Report

Standard Headers and Libraries

```
#define CL_USE_DEPRECATED_OPENCL_2_0_APIS   // using OpenCL 1.2, some functions deprecated in OpenCL 2.0
#define __CL_ENABLE_EXCEPTIONS              // enable OpenCL exemptions

// C++ standard library and STL headers
#include <iostream>
#include <vector>
#include <fstream>
#include <iomanip>

// OpenCL header, depending on OS
#ifdef __APPLE__
#include <OpenCL/cl.hpp>
#else
#include <CL/cl.hpp>
#endif

#include "common.h"
```

The #define statements enable the use of deprecated OpenCL 2.0 APIs and the use of OpenCL exceptions in the code.

The #include statements include standard C++ library headers such as iostream, vector, sstream, and fstream, as well as the OpenCL header file cl.hpp, which is specific to the OS being used.

Finally, the #include statement includes a custom header file common.h, which likely contains function declarations or macros used throughout the OpenCL program.

Common.h

```
#pragma once
#ifndef _COMMON_H_
#define _COMMON_H_

#define CL_USE_DEPRECATED_OPENCL_2_0_APIS   // using OpenCL 1.2, some functions deprecated in OpenCL 2.0
#define __CL_ENABLE_EXCEPTIONS              // enable OpenCL exemptions

// C++ standard library and STL headers
#include <iostream>
#include <vector>
#include <sstream>
#include <fstream>

// OpenCL header, depending on OS
#ifdef __APPLE__
#include <OpenCL/cl.hpp>
#else
#include <CL/cl.hpp>
#endif

// function to handle error
void handle_error(cl::Error e);

// outputs message then quits
void quit_program(const std::string str);

// looks up and displays OpenCL error code as a string
const std::string lookup_error_code(cl_int error_code);

// allows the user to select a device, displays the available platform and device options
// returns whether selection was successful, the selected device and its platform
bool select_one_device(cl::Platform* platfm, cl::Device* dev);

// builds program from given filename
bool build_program(cl::Program* prog, const cl::Context* ctx, const std::string filename);

#endif
```

The header file includes the same preprocessor directives and header files as in the previous code snippet. Additionally, it declares several functions used throughout the OpenCL program: handle_error(), quit_program(), lookup_error_code(), select_one_device(), and build_program().

The handle_error() function handles OpenCL errors by outputting the error message and code to the console. The quit_program() function outputs a message and exits the program. The lookup_error_code() function converts OpenCL error codes into human-readable strings. The select_one_device() function allows the user to select an OpenCL device and returns whether the selection was successful along with the selected device and its platform. The build_program() function builds an OpenCL program from a given filename.

Common.cpp

Select one device function

```
bool select_one_device(cl::Platform* platfm, cl::Device* dev)
{
    std::vector<cl::Platform> platforms;    // available platforms
    std::vector< std::vector<cl::Device> > platformDevices; // devices available for each platform
    std::string outputString;               // string for output
    unsigned int i, j;                      // counters
```

The function is defined with two pointer parameters, platfm and dev, which will be used to return the selected OpenCL platform and device, respectively.

The function declares several variables to be used later. platforms is a vector that will store all available OpenCL platforms, and platformDevices is a vector of vectors that will store all available devices for each platform. outputString is a string that will be used to store various output messages, and i and j are counters that will be used in the for-loops later.

```
try {
    // get the number of available OpenCL platforms
    cl::Platform::get(&platforms);
    std::cout << "Number of OpenCL platforms: " << platforms.size() << std::endl;

    // find and store the devices available to each platform
    for (i = 0; i < platforms.size(); i++)
    {
        std::vector<cl::Device> devices;        // available devices

        // get all devices available to the platform
        platforms[i].getDevices(CL_DEVICE_TYPE_ALL, &devices);

        // store available devices for the platform
        platformDevices.push_back(devices);
    }
```

The function starts by using the cl::Platform::get() method to retrieve the number of available OpenCL platforms and stores them in the platforms vector. Then, for each platform, the function retrieves all available devices using the getDevices() method and stores them in platformDevices.

```
// display available platforms and devices
std::cout << "---------------------" << std::endl;
std::cout << "Available options:" << std::endl;

// store options as platform and device indices
std::vector< std::pair<int, int> > options;
unsigned int optionCounter = 0; // option counter

// for all platforms
for (i = 0; i < platforms.size(); i++)
{
    // for all devices per platform
    for (j = 0; j < platformDevices[i].size(); j++)
    {
        // display options
        std::cout << "Option " << optionCounter << ": Platform - ";

        // platform vendor name
        outputString = platforms[i].getInfo<CL_PLATFORM_VENDOR>();
        std::cout << outputString << ", Device - ";

        // device name
        outputString = platformDevices[i][j].getInfo<CL_DEVICE_NAME>();
        std::cout << outputString << std::endl;

        // store option
        options.push_back(std::make_pair(i, j));
        optionCounter++; // increment option counter
    }
}
```

The function then displays all available options for the user to select from. It first declares an options vector that will store all the available platform and device indices for each option. It also declares an optionCounter variable that will be used to keep track of the number of available options.

For each platform and device combination, the function displays an option number, the platform vendor name, and the device name. It then stores the platform and device indices in the options vector and increments the optionCounter.

```
    std::cout << "\n----------------------" << std::endl;
    std::cout << "Select a device: ";

    std::string inputString;
    unsigned int selectedOption;    // option that was selected

    std::getline(std::cin, inputString);
    std::istringstream stringStream(inputString);

    // check whether valid option selected
    // check if input was an integer
    if (stringStream >> selectedOption)
    {
        char c;

        // check if there was anything after the integer
        if (!(stringStream >> c))
        {
            // check if valid option range
            if (selectedOption >= 0 && selectedOption < optionCounter)
            {
                // return the platform and device
                int platformNumber = options[selectedOption].first;
                int deviceNumber = options[selectedOption].second;

                *platfm = platforms[platformNumber];
                *dev = platformDevices[platformNumber][deviceNumber];

                return true;
            }
        }
    }
    // if invalid option selected
    std::cout << "\n----------------------" << std::endl;
    std::cout << "Invalid option." << std::endl;
}
// catch any OpenCL function errors
catch (cl::Error e) {
    // call function to handle errors
    handle_error(e);
}

return false;
```

Finally, the function prompts the user to select an option by entering its corresponding number. The user's input is stored in a string called inputString, which is then converted to an integer using istringstream.

The function then checks whether the selected option is a valid integer, whether there are no extra characters after the integer, and whether the selected option is within the range of available options. If all checks pass, the function returns the selected platform and device by setting the values of platfm and dev pointers and returns true to indicate success.

If any of the checks fail, the function displays an error message and returns false to indicate failure.

If there are any OpenCL function errors, they are caught by a try-catch block and handled by calling the handle_error() function.

Build program function

```
bool build_program(cl::Program* prog, const cl::Context* ctx, const std::string filename)
{
    // get devices from the context
    std::vector<cl::Device> contextDevices = ctx->getInfo<CL_CONTEXT_DEVICES>();

    // open input file stream to .cl file
    std::ifstream programFile(filename);

    // check whether file was opened
    if (!programFile.is_open())
    {
        std::cout << "File not found." << std::endl;
        return false;
    }
```

This function builds an OpenCL program from a source file. It takes three arguments, a pointer to a cl::Program object, a pointer to a cl::Context object, and a string filename of the source file. The function returns a boolean value to indicate whether the program build was successful.

The function first retrieves the devices associated with the given OpenCL context and stores them in a vector called contextDevices.

The function then opens the source file with the given filename using an input file stream called programFile. If the filename is invalid, the function outputs an error message and returns false.

```
// create program string and load contents from the file
std::string programString(std::istreambuf_iterator<char>(programFile), (std::istreambuf_iterator<char>()));

// output contents of the file
std::cout << "--------------------" << std::endl;
std::cout << "Contents of program string: " << std::endl;
std::cout << programString << std::endl;
std::cout << "--------------------" << std::endl;

// create program source from one input string
cl::Program::Sources source(1, std::make_pair(programString.c_str(), programString.length() + 1));
// create program from source
*prog = cl::Program(*ctx, source);
```

The function then creates a string called programString and loads its contents from the file using the istreambuf_iterator<char> method. This method reads the file contents and stores them in programString.
The function then outputs the contents of programString to the console to verify that the correct file was loaded.
The function creates a cl::Program::Sources object called source that contains the program source code in programString. It then creates a cl::Program object by calling the cl::Program constructor with the context and source as arguments. The resulting program is stored in the prog pointer variable.

```
// try to build program
try {
    // build the program for the devices in the context
    prog->build(contextDevices);

    std::cout << "Program build: Successful" << std::endl;
    std::cout << "----------------------" << std::endl;
}
catch (cl::Error e) {
    // if failed to build program
    if (e.err() == CL_BUILD_PROGRAM_FAILURE)
    {
        // output program build log
        std::cout << e.what() << ": Failed to build program." << std::endl;

        // check build status for all all devices in context
        for (unsigned int i = 0; i < contextDevices.size(); i++)
        {
            // get device's program build status and check for error
            // if build error, output build log
            if (prog->getBuildInfo<CL_PROGRAM_BUILD_STATUS>(contextDevices[i]) == CL_BUILD_ERROR)
            {
                // get device name and build log
                std::string outputString = contextDevices[i].getInfo<CL_DEVICE_NAME>();
                std::string build_log = prog->getBuildInfo<CL_PROGRAM_BUILD_LOG>(contextDevices[i]);

                std::cout << "Device - " << outputString << ", build log:" << std::endl;
                std::cout << build_log << "----------------------" << std::endl;
            }
        }

        return false;
    }
    else
    {
        // call function to handle errors
        handle_error(e);
    }
}

return true;
```

The function then tries to build the program using the devices in the contextDevices vector. If the build is successful, the function outputs a message indicating success.

If the build fails, the function catches a cl::Error exception and checks whether the error is due to a build failure. If so, the function outputs an error message and the build log for each device in the context that experienced a build error. The function then returns false to indicate failure.

Handle error & quit program function

```cpp
// function to handle error
void handle_error(cl::Error e)
{
    // output OpenCL function that cause the error and the error code
    std::cout << "Error in: " << e.what() << std::endl;
    std::cout << "Error code: " << e.err() << " (" << lookup_error_code(e.err()) << ")" << std::endl;
}

// function to quit program
void quit_program(const std::string str)
{
    std::cout << str << std::endl;
    std::cout << "Exiting the program..." << std::endl;

#ifdef _WIN32
    // wait for a keypress on Windows OS before exiting
    std::cout << "\npress a key to quit...";
    std::cin.ignore();
#endif

    exit(1);
}
```

The handle_error() function takes a cl::Error object as an argument and outputs the OpenCL function that caused the error and the error code. It does not return a value.

The quit_program() function takes a string argument that is printed to the console along with a message indicating that the program is exiting

Error Codes

```
// function to lookup and return error code string
const std::string lookup_error_code(cl_int error_code)
{
    // look up error codes as defined in cl.hpp
    switch (error_code) {
    case CL_SUCCESS:
        return "CL_SUCCESS";
    case CL_DEVICE_NOT_FOUND:
        return "CL_DEVICE_NOT_FOUND";
    case CL_DEVICE_NOT_AVAILABLE:
        return "CL_DEVICE_NOT_AVAILABLE";
    case CL_COMPILER_NOT_AVAILABLE:
        return "CL_COMPILER_NOT_AVAILABLE";
    case CL_MEM_OBJECT_ALLOCATION_FAILURE:
        return "CL_MEM_OBJECT_ALLOCATION_FAILURE";
    case CL_OUT_OF_RESOURCES:
        return "CL_OUT_OF_RESOURCES";
    case CL_OUT_OF_HOST_MEMORY:
        return "CL_OUT_OF_HOST_MEMORY";
    case CL_PROFILING_INFO_NOT_AVAILABLE:
        return "CL_PROFILING_INFO_NOT_AVAILABLE";
    case CL_MEM_COPY_OVERLAP:
        return "CL_MEM_COPY_OVERLAP";
    case CL_IMAGE_FORMAT_MISMATCH:
        return "CL_IMAGE_FORMAT_MISMATCH";
    case CL_IMAGE_FORMAT_NOT_SUPPORTED:
        return "CL_IMAGE_FORMAT_NOT_SUPPORTED";
    case CL_BUILD_PROGRAM_FAILURE:
        return "CL_BUILD_PROGRAM_FAILURE";
    case CL_MAP_FAILURE:
        return "CL_MAP_FAILURE";
    case CL_MISALIGNED_SUB_BUFFER_OFFSET:
        return "CL_MISALIGNED_SUB_BUFFER_OFFSET";
    case CL_EXEC_STATUS_ERROR_FOR_EVENTS_IN_WAIT_LIST:
        return "CL_EXEC_STATUS_ERROR_FOR_EVENTS_IN_WAIT_LIST";
    case CL_COMPILE_PROGRAM_FAILURE:
        return "CL_COMPILE_PROGRAM_FAILURE";
    case CL_LINKER_NOT_AVAILABLE:
        return "CL_LINKER_NOT_AVAILABLE";
    case CL_LINK_PROGRAM_FAILURE:
        return "CL_LINK_PROGRAM_FAILURE";
    case CL_DEVICE_PARTITION_FAILED:
        return "CL_DEVICE_PARTITION_FAILED";
    case CL_KERNEL_ARG_INFO_NOT_AVAILABLE:
        return "CL_KERNEL_ARG_INFO_NOT_AVAILABLE";

    case CL_INVALID_VALUE:
        return "CL_INVALID_VALUE";
    case CL_INVALID_DEVICE_TYPE:
        return "CL_INVALID_DEVICE_TYPE";
    case CL_INVALID_PLATFORM:
        return "CL_INVALID_PLATFORM";
    case CL_INVALID_DEVICE:
        return "CL_INVALID_DEVICE";
    case CL_INVALID_CONTEXT:
        return "CL_INVALID_CONTEXT";
    case CL_INVALID_QUEUE_PROPERTIES:
        return "CL_INVALID_QUEUE_PROPERTIES";
    case CL_INVALID_COMMAND_QUEUE:
        return "CL_INVALID_COMMAND_QUEUE";
    case CL_INVALID_HOST_PTR:
        return "CL_INVALID_HOST_PTR";
    case CL_INVALID_MEM_OBJECT:
        return "CL_INVALID_MEM_OBJECT";
    case CL_INVALID_IMAGE_FORMAT_DESCRIPTOR:
        return "CL_INVALID_IMAGE_FORMAT_DESCRIPTOR";
    case CL_INVALID_IMAGE_SIZE:
        return "CL_INVALID_IMAGE_SIZE";
    case CL_INVALID_SAMPLER:
        return "CL_INVALID_SAMPLER";
    case CL_INVALID_BINARY:
        return "CL_INVALID_BINARY";
```

The lookup_error_code() function takes an error code of type cl_int as an argument and returns a string representing the error code. The function contains a switch statement that maps each error code to its corresponding string representation as defined in the OpenCL header file cl.hpp.

If the error code is not defined in the switch statement, the function returns a string "Unknown error code". This function is used to provide a more user-friendly output when an OpenCL error occurs by converting the error code into a human-readable format.

Task1.cpp

declaring a number of data structures and variables

```
int main(void)
{
    cl::Platform platform;      // device's platform
    cl::Device device;          // device used
    cl::Context context;        // context for the device
    cl::CommandQueue commandQueue; // Command Queue
    std::string outputString;   // string for output
    cl::Program program;            // OpenCL program object
    cl::Kernel kernel;              // a single kernel object
    unsigned int i;             // counter
```

These lines declare variables that will be used throughout the program, such as platform, device, context, command queue, and program.

Select Device , Creating context

```
try {
    // select an OpenCL device
    if (!select_one_device(&platform, &device))
    {
        // if no device selected
        quit_program("Device not selected.");
    }

    context = cl::Context(device);

    std::vector<cl::Device> contextDevices = context.getInfo<CL_CONTEXT_DEVICES>();

    std::cout << "\nDevices in the context:" << std::endl;
```

This section of code attempts to select an OpenCL device. If no device is selected, the program will terminate with an error message. The context is constructed, getInfo is called on the context to retrieve a vector of devices in the context. The function loops through the devices in the vector and prints various properties of each device.

Device Properties

```
for (i = 0; i < contextDevices.size(); i++)
{
    std::cout << "\nSelected device information: " << std::endl;
    cl::Platform platform = contextDevices[i].getInfo<CL_DEVICE_PLATFORM>();
    std::string platformName = platform.getInfo<CL_PLATFORM_NAME>();
    std::cout << "Platform: "<< platformName << std::endl;

    cl_device_type deviceType = contextDevices[i].getInfo<CL_DEVICE_TYPE>();
    std::cout << "Device type: " << (deviceType == CL_DEVICE_TYPE_CPU ? "CPU" : "GPU") << std::endl;

    outputString = contextDevices[i].getInfo<CL_DEVICE_NAME>();
    std::cout << "Device name: " << outputString << std::endl;

    cl_uint numComputeUnits = contextDevices[i].getInfo<CL_DEVICE_MAX_COMPUTE_UNITS>();
    std::cout << "Number of computing units: " << numComputeUnits << std::endl;

    size_t maxWorkGroupSize = contextDevices[i].getInfo<CL_DEVICE_MAX_WORK_GROUP_SIZE>();
    std::cout << "Maximum work group size: " << maxWorkGroupSize << std::endl;

    cl_uint maxWorkItemDimensions = contextDevices[i].getInfo<CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS>();
    std::cout << "Maximum number of work item dimensions: " << maxWorkItemDimensions << std::endl;

    std::vector<size_t> maxWorkItemSizes = contextDevices[i].getInfo<CL_DEVICE_MAX_WORK_ITEM_SIZES>();
    std::cout << "Maximum work item sizes: ";
    for (size_t i = 0; i < maxWorkItemDimensions; i++) {
        std::cout << maxWorkItemSizes[i] << " ";
    }
    std::cout << std::endl;

    cl_uint preferredVectorWidth = contextDevices[i].getInfo<CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT>();
    std::cout << "Preferred vector width for integers: " << preferredVectorWidth << std::endl;

    cl_ulong localMemSize = contextDevices[i].getInfo<CL_DEVICE_LOCAL_MEM_SIZE>();
    std::cout << "Local memory size: " << localMemSize << std::endl;

    std::string extensionsString = contextDevices[i].getInfo<CL_DEVICE_EXTENSIONS>();
    std::istringstream iss(extensionsString);
    std::vector<std::string> extensions(std::istream_iterator<std::string>{iss}, std::istream_iterator<std::string>());
    bool fp16Supported = false, fp64Supported = false, icdSupported = false;

    for (const auto& extension : extensions) {
        if (extension == "cl_khr_fp16") {
            fp16Supported = true;
        }
        else if (extension == "cl_khr_fp64") {
            fp64Supported = true;
        }
        else if (extension == "cl_khr_icd") {
            icdSupported = true;
        }
    }
    std::cout << "\nSelected device extension support:" << std::endl;
    std::cout << "cl_khr_fp16: " << (fp16Supported ? "Supported" : "Not supported") << std::endl;
    std::cout << "cl_khr_fp64: " << (fp64Supported ? "Supported" : "Not supported") << std::endl;
    std::cout << "cl_khr_icd: " << (icdSupported ? "Supported" : "Not supported") << std::endl;
    std::cout << "----------------------" << std::endl;
}
std::cout << "----------------------" << std::endl;
```

The properties that are printed for each device include:

Platform name: The name of the platform that the device belongs to.
Device type: Whether the device is a CPU or GPU.
Device name: The name of the device.
Number of computing units: The number of compute units on the device.
Maximum work group size: The maximum size of a work group that can be used for the device.

Maximum number of work item dimensions: The maximum number of dimensions that can be used for work items.
Maximum work item sizes: The maximum size of work items in each dimension.
Preferred vector width for integers: The preferred vector width for integer operations.
Local memory size: The size of the device's local memory.
Selected device extension support: Whether the device supports the cl_khr_fp16, cl_khr_fp64, and cl_khr_icd extensions.

Kernel creation & Print Kernels

```
        // create command queue
        commandQueue = cl::CommandQueue(context, device);

        std::cout << "Command queue created." << std::endl;

        // build the program
        if (!build_program(&program, &context, "task1.cl"))
        {
            // if OpenCL program build error
            quit_program("OpenCL program build error.");
        }

        std::vector<cl::Kernel> allKernels;     // all kernels
        // create all kernels in the program
        program.createKernels(&allKernels);

        std::cout << "---------------------" << std::endl;
        std::cout << "Kernel names" << std::endl;

        // output kernel name for each index
        for (i = 0; i < allKernels.size(); i++) {
            outputString = allKernels[i].getInfo<CL_KERNEL_FUNCTION_NAME>();
            std::cout << "Kernel " << i << ": " << outputString << std::endl;
        }

        std::cout << "---------------------" << std::endl;

    }
    // catch any OpenCL function errors
    catch (cl::Error e) {
        // call function to handle errors
        handle_error(e);
    }



ifdef _WIN32
    // wait for a keypress on Windows OS before exiting
    std::cout << "\npress a key to quit...";
    std::cin.ignore();
endif

    return 0;
```

The command queue is created using the cl::CommandQueue constructor with the context and device as arguments.
the program is built using the build_program function, if invalid terminate
After the program is built, a vector of kernels is created using the createKernels function of the program object, and all kernels in the program are added to the vector.
Finally, the kernel names are printed using a for loop that iterates over the vector of kernels and prints the name of each kernel using the getInfo function with the CL_KERNEL_FUNCTION_NAME argument
If any OpenCL function errors occur, the handle_error function is called to handle them, followed by the keypress to exit program

Task 1 Program screenshots

```
Number of OpenCL platforms: 1
--------------------
Available options:
Option 0: Platform - NVIDIA Corporation, Device - NVIDIA GeForce RTX 3080

--------------------
Select a device: 0

Devices in the context:

Selected device information:
Platform: NVIDIA CUDA
Device type: GPU
Device name: NVIDIA GeForce RTX 3080
Number of computing units: 68
Maximum work group size: 1024
Maximum number of work item dimensions: 3
Maximum work item sizes: 1024 1024 64
Preferred vector width for integers: 1
Local memory size: 49152

Selected device extension support:
cl_khr_fp16: Not supported
cl_khr_fp64: Supported
cl_khr_icd: Supported
--------------------
--------------------
Command queue created.
--------------------
Contents of program string:
__kernel void copy(__global float *a,
                   __global float *b) {
   *b = *a;
}

__kernel void add(__global float *a,
                  __global float *b,
                  __global float *c) {
   *c = *a + *b;
}

__kernel void sub(__global float *a,
                  __global float *b,
                  __global float *c) {
   *c = *a - *b;
}

__kernel void mult(__global float *a,
                   __global float *b,
                   __global float *c) {
   *c = *a * *b;
}

__kernel void div(__global float *a,
                  __global float *b,
                  __global float *c) {
   *c = *a / *b;
}

--------------------
Program build: Successful
--------------------
--------------------
Kernel names
Kernel 0: copy
Kernel 1: add
Kernel 2: sub
Kernel 3: mult
Kernel 4: div
--------------------
```

Task2.cpp

declaring a number of data structures and variables

```
//declaring a number of data structures and variables
cl::Platform platform;          // device's platform
cl::Device device;              // device used
cl::Context context;            // context for the device
cl::Program program;            // OpenCL program object
cl::Kernel kernel;              // a single kernel object
cl::CommandQueue queue;         // commandqueue for a context and device

// declare vectors
std::vector<unsigned char> alphabets(52);
std::vector<unsigned char> result_char(52);
std::vector<unsigned int> integers(1024);
std::vector<unsigned int> result_int(1024);

// declare data and memory objects
cl::Buffer alphabetsBuffer;
cl::Buffer writeBuffer;
cl::Buffer readWriteBuffer;
```

Declares several data structures and variables of OpenCL classes, such as Platform, Device, Context, Program, Kernel, and CommandQueue
Declares vectors of 52 or 1024 such as alphabets, results_char,integers and result_int
Declares three buffer objects such as alphabetsBuffer, writeBuffer, readWriteBuffer

Initializing Vectors

```
// initialize alphabets vector
for (int i = 0; i < 26; i++) {
    alphabets[i] = 'z' - i;
    alphabets[i + 26] = 'Z' - i;
}

 //initialize integers vector
for (int i = 0; i < 1024; i++) {
    integers[i] = i;
}
```

the code initializes the 'alphabets' and 'integers' vectors. The 'alphabets' vector is initialized with the English alphabets (both uppercase and lowercase), while the 'integers' vector is initialized with values from 0 to 1023.

Select Device , Creating context and build program

```
// select an OpenCL device
if (!select_one_device(&platform, &device))
{
    // if no device selected
    quit_program("Device not selected.");
}

// create a context from device
context = cl::Context(device);

// build the program
if(!build_program(&program, &context, "task2.cl"))
{
    // if OpenCL program build error
    quit_program("OpenCL program build error.");
}
```

This section of code attempts to select an OpenCL device. If no device is selected, the program will terminate with an error message. The context is constructed then the program is built using the build_program function

Kernel creation & buffer creation

```
// create a kernel
kernel = cl::Kernel(program, "task2");

// create command queue
queue = cl::CommandQueue(context, device);

//// create buffers

alphabetsBuffer = cl::Buffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
    sizeof(unsigned char) * alphabets.size(), alphabets.data());

writeBuffer = cl::Buffer(context, CL_MEM_WRITE_ONLY, sizeof(unsigned char) * 52);

readWriteBuffer = cl::Buffer (context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR,
    sizeof(unsigned int) * integers.size(), integers.data());
```

kernel object is created using the OpenCL program object program and the kernel name "task2".
The command queue is created using the cl::CommandQueue constructor with the context and device as arguments.
alphabetsBuffer: a read-only buffer to hold the alphabets vector data, which is copied from host memory to device memory using CL_MEM_COPY_HOST_PTR flag. The buffer is created with a size of sizeof(unsigned char) * alphabets.size() bytes.
writeBuffer: a write-only buffer with a size of sizeof(unsigned char) * 52) bytes, to hold the output of the kernel. It will be written to by the kernel and read from by the host.

readWriteBuffer: a read-write buffer with a size of sizeof(unsigned int) * integers.size() bytes, to hold the integers vector data, which is also copied from host memory to device memory using CL_MEM_COPY_HOST_PTR flag. The buffer will be read from and written to by the kernel.

Enqueue two OpenCL commands

```
// enqueue a command to copy the content from the first buffer to the second buffer
queue.enqueueCopyBuffer(alphabetsBuffer, writeBuffer, 0, 0, sizeof(unsigned char) * alphabets.size());


// enqueue a command to write the content from the vector of 1024 integers into the third buffer
queue.enqueueWriteBuffer(readWriteBuffer, CL_TRUE, 0, sizeof(unsigned int) * integers.size(), integers.data());
```

queue.enqueueCopyBuffer is copying the content of the alphabetsBuffer to the writeBuffer with an offset of 0 and a size of sizeof(unsigned char) * alphabets.size(). This is essentially copying the content of one buffer to another.

queue.enqueueWriteBuffer is writing the content of the integers vector to the readWriteBuffer with an offset of 0 and a size of sizeof(unsigned int) * integers.size(). This is essentially writing data from the host to the device buffer.

The CL_TRUE parameter ensures that the command is blocking, meaning that the function will not return until the write operation has completed.

Kernel Execution and set Arguments

```
// set kernel arguments
float arg1 = 12.45;
kernel.setArg(0, arg1);
kernel.setArg(1, writeBuffer);
kernel.setArg(2, readWriteBuffer);

// enqueue kernel for execution
queue.enqueueTask(kernel);

std::cout << "Kernel enqueued." << std::endl;
std::cout << "--------------------" << std::endl;
```

The code sets the arguments for the OpenCL kernel named kernel. It sets the first argument to a float value of 12.45, the second argument to the writeBuffer buffer, and the third argument to the readWriteBuffer buffer. These arguments will be passed to the kernel when it is executed.

The code enqueues the kernel task2 for execution on the command queue. The previously set kernel arguments will be used during execution.

ReadBuffer Character & print Character results

```
// enqueue command to read from device to host memory
queue.enqueueReadBuffer(writeBuffer, CL_TRUE, 0, sizeof(unsigned char) * 52, result_char.data());
// display the results on screen
std::cout << "\nContents of character result: " << std::endl;
for (int i = 0; i < 52; i++)
{
        std::cout << result_char[i] << " ";
        // Check if we have printed 10 values already
        if ((i + 1) % 10 == 0) {
            std::cout << std::endl;
        }
        else {
            // If we haven't printed 10 values, print a space
            std::cout << " ";
        }
}
```

The code is reading the results of the OpenCL kernel execution from the writeBuffer and storing them in a vector called result_char. Then it is displaying the contents of the result_char vector on the screen. The vector contains 52 elements of type unsigned char, which represent the uppercase and lowercase letters of the English alphabet. The code uses a loop to print the elements of the vector and inserts a newline after every 10th element to format the output nicely.

ReadBuffer Integer & print Integer results

```
//enqueue command to read from device to host memory
queue.enqueueReadBuffer(readWriteBuffer, CL_TRUE, 0, sizeof(unsigned int) * 1024, result_int.data());
std::cout << "\nContents of integer result : " << std::endl;
for (int i = 0; i < 1024; i++)
{
        std::cout << std::setw(5) << result_int[i] << " ";
        // Check if we have printed 10 values already
        if ((i + 1) % 10 == 0) {
            std::cout << std::endl;
        }
        else {
            // If we haven't printed 10 values, print a space
            std::cout << " ";
        }
}
```

The code is similar to the previous one, but instead of reading unsigned char values from the device, it reads unsigned int values from the readWriteBuffer. The results are stored in a vector called result_int, and then displayed on the screen using a loop that formats the output to be more readable. The vector contains 1024 elements of type unsigned int, which represent the index of each integer in the integers vector that was initially passed to the kernel.

Error handling and quitting program

```
    // catch any OpenCL function errors
    catch (cl::Error e) {
        // call function to handle errors
        handle_error(e);
    }

#ifdef _WIN32
    // wait for a keypress on Windows OS before exiting
    std::cout << "\npress a key to quit...";
    std::cin.ignore();
#endif

    return 0;
}
```

If any OpenCL function errors occur, the handle_error function is called to handle them, followed by the keypress to exit program

Task 2 Program screenshots

```
Number of OpenCL platforms: 1
--------------------
Available options:
Option 0: Platform - NVIDIA Corporation, Device - NVIDIA GeForce RTX 3080

--------------------
Select a device: 0
--------------------
Contents of program string:
__kernel void task2(float value,
                                        __global unsigned char *copied,
                                        __global unsigned int *integers)
{
}

--------------------
Program build: Successful
--------------------
Kernel enqueued.
--------------------

Contents of character result:
z  y  x  w  v  u  t  s  r  q
p  o  n  m  l  k  j  i  h  g
f  e  d  c  b  a  Z  Y  X  W
V  U  T  S  R  Q  P  O  N  M
L  K  J  I  H  G  F  E  D  C
B  A
Contents of integer result :
    0       1       2       3       4       5       6       7       8       9
   10      11      12      13      14      15      16      17      18      19
   20      21      22      23      24      25      26      27      28      29
   30      31      32      33      34      35      36      37      38      39
   40      41      42      43      44      45      46      47      48      49
   50      51      52      53      54      55      56      57      58      59
   60      61      62      63      64      65      66      67      68      69
   70      71      72      73      74      75      76      77      78      79
   80      81      82      83      84      85      86      87      88      89
   90      91      92      93      94      95      96      97      98      99
  100     101     102     103     104     105     106     107     108     109
  110     111     112     113     114     115     116     117     118     119
  120     121     122     123     124     125     126     127     128     129
  130     131     132     133     134     135     136     137     138     139
  140     141     142     143     144     145     146     147     148     149
  150     151     152     153     154     155     156     157     158     159
  160     161     162     163     164     165     166     167     168     169
  170     171     172     173     174     175     176     177     178     179
  180     181     182     183     184     185     186     187     188     189
  190     191     192     193     194     195     196     197     198     199
  200     201     202     203     204     205     206     207     208     209
  210     211     212     213     214     215     216     217     218     219
  220     221     222     223     224     225     226     227     228     229
  230     231     232     233     234     235     236     237     238     239
  240     241     242     243     244     245     246     247     248     249
  250     251     252     253     254     255     256     257     258     259
  260     261     262     263     264     265     266     267     268     269
  270     271     272     273     274     275     276     277     278     279
  280     281     282     283     284     285     286     287     288     289
  290     291     292     293     294     295     296     297     298     299
  300     301     302     303     304     305     306     307     308     309
  310     311     312     313     314     315     316     317     318     319
  320     321     322     323     324     325     326     327     328     329
  330     331     332     333     334     335     336     337     338     339
  340     341     342     343     344     345     346     347     348     349
  350     351     352     353     354     355     356     357     358     359
  360     361     362     363     364     365     366     367     368     369
  370     371     372     373     374     375     376     377     378     379
  380     381     382     383     384     385     386     387     388     389
  390     391     392     393     394     395     396     397     398     399
  400     401     402     403     404     405     406     407     408     409
```

```
 400    401    402    403    404    405    406    407    408    409
 410    411    412    413    414    415    416    417    418    419
 420    421    422    423    424    425    426    427    428    429
 430    431    432    433    434    435    436    437    438    439
 440    441    442    443    444    445    446    447    448    449
 450    451    452    453    454    455    456    457    458    459
 460    461    462    463    464    465    466    467    468    469
 470    471    472    473    474    475    476    477    478    479
 480    481    482    483    484    485    486    487    488    489
 490    491    492    493    494    495    496    497    498    499
 500    501    502    503    504    505    506    507    508    509
 510    511    512    513    514    515    516    517    518    519
 520    521    522    523    524    525    526    527    528    529
 530    531    532    533    534    535    536    537    538    539
 540    541    542    543    544    545    546    547    548    549
 550    551    552    553    554    555    556    557    558    559
 560    561    562    563    564    565    566    567    568    569
 570    571    572    573    574    575    576    577    578    579
 580    581    582    583    584    585    586    587    588    589
 590    591    592    593    594    595    596    597    598    599
 600    601    602    603    604    605    606    607    608    609
 610    611    612    613    614    615    616    617    618    619
 620    621    622    623    624    625    626    627    628    629
 630    631    632    633    634    635    636    637    638    639
 640    641    642    643    644    645    646    647    648    649
 650    651    652    653    654    655    656    657    658    659
 660    661    662    663    664    665    666    667    668    669
 670    671    672    673    674    675    676    677    678    679
 680    681    682    683    684    685    686    687    688    689
 690    691    692    693    694    695    696    697    698    699
 700    701    702    703    704    705    706    707    708    709
 710    711    712    713    714    715    716    717    718    719
 720    721    722    723    724    725    726    727    728    729
 730    731    732    733    734    735    736    737    738    739
 740    741    742    743    744    745    746    747    748    749
 750    751    752    753    754    755    756    757    758    759
 760    761    762    763    764    765    766    767    768    769
 770    771    772    773    774    775    776    777    778    779
 780    781    782    783    784    785    786    787    788    789
 790    791    792    793    794    795    796    797    798    799
 800    801    802    803    804    805    806    807    808    809
 810    811    812    813    814    815    816    817    818    819
 820    821    822    823    824    825    826    827    828    829
 830    831    832    833    834    835    836    837    838    839
 840    841    842    843    844    845    846    847    848    849
 850    851    852    853    854    855    856    857    858    859
 860    861    862    863    864    865    866    867    868    869
 870    871    872    873    874    875    876    877    878    879
 880    881    882    883    884    885    886    887    888    889
 890    891    892    893    894    895    896    897    898    899
 900    901    902    903    904    905    906    907    908    909
 910    911    912    913    914    915    916    917    918    919
 920    921    922    923    924    925    926    927    928    929
 930    931    932    933    934    935    936    937    938    939
 940    941    942    943    944    945    946    947    948    949
 950    951    952    953    954    955    956    957    958    959
 960    961    962    963    964    965    966    967    968    969
 970    971    972    973    974    975    976    977    978    979
 980    981    982    983    984    985    986    987    988    989
 990    991    992    993    994    995    996    997    998    999
1000   1001   1002   1003   1004   1005   1006   1007   1008   1009
1010   1011   1012   1013   1014   1015   1016   1017   1018   1019
1020   1021   1022   1023
ress a key to quit...
```

Task3.cpp

declaring a number of data structures and variables

```
int main(void)
{
    //declaring a number of data structures and variables
    cl::Platform platform;          // device's platform
    cl::Device device;              // device used
    cl::Context context;            // context for the device
    cl::Program program;            // OpenCL program object
    cl::Kernel kernel;              // a single kernel object
    cl::CommandQueue queue;         // commandqueue for a context and device

    // declare data and memory objects
    std::vector<int> array(512);
    cl::Buffer arrayBuffer;
```

Declares several data structures and variables of OpenCL classes, such as Platform, Device, Context, Program, Kernel, and CommandQueue
Declares vector of 512 such array
Declares buffer object such as arrayBuffer

Select Device , Creating context and build program

```
try {
    // select an OpenCL device
    if (!select_one_device(&platform, &device))
    {
        // if no device selected
        quit_program("Device not selected.");
    }

    // create a context from device
    context = cl::Context(device);

    // build the program
    if(!build_program(&program, &context, "task3.cl"))
    {
        // if OpenCL program build error
        quit_program("OpenCL program build error.");
    }
```

This section of code attempts to select an OpenCL device. If no device is selected, the program will terminate with an error message. The context is constructed then the program is built using the build_program function

Choice selection num from 3 - 99

```
int num;
std::cout << "Enter a number between 2 and 99: ";
std::cin >> num;

// Check if input is valid
if (num < 2 || num > 99) {
    // if valid number was not entered
    quit_program("Invalid Number.");
}
```

If num is not from 2 to 99, quit program

Kernel creation, setting arguments, and enqueuing & buffer creation

```
// create a kernel
kernel = cl::Kernel(program, "task3");

// create command queue
queue = cl::CommandQueue(context, device);

// create buffers
arrayBuffer = cl::Buffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR,sizeof(int) * array.size(), array.data());

// set kernel arguments
kernel.setArg(0, arrayBuffer);
kernel.setArg(1, num);

cl::NDRange offset(0);
cl::NDRange globalSize(array.size());   // work-units per kernel based on array size

queue.enqueueNDRangeKernel(kernel, offset, globalSize);

std::cout << "Kernel enqueued." << std::endl;
std::cout << "--------------------" << std::endl;

// enqueue command to read from device to host memory
queue.enqueueReadBuffer(arrayBuffer, CL_TRUE, 0, sizeof(int) * array.size(), array.data());
```

- kernel object is created using the OpenCL program object program and the kernel name "task2".
- The command queue is created using the cl::CommandQueue constructor with the context and device as arguments.
- Creates a buffer object named arrayBuffer in the OpenCL context. The buffer is associated with the array vector and has a size equal to sizeof(int) * array.size(). The CL_MEM_READ_WRITE flag indicates that the buffer will be read from and written to by the kernel, and the CL_MEM_COPY_HOST_PTR flag indicates that the data from the array vector will be copied to the buffer.
- Sets the first kernel argument to arrayBuffer, representing the buffer object that will be passed to the kernel.
- Sets the second kernel argument to num, representing a value that will be passed to the kernel.
- Creates an NDRange object offset with a single dimension of size 0. This specifies the offset in work-items used when enqueuing the kernel.

- with a single dimension of size equal to the size of the array. This specifies the total number of work-items used when enqueuing the kernel.
- Enqueues the kernel for execution on the command queue. The offset and globalSize specify the work-items to be executed by the kernel.
- Enqueues a command to read the data from the arrayBuffer on the device back to the host memory. The read data is stored in the array vector.

Print results

```
// check the results
// Display contents of host-side array
for (int i = 0; i < array.size(); i++) {
    std::cout << std::setw(6) << array[i];

    // Check if we have printed 10 values already
    if ((i + 1) % 10 == 0) {
        std::cout << std::endl;
    }
}

// Add an extra newline after the final line, if needed
if (array.size() % 10 != 0) {
    std::cout << std::endl;
}
```

Error handling and quitting program

```
    // catch any OpenCL function errors
    catch (cl::Error e) {
        // call function to handle errors
        handle_error(e);
    }

#ifdef _WIN32
    // wait for a keypress on Windows OS before exiting
    std::cout << "\npress a key to quit...";
    std::cin.ignore();
#endif

    return 0;
}
```

If any OpenCL function errors occur, the handle_error function is called to handle them, followed by the keypress to exit program

Task 3 Program screenshots

```
Number of OpenCL platforms: 1
--------------------
Available options:
Option 0: Platform - NVIDIA Corporation, Device - NVIDIA GeForce RTX 3080

--------------------
Select a device: 0
Program build: Successful
--------------------
Enter a number between 2 and 99: 1
Invalid Number.
Exiting the program...

press a key to quit...
C:\Users\keiren\source\repos\Assignment\Debug\Assignment.exe (process 27844) exited with code 1.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Wrong input

```
Option 0: Platform - NVIDIA Corporation, Device - NVIDIA GeForce RTX 3080

--------------------
Select a device: 0
--------------------
Contents of program string:
__kernel void task3(__global int* array, const int num) {

   int i = get_global_id(0);
   array[i] = 3 + (num * i);
}


--------------------
Program build: Successful
--------------------
Enter a number between 2 and 99: 2
Kernel enqueued.
--------------------
     3     5     7     9    11    13    15    17    19    21
    23    25    27    29    31    33    35    37    39    41
    43    45    47    49    51    53    55    57    59    61
    63    65    67    69    71    73    75    77    79    81
    83    85    87    89    91    93    95    97    99   101
   103   105   107   109   111   113   115   117   119   121
   123   125   127   129   131   133   135   137   139   141
   143   145   147   149   151   153   155   157   159   161
   163   165   167   169   171   173   175   177   179   181
   183   185   187   189   191   193   195   197   199   201
   203   205   207   209   211   213   215   217   219   221
   223   225   227   229   231   233   235   237   239   241
   243   245   247   249   251   253   255   257   259   261
   263   265   267   269   271   273   275   277   279   281
   283   285   287   289   291   293   295   297   299   301
   303   305   307   309   311   313   315   317   319   321
   323   325   327   329   331   333   335   337   339   341
   343   345   347   349   351   353   355   357   359   361
   363   365   367   369   371   373   375   377   379   381
   383   385   387   389   391   393   395   397   399   401
   403   405   407   409   411   413   415   417   419   421
   423   425   427   429   431   433   435   437   439   441
   443   445   447   449   451   453   455   457   459   461
   463   465   467   469   471   473   475   477   479   481
   483   485   487   489   491   493   495   497   499   501
   503   505   507   509   511   513   515   517   519   521
   523   525   527   529   531   533   535   537   539   541
   543   545   547   549   551   553   555   557   559   561
   563   565   567   569   571   573   575   577   579   581
   583   585   587   589   591   593   595   597   599   601
   603   605   607   609   611   613   615   617   619   621
   623   625   627   629   631   633   635   637   639   641
   643   645   647   649   651   653   655   657   659   661
   663   665   667   669   671   673   675   677   679   681
   683   685   687   689   691   693   695   697   699   701
   703   705   707   709   711   713   715   717   719   721
   723   725   727   729   731   733   735   737   739   741
   743   745   747   749   751   753   755   757   759   761
   763   765   767   769   771   773   775   777   779   781
   783   785   787   789   791   793   795   797   799   801
   803   805   807   809   811   813   815   817   819   821
   823   825   827   829   831   833   835   837   839   841
   843   845   847   849   851   853   855   857   859   861
   863   865   867   869   871   873   875   877   879   881
   883   885   887   889   891   893   895   897   899   901
   903   905   907   909   911   913   915   917   919   921
   923   925   927   929   931   933   935   937   939   941
   943   945   947   949   951   953   955   957   959   961
   963   965   967   969   971   973   975   977   979   981
   983   985   987   989   991   993   995   997   999  1001
  1003  1005  1007  1009  1011  1013  1015  1017  1019  1021
```

Correct input