TRANSPORT AND TELECOMMUNICATION INSTITUTE

TRANSPORT AND TELECOMMUNICATION INSTITUTE

ENGINEERING FACULTY

Laboratory work N3

Course: **Programming**

Theme: **One-dimensional arrays**

Student: Igors Oļeiņikovs

Student code: 93642

Group: 4501BTA

Riga
2025

CONTENTS

1.   LABORATORY WORK TASK

Create 3 separate independent algorithms that solve individual task by using arrays. Individual task number is being defined as in previous labs. Create one (!) console program in C/C++ based on created algorithms (all algorithms must be implemented in one program) by using dynamic one-dimensional array of real (!) numbers with size n (n is entered by user). Implement two ways of array filling: manual (array elements are entered by user) and automatic (array elements are random numbers generated in user specified range). User should be able to select a way of filling (for this create a menu for the user). It is not allowed to use additional array in task N3. All transformations should be done in the same array. Add individual task number output in the beginning of the program. Prepare at least 3 test cases for each task with input and output data. Test program by using prepared test cases. Prepare a report according to general report design requirements. For each task there should be a separate block-diagram in report. It is not necessary to describe in detail array filling and array output steps. It is enough just to have a block like "Array arr input (of size n)" or "Array arr filling with n real random numbers in range [a;b]".

2. INDIVIDUAL TASK

1. Calculate sum of positive elements.
2. Calculate product of elements that are located between absolute maximum element and absolute minimum element.
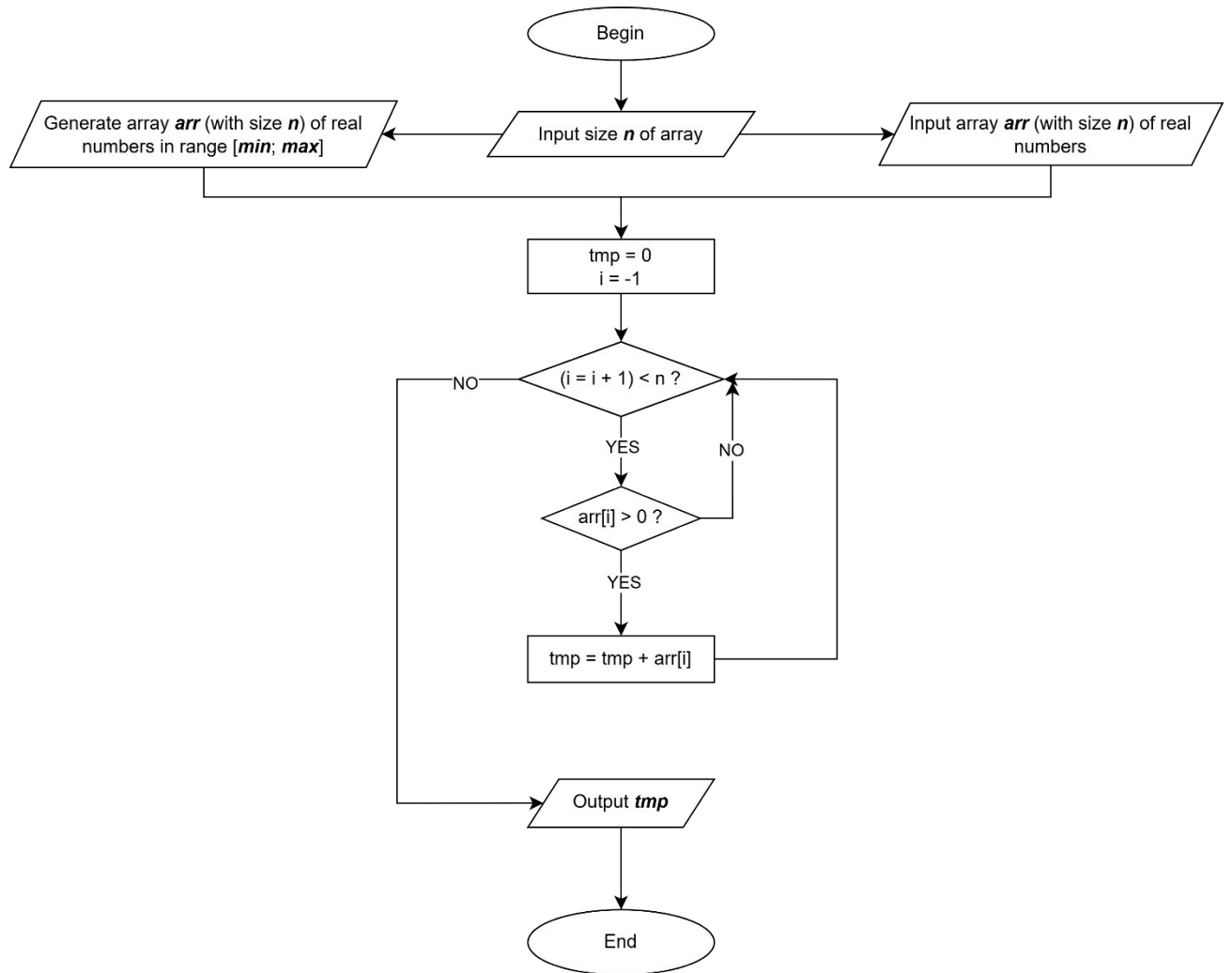3. Arrange elements in descending order.

3. ALGORITHM DATA FLOW
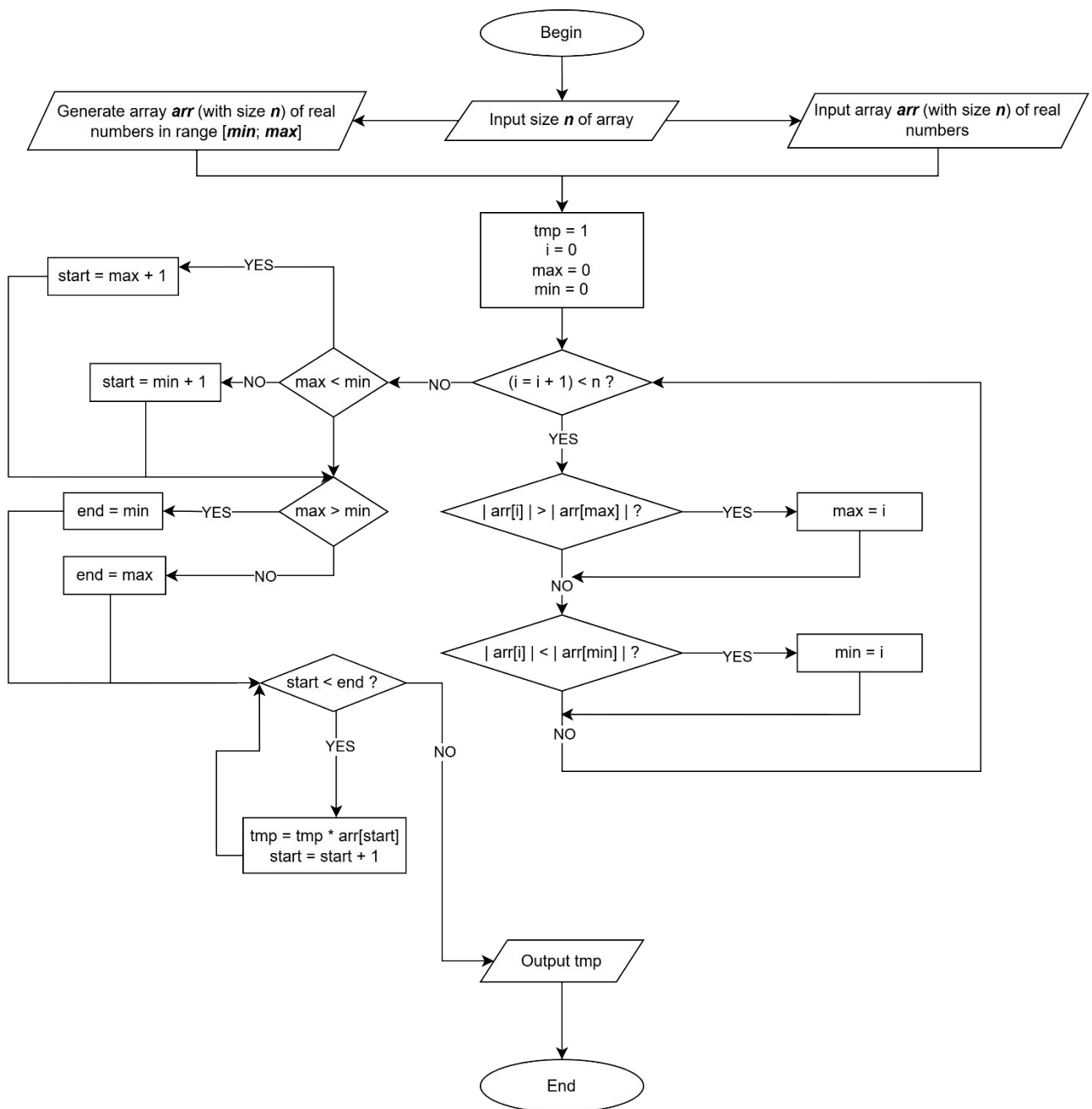

Fig. 1. **Sum of positive elements**

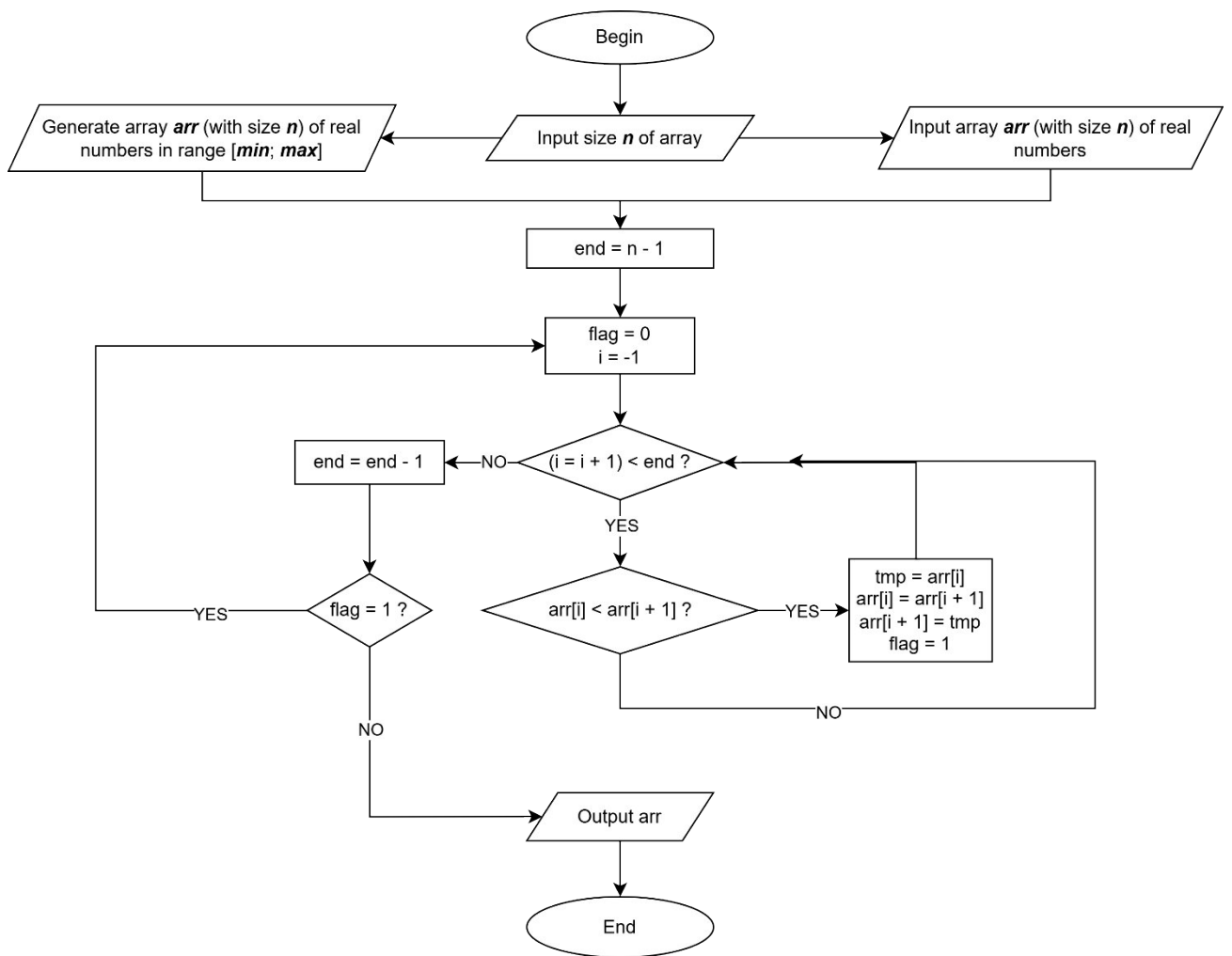Fig. 2. **Product of elements between extremes**

Fig. 3. **Bubble sort (descending)**

4. ALGORITHM DESCRIPTION

The algorithm processes an array of doubles to compute the sum of positives, product of elements positioned between absolute extremes, and descending sort. Supports manual or auto input.

Flow:

- Initialize random seed
- Print task number (93642 % 20)
- Read size n; validate > 0
- Allocate array; if fail, error
- Read flag (1 = manual, 2 = auto); validate
- If manual: for i=0 to n-1, prompt "Enter element i+1: ", read arr[i]
- If auto: read min, max; generate arr[i] = min + (max - min) * rand( ) / RAND_MAX
- Print "Array: " + elements
- Sum positives: tmp = 0; for each arr[i] > 0, tmp += arr[i]; print
- Product: find abs max/min indices; start = min, end = max; while (++start < end) tmp*= arr[start]; print
- Sort descending: bubble sort with flag; print sorted
- Free memory

5. SOURCE CODE

```cpp
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <cmath>

void    ft_error_exit_msg(const char *msg, double *arr)
{
    printf("\n%s\n", msg);
    if (arr)
        free(arr);
    exit (1);
}

int     main(void)
{
    srand(time(NULL));
    int     n, flag, start, end, i, min_i, max_i;
    double  tmp, min, max;
    double  *arr;

    arr = NULL;
    tmp = 0;
    min_i = 0;
    max_i = 0;
    i = -1;
    printf("\nTask number = %d\n\n", 93642 % 20);

printf("=========================================================\n");
    printf("Size: ");
    if (scanf("%d", &n) != 1 || n <= 0)
        ft_error_exit_msg("Invalid size", NULL);

    arr = (double *)malloc(n * sizeof(double));
    if (!arr)
        ft_error_exit_msg("Memory allocation failed", arr);

    printf("Fill array (1 = Manual, 2 = Auto): ");
    if (scanf("%d", &flag) != 1 || (flag != 1 && flag != 2))
        ft_error_exit_msg("Invalid flag", arr);
    if (flag == 1)
    {
        while (++i < n)
        {
            printf("Enter element %d: ", i + 1);
            if (scanf("%lf", &arr[i]) != 1)
```

```c
            ft_error_exit_msg("Invalid input", arr);
        }
    }
    else
    {
        printf("Min: ");
        if (scanf("%lf", &min) != 1)
            ft_error_exit_msg("Invalid min", arr);
        printf("Max: ");
        if (scanf("%lf", &max) != 1)
            ft_error_exit_msg("Invalid max", arr);
        while (++i < n) arr[i] = min + (max - min) * rand() / RAND_MAX;
    }

    printf("Array: ");
    i = -1;
    while (++i < n) printf("%.2f ", arr[i]);

printf("\n\n=====================================================\n\n");

    i = -1;
    while (++i < n)
        if (arr[i] > 0) tmp += arr[i];
    printf("Sum of positive elements: %.2f\n", tmp);

    tmp = 1;
    i = 0;
    while (++i < n)
    {
        if (fabs(arr[i]) > fabs(arr[max_i])) max_i = i;
        if (fabs(arr[i]) < fabs(arr[min_i])) min_i = i;
    }
    start = (max_i < min_i) ? max_i : min_i;
    end = (max_i < min_i) ? min_i : max_i;
    if (end - start <= 1)
    {
        printf("No elements between extremes.\n");
    }
    else
    {
        while (++start < end)
            tmp *= arr[start];

        printf("Product of elements between extremes: %.2f\n", tmp);
    }

    end = n - 1;
```

9

```c
    do
    {
        flag = 0;
        i = -1;
        while (++i < end)
        {
            if (arr[i] < arr[i + 1])
            {
                tmp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = tmp;
                flag = 1;
            }
        }
        end--;
    }
    while (flag);


    printf("Sorted (descending): ");


    i = -1;
    while (++i < n) printf("%.2f ", arr[i]);


    printf("\n============================================================\n");
    free(arr);
    return (0);
}
```

6. RUNNING PROGRAM EXAMPLE

```
altin@G3:~/TSI/Lab_3$ make
g++    -c -o main.o main.cpp
g++ -Wall -Wextra -Werror -pedantic main.o -o main
altin@G3:~/TSI/Lab_3$ valgrind ./main
==146344== Memcheck, a memory error detector
==146344== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==146344== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==146344== Command: ./main
==146344==

Task number = 2

======= Array Processing =======
Size: 10
Fill array (1 = Manual, 2 = Auto): 2
Min: 5
Max: 9
Array: 6.72 8.14 5.88 6.11 7.46 5.24 5.41 7.76 5.97 8.00

======= Results =======
Sum of positive elements: 66.69
Product of elements between extremes: 268.31
Sorted (descending): 8.14 8.00 7.76 7.46 6.72 6.11 5.97 5.88 5.41 5.24
==146344==
==146344== HEAP SUMMARY:
==146344==     in use at exit: 0 bytes in 0 blocks
==146344==   total heap usage: 3 allocs, 3 frees, 2,128 bytes allocated
==146344==
==146344== All heap blocks were freed -- no leaks are possible
==146344==
==146344== For lists of detected and suppressed errors, rerun with: -s
==146344== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
altin@G3:~/TSI/Lab_3$
```

Fig. 4. **Running program example**

7. TESTING

Testing table for tasks 1 - 3

Test N	Initial data		Results						Memory leaks/errors (Valgrind)
	array size	array	Expected			Actual			
			Task 1	Task 2	Task 3	Task 1	Task 2	Task 3	
1	7	-2 5 3 -6 4 17 8	37	-360	17 8 5 4 3 -2 -6	37	-360	17 8 5 4 3 -2 -6	No
			Actual result is correct/incorrect?			correct	correct	correct	
2	0	-	Invalid size			Invalid size			No
3	4	1 1 1 1	4	No elements between extremes	1 1 1 1	4	No elements between extremes	1 1 1 1	No
			Actual result is correct/incorrect?			correct	correct	correct	
4	4	0 5 10 3	18	5	10 5 3 0	18	5	10 5 3 0	No
			Actual result is correct/incorrect?			correct	correct	correct	
5	2	-0.5 3.7	3.7	No elements between extremes	3.7 -0.5	3.7	No elements between extremes	3.7 -0.5	No
			Actual result is correct/incorrect?			correct	correct	correct	
6	-3	-	Invalid size			Invalid size			No
7	4	0.37 9.89 3.57 8.33	22.16	No elements between extremes	9.89 8.33 3.57 0.37	22.16	No elements between extremes	9.89 8.33 3.57 0.37	No
			Actual result is correct/incorrect?			correct	correct	correct	

8. CONCLUSIONS

A C/C++ program was developed to process a one-dimensional array of real numbers, computing the sum of positive elements, the product of elements between extremes (by absolute value), and sorting in descending order. All requirements fulfilled, including dynamic memory management and support for both manual and automatic array initialization. The implementation uses an optimized Bubble Sort with a flag mechanism to detect when the array becomes sorted, reducing unnecessary iterations. The product calculation correctly identifies maximum and minimum absolute values, handling edge cases where indices are adjacent. Standard C/C++ library functions (<cstdio>, <cstdlib>, <cmath>) were used without complications. Memory management is implemented through a dedicated error handler function (ft_error_exit_msg) that conditionally frees the allocated array pointer before returning an error code. This ensures no memory leaks occur when errors are encountered during input validation or array initialization, as the function receives the array pointer and deallocates it if it was previously allocated. The dynamically allocated array is properly freed at program termination. Limitations include $O(n^2)$ worst-case complexity for Bubble Sort on large datasets, basic input validation via scanf, and potential floating-point precision loss. The program performs reliably for valid numeric inputs. Key challenges: correctly computing indices for extremes and managing product calculations for adjacent elements. No algorithmic complications met. Skills acquired include dynamic memory management with malloc/free, random number generation within specified ranges, and array traversal algorithms. The task required approximately 2 hours, primarily spent testing edge cases. This lab reinforced practical understanding of dynamic arrays, pointers, and algorithmic design in C/C++.