Assignment-54 **A Job Ready Bootcamp in c++,DSA and IOT**

DSA: Array and Dynamic Array

1. Create an ADT array without using STL.
2. Create an array and implement a search function in the array.
3. Create a function in Array to reverse an array.
4. Create a function in Array to sort the given array
5. Create a function in Array to check the size of an array.
6. Create a Dynamic array without using STL.
7. Create a function in a dynamic array to return the size of the array.
8. Create a function in a dynamic array to return the capacity of the array.
9. Create an array and implement a search function in the array.
10. Create a function in Array to reverse an array.
11. Create a function in Array to sort the given array

Array Data Structure

```
#include <iostream>
using namespace std;

class Array
{
    private:
        int *ptr;        // for dynamically creating array
        int capacity;    // for capacity of array
        int lastIndex;   // for last filed block of array
    public:
        Array(int);
        void append(int);
        bool search(int);
        void displayData();
        void reverse();
        ~Array();
        void sort();
        int size();
        int Capacity();
};
// total capacity of array
int Array::Capacity()
{
    return capacity;
}
// find size of array
int Array::size()
```

```
{
    return lastIndex+1;
}
// sorting array element
void Array::sort()
{
    int temp;
    if(lastIndex == -1)
        cout << endl << "Array is empty";
    else
    {
        for(int i = 0; i < lastIndex; i++)
        {
            for(int j = i + 1; j <= lastIndex; j++)
            {
                if(ptr[i] > ptr[j])
                {
                    temp = ptr[i];
                    ptr[i] = ptr[j];
                    ptr[j] = temp;
                }
            }
        }
    }
}

// release memory
Array::~Array()
{
    delete []ptr;
}
// reverse array
void Array::reverse()
{
    if(lastIndex == -1)
        cout << endl << "Array is empty";
    else
    {
        int temp, last = lastIndex;
        for(int i = 0; i < last; i++)
        {
```

```
            temp = ptr[i];
            ptr[i] = ptr[last];
            ptr[last] = temp;
            last--;
        }
    }
}

void Array::displayData()
{
    if(lastIndex == -1)
        cout << endl << "Array is empty";
    else
    {
        cout << endl;
        for(int i = 0; i <= lastIndex; i++)
            cout << ptr[i] << " ";
    }
}
// search array element
bool Array::search(int data)
{
    int i;
    for(i = 0; i <= lastIndex; i++)
        if(ptr[i] == data)
            return true;
    return false;
}
// append data in array
void Array::append(int data)
{
    if(capacity == lastIndex + 1)
        cout << endl << "Array is already full";
    else
    {
        lastIndex++;
        ptr[lastIndex] = data;
    }
}

// create array by passing value
```

```
Array::Array(int cap)
{
    capacity = cap;
    ptr = new int[capacity];
    lastIndex = -1; // because of after store data in array we can
simply increment by 1
}

int main()
{
    Array obj(10);
    obj.displayData();

    obj.append(10);
    obj.append(20);
    obj.append(50);
    obj.append(80);
    obj.append(210);
    obj.append(850);
    obj.append(750);

    obj.displayData();

    if(obj.search(80))
        cout << endl << "Value found";
    else
        cout << endl << "Value not found";

    obj.reverse();
    cout << endl <<"After reverse";
    obj.displayData();

    obj.sort();
    cout << endl <<"After sort";
    obj.displayData();

    cout << endl << "Size of array: " << obj.size();
    cout << endl << "Capacity of array: " << obj.Capacity();
    return 0;
}
==========================================================================
```

```
Output:
Array is empty
10 20 50 80 210 850 750
Value found
After reverse
750 850 210 80 50 20 10
After sort
10 20 50 80 210 750 850
Size of array: 7
Capacity of array: 10
```

Dynamic Array Data Structure

```
#include <iostream>
using namespace std;

class Array
{
    private:
        int *ptr;        // for dynamically creating array
        int capacity;    // for capacity of array
        int lastIndex;   // for last filed block of array
        void doubleArray();
        void halfArray();
    public:
        Array(int);
        void del(int);
        void append(int);
        bool search(int);
        void displayData();
        void reverse();
        ~Array();
        void sort();
        int size();
        int Capacity();
};
// delete array element
void Array::del(int index)
{
    if(index > lastIndex || index < 0)
        cout << endl << "Invalid index";
    else
```

```
    {
        if(lastIndex + 1 == capacity / 2 && capacity > 1)
            halfArray();
        for(int i = index; i < lastIndex; i++)
            ptr[i] = ptr[i + 1];
        lastIndex--;
    }
}

// half Array function
void Array::halfArray()
{
    int *temp;
    temp = new int[capacity/2];
    for(int i = 0; i <= lastIndex; i++)
        temp[i] = ptr[i];
    delete []ptr;
    ptr = temp;
    capacity /= 2;
}

//double array size
void Array::doubleArray()
{
    int *temp;
    temp = new int[capacity*2];
    for(int i = 0; i <= lastIndex; i++)
        temp[i] = ptr[i];
    delete []ptr;
    ptr = temp;
    capacity *= 2;
}

// total capacity of array
int Array::Capacity()
{
    return capacity;
}
// find size of array
int Array::size()
{
```

```
    return lastIndex+1;
}
// sorting array element
void Array::sort()
{
    int temp;
    if(lastIndex == -1)
        cout << endl << "Array is empty";
    else
    {
        for(int i = 0; i < lastIndex; i++)
        {
            for(int j = i + 1; j <= lastIndex; j++)
            {
                if(ptr[i] > ptr[j])
                {
                    temp = ptr[i];
                    ptr[i] = ptr[j];
                    ptr[j] = temp;
                }
            }
        }
    }
}

// release memory
Array::~Array()
{
    delete []ptr;
}
// reverse array
void Array::reverse()
{
    if(lastIndex == -1)
        cout << endl << "Array is empty";
    else
    {
        int temp, last = lastIndex;
        for(int i = 0; i < last; i++)
        {
            temp = ptr[i];
```

```
            ptr[i] = ptr[last];
            ptr[last] = temp;
            last--;
        }
    }
}

void Array::displayData()
{
    if(lastIndex == -1)
        cout << endl << "Array is empty";
    else
    {
        cout << endl;
        for(int i = 0; i <= lastIndex; i++)
            cout << ptr[i] << " ";
    }
}
// search array element
bool Array::search(int data)
{
    int i;
    for(i = 0; i <= lastIndex; i++)
        if(ptr[i] == data)
            return true;
    return false;
}
// append data in array
void Array::append(int data)
{
    if(capacity == lastIndex + 1)
        doubleArray();
    lastIndex++;
    ptr[lastIndex] = data;
}

// create array by passing value
Array::Array(int cap)
{
    capacity = cap;
    ptr = new int[capacity];
```

```
    lastIndex = -1; // because of after store data in array we can
simply increment by 1
}

int main()
{
    Array obj(6);
    obj.displayData();

    obj.append(10);
    obj.append(20);
    obj.append(50);
    obj.append(80);
    obj.append(210);
    obj.append(850);

    obj.displayData();
    cout << endl << "Size of array: " << obj.size();
    cout << endl << "Capacity of array: " << obj.Capacity();
    if(obj.search(80))
        cout << endl << "Value found";
    else
        cout << endl << "Value not found";

    obj.reverse();
    cout << endl <<"After reverse";
    obj.displayData();

    obj.sort();
    cout << endl <<"After sort";
    obj.displayData();

    obj.append(121);
    obj.displayData();
    cout << endl << "Size of array: " << obj.size();
    cout << endl << "Capacity of array: " << obj.Capacity();

    obj.del(2);
    obj.del(4);
    obj.displayData();
    cout << endl << "Size of array: " << obj.size();
```

```
    cout << endl << "Capacity of array: " << obj.Capacity();

    return 0;
}
```

==========

```
Output:
Array is empty
10 20 50 80 210 850
Size of array: 6
Capacity of array: 6
Value found
After reverse
850 210 80 50 20 10
After sort
10 20 50 80 210 850
10 20 50 80 210 850 121
Size of array: 7
Capacity of array: 12
10 20 80 210 121
Size of array: 5
Capacity of array: 6
```