```
// header files
// standard headers
#include <stdio.h>

// cuda headers
#include <cuda.h>

// global variables
const int iNumberOfArrayElements = 5;

float* hostInput1 = NULL;
float* hostInput2 = NULL;
float* hostOutput = NULL;

float* deviceInput1 = NULL;
float* deviceInput2 = NULL;
float* deviceOutput = NULL;

// CUDA kernel
__global__ void vecAddGPU(float* in1, float* in2, float* out, int len)
{
    // code
    int i = blockIdx.x * blockDim.x + threadIdx.x;

    if (i < len)
    {
        out[i] = in1[i] + in2[i];
    }
}

// entry-point function
int main(void)
{
    // function declarations
    void cleanup(void);

    // variable declarations
    int size = iNumberOfArrayElements * sizeof(float);
    cudaError_t result = cudaSuccess;

    // code
    // host memory allocation
    hostInput1 = (float*)malloc(size);
    if (hostInput1 == NULL)
    {
        printf("Host Memory allocation is failed for hostInput1 array.\n");
        cleanup();
        exit(EXIT_FAILURE);
    }

    hostInput2 = (float*)malloc(size);
    if (hostInput2 == NULL)
    {
        printf("Host Memory allocation is failed for hostInput2 array.\n");
        cleanup();
        exit(EXIT_FAILURE);
```

```
57      }
58
59      hostOutput = (float*)malloc(size);
60      if (hostOutput == NULL)
61      {
62          printf("Host Memory allocation is failed for hostOutput array.\n");
63          cleanup();
64          exit(EXIT_FAILURE);
65      }
66
67      // filling values into host arrays
68      hostInput1[0] = 101.0;
69      hostInput1[1] = 102.0;
70      hostInput1[2] = 103.0;
71      hostInput1[3] = 104.0;
72      hostInput1[4] = 105.0;
73
74      hostInput2[0] = 201.0;
75      hostInput2[1] = 202.0;
76      hostInput2[2] = 203.0;
77      hostInput2[3] = 204.0;
78      hostInput2[4] = 205.0;
79
80      // device memory allocation
81      result = cudaMalloc((void**)&deviceInput1, size);
82      if (result != cudaSuccess)
83      {
84          printf("Device Memory allocation is failed for deviceInput1 array.
              \n");
85          cleanup();
86          exit(EXIT_FAILURE);
87      }
88
89      result = cudaMalloc((void**)&deviceInput2, size);
90      if (result != cudaSuccess)
91      {
92          printf("Device Memory allocation is failed for deviceInput2 array.
              \n");
93          cleanup();
94          exit(EXIT_FAILURE);
95      }
96
97      result = cudaMalloc((void**)&deviceOutput, size);
98      if (result != cudaSuccess)
99      {
100         printf("Device Memory allocation is failed for deviceOutput array.
              \n");
101         cleanup();
102         exit(EXIT_FAILURE);
103     }
104
105     // copy data from host arrays into device arrays
106     result = cudaMemcpy(deviceInput1, hostInput1, size,
          cudaMemcpyHostToDevice);
107     if (result != cudaSuccess)
108     {
```

```
109         printf("Host to Device Data Copy is failed for deviceInput1 array.
              \n");
110         cleanup();
111         exit(EXIT_FAILURE);
112     }
113
114     result = cudaMemcpy(deviceInput2, hostInput2, size,
          cudaMemcpyHostToDevice);
115     if (result != cudaSuccess)
116     {
117         printf("Host to Device Data Copy is failed for deviceInput2 array.
              \n");
118         cleanup();
119         exit(EXIT_FAILURE);
120     }
121
122     dim3 dimGrid = dim3(iNumberOfArrayElements, 1, 1);
123     dim3 dimBlock = dim3(1, 1, 1);
124
125     // CUDA kernel for Vector Addition
126     vecAddGPU <<<dimGrid, dimBlock >>> (deviceInput1, deviceInput2,
          deviceOutput, iNumberOfArrayElements);
127
128     // copy data from device array into host array
129     result = cudaMemcpy(hostOutput, deviceOutput, size,
          cudaMemcpyDeviceToHost);
130     if (result != cudaSuccess)
131     {
132         printf("Device to Host Data Copy is failed for hostOutput array.\n");
133         cleanup();
134         exit(EXIT_FAILURE);
135     }
136
137     // vector addition on host
138     for (int i = 0; i < iNumberOfArrayElements; i++)
139     {
140         printf("%f + %f = %f\n", hostInput1[i], hostInput2[i], hostOutput[i]);
141     }
142
143     // cleanup
144     cleanup();
145
146     return(0);
147 }
148
149 void cleanup(void)
150 {
151     // code
152     if (deviceOutput)
153     {
154         cudaFree(deviceOutput);
155         deviceOutput = NULL;
156     }
157
158     if (deviceInput2)
159     {
```

```
160         cudaFree(deviceInput2);
161         deviceInput2 = NULL;
162     }
163
164     if (deviceInput1)
165     {
166         cudaFree(deviceInput1);
167         deviceInput1 = NULL;
168     }
169
170     if (hostOutput)
171     {
172         free(hostOutput);
173         hostOutput = NULL;
174     }
175
176     if (hostInput2)
177     {
178         free(hostInput2);
179         hostInput2 = NULL;
180     }
181
182     if (hostInput1)
183     {
184         free(hostInput1);
185         hostInput1 = NULL;
186     }
187 }
188
```