

```
1 #include <math.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include <time.h>
6
7 #ifndef M_PI
8 #define M_PI 3.14159265358979323846
9 #endif
10
11 #define PROGRAM_FILE "add_numbers.cl"
12 #define KERNEL_FUNC "add_numbers"
13 #define ARRAY_SIZE 64
14
15 // Device and context variables
16 cl_device_id device;
17 cl_platform_id platform;
18 cl_device_id dev;
19
20 // Context and program variables
21 cl_context ctx;
22 cl_program prog;
23 cl_kernel kernel;
24 cl_command_queue queue;
25
26 // Global and local memory sizes
27 size_t global_size;
28 size_t local_size;
29
30 // Command queue and kernel arguments
31 cl_command_queue queue;
32 cl_kernel kernel;
33
34 // Kernel arguments
35 int *input;
36 int *output;
37
38 // Main function
39 int main() {
40     // Create context
41     ctx = clCreateContext(0, 1, &device, NULL, NULL, NULL);
42     if (ctx == NULL) {
43         printf("Error: Could not create a context\n");
44         return -1;
45     }
46
47     // Create program
48     prog = clCreateProgramWithSource(ctx, 1, (const char **)&PROGRAM_FILE, NULL, NULL);
49     if (prog == NULL) {
50         printf("Error: Could not create a program\n");
51         return -1;
52     }
53
54     // Build program
55     clBuildProgram(prog, 0, NULL, NULL, NULL, NULL);
56
57     // Create kernel
58     kernel = clCreateKernel(prog, KERNEL_FUNC, &err);
59     if (err != CL_SUCCESS) {
60         printf("Error: Could not create a kernel\n");
61         return -1;
62     }
63
64     // Create command queue
65     queue = clCreateCommandQueue(ctx, device, 0, &err);
66     if (err != CL_SUCCESS) {
67         printf("Error: Could not create a command queue\n");
68         return -1;
69     }
70
71     // Allocate memory
72     input = (int *)malloc(ARRAY_SIZE * sizeof(int));
73     output = (int *)malloc(ARRAY_SIZE * sizeof(int));
74
75     // Fill input array
76     for (int i = 0; i < ARRAY_SIZE; i++) {
77         input[i] = 1.0f * i;
78     }
79
80     // Enqueue kernel
81     clEnqueueKernel(queue, 1, NULL, 0, NULL, NULL, 0, NULL, NULL, &err);
82     if (err != CL_SUCCESS) {
83         printf("Error: Could not enqueue the kernel\n");
84         return -1;
85     }
86
87     // Wait for kernel to finish
88     clWaitForEvents(1, &queue);
89
90     // Read output
91     clReadBuffer(queue, output, CL_TRUE, 0, ARRAY_SIZE * sizeof(int), &err);
92     if (err != CL_SUCCESS) {
93         printf("Error: Could not read the buffer\n");
94         return -1;
95     }
96
97     // Free memory
98     free(input);
99     free(output);
100
101     return 0;
102 }
```

IMPORTS
VARIABLES GLOBALES

SÉLECTION DU DEVICE

CONSTRUCTION DU PROGRAMME

CRÉATION DU CONTEXTE
CRÉATION DE LA QUEUE

ALLOCATION MÉMOIRE
TRANSFERTS DONNÉES VERS GPU

PASSAGE D'ARGUMENTS AU KERNEL

LANCEMENT DU KERNEL

RÉCUPÉRATION DES RÉSULTATS
DÉSALLOCATION DE RESSOURCES

```
1 extern crate ocl;
2 use ocl::ProQue;
3
4 const ARRAY_SIZE: usize = 64;
5 static SRC: &str = r#"/* ... */"#;
6
7 fn main() -> ocl::Result<> {
8     let pro_que = ProQue::builder().src(SRC).dims(ARRAY_SIZE).build()?;
9     let input = pro_que.create_buffer::<f32>()?;
10    let d_output = pro_que.create_buffer::<f32>()?;
11
12    let kernel = pro_que
13        .kernel_builder("fma")
14        .arg(&input)
15        .arg(10.0f32)
16        .arg(&d_output)
17        .build()?;
18
19    unsafe {
20        kernel.enq()?;
21    }
22
23    let mut h_output = vec![0.0f32; d_output.len()];
24    d_output.read(&mut h_output).enq()?;
25
26    Ok(())
27 }
```