Estructura básica de un programa en MPI

```
      .
      .
      .
#include “mpi.h”
      .
      .
      .

int main (int argc, char * argv[]) {
      .
      .
      .

/* No utilizar ninguna instrucción MPI antes de MPI_Init */
MPI_Init(&argc, &argv);
      .
      . /* Código utilizando MPI */
      .

MPI_Finalize();
/* No utilizar ninguna instrucción MPI detrás de MPI_Finalize */
      .
      .
      .
}
```

Compilación de un programa en MPI

```
cc fuente.c –o ejecutable.exe –lmpi
```

Ejecución de un programa en MPI

En averroes se utiliza un programa que es el encargado de lanzar el programa y construir todos los procesos que se desean:

```
mpirun –np num_procesos ejecutable
```

Envío y Recepción de mensajes en MPI

Envío del mensaje `message` de tamaño `count` elementos de tipo `datatype`. El mensaje se manda al proceso `dest` con un mensaje etiquetado por `tag` dentro del grupo de comunicación `comm`. El envío es NO bloqueante.

```
int MPI_Send (
      void *           message     /* entrada */,
      int              count       /* entrada */,
      MPI_Datatype     datatype    /* entrada */,
      int              dest        /* entrada */,
      int              tag         /* entrada */,
      MPI_Comm         comm        /* entrada */
);
```

Recepción del mensaje `message` de tamaño `count` elementos de tipo `datatype`. El mensaje se recibe desde el proceso `dest` con un mensaje etiquetado por `tag` dentro del grupo de comunicación `comm`. La recepción es BLOQUEANTE.

```
int MPI_Recv (
      void *            message     /* salida  */,
      int               count       /* entrada */,
      MPI_Datatype      datatype    /* entrada */,
      int               source      /* entrada */,
      int               tag         /* entrada */,
      MPI_Comm          comm        /* entrada */,
      MPI_Status *      status      /* salida  */
);
```

Se pueden usar las siguientes constantes previamente definidas:

Tipo de dato MPI (MPI_Datatype)	Tipo de dato en C
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Para recibir desde cualquier fuente, se puede utilizar la constante `MPI_ANY_SOURCE`. Para recibir cualquier mensaje sin importar el tag, se puede utilizar la constante `MPI_ANY_TAG`.

Para finalizar, el último parámetro de `MPI_Recv` devuelve información sobre los datos que se han recibido. `MPI_Status` es una estructura que contiene al menos tres campos:

```
MPI_SOURCE
MPI_TAG
MPI_ERROR
```

En los que se indica, respectivamente, el identificador de la fuente, de la etiqueta del mensaje recibido y, en su caso, del error producido en el envío.

También almacena el número de elementos recibidos pero no es accesible directamente por parte del usuario, para obtener dicho valor se puede utilizar la función:

```
int MPI_Get_count (
      MPI_Status *            status            /* entrada */,
      MPI_Datatype      datatype    /* entrada */,
      int *             count_ptr   /* salida */
);
```

Comunicadores y Procesos en MPI

Para conocer el número de procesos presentes en la ejecución de un programa se utiliza la función:

```
int MPI_Comm_size (
      MPI_Comm          comm              /* entrada */,
      int *             number_of_proceses      /* salida */
);
```

Un comunicador es un conjunto de procesos que pueden mandarse mensajes entre sí. Por defecto existe un comunicador `MPI_COMM_WORLD`. Para determinar el rango o identificador de un proceso dentro de un comunicador se utiliza la función:

```
int MPI_Comm_rank (
      MPI_Comm          comm        /* entrada */,
      int *             my_rank     /* salida */
);
```