```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define PAGESIZE (32)
#define PAS_FRAMES (256) //fit for unsigned char frame in PTE
#define PAS_SIZE (PAGESIZE * PAS_FRAMES) //32 * 256 = 8192 B
#define VAS_PAGES (64)
#define PTE_SIZE (4) //sizeof(pte) = 4B
#define PAGETABLE_FRAMES (VAS_PAGES * PTE_SIZE / PAGESIZE) //64 * 4 / 32 = 8 consecutive frames
#define PAGE_INVALID (0)
#define PAGE_VALID (1)
#define MAX_REFERENCES (256)
#define MAX_PROCESSES (10)

typedef struct {
    unsigned char frame; //allocated frame
    unsigned char vflag; //valid bit
    unsigned char ref; //reference bit
    unsigned char pad; //padding
} pte;

typedef struct {
    int pid;
    int ref_len; //less than 255
    unsigned char *references;
    pte *page_table;
    int page_faults;
    int ref_count;
} process;

unsigned char pas[PAS_SIZE];
int allocated_frame_count = 0;

// 실제 메모리에서 프레임 하나를 할당하고, 그 프레임 번호를 반환한다.
// 만약 모든 프레임이 이미 할당되어 있다면 -1을 반환한다.
int allocate_frame() {
    if (allocated_frame_count >= PAS_FRAMES)
        return -1; // 더 이상 할당할 프레임이 없으면 -1 반환
    return allocated_frame_count++; // 현재 인덱스를 반환하고, 다음 인덱스로 증가
}

// 페이지 테이블에 필요한 프레임들을 연속적으로 할당하고,
// 페이지 테이블의 시작 주소를 pt_out에 저장한다.
// 할당에 실패하면 -1, 성공하면 0을 반환한다.
int allocate_pagetable_frames(pte **page_table_ptr) {
    int base = allocate_frame(); // 첫 프레임 할당
    if (base == -1 || base + PAGETABLE_FRAMES - 1 >= PAS_FRAMES)
        return -1; // 할당 불가 시 -1 반환

    // 페이지 테이블이 차지하는 나머지 프레임들도 할당
    for (int i = 1; i < PAGETABLE_FRAMES; i++) {
        if (allocate_frame() == -1)
            return -1; // 중간에 할당 실패 시 -1 반환
    }

    *page_table_ptr = (pte *)&pas[base * PAGESIZE]; // 페이지 테이블의 시작 주소를 포인터에 저장
    memset(*page_table_ptr, 0, PAGESIZE * PAGETABLE_FRAMES); // 페이지 테이블 메모리 0으로 초기화
    return 0;
}

// 파일 포인터 fp에서 프로세스 정보를 읽어와 process 구조체에 저장한다.
// 1) 프로세스 ID와 참조 길이를 읽고,
// 2) 참조할 페이지 번호 배열을 동적 할당하여 읽어온다.
// 3) 페이지 폴트/참조 횟수 초기화, 페이지 테이블 프레임 할당까지 수행한다.
// 성공 시 1, 입력 끝(EOF) 시 0, 메모리 부족 등 실패 시 -1 반환
int load_process(FILE *fp, process *proc) {
    if (fread(&proc->pid, sizeof(int), 1, fp) != 1) // 프로세스 ID 읽기
        return 0;
    if (fread(&proc->ref_len, sizeof(int), 1, fp) != 1) // 참조 길이 읽기
        return 0;
    proc->references = malloc(proc->ref_len); // 참조 배열 동적 할당
    /*
    if (!proc->references) {
        perror("malloc");
        exit(1);
    }
    */
    if (fread(proc->references, 1, proc->ref_len, fp) != proc->ref_len) // 참조 배열 데이터 읽기
        return 0;

    // 읽은 참조 정보 출력
    printf("%d %d\n", proc->pid, proc->ref_len);
    for (int i = 0; i < proc->ref_len; i++) {
        printf("%02d ", proc->references[i]);
    }
    printf("\n");

    proc->page_faults = 0;
    proc->ref_count = 0;

    // 페이지 테이블 프레임 할당
    if (allocate_pagetable_frames(&proc->page_table) == -1)
        return -1;

    return 1;
}

// 여러 프로세스의 페이지 참조 시퀀스를 시뮬레이션한다.
// 각 프로세스가 자신의 참조 배열을 순서대로 접근하며,
// 페이지 폴트 발생 시 프레임을 할당하고, 이미 할당된 경우 참조 횟수만 증가시킨다.
// 모든 프로세스의 참조가 끝날 때까지 반복한다.
// simulate() 수정: 1-level page table 기준
void simulate(process *procs, int proc_count) {
    printf("simulate() start\n");

    int total_references_to_process = 0;
    for (int i = 0; i < proc_count; i++) {
        total_references_to_process += procs[i].ref_len;
    }

    int processed_refs = 0;
    while (processed_refs < total_references_to_process) {
        for (int i = 0; i < proc_count; i++) {
            if (procs[i].ref_count < procs[i].ref_len) {
                int page_num = procs[i].references[procs[i].ref_count];
                pte *pte_entry = &procs[i].page_table[page_num];

                printf("[PID %02d IDX:%03d] %03d Page access: ", procs[i].pid, procs[i].ref_count, page_num);

                if (pte_entry->vflag == PAGE_INVALID) {
                    int frame = allocate_frame();
                    if (frame == -1) {
                        printf("Out of memory!!\n");
                        printf("simulate() end\n");
                        return;
                    }
                    pte_entry->frame = frame;
                    pte_entry->vflag = PAGE_VALID;
                    pte_entry->ref = 1;
                    printf("PF -> Allocated Frame %03d\n", frame);
                    procs[i].page_faults++;
                } else {
                    pte_entry->ref++;
                    printf("Frame %03d\n", pte_entry->frame);
                }
                procs[i].ref_count++;
                processed_refs++;
            }
        }
    }
    printf("simulate() end\n");
}
```

```c

// print_page_tables() 수정
void print_page_tables(process *procs, int proc_count) {
    int total_page_faults = 0;
    int total_references = 0;

    for (int i = 0; i < proc_count; i++) {
        process *proc = &procs[i];

        int proc_allocated_frames = PAGETABLE_FRAMES;
        for (int p = 0; p < VAS_PAGES; p++) {
            if (proc->page_table[p].vflag == PAGE_VALID) {
                proc_allocated_frames++;
            }
        }

        printf("** Process %03d: Allocated Frames=%03d PageFaults/References=%03d/%03d\n",
               proc->pid, proc_allocated_frames, proc->page_faults, proc->ref_count);

        for (int p = 0; p < VAS_PAGES; p++) {
            if (proc->page_table[p].vflag == PAGE_VALID) {
                printf("[PAGE] %03d -> [FRAME] %03d REF=%03d\n",
                       p, proc->page_table[p].frame, proc->page_table[p].ref);
            }
        }
        total_page_faults += proc->page_faults;
        total_references += proc->ref_count;
    }

    printf("Total: Allocated Frames=%03d Page Faults/References=%03d/%03d\n",
           allocated_frame_count, total_page_faults, total_references);
}




// 메인 함수: 표준 입력에서 프로세스 정보를 읽고, 시뮬레이션을 수행한 뒤 결과를 출력한다.
// 1) 프로세스 정보 입력
// 2) 시뮬레이션 실행
// 3) 결과 출력 및 동적 메모리 해제
int main() {
    process procs[MAX_PROCESSES]; // 프로세스 배열
    int count = 0;

    printf("load_process() start\n");
    // 최대 MAX_PROCESSES개까지 프로세스 정보를 입력받음
    while (count < MAX_PROCESSES) {
        int ret = load_process(stdin, &procs[count]); // 표준입력에서 프로세스 정보 읽기
        if (ret == 0) // 더 이상 읽을 프로세스 없음(EOF)
            break;
        if (ret == -1) {
            printf("Out of memory!!\n"); // 프로세스 로드하거나 메모리 부족
            return 1;
        }
        count++;
    }
    printf("load_process() end\n");

    simulate(procs, count); // 시뮬레이션 실행
    print_page_tables(procs, count); // 결과 출력

    // 동적 할당된 참조 배열 해제
    for (int i = 0; i < count; i++) {
        free(procs[i].references);
    }

    return 0;
}
```