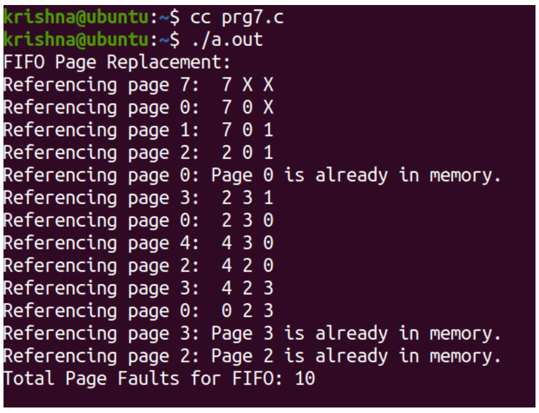
Program 7. Develop a C program to simulate page replacement algorithms:
a) FIFO   b) LRU

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_FRAMES 3 // Maximum number of frames in the memory
// Function to print the current state of frames in memory
void printFrames(int frames[], int n) {
    // Iterate through frames and print the page numbers or 'X' for empty frames
    for (int i = 0; i < n; i++) {
        if (frames[i] == -1)
            printf(" X"); // 'X' indicates an empty frame
        else
            printf(" %d", frames[i]); // Print page number in the frame
    }
    printf("\n");
}
// Function to search for a page in the frames
int search(int key, int frames[], int n) {
    // Linear search for the page in the frames
    for (int i = 0; i < n; i++)
        if (frames[i] == key)
            return i; // Return the index if page is found in frames
    return -1; // Return -1 if page is not found in frames
}
// FIFO Page Replacement Algorithm
int fifoPageReplacement(int pageReferences[], int n, int capacity) {
    int frames[capacity]; // Array to store frames in memory
    int pageFaults = 0; // Counter for page faults
    int frameIndex = 0; // Index to keep track of the next frame to replace
    // Initialize frames as empty
    for (int i = 0; i < capacity; i++)
        frames[i] = -1;

    // Iterate through page references
    for (int i = 0; i < n; i++) {
        printf("Referencing page %d: ", pageReferences[i]);
        // Check if page is already in memory
        if (search(pageReferences[i], frames, capacity) == -1) {
            // Page fault: Page is not in memory
            // Replace the oldest page with the current page using FIFO
            frames[frameIndex] = pageReferences[i];
            frameIndex = (frameIndex + 1) % capacity; // Update frame index
            pageFaults++; // Increment page fault count
            printFrames(frames, capacity); // Print current state of frames
        } else {
            printf("Page %d is already in memory.\n", pageReferences[i]);
        }
    }

    return pageFaults; // Return total number of page faults
}

// LRU Page Replacement Algorithm
int lruPageReplacement(int pageReferences[], int n, int capacity) {
    int frames[capacity]; // Array to store frames in memory
    int pageFaults = 0; // Counter for page faults
```

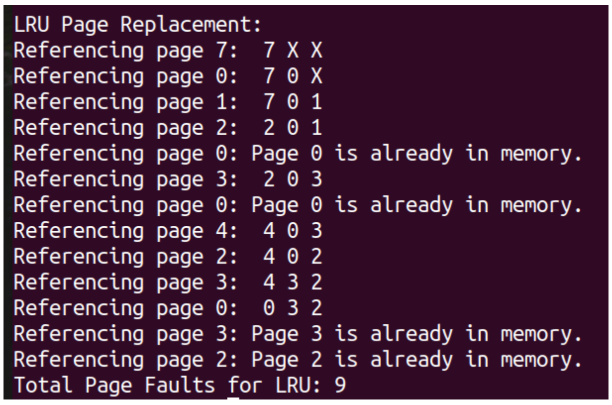
```
// Initialize frames as empty
  for (int i = 0; i < capacity; i++)
     frames[i] = -1;

  // Iterate through page references
  for (int i = 0; i < n; i++) {
     printf("Referencing page %d: ", pageReferences[i]);
     int index = search(pageReferences[i], frames, capacity);
     // Check if page is already in memory
     if (index == -1) {
        // Page fault: Page is not in memory

        int leastRecentlyUsed = n;
        int victimIndex;
        for (int j = 0; j < capacity; j++) {
           int k;
           for (k = i - 1; k >= 0; k--)
              if (frames[j] == pageReferences[k])
                 break;
           // Update least recently used page if found
           if (k < leastRecentlyUsed) {
              leastRecentlyUsed = k;
              victimIndex = j;
           }
        }
        // Replace the least recently used page with the current page
        frames[victimIndex] = pageReferences[i];
        pageFaults++; // Increment page fault count
        printFrames(frames, capacity); // Print current state of frames
     } else
```

```
        {
            printf("Page %d is already in memory.\n", pageReferences[i]);
        }
    }
    return pageFaults; // Return total number of page faults
}

int main() {
    // New page reference sequence
    int pageReferences[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
    int n = sizeof(pageReferences) / sizeof(pageReferences[0]); // Number of page
references
    int capacity = MAX_FRAMES; // Number of frames in memory

    printf("FIFO Page Replacement:\n");
    // Simulate FIFO page replacement algorithm
    int fifoFaults = fifoPageReplacement(pageReferences, n, capacity);
    printf("Total Page Faults for FIFO: %d\n\n", fifoFaults);

    printf("LRU Page Replacement:\n");
    // Simulate LRU page replacement algorithm
    int lruFaults = lruPageReplacement(pageReferences, n, capacity);
    printf("Total Page Faults for LRU: %d\n", lruFaults);

    return 0;
}
```

Output:



```
krishna@ubuntu:~$ cc prg7.c
krishna@ubuntu:~$ ./a.out
FIFO Page Replacement:
Referencing page 7:  7 X X
Referencing page 0:  7 0 X
Referencing page 1:  7 0 1
Referencing page 2:  2 0 1
Referencing page 0: Page 0 is already in memory.
Referencing page 3:  2 3 1
Referencing page 0:  2 3 0
Referencing page 4:  4 3 0
Referencing page 2:  4 2 0
Referencing page 3:  4 2 3
Referencing page 0:  0 2 3
Referencing page 3: Page 3 is already in memory.
Referencing page 2: Page 2 is already in memory.
Total Page Faults for FIFO: 10
```



```
LRU Page Replacement:
Referencing page 7:  7 X X
Referencing page 0:  7 0 X
Referencing page 1:  7 0 1
Referencing page 2:  2 0 1
Referencing page 0: Page 0 is already in memory.
Referencing page 3:  2 0 3
Referencing page 0: Page 0 is already in memory.
Referencing page 4:  4 0 3
Referencing page 2:  4 0 2
Referencing page 3:  4 3 2
Referencing page 0:  0 3 2
Referencing page 3: Page 3 is already in memory.
Referencing page 2: Page 2 is already in memory.
Total Page Faults for LRU: 9
```