Lab 5 – Dynamic Data Types and Libraries

Exercise: *pointer list implementations*

THE SOLUTION OF THIS EXERCISE NEEDS TO BE UPLOADED ON TOLEDO BEFORE THE NEXT LAB.

Your solution is only accepted if the following criteria are satisfied:

1. *Compilation with '-Wall -Werror' option generates no warnings or errors*
2. *Running 'cppcheck –enable=all --suppress=missingIncludeSystem' on the source code results in no warnings or errors*
3. *Basic implementation should contain at least the 9 methods stated in the list.h file. Write a test application to check the functionality of the insert, remove and free actions on the list.*
4. *Run your test application with Valgrind to check no memory leaks are detected by this tool.*

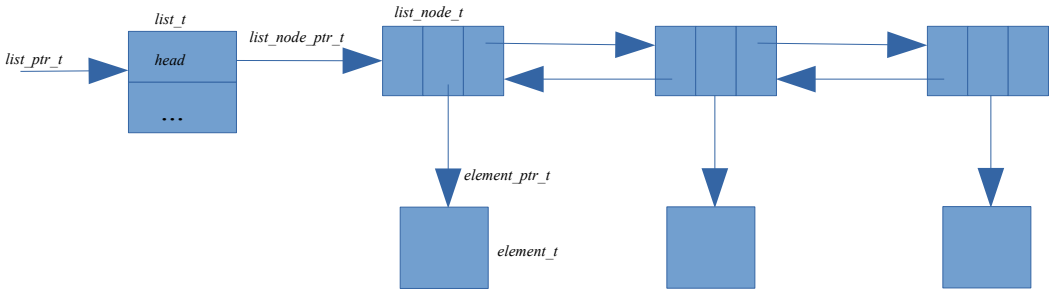
As an extra: Using conditional compilation you can also enable the extra features for the list implementation

DO NOT upload code that doesn't satisfy these criteria.

Once in your life you should have programmed a single and double linked pointer list before you can call yourself a software programmer, or, even better, a 'distinguished' C programmer! Well, this is the time to take that hurdle …

Implement a double-linked pointer list in C such that the code can be archived in a library (see the next exercise) and can be re-used in other applications. Put the code related to the linked-list in a list.c file, accompanied by a list.h header file. Use const, static and extern keywords wisely!

For simplicity you may assume that the data stored in the list is of a basic type **element_t**. Also define the type **element_ptr_t** being a pointer to a **element_t.** This will change in one of the following exercises to a more general data type but first make this code working fine before moving to the next level of complexity.



First, you have to define a data structure for the double-linked pointer list. Use the type name **list_t** for this data type. Also define the type **list_ptr_t** being a pointer to a **list_t**. A list consists basically out of list nodes. List nodes contain a reference to the element stored in the node, and *next* and *previous* references to, respectively, the next and previous list nodes. List nodes are define by the data types **list_node_t** and **list_node_ptr_t.**

Next, you have to implement the (basic) set of list operations defined in the template file 'list.h' attached to this lab document. Feel free to implement more operations if needed.

Define a set of error codes and use a global variable 'list_errno' for error messaging. Every list operator will reset list_errno to 0 at the start of the function implementing the operator. If any error happens during the execution of the function, it will set list_errno to a meaningful error code. It is the responsibility of the caller of the list operator to check the value of list_errno.

Exercise: *static library*

Create a static library containing the list implementation. Copy the library to a local directory in your home folder, e.g. /home/lucvd/mylibs. Implement a main.c function that uses a list. Build main.c and the list library to an executable (assuming that the main.c file is not in the same directory of the static library). Run and test the program. Use 'objdump' to find out if the library code is really included in the executable.

Exercise: *dynamic library*

This is a similar exercise as the previous one, but this time we build a shared library containing the list implementation. Again, copy the library to the local directory in your home folder, e.g. /home/lucvd/mylibs. Use the main.c function from the previous exercise to build an executable using the list library (again assuming that the main.c file is not in the same directory of the static library). Use 'ldd' to find out if the loader has a reference to the shared library. If that's ok, run and test the program. Again, use 'objdump' to find out that the library code is NOT really included in the executable.