#include <assert.h>

#include <limits.h>

#include <math.h>

#include <stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char\* readline();

typedef struct DoublyLinkedListNode DoublyLinkedListNode;

typedef struct DoublyLinkedList DoublyLinkedList;

struct DoublyLinkedListNode {

    int data;

    DoublyLinkedListNode\* next;

    DoublyLinkedListNode\* prev;

};

struct DoublyLinkedList {

    DoublyLinkedListNode\* head;

    DoublyLinkedListNode\* tail;

};

DoublyLinkedListNode\* create\_doubly\_linked\_list\_node(int node\_data) {

    DoublyLinkedListNode\* node = malloc(sizeof(DoublyLinkedListNode));

    node->data = node\_data;

    node->next = NULL;

    node->prev = NULL;

    return node;

}

void insert\_node\_into\_doubly\_linked\_list(DoublyLinkedList\*\* doubly\_linked\_list, int node\_data) {

    DoublyLinkedListNode\* node = create\_doubly\_linked\_list\_node(node\_data);

    if (!(\*doubly\_linked\_list)->head) {

        (\*doubly\_linked\_list)->head = node;

    } else {

        (\*doubly\_linked\_list)->tail->next = node;

        node->prev = (\*doubly\_linked\_list)->tail;

    }

    (\*doubly\_linked\_list)->tail = node;

}

void print\_doubly\_linked\_list(DoublyLinkedListNode\* node, char\* sep, FILE\* fptr) {

    while (node) {

        fprintf(fptr, "%d", node->data);

        node = node->next;

        if (node) {

            fprintf(fptr, "%s", sep);

        }

    }

}

void free\_doubly\_linked\_list(DoublyLinkedListNode\* node) {

    while (node) {

        DoublyLinkedListNode\* temp = node;

        node = node->next;

        free(temp);

    }

}

/\*

 \* Complete the 'reverse' function below.

 \*

 \* The function is expected to return an INTEGER\_DOUBLY\_LINKED\_LIST.

 \* The function accepts INTEGER\_DOUBLY\_LINKED\_LIST llist as parameter.

 \*/

/\*

 \* For your reference:

 \*

 \* DoublyLinkedListNode {

 \*     int data;

 \*     DoublyLinkedListNode\* next;

 \*     DoublyLinkedListNode\* prev;

 \* };

 \*

 \*/

DoublyLinkedListNode\* reverse(DoublyLinkedListNode\* llist) {

DoublyLinkedListNode\* curr=llist;

DoublyLinkedListNode\* temp=NULL;

while(curr!=NULL){

    temp=curr->prev;

    curr->prev=curr->next;

    curr->next=temp;

    llist=curr;

    curr=curr->prev;

}

return llist;

}

int main()

{

    FILE\* fptr = fopen(getenv("OUTPUT\_PATH"), "w");

    char\* t\_endptr;

    char\* t\_str = readline();

    int t = strtol(t\_str, &t\_endptr, 10);

    if (t\_endptr == t\_str || \*t\_endptr != '\0') { exit(EXIT\_FAILURE); }

    for (int t\_itr = 0; t\_itr < t; t\_itr++) {

        DoublyLinkedList\* llist = malloc(sizeof(DoublyLinkedList));

        llist->head = NULL;

        llist->tail = NULL;

        char\* llist\_count\_endptr;

        char\* llist\_count\_str = readline();

        int llist\_count = strtol(llist\_count\_str, &llist\_count\_endptr, 10);

        if (llist\_count\_endptr == llist\_count\_str || \*llist\_count\_endptr != '\0') { exit(EXIT\_FAILURE); }

        for (int i = 0; i < llist\_count; i++) {

            char\* llist\_item\_endptr;

            char\* llist\_item\_str = readline();

            int llist\_item = strtol(llist\_item\_str, &llist\_item\_endptr, 10);

            if (llist\_item\_endptr == llist\_item\_str || \*llist\_item\_endptr != '\0') { exit(EXIT\_FAILURE); }

            insert\_node\_into\_doubly\_linked\_list(&llist, llist\_item);

        }

        DoublyLinkedListNode\* llist1 = reverse(llist->head);

        char \*sep = " ";

        print\_doubly\_linked\_list(llist1, sep, fptr);

        fprintf(fptr, "\n");

        free\_doubly\_linked\_list(llist1);

    }

    fclose(fptr);

    return 0;

}

char\* readline() {

    size\_t alloc\_length = 1024;

    size\_t data\_length = 0;

    char\* data = malloc(alloc\_length);

    while (true) {

        char\* cursor = data + data\_length;

        char\* line = fgets(cursor, alloc\_length - data\_length, stdin);

        if (!line) { break; }

        data\_length += strlen(cursor);

        if (data\_length < alloc\_length - 1 || data[data\_length - 1] == '\n') { break; }

        size\_t new\_length = alloc\_length << 1;

        data = realloc(data, new\_length);

        if (!data) { break; }

        alloc\_length = new\_length;

    }

    if (data[data\_length - 1] == '\n') {

        data[data\_length - 1] = '\0';

    }

    data = realloc(data, data\_length);

    return data;

}