1 Examples

- A program that first broadcasts the name of the master process then each nodes send hello messages to master node.

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

#define TRUE 1
#define FALSE 0
#define MASTER_RANK 0

main(argc, argv)
int argc;
char *argv[];
{
   int count, pool_size, my_rank, my_name_length, i_am_the_master = FALSE;
   char my_name[BUFSIZ], master_name[BUFSIZ], send_buffer[BUFSIZ],
        recv_buffer[BUFSIZ];
   MPI_Status status;

   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &pool_size);
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   MPI_Get_processor_name(my_name, &my_name_length);

   if (my_rank == MASTER_RANK) {
      i_am_the_master = TRUE;
      strcpy (master_name, my_name);
   }

   MPI_Bcast(master_name, BUFSIZ, MPI_CHAR, MASTER_RANK, MPI_COMM_WORLD);

   sprintf(send_buffer, "hello %s, greetings from %s, rank = %d",
           master_name, my_name, my_rank);
   MPI_Send (send_buffer, strlen(send_buffer) + 1, MPI_CHAR,
             MASTER_RANK, 0, MPI_COMM_WORLD);

   if (i_am_the_master) {
      for (count = 1; count <= pool_size; count++) {
         MPI_Recv (recv_buffer, BUFSIZ, MPI_CHAR, MPI_ANY_SOURCE, MPI_ANY_TAG,
                   MPI_COMM_WORLD, &status);
         printf ("%s\n", recv_buffer);
```

```
      }
   }

   MPI_Finalize();
}
```

- A program that takes data from process zero and sends it to all of the other processes by sending it in a ring. That is, process **i** should receive the data and send it to process **i+1**, until the last process is reached. Assume that the data consists of a single integer. Process zero reads the data from the user.

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

int main( argc, argv )
int argc;
char **argv;
{
    int rank, value, size;
    MPI_Status status;

    MPI_Init( &argc, &argv );

    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    do {
if (rank == 0) {
    scanf( "%d", &value );
    MPI_Send( &value, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD );
}
else {
    MPI_Recv( &value, 1, MPI_INT, rank - 1, 0, MPI_COMM_WORLD,
      &status );
    if (rank < size - 1)
MPI_Send( &value, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD );
}
printf( "Process %d got %d\n", rank, value );
    } while (value >= 0);

    MPI_Finalize( );
    return 0;
}
```

- A program that reads an integer value from the terminal and distributes the value to all of the MPI processes. Each process should print out its rank and the value it received. Values should be read until a negative integer is given as input.

  You may find it helpful to include a fflush(stdout); after the printf calls in your program. Without this, output may not appear when you expect it.

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

int main( argc, argv )
int argc;
char **argv;
{
    int rank, value;
    MPI_Init( &argc, &argv );

    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    do {
if (rank == 0)
    scanf( "%d", &value );

MPI_Bcast( &value, 1, MPI_INT, 0, MPI_COMM_WORLD );

printf( "Process %d got %d\n", rank, value );
    } while (value >= 0);

    MPI_Finalize( );
    return 0;
}
```

- A program that broadcast routine is used to communicate different datatypes with a single MPI broadcast (**MPI_Bcast**) call. MPI datatypes are used. All processes exit when a negative integer is read.

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

int main( argc, argv )
int argc;
char **argv;
{
```

```
    int          rank;
    struct { int a; double b; } value;
    MPI_Datatype mystruct;
    int          blocklens[2];
    MPI_Aint     indices[2];
    MPI_Datatype old_types[2];

    MPI_Init( &argc, &argv );

    MPI_Comm_rank( MPI_COMM_WORLD, &rank );

    /* One value of each type */
    blocklens[0] = 1;
    blocklens[1] = 1;
    /* The base types */
    old_types[0] = MPI_INT;
    old_types[1] = MPI_DOUBLE;
    /* The locations of each element */
    MPI_Address( &value.a, &indices[0] );
    MPI_Address( &value.b, &indices[1] );
    /* Make relative */
    indices[1] = indices[1] - indices[0];
    indices[0] = 0;
    MPI_Type_struct( 2, blocklens, indices, old_types, &mystruct );
    MPI_Type_commit( &mystruct );

    do {
if (rank == 0)
    scanf( "%d %lf", &value.a, &value.b );

MPI_Bcast( &value, 1, mystruct, 0, MPI_COMM_WORLD );

printf( "Process %d got %d and %lf\n", rank, value.a, value.b );
    } while (value.a >= 0);

    /* Clean up the type */
    MPI_Type_free( &mystruct );
    MPI_Finalize( );
    return 0;
}
```

- Study the previous week's examples (translation from fortran to C and calculation of PI).