C Programming language exam 2

03/11/2020

- The result has to be delivered as a single C file called `exam_2.c`
- The C file must be compilable with the command `clang -c -std=c89 -pedantic -Wall -Werror exam_2.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
- You are allowed to use the C standard math library (requiring `-lm` as a flag)
- 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**
- Completing bonuses double the points of the exercise

Dynamic array structure

- Consider the following structure: `typedef struct {void* data; size_t count; size_t capacity; size_t type_size} dynamic_array`
- It defines a resizable array that can be of any C type
- `data` is a pointer to the beginning of the array
- `count` is the number of values contained in the array, also refered as the size of the array
- `capacity` is the actual number of values allocated on the memory pointed by data
- `type_size` is the size on bytes of a value of the type that constitutes the array
- `data` always points to a block of heap data of `capacity * type_size` bytes
- `data` can be `NULL`, on which case `capacity` is 0
- `count` is always inferior or equal to `capacity`
- `capacity` is here to avoid using `realloc` each time the number of elements of the array is changed
- In other terms, memory allocation should happen only when capacity is changed
- `type_size` is not supposed to change over the lifetime of the structure
- `void*` are pointers to any kind of data
- A `void*` can be casted to a pointer to any type by using the cast `(type*)` operator

Exercise 1 *(1 pt)*

- Write a function whose prototype is `dynamic_array create_dynamic_array(size_t type_size)`
- It should allocate and initialize an empty dynamic array designed to receive value of a type whose size is `type_size`
- Typical usage would be: `dynamic_array* darray = create_dynamic_array(sizeof(int));`
- It should print an error message and return NULL if the allocation failed

Exercise 2 *(0.5 pt)*

- Write a function whose prototype is `size_t dynamic_array_count(const dynamic_array* darray)`
- It should return the number of values contained in the array

Exercise 3 *(1 pt)*

- Write a function whose prototype is `size_t dynamic_array_reserve(dynamic_array* darray, size_t capacity)`
- It should extend the capacity of the array to at least the capacity passed as a parameter
- It should let the structure unchanged if the capacity passed as a parameter is smaller than the current capacity
- It should let the structure unchanged and print an error message if the memory allocation fails
- It should return the final capacity of the array regardless of if it has been changed or not
- Typical usage of this functions happens at the beginning of the lifetime of the array, to give a capacity that's large enough to contain all expected future values
- *Hint: If `realloc` is called on a null pointer, it behaves like `malloc`*

Exercise 4 *(1 pt)*

- Write a function whose prototype is `size_t dynamic_array_resize(dynamic_array* darray, size_t size)`
- It should set the new size of the array, extending its capacity if necessary
- It should return the new size
- *Bonus: It should initalize all new values to 0*
- *Advice: Don't try the bonus until you've done exercise 7*

Exercise 5 *(1 pt)*

- Write a function whose prototype is `size_t dynamic_array_shrink(dynamic_array* darray)`

- It should reduce the capacity of the array to its size
- It should print a message in the case of an error
- It should return the new capacity of the array
- Typical usage of this function happens when the array is not expected to grow anymore with the intention to reduce memory usage

Exercise 6 *(1 pt)*

- Write a function whose prototype is `void* dynamic_array_get(dynamic_array* darray, size_t index)`
- It should return a pointer to the value that is at the index `index` in the array
- It should return NULL and print an error message if the requested index is outside the array
- Typical usage would be `int a = * (int*) (dynamic_array_get (darray, 3));`
- _Hint: If you cast you void* to a char*, you can then use pointer arythmetics on it on the base of a byte_
- *Example:* `((char*) array) + 4` *gives the adress of a value that's 4 bytes away from the start*
- *Hint for later: this function returns a pointer to a value, which means it can be used to edit that value*

Exercise 7 *(1 pt)*

- Write a function whose prototype is `size_t dynamic_array_set(dynamic_array* darray, size_t index, const void* value)`
- It should set the value at the provided index to the value pointer by the pointer passed as parameter
- It should print an error message if the requested index is outside the array
- It should return 1 if the value has been set, 0 otherwise
- Typical usage would be `dynamic_array_set (darray, 3, &value);`
- *Hint: memcpy is a good way to copy arbitrary amounts of memory*

Exercise 8 *(1 pt)*

- Write a function whose prototype is `size_t dynamic_array_add(dynamic_array* darray, const void* value)`
- It should increase the size of the array by 1 and set the new value to the value pointed by the pointer passed as a parameter
- In the case of an error, it should print an error message
- It should return 1 if the value has been set, 0 otherwise
- *Bonus: if the current capacity is not enough, it should double the capacity of the array instead of just increasing it by one*

Exercise 9 *(1 pt)*

- Write a function whose prototype is `size_t dynamic_array_append(dynamic_array* darray, const dynamic_array* darray_to_append)`
- It should add all values of `darray_to_append` at the end of `darray`
- It should print an error message and not perform the append if the two arrays contain data of different type size
- It should return 1 if the append occured, 0 otherwise

Exercise 10 *(1 pt)*

- Write a function whose prototype is `void destroy_dynamic_array(dynamic_array* darray)`
- It should free all memory that's been allocated for the dynamic array

Exercise 11 *(2 pts)*

- Consider the following structures:
- `typedef struct {dynamic_array* darray} dynamic_char_array;`
- `typedef struct {dynamic_array* darray} dynamic_int_array;`
- `typedef struct {dynamic_array* darray} dynamic_double_array;`
- Write the following functions as wrappers of the previous functions:
- `dynamic_char_array* create_dynamic_char_array()`
- `size_t dynamic_char_array_count(const dynamic_char_array* darray)`
- `size_t dynamic_char_array_reserve(dynamic_char_array* darray, size_t capacity)`
- `size_t dynamic_char_array_resize(dynamic_char_array* darray, size_t size)`
- `size_t dynamic_char_array_shrink(dynamic_char_array* darray)`
- `char dynamic_char_array_get(dynamic_char_array* darray, size_t index)`
- `size_t dynamic_char_array_set(dynamic_char_array* darray, size_t index, char value)`
- `size_t dynamic_char_array_add(dynamic_char_array* darray, char value)`
- `size_t dynamic_char_array_append(dynamic_char_array* darray, dynamic_char_array darray_to_append)`
- `void destroy_dynamic_char_array(dynamic_char_array* darray)`
- `dynamic_int_array* create_dynamic_int_array()`
- `size_t dynamic_int_array_count(const dynamic_int_array* darray)`
- `size_t dynamic_int_array_reserve(dynamic_int_array* darray, size_t capacity)`
- `size_t dynamic_int_array_resize(dynamic_int_array* darray, size_t size)`
- `size_t dynamic_int_array_shrink(dynamic_int_array* darray)`
- `int dynamic_int_array_get(dynamic_int_array* darray, size_t index)`
- `size_t dynamic_int_array_set(dynamic_int_array* darray, size_t index, int value)`

- `size_t dynamic_int_array_add(dynamic_int_array* darray, int value)`
- `size_t dynamic_int_array_append(dynamic_int_array* darray, dynamic_int_array darray_to_append)`
- `void destroy_dynamic_int_array(dynamic_int_array* darray)`
- `dynamic_float_array* create_dynamic_float_array()`
- `size_t dynamic_float_array_count(const dynamic_float_array* darray)`
- `size_t dynamic_float_array_reserve(dynamic_float_array* darray, size_t capacity)`
- `size_t dynamic_float_array_resize(dynamic_float_array* darray, size_t size)`
- `size_t dynamic_float_array_shrink(dynamic_float_array* darray)`
- `float dynamic_float_array_get(dynamic_float_array* darray, size_t index)`
- `size_t dynamic_float_array_set(dynamic_float_array* darray, size_t index, float value)`
- `size_t dynamic_float_array_add(dynamic_float_array* darray, float value)`
- `size_t dynamic_float_array_append(dynamic_float_array* darray, dynamic_float_array darray_to_append)`
- `void destroy_dynamic_float_array(dynamic_float_array* darray)`