C Programming language exam 3

02/02/2021

- The result has to be delivered as a single C file called `list.c` and a header file called `list.h` inside a folder called `exam_3` in your depots folder
- The C file must be compilable with the command `clang -c -Wall -Werror list.c`
- Each exercise will ask you to create one or several functions
- You are allowed to write other functions
- You are allowed to use functions written in other exercises
- You are allowed to ask and search about C features
- Only use the C standard library (no `unistd.h`)
- Don't use global variables
- Print error messages on `stderr`
- **Don't let memory leaks happen**
- **Don't let segmentations faults happen**
- **Always check the result of memory allocations**

Linked List data structure

- Consider the following structures:
- `typedef struct {void* data; struct list_node* next} list_node;`
- `typedef struct {list_node* begin; size_t count;} list;`
- `list` defines a list of elements, wich are all pointers
- `count` is the number ot elements of the list
- `begin` is a pointer to the container of the first element of the list
- `begin` is NULL if the list is empty
- `list_node` is a container of an element of the list
- `data` is a pointer to an element of the list
- `next` is a pointer to the container of the next element of the list
- `next` is NULL if this is the last element of the list
- Note that nothing indicates the type or the size of the pointed data, as it won't be needed
- Don't forget that `begin`, `next` and `count` have to stay valid every time something changes on the list

Exercise 1 (1 pt)

- Write a function whose prototype is `list* create_list()`
- It should allocate and return an empty list

Exercise 2 (0.5 pt)

- Write a function whose prototype is `size_t list_count(const list* _list)`
- It should return the current number of elements on the list

Exercise 3 (1.5 pt)

- Write a function whose prototype is `list_node* list_get_node(list* _list, size_t _index)`

- It should return the node containing the element at given index
- The purpose of this function is to be re-used on the subsequent exercises
- If you're stuck on this, move to the next exercises and come back later

Exercise 4 (1 pt)

- Write a function whose prototype is `size_t list_add(list* _list, void* _data)`
- It should add a new element at the end of the list
- That new element should have the pointer `_data` as contained pointer
- It should return the new size of the list
- No memory copy of pointed data should be performed here

Exercise 5 (0.5 pt)

- Write a function whose prototype is `void* list_get(list* _list, size_t _index)`
- It should return the pointer contained by the element of index `_index` on the list
- It should return NULL if there is no element at given index

Exercise 6 (0.5 pt)

- Write a function whose prototype is `bool list_set(list* _list, size_t _index, void* _data)`
- It should change the pointer contained by the element at index `_index` to the given `_data` pointer, if such an element exists
- `bool` is defined in `<stdbool.h>` since C99
- It should return `true` if successfull, `false` otherwise

Exercise 7 (1.5 pt)

- Write a function whose prototype is `bool list_insert(list* _list, size_t _index, void* _data)`
- It should insert a new element on `_list` at position `_index`, shifting all following elements towards a greater index
- It should not do anything if the index is greater than the current number of elements of the list
- It should return true if the new element was inserted, false otherwise

Exercise 8 (1.5 pt)

- Write a function whose prototype is `bool list_remove(list* _list, size_t index)`
- It should remove element at given index from the list

Exercise 9 (1 pt)

- Write a function whose prototype is `voir destroy_list(list* _list)`
- It should free all memory allocated for the list and its nodes

Exercise 10 (1 pt)

- Observe the code you wrote and compare it to the code of the `dynamic_array` from

the previous exam

- Explain (inside a comment) on which cases using a `list` would result in a more optimised code than using a `dynamic_array`