```
//
There
are
some
parts
in the
file
that
you
need to
modify,
Please
do
          // not modify everything


          // Modify only between the block which mentions what to modiy
          // ===== DO NOT MODIFY BELOW THIS ======
          #include "hw04.h"

          void cleanup (FILE * fpin)
          {
            fclose (fpin);
          }


          // read the data, return true if success, false if fail
          bool readData (FILE * fpin, DataPoint * *dp, int nval, int dim)
          {
            int niter, diter;
            for (niter = 0; niter < nval; niter++)
              {
                for (diter = 0; diter < dim; diter++)
                 {
                   if (fscanf (fpin, "%d", &dp[niter]->data[diter]) == 0)
                     {
                       return false;
                     }
                 }
              }
            return true;
          }



          // write the output centroids to the file
```

```c
// check for all the NULL before calling this function, it does not check
void writeCentroids (const char *filename, Centroid * *centroids, int kval)
{
  FILE * fpout = fopen (filename, "w");
  if (fpout == NULL)
    {
      fprintf (stderr, "File %s, cannot be open\n", filename);
      exit (1);
    }
  int kiter;
  // sort the centroids for ease of grading
  qsort (centroids, kval, sizeof (Centroid *), Centroid_cmp);
  for (kiter = 0; kiter < kval; kiter++)
    {
      Centroid_print (centroids[kiter], fpout);
    }
  fclose (fpout);
}



//===== DO NOT MODIFY ABOVE THIS ======
// Modify the functions only in the enclosed box

#ifdef TEST_DIST
// distance - funtion to get the distance between present centroid and
datapoint
// @param DataPoint * - pointer to DataPoint structure from which distance
needs to be calculated
// @param Centroid * - pointer to Centroid struct from which distance is being
calculated
long long int
distance (const DataPoint * datapoint, const Centroid * centroid)
{
  // since this is for comparison only, there is no need to call sqrt
  long long int sum = 0;     // must initialize to zero
  // find Euclidean distance and then return 'sum' without calling sqrt
  int dim = datapoint->dimension;
  int diter;
  for (diter = 0; diter < dim; diter ++)
    {
      long long int diff = (datapoint->data[diter]) - (centroid->data[diter]);
      sum += (diff * diff);
    }
  return sum;
}
```

```
#endif

#ifdef TEST_CLOSESTCENTROID

// for a data point, find the closest centroid
// 1. calculate the distance between the data point and the first
// centroid, set it to the minimum distance
// 2. go through the other centroids and calculate distance from each.
// 3. Keep on updating minimum distance and index value of the
// centroid from which the distance is smaller than previously seen,
int closestCentroid (int kval, DataPoint * datapoint, Centroid * *centroids)
{
  int mindex = 0; // the index of the closest centroid
  // Please note that return value of distance is long long int, so initialize
the values with the same type
  // go through each centroid and find the distance
  // keep track of minimum difference and index of centroid which has the
smallest distance
  int kiter;
  long long int mindiff = distance(datapoint, centroids[0]);
  for (kiter = 1; kiter < kval; kiter ++)
    {
      long long int diff = distance(datapoint, centroids[kiter]);
      if (diff < mindiff)
       {
         mindiff = diff;
         mindex = kiter;
       }
    }
  return mindex;
}


#endif

#ifdef TEST_KMEAN
// kmean - function which finds the k clusters in the data set
// kval - # of clusters
// nval - # of datapoints
// datapoints - array of datapoints
// centroids - array of centroids
//
// return the total distances of datapoints from their centroids
void kmean (int kval, int nval, DataPoint * *datapoints, Centroid *
```

```
*centroids)
{
		bool finished = false;
		// initialize each data point to a cluster between 0 and kval -
1
		int kiter;
		// reset all centroids
		for (kiter = 0; kiter < kval; kiter ++)
		{
			Centroid_reset(centroids[kiter]);
		}
		int nind; // data index
		for (nind = 0; nind < nval; nind ++)
		{
			int clu = rand() % kval;
			datapoints[nind] -> cluster = clu;
			Centroid_addPoint(centroids[clu],
datapoints[nind]);
		}
		// find the centroid for
		for(kiter = 0; kiter < kval; kiter++)
		{
			Centroid_findCenter(centroids[kiter]);
		}


		// adjust the clusters and recompute the centroid
		do
		{
			finished = true;
			// for each data point, find the index of the
centroid that is the closest
			// store that index in DataPoint's structure's
cluster value.
			for (nind = 0; nind < nval; nind ++)
			{
				int mindex = closestCentroid(kval,
datapoints[nind], centroids);
				if(mindex != datapoints[nind]-
>cluster)
				{
					finished = false;
				}
				datapoints[nind]->cluster =
mindex;
			}
```

```
                        // reset all centroids
                        for (kiter = 0; kiter < kval; kiter ++)
                        {
                                Centroid_reset(centroids[kiter]);


                        }
                        // go through each datapoint
                        // add this datapoint to its centroid using
Centroid_addPoint function
                        for (nind = 0; nind < nval; nind ++)
                        {
                                int clu = datapoints[nind]-
>cluster; // this point's cluster
                                Centroid_addPoint(centroids[clu],
datapoints[nind]);
                        }
                        // find the centroid for
                        for(kiter = 0; kiter < kval; kiter++)
                        {

       Centroid_findCenter(centroids[kiter]);
                        }
                } while (finished == false);


}

#endif

/*===== DO NOT MODIFY BELOW THIS ======*/
int
main (int argc, char * *argv)
{

  // argv[1]: name of input file
  // argv[2]: value of k/ number of centroids
  // argv[3]: name of output file
  if (argc < 4)
    {
      fprintf (stderr, "argc is %d, not 4\n", argc);
      return EXIT_FAILURE;
    }

  // opening file to read the data points from
  FILE * fpin = fopen (argv[1], "r");
```

```
if (fpin == NULL)
  {
    fprintf (stderr, "fopen %s fail\n", argv[1]);
    return EXIT_FAILURE;
  }

// convert long to int
int kval = (int) strtol (argv[2], NULL, 10);
if (kval <= 0)
  {
    fprintf (stderr, "kval is %d, must be positive\n", kval);
    return EXIT_FAILURE;
  }
// control the random number sequence
int randseed = 1729;       // any integer will do, DO NOT CHANGE
srand (randseed);

// getting number of datapoints
int nval;
fscanf (fpin, "%d", &nval);
if (nval < kval)
  {
    fprintf (stderr, "nval= %d must be greater than kval = %d\n", nval,
             kval);
    cleanup (fpin);
    return EXIT_FAILURE;
  }

// getting dimensions of the data from the file
int dim;
fscanf (fpin, "%d", &dim);
if (dim < 2)
  {
    fprintf (stderr, "nval= %d must be greater than kval = %d\n", nval,
             kval);
    cleanup (fpin);
    return EXIT_FAILURE;
  }

// allocate memory for the data points
DataPoint * *datapoint_array = DataPoint_createArray (nval, dim);
if (datapoint_array == NULL)
  {
    printf ("Error in creating datapoint array\n");
    cleanup (fpin);
    return EXIT_FAILURE;
```

```
    }

  // allocate memory for array of centroids
  Centroid * *centroids = Centroid_createArray (kval, dim);
  if (centroids == NULL)
    {
      printf ("Error in creating centroids array\n");
      DataPoint_freeArray (datapoint_array, nval);
      cleanup (fpin);
      return EXIT_FAILURE;
    }

  // read the data from the file
  if (readData (fpin, datapoint_array, nval, dim) == false)
    {
      printf ("Error in reading data array\n");
      cleanup (fpin);
      DataPoint_freeArray (datapoint_array, nval);
      return EXIT_FAILURE;
    }

  // calling kmean function to find the cetroids
  kmean (kval, nval, datapoint_array, centroids);


       //writing those centroids to the file
  writeCentroids (argv[3], centroids, kval);
  // free all the allocated spaces
  DataPoint_freeArray (datapoint_array, nval);
  Centroid_freeArray (centroids, kval);
  cleanup (fpin);
  return EXIT_SUCCESS;
}
```