French Azerbaijani University
Arash Habibi
Second year
2017-2018

Advanced C Programming

Practical Work 4 : Linked lists

This practical work session starts with the implementation of what you designed in exercise session 5. But it will go way further. We will learn how to insert and remove elements and to concatenate lists.

1 Implementation of exercise session 5

1. Define the `struct floatList` structure capable of implementing a linked list of floats, with two fields : `val` and `next` ;
2. Write a function with prototype `struct floatList* FL_new1(float x);` that dynamically allocates a linked list, composed of one float `x`. What should be the value of `next` ? The return value of `FL_new1` is the address of this new list.
3. Write a function with prototype `void FL_show1(struct floatList *pf, char *label);` which prints on the terminal :
   — a character string `label` used to identify the output ;
   — the value of `pf` (use `%p` to print an address with `printf`) ;
   — the value of the `val` field of `pf` ;
   — the value of the `next` field of `pf` ;
   For example, the following code creates three lists, each one containing only one element. And it prints the results on the terminal.

```
struct floatList *pf1, *pf2, *pf3;

// create three lists with one element each
pf1 = FL_new1(5);
pf2 = FL_new1(10);
pf3 = FL_new1(3);

// show the result on the terminal
FL_show1(pf1,"first");
FL_show1(pf2,"second");
FL_show1(pf3,"third");
```

   On my computer, the output looks like :

```
first   : 0x7fbb6a404bf0 5.000000  0x0
second  : 0x7fbb6a404c00 10.000000 0x0
third   : 0x7fbb6a404c10 3.000000  0x0
```

   Test your functions and make sure that they produce similar results. You will probably have different addresses on each machine.
4. Now, we would like add `pf3` at the end of `pf2` and add `pf2` at the end of `pf1`. In this way, `pf1` would be a list containing three floats : 5, 10 and 3. Once again, use `FL_show1` to write the attributes of `pf1`, `pf2` and `pf3` and make sure that :
   — the `next` field of `pf1` is the address of `pf2` ;
   — the `next` field of `pf2` is the address of `pf3` .

5. Write a function with prototype `void FL_show(struct floatList *plist, char *label);` which does the same thing as `FL_show1` but it does not only show the fields of the first element of the list. It also shows the fields of the next element and the next one until the end of the list. For example, if you succeeded in answering the last question, the call `FL_show(pf1, "first list");` should produce something like :

   ```
   ------- Show floatList : first list ---------
   0x7fb131404c10 5.000  0x7fb131404c00
   0x7fb131404c00 10.000 0x7fb131404bf0
   0x7fb131404bf0 3.000  0x0
   ```

   In order to achieve this, I suggest that you define `pf` as a pointer to a float list. `struct floatList *pf;`. This variable would represent the *current* list element. In the beginning, `pf` would be the first element, then it would be the second element etc until the value of `pf` is `NULL` (end of list). You can achieve this result with a `while` loop.

6. Write another version of the previous function with a `for` loop.

2 Manipulating linked lists

I strongly recommend that after writing each function you test its behaviour by calling `FL_show`.

1. Change the previous function so that :
   — on each line, it prints the index of the element ;
   — after the last element, it prints the total number of elements in the list.
   For example for the previous list, it would print :

   ```
   ------- Show floatList : first list ---------
   0 : 0x7fb131404c10 5.000  0x7fb131404c00
   1 : 0x7fb131404c00 10.000 0x7fb131404bf0
   2 : 0x7fb131404bf0 3.000  0x0
   3 elements in all
   ```

2. Write a function with prototype `struct floatList* FL_newEmpty();` which returns a pointer to an empty list ;
3. Write a function with prototype `int FL_isEmpty(struct floatList *plist);` which returns `1` if `plist` is an empty list and `0` otherwise.
4. Write a function with prototype `struct floatList* FL_add(struct floatList *plist, float val);` which allocates a new float list containing one element with value `val`, adds this element at the beginning of `plist` and returns the pointer to the new list. For example the following code :

   ```
   struct floatList *plist;
   plist = FL_newEmpty();
   plist = FL_add(plist,3);
   plist = FL_add(plist,5);
   plist = FL_add(plist,7);
   FL_show(list,"list made by FL_add");
   ```

   should output :

   ```
   ------- Show floatList : list made by FL_add ---------
   0 : 0x7fad88404c10 7.000000 0x7fad88404c00
   1 : 0x7fad88404c00 5.000000 0x7fad88404bf0
   2 : 0x7fad88404bf0 3.000000 0x0
   3 elements in all.
   ```

5. Write a function with prototype `struct floatList* FL_firstInts(int n);` which returns a list containing the `n` first integers. You may use `FL_newEmpty` and `FL_add`. For example :

   ```
   struct floatList *ten = FL_firstInts(10);
   FL_show(ten,"10 first integers");
   ```

   should output :

   ```
   ------- Show floatList : 10 first integers ---------
   0 : 0x7fa53ac04c80 9.000000 0x7fa53ac04c70
   1 : 0x7fa53ac04c70 8.000000 0x7fa53ac04c60
   2 : 0x7fa53ac04c60 7.000000 0x7fa53ac04c50
   3 : 0x7fa53ac04c50 6.000000 0x7fa53ac04c40
   4 : 0x7fa53ac04c40 5.000000 0x7fa53ac04c30
   5 : 0x7fa53ac04c30 4.000000 0x7fa53ac04c20
   6 : 0x7fa53ac04c20 3.000000 0x7fa53ac04c10
   7 : 0x7fa53ac04c10 2.000000 0x7fa53ac04c00
   8 : 0x7fa53ac04c00 1.000000 0x7fa53ac04bf0
   9 : 0x7fa53ac04bf0 0.000000 0x0
   10 elements in all.
   ```

6. Write a function with prototype `struct floatList* FL_nth(struct floatList *plist, int n);` which returns a pointer to element number `n` in the list. For `n=0`, the function should return the first element. If `list` contains `n` elements or less, the function should print an error message and return `NULL`. Test the function with `ten`.
7. Write a function with prototype `struct floatList* FL_last(struct floatList *plist);` which returns a pointer to the last element of the list. It should return `NULL` if `plist` is empty. Avoid using `FL_nth` since this would imply that we loop through the list twice. Once for measuring the size and once more to find the last element. Test the function with `ten`.
8. Write a function with prototype `struct floatList* FL_removeFirst(struct floatList *plist);` which removes the first element of the list and returns the address of the new list. If `plist` is an empty list, then the function should return `plist` itself. Test your function with `ten`. The function should remove 9 and the resulting list should contain integers 8 to 0.
9. Write a function with prototype `struct floatList* FL_removeSecond(struct floatList *plist);` which removes the second element of the list and returns the address of the new list. If `plist` has 0 or 1 element, then the function should return `plist` itself. Test your function with `ten`. The function should remove 8 and the resulting list should contain integers 9 followed by integers 7 to 0.
10. Write a function with prototype :

    ```
    struct floatList* FL_insertAfterFirst(struct floatList *plist, float val);
    ```

    which inserts a new element with value `val` after the first element of `plist`. If `plist` is empty then the function should print an error message and return `plist` itself.

3 Mastering linked lists

In this section, we will see more complex functions which imply looping through a linked list.

1. Write a function with prototype :

   ```
   struct floatList* FL_previous(struct floatList *plist, struct floatList *pel);
   ```

   in which `plist` represents a linked list of floats and `pel` an element of that list. This function should return the address of the element just before `pel`. It should return `NULL` if `pel` is the first element or if `pel` was not found in `plist`. Test the function with `ten`. Choose the first or second element for `pel`.

2. Write a function with prototype

   ```
   struct floatList* FL_remove(struct floatList *plist, struct floatList *pel);
   ```

   in which `plist` represents a linked list of floats and `pel` an element of that list. This function should remove the element `pel` in the list and return a pointer to the resulting list. If `pel` does not belong to `plist`, this function should print an error message and return `plist`. You may use `FL_previous` to achieve the result. Test this function with `ten`. Use `FL_nth` to find different values for `pel`. Make sure that your function also works if `pel` is the first element of the list.

3. The use of `FL_previous` is not very satisfactory since it loops through the list. Write another version of `FL_remove` which does not loop through the list in order to find the previous element ;

4. Write a function with prototype :

   ```
   struct floatList* FL_insertAfter(struct floatList *plist,
                                    struct floatList *pel,
                                    float val);
   ```

   in which `plist` represents a list of floats and `pel` represents an element of `plist`. This function should insert a new element of value `val` after element `pel`. If `pel` does not belong to `plist`, then the function should print an error message and return `plist`. Test this function with `ten`. Use `FL_nth` to find different values for `pel`.

5. Write a function with prototype :

   ```
   struct floatList* FL_append(struct floatList *plist, float val);
   ```

   which adds a new element of value `val` at the end of `plist`. Test this function with `ten`.

6. Write a function with prototype :

   ```
   struct floatList* FL_concat(struct floatList *l1, struct floatList *l2);
   ```

   which concatenates both lists and returns the produced list. For example the following code :

   ```
   struct floatList *lst1 = FL_firstInts(3);
   struct floatList *lst2 = FL_firstInts(2);
   struct floatList *cat = FL_concat(lst1,lst2);
   FL_show(cat,"cat");
   ```

   should output :

   ```
   ------- Show floatList : cat ---------
   0 : 0x7fa9d1c04d50 2.000000 0x7fa9d1c04d40
   1 : 0x7fa9d1c04d40 1.000000 0x7fa9d1c04ce0
   2 : 0x7fa9d1c04ce0 0.000000 0x7fa9d1c04d70
   3 : 0x7fa9d1c04d70 1.000000 0x7fa9d1c04d60
   4 : 0x7fa9d1c04d60 0.000000 0x0
   5 elements in all.
   ```