Exercise 11

Task 1 (`distribute_collect.c`)

Write an **MPI program** `distribute_collect.c`:

1. Process 0 initializes an integer variable `base_value` with the value `100`.
2. Distribute `base_value` from process 0 to all other processes.
3. Each process computes a **local result** by adding its rank multiplied by 5 to the received value:

```
local_result = base_value + (rank * 5)
```

4. Each process prints its local result:

```
Rank X: local_result = Y
```

5. Collect all local results at process 0.
6. Process 0 prints all collected results:

```
Collected results: [100, 105, 110, 115, ...]
```

7. Process 0 computes and prints the **sum** of all collected values.

Task 2 (`chunk_exchange.c`)

Write an **MPI program** `chunk_exchange.c`:
Assume 4 processes and `N = 4`.

1. Process 0 initializes an array of size `N * size` (where `size` is the number of processes and `N = 4`). Initialize this array with consecutive integers starting from 1:

```
global_array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

2. Distribute the array so that each process receives exactly `N` elements.
3. Each process prints the chunk it received:

```
Rank 0 received: [1, 2, 3, 4]
Rank 1 received: [5, 6, 7, 8]
Rank 2 received: [9, 10, 11, 12]
Rank 3 received: [13, 14, 15, 16]
```

4. Each process multiplies every element in its chunk by `(rank + 1)`:

```
local_chunk[i] = local_chunk[i] * (rank + 1)
```

After multiplication:

```
Rank 0: [1, 2, 3, 4]       (multiplied by 1)
Rank 1: [10, 12, 14, 16]   (multiplied by 2)
Rank 2: [27, 30, 33, 36]   (multiplied by 3)
Rank 3: [52, 56, 60, 64]   (multiplied by 4)
```

5. Exchange data such that each process sends one element to every other process and receives one element from each.
6. Each process prints the data it received after the exchange:

```
Rank 0 after exchange: [1, 10, 27, 52]
Rank 1 after exchange: [2, 12, 30, 56]
Rank 2 after exchange: [3, 14, 33, 60]
Rank 3 after exchange: [4, 16, 36, 64]
```

7. Each process computes and prints the sum of its received elements:

```
Rank 0 sum: 90
Rank 1 sum: 100
Rank 2 sum: 110
Rank 3 sum: 120
```

Task 3 (`global_ops.c`)

Write an **MPI program** `global_ops.c`:

1. Each process initializes a local array of `N = 5` integers. The values should depend on the rank:

```
local_array[i] = (rank + 1) * (i + 1)    for i = 0, 1, ..., N-1
```

   For example, rank 0 has `[1, 2, 3, 4, 5]`, rank 1 has `[2, 4, 6, 8, 10]`, etc.

2. Each process prints its local array:

```
Rank 1 local array: [2, 4, 6, 8, 10]
Rank 0 local array: [1, 2, 3, 4, 5]
Rank 3 local array: [4, 8, 12, 16, 20]
Rank 2 local array: [3, 6, 9, 12, 15]
```

3. Compute the element-wise sum of all local arrays. The result should be stored at process 0.
4. Process 0 prints the summed array:

```
Sum (at root): [10, 20, 30, 40, 50]
```

5. Find the element-wise maximum across all processes. Every process should receive the result.

6. All processes print the maximum array:

```
Rank 0 - Max (global): [4, 8, 12, 16, 20]
Rank 2 - Max (global): [4, 8, 12, 16, 20]
Rank 1 - Max (global): [4, 8, 12, 16, 20]
Rank 3 - Max (global): [4, 8, 12, 16, 20]
```

Task 4 (`uneven_distribution.c`)

Write an **MPI program** `uneven_distribution.c`:

1. Process 0 initializes a global array of size `N = 22` with values from 1 to 22:

```
global_array = [1, 2, 3, ..., 22]
```

2. Distribute the array **unevenly** across processes. Each process should receive $\lfloor N/\texttt{size} \rfloor$ elements, with the **first** $N \mod \texttt{size}$ processes receiving one extra element.

3. Each process prints the elements it received. For example, with 4 processes and $N = 22$:
   - Process 0 receives 6 elements: `[1, 2, 3, 4, 5, 6]`
   - Process 1 receives 6 elements: `[7, 8, 9, 10, 11, 12]`
   - Process 2 receives 5 elements: `[13, 14, 15, 16, 17]`
   - Process 3 receives 5 elements: `[18, 19, 20, 21, 22]`

4. Each process squares all elements in its local chunk:

```
local_chunk[i] = local_chunk[i] * local_chunk[i]
```

5. Collect all squared values back at process 0.

6. Process 0 prints the collected result:

```
Collected squared values: [1, 4, 9, 16, ..., 484]
```

7. Process 0 verifies correctness by computing and printing the sum: $\sum_{i=1}^{22} i^2 = 3795$.