```
class CClustSet : public CObject
{
public:
      CClustSet() {}
      DECLARE_DYNCREATE(CClustSet)

      CClustSet(long nPoints, long feet);
      CClustSet(CClustSet *CSet);
      ~CClustSet();

      int GetSize() {return m_nClusters;}
      int GetCluster(int pos) {return m_Cluster[pos];}
      void SetCluster(int pos, int cluster) {m_Cluster[pos] = cluster;}

      int pCluster(long Point) {return m_pCluster[Point];}  // Point Cluster
      BOOL Add(unsigned int  CurClust, long CellCount, long LineCount);
      void SetPointClust(long  point,  unsigned int cluster) {m_pCluster[point] = cluster;}
      void ResetClusts();

      BOOL IsNoMerge(int Clust, unsigned int NM_Count);
      BOOL IsMergable(int Clust, unsigned int LM_Count);
      long GetClustLines(unsigned int Cluster);

      void ClusConsolidate();
      void SetOutlier(BOOL Outliers);

      int GetOutStart() {return m_OutlStart;}

//Attributes
      long m_Feet;      // Distance this set created at

      // Cluster Information
      unsigned int m_nMaxClusters;
      unsigned int m_nClusters;

      int m_OutlStart;

      long *m_CellCount;
      long *m_NumLines;
```

```
        unsigned int *m_Cluster;

        // Point Data
        long m_nPoints;
        int *m_pCluster;
        CUIntArray m_Perimeter;

        // Size of number of clusters
        CDblArray m_LongCentroids;
        CDblArray m_LatCentroids;
        CDblArray m_ServeLongCentroids;
        CDblArray m_ServeLatCentroids;
        CDblArray m_ClusArea;
        CStringArray m_ClusLabels;

public:
        void AllocOnClus(void);
};

class CClustComp
{
public:
//////////////// CHANGED ////////////////////////////////////////////////
//      CClustComp(int nPoints=0, double *Long=NULL, double *Lat=NULL, float *Weight=NULL, long feet=0);
        CClustComp(CData* Data = NULL, SPECS* Specs = NULL);
        ~CClustComp();

//////////////// CHANGED ////////////////////////////////////////////////
//  void Initialize(long nPoints, double *Long, double *Lat, float *Weight, long feet);
        void Initialize(CData* Data, SPECS* Specs);

// Attributes
        // Grid Specific
        CClustGrid m_Lines;
        CClustGrid m_Member;

        // Point Data
        double m_MinLong;
        double m_MaxLong;
        double m_MinLat;
```

```
      double m_MaxLat;
      double m_Width;
      double m_Height;

      unsigned long m_MaxPoints;
      unsigned long m_nPoints;
    double *m_pLong;
    double *m_pLat;
    float *m_pWeight;

    // Cluster Data
    unsigned int m_LastCluster;           // number clusters (total count)
      CClustSet *m_CurClust;

      // Parameters
      int m_MinLines;
      unsigned int m_LM_Count;
      unsigned int m_NM_Count;
      long m_MaxClustDist;

// Implementation
      void Clear();
      long Index(unsigned int x,unsigned int y) {return x + y*m_Lines.GetNCols();}
    void SetGrid(long feet);
      void CheckAdjacent(unsigned int iStart, unsigned int jStart, char *Checked, CDWordArray &CellList,
            int perim, int *xLoop, int *yLoop, BOOL &Mergable, BOOL NoMerge, CWordArray &WithinClusts);


      void ComputeClusters(SPECS* Specs, BOOL Outliers);
      void Finalize();

      BOOL WithinMaxDist(CDWordArray &CellList, long CurCell, CWordArray &WithinClusts);
      void GetMemberGrid(long feet, CClustGrid &Grid, int Cluster);

      void ComputePerimeter(void);
      void ComputeArea(void);
      void ComputeCentroids(BOOL OfMaxPtDist = FALSE);
      void AssignLabels(void);
};
```

```
#endif


#include "stdafx.h"
#include "GrdClust.h"

#include "Perimeter.h"
#include "Area.h"
#include "Data.h"

#include "string.h"
#include "math.h"

////////////////////////////////////////////////////////////////////////////////////////////////
//
//  Distance functions
//
////////////////////////////////////////////////////////////////////////////////////////////////

static double Radius = 3956.0;                          // approx earth radius in miles
static double Radians = acos(-1.0) / 180.0;     // pi / 180
static double Distance(double x1, double y1, double x2, double y2)
{
      double Distance = 0.0;

      if (x1 != x2 || y1 != y2) {
            x1 *= Radians;
            x2 *= Radians;
            y1 *= Radians;
            y2 *= Radians;

            Distance = pow(sin((y2-y1) / 2.0), 2.0) + cos(y1) * cos(y2) * pow(sin((x2-x1) / 2.0), 2.0);
            Distance = (2 * asin(min(1, pow(Distance, .5)))) * Radius;
      }

      return Distance;
}
```

```
static double LatFromDist(double Lat, double Distance)
{
      return (Distance/Radius + Lat*Radians) / Radians;
}

static double LongFromDist(double x1, double y1, double Distance)
{
      x1 *= Radians;
      y1 *= Radians;

      return (x1 + 2 * asin(pow( pow(sin(Distance/(2*Radius)), 2.) / (cos(y1)*cos(y1)), .5))) / Radians;
}

//////////////////////////////////////////////////////////////////////////////////////////////
//
//  CClustGrid:  Grid class for cluster computation
//
//////////////////////////////////////////////////////////////////////////////////////////////

CClustGrid::CClustGrid()
{
      m_dFeet = 0;
      m_Height = 0;
      m_Width = 0;
      m_nPoints = 0;

      m_Grid = NULL;
      m_nRows = NULL;
      m_nCols = NULL;
      m_pRow = NULL;
      m_pCol = NULL;
}

CClustGrid::CClustGrid(double MinLong, double MinLat, double MaxLong, double MaxLat, long feet, long
nPoints, double *pLong, double *pLat)
{
      m_dFeet = 0;
      m_Height = 0;
      m_Width = 0;
      m_nPoints = 0;
```

```
	m_Grid = NULL;
	m_nRows = NULL;
	m_nCols = NULL;
	m_pRow = NULL;
	m_pCol = NULL;

	Initialize(MinLong, MinLat, MaxLong, MaxLat, feet, nPoints, pLong, pLat);
}

CClustGrid::~CClustGrid()
{
	delete[] m_Grid;
	delete[] m_pRow;
	delete[] m_pCol;
}

long CClustGrid::GetDistance(long Cell1, long Cell2)
{
	long row1 = floor(Cell1 / m_nCols);
	long col1 = Cell1 - row1*m_nCols;
	long row2 = floor(Cell2 / m_nCols);
	long col2 = Cell2 - row2*m_nCols;

	float distance = pow(pow(row2-row1,2) + pow(col2-col1,2), .5) * m_dFeet;


	return (long)distance;
}

long CClustGrid::GetRLDistance(long Cell1, long Cell2)
{
	long row1 = floor(Cell1 / m_nCols);
	long col1 = Cell1 - row1*m_nCols;
	long row2 = floor(Cell2 / m_nCols);
	long col2 = Cell2 - row2*m_nCols;

	float distance = (labs(row2-row1) + labs(col2-col1)) * m_dFeet;
```

```
	return (long)distance;
}

void CClustGrid::Initialize(double MinLong, double MinLat, double MaxLong, double MaxLat, long feet,
                            long nPoints, double *pLong, double *pLat, CULongArray* pPtMask /*NULL*/)
{
	if (!nPoints || !feet)
		return;

	// Determine width and height of grid
	double dist1 = Distance(MinLong, MinLat, MaxLong, MinLat);
	double dist2 = Distance(MinLong, MaxLat, MaxLong, MaxLat);
	m_Width = (dist1 > dist2) ? dist1 : dist2;
	m_Height = Distance(MinLong, MinLat, MinLong, MaxLat);

	// Deallocate
	delete[] m_Grid;
	delete[] m_pRow;
	delete[] m_pCol;

	// Initialize member variables
	m_dFeet = feet;
	m_nPoints = nPoints;
	m_nRows = (unsigned int)(m_Height / ((double)m_dFeet/5280.) + 1);
	m_nCols = (unsigned int)(m_Width / ((double)m_dFeet/5280.) + 1);

	// Allcoate member variables
	m_Grid = new long[m_nRows * m_nCols];
	for (long Point = 0; Point < long(m_nRows * m_nCols); Point++)
		m_Grid[Point] = 0;

	m_pRow = new unsigned int[m_nPoints];
	m_pCol = new unsigned int[m_nPoints];
	for (Point = 0; Point < m_nPoints; Point++)
		m_pRow[Point] = m_pCol[Point] = 0;

	// Assign point cell locations
	BOOL bFound;
	long CurPoint = 0, nMaskIndex = 0;
	double miles = (double)m_dFeet/5280.;
```

```
double BegLat, BegLong, EndLat, EndLong;
for (Point = 0; Point < m_nPoints; Point++)
{// Loop works because data sorted by Lat then Long

      bFound = FALSE;
      if (pPtMask)
      {
            if (pPtMask->GetAt(nMaskIndex) == Point)
                  ++nMaskIndex;
            else
                  bFound = TRUE;
      }

      if (!bFound)
      {
            BegLat = EndLat = MinLat;
            for (unsigned int i = 0; i < m_nRows; i++)
            {
                  BegLat = EndLat;
                  EndLat = LatFromDist(BegLat, miles);

                  if (!(pLat[Point] < EndLat))
                        continue;

                  BegLong = EndLong = MinLong;
                  for (unsigned int j = 0; j < m_nCols; j++)
                  {
                        BegLong = EndLong;
                        EndLong = LongFromDist(BegLong, (BegLat + EndLat) / 2.0, miles);
                        if (pLong[Point] < EndLong)
                        {
                              m_pRow[Point] = i;
                              m_pCol[Point] = j;
                              bFound = TRUE;
                              break;
                        }
                  }
                  if (bFound)
                        break;
            }
```

```
            }
            if (pPtMask && nMaskIndex == pPtMask->GetSize())
                break;
        }
}

/////////////////////////////////////////////////////////////////////////////
//
//  CClustSet:  Storage for cluster information
//
/////////////////////////////////////////////////////////////////////////////

IMPLEMENT_DYNCREATE(CClustSet, CObject)

CClustSet::CClustSet(CClustSet *CSet)
{
        m_Feet = CSet->m_Feet;

        m_nMaxClusters = CSet->m_nMaxClusters;

        m_CellCount = new long[m_nMaxClusters];
        m_NumLines = new long[m_nMaxClusters];
        m_Cluster = new unsigned int[m_nMaxClusters];

        m_nPoints = CSet->m_nPoints;
        m_pCluster = new int[m_nPoints];
        for (long i=0; i<m_nPoints; i++)
                m_pCluster[i] = CSet->m_pCluster[i];

        m_OutlStart = CSet->m_OutlStart;
        m_nClusters = CSet->m_nClusters;
        for (i=0; i<m_nClusters; i++) {
                m_CellCount[i] = CSet->m_CellCount[i];
                m_NumLines[i] = CSet->m_NumLines[i];
                m_Cluster[i] = CSet->m_Cluster[i];
        }


        long sz = m_ClusLabels.GetSize();
        m_ClusLabels.SetSize(sz);
```

```
        for (i = 0; i < sz; i++)
              m_ClusLabels[i] = CSet->m_ClusLabels[i];

        sz = m_ClusArea.GetSize();
        m_ClusArea.SetSize(sz);
        for (i = 0; i < sz; i++)
              m_ClusArea[i] = CSet->m_ClusArea[i];

        sz = m_LongCentroids.GetSize();
        m_LongCentroids.SetSize(sz);
        for (i = 0; i < sz; i++)
              m_LongCentroids[i] = CSet->m_LongCentroids[i];

        sz = m_LatCentroids.GetSize();
        m_LatCentroids.SetSize(sz);
        for (i = 0; i < sz; i++)
              m_LatCentroids[i] = CSet->m_LatCentroids[i];

        sz = m_ServeLongCentroids.GetSize();
        m_ServeLongCentroids.SetSize(sz);
        for (i = 0; i < sz; i++)
              m_ServeLongCentroids[i] = CSet->m_ServeLongCentroids[i];

        sz = m_ServeLatCentroids.GetSize();
        m_ServeLatCentroids.SetSize(sz);
        for (i = 0; i < sz; i++)
              m_ServeLatCentroids[i] = CSet->m_ServeLatCentroids[i];
}

CClustSet::CClustSet(long nPoints, long feet)
{
        m_Feet = feet;

        m_nMaxClusters = 10000;
        m_nClusters = 0;

        m_CellCount = new long[m_nMaxClusters];
        m_NumLines = new long[m_nMaxClusters];
        m_Cluster = new unsigned int[m_nMaxClusters];
```

```
	m_nPoints = nPoints;
	m_pCluster = new int[nPoints];
	for (long i=0; i<nPoints; i++)
		m_pCluster[i] = 0;

	m_OutlStart = 0;
}

CClustSet::~CClustSet()
{
	delete[] m_CellCount;
	delete[] m_NumLines;
	delete[] m_Cluster;
	delete[] m_pCluster;
}


void CClustSet::ClusConsolidate()
{
	for (unsigned int i=0; i<m_nClusters; i++)
	{
		long count=0;
		for (long n=0; n<(long)m_nPoints; n++)
		{
			if (m_pCluster[n] == m_Cluster[i])
			{
				count++;
				m_pCluster[n] = 10000+i+1;
			}
		}
		m_Cluster[i] = i+1;
	}

	for (long n=0; n<(long)m_nPoints; n++)
		if (m_pCluster[n])
			m_pCluster[n] -= 10000;
}

void CClustSet::AllocOnClus()
{
```

```
	m_ClusLabels.RemoveAll();
	m_ClusArea.RemoveAll();
	m_LongCentroids.RemoveAll();
	m_LatCentroids.RemoveAll();
	m_ServeLongCentroids.RemoveAll();
	m_ServeLatCentroids.RemoveAll();

	m_ClusLabels.SetSize(m_nClusters);
	m_ClusArea.SetSize(m_nClusters);
	m_LongCentroids.SetSize(m_nClusters);
	m_LatCentroids.SetSize(m_nClusters);
	m_ServeLongCentroids.SetSize(m_nClusters);
	m_ServeLatCentroids.SetSize(m_nClusters);
}


void CClustSet::ResetClusts()
{
	if (m_OutlStart)
		m_nClusters = m_OutlStart-1;
	else
		m_nClusters = 0;
}

BOOL CClustSet::Add(unsigned int CurClust, long CellCount, long LineCount)
{
	if (m_nClusters+1 >= m_nMaxClusters) {
		AfxMessageBox("Error, Max Clusters Exceeded");
		return FALSE;
	}

	for (unsigned int i=0; i<m_nClusters; i++) {
		if (CurClust == m_Cluster[i]) {
			m_CellCount[i] += CellCount;
			m_NumLines[i] += LineCount;
			break;
		}
	}

	if (i == m_nClusters) {
```

```
            m_CellCount[m_nClusters] = CellCount;
            m_NumLines[m_nClusters] = LineCount;
            m_Cluster[m_nClusters] = (unsigned int)CurClust;

            m_nClusters++;
      }

      return TRUE;
}

void CClustSet::SetOutlier(BOOL Outliers)
{
      if (Outliers)
            m_OutlStart = m_nClusters+1;
      else
            m_OutlStart = FALSE;
}

BOOL CClustSet::IsNoMerge(int Clust, unsigned int NM_Count)
{
      BOOL NoMerge = FALSE;

      for (int i=0; i<(int)m_nClusters; i++) {
            if ((int)m_Cluster[i] == Clust) {
                  if (m_NumLines[i] > NM_Count || (Clust < m_OutlStart))
                        NoMerge = TRUE;
                  break;
            }
      }

      return NoMerge;
}

BOOL CClustSet::IsMergable(int Clust, unsigned int LM_Count)
{
      BOOL IsMergable = TRUE;

      for (int i=0; i<(int)m_nClusters; i++) {
            if ((int)m_Cluster[i] == Clust) {
                  if (m_NumLines[i] > LM_Count)
```

```
                        IsMergable = FALSE;
                    break;
                }
        }

        return IsMergable;
}

long CClustSet::GetClustLines(unsigned int Cluster)
{
        long nLines = 0;

        for (unsigned int i=0; i<m_nClusters; i++) {
                if (Cluster == m_Cluster[i]) {
                        nLines = m_NumLines[i];
                        break;
                }
        }

        return nLines;
}

////////////////////////////////////////////////////////////////////////////////////////////
//
//  CClustComp:  Calculates cluster solution using grid and storage classes
//
////////////////////////////////////////////////////////////////////////////////////////////

CClustComp::CClustComp(CData* Data, SPECS* Specs)
{
        m_pWeight = NULL;
        m_CurClust = NULL;

        Clear();

        if (Data && Specs)
                Initialize(Data, Specs);
}

CClustComp::~CClustComp()
```

```
{
	delete m_pWeight;
	delete m_CurClust;
}

void CClustComp::Clear()
{
	m_nPoints = 0;
	m_pLong = NULL;
	m_pLat = NULL;
	m_MinLong = 0;
	m_MaxLong = 0;
	m_MinLat = 0;
	m_MaxLat = 0;

	m_Width = 0;
	m_Height = 0;
	m_LastCluster = 0;
	m_MinLines = 0;
	m_LM_Count = 0;
	m_NM_Count = 0;

	m_MaxClustDist = 36000; // Should be a user accessable parameter

	delete m_pWeight;
	m_pWeight = NULL;

	delete m_CurClust;
	m_CurClust = NULL;
}

void CClustComp::Initialize(CData* Data, SPECS* Specs)
{
	if (m_nPoints)
		Clear();

	m_nPoints = Data->GetNCases();
	m_pLong = Data->GetLongitude();
	m_pLat = Data->GetLatitude();
```

```
        m_pWeight = new float[m_nPoints];
        for (long i = 0; i < m_nPoints; i++)
              m_pWeight[i] = (float)Data->GetWt(i);

        // Determine the minimum/maximum longitudes and latitudes for grid compilation
        m_MinLong = m_pLong[0];
        m_MaxLong = m_pLong[0];
        m_MinLat = m_pLat[0];
        m_MaxLat = m_pLat[0];
        for (i = 1; i < m_nPoints; i++)
        {
              if (m_MinLong > m_pLong[i])
                    m_MinLong = m_pLong[i];

              if (m_MaxLong < m_pLong[i])
                    m_MaxLong = m_pLong[i];

              if (m_MinLat > m_pLat[i])
                    m_MinLat = m_pLat[i];

              if (m_MaxLat < m_pLat[i])
                    m_MaxLat = m_pLat[i];
        }

        // Initialize grid that will contain cluster solutions
        m_Member.Initialize(m_MinLong, m_MinLat, m_MaxLong, m_MaxLat, Specs->nGridSize, m_nPoints,
                                m_pLong, m_pLat);

        // Initialize and set grid containing number of weighted lines
        m_Lines.Initialize(m_MinLong, m_MinLat, m_MaxLong, m_MaxLat, Specs->nGridSize, m_nPoints,
                                m_pLong, m_pLat);

        for (i = 0; i < m_nPoints; i++)
        {
              unsigned int row = m_Lines.GetPRow(i);
              unsigned int col = m_Lines.GetPCol(i);
              m_Lines.SetValue(Index(col, row), m_Lines.GetValue(Index(col, row)) + m_pWeight[i]);
        }
}
```

```
//    !!!!!!!  must clear after each run

BOOL CClustComp::WithinMaxDist(CDWordArray &CellList, long CurCell, CWordArray &WithinClusts)
{
      long i;

      int NewClust = m_Member.GetValue(CurCell);

      // Check if Current Cell is a member of the approved cluster list
      if (NewClust)
            for (i=0; i<WithinClusts.GetSize(); i++)
                  if (WithinClusts[i] == NewClust)
                        return TRUE;

      // Compare distance with all current members
      for (i=0; i<CellList.GetSize(); i++) {
            if (m_Member.GetRLDistance(CellList[i], CurCell) > m_MaxClustDist) {
                  return FALSE;
            }
      }

      // Compare distance of Existing cluster members with Current Cell
      CDWordArray NewClustList;
      for (i=0; i<(long)m_nPoints; i++) {
            int index = Index(m_Member.GetPCol(i), m_Member.GetPRow(i));

            if (NewClust && m_Member.GetValue(index) == NewClust && index != CurCell)
                  NewClustList.Add(index);

            for (int j=0; j<WithinClusts.GetSize(); j++) {
                  if (WithinClusts[j] == m_Member.GetValue(index)) {
                        if (m_Lines.GetRLDistance(index, CurCell) > m_MaxClustDist) {
                              return FALSE;
                        }
                        break;
                  }
            }
      }
```

```
	// Compare distance of New Cluster members with Current Cell List
	for (int n=0; n<NewClustList.GetSize(); n++) {
		long CheckCell = NewClustList[n];

		for (i=0; i<CellList.GetSize(); i++) {
			if (m_Member.GetRLDistance(CellList[i], CheckCell) > m_MaxClustDist) {
				return FALSE;
			}
		}
	}

	// Compare distance of New Cluster members with existing Cluster members
	for (n=0; n<NewClustList.GetSize(); n++) {
		long CheckCell = NewClustList[n];

		for (i=0; i<(long)m_nPoints; i++) {
			int index = Index(m_Member.GetPCol(i), m_Member.GetPRow(i));

			for (int j=0; j<WithinClusts.GetSize(); j++) {
				if (WithinClusts[j] == m_Member.GetValue(index)) {
					if (m_Lines.GetRLDistance(index, CheckCell) > m_MaxClustDist) {
						return FALSE;
					}
					break;
				}
			}
		}
	}


	// If cluster passes the test, insert into Cluster list
	if (NewClust)
		WithinClusts.Add(NewClust);

	return TRUE;
}


void CClustComp::CheckAdjacent(unsigned int iStart, unsigned int jStart, char *Checked, CDWordArray
&CellList, int perim, int *xLoop, int *yLoop, BOOL &Mergable, BOOL NoMerge, CWordArray &WithinClusts)
```

```
{
	unsigned int i, j;

	CellList.Add(Index(jStart,iStart));

	for (int l=0; l<perim; l++)
	{
		j = jStart + xLoop[l];
		i = iStart + yLoop[l];

		if (i<0 || j<0 || i>=m_Lines.GetNRows() || j>=m_Lines.GetNCols())
			continue;

		if (!Checked[Index(j,i)])
		{
			BOOL NewMergable = m_CurClust->IsMergable((int)m_Member.GetValue(Index(j,i)), m_LM_Count);
			BOOL CurNoMerge = m_CurClust->IsNoMerge((int)m_Member.GetValue(Index(j,i)), m_NM_Count);
			if ((m_Member.GetValue(Index(jStart,iStart)) == m_Member.GetValue(Index(j,i))) ||
				((Mergable || (!Mergable && NewMergable)) &&
				(!NoMerge || m_Member.GetValue(Index(j,i))==0) && !CurNoMerge))
			{
				BOOL CloseEnough = TRUE;

				if (m_Lines.GetValue(Index(j,i)))
					CloseEnough = WithinMaxDist(CellList, Index(j,i), WithinClusts);

				if (CloseEnough) {
					Checked[Index(j,i)] = TRUE;

					if (!NewMergable)
						Mergable = NewMergable;

					if (m_Lines.GetValue(Index(j,i)))
						CheckAdjacent(i, j, Checked, CellList, perim, xLoop, yLoop, Mergable,
NoMerge, WithinClusts);

				}
			}
		}
	}
```

```
}

void CClustComp::ComputeClusters(SPECS* Specs, BOOL Outliers)
{
	// Assign member variables
	m_NM_Count = Specs->nMaxMerge;
	if (!Outliers)
	{
		m_MinLines = Specs->nMinN;
		m_LM_Count = Specs->nLimitedMerge;
	}
	else
	{
		m_MinLines = 1;
		m_LM_Count = Specs->nMinN;
	}
	long iterations = Specs->nMaxClusFt / Specs->nGridSize;

	// Declare and initialize
	unsigned int nRows = m_Lines.GetNRows();
	unsigned int nCols = m_Lines.GetNCols();
	long dFeet = m_Lines.GetDistance();

	if (!Outliers)
		delete m_CurClust;
	if (!m_CurClust)
		m_CurClust = new CClustSet(m_nPoints, dFeet);

	m_CurClust->SetOutlier(Outliers);
	CClustSet *NextClust = new CClustSet(m_CurClust);


	char *Checked = new char[nRows * nCols];

	long n,	CurClust;
	long MaxLoop = 20000;
	int LoopPos = 0;
	unsigned int TempClust = 10000;

	int *xLoop = NULL;
```

```
int *yLoop = NULL;
xLoop = new int[MaxLoop];
yLoop = new int[MaxLoop];


// Cluster
for (int l=0; l<iterations; l++)
{
      for (n=0; n<(long)nRows*nCols; n++)
            Checked[n] = FALSE;

      if (Outliers) {
            for (n=0; n<(long)m_nPoints; n++) {
                  unsigned int row = m_Member.GetPRow(n);
                  unsigned int col = m_Member.GetPCol(n);
                  long Member = m_Member.GetValue(Index(col,row));
                  if (Member && Member < m_CurClust->GetOutStart())
                        Checked[Index(col, row)] = TRUE;
                  else
                        n=n;
            }
      }

      int pos = 0;
      int PerSide = (l+1)*2+1;
      int Perim = 4*PerSide-4;

      if (LoopPos + Perim >= MaxLoop)
      {
            AfxMessageBox("ERROR: Loop exceed max size");
            return;
      }

      for (int i2=0; i2<PerSide; i2++)
      {
            xLoop[LoopPos + i2] = l+1;
            xLoop[LoopPos + i2 + Perim/2] = -(l+1);
      }

      for (i2=0; i2<PerSide-2; i2++)
```

```
            xLoop[LoopPos + i2+PerSide] = xLoop[LoopPos + Perim-i2-1] = l-i2;

        for (i2=0; i2<Perim; i2++)
            yLoop[LoopPos + i2] = xLoop[LoopPos + (i2+PerSide-1) % Perim];

        LoopPos += Perim;

        CClustSet *MoveClust = NextClust;
        NextClust = m_CurClust;
        m_CurClust = MoveClust;
        NextClust->ResetClusts();

        for (unsigned int i=0; i<nRows; i++)
            for (unsigned int j=0; j<nCols; j++)
            {
                if (!Checked[Index(j,i)])
                {
                    Checked[Index(j,i)] = TRUE;
                    if (m_Lines.GetValue(Index(j,i)))
                    {
                        CDWordArray CellList;
                        CWordArray WithinClusts;

                        if (m_Member.GetValue(Index(j,i)))
                            WithinClusts.Add(m_Member.GetValue(Index(j,i)));

#ifdef _DEBUG
        static int Break_l=-1;
        static int Break_i=-1;
        static int Break_j=-1;

        if (l==Break_l && i==Break_i && j==Break_j)
            l=l;
#endif

                        BOOL Mergable = m_CurClust->IsMergable(m_Member.GetValue(Index(j,i)),
                                                               m_LM_Count);
                        BOOL NoMerge = m_CurClust->IsNoMerge(m_Member.GetValue(Index(j,i)),
                                                             m_NM_Count);
```

```
CheckAdjacent(i, j, Checked, CellList, LoopPos, xLoop, yLoop, Mergable,
                    NoMerge, WithinClusts);

CurClust = 0;

CWordArray ClustList;
for (n=0; n<CellList.GetSize(); n++)
{
      int Cluster = m_Member.GetValue(CellList[n]);
      if (Cluster != -1)
      {
            for (int i=0; i<ClustList.GetSize(); i++)
                  if (Cluster == ClustList[i])
                        break;

            if (i == ClustList.GetSize())
                  ClustList.Add(Cluster);
      }
}

long MaxLines = 0;
for (n=0; n<ClustList.GetSize(); n++)
{
      long Lines = m_CurClust->GetClustLines(ClustList[n]);
      if (Lines > MaxLines)
      {
            MaxLines = Lines;
            CurClust = ClustList[n];
      }
}

long CellCount = CellList.GetSize();
long LineCount = 0;
for (n=0; n<CellList.GetSize(); n++)
      LineCount += m_Lines.GetValue(CellList[n]);

if (LineCount < m_MinLines)
      CurClust = 0;
else if (!CurClust || CurClust == -1)
      CurClust = ++m_LastCluster;
```

```
                                    if (CurClust)
                              NextClust->Add(CurClust, CellCount, LineCount);

                                    for (n=0; n<CellList.GetSize(); n++)
                                          m_Member.SetValue(CellList[n], CurClust);
                              }
                        }
                  }

#ifdef _DEBUG
{
      for (n=0; n<(long)m_nPoints; n++)
      {
            unsigned int row = m_Member.GetPRow(n);
            unsigned int col = m_Member.GetPCol(n);
            NextClust->SetPointClust(n, m_Member.GetValue(Index(col, row)));
      }

      CString Output;
      char buff[360];

      FILE* trace = fopen("TRACE.TXT", "a+");
      TRACE("%d Clusters at %ld feet\n", NextClust->m_nClusters, NextClust->m_Feet);
      fprintf(trace, "%d Clusters at %ld feet\n", NextClust->m_nClusters, NextClust->m_Feet);

      for (unsigned int i=0; i<NextClust->m_nClusters; i++) {
            sprintf(buff, "%02d)   %2d   %4ld   %4ld\n", i+1, NextClust->m_Cluster[i],
                        NextClust->m_CellCount[i], NextClust->m_NumLines[i]);
            TRACE(buff);
            fprintf(trace,"%s", buff);
      }

      unsigned int count=0;
      for (i=0; i<NextClust->m_nPoints; i++) {
            if (!NextClust->pCluster(i))
                  count++;
      }
      TRACE("Outliers = %d\n\n", count);
      fprintf(trace, "Outliers = %d\n", count);
```

```
        fclose(trace);
        TRACE("\n");
    }
    #endif
        }

        CClustSet *MoveClust = NextClust;
        NextClust = m_CurClust;
        m_CurClust = MoveClust;
        NextClust->ResetClusts();

    //=========================================
    //=========================================
    //=========================================
        for (unsigned int i=0; i<m_CurClust->GetSize(); i++)
        {
            long count=0;
            for (n=0; n<m_Member.GetSize(); n++)
            {
                if (m_Member.GetValue(n) == m_CurClust->GetCluster(i))
                {
                    m_Member.SetValue(n, 10000+i+1);
                    count++;
                }
            }

            if (!count)
                count = 0;

            m_CurClust->SetCluster(i, i+1);
        }

        for (n=0; n<m_Member.GetSize(); n++)
        {
            if (m_Member.GetValue(n))
                m_Member.SetValue(n, m_Member.GetValue(n)-10000);
        }
    //=========================================
    //=========================================
    //=========================================
```