Which one of these sentences is true and which is false?

- All the threads of a thread block execute in lock-step: False
- `syncthreads()` is a barrier for all the threads in a thread block: True
- Variable declared as `_global_` in a CUDA kernel are allocated in the shared memory: False
- Shared memory in CUDA is shared by all the threads in a kernel: False
- Global memory in CUDA is shared by all the threads: True
- `cudaMemcpy()` can be called from a Kernel to copy data between host and global memory: False
- `cudaMemcpy()` is used to copy data between host and global memory: True

Assuming that you wrote a cuda kernel that declares a shared memory array consisting of 4K bytes and that the compiler determined that each thread in that kernel needs 16 integer registers. Assume also that your GPU has 4 SMs, each with a register file of 2048 integer registers and a shared memory of 16K bytes. If your application will execute kernel `<<<nblocks, blksize>>>'s, answer the following questions:

- What is the maximum number of threads that can execute simultaneously on the GPU?
  Each SM has 2048 registers and each thread needs 16 registers
  → each SM can support 128 threads → 4 SMs can support 512 threads

- What is the maximum number of thread blocks that can execute simultaneously on an SM?
  Each SM has 16K bytes of shared memory and each thread block needs 4K bytes
  → each SM can support at most 4 thread blocks simultaneously.

- To execute the maximum number of threads simultaneously what is the value of `nblocks` and `blksize` that you would use when launching the kernel
  `Kernel<<<16,32>>` or `<<<8, 64>>` or `<<4, 128>>`
Show the output of the content of array A after the execution of the following program:

```c
_global_ F(int *A)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    A[idx] = idx;
    A[blockIdx.x] = blockIdx.x;
}

void main()
{
    Allocate a 16 element int array A in the GPU global memory and initialize its elements to 0;
    F<<<2,4>>>(A);
}
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Rewrite the following cuda kernel without using shared memory. The kernel adds N integers stored in the global array “input[]” into a global variable, “total”, and is called as

```
plus_reduce<<<NB, N/NB>>>(input, N, total)
```

Where N is multiple of NB.

```c
_global_void plus_reduce(int *input, int *N, int *total)
{
    int tid = threadIdx.x;
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    _shared_ int x[blocksize];
    x[tid] = input[i];
    __syncthreads();

    for(int s=blockDim.x/2; s>0; s=s/2)
    {
        if(tid < s) x[tid] += x[tid + s];
        __syncthreads();
    }
    if (tid == 0) atomicAdd(total, x[tid]);
}
```
What is wrong with the following code?

```c
__global__ F(int *A)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    A[idx] = idx;
    if (idx < blockDim * gridDim / 2)_syncthreads();
    A[blockIdx.x] = blockIdx.x;
}
```