

03 oct. 17 9:41

list.h

Page 1/1

```
/* Structure of an element of a linked list */
typedef struct s_list {
    int value;
    struct s_list* next;
} list_elem_t;

/* Prototypes */
int insert_head(list_elem_t** l, int value);
int insert_tail(list_elem_t** l, int value);
int remove_element(list_elem_t** ppl, int value);
list_elem_t* get_tail(list_elem_t* l);
void reverse_list(list_elem_t** l);
list_elem_t* find_element(list_elem_t* l, int index);
int list_size(list_elem_t* l);
```

03 oct. 17 9:42

list.c

Page 1/3

```

/*****
 * L3Informatique
 *
 * TP linked lists
 *
 * Group      :
 * Name 1 :
 * Name 2 :
 *****/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "list.h"

int nb_malloc = 0;      // Global counter of the number of allocations

/*
 * SYNOPSIS :
 * list_elem_t * create_element(int value)
 * DESCRIPTION :
 * creates a new element, whose field next is initialized with NULL and the fi
eld
 * value is initialized with the integer passed as argument.
 * PARAMETERS :
 * value : value of the element
 * RESULT :
 * NULL in case of error, otherwise a pointer over one struture of type list_
elem_t
 */
static list_elem_t *
create_element (int value)
{
    list_elem_t * newelt = malloc (sizeof (list_elem_t));
    if (newelt != NULL) {
        ++nb_malloc;
        newelt->value = value;
        newelt->next = NULL;
    }
    return newelt;
}

/*
 * SYNOPSIS :
 * void free_element(list_elem_t * l)
 * DESCRIPTION :
 * frees an element of the list.
 * PARAMETERS :
 * l : the pointer of the element to be freed
 * RESULT :
 * nothing
 */
/*
static void
free_element (list_elem_t * l)
{
    --nb_malloc;
    free (l);
}
*/
/*
 * SYNOPSIS :

```

03 oct. 17 9:42

list.c

Page

```

 * int insert_head(list_elem_t * * l, int value)
 * DESCRIPTION :
 * Insert an element at the head of the list
 * *l points the new head of the list.
 * PARAMETERS :
 * list_elem_t * * l : pointer to the pointer of the head of the list
 * int value : value of the element that is added
 * RESULT :
 * 0 in case of successful insertion
 * -1 if the insertion is impossible
 */
int
insert_head (list_elem_t * * l, int value)
{
    if (l == NULL) { return -1; }
    list_elem_t * new_elt = create_element (value);
    if (new_elt == NULL) { return -1; }

    new_elt->next = *l;
    *l = new_elt;
    return 0;
}

/*
 * SYNOPSIS :
 * int insert_tail(list_elem_t * * l, int value)
 * DESCRIPTION :
 * Insert an element at the tail of the list (* l points the head of the
 * PARAMETERS :
 * list_elem_t * * l : pointer to the pointer of the head of the list
 * int value : value of the element that is added
 * RESULT :
 * 0 in case of successful insertion
 * -1 if the insertion is impossible
 */
int
insert_tail(list_elem_t * * l, int value) {
    // TO BE COMPLETED
    return -1;
}

/*
 * SYNOPSIS :
 * list_elem_t * find_element(list_elem_t * l, int index)
 * DESCRIPTION :
 * Return a pointer of the element at the position n°i of the list
 * (The first element is situated at the position 0).
 * PARAMETERS :
 * int index : position of the element to be found
 * list_elem_t * l : pointer of the head of the list
 * RESULTAT :
 * - a pointer to the element of the list in cas of success
 * - NULL in case of error
 */
list_elem_t *
find_element(list_elem_t * l, int index) {
    // TO BE COMPLETED
    return NULL;
}

/*
 * SYNOPSIS :

```

03 oct. 17 9:42

list.c

Page 3/3

```

*   int remove_element(list_elem_t * * ppl, int value)
*   DESCRIPTION :
*       Removes from the list the first element with a value equal to the argument
value and
*       frees the memory space reserved by this element.
*       Attention : Depending on the position, the head of the list may need to be
modified
*   PARAMETERS :
*       list_elem_t * * ppl : pointer to the pointer of the head of the list
*       int value : value to be removed from the list
*   RESULT :
*       0 in case of success
*       -1 in case of error
*/
int
remove_element(list_elem_t * * ppl, int value) {
    // TO BE COMPLETED
    return -1;
}

/*
*   SYNOPSIS :
*       void reverse_list(list_elem_t * * l)
*   DESCRIPTION :
*       Modifies the list by inversing the order of the elements.
*       So the 1st element becomes the last element, the 2nd becomes the before las
t element, etc.)
*   PARAMETRES :
*       list_elem_t * * l : pointer to the pointer of the head of the list
*   RESULTAT :
*       aucun
*/
void
reverse_list(list_elem_t * * l) {
    // TO BE COMPLETED
}

```

12 oct. 17 8:22

test_list.c

Page 1/3

```

/*****
 * L3Informatique
 *
 * TP linked lists
 *
 * Group      :
 * Name 1 :
 * Name 2 :
 *
 *****/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <termios.h>
#include <unistd.h>
#include "list.h"

/* Compute the number of elements of the list */
int
list_size(list_elem_t * p_list)
{
    int nb = 0;
    while (p_list != NULL) {
        nb += 1;
        p_list = p_list->next;
    }
    return nb;
}

/* Print the elements of the list */
void
print_list(list_elem_t* p_list) {
    list_elem_t * pl = p_list;
    printf("The list contains %d element(s)\n", list_size(p_list));
    while(pl != NULL) {
        printf("[%d]", pl->value);
        pl = pl->next;
        if (pl != NULL) {
            printf("->");
        }
    }
}

/* Compute the number of memory allocations */
extern int nb_malloc;

int
main(int argc, char **argv)
{
    list_elem_t * la_liste = NULL;      // The pointer to the head of the list
    char menu[] =
        "\n Program of chained list\n\n\
        " 'h/t' : Insert an element to the head/tail of the list\n\n\
        " 'f' : search of a list element\n\n\
        " 's' : suppression of a list element\n\n\
        " 'r' : reverse the order of the list elements\n\n\
        " 'x' : exit the program\n\n\
        " What is your choice ?";
    int choice=0;                        // choice from the menu
    int value=0;                         // inserted value

    printf("%s", menu);

```

12 oct. 17 8:22

test_list.c

Page

```

fflush(stdout);

while(1) {
    fflush(stdin);
    choice = getchar();
    printf("\n");

    switch(choice) {
        case 'H' :
        case 'h' :
            printf("Value of the new element ? ");
            scanf("%d",&value);
            if (insert_head(&la_liste,value)!=0) {
                printf("Error: impossible to add the element %d\n", value);
            };
            break;

        case 'T' :
        case 't' :
            // TO BE COMPLETED
            break;

        case 'F' :
        case 'f' :
            // TO BE COMPLETED
            break;

        case 's' :
        case 'S' :
            // TO BE COMPLETED
            break;

        case 'r' :
        case 'R' :
            // TO BE COMPLETED
            break;

        case 'x' :
        case 'X' :
            return 0;

        default:
            break;
    }
    print_list(la_liste);

    if (nb_malloc!=list_size(la_liste)) {
        printf("\n Attention : There is a leak memory in your program !\n");
        printf("The list has %d elements, but %d elements are allocated in memory!\n",
            list_size(la_liste),nb_malloc);
    }
    getchar(); // to consume a return character and avoid double display of
menu
    printf("%s\n",menu);
}
return 0;
}

```

12 oct. 17 8:22

test_list.c

Page 3/3