TWO DIMENSIONAL ARRAY OR MATRIX common name scores
Defined as scores[10][5] IE 10 rows x 5 columns
showing name of each position; some values below

| | col. 0 | col. 1 | col. 2 | col. 3 | col. 4 |
|---|---|---|---|---|---|
| row 0 | scores[0][0] | scores[0][1] | scores[0][2] | scores[0][3] | scores[0][4] |
| row 1 | scores[1][0] | scores[1][1] | scores[1][2] | scores[1][3] | scores[1][4] |
| row 2 | scores[2][0] | scores[2][1] | scores[2][2] | scores[2][3] | scores[2][4] |
| row 3 | scores[3][0] | scores[3][1] | scores[3][2] | scores[3][3] | scores[3][4] |
| row 4 | scores[4][0] | scores[4][1] | scores[4][2] | scores[4][3] | scores[4][4] |
| row 5 | scores[5][0] | scores[5][1] | scores[5][2] | scores[5][3] | scores[5][4] |
| row 6 | scores[6][0] | scores[6][1] | scores[6][2] | scores[6][3] | scores[6][4] |
| row 7 | scores[7][0] | scores[7][1] | scores[7][2] | scores[7][3] | scores[7][4] |
| row 8 | scores[8][0] | scores[8][1] | scores[8][2] | scores[8][3] | scores[8][4] |
| row 9 | scores[9][0] | scores[9][1] | scores[9][2] | scores[9][3] | scores[9][4] |

Figure 9.1: Rows and Columns in A Two Dimensional Array

Values in the memory could be scores[1][2]=78 scores[0][2]=56 scores[7][0] = 91 Etc.

**HOW MANY MEMBERS IN THIS 2D ARRAY???**

# Outline

Objectives READ STDY FOLLOWING SECTIONS ONLY

1. Two Dimensional Arrays

4. Matrices as 2D arrays ( The mathematics of Matrices)

5. Numerical technique: Solution to Simultaneous Equations

6. Problem Solving Applied: Electrical-Circuit Analysis

**(Kirchoffs rules for two voltage sources an application Of a solution to simultaneous equations)**

**Be sure to run youtube video(s) in the online syllabus**

# Objectives

Develop problem solutions in C++
containing:

- Matrix computations
- Input from Data Files
- Functions to Compute Sums and Averages
- Techniques for solving a system of simultaneous equations

# Two Dimensional Arrays

- Declaration and Initialization
- Computation and Output
- Function Arguments

# Two Dimensional Arrays

- A two dimensional array stores data as a logical collection of **rows** and **columns**.
- Each element of a two-dimensional array has a row position and a column position.
- To access an element in a two-dimensional array, you must specify the name of the array followed by:
  - a **row offset**
  - a **column offse**t

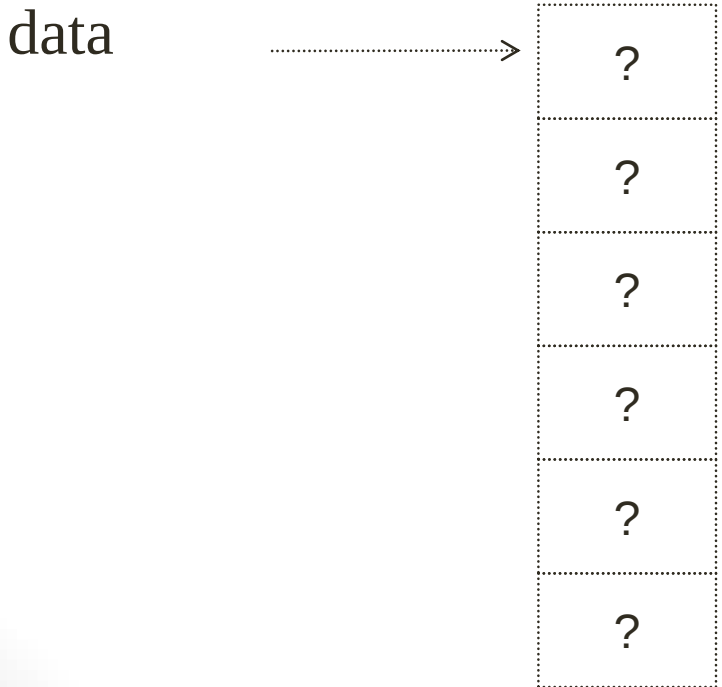
# Declaration and Initialization

- The declaration of a two-dimensional array requires a **row size** and a **column size**.
- A consecutive block of (**row size**)*(**column size**) memory locations are allocated.
- All array elements must be of the same type.
- Elements accessed by two offsets – a row offset and a column offset.
- The name of the array holds the address of the first byte of memory

# Example

```
//Declaration
int data[2][3];   2 X 3  = 6 VALUES
```

**Memory Snapshot 6 VALUES ARE SEQUENCIAL IN MEMORY**



# Example

```
//Declaration  VALUES IN MATRIX
FORM IS THE PROGRAMMERS CONCERNS
int data[2][3];
```

**row/column form:**

| | col 0 | col 1 | col 2 |
|---|---|---|---|
| row 0 | ? | ? | ? |
| row 1 | ? | 98 | ? |

THESE ROW AND COLUMN NUMBERS ARE KNOWN AS THE OFFSETS AND ARE USED TO GET THE VALUE IN MEMORY OF THE ARRAY at their position.
FOR EXAMPLE  data[1][1]=98 as illustrated

# 2D Array Definition Syntax

**Syntax:**

data_type identifier**[** [row_size] **][**column_size**]** [= initialization_list ];

//row_size and column_size must be integer constants

Examples

```
int data[2][5]; //allocates consecutive memory for 10 integer values
double t[2][2] = {{3.0,5.0},{2.1,7.2}};
//allocates and initializes 4 VALUES     T[0][0]=3.0   T[1][1]=7.2 ETC
```

Valid References

```
cout << data[1][3];
cout << t[1][0];
```

Invalid References

```
cout << data[2][5]; //invalid offset NO ROW 2 OR COLUMN 5   0 TO 1 FOR ROWS AND 0 TO 4 ARE THE OFFSETS ONLY.
cout << t[-1][-1]; //invalid offset NO SUCH THING AS NEGATIVE OFFSETS
```

# Initialization Examples

```
 int temp[4][3] = {50, 70, 60, 48, 75,
   62, 51, 69, 60, 52, 78, 63};
int temp[4][3] = {{50, 70, 60}, {48, 75, 62},
   {51, 69, 60}, {52, 78, 63}};
//  last is the same as above
```

Only the row size can be left blank and the System will use the appropriate number= 4 here!

```
int temp[][3] = {{50, 70, 60}, {48, 75, 62},
                 {51, 69, 60}, {52, 78, 63}};
int temp[][3] = {50, 70, 60, 48, 75, 62, 51,
   69, 60, 52, 78, 63};
```

NOTE: ONLY ROW SIZE DIMENSION CAN BE EMPTY
USING AN EMPTY COLUMN SIZE GETS A COMPILATION ERROR

# Initialization Examples

- `int t2[7][4] = {{50, 70, 60}, {48, 75, 62},`
  `{51, 69, 60}, {52, 78, 63}};`
- IF the Number of Values assigned are shorter than the size of the Array all other values are assigned 0!
- Values of 2D array t2 are thus =

| 50 | 70 | 60 | 0 |
|---|---|---|---|
| 48 | 75 | 62 | 0 |
| 51 | 69 | 60 | 0 |
| 52 | 78 | 63 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

# Example: Input Using cin

- **Nested `for` loops** are often used when inputting and assigning values to a two-dimensional array.
  - Nested loops are generally useful for getting around the 2D arrays…

```
//Declaration
double table[RSIZE][CSIZE];

for (int i=0; i<RSIZE; ++i) //every row
  for (int j=0; j<CSIZE; ++j )//every col
    cin >> table[i][j];
```

# Example: Assignment of values using nested “for” loops for 2D!

```
//Declaration
const int RSIZE(3),CSIZE(2);
double v[RSIZE][CSIZE];

for (int i=0; i<RSIZE; ++i) //every row
  for (int j=0; j<CSIZE; ++j )//every col
    v[i][j] = i+j;
```

v

| | |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 3 |

# Example: Computations

- Compute the average value of an array with n rows and m columns.

```
double sum(0), average;
for (int i=0; i<n; ++i)//every row
  for (int j=0; j<m; ++j )//every col
    sum += array[i][j];
average = sum / (n*m);
```

# Example: Computations

- Compute the average value of the nth row of a 2D array with r rows and c columns.

```
double sum(0), rowAverage;
for (int j=0; j<c; ++j ) //every col
    sum += array[n][j];
average = sum / c;
```

# Modify!

- How would you Modify the C++ statements on the previous slide to compute the average of the mth column.

# Outputting 2D Arrays

- Two dimensional arrays are often printed  in a row by row format, using nested `for` statements.
- When printing the row values of an array, be sure to print:
  - whitespace between the values in a row.
  - a newline character at the end of each row.

# Example: output of the array values row by row

```
for (int i=0; i<n; ++i) {//every row
  for (int j=0; j<m; ++j )//every col
    cout << array[i][j] << ' ';
  cout << endl; //add end-of-line each row
}
```

# Class exercise 2D array with for loops!

VERY IMPORTANT FOR ARRAY (MATRICES)
HOW MANY TIMES DOES EACH LOOP EXECUTE & output =?

```
for (int k=0; k<3; ++k)
{        for (int j=0; j<2; ++j)
           {
          ++ count
           cout <<k<<“  “<<j<<count<<endl;
           }
}
```

Double sum2(0);

```
int ving[2] [3] = {2, -8, 9, 3, 6, 4};      //   PLAY COMPUTER NOW!  k, j, ving sum2
for (int k = 0; k < 2; ++k)
  {
    for (int j = 0; j < 3; ++j)
      {
         cout << "    "<<  ving[k][j];
         sum2 = sum2 + ving[k][j];
      }
    cout << endl;
```

# 2D Arrays as Function Parameters

- 2-D arrays are always passed by reference.
- The **column dimension must be specified**. The leftmost dimension (row) may be empty [].
- **Function prototype** example:
  - int rowAverage(**int Arr[][COLSIZE],** int whichRow);
- Array declaration in main:
  - **int table [ROWSIZE][COLSIZE];**
- Function invocation(call!) example:

  avg = rowAverage(**table**, 3);

# Documentation Style

- Well-documented functions always include pre-conditions and post-conditions in the function's comment header block.
  - Pre-conditions describe the conditions assumed to be true at the time the function is called.
    - If pre-conditions are not met, there is no guarantee that the function will work correctly.
  - Post-conditions describe the changes that will be made to the arguments during the execution of the function.

# Function example: Given array int a{NROWS][NCOLS] CALL TO sum use cout << sum(a)<<endl;

- /* This function returns the sum of the values in */
- /* an array with NROWS rows and NCOLS columns. */
- // PreCondition: Array X has NROWS and NCOLS.
- // PostCondition: Sum of integer Values is returned.
- int sum(int x[][NCOLS])  // rows do not have to be specified
- {
- int total(0); // Declare and initialize local objects.
- for (int i=0; i<NROWS; ++i) // Compute a sum of the array values.
- {
- for (int j=0; j<NCOLS; ++j)
- { total += x[i][j];
- }
- }
- return total; // Return sum of array values.
- }

# HAND IN HW # 16 3 each 24 pts

- Do the following Exam practice problems in chapter 8
- 1 to 4 and 7 to 10

# Matrices

- A matrix is a set of numbers arranged in a rectangular grid with rows and columns.
- A square matrix has the same number of rows as columns.
- Row and Column offsets in Matrices are 1-based.
- 2D Arrays are useful for representing matrices.
  - Remember that C++ uses 0-based offsets!
  - Other languages like MATLAB use 1 as the base

- Example 0f 1-based and 0-based members : consider matrix A =( $\begin{matrix} 3 & 4 \\ 7 & 1 \end{matrix}$ )
- in math the value 3 is member a(1,1 ) and value 7 is a(2,1)

# Matrices as 2 D arrays! Study section 8.4 on matrices

# Some matrix magic!

How do you solve the following ( The short way is determinants as follows) ?
2x -8y =9
3x +6y =4
x=?  y=?

DETERMINANT! Using rows and columns of numbers we create matrices and evaluate them
By the determinate rules. With answers only to 3 decimals here (could be better

$$x = \frac{\begin{vmatrix} 9 & -8 \\ 4 & 6 \end{vmatrix}}{\begin{vmatrix} 2 & -8 \\ 3 & 6 \end{vmatrix}} = \frac{54 - -32}{12 - -24} = \frac{86}{36} = 2.3888\ldots \qquad y = \frac{8-27}{36} = \frac{-19}{36} = -0.52777\ldots$$

check first equation 2 *2.3888 – 8* (-0.5278) = 9.0004 (depends on how many decimals you keep in the intermediate calculations =9 if you keep all decimals
But the number depends on accuracy of coefficients to begin with

How do you solve the following 2 simultaneous equations
Here the Gauss Elimination technique is shown.

1. Elimination step use multiplies to eliminate variables
2. Back substitute to get answers
3. We use functions to be presented in an example to follow that do the above two steps once we put the coefficients into a 2 dimensional array for this problem and for 3 simultaneous equations to follow.

2x -8y =9 a.
3x +6y =4 b.
x=? y=?

| | | |
|---|---|---|
| | 1. | 2x – 8y =9 |
| mult b. | by 2/3 | 2x +4y = 8/3 |
| SubTract | | -12y = 9-2 2/3 = 6.333333…… |
| | | y= - 6.333…/12 = - .5277777…=- as before |

Back substitute y into equation 1

2x = 8(-.5275) +9 = 4.777777….
x = 4.77777/2 = 2.388888…. As before!

Consider again
2x -8y =9
3x +6y =4
x=?  y=?
We consider in C++ an entity that keeps track of values in rows and columns
an **array**!
So we can create a 2 Dimensional array from the above equations as

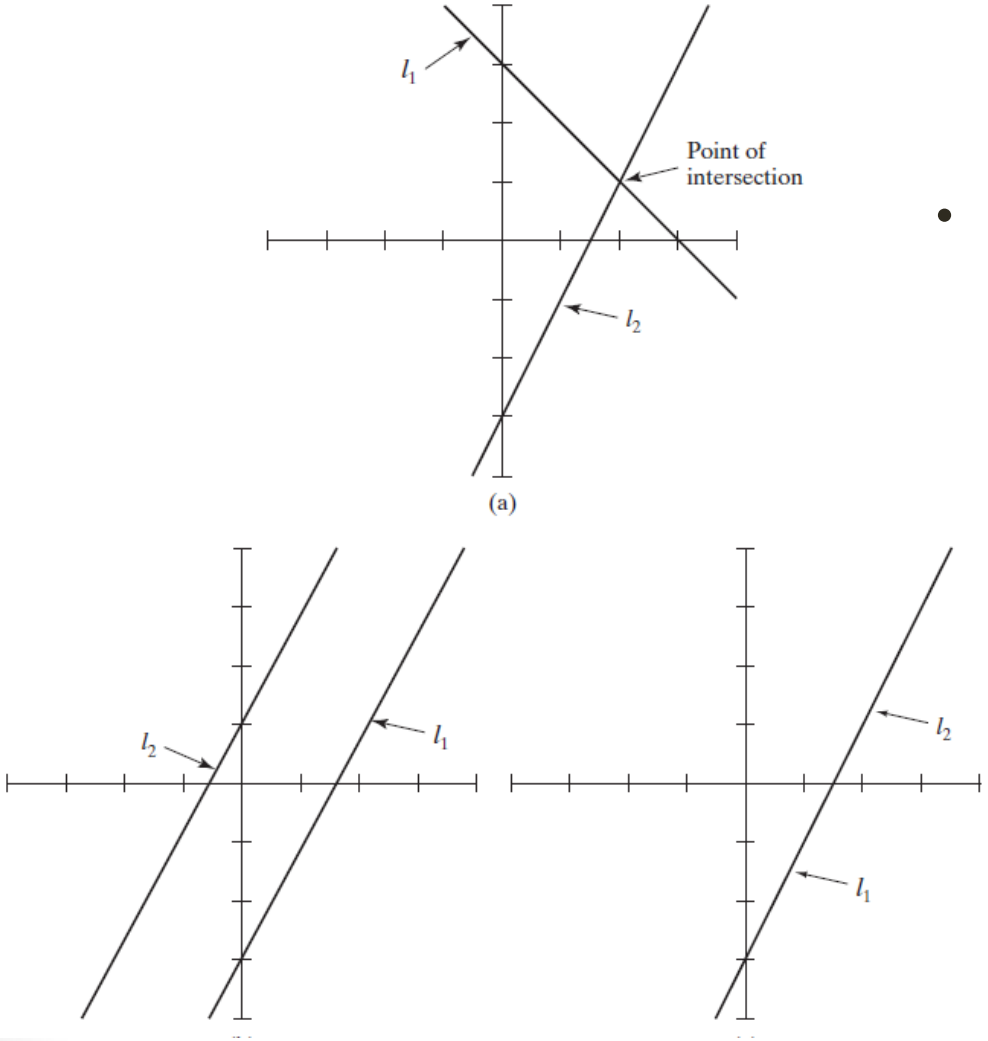
2  -8  9  or two rows and three columns each position can be referred
3  6  4  to the by what row number and what column number the
value is in computers the first row is the $0^{th}$ row ,
as is the first columns the $0^{th}$ column.
(MATLAB uses 1 to define the first row And column).
So the member at
0, 2 = 9 in this case,
1,2 =4  so  0,1 =?  1,0 =?

We define an array in C++ with the declaration giving its dimension
Initializing the values like below, or from input from the keypad or a file!
for the values above  2 rows and 3 columns we can use a row by row
Initialization. So in dealing with 2 D matrices 2D arrays are perfect!
int ving[2] [3] = {2, -8, 9, 3, 6, 4};  or int ving[2] [3]  ={ (2, -8, 9), (3, 6, 4) };

```cpp
// another DEMO OF ARRAY AND MATRIX
#include<iostream> //Required for cout
using namespace std;
int main()
{
  // Declare and initialize objects.
  double s[6] = { 1, 4, 7, 4.9, 5.1, 20.45 };
  double sum1(0), sum2(0);
  int ving[2][3] = { 2,-8,9,3,6,4 };
  // or int ving[2][3] = { {2, -8, 9}, {3, 6, 4} };
  for (int i = 0; i < 6; ++i)
    sum1 = sum1 + s[i];
  cout << endl;
  cout << "sum1=  " << sum1 << endl;
  for (int k = 0; k < 2; ++k)
  {
    for (int j = 0; j < 3; ++j)
      {
         cout << "   "<< ving[k][j];
         sum2 = sum2 + ving[k][j];
      }
    cout << endl;
  }
  cout << "sum1 = " << sum1 << "    sum2= " << sum2 << endl;
  return(0); // Exit program
}
```

# 8.5 Numerical Technique: Solution to Simultaneous Equations

# Graphical Interpretation



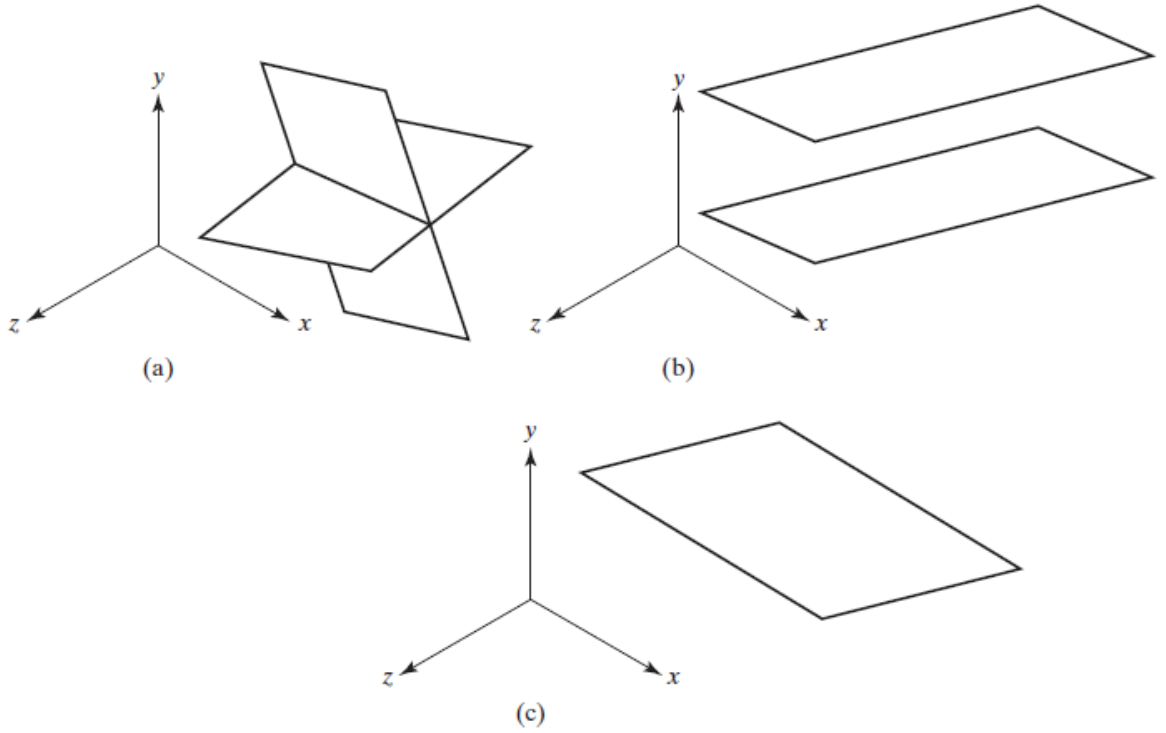

- Solving two simultaneous (linear) equations is finding the point at which the lines intersect.
  - Not all lines intersect, therefore there is not always a solution.
    - In 2D, parallel lines do not intersect unless they are identical lines.
  - When the lines are identical, there is no single point at which they intersect.

Graphical Interpretation (2 equations

$ax + by + cz = d$

$ex + fy + gz = h$

each defines plane whose solutions look like) First image 9 (a) is a line second (b) are no solutions of x,y,z that are simultaneous on both equations ie intersecting planes. Last both equations are really the same plane



$3x + 2y - z = 10$

$-x + 3y + 2z = 5$

$x - y - x = -1$

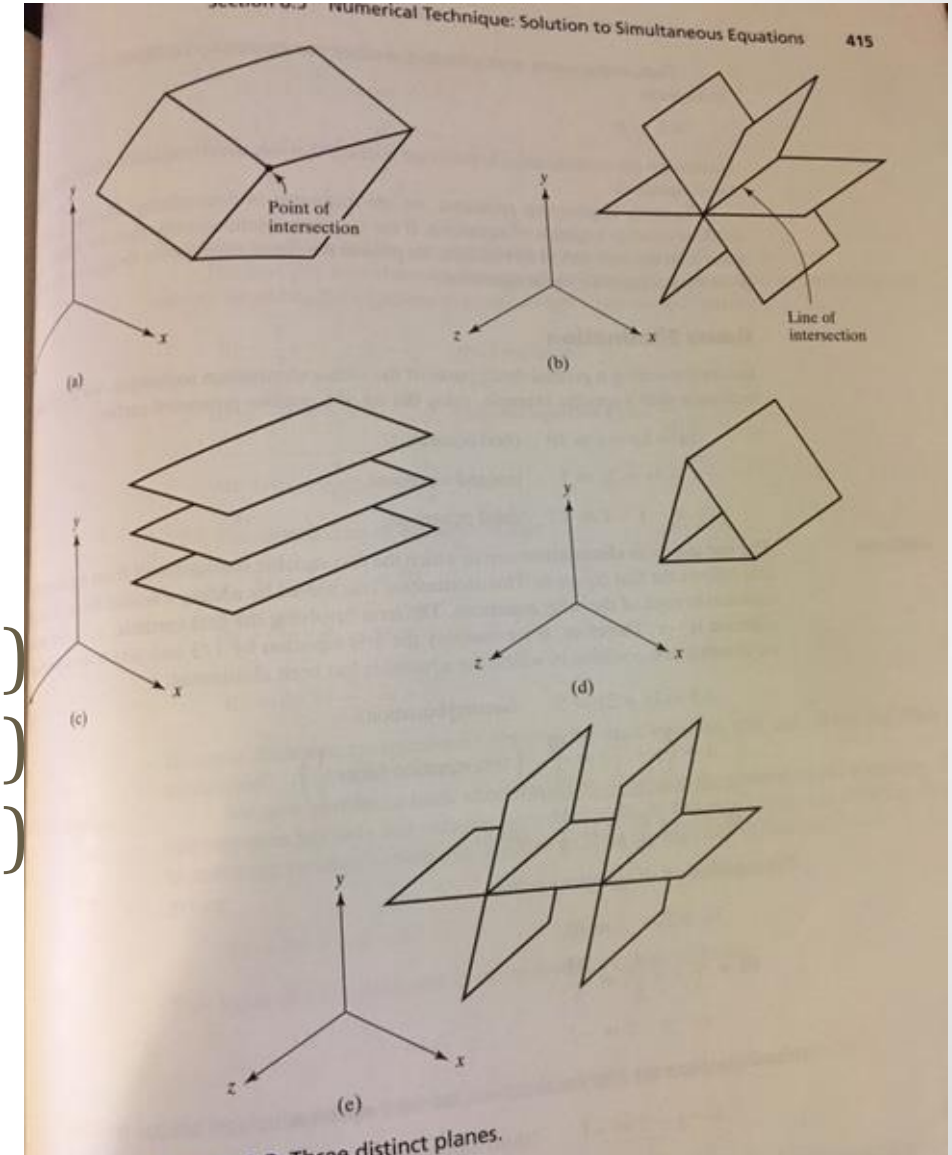
other possibilities are not a unique set of x,y,z. but a line fig (b) no solution fig (c) three lines fig (d) or two lines fig (e)

2 Dim array form to solution to follow

$a(0,0)x+a(0,1)y+a(0,2)z=a(0,3)$

$a(1,0)x+a(1,1)y+a(1,2)z=a(1,3)$

$a(2,0)x+a(2,1)y+a(2,2)z=a(2,3)$



presented works with many situation all you need to get x,y,z is the values of the two dimensional array (matrix).

$a(0,0)x+a(0,1)y+a(0,2)z=a(0,3)$

$a(1,0)x+a(1,1)y+a(1,2)z=a(1,3)$

$a(2,0)x+a(2,1)y+a(2,2)z=a(2,3)$

# Problem Solving Applied: Electrical-Circuit Analysis using Kirchoff laws we get 3 simultaneous equations. We know Vs and Rs and have to solve for 3 currents i1, i2 and i3 whichare the unkowns x,y,z in previous slide. Original physics equations are transformed to the a(i,j) array form as follows.

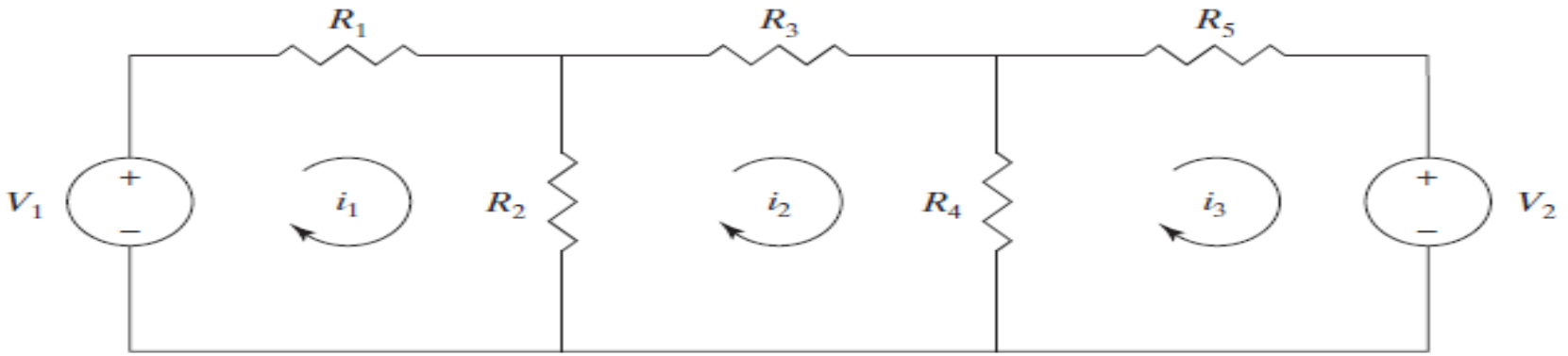



**Figure 8.4** Circuit with two voltage sources.

$$-V_1 + R_1 i_1 + R_2(i_1 - i_2) = 0,$$
$$R_2(i_2 - i_1) + R_3 i_2 + R_4(i_2 - i_3) = 0,$$
$$R_4(i_3 - i_2) + R_5 i_3 + V_2 = 0.$$

(R1 +R2)i1 – R2i2 +0i3 =V1 a(0,0) =R1+R2 a(0,1)= - R2 a(0,2)=0 a(0,3)=V1
-R2i1 +(R2+R3+R4)i2 –R4i3=0 a(1,0) = -R2 a(1,1)=R2+R3+R4 a(1,2)=-R4 a(1,3)=0
0i1-R4i2 +(R4+R5)i3 = -V2 a(2,0) = 0 a(2,1)= -R4 a(2,2)=(R4+R5) a(2,3)= -V2

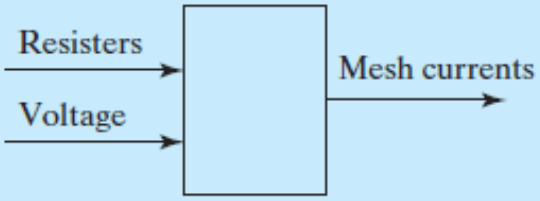
# Problem Solving Applied: Electrical-Circuit Analysis

**1. PROBLEM STATEMENT**

Compute the three mesh currents in the circuit shown in Figure 8.4.

**2. INPUT/OUTPUT DESCRIPTION**

The I/O diagram shows that the resistor values and the voltage values are the input values. The three mesh currents are the output values.



# Problem Solving Applied: Electrical-Circuit Analysis

**3. HAND EXAMPLE**

By using the resistor values and the voltage values, a system of three equations can be defined, using this rearranged set of equations from the definition of the problem:

$$(R_1 + R_2)i_1 - R_2 i_2 + 0i_3 = V_1,$$
$$-R_2 i_1 + (R_2 + R_3 + R_4)i_2 - R_4 i_3 = 0,$$
$$0i_1 - R_4 i_2 + (R_4 + R_5)i_3 = -V_2.$$

For example, suppose that each of the resistor values is 1 ohm and that both of the voltage sources are 5 volts. Then the corresponding set of equations is the following:

$$2i_1 - i_2 + 0i_3 = 5,$$
$$-i_1 + 3i_2 - i_3 = 0,$$
$$0i_1 - i_2 + 2i_3 = -5.$$

Once the system of equations is determined, the solution follows the steps illustrated in the hand example in the previous section. For this set of equations, the solution is $i_1 = 2.5$, $i_2 = 0$, and $i_3 = -2.5$.

# Problem Solving Applied: Electrical-Circuit Analysis

**4. ALGORITHM DEVELOPMENT**

We first develop the decomposition outline, because it breaks the solution into a series of sequential steps:

*Decomposition Outline*

1. Read the resistor values and the voltage values.
2. Specify the coefficients for the system of equations.
3. Perform Gauss elimination to determine currents.
4. Print currents.

In step 1 we read the information necessary to specify the circuit values, and in step 2 we use this information to specify the coefficients for the system of equations. Then, in step 3, we develop the details of the elimination and back-substitution steps. To keep the main function short and readable. functions are used for both the elimination and the back substitution.

```
/* Program chapter8_6                          */
/* This program uses Gauss elimination to  determine the mesh currents for
a circuit. */
#include <iostream> //Required for cin, cout
using namespace std;
const int N = 3; // Define global constant for number of unknowns.
void eliminate(double a[][N+1], int n, int index);
// Declare function prototypes.
void back_substitute(double a[][N+1],
int n, double soln[N]);
int main()
{
   double r1, r2, r3, r4, r5, v1, v2, a[N][N+1], soln[N];   // Declare objects.
"// Get user input. Specific for   this problem
 cout << "Enter resistor values in ohms: \n     << "(R1, R2, R3, R4, R5) \n";
  cin >> r1 >> r2 >> r3 >> r4 >> r5;
  cout << "Enter voltage values in volts: \n«   << "(V1, V2) \n";
  cin >> v1 >> v2;
```

```
// Specify equation coefficients.
 a[0][0] = r1 + r2;
a[0][1] = a[1][0] = -r2;
a[0][2] = a[2][0] = a[1][3] = 0;
 a[1][1] = r2 + r3 + r4;
a[1][2] = a[2][1] = -r4;
a[2][2] = r4 + r5;
a[0][3] = v1;
a[2][3] = -v2;

// Perform elimination step.
 for (int index=0; index<N-1; index++)
 {
  eliminate(a,N,index);
 }
```

```
// Perform back substitution step.  Get answers
  back_substitute(a,N,soln);
  // Print solution.
  cout << "\nSolution: \n";
  for (int i=0; i<N; ++i)
  {
    cout << "Mesh Current " << i+1 << ": "<< soln[i] <<
endl;
  }
  // Exit program.
  return 0;
}
/*-----------------------------------------------*/
```

```
/* This function performs the elimination step. */
void eliminate(double a[][N+1], int n, int index)
{  double scale_factor; // Declare objects.
  // Eliminate object from equations.
  for (int row=index+1; row<n; ++row)
  {
    scale_factor = -a[row][index]/a[index][index];
    a[row][index] = 0;
    for (int col=index+1; col<=n; ++col)
    {
      a[row][col] += a[index][col]*scale_factor;
    }
  }
  // Void return.
  return;
}
/*-------------------------------------------------------*/
```

```
/* This function performs the back substitution.
*/
void back_substitute(double a[][N + 1], int n,
double soln[])
{
// Perform back substitution in each equation.
soln[n - 1] = a[n - 1][n] / a[n - 1][n - 1];
for (int row = n - 2; row >= 0; --row)
{
for (int col = n - 1; col >= row + 1; --col)
{
a[row][n] -= soln[col] * a[row][col];
}
soln[row] = a[row][n] / a[row][row];
}
// Void return.
return;
}
```

***HAND IN LABORATORY TASK: LAB #28***

The text solve by hand using Gauss elimination the following 3 simultaneous equations in section 8.5.

Study the text solution and then use the previous program, set up the matrix a(i,j) appropriate for these equations and solve for x, y, z. (just consider i1 is x, i2 is y and i3 is z. You are setting The program for any a(i,j) for this problem. Be sure to keep it as An important example

3x +2y-z=10

-x +3y +2z =5

x – y –z =-1

Use code like this to load the values of a(i,j) instead of loading One at a time as in the original program code 8_6

```
for (int i=0; i<N; ++i)   {//every row 3
  for (int j=0; j<N+1; ++j )//every col 4!
    cin >> array[i][j];
}
```

Check your answers to the text solution.. Did you get the right ones?? Yes or NO!

If not then check your matrix to be sure the values are correct