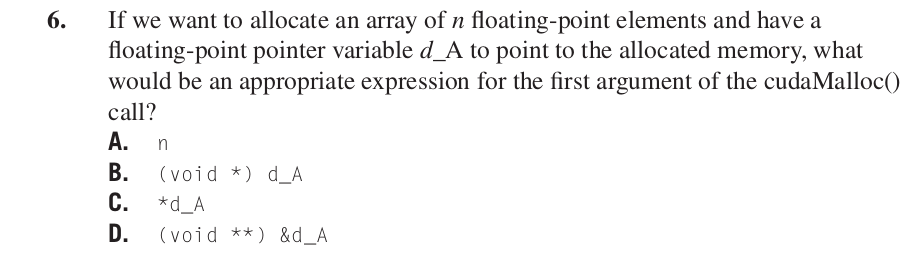


Answer: **C**

My interpretation of the question and the process of arriving at the solution is as follows. The elements are present in an array with starting index 0 and ending index n. As each thread processes two adjacent elements from the array, the index of the first element to be processed by each thread compared to the next thread should have a difference of 2. For instance, the first element of thread 0 should be 0, the first element of thread 1 should be 2, the first element of thread 2 must be 4, etc. The expression given in answer choice C satifies this contraint. A few results are given in the table below.

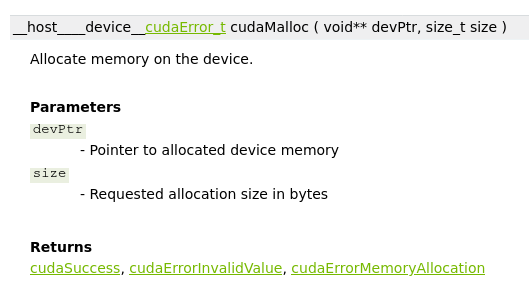
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **blockDim** | 256 | | | | | | | | |
| **blockIdx** | 0 | | | | | 1 | | | |
| **threadIdx** | 0 | 1 | 2 | ... | 255 | 0 | 1 | 2 | ... |
| **i** | 0 | 2 | 4 | ... | 510 | 512 | 514 | 516 | ... |



Answer: **D**

The first parameter to the cudaMalloc function is the address of a pointer variable that will be set to point to the allocated object. The address of the pointer variableshould be cast to (void \*\*) because the function expects a generic pointer; the memory allocation function is a generic function that is not restricted to any particular type of objects.

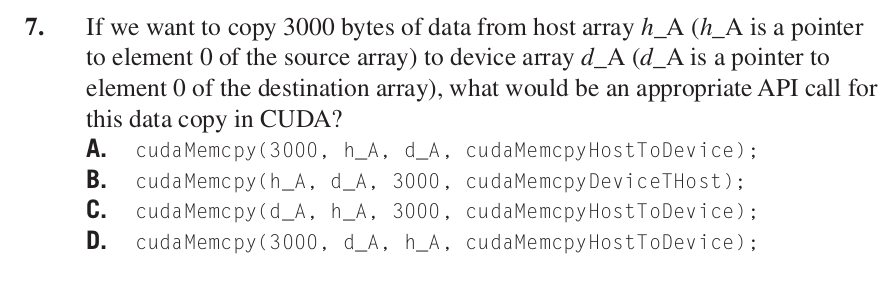
The CUDA toolkit documentation from Nvidia states the following.



Reference:

Programming Massively Parallel Processors: A Hands-on Approach by Wen-mei Hwu and David Kirk

NVIDIA CUDA Toolkit Documentation.



Answer: **C**

The cudaMemcpy function takes in four arguments namely the destination memory address, the source memory address, the size in bytes to copy, and the type of transfer. For the scenario given in the question the values of the arguments are as follows. Hence the answer choice C.

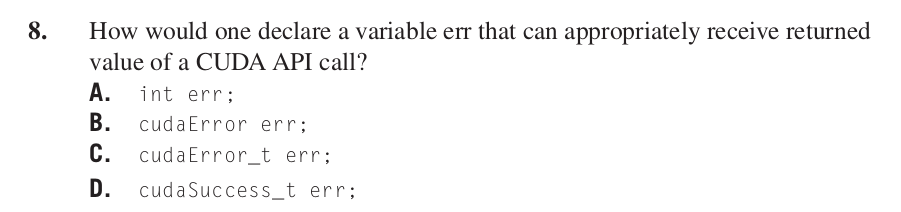
1. Destination memory address – d\_A

2. Source memory address – h\_A

3. Size in bytes to copy – 3000

4. Type of transfer – cudaMemcpyHostToDevice

Reference: Nvidia CUDA Toolkit Documentation



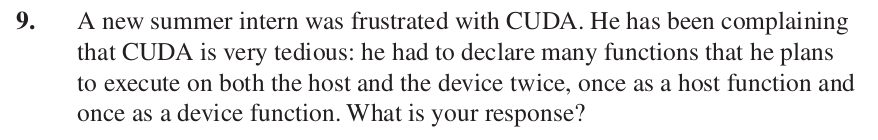
Answer: **C**

CUDA API calls have a return type of cudaError\_t which is a typedef.

typedef enumcudaError cudaError\_t

enum cudaError

Reference: Nvidia CUDA Toolkit Documentation



**My response:**

The intern could use both “\_\_host\_\_” and “\_\_device\_\_” keywords in his function declarations. This combination tells the compiler to create two versions of object file, one to be executed on device and the other to be executed on host, for the same function.

Reference:

Programming Massively Parallel Processors: A Hands-on Approach by Wen-mei Hwu and David Kirk