CHAPTER 5

ARRAYS AND MATRICES

One-Dimensional Arrays

•When solving engineering problems the data consist of just a single number, and some other times we have hundreds of numbers that need to be identified individually

•So we need a method to work with a large group of values using a single identifier. The solution to this problem is to use a data structure called an **array**.

•One-dimensional arrays can be visualized as a list of values arranged in a row or column form:

5	0	-1
s[0]	s[1]	s[2]

0.0	t[0]
0.1	t[1]
0.2	t[2]

Definition and Initialization

- An array is defined using declaration statements

- The declaration statements for the previous arrays are:

```
int s[3];
double t[3];
```

- Arrays can be initialized with declaration statements or with program statements:

```
int s[3]={5,0,-1}                or     int s[]={5,0,-1}
double t[3]={0.0,0.1,0.2}               double t[]={0.0,0.1,0.2}
```

Definition and Initialization

•Arrays are often used to store information that is read form data files. For example suppose that we have a data file named that contains 10 time and motions measurements collected from a seismometer:

•To read these values we could use the following statements:

```
int k;
double time[10], motion[10];
        …
sensor = fopen(“sensor3.txt”,”r”);
for(k=0; k<=9; k++)
        fscanf(sensor,”%lf %lf”,&time[k],&motion[k]);
        …
```

Computation and Output

•To print 100 values of the array, one per line, we could use the following statements:

•#define N 100

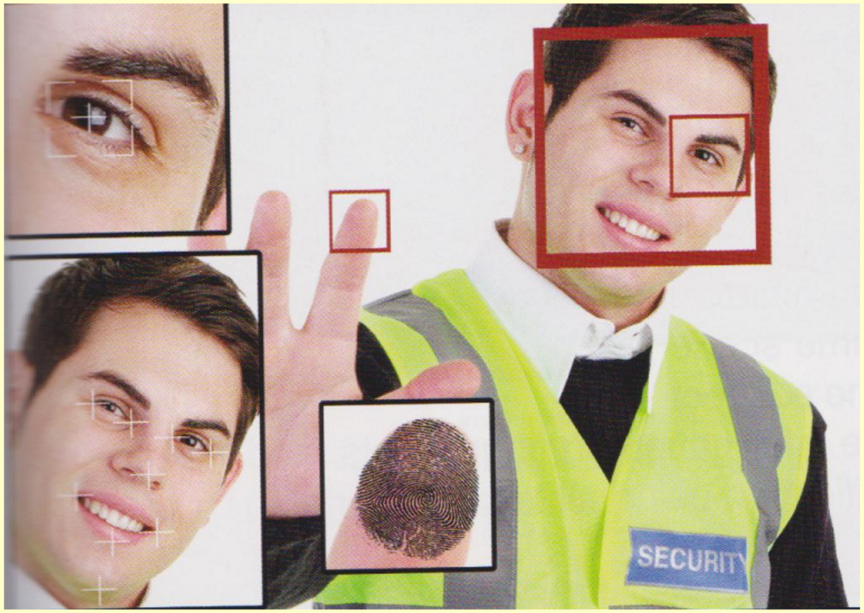
```
…

printf("y values: \n");
for(k=0; k<=N-1; k++)
        printf("%f \n",y[k]);
…
```

Crime Scene Investigation

•The goal of the text is to teach the student how to solve problems using the C language. To make the problems more interesting, the author is using a them of Crime Scene Investigacion, or CSI. Biometrics can be used to identify a person (either a victim, a suspect, or a witness).

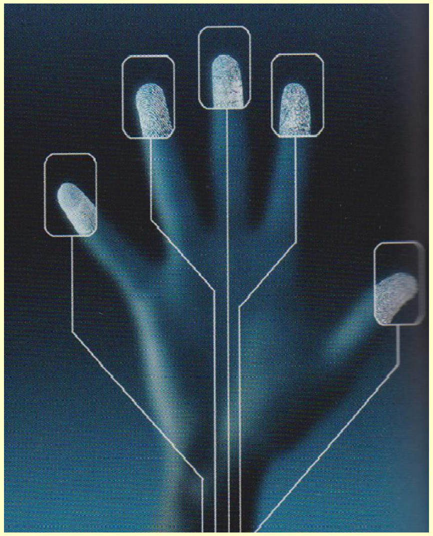
•Common biometrics are:

1. Fingerprint
2. Face
3. Iris
4. Speech



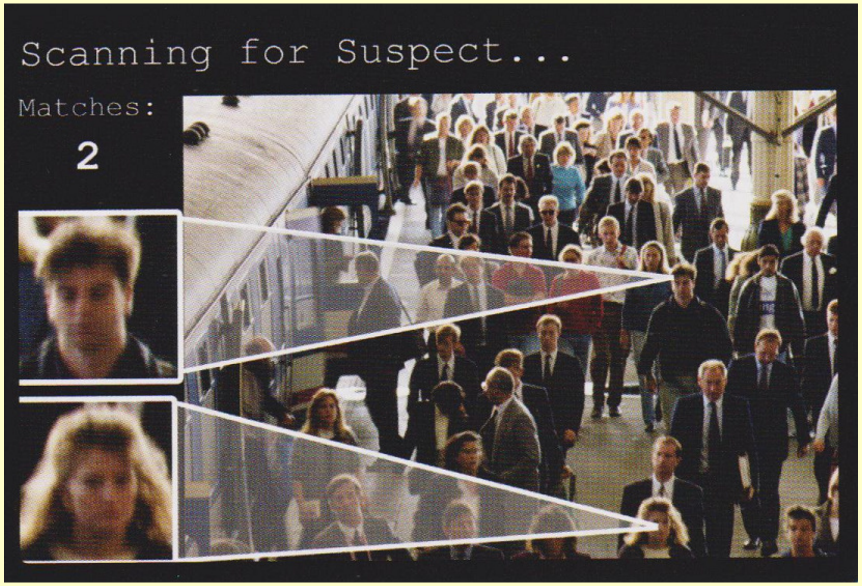
Crime Scene Investigation

1. Fingerprints
   Fingerprints are the oldest method of identification. A common method used by the FBI is based on the Henry classification system: identifying whorl, arch or loops.



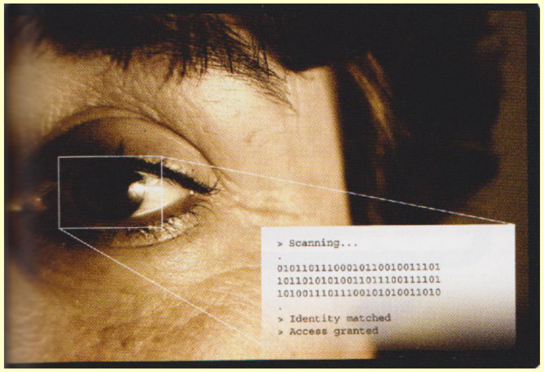
Crime Scene Investigation

2. Face
   Face recognition is another commonly used technique to identify an individual form an image or from a single frame (or image) taken from a surveillance video



Crime Scene Investigation

3. Iris
   Iris recognition is one the most accurate biometrics. Commercial iris recognition systems use an infrared camera to collect an image of the eye because an infrared image is not affected by color



Crime Scene Investigation

4. Speech
   Speech is also a biometric. Your speech is affected by your vocal cords, your mouth, your tongue, your teeth, your nasal cavity, and other parts of your anatomy. Therefore, it can also be used to identify a person.

Statistical Measurements

•Analyzing data collected from engineering experiments is an important part of evaluating the experiments. Many of the computations or measurements using data are statistical measurements.

Simple Analysis

When evaluating a set of experimental data, we often compute the following statistical values:

1. Maximum
2. Minimum
3. Mean or average
4. Median
5. Variance and standard deviation

Statistical Measurements

•Maximum and Minimum. Are the maximum and minimum values in an array

•Mean. Is an average value in an array:

$$\mu = \frac{1}{N} \cdot \sum_{i=1}^{N} x_i$$

•Median. The median is the value in the middle of a group of values, assuming that the values are sorted

•Variance and Standard Deviation. The variance is defined as the average squared deviation from the mean

$$\sigma^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu)^2$$

Problem Solving Applied: Speech Signal Analysis

•Suppose we are interested in analyzing speech signals for the words "zero", "one", "two", …,"nine." We need to develop ways of identifying the correct digit from a data file containing utterance of an unknown digit

•The analysis of this type of complicated signal starts with computing some statistical measurements. Other measurements used in speech recognition are:

•Average magnitude

$$Average\ magnitude = \frac{\sum_{k=1}^{N} |x_k|}{N}$$

Problem Solving Applied: Speech Signal Analysis

- Average power:

$$Average\ magnitude = \frac{\sum_{k=1}^{N} x^2}{N}$$

- Zero crossings: The number of zero crossings is the number of times that the speech signal makes a transition from a negative to a positive value or from a positive to a negative value.

Problem Solving Applied: Speech Signal Analysis

•Following figure contains a plot of an utterance of the digit "zero"

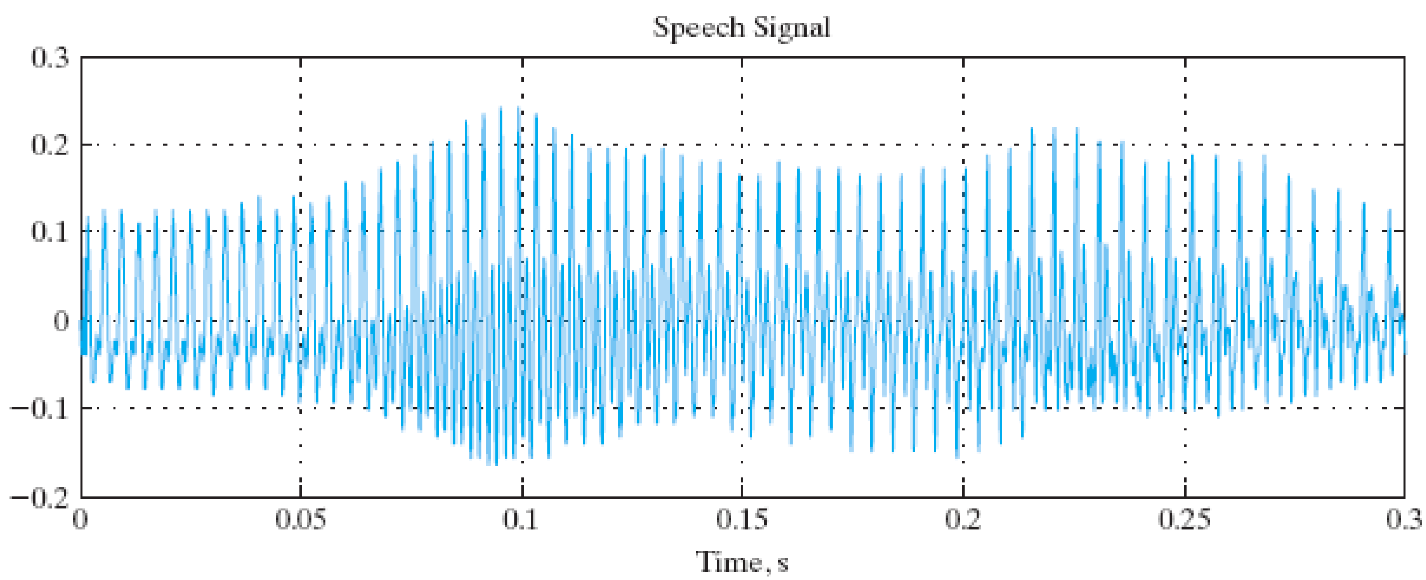


Figure 5.2 *Utterance of the word "zero."*

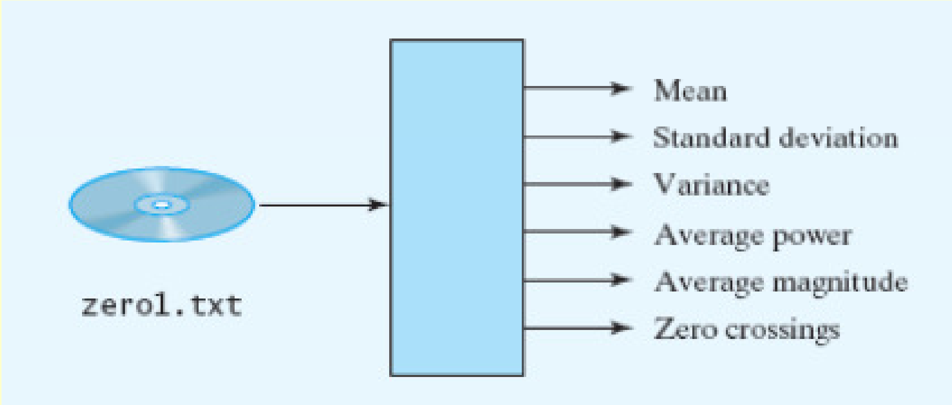
Problem Solving Applied: Speech Signal Analysis

•Average power

$$Average\ magnitude = \frac{\sum_{k=1}^{N} x^2}{N}$$

1. Problem Description
   Compute several statistical measurements for speech utterance.

2. Input/Output Description



Problem Solving Applied: Speech Signal Analysis

3. Hand Example
   Assume that the file contains the following values:
   2.5   8.2   -1.1   -0.2   1.5

   Using the equation described above, we can compute the following values:

   Mean = 2.18, Variance = 13.307, Standard deviation = 3.648
   Average power = 15.398, Average magnitude = 2.7
   Number of zero crossings = 2

Problem Solving Applied: Speech Signal Analysis

4. Algorithm Development

   Decomposition Outline

   1. Read the speech signal into an array
   2. Compute and print statistical measurements

```
#include <stdio.h>
#include <math.h>
#define MAX 2500
#define FILENAME "zero1.txt"

int main(void)
{
   /*  Declare and initialize variables.  */
   int k=0, N;
   double speech[MAX];
   FILE *filein;
   double max(double x[], int N);
   double min(double x[], int N);
   double mean(double x[], int N);
   double median(double x[], int N);
   double variance(double x[], int N);
   double std_dev(double x[], int N);
   double avg_power(double x[], int N);
   double avg_magn(double x[], int N);
   int    crossings(double x[], int N);

   /*  Read information from data file    */
   filein = fopen(FILENAME,"r");
   if (filein == NULL) printf("Error opening file. \n");
   else
   {
       while ((fscanf(filein,"%lf",&speech[k])) == 1) k++;
       N = k;

       /*  Compute and print statistics   */
       printf("Speech statistics \n");
       printf("  Mean     : %f\n", mean(speech,N));
       printf("  Std. dev.: %f\n", std_dev(speech,N));
       printf("  Variance : %f\n", variance(speech,N));
       printf("  Avg. pwr.: %f\n", avg_power(speech,N));
       printf("  Avg. mgn.: %f\n", avg_magn(speech,N));
       printf("  Crossings: %d\n", crossings(speech,N));

   }

   /*  Close data file and exit  */
   fclose(filein);
   getch();
   return 0;
}
```

```c
/*-----------------------------------------------------*/
/*  This function returns the maximum                  */
/*  value in the array x with n elements.              */

double max(double x[], int n)
{
   /*  Declare variables.  */
   int k;
   double max_x;

   /*  Determine maximum value in the array.  */
   max_x = x[0];
   for (k=1; k<=n-1; k++)
   {
      if (x[k] > max_x) max_x = x[k];
   }

   /*  Return maximum value.  */
   return max_x;
}
```

```c
/*-----------------------------------------------------*/
/*  This function returns the minimum                  */
/*  value in an array x with n elements.               */

double min(double x[], int n)
{
   /*  Declare variables.  */
   int k;
   double min_x;

   /*  Determine minimum value in the array.  */
   min_x = x[0];
   for (k=1; k<=n-1; k++)
   {
      if (x[k] < min_x) min_x = x[k];
   }

   /*  Return minimum value.  */
   return min_x;
}
```

```c
/*-----------------------------------------------------*/
/*  This function returns the average or               */
/*  mean value in an array with n elements.            */

double mean(double x[], int n)
{
   /*  Declare and initialize variables.  */
   int k;
   double sum=0;

   /*  Determine mean values.  */
   for (k=0; k<=n-1; k++)
   {
      sum += x[k];
   }

   /*  Return mean value.  */
   return sum/n;
}
```

```c
/*-----------------------------------------------------*/
/*  This function returns the median                   */
/*  value in an array x with n elements.               */

double median(double x[], int n)
{
   /*  Declare variables.  */
   int k;
   double median_x;

   /*  Determine median value.  */
   k = floor(n/2);
   if (n%2 != 0)
      median_x = x[k];
   else
      median_x = (x[k-1] + x[k])/2;

   /*  Return median value.  */
   return median_x;
}
```

```c
/*-----------------------------------------------------*/
/*  This function returns the variance                 */
/*  of an array with n elements.                       */

double variance(double x[], int n)
{
   /*  Declare variables and function prototypes.  */
   int k;
   double sum=0, mu;
   double mean(double x[], int n);

   /*  Determine variance.  */
   mu = mean(x,n);
   for (k=0; k<=n-1; k++)
   {
      sum += (x[k] - mu)*(x[k] - mu);
   }

   /*  Return variance.  */
   return sum/(n-1);
}
```

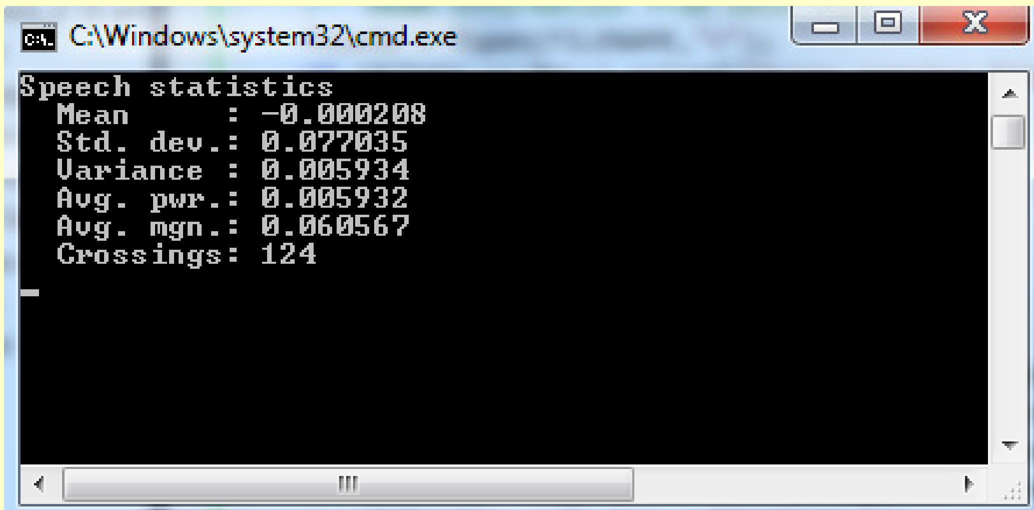
```c
/*-----------------------------------------------------*/
/*  This function returns the average magnitude        */
/*  of an array with n elements.                       */

double avg_magn(double x[], int n)
{
   /*  Declare variables and function prototypes.  */
   int k;
   double sum=0;

   /*  Determine ave magnitude  */
   for (k=0; k<=n-1; k++) sum += fabs(x[k]);

   /*  Return average magnitude.  */
   return sum/n;
}
```

```c
/*-----------------------------------------------------*/
/*  This function returns the standard deviation       */
/*  of an array with n elements.                       */

double std_dev(double x[], int n)
{
   /*  Declare function prototypes.  */
   double variance(double x[], int n);

   /*  Return standard deviation.  */
   return sqrt(variance(x,n));
}

/*-----------------------------------------------------*/
/*  This function returns the average power            */
/*  of an array with n elements.                       */

double avg_power(double x[], int n)
{
   /*  Declare variables and function prototypes.  */
   int k;
   double sum=0;

   /*  Determine ave power  */
   for (k=0; k<=n-1; k++) sum += x[k]*x[k];

   /*  Return average power.  */
   return sum/n;
}
/*-----------------------------------------------------*/
```

```c
/*-------------------------------------------------*/
/*  This function returns the crossings            */
/*  of an array with n elements.                   */

int crossings(double x[], int n)
{
   /*  Declare variables and function prototypes.  */
   int k, cnt=0;

   /*  Determine the zero crossings   */
   for (k=0; k<=n-2; k++)
      if (x[k]*x[k+1] < 0) cnt++;

   /*  Return zero crossings   */
   return cnt;
}
/*-------------------------------------------------*/
```

Problem Solving Applied: Speech Signal Analysis

5. Testing
   The following values were computed for the utterance “zero” using the file zero1.txt



Two-Dimensional Arrays

•A set of data values that is visualized as a row or column is easily represented by a one-dimensional array. An array with four rows and three column (let us call it x) is shown in the following diagram

Row 0	**2**	**3**	**-1**
…	0	-3	5
	2	6	3
Row 3	-2	10	4
	Column 0	…	Column 2

Definition and Initialization

- To define a two-dimensional array, we specify the number of rows and columns in the declaration statement. The row number is written first. Both the row and column number are in brackets:

```
int x[4][3];
```

- A two-dimensional array can be initialized with a declaration statement:

```
int x[4][3]={{2,3,-1},{0,-3,5},{2,6,3},{-2,10,4}};

int x[][3]={{2,3,-1},{0,-3,5},{2,6,3},{-2,10,4}};
```

Definition and Initialization

•Arrays can also be initialized with program statements (using nested loops)

•For example to initialize an array such that each row contains the row number, use the following statements:

```
/*  Declare variables  */
int i, j, t[5][4];
     …

/*  Initialize array  */
for (i=0; i<=4; i++)
          for (j = 0; j<=3; j++)     t[i][j] = i;
```

Function Arguments

•When using a two-dimensional array as a function argument, the function also needs information about the size of the array. Suppose we need to write a program that computes the sum of the elements of an array. We would need to use the following statements:

```
/*  Declare variables  */
int a[4][4];
int sum(int x[4][4]);
      …

/*  Use function to compute the array sum.  */
Printf("Array sum = %i \n",sum(a));
```

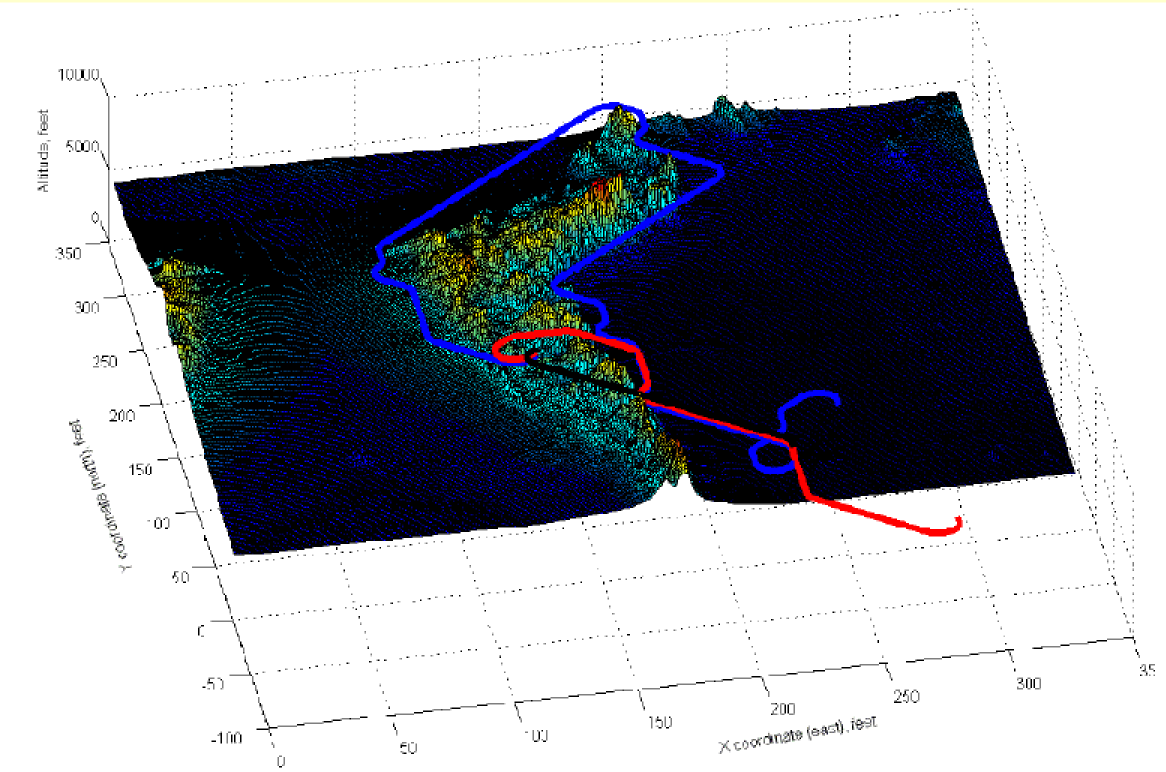
Problem Solving Applied: Terrain Navigation

•Terrain navigation is a key component in the design of unmanned aerial vehicles (UAVs). Vehicles such as a robot or a car, can travel on land; and a drone or plane, can fly above the land.

•A UAV contains an onboard computer that has stored terrain information for the area in which is to be operated. The computer has elevation information that allows for a safe navigation of the UAV

Problem Solving Applied: Terrain Navigation

Real-time path planning



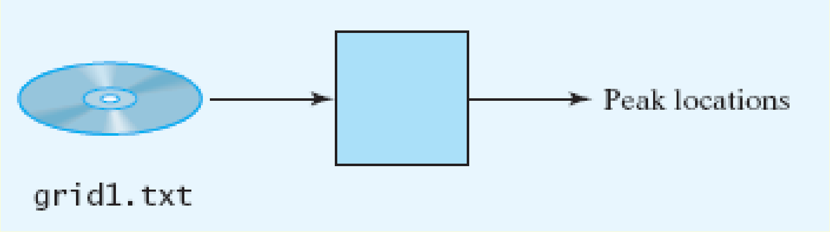
MQM-107 UAV

Problem Solving Applied: Terrain Navigation

1. Problem Statement
   For this problem we want to determine whether the value in grid position [m][n] is a peak.

   Determine and print the number of peaks and their locations in an elevation grid

2. Input/Output Description



Problem Solving Applied: Terrain Navigation

3. Hand Example
   Assume that the following data represent the elevation for a grid that has six points along the side and seven points along the top (the peaks are underlined)

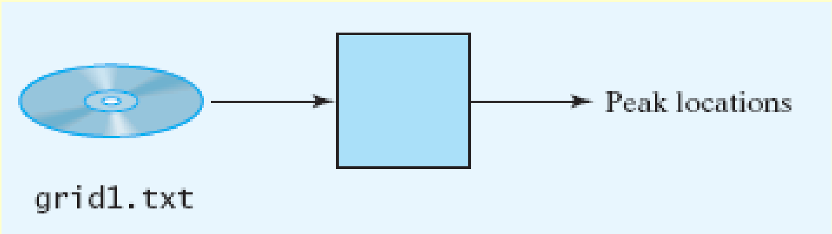
5039	5127	5238	5259	5248	5310	5299
5150	5392	5410	5401	5320	5820	5321
5290	<u>5560</u>	5490	5421	5530	<u>5831</u>	5210
5110	5429	5430	5411	5459	5630	5319
4920	5129	4921	<u>5821</u>	4722	4921	5129
5023	5129	4822	4872	4794	4862	4245

Problem Solving Applied: Terrain Navigation

1. Problem Statement
   For this problem we want to determine whether the value in grid position [m][n] is a peak.

   Determine and print the number of peaks and their locations in an elevation grid

2. Input/Output Description



Problem Solving Applied: Terrain Navigation

4. Algorithm Development

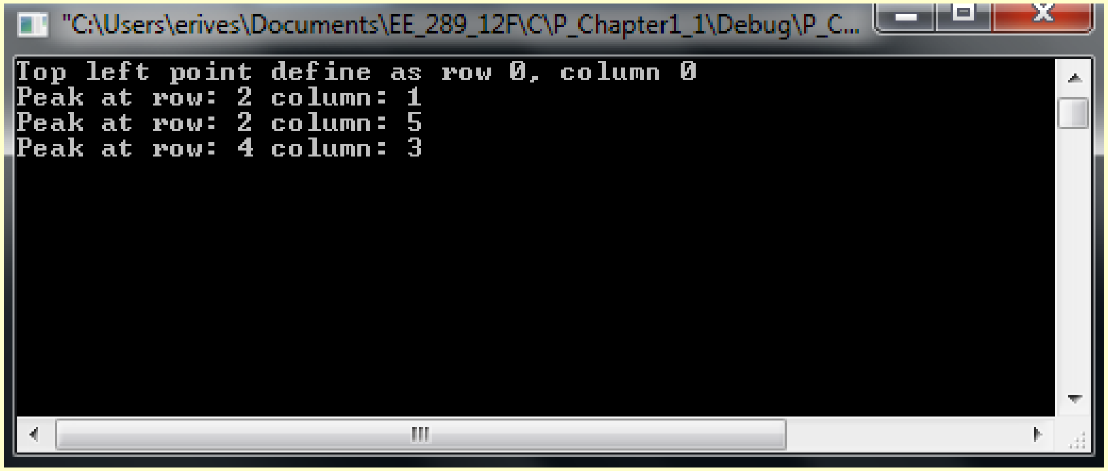
   Decomposition Outline

   1. Dread the terrain data into an array
   2. Determine and print the location of the peaks

```c
#include <stdio.h>
#define N 25
#define FILENAME "grid1.txt"

int main(void)
{
   /*  Declare and initialize variables.  */
   int nr, nc, i, j;
   double elevation[N][N];
   FILE *grid;

   /*  Read information from data file  */
   grid = fopen(FILENAME,"r");
   if (grid == NULL) printf("Error opening the file \n");
   else
   {
       fscanf(grid,"%d %d",&nr,&nc);
       for(i=0; i<=nr-1; i++)
           for(j=0; j<=nc-1; j++)
               fscanf(grid,"%lf",&elevation[i][j]);

       /*  Determine and print peak locations. */
       printf("Top left point define as row 0, column 0 \n");
       for (i=1; i<=nr-2; i++)
           for (j=1; j<=nc-2; j++)
               if((elevation[i-1][j]<elevation[i][j]) &&
                  (elevation[i+1][j]<elevation[i][j]) &&
                  (elevation[i][j-1]<elevation[i][j]) &&
                  (elevation[i][j+1]<elevation[i][j]))
                printf("Peak at row: %d column: %d \n",i,j);

       /*  Close file  */
       fclose(grid);
   }

   /*  Exit program   */
   getch();
   return 0;
}
```

Problem Solving Applied: Terrain Navigation

5. Testing
   The following output was printed using a data file that corresponds to the hand example:



Homework on Chapter 5 is posted on the website:

http://www.ee.nmt.edu/~erives/289_F12/EE289.html

Homework is due within a week