



# Fundamentals of Data Structures

S. Y. B. Tech CSE

Semester – III

---

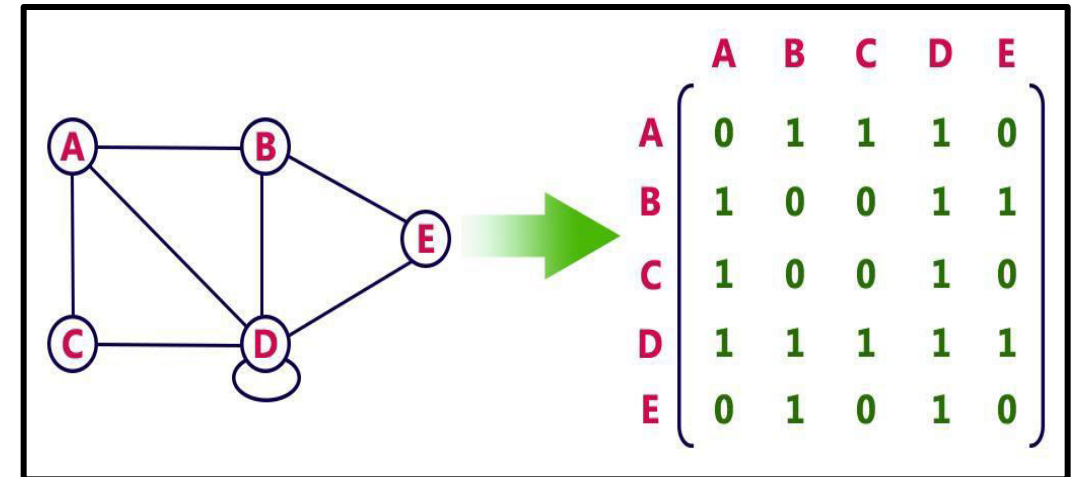
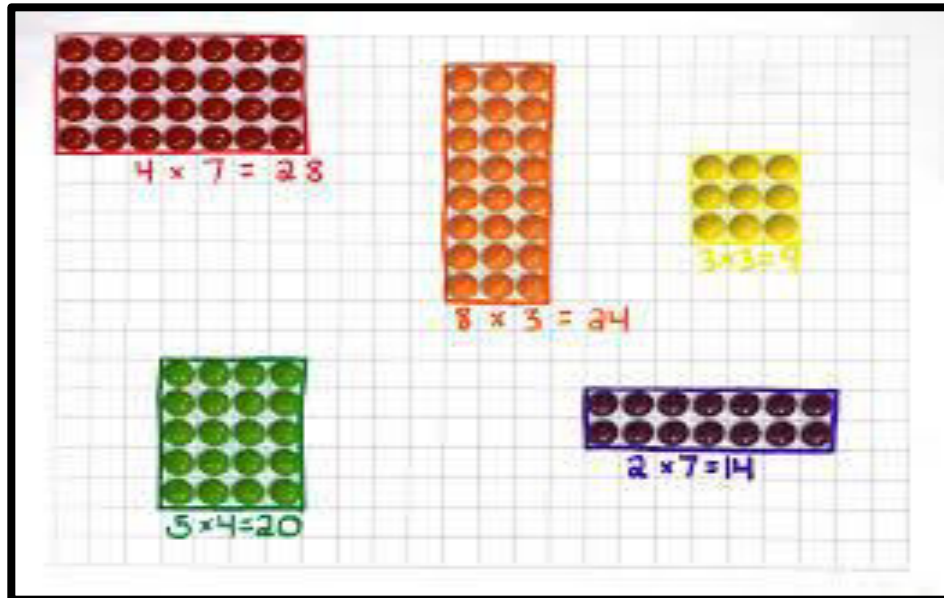
SCHOOL OF COMPUTER ENGINEERING AND TECHNOLOGY

# Linear Data Structures

---

**Linear Data Structures:** Array as an Abstract Data Type, Sequential Organization, Storage Representation and their Address Calculation: Row major and Column Major, Multidimensional Arrays: Concept of Ordered List, Single Variable Polynomial: Representation using arrays, Polynomial as array of structure, Polynomial addition, Polynomial evaluation and Polynomial multiplication. Sparse Matrix: Sparse matrix representation using array, Sparse matrix addition, Transpose of sparse matrix- Simple and Fast Transpose, Time and Space tradeoff.

# Applications of an Array



# Array as an Abstract Data Type

Array is set of pairs – index and value.

For each index, there is a value associated with it.

Arrays are often stored in consecutive set of memory locations.

Mainly two operations perform on array are retrieve and store values.

Array :

3	8	1	0	5	-2	32
0	1	2	3	4	5	6

# Array as an Abstract Data Type

```
structure ARRAY (value, index)
  Declare CREATE() -> array
           RETRIEVE(array, index) ->
value
           STORE(array, index, value) ->
array;
  for all  $A \in \text{array}$ ,  $i, j \in \text{index}$ ,  $x \in \text{value}$  let
    RETRIEVE(CREATE() , i) ::= error
    RETRIEVE(STORE(A, i, x) , j) ::= val
  end
end ARRAY
```

The function CREATE produce a new empty array.

RETRIEVE takes as input an array and an index and either returns the appropriate values or an error.

STORE is used to enter new index-value pairs

# Sequential Organization

---

In sequential organization, elements are stored in consecutive memory locations. Ex- array

Properties of sequential data structure are:

- Simple to use
- Simple to define
- Constant access time
- Mapping by compiler

An array element can be accessed by  $a[i]$  where  $a$  is the name of the array and  $i$  is the index.

# Sequential Organization

---

Compiler maps  $a[i]$  to its physical location in memory.

**Address of  $a[i]$  is given by**  
**starting address of  $a$  +  $i$  \* size of array elements in bytes**

This mapping is carried out in constant time, irrespective of which element is being accessed.

Limitations:

- Size of an array is defined at the time of programming
- Insertion and deletion is time consuming
- Requires contiguous memory

# Memory Representation and Address Calculation

---

## 1-D Array

1-D array is a list of elements or simply a row of elements.

In mathematic, we often deal with variable that are simple scripted

—

Where  $x_i$  refers to the  $i^{\text{th}}$  element of  $x$  variables can be expressed as –  $x[0]$ ,  $x[1]$ , ....  $x[4]$

$$\sum_{i=1}^4 x_i = x_1 + x_2 + x_3 + x_4$$



# Memory Representation and Address Calculation

Example: 1-D array A[7] where A is array name and 7 is size.

Syntax: `int A[7];`

index	value	Memory address
[0]	10	1024
[1]	20	1028
[2]	30	1032
[3]	40	1036
[4]	50	1040
[5]	60	1044
[6]	70	1048

# Memory Representation and Address Calculation

---

**Address of 1<sup>st</sup> element of an array is called Base Address (BA).**

Address of i<sup>th</sup> element = BA + offset of the i<sup>th</sup> element from BA

Where offset = (number of element before i<sup>th</sup> element) \* size of each element

Ex: consider an array : `int a[10];` and BA = 403, size of element = 4 bytes. Calculate address of `a[3]`?

$$a[3] = 403 + (3 * 4) = 403 + 12 = 415$$

# Memory Representation and Address Calculation

---

## 2-D Array

2-D arrays can be thought of as a table consisting of rows and columns.

It is also called as matrix.

1<sup>st</sup> dimension is referred as row and 2<sup>nd</sup> dimension is referred as column.

The elements of 2D array may be arranged either row wise or column wise.

# Memory Representation and Address Calculation

Ex: `int a[3][4];` //declares an integer array of 3 rows and 4 columns

<code>int a[3][4];</code>	Col 0	Col 1	Col 2	Col 3
Row 0	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[0][3]</code>
Row 1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[1][3]</code>
Row 2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>	<code>a[2][3]</code>

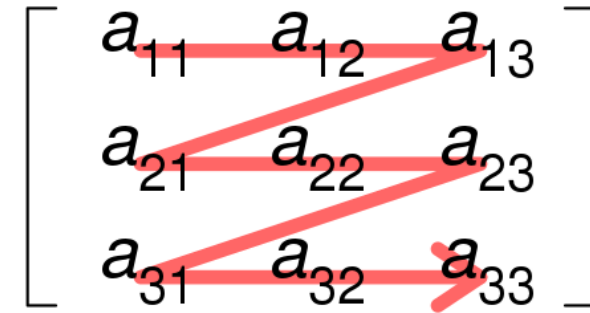
In memory, all elements are stored linearly using contiguous addresses.

# Memory Representation and Address Calculation

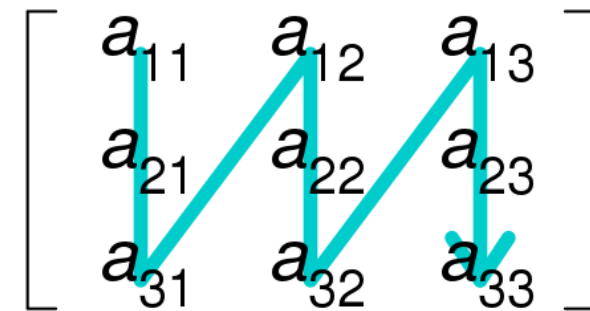
In order to store 2D matrix, a 2D address space must be mapped to 1D address space.

In computer memory, matrices are stored in either row-major form or column-major form.

Row-major order



Column-major order



# Memory Representation and Address Calculation

Column-major representation:

if the elements are stored column-wise manner then it is called column major representation.

`int A[3][4];`

$A =$

1	2	3	4
5	6	7	8
9	10	11	12

Column-major:

$A =$

1	4	7	10
2	5	8	11
3	6	9	12

# Memory Representation and Address Calculation

- Address of an element  $a_{ij}$  (row-major)  
 $= BA + (\text{row\_index} * \text{total\_no\_of\_col} + \text{col\_index}) * S$ ; **or**  
 $B + (i - L1) * (U2 - L2 + 1) * S + (j - L2) * S$ 
  - Where B = base address
  - L1 & U1 are lower and upper bound values for rows
  - L2 & U2 are lower and upper bound values for columns
  - S is Number of bytes taken to store element
- Ex: `int a[3][4];` BA = 100; S=4 bytes; Find location of `a[2][1]` row-wise?

$$\begin{aligned} A[2][1] &= 100 + (2-0) * (4 - 0 + 1) * 4 + (1 - 0) * 4 && 100 + (2*4+1)*4 \\ &= 100 + 2 * 5 * 4 + 4 && 100+44 \\ &= 100 + 40 + 4 && 144 \\ &= 144 \end{aligned}$$

# Memory Representation and Address Calculation

- Address of an element  $a_{ij}$  (column-major) =
- $BA + (\text{col\_index} * \text{total\_no\_of\_row} + \text{row\_index}) * S$ ; or
  - $B + (j - L2) * (U1 - L1 + 1) * S + (i - L1) * S$ 
    - Where B = base address
    - L1 & U1 are lower and upper bound values for rows
    - L2 & U2 are lower and upper bound values for columns
    - S is Number of bytes taken to store element
- Ex: `int a[3][4]`; BA = 100; S=4 bytes; find location of `a[2][1]` col-wise?

$$\begin{aligned} A[2][1] &= 100 + (1-0) * (3 - 0 + 1) * 4 + (2 - 0) * 4 \\ &= 100 + 1 * 4 * 4 + 8 \\ &= 100 + 16 + 8 \\ &= 124 \end{aligned}$$

$$\begin{aligned} &100 + (1*3+2)*4 \\ &100+24 \\ &124 \end{aligned}$$



# Row -major and Column -major Address

---

## Example:

int a[3][4], Base address 1050 s=4, find a[2][3] in row major and column major representation

### Row-major :

$$\begin{aligned}A[2][3] &= 1050 + (2 \times 4 + 3) \times 4 \\&= 1050 + 44 \\&= 1094\end{aligned}$$

### Column-major:

$$\begin{aligned}A[2][3] &= 1050 + (3 \times 3 + 2) \times 4 \\&= 1050 + 44 \\&= 1094\end{aligned}$$

# Multidimensional Array

- In C/C++, multidimensional arrays can be used as array of arrays.
- Data in multidimensional arrays are stored in tabular form (in row major order).
- Declaration
  - General form of declaring N-dimensional arrays:
    - `data_type array_name[size1][size2]....[sizeN];`  
e.g. `int a[5][10][20];`
- Size of multidimensional arrays
  - Total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensions.
  - `int a[5][10][20];`
    - 1000 elements in an array 'a'.

# Memory Representation and Address Calculation (row-major)

- $a[S_1][S_2] \dots [S_N]$
- Address of an element  $a_{ij}$
- Address of  $a[l_1][l_2] \dots [l_N] = B + W * [((E_1 S_2 + E_2) S_3 + E_3) S_4 \dots + E_{N-1} S_N + E_N]$ 
  - Where  $B$  = base address
  - $W$  represents the size of an element in bytes
  - $E_i$  is given by  $E_i = l_i - t_i$ , where  $l_i$  and  $t_i$  are the calculated indexes (indices of array element which needs to be determined) and lower bounds respectively.
  - $S_1, S_2, S_3, \dots, S_N$  are dimensions
  - $l_1, l_2, l_3, \dots, l_N$  are indices of the element whose address needs to be calculated.

# Memory Representation and Address Calculation (row-major)

- Example
- `int A[10][20][30][40]` with base address 1200
- Find the address of element `A[1][3][5][6]`
- $B = 1200$  and  $W = 4$ ,  $S_1 = 10$ ,  $S_2 = 20$ ,  $S_3 = 30$ ,  $S_4 = 40$
- Lower bounds is zero.
  - $E_1 = 1 - 0 = 1$
  - $E_2 = 3 - 0 = 3$
  - $E_3 = 5 - 0 = 5$
  - $E_4 = 6 - 0 = 6$
- $A[I_1][I_2] \dots [I_N] = B + W * [((E_1 S_2 + E_2) S_3 + E_3) S_4 \dots + E_{N-1} S_N + E_N]$
- $A[1][3][5][6] = 1200 + 4(((1 \times 20 + 3)30 + 5)40 + 6) = 112424$

# Memory Representation and Address Calculation (column-major)

- Address of  $a[I_1][I_2] \dots [I_N] = B + W * [((\dots E_N S_{N-1} + E_{N-1}) S_{N-2} + \dots E_3) S_2 + E_2) S_1 + E_1]$
- Example
- `int A[10][20][30][40]` with base address 1200
- Find the address of element `A[1][3][5][6]`
- $B = 1200$  and  $W = 4$ ,  $S_1 = 10$ ,  $S_2 = 20$ ,  $S_3 = 30$ ,  $S_4 = 40$
- Lower bounds is zero.
  - $E_1 = 1 - 0 = 1$
  - $E_2 = 3 - 0 = 3$
  - $E_3 = 5 - 0 = 5$
  - $E_4 = 6 - 0 = 6$
- $A[I_1][I_2] \dots [I_N] = B + W * [((\dots E_N S_{N-1} + E_{N-1}) S_{N-2} + \dots E_3) S_2 + E_2) S_1 + E_1]$
- $A[1][3][5][6] = 1200 + 4(((6*30+5)20+3)10+1) = 149324$

# Concept of Ordered List

- One of the simplest and most commonly found data object is the ordered or linear list.
- Examples
  - Days of the week
    - (MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY)
  - The values in a card deck
    - (2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace)
- If we consider an ordered list more abstractly, we say that it is either empty or it can be written as
  - $(a_1, a_2, a_3, \dots, a_n)$ 
    - where the  $a_i$  are atoms from some set  $S$ .
- There are a variety of operations that are performed on these lists.
  - find the length of the list,  $n$ ;
  - read the list from left-to-right (or right-to-left);
  - retrieve the  $i$ -th element, ;
  - store a new value into the  $i$ -th position, ;
  - insert a new element at position causing elements numbered  $i, i + 1, \dots, n$  to become numbered  $i + 1, i + 2, \dots, n + 1$ ;
  - delete the element at position causing elements numbered  $i + 1, \dots, n$  to become numbered  $i, i + 1, \dots, n - 1$ .

# Representation of Ordered List using an Array

---

- ☐ Associate the list element  $a_i$  with the array index  $i$ .
- ☐ This we will refer to as a sequential mapping
  - ☐ Using the conventional array representation we are storing  $a_i$  and  $a_{i+1}$  into consecutive locations  $i$  and  $i + 1$  of the array.
- ☐ Retrieve or modify the values of random elements in the list is done in a constant amount of time.

# Representation of Polynomials using arrays

---

Polynomial is one of the classic example of an Ordered List.

An ordered list is set of elements where set may be empty or it can be written as a collection of elements such as  $(a_1, a_2, a_3, \dots, a_n)$ , where  $a_i$  are atoms from some set  $S$ .

A polynomial is the sum of terms where each term consists of variable, coefficient and exponent.

Ex:  $A(x) = 3x^2 + 2x + 4$  and  $B(x) = x^4 + 10x^3 + 3x^2 + 1$



# Representation of Polynomials of Degree n in an array

We have a Polynomial-

$$C_{m-1} X^{m-1} + C_{m-2} X^{m-2} + \dots + C_0 X^0$$

Polynomial with n terms can be represented by the ordered list of length 2n+1

(n, (power, coeff), (power, coeff), .....)

Ex:  $x^3 + 5x^2 + 9$  can be represented as –

(3, (3,1), (2,5), (0,9)) or

3	3	1	2	5	0	9
---	---	---	---	---	---	---

$x^4 + 59x + 10$

(3, (4, 1), (1, 59), (0, 10)) or


3	4	1	1	59	0	10
---	---	---	---	----	---	----

# Representation of Polynomials

## $A(x, y)$

If the polynomial has  $m$  non-zero terms then the polynomial can be represented as an ordered list of  $3m + 1$  terms.

$m$ , power of  $x$ , power of  $y$ , coeff, power of  $x$ , power of  $y$ , coeff,  
...



1<sup>st</sup> term

2<sup>nd</sup> term

1<sup>st</sup> entry is the number of non-zero terms. For each term, there are 3 entries- an exponent of  $x$ ,  $y$  and coefficient triple.

# Representation of Polynomials

## $A(x, y)$

$$3x^3y^2 + 10xy - 1$$

$$(3, (3, 2, 3), (1, 1, 10), (0, 0, -1))$$

OR

3	3	2	3	1	1	10	0	0	-1
---	---	---	---	---	---	----	---	---	----

$$X^2+5xy+y^2-y-x$$

$$(5, (2, 0, 1), (1, 1, 5), (0, 2, 1), (0, 1, -1), (1, 0, -1))$$

OR

5	2	0	1	1	1	5	0	2	1	0	1	-1	0	0	-1
---	---	---	---	---	---	---	---	---	---	---	---	----	---	---	----

# Representation of Polynomials A (x, y, z)

If the polynomial has  $m$  non-zero terms then the polynomial can be represented as an ordered list of  $4m + 1$  terms.

... m, power of x, power of y, power of z, coeff, power of x, power of y, power of z coeff,

1<sup>st</sup> term

2<sup>nd</sup> term

1<sup>st</sup> entry is the number of non-zero terms. For each term, there are 4 entries- an exponent of x, y, z and coefficient.

# Representation of Polynomials A (x, y)

$$5x^3y^2z + 3x^2y^3z^2 + 6xyz^3 + 10$$

(4, (3, 2, 1, 5), (2, 3, 2, 3), (1, 1, 3, 6), (0, 0, 0, 10))

OR

4	3	2	1	5	2	3	2	3	1	1	3	6	0	0	0	10
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----

$$-8x^3y^3z^3 + 10x^3y^2z + 5xyz^2 + 10$$

(4, (3,3,3, -8), (3,2,1, 10), (1,1,2,5), (0,0,0,10))

OR

4	3	3	3	-8	3	2	1	10	1	1	2	5	0	0	0	10
---	---	---	---	----	---	---	---	----	---	---	---	---	---	---	---	----

# Addition and Evaluation of Polynomials

---

Add  $A(x) = 3x^2 + 2x + 4$  and  $B(x) = x^4 + 10x^3 + 3x^2 + 1$

Result  $C(x) = x^4 + 10x^3 + 5x^2 + 2x + 5$

While storing the polynomials in the arrays, we have to store them in the descending order of the exponents.

# Adding Polynomials

## Horizontal Method

$$(4x^3 + x^2 - 5x - 7) + (-8x^3 - 2x + 1)$$

$$-4x^3 + x^2 - 7x - 6$$

## Vertical Method

$$4x^3 + x^2 - 5x - 7$$

$$-8x^3 \quad - 2x + 1$$

$$-4x^3 + x^2 - 7x - 6$$

Add the polynomials.

$$1) (6x^3 - 4x + 3) + (9x^3 + 6x - 11)$$

$$15x^3 + 2x - 8$$

$$2) (7y^4 + 9y^2 - 6) + (3y^3 - 10y^2 - 8)$$

$$7y^4 + 3y^3 - y^2 - 14$$

$$3) \begin{array}{r} 4x^2 - 5x + 2 \\ 3x^2 - 4x - 7 \\ \hline \end{array}$$

$$7x^2 - 9x - 5$$

$$4) \begin{array}{r} 6m^3 - 2m^2 + m \\ -m^3 \qquad \qquad -7m + 10 \\ \hline \end{array}$$

$$5m^3 - 2m^2 - 6m + 10$$



# Representing Polynomial using array of Structure

Consider polynomial  $A(x) = 2x^{1000} + 1$ , representing using array of structure:

```
#define Max_Terms 100;
```

```
struct polynomial  
{  
    int coef;  
    int exp;  
};
```

```
typedef struct polynomial poly[Max_Terms];
```

	Coeff	Expo
[0]	2	1000
[1]	1	0

## Polynomial Addition

$$P1(x) = 3x^2 + 2x + 4$$

$$P2(x) = x^4 + 10x^3 + 3x^2 - 2x + 1$$

$$P3(x) = x^4 + 10x^3 + 6x^2 + 5$$

Case 1: exponent of  $p1 >$  exponent of  $p2$   
copy term of  $p1$  to end of  $p3$ .  
[go to next term in  $p1$ ]

Case 2: exponent of  $p1 <$  exponent of  $p2$   
copy term of  $p2$  to end of  $p3$ .  
[go to next term in  $p2$ ]

Case 3: exponent of  $p1 =$  exponent of  $p2$   
copy term in  $p3$  with the same exponent and with the sum of the coefficients of  $p1$  and  $p2$ .  
[go to next term in  $p1$  &  $p2$ ]

Algorithm polyadd (p1, p2, max1, max2)

```
{
    i = j = k = 0;
while ( i < max1 && j < max2)
{
    if p1[i].exp > p2[j].exp
    {
        p3[k] = p1[i];
        k++; i++;
    }
    else if( p1[i].exp < p2[j].exp)
    {
        p3[k] = p2[j];
        k++; j++;
    }
    else // same exponents
    {
        temp = p1[i].coef + p2[j].coef;
        if(temp != 0)
        {
            p3[k].exp = p1[i].exp;
            p3[k].coef = temp;
            i++; j++; k++;
        }
    }
} // end while
```

```
while( i < max1 )
```

```
{
```

```
    p3[k] = p1[i];
```

```
    k++;
```

```
    i++;
```

```
} //while
```

```
while( j < max2 )
```

```
{
```

```
    p3[k] = p2[j];
```

```
    k++;
```

```
    j++;
```

```
} //while
```

```
}
```

# Evaluation of Polynomial

"Evaluating" a polynomial is the same as evaluating anything else: you plug in the given value of  $x$ , and figure out what the answer will be, or, in some cases, what  $y$  is supposed to be. For instance:

**Evaluate  $2x^3 - x^2 - 4x + 2$  at  $x = -3$**

Plug in  $-3$  for  $x$ , remembering to be careful with parentheses and negatives:

$$\begin{aligned} & 2(-3)^3 - (-3)^2 - 4(-3) + 2 \\ &= 2(-27) - (9) + 12 + 2 \\ &= -54 - 9 + 14 \\ &= -63 + 14 \\ &= -49 \end{aligned}$$

Always remember to be careful with the minus signs!

# Evaluation of Polynomials

---

## Algorithm Eval\_Poly

- Step 1: Read the polynomial array A
- Step 2: Read the value of x
- Step 3: Initialize the variable sum to zero
- Step 4: calculate  $\text{coeff} * \text{pow}(x, \text{exp})$  of each term and add the result to sum
- Step 5: display sum
- Step 6: end

# Polynomial Multiplication

- Given polynomials  $A(x)$ ,  $B(x)$

$$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

$$B(x) = b_0 + b_1x + b_2x^2 + \dots + b_mx^m$$

- Find their product  $C(x) = A(x) B(x)$

$$C(x) = c_0 + c_1x + c_2x^2 + \dots + c_{m+n}x^{m+n}$$

- Given  $a_0, \dots, a_n$  and  $b_0, \dots, b_m$  find  $c_0, \dots, c_{m+n}$ 
  - Natural to use an array of  $n+1$  elements to store the coefficients of a degree  $n$  polynomial.
- Algorithm idea:
  - Each term  $a_ix^i$  in  $A(x)$  will multiply each term  $b_jx^j$  in  $B(x)$  and the product  $a_ib_jx^{i+j}$  will contribute to the term  $c_{i+j}x^{i+j}$ .

□ Polynomial 1:

□  $6x^3 + 10x^2 + 0x^1 + 5$

□ Polynomial 2:

□  $4x^2 + 2x^1 + 1$

□ Multiplication of Polynomials 1 & 2:

□  $24x^5 + 52x^4 + 26x^3 + 30x^2 + 10x^1 + 5$

Algorithm polymultiplication (p1, p2, max1, max2)

```
{
    i = j = k = 0;
    while ( i < max1){
        j = 0;
        while( j < max2)
        {
            temp = p1[i].coef * p2[j].coef;
            if(temp != 0)
            {
                flag = 0;
                exp = p1[i].exp + p2[j].exp;
                for(x = 0; x < k; x++){
                    if(exp == p3[x].exp){
                        flag = 1;
                        break; }
                }
                if(flag == 1){
                    p3[x].coef = p3[x].coef + temp;
                    j++;
                }
            }
        }
    }
}
```

```
else{
    p3[k].exp = exp;
    p3[k].coef = temp;
    j++; k++;
}
} // end if
} // end while
i++;
} // end while
```

# Representation of sparse matrix

A matrix contains m rows and n columns of elements.

	Col0	Col1	Col2
Row0	-27	3	4
Row1	6	82	-2
Row2	109	-64	11
Row3	12	8	9
Row4	48	27	47

mat1(5\*3)

	Col0	Col1	Col2	Col3	Col4	Col5
Row0	15	0	0	22	0	-15
Row1	0	11	3	0	0	0
Row2	0	0	0	-6	0	0
Row3	0	0	0	0	0	0
Row4	91	0	0	0	0	0
Row5	0	0	28	0	0	0

mat2(6\*6)

We write m\*n to designate a matrix with m rows and n columns



- In mat1 contains 15 non-zero elements but in mat2 contains 8 non-zero elements.
- Any  $m \times n$  matrix which contains a large number of zeros is called as **sparse matrix**.
- We can characterize uniquely any element within a matrix by using the triple  $\langle \text{row}, \text{col}, \text{value} \rangle$ .
- We can use an array of triples to represent a sparse matrix.
- An array of triples called as compact form.

- Each triple contains row, column and value of the non-zero elements.
- The first row of compact form contains number of rows, no of columns and number of nonzero elements of original sparse matrix.
- If sparse matrix contains 't' non-zero elements, then the compact form has  $t+1$  rows and 3 columns.

# Sparse matrix in triplet form

	Col0	Col1	Col2	Col3	Col4	Col5
Row0	15	0	0	22	0	-15
Row1	0	11	3	0	0	0
Row2	0	0	0	-6	0	0
Row3	0	0	0	0	0	0
Row4	91	0	0	0	0	0
Row5	0	0	28	0	0	0

Matrix1[6][6]

	Col0	Col1	Col2
Row0	6	6	8
Row1	0	0	15
Row2	0	3	22
Row3	0	5	-15
Row4	1	1	11
Row5	1	2	3
Row6	2	3	-6
Row7	4	0	91
Row8	5	2	28

Matrix2[9][3]

- Matrix1 is a normal matrix and Matrix2 is a compact form of Sparse matrix stored as triples
- Row0 of compact form matrix contains total number of rows, columns and non-zero elements of sparse matrix.
- In Normal matrix representation:  
 $6 * 6 * 2(**) = 72$  bytes are required for storage.
- In Sparse matrix (Triplet) representation:  
 $(8+1) * 3 * 2(**) = 54$  bytes are required for storage.
- ( \*\* 2 bytes are required to store Integer val in 32 bits compiler)

Algorithm compact(A,m,n,B)

{ //m and n are tot. no. of rows. & cols. of original matrix

B(0,0)=m;

B(0,1)=n

k=1;

for i=0 to m

{

for j=0 to n

{

if(A(i,j)!=0)

{

B(k,0)=i;

B(k,1)=j;

B(k,2)=A(i,j);

k++;

} //end for j

} //end for i

}

B(0,2)=k-1;

}

	[0]	[1]	[2]	[3]	[4]	[5]
A[0]	15	0	0	22	0	-15
A[1]	0	11	3	0	0	0
A[2]	0	0	0	-6	0	0
A[3]	0	0	0	0	0	0
A[4]	91	0	0	0	0	0
A[5]	0	0	28	0	0	0

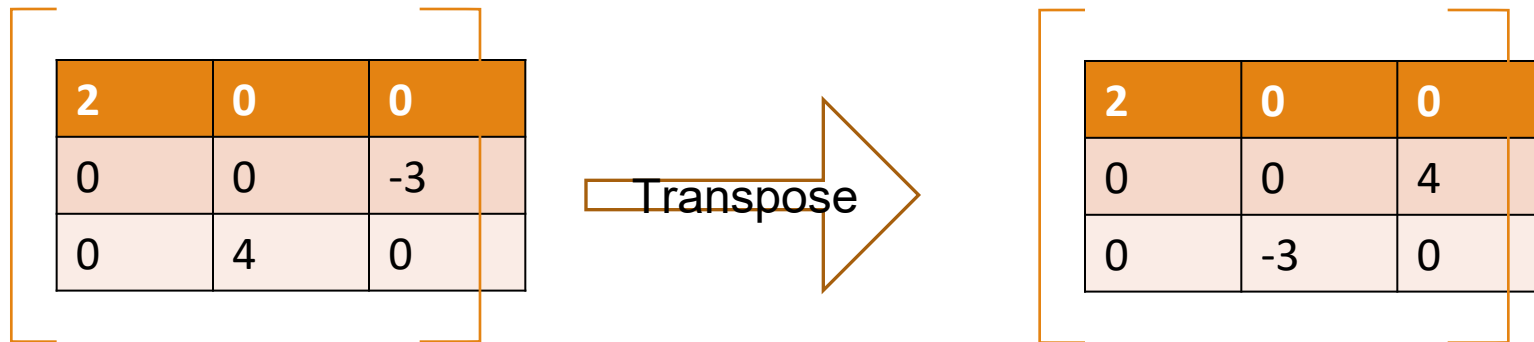


	Col0	Col1	Col2
B[0]	6	6	8
B[1]	0	0	15
B[2]	0	3	22
B[3]	0	5	-15
B[4]	1	1	11
B[5]	1	2	3
B[6]	2	3	-6
B[7]	4	0	91
B[8]	5	2	28

# Simple Transpose

Transpose is an operation which exchanges rows and columns.

For Example:

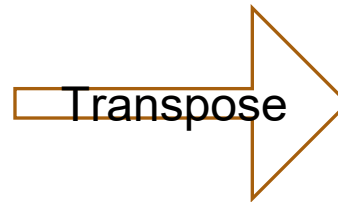


# Simple Transpose

## Transpose of Sparse Matrix

Exchanging rows and columns

3	3	3
0	0	2
1	2	-3
2	1	4




3	3	3
0	0	2
2	1	-3
1	2	4

Is this a correct matrix?

No, in a matrix values needs to stored sequentially(row & column wise)

# Simple Transpose

Now we will see how to find out simple transpose of sparse matrix.



3	3	9
0	0	2
0	1	-3
0	2	4
1	0	5
1	1	6
1	2	1
2	0	7
2	1	8
2	2	3

Transpose

1. Traverse Second column of sparse matrix

2. Search for column indexes in sequence from 0 to number of columns-1

3. Copy the rows (from 1 to number of non-zero terms) in sequence to transpose matrix.



# Simple Transpose of Sparse Matrix

	Col0	Col1	Col2			Col0	Col1	Col2
Row0	6	6	8	→	Row0	6	6	8
Row1	0	0	15	→	Row1	0	0	15
Row2	0	3	22	→	Row2	0	4	91
Row3	0	5	-15	→	Row3	1	1	11
Row4	1	1	11	→	Row4	2	1	3
Row5	1	2	3	→	Row5	2	5	28
Row6	2	3	-6	→	Row6	3	0	22
Row7	4	0	91	→	Row7	3	2	-6
Row8	5	2	28	→	Row8	5	0	-15

(a) (b)

(a) shows a sparse matrix and (b) shows its transpose stored as triples.

# Simple Transpose of Sparse Matrix

- Transpose of a matrix is obtained by interchanging rows and columns.
- In another way, we can say that element in the  $i, j$  position gets put in the  $j, i$  position.
- But this is not the case with sparse matrix triple form.
- To find transpose of sparse matrix triple form, find the all elements in  $col0$  and store then in  $row0$  of the transpose matrix, find the all elements in  $col1$  and store then in  $row1$  of the transpose matrix, and so on.
- Since the original matrix ordered the rows, the columns within each row of the transpose matrix will be arranged in ascending order as well.

# Algorithm of Simple Transpose Sparse Matrix

Algorithm Transpose(A,B)

// A is a matrix represented in sparse form. B is set to be its transpose

```

1      (m,n,t) := (A(0,0), A(0,1), A(0,2))
2      (B(0,0), B(0,1), B(0,2)) := (n,m,t)
3      if t <= 0 then return          //check for zero matrix
4      q := 1                        //q is position of next term in B
5      for col := 0 to n do          //transpose by column
6          for p := 1 to t do        //for all nonzero term do
7              if A(p,1) = col       //correct column
8                  then [(B(q,0), B(q,1), B(q,2)) := (A(p,1), A(p,0), A(p,2))
9                      q := q + 1 ]
10         end
11     end
12 end Transpose

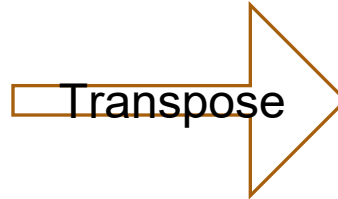
```

A[0]	6	6	8
A[1]	0	0	15
A[2]	0	3	22
A[3]	0	5	-15
A[4]	1	1	11
A[5]	1	2	3
A[6]	2	3	-6
A[7]	4	0	91
A[8]	5	2	28
	Col0	Col1	Col2
B[0]	6	6	8
B[1]	0	0	15
B[2]	0	4	91
B[3]	1	1	11
B[4]	2	1	3
B[5]	2	5	28
B[6]	3	0	22
B[7]	3	2	-6
B[8]	5	0	-15

# Fast Transpose

## Transpose of Sparse Matrix

3	3	3
0	0	2
1	2	-3
2	1	4



3	3	3
0	0	2
1	2	4
2	1	-3


Exchanging rows and columns

Is this a correct matrix?

Yes, because row and column indices are in sequence.

# Fast Transpose

Now we will see how to find out fast transpose of sparse matrix.



3	3	9
0	0	2
0	1	-3
0	2	4
1	0	5
1	1	6
1	2	1
2	0	7
2	1	8
2	2	3

1. Calculate frequency of each column value

2. Use frequency of each column value to find

location of each row in transpose matrix.

Transpose

3. Copy row containing first zero at location zero.

# Fast Transpose of Sparse Matrix

---

- Time complexity of simple transpose, is  $O(\text{columns} * \text{elements})$
- We can transpose a matrix as a sequence of triples in time  $(\text{columns} + \text{elements})$
- Fast\_Transpose proceeds by first determining the number of elements in each column of A
- This gives us the number of elements in each row of B. From this information, the starting point in B of each of its rows is easily obtained.
- We can now move the elements of A one by one into their correct position in B.

# Fast Transpose of Sparse Matrix

	Col0	Col1	Col2			Col0	Col1	Col2
Row0	6	6	8	→	Row0	6	6	8
Row1	0	0	15	→	Row1	0	0	15
Row2	0	3	22	→	Row2	0	4	91
Row3	0	5	-15	→	Row3	1	1	11
Row4	1	1	11	→	Row4	2	1	3
Row5	1	2	3	→	Row5	2	5	28
Row6	2	3	-6	→	Row6	3	0	22
Row7	4	0	91	→	Row7	3	2	-6
Row8	5	2	28	→	Row8	5	0	-15

(a) (b)

(a) shows a sparse matrix and (b) shows its transpose stored as triples.

# Algorithm of Fast Transp Sparse Matrix

procedure FAST\_TRANSPOSE(A,B)

// A is a matrix represented in sparse form. B is set to be its transpose. t is r

```

    declare S(n), T(n);           //local array
1   (m,n,t) := (A(0,0), A(0,1), A(0,2))
2   (B(0,0), B(0,1), B(0,2)) := (n,m,t)
3   if t<=0 then return //check for zero matrix
4   for i:=0 to n do S(i) :=0 end
5   for i:= 1 to t do //S(k) is the number of elements in row k of B
6       S(A(i, 1)) := S(A(i, 1)) + 1 //elements in row k of B
7   end
8   T(0):=1 //T(i) is the starting position of row i in B
9   for i:=1 to n do
10      T(i) := T(i -1 ) + S(i - 1)
11  end
12  for i:=1 to t do //move all t elements of A to B//
13      j := A(i,1) //j is the row in B//
14      (B(T(j),0), B(T(j),1), B(T(j),2)) := (A(i,1), A(i,0), A(i,2)) //store in tr
15      T(j) :=T(j) + 1 //increase row j to next spot//
16  end
17 end FAST_TRANSPOSE

```

A[0]	6	6	8
A[1]	0	0	15
A[2]	0	3	22
A[3]	0	5	-15
A[4]	1	1	11
A[5]	1	2	3
A[6]	2	3	-6
A[7]	4	0	91
A[8]	5	2	28

	Col0	Col1	Col2
Row0	6	6	8
Row1	0	0	15
Row2	0	4	91
Row3	1	1	11
Row4	2	1	3
Row5	2	5	28
Row6	3	0	22
Row7	3	2	-6
Row8	5	0	-15



# Fast Transpose

```
void fasttranspose(a,b)
```

```
{
    for (i = 1; i<= num_terms; i++)
```

```
num_cols = a[0][0];
```

```
num_term = a[0][2];
```

```
    for (i = 0; i< num_cols; i++)
```

```
        s[i] = 0;
```

```
    for (i = 1; i<= num_terms; i++)
```

```
    s[a[i][1]]++;
```

```
    T[0] = 1;
```

```
    for (i = 1; i< num_cols;i++)
```

```
    T[i] = T[i-1] + s[i-1];
```

```
{
```

```
    j = T[a[i][1]];
```

```
    b[j][0] = a[i][1];
```

```
    b[j][1] = a[i][0];
```

```
    b[j][2] = a[i][2];
```

```
    T[a[i][1]]=T[a[i][1]]++;
```

```
}
```

terms =

ng\_pos  
=

	[0]	[1]	[2]	[3]	[4]	[5]
terms =	2	1	2	2	0	1
ng_pos =	1	3	4	6	8	8

	row	col	value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	-15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	-6
[7]	4	0	91
[8]	5	2	28

# Difference between Fast and Simple Transpose

---

Time Complexity of Simple Transpose  
 $= O(n*t)$

Time Complexity of Fast Transpose  
 $= O(n+t)$



10

# Algorithm - Addition of Sparse Matrix

---

Algorithm Addition(a,b,c)

//a and b are two sparse matrices in compact form. C is the resultant sparse matrix in compact form.

if  $a(0,0) = b(0,0)$  and  $a(0,1) = b(0,1)$

    if  $t1=0$  and  $t2 = 0$  then stop

$c(0,0) = a(0,0)$  ;  $c(0,1) = a(0,1)$  ;

End

$i = j = k = 1$

while (  $i \leq t1$  and  $j \leq t2$  )

    Case A :  $a(i,0) = b(j,0)$  // row position

        Case 1 :  $a(i,1) = b(j,1)$  //col position

$temp = a(i,2) + b(j,2)$

            if  $temp \neq 0$

$c(k,0) = a(i,0)$

$c(k,1) = a(i,1)$

$c(k,2) = temp$

$k = k+1$

            end if

$i = i + 1; j = j + 1;$

# Algorithm - Addition of Sparse Matrix

Case 2 : if  $a(i, 1) < b(j, 1)$

$c(k, 0) = a(i, 0)$

$c(k, 1) = a(i, 1)$

$c(k, 2) = a(i, 2)$

$k = k + 1$

$i = i + 1$

Case 3 : if  $a(i, 1) > b(j, 1)$

$c(k, 0) = b(j, 0)$

$c(k, 1) = b(j, 1)$

$c(k, 2) = b(j, 2)$

$k = k + 1$

$j = j + 1$

Case B : if  $a(i, 0) < b(j, 0)$

$c(k, 0) = a(i, 0)$

$c(k, 1) = a(i, 1)$

$c(k, 2) = a(i, 2)$

$k = k + 1$

$i = i + 1$

# Algorithm - Addition of Sparse Matrix

---

Case C : if  $a(i, 0) > b(j, 0)$

$c(k, 0) = b(j, 0)$

$c(k, 1) = b(j, 1)$

$c(k, 2) = b(j, 2)$

$k = k + 1$

$j = j + 1$

end while

while ( $i \leq t1$ )

{

$c := a$

$i++$ ;  $k++$ ;

}

while ( $j \leq t2$ )

{

$c := b$

$j++$ ;  $k++$ ;

}

$c(0, 2) = k - 1$

**end Addition**

# FAQ

---

1. What is sparse matrix? List the applications?

2. Represent sparse matrix with suitable data structures?  
Explain with example simple and fast transpose?

3. Explain sequential memory organization with suitable example?

4. Explain row major and column major representation of a matrix? Show address calculation with example?

# FAQ

Find out the addition of two sparse matrices in triplet form and also find Simple and Fast transpose?

4	5	6
0	3	7
0	4	6
1	4	4
2	1	8
3	2	45
4	4	21

4	5	6
0	3	5
1	3	8
1	4	45
2	3	4
3	2	45
4	1	2

Represent the following polynomial using array:

$$3x^3y^2 + 10xy^4 + 10x + 1$$

$$21 + x^4 - 5x^7 + 18x^6$$

$$5x^2 + 10xy + y^2 + 20$$

$$5x^3y^2z + 3x^2y^3z^2 + 6xyz^3 + 10$$

$$10x^3y^2z - 8x^3y^3z^3 + 5xyz^2 + 10$$



# FAQ

---

Each element of an array `Data[20][50]` requires 4 bytes of storage. Base address of data is 2000. determine the location of `Data[10][10]` when the array is stored as i) row major and ii) column major?

Consider the liner array `A(5:50)`. `BA=300` and the number of words per memory cell is 4 bytes. Find address of `A[15]`?

Consider `int arr[4][5]`. `BA=1020`. Find the address of `arr[3][4]` with row-major and column major representation?

Consider `int arr[5][4]`. `BA=510`. Find the address of `arr[3][2]` with row-major and column major representation?

# FAQ

---

7. Each element of an array `Data[20][50]` requires 4 bytes of storage. Base address of data is 2000. determine the location of `Data[10][10]` when the array is stored as i) row major and ii) column major?

8. Consider the liner array `A(5:50)`. `BA=300` and the number of words per memory cell is 4 bytes. Find address of `A[15]`?

9. Consider `int arr[4][5]`. `BA=1020`. Find the address of `arr[3][4]` with row-major and column major representation?

10. Consider `int arr[5][4]`. `BA=510`. Find the address of `arr[3][2]` with row-major and column major representation?

# Examples

---

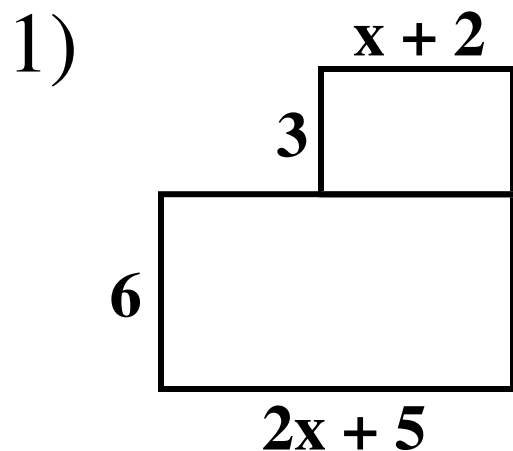
A farmer must add the areas of two plots of land to determine the amount of seed to plant. The area of plot A can be represented by  $3x^2 + 7x - 5$ , and the area of plot B can be represented by  $5x^2 - 4x + 11$ . Write a polynomial that represents the total area of both plots of land.

$$\begin{array}{r} (3x^2 + 7x - 5) \\ + (5x^2 - 4x + 11) \\ \hline 8x^2 + 3x + 6 \end{array}$$

*Plot A.*

*Plot B.*

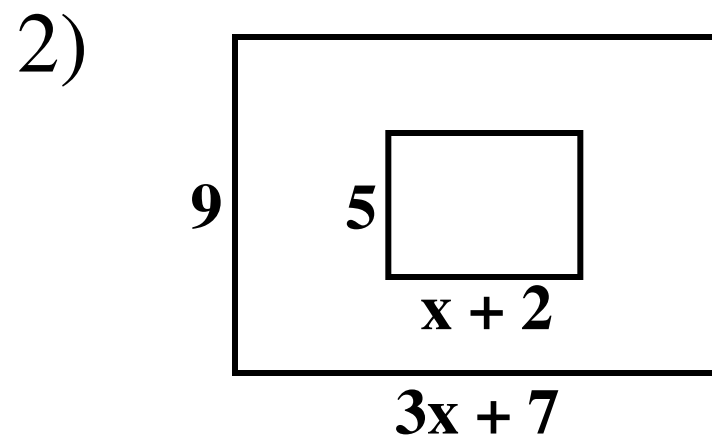
*Combine like terms.*



$$6(2x + 5) + 3(x + 2)$$

$$12x + 30 + 3x + 6$$

$$15x + 36$$

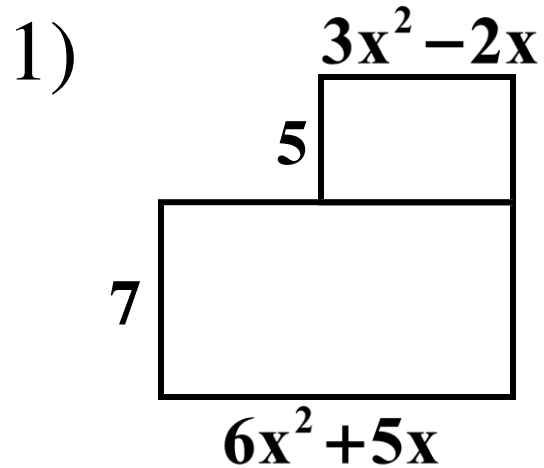


$$9(3x + 7) - 5(x + 2)$$

$$27x + 63 - 5x - 10$$

$$22x + 53$$

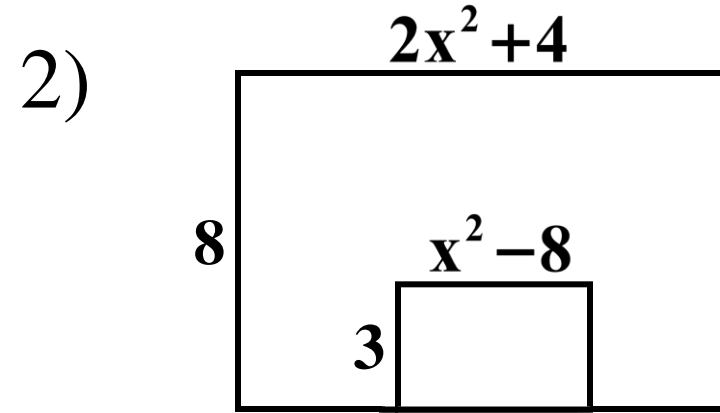
Write an expression that represents the area of the shaded region in terms of  $x$ .



$$7(6x^2 + 5x) + 5(3x^2 - 2x)$$

$$42x^2 + 35x + 15x^2 - 10x$$

$$57x^2 + 25x$$



$$8(2x^2 + 4) - 3(x^2 - 8)$$

$$16x^2 + 32 - 3x^2 + 24$$

$$13x^2 + 56$$

Add or subtract.

1.  $7m^2 + 3m + 4m^2$      $11m^2 + 3m$

2.  $(r^2 + s^2) - (5r^2 + 4s^2)$      $-4r^2 - 3s^2$

3.  $(10pq + 3p) + (2pq - 5p + 6pq)$      $18pq - 2p$

4.  $(14d^2 - 8) + (6d^2 - 2d + 1)$      $20d^2 - 2d - 7$

5.  $(2.5ab + 14b) - (-1.5ab + 4b)$      $4ab + 10b$

6. A painter must add the areas of two walls to determine the amount of paint needed. The area of the first wall is modeled by  $4x^2 + 12x + 9$ , and the area of the second wall is modeled by  $36x^2 - 12x + 1$ . Write a polynomial that represents the total area of the two walls.

$$40x^2 + 10$$

# *Practice Assignments*

---

- Implement Array as an Abstract Data Type using Sequential Organization?
- Representation of Polynomials using arrays and perform operations
  - Addition of two polynomials
  - Evaluation of Polynomial



# *Takeaway*

---

- Study the concept of Sequential Organization and its Memory Representation and Address Calculation.
- Learn Representation and operations of polynomial such as Addition and Evaluation.
- Study and implement sparse matrix concept and operations such as Simple Transpose, Fast Transpose and Addition

# *References*

---

1. E. Horowitz, S. Sahani, S. Anderson-Freed, "Fundamentals of Data Structures in C", Universities Press, 2008
2. Treamblay, Sorenson, "An introduction to data structures with applications", Tata McGraw Hill, Second Edition
3. Aaron Tanenbaum, "Data Structures using C", Pearson Education
4. R. Gilberg, B. Forouzan, "Data Structures: A pseudo code approach with C", Cenage Learning, ISBN 9788131503140