Lab 1:

Objectives

- To perform basic operations in a one dimensional array such as insertion, deletion, update, traversal & merging.

An array is a data structure that stores elements of same type & in contigueous memory locations, allowing efficient access to individual elements using an index. Arrays provide a straightforward way to store & manipulate collections of data, but their size fixed at the time of declaration Hence, dynamic operations like insertion, deletion & updating require careful manipulation of the array's elements.

Algorithms of different operations

a) Insertion

1). Input the total number of elements n & the array elements.
2). Input the elements to be inserted & the index i where insertion is to occur
3). If $i<0$ or $i>0$, output "invalid index" & terminate.
4). Shift all elements from index i to the right by one position.
5) Insert the new element at index i.
6). Increment n by 1.
7). Display the updated array.

b). Deletion

1). Input the total number of elements n & the array elements
2). Input the index i of the element to be deleted
3). If i<0 or i>=n, output "invalid index" & teominate.
4). Shift all elements from index i+1 to the left.
5). Decrease n by 1.
6). Display the updated array.

c). Updatee

1). Input the total number of elements n & the array elements.
2). Input the index i of the element to update.
3). If i<0 or i>=n, output "invalid index" & terminate
4). Input the new value.
5). Replace the value at index i with the new value.
6). Display the updated array.

d). Traverse

1). Input the total number of elements n & the array elements.
2). Iterate through the array from index 0 to index n-1.
3). Output each element.

e) Merge

a). Input the total number of elements in the first array, n1 &

Merge

1). Input the total number of elements in the first array, n1, & its elements.
2). Input the total number of elements in the second array, n2, & its elements.
3). Initialize a new array merged of size n1 + n2.
4). Copy elements of the first array to merged starting from index 0.
   • For each i from 0 to n1-1, assign merged[i] = arr1[i]
5). Append elements of the second array to merged
   • for each j from 0 to n2-1, assign merged[n1+j] = arr2[j]
6). Output the elements of merged by traversing it from index 0 to index [n1+ n2-1]

These operations illustrate how many arrays can be manipulated to accomodate varying program requirements emphasizing the importance of memory management & efficient indexing. While arrays are simple & versatile, dynamic memory allocation (using functions malloc & realloc) can enhance the flexibility

This lab also plays a foundation for understanding more complex data structures such as stacks, queus & linked list which ~~build~~ build upon these basic operations.

C program to perform a). insert, b) Delete c) Update d) Traverse

```
#include <stdio.h>
#include <stdlib.h>

a) void insert{ (int **ptr, int size) {
       *ptr = (int *)malloc( size * sizeof(int));
   if  (*ptr == = NULL) {
         printf ("Memory Allocation Failed! ");
         exit (1);
   }
   printf("Enter the elements of array: \n");
   for (int i =0; i < size; ++i) {
         scanf("%d", (*ptr) +i);
   }
}

int indexCheck (int i, int size) {
    if ( i >=  size || i < 0) {
          printf ("The given index doesn't exist. \n");
          return 1;
    }
    return 0;
}

void delete (int **ptr, int * size) {
    int i;
    printf("Enter the index that you want to delete: \n");
    scanf(" %d", &i);
    if (indexCheck( i,  *size))
        return;
    while  (i < *size - 1) {
        (*ptr)[i] = (*ptr)[i+1];
        ++i
    }
    int *temp = realloc ( *ptr, (*size -1) * sizeof(int));
```

```
if (temp == NULL && *size > 1) {
    printf("Memory Allocation failed! \n");
    return;
}
*ptr = temp;
(*size)--;
}
void update(int *ptr, int size) {
  int i;
  printf("Enter the index that you want to update: \n");
  scanf("%d", &i);
  if (indexCheck(i, size))
     return;
  printf("Enter the new value: \n");
  scanf("%d", (ptr+i));
}
void traverse(int *ptr, int size) {
     printf("The elements of array: \n");
     for (int i =0; i < size; ++i) {
          printf("%d", *(ptr+i));
     }
}

int main() {
    int *ptr = NULL, size;

    printf("Enter the number of elements: \n");
    scanf("%d", &size);

    insert(&ptr, size);
    delete(&ptr, &size);
    update(ptr, size);
    traverse(ptr, size);
    free(ptr);
    return 0;
}
```

Merge

```
#include <stdio.h>
int main() {
    int n1, n2;

    printf("Enter the total no. of elements in 1st array: \n");
    scanf("%d", &n1);
    printf("Enter the total no. of elements in 2nd array: \n");
    scanf("%d", &n2);

    int arr1[n1], arr2[n2];
    printf("Enter the elements of the 1st array: \n");
    for (int i=0; i<n1; ++i) {
        scanf("%d", &arr1[i]);
    }
    printf("Enter the elements of the 2nd array: \n");
    for (int j=0; j<n2; ++j) {
        scanf("%d", &arr2[j]);
    }

    int merged[n1+n2];
    for (int i=0; i<n1; ++i) {
        merged[i] = arr1[i];
    }
    for (int j=0; j<n2; ++j) {
        merged[j+n1] = arr2[j];
    }

    printf("Merged array: ");
    for(int k=0; k<n1+n2; ++k) {
        printf("%d", merged[k]);
    }
    printf("\n");

    return 0;
}
```

Conclusion

In this lab, fundamental operations on arrays, including insertion, deletion, updating, & traversal, were performed & analyzed. By implementing these operations using both static & dynamic arrays, the importance of efficient memory management & error handling in C programming was emphasized. The use of pointers & dynamic allocation (via malloc & realloc) highlighted how arrays can be adapted for varying program requirements. This lab serves as a stepping stone to more advanced data structures, enhancing understanding of essential programming concepts.