PW SKILLS

Lesson Plan

Array and Vector

PW SKILLS

Array Introduction

1. Arrays can be described as a type of data structure used to store a group or collection of items or elements sequentially inside a memory.
2. The main function of an Array is to store a collection of homogenous data of the same type. For example, integers, strings, floating numbers, etc
3. The indexing of an Array is 0 based indexing ie the first element is at index 0 and the second element is at index 1 and so forth. We can directly access the required elements using the indexes.
4. The memory allocation in Arrays is contiguous in nature.
5. We can create both single-dimensional and multidimensional arrays.

Array Declaration and Creation

Syntax for creating a new Array in C++ is data-type array-name[array-size];

Some examples:
int arr[10]; , float array[15];
Array Literal
With curly braces we can initialize the array and add value to it during initialization without defining the size.
int arr[] = { 1,2,3,4,5,6,7,8,9,10 };

Array Types

- Single dimensional or one-dimensional array
- Multidimensional Array
- Single dimensional Array:

1. When we have elements stored in a single dimension sequentially, they are called single dimensional arrays.
2. We can declare and allocate memory to a single-dimensional array using a single variable in C++.

Here is an example,

```
string colours[] = {"Red", "Green", "Blue"};
// To print the elements in the console:
for(string colour : colours)
cout << colour << ", ";
```

Output: Red, Green, Blue

Syntax for declaring an single dimension array -
int arrayName[] = {element_0, element_1, element_2, element_3, ...elementN};

Since we have come this far in arrays, it is worth mentioning here that there is a provision in C++ wherein we can store the elements/data dynamically and sequentially. That means the length of this container is dynamic. These are called vectors ! Let's learn about them.

Vectors in C++

Vectors are dynamic arrays that can be resized whenever an element is inserted or deleted. Vectors also have contiguous storage like arrays but their size can change dynamically according to the requirements of user. Another important point to note here is that vectors are not compulsorily bound by index values.

Declaration Syntax: vector < data_type > name;

Example: vector<int>myVec;

Note: We can even declare a vector of fixed size also with the following syntax:
vector<int> v(n);

Here, a vector of size n is declared.

Basic Operations on Vectors

Just like in arrays, it returns the size of vector.

Example:
```
vector<int>v = {10,20,30,40,50};
cout << v.size() << endl;
```

Output : 5

Resizing operation in vector:

Unlike the fixed size approach of arrays, we can resize the vector according to our requirement.

Example:
```
vector<int> v;
v.resize(n);
```

Note: This is extremely useful when we do not know the size initially (at the time of creating the vector). It can be resized anytime as per need.

Vector capacity() function: It returns the size of the storage space allocated to the vector. This capacity is greater than or equal to the size of the vector catering the need for extra space to allow growth without the need to reallocate on each insertion.

Note: This capacity is not the limit for growth and can be automatically expanded.

Code Example:
```
vector<int> v;
v.resize(10);
cout << v.capacity() << endl;
v.resize(90);
cout << v.capacity() << endl;
```

Output:
16
128

Note: Whenever we resize and increase the size of the vector, the total capacity gets to the next power of 2 or the remaining memory. For example, in the above case, the vector size is 10 but its capacity is the closest next power of 2 i.e.16. When increased to 90, its capacity went to the closest next power of 2 i.e. 128.

Inserting an element at the back of the vector using push_back operation:

Code:
vector<int> v = {10,20,30,40};
v.push_back(50);
Now the new array will have values: {10,20,30,40,50} .

Popping the last element of the vector using pop_back operation:

Code:
vector<int> v = {10,20,30,40,50};
v.pop_back();
Now the new array will have values: {10,20,30,40} .

Insert an element somewhere in the vector's middle/ (in between the elements):

Syntax:
vector<int> v;
v.insert(position, value);
Position specifies to the iterator which points to the position where the insertion is to be done.

Example Code:
vector<int> v = {10,20,30,40,50};
v.insert(v.begin() + 2, 100);
The new vector will look like this:
{10,20,100,30,40,50}

Deleting an element in a vector at a specific position:

Syntax:
v.erase(position);

Example Code:
vector<int> v = {10,20,30,40,50};
v.erase(v.begin() + 2);

The new vector will look like this:
{10,20,40,50}

Clearing the vector : clear() function is used to remove all the elements of the vector container, thus making its size 0.

Code:

```
vector<int> v = {10,20,30,40,50};
v.clear();
```

Output:
Now v will be empty.

Looping in Vector

There are many ways to traverse through the elements of a vector. The most common and widely used ways of looping are:

For Loop:

```
#include<iostream>
using namespace std;
int main(){
vector<int> v;
for(int i = 0; i < 5; i++) {
v.push_back(i);
}
// now by the above loop v will have the values : {0,1,2,3,4}
for(int i = 0; i < v.size(); i++){
cout << v[i] << " ";
}
}
```

Output : 0 1 2 3 4

For each loop: This loop helps in iterating over iterable objects like string, array and so on.

```
vector<int> a = { 1, 2, 3, 4, 5, 6, 7, 8 };
for (int i : a) {
// accessing each element of the vector
cout << i << endl;
}
```

Output :
1
2

3
4
5
6
7
8

Explanation : Here, all the elements present in the array will be printed.

While loop:

```
int main(){
int i = 5;
vector<int> v;
while(i > 0) {
v.push_back(i--);
}
i = 0;
while(i < v.size()){
cout<<v[i]<<" ";
I++;
}
}
```

Output : 5 4 3 2 1

Explanation: Here, we initialize the value of 'i' to be 5 and keep decrementing it using the post-decrement operator and at each step insert it at the back of the vector. Therefore, in the end, the vector will contain elements from 5 to 0.

Q1. Given an array of marks of students, if the mark of any student is less than 35 print its roll number. [roll number here refers to the index of the array.]

```
#include <iostream>
#include <vector>

// Function to print roll numbers of students with marks less
than 35
void printFailingStudents(const std::vector<int>& marks) {
    std::cout << "Roll numbers of students with marks less than
35:" << std::endl;
    for (size_t i = 0; i < marks.size(); ++i) {
        if (marks[i] < 35) {
            std::cout << "Roll Number: " << i << std::endl;
        }
    }
}

int main() {
    std::vector<int> studentMarks;

    // Input marks for students
    int numStudents;
    std::cout << "Enter the number of students: ";
    std::cin >> numStudents;
```

```
    std::cout << "Enter the marks of students:" << std::endl;
    for (int i = 0; i < numStudents; ++i) {
        int mark;
        std::cout << "Enter the mark for student " << i << ": ";
        std::cin >> mark;
        studentMarks.push_back(mark);
    }

    // Call the function to print roll numbers of failing
students
    printFailingStudents(studentMarks);

    return 0;
}
```

Q2. Calculate the sum of all the elements in the given array.

```
#include <iostream>

// Function to calculate the sum of elements in an array
int calculateSum(const int arr[], int size) {
    int sum = 0;
    for (int i = 0; i < size; ++i) {
        sum += arr[i];
    }
    return sum;
}

int main() {
    const int size = 5; // Adjust the size based on your array
size
    int arr[size];

    // Input elements of the array
    std::cout << "Enter " << size << " elements of the array:" <<
std::endl;
    for (int i = 0; i < size; ++i) {
        std::cout << "Element " << i + 1 << ": ";
        std::cin >> arr[i];
    }

    // Call the function to calculate the sum
    int sum = calculateSum(arr, size);

    // Print the sum
    std::cout << "Sum of elements in the array: " << sum <<
std::endl;

    return 0;
}
```

Q3. Find the element x in the array . Take array and x as input.

```
#include <iostream>

// Function to search for an element x in the array
bool searchElement(const int arr[], int size, int x) {
    for (int i = 0; i < size; ++i) {
        if (arr[i] == x) {
            return true; // Element found
        }
    }
    return false; // Element not found
}

int main() {
    const int size = 5; // Adjust the size based on your array
size
    int arr[size];

    // Input elements of the array
    std::cout << "Enter " << size << " elements of the array:" <<
std::endl;
    for (int i = 0; i < size; ++i) {
        std::cout << "Element " << i + 1 << ": ";
        std::cin >> arr[i];
    }

    int x;

    // Input the element to search
    std::cout << "Enter the element to search: ";
    std::cin >> x;

    // Call the function to search for the element
    bool isElementFound = searchElement(arr, size, x);

    // Print the result
    if (isElementFound) {
        std::cout << "Element " << x << " is found in the array."
<< std::endl;
    } else {
        std::cout << "Element " << x << " is not found in the
array." << std::endl;
    }

    return 0;
}
```

Q4. Find the maximum value out of all the elements in the array.

```
#include <iostream>

// Function to find the maximum value in an array
int findMaxValue(const int arr[], int size) {
    // Assuming the array has at least one element
    int maxVal = arr[0];

    for (int i = 1; i < size; ++i) {
        if (arr[i] > maxVal) {
            maxVal = arr[i];
        }
    }

    return maxVal;
}

int main() {
    const int size = 5; // Adjust the size based on your array
size
    int arr[size];

    // Input elements of the array
    std::cout << "Enter " << size << " elements of the array:" <<
std::endl;
    for (int i = 0; i < size; ++i) {
        std::cout << "Element " << i + 1 << ": ";
        std::cin >> arr[i];
    }

    // Call the function to find the maximum value
    int maxVal = findMaxValue(arr, size);

    // Print the maximum value
    std::cout << "Maximum value in the array: " << maxVal <<
std::endl;

    return 0;
}
```

Q5. Find the second largest element in the given Array.

```
#include <iostream>

// Function to find the second largest element in an array
int findSecondLargest(const int arr[], int size) {
    // Assuming the array has at least two elements
    int firstLargest = arr[0];
    int secondLargest = arr[1];
```

```
    // Ensure firstLargest is greater than secondLargest
    if (firstLargest < secondLargest) {
        std::swap(firstLargest, secondLargest);
    }

    for (int i = 2; i < size; ++i) {
        if (arr[i] > firstLargest) {
            secondLargest = firstLargest;
            firstLargest = arr[i];
        } else if (arr[i] > secondLargest && arr[i] <
firstLargest) {
            secondLargest = arr[i];
        }
    }

    return secondLargest;
}

int main() {
    const int size = 5; // Adjust the size based on your array
size
    int arr[size];

    // Input elements of the array
    std::cout << "Enter " << size << " elements of the array:" <<
std::endl;
    for (int i = 0; i < size; ++i) {
        std::cout << "Element " << i + 1 << ": ";
        std::cin >> arr[i];
    }

    // Call the function to find the second largest element
    int secondLargest = findSecondLargest(arr, size);

    // Print the second largest element
    std::cout << "Second largest element in the array: " <<
secondLargest << std::endl;

    return 0;
}
```

Q6. Count the number of elements in given array greater than a given number x.

```
#include <iostream>

// Function to count the number of elements greater than x in an
array
int countElementsGreaterThanX(const int arr[], int size, int x) {
    int count = 0;
```

```
    for (int i = 0; i < size; ++i) {
        if (arr[i] > x) {
            count++;
        }
    }

    return count;
}

int main() {
    const int size = 5; // Adjust the size based on your array
size
    int arr[size];

    // Input elements of the array
    std::cout << "Enter " << size << " elements of the array:" <<
std::endl;
    for (int i = 0; i < size; ++i) {
        std::cout << "Element " << i + 1 << ": ";
        std::cin >> arr[i];
    }

    int x;

    // Input the value of x
    std::cout << "Enter the value of x: ";
    std::cin >> x;

    // Call the function to count elements greater than x
    int count = countElementsGreaterThanX(arr, size, x);

    // Print the result
    std::cout << "Number of elements greater than " << x << ": "
<< count << std::endl;

    return 0;
}
```

Q7. Find the last occurrence of x in the array.

```
#include <iostream>
#include <vector>

// Function to find the last occurrence of x in a vector
int findLastOccurrence(const std::vector<int>& vec, int x) {
    for (int i = vec.size() - 1; i >= 0; --i) {
        if (vec[i] == x) {
            return i; // Return the index of the last occurrence
        }
    }
```

```
    return -1; // Return -1 if x is not found in the vector
}

int main() {
    std::vector<int> vec;

    // Input elements of the vector
    int size;
    std::cout << "Enter the size of the vector: ";
    std::cin >> size;

    std::cout << "Enter elements of the vector:" << std::endl;
    for (int i = 0; i < size; ++i) {
        int element;
        std::cout << "Element " << i + 1 << ": ";
        std::cin >> element;
        vec.push_back(element);
    }

    int x;

    // Input the value of x
    std::cout << "Enter the value of x: ";
    std::cin >> x;

    // Call the function to find the last occurrence of x
    int lastOccurrence = findLastOccurrence(vec, x);

    // Print the result
    if (lastOccurrence != -1) {
        std::cout << "Last occurrence of " << x << " is at index:
" << lastOccurrence << std::endl;
    } else {
        std::cout << x << " not found in the vector." <<
std::endl;
    }

    return 0;
}
```

Q8. Find the doublet in the Array whose sum is equal to the given value x.

```
#include <iostream>
#include <vector>

// Function to find doublets with sum equal to x in a vector
void findDoublets(const std::vector<int>& vec, int x) {
    bool found = false;
```

```
    // Iterate through the vector
    for (size_t i = 0; i < vec.size() - 1; ++i) {
        for (size_t j = i + 1; j < vec.size(); ++j) {
            if (vec[i] + vec[j] == x) {
                std::cout << "Doublet found: (" << vec[i] << ", "
<< vec[j] << ")" << std::endl;
                found = true;
            }
        }
    }

    if (!found) {
        std::cout << "No doublet found with sum equal to " << x
<< "." << std::endl;
    }
}

int main() {
    std::vector<int> vec;

    // Input elements of the vector
    int size;
    std::cout << "Enter the size of the vector: ";
    std::cin >> size;

    std::cout << "Enter elements of the vector:" << std::endl;
    for (int i = 0; i < size; ++i) {
        int element;
        std::cout << "Element " << i + 1 << ": ";
        std::cin >> element;
        vec.push_back(element);
    }

    int x;

    // Input the value of x
    std::cout << "Enter the value of x: ";
    std::cin >> x;

    // Call the function to find doublets with sum equal to x
    findDoublets(vec, x);

    return 0;
}
```

Q9. Write a program to copy the contents of one array into another in the reverse order.

```
#include <iostream>
#include <vector>

// Function to copy vector elements in reverse order
void copyVectorReverse(const std::vector<int>& original,
std::vector<int>& reversed) {
    for (int i = original.size() - 1; i ⩾ 0; --i) {
        reversed.push_back(original[i]);
    }
}

int main() {
    std::vector<int> originalVector;
    std::vector<int> reversedVector;

    // Input elements of the original vector
    int size;
    std::cout << "Enter the size of the vector: ";
    std::cin >> size;

    std::cout << "Enter elements of the vector:" << std::endl;
    for (int i = 0; i < size; ++i) {
        int element;
        std::cout << "Element " << i + 1 << ": ";
        std::cin >> element;
        originalVector.push_back(element);
    }

    // Call the function to copy vector elements in reverse order
    copyVectorReverse(originalVector, reversedVector);

    // Print the reversed vector
    std::cout << "Reversed vector:" << std::endl;
    for (size_t i = 0; i < reversedVector.size(); ++i) {
        std::cout << reversedVector[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Q10. Write a program to copy the contents of one array into another in the reverse order.

```
#include <iostream>
#include <vector>

// Function to reverse a vector in-place
void reverseVector(std::vector<int>& vec) {
    size_t start = 0;
    size_t end = vec.size() - 1;
```

```
    while (start < end) {
        // Swap elements at start and end indices
        std::swap(vec[start], vec[end]);

        // Move indices towards the center
        ++start;
        --end;
    }
}

int main() {
    std::vector<int> myVector;

    // Input elements of the vector
    int size;
    std::cout << "Enter the size of the vector: ";
    std::cin >> size;

    std::cout << "Enter elements of the vector:" << std::endl;
    for (int i = 0; i < size; ++i) {
        int element;
        std::cout << "Element " << i + 1 << ": ";
        std::cin >> element;
        myVector.push_back(element);
    }

    // Call the function to reverse the vector in-place
    reverseVector(myVector);

    // Print the reversed vector
    std::cout << "Reversed vector:" << std::endl;
    for (size_t i = 0; i < myVector.size(); ++i) {
        std::cout << myVector[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Q11. Rotate the given array 'a' by k steps, where k is non-negative.

Note : k can be greater than n as well where n is the size of array 'a'.

```
#include <iostream>
#include <vector>

// Function to reverse a portion of the array
void reverseArray(std::vector<int>& arr, int start, int end) {
    while (start < end) {
```

```
        std::swap(arr[start], arr[end]);
        ++start;
        --end;
    }
}

// Function to rotate the array 'a' by 'k' steps
void rotateArray(std::vector<int>& arr, int k) {
    int n = arr.size();

    // Handle cases where k is greater than the array size
    k = k % n;

    // Reverse the first 'n-k' elements
    reverseArray(arr, 0, n - k - 1);

    // Reverse the last 'k' elements
    reverseArray(arr, n - k, n - 1);

    // Reverse the entire array
    reverseArray(arr, 0, n - 1);
}

int main() {
    std::vector<int> myArray;

    // Input elements of the array
    int size;
    std::cout << "Enter the size of the array: ";
    std::cin >> size;

    std::cout << "Enter elements of the array:" << std::endl;
    for (int i = 0; i < size; ++i) {
        int element;
        std::cout << "Element " << i + 1 << ": ";
        std::cin >> element;
        myArray.push_back(element);
    }

    int k;
    std::cout << "Enter the number of steps to rotate (k): ";
    std::cin >> k;

    // Call the function to rotate the array by k steps
    rotateArray(myArray, k);

    // Print the rotated array
    std::cout << "Rotated array:" << std::endl;
    for (size_t i = 0; i < myArray.size(); ++i) {
        std::cout << myArray[i] << " ";
    }
```

```
    std::cout << std::endl;

    return 0;
}
```

Q12. Sort the array of 0's and 1's

```
#include <iostream>
#include <vector>

// Function to sort an array of 0s and 1s
void sortZerosOnes(std::vector<int>& arr) {
    int left = 0;
    int right = arr.size() - 1;

    while (left < right) {
        // Move left pointer to the right until finding the first
'1'
        while (arr[left] == 0 && left < right) {
            left++;
        }

        // Move right pointer to the left until finding the first
'0'
        while (arr[right] == 1 && left < right) {
            right--;
        }

        // Swap elements at left and right pointers
        std::swap(arr[left], arr[right]);
        left++;
        right--;
    }
}

int main() {
    std::vector<int> myArray;

    // Input elements of the array
    int size;
    std::cout << "Enter the size of the array: ";
    std::cin >> size;

    std::cout << "Enter elements of the array (0 or 1 only):" <<
std::endl;
    for (int i = 0; i < size; ++i) {
        int element;
        std::cout << "Element " << i + 1 << ": ";
        std::cin >> element;
        myArray.push_back(element);
```

```
    }

    // Call the function to sort the array of 0s and 1s
    sortZerosOnes(myArray);

    // Print the sorted array
    std::cout << "Sorted array:" << std::endl;
    for (size_t i = 0; i < myArray.size(); ++i) {
        std::cout << myArray[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Q13. Move all negative numbers to beginning and positive to end with constant extra space.

```
#include <iostream>
#include <vector>

// Function to rearrange the array with negative numbers at the
beginning and positive at the end
void rearrangeNegativesPositives(std::vector<int>& arr) {
    int n = arr.size();
    int left = 0;
    int right = n - 1;

    while (left ⩽ right) {
        // Find the first positive number from the left
        while (left ⩽ right && arr[left] < 0) {
            left++;
        }

        // Find the first negative number from the right
        while (left ⩽ right && arr[right] ⩾ 0) {
            right--;
        }

        // Swap elements at left and right pointers
        if (left ⩽ right) {
            std::swap(arr[left], arr[right]);
            left++;
            right--;
        }
    }
}

int main() {
    std::vector<int> myArray;
```

```
    // Input elements of the array
    int size;
    std::cout << "Enter the size of the array: ";
    std::cin >> size;

    std::cout << "Enter elements of the array:" << std::endl;
    for (int i = 0; i < size; ++i) {
        int element;
        std::cout << "Element " << i + 1 << ": ";
        std::cin >> element;
        myArray.push_back(element);
    }

    // Call the function to rearrange the array
    rearrangeNegativesPositives(myArray);

    // Print the rearranged array
    std::cout << "Rearranged array:" << std::endl;
    for (size_t i = 0; i < myArray.size(); ++i) {
        std::cout << myArray[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Q14. Sort the array of 0's , 1's and 2's .

```
#include <iostream>
#include <vector>

// Function to sort an array of 0s, 1s, and 2s
void sortColors(std::vector<int>& arr) {
    int low = 0;
    int high = arr.size() - 1;
    int mid = 0;

    while (mid <= high) {
        switch (arr[mid]) {
            case 0:
                std::swap(arr[low], arr[mid]);
                low++;
                mid++;
                break;
            case 1:
                mid++;
                break;
            case 2:
                std::swap(arr[mid], arr[high]);
                high--;
```

```
                break;
            }
        }
    }

int main() {
    std::vector<int> myArray;

    // Input elements of the array
    int size;
    std::cout << "Enter the size of the array: ";
    std::cin >> size;

    std::cout << "Enter elements of the array (0, 1, or 2 only):"
<< std::endl;
    for (int i = 0; i < size; ++i) {
        int element;
        std::cout << "Element " << i + 1 << ": ";
        std::cin >> element;

        if (element < 0 || element > 2) {
            std::cout << "Invalid input. Please enter 0, 1, or 2
only." << std::endl;
            return 1; // Exit with an error code
        }

        myArray.push_back(element);
    }

    // Call the function to sort the array of 0s, 1s, and 2s
    sortColors(myArray);

    // Print the sorted array
    std::cout << "Sorted array:" << std::endl;
    for (size_t i = 0; i < myArray.size(); ++i) {
        std::cout << myArray[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Q15. Merge two sorted arrays .

```
#include <iostream>
#include <vector>

// Function to merge two sorted arrays
```

```
void mergeSortedArrays(const std::vector<int>& arr1, const
std::vector<int>& arr2, std::vector<int>& mergedArray) {
    int i = 0; // Pointer for arr1
    int j = 0; // Pointer for arr2
    int k = 0; // Pointer for mergedArray

    while (i < arr1.size() && j < arr2.size()) {
        if (arr1[i] ⩽ arr2[j]) {
            mergedArray[k++] = arr1[i++];
        } else {
            mergedArray[k++] = arr2[j++];
        }
    }

    // Copy the remaining elements of arr1, if any
    while (i < arr1.size()) {
        mergedArray[k++] = arr1[i++];
    }

    // Copy the remaining elements of arr2, if any
    while (j < arr2.size()) {
        mergedArray[k++] = arr2[j++];
    }
}

int main() {
    std::vector<int> arr1, arr2;

    // Input elements of the first sorted array
    int size1;
    std::cout << "Enter the size of the first sorted array: ";
    std::cin >> size1;

    std::cout << "Enter elements of the first sorted array:" <<
std::endl;
    for (int i = 0; i < size1; ++i) {
        int element;
        std::cout << "Element " << i + 1 << ": ";
        std::cin >> element;
        arr1.push_back(element);
    }

    // Input elements of the second sorted array
    int size2;
    std::cout << "Enter the size of the second sorted array: ";
    std::cin >> size2;

    std::cout << "Enter elements of the second sorted array:" <<
std::endl;
    for (int i = 0; i < size2; ++i) {
        int element;
```

```
        std::cout << "Element " << i + 1 << ": ";
        std::cin >> element;
        arr2.push_back(element);
    }

    // Create a vector to store the merged array
    std::vector<int> mergedArray(size1 + size2);

    // Call the function to merge two sorted arrays
    mergeSortedArrays(arr1, arr2, mergedArray);

    // Print the merged array
    std::cout << "Merged sorted array:" << std::endl;
    for (size_t i = 0; i < mergedArray.size(); ++i) {
        std::cout << mergedArray[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Q16. Find the next permutations of Array .

Note :- If not possible then print the sorted order in ascending order.

```
#include <iostream>
#include <algorithm>
#include <vector>

// Function to find the next permutation or print the sorted
order
void findNextPermutation(std::vector<int>& arr) {
    if (std::next_permutation(arr.begin(), arr.end())) {
        std::cout << "Next permutation: ";
        for (int element : arr) {
            std::cout << element << " ";
        }
        std::cout << std::endl;
    } else {
        std::cout << "No next permutation. Sorted order: ";
        std::sort(arr.begin(), arr.end());
        for (int element : arr) {
            std::cout << element << " ";
        }
        std::cout << std::endl;
    }
}

int main() {
    std::vector<int> myArray;
```

```
    // Input elements of the array
    int size;
    std::cout << "Enter the size of the array: ";
    std::cin >> size;

    std::cout << "Enter elements of the array:" << std::endl;
    for (int i = 0; i < size; ++i) {
        int element;
        std::cout << "Element " << i + 1 << ": ";
        std::cin >> element;
        myArray.push_back(element);
    }

    // Call the function to find the next permutation or print
the sorted order
    findNextPermutation(myArray);

    return 0;
}
```

Q17. trapping rain water problem

```
#include <iostream>
#include <vector>

int trapRainWater(const std::vector<int>& height) {
    int n = height.size();

    if (n <= 2) {
        return 0; // Cannot trap water with less than three bars
    }

    std::vector<int> leftMax(n, 0);
    std::vector<int> rightMax(n, 0);

    // Calculate the maximum height to the left of each bar
    leftMax[0] = height[0];
    for (int i = 1; i < n; ++i) {
        leftMax[i] = std::max(leftMax[i - 1], height[i]);
    }

    // Calculate the maximum height to the right of each bar
    rightMax[n - 1] = height[n - 1];
    for (int i = n - 2; i >= 0; --i) {
        rightMax[i] = std::max(rightMax[i + 1], height[i]);
    }

    int trappedWater = 0;
```

```
    // Calculate the trapped water above each bar
    for (int i = 0; i < n; ++i) {
        int minHeight = std::min(leftMax[i], rightMax[i]);
        trappedWater += std::max(0, minHeight - height[i]);
    }

    return trappedWater;
}

int main() {
    std::vector<int> height;

    // Input heights of the bars
    int size;
    std::cout << "Enter the number of bars: ";
    std::cin >> size;

    std::cout << "Enter the heights of the bars:" << std::endl;
    for (int i = 0; i < size; ++i) {
        int h;
        std::cout << "Height of bar " << i + 1 << ": ";
        std::cin >> h;
        height.push_back(h);
    }

    // Calculate and print the trapped rainwater
    int trappedWater = trapRainWater(height);
    std::cout << "Trapped rainwater: " << trappedWater << "
units." << std::endl;

    return 0;
}
```

PW SKILLS

THANK YOU !

PW SKILLS