```cpp
#define __CL_ENABLE_EXCEPTIONS
#include <cl.hpp>
#include <iostream>
#include <vector>
#include <utility>
#include <cstdlib>
using namespace std;
const char * kernel_str  =
   "__kernel void "
   "raiz( __global const float * x, __global float * y ) "
   "{ "
   "   int i = get_global_id(0); "
   "   y[i] = sqrt( x[i] ); "
   "} ";
int main( int argc, char* argv[] )
{
   const int elementos = atoi( argv[1] );
   float *X = new float[elementos];
   for( int i = 0; i < elementos; ++i ) X[i] = i;
   float *Y = new float[elementos];
   // --- Inicialização:
   vector<cl::Platform> plataformas;
   vector<cl::Device> dispositivos;
   cl::Platform::get( &plataformas );
   plataformas[0].getDevices( CL_DEVICE_TYPE_ALL, &dispositivos );
   cl::Context contexto( dispositivos );
   cl::CommandQueue fila( contexto, dispositivos[0] );
   cl::Program::Sources fonte( 1, make_pair( kernel_str, strlen( kernel_str ) ) );
   cl::Program programa( contexto, fonte );
   programa.build( vector<cl::Device>() );
   cl::Kernel kernel( programa, "raiz" );
   // --- Preparação da memória:
   cl::Buffer bufferX( contexto, CL_MEM_READ_ONLY, elementos * sizeof( float ) );
   cl::Buffer bufferY( contexto, CL_MEM_WRITE_ONLY, elementos * sizeof( float ) );
   // --- Execução:
   fila.enqueueWriteBuffer( bufferX, CL_TRUE, 0, elementos * sizeof( float ), X );
   kernel.setArg( 0, bufferX );
   kernel.setArg( 1, bufferY );
   fila.enqueueNDRangeKernel( kernel, cl::NDRange(), cl::NDRange( elementos ), cl::NDRange() );
   fila.finish();
   fila.enqueueReadBuffer( bufferY, CL_TRUE, 0, elementos * sizeof( float ), Y );
   for( int i = 0; i < elementos; ++i ) cout << '[' << Y[i] << ']'; cout << endl;
   delete[] X, Y;
   return 0;
}
```