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 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;  
}

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

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

#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;  
}

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

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 c 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;  
}

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

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

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.

#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() {  
 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;  
}

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

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**

****

#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:

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

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:

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

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 1.264 \*

Относительная погрешность для первого уравнения получается:

\* 100 7.78\*

Абсолютная погрешность для второго уравнения:

16236.880859–16236.878619071545 2.24\*

Относительная погрешность для второго уравнения получается:

\* 100 1.37 \*

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.