Basic Queue and Command Group Usage

Objective: Learn how to create and use queues and command groups to manage tasks in SYCL.

Task:

1. Write a kernel using parallel_for to initialize a buffer of size 1024 with values from 0 to 1023.

```
#include <sycl/sycl.hpp>
#include <vector>
#include <iostream>

using namespace sycl;

int main() {
    const int N = 1024;
    std::vector<int> data(N);

    buffer<int, 1> buf(data.data(), range<1>(N));
    queue q;

    q.submit([&](handler& cgh) {
        auto aA = buf.get_access<access::mode::write>(cgh);

       //TODO: Kernel goes here => aA[i]=id

    }).wait();

    host_accessor h_acc(buf);
    for (int i = 0; i < N; ++i) {
        std::cout << h_acc[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Using Function Objects for Kernels

Objective: Understand how to define and use function objects (functors) for more complex kernel logic.

Task:

1. Define a function object to perform the square each element on a buffer.
2. Use a command group to execute the kernel defined by the function object.

```
// Fill in the code parts (____)
#include <sycl/sycl.hpp>
#include <vector>
#include <iostream>
#include <random>

using namespace sycl;

class ____ {
public:
    void ____()(id<1> i, accessor<int, 1, access::mode::read_write> a)
    ↪  const {
        a[i] = ____;
    }
};

int main() {
    const int N = 512;
    std::vector<int> data(N);
    std::mt19937 gen;
    std::uniform_int_distribution<int> dis(1, 100);
    for(int i = 0; i < N; ++i) data[i] = dis(gen);

    buffer<int, 1> buf(data.data(), range<1>(N));
    queue q;

    q.submit([&](handler& cgh) {
        auto aA = buf.get_access<access::mode::read_write>(cgh);
        cgh.parallel_for<____>(range<1>(N), [=](id<1> i) {
            ____()(____,____); // Que argumentos passamos ?? (acedemos ao
↪  vector atraves do aA);
        });
    }).wait();

    host_accessor h_acc(buf);
    for (int i = 0; i < N; ++i) {
        std::cout << h_acc[i] << " ";
    }
    std::cout << std::endl;
```

```
    return 0;
}
```

Combining Multiple Commands in a Queue

Objective: Explore how to combine multiple commands and manage synchronization and dependencies.

Task:

1. Write kernels that:
    - Increments each element by 1 using a parallel_for.
    - Copies the buffer to a new buffer.
    - Fills the original buffer with 42.
2. Add the needed dependencies between actions
3. Print the final values of both buffers on the host.
    1. Is it needed to add the dependencies manually ?

```
#include <sycl/sycl.hpp>
#include <vector>
#include <iostream>

using namespace sycl;

int main() {
    const int N = 256;
    std::vector<int> data(N,0.0);
    std::vector<int> data_old(N,0.0);
    for(int i = 0; i < N; ++i) data[i] = i;

    buffer<int, 1> buf(data.data(), range<1>(N));
    buffer<int, 1> buf_copy(data_old.data(),range<1>(N));
    queue q;

    q.submit([&](handler& cgh) {
        auto aA = buf.get_access(cgh);
        // TODO: kernel increment one here we access using aA
    });

    q.submit([&](handler& cgh) {
        auto aB = buf.get_access(cgh);
```

```
        auto aC = buf_copy.get_access(cgh);
         // TODO: kernel that copies from buffer aB to buffer aC
    });

    q.submit([&](handler& cgh) {
        auto aD = buf.get_access(cgh);
        // TODO: kernel that fills buffer aD to 42
    });


    // how can we force the last action to be completed before printing ?

    host_accessor h_acc(buf);
    host_accessor h_acc_copy(buf_copy);
    std::cout << "Original buffer after operations: ";
    for (int i = 0; i < N; ++i) {
        std::cout << h_acc[i] << " ";
    }
    std::cout << std::endl;

    std::cout << "Copied buffer: ";
    for (int i = 0; i < N; ++i) {
        std::cout << h_acc_copy[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```