A+

Printed by Pierre Tirilly

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


int
ga_set(struct ga_s *ga, unsigned int index, int val)
{
    if (index < ga -> ga_size){
        ga -> ga_elements[index] = val;
    }

    else{
        int i;
        int * new_mem = malloc(sizeof(int) * (index*2));
        if(new_mem == NULL){
        return -1;
        }

        for(i = 0; i < ga -> ga_size; i++){
            new_mem[i] = ga -> ga_elements[i];
        }

        free(ga -> ga_elements);
        ga -> ga_elements = new_mem;
        ga -> ga_elements[index] = val;
        ga -> ga_size = index * 2;
    }
    return 1;

}

int
ga_get(struct ga_s *ga, unsigned int index, int *val)
{
    *val = ga -> ga_elements[index];
    return 1;
}

int
ga_new(struct ga_s *ga)
{
    ga -> ga_size = 10;
    ga -> ga_elements = malloc(sizeof(int) * ga -> ga_size);
    if(ga -> ga_elements == NULL){
        return -1;
    }
    return  1;
}

int
ga_del(struct ga_s *ga)
{
    ga -> ga_size = 0;
    free(ga -> ga_elements);
    return 1;
}
```

pas d'espace ici (style).

bien

indentation

factoriser

Vérifier la validité de l'indice (index < ga->ga_size).

Eviter les "nombres magiques" : utiliser une constante ou #define.

Inutile : après l'appel de ga_del(), ga doit être considéré comme invalide.

```
#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
/*
  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);
}
```

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

/* QUESTION 1 : */

typedef struct ififo_node_s ififo_node_s;
struct ififo_node_s {
    int nombre;
    ififo_node_s *noeud;
};

/* QUESTION 2 : */

typedef struct ififo_s ififo_s;
struct ififo_s {
    ififo_node_s *suivant;
    ififo_node_s *dernier;
};

/* QUESTION 3 :

La file vide est reprÃ©sentÃ©e par ififo_s avec comme pointeur NULL suivant.

*/

/* QUESTION 4 : */

ififo_s *ififo_new(){

    ififo_node_s *newnoeud = NULL;
    ififo_s *files = malloc(sizeof(*files));

    if(files == NULL){
        return NULL;
    }

    files -> suivant = newnoeud;
    files -> dernier = newnoeud;

    return files;
}

/* QUESTION 5 : */

int ififo_is_empty(struct ififo_s *f){

    return ((f -> suivant == NULL) && (f -> dernier == NULL));
}

/* QUESTION 6 : */

int ififo_enqueue(struct ififo_s *f, int nb){

    ififo_node_s *new = malloc(sizeof(*new));

    if (f == NULL || new == NULL){
        return -1;
    }

    new->nombre = nb;
    if (ififo_is_empty(f)){
        f -> suivant = new;
        f -> dernier = new;
    }
    else{
        f -> dernier -> noeud = new;
        f -> dernier = new;
    }
```



```c
    new -> noeud = NULL;
    return 0;
}

/* QUESTION 7 : */

int ififo_dequeue(struct ififo_s *f, int *nb){

    if (ififo_is_empty(f)){
        return -1;
    }
    ififo_node_s *new = f -> suivant;
    if (f -> suivant == f -> dernier){
        ififo_node_s *new2 = NULL;

        *nb = f -> suivant -> nombre;
        f -> suivant = f -> dernier = new2;
    }
    else {
        *nb = new -> nombre;
        f -> suivant = new -> noeud;
    }

    free(new);

    return 0;
}

/* QUESTION 8 : */

int ififo_head(const struct ififo_s *f){

    return f -> suivant -> nombre;
}

/* QUESTION 9 : */

typedef void(func_t)(int);

int ififo_apply(struct ififo_s *f, func_t *fn){

    ififo_node_s *apply = f -> suivant;

    while (apply != NULL){
        fn(apply -> nombre);
        apply = apply -> noeud;
    }
    return 0;
}

/* QUESTION 10 : */

void ififo_del(struct ififo_s *f){

    ififo_node_s *new = f -> suivant;
    ififo_node_s *del;


    while(new != NULL){
        del = new;
        new = new -> noeud;
        free(del);
    }
    new = NULL;
    free(f);
}
```

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

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-^FM-^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(){
    test_fifo_int();
    return 0;
}
```