Data Structures

Md. Jalal Uddin

Lecturer

Department of CSE

City University

Email: jalalruice@gmail.com

No: 01717011128 (Emergency Call)

Array in Data Structure:

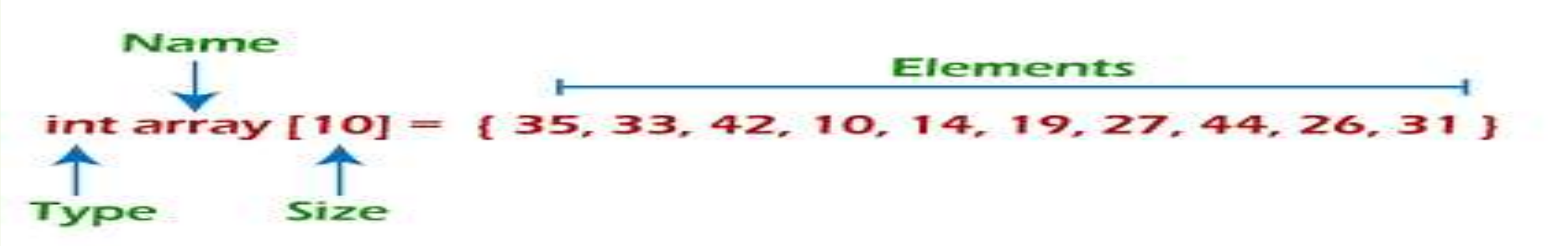
Arrays are defined as the collection of similar types of data items stored at contiguous memory locations. It is one of the simplest data structures where each data element can be randomly accessed by using its index number.

Properties of array

There are some of the properties of an array that are listed as follows -

- Each element in an array is of the same data type and carries the same size that is 4 bytes.
- Elements in the array are stored at contiguous memory locations from which the first element is stored at the smallest memory location.
- Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of the data element.

Representation of Array:



```
int array [10] = { 35, 33, 42, 10, 14, 19, 27, 44, 26, 31 }
```

As per the above illustration,

- Index starts with 0.
- The array's length is 10, which means we can store 10 elements.
- Each element in the array can be accessed via its index

Linear Arrays

A linear array is a list of finite number n of homogeneous data elements such that :

a) The elements of the array are referenced respectively by an index set consisting of n consecutive numbers.

b) The elements of the array are stored respectively in successive memory locations.

The number n of elements is called the length or size of the array.

Three numbers define an array : lower bound, upper bound, size.
a. The lower bound is the smallest subscript you can use in the array (usually 0)
b. The upper bound is the largest subscript you can use in the array
c. The size / length of the array refers to the number of elements in the array , It can be computed as upper bound - lower bound + 1

Let, Array name is A then the elements of A is : a_1,a_2….. A_n
Or by the bracket notation A[1], A[2], A[3],…………., A[n]
The number k in A[k] is called a subscript and A[k] is called a subscripted variable.

Linear Arrays

Example :

A linear array DATA consisting of six elements

DATA

1	247
2	56
3	429
4	135
5	87
6	156

DATA[1] = 247

DATA[2] = 56

DATA[3] = 429

DATA[4] = 135

DATA[5] = 87

DATA[6] = 156

Linear Arrays

Example :

An automobile company uses an array AUTO to record the number of auto mobile sold each year from 1932 through 1984. Find the total elements of the array AUTO.

AUTO[k] = Number of auto mobiles sold in the year K

LB = 1932

UB = 1984

Length = UB – LB+1 = 1984 – 1930+1 =55

That is AUTO contains 55 elements .

Representation of linear array in memory

Let LA be a linear array in the memory of the computer. The memory of the computer is a sequence of addressed locations.

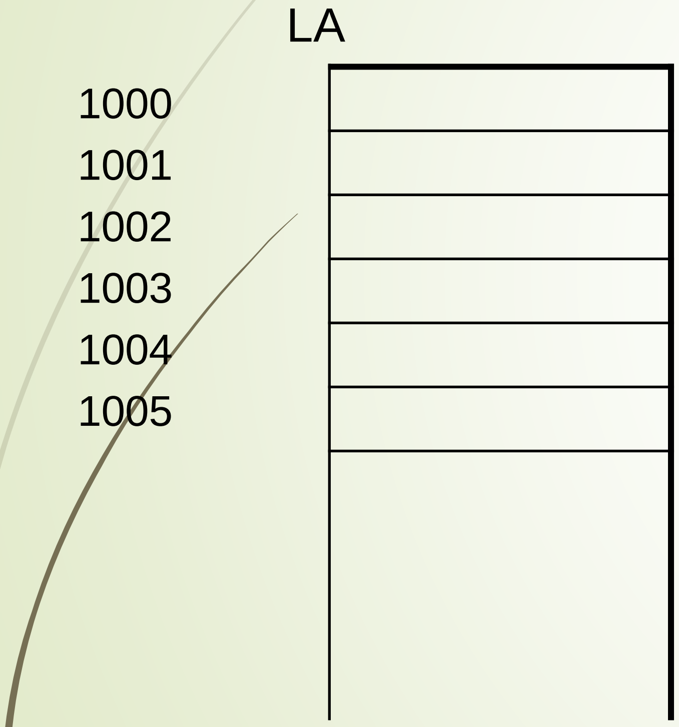


Fig : Computer memory

The computer does not need to keep track of the address of every element of LA, but needs to keep track only of the first element of LA, denoted by

Base(LA)

Called the base address of LA. Using this address Base(LA), the computer calculates the address of any element of LA by the following formula :

LOC(LA[k]) = Base(LA) + w(K – lower bound)

Where w is the number of words pr memory cell for the array LA

Representation of linear array in memory

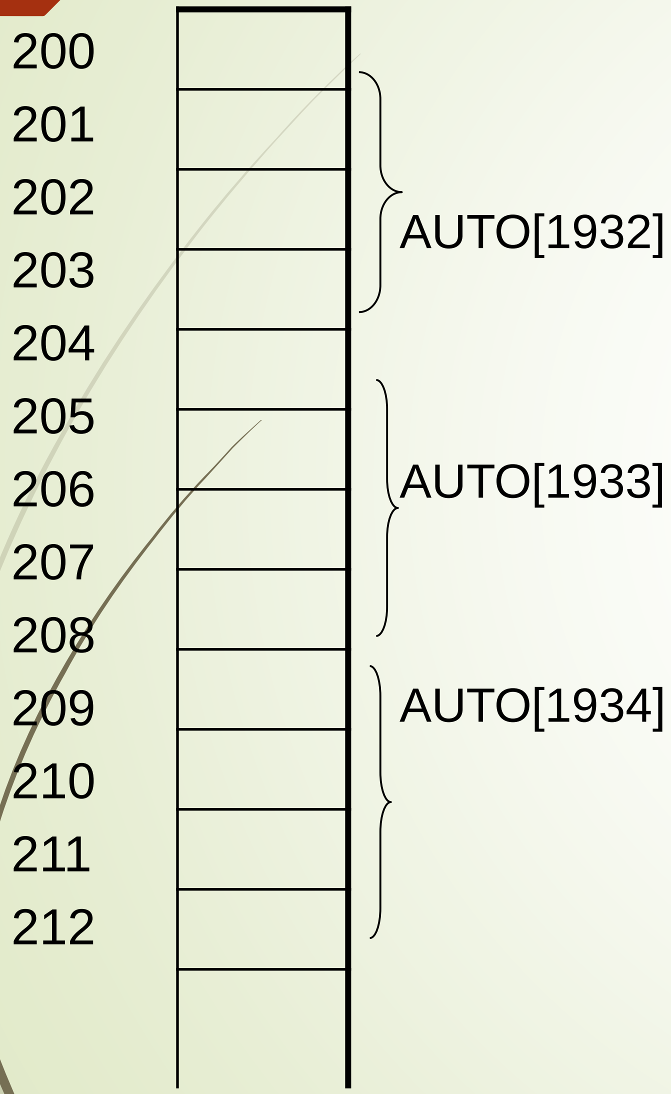


Fig : A

Example :
An automobile company uses an array AUTO to record the number of auto mobile sold each year from 1932 through 1984. Suppose AUTO appears in memory as pictured in fig A . That is Base(AUTO) = 200, and w = 4 words per memory cell for AUTO. Then,
LOC(AUTO[1932]) = 200, LOC(AUTO[1933]) =204
LOC(AUTO[1934]) = 208
the address of the array element for the year K = 1965 can be obtained by using :
LOC(AUTO[1965]) = Base(AUTO) + w(1965 – lower bound)
=200+4(1965-1932)=332

Traversing linear arrays

Print the contents of each element of DATA or Count the number of elements of DATA with a given property. This can be accomplished by traversing DATA, That is, by accessing and processing (visiting) each element of DATA exactly once.

Algorithm 2.3: Given DATA is a linear array with lower bound LB and upper bound UB . This algorithm traverses DATA applying an operation PROCESS to each element of DATA.

```
1.  [Assign counter]
   K=LB   // LB = 0
2. Repeat step 2.1 and 2.2 while K <= UB   // If LB = 0
    2.1    [visit element]
     do PROCESS on LA[K]
           2.2    [increase counter]
            K=K+1
3. end of step 2
4. exit
```

Traversal operation

This operation is performed to traverse through the array elements. It prints all array elements one after another. We can understand it with the below program -

```
#include <stdio.h>
void main() {
int Arr[5] = {18, 30, 15, 70, 12};
int i;
printf("Elements of the array are:\n");
for(i = 0; i<5; i++) {
printf("Arr[%d] = %d,  ", i, Arr[i]);
   }
}
```

Inserting and Deleting

Inserting refers to the operation of adding another element to the Array

Deleting refers to the operation of removing one element from the Array

Inserting an element somewhere in the middle of the array require that each subsequent element be moved downward to new locations to accommodate the new element and keep the order of the other elements.

Deleting an element somewhere in the middle of the array require that each subsequent element be moved one location upward in order to “fill up” the array. Fig shows Milon Inserted, Sumona deleted.

STUDENT

1	Dalia Rahaman
2	Sumona
3	Mubtasim Fuad
4	Anamul Haque
5	
6	

STUDENT

1	Dalia Rahaman
2	Sumona
3	Milon
4	Mubtasim Fuad
5	Anamul Haque
6	

STUDENT

1	Dalia Rahaman
2	Milon
3	Mubtasim Fuad
4	Anamul Haque
5	
6	

Insertion Algorithm

- Insert item at the back is easy if there is a space. Insert item in the middle requires the movement of all elements to the right as in Figure 1.

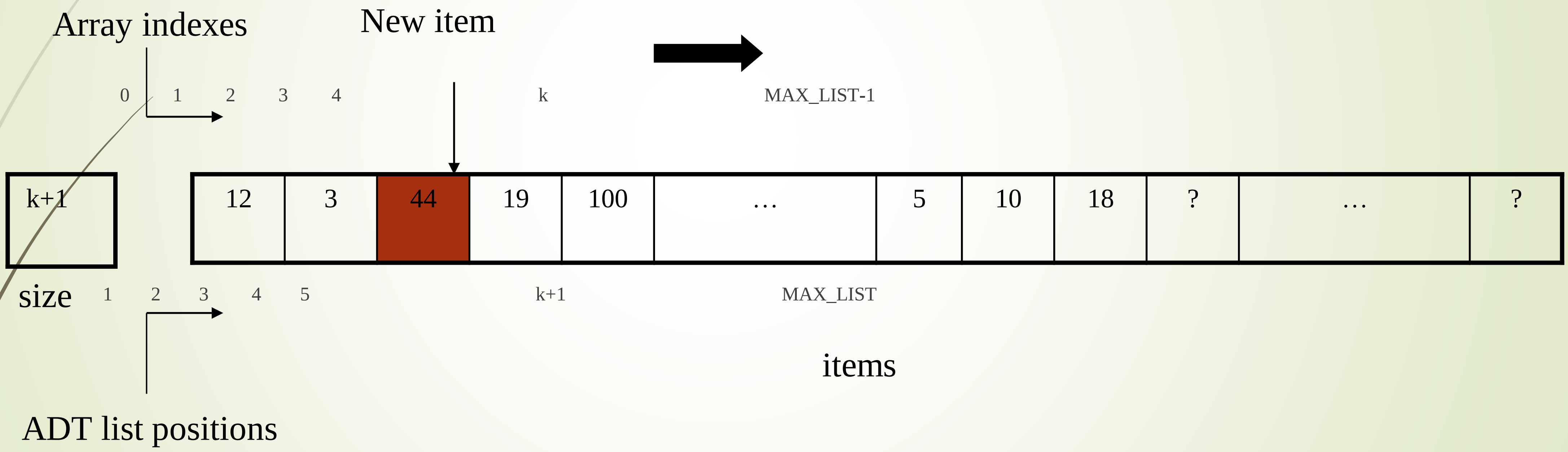


Figure 1: Shifting items for insertion at position 3

Insertion

```
INSERT(LA, N, K, ITEM)
//LA is a linear array with N element
//K is integer positive where K < N and LB = 0
//Insert an element, ITEM in index K
    1. [Assign counter]
       J = N – 1;    // LB = 0
    2. Repeat step 2.1 and 2.2 while J >= K
      2.1 [shift to the right all elements from J]
          LA[J+1] = LA[J]
      2.2 [decrement counter]   J = J – 1
    3. [Stop repeat step 2]
    4. [Insert element]   LA[K] = ITEM
    5. [Reset N]   N = N + 1
    6. Exit
```

Deletion Algorithm

- Delete item.

(a)

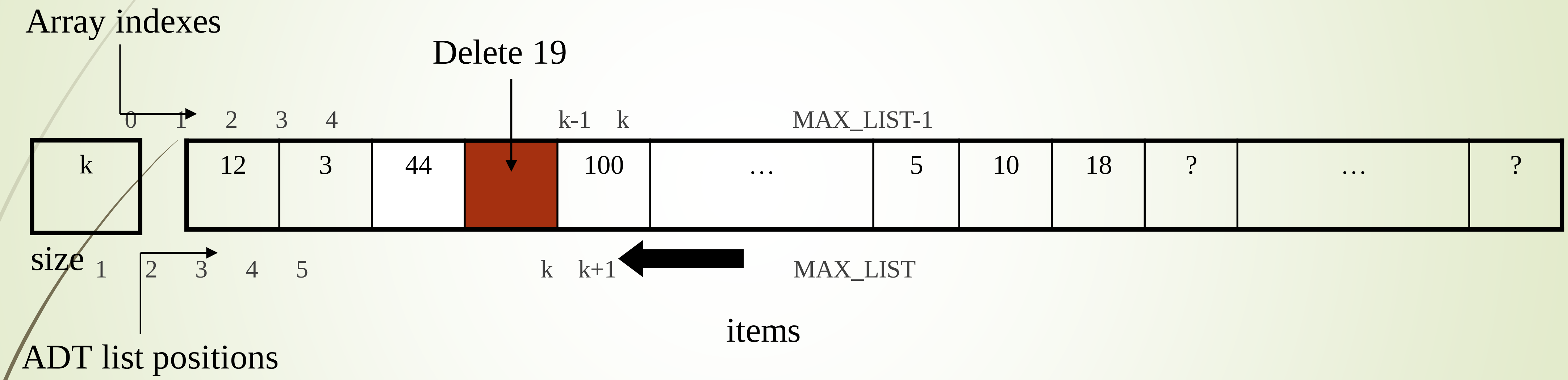
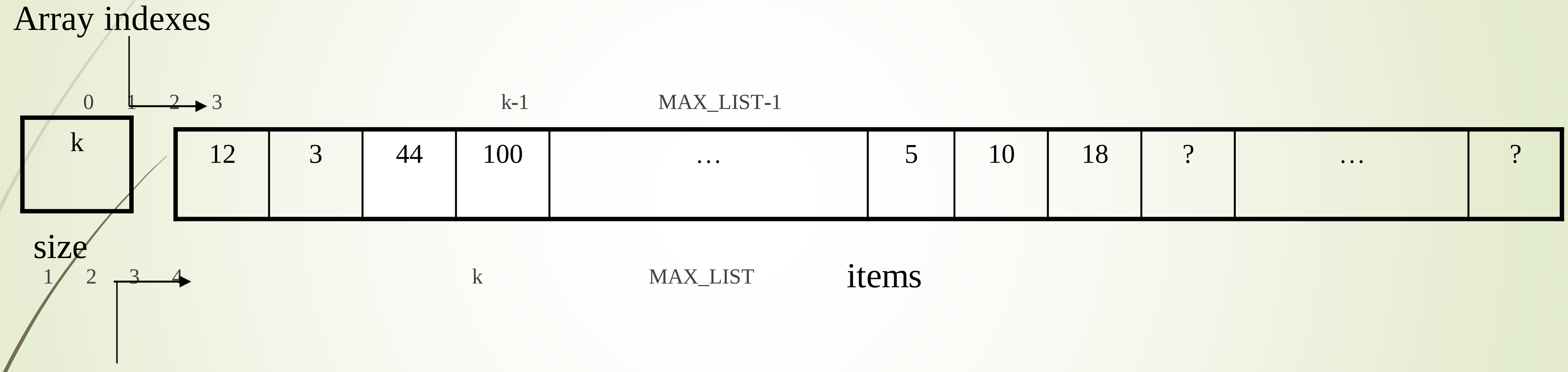


Figure 2: Deletion causes a gap

Deletion Algorithm

(b)



Array indexes

0 1 2 3 k-1 MAX_LIST-1

k

12	3	44	100	…	5	10	18	?	…	?

size

1 2 3 4 k MAX_LIST

items

ADT list positions

Figure 3: Fill gap by shifting

Deletion Algorithm

Example algorithm:

```
DELETE(LA, N, K, ITEM)
  1.  ITEM = LA[K]
  2.  Repeat for I = K to N–2     // If LB = 0
      2.1 [Shift element J + 1, forward]
            LA[I] = LA[I+1]
  3.  [end of loop]
  4.  [Reset N in LA]
      N = N – 1
  5.  Exit
```

Insertion operation:

This operation is performed to insert one or more elements into the array. As per the requirements, an element can be added at the beginning, end, or at any index of the array. Now, let's see the implementation of inserting an element into the array.

```
#include <stdio.h>
int main()
{
    int arr[20] = { 18, 30, 15, 70, 12 };
    int i, x, pos, n = 5;
    printf("Array elements before insertion\n");
    for (i = 0; i < n; i++)
    printf("%d ", arr[i]);
    printf("\n");

    x = 50; // element to be inserted
    pos = 4;
    n++;
for (i = n-1; i >= pos; i--)
    arr[i] = arr[i - 1];
    arr[pos - 1] = x;
    printf("Array elements after insertion\n");
    for (i = 0; i < n; i++)
    printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```

Deletion Operation:

This operation removes an element from the array and then reorganizes all of the array elements.

```
#include <stdio.h>
void main() {
   int arr[] = {18, 30, 15, 70, 12};
   int k = 30, n = 5;
   int i, j;
   printf("Given array elements are :\n");

   for(i = 0; i<n; i++) {
   printf("arr[%d] = %d,  ", i, arr[i]);
   }
while( j < n)
{
    arr[j-1] = arr[j];
    j = j + 1;
}
    n = n -1;
    printf("\nElements of array after deletion:\n");
    for(i = 0; i<n; i++) {
    printf("arr[%d] = %d,  ", i, arr[i]);
   }
}
```

Search operation:

This operation is performed to search an element in the array based on the value or index.

```
#include <stdio.h>
void main() {
   int arr[5] = {18, 30, 15, 70, 12};
   int item = 70, i, j=0 ;
   printf("Given array elements are :\n");
   for(i = 0; i<5; i++) {
   printf("arr[%d] = %d,  ", i, arr[i]);
   }
   printf("\nElement to be searched = %d", item);
   while( j < 5){
   if( arr[j] == item ) {
   break;
      }
   j = j + 1;
   }
```

References: