# Быстрый старт.

Устанавливаем все необходимые для работы инструменты:

1. [Visual Studio 2019 Community](https://visualstudio.microsoft.com/ru/vs/community/) – устанавливается первой, при установке выбрать пункты:
   1. «Разработка классических приложений на C++»;
   2. «Разработка для Linux на C++».
2. [CUDA Toolkit](https://developer.nvidia.com/cuda-toolkit) – при установке увидит установленную Visual Studio, в противном случае следует вручную выбрать средства разработки для Visual Studio, в том числе Nsight Visual Studio Edition.

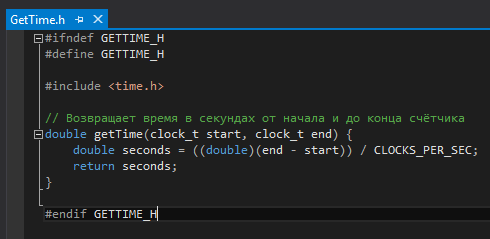
# Module 2. Device Query.

1. Замена функции *wbArg\_read()*.

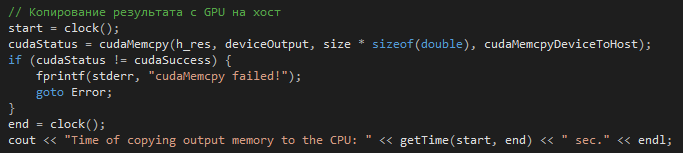
В данном случае не требуется, так как в программу не передаются никакие данные, и её задача – вывести информацию об устройстве.

1. Замена функции *wbTime\_start()*.

Самое простое решение – создать заголовочный файл *GetTime.h* со следующим кодом:



Данная функция принимает на вход 2 аргумента типа *clock\_t*, один из которых инициализируется перед запускаемой операцией (*start*), а второй – после запускаемой операции (*end*). Для работы функции необходимо подключить заголовочный файл *time.h* или *ctime*. Пример использования функции:



1. Замена функции *wbLog()*.

По сути, данная функция выводит указанный аргумент в поток вывода *std::cout*. Вместо неё можно сразу использовать *std::cout*, [*printf()*](https://www.cplusplus.com/reference/cstdio/printf/)или [*fprintf()*](https://www.cplusplus.com/reference/cstdio/fprintf/)с записью в потоки [*stdout*](https://www.cplusplus.com/reference/cstdio/stdout/)или [*stderr*](https://www.cplusplus.com/reference/cstdio/stderr/)(используется в примере выше для функции *getTime()*). При этом не забываем, что *cudaDeviceProp* – всего лишь структура, к полям которой мы получаем доступ (т.е. выводим их в консоль). Пример вывода информации через обычный *printf()*: [ссылка](https://habr.com/ru/post/54330/).

# Module 3. Vector Add.

В данной лабораторной работе векторы передаются через текстовые файлы, которая программа считывает через аргументы командной строки. Попытайтесь обойтись без *wbArg\_read()* следующим образом – количество элементов в обоих векторах вводится пользователем, затем на хосте создаются 2 динамических массива типа *float* или *double*, элементы которых генерируются случайным образом (можно определить функцию, возвращаемое случайное число типа *int* в заданном диапазоне, а при присваивании использовать *static* *cast* к типу *float* или *double*).

Пример сложения векторов без проверки выполнения программы и замеров времени – [ссылка](https://habr.com/ru/post/54707/) (ближе к концу статьи). Также стоит обратить на шаблонный пример, генерируемый при создании CUDA-проекта в Visual Studio – там как раз реализовано сложение векторов, только заданных заранее, с постоянным размером.

# Module 3. Thrust Vector Add.

*Thrust* – это библиотека шаблонов C++ для CUDA, основанная на стандартной библиотеке шаблонов (STL) C++. *Thrust* позволяет реализовывать высокопроизводительные параллельные приложения с минимальными усилиями программирования через высокоуровневый интерфейс, полностью совместимый с CUDA C.

*Thrust* предоставляет богатую коллекцию параллельных структур данных, алгоритмов и примитивов, таких как *scan*, *sort* и *reduce*, которые могут быть использованы вместе для реализации сложных алгоритмов с кратким, читаемым исходным кодом. В результате *Thrust* может быть использован в быстром прототипировании приложений CUDA, где производительность программиста имеет наибольшее значение.

По сути, эта библиотека предоставляет реализацию многих структур данных и алгоритмов из STL. В данной лабораторной работе надо использовать thrust::host\_vector и thrust::device\_vector. Документация на английском языке с примерами [расположена тут](https://docs.nvidia.com/cuda/thrust/index.html).