Functional specifications for `oclude` OpenCL kernel driver

- Inputs:
  – **A *.cl file** containing one or more OpenCL kernel(s).
  – **The name of one of the kernels** included in the *.cl file that needs to be run.
  – **Platform ID (pid)** and **device ID (did)** specifing the hardware on which to run the selected kernel. The user could be prompted for these arguments interactivelly, after identifing the platforms and the devices that exist on the current machine.
  – **A random number** used as seed to populate OpenCL memory objects (the kernel arguments) with random* values. Apart from that, this number could be used to decide the NDRange.
  – **Number of work groups**. Not necessary.

- Outputs:
  – **Time elapsed** while the kernel was executing on the specified platform and device. Pretty straight-forward.
  – **LLVM IR instructions count** of the kernel execution. Not straight-forward at all. Could be undoable given that we want a compiled kernel to actually run it on an OpenCL device.
  – (optionally) **A memory dump** of the kernel arguments, as a kind of kernel execution output.

- Assumptions:
  – No `Image` objects (part of the OpenCL specification, but very rarely used)
  – No `vector` objects (part of the OpenCL specification here, sometimes used, `mergesort.cl` uses them)
  – No user-defined structs (part of the OpenCL specification, sometimes used, `nearestNeighbor_kernel.cl` uses them)

- Possible extentions:
  – **A plugin API**: Instead of random initialization of the kernel arguments, a user may want to run a kernel on real input data. These could be formatted in arbitrary ways (e.g. a csv file). A plugin API could give the user the ability to define a C++ class, say `InputFormatter`, which could process their data and transform them into a form that `oclude` can use (a specification is needed, obviously). This class could be compiled as a shared library and loaded at runtime in order to process the real input data of the user. The same could be done for the output, with an `OutputFormatter`. Click here here to see how `oclgrind` does the same thing.
  – Implementation of **`vector` objects** as kernel arguments.
  – Implementation of **user-defined structs** as kernel arguments.

* Not *entirely* random. The value range for each argument should be dictated by its type, e.g. an argument of the `uint` type should be populated with non-negative integers.