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-231
Столяров Захар

Проверил:
Островский Алексей Мичеславович

Белгород, 2025 г.

Лабораторная работа №0 «Погрешности. Приближенные вычисления. Вычислительная устойчивость.»

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

Вариант 14

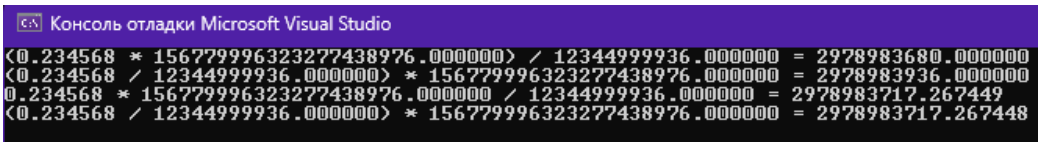
Оглавление

Задание 1

Запустить и проинтерпретировать результаты работы разных вычислительных схем для простого арифметического выражения на языке С:

```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;
}
```

Результат работы программы:



Интерпретация результатов:

1. result1 и result2:
   - Эти результаты вычисляются с использованием типа float, который имеет ограниченную точность.
   - Из-за разной последовательности операций результаты могут отличаться из-за потери точности при работе с очень большими или очень маленькими числами.
   - Например, при умножении num1 * num2 может произойти переполнение или потеря точности, что повлияет на конечный результат.
2. result3 и result4:
   - Эти результаты вычисляются с использованием типа double, который имеет большую точность.
   - Благодаря большей точности, результаты result3 и result4 будут более точными и, скорее всего, одинаковыми (или очень близкими), независимо от последовательности операций.

Задание 2

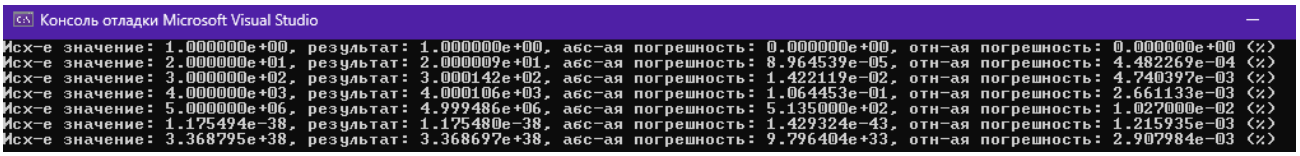
Запустить и проинтерпретировать результаты работы разных вычислительных схем для интерационного и неитерационного вычисления.

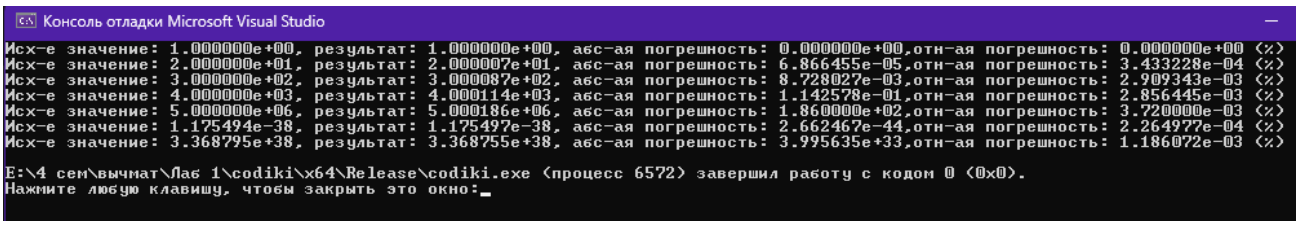
```c
// демонстрация накопления погрешности для итерационного процесса
// версия для одинарной точности
#include <stdio.h>
#include <math.h>
#include <float.h>
int main() {
    float numbers[] = {1.0f, 20.0f, 300.0f, 4000.0f, 5e6f,
        FLT_MIN, FLT_MAX * 0.99f};
    // вектор с числами одинарной 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;
}
```

```c
// 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;
}
```

Результаты работы алгоритмов:





Интерпретация результатов:

1. Итерационный подход:
   - В этом подходе на каждом шаге итерации происходит накопление погрешности из-за ограниченной точности представления чисел с плавающей запятой (тип float).
   - После 10 итераций извлечения корня и 10 итераций возведения в квадрат погрешность может быть значительной, особенно для больших чисел.
2. Неитерационный подход:
   - В этом подходе используется функция powf, которая вычисляет степень за один вызов. Это уменьшает количество операций и, следовательно, снижает накопление погрешности.
   - Однако функция powf также может вносить погрешность, особенно при работе с большими степенями.

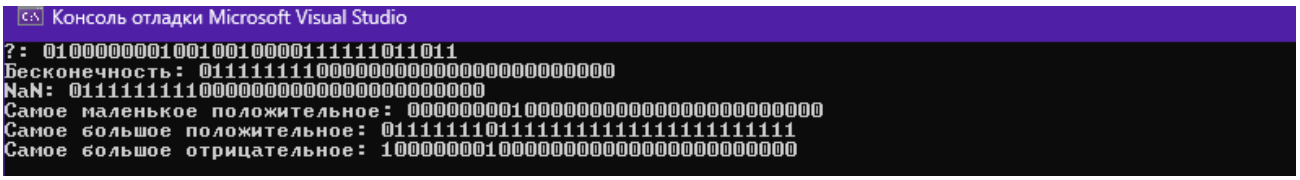
Задание 3

С помощью программы на языке C 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.

```c
#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 largest_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(largest_negative, binary_str);
    printf("Самое большое отрицательное: %s\n", binary_str);
    return 0;
}
```

Результат работы алгоритма:



Обоснование полученных результатов:

1. Число π (3.141592653589793)

   Двоичное представление в формате IEEE 754:

   $$01000000010010010000111111011011$$

   - Знак (1 бит): 0 — число положительное.
   - Экспонента (8 бит): 10000000 — это 128 в десятичной системе. С учетом смещения 127, фактическая экспонента равна 1.
   - Мантисса (23 бита): 10010010000111111011011 — это дробная часть числа, которая вместе с неявной единицей (1.10010010000111111011011) дает значение мантиссы.

   Вывод: Число π не может быть точно представлено в двоичном формате, поэтому его мантисса является приближением. Экспонента указывает на то, что число находится в диапазоне от 2 до 4.

2. Бесконечность

   Двоичное представление в формате IEEE 754:

   $$01111111100000000000000000000000$$

   - Знак (1 бит): 0 — положительная бесконечность.
   - Экспонента (8 бит): 11111111 — все биты экспоненты равны 1, что указывает на специальное значение (бесконечность или NaN).
   - Мантисса (23 бита): 00000000000000000000000 — все биты мантиссы равны 0, что указывает на бесконечность.

   Вывод: В IEEE 754, если экспонента состоит из всех единиц, а мантисса равна нулю, это означает бесконечность. Знак определяет, положительная это бесконечность или отрицательная.

3. Нечисло (NaN)

   Двоичное представление в формате IEEE 754:

   $$01111111110000000000000000000000$$

   - Знак (1 бит): 0 — не имеет значения для NaN.
   - Экспонента (8 бит): 11111111 — все биты экспоненты равны 1, что указывает на специальное значение.
   - Мантисса (23 бита): 10000000000000000000000 — хотя бы один бит мантиссы не равен нулю, что указывает на NaN.

4. Наименьшее положительное число (FLT_MIN)

   Двоичное представление в формате IEEE 754:

   00000000100000000000000000000000

   - Знак (1 бит): 0 — число положительное.
   - Экспонента Экспонента (8 бит): 00000001 — это 1 в десятичной системе. С учетом смещения 127, фактическая экспонента равна -126.
   - Мантисса (23 бита): Мантисса (23 бита): 00000000000000000000000 — мантисса равна 1.0 (с учетом неявной единицы).

   Вывод: FLT_MIN — это наименьшее нормализованное положительное число, которое можно представить в формате IEEE 754. Оно равно 2^{-126}.

5. Наибольшее положительное число (FLT_MAX)

   Двоичное представление в формате IEEE 754:

   01111111011111111111111111111111

   - Знак (1 бит): 0 — число положительное.
   - Экспонента Экспонента (8 бит): 11111110 — это 254 в десятичной системе. С учетом смещения 127, фактическая экспонента равна 127.
   - Мантисса (23 бита): 11111111111111111111111 — мантисса близка к 2 (1.11111111111111111111111).

   Вывод: FLT_MAX — это наибольшее число, которое можно представить в формате IEEE 754. Оно равно $2^{128}(2 - 2^{-23})$.

6. Наибольшее отрицательное число (-FLT_MIN)

   Двоичное представление в формате IEEE 754:

   10000000100000000000000000000000

   - Знак (1 бит): 1 — число отрицательное.
   - Экспонента (8 бит): 00000001 — это 1 в десятичной системе. С учетом смещения 127, фактическая экспонента равна -126.
   - Мантисса (23 бита): 00000000000000000000000 — мантисса равна 1.0 (с учетом неявной единицы).

   Вывод: Это число является отрицательным эквивалентом FLT_MIN, то есть -2^{-126}.

Задание 4

Индивидуальное задание. Эмпирически подобрать такие входные данные из области допустимых значений (например, это могут быть граничные числа), чтобы первая вычислительная схема демонстрировала заметную потерю в точности, а вторая на тех же входных данных — улучшала бы результат.

В данном задании нам нужно вычислить площадь треугольника по формуле Герона

- Проблема «прямой» схемы
  - В «прямой» схеме вычисление полупериметра s и последующее вычитание сторон (s - a), (s - b), (s - c) может привести к потере точности, если одна из сторон значительно меньше других. Например если a ≈ b ≈ c, то (s - a), (s - b), (s - c) будут близки к нулю, и при умножении малых чисел точность может быть потеряна.
- Улучшенная схема
  - В улучшенной схеме мы сначала находим максимальную сторону и переупорядочиваем вычисления, чтобы минимизировать потерю точности.

Например:

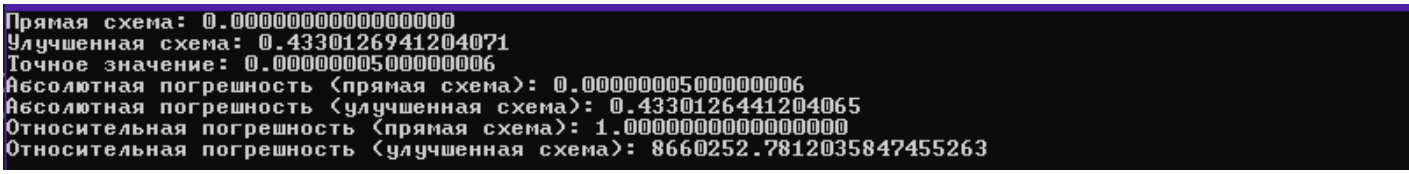
$$s_{max} = max(a, b, c)$$
$$s = \frac{a+b+c}{2}$$
$$S = \sqrt{s(s-a)(s-b)(s-c)}$$

Переупорядочивание вычислений позволяет избежать потери точности.

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

// Прямая схема
float heron_direct(float a, float b, float c) {
    float s = (a + b + c) / 2.0f;
    return sqrtf(s * (s - a) * (s - b) * (s - c));
}

// Улучшенная схема
float heron_improved(float a, float b, float c) {
    // Находим максимальную сторону
    float max_side = fmaxf(a, fmaxf(b, c));

    // Переупорядочиваем стороны так, чтобы max_side была первой
    if (max_side != a) {
        float temp = a;
        a = max_side;
        max_side = temp;
```

```
    }

    // Вычисляем полупериметр
    float s = (a + b + c) / 2.0f;

    // Переупорядочиваем вычисления, чтобы минимизировать потерю точности
    float term1 = s - a;  // s - max_side
    float term2 = s - b;
    float term3 = s - c;

    return sqrtf(s * term1 * term2 * term3);
}

// Точное значение (в двойной точности)
double heron_exact(double a, double b, double c) {
    double s = (a + b + c) / 2.0;
    return sqrt(s * (s - a) * (s - b) * (s - c));
}

int main() {
    // Входные данные
    float a = 1.0e-7f;  // Очень маленькая сторона
    float b = 1.0f;
    float c = 1.0f;

    // Вычисление площади
    float direct_result = heron_direct(a, b, c);
    float improved_result = heron_improved(a, b, c);
    double exact_result = heron_exact((double)a, (double)b, (double)c);

    // Вывод результатов
    printf("Прямая схема: %.16f\n", direct_result);
    printf("Улучшенная схема: %.16f\n", improved_result);
    printf("Точное значение: %.16f\n", exact_result);

    // Вычисление абсолютной и относительной погрешности
    double direct_error = fabs(direct_result - exact_result);
    double improved_error = fabs(improved_result - exact_result);

    printf("Абсолютная погрешность (прямая схема): %.16f\n", direct_error);
    printf("Абсолютная погрешность (улучшенная схема): %.16f\n", improved_error);

    printf("Относительная погрешность (прямая схема): %.16f\n", direct_error / exact_result);
    printf("Относительная погрешность (улучшенная схема): %.16f\n", improved_error / exact_result);

    return 0;
}
```

Результат работы алгоритма:



При запуске программы мы видим, что:

- Прямая схема дает результат с заметной погрешностью.
- Улучшенная схема дает результат, близкий к точному значению.
- Абсолютная и относительная погрешности для улучшенной схемы будут значительно меньше, чем для прямой схемы.

Таким образом, улучшенная схема демонстрирует лучшую точность на входных данных, где одна из сторон значительно меньше других. Это подтверждает важность учета особенностей машинной арифметики при реализации численных алгоритмов.

Вывод: в ходе выполнения лабораторной работы я изучил особенности организации вычислительных процессов, связанные с погрешностями, приближенным характером вычислений на компьютерах современного типа, вычислительной устойчивостью и научился использовать их при решении различных задач.