French Azerbaijani University
Arash Habibi
Second year
2017-2018

Advanced C Programming

Exercice session 5 : Linked lists

Before you start this session, make sure you have read chapter 7.

1 Incomplete specifications

This section involves no programming, but only specifications. A complete specification is a specification which indicates what should be done in all cases. An incomplete specification may indicate what should be done in *most* cases. But some cases remain unspecified. For example :

complete specification : when I ask you to write a function that returns the size of a list, the specification is complete since every list is characterized by a size (number of elements). Even an empty list has a size (0).

Incomplete specification : when I ask you to write a function that returns the value of the first element of a list, the specification is incomplete since you can rightfully ask : *What do we do when there is no first element (when the list is empty) ?*

You must always ask that question, hopefully *before* you write and use your functions. Why ? Let us take an example : let us assume that a linked list of integers is defined by the following structure :

```
struct intList
{   int value;
    struct intList *next;
};
```

If I ask you to write a function with prototype `int first(struct intList *list);` that returns the value of the first element of the list, you would probably write :

```
int first(struct intList *list)
{
   return(list->value);
}
```

which would work in almost all cases, but which would make your program crash if `list` is empty. In the following questions, determine which specifications are incomplete and find what cases were not specified :

1. Write a function that, for a given list of integers `list`, returns the last element of `list` ;
2. Write a function that creates a new empty list and returns the address of this list ;
3. Write a function which, for a given list of integers `list` and an integer `n` returns a new list beginning with `n` and followed by the elements of `list` if any.
4. Write a function which, for a given list of integers `list` prints on the terminal the values of all of its elements if any.
5. Write a function which, for a given list of integers `list` and an integer `n`, returns the value of element number `n` in the list. For `n=0`, the function should return the value of the first element of the list. For `n=1`, the function should return the value of the second element of the list and so on. If `n` is negative, the function should print an error message.

6. Write a function which, for a given list of integers `list` and `el`, an element of `list` returns the value of the element located just before `el`
7. Write a function which, for a given list of integers `list` returns 0 if the list is empty and 1 if it is not ;
8. Write a function which, for a given list of integers `list` and an integer `n` returns a new list beginning with the elements of `list`, followed by `n`.
9. Write a function which, for a given list of integers `list` removes the last element of `list`.

2 Linked lists

1. Define the `struct floatList` structure capable of implementing a linked list of floats ;
2. Define the `struct charlist` structure capable of implementing a linked list of chars ;
3. Define the `struct int5Arraylist` structure capable of implementing a linked list of arrays containing 5 integers ;
4. Define the `struct intArraylist` structure capable of implementing a linked list of pointers to integers ;
5. Let the color structure be defined as follows :

   ```
   struct color { unsigned char red, green, blue; };
   ```

   Define the `struct colorlist` structure capable of implementing a linked list of colors.

3 Linked list of floats

In this section, our aim is to make a linked list of floats.

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 *f, char *label);` which prints on the terminal :
   — a character string `label` used to identify the output ;
   — the value of `f` (use `%p` to print an address with `printf`) ;
   — the value of the `val` field of `f` ;
   — the value of the `next` field of `f` ;
   For example, the following code creates three lists, each one containing only one element. And it prints the results on the terminal.

   ```
   struct floatList *f1, *f2, *f3;

   // create three lists with one element each
   f1 = FL_new1(5);
   f2 = FL_new1(10);
   f3 = FL_new1(3);

   // show the result on the terminal
   FL_show1(f1,"first");
   FL_show1(f2,"second");
   FL_show1(f3,"third");
   ```

   On my computer, the output looks like :

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

4. Now, we would like add `f3` at the end of `f2` and add `f2` at the end of `f1`. In this way, `f1` would be a list containing three floats : 5, 10 and 3. Once again, use `FL_show1` to write the attributes of `f1`, `f2` and `f3` and make sure that :
   — the `next` field of `f1` is the address of `f2` ;
   — the `next` field of `f2` is the address of `f3` .
5. Write a function with prototype `void FL_show(struct floatList *list, 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(f1, "first list");` should produce something like :

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

   I suggest that you define `struct floatList *f;` representing the *current* list element. In the beginning, `f` would be the first element, then it would be the second element etc until the value of `f` is `NULL` (end of list). You can achieve this result with a `while` loop.
6. Write another version of the pervious function with a `for` loop.