EI1024/MT1024 "Programación Concurrente y Paralela" 2025–26	Entregable para Casa ec_t10
Nombre y apellidos del alumno: Víctor-Nesta Reig Buendía	
Tiempo empleado en formato *H:MM* (obligatorio): 1:30	
Nota: Para ser evaluado en esta actividad (y en el test correspondiente) es obligatorio completar todos los apartados de todos los ejercicios y rellenar el campo del tiempo empleado (justo arriba). La no realización de un ejercicio o de un apartado de un ejercicio conllevará la no evaluación de esta actividad (y del test correspondiente).	

Tema 10. Comunicaciones Punto a Punto en MPI

1 (*Tiempo estimado: 10'*) Responde a las preguntas siguientes.

1.1) ¿Se puede emplear la constante `MPI_ANY_SOURCE` como destinatario en una operación de envío de MPI?

No, el destino tiene que ser una direccion concreta

1.2) ¿Se puede emplear la constante `MPI_ANY_SOURCE` como origen en una operación de recepción de MPI?

Sí

1.3) ¿Se puede emplear la constante `MPI_ANY_TAG` como etiqueta en una operación de envío de MPI?

No, la tag de envio debe ser concreta

1.4) ¿Se puede emplear la constante `MPI_ANY_TAG` como etiqueta en una operación de recepción de MPI?

Sí

2 (*Tiempo estimado: 10'*) En el tema anterior se ha trabajado con las dos funciones siguientes:

```
1    SendInt( int * dato, int procDestino );
2    ReceiveInt( int * dato, int procOrigen );
```

Completa la implementación de dichas funciones empleando únicamente funciones de MPI dentro de ellas.

```
1  void SendInt( int * dato, int procDestino ) {
2    ...
3  }
4
5  void ReceiveInt( int * dato, int procOrigen ) {
```

```
6       ...
7     }
```

```
void SendInt(int dato, int procDestino) {
    MPI_Send(&dato, 1, MPI_INT, procDestino, 33, MPI_COMM_WORLD);
}
void ReceiveInt(int dato, int procOrigen) {
    MPI_Status s;
    MPI_Recv(&dato, 1, MPI_INT, procOrigen, 33, MPI_COMM_WORLD, &s);
}
```

no hace falta & porque int * dato, en status sí hace falta.

3 (*Tiempo estimado: 20'*) A partir de este ejercicio ya **NO** se pueden emplear las funciones `SendInt` y `ReceiveInt`, sino que habrá que emplear las funciones estándares de MPI.

Se supone que tras realizar ciertos cálculos costosos, todos los procesos almacenan un valor (posiblemente distinto) en una variable entera denominada `dato`. Se desea que al término del programa el proceso 0 contenga **el máximo de todos esos valores y el identificador del proceso que lo contiene**. Finalmente, el proceso 0 deberá imprimir ambos valores en pantalla.

Hay que tener en cuenta que el dato contenido en la variable `dato` del proceso 0 también debe intervenir en los cálculos.

3.1) Existen muchas formas de llevar a cabo tal tarea. A continuación se describe el método que se debe seguir en este apartado.

En primer lugar, todos los procesos excepto el proceso 0 deberán enviar su respectivo dato al proceso 0. Cada vez que el proceso 0 recibe un dato, deberá actualizar el máximo hasta el momento en la variable `maximo` y el identificador del proceso que contiene dicho máximo en la variable `p_max`. El proceso 0 deberá recibir los datos de los restantes procesos **uno a uno por orden de proceso**.

```
if (miId == 0) {
    int recibido;
    int maximo = dato;
    int p_max = 0;
    MPI_Status s;
    for (int i = 1; i < numProcs; i++) {
        MPI_Recv(&recibido, 1, MPI_INT, i, 33, MPI_COMM_WORLD, &s);
        if (recibido > maximo) {
            maximo = recibido;
            p_max = i;
        }
    }
    printf("El máximo es: %d. Proceso: %d\n", maximo, p_max);
} else {
    MPI_Send(&dato, 1, MPI_INT, 0, 33, MPI_COMM_WORLD);
}
```

3.2) Reescribe el código anterior de tal forma que ahora el proceso 0 reciba los datos de los restantes procesos **en cualquier orden**.

```
if (miId == 0) {
    int recibido;
    int maximo = dato;
    int p_max = 0;
    MPI_Status s;
    for (int i = 1; i < numProcs; i++) {
        MPI_Recv(&recibido, 1, MPI_INT, MPI_ANY_SOURCE, 33, MPI_COMM_WORLD, &s);
        if (recibido > maximo) {
            maximo = recibido;
            p_max = s.MPI_SOURCE;
        }
    }
    printf("El máximo es: %d. Proceso: %d\n", maximo, p_max);
} else {
    MPI_Send(&dato, 1, MPI_INT, 0, 33, MPI_COMM_WORLD);
}
```

4 (*Tiempo estimado: 20'*) Se supone que tras realizar ciertos cálculos costosos, todos los

procesos almacenan un valor (posiblemente distinto) en una variable real con precisión doble denominada `dato`. Se desea que al término del programa el proceso 0 contenga **la suma de dichos valores de los procesos PARES** en una variable denominada `suma`. Finalmente, el proceso 0 deberá imprimir dicho valor en pantalla. Los valores guardados en los procesos impares no deberán ser tenidos en cuenta.

Hay que tener en cuenta que el dato contenido en la variable `dato` del proceso 0 también debe intervenir en los cálculos.

El programa debe funcionar correctamente para cualquier número de procesos mayor o igual a 2.

Existen muchas formas de llevar a cabo tal tarea. A continuación se describe el método que se debe seguir.

Todos los procesos pares menos el proceso 0 deberán enviar su dato al proceso 0. El proceso 0 deberá recibir y acumular los datos de todos los procesos pares excepto él mismo. Una vez realizadas todas las recepciones y la acumulación de su propio dato, el proceso 0 imprimirá en pantalla la suma total.

```
double dato = ...;
if (miId == 0) {
    double recibido;
    double suma = dato;
    MPI_Status s;
    for (int i = 2; i < numProcs; i += 2) {
        MPI_Recv(&recibido, 1, MPI_DOUBLE, MPI_ANY_SOURCE, 33, MPI_COMM_WORLD, &s);
        suma += recibido;
    }
    printf("La suma de los procesos pares es: %f\n", suma);
} else if (miId % 2 == 0) {
    MPI_Send(&dato, 1, MPI_DOUBLE, 0, 33, MPI_COMM_WORLD);
}
```

5 (*Tiempo estimado: 15'*) Se supone que tras realizar ciertos cálculos costosos, todos los procesos almacenan un valor (posiblemente distinto) en una variable real con precisión doble denominada `dato`. Se desea que al término del programa, cada proceso contenga **la suma de dichos valores de todos los procesos con índice menor o igual al suyo** en una variable `suma`. Es decir, el proceso 0 debe contener en `suma` su propio valor. El proceso 1 debe contener la suma del valor del proceso 0 y de su propio valor. El proceso 2 debe contener la suma de su propio valor y los valores de los procesos 0 y 1. El proceso 3 debe contener la suma de su propio valor y los valores de los procesos 0, 1 y 2. Y así sucesivamente.

A continuación se muestra un ejemplo de los datos iniciales y de las sumas finales en 5 procesos.

	P_0	P_1	P_2	P_3	P_4
Dato inicial:	5.0	4.0	3.0	2.0	1.0
Suma final:	5.0	9.0	12.0	14.0	15.0

Al final del programa, todos los procesos deben imprimir su dato y su suma.

El programa debe funcionar correctamente para cualquier número de procesos mayor o igual a 2.

Existen muchas formas de llevar a cabo tal tarea. A continuación se describe el método que se debe seguir en este apartado. El proceso 0 debe enviar su dato al proceso 1. El proceso 1 debe enviar su suma al proceso 2. El proceso 2 debe enviar su suma al proceso 3. Y así sucesivamente.

```
MPI_Status s;
if (miId == 0) {
    MPI_Send(&dato, 1, MPI_DOUBLE, 1, 33, MPI_COMM_WORLD);
    suma = dato;
} else if (miId == numProcs - 1) {
    suma = dato;
    MPI_Recv(&otro, 1, MPI_DOUBLE, miId-1, 33, MPI_COMM_WORLD, &s);
    suma += otro;
} else {
    suma = dato;
    MPI_Recv(&otro, 1, MPI_DOUBLE, miId-1, 33, MPI_COMM_WORLD, &s);
    suma += otro;
    MPI_Send(&suma, 1, MPI_DOUBLE, miId+1, 33, MPI_COMM_WORLD);
}
printf("Suma total: %d", suma);
```

6 (*Tiempo estimado: 20'*) Se supone que tras realizar ciertos cálculos costosos, todos los procesos almacenan un valor (posiblemente distinto) en una variable real con precisión doble denominada `dato`. Se desea que al término del programa **los procesos intercambien sus datos** de la siguiente forma: El primer proceso (identificador 0) debe intercambiar su valor con el último proceso (identificador `numProcs - 1`), el segundo proceso (identificador 1) debe intercambiar su valor con el penúltimo proceso (identificador `numProcs - 2`), y así sucesivamente.

A continuación se muestra un posible ejemplo de los datos iniciales y de los datos finales.

	P_0	P_1	P_2	P_3	P_4
Dato inicial:	5.0	4.0	3.0	2.0	1.0
Dato final:	1.0	2.0	3.0	4.0	5.0

Hay que tener en cuenta que en el caso de que el número de procesos sea impar, el proceso central no deberá realizar nada.

El programa debe funcionar correctamente para cualquier número de procesos mayor o igual a 2.

6.1) Existen muchas formas de llevar a cabo tal tarea. A continuación se describe el método que se debe seguir en este apartado.

Se deben emplear **operaciones de envío y recepción habituales** (`MPI_Send` y `MPI_Recv`).

Para evitar interbloqueos, se recomienda que los procesos realicen tareas distintas según el valor de su identificador: la primera mitad deben realizar un envío y después una recepción; la segunda mitad deben realizar una recepción y después un envío.

```
double dato_recibido;
MPI_Status s;
int partner = numProcs - 1 - miId;
if (miId < partner) {
   MPI_Send(&dato, 1, MPI_DOUBLE, partner, 33, MPI_COMM_WORLD);

   MPI_Recv(&dato_recibido, 1, MPI_DOUBLE, partner, 33, MPI_COMM_WORLD, &s);
   dato = dato_recibido;

} else if (miId > partner) {
   MPI_Recv(&dato_recibido, 1, MPI_DOUBLE, partner, 33, MPI_COMM_WORLD, &s);

   MPI_Send(&dato, 1, MPI_DOUBLE, partner, 33, MPI_COMM_WORLD);

   dato = dato_recibido;

}
```

6.2) Existen muchas formas de llevar a cabo tal tarea. A continuación se describe el método que se debe seguir en este apartado.

En este aparado se deben emplear **comunicaciones punto a punto que realicen un envío y una recepción en una única operación**.

```
MPI_Status s;
int partner = numProcs - 1 - miId;
if (miId != partner) {

    MPI_Sendrecv_replace(&dato, 1, MPI_DOUBLE,
                         partner, 33,
                         partner, 33,
                         MPI_COMM_WORLD, &s);

} else {
    // miId == partner: Proceso central. No realiza ninguna acción.
}
```

Nota: Esta entrega forma parte de la evaluación de la asignatura. Debe ser guardado por el estudiante junto con el resto de entregas en una carpeta. El profesor podrá pedir al estudiante que le entregue dicha carpeta en cualquier momento.