# Background

Modern processors have multiple cores. It is not efficient way to run heavy calculations on a single thread – most of the cores will be idle and underutilized. Running calculations on several threads is better, but not elegant from architecture point of view – there will be too much threads if every calculation module will run its own set of threads.

Instead, we are using deferred tasks approach. There is a single component (DeferredTasksExecutor) that owns a thread pool. Calculation algorithm is split into set of tasks which are submitted to DeferredTasksExecutor. Task is processed on first available idle thread within executor’s thread pool.

# Exercise

Develop DeferredTasksExecutor class that

* Owns a thread pool
* Accepts DeferredTasks
* Manages FIFO tasks queue
* Processes task on first available idle thread

Client code should use a simple interface for submitting and waiting for deferred tasks.

Example tasks may be implemented as usleep(random(1000000)) simulating real heavy calculations

# Advanced requirements

* Code is covered with unit tests (gtest, cppunit, qt test or similar)
* Number of threads in the pool is reasonable for current machine
* Clients have an interface to get the task progress (in queue, processing, done).
* Task has a start precondition - task does not start until precondition is met. This allows sequential task processing - one task waits for other task completion
* There is a way to cancel a task
* Each task has execution priority

# Coding guidelines

* Using of STL, boost or Qt frameworks is a plus
* However, do not use boost::future, QtConcurrent, QThreadPool or QRunnable – we are interested in your own DeferredTasksExecutor implementation. You may use these frameworks as a reference, though.