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: ст. группы ПВ-233

Ситников Алексей Павлович

Проверил:

Горбов Даниил Игоревич

Белгород 2025 г.

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 С:

```c
#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;
}
```

```
C:\Users\admin\CLionProjects\Project17\cmake-build-debug\Project17.exe
(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

Process finished with exit code 0
```

Возьмём первые два уравнения, они эквивалентны, но имеют разные ответы, это связано с тем, что тип float имеет точность 6-7 знаков, а числа num2 и num3 очень большие. В первом случае теряется точность из-за num1 * num2, а во втором случае num1 / num3 уменьшает значение, но затем все равно теряется точность из-за того же * num2. В случае с double точность около 15 знаков поэтому ответы почти совпали.

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

// демонстрация накопления погрешности для итерационного процесса
// версия для одинарной точности

```c
#include <stdio.h>
#include <math.h>
#include <float.h>
#include <windows.h>

int main() {
    SetConsoleOutputCP(CP_UTF8);
    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;
}
```

```
C:\Users\admin\CLionProjects\Project17\cmake-build-debug\Project17.exe
Исх-е значение: 1.000000e+00, результат: 1.000000e+00, абс-ая погрешность: 0.000000e+00, отн-ая погрешность: 0.000000e+0
0 (%)
Исх-е значение: 2.000000e+01, результат: 2.000009e+01, абс-ая погрешность: 8.964539e-05, отн-ая погрешность: 4.482269e-0
4 (%)
Исх-е значение: 3.000000e+02, результат: 3.000142e+02, абс-ая погрешность: 1.422119e-02, отн-ая погрешность: 4.740397e-0
3 (%)
Исх-е значение: 4.000000e+03, результат: 4.000106e+03, абс-ая погрешность: 1.064453e-01, отн-ая погрешность: 2.661133e-0
3 (%)
Исх-е значение: 5.000000e+06, результат: 4.999486e+06, абс-ая погрешность: 5.135000e+02, отн-ая погрешность: 1.027000e-0
2 (%)
Исх-е значение: 1.175494e-38, результат: 1.175480e-38, абс-ая погрешность: 1.429324e-43, отн-ая погрешность: 1.215935e-0
3 (%)
Исх-е значение: 3.368795e+38, результат: 3.368697e+38, абс-ая погрешность: 9.796404e+33, отн-ая погрешность: 2.907984e-0
3 (%)

Process finished with exit code 0
```

Исходя из выводов программы понятно что чем больше знаков в числе тем меньше точность, но если взять крайнее значение FLT_MIN то погрешность будет очень маленькая, а FLT_MAX * 0.99f наоборот погрешность будет очень большая.

// замена итерации функцией
// версия для одинарной точности с powf

```
#include <stdio.h>
#include <math.h>
#include <float.h>
#include <windows.h>

int main() {
    SetConsoleOutputCP(CP_UTF8);
    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;
}
```

```
C:\Users\admin\CLionProjects\Project17\cmake-build-debug\Project17.exe
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: 3.995635e+33,отн-ая погрешность: 1.186072e-03
 (%)

Process finished with exit code 0
```

Картина практически не изменилась после поправки в функции возведения в степень и под корень.

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>
#include <windows.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() {
```

```c
    SetConsoleOutputCP(CP_UTF8);
    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;
}
```

```
C:\Users\admin\CLionProjects\Project17\cmake-build-debug\Project17.exe
π: 01000000010010010000111111011011
Бесконечность: 01111111100000000000000000000000
NaN: 01111111110000000000000000000000
Самое маленькое положительное: 00000000100000000000000000000000
Самое большое положительное: 01111111011111111111111111111111
Наименьшее отрицательное: 10000000100000000000000000000000

Process finished with exit code 0
```

Потеря точности в пункте 1 результаты вычислений result1 и result2 отличаются из-за разного порядка операций. Это связано с тем, что промежуточные результаты округляются до ближайшего числа, представимого в формате float.

В пункте 2 рассматриваются итерационные вычисления, такие как последовательное извлечение квадратного корня и возведение в квадрат. Эти операции также зависят от точности представления чисел и накопления ошибок округления.

Индивидуальное задание, вариант 13

13.	res = (a/b) * (c/d) * (e/f)	res = (a * c * e) / (b * d * f)

```c
#include <stdio.h>
int main() {
    float a = 0.234569;
    float b = 1.5678e+4f;
    float c = 1.2345e+1f;
    float d = 1.2345678;
    float e = 1.3278e+11f;
    float f = 1.22345e+3f;

    float result1 = (a/b) * (c/d) * (e/f);
    float result2 = (a*c*e) / (b*d*f);

    printf("(%f/%f) * (%f/%f) * (%f/%f) = %f\n", a, b, c, d,e,f,result1);
    printf("(%f*%f*%f) / (%f*%f*%f) = %f\n", a, c, e, b, d, f, result2);


    return 0;
}
```

Вывод на си:

```
C:\Users\admin\CLionProjects\Project17\cmake-build-debug\Project17.exe
(0.234569/15678.000000) * (12.345000/1.234568) * (132779999232.000000/1223.449951) = 16236.879883
(0.234569*12.345000*132779999232.000000) / (15678.000000*1.234568*1223.449951) = 16236.880859

Process finished with exit code 0
```

```python
a = 0.234569
b = 1.5678e+4
c = 1.2345e+1
d = 1.2345678
e = 1.3278e+11
f = 1.22345e+3

result1 = (a / b) * (c / d) * (e / f)
result2 = (a * c * e) / (b * d * f)

print(f"({a}/{b}) * ({c}/{d}) * ({e}/{f}) = {result1}")
print(f"({a}*{c}*{e}) / ({b}*{d}*{f}) = {result2}")
```

Вывод на интерпретируемом языке python:

```
C:\Users\admin\PycharmProjects\pythonProject2\.venv\Scripts\python.exe C:\Users\admin\PycharmProjects\pythonProject\main.py
(0.234569/15678.0) * (12.345/1.2345678) * (132780000000.0/1223.45) = 16236.878619071544
(0.234569*12.345*132780000000.0) / (15678.0*1.2345678*1223.45) = 16236.878619071545

Process finished with exit code 0
```

Число, полученное на Python, будем считать точным, так как Python использует числа с плавающей точкой с двойной точностью (тип float), что позволяет добиться высокой степени точности в большинстве вычислений.

В таком случае посчитаем погрешность:

Абсолютная погрешность для первого уравнения:
$16236.878619071545 – 16236.878619071544 \approx 1.264 * 10^{-3}$

Относительная погрешность для первого уравнения получается:
$\frac{1.264 * 10^{3}}{16236.878619071544} * 100 \approx 7.78*10^{-6}$

Абсолютная погрешность для второго уравнения:
$16236.880859 – 16236.878619071545 \approx 2.24*10^{-3}$

Относительная погрешность для второго уравнения получается:
$\frac{2.24*10^{-3}}{16236.878619071544} * 100 \approx 1.37 * 10^{-5}$

По результатам можно делать такой вывод: формулы res = (a/b) * (c/d) * (e/f) и res = (a * c * e) / (b * d * f) эквивалентны, но первый вариант посчитал точнее, скорее всего это случилось из-за порядка чисел, в первом варианте он более сбалансированный что даёт лучшую точность.

Вывод: в ходе проделанной работы я изучил особенности организации вычислительных процессов, связанные с погрешностями, приближенным характером вычислений на компьютерах современного типа, вычислительной устойчивостью. Я понял как хранение чисел с плавающей точкой влияет на ответ, узнал точность типа float 6–7 знаков, и double +-15 знаков. Также определил, что операции между числами у которых порядок различается на большое количество знаков, плохо влияют на точность ответа, для улучшения точности стоит постепенно увеличивать или снижать порядок чисел в операции.