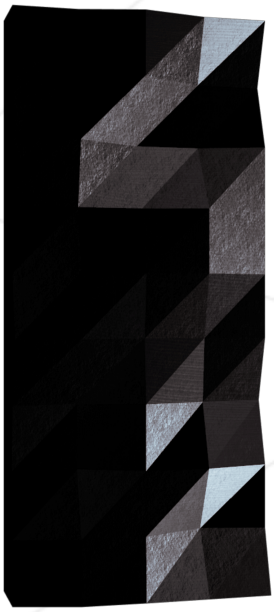


# C++ Pool - rush01

## ft_gkrellm

Staff 42 bocal@staff.42.fr

*Abstract:* *This document contains the subject for rush 01 of 42's C++ pool, which is a very good reason to start hating the Bocal with all your being.*

# Contents

# Chapter I

# General rules

- Any function implemented in a header (except in the case of templates), and any unprotected header, means 0 to the exercise.
- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
- The imposed filenames must be followed to the letter, as well as class names, function names and method names.
- Remember: You are coding in `C++` now, not in `C` anymore. Therefore:
  - The following functions are FORBIDDEN, and their use will be punished by a `-42`, no questions asked: `*alloc`, `*printf` and `free`.
  - You are allowed to use basically everything in the standard library. HOWEVER, it would be smart to try and use the C++-ish versions of the functions you are used to in C, instead of just keeping to what you know, this is a new language after all. And NO, you are not allowed to use the STL until you actually are supposed to (that is, until d08). That means no vectors/lists/maps/etc... or anything that requires an include <algorithm> until then.
- Actually, the use of any explicitly forbidden function or mechanic will be punished by a `-42`, no questions asked.
- Also note that unless otherwise stated, the `C++` keywords `"using namespace"` and `"friend"` are forbidden. Their use will be punished by a `-42`, no questions asked.
- Files associated with a class will always be `ClassName.hpp` and `ClassName.cpp`, unless specified otherwise.
- Turn-in directories are `ex00/`, `ex01/`, ..., `exn/`.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description. If something seems ambiguous, you don't understand `C++` enough.

- Since you are allowed to use the `C++` tools you learned about since the beginning of the pool, you are not allowed to use any external library. And before you ask, that also means no `C++11` and derivates, nor `Boost` or anything your awesomely skilled friend told you `C++` can't exist without.

- You may be required to turn in an important number of classes. This can seem tedious, unless you're able to script your favorite text editor.

- Read each exercise FULLY before starting it ! Really, do it.

- The compiler to use is `clang++`.

- Your code has to be compiled with the following flags : `-Wall -Wextra -Werror`.

- Each of your includes must be able to be included independently from others. Includes must contains every other includes they are depending on, obviously.

- The subject can be modified up to 4h before the final turn-in time.

- In case you're wondering, no coding style is enforced during the `C++` pool. You can use any style you like, no restrictions. But remember that a code your peer-evaluator can't read is a code she or he can't grade.

- Important stuff now : You will NOT be graded by a program, unless explictly stated in the subject. Therefore, you are afforded a certain amount of freedom in how you choose to do the exercises. However, be mindful of the constraints of each exercise, and DO NOT be lazy, you would miss a LOT of what they have to offer !

- It's not a problem to have some extraneous files in what you turn in, you may choose to separate your code in more files than what's asked of you. Feel free, as long as the day is not graded by a program.

- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.

- By Odin, by Thor ! Use your brain !!!

# Chapter II

# Day-specific rules

- Classes will have to be in Coplien's form, without us asking. It's a pre-requisite to have a positive grade. Be thorough.
- C-style casts are forbidden. You have to use C++-style casts.
- You should try to use the STL as much as possible. It's worth it.
- Your program must compile with a proper Makefile, and the executable must be named `ft_gkrellm`.

# Chapter III

# Foreword

Here's what Wikipedia has to say on the `crowbar`:

`A crowbar, wrecking bar, pry bar, or prybar, pinch-bar or sometimes (in British usage) a prise bar, prisebar, and more informally a jimmy, jimmy bar, jemmy or PikiPiki or gooseneck is a tool consisting of a metal bar with a single curved end and flattened points, often with a small fissure on one or both ends for removing nails. In the United Kingdom, Ireland and Australia, "crowbar" may occasionally be used loosely for this tool, but may also be used to mean a larger straighter tool (see spud bar). The term jemmy or jimmy most often refers to the tool when used for burglary.`

`It is used as a lever either to force apart two objects or to remove nails. Crowbars are commonly used to open nailed wooden crates. Common uses for larger crowbars are: removing nails, prying apart boards, and generally breaking things. Crowbars can be used as any of the three lever classes but the curved end is usually used as a first-class lever, and the flat end as a second class lever. In mining, crowbars are used to break blasted rocks and to remove loose rock on roof sides and the working face, but not as much in modern mining.`

You will fail this project if you do not have a `crowbar`. Maybe `zaz` could lend you his `crowbar` if you offer him enough money.

# Chapter IV

# Subject

## IV.1 Introduction

Let's be clear : We don't want you to fully recode `GKrellM`. But we want you to make a clone of it with at least a few features. This subject is not restrictive at all, so feel free to add whatever you feel is necessary, fun, or interesting. However, a certain number of mandatory steps will have to be validated first !

So, well, have fun and make something cool, but please be rigorous anyway.

## IV.2 Display

Once properly configured and with a sexy skin, `GKrellM` is a very nice system monitor. But to reach this point, there's work to be done!

Your system monitor should be able to be launched in text mode or in graphical mode and retain the same features. So you must keep this in mind when designing your program.

- In text mode, your monitor (read: "System Monitor", not the monitor you plug into a computer to display stuff...) will output to your terminal. You have to use the `ncurses` library for this.

- In graphical mode, your monitor will display in its own window(s). You are entirely free to choose whatever library you want for the graphical interface. Yes, even this graphical library that you are the only one to know and you can't live without. Just remember that you need to install the library yourself without any privilege on your user account.

Whatever the chosen mode, the visual quality and the ergonomics of the display will have a non-negligible impact on your grade. For example, some pieces of data are more suited to a text/numerical display, others will be easier to read as a curve or a bar graph. Also, if it's just plain ugly, it's not worth it. Yes, this is a very subjective term. Yes, whoever grades you will arbitrarily decide if what he sees is ugly or not. No, this is not fair.

Also, `GKrellM` has an awesome skin system. What about your program ?

## IV.3 Design

The software design part is almost entirely at your discretion. However, you MUST create and use at least the two following interfaces :

- `IMonitorModule`, which describes the behavior of one or your monitor's "modules". `IMonitorModule` is an abstraction to the available modules. Implementing this interface allows a class to behave like a module of the monitor, and allows the monitor to handle every modules in an unified way. This, students. This is Object Oriented Programming.

- `IMonitorDisplay`, which describes the behavior of a display mode. `IMonitorDisplay` is an abstraction between the two available displays. Thus, at least two class should implement this interface : the class that handles the shell UI and the one that handles the graphical UI.

What these interfaces actually contain is your problem. Ask yourself why we decided to make them mandatory ...

The bare minimum is to be able to choose which modules are used, and in which order they are displayed, at compile time (We'd call that the "Peasant-mode configuration"). It'd be better if it was possible to select modules when running the program or with a configuration file ("Vaguely-good-programmer mode"), but it'd be best to be able to change this during the execution ("Awesome-mode") ...

## IV.4 Allowed stuff

You will require some system calls to do this subject, such as `fork` or `exec*`. They are all allowed as long as you can explain why you need them. Moreover, everything from `C++(98)` is allowed as long as you respect the instuctions.

## IV.5 Step 1

- Hostname/username module
- OS info module
- Date/time module

## IV.6 Step 2

- CPU module (Model, clock speed, number of cores, activity, ...)
- RAM module

## IV.7 Step 3

- Network throughput module

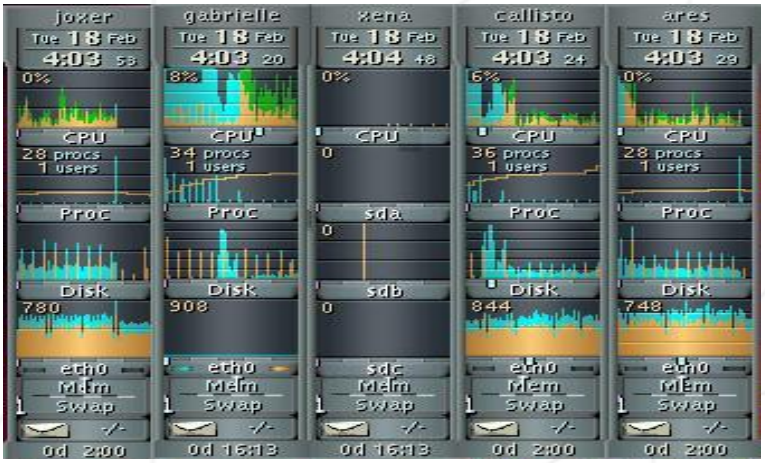
## IV.8 Step 4

Last step of this rush : Create a `metric fuckton` of cool and `USEFUL` additional modules. The definition of "useful" is at the discretion of whoever grades you. So be smart about it. Don't forget that the three first steps must be perfect before this fourth step worth anything ...

`Bonus` : One and only ONE module that represents `zaz`'s favorite animal will ALWAYS count as a useful module. However, it must be animated to be worth anything. If you don't know what `zaz`'s favorite animal is, tough luck, find out.

# Chapter V

# Inspiration




http://en.wikipedia.org/wiki/GKrellM

Have fun !