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

#define MAX_PAGES 100
#define MAX_FRAMES 100
#define INFINITY INT_MAX

// CS420 | Assignment 4
// Group 20
//
// Drew Ridley, Ryan Ferguson

// Page information
typedef struct {
    int page_number;
    int time_loaded;
    bool is_loaded;
} Frame;

// shart emoji
void initializeFrames(Frame frames[], int num_frames);
int findFreeFrame(Frame frames[], int num_frames);
int findFIFOVictim(Frame frames[], int num_frames);
int findOptimalVictim(Frame frames[], int num_frames, int future_requests[], int
current_pos, int num_requests);
int findLRUVictim(Frame frames[], int num_frames, int current_time);
int findFrame(Frame frames[], int num_frames, int page);
void simulateFIFO(int requests[], int num_pages, int num_frames, int num_requests);
void simulateOptimal(int requests[], int num_pages, int num_frames, int
num_requests);
void simulateLRU(int requests[], int num_pages, int num_frames, int num_requests);

int main() {
    FILE *input = fopen("input.txt", "r");
    FILE *output = fopen("output.txt", "w");
    if (!input || !output) {
        printf("Error opening files!\n");
        return 1;
    }

    int num_pages, num_frames, num_requests;
    fscanf(input, "%d %d %d", &num_pages, &num_frames, &num_requests);

    int requests[MAX_PAGES];
    for (int i = 0; i < num_requests; i++) {
        fscanf(input, "%d", &requests[i]);
    }

    fclose(input);
    freopen("output.txt", "w", stdout);  // Redirect stdout to output file

    simulateFIFO(requests, num_pages, num_frames, num_requests);
    printf("\n");
    simulateOptimal(requests, num_pages, num_frames, num_requests);
    printf("\n");
    simulateLRU(requests, num_pages, num_frames, num_requests);
    fclose(output);
    return 0;
}

void initializeFrames(Frame frames[], int num_frames) {
    for (int i = 0; i < num_frames; i++) {
        frames[i].page_number = -1;
        frames[i].time_loaded = 0;
        frames[i].is_loaded = false;
    }
}

int findFreeFrame(Frame frames[], int num_frames) {
    for (int i = 0; i < num_frames; i++) {
        if (!frames[i].is_loaded) {
            return i;
        }
    }
    return -1;
}

int findFrame(Frame frames[], int num_frames, int page) {
    for (int i = 0; i < num_frames; i++) {
        if (frames[i].is_loaded && frames[i].page_number == page) {
            return i;
        }
    }
    return -1;
}

int findFIFOVictim(Frame frames[], int num_frames) {
    int victim = 0;
    int min_time = frames[0].time_loaded;

    for (int i = 1; i < num_frames; i++) {
        if (frames[i].time_loaded < min_time) {
            min_time = frames[i].time_loaded;
            victim = i;
        }
    }
    return victim;
}

int findOptimalVictim(Frame frames[], int num_frames, int future_requests[], int
current_pos, int num_requests) {
    int max_future = -1;
    int victim = 0;

    for (int i = 0; i < num_frames; i++) {
        int future_pos = num_requests;  // Default to end if not found
        for (int j = current_pos; j < num_requests; j++) {
            if (frames[i].page_number == future_requests[j]) {
                future_pos = j;
                break;
            }
        }
        if (future_pos > max_future) {
            max_future = future_pos;
            victim = i;
        }
    }
    return victim;
}

int findLRUVictim(Frame frames[], int num_frames, int current_time) {
    int victim = 0;
    int oldest_time = INFINITY;

    for (int i = 0; i < num_frames; i++) {
        if (frames[i].time_loaded < oldest_time) {
            oldest_time = frames[i].time_loaded;
            victim = i;
        }
    }
    return victim;
}

void simulateFIFO(int requests[], int num_pages, int num_frames, int num_requests)
{
    printf("FIFO\n");
    Frame frames[MAX_FRAMES];
    initializeFrames(frames, num_frames);
    int page_faults = 0;
    int time = 0;

    for (int i = 0; i < num_requests; i++) {
        int page = requests[i];
        int frame = findFrame(frames, num_frames, page);

        if (frame == -1) {  // Page fault
            page_faults++;
            int free_frame = findFreeFrame(frames, num_frames);

            if (free_frame != -1) {
                frames[free_frame].page_number = page;
                frames[free_frame].time_loaded = time++;
                frames[free_frame].is_loaded = true;
                printf("Page %d loaded into Frame %d\n", page, free_frame);
            } else {
                int victim = findFIFOVictim(frames, num_frames);
                printf("Page %d unloaded from Frame %d, Page %d loaded into Frame
%d\n",
                       frames[victim].page_number, victim, page, victim);
                frames[victim].page_number = page;
                frames[victim].time_loaded = time++;
            }
        } else {
            printf("Page %d already in Frame %d\n", page, frame);
        }
    }
    printf("%d page faults\n", page_faults);
}

void simulateOptimal(int requests[], int num_pages, int num_frames, int
num_requests) {
    printf("Optimal\n");
    Frame frames[MAX_FRAMES];
    initializeFrames(frames, num_frames);
    int page_faults = 0;
    int time = 0;

    for (int i = 0; i < num_requests; i++) {
        int page = requests[i];
        // oogah boogah
        int frame = findFrame(frames, num_frames, page);

        if (frame == -1) {  // We had a fault :(
            page_faults++;
            int free_frame = findFreeFrame(frames, num_frames);

            if (free_frame != -1) {
                frames[free_frame].page_number = page;
                frames[free_frame].time_loaded = time++;
                frames[free_frame].is_loaded = true;
                printf("Page %d loaded into Frame %d\n", page, free_frame);
            } else {
                //'victim' is a fun word :)
                int victim = findOptimalVictim(frames, num_frames, requests, i + 1,
num_requests);
                printf("Page %d unloaded from Frame %d, Page %d loaded into Frame
%d\n",
                       frames[victim].page_number, victim, page, victim);
                frames[victim].page_number = page;
                frames[victim].time_loaded = time++;
            }
        } else {
            printf("Page %d already in Frame %d\n", page, frame);
        }
    }
    printf("%d page faults\n", page_faults);
}

void simulateLRU(int requests[], int num_pages, int num_frames, int num_requests) {
    printf("LRU\n");
    Frame frames[MAX_FRAMES];
    initializeFrames(frames, num_frames);
    int page_faults = 0;
    int time = 0;

    for (int i = 0; i < num_requests; i++) {
        int page = requests[i];
        int frame = findFrame(frames, num_frames, page);

        if (frame == -1) {  // Page fault
            page_faults++;
            int free_frame = findFreeFrame(frames, num_frames);

            if (free_frame != -1) {
                //Update the page after finding free frame.
                frames[free_frame].page_number = page;
                frames[free_frame].time_loaded = time++;
                frames[free_frame].is_loaded = true;
                printf("Page %d loaded into Frame %d\n", page, free_frame);
            } else {
                int victim = findLRUVictim(frames, num_frames, time);
                //Unloaded.
                printf("Page %d unloaded from Frame %d, Page %d loaded into Frame 
%d\n",
                       frames[victim].page_number, victim, page, victim);
                frames[victim].page_number = page;
                frames[victim].time_loaded = time++;
            }
        } else {
            //its already there :)
            printf("Page %d already in Frame %d\n", page, frame);
            frames[frame].time_loaded = time++;  // Update access time for LRU
        }
    }
    printf("%d page faults\n", page_faults);
}

// nae nae
```