Федеральное государственное автономное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э. Баумана»
(национальный исследовательский университет)
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

О т ч е т

по домашнему заданию № 1

Название домашнего задания: Программирование с использованием разветвленных и циклических процессов

Дисциплина: Алгоритмизация и программирование

Студент гр. ИУ6-13Б _____ _____
(Подпись, дата) (И.О. Фамилия)

Преподаватель _____ _____
(Подпись, дата) (И.О. Фамилия)

Москва, 2025

Часть 1. Вычисления. Погрешности вычислений

Задача 1

Цель работы: изучение и оценка точности представления чисел.

Выполнение: текст программы на рисунке 1.

```cpp
#include <iostream>
#include <cmath>

using namespace std;


int main() {
    float y;
    cout << "До инициализации y = " << y << endl;

    y = 1;
    cout << "После инициализации y = " << y << endl;

    y = y / 6000;
    y = exp(y);
    y = sqrt(y);
    y = y / 14;

    y = 14 * y;
    y = y * y;
    y = log(y);
    y = 6000 * y;

    cout << "После преобразований y = " << y << endl;

    return 0;
}
```

Рисунок 1 — Текст программы

Примерные результаты при запуске программы:

До инициализации y = 4.59163e-41 После инициализации y = 1

После преобразований y = 0.999844

Абсолютная погрешность: Δ = |1 — 0.999844| = 0.000156 Относительная погрешность: δ = Δ / |A| = 0.000156 / 1 = 0.000156

В ходе выполнения лабораторной работы основными факторами, влияющими на точность вычислений, стали:

•Погрешности округления. Обусловлены использованием ограниченного количества разрядов для представления чисел с плавающей точкой. Данные погрешности возникают при выполнении каждой арифметической операции и имеют свойство накапливаться в процессе вычислений.

• Погрешности математических операций. Связаны с приближённым характером вычисления элементарных функций, таких как exp, sqrt и log, реализованных в стандартной библиотеке C++. Даже оптимизированные алгоритмы их расчёта вносят некоторую ошибку.

Вывод: в рамках лабораторной работы была разработана и протестирована программа на C++, предназначенная для оценки погрешностей представления чисел и арифметических операций с типом float. Результаты показали существенную величину абсолютной и относительной погрешностей, что подчеркивает важность учёта подобных погрешностей при выполнении точных вычислений.

Задача 2

Цель работы: разработка программы для вычисления значений гиперболических функций и оценки погрешности вычислений при использовании различных типов данных (float, double, long double).

Выполнение: текст программы на рисунке 2.

```cpp
#include <iostream>
#include <cmath>
#include <iomanip>

using namespace std;

int main() {
    float x;
    cout << "Enter x: ";
    cin >> x;

    float y1 = (exp(x) - exp(-x)) / 2;
    float y2 = (exp(x) + exp(-x)) / 2;
    float y = pow(y2, 2) - pow(y1, 2);

    cout << setprecision(18);
    cout << "x: " << x << endl;
    cout << "y1 (sh(x)) = " << setw(22) << y1 << endl;
    cout << "y2 (ch(x)) = " << setw(22) << y2 << endl;
    cout << "y (y2^2 - y1^2) = " << setw(22) << y << endl;
    cout << "Абсолютная погрешность: " << setw(22) << fabs(1 - y) << endl;
    cout << "Относительная погрешность: " << setw(22) << fabs(1 - y) / 1 << endl;
}
```

Рисунок 2 — Текст программы

Результаты при использовании float представлены в таблице 1.

Таблица 1 — Результаты при использовании float

x	y1	y2	y	Δ	δ
5	74.2032089233398438	74.2099533081054688	1.00095546245574951	0.000955462455749511719	0.000955462455749511719
10	11013.232421875	11013.232421875	0	1	1
15	1634508.625	1634508.625	0	1	1
20	242582592	242582592	0	1	1
25	36002451456	36002451456	0	1	1

Результаты при использовании double представлены в таблице 2.

Таблица 2 — Результаты при использовании double

x	y1	y2	y	Δ	δ
5	74.2032105777887523	74.2099485247878476	1.00000000000181899	1.81898940354585648e-12	1.81898940354585648e-12
10	11013.2328747033935	11013.2329201033244	1.00000002980232239	2.98023223876953125e-08	2.98023223876953125e-08
15	1634508.68623590237	1634508.6862362083	1	0	0
20	242582597.704895139	242582597.704895139	0	1	1
25	36002449668.6929398	36002449668.6929398	0	1	1

Результаты при использовании long double представлены в таблице 3.

Таблица 3 — Результаты при использовании long double

x	y1	y2	y	Δ	δ
5	74.203210577788759	74.2099485247878444	0.999999999999999112	8.88178419700125232e-16	8.88178419700125232e-16
10	11013.2328747033934	11013.2329201033231	1	0	0
15	1634508.68623590237	1634508.68623620827	1.0000002384185791	2.384185791015625e-07	2.384185791015625e-07
20	242582597.704895138	242582597.70489514	1.00390625	0.00390625	0.00390625
25	36002449668.6929363	36002449668.6929363	0	1	1

Вывод: проведённые вычисления демонстрируют, что выбор типа данных существенно влияет на точность результатов. Переход от float к типам double и long double позволяет значительно снизить погрешность, особенно при работе с большими значениями аргумента. Это подтверждает целесообразность их применения в задачах, требующих высокой вычислительной точности и минимизации накопления ошибок округления.

Ответ на вопрос: наибольшее влияние на точность оказывают тип переменной x (исходные данные) и типы переменных y1, y2 (промежуточные вычисления). Использование double или long double для этих переменных позволяет сохранить точность за счет большего количесвтва значащих цифр и уменьшения ошибок округления в критических операциях с экспонентами.

Задача 3

Цель работы: проверка основного тригонометрического тождества с применением численных методов.

Выполнение: для минимизации погрешности вычислений был использован тип данных long double, обеспечивающий наибольшую точность. Исходный код программы представлен на рисунке 3.

```cpp
#include <iostream>
#include <cmath>
#include <iomanip>

using namespace std;

int main() {
    long double x;
    cout << "Введите значение x (в радианах): ";
    cin >> x;

    long double sin2x = pow(sin(x), 2);
    long double cos2x = pow(cos(x), 2);
    long double result = sin2x + cos2x;

    cout << fixed << setprecision(16);
    cout << "x: " << x << endl;
    cout << "sin^2(x) = " << setw(20) << sin2x << endl;
    cout << "cos^2(x) = " << setw(20) << cos2x << endl;
    cout << "sin^2(x) + cos^2(x) = " << setw(20) << result << endl;

    // Оценка погрешности
    long double delta = fabs(result - 1);
    cout << "Погрешность = " << setw(20) << delta << endl;
}
```

Рисунок 3 — Текст программы

Результаты работы в таблице 4 (т. к. A = 1, абсолютная погрешность и относительная совпадают)

Таблица 4 — Результаты выполнения программы

x	$\sin^2(x)$	$\cos^2(x)$	$\sin^2(x)+\cos^2(x)$	Δ δ
0	0.0000000000000000	1.0000000000000000	1.0000000000000000	0.0000000000000000
1.04	0.7437410511671797	0.2562589488328203	1.0000000000000000	0.0000000000000000
1.57	0.9999993658637698	0.0000006341362302	1.0000000000000000	0.0000000000000000
3.14	0.0000025365433124	0.9999974634566876	1.0000000000000000	0.0000000000000000

Вывод: в рамках лабораторной работы была создана и протестирована программа для проверки тождества $\sin^2(x) + \cos^2(x) = 1$. Результаты тестирования показали высокую точность вычислений — полученные значения практически идеально соответствуют теоретическому ожиданию, что свидетельствует о надежности встроенных математических функций языка C++. Проведенная работа также наглядно демонстрирует важность учета и оценки вычислительной погрешности при выполнении расчетов.

Часть 2. Программирование разветвляющегося вычислительного процесса

Цель работы: разработать программу для вычисления значения функции **f(x)**, определённой по частям согласно формуле, приведённой на рисунке 4. Функция задаётся тремя разными выражениями в зависимости от значения **x**.

$$f(x) = \begin{cases} 0, & \text{при} \quad x < 0; \\ (\sin x + \cos x)^2, & \text{при} \quad 0 \le x < 1{,}5; \\ \sin x - \sqrt{x + \cos(\pi x^2)}, & \text{при} \quad x \ge 1{,}5 \end{cases}$$

Рисунок 4 — Значение функции f(x), определенной по частям

Задание: написать функцию, которая по входному значению **x** рассчитывает значение функции **f(x)** согласно указанным в формуле случаям из фото и проверить её работу на различных значениях **x**.

Проект программы: проект программы изображен на рисунке 5.

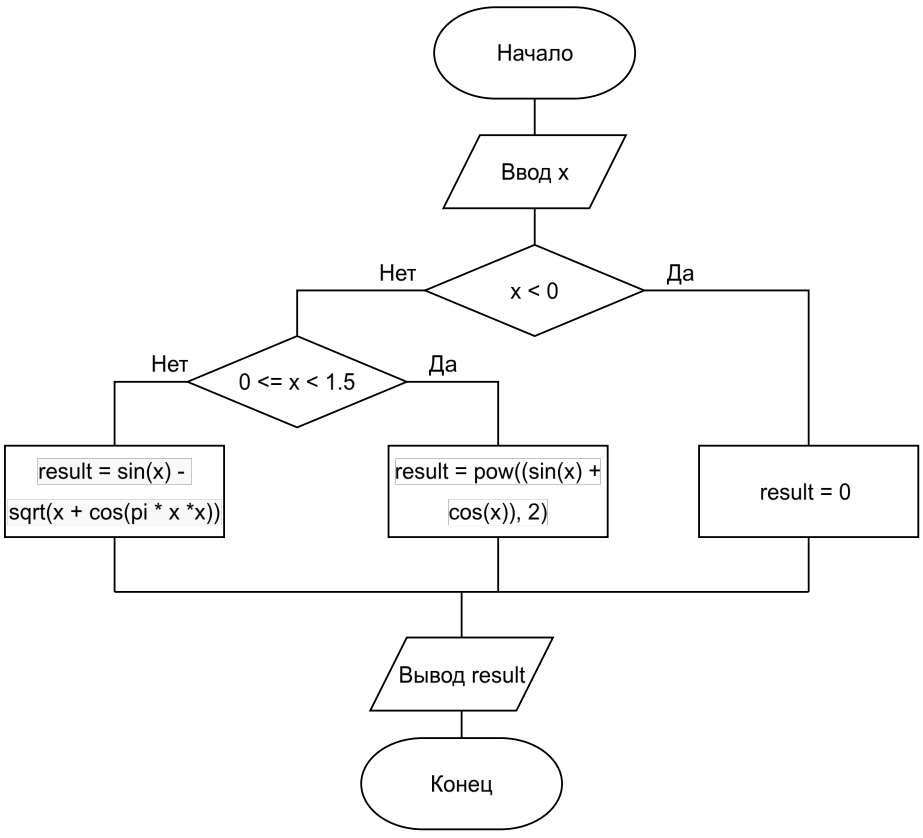

Рисунок 5 — Блок-схема алгоритма

Текст программы: текст программы изображен на рисунке 6.

```cpp
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    double x, result;
    const double pi = 3.141592653589793;

    cout << "Введите x: " << endl;
    cin >> x;

    if (x<0) result = 0;
    else if (0 <= x < 1.5) result = pow((sin(x) +cos(x)), 2);
    else result = sin(x) - sqrt(x + cos(pi * x *x));

    cout << "Значение функции= " << result << endl;
    return 0;
}
```

Рисунок 6 — Текст программы

Тестовые данные и результаты тестирования: тестовые данные и результаты тестирования представлены в таблице 5.

Таблица 5 — Тестовые данные и результат выполнения

x	Результат
-1	0
0	1
1.4	1.33499
1.5	1.14112
2	0.243198

Вывод: в результате выполнения лабораторной работы была разработана программа для вычисления значения функции, определённой по частям согласно заданной формуле. Программа корректно обрабатывает все случаи в зависимости от входного значения **x**, включая ветви для **x<0**, **0≤x<1.5** и **x≥1.5**. Тестирование показало, что программа работает правильно и выдает ожидаемые результаты для различных входных данных, что подтверждает корректность реализованного алгоритма.

Часть 3. Программирование циклического процесса.

Цель работы: решить задачу с заданной точностью ξ, организовав итерационный цикл. Значение точности вводится с клавиатуры. Найти первый член последовательности $\mathbf{y=(n+10)/n^3}$, для которого выполняется условие $\mathbf{y \le \xi}$. Провести проверку программы при $\mathbf{\xi = 10^{-2}, 10^{-4}}$. Определить, как изменяется количество итераций при изменении точности вычислений.

Задание: Организовать итерационный цикл для вычисления членов последовательности $y=(n+10)/n^3$ с увеличением n. Найти и вывести первый член последовательности, для которого значение y становится меньше либо равно заданной точности ξ. Выполнить тестирование программы при $\xi = \mathbf{10^{-2}}$ и $\mathbf{10^{-4}}$. Провести анализ и сделать выводы о зависимости количества итераций от точности расчета.

Выполнение: схема алгоритма изображена на рисунке 7, текст программы изображен на рисунке 8.

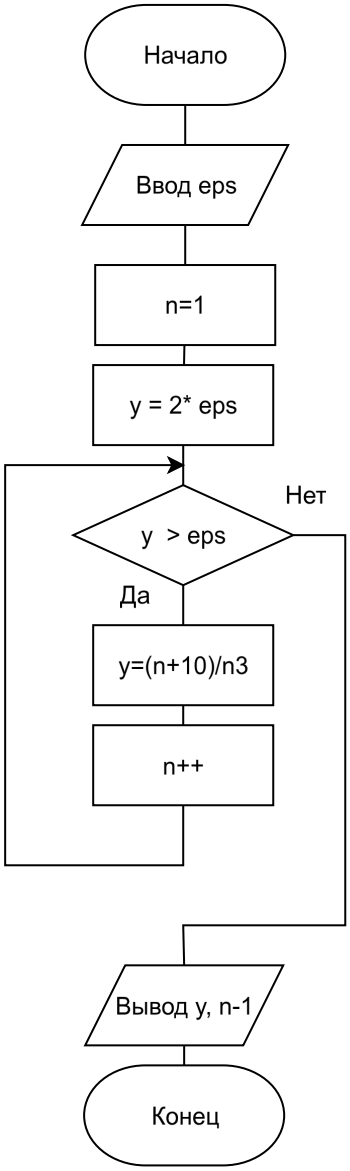

Рисунок 7 — Схема алгоритма

```cpp
#include <cmath>
#include <iostream>
using namespace std;

int main()
{
    long double eps, y;
    int n = 1;

    cout << "Введите эпсилон" << endl;
    cin >> eps;

    y = 2*eps;

    while (y > eps)
    {
        y = (n + 10) / (double)(n*n*n); // без double происходило бы целочисле
        n++;
    }

    cout << "первый член последовательности которогый <= eps: " << y << endl;
    cout << "Количество итераций: " << n - 1 << endl;
}
```

Рисунок 8 — Текст программы

Тестовые данные и результат выполнения: в таблице 6.

Таблица 6 - Тестовые данные и результат выполнения

eps	y	Кол-во итераций
1,00E-01	0.0740741	6
1,00E-03	0.00098594	36

Вывод: программа реализует итерационный алгоритм для нахождения первого члена последовательности **y=n3n+10**, который становится меньше или равен заданной точности **ϵ**. Результаты показывают, что при уменьшении значения **ϵ** значительно увеличивается количество итераций для достижения заданной точности, что подтверждает зависимость вычислительной нагрузки от точности. Это свидетельствует о корректной работе программы и правильной реализации алгоритма итерационного поиска.