- 代码思路分析：

首先，我们首先要从用户获得各输入，即调用$Get_input()$函数，通过传引用读入桶的数量、数据区间左右值、数据个数等，同时保证进程数能整除数据数。

接下来是确定各桶的区间左右值，进入$Set_bin()$函数，仅在0号进程中算出各桶左右值，并利用$MPI_Bcast()$将bin_maxes, bin_count广播到各进程

```
// set bin_maxes for each proc
MPI_Bcast(bin_maxes, bin_count,
MPI_FLOAT, 0, comm);

// reset loc_bin_cts of each proc
MPI_Bcast(loc_bin_cts, bin_count,
MPI_INT, 0, comm);
```

再者是生成源数据并通过$MPI_Scatter()$划分源数据，并将划分后的数据分配给各进程，即Foster方法论中的分配任务一步

```
MPI_Scatter(data, local_data_count,
MPI_FLOAT, local_data,
            local_data_count,
MPI_FLOAT, 0, comm);
      free(data);
} else {
      MPI_Scatter(data,
local_data_count, MPI_FLOAT,
local_data,
            local_data_count,
MPI_FLOAT, 0, comm);
}
```

之后便是主体部分，也就是对输入数据进行处理，进入$Find_bins()$函数，对每个读入的$data$，我们首先确定其应落在哪个桶的区间内

```
for (i = 0; i < local_data_count;
i++) {
   bin = Which_bin(local_data[i],
bin_maxes, bin_count, min_meas);
   loc_bin_cts[bin]++;
}
```

进入$Which_bin()$子函数，这里就需要我们填充代码，我们约定各个桶的区间是左闭右开的，采用对桶号进行二分查找的方法找到一个数据在哪个桶中，因为当桶的标号为$m == 0$时，不存在上一个桶，因此这里需要特判，此时将左区间置为min_meas，其他部分就是正常的二分查找。详细代码及注释如下：

```
int Which_bin(float data, float
bin_maxes[], int bin_count,
      float min_meas) {

   // PLEASE ADD THE RIGHT CODES

   int left = 0;                    // 初
始化查询区间左值
   int right = bin_count - 1;    // 初
始化查询区间右值
   while (left <= right) {
      int mid = (left + right) / 2;
// 每次都求中值
      float l1 = (mid == 0) ?
min_meas : bin_maxes[mid - 1];    //
特判第0个桶的情况
      float r1 = bin_maxes[mid];
```

```
        if (l1 <= data && data < r1) {
// 找到data落入的区间
            return mid;
        }
        else if (data < l1) { // 区间左
移
            right = mid - 1;
        }
        else left = mid + 1;  // 区间右
移
    }
    printf("Uh oh . . .\n"); // 找不到
    return 0;

}  /* Which_bin */
```

之后回到$Find_bins()$中，利用$MPI_Reduce()$函数，并将MPI_OP参数置为MPI_SUM对各进程的loc_bin_cts进行求和运算，并将结果送到0号进程的bin_cts中

```
MPI_Reduce(loc_bin_cts, bin_counts,
bin_count, MPI_FLOAT, MPI_SUM, 0,
comm);
```

最后由0号进程调用$Print_histo$进行输出。

然后是内存的释放，以及结束MPI。

- 程序输入、运行结果及分析：

- 编译

```
mpicc -Wall -g -o mpi mpi_histogram.c
```

- 运行

```
mpiexec -n 4 ./mpi
```

- 输入：

```
Enter the number of bins
5
Enter the minimum measurement
0.0
Enter the maximum measurement
10.0
Enter the number of data
12
```

- 输出：

```
Generated data:
   8.402 3.944 7.831 7.984 9.116
1.976 3.352 7.682 2.778 5.540 4.774
6.289

0.000-2.000:    X
2.000-4.000:    XXX
4.000-6.000:    XX
6.000-8.000:    XXXX
8.000-10.000:   XX
```

- 分析：

> 为了方便检查，这里我开启了程序中写好的Debug模式，能够输出生成各data的值，对比输入、生成代码，以及最终打印出来的结果，我们可以看到程序运行正常。