



SAPIENZA
UNIVERSITÀ DI ROMA

Algorithms

Exercises 2 — Programming languages

Worked exercises on chapters 6–8: programs in C++, Python and Java, from pseudo-code to code. Every exercise has a complete solution.

Fabio Furini

fabiofurini.github.io

Department of Computer, Control and Management Engineering — Sapienza University of Rome

Online version and downloadable material:

fabiofurini.github.io/algorithms/
github.com/fabiofurini/algorithms

Academic year 2026–2027

Contents

6	Introduction to C++	2
7	Introduction to Python	26
8	Introduction to Java	39



Introduction to C++

In the exercises of this chapter the programs are written, compiled and executed with the conventions of the chapter «Introduction to C++», summarized below.

Warning

- The program of an exercise is saved in a file `<name>.cpp`, compiled and executed with the commands

```
g++ -Wall -std=c++17 -o <name> <name>.cpp
./<name>
```

- The test instances are fixed-size arrays declared in `main`, for example `int arr1[] = {9, 12, 3, 16, 19, 11, 12};`, and are passed to the functions through the pointer to their first element (`int *a`) together with their size `n`.
- The elements of an array of size n are `a[0]`, `a[1]`, $\dots$, `a[n-1]`: the entry `a[i]` of an instance written as `a = [...]` in the text corresponds to `a[i-1]` in the code.
- The results that a function has to give back besides its return value are passed by reference (`int& x`, `double& x`).
- When an array is printed, every value is followed by a space, so the printed lines end with a space.

Exercise 6.1 — Introduction to C++ (E1)

Given an array a of $n \geq 1$ integer numbers, we consider the problem of finding the maximum value in the array.

Questions

1. Write a complete C++ program that finds the maximum value in an array a of n integer numbers. The program should:

- Define a function `int ArrayMaxValue(int n, int *a)` that implements the algorithm
- In the `main()` function, create test arrays and call the function
- Print the results for each test case

2. Execute the program on the following instances:

- Instance 1: $a = [9, 12, 3, 16, 19, 11, 12]$ with $n = 7$
- Instance 2: $a = [5, 2, 8, 1]$ with $n = 4$
- Instance 3: $a = [42]$ with $n = 1$

Show the complete compilation commands and output for all test cases.

3. Modify the program to create a new function `int ArrayMaxValueWithIndex(int n, int *a, int& idx)` that returns both the maximum value and stores its index in the reference parameter `idx`. Execute this modified version and show the output.

Solution

1. Complete C++ program:

```

1  #include <iostream>
2  using namespace std;
3
4  // Function to find maximum value in array
5  int ArrayMaxValue(int n, int *a)
6  {
7      int max = a[0]; // Initialize with first element
8
9      for (int i = 1; i < n; i++)
10     {
11         if (a[i] > max)
12         {
13             max = a[i]; // Update maximum
14         }
15     }
16
17     return max;
18 }
19
20 int main()
21 {
22     // Test case 1
23     int arr1[] = {9, 12, 3, 16, 19, 11, 12};
24     int n1 = 7;
25
26     cout << "Test 1 - Array: ";
27     for (int i = 0; i < n1; i++)
28     {
29         cout << arr1[i] << " ";

```

```

30     }
31     cout << endl;
32
33     int max1 = ArrayMaxValue(n1, arr1);
34     cout << "Maximum value: " << max1 << endl;
35     cout << endl;
36
37     // Test case 2
38     int arr2[] = {5, 2, 8, 1};
39     int n2 = 4;
40
41     cout << "Test 2 - Array: ";
42     for (int i = 0; i < n2; i++)
43     {
44         cout << arr2[i] << " ";
45     }
46     cout << endl;
47
48     int max2 = ArrayMaxValue(n2, arr2);
49     cout << "Maximum value: " << max2 << endl;
50     cout << endl;
51
52     // Test case 3
53     int arr3[] = {42};
54     int n3 = 1;
55
56     cout << "Test 3 - Array: ";
57     for (int i = 0; i < n3; i++)
58     {
59         cout << arr3[i] << " ";
60     }
61     cout << endl;
62
63     int max3 = ArrayMaxValue(n3, arr3);
64     cout << "Maximum value: " << max3 << endl;
65
66     return 0;
67 }

```

2. Compilation and execution:

To compile the program:

```
g++ -Wall -std=c++17 -o max_value max_value.cpp
```

To run the program:

```
./max_value
```

Output:

Test 1 - Array: 9 12 3 16 19 11 12
Maximum value: 19

Test 2 - Array: 5 2 8 1
Maximum value: 8

Test 3 - Array: 42
Maximum value: 42

3. Modified program with index:

```

1  #include <iostream>
2  using namespace std;
3
4  // Function to find maximum value and its index
5  int ArrayMaxValueWithIndex(int n, int *a, int& idx)
6  {
7      int max = a[0];
8      idx = 0;
9
10     for (int i = 1; i < n; i++)
11     {
12         if (a[i] > max)
13         {
14             max = a[i];
15             idx = i; // Store the index
16         }
17     }
18
19     return max;
20 }
21
22 int main()
23 {
24     int arr[] = {9, 12, 3, 16, 19, 11, 12};
25     int n = 7;
26     int index;
27
28     cout << "Array: ";
29     for (int i = 0; i < n; i++)
30     {
31         cout << arr[i] << " ";
32     }
33     cout << endl;
34
35     int max = ArrayMaxValueWithIndex(n, arr, index);
36
37     cout << "Maximum value: " << max << endl;
38     cout << "Position (index): " << index << endl;
39
40     return 0;
41 }

```

Output:

Array: 9 12 3 16 19 11 12

Maximum value: 19

Position (index): 4

Note: The maximum value 19 is found at index 4 (the 5th element, since arrays are indexed from 0 in C++). If the maximum value appears more than once, the function stores the index of its first occurrence, because `max` and `idx` are updated only when `a[i] > max` (strict inequality).

Exercise 6.2 — Introduction to C++ (E2)

Given an array a of $n \geq 1$ integer numbers, we consider the problem of finding its minimum value.

Questions

1. Write a complete C++ program that finds the minimum value in an array a of n integer numbers. The program should:
 - Define a function `int ArrayMinValue(int n, int *a)` that implements the algorithm
 - In the `main()` function, create test arrays and call the function
 - Print the results for each test case
2. Execute the program on the following instances:
 - Instance 1: $a = [9, 12, 3, 16, 19, 11, 12]$ with $n = 7$
 - Instance 2: $a = [15, 7, 22, 4]$ with $n = 4$
 - Instance 3: $a = [100]$ with $n = 1$

Show the complete compilation commands and output for all test cases.

3. Modify the program to create a new function `void ArrayMaxMin(int n, int *a, int& max, int& min)` that finds both the maximum and minimum values in a single pass through the array. Execute this modified version and show the output.

Solution

1. **Complete C++ program:**

```

1  #include <iostream>
2  using namespace std;
3
4  // Function to find minimum value in array
5  int ArrayMinValue(int n, int *a)
6  {
7      int m = a[0]; // Initialize with first element
8
9      for (int i = 1; i < n; i++)
10     {
11         if (a[i] < m)
12         {
13             m = a[i]; // Update minimum
14         }
15     }
16
17     return m;
18 }
19
20 int main()
21 {
22     // Test case 1
23     int arr1[] = {9, 12, 3, 16, 19, 11, 12};
24     int n1 = 7;
25
26     cout << "Test 1 - Array: ";
27     for (int i = 0; i < n1; i++)
28     {
29         cout << arr1[i] << " ";
30     }
31     cout << endl;
32
33     int min1 = ArrayMinValue(n1, arr1);
34     cout << "Minimum value: " << min1 << endl;
35     cout << endl;
36
37     // Test case 2
38     int arr2[] = {15, 7, 22, 4};
39     int n2 = 4;
40
41     cout << "Test 2 - Array: ";
42     for (int i = 0; i < n2; i++)
43     {
44         cout << arr2[i] << " ";
45     }
46     cout << endl;
47
48     int min2 = ArrayMinValue(n2, arr2);
49     cout << "Minimum value: " << min2 << endl;
50     cout << endl;
51
52     // Test case 3
53     int arr3[] = {100};
54     int n3 = 1;
55
56     cout << "Test 3 - Array: ";

```

```

57     for (int i = 0; i < n3; i++)
58     {
59         cout << arr3[i] << " ";
60     }
61     cout << endl;
62
63     int min3 = ArrayMinValue(n3, arr3);
64     cout << "Minimum value: " << min3 << endl;
65
66     return 0;
67 }

```

2. Compilation and execution:

To compile the program:

```
g++ -Wall -std=c++17 -o min_value min_value.cpp
```

To run the program:

```
./min_value
```

Output:

```
Test 1 - Array: 9 12 3 16 19 11 12
Minimum value: 3
```

```
Test 2 - Array: 15 7 22 4
Minimum value: 4
```

```
Test 3 - Array: 100
Minimum value: 100
```

3. Modified program to find both max and min:

```

1  #include <iostream>
2  using namespace std;
3
4  // Function to find both maximum and minimum values
5  void ArrayMaxMin(int n, int *a, int& max, int& min)
6  {
7      max = a[0];
8      min = a[0];
9
10     for (int i = 1; i < n; i++)
11     {
12         if (a[i] > max)
13         {
14             max = a[i];

```

```

15     }
16     if (a[i] < min)
17     {
18         min = a[i];
19     }
20 }
21 }
22
23 int main()
24 {
25     int arr[] = {9, 12, 3, 16, 19, 11, 12};
26     int n = 7;
27     int maximum, minimum;
28
29     cout << "Array: ";
30     for (int i = 0; i < n; i++)
31     {
32         cout << arr[i] << " ";
33     }
34     cout << endl;
35
36     ArrayMaxMin(n, arr, maximum, minimum);
37
38     cout << "Maximum value: " << maximum << endl;
39     cout << "Minimum value: " << minimum << endl;
40
41     return 0;
42 }

```

Output:

```

Array: 9 12 3 16 19 11 12
Maximum value: 19
Minimum value: 3

```

Note: This implementation finds both the maximum and the minimum value in a single pass through the array. The number of comparisons is the same as with two separate functions, $2(n - 1)$, but the array is traversed once instead of twice.

Exercise 6.3 — Introduction to C++ (E3)

Given an array a of $n \geq 1$ integer numbers, we consider the problem of computing the sum and average of all its elements.

Questions

1. Write a complete C++ program that computes the sum and average of elements in an array a of n integer numbers. The program should:

- Define a function `int ArraySum(int n, int *a)` that computes the sum of array elements
- In the `main()` function, create test arrays and call the function
- Compute and print both the sum and average for each test case

2. Execute the program on the following instances:

- Instance 1: $a = [5, 12, 3, 8]$ with $n = 4$
- Instance 2: $a = [10, 20, 30]$ with $n = 3$
- Instance 3: $a = [7]$ with $n = 1$

Show the complete compilation commands and output for all test cases.

3. Modify the program to create a new function `void ArraySumAndAverage(int n, int *a, int& sum, double& avg)` that computes both the sum and average in one function. Execute this modified version and show the output.

Solution

1. Complete C++ program:

```

1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4
5  // Function to compute sum of array elements
6  int ArraySum(int n, int *a)
7  {
8      int S = 0;
9
10     for (int i = 0; i < n; i++)
11     {
12         S = S + a[i];
13     }
14
15     return S;
16 }
17
18 int main()
19 {
20     // Test case 1
21     int arr1[] = {5, 12, 3, 8};
22     int n1 = 4;
23
24     cout << "Test 1 - Array: ";
25     for (int i = 0; i < n1; i++)
26     {
27         cout << arr1[i] << " ";
28     }
29     cout << endl;

```

```

30
31     int sum1 = ArraySum(n1, arr1);
32     double avg1 = (double)sum1 / n1;
33     cout << "Sum: " << sum1 << endl;
34     cout << "Average: " << fixed << setprecision(2)
35         << avg1 << endl;
36     cout << endl;
37
38     // Test case 2
39     int arr2[] = {10, 20, 30};
40     int n2 = 3;
41
42     cout << "Test 2 - Array: ";
43     for (int i = 0; i < n2; i++)
44     {
45         cout << arr2[i] << " ";
46     }
47     cout << endl;
48
49     int sum2 = ArraySum(n2, arr2);
50     double avg2 = (double)sum2 / n2;
51     cout << "Sum: " << sum2 << endl;
52     cout << "Average: " << fixed << setprecision(2)
53         << avg2 << endl;
54     cout << endl;
55
56     // Test case 3
57     int arr3[] = {7};
58     int n3 = 1;
59
60     cout << "Test 3 - Array: ";
61     for (int i = 0; i < n3; i++)
62     {
63         cout << arr3[i] << " ";
64     }
65     cout << endl;
66
67     int sum3 = ArraySum(n3, arr3);
68     double avg3 = (double)sum3 / n3;
69     cout << "Sum: " << sum3 << endl;
70     cout << "Average: " << fixed << setprecision(2)
71         << avg3 << endl;
72
73     return 0;
74 }

```

2. Compilation and execution:

To compile the program:

```
g++ -Wall -std=c++17 -o array_sum array_sum.cpp
```

To run the program:

```
./array_sum
```

Output:

Test 1 - Array: 5 12 3 8

Sum: 28

Average: 7.00

Test 2 - Array: 10 20 30

Sum: 60

Average: 20.00

Test 3 - Array: 7

Sum: 7

Average: 7.00

3. Modified program with combined function:

```

1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4
5  // Function to compute both sum and average
6  void ArraySumAndAverage(int n, int *a, int& sum, double& avg)
7  {
8      sum = 0;
9
10     for (int i = 0; i < n; i++)
11     {
12         sum = sum + a[i];
13     }
14
15     avg = (double)sum / n;
16 }
17
18 int main()
19 {
20     int arr[] = {5, 12, 3, 8};
21     int n = 4;
22     int sum;
23     double average;
24
25     cout << "Array: ";
26     for (int i = 0; i < n; i++)
27     {
28         cout << arr[i] << " ";
29     }
30     cout << endl;
31
32     ArraySumAndAverage(n, arr, sum, average);
33

```

```

34     cout << "Sum: " << sum << endl;
35     cout << "Average: " << fixed << setprecision(2)
36         << average << endl;
37
38     return 0;
39 }

```

Output:

Array: 5 12 3 8

Sum: 28

Average: 7.00

Note: The `setprecision(2)` manipulator is used to format the average with 2 decimal places. The conversion (`double`) ensures the floating-point division: without it, `sum / n` would be the integer division. Since the average divides by n , the functions require $n \geq 1$.

Exercise 6.4 — Introduction to C++ (E4)

Given an array a of n integer numbers and a value m , we consider the problem of determining whether the value m is present or not in the array.

Questions

1. Write a complete C++ program that implements the linear search algorithm for an array a of n integer numbers. The program should:
 - Define a function `int LinearSearch(int n, int *a, int m)` that searches for value m in array a
 - Return the index of the element if found, or `-1` if not found
 - In the `main()` function, create a test array and search for different values
 - Print the results for each search
2. Execute the program with the following instance:
 - Array: $a = [9, 12, 3, 16, 19, 11, 12]$ with $n = 7$
 - Search for: $m = 16$ (element present)
 - Search for: $m = 25$ (element not present)
 - Search for: $m = 9$ (first element)
 - Search for: $m = 12$ (duplicate element)

Show the complete compilation commands and output for all test cases.

3. Modify the program to create a function `int CountOccurrences(int n, int *a, int m)` that counts how many times value `m` appears in the array. Execute this modified version with value `m = 12` in the test array and show the output.

Solution

1. Complete C++ program:

```

1  #include <iostream>
2  using namespace std;
3
4  // Function to perform linear search
5  int LinearSearch(int n, int *a, int m)
6  {
7      for (int i = 0; i < n; i++)
8      {
9          if (a[i] == m)
10         {
11             return i; // Found, return index
12         }
13     }
14
15     return -1; // Not found
16 }
17
18 int main()
19 {
20     int arr[] = {9, 12, 3, 16, 19, 11, 12};
21     int n = 7;
22
23     cout << "Array: ";
24     for (int i = 0; i < n; i++)
25     {
26         cout << arr[i] << " ";
27     }
28     cout << endl << endl;
29
30     // Test 1: Search for 16 (present)
31     int m1 = 16;
32     int result1 = LinearSearch(n, arr, m1);
33     cout << "Search for " << m1 << ": ";
34     if (result1 != -1)
35     {
36         cout << "Found at index " << result1 << endl;
37     }
38     else
39     {
40         cout << "Not found" << endl;
41     }
42
43     // Test 2: Search for 25 (not present)
44     int m2 = 25;

```

```

45     int result2 = LinearSearch(n, arr, m2);
46     cout << "Search for " << m2 << ": ";
47     if (result2 != -1)
48     {
49         cout << "Found at index " << result2 << endl;
50     }
51     else
52     {
53         cout << "Not found" << endl;
54     }
55
56     // Test 3: Search for 9 (first element)
57     int m3 = 9;
58     int result3 = LinearSearch(n, arr, m3);
59     cout << "Search for " << m3 << ": ";
60     if (result3 != -1)
61     {
62         cout << "Found at index " << result3 << endl;
63     }
64     else
65     {
66         cout << "Not found" << endl;
67     }
68
69     // Test 4: Search for 12 (duplicate)
70     int m4 = 12;
71     int result4 = LinearSearch(n, arr, m4);
72     cout << "Search for " << m4 << ": ";
73     if (result4 != -1)
74     {
75         cout << "Found at index " << result4
76             << " (first occurrence)" << endl;
77     }
78     else
79     {
80         cout << "Not found" << endl;
81     }
82
83     return 0;
84 }

```

2. Compilation and execution:

To compile the program:

```
g++ -Wall -std=c++17 -o linear_search linear_search.cpp
```

To run the program:

```
./linear_search
```

Output:

Array: 9 12 3 16 19 11 12

Search for 16: Found at index 3

Search for 25: Not found

Search for 9: Found at index 0

Search for 12: Found at index 1 (first occurrence)

3. Modified program to count occurrences:

```

1  #include <iostream>
2  using namespace std;
3
4  // Function to count occurrences of value m
5  int CountOccurrences(int n, int *a, int m)
6  {
7      int count = 0;
8
9      for (int i = 0; i < n; i++)
10     {
11         if (a[i] == m)
12         {
13             count++;
14         }
15     }
16
17     return count;
18 }
19
20 int main()
21 {
22     int arr[] = {9, 12, 3, 16, 19, 11, 12};
23     int n = 7;
24
25     cout << "Array: ";
26     for (int i = 0; i < n; i++)
27     {
28         cout << arr[i] << " ";
29     }
30     cout << endl;
31
32     int x = 12;
33     int count = CountOccurrences(n, arr, x);
34
35     cout << "Value " << x << " appears " << count
36         << " times in the array" << endl;
37
38     return 0;
39 }

```

Output:

Array: 9 12 3 16 19 11 12

Value 12 appears 2 times in the array

Note: The `LinearSearch` function returns the index of the first occurrence when duplicates exist. The `CountOccurrences` function scans the entire array to count all occurrences.

Exercise 6.5 — Introduction to C++ (E5)

Given an array a of $n \geq 1$ integer numbers, we consider the problem of computing in a single pass the maximum, minimum, sum and average of all elements in the array.

Questions

- Write a complete C++ program that computes all statistics for an array a of n integer numbers in a single pass. The program should:
 - Define a function `void ArrayStatistics(int n, int *a, int& max, int& min, int& sum, double& avg)` that computes all four statistics
 - In the `main()` function, create test arrays and call the function
 - Print all statistics for each test case
- Execute the program on the following instances:
 - Instance 1: $a = [9, 12, 3, 16, 19, 11, 12]$ with $n = 7$
 - Instance 2: $a = [5, 2, 8, 1]$ with $n = 4$

Show the complete compilation commands and output for all test cases.

- Discuss the advantages of computing all statistics in a single pass versus computing them separately with multiple functions.

Solution

1. Complete C++ program:

```

1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4
5 // Function to compute all statistics in a single pass
6 void ArrayStatistics(int n, int *a, int& max, int& min,
7                     int& sum, double& avg)
8 {
9     // Initialize with first element
10    max = a[0];
11    min = a[0];
12    sum = a[0];

```

```

13
14     // Process remaining elements
15     for (int i = 1; i < n; i++)
16     {
17         if (a[i] > max)
18         {
19             max = a[i];
20         }
21         if (a[i] < min)
22         {
23             min = a[i];
24         }
25         sum = sum + a[i];
26     }
27
28     // Compute average
29     avg = (double)sum / n;
30 }
31
32 int main()
33 {
34     // Test case 1
35     int arr1[] = {9, 12, 3, 16, 19, 11, 12};
36     int n1 = 7;
37     int max1, min1, sum1;
38     double avg1;
39
40     cout << "Test 1 - Array: ";
41     for (int i = 0; i < n1; i++)
42     {
43         cout << arr1[i] << " ";
44     }
45     cout << endl;
46
47     ArrayStatistics(n1, arr1, max1, min1, sum1, avg1);
48
49     cout << "Maximum: " << max1 << endl;
50     cout << "Minimum: " << min1 << endl;
51     cout << "Sum: " << sum1 << endl;
52     cout << "Average: " << fixed << setprecision(2)
53         << avg1 << endl;
54     cout << endl;
55
56     // Test case 2
57     int arr2[] = {5, 2, 8, 1};
58     int n2 = 4;
59     int max2, min2, sum2;
60     double avg2;
61
62     cout << "Test 2 - Array: ";
63     for (int i = 0; i < n2; i++)
64     {
65         cout << arr2[i] << " ";
66     }
67     cout << endl;
68

```

```
69     ArrayStatistics(n2, arr2, max2, min2, sum2, avg2);
70
71     cout << "Maximum: " << max2 << endl;
72     cout << "Minimum: " << min2 << endl;
73     cout << "Sum: " << sum2 << endl;
74     cout << "Average: " << fixed << setprecision(2)
75           << avg2 << endl;
76
77     return 0;
78 }
```

2. Compilation and execution:

To compile the program:

```
g++ -Wall -std=c++17 -o array_stats array_stats.cpp
```

To run the program:

```
./array_stats
```

Output:

```
Test 1 - Array: 9 12 3 16 19 11 12
Maximum: 19
Minimum: 3
Sum: 82
Average: 11.71
```

```
Test 2 - Array: 5 2 8 1
Maximum: 8
Minimum: 1
Sum: 16
Average: 4.00
```

3. Advantages of single-pass approach:

Advantages:

- **Fewer array traversals:** The array is traversed only once instead of once for each statistic
- **More compact code:** Single function and single loop instead of multiple function calls
- **Better for streaming data:** Can process data that arrives sequentially or doesn't fit entirely in memory

- **Reduced overhead:** Only one loop initialization and termination instead of multiple loops

Trade-offs:

- **Less modular:** Combines multiple operations in one function, which may be harder to maintain
- **Less flexible:** If only one statistic is needed, the function still computes all of them
- **Code reusability:** Cannot easily reuse individual statistic computations in other contexts

Recommendation: The single-pass approach is preferable when:

- All statistics are needed together
- Working with large arrays where minimizing array accesses is important
- Processing streaming data or data that cannot fit in memory

Separate functions are better when:

- Only specific statistics are needed
- Code clarity and modularity are higher priorities
- Individual functions may be reused in different contexts

Exercise 6.6 — Introduction to C++ (E6)

Given a two-dimensional array a of $m \times n$ integer numbers, we consider the problem of computing the sum of all its elements.

Questions

1. Write a complete C++ program that computes the sum of all elements of a two-dimensional array a of $m \times n$ integers. The program should:
 - Define a function `int SumMatrix(int m, int n, int **matrix)` that computes the sum
 - In the `main()` function, allocate test matrices using dynamic memory, initialize them, call the function, and print the results
 - Free all allocated memory after use
2. Execute the program on the following instances:

- Instance 1: a 3×4 matrix with sequential values $a[i][j] = (i - 1) \cdot n + j$ for $i \in \{1, 2, 3\}$ and $j \in \{1, 2, 3, 4\}$ (rows and columns numbered from 1, as in pseudo-code), i.e.,

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

- Instance 2: a 2×3 matrix with values $\begin{bmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \end{bmatrix}$

Show the complete compilation commands and output for both test cases.

3. Modify the program to add a function `void RowSums(int m, int n, int **matrix, int *rowSums)` that computes the sum of each row and stores the results in the array `rowSums`. Execute this modified version on Instance 1 and show the output.

Solution

1. Complete C++ program:

```

1  #include <iostream>
2  using namespace std;
3
4  // Function to compute the sum of all elements in a matrix
5  int SumMatrix(int m, int n, int **matrix)
6  {
7      int sum = 0;
8      for (int i = 0; i < m; i++)
9      {
10         for (int j = 0; j < n; j++)
11         {
12             sum += matrix[i][j];
13         }
14     }
15     return sum;
16 }
17
18 int main()
19 {
20     // Test case 1: 3x4 matrix with sequential values
21     int m1 = 3, n1 = 4;
22
23     int** mat1 = new int*[m1];
24     for (int i = 0; i < m1; i++)
25     {
26         mat1[i] = new int[n1];
27     }
28
29     int val = 1;
30     for (int i = 0; i < m1; i++)
31     {

```

```

32     for (int j = 0; j < n1; j++)
33     {
34         mat1[i][j] = val++;
35     }
36 }
37
38 cout << "Test 1 - Matrix " << m1 << "x" << n1 << ":" << endl;
39 for (int i = 0; i < m1; i++)
40 {
41     for (int j = 0; j < n1; j++)
42     {
43         cout << mat1[i][j] << " ";
44     }
45     cout << endl;
46 }
47 cout << "Sum: " << SumMatrix(m1, n1, mat1) << endl << endl;
48
49 for (int i = 0; i < m1; i++)
50 {
51     delete[] mat1[i];
52 }
53 delete[] mat1;
54
55 // Test case 2: 2x3 matrix
56 int m2 = 2, n2 = 3;
57 int data[2][3] = {{3, 1, 4}, {1, 5, 9}};
58
59 int** mat2 = new int*[m2];
60 for (int i = 0; i < m2; i++)
61 {
62     mat2[i] = new int[n2];
63     for (int j = 0; j < n2; j++)
64     {
65         mat2[i][j] = data[i][j];
66     }
67 }
68
69 cout << "Test 2 - Matrix " << m2 << "x" << n2 << ":" << endl;
70 for (int i = 0; i < m2; i++)
71 {
72     for (int j = 0; j < n2; j++)
73     {
74         cout << mat2[i][j] << " ";
75     }
76     cout << endl;
77 }
78 cout << "Sum: " << SumMatrix(m2, n2, mat2) << endl;
79
80 for (int i = 0; i < m2; i++)
81 {
82     delete[] mat2[i];
83 }
84 delete[] mat2;
85
86 return 0;
87 }

```

2. Compilation and execution:

To compile the program:

```
g++ -Wall -std=c++17 -o sum_matrix sum_matrix.cpp
```

To run the program:

```
./sum_matrix
```

Output:

Test 1 - Matrix 3x4:

```
1 2 3 4
5 6 7 8
9 10 11 12
Sum: 78
```

Test 2 - Matrix 2x3:

```
3 1 4
1 5 9
Sum: 23
```

3. Row sums function:

```
1 #include <iostream>
2 using namespace std;
3
4 // Function to compute the sum of all elements in a matrix
5 int SumMatrix(int m, int n, int **matrix)
6 {
7     int sum = 0;
8     for (int i = 0; i < m; i++)
9     {
10         for (int j = 0; j < n; j++)
11         {
12             sum += matrix[i][j];
13         }
14     }
15     return sum;
16 }
17
18 // Function to compute the sum of each row
19 void RowSums(int m, int n, int **matrix, int *rowSums)
20 {
21     for (int i = 0; i < m; i++)
22     {
```

```

23     rowSums[i] = 0;
24     for (int j = 0; j < n; j++)
25     {
26         rowSums[i] += matrix[i][j];
27     }
28 }
29 }
30
31 int main()
32 {
33     // Test case 1: 3x4 matrix with sequential values
34     int m1 = 3, n1 = 4;
35
36     int** mat1 = new int*[m1];
37     for (int i = 0; i < m1; i++)
38     {
39         mat1[i] = new int[n1];
40     }
41
42     int val = 1;
43     for (int i = 0; i < m1; i++)
44     {
45         for (int j = 0; j < n1; j++)
46         {
47             mat1[i][j] = val++;
48         }
49     }
50
51     cout << "Matrix " << m1 << "x" << n1 << ":" << endl;
52     for (int i = 0; i < m1; i++)
53     {
54         for (int j = 0; j < n1; j++)
55         {
56             cout << mat1[i][j] << " ";
57         }
58         cout << endl;
59     }
60     cout << "Total sum: " << SumMatrix(m1, n1, mat1) << endl << endl;
61
62     int* rowSums = new int[m1];
63     RowSums(m1, n1, mat1, rowSums);
64
65     cout << "Row sums:" << endl;
66     for (int i = 0; i < m1; i++)
67     {
68         cout << "Row " << (i + 1) << ": " << rowSums[i] << endl;
69     }
70
71     delete[] rowSums;
72     for (int i = 0; i < m1; i++)
73     {
74         delete[] mat1[i];
75     }
76     delete[] mat1;
77
78     return 0;

```

```
79 }
```

Output:

Matrix 3x4:

1 2 3 4

5 6 7 8

9 10 11 12

Total sum: 78

Row sums:

Row 1: 10

Row 2: 26

Row 3: 42

Introduction to Python

In the exercises of this chapter the programs are written and executed with the conventions of the chapter «Introduction to Python», summarized below.

Warning

- The program of an exercise is saved in a file `<name>.py` and executed with the command

```
python3 <name>.py
```

- The test instances are lists written in the code, for example `arr1 = [9, 12, 3, 16, 19, 11, 12]`, and are passed to the functions without their size: a function obtains the size n of a list `a` with `len(a)`.
- The elements of a list of size n are `a[0]`, `a[1]`, ..., `a[n-1]`: the entry `a[i]` of an instance written as `a = [...]` in the text corresponds to `a[i-1]` in the code, and the indices printed by the programs start from 0.
- The main instructions are written in the block `if __name__ == "__main__":`; a list is printed with `print`, which displays it in square brackets.
- The results that a function has to give back together are returned as a tuple (`return x, y`) and received with a multiple assignment (`x, y = f(a)`).
- The format `:.4f` inside an f-string displays a floating-point number with exactly 4 digits after the decimal point.

Exercise 7.1 — Introduction to Python (E1)

Given an array a of $n \geq 1$ integer numbers, we consider the problem of finding the maximum value in the array.

Questions

1. Write a complete Python program that finds the maximum value in a list a of n integer numbers. The program should:
 - define a function `array_max_value(a)` that implements the algorithm;
 - in the `__main__` block, create test lists and call the function;
 - print the results for each test case.
2. Execute the program on the following instances:
 - Instance 1: $a = [9, 12, 3, 16, 19, 11, 12]$ with $n = 7$
 - Instance 2: $a = [5, 2, 8, 1]$ with $n = 4$
 - Instance 3: $a = [42]$ with $n = 1$

Show the command that executes the program and its output for all test cases.

3. Modify the function so that it returns both the maximum value and its index in the list. Execute the modified version on Instance 1 and show the output.

Solution

1. Complete Python program:

```

1 def array_max_value(a):
2     """Find and return the maximum value in list a."""
3     max_val = a[0] # Initialize with the first element
4
5     for i in range(1, len(a)):
6         if a[i] > max_val:
7             max_val = a[i] # Update the maximum
8
9     return max_val
10
11
12 if __name__ == "__main__":
13     # Test case 1
14     arr1 = [9, 12, 3, 16, 19, 11, 12]
15     print(f"Test 1 - List: {arr1}")
16     print(f"Maximum value: {array_max_value(arr1)}")
17     print()
18
19     # Test case 2
20     arr2 = [5, 2, 8, 1]
21     print(f"Test 2 - List: {arr2}")
22     print(f"Maximum value: {array_max_value(arr2)}")
23     print()
24
25     # Test case 3
26     arr3 = [42]
```

```

27     print(f"Test 3 - List: {arr3}")
28     print(f"Maximum value: {array_max_value(arr3)}")

```

For Instance 3 the loop `for i in range(1, 1)` performs no iteration, and the function returns the only element `a[0]`.

2. **Execution.** If the program is saved in the file `max_value.py`, it is executed with the command

```
python3 max_value.py
```

Output:

```
Test 1 - List: [9, 12, 3, 16, 19, 11, 12]
Maximum value: 19
```

```
Test 2 - List: [5, 2, 8, 1]
Maximum value: 8
```

```
Test 3 - List: [42]
Maximum value: 42
```

3. **Modified function returning the maximum value and its index:**

```

1 def array_max_value_with_index(a):
2     """Return (max_value, index) of the maximum element in list a."""
3     max_val = a[0]
4     idx = 0
5
6     for i in range(1, len(a)):
7         if a[i] > max_val:
8             max_val = a[i]
9             idx = i  # Store the index
10
11     return max_val, idx
12
13
14 if __name__ == "__main__":
15     arr1 = [9, 12, 3, 16, 19, 11, 12]
16     max_val, idx = array_max_value_with_index(arr1)
17     print(f"Maximum value: {max_val}, at index: {idx}")

```

Output:

```
Maximum value: 19, at index: 4
```

For Instance 1 the function returns the tuple (19, 4): the maximum value 19 is at index 4 of the list (indices from 0), that is, in position 5 of the array `a[1..7]` of the text. If the maximum value appears more than once, the function returns the index of its first occurrence, because `max_val` and `idx` are updated only when `a[i] > max_val` (strict inequality).

Exercise 7.2 — Introduction to Python (E2)

Given an array a of $n \geq 1$ integer numbers, we consider the problem of finding its minimum value.

Questions

1. Write a complete **Python** program that finds the minimum value in a list a of n integer numbers. The program should:
 - define a function `array_min_value(a)` that implements the algorithm;
 - in the `__main__` block, create test lists and call the function;
 - print the results for each test case.
2. Execute the program on the following instances:
 - Instance 1: $a = [9, 12, 3, 16, 19, 11, 12]$ with $n = 7$
 - Instance 2: $a = [5, 2, 8, 1]$ with $n = 4$
 - Instance 3: $a = [42]$ with $n = 1$

Show the command that executes the program and its output for all test cases.

3. Modify the function so that it returns both the minimum value and its index in the list. Execute the modified version on Instance 1 and show the output.

Solution

1. Complete Python program:

```

1 def array_min_value(a):
2     """Find and return the minimum value in list a."""
3     min_val = a[0] # Initialize with the first element
4
5     for i in range(1, len(a)):
6         if a[i] < min_val:
7             min_val = a[i] # Update the minimum
8
9     return min_val
10
11
12 if __name__ == "__main__":
13     # Test case 1
14     arr1 = [9, 12, 3, 16, 19, 11, 12]
15     print(f"Test 1 - List: {arr1}")
16     print(f"Minimum value: {array_min_value(arr1)}")

```

```

17     print()
18
19     # Test case 2
20     arr2 = [5, 2, 8, 1]
21     print(f"Test 2 - List: {arr2}")
22     print(f"Minimum value: {array_min_value(arr2)}")
23     print()
24
25     # Test case 3
26     arr3 = [42]
27     print(f"Test 3 - List: {arr3}")
28     print(f"Minimum value: {array_min_value(arr3)}")

```

For Instance 3 the loop `for i in range(1, 1)` performs no iteration, and the function returns the only element `a[0]`.

2. **Execution.** If the program is saved in the file `min_value.py`, it is executed with the command

```
python3 min_value.py
```

Output:

```
Test 1 - List: [9, 12, 3, 16, 19, 11, 12]
Minimum value: 3
```

```
Test 2 - List: [5, 2, 8, 1]
Minimum value: 1
```

```
Test 3 - List: [42]
Minimum value: 42
```

3. **Modified function returning the minimum value and its index:**

```

1 def array_min_value_with_index(a):
2     """Return (min_value, index) of the minimum element in list a."""
3     min_val = a[0]
4     idx = 0
5
6     for i in range(1, len(a)):
7         if a[i] < min_val:
8             min_val = a[i]
9             idx = i # Store the index
10
11     return min_val, idx
12
13

```

```

14 if __name__ == "__main__":
15     arr1 = [9, 12, 3, 16, 19, 11, 12]
16     min_val, idx = array_min_value_with_index(arr1)
17     print(f"Minimum value: {min_val}, at index: {idx}")

```

Output:

Minimum value: 3, at index: 2

For Instance 1 the function returns the tuple (3, 2): the minimum value 3 is at index 2 of the list (indices from 0), that is, in position 3 of the array $a[1..7]$ of the text. If the minimum value appears more than once, the function returns the index of its first occurrence, because `min_val` and `idx` are updated only when $a[i] < \text{min_val}$ (strict inequality).

Exercise 7.3 — Introduction to Python (E3)

Given an array a of $n \geq 1$ integer numbers, we consider the problem of computing the sum and average of all its elements.

Questions

1. Write a complete **Python** program that computes the sum and the average of the elements of a list a of n integer numbers. The program should:
 - define a function `array_sum(a)` that returns the sum of all elements;
 - define a function `array_average(a)` that returns the average of all elements;
 - in the `__main__` block, create test lists, call both functions, and print the results, with the average rounded to 4 decimal digits.
2. Execute the program on the following instances:
 - Instance 1: $a = [9, 12, 3, 16, 19, 11, 12]$ with $n = 7$
 - Instance 2: $a = [5, 2, 8, 1]$ with $n = 4$
 - Instance 3: $a = [42]$ with $n = 1$

Show the command that executes the program and its output for all test cases.

3. Combine the two functions into a single function `array_sum_and_average(a)` that returns both the sum and the average as a tuple. Execute it on Instance 1 and show the output.

Solution

1. Complete Python program:

```

1 def array_sum(a):
2     """Compute and return the sum of all elements in list a."""
3     total = 0
4     for x in a:
5         total += x
6     return total
7
8
9 def array_average(a):
10    """Compute and return the average of all elements in list a."""
11    return array_sum(a) / len(a)
12
13
14 if __name__ == "__main__":
15     # Test case 1
16     arr1 = [9, 12, 3, 16, 19, 11, 12]
17     print(f"Test 1 - List: {arr1}")
18     print(f"Sum:      {array_sum(arr1)}")
19     print(f"Average: {array_average(arr1):.4f}")
20     print()
21
22     # Test case 2
23     arr2 = [5, 2, 8, 1]
24     print(f"Test 2 - List: {arr2}")
25     print(f"Sum:      {array_sum(arr2)}")
26     print(f"Average: {array_average(arr2):.4f}")
27     print()
28
29     # Test case 3
30     arr3 = [42]
31     print(f"Test 3 - List: {arr3}")
32     print(f"Sum:      {array_sum(arr3)}")
33     print(f"Average: {array_average(arr3):.4f}")

```

The loop `for x in a` visits the elements of the list directly. The operator `/` always returns a float, so no type conversion is needed (in C++ the sum must be converted to `double` before the division). The function `array_average` requires $n \geq 1$: for an empty list the division by `len(a) = 0` stops the program with a `ZeroDivisionError`.

2. **Execution.** If the program is saved in the file `array_sum.py`, it is executed with the command

```
python3 array_sum.py
```

Output:

Test 1 - List: [9, 12, 3, 16, 19, 11, 12]
 Sum: 82
 Average: 11.7143

Test 2 - List: [5, 2, 8, 1]
 Sum: 16
 Average: 4.0000

Test 3 - List: [42]
 Sum: 42
 Average: 42.0000

For Instance 1 the exact average is $82/7 = 11.714285\dots$, displayed as 11.7143.

3. Combined function:

```

1 def array_sum_and_average(a):
2     """Return (sum, average) of list a."""
3     total = 0
4     for x in a:
5         total += x
6     avg = total / len(a)
7     return total, avg
8
9
10 if __name__ == "__main__":
11     arr1 = [9, 12, 3, 16, 19, 11, 12]
12     s, avg = array_sum_and_average(arr1)
13     print(f"Sum: {s}, Average: {avg:.4f}")

```

Output:

Sum: 82, Average: 11.7143

The combined function visits the list only once, while `array_average` of the first program calls `array_sum` and visits the list a second time.

Exercise 7.4 — Introduction to Python (E4)

Given an array a of n integer numbers and a value m , we consider the problem of determining whether the value m is present or not in the array.

Questions

1. Write a complete Python program that performs a **linear search** in a list a . The program should:

- define a function `linear_search(a, m)` that searches for the value `m` in the list `a`;
 - return the index of the first occurrence of `m` if it is found, or `-1` if it is not found;
 - in the `__main__` block, run the test cases and print the results.
2. Execute the program on the list `a = [9, 12, 3, 16, 19, 11, 12]` with the following values:
- `m = 16` (present in the list)
 - `m = 12` (present twice: the index of the first occurrence is returned)
 - `m = 7` (not present in the list)
- Show the command that executes the program and its output for all test cases.
3. Modify the function so that it returns the **list of all indices** where the value `m` appears (instead of only the first occurrence). Execute it with `m = 12` and show the output.

Solution

1. Complete Python program:

```

1 def linear_search(a, m):
2     """Search for the value m in list a.
3     Return the index of the first occurrence, or -1 if not found."""
4     for i in range(len(a)):
5         if a[i] == m:
6             return i # Found: return the index
7     return -1 # Not found
8
9
10 def print_result(a, m):
11     """Search for m in a and print the result."""
12     idx = linear_search(a, m)
13     if idx != -1:
14         print(f"Value {m} found at index {idx}")
15     else:
16         print(f"Value {m} not found")
17
18
19 if __name__ == "__main__":
20     a = [9, 12, 3, 16, 19, 11, 12]
21     print(f"List: {a}")
22     print()
23
24     print_result(a, 16)
25     print_result(a, 12)
26     print_result(a, 7)

```

The instruction `return i` terminates the function, and therefore the loop, at the first occurrence of `m`; the instruction `return -1` is reached only when the loop has visited the whole list without finding `m`. The value `-1` is used because it cannot be an index of the list. The auxiliary function `print_result` avoids writing three times the same instructions.

2. **Execution.** If the program is saved in the file `linear_search.py`, it is executed with the command

```
python3 linear_search.py
```

Output:

```
List: [9, 12, 3, 16, 19, 11, 12]
```

```
Value 16 found at index 3
```

```
Value 12 found at index 1
```

```
Value 7 not found
```

The value 12 is at the indices 1 and 6, and the function returns the first one. In the array `a[1..7]` of the text, the indices 3 and 1 correspond to the positions 4 and 2.

3. **Function returning all the occurrences:**

```
1 def linear_search_all(a, m):
2     """Return the list of all indices where the value m appears in a.
3     Return an empty list if m is not found."""
4     indices = []
5     for i in range(len(a)):
6         if a[i] == m:
7             indices.append(i)
8     return indices
9
10
11 if __name__ == "__main__":
12     a = [9, 12, 3, 16, 19, 11, 12]
13
14     m = 12
15     result = linear_search_all(a, m)
16     if result: # A non-empty list is true, an empty list is false
17         print(f"Value {m} found at indices: {result}")
18     else:
19         print(f"Value {m} not found")
```

Output:

```
Value 12 found at indices: [1, 6]
```

Unlike `linear_search`, this function always visits the whole list, because it cannot stop at the first occurrence. The list `indices` starts empty and grows with `append`; when it is used as a condition, an empty list counts as `False`.

Exercise 7.5 — Introduction to Python (E5)

Given an array a of $n \geq 1$ integer numbers, we consider the problem of computing in a single pass the maximum, minimum, sum and average of all elements in the array.

Questions

- Write a complete Python program that computes in a **single pass** the four statistics (maximum, minimum, sum, average) of a list a of n integer numbers. The program should:
 - define a function `array_stats(a)` that computes the four values with a single loop;
 - return the four values as a tuple `(max_val, min_val, total, avg)`;
 - in the `__main__` block, run the test cases and print the results.
- Execute the program on the following instances:
 - Instance 1: $a = [9, 12, 3, 16, 19, 11, 12]$ with $n = 7$
 - Instance 2: $a = [5, 2, 8, 1]$ with $n = 4$
 - Instance 3: $a = [42]$ with $n = 1$

Show the command that executes the program and its output for all test cases.

- Modify the function so that it also returns the indices of the maximum and of the minimum element. Execute it on Instance 1 and show the output.

Solution

1. Complete Python program:

```

1 def array_stats(a):
2     """Compute max, min, sum, and average in a single pass.
3     Return (max_val, min_val, total, avg)."""
4     max_val = a[0]
5     min_val = a[0]
6     total = a[0]
7
8     for i in range(1, len(a)):
9         if a[i] > max_val:
10             max_val = a[i]
11         if a[i] < min_val:
12             min_val = a[i]
13         total += a[i]
14 
```

```

15     avg = total / len(a)
16     return max_val, min_val, total, avg
17
18
19 def print_stats(k, a):
20     """Print the statistics of the test case k."""
21     max_v, min_v, s, avg = array_stats(a)
22     print(f"Test {k} - List: {a}")
23     print(f"  Maximum: {max_v}")
24     print(f"  Minimum: {min_v}")
25     print(f"  Sum:      {s}")
26     print(f"  Average: {avg:.4f}")
27
28
29 if __name__ == "__main__":
30     print_stats(1, [9, 12, 3, 16, 19, 11, 12])
31     print()
32     print_stats(2, [5, 2, 8, 1])
33     print()
34     print_stats(3, [42])

```

The three variables are initialized with the first element, and the loop visits the other $n - 1$ elements once: at each iteration the current element is compared with the maximum and with the minimum, and it is added to the sum.

2. **Execution.** If the program is saved in the file `array_stats.py`, it is executed with the command

```
python3 array_stats.py
```

Output:

```
Test 1 - List: [9, 12, 3, 16, 19, 11, 12]
```

```
Maximum: 19
Minimum: 3
Sum:      82
Average: 11.7143
```

```
Test 2 - List: [5, 2, 8, 1]
```

```
Maximum: 8
Minimum: 1
Sum:      16
Average: 4.0000
```

```
Test 3 - List: [42]
```

```
Maximum: 42
Minimum: 42
Sum:      42
Average: 42.0000
```

3. Extended function also returning the indices:

```

1 def array_stats_with_indices(a):
2     """Compute all the statistics, including the positions.
3     Return (max_val, max_idx, min_val, min_idx, total, avg)."""
4     max_val = a[0]
5     max_idx = 0
6     min_val = a[0]
7     min_idx = 0
8     total = a[0]
9
10    for i in range(1, len(a)):
11        if a[i] > max_val:
12            max_val = a[i]
13            max_idx = i
14        if a[i] < min_val:
15            min_val = a[i]
16            min_idx = i
17        total += a[i]
18
19    avg = total / len(a)
20    return max_val, max_idx, min_val, min_idx, total, avg
21
22
23 if __name__ == "__main__":
24     a = [9, 12, 3, 16, 19, 11, 12]
25     max_v, max_i, min_v, min_i, s, avg = array_stats_with_indices(a)
26     print(f"List:      {a}")
27     print(f"Max:       {max_v} at index {max_i}")
28     print(f"Min:       {min_v} at index {min_i}")
29     print(f"Sum:        {s}")
30     print(f"Average: {avg:.4f}")

```

Output:

```

List:      [9, 12, 3, 16, 19, 11, 12]
Max:       19 at index 4
Min:       3 at index 2
Sum:       82
Average: 11.7143

```

The maximum 19 is at index 4 and the minimum 3 at index 2 (indices from 0), that is, in the positions 5 and 3 of the array $a[1..7]$ of the text.

Introduction to Java

In the exercises of this chapter the programs are written, compiled and executed with the conventions of the chapter «Introduction to Java», summarized below.

Warning

- The program of an exercise is a class `Main`, saved in the file `Main.java` (each exercise in its own folder), compiled and executed with the commands

```
javac Main.java
java Main
```

- The methods are `static` methods of the class `Main`. The test instances are arrays written in the code, for example `int[] arr1 = {9, 12, 3, 16, 19, 11, 12};`, and are passed to the methods without their size: a method obtains the size n of an array `a` with `a.length`.
- The elements of an array of size n are `a[0]`, `a[1]`, ..., `a[n-1]`: the entry `a[i]` of an instance written as `a = [...]` in the text corresponds to `a[i-1]` in the code, and the indices printed by the programs start from 0.
- A method returns at most one value: the results that a method has to give back together are returned in an array (`int[]` or `double[]`).
- When an array is printed with a loop, every value is followed by a space, so the printed lines end with a space.
- The format `%.4f` of `printf` displays a floating-point number with exactly 4 digits after the decimal point. The outputs are the ones of a computer with English language settings: with Italian settings `printf` writes the decimal comma (11,7143).

Exercise 8.1 — Introduction to Java (*E1*)

Given an array a of $n \geq 1$ integer numbers, we consider the problem of finding the maximum value in the array.

Questions

- Write a complete Java program that finds the maximum value in an array a of n integer numbers. The program should:
 - define a static method `int arrayMaxValue(int[] a)` inside the class `Main`;
 - in the `main` method, create test arrays and call the method;
 - print the results for each test case.
- Execute the program on the following instances:
 - Instance 1: $a = [9, 12, 3, 16, 19, 11, 12]$ with $n = 7$
 - Instance 2: $a = [5, 2, 8, 1]$ with $n = 4$
 - Instance 3: $a = [42]$ with $n = 1$

Show the compilation and execution commands and the output for all test cases.
- Modify the method so that it returns both the maximum value and its index. Since a Java method returns only one value, return an array `int[]` of two elements: `{maxValue, maxIndex}`. Execute the modified version on Instance 1 and show the output.

Solution

1. Complete Java program:

```

1 public class Main {
2
3     static int arrayMaxValue(int[] a) {
4         int max = a[0]; // Initialize with the first element
5
6         for (int i = 1; i < a.length; i++) {
7             if (a[i] > max) {
8                 max = a[i]; // Update the maximum
9             }
10        }
11        return max;
12    }
13
14    public static void main(String[] args) {
15        int[] arr1 = {9, 12, 3, 16, 19, 11, 12};
16        System.out.print("Test 1 - Array: ");
17        for (int x : arr1) System.out.print(x + " ");
18        System.out.println();
19        System.out.println("Maximum value: " + arrayMaxValue(arr1));
20        System.out.println();
21
22        int[] arr2 = {5, 2, 8, 1};
23        System.out.print("Test 2 - Array: ");
24        for (int x : arr2) System.out.print(x + " ");

```

```

25     System.out.println();
26     System.out.println("Maximum value: " + arrayMaxValue(arr2));
27     System.out.println();
28
29     int[] arr3 = {42};
30     System.out.print("Test 3 - Array: ");
31     for (int x : arr3) System.out.print(x + " ");
32     System.out.println();
33     System.out.println("Maximum value: " + arrayMaxValue(arr3));
34 }
35 }

```

The arrays are printed with the enhanced `for` loop, which visits their elements directly. For Instance 3 the condition `1 < a.length` is false from the beginning, the loop performs no iteration, and the method returns the only element `a[0]`.

2. Compilation and execution:

```

javac Main.java
java Main

```

Output:

```

Test 1 - Array: 9 12 3 16 19 11 12
Maximum value: 19

```

```

Test 2 - Array: 5 2 8 1
Maximum value: 8

```

```

Test 3 - Array: 42
Maximum value: 42

```

3. Modified method returning the maximum value and its index:

```

1 public class Main {
2
3     // Returns int[]{maxValue, maxIndex}
4     static int[] arrayMaxValueWithIndex(int[] a) {
5         int max = a[0];
6         int idx = 0;
7
8         for (int i = 1; i < a.length; i++) {
9             if (a[i] > max) {
10                 max = a[i];
11                 idx = i; // Store the index
12             }
13         }
14         return new int[]{max, idx};
15     }

```

```

16
17     public static void main(String[] args) {
18         int[] arr1 = {9, 12, 3, 16, 19, 11, 12};
19         int[] result = arrayMaxValueWithIndex(arr1);
20         System.out.println("Maximum value: " + result[0]
21                             + ", at index: " + result[1]);
22     }
23 }

```

Output:

Maximum value: 19, at index: 4

For Instance 1 the method returns the array {19, 4}: the maximum value 19 is at index 4 (indices from 0), that is, in position 5 of the array $a[1..7]$ of the text. If the maximum value appears more than once, the method returns the index of its first occurrence, because `max` and `idx` are updated only when $a[i] > \text{max}$ (strict inequality).

Exercise 8.2 — Introduction to Java (E2)

Given an array a of $n \geq 1$ integer numbers, we consider the problem of finding its minimum value.

Questions

1. Write a complete Java program that finds the minimum value in an array a of n integer numbers. The program should:
 - define a static method `int arrayMinValue(int[] a)` inside the class `Main`;
 - in the `main` method, create test arrays and call the method;
 - print the results for each test case.
2. Execute the program on the following instances:
 - Instance 1: $a = [9, 12, 3, 16, 19, 11, 12]$ with $n = 7$
 - Instance 2: $a = [5, 2, 8, 1]$ with $n = 4$
 - Instance 3: $a = [42]$ with $n = 1$

Show the compilation and execution commands and the output for all test cases.

3. Modify the method so that it returns both the minimum value and its index. Since a Java method returns only one value, return an array `int[]` of two elements: `{minValue, minIndex}`. Execute the modified version on Instance 1 and show the output.

Solution

1. Complete Java program:

```

1 public class Main {
2
3     static int arrayMinValue(int[] a) {
4         int min = a[0]; // Initialize with the first element
5
6         for (int i = 1; i < a.length; i++) {
7             if (a[i] < min) {
8                 min = a[i]; // Update the minimum
9             }
10        }
11        return min;
12    }
13
14    public static void main(String[] args) {
15        int[] arr1 = {9, 12, 3, 16, 19, 11, 12};
16        System.out.print("Test 1 - Array: ");
17        for (int x : arr1) System.out.print(x + " ");
18        System.out.println();
19        System.out.println("Minimum value: " + arrayMinValue(arr1));
20        System.out.println();
21
22        int[] arr2 = {5, 2, 8, 1};
23        System.out.print("Test 2 - Array: ");
24        for (int x : arr2) System.out.print(x + " ");
25        System.out.println();
26        System.out.println("Minimum value: " + arrayMinValue(arr2));
27        System.out.println();
28
29        int[] arr3 = {42};
30        System.out.print("Test 3 - Array: ");
31        for (int x : arr3) System.out.print(x + " ");
32        System.out.println();
33        System.out.println("Minimum value: " + arrayMinValue(arr3));
34    }
35 }

```

The arrays are printed with the enhanced for loop, which visits their elements directly. For Instance 3 the condition `1 < a.length` is false from the beginning, the loop performs no iteration, and the method returns the only element `a[0]`.

2. Compilation and execution:

```

javac Main.java
java Main

```

Output:

```

Test 1 - Array: 9 12 3 16 19 11 12
Minimum value: 3

```

Test 2 - Array: 5 2 8 1
Minimum value: 1

Test 3 - Array: 42
Minimum value: 42

3. Modified method returning the minimum value and its index:

```

1 public class Main {
2
3     // Returns int[]{minValue, minIndex}
4     static int[] arrayMinValueWithIndex(int[] a) {
5         int min = a[0];
6         int idx = 0;
7
8         for (int i = 1; i < a.length; i++) {
9             if (a[i] < min) {
10                 min = a[i];
11                 idx = i; // Store the index
12             }
13         }
14         return new int[]{min, idx};
15     }
16
17     public static void main(String[] args) {
18         int[] arr1 = {9, 12, 3, 16, 19, 11, 12};
19         int[] result = arrayMinValueWithIndex(arr1);
20         System.out.println("Minimum value: " + result[0]
21                             + ", at index: " + result[1]);
22     }
23 }

```

Output:

Minimum value: 3, at index: 2

For Instance 1 the method returns the array {3, 2}: the minimum value 3 is at index 2 (indices from 0), that is, in position 3 of the array $a[1..7]$ of the text. If the minimum value appears more than once, the method returns the index of its first occurrence, because `min` and `idx` are updated only when $a[i] < \text{min}$ (strict inequality).

Exercise 8.3 — Introduction to Java (E3)

Given an array a of $n \geq 1$ integer numbers, we consider the problem of computing the sum and average of all its elements.

Questions

1. Write a complete Java program that computes the sum and the average of the elements of an array a of n integer numbers. The program should:
 - define a static method `int arraySum(int[] a)` that returns the sum;
 - define a static method `double arrayAverage(int[] a)` that returns the average;
 - in the `main` method, create test arrays, call both methods, and print the results, with the average rounded to 4 decimal digits.
2. Execute the program on the following instances:
 - Instance 1: $a = [9, 12, 3, 16, 19, 11, 12]$ with $n = 7$
 - Instance 2: $a = [5, 2, 8, 1]$ with $n = 4$
 - Instance 3: $a = [42]$ with $n = 1$

Show the compilation and execution commands and the output for all test cases.

3. Write a single method `arraySumAndAverage(int[] a)` that returns both the sum and the average. Since a Java method returns only one value, return the result as an array `double[]` of two elements: {sum, average}. Execute it on Instance 1 and show the output.

Solution

1. Complete Java program:

```

1 public class Main {
2
3     static int arraySum(int[] a) {
4         int total = 0;
5         for (int i = 0; i < a.length; i++) {
6             total += a[i];
7         }
8         return total;
9     }
10
11     static double arrayAverage(int[] a) {
12         return (double) arraySum(a) / a.length;
13     }
14
15     static void printTest(int k, int[] a) {
16         System.out.print("Test " + k + " - Array: ");
17         for (int x : a) System.out.print(x + " ");
18         System.out.println();
19         System.out.println("Sum:      " + arraySum(a));
20         System.out.printf("Average: %.4f%n", arrayAverage(a));
21     }
22
23     public static void main(String[] args) {

```

```

24     int[] arr1 = {9, 12, 3, 16, 19, 11, 12};
25     printTest(1, arr1);
26     System.out.println();
27
28     int[] arr2 = {5, 2, 8, 1};
29     printTest(2, arr2);
30     System.out.println();
31
32     int[] arr3 = {42};
33     printTest(3, arr3);
34 }
35 }

```

The conversion (`double`) is necessary: without it, `arraySum(a) / a.length` would be the integer division, and for Instance 1 the average would be $82/7 = 11$. The method `printTest` prints the array and the two results, and avoids writing three times the same instructions.

2. Compilation and execution:

```

javac Main.java
java Main

```

Output:

```

Test 1 - Array: 9 12 3 16 19 11 12
Sum:      82
Average: 11.7143

```

```

Test 2 - Array: 5 2 8 1
Sum:      16
Average: 4.0000

```

```

Test 3 - Array: 42
Sum:      42
Average: 42.0000

```

For Instance 1 the exact average is $82/7 = 11.714285\dots$, displayed as 11.7143.

3. Combined method:

```

1 public class Main {
2
3     // Returns double[]{sum, average}
4     static double[] arraySumAndAverage(int[] a) {
5         int total = 0;
6         for (int x : a) total += x;
7         double avg = (double) total / a.length;
8         return new double[]{total, avg};
9     }
10 }

```

```

9      }
10
11     public static void main(String[] args) {
12         int[] arr1 = {9, 12, 3, 16, 19, 11, 12};
13         double[] result = arraySumAndAverage(arr1);
14         System.out.printf("Sum: %.0f, Average: %.4f\n",
15                             result[0], result[1]);
16     }
17 }

```

Output:

Sum: 82, Average: 11.7143

The integer `total` is converted automatically into a `double` when it is stored in the array `double[]`; for this reason it is printed with the format `%.0f` (no decimal digits) instead of `%d`, which accepts only integers.

Exercise 8.4 — Introduction to Java (E4)

Given an array a of n integer numbers and a value m , we consider the problem of determining whether the value m is present or not in the array.

Questions

- Write a complete Java program that performs a **linear search** in an array a . The program should:
 - define a static method `int linearSearch(int[] a, int m)` that searches for the value m in the array a ;
 - return the index of the first occurrence of m if it is found, or `-1` if it is not found;
 - in the `main` method, run the test cases and print the results.
- Execute the program on the array $a = [9, 12, 3, 16, 19, 11, 12]$ with the following values:
 - $m = 16$ (present in the array)
 - $m = 12$ (present twice: the index of the first occurrence is returned)
 - $m = 7$ (not present in the array)

Show the compilation and execution commands and the output for all test cases.

- Modify the method so that it returns an array `int[]` containing **all the indices** where the value m appears (instead of only the first occurrence). Use an `ArrayList<Integer>` inside the method and convert it into an `int[]` before returning. Execute it with $m = 12$ and show the output.

Solution

1. Complete Java program:

```

1 public class Main {
2
3     static int linearSearch(int[] a, int m) {
4         for (int i = 0; i < a.length; i++) {
5             if (a[i] == m) {
6                 return i; // Found: return the index
7             }
8         }
9         return -1; // Not found
10    }
11
12    static void printResult(int[] a, int m) {
13        int idx = linearSearch(a, m);
14        if (idx != -1) System.out.println("Value " + m + " found at index " + idx);
15        else          System.out.println("Value " + m + " not found");
16    }
17
18    public static void main(String[] args) {
19        int[] a = {9, 12, 3, 16, 19, 11, 12};
20        System.out.print("Array: ");
21        for (int x : a) System.out.print(x + " ");
22        System.out.println();
23        System.out.println();
24
25        printResult(a, 16);
26        printResult(a, 12);
27        printResult(a, 7);
28    }
29 }

```

The instruction `return i` terminates the method, and therefore the loop, at the first occurrence of `m`; the instruction `return -1` is reached only when the loop has visited the whole array without finding `m`. The value `-1` is used because it cannot be an index of the array.

2. Compilation and execution:

```

javac Main.java
java Main

```

Output:

```
Array: 9 12 3 16 19 11 12
```

```
Value 16 found at index 3
```

```
Value 12 found at index 1
```

```
Value 7 not found
```

The value 12 is at the indices 1 and 6, and the method returns the first one. In the array `a[1..7]` of the text, the indices 3 and 1 correspond to the positions 4 and 2.

3. Method returning all the occurrences:

```

1  import java.util.ArrayList;
2
3  public class Main {
4
5      static int[] linearSearchAll(int[] a, int m) {
6          ArrayList<Integer> indices = new ArrayList<>();
7          for (int i = 0; i < a.length; i++) {
8              if (a[i] == m) {
9                  indices.add(i);
10             }
11         }
12         // Conversion of the ArrayList<Integer> into an int[]
13         int[] result = new int[indices.size()];
14         for (int i = 0; i < indices.size(); i++) {
15             result[i] = indices.get(i);
16         }
17         return result;
18     }
19
20     public static void main(String[] args) {
21         int[] a = {9, 12, 3, 16, 19, 11, 12};
22         int m = 12;
23         int[] occurrences = linearSearchAll(a, m);
24
25         if (occurrences.length > 0) {
26             System.out.print("Value " + m + " found at indices: ");
27             for (int idx : occurrences) System.out.print(idx + " ");
28             System.out.println();
29         } else {
30             System.out.println("Value " + m + " not found");
31         }
32     }
33 }

```

Output:

Value 12 found at indices: 1 6

The number of occurrences is not known in advance, so the indices are first collected in an `ArrayList`, which grows with `add`; then an array of the right size `indices.size()` is allocated and filled. Unlike `linearSearch`, this method always visits the whole array. If `m` is not present, the method returns an array of length 0.

Exercise 8.5 — Introduction to Java (E5)

Given an array a of $n \geq 1$ integer numbers, we consider the problem of computing in a single pass the maximum, minimum, sum and average of all elements in the array.

Questions

- Write a complete Java program that computes in a **single pass** the four statistics (maximum, minimum, sum, average) of an array a of n integer numbers. The program should:
 - define a static method `arrayStats(int[] a)` that computes the four values with a single loop;
 - return the results as an array `double[]` of four elements: `{max, min, sum, average}`;
 - in the `main` method, run the test cases and print the results.
- Execute the program on the following instances:
 - Instance 1: $a = [9, 12, 3, 16, 19, 11, 12]$ with $n = 7$
 - Instance 2: $a = [5, 2, 8, 1]$ with $n = 4$
 - Instance 3: $a = [42]$ with $n = 1$

Show the compilation and execution commands and the output for all test cases.

- Modify the method so that it also returns the indices of the maximum and of the minimum element, as an array `double[]` of six elements: `{max, maxIdx, min, minIdx, sum, average}`. Execute it on Instance 1 and show the output.

Solution

1. Complete Java program:

```

1 public class Main {
2
3     // Returns double[]{max, min, sum, average}
4     static double[] arrayStats(int[] a) {
5         int max = a[0];
6         int min = a[0];
7         int total = a[0];
8
9         for (int i = 1; i < a.length; i++) {
10             if (a[i] > max) max = a[i];
11             if (a[i] < min) min = a[i];
12             total += a[i];
13         }

```

```

14
15     double avg = (double) total / a.length;
16     return new double[]{max, min, total, avg};
17 }
18
19 static void printStats(int k, int[] a) {
20     double[] s = arrayStats(a);
21     System.out.print("Test " + k + " - Array: ");
22     for (int x : a) System.out.print(x + " ");
23     System.out.println();
24     System.out.println("    Maximum: " + (int) s[0]);
25     System.out.println("    Minimum: " + (int) s[1]);
26     System.out.println("    Sum:      " + (int) s[2]);
27     System.out.printf("    Average: %.4f%n", s[3]);
28 }
29
30 public static void main(String[] args) {
31     printStats(1, new int[]{9, 12, 3, 16, 19, 11, 12});
32     System.out.println();
33     printStats(2, new int[]{5, 2, 8, 1});
34     System.out.println();
35     printStats(3, new int[]{42});
36 }
37 }

```

The three variables are initialized with the first element, and the loop visits the other $n - 1$ elements once. The four results have different data types, so they are all stored as `double` in the returned array; the integer results are converted back with `(int)` before printing, so that they are displayed without the decimal part (19 instead of 19.0). The expression `new int[]{...}` creates an array directly in the call.

2. Compilation and execution:

```

javac Main.java
java Main

```

Output:

```

Test 1 - Array: 9 12 3 16 19 11 12
    Maximum: 19
    Minimum: 3
    Sum:      82
    Average: 11.7143

Test 2 - Array: 5 2 8 1
    Maximum: 8
    Minimum: 1
    Sum:      16
    Average: 4.0000

Test 3 - Array: 42

```

Maximum: 42
 Minimum: 42
 Sum: 42
 Average: 42.0000

3. Extended method also returning the indices:

```

1 public class Main {
2
3     // Returns double[]{max, maxIdx, min, minIdx, sum, average}
4     static double[] arrayStatsWithIndices(int[] a) {
5         int max = a[0], maxIdx = 0;
6         int min = a[0], minIdx = 0;
7         int total = a[0];
8
9         for (int i = 1; i < a.length; i++) {
10             if (a[i] > max) { max = a[i]; maxIdx = i; }
11             if (a[i] < min) { min = a[i]; minIdx = i; }
12             total += a[i];
13         }
14
15         double avg = (double) total / a.length;
16         return new double[]{max, maxIdx, min, minIdx, total, avg};
17     }
18
19     public static void main(String[] args) {
20         int[] a = {9, 12, 3, 16, 19, 11, 12};
21         double[] r = arrayStatsWithIndices(a);
22         System.out.println("Max: " + (int) r[0] + " at index " + (int) r[1]);
23         System.out.println("Min: " + (int) r[2] + " at index " + (int) r[3]);
24         System.out.println("Sum: " + (int) r[4]);
25         System.out.printf("Avg: %.4f%n", r[5]);
26     }
27 }

```

Output:

Max: 19 at index 4
 Min: 3 at index 2
 Sum: 82
 Avg: 11.7143

The maximum 19 is at index 4 and the minimum 3 at index 2 (indices from 0), that is, in the positions 5 and 3 of the array $a[1..7]$ of the text.