

Informatics II, Spring 2024, Exercise 6

Publication of exercise: April 8, 2024

Publication of solution: April 15, 2024

Exercise classes: April 15 - April 19, 2024

Learning Goal

- Deepen the understanding of Pointers and Linked List.
- Being able to apply them to further advanced ADTs.

Task 1 [Easy]

Complete the following table. *Hint: Draw or try it out if it helps.*

C code	What is it?
<pre>int i; int *p; &i p *p &p int **q; *q **q</pre>	<i>Integer somewhere in memory</i>

(Bonus) Pointer Arithmetic

Let `a` be an integer array with length of 10.

C code	What is it?
<pre>int a[10]; a[0] int *p = &a[0] *p int *pa = a; *pa *(pa+1) *pa+1</pre>	<i>Integer array <code>a</code> of size 10</i>

Task 2 [Medium]

Given is the following struct of a node. Each node represents a month, with some properties, and is in a linked list.

```
1 struct month {
2     int month_number;
3     char *month_name;
4     struct month *next;
5 };
```

After compiling the script, a linked list of these month nodes is initialized in a random order. (You do not need to understand how they are initialized for this exercise.) The linked list is stored in the `head` pointer, which points to the head of this month linked list. Please note that this is a single linked list, with only a pointer to the head (no tail pointer).

Use the template C code in the `./task2.c` file. You do not have to modify the `init_months()` and `free_months()` function. These only serve as setup and teardown of the linked list. Your tasks are listed below.

(a) Print Linked List

As a first task, implement the `print_months()` function. This function should print each month node in the linked list. Use the following format for the print statements: "January (1)".

```
1 void print_months(struct month *head) {
2     printf("months: ");
3 }
```

(b) Get Previous Node

Sometimes we want to know the preceding month of a given month in our linked list. Implement the `get_previous_month()`, which returns a pointer to the preceding month. *Hint: You might want to add some parameters.*

```
1 struct month * get_previous_month() {
2     return NULL;
3 }
```

Hint: Test your implementation.

(c) Swapping Nodes

It is useful to be able to swap two months in the linked list. Implement the `swap_month(struct month *head)` function, which returns a pointer to the (new) head. The swapping should swap the entire node, **not** the node values.

```
1 struct month* swap_month(struct month *head, struct month *a, struct month *b) {
2     return head;
3 }
```

Think about different cases which might occur. You can assume, that `a` comes always before `b` in the linked list.

(d) Selection Sort

Now we want to combine the previously implemented functions to create a **in order** selection sort algorithm. For that implement the `selection_sort()` function, which returns a pointer to the (changed) head. Keep in mind that this implementation is in order, which means that you should not create a second head or linked list.

```
1 struct month *selection_sort(struct month *head) {  
2     return head;  
3 }
```

Hint: Use the previously implemented functions and print for every step the current linked list with `print_months()`.

Task 3 [Medium]

(a) Merge

Consider two sorted doubly linked lists. Implement a function `merge()`, which merges those two lists into a single doubly linked list. The resulting, merged, list should be sorted and containing all elements from the input linked lists. The `merge()` function should not copy values of the input linked lists. Use the following `struct` for a node.

```
1 struct node {  
2     int val;  
3     struct node *next;  
4     struct node *prev;  
5 };
```

(b) Tenet

Let the same `struct` as in Task 3(a) define a node in a **doubly linked list**. Implement a function `reverse()` which takes a pointer to a node and returns the pointer to the head of the reversed list. The `reverse()` function should only change pointers. It should not create new nodes.

There are two ways to implement this. The easier way is to implement an iterative algorithm. A harder, but more elegant solution is to implement a recursive function.