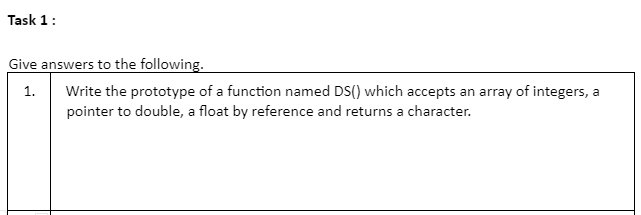
Submitted By: Zainab

Submitted To: Sir Rehan

Roll no: 2022-BSE-076

DS LAB 01

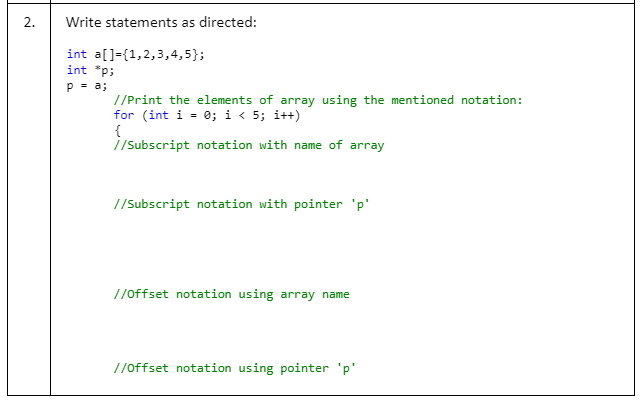


**char DS(int arr[], double \*ptrToDouble, float &floatByReference);**

**“arr”: An array of integers.**

**“ptrToDouble”: A pointer to a double.**

**“floatByReference”: A float passed by reference.**



**int a[] = {1, 2, 3, 4, 5};**

**int \*p;**

**p = a;**

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

**// Subscript notation with name of array**

**int subscriptNotationArray = a[i];**

**// Subscript notation with pointer 'p'**

**int subscriptNotationPointer = p[i];**

**// Offset notation using array name**

**int offsetNotationArray = \*(a + i);**

**// Offset notation using pointer 'p'**

**int offsetNotationPointer = \*(p + i);**

**// Print the values obtained from different notations**

**printf("Subscript notation with array: %d\n", subscriptNotationArray);**

**printf("Subscript notation with pointer 'p': %d\n", subscriptNotationPointer);**

**printf("Offset notation using array name: %d\n", offsetNotationArray);**

**printf("Offset notation using pointer 'p': %d\n", offsetNotationPointer);**

**}**

**Function Overloading:**

* **Definition: Function overloading allows you to define multiple functions in the same scope with the same name but different parameter lists (number or type of parameters).**
* **Overloaded functions have the same name but different parameters.**
* **The compiler determines which overloaded function to call based on the number and types of arguments passed.**
* **Overloading is resolved at compile-time and is also known as compile-time polymorphism or static polymorphism.**
* **Example:**

**void print(int num);**

**void print(float num);**

**void print(string text);**

**Function Overriding:**

* **Function overriding occurs in inheritance when a derived class provides a specific implementation for a function that is already defined in its base class. The function signatures (name and parameters) in the base and derived classes must match.**
* **Function overriding is used to achieve runtime polymorphism or dynamic polymorphism.**
* **The base class function must be declared as virtual (or part of an interface in Java) to enable overriding in derived classes.**
* **function overloading is about defining multiple functions with the same name in the same scope but with different parameters, while function overriding is about providing a new implementation for a base class function in a derived class to achieve polymorphism. Overloading is resolved at compile-time, whereas overriding is resolved at runtime.**
* **Example:**

**class Base {**

**public:**

**virtual void display() {**

**cout << "Base class display()" << endl;**

**}**

**};**

**class Derived : public Base {**

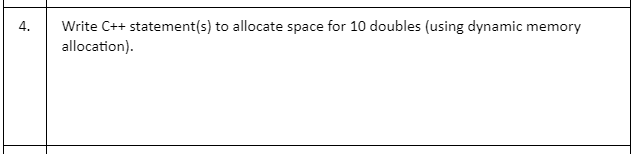
**public:**

**void display() override {**

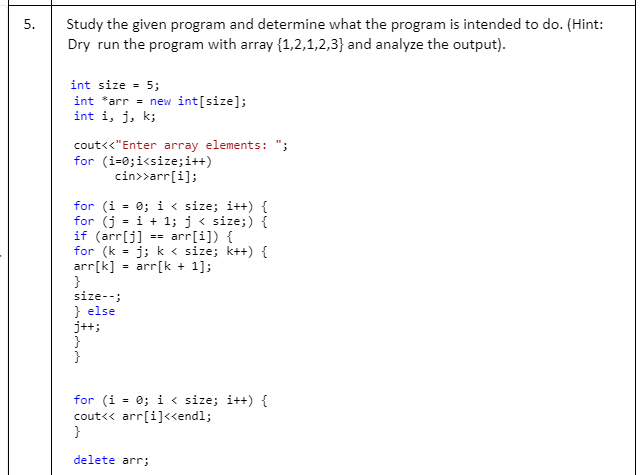
**cout << "Derived class display()" << endl;**

**}**

**};**

**Allocating space for 10 doubles using dynamic memory allocation in C++ using the new keyword. Here's how you can do it**

**double \*doubleArray = new double[10];**



**The program is designed to remove duplicate elements from an integer array and print the unique elements. Here's a step-by-step breakdown:**

1. **Initialize an array of integers arr with a size of 5.**
2. **Read integer values from the user to fill the array.**
3. **Loop through the array to remove duplicate elements:**
   * **For each element at index i, compare it with elements at subsequent indices (j).**
   * **If a duplicate is found, shift the elements to the left to fill the gap and reduce the array size.**
   * **Repeat this process until all duplicates are removed.**

**Print the unique elements from the modified array. Deallocate the dynamically allocated memory.**

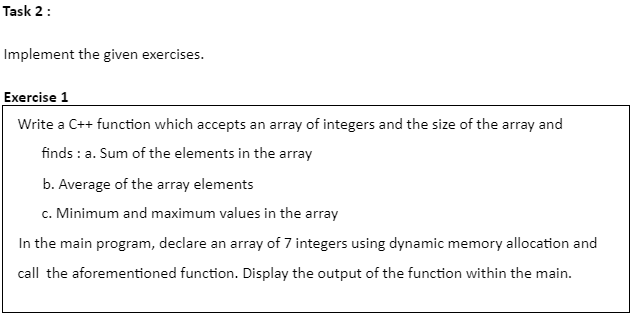
**For the input {1, 2, 1, 2, 3}, the program will output:**

**1**

**2**

**3**

**This output contains the unique elements from the input array, with duplicates removed.**



**#include <iostream>**

**#include <limits> // For INT\_MAX and INT\_MIN**

**using namespace std;**

**int main() {**

**const int size = 7;**

**int \*arr = new int[size]; // Create an array of 7 integers**

**// Read values into the array**

**cout << "Enter 7 integers:" << endl;**

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

**cin >> arr[i];**

**}**

**// Initialize variables**

**int sum = 0;**

**double average;**

**int minValue = INT\_MAX; // Initialize to a large value**

**int maxValue = INT\_MIN; // Initialize to a small value**

**// Calculate sum, average, min, and max**

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

**sum += arr[i];**

**if (arr[i] < minValue) {**

**minValue = arr[i];**

**}**

**if (arr[i] > maxValue) {**

**maxValue = arr[i];**

**}**

**}**

**average = static\_cast<double>(sum) / size; // Calculate average**

**// Display the results**

**cout << "Sum: " << sum << endl;**

**cout << "Average: " << average << endl;**

**cout << "Minimum: " << minValue << endl;**

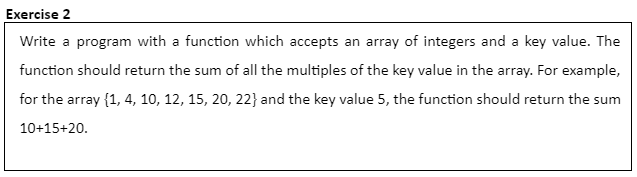
**cout << "Maximum: " << maxValue << endl;**

**// Deallocate memory**

**delete[] arr;**

**return 0;**

**}**

 **#include <iostream>**

**using namespace std;**

**// Function to calculate the sum of multiples of the key in the array**

**int sumOfMultiples(int arr[], int size, int key) {**

**int sum = 0;**

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

**if (arr[i] % key == 0) {**

**sum += arr[i];**

**}**

**}**

**return sum;**

**}**

**int main() {**

**const int size = 7;**

**int arr[size] = {1, 4, 10, 12, 15, 20, 22};**

**int key = 5;**

**// Calculate the sum of multiples of the key in the array**

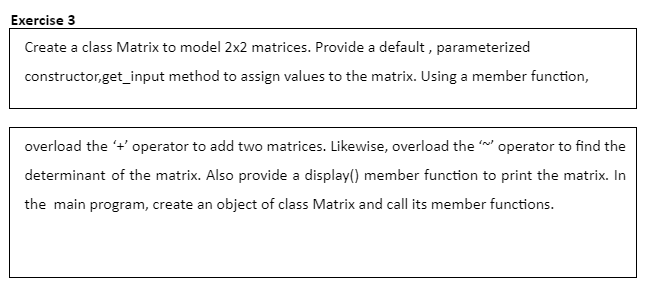
**int result = sumOfMultiples(arr, size, key);**

**// Display the result**

**cout << "Sum of multiples of " << key << ": " << result << endl;**

**return 0;**

**}**



**#include <iostream>**

**using namespace std;**

**class Matrix {**

**private:**

**int mat[2][2];**

**public:**

**// Default constructor**

**Matrix() {**

**// Initialize matrix with zeros**

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

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

**mat[i][j] = 0;**

**}**

**}**

**}**

**// Parameterized constructor to set matrix values**

**Matrix(int a, int b, int c, int d) {**

**mat[0][0] = a;**

**mat[0][1] = b;**

**mat[1][0] = c;**

**mat[1][1] = d;**

**}**

**// Member function to input matrix values**

**void get\_input() {**

**cout << "Enter matrix elements (2x2):" << endl;**

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

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

**cin >> mat[i][j];**

**}**

**}**

**}**

**// Overload the '+' operator to add two matrices**

**Matrix operator+(const Matrix& other) {**

**Matrix result;**

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

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

**result.mat[i][j] = mat[i][j] + other.mat[i][j];**

**}**

**}**

**return result;**

**}**

**// Overload the '~' operator to find the determinant**

**int operator~() {**

**return (mat[0][0] \* mat[1][1] - mat[0][1] \* mat[1][0]);**

**}**

**// Member function to display the matrix**

**void display() {**

**cout << "Matrix:" << endl;**

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

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

**cout << mat[i][j] << " ";**

**}**

**cout << endl;**

**}**

**}**

**};**

**int main() {**

**Matrix matrix1, matrix2, result;**

**cout << "Enter values for the first matrix:" << endl;**

**matrix1.get\_input();**

**cout << "Enter values for the second matrix:" << endl;**

**matrix2.get\_input();**

**result = matrix1 + matrix2;**

**cout << "Matrix Addition:" << endl;**

**result.display();**

**int determinant = ~matrix1;**

**cout << "Determinant of the first matrix: " << determinant << endl;**

**return 0;**

**}**