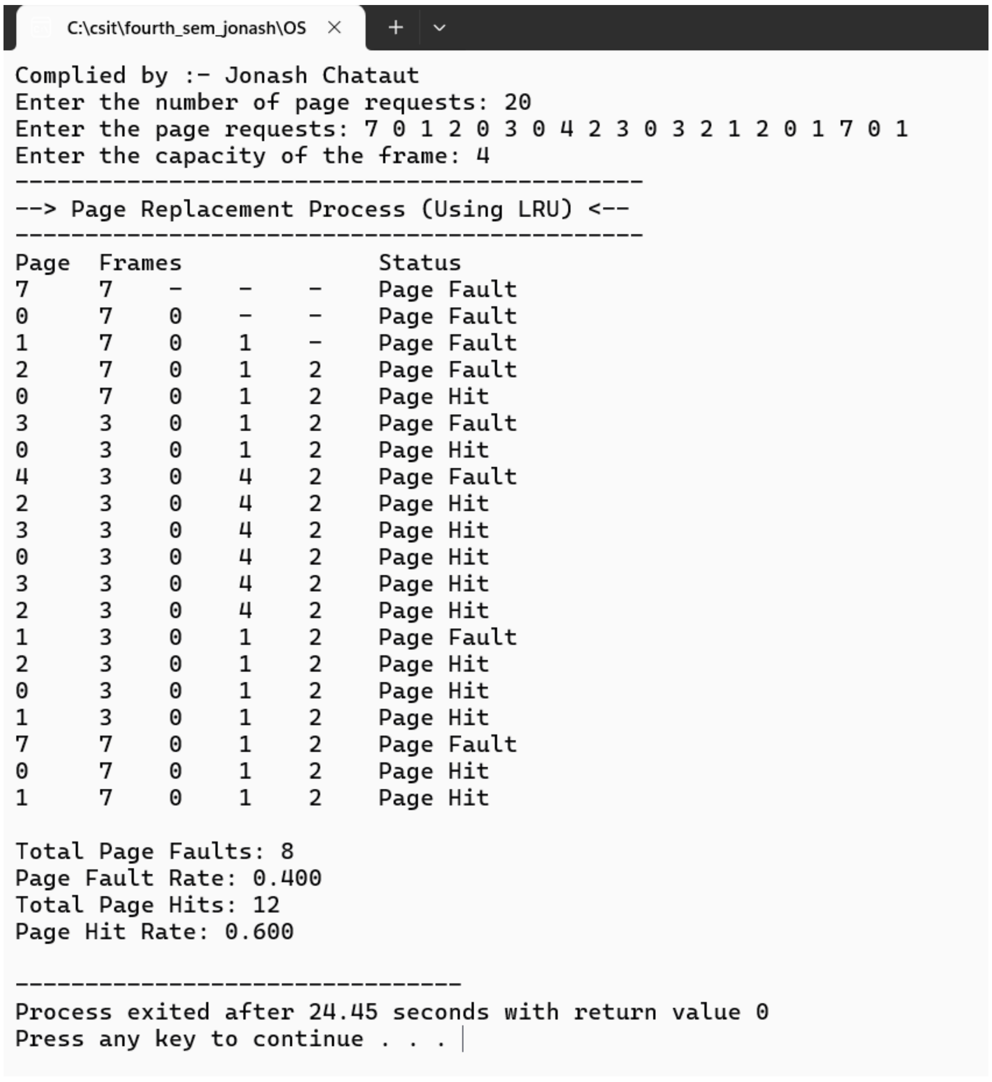
```
#include <iostream>
#include <iomanip>
using namespace std;
// function to check pagehit
bool isPageHit(int frames[], int n, int page)
{
   for (int i = 0; i < n; i++)
      if (frames[i] == page)
         return true;
   return false;
}
// function to get index of least used page
int getLRUPageIndex(int count[], int frameCount)
{
   int minIndex = 0;
   for (int i = 1; i < frameCount; i++)
   {
      if (count[i] < count[minIndex])
         minIndex = i;
   }
   return minIndex;
}
void printFrames(int frames[], int frame_size, int frame_occupied, bool page_hit)
{
   for (int i = 0; i < frame_size; i++)
   {
      if (i < frame_occupied)
         cout << setw(5) << frames[i];
      else
         cout << setw(5) << "-";
   }
   cout << setw(15) << (page_hit ? "Page Hit" : "Page Fault") << "\n";
}
void lru(int pages[], int np, int capacity)
{
   int frames[capacity];   // stores pages in frames
   int count[capacity];    // stores last used time for each frame
   int frame_occupied = 0; // how many frames currently occupied
   int pageFaults = 0;     // count page faults
   int time = 0;           // global counter to track usage
   // Initialize frames as empty (-1) and count as 0
   for (int i = 0; i < capacity; i++)
   {
      frames[i] = -1;
      count[i] = 0;
   }
   // Print headers for output table
   cout << "-------------------------------------------\n";
   cout << "--> Page Replacement Process (Using LRU) <--\n";
   cout << "-------------------------------------------\n";
```

```
    cout << setw(6) << left << "Page" << setw(5 * capacity) << left << "Frames" << "Status\n";
    for (int i = 0; i < np; i++)
    {
        int page = pages[i];
        cout << setw(6) << left << page;
        // Check if page is already in frames (page hit)
        bool page_hit = isPageHit(frames, frame_occupied, page);
        if (!page_hit)
        {
            // Page fault occurred so incrementing
            pageFaults++;
            if (frame_occupied < capacity)
            {
                // Still space in frames so adding page directly
                frames[frame_occupied] = page;
                count[frame_occupied] = time++; // record last used time
                frame_occupied++;
            }
            else
            {
                // Need to replace the LRU page
                int lruIndex = getLRUPageIndex(count, capacity);
                frames[lruIndex] = page;
                count[lruIndex] = time++;
            }
        }
        else
        {
            // Page hit: update last used time of that frame
            for (int j = 0; j < frame_occupied; j++)
            {
                if (frames[j] == page)
                {
                    count[j] = time++;
                    break;
                }
            }
        }

        // Print current state of frames and status
        printFrames(frames, capacity, frame_occupied, page_hit);
    }
    // Print stats
    cout << "\nTotal Page Faults: " << pageFaults << "\n";
    cout << "Page Fault Rate: " << fixed << setprecision(3) << (float)pageFaults / np << "\n";
    cout << "Total Page Hits: " << np - pageFaults << "\n";
    cout << "Page Hit Rate: " << fixed << setprecision(3) << (float)(np - pageFaults) / np << "\n";
}
int main()
{
    cout << "Complied by :- Jonash Chataut\n";
    int np, capacity;
```

```
        cout << "Enter the number of page requests: ";
        cin >> np;
        if (np <= 0)
        {
          cout << "Invalid number of page requests.\n";
          return 0;
        }
        int pages[np];
        cout << "Enter the page requests: ";
        for (int i = 0; i < np; i++)
        {
          cin >> pages[i];
        }
        cout << "Enter the capacity of the frame: ";
        cin >> capacity;
        if (capacity <= 0)
        {
          cout << "Invalid capacity.\n";
          return 0;
        }
        lru(pages, np, capacity);
        return 0;
      }
```

Output:



C:\csit\fourth_sem_jonash\OS

```
Complied by :- Jonash Chataut
Enter the number of page requests: 20
Enter the page requests: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter the capacity of the frame: 4
---------------------------------------------
--> Page Replacement Process (Using LRU) <--
---------------------------------------------
Page  Frames              Status
7     7    -    -    -    Page Fault
0     7    0    -    -    Page Fault
1     7    0    1    -    Page Fault
2     7    0    1    2    Page Fault
0     7    0    1    2    Page Hit
3     3    0    1    2    Page Fault
0     3    0    1    2    Page Hit
4     3    0    4    2    Page Fault
2     3    0    4    2    Page Hit
3     3    0    4    2    Page Hit
0     3    0    4    2    Page Hit
3     3    0    4    2    Page Hit
2     3    0    4    2    Page Hit
1     3    0    1    2    Page Fault
2     3    0    1    2    Page Hit
0     3    0    1    2    Page Hit
1     3    0    1    2    Page Hit
7     7    0    1    2    Page Fault
0     7    0    1    2    Page Hit
1     7    0    1    2    Page Hit

Total Page Faults: 8
Page Fault Rate: 0.400
Total Page Hits: 12
Page Hit Rate: 0.600

--------------------------------
Process exited after 24.45 seconds with return value 0
Press any key to continue . . .
```