```c
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>
#include <stdlib.h>

void fifo(int pages, int frame, int inputs[]){
    printf("Incoming Frame 1 Frame 2 Frame 3\n");
    int temp[frame];
    int pagefault = 0;
    for (int i = 0; i < frame; i++)
    {
        temp[i] = -1;
    }
    for (int i = 0; i < pages; i++)
    {
        int s = 0;
        for (int j = 0; j < frame; j++)
        {
            if (temp[j] == inputs[i])
            {
                s++;
                pagefault--;
            }
        }
        pagefault++;
        if (pagefault <= frame && s == 0)
        {
            temp[i % frame] = inputs[i];
        }
        else if (s == 0)
        {
            temp[(pagefault - 1) % frame] = inputs[i];
        }
        printf("%d\t", inputs[i]);
        for (int j = 0; j < frame; j++)
        {
            if (temp[j] != -1)
                printf(" %d\t", temp[j]);
            else
                printf(" - \t");
        }
        printf("\n");
    }
    printf("Total Page Faults:\t%d\n", pagefault);
}


int checkHit(int page, int queue[], int occupied){
    for (int i = 0; i < occupied; i++)
    {
        if (queue[i] == page)
        {
```

```
53              return 1;
54          }
55      }
56      return 0;
57  }
58  void printFrame(int queue[], int occupied, int currentPage){
59      printf("%d\t", currentPage);
60      for (int i = 0; i < occupied; i++)
61      {
62          if (queue[i] != -1)
63              printf(" %d\t", queue[i]);
64          else
65              printf(" - \t");
66      }
67      printf("\n");
68  }
69  void lru(int pages, int frames, int incomingStream[]){
70      int queue[frames];
71      int occupied = 0;
72      int pagefault = 0;
73      int distance[frames];
74      printf("\nIncoming Frame 1 Frame 2 Frame 3\n");
75      for (int i = 0; i < pages; i++)
76      {
77          if (checkHit(incomingStream[i], queue, occupied))
78          {
79              printFrame(queue, occupied, incomingStream[i]);
80              continue;
81          }
82          if (occupied < frames)
83          {
84              queue[occupied] = incomingStream[i];
85              pagefault++;
86              occupied++;
87              printFrame(queue, occupied, incomingStream[i]);
88          }
89          else
90          {
91              int max = INT_MIN;
92              int index = 0;
93              for (int j = 0; j < frames; j++)
94              {
95
96                  distance[j] = 0;
97                  for (int k = i - 1; k >= 0; k--)
98                  {
99                      distance[j]++;
100                     if (queue[j] == incomingStream[k])
101                         break;
102                 }
103                 if (distance[j] > max)
104                 {
105                     max = distance[j];
106                     index = j;
```

```
                }
            }
            queue[index] = incomingStream[i]; // Replace LRU page pagefault++;
            printFrame(queue, occupied, incomingStream[i]);
        }
    }
    printf("Total Page Faults: %d\n", pagefault);
}
bool search(int key, int fr[], int size){
    for (int i = 0; i < size; i++)
        if (fr[i] == key)
            return true;
    return false;
}
int predict(int pg[], int fr[], int pn, int index, int fn){
    int res = -1, farthest = index;
    for (int i = 0; i < fn; i++)
    {
        int j;
        for (j = index; j < pn; j++)
        {
            if (fr[i] == pg[j])
            {
                if (j > farthest)
                {
                    farthest = j;
                    res = i;
                }
                break;
            }
        }
        if (j == pn)
            return i;
    }
    return (res == -1) ? 0 : res;
}

void optimal(int pg[], int pn, int fn){
    int fr[fn];
    int pagefault = 0;
    int occupied = 0;
    printf("\nIncoming Frame 1 Frame 2 Frame 3\n");
    for (int i = 0; i < pn; i++)
    {
        if (search(pg[i], fr, occupied))
        {
            printFrame(fr, occupied, pg[i]);
            continue;
        }
        if (occupied < fn)
        {
            fr[occupied] = pg[i];
            pagefault++;
```

```
            occupied++;
            printFrame(fr, occupied, pg[i]);
        }
        else
        {
            int j = predict(pg, fr, pn, i + 1, fn);
            fr[j] = pg[i];
            pagefault++;
            printFrame(fr, occupied, pg[i]);
        }
    }
    printf("Total Page Faults: %d\n", pagefault);
}
int main(){
    int pages;
    int frames;
    int op;
    printf("Enter the number of pages: ");
    scanf("%d", &pages);
    int arr[pages];
    printf("Enter the number of frames: ");
    scanf("%d", &frames);
    for (int i = 0; i < pages; i++)
    {
        printf("Enter the input %d: ", i + 1);

        scanf("%d", &arr[i]);
    }
    do
    {
        printf("\n================================\n");
        printf("1. FIFO\n2. LRU\n3. Optimal\n4. Exit\n");
        printf("Enter the option: ");
        scanf("%d", &op);
        switch (op)
        {
        case 1:
            printf("\nFirst In First Out\n");
            fifo(pages, frames, arr);
            break;
        case 2:
            printf("\nLeast Recently Used\n");
            lru(pages, frames, arr);
            break;
        case 3:
            printf("\nOptimal Page Replacement\n");
            optimal(arr, pages, frames);
            break;
        case 4:
            break;
        default:
            printf("Enter a valid option\n");
        }
```

```
213     } while (op != 4);
214     return 0;
215 }
```

```
tanishkhot@Tanishs-MacBook-Air A5 % gcc main.c
tanishkhot@Tanishs-MacBook-Air A5 % ./a.out
Enter the number of processes: 3
Enter the number of resources: 2
Enter the available resources for each resource type:
Resource 0: 3
Resource 1: 2
Enter the maximum demand matrix:
Process 0: 4 3
Process 1: 3 4
Process 2: 5 6
Enter the allocation matrix:
Process 0: 4 5
Process 1: 3 6
Process 2: 3 2
System is not in a safe state.
```