APA 254
Data Structures

Lecture 4.1
(Array and Matrices)

Dept. of Information Systems
Hanyang University

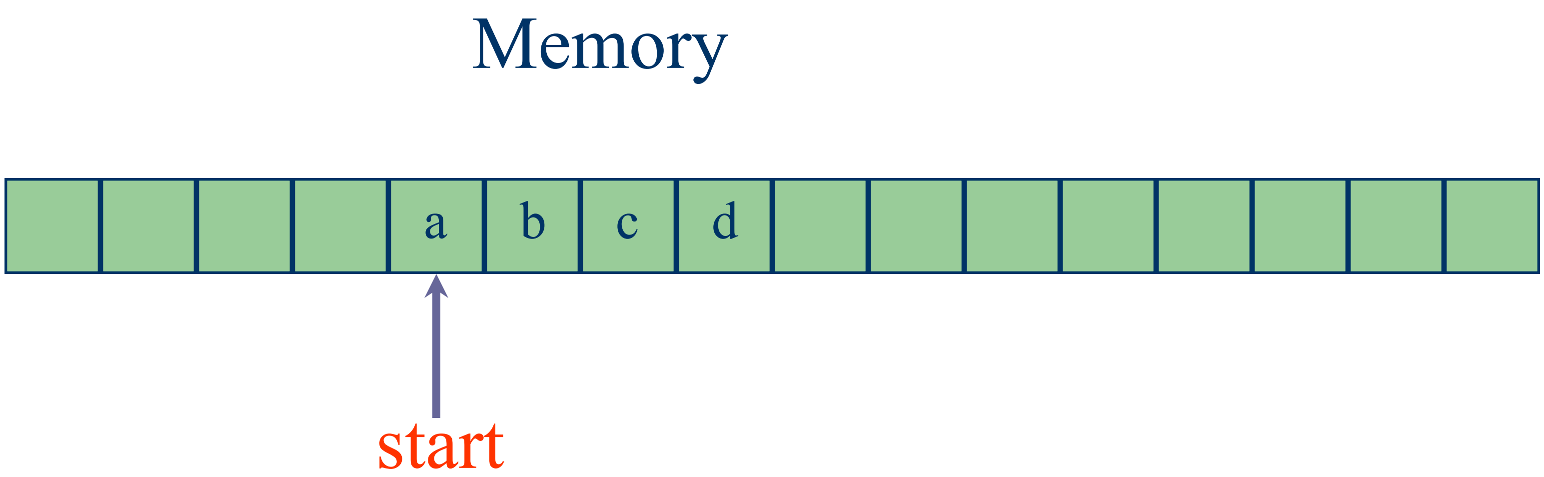
Introduction

- Data is often available in tabular form
- Tabular data is often represented in arrays
- Matrix is an example of tabular data and is often represented as a 2-dimensional array
  - Matrices are normally indexed beginning at 1 rather than 0
  - Matrices also support operations such as **add**, **multiply**, and **transpose**, which are NOT supported by C++'s 2D array

Introduction

- It is possible to **reduce time and space** using a **customized representation** of multidimensional arrays
- This chapter focuses on
  - Row- and column-major mapping and representations of multidimensional arrays
  - the class Matrix
  - Special matrices
    - ✓ Diagonal, tridiagonal, triangular, symmetric, sparse

1D Array Representation in C++



- 1-dimensional array x = [a, b, c, d]
- map into contiguous memory locations
- location(x[i]) = start + i

2D Arrays

The elements of a 2-dimensional array a declared as:

int a[3][4];

may be shown as a table

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

Rows of a 2D Array

a[0][0]	a[0][1]	a[0][2]	a[0][3]	row 0
a[1][0]	a[1][1]	a[1][2]	a[1][3]	row 1
a[2][0]	a[2][1]	a[2][2]	a[2][3]	row 2

Columns of a 2D Array

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]
column 0	column 1	column 2	column 3

2D Array Representation in C++

2-dimensional array x

a, b, c, d

e, f, g, h

i, j, k, l

view 2D array as a 1D array of rows

x = [row0, row1, row 2]

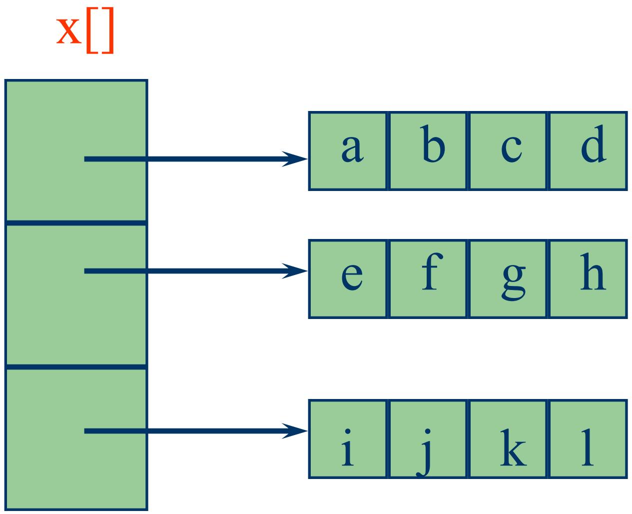
row 0 = [a, b, c, d]

row 1 = [e, f, g, h]

row 2 = [i, j, k, l]

and store as 4 1D arrays

2D Array Representation in C++

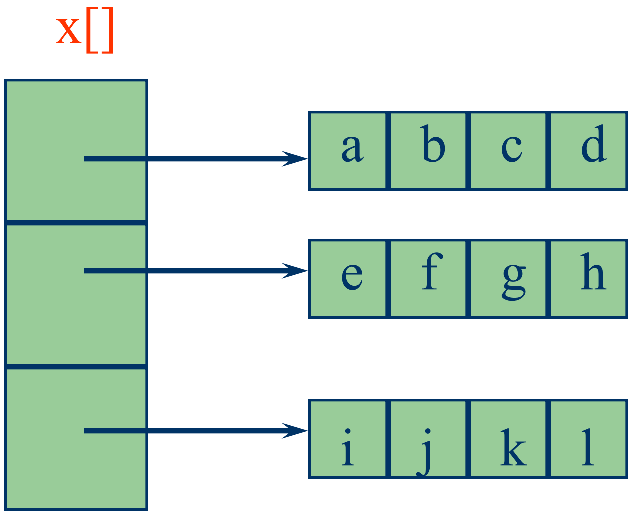


4 separate

1-dimensional arrays

- space overhead = overhead for 4 1D arrays

Array Representation in C++



- This representation is called the array-of-arrays representation.
- Requires contiguous memory of size 3, 4, 4, and 4 for the 4 1D arrays.
- 1 memory block of size number of rows and number of rows blocks of size number of columns

Row-Major Mapping

- Example 3 x 4 array:

  a b c d

  e f g h

  i j k l

- Convert into 1D array y by collecting elements by rows.
- Within a row elements are collected from left to right.
- Rows are collected from top to bottom.
- We get y[] = {a, b, c, d, e, f, g, h, i, j, k, l}

row 0	row 1	row 2	…	row i		

Locating Element x[i][j]

row 0	row 1	row 2	…	row i		

- assume x has r rows and c columns
- each row has c elements
- i rows to the left of row i
- so ic elements to the left of x[i][0]
- x[i][j] is mapped to position

  ic + j of the 1D array

Column-Major Mapping

a b c d

e f g h

i j k l

- Convert into 1D array y by collecting elements by columns.
- Within a column elements are collected from top to bottom.
- Columns are collected from left to right.
- We get y = {a, e, i, b, f, j, c, g, k, d, h, l}

Row- and Column-Major Mappings

2D Array int a[3][6];

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0][5]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]	a[1][5]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]	a[2][5]

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17

(a) Row-major mapping

0	3	6	9	12	15
1	4	7	10	13	16
2	5	8	11	14	17

(b) Column-major mapping

Row- and Column-Major Mappings

- Row-major order mapping functions

  $map(i_1,i_2) = i_1u_2+i_2$ for 2D arrays

  $map(i_1,i_2,i_3) = i_1u_2u_3+i_2u_3+i_3$ for 3D arrays

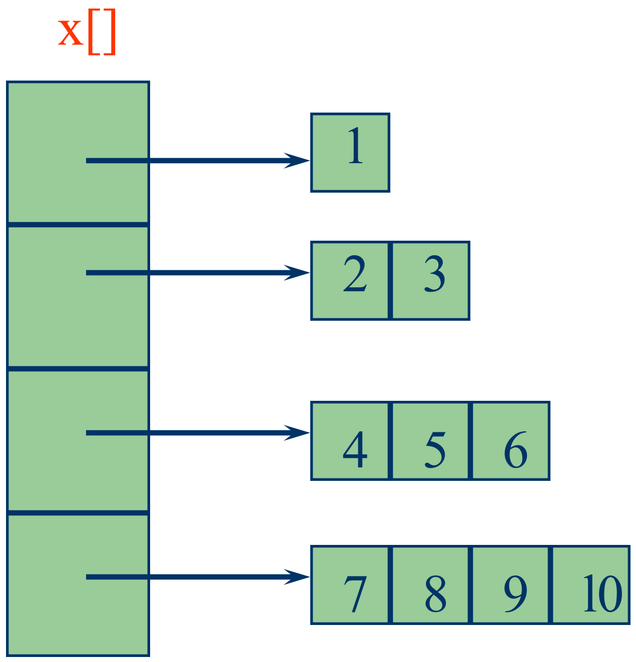
- What is the mapping function for Figure 7.2(a)?

  $map(i_1,i_2) = 6i_1+i_2$

  $map(2,3) = ?$

- Column-major order mapping functions

Irregular 2D Arrays



Irregular 2-D array: the length of rows is not required to be the same.

Matrices

- *m x n* matrix is a table with *m* rows and *n* columns.
- *M(i,j)* denotes the element in row *i* and column *j*.
- Common matrix operations
  - transpose
  - addition
  - multiplication

$$
\begin{array}{cc}
 & \begin{array}{cccc} \text{col } 1 & \text{col } 2 & \text{col } 3 & \text{col } 4 \end{array} \\
\begin{array}{c} \text{row } 1 \\ \text{row } 2 \\ \text{row } 3 \\ \text{row } 4 \\ \text{row } 5 \end{array} &
\begin{bmatrix} 7 & 2 & 0 & 9 \\ 0 & 1 & 0 & 5 \\ 6 & 4 & 2 & 0 \\ 8 & 2 & 7 & 3 \\ 1 & 4 & 9 & 6 \end{bmatrix}
\end{array}
$$

Matrix Operations

- Transpose
  - The result of transposing an *m x n* matrix is an *n x m* matrix with property:

    $M^T(j,i) = M(i,j),\ 1 <= i <= m,\ 1 <= j <= n$

- Addition
  - The sum of matrices is only defined for matrices that have the same dimensions.
  - The sum of two *m x n* matrices A and B is an *m x n* matrix with the property:

    $C(i,j) = A(i,j) + B(i,j),\ 1 <= i <= m,\ 1 <= j <= n$

Matrix Operations

- Multiplication
  - The product of matrices A and B is only defined when the number of columns in A is equal to the number of rows in B.
  - Let A be *m x n* matrix and B be a *n x q* matrix. A*B will produce an *m x q* matrix with the following property:

$$C(i,j) = \Sigma(k=1\ldots n)\ A(i,k) * B(k,j)$$

where $1 <= i <= m$ and $1 <= j <= q$

- Read Example 7.2

A Matrix Class

- There are many possible implementations for matrices.

```
// use a built-in 2 dimensional array
T matrix[m][n]

// use the Array2D class
Array2D<T> matrix(m,n)

// or flatten the matrix into a one-dimensional array
template<class T>
class Matrix {
      private:        int rows, columns;
                      T *data;
};
```

Shortcomings of using a 2D Array for a Matrix

- Indexes are off by 1.
- C++ arrays do not support matrix operations such as add, transpose, multiply, and so on.
  - Suppose that x and y are 2D arrays. Cannot do x + y, x –y, x * y, etc. in C++.
- We need to develop a class matrix for object-oriented support of all matrix operations.
- See Programs 7.2-7.7
- Read Sections 7.1-7.2

Special Matrices

- A square matrix has the same number of rows and columns.
- Some special forms of square matrices are
  - Diagonal: $M(i,j) = 0$ for $i \neq j$
  - Tridiagonal: $M(i,j) = 0$ for $|i-j| < 1$
  - Lower triangular: $M(i,j) = 0$ for $i < j$
  - Upper triangular: $M(i,j) = 0$ for $i > j$
  - Symmetric $M(i,j) = M(j,i)$ for all i and j
- See Figure 7.7

Special Matrices

$$\begin{matrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 6 \end{matrix}$$

(a) Diagonal

$$\begin{matrix} 2 & 1 & 0 & 0 \\ 3 & 1 & 3 & 0 \\ 0 & 5 & 2 & 7 \\ 0 & 0 & 9 & 0 \end{matrix}$$

(b) Tridiagonal

$$\begin{matrix} 2 & 0 & 0 & 0 \\ 5 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 4 & 2 & 7 & 0 \end{matrix}$$

(c) Lower triangular

$$\begin{matrix} 2 & 1 & 3 & 0 \\ 0 & 1 & 3 & 8 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 0 \end{matrix}$$

(d) Upper triangular

$$\begin{matrix} 2 & 4 & 6 & 0 \\ 4 & 1 & 9 & 5 \\ 6 & 9 & 4 & 7 \\ 0 & 5 & 7 & 0 \end{matrix}$$

(e) Symmetric

Special Matrices

- Why are we interested in these "special" matrices?
  - We can provide more efficient implementations for specific special matrices.
  - Rather than having a space complexity of $O(n^2)$, we can find an implementation that is $O(n)$.
  - We need to be clever about the "store" and "retrieve" operations to reduce time.
- Read Examples 7.4 & 7.5

Diagonal Matrix

$$\begin{matrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 6 \end{matrix}$$

- Naive way to represent *n x n* diagonal matrix
  - T d[n][n]
  - d[i-1][j-1] for D(i,j)
  - requires n^2 x sizeof(T) bytes of memory
- Better way
  - T d[n]
  - d[i-1] for D(i,j) where i = j
    0 for D(i,j) where i ≠ j
  - requires n x sizeof(T) bytes of memory
- See Program 7.8 for the class diagonalMatrix

Tridiagonal Matrix

$$\begin{matrix} 2 & 1 & 0 & 0 \\ 3 & 1 & 3 & 0 \\ 0 & 5 & 2 & 7 \\ 0 & 0 & 9 & 0 \end{matrix}$$

- Nonzero elements lie on one of three diagonals:
  - main diagonal: $i = j$
  - diagonal below main diagonal: $i = j+1$
  - diagonal above main diagonal: $i = j-1$
- *3n-2* elements on these three diagonals: T *t[3n-2]*
- Mappings of Figure 7.2(b)
  - by row [2,1,3,1,3,5,2,7,9,0]
  - by column [2,3,1,1,5,3,2,9,7,0]
  - by diagonal [3,5,9,2,1,2,0,1,3,7]
    - ✓ more on diagonal mapping on the next page

Tridiagonal Matrix

$$\begin{matrix} 2 & 1 & 0 & 0 \\ 3 & 1 & 3 & 0 \\ 0 & 5 & 2 & 7 \\ 0 & 0 & 9 & 0 \end{matrix}$$

- Mapping by diagonals beginning with the lowest

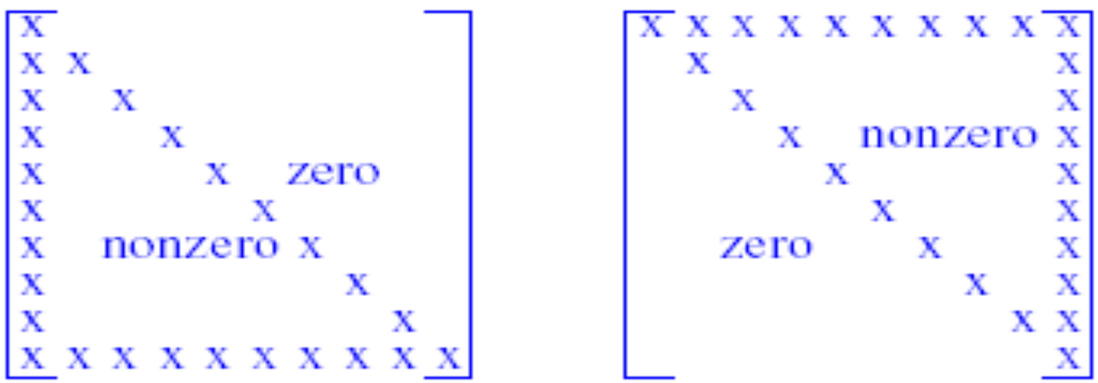
  D(2,1) -> t[0]
  D(3,2) -> t[1]
  …
  D(n,n-1) -> t[n-2]
  D(1,1) -> t[n-1]
  D(2,2) -> t[n]
  …
  D(n, n) -> t[(n-2)+n] = t[2n-2]
  D(1,2) -> t[2n-1]
  D(2,3) -> t[2n]
  …
  D(n-1,n) -> t[(2n-2)+(n-1)] = t[3n-3]

```
switch (i - j) {
  case 1:  // lower diagonal
   return t[i - 2];
  case 0:  // main diagonal
   return t[n + i - 2];
  case -1:  // upper diagonal
   return t[2 * n + i - 2];
  default:  return 0;
  }
```

- See Program 7.11

Triangular Matrix

- Nonzero elements lie in the region marked "nonzero" in the figure below



lower triangular   upper triangular

- *1+2+…+n* = *Σ(i=1..n)* = *n(n+1)/2* elements in the nonzero region

Triangular Matrix

- Both triangular matrices may be represented using 1-D array → T t[n(n+1)/2]

$$\begin{matrix} 2 & 0 & 0 & 0 \\ 5 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 4 & 2 & 7 & 0 \end{matrix}$$

- Mappings
  - by row?

    → [2,5,1,0,3,1,4,2,7,0]
  - by column?

    → [2,5,0,4,1,3,2,1,7,0]

Lower Triangular Matrix

$$\begin{matrix} 2 & 0 & 0 & 0 \\ 5 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 4 & 2 & 7 & 0 \end{matrix}$$

- Mapping by row

$$L(i,j) = 0 \qquad \text{if } i < j$$

$$L(i,j) = t[1+2+\ldots+(i-1)+(j-1)] \qquad \text{if } i \geq j$$

$$= t[i(i-1)/2 + j-1]$$

- See Program 7.12 for the method lowerTriangularMatrix<T>::set

Upper Triangular Matrix

$$\begin{matrix} 2 & 1 & 3 & 0 \\ 0 & 1 & 3 & 8 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 0 \end{matrix}$$

- Mapping by column

  → [2, 1, 1, 3, 3, 1, 0, 8, 6, 0]

  $L(i,j) = ?$ *if* $i > j$

  $L(i,j) = ?$ *if* $i \le j$

- Exercise: Write the method for upperTriangularMatrix<T>::set

Symmetric Matrix

- An *n x n* matrix can be represented using 1-D array of size *n(n+1)/2* by storing either the lower or upper triangle of the matrix
- Use one of the methods for a triangular matrix
- The elements that are not explicitly stored may be computed from those that are stored
  - How do we compute this?

$$\begin{matrix} 2 & 4 & 6 & 0 \\ 4 & 1 & 9 & 5 \\ 6 & 9 & 4 & 7 \\ 0 & 5 & 7 & 0 \end{matrix}$$

Sparse Matrix

- A matrix is sparse if many of its elements are zero
- A matrix that is not sparse is dense
- The boundary is not precisely defined
  - Diagonal and tridiagonal matrices are sparse
  - We classify triangular matrices as dense
- Two possible representations
  - array
  - linked list

- Read Example 7.6

Array Representation of Sparse Matrix

- The nonzero entries may be mapped into a 1D array in row-major order
- To reconstruct the matrix structure, need to record the row and column each nonzero comes from

$$\begin{matrix} 0 & 0 & 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 6 & 0 & 0 & 7 & 0 & 0 & 3 \\ 0 & 0 & 0 & 9 & 0 & 8 & 0 & 0 \\ 0 & 4 & 5 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

(a) A 4×8 matrix

a[]	0	1	2	3	4	5	6	7	8
row	1	1	2	2	2	3	3	4	4
col	4	7	2	5	8	4	6	2	3
value	2	1	6	7	3	9	8	4	5

(b) Its representation

Array Representation of Sparse Matrix

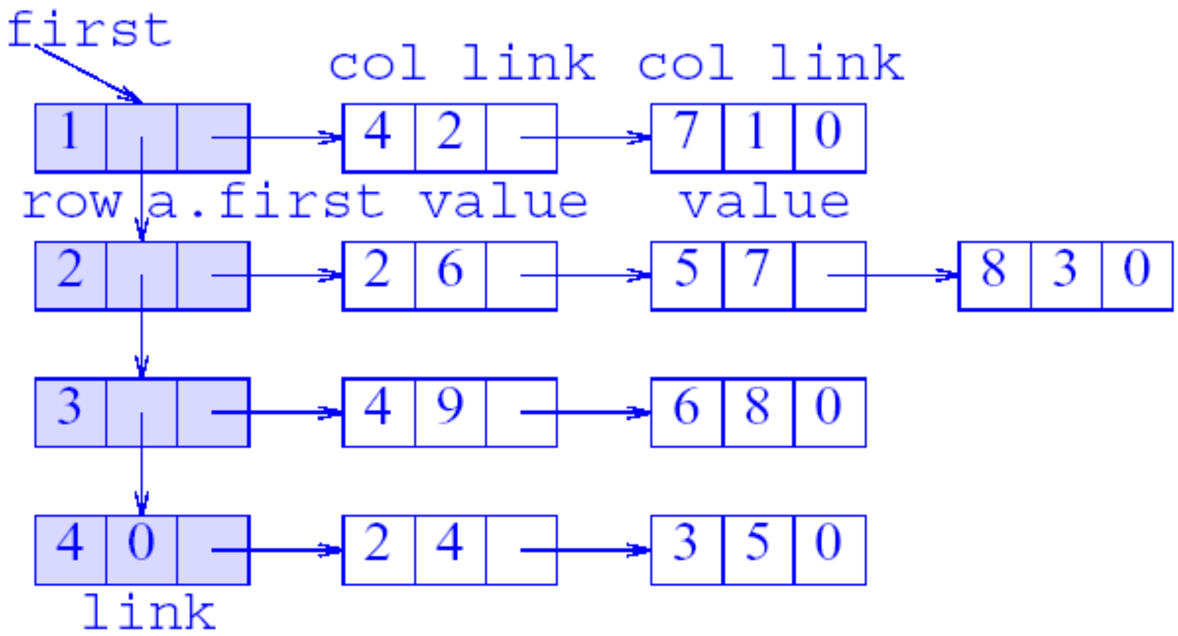
```
template<class T>
class Term {
private:
        int row, col;
        T value;
};
```

```
template<class T>
class sparseMatrix {
private:
        int rows, cols,
        int terms;
        Term<T> *a;
        int MaxTerms;
public:
        //…
};
```

- See Programs 7.13~7.17 for the class definition and methods of sparseMatrix

Linked Representation of Sparse Matrix

- A shortcoming of the 1-D array of a sparse matrix is that we need to know the number of nonzero terms in each of the sparse matrices when the array is created
- A linked representation can overcome this shortcoming



Linked Representation of Sparse Matrix

- See Program 7.18 for linked representation of sparse matrix
- Read Chapter 7