A+


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

int ga_set(struct ga_s *ga, unsigned int index, int val){
    int i, *new = NULL;
    if(index < ga->ga_size){
        *(ga->ga_elements + index) = val;
        return 0;
    }
    else{
        new = malloc(sizeof(int) * (index+1));
        if(new == NULL){
            return -1;
        }

        for(i=0; i<ga->ga_size; i++){
            *(new + i) = *(ga->ga_elements + i);
        }
        *(new + index) = val;
        ga->ga_elements = new;
        ga->ga_size = index+1;
        return 0;
    }
    return -1;
}

int ga_get(struct ga_s *ga, unsigned int index, int *val){
    if(index >= ga->ga_size){
        return -1;
    }
    *val = *(ga->ga_elements + index);
    return 0;
}

int ga_new(struct ga_s *ga){
    ga->ga_size = 1;
    ga->ga_elements = malloc(sizeof(int));
    if(ga->ga_elements == NULL){
        return -1;
    }
    return 0;
}

int ga_del(struct ga_s *ga){
    free(ga->ga_elements);
    ga->ga_size = 0;
    return 0;
}
```

factorisation possible

free(ga->ga_elements);

Incohérent : si on alloue un entier, la taille devrait être de 1.

→ inutile : après ga_del(), ga doit être considéré comme invalide.

```c
/*
  Validation (Ã  minima) de la bibliothÃ¨que de gestion de
  tableaux grandissants.

  Principe :
  - lecture d'une sÃ©quenece de valeurs entiÃ¨res sur l'entrÃ©e standard
    et rangement dans un tableau grandissant
  - tri Ã  bulle de ce tableau grandissant
  - affichage des Ã©lÃ©ments ce tableau

*/

#include <stdio.h>
#include <stdlib.h>
#include "libga.h"

int
main(void) {
    struct ga_s tab;            /* le tableau grandissant */
    int val;
    int i, j;
    int count;

    /* Initialisation et allocation mÃ©moire du tableau grandissant */
    ga_new(&tab);

    /* Lecture d'entiers, un par ligne, et mÃ©morisation dans le
       tableau grandissant */
    count = 0;
    while (scanf("%u\n", &val) == 1) {
        ga_set(&tab, count, val);
        count++;
    }

    /* Tri Ã  bulles du tableau */
    for (i = 0; i < count - 1; i++) {
        for (j = i + 1; j < count; j++) {
            int vali, valj;         /* valeurs aux indices i et j du tableau */

            ga_get(&tab, i, &vali); /* vali = tab[i] */
            ga_get(&tab, j, &valj); /* valj = tab[j] */

            if (vali > valj) {          /*  Ã©change */
                ga_set(&tab, i, valj); /* tab[i] = valj */
                ga_set(&tab, j, vali); /* tab[j] = vali */
            }
        }
    }

    /* Affichage du tableau triÃ©, une valeur par ligne */
    for (i = 0; i < count; i++) {
        ga_get(&tab, i, &val);
        printf("%u\n", val);
    }

    /* LibÃ©ration mÃ©moire */
    ga_del(&tab);

    exit(EXIT_SUCCESS);
}
```

```
#ifndef LIBGA_H
#define LIBGA_H

struct ga_s {
    unsigned int ga_size;              /* nombre d'Ã©lÃ©ments allouÃ©s */
    int *ga_elements;                  /* les Ã©lÃ©ments */
};


int ga_set(struct ga_s *ga, unsigned int index, int val);
int ga_get(struct ga_s *ga, unsigned int index, int *val);

int ga_new(struct ga_s *ga);
int ga_del(struct ga_s *ga);

#endif
```

```c
#include "files_entiers.h"

/*Q4*/
struct ififo_s *ififo_new(){
    struct ififo_s *newHead = NULL;
    newHead = malloc(sizeof(struct ififo_s));
    if(newHead == NULL){
        return NULL;
    }
    newHead->debut = NULL;
    newHead->fin = NULL;
    return newHead;
}

/*Q5*/
int ififo_is_empty(struct ififo_s *f){
    if(f == NULL){
        return -1;
    }
    return f->debut == NULL && f->fin == NULL;
}

/*Q6*/
int ififo_enqueue(struct ififo_s *f, int val){
    struct ififo_node_s *data;

    data = malloc(sizeof(struct ififo_node_s));
    if(data == NULL){
        return -1;
    }
    data->valeur = val;
    data->suivant = NULL;
    if(ififo_is_empty(f)){
        f->fin = f->debut = data;
    }
    else{
        f->debut->suivant= data;
        f->debut=data;
    }
    return 0;
}

/*Q7*/
int ififo_dequeue(struct ififo_s *f, int *val){
    struct ififo_node_s *data;
    if(ififo_is_empty(f)){
        return -1;
    }
    data = f->fin;
    *val = data->valeur;
    f->fin = data->suivant;
    free(data);
    return 0;
}

/*Q8*/
int ififo_head(const struct ififo_s *f){
    if(f->debut == NULL){
        return -1;
    }
    return f->debut->valeur;
}

/*Q9*/
int ififo_apply(struct ififo_s *f, func_t *fn){
    struct ififo_node_s *newhead = f->fin;
    while(newhead != NULL){
        fn(newhead->valeur);
        newhead = newhead->suivant;
```



```c
    }
    return 0;
}

/*Q10*/
void ififo_del(struct ififo_s *f){
    struct ififo_node_s *head = f->fin;
    while(head != NULL){
        free(head);
        f->fin = f->fin->suivant;
        head = f->fin;
    }
    free(f);
}

void print_int(int i){
    printf("âM-^FM-^R %d ", i);
}
```

Bien

de mais f->début est en fait la fin de file. Le nommage devrait refléter la fonction des objets.

Même remarque sur le nommage.

```c
#ifndef FILES_ENTIERS
#define FILES_ENTIERS
#include <stdio.h>
#include <stdlib.h>

/*Q1*/
struct ififo_node_s {
    int valeur;
    struct ififo_node_s * suivant;
};

/*Q2*/
struct ififo_s{
    struct ififo_node_s * debut;
    struct ififo_node_s * fin;
};

/*Q3
Comment est reprÃ©sentÃ©e la file vide ?
- la file est vide quand debut == NULL / debut = fin = NULL
*/

struct ififo_s *ififo_new();
int ififo_is_empty(struct ififo_s *f);
int ififo_enqueue(struct ififo_s *f, int val);
int ififo_dequeue(struct ififo_s *f, int *val);
int ififo_head(const struct ififo_s *f);
typedef void (func_t)(int);
int ififo_apply(struct ififo_s *f, func_t *fn);
void ififo_del(struct ififo_s *f);
void print_int(int i);

#endif
```

```c
#include "files_entiers.h"

/*
> make main_files_entiers
*/

void test_fifo_int(){
    struct ififo_s *fifo;
    int i;

    fifo = ififo_new();

    ififo_enqueue(fifo, 12);   /* âM-^FM-^R 12 âM-^FM-^R */
    ififo_enqueue(fifo, 13);   /* âM-^FM-^R 13 âM-^FM-^R 12 âM-^FM-^R */

    ififo_apply(fifo, print_int); putchar('\n');

    ififo_enqueue(fifo, 14);   /* âM-^FM-^R 14 âM-^FM-^R 13 âM-^FM-^R 12 âM-^F
M-^R */
    ififo_dequeue(fifo, &i);   /* 12 & âM-^FM-^R 14 âM-^FM-^R 13 âM-^FM-^R */

    printf("%d\n", i);
    ififo_apply(fifo, print_int); putchar('\n');

    ififo_dequeue(fifo, &i);   /* 13 & âM-^FM-^R 14 âM-^FM-^R */
    ififo_dequeue(fifo, &i);   /* 14 & âM-^FM-^R âM-^FM-^R */
    ififo_apply(fifo, print_int); putchar('\n');

    ififo_del(fifo);
}

int main(void){
    test_fifo_int();
    return 0;
}
```