1 Bachelor Game Programming

C Programming language exam

20/10/2020

- The result has to be delivered as a single C file called `exam_1.c`
- The C file must be compilable with the command `clang -c -std=c89 -pedantic -Wall -Werror exam_1.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
- **Don't let memory leaks happen**
- **Don't let segmentations faults happen**
- Completing bonuses double the points of the exercise

Exercise 1 *(.5 pt)*

- Write a function whose signature is `int sum_first_integers (int n)` which returns the sum of the n first integer numbers
- It should return `0` if `n` is `0`
- It should return `-1` if n is negative
- It should return `-1` or if the sum overflows `INT_MAX`
- *Hint: if `a + n` with `b` being positive is inferior to `a`, it means that the sum overflowed*
- *Bonus:* Don't use a loop

Exercise 2 *(1 pt)*

- Write a function whose signature is `int sum_int_array(const int* array, size_t size)` which returns the sum of all the values of the array passed as parameter
- `array` is guaranteed to point at `size` integer values (this also true for all subsequent exercises)
- Values are guaranteed to be at a scale and at a number that would prevent overflows

Exercise 3 *(1 pt)*

- Write a function whose signature is `int min_int_array(const int* array, size_t size)` which returns the minimum value contained in the array passed as parameter
- Write a function whose signature is `int max_int_array(const int* array, size_t size)` which returns the maximum value contained in the array passed as parameter
- *Bonus:* Write a function whose signature is `void min_max_int_array(const int* array, size_t size, int* min, int* max)` which writes the mininum value and the maximum value contained in the array to the variables respectively pointed by `min` and `max`
- `min` and `max` are guaranteed to be valid pointers to integers
- `min_int_array` and `max_int_array` should be implemented by calling `min_max_int_array`

Exercise 4 *(1 pt)*

- Write a function whose signature is `void heap_10M()` that allocates exactly 10 000 bytes of heap memory, prints a message to the standard output, then releases the memory
- It should print an error message if there is not enough available memory
- *Bonus:* Write a function whose signature is `void stack_10M()` that allocates exactly 10 000 bytes of stack memory, prints a message to the standard output, then releases the memory

Exercise 5 *(1 pt)*

- Consider the following structure : `typedef struct {int* array, size_t size} dynamic_int_array;`
- Write a function whose signature is `dynamic_int_array* create_dynamic_int_array(size_t size)` which creates an array of

integers of the provided size and returns a pointer to a `dynamic_int_array` containing the pointer to the start of the array and its size
- All values in the array should be initalized to 0
- Write a function whose signature is `void destroy_dynamic_int_array(dynamic_int_array* darray)` which releases the memory of the array and the memory of the structure

Exercise 6 *(1 pt)*

- Write a function whose signature is `int dynamic_int_array_get(const dynamic_int_array* darray, size_t index)` which returns the int that's in the position `index` of the array
- It should return 0 if the provided index is negative or overflows the array
- Write a function whose signature is `size_t dynamic_int_array_add(dynamic_int_array* darray, int value)` which increases the size of the array by 1 and sets the end value to `value`, then returns the index of that last value
- *Hint: `realloc` returns a new pointer to the data with a new size, for a `malloc`-allocated array*

Exercise 7 *(2 pts)*

- Write a function whose signature is `void dynamic_int_array_set(dynamic_int_array* darray, size_t index, int value)` which sets `value` to the position `index` of the array
- It should resize the array if it isn't large enough, filling all intermediate new values with 0
- Write a function whose signature is `int dynamic_int_array_remove(dynamic_int_array* darray, size_t index)` which returns the value at the position `index` from the array and removes it
- It should return 0 if the index has an invalid value
- It should resize down the array and shift towards the begining every value following the removed one

Exercise 8 *(1 pt)*

- Write a function whose signature is `dynamic_int_array* copy_dynamic_int_array(const dynamic_int_array* darray)` which creates and returns a copy of the array, containing the same values and of the same size
- Write a function whose signature is `dynamic_int_array* sub_dynamic_int_array(const dynamic_int_array* darray, size_t start, size_t end)` which creates and returns a copy of a portion of the array
- The copied portion should start at the index `start`
- The copied portion should end just before the index `end`, meaning the value at the position `end` should not be included
- If the indices of the requested portion to copy overflow the array, only values that exist should be copied
- *Bonus:* Implement `copy_dynamic_int_array` using `sub_dynamic_int_array`

Exercise 9 *(1 pt)*

- Write a function whose signature is `char* read_file(const char* path)` which returns a null-terminated char array containing all the text of the file
- It should return a pointer to data that can be released later using `free`
- *Hint: One way to do this is by having a fixed-size char array that acts as a buffer, using `fgets` to read the file chunk by chunk until it ends, `realloc` and `strncat` to append the buffer text to the final string*

Exercise 10 *(1 pt)*

- Consider the following structure : `typedef struct {int number, char * file_path, char * text, int * links} interactive_story_paragraph;`
- It represents a pragraph of a text-based interactive story
- `number` is the number of the paragraph in the story
- `file_path` is the full path (including the filename) of the file that contains the text of the paragraph
- `text` is a null-terminated char array containing the entire text of the paragraph if the text has already been loaded, or a null pointer otherwise
- `links` is a null-terminated int array containing the list of the numbers or the paragraphs linked by the paragraph if the texte has already been loaded, or a null pointer otherwise
- Write a function whose signature is `interactive_story_paragraph* create_interactive_story(char* folder_path, int paragraph_count)` which returns an array of `interactive_story_paragraph`
- It should not load the text of the paragraph
- Each paragraph file is expected to exist inside the folder whose address is `folder_path`
- Each paragraph file is the paragraph number

- The paragraph count starts at 0
- *Hint : On POSIX-compliant systems the directory separator character is* `/`

Exercise 11 *(2 pts)*

- Write a function whose signature is `void interactive_story_chapter_load(interactive_story_chapter* chapter)` which loads the text from the file
- It should print a message to the standard input instructing which file is being read and if the operation is successful or not
- It should let the structure untouched if the file didn't exist
- It should not reload the text if it's already been loaded
- It should fill the `links` array with every number contained in the text enclosed by `*`
- Example: `"To open the left door, go to *12*."` would add 12 to the `links` array
- The `links` array should have at least one more slot than the total number of linked paragraphs, having a value of 0
- Write a function whose signature is `void interactive_story_load_all(interactive_story_chapter* story, int chapters_count)` which loads all the chapters of the story
- `story` is an array of `interactive_story_paragraph` which size is `chapters_count`

Exercise 12 *(1.5 pts)*

- Write a function whose signature is `void interactive_story_chapter_print(const interactive_story_chapter* chapter)` which prints the text of the chapter to the standard output
- It should print an error message if the chapter text has not been loaded
- Write a function whose signature is `void interactive_story_chapter_load_print(interactive_story_chapter* chapter)` which loads the chapter if not loaded then prints the text of the chapter to the standard input

Exercise 13 *(1.5 pts)*

- Write a function whose signature is `int interactive_story_chapter_scan_player_choice(const interactive_story_chapter* chapter)` which asks the player for their choice and returns it
- It should first print to the standard output a message asking the player to write the number of the paragraph they want to go next
- It should then read from the standand input the number of the paragraph
- If the paragraph is not listed in the `link` array, it should ask again until the player inputs a valid paragraph number
- If the `link` array is empty, it should print to the standard output that the story is over and return 0
- If the chapter is not loaded, it should print an error message to the standard output and return 0

Exercise 14 *(1 pt)*

- Write a function whose signature is `void interactive_story_chapter_unload(interactive_story_chapter* chapter)` which resets the chapter to the unloaded state
- It should free all memory dynamically allocated by the `interactive_story_chapter_load` function
- Write a function whose signature is `void destroy_interactive_story(interactive_story_chapter* story, int chapters_count)` which unloads every chapter and completly releases the memory of the story