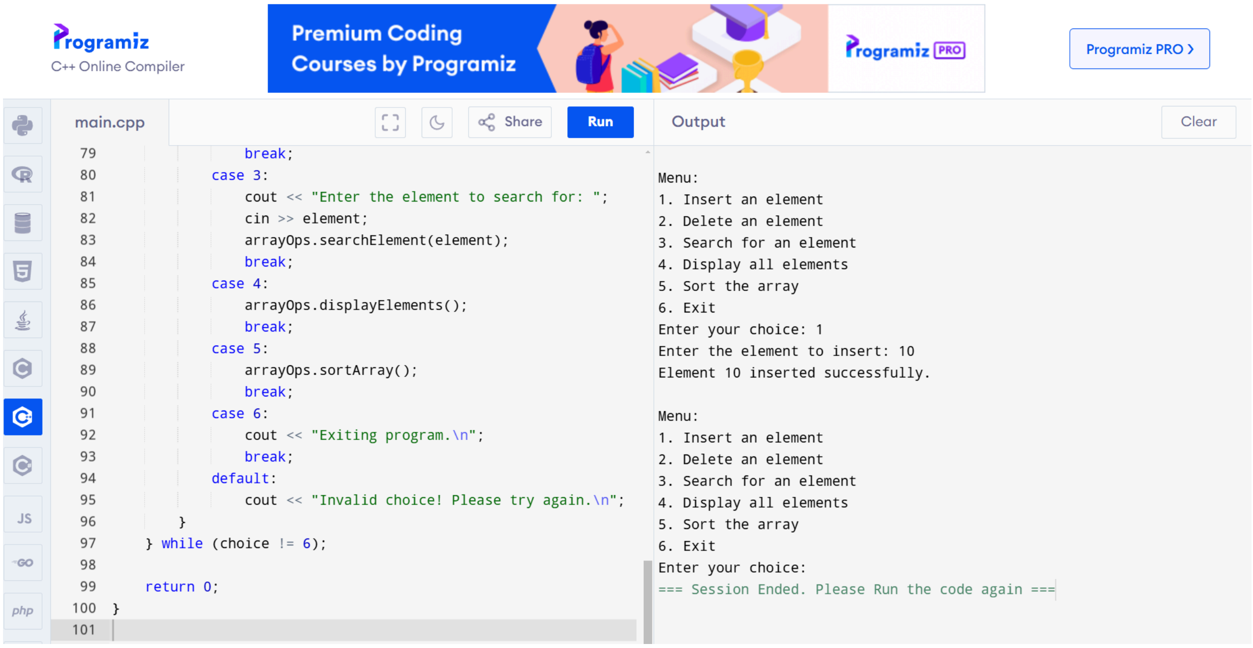
WEEK 2 TUTORIAL ASSIGNMENT

1. Write a menu driven C++ program with following option
a. Insert an element – Add a new integer to the array (at the next available position).
b. Delete an element – Remove a specific integer from the array (if it exists).
c. Search for an element – Check if a given integer is present in the array.
d. Display all elements – Print all stored integers in the array.
e. Sort the array – Sort the array in ascending order.
f. Exit – Terminate the program.

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

using namespace std;

class ArrayOperations {
private:
    vector<int> arr;

public:
    void insertElement(int element) {
        arr.push_back(element);
        cout << "Element " << element << " inserted successfully.\n";
    }

    void deleteElement(int element) {
        auto it = find(arr.begin(), arr.end(), element);
        if (it != arr.end()) {
            arr.erase(it);
            cout << "Element " << element << " deleted successfully.\n";
        } else {
            cout << "Element " << element << " not found in the array.\n";
        }
    }

    void searchElement(int element) {
        auto it = find(arr.begin(), arr.end(), element);
        if (it != arr.end()) {
            cout << "Element " << element << " found in the array.\n";
        } else {
            cout << "Element " << element << " not found in the array.\n";
        }
    }

    void displayElements() {
        if (arr.empty()) {
            cout << "The array is empty.\n";
            return;
        }
        cout << "Elements in the array: ";
        for (int num : arr) {
            cout << num << " ";
        }
        cout << endl;
    }

    void sortArray() {
        sort(arr.begin(), arr.end());
        cout << "Array sorted in ascending order.\n";
    }
};

int main() {
    ArrayOperations arrayOps;
    int choice, element;

    do {
        cout << "\nMenu:\n";
        cout << "1. Insert an element\n";
        cout << "2. Delete an element\n";
        cout << "3. Search for an element\n";
        cout << "4. Display all elements\n";
        cout << "5. Sort the array\n";
        cout << "6. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter the element to insert: ";
                cin >> element;
                arrayOps.insertElement(element);
                break;
            case 2:
                cout << "Enter the element to delete: ";
                cin >> element;
                arrayOps.deleteElement(element);
                break;
            case 3:
                cout << "Enter the element to search for: ";
                cin >> element;
                arrayOps.searchElement(element);
                break;
            case 4:
                arrayOps.displayElements();
                break;
            case 5:
                arrayOps.sortArray();
                break;
            case 6:
                cout << "Exiting program.\n";
                break;
            default:
                cout << "Invalid choice! Please try again.\n";
        }
    } while (choice != 6);

    return 0;
}
```

Output:



```
                break;
            case 3:
                cout << "Enter the element to search for: ";
                cin >> element;
                arrayOps.searchElement(element);
                break;
            case 4:
                arrayOps.displayElements();
                break;
            case 5:
                arrayOps.sortArray();
                break;
            case 6:
                cout << "Exiting program.\n";
                break;
            default:
                cout << "Invalid choice! Please try again.\n";
        }
    } while (choice != 6);

    return 0;
}
```

```
Menu:
1. Insert an element
2. Delete an element
3. Search for an element
4. Display all elements
5. Sort the array
6. Exit
Enter your choice: 1
Enter the element to insert: 10
Element 10 inserted successfully.

Menu:
1. Insert an element
2. Delete an element
3. Search for an element
4. Display all elements
5. Sort the array
6. Exit
Enter your choice:
=== Session Ended. Please Run the code again ===
```

2. Develop a system to manage students' marks in a class. The C++ program should:

• Use an array to store marks of N students.

• Provide a menu-driven system with options to:

a. Enter marks of N students.

b. Calculate the average marks of the class

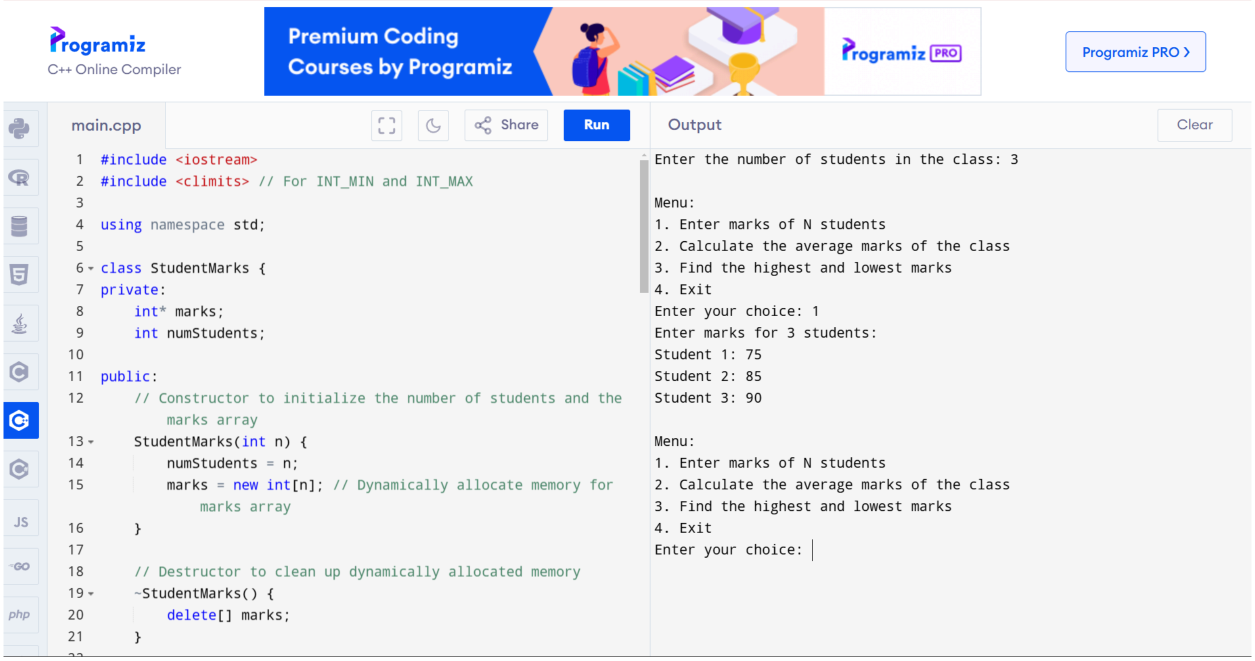
. c. Find the highest and lowest marks.

d. Exit.

```
#include <iostream>
#include <climits> // For INT_MIN and INT_MAX

using namespace std;
```

```
class StudentMarks {
private:
    int* marks;
    int numStudents;

public:
    // Constructor to initialize the number of students and the marks array
    StudentMarks(int n) {
        numStudents = n;
        marks = new int[n]; // Dynamically allocate memory for marks array
    }

    // Destructor to clean up dynamically allocated memory
    ~StudentMarks() {
        delete[] marks;
    }

    // Function to enter marks of N students
    void enterMarks() {
        cout << "Enter marks for " << numStudents << " students:\n";
        for (int i = 0; i < numStudents; i++) {
            cout << "Student " << i + 1 << ": ";
            cin >> marks[i];
        }
    }

    // Function to calculate and return the average marks
    double calculateAverage() {
        int sum = 0;
        for (int i = 0; i < numStudents; i++) {
            sum += marks[i];
        }
        return (double)sum / numStudents;
    }

    // Function to find and display the highest and lowest marks
    void findHighestLowest() {
        int highest = INT_MIN, lowest = INT_MAX;
        for (int i = 0; i < numStudents; i++) {
            if (marks[i] > highest) highest = marks[i];
            if (marks[i] < lowest) lowest = marks[i];
        }
        cout << "Highest Marks: " << highest << endl;
        cout << "Lowest Marks: " << lowest << endl;
    }
};

int main() {
    int choice, numStudents;

    // Take the number of students as input
    cout << "Enter the number of students in the class: ";
    cin >> numStudents;

    // Create an object of StudentMarks
    StudentMarks sm(numStudents);

    do {
        // Display the menu
        cout << "\nMenu:\n";
        cout << "1. Enter marks of N students\n";
        cout << "2. Calculate the average marks of the class\n";
        cout << "3. Find the highest and lowest marks\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                sm.enterMarks();
                break;
            case 2:
                {
                    double average = sm.calculateAverage();
                    cout << "The average marks of the class: " << average << endl;
                }
                break;
            case 3:
                sm.findHighestLowest();
                break;
            case 4:
                cout << "Exiting the program.\n";
                break;
            default:
                cout << "Invalid choice! Please try again.\n";
        }
    } while (choice != 4);

    return 0;
}
```

Output:



```
#include <iostream>
#include <climits> // For INT_MIN and INT_MAX

using namespace std;

class StudentMarks {
private:
    int* marks;
    int numStudents;

public:
    // Constructor to initialize the number of students and the marks array
    StudentMarks(int n) {
        numStudents = n;
        marks = new int[n]; // Dynamically allocate memory for marks array
    }

    // Destructor to clean up dynamically allocated memory
    ~StudentMarks() {
        delete[] marks;
    }
```

```
Enter the number of students in the class: 3

Menu:
1. Enter marks of N students
2. Calculate the average marks of the class
3. Find the highest and lowest marks
4. Exit
Enter your choice: 1
Enter marks for 3 students:
Student 1: 75
Student 2: 85
Student 3: 90

Menu:
1. Enter marks of N students
2. Calculate the average marks of the class
3. Find the highest and lowest marks
4. Exit
Enter your choice: 
```

3. Write a C++ program that reverses an integer array using pointers. The program should:

• Accept N integers from the user and store them in an array.

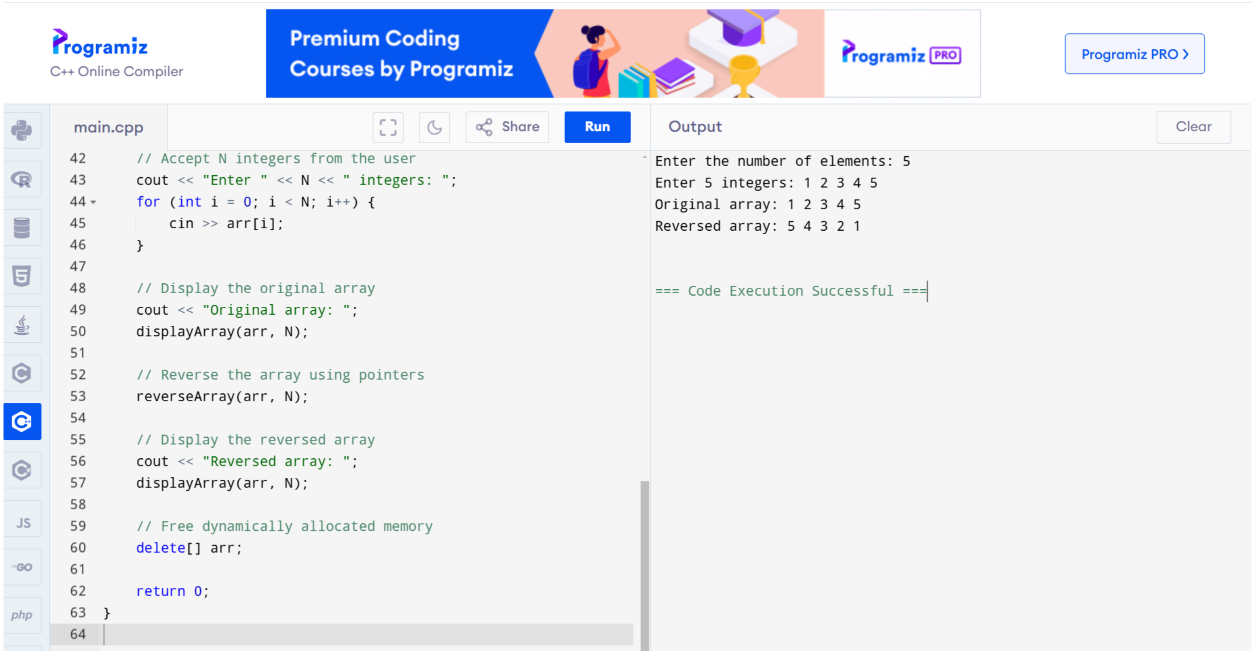
• Use a pointer approach to swap elements in place (without using another array).

• Display the original and reversed arrays.

```
#include <iostream>
```

```
using namespace std;

void reverseArray(int* arr, int size) {
   // Pointer to the start of the array
   int* start = arr;
   // Pointer to the end of the array
   int* end = arr + size - 1;

   // Swapping elements in place using pointers
   while (start < end) {
      // Swap the values pointed to by start and end
      int temp = *start;
      *start = *end;
      *end = temp;

      // Move the pointers towards each other
      start++;
      end--;
   }
}

void displayArray(int* arr, int size) {
   // Display all elements of the array
   for (int i = 0; i < size; i++) {
      cout << arr[i] << " ";
   }
   cout << endl;
}

int main() {
   int N;

   // Accept the size of the array
   cout << "Enter the number of elements: ";
   cin >> N;

   // Dynamically allocate memory for the array
   int* arr = new int[N];
```

```
    // Accept N integers from the user
    cout << "Enter " << N << " integers: ";
    for (int i = 0; i < N; i++) {
        cin >> arr[i];
    }

    // Display the original array
    cout << "Original array: ";
    displayArray(arr, N);

    // Reverse the array using pointers
    reverseArray(arr, N);

    // Display the reversed array
    cout << "Reversed array: ";
    displayArray(arr, N);

    // Free dynamically allocated memory
    delete[] arr;

    return 0;
}
```



```
    // Accept N integers from the user
    cout << "Enter " << N << " integers: ";
    for (int i = 0; i < N; i++) {
        cin >> arr[i];
    }

    // Display the original array
    cout << "Original array: ";
    displayArray(arr, N);

    // Reverse the array using pointers
    reverseArray(arr, N);

    // Display the reversed array
    cout << "Reversed array: ";
    displayArray(arr, N);

    // Free dynamically allocated memory
    delete[] arr;

    return 0;
}
```

```
Enter the number of elements: 5
Enter 5 integers: 1 2 3 4 5
Original array: 1 2 3 4 5
Reversed array: 5 4 3 2 1


=== Code Execution Successful ===
```

4. What is the data type of ‘result’ in the below code? Justify your answer based on C++'s type
conversion rules.
float x = 2.5;
int y = 3;
auto result = x / y;

In the give code:

```
float x = 2.5;
int y = 3;
auto result = x / y;
```

Data type of `result`:

The data type of `result` will be **`float`**.

Justification based on C++'s type conversion rules:

1. **Type of `x`**: `x` is of type `float` (as `2.5` is a floating-point literal).
2. **Type of `y`**: `y` is of type `int`.

When you perform the operation `x / y`, the **C++ type promotion rules** come into play:

- **Implicit Type Conversion**: In an arithmetic operation where an `int` and a `float` are involved, C++ automatically promotes the `int` to a `float` to perform the operation in a consistent floating-point context.
- Therefore, the operation `x / y` (which is `float / int`) results in a `float` because the `int` (`y`) is promoted to `float` before performing the division.

3. **Result Type**: Since `x` is a `float` and `y` is implicitly converted to a `float`, the result of `x / y` is a `float`. This result is assigned to the result.
4. **Use of `auto`**: The `auto` keyword allows the compiler to deduce the type of `result` based on the expression on the right-hand side. Since the result of `x / y` is a `float`, `the result` will also be deduced as a `float`.

Final Answer:

The data type of `result` is **`float`** because of the implicit promotion of the `int` (`y`) to `float` during the division operation, and `auto` deduces the type based on the result of the expression.

5. Consider this code snippet: double pi = 3.14159; int approx_pi = (int)pi + 0.5; std::cout << approx_pi; What is the expected output? How would you modify the code to ensure correct rounding to the nearest integer?

given code:

double pi = 3.14159;

int approx_pi = (int)pi + 0.5;

std::cout << approx_pi;

Expected Output:

1. First Line: `pi` is a `double` with the value `3.14159`.
2. Second Line: The code attempts to approximate `pi` as an `int` using the following expression:
   - `(int)pi` performs a type cast of `pi` to an `int`, which truncates the decimal part of `pi`. Thus, `3.14159` becomes `3`.
   - The value `0.5` is then added to this truncated result: `3 + 0.5 = 3.5`.
   - The result `3.5` is assigned to `approx_pi`, which is an `int`. Since `approx_pi` is an `int`, it truncates `3.5` to `3` (losing the decimal part).

Thus, the final value of `approx_pi` will be `3`, and this value is printed to the console.

Expected Output:

Copy

```
3
```

How to Ensure Correct Rounding:

To ensure correct rounding to the nearest integer, you can modify the code as follows:

cpp

Copy

```
double pi = 3.14159;

int approx_pi = static_cast<int>(pi + 0.5);  // Round to nearest integer

std::cout << approx_pi;
```

Explanation of the Modification:

- **Rounding before casting**: Instead of truncating the decimal part first and then adding `0.5`, we add `0.5` to `pi` first and then cast it to an `int`. This ensures that values with decimals above `.5` are rounded up, and values below `.5` are rounded down. For example:
    - For `pi = 3.14159`, the expression `pi + 0.5` becomes `3.64159`, and casting this to `int` results in `3` (correct rounding).
    - For `pi = 3.6`, the expression `3.6 + 0.5 = 4.1`, which gets truncated to `4`.
- **`static_cast<int>`**: While this is not strictly necessary, using `static_cast<int>` is preferred over a C-style cast `(int)` because it is more explicit and safer in C++.

Corrected Output with Proper Rounding:

For `pi = 3.14159`, after the modification, the output will be `3`. If `pi` were `3.6`, the output would be `4`.