Array

- An array is defined as the collection of similar type of data items stored at contiguous memory locations.
- These entities or elements can be of int, float, char, or double data type or can be of user-defined data types too like structures.
- However, in order to be stored together in a single array, all the elements should be of the same data type.
- The elements are stored from left to right with the left-most index being the 0^{th} index and the rightmost index being the (n-1) index.

Properties:

i. Each element of an array is of same data type and carries the same size, i.e., int = 4 bytes.

ii. Elements of the array are stored at contiguous memory locations where the first element is stored at the smallest memory location.

iii. 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.

Advantages of C++ Array

a = {2, 3, 4, 5, 6}

1) ***Code Optimization***: Less code to the access the data.

2) ***Ease of traversing***: By using the for loop, we can retrieve the elements of an array easily.

3) ***Ease of sorting***: To sort the elements of the array, we need a few lines of code only.

4) ***Random Access***: We can access any element randomly using the array.

Disadvantage of C++ Array

1) Fixed Size: Whatever size, we define at the time of declaration of the array, we can't exceed the limit. So, it doesn't grow the size dynamically like LinkedList which we will learn later.

Types of Array in C++

i. One Dimensional Arrays (1D Array)

ii. Multidimensional Arrays

Declaring Arrays:

- Array is a type consisting of a contiguously allocated nonempty sequence of objects with a particular element type. The number of those objects (the array size) never changes during the array lifetime.
- Array variables are declared identically to variables of their data type, except that the variable name is followed by one pair of square [ ] brackets for each dimension of the array.
- Uninitialized arrays must have the dimensions of their rows, columns, etc. listed within the square brackets.
- Dimensions used when declaring arrays in C++ must be positive integral constants or constant expressions.

Syntax:

Data_type array_name [array_size];

Where,

Data_type can be int, float, double, char etc.

array_name should be a valid variable name.

array_size is an integer value.

Declaring Arrays:

- Array is a type consisting of a contiguously allocated nonempty sequence of objects with a particular element type. The number of those objects (the array size) never changes during the array lifetime.
- Array variables are declared identically to variables of their data type, except that the variable name is followed by one pair of square [ ] brackets for each dimension of the array.
- Uninitialized arrays must have the dimensions of their rows, columns, etc. listed within the square brackets.
- Dimensions used when declaring arrays in C++ must be positive integral constants or constant expressions.

Syntax:

Data_type array_name [array_size];

Where,

Data_type can be int, float, double, char etc.

array_name should be a valid variable name.

array_size is an integer value.

nitializing Arrays:

- The elements of an array can be initialized by using an initialization list. An initialization list is a comma-separated list of initializers enclosed within braces.
- An initializer is an expression that determines the initial value of an element of the array.
- If the type of initializers is not the same as the element type of an array, implicit type casting will be done, if the types are compatible. If types are not compatible, there will be a compilation error. this fact.

Syntax:

Data_type array_name [array_size]={value1, value2, ... valueN};

Where,

Values are of same data type or implicit data type.

- There are different ways through which we can initialize the array as follows.

Method 1: Initialize an array using an Initializer List

- An initializer list initializes elements of an array in the order of the list.

Example-1:

```
int arr[5] = {1, 2, 3, 4, 5};        //a[0]=1, a[1]=2....
```

Example-2:

```
int arr[5];

a[0]=1; a[1]=2; a[2]=3; a[3]=4; a[4]=5;
```

Example-2:

```
int arr[5] = {1, 2, 3};
```

1	2	3	0	0

Method 2: Initialize an array in C++ using a for loop

```
#include<iostream>
using namespace std;
int main()
{
   int n;
   cout<<"Enter the number of elements: ";
   cin>>n;
   int a[n];
   for(int i=0;i<n;i++)
   {
      cout<<"Enter the "<< i<<"th element of a: ";
      cin>>a[i];
   }
   return(0);
}
```

Input:

Enter the number of elements: 4

Enter the 0th element of a: 14

Enter the 1th element of a: 15

Enter the 2th element of a: 22

Enter the 3th element of a: 24

Accessing Array:

- We can access any element of an array in C++ using the array subscript operator [ ] and the index value 'i' of the element.
- One thing to note is that the indexing in the array always starts with 0, i.e., the first element is at index 0 and the last element is at N – 1 where N is the number of elements in the array.

array_name [index];

```
int a[4] = {12, 5, 9, 14}
cout<<a[2]; 19
```

```
#include<iostream>
using namespace std;
int main()
{
  //Array declaration and initialization
  int arr[5]={14, 21,11,5,47};
  //Accessing array elements.
  cout<<"Element at arr[0] is "<<arr[0]<<endl;
  cout<<"Element at arr[2] is "<<arr[2]<<endl;
  cout<<"Element at arr[4] is "<<arr[4]<<endl;
  return(0);
}
```

Output:

Element at arr[0] is 14

Element at arr[2] is 11

Element at arr[4] is 47

- We can access all elements of an array in C++ using loop.

```
#include<iostream>
using namespace std;
int main()
{
   //Array declaration and initialization
   int arr[5]={14, 21,11,5,47};
   //Accessing array elements
   for(int i=0;i<5;i++)
   {
      cout<<arr[i]<<'\t';
   }
   return(0);
}
```

Output:

14 21 11 5 47

- Array of string.

Syntax:

string array_name[string_numbers] = {string1,string2,....};

```
#include<iostream>
using namespace std;
int main()
{
   //Array declaration and initialization
   string arr[5]={"ITER","SOA","Training","C++","Class"};
   //Accessing array elements
   cout<<arr[0]<<endl;
   cout<<arr[2]<<endl;
   cout<<arr[4]<<endl;
   arr[0]="CSE";
   cout<<arr[0]<<endl;
   return(0);
 }
```

Output:

```
ITER
Training
Class
CSE
```

Operation on Arrays in C++

There are a number of operations that can be performed on an array which are:

1. Traversal
2. Copying
3. Sorting
4. Searching
5. Reversing
6. Insertion
7. Deletion
8. Merging

1. Traversal

- Traversing basically means the accessing the each and every element of the array at least once. Traversing is usually done to be aware of the data elements which are present in the array.

Algorithm:

Step 01: Start

Step 02: [Initialize counter variable.] Set i = 0 (Indexing starts from 0 to n-1).

Step 03: Repeat for i = 0 to n-1.

Step 04: Apply process to arr[i].

Step 05: [End of loop.]

Step 06: Stop

2. Copying

- Copying an array involves index-by-index copying.
- For this to work we shall know the length of array in advance, which we shall use in iteration.

Algorithm:

Step 01: Start

Step 02: Take two arrays, A and B.

Step 03: Store values in A

Step 04: Loop for each value of A

Step 05: Copy each index value to B array at the same index location

Step 06: Stop

Copying all elements of A to B.

```
using namespace std;
int main()
{
   int A[5]={5,14,24,63,7};
   int B[5]={15,4,34,13,27};
   for (int i=0;i<5;i++)
   {
      B[i]=A[i];
      cout<<B[i]<<'\t';
   }
   return(0);
}
```

Output:

5 14 24 63 7

Copying first 3 elements of A to B.

```
#include<iostream>
using namespace std;
int main()
{
   int A[5]={5,14,24,63,7};
   int B[5]={15,4,34,13,27};
   for (int i=0;i<3;i++)
   {
      B[i]=A[i];
   }
   for (int i=0;i<5;i++)
   {
      cout<<B[i]<<'\t';
   }
   return(0);
}
```

Output:

5 14 24 13 27

3. Sorting

- Sorting is the process of arranging elements in a list or array in a specific order, typically in ascending or descending order.

    i. Selection Sort

    ii. Bubble Sort

    iii. Insertion Sort

i. Selection Sort

- Selection sort is a sorting algorithm that selects the smallest or largest element from an unsorted list in each iteration and places that element at the beginning or end position of the unsorted list.

Algorithm:

Step 1: Set Min to location 0 in Step 1.

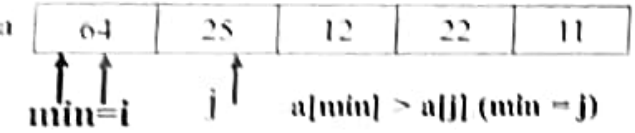
Step 2: Look for the smallest element on the list.

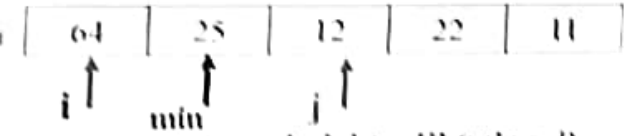
Step 3: Replace the value at location Min with a different value.

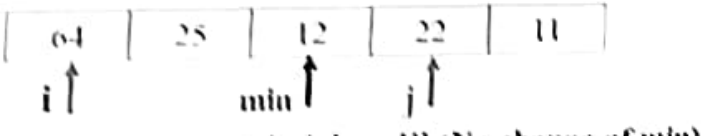
Step 4: Increase Min to point to the next element

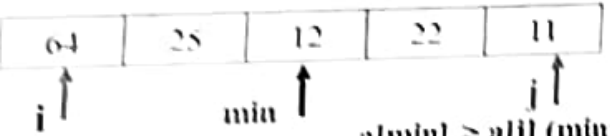
Step 5: Continue until the list is sorted.

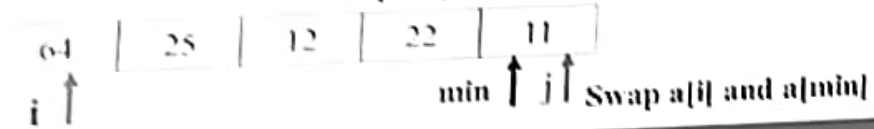
Example: Sorting of an array in ascending order using selection sort

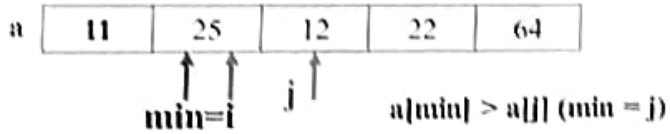


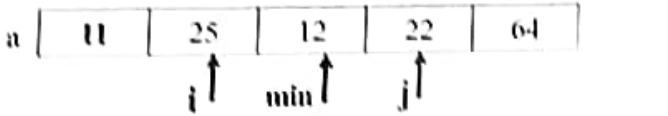


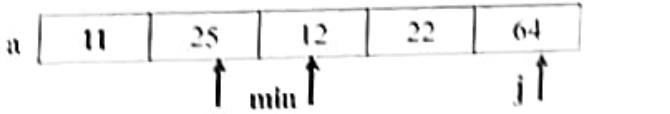




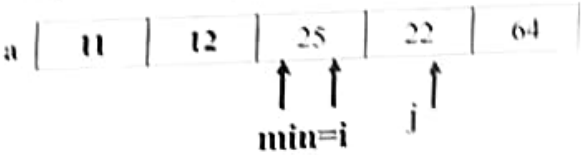




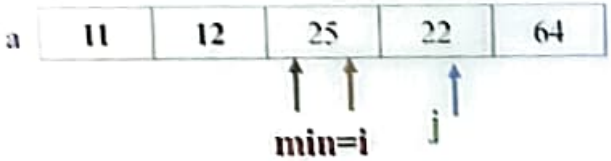




Swap a[i] and a[min]



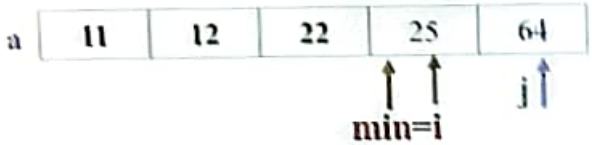
Repeat the steps upto i < 5



a[min] > a[j] (min=j)

a[min] > a[j] (No change)

Swap a[i] and a[min]



a[min] > a[j] (No change)

11	12	22	25	64

```
#include<iostream>
using namespace std;
int main()
{
  int arr[5]={5,14,24,63,7};
  int i,j,min,temp;
  for(i=0;i<4;i++)
  {
    min=i;
for(j=i+1;j<=4;j++)
    {
      if (arr[min]>arr[j])
      {
        min=j;
      }
    }
      if (arr[min]<arr[i])
      {
        temp=arr[min];
        arr[min]=arr[i];
        arr[i]=temp;
      }
  }
  for (i=0;i<5;i++)
  {
    cout<<arr[i]<<'\t';
  }
  return(0);
}
```

10-9

Output:

5 7 14 24 63

ii. Bubble Sort

- Bubble sort is a sorting algorithm that compares two adjacent elements and swaps them until they are in the intended order.

In this algorithm,

- Traverse from left and compare adjacent elements and the higher one is placed at right side.
- In this way, the largest element is moved to the rightmost end at first.
- This process is then continued to find the sec  e it and so on until the data is sorted.

Algorithm:

```
for all elements of list
  if list[i] > list[i+1]
    swap(list[i], list[i+1])
  end if
end for
return list
```