```
#include <stdio.h>
#include <mpi.h>


int main (argc, argv)
     int argc;
     char *argv[];
{
  int rank, size;

  MPI_Init (&argc, &argv);          /* starts MPI */
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);     /* get current process id */
  MPI_Comm_size (MPI_COMM_WORLD, &size);      /* get number of processes */
  printf( "Hello world from process %d of %d\n", rank, size );
  MPI_Finalize();
  return 0;
}
```

```
mpicc ./hello.c -o sayhello
mpiexec -n 7 ./sayhello
```

```
MPI_Send(
    void* data,
    int count,
    MPI_Datatype datatype,
    int destination,
    int tag,
    MPI_Comm communicator)
```

```
MPI_Recv(
    void* data,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm communicator,
    MPI_Status* status)
```

```
int number;
if (rank == 0) {
    number = 5;
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
} else if (rank == 1) {
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
    printf("Process 1 received number %d from process 0\n",
           number);
}
```

```
MPI_Barrier(MPI_Comm communicator)
```

```
MPI_Bcast(void* buffer, int count, MPI_Datatype datatype,int root, MPI_Comm comm)
```

```
if (rank == root)
  {
    number=5;
```

```
    }
    MPI_Bcast(number, 1, MPI_INT, root, MPI_COMM_WORLD);
    printf( "Message from process %d : %s\n", rank, message);
```

```
MPI_Reduce(
    void* send_data,
    void* recv_data,
    int count,
    MPI_Datatype datatype,
    MPI_Op op,
    int root,
    MPI_Comm communicator)
```

```
MPI_Allreduce(
    void* send_data,
    void* recv_data,
    int count,
    MPI_Datatype datatype,
    MPI_Op op,
    MPI_Comm communicator)
```

```
float *rand_nums = NULL;
rand_nums = create_rand_nums(num_elements_per_proc);

// Sum the numbers locally
float local_sum = 0;
int i;
for (i = 0; i < num_elements_per_proc; i++) {
  local_sum += rand_nums[i];
}

// Print the random numbers on each process
printf("Local sum for process %d - %f, avg = %f\n",
       world_rank, local_sum, local_sum / num_elements_per_proc);

// Reduce all of the local sums into the global sum
float global_sum;
MPI_Reduce(&local_sum, &global_sum, 1, MPI_FLOAT, MPI_SUM, 0,
           MPI_COMM_WORLD);

// Print the result
if (world_rank == 0) {
  printf("Total sum = %f, avg = %f\n", global_sum,
         global_sum / (world_size * num_elements_per_proc));
}
```

```
int MPI_Iprobe(
  int source,
  int tag,
  MPI_Comm comm,
  int *flag,
  MPI_Status *status
);
```

Allow checking of incoming messages without actual receipt of them. returns flag = true if there is a message that can be received and that matches the pattern specified by the arguments source, tag, and comm. It is a non-blocking operation. The blocking counterpart of MPI_Probe hangs on until a matching message has been found.