Code

The common approach to solve issues coming from the coding part is to establish so-called **Code Guidelines** within an organization: a set of mandatory rules and optional recommendations with the goal of having uniform code style that is more robust to errors.

The real-life full code guidelines are quite big and complicated and yes, they suppress many individual “likes” and “wants” for the sake of higher overall quality and easier integration of new members. But to start with, concerning style (code formatting) the suggestion will be: **pay attention to your style, use the same style consistently throughout your code**, including many very small things like identifiers naming (small and capital letters, underscores), positioning of curly brackets, writing long lists of function arguments, spaces or their absence between brackets and identifiers, spaces or their absence around mathematical operators, etc. If you find it difficult - there are tools called “beautifiers” or “code formatters” which can ensure consistent style for you.

Some important points here:

1. **Turn on maximal warnings level** in your compiler (for GCC the option is -Wall) and solve as many as possible. Some of them show potential problems, i.e. that on different platforms the code behaves differently, some of them actually indicate the common mistakes (like ‘=’ instead of ‘==’ in the condition statement, unused variables). *If you have questions like “why is it a warning?” or “how can I solve it?”, I am willing to help*.

2. Make sure to **implement error checks properly** from the very beginning, if you decide to check for errors. It will save you time in case your program behaves in an unexpected way, as error checking code may be the last part you would check for mistakes.

3. **Initialize all pointers with zero** (NULL if you like), set them to zero when you delete them and **make sure that a pointer is non-zero when you call a method on it or write to it**. For prototype-level code, assertions may suffice (#include <cassert> at the beginning, assert(ptr != 0) in the code).
   For those who do not know assertions: it is a way to check a condition, mostly used during debugging process to find programmers’ errors. If condition is true, the assert does nothing. If condition is false, it breaks the execution and shows the place (name of the file and number of line) where the condition failed. If you run the program under debugger, it serves similar to a breakpoint. Otherwise it aborts execution of the program (which for the case with zero pointer is nicer than just receive “segmentation fault” message).

4. **Write simple clean code first**, make sure it compiles and executes as intended. Only after ensuring that it works you may start to optimize it, documenting your optimizations (probably, leaving the simple code version as comments or referencing it like “for non-optimized version, see file ‘project/file.cpp’ in revision ‘xx’ of repository ‘vcgl.jacobs-university.de/svn/my_repo’ ” - *this is to the proper use of SVN*).