Лабораторная работа 2.

Операторы цикла

Часть 0.

Задача 1. Цикл for: «Сумма кратных 3 — почему результат “съезжает”?»

Что нужно сделать

1. Скопируйте и запустите «битый» код без правок.
2. Прогоните контрольные входы (ниже) и зафиксируйте вывод.
3. Поставьте «зонды» (временные печати) и объясните, где и почему ломается.
4. Исправьте все проблемы и подтвердите тестами.
5. (Бонус) Упростите цикл до шага +3, без if

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

// Задача: для целых A и B вывести сумму и количество чисел кратных 3 на [A, B].
int main() {
    int A, B;
    cin >> A >> B;

    int sum = 0;
    int cnt = 0;

    // BUGS: неверная граница; нет скобок у if; лишний i++; постинкремент в присваивании
    for (int i = A; i < B; i++)           // должно охватывать B тоже
        if (i % 3 == 0)
            sum += i;                     // из-за отсутствия {} только ЭТА строка в if
            cnt = cnt++;                  // cnt не увеличивается из-за присваивания
            if (i % 2 == 0) i++;          // меняем i внутри цикла -> пропуски значений

    cout << "sum=" << sum << " cnt=" << cnt << "\n";
    return 0;
}
```

Контрольные входы и ожидаемая логика

- A=3, B=9 → члены: 3, 6, 9 → sum=18, cnt=3
- A=-2, B=4 → члены: 0, 3 → sum=3, cnt=2
- A=7, B=8 → нет членов → sum=0, cnt=0

Диагностика (что проверить зондами)

Добавьте перед cout:

```
// пример «зондов»
/*
for (int i = A; i < B; i++) {
    bool mult3 = (i % 3 == 0);
    cout << "i=" << i << " mult3=" << mult3
         << " sum=" << sum << " cnt=" << cnt << "\n";
    ...
}
*/
```

Объясните:

- почему из-за отсутствия {} только одна строка относится к if, а cnt=cnt++; выполняется **всегда**;
- почему cnt = cnt++; не меняет cnt (пост-инкремент возвращает старое значение, затем присваивание его же затирает);
- как i++ внутри тела «перепрыгивает» числа;
- почему i < B исключает B.

Задача 2. Цикл while: «Последовательность до 0 — почему зависает/теряет числа?»

Цель: понять типичный паттерн ввода в while: «прайминг» чтением/чтение в условии, работа с continue, защита от деления на ноль, корректное завершение по «сторожу» (sentinel).

Задача: читаем вещественные числа до ввода 0. Считаем:

- avg_pos — среднее положительных;
- neg — количество отрицательных. Требуется вывести avg_pos (или NA, если положительных нет) и neg.

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    double x;                // BUG: не инициализировано
    int neg = 0, pos = 0;
    double sum = 0.0;

    cout << fixed << setprecision(3);

    while (x != 0) {         // BUG: проверка до чтения -> поведение неопределено
        if (x > 0) {
            sum += x;
            pos++;
            continue;        // BUG: из-за continue следующее чтение пропускается
        }
        if (x < 0) neg++;

        cin >> x;            // BUG: читаем только тут; при continue ввода не будет
    }

    cout << "avg_pos=" << (sum / pos) << " neg=" << neg << "\n"; // BUG: деление на 0
    return 0;
}
```

Контрольные сценарии и ожидаемый вывод

1. Ввод: 5 4 -2 0 → avg_pos=4.500 neg=1
2. Ввод: -1 -2 0 → avg_pos=NA neg=2
3. Ввод: 0 → avg_pos=NA neg=0

Шаги диагностики

1. Добавьте «зонд» перед телом цикла: печатайте текущее x. Объясните, почему цикл стартует с мусорным значением.
2. Покажите, как continue приводит к пропуску следующего ввода и зависанию.
3. Обнаружьте деление на ноль при отсутствии положительных.

Исправление: два канонических паттерна

Паттерн A: «прайминг» (предварительное чтение)

```
double x;
int neg = 0, pos = 0;
double sum = 0.0;

if (!(cin >> x)) return 0;          // вход закончился внезапно
while (x != 0) {
    if (x > 0) { sum += x; pos++; }
    else if (x < 0) { neg++; }

    if (!(cin >> x)) break;         // читаем СЛЕДУЮЩЕЕ значение в конце итерации
}

cout << fixed << setprecision(3);
if (pos == 0) cout << "avg_pos=NA ";
else          cout << "avg_pos=" << (sum / pos) << " ";
cout << "neg=" << neg << "\n";
```

Паттерн B: чтение прямо в условии

```
double x;
int neg = 0, pos = 0;
double sum = 0.0;

while (cin >> x && x != 0) {        // читаем и проверяем сторожа в ОДНОМ месте
    if (x > 0) { sum += x; pos++; }
    else if (x < 0) { neg++; }
}

cout << fixed << setprecision(3);
cout << (pos ? "avg_pos=" + to_string(sum / pos) : string("avg_pos=NA"));
cout << " neg=" << neg << "\n";
```

Обсудите, почему во втором паттерне continue не ломает ввод (чтение — в условии), а в первом — оно безопасно, потому что чтение находится в **одном месте** внизу цикла.

Задача 3. i++ vs ++i: почему печатается «через один», а сумма не сходится?

Цель: на практике понять:

- постфиксный инкремент i++ возвращает старое значение, затем увеличивает i;
- префиксный ++i сначала увеличивает i, затем возвращает новое значение;
- инкремент в **условии цикла** тоже меняет i (даже если тело не выполнялось)

Что сделать

1. Скомпилируйте и запустите код ниже **без правок**.
2. Сравните фактический вывод с ожидаемым для контрольных входов.
3. Добавьте «зонды» (временные cout) и объясните, где именно срабатывает постфикс/префикс.
4. Исправьте код (см. подсказки и эталон) так, чтобы печатался ряд 1..N, а сумма совпала с формулой N*(N+1)/2.
5. Сравните две корректные версии: «префиксная» и «постфиксная»

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

// Задача: распечатать 1..N и посчитать их сумму.
int main() {
    int N;
    cin >> N;

    int i = 0;
    long long sum = 0;

    // ОЖИДАЛОСЬ: 1 2 3 ... N
    // ФАКТИЧЕСКИ: печатаются не все числа + неверная сумма
    while (i++ < N) {           // постфикс меняет i уже в условии
        cout << i;              // здесь i уже увеличен
        if (i < N) cout << ' '; // попытка без "висячего" пробела

        sum += i++;             // постфикс: к сумме идёт СТАРОЕ i, потом i ещё раз растёт
    }
    cout << "\n";

    cout << "sum=" << sum << "\n";
    cout << "i_final=" << i << "\n";  // обратите внимание на конечное значение i
    return 0;
}
```

Контрольные входы и что вы увидите

- N = 5

  Ожидание: 1 2 3 4 5, sum=15.

  Факт: печатается 1 3 5 (идём «через один»), sum не равен 15, i_final = 6.

- N = 0

  Ожидание: пустая строка, sum=0, i_final=0.

  Факт: цикл **не выполнялся**, но i всё равно стал 1 (из-за i++ в условии).

Шаги диагностики (добавьте «зонды»)

Временно распечатайте состояние в начале итерации:

```
/*
int before = i;
bool cond = (i++ < N);
cerr << "[check] before=" << before << "  cond=" << cond << "  afterCond(i)=" << i << "\n";
if (!cond) break;
cerr << "  body: i=" << i << "\n";
sum += i++;
*/
```

Отметьте:

- значение i **до** и **после** проверки условия;
- что суммирование использует **старое** значение i (постфикс), а затем ещё увеличивает счётчик.

Эталонные исправления

Вариант A (префиксный, «чистый»)

Одно место инкремента — в начале тела; условие не трогает i.

```
int i = 0;
long long sum = 0;
while (i < N) {
    ++i;                // сначала увеличили до 1..N
    cout << i << (i < N ? ' ' : '\n');
    sum += i;           // складываем текущее i
}
cout << "sum=" << sum << "\n";
cout << "i_final=" << i << "\n";     // ровно N
```

Вариант B (постфиксный, аккуратный)

Если очень хочется держать инкремент в условии — **уберите** любые инкременты внутри тела.

```
int i = 0;
long long sum = 0;
while (i++ < N) {        // сравнили старое i (0..N-1), затем i стало 1..N
    cout << i << (i < N ? ' ' : '\n');  // печатаем 1..N
    sum += i;            // НИКАКИХ i++ здесь!
}
cout << "sum=" << sum << "\n";
cout << "i_final=" << i << "\n";     // будет N+1 — это ожидаемо для такой схемы
```

В Варианте B финальное i на 1 больше N — это нормально, т.к. постфикс в условии увеличил счётчик на «лишний» шаг уже на проверке выхода.

Сравнение i++ и ++i в выражениях (крошечный пример)

```
int a = 0;
int b = a++;   // b = 0; a = 1
int c = ++a;   // a = 2; c = 2
```

i++ возвращает старое значение, затем увеличивает i.

++i сначала увеличивает, затем возвращает новое.

Часть 1.

Выполните следующие задачи по ссылке:

https://informatics.msk.ru/mod/statements/view.php?id=86831#1

Часть 2.

Задача 1.

Ввод: диалогово: N (1..1000), затем cols (1..8), w (3..6).

Вывод: искомые числа из диапазона 1..N, выведенные по строкам в cols колонок, каждая ячейка шириной w, **ровно выровнены по правому краю**, без лишних пробелов в конце строки.

Требования: использовать циклы; не «ломать» последнюю строку; не выводить пустые колонки.

Варианты:

1. Простые числа.
2. Взаимно простые с N: gcd(i, N) = 1.
3. Ровно 4 делителя.
4. Полупростые (semiprime): i = p*q, где p, q — простые (допускаются p=q).
5. Палиндромы в десятичной записи.
6. Все цифры различны.
7. Сумма цифр чётная.
8. Сумма цифр — простое число.
9. Числа Харшада: i делится на сумму своих цифр.
10. Нечётное число единиц в двоичной записи.
11. Ровно две единицы в двоичной записи.
12. Автоморфные: последние разряды i^2 совпадают с i.
13. Треугольные числа: i = k(k+1)/2 для некоторого k.
14. Числа Фибоначчи ≤ N.
15. Совершенные квадраты или кубы (проверка r*r==i или c*c*c==i).
16. Первая и последняя цифра совпадают.
17. Чередование чётной/нечётной цифры во всей записи числа.
18. gcd(i, S(i)) = 1, где S(i) — сумма цифр числа.
19. Ровно 3 различных делителя.
20. Совершенные числа: i равно сумме своих собственных делителей.

Задача 2.

Ввод: A B H (H>0). Проход по сетке x = A, A+H, … пока x не превышает B на шаг (аккуратно с накопленной погрешностью).

Вывод: таблица из двух столбцов x и f(x) в формате fixed, setprecision(3). В конце выполните дополнительную операцию варианта. При недопустимом аргументе (sqrt от отрицательного, ln ≤ 0) — строка x пропускается и увеличивается счётчик ошибок; выведите этот счётчик в конце.

Варианты:

1. f(x) = x*x - 3x + 2. Дополнительно: вывести min f, max f.
2. f(x) = sin(x)/x, считать f(0)=1. Дополнительно: посчитать число смен знака f.
3. f(x) = |x| + ln(x+2) (домен: x>-2). Дополнительно: сколько строк пропущено из-за домена.
4. f(x) = sqrt(5 - x) (домен: x ≤ 5). Дополнительно: среднее значение f.
5. f(x) = e^(-x) * cos(x). Дополнительно: x в котором модуль f максимален.
6. f(x) = ln(1 + x*x) (домен: всегда). Дополнительно: трапецеидурная оценка интеграла по сетке.

7. f(x) = x / (1 + x*x). Дополнительно: количество x, где |f(x)| < 0.1.
8. f(x) = sqrt(|x-1|) + ln(x+3) (домен: x>-3). Дополнительно: медиана по значениям f на сетке*
9. (упрощение: можно собрать сумму и счётчик, медиану — не требуем, если не храните; вместо медианы выведите min/max).
10. f(x) = tanh(x) = (e^x - e^{-x})/(e^x + e^{-x}). Дополнительно: первая точка, где f(x) > 0.9.
11. f(x) = sqrt(4 - x*x) (полукруг; домен: |x| ≤ 2). Дополнительно: площадь по формуле трапеций.
12. f(x) = ln(x) (домен: x>0). Дополнительно: количество x, где дробная часть f < 0.5.
13. f(x) = cosh(x) - 1. Дополнительно: посчитать max при x∈[A,B].
14. f(x) = 1 / (1 + x*x). Дополнительно: число точек, где f(x) ≥ 0.5.
15. f(x) = x^3 - 2x. Дополнительно: количество локальных экстремумов по сетке (сравнивая соседей).
16. f(x) = ln(x+1) / (x+1) (домен: x>-1). Дополнительно: сумма положительных значений f.
17. f(x) = |sin(3x)|. Дополнительно: доля точек, где f(x) > 0.8.
18. f(x) = x * e^{-x}. Дополнительно: x максимума (по сетке).
19. f(x) = atan(x). Дополнительно: число попаданий f в интервал (0.5, 1.0).
20. f(x) = ln(2 - x) (домен: x<2). Дополнительно: вывести min f.
21. f(x) = sqrt(x) + sqrt(3 - x) (домен: 0 ≤ x ≤ 3). Дополнительно: max f.

Задача 3. Сумма ряда (печать шага, слагаемого и накопленной суммы)

Общие правила:
Варианты указывают ряд, правило слагаемого t_k и условие остановки: либо по числу членов n, либо по точности eps (0<eps<1).
Печатать таблицу: k, t_k, S_k (накопленная сумма). В конце — min |t_k|, max |t_k|.
Все вычисления — циклами; факториалы и степени считать итеративно.

Варианты:

1. e^x = Σ_{k=0..n} x^k / k! (ввод x, n).
2. sin x = Σ_{k=0..n} (-1)^k x^{2k+1} / (2k+1)! (ввод x, n).
3. cos x = Σ_{k=0..n} (-1)^k x^{2k} / (2k)! (ввод x, n).
4. ln(1+x) = Σ_{k=1..n} (-1)^{k+1} x^k / k, |x|<1 (ввод x, n).
5. arctan x = Σ_{k=0..n} (-1)^k x^{2k+1} / (2k+1), |x|≤1 (ввод x, n).
6. sinh x = Σ_{k=0..n} x^{2k+1} / (2k+1)! (ввод x, n).
7. cosh x = Σ_{k=0..n} x^{2k} / (2k)! (ввод x, n).
8. Геометрический: S = Σ_{k=0..n} q^k = (1 - q^{n+1})/(1 - q) при q≠1 (ввод q, n). Печатать также теоретическую сумму для сравнения.
9. Альт. гармонический: S = Σ_{k=1..n} (-1)^{k+1} / k (ввод n).

10. $\ln((1+x)/(1-x)) = 2\,\Sigma_{k=0..n}\, x^{2k+1}/(2k+1)$, $|x|<1$ (ввод x, n).
11. $\ln x = 2\,\Sigma_{k=0..n}\, (y^{2k+1})/(2k+1)$, где $y=(x-1)/(x+1)$, $x>0$ (ввод x, n).
12. $e^{-x} = \Sigma_{k=0..n}\, (-1)^k\, x^k / k!$ (ввод x, n).
13. $1/(1-x) = \Sigma_{k=0..n}\, x^k$, $|x|<1$ (ввод x, n).
14. $\sqrt{(1+x)} = 1 + \Sigma_{k=1..n}\, C(1/2, k)\, x^k$, $|x|<1$ (ввод x, n), где $C(\alpha,k) = \alpha(\alpha-1)\ldots(\alpha-k+1)/k!$.
15. $\arcsin x = \Sigma_{k=0..n}\, ((2k)! / (4^k\, (k!)^2\, (2k+1)))\, x^{2k+1}$, $|x|\le 1$ (ввод x, n).
16. $\pi/4 = \Sigma_{k=0..n}\, (-1)^k / (2k+1)$ (ввод n).
17. $s = \Sigma_{k=1..n}\, 1/(k(k+1))$ (ввод n) — простой телескопический ряд; печатать также «закрытую» сумму $1 - 1/(n+1)$.
18. $s = \Sigma_{k=1..n}\, 1/k!$ (ввод n) — печатать оценку $s \approx e - 1$.
19. Сумма степенного ряда до точности: e^x, складывать, пока $|t_k| \ge eps$ (ввод x, eps).
20. sin x с остановом по точности: добавлять члены, пока $|t_k| \ge eps$ (ввод x, eps).

Задача 4. Поток чисел без массивов (K элементов)

Ввод: K (количество), затем K значений (целые или вещественные — указано в варианте).
Вывод: только то, что требует вариант. **Нельзя** сохранять весь поток — только текущие агрегаты/флаги.

Варианты:

1. Количество **смен знака** соседних элементов.
2. Длина **максимальной серии равных** подряд.
3. Количество **локальных максимумов** (строго больше соседей).
4. Индекс первого элемента, большего среднего всех предыдущих; если нет — -1.
5. Сумма элементов на **чётных позициях** и на **нечётных**; вывести большее из них.
6. Количество **строго возрастающих пар** ($a_i < a_{i+1}$).
7. Максимальная по модулю величина и её индекс (первое вхождение).
8. Количество элементов, у которых **дробная часть < 0.5** (вещественный поток).
9. Сумма всех **неотрицательных**, произведение всех **отрицательных** (если отрицательных нет — вывести NA).
10. Проверка на **чередование знаков** по всему потоку (да/нет).

11. Максимальная длина **монотонного** (неубывающего) фрагмента подряд.
12. Количество элементов, у которых **младшая десятичная цифра** равна 7.
13. Сумма **простых** чисел в потоке.
14. Количество **совпадений** a_i == a_{i+1}.
15. Сумма элементов, попавших в интервал (L, R) (ввести L,R).
16. Количество **квадратов целых** (…, 1, 4, 9, …) в потоке.
17. Количество индексов i, где a_i * a_{i+1} < 0 и |a_i| > |a_{i+1}|.
18. Первая позиция, где a_i = 0; иначе -1.
19. Количество элементов, у которых сумма цифр — **кратна 3** (целый поток).
20. Количество **чётных** перед первым **отрицательным** (если отрицательных нет — считать все).

Задание 5. Вложенные циклы

Ввод: параметры узора (обычно n, иногда m); символы рамки/заполнителя при необходимости.
Вывод: рисунок/таблица в консоль. Все фигуры строить **циклами**, без «готовых строк повторов».

Варианты:

1. Таблица умножения n×n с заголовками и выравниванием по правому краю.
2. «Шахматная доска» n×n символами #/. (верхний левый — #).
3. Полая рамка n×m с диагоналями \ и /.
4. Правый прямоугольный треугольник из * высоты n.
5. Равнобедренный треугольник из * высоты n (центрировать по ширине).
6. Ромб из * с диагоналями (нечётная сторона 2k+1).
7. «Лесенка» из чисел: в строке i печатать 1..i.
8. Таблица степеней: столбцы — k=1..m, строки — i^k для i=1..n.
9. Таблица остатков: в ячейке (i,j) — i % j для i,j=1..n.
10. Таблица булевой функции: AND/OR/XOR для всех пар битов 0/1.

11. «Песочные часы» из * в квадрате n×n.
12. Полосатый прямоугольник n×m: чередовать строки из # и ..
13. Лестница с пропусками: строка i содержит числа i, i+2, i+4, … до n.
14. Треугольник Паскаля (первые n строк), выравнивание по ширине.
15. Таблица делителей: в строке i вывести все делители i (через пробел).
16. «Крест» в квадрате n×n: вертикаль и горизонталь через центр.
17. Таблица min(i,j) для i,j=1..n.
18. Градиент символов: от 'A' до 'A'+m-1 по колонкам, повторять по строкам n.
19. Полый прямоугольник с толщиной рамки t (≥1).
20. «Змейка» чисел 1..n*m по прямоугольнику n×m (чётные строки печатать справа-налево).