PROGRAM 10
Write a C program to simulate page replacement algorithms
a) FIFO
b) LRU
c) Optimal

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

void print_frames(int frame[], int capacity, int page_faults) {
    for (int i = 0; i < capacity; i++) {
        if (frame[i] == -1)
            printf("- ");
        else
            printf("%d ", frame[i]);
    }
    if (page_faults > 0)
        printf("PF No. %d", page_faults);
    printf("\n");
}

void fifo(int pages[], int n, int capacity) {
    int frame[capacity], index = 0, page_faults = 0;
    for (int i = 0; i < capacity; i++)
        frame[i] = -1;

    printf("FIFO Page Replacement Process:\n");
    for (int i = 0; i < n; i++) {
        int found = 0;
        for (int j = 0; j < capacity; j++) {
            if (frame[j] == pages[i]) {
                found = 1;
                break;
            }
        }
        if (!found) {
            frame[index] = pages[i];
            index = (index + 1) % capacity;
            page_faults++;
        }
        print_frames(frame, capacity, found ? 0 : page_faults);
    }
    printf("Total Page Faults using FIFO: %d\n\n", page_faults);
```

```
}

void lru(int pages[], int n, int capacity) {
    int frame[capacity], counter[capacity], time = 0, page_faults = 0;
    for (int i = 0; i < capacity; i++) {
        frame[i] = -1;
        counter[i] = 0;
    }

    printf("LRU Page Replacement Process:\n");
    for (int i = 0; i < n; i++) {
        int found = 0;
        for (int j = 0; j < capacity; j++) {
            if (frame[j] == pages[i]) {
                found = 1;
                counter[j] = time++;
                break;
            }
        }
        if (!found) {
            int min = INT_MAX, min_index = -1;
            for (int j = 0; j < capacity; j++) {
                if (counter[j] < min) {
                    min = counter[j];
                    min_index = j;
                }
            }
            frame[min_index] = pages[i];
            counter[min_index] = time++;
            page_faults++;
        }
        print_frames(frame, capacity, found ? 0 : page_faults);
    }
    printf("Total Page Faults using LRU: %d\n\n", page_faults);
}

void optimal(int pages[], int n, int capacity) {
    int frame[capacity], page_faults = 0;
    for (int i = 0; i < capacity; i++)
        frame[i] = -1;

    printf("Optimal Page Replacement Process:\n");
    for (int i = 0; i < n; i++) {
        int found = 0;
```

```
        for (int j = 0; j < capacity; j++) {
            if (frame[j] == pages[i]) {
                found = 1;
                break;
            }
        }
        if (!found) {
            int farthest = i + 1, index = -1;
            for (int j = 0; j < capacity; j++) {
                int k;
                for (k = i + 1; k < n; k++) {
                    if (frame[j] == pages[k])
                        break;
                }
                if (k > farthest) {
                    farthest = k;
                    index = j;
                }
            }
            if (index == -1) {
                for (int j = 0; j < capacity; j++) {
                    if (frame[j] == -1) {
                        index = j;
                        break;
                    }
                }
            }
            frame[index] = pages[i];
            page_faults++;
        }
        print_frames(frame, capacity, found ? 0 : page_faults);
    }
    printf("Total Page Faults using Optimal: %d\n\n", page_faults);
}

int main() {
    int n, capacity;
    printf("Enter the number of pages: ");
    scanf("%d", &n);
    int *pages = (int*)malloc(n * sizeof(int));
    printf("Enter the pages: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &pages[i]);
    printf("Enter the frame capacity: ");
```

```
    scanf("%d", &capacity);

    printf("\nPages: ");
    for (int i = 0; i < n; i++)
        printf("%d ", pages[i]);
    printf("\n\n");

    fifo(pages, n, capacity);
    lru(pages, n, capacity);
    optimal(pages, n, capacity);

    free(pages);
    return 0;
}
```

OUTPUT:

```
Enter the number of pages: 20
Enter the pages: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter the frame capacity: 3

Pages: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

FIFO Page Replacement Process:
7 - - PF No. 1
7 0 - PF No. 2
7 0 1 PF No. 3
2 0 1 PF No. 4
2 0 1
2 3 1 PF No. 5
2 3 0 PF No. 6
4 3 0 PF No. 7
4 2 0 PF No. 8
4 2 3 PF No. 9
0 2 3 PF No. 10
0 2 3
0 2 3
0 1 3 PF No. 11
0 1 2 PF No. 12
0 1 2
0 1 2
7 1 2 PF No. 13
7 0 2 PF No. 14
7 0 1 PF No. 15
Total Page Faults using FIFO: 15

LRU Page Replacement Process:
7 - - PF No. 1
0 - - PF No. 2
0 1 - PF No. 3
0 1 2 PF No. 4
0 1 2
0 3 2 PF No. 5
0 3 2
0 3 4 PF No. 6
0 2 4 PF No. 7
3 2 4 PF No. 8
3 2 0 PF No. 9
3 2 0
3 2 0
3 2 1 PF No. 10
3 2 1
0 2 1 PF No. 11
0 2 1
0 7 1 PF No. 12
0 7 1
0 7 1
Total Page Faults using LRU: 12
```

```
Optimal Page Replacement Process:
7 - - PF No. 1
7 0 - PF No. 2
7 0 1 PF No. 3
2 0 1 PF No. 4
2 0 1
2 0 3 PF No. 5
2 0 3
2 4 3 PF No. 6
2 4 3
2 4 3
2 0 3 PF No. 7
2 0 3
2 0 3
2 0 1 PF No. 8
2 0 1
2 0 1
2 0 1
7 0 1 PF No. 9
7 0 1
7 0 1
Total Page Faults using Optimal: 9


Process returned 0 (0x0)   execution time : 31.720 s
Press any key to continue.
```