DSA Lab

**Assignment 2**

**Name:** Madhav Jha

**Roll no.:** 48

**Branch:** CSE (AI & ML)

# ADT Array

## AIM:

Use ADT Array with Structures and perform following operations

1. Create an array using dynamic memory allocation function  
   2. Insert an element in an array at specific position.   
   3. Insert an element at the end of an array   
   4. delete an element from array   
   5.  Find maximum number  
   5.  Find minimum number  
   6.  Linear search and binary search  
   7.  Merging to array  
   8.  Array intersection   
   9. Sum up all the elements in the array  
   10. Average all the elements of the array  
   11. Reverse all the elements of the array (Using new array)  
   12. Reversing array using original array only  
   13. Insert element in a sorted array  
   14. Check if array is sorted or not

## Code:

#include <stdio.h>

#include <stdlib.h>

void error(char e[]) {

    printf("\nERROR: %s!!\n", e);

}

typedef struct

{

    int\* arr;

    int maxLen, i;

}Array;

void printArray(Array\* a) {

    printf("[ ");

    for (int j = 0;j < (a->i);j++) {

        printf("%d ", (a->arr)[j]);

    }

    printf("]\n");

}

// initiate the Array object

Array\* initializeArray(int n) {

    Array\* a;

    a = (Array\*)malloc(sizeof(Array));

    a->maxLen = n;

    a->i = 0;

    a->arr = (int\*)calloc(n, sizeof(int));

    return a;

}

// increase the size of array

void expandArray(Array\* a) {

    a->maxLen \*= 2;

    a->arr = (int\*)realloc(a->arr, (a->maxLen) \* sizeof(int));

}

// add element at the end of the array

void push\_back(Array\* a, int num) {

    if (a->i >= a->maxLen - 1) {

        expandArray(a);

    }

    (a->arr)[a->i] = num;

    (a->i)++;

}

// return current length of the Array

int len(Array\* a) {

    return a->i;

}

// insert at index

void insertAt(Array\* a, int index, int num) {

    if (index < 0 || index >= a->i) {

        error("index out of bound");

        return;

    }

    (a->maxLen)++;

    for (int i = a->i;i > index;i--) {

        (a->arr)[i] = (a->arr)[i - 1];

    }

    (a->arr)[index] = num;

    (a->i)++;

}

int linerSearch(Array\* a, int num) {

    int index = -1;

    for (int i = 0;i < a->i;i++) {

        if ((a->arr)[i] == num) {

            index = i;

            break;

        }

    }

    return index;

}

void delete(Array\* a, int num) {

    int index = linerSearch(a, num);

    if (index == -1) {

        error("Num not found");

        return;

    }

    for (int i = index;i < a->i;i++) {

        (a->arr)[i] = (a->arr)[i + 1];

    }

    (a->i)--;

}

int minElem(Array\* a) {

    int min = INT\_MAX;

    for (int i = 0;i < a->i;i++) {

        min = ((a->arr)[i] < min) ? (a->arr)[i] : min;

    }

    return min;

}

int maxElem(Array\* a) {

    int max = -1 \* INT\_MAX;

    for (int i = 0;i < a->i;i++) {

        max = ((a->arr)[i] > max) ? (a->arr)[i] : max;

    }

    return max;

}

int sum(Array\* a) {

    int s = 0;

    for (int i = 0;i < a->i;i++) {

        s += (a->arr)[i];

    }

    return s;

}

double avg(Array\* a) {

    int s = sum(a);

    return (double)s / (double)(a->i);

}

void reverseNew(Array\* a) {

    int\* t;

    t = (int\*)calloc((a->maxLen), sizeof(int));

    for (int i = (a->i) - 1, j = 0;i >= 0;i--, j++) {

        t[j] = (a->arr)[i];

    }

    free(a->arr);

    a->arr = t;

}

void reverse(Array\* a) {

    int n = a->i, temp = 0;

    for (int i = 0;i < n / 2;i++) {

        temp = (a->arr)[i];

        (a->arr)[i] = (a->arr)[n - 1 - i];

        (a->arr)[n - 1 - i] = temp;

    }

}

void bubbleSort(Array\* a) {

    int n = a->i;

    for (int i = 0;i < n;i++) {

        for (int j = 0;j < n;j++) {

            if (a->arr[i] < a->arr[j]) {

                int temp = a->arr[i];

                a->arr[i] = a->arr[j];

                a->arr[j] = temp;

            }

        }

    }

}

int binarySearch(Array\* a, int num) {

    // sort

    bubbleSort(a);

    int l = 0, r = a->i - 1, m, index = -1;

    int\* arr;

    arr = a->arr;

    while (l <= r) {

        m = (l + r) / 2;

        if (arr[m] == num) {

            index = m;

            break;

        }

        else if (arr[m] < num) {

            l = m + 1;

        }

        else {

            r = m;

        }

    }

    return index;

}

int isSorted(Array\* a) {

    int flag = 1;

    for (int i = 0;i < a->i - 1;i++) {

        if ((a->arr)[i] > (a->arr)[i + 1]) {

            flag = 0;

            break;

        }

    }

    return flag;

}

void mergeArr() {

    int a1 = 2;

    printf("\nEnter number of elements in array 1: ");

    scanf("%d", &a1);

    int\* arr1, \* arr2, \* arr3;

    arr1 = (int\*)calloc(a1, sizeof(int));

    printf("Enter array elements: ");

    for (int i = 0;i < a1;i++) {

        scanf("%d", &arr1[i]);

    }

    int a2 = 2;

    printf("\nEnter number of elements in array 2: ");

    scanf("%d", &a2);

    arr2 = (int\*)calloc(a2, sizeof(int));

    printf("Enter array elements: ");

    for (int i = 0;i < a2;i++) {

        scanf("%d", &arr2[i]);

    }

    int a3 = a1 + a2;

    arr3 = (int\*)calloc(a3, sizeof(int));

    for (int i = 0; i < a1; i++)

    {

        arr3[i] = arr1[i];

    }

    for (int i = 0; i < a2; i++)

    {

        arr3[a1 + i] = arr2[i];

    }

    printf("\nArray after merge: ");

    printf("[ ");

    for (int j = 0;j < a3;j++) {

        printf("%d ", arr3[j]);

    }

    printf("]\n");

}

void intersection(Array\* a) {

    int a1 = 2;

    printf("\nEnter number of elements in array: ");

    scanf("%d", &a1);

    int arr1[a1];

    printf("Enter array elements: ");

    for (int i = 0;i < a1;i++) {

        scanf("%d", &arr1[i]);

    }

    printf("\nIntersection: ");

    for (int i = 0;i < a1;i++) {

        int index = linerSearch(a, arr1[i]);

        if (index != -1) {

            printf("%d ", arr1[i]);

        }

    }

    printf("\n");

}

int main() {

    printf("\n1.Create a dynamic array\n");

    int n = 2;

    printf("\nEnter number of elements: ");

    scanf("%d", &n);

    Array\* a;

    a = initializeArray(n);

    printf("Enter array elements: ");

    for (int i = 0;i < n;i++) {

        int t;

        scanf("%d", &t);

        push\_back(a, t);

    }

    printf("Array: ");

    printArray(a);

    printf("\n2.Insert element(40) at an index(1) of the array\n");

    insertAt(a, 1, 40);

    printArray(a);

    printf("\n3.Insert element(50) at end of the array\n");

    push\_back(a, 50);

    printArray(a);

    printf("\n4.Delete an element(50)\n");

    delete(a, 50);

    printArray(a);

    printf("\n5.1.Maximum element: %d\n", maxElem(a));

    printf("5.2.Minimum element: %d\n", minElem(a));

    printf("\n6.1.Linear search element(40): %d\n", linerSearch(a, 40));

    printf("6.2.Binary search element(40): %d\n", binarySearch(a, 40));

    printf("\n7.Merging two arrays\n");

    mergeArr();

    printf("\n8.Intersection between arrays\n");

    intersection(a);

    printf("\n9.Sum of array: %d\n", sum(a));

    printf("\n10.Average of array: %f\n", avg(a));

    printf("\n11.Reverse of array (using new array): ");

    reverseNew(a);

    printArray(a);

    printf("\n12.Reverse of array (using original array): ");

    reverse(a);

    printArray(a);

    printf("\n13.Insert element(50) in a sorted array at the end\n");

    printf("Before sort: ");

    printArray(a);

    printf("After sort: ");

    bubbleSort(a);

    printArray(a);

    printf("After Adding element: ");

    push\_back(a, 50);

    printArray(a);

    printf("\n14.Is the array sorted: %s\n", ((isSorted(a) == 1 ? "Yes" : "No")));

    printf("\n");

    free(a);

    return 0;

}

## Output:

