Table of Content

Page Numbers

[Lab 01 2](#_Toc184686799)

[Lab 02 7](#_Toc184686800)

[Lab Programs 16](#_Toc184686801)

[Lab 03 19](#_Toc184686802)

[Lab Programs 28](#_Toc184686803)

[Lab 05 34](#_Toc184686804)

[Lab Programs 35](#_Toc184686805)

[Lab 06 42](#_Toc184686806)

[Lab Programs 44](#_Toc184686807)

[Lab 07 51](#_Toc184686808)

[Lab Programs 53](#_Toc184686809)

# **Lab 01**

**Introductory Lab**

**First C++ program:**

13

**Variables in C++:**

* A variable can be of any data type.
* Variables must be declared before they can be used.
* Multiple variables can also be declared at one time.
* The name of a variable can contain alphabets, digits, and an underscore but the name of a variable must start with an alphabet or an underscore.

Syntax to Declare a variable



**Data Types in C++:**

* **char:** Used to represent characters.
* **int:** Used to represent integral numbers.
* **float:** Used to represent decimal numbers up to 6-7 precision digits.
* **double:** Used to represent decimal numbers up to 15 precision digits.
* **void:** Used to represent the valueless entity.

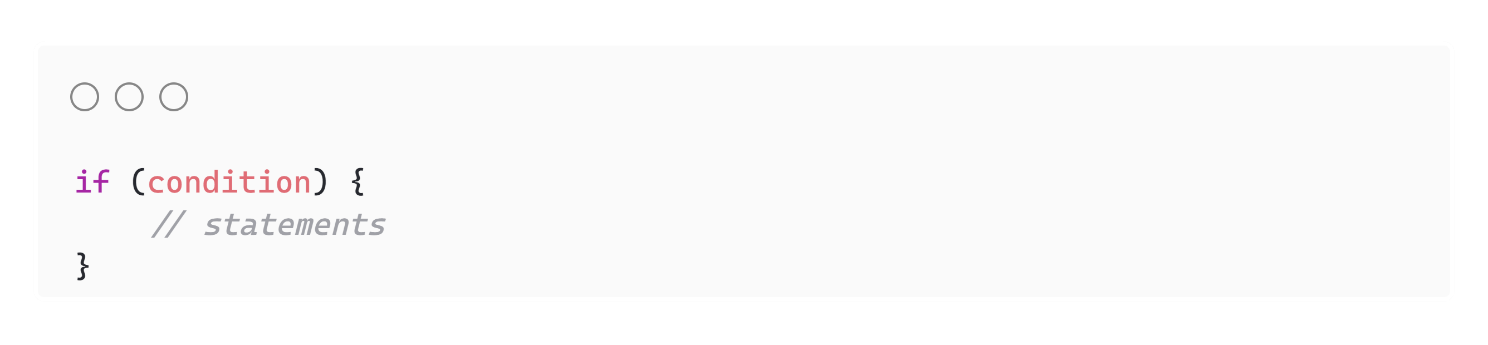
****

**Operators:**

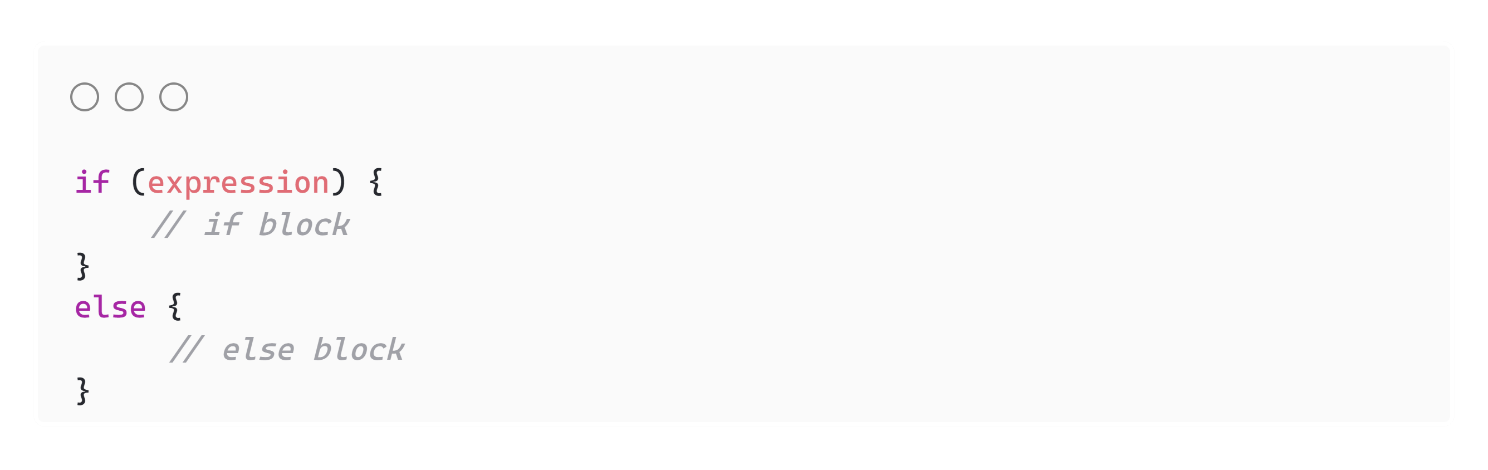
| **S.No.** | **Operator Type** | **Description** | **Example** |
| --- | --- | --- | --- |
| **1.** | **Arithmetic Operators** | Operators that perform arithmetic operations. | +, -, \*, /, % |
| **2.** | **Relational Operators** | They are used to compare two values. | <, >, <=, >=, ==, != |
| **3.** | **Bitwise Operators** | They are used to perform bit-level operations on integers. | &, ^, |, <<, >>, ~ |
| **4.** | **Logical Operators** | They perform logical operations such as logical AND, logical OR, etc. | &&, ||, ! |
| **5.** | **Conditional Operators** | The conditional Operator is used to insert conditional code. | ? : |
| **6.** | **Assignment Operators** | They are used to assign some value to the variables. | =, +=, -=, <<= |
| **7.** | **Miscellaneous Operators** | comma, addressof, sizeof, etc. are some other types of operators. | , sizeof, &, \*, ->, . |

**Conditional Statements:**

1. If Statement

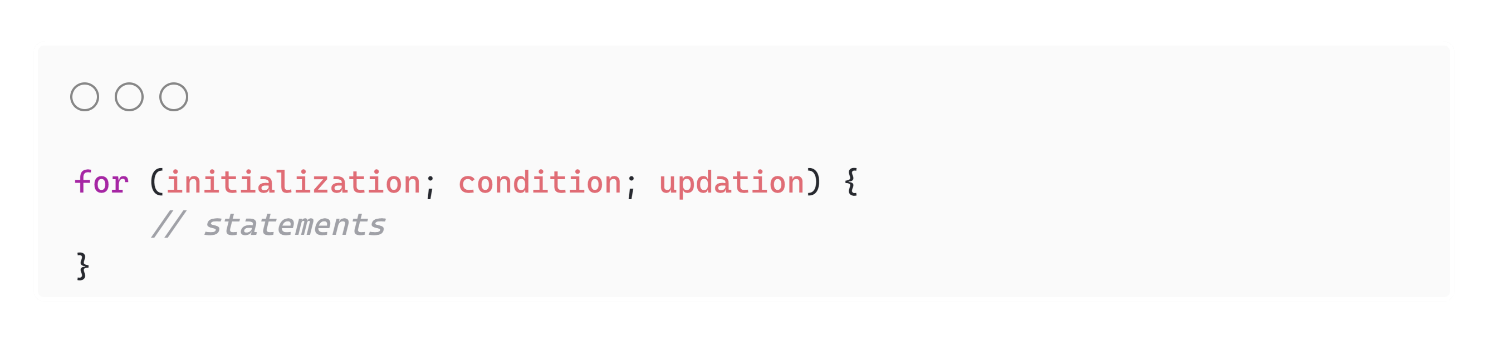


1. Else Statement

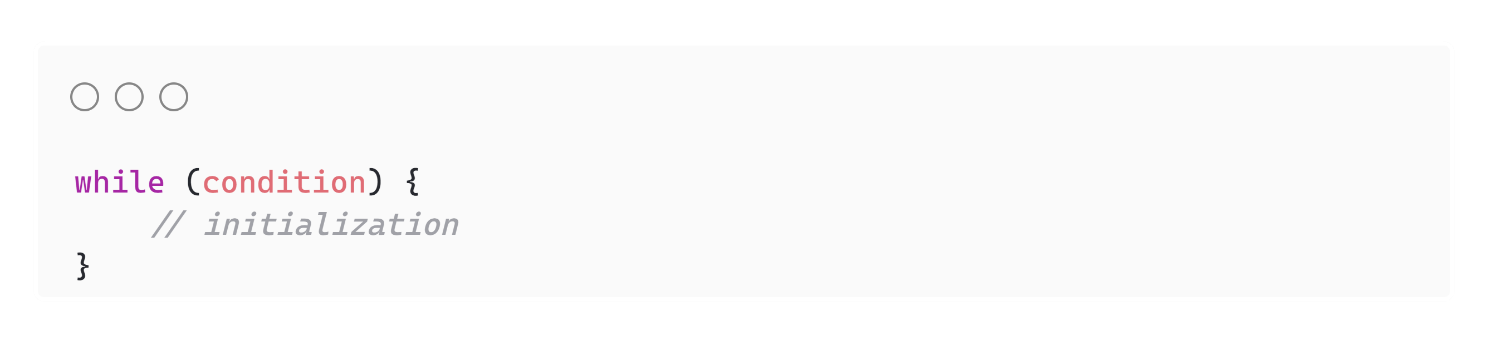
****

**Loops:**

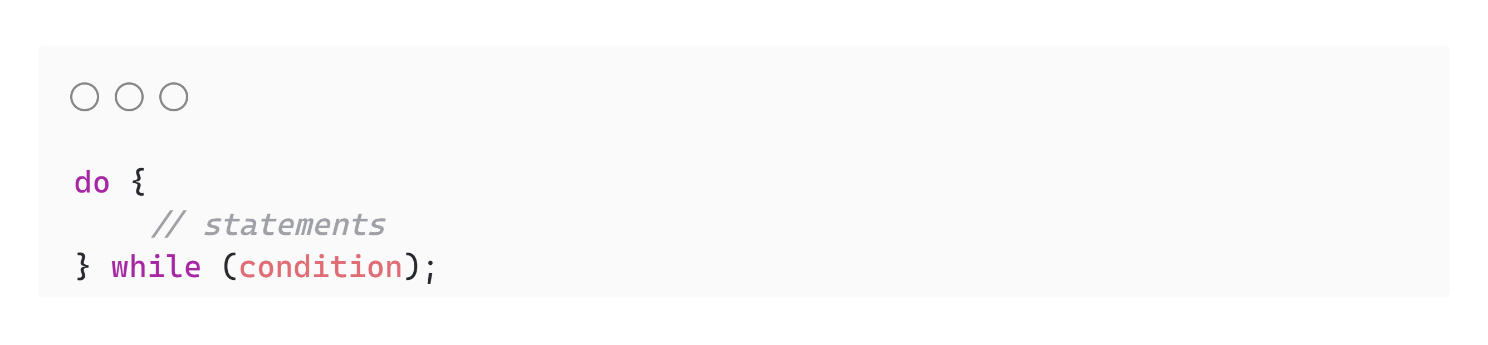
1. For Loop



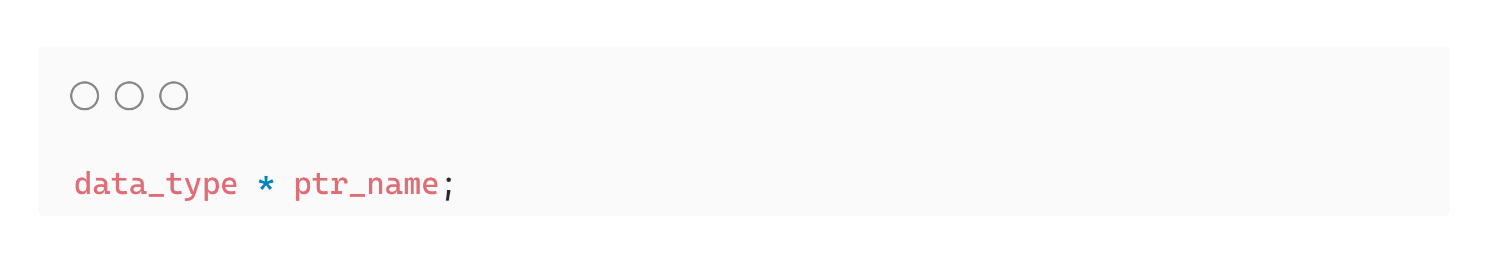
1. While Loop



1. Do While Loop



**Pointers:**

**** **Functions:**

**Functions Prototype:**

1. **Return Type:** It is the type of optional value returned by the function. Only one value can be returned.
2. **Parameters:** It is the data passed to the function by the caller.

****

**Function Definition:**



**Function Call:**

****

# **Lab 02**

**Introduction to Arrays and its Implementation**

Example: 1

#include <iostream>

#include <string>

using namespace std;

int main()

{

string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};

cout << cars[0];

return 0;

}

Explained:

#include <iostream>

// Includes the input/output stream library for using cout

#include <string>

// Includes the string library for using the string type

using namespace std;

// Allows us to avoid using the std:: prefix before standard library objects like cout

int main()

{

// Declare and initialize an array of strings named 'cars' with 4 elements

string cars[4] = {"Suzuki", "Corolla", "Ford", "Mazda"};

// Output the first element of the 'cars' array (which is "Volvo") to the console

cout << cars[0];

return 0; // Return 0 indicates that the program ended successfully

}

Example: 2

#include <iostream>

#include <string>

using namespace std;

int main()

{

string cars[5] = {"Volvo", "BMW", "Ford", "Mazda", "Tesla"};

for (int i = 0; i < 5; i++)

{

cout << cars[i] << "\n";

}

return 0;

}

Explained:

#include <iostream> // Includes the input/output stream library for using cout

#include <string> // Includes the string library for using the string type

using namespace std; // Allows us to avoid using the std:: prefix before standard library objects like cout

int main()

{

// Declare and initialize an array of strings named 'cars' with 5 elements

string cars[5] = {"Volvo", "BMW", "Ford", "Mazda", "Tesla"};

// Loop through each element in the array

// The loop starts at index 0 and runs until it reaches index 4 (less than 5)

for (int i = 0; i < 5; i++)

{

// Output each element in the 'cars' array followed by a newline character

cout << cars[i] << "\n";

}

return 0; // Return 0 indicates that the program ended successfully

}

Example: 3

#include <iostream>

#include <string>

using namespace std;

int main()

{

string cars[5] = {"Volvo", "BMW", "Ford", "Mazda", "Tesla"};

for (int i = 0; i < 5; i++)

{

cout << i << " = " << cars[i] << "\n";

}

return 0;

}

Explained:

#include <iostream> // Includes the input/output stream library for using cout

#include <string> // Includes the string library for using the string type

using namespace std; // Allows us to avoid using the std:: prefix before standard library objects like cout

int main()

{

// Declare and initialize an array of strings named 'cars' with 5 elements

string cars[5] = {"Volvo", "BMW", "Ford", "Mazda", "Tesla"};

// Loop through each element in the array

// The loop runs from index 0 to 4 (less than 5)

for (int i = 0; i < 5; i++)

{

// Output the index (i) and corresponding element in the 'cars' array

// Each line will print in the format: index = car\_name

cout << i << " = " << cars[i] << "\n";

}

return 0; // Return 0 indicates that the program ended successfully

}

Example: 4

#include <iostream>

using namespace std;

int main()

{

int myNumbers[5] = {10, 20, 30, 40, 50};

for (int i = 0; i < 5; i++)

{

cout << myNumbers[i] << "\n";

}

return 0;

}

Explained

#include <iostream> // Includes the input/output stream library for using cout

using namespace std; // Allows us to avoid using the std:: prefix before standard library objects like cout

int main()

{

// Declare and initialize an array of integers named 'myNumbers' with 5 elements

int myNumbers[5] = {10, 20, 30, 40, 50};

// Loop through each element in the 'myNumbers' array

// The loop runs from index 0 to 4 (less than 5)

for (int i = 0; i < 5; i++)

{

// Output each element in the 'myNumbers' array followed by a newline character

cout << myNumbers[i] << "\n";

}

return 0; // Return 0 indicates that the program ended successfully

}

Example: 5

#include <iostream>

#include <string>

using namespace std;

int main()

{

string cars[5];

cars[0] = "Volvo";

cars[1] = "BMW";

cars[2] = "Ford";

cars[3] = "Mazda";

cars[4] = "Tesla";

for (int i = 0; i < 5; i++)

{

cout << cars[i] << "\n";

}

return 0;

}

Explained

#include <iostream> // Includes the input/output stream library for using cout

#include <string> // Includes the string library for using the string type

using namespace std; // Allows us to avoid using the std:: prefix before standard library objects like cout

int main()

{

// Declare an array of strings named 'cars' with 5 elements

string cars[5];

// Assign values to each element of the 'cars' array

cars[0] = "Volvo";

cars[1] = "BMW";

cars[2] = "Ford";

cars[3] = "Mazda";

cars[4] = "Tesla";

// Loop through each element in the 'cars' array

// The loop runs from index 0 to 4 (less than 5)

for (int i = 0; i < 5; i++)

{

// Output each element in the 'cars' array followed by a newline character

cout << cars[i] << "\n";

}

return 0; // Return 0 indicates that the program ended successfully

}

Example: 6

#include <iostream>

using namespace std;

int main()

{

int myNumbers[5] = {10, 20, 30, 40, 50};

for (int i = 0; i < 5; i++)

{

cout << myNumbers[i] << "\n";

}

return 0;

}

Explained

#include <iostream> // Includes the input/output stream library for using cout

using namespace std; // Allows us to avoid using the std:: prefix before standard library objects like cout

int main()

{

// Declare and initialize an array of integers named 'myNumbers' with 5 elements

int myNumbers[5] = {10, 20, 30, 40, 50};

// Loop through each element in the 'myNumbers' array

// The loop runs from index 0 to 4 (less than 5)

for (int i = 0; i < 5; i++)

{

// Output each element in the 'myNumbers' array followed by a newline character

cout << myNumbers[i] << "\n";

}

return 0; // Return 0 indicates that the program ended successfully

}

Example: 7

#include <iostream>

using namespace std;

int main()

{

int myNumbers[5] = {10, 20, 30, 40, 50};

cout << sizeof(myNumbers);

return 0;

}

Explained

#include <iostream> // Includes the input/output stream library for using cout

using namespace std; // Allows us to avoid using the std:: prefix before standard library objects like cout

int main()

{

// Declare and initialize an array of integers named 'myNumbers' with 5 elements

int myNumbers[5] = {10, 20, 30, 40, 50};

// Output the size (in bytes) of the entire 'myNumbers' array

// The sizeof operator returns the total number of bytes occupied by the array

cout << sizeof(myNumbers);

return 0; // Return 0 indicates that the program ended successfully

}

Example: 8

#include <iostream>

using namespace std;

int main()

{

int numbers[5] = {10, 20, 30, 40, 50};

for (int i = 0; i < 5; i++)

{

cout << "Element at index " << i << " is: " << numbers[i] << endl;

}

return 0;

}

Explained

#include <iostream> // Include the input-output stream library for standard I/O

using namespace std; // Allows the use of standard names without the 'std::' prefix

int main()

{

// Initializing an array of 5 integers with predefined values

int numbers[5] = {10, 20, 30, 40, 50}; // Loop through the array elements

for (int i = 0; i < 5; i++)

{

// Output the current element with its index

cout << "Element at index " << i << " is: " << numbers[i] << endl;

}

return 0; // Return 0 to indicate successful program execution

}

Example: 9

#include <iostream>

using namespace std;

int main()

{

int arr[5], largest;

cout << "Enter 5 numbers:" << endl;

for (int i = 0; i < 5; i++)

{

cin >> arr[i];

}

largest = arr[0]; // Assume first element is the largest

for (int i = 1; i < 5; i++)

{

if (arr[i] > largest)

{

largest = arr[i];

}

}

cout << "The largest number is: " << largest << endl;

return 0;

}

Explained

#include <iostream> // Include the input-output stream library

using namespace std; // Allow usage of standard names without the 'std::' prefix

int main()

{

int arr[5], largest; // Declare an array to store 5 integers and a variable to hold the largest number

// Prompt the user to enter 5 numbers

cout

<< "Enter 5 numbers:" << endl;

// Loop to input 5 numbers from the user and store them in the array

for (int i = 0; i < 5; i++)

{

cin >> arr[i]; // Store each input in the corresponding index of the array

}

largest = arr[0]; // Assume the first element is the largest initially

// Loop through the remaining elements of the array to find the largest

for (int i = 1; i < 5; i++)

{

if (arr[i] > largest)

{

// If the current element is larger than 'largest'

largest = arr[i];

// Update 'largest' with the current element

}

}

// Output the largest number found

cout << "The largest number is: " << largest << endl;

return 0;

// Return 0 to indicate successful program execution

}

# Lab Programs

Self-Programs of Lab 2

Program No 1:

#include <iostream>

using namespace std;

int main()

{

string grocery[5] = {"bread", "jam", "milk", "butter", "rice"};

for (int i = 0; i < 5; i++)

{

cout << " " << grocery[i];

}

}

Output:

bread jam milk butter rice

Program No 2:

#include <iostream>

using namespace std;

int main ()

{

string subjects[7] = {"Urdu","English", "Math", "Physics", "Chemistry", "Islamiat", "Ai"};

int number[7] = {10,30,50,70,90,110,130};

for (int i = 0; i<7; i++)

{

cout << subjects[i] << " " << number[i];

}

}

Output:

Urdu 10English 30Math 50Physics 70Chemistry 90Islamiat 110Ai 130

Program No 3:

#include <iostream>

using namespace std;

int main()

{

string grocery[] = {"bread", "jam", "milk", "butter", "rice"};

cout << sizeof("jam");

}

Output:

4

Program No 4:

#include <iostream>

using namespace std;

int main()

{

int arr[10], n, x, count = 0;

cout << "Enter Size of Array: ";

cin >> n;

cout << "Enter Values of Array (in sorted order): ";

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

{

cin >> arr[i];

}

cout << "Enter Value to Search: ";

cin >> x;

int low = 0, high = n - 1, result = -1;

while (low <= high)

{

int mid = low + (high - low) / 2;

if (arr[mid] == x)

{

result = mid;

break;

}

else if (arr[mid] < x)

{

low = mid + 1;

}

else

{

high = mid - 1;

}

}

if (result != -1)

{

cout << "Value found at Index: " << result << endl;

for (int i = result - 1; i >= 0 && arr[i] == x; i--)

count++;

for (int i = result + 1; i < n && arr[i] == x; i++)

count++;

cout << "Value Duplicated in Array " << count << " Times";

}

else

{

cout << "Value not found in the array.";

}

return 0;

}

Output:

Enter Size of Array: 1

Enter Values of Array (in sorted order): 1

Enter Value to Search: 1

Value found at Index: 0

Value Duplicated in Array 0 Times

# **Lab 03**

**Implementation of Multi-Dimensional Array**

**Definition:**

A **multidimensional array** is an array of arrays, where each element is itself an array. In a **2D array**, elements are arranged in rows and columns, forming a matrix-like structure. This allows you to store data in a tabular form, making it ideal for scenarios like storing matrices, tables, or grids.

Example 1 (2D Array Declaration and Initialization):

#include <iostream>

using namespace std;

int main()

{

// Declare and initialize a 2x3 array (2 rows, 3 columns)

int matrix[2][3] = {

{1, 2, 3}, // First row

{4, 5, 6} // Second row

};

// Print the 2D array

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 3; j++)

{

cout << matrix[i][j] << " "; // Access elements using row and column indices

}

cout << endl; // Newline after each row

}

return 0;

}

Output

1 2 3

4 5 6

**Example 2** (**2D Array Declaration and Initialization**):

#include <iostream>

using namespace std;

int main()

{

// Declare and initialize a 3x2 array to store marks for 3 students in 2 subjects

int marks[3][2] = {

{80, 87}, // Marks for Student 1 in Subject 1 and Subject 2

{78, 82}, // Marks for Student 2 in Subject 1 and Subject 2

{97, 78} // Marks for Student 3 in Subject 1 and Subject 2

};

// Display the marks

for (int i = 0; i < 3; i++)

{

cout << "Student " << i + 1 << " Marks: ";

for (int j = 0; j < 2; j++)

{

cout << marks[i][j] << " "; // Print each student's marks

}

cout << endl;

}

return 0;

}

**Output**

Student 1 Marks: 80 87

Student 2 Marks: 78 82

Student 3 Marks: 97 78

**Example 3:**

#include <iostream>

using namespace std;

int main() {

// Declare and initialize a 3x2 array to store marks for 3 students in 2 subjects

int marks[6][2] = {

{85, 90}, // Marks for Student 1 in Subject 1 and Subject 2

{78, 82}, // Marks for Student 2 in Subject 1 and Subject 2

{92, 88}, // Marks for Student 3 in Subject 1 and Subject 2

{56, 33},

{66, 99},

{89, 88},

};

// Display the marks

for (int i = 0; i < 6; i++) {

cout << "Student " << i+1 << " Marks: ";

for (int j = 0; j < 2; j++) {

cout << marks[i][j] << " "; // Print each student's marks

}

cout << endl;

}

return 0;

}

**Output**

Student 1 Marks: 85 90

Student 2 Marks: 78 82

Student 3 Marks: 92 88

Student 4 Marks: 56 33

Student 5 Marks: 66 99

Student 6 Marks: 89 88

**Example 4:**

#include <iostream>

#include <vector> // Include the vector header

using namespace std;

int main()

{

// Create a vector to store integers

vector<int> numbers;

// Add elements to the vector

numbers.push\_back(10); // Add 10

numbers.push\_back(20); // Add 20

numbers.push\_back(30); // Add 30

// Display the elements of the vector

cout << "Vector elements: ";

for (int i = 0; i < numbers.size(); i++)

{ // Use size() to get the number of elements

cout << numbers[i] << " "; // Access elements using the index

}

cout << endl;

// Remove the last element

numbers.pop\_back(); // Removes 30

// Display the updated vector

cout << "After pop\_back, elements: ";

for (int i = 0; i < numbers.size(); i++)

{

cout << numbers[i] << " "; // Print updated vector

}

cout << endl;

return 0;

}

**Output**

Vector elements: 10 20 30

After pop\_back, elements: 10 20

Example 5:

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};

// Change the value of the first element

cars[0] = "Corolla";

cout << cars[0];

return 0;

}

Output

Corolla

**Example 6:**

In C++, the .at() function is a member function of the std::vector class that provides access to the elements of the vector. It allows you to retrieve or modify an element at a specific index while performing bounds checking.

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};

// Change the value of the first element

cars.at(0) = "Opel";

cout << cars.at(0);

return 0;

}

**Example 7:**

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};

cars.push\_back("Tesla");

for (string car : cars)

{

cout << car << "\n";

}

return 0;

}

**Example 8:**

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};

cars.push\_back("Tesla");

cars.push\_back("VW");

cars.push\_back("Mitsubishi");

cars.push\_back("Mini");

for (string car : cars)

{

cout << car << "\n";

}

return 0;

}

**Output**

Volvo

BMW

Ford

Mazda

Tesla

VW

Mitsubishi

Mini

**Example 9:**

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};

cars.pop\_back();

for (string car : cars)

{

cout << car << "\n";

}

return 0;

}

**Example 10:**

#include <iostream> // Include the standard input-output library

#include <vector> // Include the vector library to use the vector data structure

using namespace std; // Use the standard namespace for convenience

int main()

{

// Declare and initialize a vector of strings with car brand names

vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};

// Output the size of the vector using the .size() function

// This function returns the number of elements currently stored in the vector

cout << cars.size();

return 0; // Return 0 to indicate successful execution of the program

}

**Explanation**

* **#include <vector>:** This includes the vector library, allowing you to use the std::vector class.
* **vector<string> cars = {...};**: Initializes a vector of strings called cars with four elements.
* **cars.size():** The .size() function returns the number of elements in the vector, which in this case is 4.
* **cout << cars.size();:** Prints the size of the vector (4) to the console.

# **Lab Programs**

**Self-Programs of Lab 3**

**Program No 1:**

#include <iostream>

using namespace std;

int main()

{

float matrix[2][3] = {

{1.1, 2.2, 3.3},

{4.4, 5.5, 6.6}

};

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 3; j++)

{

cout << matrix[i][j] << " ";

}

cout << endl;

}

return 0;

}

**Output**

1.1 2.2 3.3

4.4 5.5 6.6

**Program No 2:**

#include <iostream>

using namespace std;

int main()

{

double marks[3][2] = {

{85.5, 90.2},

{78.4, 82.1},

{92.3, 88.7}

};

for (int i = 0; i < 3; i++)

{

cout << "Student " << i + 1 << " Marks: ";

for (int j = 0; j < 2; j++)

{

cout << marks[i][j] << " ";

}

cout << endl;

}

return 0;

}

**Output**

Student 1 Marks: 85.5 90.2

Student 2 Marks: 78.4 82.1

Student 3 Marks: 92.3 88.7

**Program No 3:**

#include <iostream>

using namespace std;

int main() {

float marks[6][2] = {

{85.5, 90.0},

{78.3, 82.7},

{92.1, 88.4},

{56.8, 33.5},

{66.2, 99.9},

{89.6, 88.3}

};

for (int i = 0; i < 6; i++) {

cout << "Student " << i+1 << " Marks: ";

for (int j = 0; j < 2; j++) {

cout << marks[i][j] << " ";

}

cout << endl;

}

return 0;

}

**Output**

Student 1 Marks: 85.5 90

Student 2 Marks: 78.3 82.7

Student 3 Marks: 92.1 88.4

Student 4 Marks: 56.8 33.5

Student 5 Marks: 66.2 99.9

Student 6 Marks: 89.6 88.3

**Program No 4:**

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<string> words;

words.push\_back("Hello");

words.push\_back("World");

words.push\_back("C++");

cout << "Vector elements: ";

for (int i = 0; i < words.size(); i++)

{

cout << words[i] << " ";

}

cout << endl;

words.pop\_back();

cout << "After pop\_back, elements: ";

for (int i = 0; i < words.size(); i++)

{

cout << words[i] << " ";

}

cout << endl;

return 0;

}

**Output**

Vector elements: Hello World C++

After pop\_back, elements: Hello World

**Program No 5:**

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<char> letters = {'A', 'B', 'C', 'D'};

letters[0] = 'Z';

cout << letters[0];

return 0;

}

**Output**

Z

**Program No 6:**

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<float> numbers = {1.1, 2.2, 3.3, 4.4};

numbers.at(0) = 5.5;

cout << numbers.at(0);

return 0;

}

**Output**

5.5

**Program No 7:**

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<char> letters = {'A', 'B', 'C', 'D'};

letters.push\_back('E');

for (char letter : letters)

{

cout << letter << "\n";

}

return 0;

}

**Output**

A

B

C

D

E

**Program No 8:**

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<int> numbers = {1, 2, 3, 4};

numbers.push\_back(5);

numbers.push\_back(6);

numbers.push\_back(7);

numbers.push\_back(8);

for (int number : numbers)

{

cout << number << "\n";

}

return 0;

}

**Output**

1

2

3

4

5

6

7

8

**Program No 9:**

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<double> prices = {19999.99, 25999.99, 29999.99, 34999.99};

prices.pop\_back();

for (double price : prices)

{

cout << price << "\n";

}

return 0;

}

**Output**

20000

26000

30000

**Program No 10:**

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<bool> flags = {true, false, true, false};

cout << flags.size();

return 0;

}

**Output**

4

# **Lab 05**

**Implementation of Multi-Dimensional Array**

**Two-Dimensional Array:**

A multi-dimensional array is an array of arrays.

To declare a multi-dimensional array, define the variable type, specify the name of the array followed by square brackets which specify how many elements the main array has, followed by another set of square brackets which indicates how many elements the sub-arrays have:

string letters[2][4] = {

{ "A", "B", "C", "D" },

{ "E", "F", "G", "H" }

};

Three-Dimensional Array

Arrays can have any number of dimensions. The more dimensions an array has, the more complex the code becomes. The following array has three dimensions:

string letters[2][2][2] = {

{

{ "A", "B" },

{ "C", "D" }

},

{

{ "E", "F" },

{ "G", "H" }

}

};

Access the Elements of a Multi-Dimensional Array

To access an element of a multi-dimensional array, specify an index number in each of the array's dimensions.

This statement accesses the value of the element in the **first row (0)** and **third column (2)** of the **letters** array.

string letters[2][4] = {

{ "A", "B", "C", "D" },

{ "E", "F", "G", "H" }

};

cout << letters[0][2]; // Outputs "C"

**Loop Through a Multi-Dimensional Array**

To loop through a multi-dimensional array, you need one loop for each of the array's dimensions.

The following example outputs all elements in the letters array:

string letters[2][4] = {

{ "A", "B", "C", "D" },

{ "E", "F", "G", "H" }

};

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 4; j++) {

cout << letters[i][j] << "\n";

}

}

# **Lab Programs**

**Self-Programs of Lab 5**

**Program No 1:**

**Integers with Addition**

#include <iostream>

using namespace std;

void printAndAdd(int nums[2][2][2]) {

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

for (int k = 0; k < 2; k++) {

nums[i][j][k] += 10; // Add 10 to each element

cout << nums[i][j][k] << "\n";

}

}

}

}

int main() {

int nums[2][2][2] = {

{ {1, 2}, {3, 4} },

{ {5, 6}, {7, 8} }

};

printAndAdd(nums);

return 0;

}

**Output:**

11

12

13

14

15

16

17

18

**Program No 2:**

**Floating-Point Numbers with Multiplication**

#include <iostream>

using namespace std;

void printAndMultiply(float nums[2][2][2])

{

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

for (int k = 0; k < 2; k++)

{

nums[i][j][k] \*= 2.5; // Multiply each element by 2.5

cout << nums[i][j][k] << "\n";

}

}

}

}

int main()

{

float nums[2][2][2] = {

{{1.1, 2.2}, {3.3, 4.4}},

{{5.5, 6.6}, {7.7, 8.8}}};

printAndMultiply(nums);

return 0;

}

Output:

2.75

5.5

8.25

11

13.75

16.5

19.25

22

**Program No 3:**

**Characters with Uppercase Conversion**

#include <iostream>

using namespace std;

void printUppercase(char chars[2][2][2])

{

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

for (int k = 0; k < 2; k++)

{

chars[i][j][k] = toupper(chars[i][j][k]); // Convert to uppercase

cout << chars[i][j][k] << "\n";

}

}

}

}

int main()

{

char chars[2][2][2] = {

{{'a', 'b'}, {'c', 'd'}},

{{'e', 'f'}, {'g', 'h'}}};

printUppercase(chars);

return 0;

}

**Output:**

A

B

C

D

E

F

G

H

**Program No 4:**

**Characters with Uppercase Conversion**

#include <iostream>

using namespace std;

void printAndConcat(string words[2][2][2])

{

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

for (int k = 0; k < 2; k++)

{

words[i][j][k] += "!";

cout << words[i][j][k] << "\n";

}

}

}

}

int main()

{

string words[2][2][2] = {

{{"Hello", "World"}, {"Good", "Morning"}},

{{"C++", "Programming"}, {"Nested", "Arrays"}}};

printAndConcat(words);

return 0;

}

**Output:**

Hello!

World!

Good!

Morning!

C++!

Programming!

Nested!

Arrays!

**Program No 5:**

**Booleans with Logical Negation**

#include <iostream>

using namespace std;

void printAndNegate(bool flags[2][2][2])

{

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

for (int k = 0; k < 2; k++)

{

flags[i][j][k] = !flags[i][j][k]; // Negate the boolean value

cout << flags[i][j][k] << "\n";

}

}

}

}

int main()

{

bool flags[2][2][2] = {

{{true, false}, {true, true}},

{{false, false}, {true, false}}};

printAndNegate(flags);

return 0;

}

**Output:**

0

1

0

0

1

1

0

1

**Program No 6:**

**Double with Arithmetic Average**

#include <iostream>

using namespace std;

void printAndAverage(double values[2][2][2])

{

double total = 0.0;

int count = 0;

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

for (int k = 0; k < 2; k++)

{

total += values[i][j][k];

count++;

cout << values[i][j][k] << "\n";

}

}

}

cout << "Average: " << total / count << "\n";

}

int main()

{

double values[2][2][2] = {

{{1.1, 2.2}, {3.3, 4.4}},

{{5.5, 6.6}, {7.7, 8.8}}};

printAndAverage(values);

return 0;

}

**Output:**

1.1

2.2

3.3

4.4

5.5

6.6

7.7

8.8

Average: 4.95

# **Lab 06**

**Vector**

**Program No 1:**

#include <iostream>

using namespace std;

struct Vector

{

int \*data; // Dynamic array to store integers

int size; // Current number of elements

int capacity; // Maximum capacity

// Initialize the vector

void initialize()

{

size = 0;

capacity = 2; // Start with a small capacity

data = new int[capacity]; // Allocate memory

}

// Add an integer to the vector

void push(int value)

{

if (size == capacity)

{

// Increase capacity if the vector is full

capacity \*= 2;

int \*newData = new int[capacity];

// Copy old elements to the new array

for (int i = 0; i < size; i++)

{

newData[i] = data[i];

}

delete[] data; // Free the old memory

data = newData;

}

// Add the new integer

data[size] = value;

size++;

cout << "Pushed " << value << " to the vector.\n";

}

// Remove the last element from the vector

void pop()

{

if (size > 0)

{

cout << "Popped " << data[size - 1] << " from the vector.\n";

size--; // Reduce the size

}

else

{

cout << "Vector is empty. Nothing to pop.\n";

}

}

// Free the memory allocated for the vector

void destroy()

{

delete[] data;

data = nullptr;

size = 0;

capacity = 0;

}

};

int main()

{

Vector numbers;

numbers.initialize(); // Initialize the vector

// Perform push operations

numbers.push(10);

numbers.push(20);

numbers.push(30);

// Perform pop operations

numbers.pop();

numbers.pop();

numbers.pop();

numbers.pop(); // Attempt to pop from an empty vector

// Clean up memory

numbers.destroy();

return 0;

}

**Output:**

Pushed 10 to the vector.

Pushed 20 to the vector.

Pushed 30 to the vector.

Popped 30 from the vector.

Popped 20 from the vector.

Popped 10 from the vector.

Vector is empty. Nothing to pop.

**Program No 2:**

#include <iostream>

using namespace std;

struct CharVector

{

char \*data;

int size;

int cap;

void initialize()

{

size = 0;

cap = 2;

data = new char[cap];

}

void pushBack(char value)

{

if (size == cap)

{

cap \*= 2;

char \*n = new char[cap];

for (int i = 0; i < size; i++)

{

n[i] = data[i];

}

delete[] data;

data = n;

}

data[size++] = value;

cout << "Pushed '" << value << "' into the vector.\n";

}

void popBack()

{

if (size > 0)

{

cout << "Popped '" << data[size - 1] << "' from the vector.\n";

size--;

}

else

{

cout << "Vector is empty. Cannot pop.\n";

}

}

char peek()

{

if (size > 0)

{

return data[size - 1];

}

else

{

cout << "Vector is empty. Nothing to peek.\n";

return '\0'; // Return null character if empty

}

}

void destroy()

{

delete[] data;

data = nullptr;

size = 0;

cap = 0;

}

};

int main()

{

CharVector vec;

vec.initialize();

vec.pushBack('A');

vec.pushBack('B');

vec.pushBack('C');

cout << "Last element in the vector: '" << vec.peek() << "'\n";

vec.popBack();

vec.popBack();

vec.destroy();

return 0;

}

**Program No 3:**

#include <iostream>

using namespace std;

// A simple dynamic vector to store boolean values

struct VectorBool {

bool\* data; // Pointer to dynamically allocated array

int size; // Current number of elements

int capacity; // Maximum capacity of the array

// Initialize the vector

void initialize() {

size = 0;

capacity = 2; // Start with a small capacity

data = new bool[capacity]; // Allocate initial memory

}

// Add a boolean value to the end of the vector

void push\_back(bool value) {

// Check if we need to increase capacity

if (size == capacity) {

capacity \*= 2; // Double the capacity

bool\* newData = new bool[capacity]; // Allocate larger memory

// Copy existing data to the new array

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

newData[i] = data[i];

}

delete[] data; // Free old memory

data = newData; // Point to the new memory

}

data[size] = value; // Add the new value

size++; // Increase the size

}

// Remove the last element

void pop\_back() {

if (size > 0) {

size--; // Reduce the size (no need to delete explicitly)

}

}

// Get the first element (safe access)

bool front() const {

if (size > 0) {

return data[0];

}

return false; // Return false if empty

}

// Get the last element (safe access)

bool back() const {

if (size > 0) {

return data[size - 1];

}

return false; // Return false if empty

}

// Get an element at a specific index (safe access)

bool at(int index) const {

if (index >= 0 && index < size) {

return data[index];

}

return false; // Return false if index is invalid

}

// Get the current size of the vector

int getSize() const {

return size;

}

// Check if the vector is empty

bool empty() const {

return size == 0;

}

// Free the allocated memory

void destroy() {

delete[] data; // Release memory

data = nullptr; // Avoid dangling pointers

size = 0;

capacity = 0;

}

};

// Custom function to print a bool as "true" or "false"

void printBool(bool value) {

if (value) {

cout << "true";

} else {

cout << "false";

}

}

int main() {

VectorBool vec; // Create a vector

vec.initialize(); // Initialize it

// Add some values

vec.push\_back(true);

vec.push\_back(false);

vec.push\_back(true);

// Print elements using the custom function

cout << "First Element: ";

printBool(vec.front());

cout << endl;

cout << "Last Element: ";

printBool(vec.back());

cout << endl;

cout << "Element at index 1: ";

printBool(vec.at(1));

cout << endl;

cout << "Size of vector: " << vec.getSize() << endl;

cout << "Is vector empty? ";

printBool(vec.empty());

cout << endl;

// Remove the last element and print the size again

vec.pop\_back();

cout << "Size after pop: " << vec.getSize() << endl;

// Clean up resources

vec.destroy();

return 0;

}

**Output:**

First Element: true

Last Element: true

Element at index 1: false

Size of vector: 3

Is vector empty? false

Size after pop: 2

# **Lab 07**

**Stack**

A stack is a linear data structure in which all the insertion and deletion operations are performed only at one end, called the **top** of the stack. It works on the principle of **Last In First Out (LIFO)**. This means that the last item put on the stack is the first item that can be taken off.

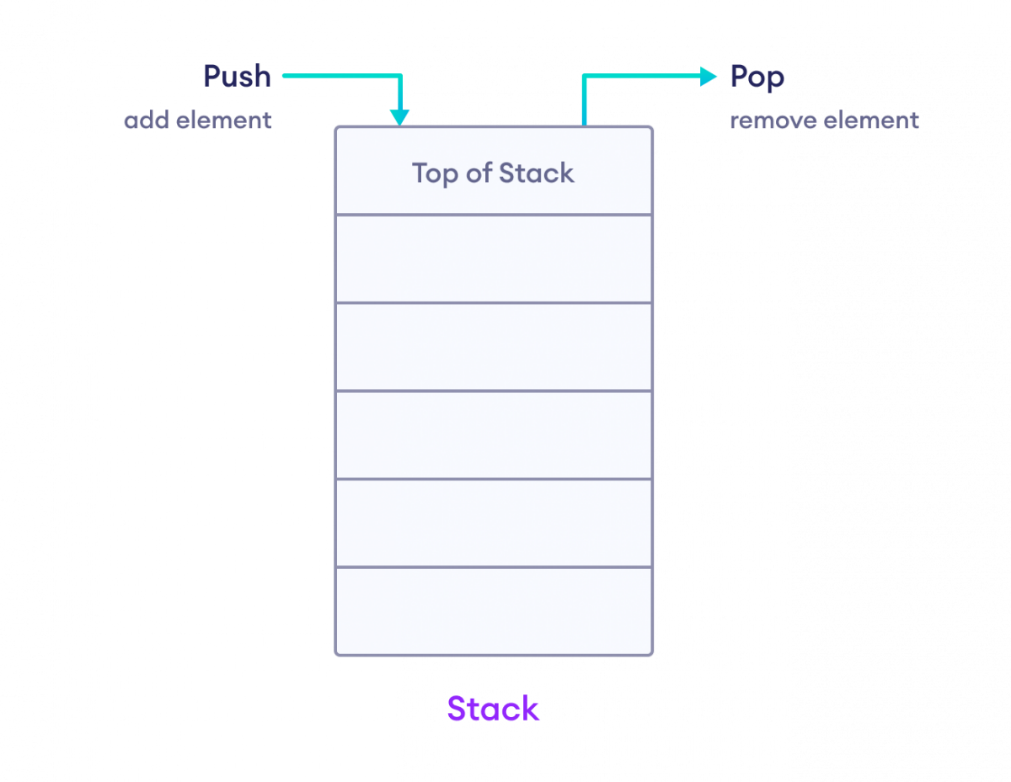
The fundamental operations on stack are:

* PUSH - To add an element to the top of the stack.
* POP - To remove an element from the top of the stack.
* PEEK - To see the topmost element of stack.

A stack can be implemented in the following ways:

* Using one-dimensional arrays.
* Using linked lists.

Note: *When implemented using linked lists, the size of the stack never becomes full.*



#include <iostream>

using namespace std;

class stack

{

private:

int \*arr;

int capacity;

int top;

public:

stack(int a)

{

capacity = a;

arr = new int[capacity];

top = -1;

}

void push(int a)

{

if (top == capacity - 1)

{

cout << "stack overflow";

return;

}

arr[++top] = a;

cout << a << "pushed";

}

int pop()

{

if (top == -1)

{

cout << "Empty";

return -1;

}

int popedvalue = arr[top--];

return popedvalue;

}

};

int main() {

// Create a stack with a maximum capacity of 5 elements

stack s(5);

// Test stack operations

s.push(10);

s.push(20);

s.push(30);

s.push(40);

s.push(50);

cout << "Popped element: " << s.pop() << "\n";

cout << "Popped element: " << s.pop() << "\n";

return 0;

}

# Lab Programs

Self-Programs of Lab 7

Program No 1:

#include <iostream>

using namespace std;

class stack {

private:

int \*arr;

int capacity;

int top;

public:

stack(int a) {

capacity = a;

arr = new int[capacity];

top = -1;

}

// Push function with a different style of overflow check

void push(int a) {

if (top < capacity - 1) {

arr[++top] = a;

cout << a << " pushed\n";

} else {

cout << "Stack is full! Cannot push " << a << "\n";

}

}

// Display function after each push

void display() {

if (top == -1) {

cout << "Stack is empty\n";

} else {

cout << "Stack content: ";

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

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

}

cout << "\n";

}

}

};

int main() {

stack s(5);

s.push(10);

s.push(20);

s.push(30);

s.push(40);

s.push(50);

// Display stack after pushing

s.display();

return 0;

}

Output:

10 pushed

20 pushed

30 pushed

40 pushed

50 pushed

Stack content: 10 20 30 40 50

Program No 2:

#include <iostream>

using namespace std;

class stack {

private:

int \*arr;

int capacity;

int top;

public:

stack(int a) {

capacity = a;

arr = new int[capacity];

top = -1;

}

// Pop function that handles underflow condition differently

int pop() {

if (top == -1) {

cout << "Stack is empty. Cannot pop.\n";

return -1;

}

int poppedValue = arr[top--];

cout << poppedValue << " popped\n";

return poppedValue;

}

// Display function after each pop

void display() {

if (top == -1) {

cout << "Stack is empty\n";

} else {

cout << "Current stack: ";

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

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

}

cout << "\n";

}

}

};

int main() {

stack s(5);

s.push(10);

s.push(20);

s.push(30);

s.push(40);

s.push(50);

s.display();

// Pop elements from the stack

s.pop();

s.pop();

// Display stack after popping

s.display();

return 0;

}

Output:

10 pushed

20 pushed

30 pushed

40 pushed

50 pushed

Stack content: 10 20 30 40 50

Program No 3:

#include <iostream>

using namespace std;

class stack {

private:

char \*arr;

int capacity;

int top;

public:

stack(int a) {

capacity = a;

arr = new char[capacity];

top = -1;

}

// Push function to add a character to the stack

void push(char a) {

if (top == capacity - 1) {

cout << "Stack overflow\n";

return;

}

arr[++top] = a;

cout << a << " pushed\n";

}

// Pop function to remove and return the top element

char pop() {

if (top == -1) {

cout << "Stack is empty\n";

return -1; // Return an invalid value if empty

}

char poppedValue = arr[top--];

cout << poppedValue << " popped\n";

return poppedValue;

}

// Display function to show the stack contents

void display() {

if (top == -1) {

cout << "Stack is empty\n";

} else {

cout << "Current stack: ";

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

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

}

cout << "\n";

}

}

};

int main() {

stack s(5);

// Push characters "H", "A", "M", "N", "A" into the stack

s.push('H');

s.push('A');

s.push('M');

s.push('N');

s.push('A');

// Display the stack after pushing

s.display();

// Pop elements until we pop "H"

cout << "Popping elements until we reach 'H':\n";

s.pop(); // Pops 'A'

s.pop(); // Pops 'N'

s.pop(); // Pops 'M'

s.pop(); // Pops 'A'

s.pop(); // Pops 'H'

return 0;

}

Output:

H pushed

A pushed

M pushed

N pushed

A pushed

Current stack: H A M N A

Popping elements until we reach 'H':

A popped

N popped

M popped

A popped

H popped

Program No 4:

#include <iostream>

#include <string>

using namespace std;

class stack

{

private:

string \*arr;

int capacity;

int top;

public:

stack(int a)

{

capacity = a;

arr = new string[capacity];

top = -1;

}

// Push function to add a string to the stack

void push(string a)

{

if (top == capacity - 1)

{

cout << "Stack overflow\n";

return;

}

arr[++top] = a;

cout << a << " pushed\n";

}

// Pop function to remove and return the top element

string pop()

{

if (top == -1)

{

cout << "Stack is empty\n";

return "";

}

string poppedValue = arr[top--];

cout << poppedValue << " popped\n";

return poppedValue;

}

// Display function to show the stack contents

void display()

{

if (top == -1)

{

cout << "Stack is empty\n";

}

else

{

cout << "Current stack: ";

for (int i = 0; i <= top; i++)

{

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

}

cout << "\n";

}

}

};

int main()

{

stack s(5);

// Push strings into the stack

s.push("hamna");

s.push("fizza");

s.push("maham");

s.push("tehreem");

// Display the stack after pushing

s.display();

// Pop elements until we reach "maham"

cout << "Popping elements until we reach 'maham':\n";

s.pop(); // Pops 'tehreem'

s.pop(); // Pops 'maham'

// Stop popping once we reach "maham"

return 0;

}

Output:

hamna pushed

fizza pushed

maham pushed

tehreem pushed

Current stack: hamna fizza maham tehreem

Popping elements until we reach 'maham':

tehreem popped

maham popped