**Arrays**

**Ques1. Explain the term arrays as a linear data structure?**

Ans. An array is a collection of homogeneous or similar data elements stored in contiguous memory locations.

**Ques2. What are sparse matrices? Give an example.**

Ans. When the number of zero elements in a matrix is more than the number of non-zero elements it is called a sparse matrix. Example - .

**Ques3. How 2-Dimensional arrays are represented in memory? Also obtain the formula for calculating the address of any element stored in the array, in case of column major?**

Ans. 2-Dimensional matrices are represented in either row major or column major order. To calculate the address of an element in column major format, the formula is:

Where L0 = Base address

m= number of rows

n= number of columns

W= size of each element

**Add= L­0 + [ m\*j + i ] \* W**

**Ques4. Derive the formula to find physical address of an element in 3-Dimensional arrays stored in row major order?**

Ans. In 3-dimensional matrix the Physical address **P** of an element at position **(i,j,k)** can be given by **P= W \* (((i\*n2)+j)\*n3+k)**

Where W= size of each element

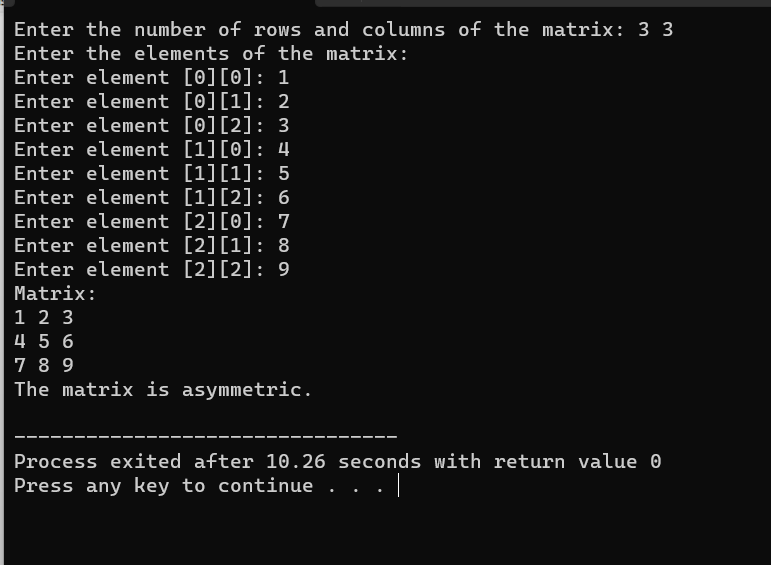
N1, N2, N3 are the dimensions of arrays in 1st 2nd and 3rd dimensions respectively.

i, j,k are the indices of element in 1st 2nd and 3rd dimensions respectively.

**Ques5. Write a program to enter a matrix and determine if it is an asymmetrical matrix?**

Ans.

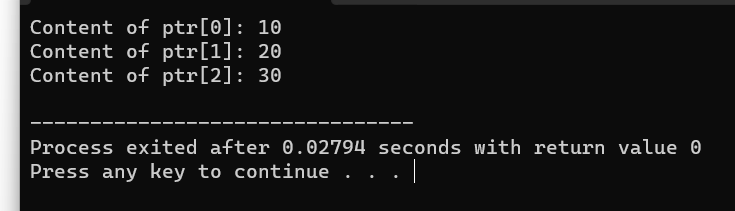
1. #include <stdio.h>
2. #define MAX\_ROWS 10
3. #define MAX\_COLS 10
4. void inputMatrix(int matrix[MAX\_ROWS][MAX\_COLS], int rows, int cols) {
5. printf("Enter the elements of the matrix:\n");
6. for (int i = 0; i < rows; i++) {
7. for (int j = 0; j < cols; j++) {
8. printf("Enter element [%d][%d]: ", i, j);
9. scanf("%d", &matrix[i][j]);
10. }
11. }
12. }
13. void printMatrix(int matrix[MAX\_ROWS][MAX\_COLS], int rows, int cols) {
14. printf("Matrix:\n");
15. for (int i = 0; i < rows; i++) {
16. for (int j = 0; j < cols; j++) {
17. printf("%d ", matrix[i][j]);
18. }
19. printf("\n");
20. }
21. }
22. int isAsymmetric(int matrix[MAX\_ROWS][MAX\_COLS], int rows, int cols) {
23. int transpose[MAX\_ROWS][MAX\_COLS];
24. for (int i = 0; i < rows; i++) {
25. for (int j = 0; j < cols; j++) {
26. transpose[j][i] = matrix[i][j];
27. }
28. }
29. for (int i = 0; i < rows; i++) {
30. for (int j = 0; j < cols; j++) {
31. if (matrix[i][j] != transpose[i][j]) {
32. return 1;
33. }
34. }
35. }
36. return 0;
37. }
38. int main() {
39. int matrix[MAX\_ROWS][MAX\_COLS];
40. int rows, cols;
41. printf("Enter the number of rows and columns of the matrix: ");
42. scanf("%d %d", &rows, &cols);
43. if (rows <= 0 || rows > MAX\_ROWS || cols <= 0 || cols > MAX\_COLS) {
44. printf("Invalid matrix dimensions.\n");
45. return 1;
46. }
47. inputMatrix(matrix, rows, cols);
48. printMatrix(matrix, rows, cols);
49. if (isAsymmetric(matrix, rows, cols)) {
50. printf("The matrix is asymmetric.\n");
51. } else {
52. printf("The matrix is symmetric.\n");
53. }
54. return 0;
55. }

****

**Ques6. Write a C program to display the content of pointers using array of pointers?**

Ans.

1. #include <stdio.h>
2. int main() {
3. int num1 = 10, num2 = 20, num3 = 30;
4. int \*ptr[3]= {&num1,&num2,&num3};
5. for (int i = 0; i < 3; i++) {
6. printf("Content of ptr[%d]: %d\n", i, \*ptr[i]);
7. }
8. return 0;
9. }



**Ques7. Write an algorithm to count the common elements of two one dimensional arrays?**

Ans.

1. Start

2. Initialize variables:

count = 0 (to count the common elements)

n1 = length of the first array

n2 = length of the second array

3. Read elements of the first array (arr1)

4. Read elements of the second array (arr2)

5. Loop through each element of arr1:

a. Loop through each element of arr2:

i. If arr1[i] is equal to arr2[j], increment count and break the inner loop

6. Display the value of count as the count of common elements

7. Stop

**Ques8. Write a short note on**

1. **Static and Dynamic memory allocation**

Ans. In static memory allocation, memory for variables is allocated at compile time and remains fixed throughout the program’s execution. This means that the size of memory block needed for each variable is determined before the program runs and does not change during runtime.

Dynamic memory allocation allows programs to allocate memory during runtime. It allows creation of variables whose size is not known until the program is executed. This flexibility is crucial for handling data structures like arrays and linked lists whose size may vary based on user input and other factors.

1. **Arrays of pointers**

Ans. An array of pointers is a data structure in which each element of the array holds memory address of another variable or data structure.

**Ques9. Explain the memory representation of two-dimensional arrays with the help of examples.**

Ans. In memory a two-dimensional array is represented as a contiguous block of memory, with elements arranged in rows and columns. It can be represented in either row major or column major order.

1. **Row major format**

In row major format, elements of each row are stored in contiguous memory locations. Example: 2x3 array

int arr[2][3] = { {1, 2, 3},

{4, 5, 6} };

| arr | 1 | 2 | 3 | 4 | 5 | 6 |<-- Elements of the array

1. **Column major format**

In column major format, elements of each column are stored in contiguous memory locations.

Example: 2x3 array

int arr[2][3] ={{1}{2}{3}

{4}{5}{6}};

|arr| 1 | 4 | 2 | 5 | 3 | 6 | 🡨 Elements of the array

**Q. 10. Draw a linked list and vector representation of the following sparse matrix:--**

**0 12 0 0**

**0 11 0 0**

**9 0 0 0**

**0 0 0 0**

Ans. Linked list representation:

(0,12,0,0) -> (0,11,0,0) -> (9,0,0,0)-> (0,0,0,0)

Vector representation:

[ (0,12,0,0), (0,11,0,0), (9,0,0,0), (0,0,0)]

**Ques11. Explain the application areas of the array with a suitable example.**

Ans. Data Structures: Arrays are widely used as the underlying data structure for implementing other data structures such as stacks, queues, heaps, and hash tables.

Algorithms: Arrays are essential for implementing and manipulating data in algorithms. Many sorting, searching, and graph algorithms rely on arrays for data storage and manipulation.

Dynamic Memory Allocation: Arrays are used in dynamic memory allocation to represent resizable data structures such as dynamic arrays, lists, and resizable buffers.

Multidimensional Data: Arrays are used to represent multidimensional data structures such as matrices, grids, and images.

**Ques12. Write an algorithm for matrix operations such as the addition of two matrices, multiplication, subtraction, and transpose.**

Ans. **1. Addition of matrices**

Algorithm: MatrixAddition(A, B, C)

Input: Matrices A and B of dimensions m x n

Output: Matrix C containing the sum of matrices A and B

Step 1: Initialize matrix C of dimensions m x n.

Step 2: for i = 1 to m do

for j = 1 to n do

C[i][j] = A[i][j] + B[i][j]

end for

end for

Step 3: Return matrix C.

**2. Multiplication of matrices**

Algorithm: MatrixMultiplication(A, B, C)

Input: Matrices A and B of dimensions m x n and n x p respectively

Output: Matrix C containing the product of matrices A and B

Step 1: Initialize matrix C of dimensions m x p.

Step 2: for i = 1 to m do

for j = 1 to p do

C[i][j] = 0

for k = 1 to n do

C[i][j] += A[i][k] \* B[k][j]

end for

end for

end for

Step 3: Return matrix C**.**

**3. Subtraction of two matrices**

Algorithm: MatrixSubtraction(A, B, C)

Input: Matrices A and B of dimensions m x n

Output: Matrix C containing the difference of matrices A and B

Step 1: Initialize matrix C of dimensions m x n.

Step 2: for i = 1 to m do

for j = 1 to n do

C[i][j] = A[i][j] - B[i][j]

end for

end for

Step 3: Return matrix C.

**4. Transpose of a Matrix**

Algorithm: MatrixTranspose(A, B)

Input: Matrix A of dimensions m x n

Output: Matrix B containing the transpose of matrix A

Step 1: Initialize matrix B of dimensions n x m.

Step 2: for i = 1 to m do

for j = 1 to n do

B[j][i] = A[i][j]

end for

end for

Step 3: Return matrix B.

**Ques14. Write and explain the algorithm to find the 7th smallest element in an array.**

Ans.

Algorithm: FindSeventhSmallestElement(arr)

Input: Array arr of size n

Output: The 7th smallest element in the array

Step 1: Check if the length of the array is less than 7. If so, return an error message indicating that the array is too small.

Step 2: Sort the array in non-decreasing order using any sorting algorithm, such as Quick Sort or Merge Sort.

Step 3: Retrieve the element at index 6 (since array indices start from 0, the 7th smallest element will be at index 6).

Step 4: Return the element obtained in Step 3 as the 7th smallest element.

Explanation:

1. We first check if the array has at least 7 elements. If it has fewer than 7 elements, we cannot find the 7th smallest element, so we return an error message.

2. Next, we sort the array in non-decreasing order. Sorting the array ensures that the smallest elements are towards the beginning of the array.

3. Once the array is sorted, we retrieve the element at index 6. Since array indices start from 0, the 7th smallest element will be at index 6.

4. Finally, we return the element obtained in Step 3 as the 7th smallest element in the array.

**Ques15. Write a program in C to create an array of 10 elements. Take 10 elements in the array pass them into a function that prints them in reverse order.**

Ans.

1. #include <stdio.h>
2. void printReverse(int arr[], int size) {
3. printf("Elements in reverse order:\n");
4. for (int i = size - 1; i >= 0; i--) {
5. printf("%d ", arr[i]);
6. }
7. printf("\n");
8. }
9. int main() {
10. int arr[10];
11. printf("Enter 10 elements:\n");
12. for (int i = 0; i < 10; i++) {
13. scanf("%d", &arr[i]);
14. }
15. printReverse(arr, 10);
16. return 0;
17. }

**Ques16. You have a row of binary digits arranged randomly. Arrange them in such an order that all 0’s precede all 1’s or vice versa. The only constraint in arranging them is that you are allowed to interchange the positions of binary digits if they are not similar. Write a program in C using the pointer to solve the problem. Eg: 11010100 ◊ 00001111 OR 11110000**

Ans.

1. #include <stdio.h>
2. void swap(int \*a, int \*b) {
3. int temp = \*a;
4. \*a = \*b;
5. \*b = temp;
6. }
7. void rearrangeBinary(int \*arr, int n) {
8. int i = 0, j = n - 1;
9. while (i < j) {
10. while (arr[i] == 0 && i < j) {
11. i++;
12. }
13. while (arr[j] == 1 && i < j) {
14. j--;
15. }
16. if (i < j) {
17. swap(&arr[i], &arr[j]);
18. i++;
19. j--;
20. }
21. }
22. }
23. void printArray(int \*arr, int n) {
24. printf("Arranged binary digits: ");
25. for (int i = 0; i < n; i++) {
26. printf("%d", arr[i]);
27. }
28. printf("\n");
29. }
30. int main() {
31. int n;
32. printf("Enter the number of binary digits: ");
33. scanf("%d", &n);
34. int arr[n];
35. printf("Enter the binary digits (0s and 1s): ");
36. for (int i = 0; i < n; i++) {
37. scanf("%d", &arr[i]);
38. }
39. rearrangeBinary(arr, n);
40. printArray(arr, n);
41. return 0;
42. }

**Ques17. GAME OF CHOCOLATES**