**Contents**

[Input Buffer 2](#_Toc148736705)

[ String Input : 2](#_Toc148736706)

[1. Line Buffering : 2](#_Toc148736707)

[2. Stream Buffer: 2](#_Toc148736708)

[3. Input Buffer: 3](#_Toc148736709)

[ Common techniques to handle input buffer-related issues: 3](#_Toc148736710)

[Permutation 4](#_Toc148736711)-

[Number of Operations on Array 8](#_Toc148736712)

[STL 8](#_Toc148736713)

[ set <char> a; 8](#_Toc148736714)

[ a.insert(s[i]); 8](#_Toc148736715)

[ Sort; 8](#_Toc148736716)

[ Permutaion 8](#_Toc148736717)

[ Gcd: 10](#_Toc148736718)

# Input Buffer

## String Input :

* While using scanf to input a string > scanf(“%s”,string);

No “&” sign will be used.

* The function fflush(stdin) is used to flush the output buffer of the stream.

fflush(stdin);

getchar();

* Variable-length arrays were introduced in the C99 standard but are not a part of the C++ standard. Although some compilers might support it as an extension, it's generally not recommended to use them.
* In C++, input buffers are used to temporarily store data read from input streams, such as std::cin, before being processed. Understanding input buffers is important for handling input correctly and avoiding unexpected behavior. Here are the different kinds of input buffers in C++:

## Line Buffering :

* The default buffering mode for std::cin.
* Reads input line by line, delimited by the newline character ('\n').
* Line buffering occurs when std::cin encounters a newline character or when the buffer is full.
* Can be bypassed using std::cin.ignore() or std::cin.getline().

## Stream Buffer:

* The underlying buffer associated with input/output streams.
* Controlled by the standard library's streambuf class.
* Generally, you don't directly interact with the stream buffer unless you're implementing custom stream behavior.

## Input Buffer:

* A buffer used to hold characters temporarily while reading input.
* Managed by the input stream objects, such as std::cin.
* Can cause issues like incomplete or unexpected input if not handled properly.

## Common techniques to handle input buffer-related issues:

* std::cin.ignore(): Ignores a specific number of characters or until a specific delimiter is encountered. Useful to skip unwanted characters or consume newline characters.
* std::cin.get(): Reads a single character from the input stream.
* std::cin.getline(): Reads an entire line of input into a character array or std::string. It discards the newline character.
* std::cin >> variable: Reads input until whitespace is encountered, but does not consume the newline character. Consider using std::cin.ignore() to consume the newline if necessary.
* std::cin.clear(): Clears any error flags on the input stream.
* std::cin.sync(): Flushes the input buffer, discarding any unread characters.

# Permutation

**Swap Function:** Create a function to swap two elements in an array. This function will be used to generate permutations by swapping elements at different positions.

*Code :*

void swap(int &a, int &b) {

int temp = a;

a = b;

b = temp;

}

**Generate Permutations Function:** Create a recursive function that generates permutations by swapping elements and recursively processing the remaining part of the array.

*Code :*

void generate\_permutations(int arr[], int start, int end) {

if (start == end) {

// Print the permutation

for (int i = 0; i <= end; ++i) {

cout << arr[i] << " ";

}

cout << endl;

} else {

for (int i = start; i <= end; ++i) {

bool should\_swap = true;

// Check if arr[i] is already placed before start

for (int j = start; j < i; ++j) {

if (arr[j] == arr[i]) {

should\_swap = false;

break;

}

}

if (should\_swap) {

swap(arr[start], arr[i]); // Swap elements

generate\_permutations(arr, start + 1, end); // Recurse on the next index

swap(arr[start], arr[i]); // Swap back to backtrack

}

}

}

}

**Main Function:** In the main function, create an array of integers and call the generate\_permutations function to start generating permutations.

*Code :*

#include <iostream>

using namespace std;

int main() {

int input\_array[] = {1, 2, 3}; // Change this array as needed

int n = sizeof(input\_array) / sizeof(input\_array[0]);

generate\_permutations(input\_array, 0, n - 1);

return 0;

}

***Main Code :***

#include <iostream>

using namespace std;

*// Swap two elements in an array*

void swap(int &a, int &b) {

int temp = a;

a = b;

b = temp;

}

*// Recursive function to generate permutations*

void generate\_permutations(int arr[], int start, int end) {

*// Base case: When only one element is left to be considered*

if (start == end) {

// Print the permutation

for (int i = 0; i <= end; ++i) {

std::cout << arr[i] << " ";

}

std::cout << std::endl;

} else {

*// Loop through elements to swap and generate permutations*

for (int i = start; i <= end; ++i) {

bool should\_swap = true;

*// Check if arr[i] is already placed before start*

for (int j = start; j < i; ++j) {

if (arr[j] == arr[i]) {

should\_swap = false;

break;

}

}

*// If not a duplicate, proceed with swapping*

if (should\_swap) {

swap(arr[start], arr[i]); *// Swap elements*

generate\_permutations(arr, start + 1, end); *// Recurse on the next index*

swap(arr[start], arr[i]); *// Swap back to backtrack*

}

}

}

}

int main() {

int input\_array[] = {1, 2, 2}; *// Replace with your input array*

int n = sizeof(input\_array) / sizeof(input\_array[0]);

generate\_permutations(input\_array, 0, n - 1);

return 0;

}

# Number of Operations on Array

------------------- tbc….

# STL

## set <char> a;

* This line declares a variable named a, which is of type "set" and holds elements of type char.
* Stores unique elements in a sorted order.
* #include<set>

## a.insert(s[i]);

This line inserts a character s[i] into the set a

## Sort;

*String Class:*sort(str.begin(), str.end());

*Array :* sort(strArray, strArray + size);

## Permutaion

First sort the array - sort(arr, arr + n);

Then - next\_permutation(arr, arr + n));

Ex :

#include <iostream>

#include <algorithm

using namespace std

int main() {

int n;

cin >> n;

int arr[n];

for (int i = 0; i < n; ++i) {

cin >> arr[i];

}

sort(arr, arr + n); *// Ensure the array is sorted for permutations*

do {

for (int i = 0; i < n; ++i) {

cout << arr[i] << " ";

}

cout << endl;

} while (next\_permutation(arr, arr + n));

return 0;

}

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

int n;

cin >> n;

int arr[n];

for (int i = 0; i < n; ++i) {

cin >> arr[i];

}

sort(arr, arr + n); // Ensure the array is sorted for permutations

int num\_permutations = 0;

// Generate and count permutations using next\_permutation

do {

num\_permutations++;

} while (next\_permutation(arr, arr + n));

cout << "Number of permutations: " << num\_permutations << endl;

return 0;

}

## Gcd:

\_\_gcd(a,b);

The maximum GCD occurs when the two numbers have the maximum difference

int maxGCD = n / 2;

# Math

## Typecasting :

int number = static\_cast<int>(max);

To convert a string that represents an integer into an actual integer in C++, you can use the **std::stoi** function. Here's an example of how to do it:

#include <iostream>

#include <string>

int main() {

std::string s = "170";

int number = std::stoi(s);

std::cout << "The integer value is: " << number << std::endl;

return 0;

}

In this code, **std::stoi** is used to convert the string **s** into an integer **number**. After the conversion, you can use the **number** variable as an integer in your program.

* In a rectangular table ,

Number of cells = row \* column

Number of edges = 2 \* (r \* c) + r + c