MPI_Send()

int MPI_Send(  
    void *buffer,  // Address of the data to send
    int count,    // Number of data elements to send
    MPI_Datatype type,  // Type of data elements to send
    int dest,      // ID of destination process
    int tag,       // Tag to distinguish this message
    MPI_Comm *comm  // An MPI communicator
);

Arguments:
- The address of the data to send.
- The number of data elements to send.
- The type of data elements to send.
- The ID of the process that should receive this message.
- A message tag that distinguishes this message from others that may be sent to the same process.
- The MPI communicator to use.

Notes:
This routine sends data to another process. This routine has blocking semantics, which means that the routine does not return until the message has been sent. MPI_Isend() is a non-blocking version of the send operation; it takes a seventh parameter of type MPI_Request that is used to differentiate this send from other invocations of MPI_Isend() when waiting for completion.

Return value:
An MPI error code.
MPI_Recv()

```c
int MPI_Recv(
  void *buffer, // Address at which to receive data
  int count, // Number of elements to receive
  MPI_Datatype type, // Type of each element
  int source, // ID of sending process
  int tag, // Identifier to distinguish message
  MPI_Comm comm, // MPI communicator
  MPI_Status *status // Status of this receive operation
);
```

Arguments:
- The first six arguments correspond to MPI_Send().
- To receive a message from any other process, use MPI_ANY_SOURCE as the source.
- To match on any tag, use MPI_ANY_TAG as the fifth parameter.

Notes:
This routine receives data from another process. This routine has blocking semantics—it does not return until the message is received. MPI_Recv() is a non-blocking version of the receive operation; it takes a seventh parameter of type MPI_Request that is used to differentiate this receive from other invocations of MPI_Recv() when waiting for completion.

Return value:
An MPI error code.
Count 3’s

Using point-to-point communication
MPI_Scatter()

```c
int MPI_Scatter(
    void   *sendbuffer, // Address of the data to send
    int    sendcount,  // Number of data elements to send
    MPI_Datatype sendtype, // Type of data elements to send
    void   *destbuffer, // Address of buffer to receive data
    int    destcount,  // Number of data elements to receive
    MPI_Datatype desttype, // Type of data elements to receive
    int    root,       // Rank of the root process
    MPI_Comm *comm     // An MPI communicator
);```

Arguments:

- The first three arguments specify the address, size, and type of the data elements to send to each process. These arguments only have meaning for the root process.
- The second three arguments specify the address, size, and type of the data elements for each receiving process. The size and type of the sending data and the receiving data may differ as a means of converting data types.
MPI_Scatter()

- The seventh argument specifies the root process that is the source of the data.
- The eighth argument specifies the MPI communicator to use.

Notes:
This routine distributes data from the root process to all other processes, including the root. A more sophisticated version of the routine, MPI_Scatterv(), allows the root process to send different amounts of data to the various processes. Details can be found in the MPI standard.

Return value:
An MPI error code.
MPI_Reduce()

```c
int MPI_Reduce(
    void *sendBuffer,  // Address at which to receive data
    void *recvBuffer,  // Number of elements to receive
    int count,         // Type of each element
    MPI_Datatype datatype,  // ID of sending process
    MPI_OP op,         // MPI operator
    int root,          // Process that will contain result
    MPI_Comm comm      // MPI communicator
);
```

Notes:

This routine implements a reduce operation. A special form of this routine, MPI_Allreduce(), treats all processes as if they were the root, meaning that the reduced value will be passed to all processes at the address specified by the second argument. MPI_Allreduce() is equivalent to a call to MPI_Reduce() followed by a call to MPI_Bcast(), which broadcasts values to all processes within a communicator.

Return value:

An MPI error code.
Count 3’s

Using MPI_Reduce and MPI_Scatter
MPI_Gather()

```c
int MPI_Gather(
    void *sendbuffer,  // Address of the data to send
    int sendcount,     // Number of data elements to send
    MPI_Datatype sendtype,  // Type of data elements to send
    void *recvbuffer,   // Address of buffer to receive data
    int recvcount,      // Number of data elements to receive
    MPI_Datatype recvtype,  // Type of data elements to receive
    int root,           // Rank of the root process
    MPI_Comm *comm      // An MPI communicator
);  
```

Arguments:

- The first three arguments specify the address, size, and type of the data elements that will be sent from each process.
- The second three arguments specify the address, size, and type of the data elements of the receiving process, also known as the root process. The size and type of the sending data and the receiving data may differ as a means of converting data types.
- The seventh argument specifies the root process that will receive the data.
- The eighth argument specifies the MPI communicator to use.

Notes:

This routine gathers data from all processes in a communicator, leaving the result in the root process. A more sophisticated version of the routine, `MPI_Gatherv()`, allows the root process to send different amounts of data to the various processes. Details can be found in the MPI standard.

Return value:

An MPI error code.
int MPI_Allreduce(
    void *sendBuffer,
    void *recvBuffer,
    int count,
    MPI_Datatype datatype,
    MPI_OP op,
    MPI_Comm comm);

-- every process sends data, every process receives the result of the reduction using the specified operation
MPI_Bcast()

```c
int MPI_Bcast(
    void *buffer,       // Address of the data to send
    int count,          // Number of data elements to send
    MPI_Datatype datatype,  // Type of data elements to send
    int root,           // Rank of the root task
    MPI_Comm *comm      // An MPI communicator
);
```

Arguments:
- The first three arguments specify the address, size, and type of the data elements to send to each process.
- The fourth argument specifies the rank of the root, or sending, process.
- The fifth argument specifies MPI communicator to use.

Notes:
This routine broadcasts data from the root process to all other processes in the communicator. Unlike MPI_Scatter() and MPI_Gather(), the number of elements and the types of the elements must be the same for the root process and the receiving processes.

Return value:
An MPI error code.
**MPI_Scan()**

```c
MPI_Scan()
int MPI_Scan(
    void *sendBuffer,
    void *recvBuffer,
    int count,
    MPI_Datatype datatype,
    MPI_OP op,
    MPI_Comm comm
);
```

**Notes:**

This routine has the same interface as the reduce operation except it does not require a root process.

**Return value:**

An MPI error code.

**MPI_scan** is an inclusive scan, i.e. the prefix reduction on process i includes the data from process i.
MPI_Barrier()

```c
int MPI_Barrier( 
    MPI_Comm *comm 
) ;
```

**Arguments:**

This argument specifies the MPI communicator to use.

**Notes:**

This routine blocks until all processes in the communicator have arrived at this point.

**Return value:**

An MPI error code.