```
#include <stdio.h>

// We will manage the array as a queue
// (see Tuesday Question 1 p2v1d1)
// to keep track of unfulfilled requests
const int MAX = 100;
int size = 0, queue[MAX], mem, memTot;

// Is the queue full? This will never happen in this question
int isFull(){
    return size == MAX;
}

// No more requests left to fulfill
int isEmpty(){
    return size == 0;
}

void enqueue(int n){
    if(isFull())
        printf("Full\n");
    else
        queue[size++] = n;
}

// Careful: the dequeue function is effectively returning two integers
// One is returned explicitly as 0/1 depending on whether queue is
// empty or not. The other is returned implicity by changing the value
// of the variable n which is reflected in the function calling dequeue
int dequeue(int *n){
    int i;
    if(isEmpty())
        return 0;
    else{
        *n = queue[0];
        for(i = 0; i < MAX - 1; i++)
            queue[i] = queue[i+1];
        size--;
        return 1;
    }
}

int execute(){
    char c = getchar();
    int x;
    switch(c){
        case 'M':
            scanf("%d", &x);
            getchar();
            enqueue(x); // Store malloc request bytes as positive bytes
            return 1;
        case 'F':
            scanf("%d", &x);
            getchar();
            enqueue(-x); // Store free request bytes as negative bytes
            return 1;
        case 'X':
            getchar();
            if(!dequeue(&x))
                printf("NO MORE INSTRUCTIONS\n");
            else{
                // x now stores the number of bytes
                // If x > 0 => memory allocation request
                // If x < 0 => memory free request
                if(mem - x > memTot)
                    printf("SEGFAULT\n");
                else if(mem - x < 0)
                    printf("NOT ENOUGH MEMORY\n");
                else{
```

```
                    mem -= x;
                    printf("%d BYTES LEFT\n", mem);
                }
            }
            return 1;
        case '#': // Nothing more to execute
            return 0;
        default:
            printf("Error!!");
            return -1;
    }
}

int main(){
    scanf("%d\n", &memTot);
    mem = memTot;

    // ; is an empty statement
    while(execute());

    return 0;
}
```