THEORY – 3

PROGRAM -1

AIM- ARRAY OPERATIONS

|  |
| --- |
| Array operations refer to various actions that can be performed on arrays, which are data structures that store a fixed-size sequential collection of elements of the same type. Common array operations include initializing an array, accessing elements, modifying elements, searching for elements, inserting elements, deleting elements, and sorting elements. Below are detailed descriptions of common array operations:  1. \*\*Initializing an Array\*\*:  - Arrays can be initialized either statically (at compile-time) or dynamically (at run-time).  - Static initialization involves specifying the size of the array and initializing each element explicitly.  - Dynamic initialization involves creating an array with a specified size using memory allocation functions.  Example of static initialization:  ```c  int arr[5] = {10, 20, 30, 40, 50}; // Array with 5 elements initialized  ```  2. \*\*Accessing Elements\*\*:  - Elements in an array can be accessed using their indices.  - Array indices are zero-based, meaning the first element is at index 0, the second element is at index 1, and so on.  Example of accessing elements:  ```c  int value = arr[2]; // Access the element at index 2 (third element)  ```  3. \*\*Modifying Elements\*\*:  - Elements in an array can be modified by assigning new values to them using their indices.  Example of modifying elements:  ```c  arr[1] = 100; // Change the value of the second element to 100  ```  4. \*\*Searching for Elements\*\*:  - Searching for an element in an array involves iterating through the array to find the index of a specific value.  - Common search algorithms include linear search (sequential search) and binary search (for sorted arrays).  Example of linear search:  ```c  int search(int arr[], int size, int key) {  for (int i = 0; i < size; i++) {  if (arr[i] == key) {  return i; // Return index of the key if found  }  }  return -1; // Return -1 if key is not found  }  ```  5. \*\*Inserting Elements\*\*:  - Inserting an element into an array involves shifting existing elements to make space for the new element.  - Insertion can occur at the beginning, end, or any specific position in the array.  Example of inserting an element at the end:  ```c  void insertAtEnd(int arr[], int \*size, int value) {  arr[\*size] = value; // Assign value to the end of the array  (\*size)++; // Increment the size of the array  }  ```  6. \*\*Deleting Elements\*\*:  - Deleting an element from an array involves shifting remaining elements to close the gap left by the deleted element.  - Deletion can occur from the beginning, end, or any specific position in the array.  Example of deleting an element at a specific position:  ```c  void deleteAtIndex(int arr[], int \*size, int index) {  if (index < 0 || index >= \*size) {  printf("Invalid index\n");  return;  }  for (int i = index; i < \*size - 1; i++) {  arr[i] = arr[i + 1]; // Shift elements to the left  }  (\*size)--; // Decrement the size of the array  }  ```  7. \*\*Sorting Elements\*\*:  - Sorting elements in an array arranges them in either ascending or descending order.  - Common sorting algorithms include bubble sort, selection sort, insertion sort, merge sort, quick sort, etc.  Example of sorting elements using bubble sort:  ```c  void bubbleSort(int arr[], int size) {  for (int i = 0; i < size - 1; i++) {  for (int j = 0; j < size - i - 1; j++) {  if (arr[j] > arr[j + 1]) {  int temp = arr[j];  arr[j] = arr[j + 1];  arr[j + 1] = temp;  }  }  }  }  These are fundamental array operations that are widely used in programming to manipulate and process data stored in arrays efficiently. The choice of operation depends on the specific requirements and constraints of the problem being solved. |