Page Replacement using **FIFO**

```
#include <stdio.h>
int n, pg[30], fr[10];
void fifo();

int main()
{
   int i;
   printf("Enter total number of pages: ");
   scanf("%d", &n);

   printf("Enter page sequence:\n");
   for (i = 0; i < n; i++) {
      scanf("%d", &pg[i]);
   }

   fifo();
   return 0;
}

void fifo()
 {
   int i, f = 0, r = 0, s = 0, count = 0, flag = 0, num, psize;

   printf("Enter the size of page frame: ");
   scanf("%d", &psize);
```

```
  for (i = 0; i < psize; i++) {
      fr[i] = -1;  // Initialize all frames to -1 (empty)
   }

   while (s < n)
{
      flag = 0;
      num = pg[s];

      // Check if page is already in frame
      for (i = 0; i < psize; i++)
 {
         if (num == fr[i])
{
            flag = 1;
            break;
         }
      }

      // If page is not in frame, we have a page fault
      if (flag == 0)
{
         fr[f] = pg[s];   // Replace page in the FIFO order
         f = (f + 1) % psize;  // Move FIFO pointer
         count++;   // Increment page fault counter
      }
      s++;  // Move to next page in sequence
```

```
        // Print the current page frame state
        printf("\nPage Frame: ");
        for (i = 0; i < psize; i++)
{
          if (fr[i] != -1)
{
              printf("%d ", fr[i]);
          }
else
{
              printf("- ");
          }
        }
    }

    printf("\n\nTotal Page Faults: %d\n", count);
}
```

optimal page replacement using (least recently used)

```
#include <stdio.h>

int n, pg[30], fr[10];
void lru();

int main()
 {
   int i;
   printf("Enter total number of pages: \n");
   scanf("%d", &n);
   printf("Enter page sequence:\n");
   for (i = 0; i < n; i++)
      scanf("%d", &pg[i]);
   lru();  // Call the lru function
   return 0;
}

void lru()
{
   int count[10], i, j, fault = 0, flag, f, temp, min, m, x;

   printf("Enter the frame size: \n");
   scanf("%d", &f);

   // Initialize frame array and count array
   for (i = 0; i < f; i++)
```

```
{
    fr[i] = -1;    // Set each frame initially to -1 (empty)
    count[i] = -1; // Initialize the count array to -1 for clarity
  }

  for (i = 0; i < n; i++)
{
    flag = 0;
    temp = pg[i];

    // Check if the page is already in a frame
    for (j = 0; j < f; j++)
{
      if (fr[j] == temp)
{
        flag = 1;        // Page hit
        count[j] = i;     // Update the most recent usage time
        break;
      }
    }

    // Page fault: the page is not in any frame
    if (flag == 0)
{
      // If there's still space in frames, place page in the next empty slot
      if (fault < f)
{
```

```
            fr[fault] = temp;
            count[fault] = i;  // Record the time of usage for LRU
            fault++;
        }
else
{
            // LRU replacement: find the least recently used page
            min = 0;
            for (m = 1; m < f; m++)
 {
                if (count[m] < count[min])
{
                    min = m;
                }
            }
            fr[min] = temp;     // Replace the least recently used page
            count[min] = i;     // Update usage time
            fault++;
        }
    }

    // Print the current state of frames
    printf("\nPage frames after accessing page %d:\t", temp);
    for (x = 0; x < f; x++)
{
        if (fr[x] != -1)
            printf("%d\t", fr[x]);
```

```
            else
                printf("-\t");     // Show empty slots
        }
    }

    printf("\n\nTotal number of page faults = %d\n", fault);
}
```

optimal page replacement using (Optimal Page Replacement)

```
#include <stdio.h>
#include <stdbool.h>

bool contains(int frame[], int frameSize, int page)
{
   for (int i = 0; i < frameSize; i++)
{
      if (frame[i] == page)
{
         return true;
      }
   }
   return false;
}

int findOptimalPage(int pages[], int n, int frame[], int frameSize, int currentIndex)
{
   int res = -1, farthest = currentIndex;
   for (int i = 0; i < frameSize; i++)
{
      int j;
      for (j = currentIndex; j < n; j++)
{
         if (frame[i] == pages[j])
{
            if (j > farthest)
```

```
{
                farthest = j;
                res = i;
            }
            break;
        }
    }
    if (j == n)
        return i;
  }
  return (res == -1) ? 0 : res;
}

void optimalPageReplacement(int pages[], int n, int frameSize)
{
  int frame[frameSize];
  bool isPageFault[n];
  int pageFaults = 0;

  // Initialize frame and page faults
  for (int i = 0; i < frameSize; i++)
    frame[i] = -1;
  for (int i = 0; i < n; i++)
    isPageFault[i] = false;

  for (int i = 0; i < n; i++)
 {
```

```
        // Check if page is already in the frame
        if (!contains(frame, frameSize, pages[i]))
{
            // Page fault occurs
            int replaceIndex;
            if (i < frameSize)
{
                // If there's an empty slot in the frame, use it
                replaceIndex = i;
            }
else
 {
                // Otherwise, find the optimal page to replace
                replaceIndex = findOptimalPage(pages, n, frame, frameSize, i + 1);
            }
            frame[replaceIndex] = pages[i];
            isPageFault[i] = true;
            pageFaults++;

            // Print current frame state
            printf("Frame state after inserting page %d: ", pages[i]);
            for (int j = 0; j < frameSize; j++)
 {
                if (frame[j] != -1)
                    printf("%d ", frame[j]);
                else
                    printf("- ");
```

```
            }
            printf("\n");
        }
    }

    printf("Total Page Faults: %d\n", pageFaults);
}

int main()
{
    int pages[50], n, frameSize, i;
    printf("Enter the number of pages: ");
    scanf("%d", &n);

    printf("Enter the page sequence of %d pages: ", n);
    for (i = 0; i < n; i++)
        scanf("%d", &pages[i]);

    printf("Enter the frame size: ");
    scanf("%d", &frameSize);

    optimalPageReplacement(pages, n, frameSize);
    return 0;
}
```