

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В. Г. ШУХОВА» (БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения

вычислительной техники и автоматизированных

систем

Лабораторная работа №0

по дисциплине: Вычислительная математика

тема: **«Погрешности. Приближенные вычисления.
Вычислительная устойчивость.»**

Выполнил: студент группы
ПВ-233

Мороз Роман Алексеевич

Проверили:

Белгород 2025

Цель работы: Изучить особенности организации вычислительных процессов, связанные с погрешностями, приближенным характером вычислений на компьютерах современного типа, вычислительной устойчивостью.

1) Запустить и проинтерпретировать результаты работы разных вычислительных схем для простого арифметического выражения на языке C:

```
// демонстрация чувствительности результата вычисления к
последовательности
// арифметических операций
#include <stdio.h>

int main() {
    float num1 = 0.23456789;
    float num2 = 1.5678e+20f;
    float num3 = 1.2345e+10f;

    float result1 = (num1 * num2) / num3;
    float result2 = (num1 / num3) * num2;

    double result3 = (double)num1 * (double)num2 / (double)num3;
    double result4 = ((double)num1 / (double)num3) * (double)num2;

    printf("(%.f * %.f) / %.f = %.f\n", num1, num2, num3, result1);

    printf("(%.f / %.f) * %.f = %.f\n", num1, num3, num2, result2);

    printf("%.f * %.f / %.f = %.lf\n", num1, num2, num3, result3);

    printf("(%.f / %.f) * %.f = %.lf\n", num1, num3, num2, result4);

    return 0;
}
```

```
(0.234568 * 156779996323277438976.000000) / 12344999936.000000 = 2978983680.000000
(0.234568 / 12344999936.000000) * 156779996323277438976.000000 = 2978983936.000000
0.234568 * 156779996323277438976.000000 / 12344999936.000000 = 2978983717.267449
(0.234568 / 12344999936.000000) * 156779996323277438976.000000 = 2978983717.267448
```

2) Запустить и проинтерпретировать результаты работы разных вычислительных схем для итерационного и неитерационного вычисления.

```
// демонстрация накопления погрешности для итерационного процесса
// версия для одинарной точности
#include <stdio.h>
#include <math.h>
#include <float.h>

int main() {
    float numbers[] = {1.0f, 20.0f, 300.0f, 4000.0f, 5e6f,
        FLT_MIN, FLT_MAX * 0.99f};
    // вектор с числами одинарной точности

    int iterations = 10;
    int size = sizeof(numbers) / sizeof(numbers[0]);

    for (int iter = 0; iter < size; iter++) {
        float number = numbers[iter];
        float result = number;

        for (int it = 0; it < iterations; it++)
            result = sqrtf(result);

        // послед. извлечение квадратного корня
        for (int it = 0; it < iterations; it++)
            result = result * result;

        // послед. возведение числа в квадрат
        float error = fabsf(number - result);
        float relative_error = (error * 100.0f) / number;

        printf("Исх-е значение: %e, результат: %e, "
            "абс-ая погрешность: %e, отн-ая погрешность: %e (%%)\n",
            number, result, error, relative_error);
    }

    return 0;
}
```

```
> ./a.out
```

```
Исх-е значение: 1.000000e+00, результат: 1.000000e+00, абс-ая погрешность: 0.000000e+00, отн-ая погрешность: 0.000000e+00 (%)  
Исх-е значение: 2.000000e+01, результат: 2.000009e+01, абс-ая погрешность: 8.964539e-05, отн-ая погрешность: 4.482269e-04 (%)  
Исх-е значение: 3.000000e+02, результат: 3.000142e+02, абс-ая погрешность: 1.422119e-02, отн-ая погрешность: 4.740397e-03 (%)  
Исх-е значение: 4.000000e+03, результат: 4.000106e+03, абс-ая погрешность: 1.064453e-01, отн-ая погрешность: 2.661133e-03 (%)  
Исх-е значение: 5.000000e+06, результат: 4.999486e+06, абс-ая погрешность: 5.135000e+02, отн-ая погрешность: 1.027000e-02 (%)  
Исх-е значение: 1.175494e-38, результат: 1.175480e-38, абс-ая погрешность: 1.429324e-43, отн-ая погрешность: 1.215935e-03 (%)  
Исх-е значение: 3.368795e+38, результат: 3.368697e+38, абс-ая погрешность: 9.796404e+33, отн-ая погрешность: 2.907984e-03 (%)
```

```
// замена итерации функцией  
// версия для одинарной точности с powf  
#include <stdio.h>  
#include <math.h>  
#include <float.h>  
  
int main() {  
    float numbers[] = {1.0f, 20.0f, 300.0f, 4000.0f, 5e6f,  
        FLT_MIN, FLT_MAX * 0.99f};  
  
    int iterations = 10;  
    int size = sizeof(numbers) / sizeof(numbers[0]);  
  
    for (int iter = 0; iter < size; iter++) {  
        float number = numbers[iter];  
  
        // Извлекаем корень  
        float intermediate = powf(number, 1.0f / (1 << iterations));  
  
        // Восстанавливаем значение  
        float result = powf(intermediate, (1 << iterations));  
  
        float error = fabsf(number - result);  
  
        float relative_error = (error * 100.0f) / number;  
  
        printf("Исх-е значение: %e, результат: %e, абс-ая погрешность: %e, "  
            "отн-ая погрешность: %e (%)\\n",  
            number, result, error, relative_error);  
    }  
  
    return 0;  
}
```

```

> ./a.out
Исх-е значение: 1.000000e+00, результат: 1.000000e+00, абс-ая погрешность: 0.000000e+00, отн-ая погрешность: 0.000000e+00 (%)
Исх-е значение: 2.000000e+01, результат: 2.000007e+01, абс-ая погрешность: 6.866455e-05, отн-ая погрешность: 3.433228e-04 (%)
Исх-е значение: 3.000000e+02, результат: 3.000087e+02, абс-ая погрешность: 8.728027e-03, отн-ая погрешность: 2.909343e-03 (%)
Исх-е значение: 4.000000e+03, результат: 4.000114e+03, абс-ая погрешность: 1.142578e-01, отн-ая погрешность: 2.856445e-03 (%)
Исх-е значение: 5.000000e+06, результат: 5.000186e+06, абс-ая погрешность: 1.860000e+02, отн-ая погрешность: 3.720000e-03 (%)
Исх-е значение: 1.175494e-38, результат: 1.175497e-38, абс-ая погрешность: 2.662467e-44, отн-ая погрешность: 2.264977e-04 (%)
Исх-е значение: 3.368795e+38, результат: 3.368755e+38, абс-ая погрешность: 4.015917e+33, отн-ая погрешность: 1.192093e-03 (%)

```

3) С помощью программы на языке С вывести на экран двоичное представление машинных чисел одинарной точности стандарта IEEE 754 для записи: числа π , бесконечности, нечисла (NaN), наименьшего положительного числа, наибольшего положительного числа, наименьшего отрицательного числа. Сформулировать обоснование полученных результатов в пунктах 1 и 2, опираясь на двоичное представление машинных чисел.

```

#include <stdio.h>
#include <math.h>
#include <float.h>
#include <stdint.h>

void float_to_binary_string(float num, char *buffer) {
    union {
        float f;
        uint32_t u;
    } converter;

    converter.f = num;

    for (int iter = 31; iter >= 0; iter--)
        buffer[31 - iter] = (converter.u & (1U << iter)) ? '1' : '0';

    buffer[32] = '\0';
}

int main() {

    float pi = M_PI;
    float infinity = INFINITY;
    float nan_value = NAN;
    float smallest_positive = FLT_MIN;
    float largest_positive = FLT_MAX;

```

```

float smallest_negative = -FLT_MIN;
char binary_str[33];

float_to_binary_string(pi, binary_str);
printf("\u03C0: %s\n", binary_str);

float_to_binary_string(infinity, binary_str);
printf("Бесконечность: %s\n", binary_str);

float_to_binary_string(nan_value, binary_str);
printf("NaN: %s\n", binary_str);

float_to_binary_string(smallest_positive, binary_str);
printf("Самое маленькое положительное: %s\n", binary_str);

float_to_binary_string(largest_positive, binary_str);
printf("Самое большое положительное: %s\n", binary_str);

float_to_binary_string(smallest_negative, binary_str);
printf("Наименьшее отрицательное: %s\n", binary_str);

return 0;
}

```

```

> gcc calc_math_lab_0.c -lm
> ./a.out
п: 01000000010010010000011111011011
Бесконечность: 01111111000000000000000000000000
NaN: 0111111110000000000000000000000000
Самое маленькое положительное: 00000000100000000000000000000000
Самое большое положительное: 01111110111111111111111111111111
Наименьшее отрицательное: 10000000100000000000000000000000

```

Индивидуальные задания Вариант 9

from math import

log res = log(a * b / c)

from math import

log res = log(a) + log(b) – log(c)

```

#include <stdio.h>
#include <math.h>

int main() {
    double a = 1007e-13;
    double b = 103e10;
    double c = 1.00731;

    double direct_res = log(a * b / c);

    double improved_res = log(a) + log(b) - log(c);

    printf("Прямой метод: log(a * b / c) = %e\n", direct_res);
    printf("Улучшенный метод: log(a) + log(b) - log(c) = %e\n",
improved_res);

    double difference = fabs(direct_res - improved_res);
    printf("Разница между методами: %e\n", difference);

    return 0;
}

```

```

Прямой метод: log(a * b / c) = 4.634421e+00
Улучшенный метод: log(a) + log(b) - log(c) = 4.634421e+00
Разница между методами: 8.881784e-16

```

Вывод: Изучили особенности организации вычислительных процессов, связанные с погрешностями, приближенным характером вычислений на компьютерах современного типа, вычислительной устойчивостью.