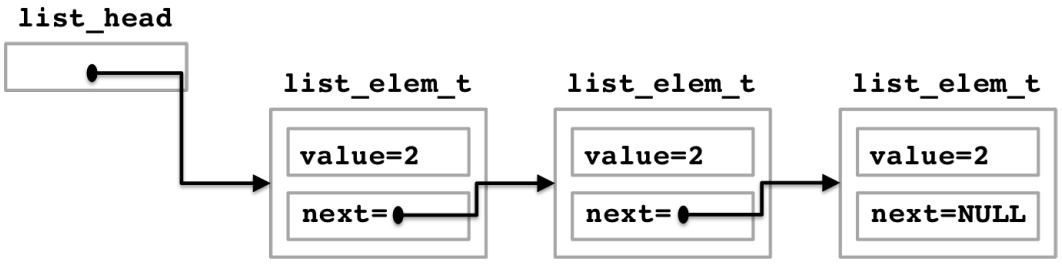
TP6: Linked lists

1 Getting started

The object of this lab is the implementation of linked lists in C language. The codes for this lab are in the directory **/share/l3info/CUnix/tp6**. The list type is represented using a **list_elem_t** structure whose definition is given as:

```
typedef struct s_list {
        int   value;              //value of the element
        struct s_list * next;     //pointer to the next element
} list_elem_t;
```

This representation is illustrated in Fig.1:



2 Implementation of linked lists

The file **list.c** provides the following functions (for a more detailed description, please refer to the source code of the **list.c** file. The functions **insert_head**, **create_element**, and **free_element** have been already implemented.

- **list_elem_t * create_element (int val):** creates a new element, whose field **next** is set to NULL and field **value** is set to the integer value **val** passed as a parameter. The function returns NULL on failure, otherwise the pointer to the new element.
- **void free_element (list_elem_t * l):** frees the memory allocated to an element
- **int insert_head (list_elem_t * * l, int val):** inserts an element at the head of the list (*** l** points to the head of the list). At the end of the execution, *** l** points to the new head of the list. The function returns 0 on success, -1 on failure.

- **int insert_tail (list_elem_t * * l, int val):** inserts an element at the tail of the list (*** l** points to the head of the list). The function returns 0 on success, -1 on failure.
- **list_elem_t * find_element (list_elem_t * l, int pos):** returns the pointer to the position **pos** of the list (the first element is at position 0). The function returns this pointer on success, NULL on failure.
- **int remove_element (list_elem_t * * l, int val):** removes from the list the first element found in the list with a value equal to the value **val** passed as parameter. *** l** points to the head of the list. It frees the memory space dedicated to this element. The function returns 0 on success (an element with a value **val** has been found in the list), -1 on failure (no element with value **val** exists in the list)
- **void reverse_list (list_elem_t * * l):** modifies the list by reversing the order of its elements (the first becomes the last, the second the before last , etc.). ATTENTION: Be careful not to create a "memory leak".

3 Testing of linked lists

You also have a **test_list** test program located in the subdirectory **./bin**. To launch it, you just need to write **./bin/test_list**. This program allows (via keyboard commands) to add / remove items to/from the list. The operations over the list are summarized below:

- **'h'**: insertion to the head of the list of a value given by the keyboard
- **'t'**: insertion to the tail of the list of a value given by the keyboard
- **'f'**: search (and display) the i^{th} element of the list
- **'s'**: delete the first element equal to a given value
- **'r'**: reverse the list
- **'x'**: end of the program

The content of the list is displayed at the end of each operation in the form: [value_1] $\rightarrow$ [value_2] $\rightarrow \ldots \rightarrow$ [value_n]
In addition, the program reports any memory leaks caused by any function.

4 Exercise

1. Complete the **list.c** file to implement the functions of the linked lists and the **test_list.c** file which calls the corresponding functions. Take care the following remarks:
   - Read the specification of each function (detailed in the **list.c** file) before starting coding!
   - Each of these functions has to go through the list **only once**, either directly or through the use of other functions.

- When manipulating a pointer of a structure, to access its fields we have to use the '$\rightarrow$' operator and not by the '**.**' operator.
- Special care must be taken in the management of pointers. For example, you should always ensure that the variable **list_elem_t * l** is different from NULL before performing any operation, such as **l** $\rightarrow$ **next**.
- Make sure your sources compile without causing any warning messages!
- Check your program for correct operation by completing the target to the **makefile** called **test**, which tests in a non-interactive way your program. To do so, you have to use the input/output redirections of the **test_list** program and to create and use a command file **commands.txt**