TRANSPORT AND TELECOMMUNICATION INSTITUTE

ENGINEERING FACULTY

Laboratory work N2

Course: **Programming**

Theme: Loops. while and do ... while statements

Student: Igors Oļeiņikovs

Student code: 93642

Group: 4501BTA

Riga

2025

Contents

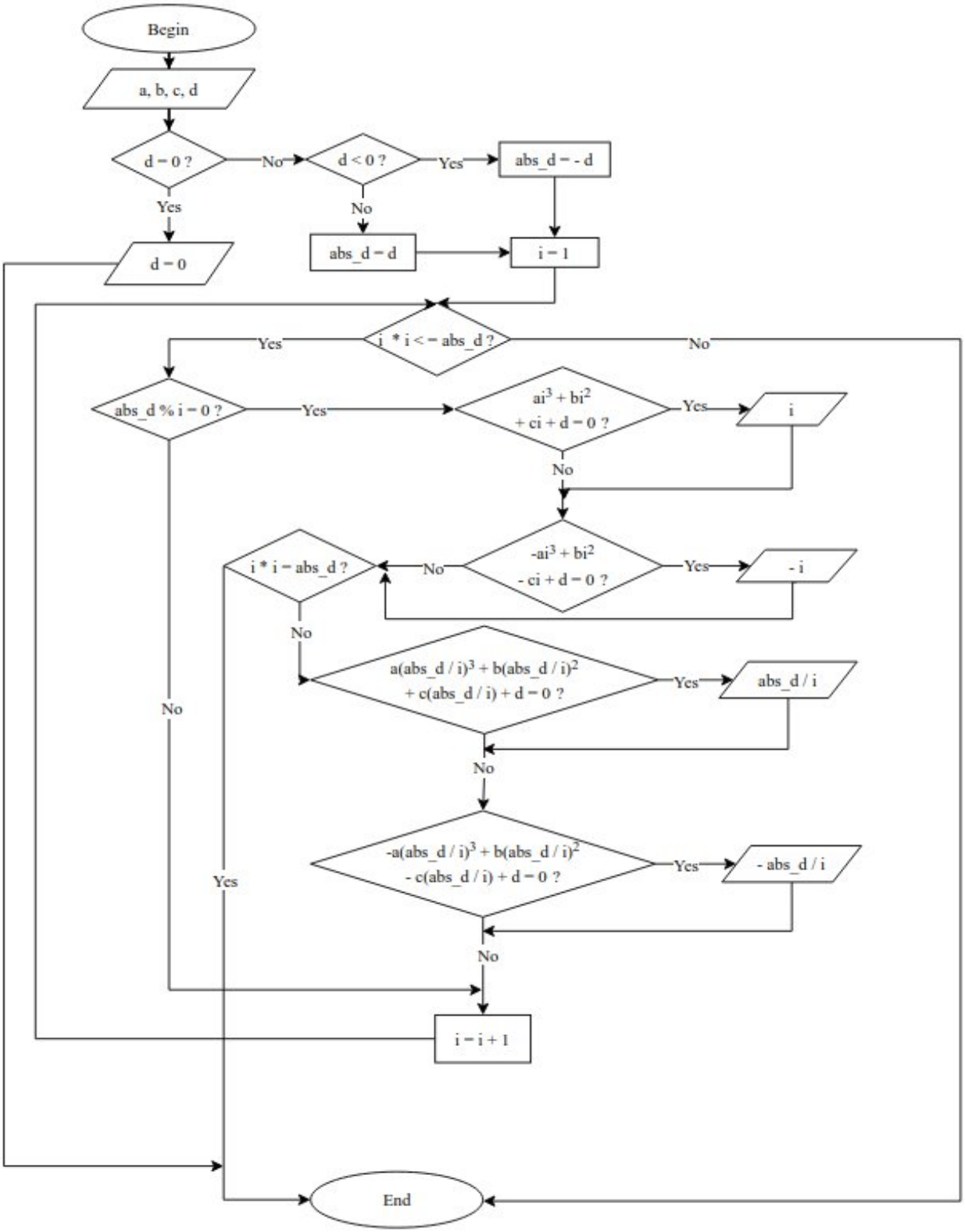
1. Laboratory work task

Create an algorithm that solves individual task using loops. Create a console application in C/C++ based on created algorithm using while or/and do ... while statements. Add individual task number output in the beginning of the program. Test application. Prepare a report according to general report design requirements. Testing table for task should have at least 5 test cases with input data, expected result and actual result for each test case.

2. Individual task

Input: integer numbers a, b, c, d. Find all integer roots of equation $ax^3 + bx^2 + cx + d = 0$ (search roots among possible divisors of number d).

3. Algorithm data flow



Pic.1 **Data flow**

4. Algorithm description

The algorithm is based on the rational root theorem, which states that any rational root of a polynomial with integer coefficients is of the form p/q where p divides the constant term and q divides the leading coefficient. For integer roots (q=1), check all divisors of |d| as potential roots x. For each x, evaluate $ax^3 + bx^2 + cx + d$. Also check -x. If the equation equals zero, print x / -x. Handle d = 0 separately. Flow:

- Read a, b, c, d

- If d = = 0, print "d = 0" and exit

- Compute abs_d = | d |

- For i from 1 to sqrt(abs_d):

    - If abs_d % i = = 0:

    - Test i and abs_d/i as roots

- Print newline

5. Source code

```cpp
#include <cstdio>

struct CubEq
{
    int     a;
    int     b;
    int     c;
    int     d;
};

void    test(const CubEq *eq, int x)
{
    if (eq->a * x * x * x + eq->b * x * x + eq->c * x + eq->d == 0)
        printf("%d ", x);
    if (-eq->a * x * x * x + eq->b * x * x - eq->c * x + eq->d == 0)
        printf("%d ", -x);
}

int     main(void)
{
    CubEq   eq;
    int     abs_d;
    int     i;

    printf("Task number = %d\n", 93642 % 16);
    printf("Enter coefficients a, b, c, d: ");
    scanf("%d %d %d %d", &eq.a, &eq.b, &eq.c, &eq.d);

    if (eq.d == 0)
    {
        printf("d = 0\n");
        return (0);
    }

    abs_d = eq.d < 0 ? -eq.d : eq.d;
    i = 1;

    while (i * i <= abs_d)
    {
        if (abs_d % i == 0)
        {
            test(&eq, i);
            if (i * i == abs_d) break;
            test(&eq, abs_d / i);
        }
        i++;
    }

    printf("\n");
    return (0);
}
```

6. Running program example

```
altin@G3:~/TSI/Lab_2$ g++ -Wall -Wextra -Werror -pedantic Lab_N2.cpp
altin@G3:~/TSI/Lab_2$ ./a.out
Task number = 10
Enter coefficients a, b, c, d: 1 0 0 -1
1
altin@G3:~/TSI/Lab_2$ ./a.out
Task number = 10
Enter coefficients a, b, c, d: 1 1 1 1
-1
altin@G3:~/TSI/Lab_2$ ./a.out
Task number = 10
Enter coefficients a, b, c, d: 2 3 4 0
d = 0
altin@G3:~/TSI/Lab_2$ ./a.out
Task number = 10
Enter coefficients a, b, c, d: 1 -6 11 -6
1 2 3
altin@G3:~/TSI/Lab_2$ ./a.out
Task number = 10
Enter coefficients a, b, c, d: 2 0 0 -16
2
altin@G3:~/TSI/Lab_2$ ./a.out
Task number = 10
Enter coefficients a, b, c, d: 2 -5 -22 24

altin@G3:~/TSI/Lab_2$ ./a.out
Task number = 10
Enter coefficients a, b, c, d: 1 3 3 1
-1
altin@G3:~/TSI/Lab_2$
```

Pic. 2. **Running program example**

7. Testing

Table 1

Testing

Input (a b c d)	Expected Output	Actual Output
1 0 0 -1	1	1
1 1 1 1	-1	-1
1 0 0 2	no output	no output
1 0 0 0	d = 0	d = 0
2 -5 -22 24	no output	no output
0 1 2 3	no output	no output
1 -6 11 -6	1 2 3	1 2 3
1 3 3 1	-1	-1
1 0 0 -8	2	2
-1 0 0 1	1	1
2 0 0 -16	2	2

8. Conclusions

A C/C++ program was developed to identify integer roots of cubic equations by systematically checking divisors of the constant term as potential roots, based on the rational root theorem. Everything required and planned was completed, including reading coefficients, handling the special case of d = 0, and printing found roots. The program has limitations: it only detects integer roots (not all rational roots, as it doesn't check denominators dividing the leading coefficient), may miss irrational or complex roots, assumes integer coefficients (scanf may fail on non-integer input), and could have integer overflow for large coefficients or divisors. It works correctly for valid integer inputs but may not handle invalid data gracefully. The task was moderately challenging. Problems faced included compilation errors due to Makefile issues (mismatched file extensions and object file handling), which were resolved by correcting the build script. No major complications occurred in the algorithm design, as the rational root theorem provided a clear framework, but careful implementation was needed to avoid duplicates and ensure efficiency. Writing the program involved basic C features, with no unresolved issues. New knowledge and skills gained include understanding the rational root theorem for polynomial root finding, practical use of structs for data organization, loop optimization for divisor checking, and debugging build systems. The struct was chosen to logically group the four coefficients, simplifying parameter passing, and passing by pointer to the test function avoids copying the entire struct on each call, enhancing efficiency. The loop iterates only up to the square root of | d | because integer divisors form pairs (if i divides d, so does d/i), allowing both factors to be checked in a single iteration, which optimizes performance for larger values of d. Information resources used were C documentation, online tutorials for the rational root theorem, and compiler error messages for troubleshooting. The work took approximately 2 hours to complete, with the most time-consuming part being writing the report and testing various inputs to ensure correctness. Overall, the lab reinforced programming fundamentals and algorithmic thinking in a practical context.