C Programming language exam 4

23/02/2021

- The result has to be delivered as a single header file called `list_generic.h` inside a folder called `exam_4` in your depots folder
- Files `list.c` and `list.h` from previous exam should be used and included, with `list.h` including all definitions needed in `list_generic.h`
- The header file must be compilable with `-Wall -Werror` with clang when included by a C file
- The header file should contain include guards
- Each exercise will ask you to create one or several macros
- 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**
- Remember that the `-E` flag on `gcc` can show you the preprocessour output of your code

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 macro whose name and arguments are `LIST_TYPE(TYPE)`
- It should define a type whose name is `list_` followed by `TYPE`
- For instance, `LIST_TYPE(float)` should define a type whose name is `list_float`
- Don't take into consideration type names that contain spaces

Exercise 2: 1 pt

- Write a macro whose name and arguments are `LIST_METHOD_DECL_CREATE(TYPE)`
- It should **declare** a function whose prototype is `list_type* create_list_type`

`()` with `type` replaced by the parameter `TYPE`

- Write a macro whose name and arguments are `LIST_METHOD_DECL_DESTROY(TYPE)`
- It should **declare** a function whose prototype is `void destroy_list_type (list_type* _list)` with `type` replaced by the parameter `TYPE`

Exercise 3: 1 pt

- Write a macro whose name and arguments are `LIST_METHOD_DEFN_CREATE(TYPE)`
- It should **define** the function `list_type* create_list_type()` declared on the previous exercise
- It should wrap the function `list* create_list()` from the previous exam
- Write a macro whose name and arguments are `LIST_METHOD_DEFN_DESTROY(TYPE)`
- It should **define** the function `void destroy_list_type (list_type* _list)` declared on the previous exercise
- It should wrap the function `void* destroy_list(list*)` from the previous exam

Exercise 4: 1 pt

- Write the following macros:
- `LIST_METHOD_DECL_COUNT(TYPE)` which declares `size_t list_type_count (const list_type* _list)`
- `LIST_METHOD_DECL_ADD(TYPE)` which declares `size_t list_type_add (list_type* _list, type* _data)`
- `LIST_METHOD_DECL_GET(TYPE)` which declares `type* list_type_get (list_type* _list, size_t _index)`
- `LIST_METHOD_DECL_SET(TYPE)` which declares `bool list_type_set (list_type* _list, size_t _index, type* _data)`
- `LIST_METHOD_DECL_INSERT(TYPE)` which declares `bool list_type_insert (list_type* _list, size_t _index, type* _data)`
- `LIST_METHOD_DECL_REMOVE(TYPE)` which declares `bool list_type_remove (list_type* _list, size_t _index)`

Exercise 5: 1 pt

- Write the following macros defining the functions declared on the previous exercise
- `LIST_METHOD_DEFN_COUNT(TYPE)`
- `LIST_METHOD_DEFN_ADD(TYPE)`
- `LIST_METHOD_DEFN_GET(TYPE)`
- `LIST_METHOD_DEFN_SET(TYPE)`
- `LIST_METHOD_DEFN_INSERT(TYPE)`
- `LIST_METHOD_DEFN_REMOVE(TYPE)`
- All of them should be wrappers around functions from the previous exam

Exercise 6: 1.5 pt

- Write the following macros:
- `LIST_METHOD_DECL_VAL_ADD(TYPE)` which declares `size_t list_type_add (list_type* _list, type _value)`
- `LIST_METHOD_DECL_VAL_GET(TYPE)` which declares `type list_type_get (list_type* _list, size_t _index)`
- `LIST_METHOD_DECL_VAL_SET(TYPE)` which declares `bool list_type_set (list_type* _list, size_t _index, type _value)`
- `LIST_METHOD_DECL_VAL_INSERT(TYPE)` which declares `bool`

`list_type_insert (list_type* _list, size_t _index, type _value)`

- `LIST_METHOD_DECL_VAL_REMOVE(TYPE)` which declares `bool list_type_remove (list_type* _list, size_t _index)`
- Note that those functions are not compatible with the functions declared at exercise 4
- Write the follwing macros defining the functions declared above:
- `LIST_METHOD_DEFN_VAL_ADD(TYPE)`
- `LIST_METHOD_DEFN_VAL_GET(TYPE)`
- `LIST_METHOD_DEFN_VAL_SET(TYPE)`
- `LIST_METHOD_DEFN_VAL_INSERT(TYPE)`
- `LIST_METHOD_DEFN_VAL_REMOVE(TYPE)`
- The definitions should wrap arount the functions from the previous exam and manage the allocation and deallocation of the memory containing the values
- What happens when invalid indices are accessed is considered undefined behavior
- Note: The test program should generate exactly 3 memory leaks, this is addressed in ex9