**Batch: C3 Roll No.: 16010123217**

**Experiment No. 1**

**Grade: AA / AB / BB / BC / CC / CD /DD**

|  |
| --- |
| **Title:**  Implementation of Abstract Data Type |

**Objective:** Implementation of ADT without using any standard library function

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| **CO 1** | Explain the different data structures used in problem solving. |

**Books/ Journals/ Websites referred:**

**Abstract**:-

*(Define ADT. Why are they important in data structures?)*

*Ans.*

* *Abstract Data Types (ADTs) stores data and allow various operations on the data to access and change it.*
* *A mathematical model, together with various operations defined on the model*
* *An ADT is a collection of data and associated operations for manipulating that data*
* ADTs support abstraction, encapsulation, and information hiding.
* Abstraction is the structuring of a problem into well-defined entities by defining their data and operations.
* The principle of hiding the used data structure and to only provide a well-defined interface is known as encapsulation.

**Abstract Data Type for Array**

*[For the assigned data type, write value definition and operator definition)*

***Value Definition:***

Abstract typedef ArrayType <<ElementType ele>>

Precondition : none

**Operator Definitions:**

1. **CreateArray**

Abstract ArrayType CreateArray <ElementType ele, Integer Size>

Precondition: memory of size should be available

Postcondition: CreateArray: An array of the specified size and ElementType is created.

1. **AddToArrayAtPos**

Abstract ArrayType AddToArrayAtPos <ArrayType A, ElementType ele, Integer Pos>

Precondition: !full(A)

Postcondition: AddToArrayAtPos: A = A + ele with ele at position pos in A.

1. **DeleteFromArray**

Abstract ElementType DeleteFromArray <ArrayType A, ElementType ele>

Precondition: !empty(A)

Postcondition: DeleteFromArray: Ele = A – ele

1. **CountElementsOfArray**

Abstract Integer CountElementsOfArray <ArrayType A>

Precondition: A is not null

Postcondition: CountElementsOfArray: Returns the number of elements present in the array A.

1. **SumofArray**

Abstract ElementType SumofArray <ArrayType A>

Precondition: A is not null and elements of A support addition

Postcondition: SumofArray: Returns the sum of all elements in the array A.

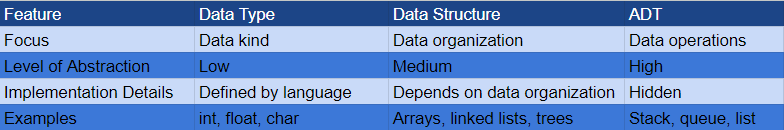
**Post lab questions:**

1. Discuss the advantages of list

Ans.

* A linked list is a dynamic data structure that can expand and shrink in size at runtime by allocating and deallocating memory. As a result, there is no need to specify the linked list's initial size.
* There is no memory wastage in the linked list because the size of the linked list increases or decreases at run time. Hence there is no memory wastage and no need to pre-allocate memory.
* Implementation: A connected list is often used to implement linear data structures such as stacks and queues.
* Service of insertion and deletion: The connected list makes insertion and deletion far simpler. After an element is inserted or deleted, there is no need to shift it; the address in the next pointer only needs to be changed.

2. Compare and contrast between ADT, Data types and Data structure.

Ans. 

3. Define 5 ADT functions for lists

Ans.

**1. isEmpty()**

Purpose: Checks if the list is empty.

Returns: A boolean value, True if the list is empty, False otherwise.

**2. size()**

Purpose: Returns the number of elements in the list.

Returns: An integer representing the size of the list.

**3. insert(index, element)**

Purpose: Inserts an element at a specific index in the list.

Parameters:

index: The position where the element should be inserted.

element: The value to be inserted.

Returns: None or a boolean indicating success/failure.

**4. remove(index)**

Purpose: Removes the element at a specific index from the list.

Parameters:

index: The index of the element to be removed.

Returns: The removed element or None if the index is out of bounds.

**5. get(index)**

Purpose: Returns the element at a specific index in the list.

Parameters:

index: The index of the element to be retrieved.

Returns: The element at the specified index.

**Program code and Output screenshots:**

**Program Code:**

#include<stdio.h>

#include<stdlib.h>

// Structure to represent an array with dynamic size

struct Array {

int capacity; // Maximum capacity of the array

int used\_size; // Current number of elements in the array

int \*ptr; // Pointer to the array elements

};

// Function to create an array with a given initial size

void createArray(struct Array \*a, int uSize) {

(\*a).capacity = 100; // Set the capacity to 100

(\*a).used\_size = uSize; // Set the initial used size

(\*a).ptr = (int \*)malloc(100 \* sizeof(int)); // Allocate memory for 100 integers

}

// Function to display the elements of the array

void display(struct Array \*a) {

for (int i = 0; i < (\*a).used\_size; i++) {

printf("%d ", (\*a).ptr[i]);

}

printf("\n");

}

// Function to set values in the array

void setVal(struct Array \*a) {

int n;

for (int i = 0; i < (\*a).used\_size; i++) {

printf("Enter element at position %d: ", i + 1);

scanf("%d", &n);

(\*a).ptr[i] = n;

}

}

// Function to add an element to the array at a specified position

void AddElementToArrayAtPos(struct Array \*a, int element, int index) {

if (index > (\*a).used\_size || index < 0) { // Check if the index is valid

printf("Invalid index\n");

return;

}

if ((\*a).used\_size == (\*a).capacity) { // Check if the array is full

printf("Array is full\n");

return;

}

for (int i = (\*a).used\_size; i > index; i--) { // Shift elements to the right

(\*a).ptr[i] = (\*a).ptr[i - 1];

}

(\*a).ptr[index] = element; // Insert the new element

(\*a).used\_size++; // Increase the used size

}

// Function to delete the first occurrence of an element from the array

void DeleteFromArray(struct Array \*a, int element) {

int found = 0;

for (int i = 0; i < (\*a).used\_size; i++) {

if ((\*a).ptr[i] == element) {

found = 1; // Mark the element as found

}

if (found && i < (\*a).used\_size - 1) {

(\*a).ptr[i] = (\*a).ptr[i + 1]; // Shift elements to the left

}

}

if (found) {

(\*a).used\_size--; // Decrease the used size if the element was found

} else {

printf("Element not found\n");

}

}

// Function to count the number of elements in the array

int CountElementsOfArray(struct Array \*a) {

return (\*a).used\_size;

}

// Function to calculate the sum of the elements in the array

int SumOfArray(struct Array \*a) {

int sum = 0;

for (int i = 0; i < (\*a).used\_size; i++) {

sum += (\*a).ptr[i];

}

return sum;

}

int main() {

struct Array elements; // Declare an array

printf("Enter the number of Elements: ");

int c;

scanf("%d", &c);

if (c > 100) {

printf("Memory space exceeds the allotted space.\nEnter a number less than 100\n");

return 1;

}

createArray(&elements, c); // Create an array with the specified size

printf("Enter Elements\n");

setVal(&elements); // Set the values of the elements

printf("Displaying the elements: ");

display(&elements); // Display the elements

int option;

while (1) {

printf("\n====================================================================================================\n");

printf("\nMenu:\n");

printf("1. Add Element\n");

printf("2. Delete Element\n");

printf("3. Count Elements\n");

printf("4. Sum of Elements\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &option);

switch(option) {

case 1: {

int element, index;

printf("Enter the element to be added: ");

scanf("%d", &element);

printf("Enter the position at which it needs to be inserted: ");

scanf("%d", &index);

AddElementToArrayAtPos(&elements, element, index - 1);

printf("Displaying the elements: ");

display(&elements);

break;

}

case 2: {

int element;

printf("Enter the element to be deleted: ");

scanf("%d", &element);

DeleteFromArray(&elements, element);

printf("Displaying the elements: ");

display(&elements);

break;

}

case 3: {

int count = CountElementsOfArray(&elements);

printf("The number of elements in the array is: %d\n", count);

break;

}

case 4: {

int sum = SumOfArray(&elements);

printf("The sum of the elements in the array is: %d\n", sum);

break;

}

case 5: {

printf("Exiting...\n");

free(elements.ptr); // Free the allocated memory

return 0;

}

default: {

printf("Invalid option. Please try again.\n");

break;

}

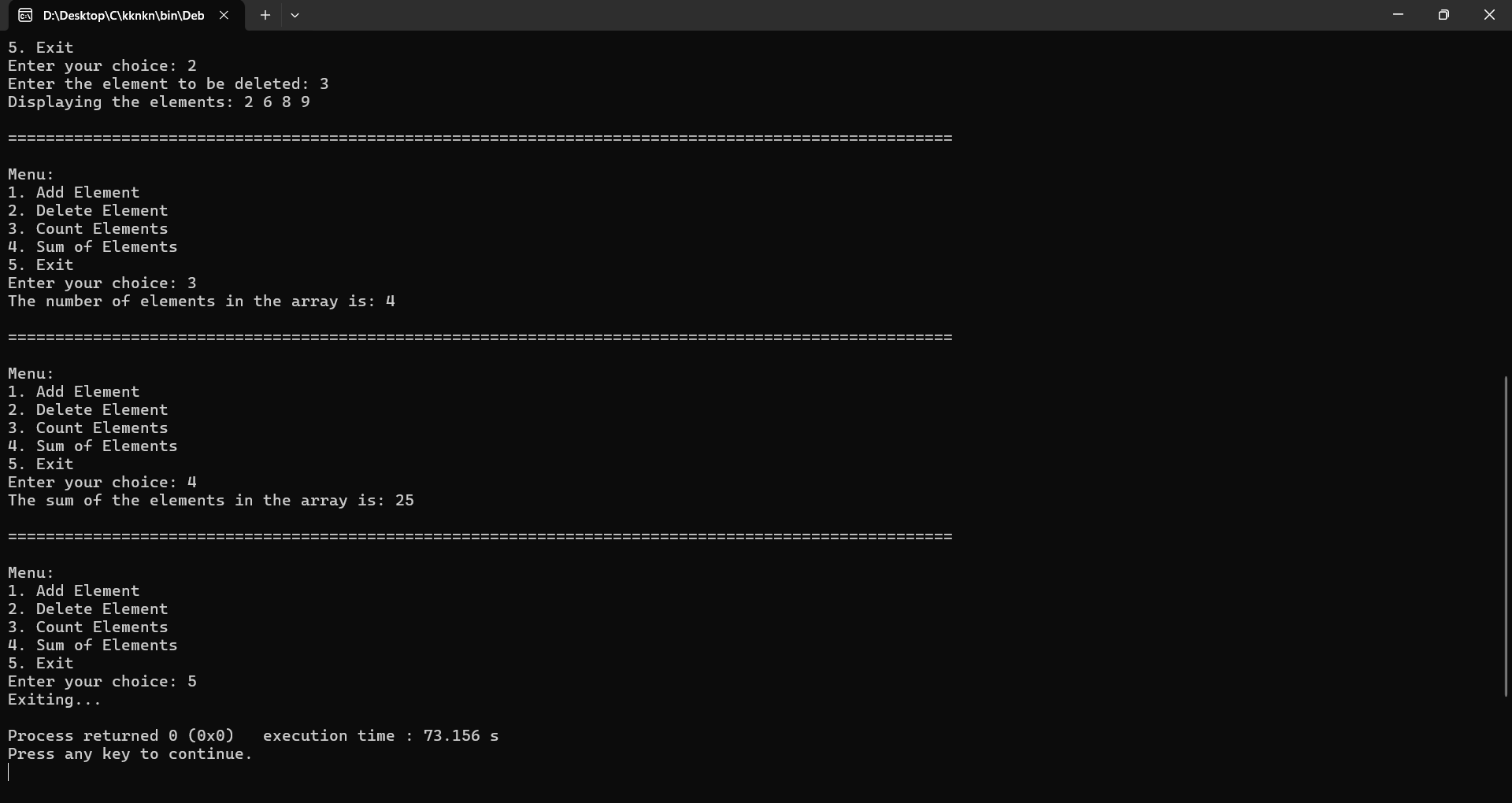
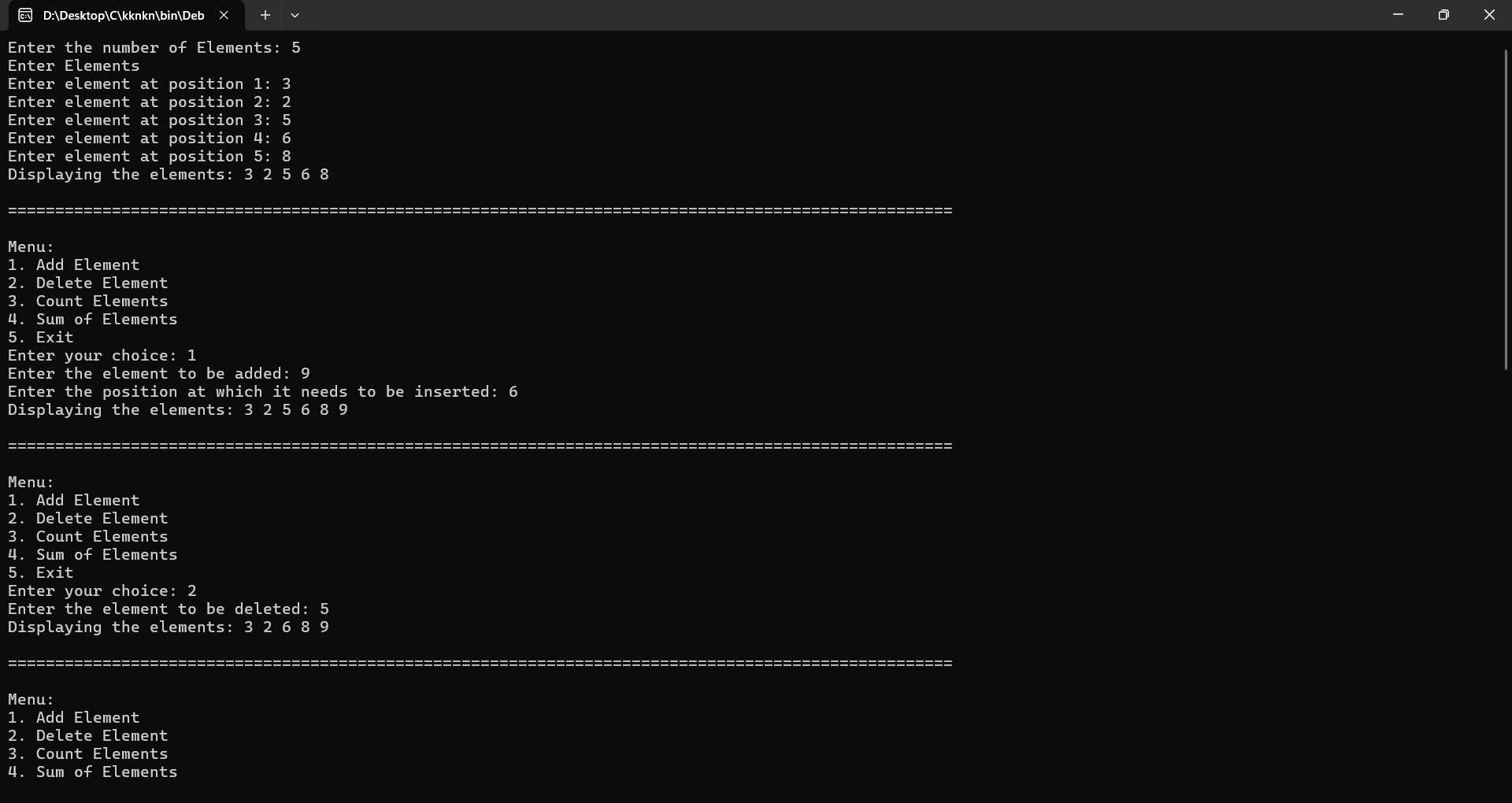
}

}

return 0;

}

**Output:**



**Conclusion:-**

From this experiment, we learnt about Abstract Data Type which helped to understand the process behind commonly used operations like creating, adding and deleting an element in an array, etc from scratch