День 1 — Первый запуск: «Привет, C!» на WSL + Ubuntu

Цель дня

К концу дня вы:

- Откроете **WSL → Ubuntu 22.04** и установите нужные инструменты.
- Создадите **первый файл с кодом** и запустите программу.
- Научитесь вызывать компилятор gcc и редактор vim.

---

1) Открываем Ubuntu (WSL)

1. В Windows нажмите **Win**, введите **Ubuntu**, откройте **Ubuntu 22.04**.
2. Появится терминал с приглашением вида:

```
yourname@PC:~$
```

2. Знак $ означает, что можно печатать команду.

> Подсказка: в терминале **не печатайте сам символ $**, набирайте только команды после него.

---

2) Ставим инструменты

В терминале выполните по очереди:

```
sudo apt update
sudo apt install -y build-essential gdb make vim
```

Проверим:

```
gcc --version
vim --version | head -n 1
```

---

3) Создаём папку для занятий

```
mkdir -p ~/c-course/day01
cd ~/c-course/day01
pwd
```

---

4) Пишем самую первую программу

Открываем файл в vim

```
vim main.c
```

Как печатать в vim (минимум)

- Нажмите i — режим **ввода** (-- INSERT -- внизу).
- Вставьте код ниже целиком.
- Нажмите Esc, затем :wq и Enter (сохранить и выйти).

Код «Привет, C!»

```
#include <stdio.h>   // подключаем библиотеку для printf

int main(void) {     // точка входа в программу
    printf("Hello, C!\n");  // \n — перевод строки
    return 0;        // 0 = всё хорошо
}
```

---

5) Собираем (компилируем) и запускаем

```
gcc main.c -o hello
./hello
```

Ожидаемый вывод:

```
Hello, C!
```

Если ./hello не запускается, проверьте, что вы в ~/c-course/day01 и файл существует:

```
ls -l
```

---

6) Меняем текст и запускаем снова

```
vim main.c
```

- Нажмите i, измените текст в кавычках (например, своё имя).
- Esc :wq — сохранить и выйти.
- Пересоберите и запустите:

```
gcc main.c -o hello
./hello
```

---

7) Пояснение простыми словами

- **main.c** — текст с инструкциями для компьютера.
- **gcc** переводит этот текст в программу (hello).
- **./hello** — команда «запусти программу из текущей папки».

---

8) Примеры (

printf

)

Замените код в main.c на это и посмотрите вывод:

```
#include <stdio.h>

int main(void) {
    printf("Одна строка\n");
    printf("Другая строка\n");
    printf("Число: %d\n", 42);                // %d — целое
    printf("Два числа: %d и %d\n", 3, 7);
    printf("Текст и кавычки: \"%s\"\n", "C — это просто");
```

```
    return 0;
}
```

Сборка и запуск:

```
gcc main.c -o hello
./hello
```

---

9) Практика (по возрастанию)

A. Разогрев

1. Измените сообщение на своё имя и любимый фильм (на двух строках).
2. Добавьте printf, который печатает ваше любимое число.

B. Экранирование символов

- Напечатайте строку с кавычками внутри — используйте ".
- Напечатайте табуляцию между словами — используйте \t:

```
printf("Имя:\tИван\n");
```

C. Мини-«визитка»

Напечатайте три строки:

```
Имя: ...
Город: ...
Хобби: ...
```

---

10) Мини-экзамен Дня 1

Задание: программа **card**, которая печатает ровно:

```
=== CARD ===
Name: <ваше имя>
City: <ваш город>
```

Требования:

- Файл: main.c.
- Сборка и запуск:

```
gcc main.c -o card
./card
```

Подсказка: используйте три printf.

---

11) Частые вопросы

- **Как выйти из vim, если запутался?** Esc :q! Enter — выйти без сохранения.
- **Почему не запускается hello просто по имени?** В Linux текущая папка не ищется. Используйте ./hello.
- **Где хранить файлы?** Внутри Ubuntu (например, ~/c-course/...), а не на C:\ — так надёжнее и быстрее.

День 2 — Переменные, числа и простая математика

Цель дня

К концу дня вы:

- Поймёте, что такое **переменная** и как ей присваивать значение.
- Научитесь печатать числа разного вида (целые и с точкой).
- Научитесь **вводить числа с клавиатуры** с помощью scanf.
- Напишете простые программы с вычислениями.

---

1) Готовим папку

```
mkdir -p ~/c-course/day02
cd ~/c-course/day02
```

---

2) Что такое переменная (очень просто)

Переменная — это «коробочка» в памяти у компьютера. У коробочки есть **имя** и **тип** (что в ней лежит).

Создайте файл vars.c:

```
#include <stdio.h>

int main(void) {
    int age;        // коробочка для целого числа
    age = 25;       // положили 25

    double pi;      // коробочка для числа с точкой
    pi = 3.14;

    char letter;    // один символ
    letter = 'A';

    printf("age = %d\n", age);        // %d — целое
    printf("pi = %f\n", pi);          // %f — число с точкой
    printf("letter = %c\n", letter);  // %c — символ
    return 0;
}
```

Сборка и запуск:

```
gcc vars.c -o vars
./vars
```

---

3) Печать чисел аккуратно

Создайте print_numbers.c:

```
#include <stdio.h>

int main(void) {
    int a = 10;
    double b = 3.14159;

    printf("a = %d\n", a);            // целое
    printf("b = %f\n", b);            // по умолчанию 6 знаков после точки
    printf("b (2 знака) = %.2f\n", b);// ровно 2 знака после точки
```

```
    return 0;
}
```

Сборка и запуск:

```
gcc print_numbers.c -o print_numbers
./print_numbers
```

> Подсказка: %.1f, %.2f, %.3f — управляют количеством знаков после точки.

---

4) Простая математика (целые и вещественные)

Создайте math_demo.c:

```
#include <stdio.h>

int main(void) {
    int x = 8;
    int y = 3;

    printf("Сложение: %d\n", x + y);       // 11
    printf("Вычитание: %d\n", x - y);      // 5
    printf("Умножение: %d\n", x * y);      // 24
    printf("Деление целое: %d\n", x / y);  // 2 (дробная часть
отбрасывается)
    printf("Остаток: %d\n", x % y);        // 2

    double a = 8.0;
    double b = 3.0;
    printf("Деление с дробью: %f\n", a / b); // 2.666667
    return 0;
}
```

Сборка и запуск:

```
gcc math_demo.c -o math_demo
./math_demo
```

---

5) Ввод чисел с клавиатуры:

scanf

Программа задаёт вопрос, вы вводите число и нажимаете Enter.

Создайте sum_two.c:

```
#include <stdio.h>

int main(void) {
    int a, b;

    printf("Введите первое целое число: ");
    scanf("%d", &a);  // &a — адрес коробочки a

    printf("Введите второе целое число: ");
    scanf("%d", &b);

    int s = a + b;
    printf("Сумма: %d\n", s);
    return 0;
}
```

Сборка и запуск:

```
gcc sum_two.c -o sum_two
./sum_two
```

Ввод вещественного числа (double):

```
#include <stdio.h>

int main(void) {
    double t;
    printf("Температура в °C: ");
    scanf("%lf", &t);              // для double — именно %lf
    printf("Введено: %.1f °C\n", t);
    return 0;
}
```

Сборка и запуск:

```
gcc temp.c -o temp
./temp
```

> Важно:

- > В scanf для int используем %d, для double — %lf.
- > Всегда ставьте & перед именем переменной в scanf.

---

6) Мини-примеры для тренировки

6.1 Перевод километров в мили

```
#include <stdio.h>

int main(void) {
    double km;
    printf("Километры: ");
    scanf("%lf", &km);

    double miles = km * 0.621371;
    printf("Мили: %.2f\n", miles);
    return 0;
}
```

6.2 Цельсий → Фаренгейт

```
#include <stdio.h>

int main(void) {
    double c;
    printf("Градусы Цельсия: ");
    scanf("%lf", &c);

    double f = c * 9.0/5.0 + 32.0;
    printf("Фаренгейты: %.1f\n", f);
    return 0;
}
```

7) Практика (по возрастанию)

A. Разогрев

1. Программа спрашивает ваше любимое **целое** число и печатает:

   Ваше число: <...>
2. Программа спрашивает два double и печатает их **среднее** с двумя знаками после точки.

B. Калькуляция прямоугольника

1. Спросите у пользователя **ширину** и **высоту** (целые).
2. Выведите **периметр** P = 2*(w + h) и **площадь** S = w*h.

C. Деление целых

1. Спросите два целых.
2. Выведите **частное** и **остаток** (используйте / и %).
3. Если второе число равно 0 — напечатайте «Деление на 0 запрещено».

D. Форматирование

1. Попросите double и выведите его тремя способами:

   %.0f, %.1f, %.2f.

---

8) Мини-экзамен Дня 2 — «Разделим минуты»

Задание: программа **time_split**.

Пользователь вводит **общее количество минут** (целое).

Программа выводит, сколько это **часов** и **минут**.

Пример:

```
Enter minutes: 130
Hours: 2
Minutes: 10
```

Шаблон (скопируйте в main.c):

```c
#include <stdio.h>

int main(void) {
    int minutes;
    printf("Enter minutes: ");
    scanf("%d", &minutes);

    int hours = minutes / 60;  // целое деление
    int mins  = minutes % 60;  // остаток

    printf("Hours: %d\n", hours);
    printf("Minutes: %d\n", mins);
    return 0;
}
```

Сборка и запуск:

```
gcc main.c -o time_split
./time_split
```

Критерии зачёта:

- Корректно считает для 0, для чисел меньше 60 и больше 60.
- Вывод в точности как в примере.
- Программа компилируется и запускается без ошибок.

---

9) Частые вопросы

- **Зачем & в scanf?** Так программа получает **адрес** переменной — куда положить введённое число.
- **Почему так много знаков после точки?** По умолчанию %f печатает 6 знаков. Используйте %.2f и т.п.
- **Если ввёл буквы вместо числа — программа ведёт себя странно?** Пока мы не проверяем ошибки ввода — это нормально на этом этапе. Позже научимся.

День 3 — Сравнения и ветвления:

if

/

else

Цель дня

К концу дня вы:

- Научитесь сравнивать числа (> < >= <= == !=).
- Поймёте, как работает if / else и «лесенка» else if.
- Сумеете комбинировать условия (&&, ||, !).
- Напишете программы: «какое число больше», «чёт/нечёт», «классификация по диапазонам».
- Сдадите мини-экзамен: **перевод баллов 0..100 в оценку A/B/C/D/F** с проверкой диапазона.

---

1) Готовим папку

```
mkdir -p ~/c-course/day03
cd ~/c-course/day03
```

---

2) Что такое сравнение

В C сравнения — это выражения, которые дают **истину** (не ноль) или **ложь** (ноль):

- Равно: **==**
- Не равно: !=
- Больше / Меньше: > <
- Больше или равно / Меньше или равно: >= <=

> Важно: **= — это присваивание**, а **== — сравнение**. Это частая ошибка!

Пример (скопируйте в cmp_demo.c):

```
#include <stdio.h>

int main(void) {
    int a = 5, b = 7;

    printf("a == b ? %d\n", a == b);   // 0 (ложь)
    printf("a != b ? %d\n", a != b);   // 1 (истина)
```

```
    printf("a <  b ? %d\n", a <  b);   // 1
    printf("a >  b ? %d\n", a >  b);   // 0
    return 0;
}
```

Сборка и запуск:

```
gcc cmp_demo.c -o cmp_demo
./cmp_demo
```

---

3) Простое ветвление

if

/

else

Форма:

```
if (условие) {
    // этот блок выполнится, если условие истинно (не ноль)
} else {
    // этот — если условие ложно (ноль)
}
```

Пример «кто больше» (max2.c):

```
#include <stdio.h>

int main(void) {
    int x, y;
    printf("Введите два целых: ");
    scanf("%d %d", &x, &y);

    if (x > y) {
        printf("Больше: %d\n", x);
    } else if (x < y) {
        printf("Больше: %d\n", y);
    } else {
```

```
        printf("Равны: %d и %d\n", x, y);
    }
    return 0;
}
```

```
gcc max2.c -o max2
./max2
```

> Советы:

- > **Скобки {}** ставьте всегда, даже если в блоке одна строка — так безопаснее и понятнее.

- > Условие в if (...) должно быть **в скобках**.

---

4) Чётное / нечётное: оператор %

Остаток от деления: x % 2 равно 0 для чётных чисел.

```
// even_odd.c
#include <stdio.h>

int main(void) {
    int n;
    printf("Введите целое число: ");
    scanf("%d", &n);

    if (n % 2 == 0) {
        printf("Чётное\n");
    } else {
        printf("Нечётное\n");
    }
    return 0;
}
```

---

5) «Лесенка»

else if

: диапазоны

Пример: классифицируем температуру (temp_class.c):

```
#include <stdio.h>

int main(void) {
    int t;
    printf("Температура (°C): ");
    scanf("%d", &t);

    if (t < 0) {
        printf("Холодно\n");
    } else if (t <= 30) {
        printf("Нормально\n");
    } else {
        printf("Жарко\n");
    }
    return 0;
}
```

Порядок условий имеет значение — проверка идёт сверху вниз, первый «подходящий» блок выполняется.

---

6) Комбинируем условия:

&&

,

||

,

!

- && — «и» (оба условия истинны)

- || — «или» (хотя бы одно истинно)
- ! — «не» (меняет истину на ложь и наоборот)

Примеры (logic_demo.c):

```
#include <stdio.h>

int main(void) {
    int age;
    printf("Возраст: ");
    scanf("%d", &age);

    // В диапазоне [18..60] включительно
    if (age >= 18 && age <= 60) {
        printf("Рабочий возраст\n");
    } else {
        printf("Вне диапазона 18..60\n");
    }

    int x;
    printf("Введите число x: ");
    scanf("%d", &x);

    // Кратно 3 ИЛИ 5
    if ((x % 3 == 0) || (x % 5 == 0)) {
        printf("Кратно 3 или 5\n");
    } else {
        printf("Не кратно 3 и не кратно 5\n");
    }

    // Отрицание
    if (!(x == 0)) {
        printf("x не равен нулю\n");
    }
    return 0;
}
```

---

7) Ещё пара полезных шаблонов

7.1 Проверка ввода на деление

```c
// safe_div.c
#include <stdio.h>

int main(void) {
    int a, b;
    printf("Введите два целых: ");
    scanf("%d %d", &a, &b);

    if (b == 0) {
        printf("Деление на 0 запрещено\n");
    } else {
        printf("Частное: %d\n", a / b);  // целое деление
        printf("Остаток: %d\n", a % b);
    }
    return 0;
}
```

7.2 Мини-меню (без switch, пока рано)

```c
// mini_menu.c
#include <stdio.h>

int main(void) {
    int choice;
    printf("Меню: 1 – привет, 2 – пока\n");
    printf("Ваш выбор: ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Привет!\n");
    } else if (choice == 2) {
        printf("Пока!\n");
    } else {
        printf("Неизвестный пункт\n");
    }
    return 0;
}
```

8) Практика (по возрастанию)

A. Больше/меньше/равно

1. Спросите два целых.
2. Напечатайте «Первое больше», «Второе больше» или «Равны».

B. Диапазон

1. Спросите число x.
2. Напечатайте, попадает ли x в диапазон [10..20] (включительно).

C. Знак числа

1. Спросите число.
2. Напечатайте «Положительное», «Отрицательное» или «Ноль».

D. Скидка

1. Спросите сумму покупки (целое).
2. Если >= 1000 — «Скидка 10%», иначе — «Без скидки».

E. Максимум из трёх

1. Спросите три целых a, b, c.
2. Напечатайте максимальное из них (можно несколькими if).

---

9) Мини-экзамен Дня 3 — «Оценка по баллам»

Задание: программа **grade**.

Пользователь вводит целое число **0..100** (баллы).

Нужно вывести оценку по шкале:

- 90..100 → Grade: A
- 80..89 → Grade: B
- 70..79 → Grade: C
- 60..69 → Grade: D
- 0..59 → Grade: F
- Вне диапазона 0..100 → Invalid score

Шаблон (скопируйте в main.c):

```c
#include <stdio.h>

int main(void) {
    int s;
    printf("Enter score (0..100): ");
    scanf("%d", &s);

    if (s < 0 || s > 100) {
        printf("Invalid score\n");
    } else if (s >= 90) {
        printf("Grade: A\n");
    } else if (s >= 80) {
        printf("Grade: B\n");
    } else if (s >= 70) {
        printf("Grade: C\n");
    } else if (s >= 60) {
        printf("Grade: D\n");
    } else {
        printf("Grade: F\n");
    }
    return 0;
}
```

Сборка и запуск:

```
gcc main.c -o grade
./grade
```

Проверьте случаи: -5, 0, 59, 60, 75, 89, 90, 100, 101.

---

10) Частые ошибки и как их избежать

- **Путаете = и ==:**
  - Неверно: if (x = 0) { ... } — это присваивание (и условие всегда ложно).
  - Верно: if (x == 0) { ... }.
- **Забыли фигурные скобки {}:** Без скобок if применится только к одной ближайшей строке — легко ошибиться. Ставьте {} всегда.
- **Сравнение строк как чисел:** В C строки — это массивы символов; сравнение == между строками не работает как «содержимое равно». До строк мы ещё дойдём (День 7).

- **Остаток % только для целых.** Для double так нельзя.
- **Точное сравнение double через ==** — часто плохая идея (ошибки округления). Пока избегаем этого.

---

11) Маленький бонус: «булево» в C

В C можно подключить «настоящий» тип bool:

```
#include <stdio.h>
#include <stdbool.h>

int main(void) {
    bool ok = true;
    if (ok) {
        printf("OK!\n");
    }
    return 0;
}
```

Сборка:

```
gcc bool_demo.c -o bool_demo
./bool_demo
```

День 4 — Повторение действий: циклы while и for

Цель дня

К концу дня вы:

- Поймёте, зачем нужны **циклы** и как они работают.
- Научитесь писать циклы **while** и **for**, увеличивать счётчик i++.

- Освоите **break** (выйти из цикла) и **continue** (пропустить шаг).
- Сделаете программы: сумма 1..N, таблица умножения, чтение чисел до нуля.
- Сдадите мини-экзамен: игра **«Угадай число»**.

---

1) Готовим папку

```
mkdir -p ~/c-course/day04
cd ~/c-course/day04
```

---

2) Цикл

while

— «пока условие истинно, повторяй»

Шаблон:

```
while (условие) {
    // действия
}
```

Пример: считаем от 1 до 5

```
// while_count.c
#include <stdio.h>

int main(void) {
    int i = 1;
    while (i <= 5) {
        printf("%d\n", i);
        i++;                 // не забудьте шаг!
    }
    return 0;
}
```

Сборка и запуск:

```
gcc while_count.c -o while_count
./while_count
```

---

3) Цикл

for

— «старт; проверка; шаг»

Шаблон:

```
for (старт; условие; шаг) {
    // действия
}
```

Тот же счётчик 1..5:

```
// for_count.c
#include <stdio.h>

int main(void) {
    for (int i = 1; i <= 5; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```

---

4) Сумма чисел 1..N (два способа)

while:

```
// sum_while.c
#include <stdio.h>

int main(void) {
    int N;
    printf("N: ");
    scanf("%d", &N);
```

```
    int i = 1, sum = 0;
    while (i <= N) {
        sum += i;
        i++;
    }
    printf("Sum 1..N = %d\n", sum);
    return 0;
}
```

for:

```
// sum_for.c
#include <stdio.h>

int main(void) {
    int N;
    printf("N: ");
    scanf("%d", &N);

    int sum = 0;
    for (int i = 1; i <= N; i++) {
        sum += i;
    }
    printf("Sum 1..N = %d\n", sum);
    return 0;
}
```

---

5) Таблица умножения

Для одного числа:

```
// mult_table.c
#include <stdio.h>

int main(void) {
    int x;
    printf("Число: ");
    scanf("%d", &x);

    for (int i = 1; i <= 10; i++) {
```

```
        printf("%d x %d = %d\n", x, i, x * i);
    }
    return 0;
}
```

Небольшая таблица 1..5 (вложенные циклы):

```
// mult_table_5x5.c
#include <stdio.h>

int main(void) {
    for (int r = 1; r <= 5; r++) {
        for (int c = 1; c <= 5; c++) {
            printf("%4d", r * c);
        }
        printf("\n");
    }
    return 0;
}
```

---

6) break и continue

- break; — выйти из ближайшего цикла.
- continue; — перейти сразу к следующей итерации.

Останавливаемся, когда сумма превысила 100:

```
// stop_over_100.c
#include <stdio.h>

int main(void) {
    int n, sum = 0;
    while (1) {
        printf("Введите число (0 — выход): ");
        scanf("%d", &n);
```

```
        if (n == 0) break;
        sum += n;
        if (sum > 100) {
            printf("Сумма > 100, стоп.\n");
            break;
        }
    }
    printf("Итоговая сумма: %d\n", sum);
    return 0;
}
```

Пропускаем отрицательные:

```
// skip_negative.c
#include <stdio.h>

int main(void) {
    int n, count = 0, sum = 0;

    for (int i = 1; i <= 5; i++) {
        printf("Введите число %d из 5: ", i);
        scanf("%d", &n);
        if (n < 0) continue;
        sum += n;
        count++;
    }
    printf("Положительных: %d, сумма: %d\n", count, sum);
    return 0;
}
```

---

7) Читаем до нуля: суммирование ряда

```
// sum_until_zero.c
#include <stdio.h>

int main(void) {
    int x, sum = 0;
    printf("Вводите целые (0 — конец):\n");
    while (1) {
        scanf("%d", &x);
```

```
        if (x == 0) break;
        sum += x;
    }
    printf("Сумма = %d\n", sum);
    return 0;
}
```

---

8) Частые ошибки

1. **Бесконечный цикл** — забыли i++.
2. **Лишний/недостающий шаг** — перепутали i < N и i <= N.
3. **Неинициализированная переменная** — всегда задавайте стартовое значение: int sum = 0;.

---

9) Практика (по возрастанию)

A. Сумма чётных 1..N — складывайте только i % 2 == 0.

B. Количество цифр — для числа x ≥ 0 делите на 10 в цикле; для 0 ответ 1.

C. Степени двойки — выведите 10 значений: 1 2 4 8 ....

D. Список покупок — спросите n ≤ 10, затем n цен; выведите сумму.

E. Квадраты — таблица i и i*i для i = 1..10.

---

10) Мини-экзамен Дня 4 — «Угадай число»

Программа **guess**.

Компьютер «задумал» число от 1 до 100 (пусть фиксированное secret = 37).

Пользователь вводит варианты; программа подсказывает:

- Больше! — если нужно больше,
- Меньше! — если нужно меньше,
- Угадали за X попыток — если попали.

Код (скопируйте в main.c):

```c
#include <stdio.h>

int main(void) {
    const int secret = 37;
    int tries = 0;
    int x;

    printf("Угадайте число от 1 до 100\n");
    while (1) {
        printf("Ваш вариант: ");
        scanf("%d", &x);
        tries++;

        if (x < secret) {
            printf("Больше!\n");
        } else if (x > secret) {
            printf("Меньше!\n");
        } else {
            printf("Угадали за %d попыток\n", tries);
            break;
        }
    }
    return 0;
}
```

Сборка и запуск:

```
gcc main.c -o guess
./guess
```

Критерии зачёта:

- Корректные подсказки «Больше/Меньше».
- Счётчик попыток.
- Завершение при угадывании.

(По желанию: проверяйте, что ввод в диапазоне 1..100.)

---

11) Бонус: цикл

do ... while

Сначала выполняет тело, потом проверяет условие:

```
// do_while_demo.c
#include <stdio.h>

int main(void) {
    int n = 0;
    do {
        printf("n = %d\n", n);
        n++;
    } while (n < 3);
    return 0;
}
```

День 5 — Функции: разбиваем задачи на маленькие шаги

Цель дня

К концу дня вы:

- Поймёте, что такое **функция** и зачем она нужна.
- Научитесь объявлять, вызывать и возвращать значения из функций.
- Напишете несколько маленьких функций (сумма, максимум, среднее и т.п.).
- Сдадите мини-экзамен: **калькулятор a <op> b**, где операции реализованы функциями.

---

1) Готовим папку

```
mkdir -p ~/c-course/day05
cd ~/c-course/day05
```

---

2) Что такое функция (очень просто)

Функция — это «мини-программа» с **именем**, которая:

- может **получить входы** (параметры),
- может **вернуть результат** (через return),
- помогает **не повторять код** и делает программу понятнее.

Шаблон:

```
тип_результата имя(типы_параметров) {
    // тело
    return значение;   // если тип результата не void
}
```

---

3) Самая простая функция: сумма двух чисел

Создайте add_demo.c:

```
#include <stdio.h>

int add(int a, int b) {   // принимает два int, возвращает int
    return a + b;         // результат кладём в return
}

int main(void) {
    int x = 6, y = 7;
    int s = add(x, y);    // вызываем функцию по имени
    printf("sum = %d\n", s);
    return 0;
}
```

Сборка и запуск:

```
gcc add_demo.c -o add_demo
./add_demo
```

---

4) Функции, которые ничего не возвращают: void

Иногда нужно просто «что-то сделать», например распечатать линию.

```c
// void_demo.c
#include <stdio.h>

void print_line(void) {     // нет параметров и нет возвращаемого значения
    printf("------------\n");
}

int main(void) {
    print_line();
    printf("Hello\n");
    print_line();
    return 0;
}
```

---

5) Параметры и тип результата: аккуратно с типами

Максимум из двух

```c
// max2.c
#include <stdio.h>

int max2(int a, int b) {
    if (a > b) return a;
    else       return b;
}

int main(void) {
    int x, y;
    printf("Введите два целых: ");
    scanf("%d %d", &x, &y);
    printf("max = %d\n", max2(x, y));
    return 0;
}
```

Среднее двух чисел (с точкой!)

Важно: если делить два int, получится **целое**. Для среднего лучше double.

```c
// avg2.c
#include <stdio.h>
```

```
double avg2(double a, double b) {
    return (a + b) / 2.0;     // 2.0 делает деление вещественным
}

int main(void) {
    double x, y;
    printf("Введите два числа: ");
    scanf("%lf %lf", &x, &y);       // для double — %lf
    printf("avg = %.2f\n", avg2(x, y));
    return 0;
}
```

---

6) Порядок функций и «объявления» (прототипы)

Компилятор должен **знать** про функцию до её первого использования.

Варианты:

1. Сначала написать функцию, потом вызывать (как мы делали выше).
2. Либо дать **объявление** (прототип) сверху, а реализацию — ниже.

```
// proto_demo.c
#include <stdio.h>

int square(int x);      // ← объявление (прототип): имя и типы, с точкой с
запятой

int main(void) {
    printf("%d\n", square(5));  // можно вызывать: компилятор знает
сигнатуру
    return 0;
}

int square(int x) {     // реализация позже
    return x * x;
}
```

> На этом этапе **всё в одном файле**. Про отдельные .h/.c поговорим позже (День 12).

---

7) Ещё несколько полезных функций

```
// more_funcs.c
#include <stdio.h>

int min2(int a, int b) {
    return (a < b) ? a : b;     // тернарный оператор: (условие) ? если_да
: если_нет
}

int abs_int(int x) {
    if (x < 0) return -x;
    return x;
}

int is_even(int x) {
    return (x % 2 == 0);        // вернёт 1 или 0
}

int main(void) {
    printf("min2(3,7) = %d\n", min2(3,7));
    printf("abs_int(-5) = %d\n", abs_int(-5));
    printf("is_even(10) = %d\n", is_even(10));
    return 0;
}
```

Сборка и запуск:

```
gcc more_funcs.c -o more_funcs
./more_funcs
```

---

8) Практика (по возрастанию)

A. Простые функции

1. int inc(int x) — вернуть x + 1.
2. int last_digit(int x) — вернуть **последнюю цифру** (для отрицательных сначала сделайте x = abs(x) с помощью abs_int).
3. double c_to_f(double c) — перевести °C → °F по формуле F = C * 9/5 + 32.

B. Проверки

1. int in_range(int x, int lo, int hi) — вернуть 1, если lo ≤ x ≤ hi, иначе 0.
2. int max3(int a, int b, int c) — максимум из трёх (можно вызвать max2 дважды).

C. Маленькая задачка

1. Функция double triangle_area(double w, double h) — площадь прямоугольного треугольника w*h/2.
2. Программу, которая спрашивает w и h, печатает площадь с двумя знаками после точки.

---

9) Мини-экзамен Дня 5 — калькулятор a <op> b

Задание: программа **calc2**.

Пользователь вводит: целое a, **операцию** + - * / (одним символом), целое b.

Нужно распечатать результат. Каждая операция реализована **отдельной функцией**.

Требования:

- Чтение оператора делайте так, чтобы пропустить перевод строки/пробелы:

```
scanf(" %c", &op);   // пробел перед %c — важно!
```

-
- Деление целочисленное. Если b == 0 и операция / — вывести Division by zero и завершить.
- Структура кода: **прототипы** функций сверху, **реализации** ниже main.

Шаблон (скопируйте в main.c и допишите):

```
#include <stdio.h>

/* Прототипы */
int add(int a, int b);
int sub(int a, int b);
int mul(int a, int b);
int divide_int(int a, int b);   // предположим, что b != 0

int main(void) {
    int a, b;
```

```
    char op;

    printf("Enter: a op b (e.g. 3 * 4): ");
    if (scanf("%d", &a) != 1) { printf("Bad a\n"); return 1; }
    if (scanf(" %c", &op) != 1) { printf("Bad op\n"); return 1; }
    if (scanf("%d", &b) != 1) { printf("Bad b\n"); return 1; }

    int result = 0;

    if (op == '+') {
        result = add(a, b);
    } else if (op == '-') {
        result = sub(a, b);
    } else if (op == '*') {
        result = mul(a, b);
    } else if (op == '/') {
        if (b == 0) { printf("Division by zero\n"); return 1; }
        result = divide_int(a, b);
    } else {
        printf("Unknown op\n");
        return 1;
    }

    printf("Result: %d\n", result);
    return 0;
}

/* Реализации */
int add(int a, int b) { return a + b; }
int sub(int a, int b) { return a - b; }
int mul(int a, int b) { return a * b; }
int divide_int(int a, int b) { return a / b; }  // целое деление
```

Сборка и запуск:

```
gcc main.c -o calc2
./calc2
```

Проверьте:

```
3 + 4      → 7
10 - 25    → -15
6 * 7      → 42
```

```
8 / 3      → 2
5 / 0      → Division by zero
```

---

10) Частые ошибки (и как их избежать)

- **Забытый return** в функции, которая должна что-то вернуть.
- **Неверный тип**: вернули double, а функция объявлена как int. Тип результата и printf должны совпадать.
- **Целочисленное деление вместо вещественного**: в среднем используйте double и делите на 2.0.
- **Чтение символа оператора**: без пробела в " %c" часто считывается предыдущий перевод строки.
- **Порядок функций**: если вызываете функцию до её реализации — добавьте **прототип** сверху.

---

11) Бонус (по желанию)

- Сделайте версию калькулятора, которая **повторяет запрос** до команды q (понадобится цикл из Дня 4).
- Добавьте операцию % (остаток) и аккуратную проверку b == 0.
- Напишите функцию print_menu(void) и выведите подсказку для пользователя.

День 6 — Массивы: храним много чисел и работаем с ними

Цель дня

К концу дня вы:

- Поймёте, что такое **массив** (набор однотипных значений).
- Научитесь создавать массивы int a[...];, читать в них данные и печатать.
- Посчитаете **сумму, среднее, минимум, максимум**.
- Научитесь **линейному поиску** (найти число в массиве).
- Сдадите мини-экзамен: **анализ оценок** (статистика + поиск + разворот).

---

1) Готовим папку

```
mkdir -p ~/c-course/day06
cd ~/c-course/day06
```

---

2) Что такое массив (очень просто)

Массив — это «ряд коробочек» **одного типа**, стоящих подряд в памяти.

- Объявление: int a[5]; — 5 целых чисел.
- Индексы **с нуля**: a[0] ... a[4].
- Важно: **не выходите за границы**.

```
// arr_basics.c
#include <stdio.h>

int main(void) {
    int a[5];
    a[0] = 10; a[1] = 20; a[2] = 30; a[3] = 40; a[4] = 50;
    printf("%d %d %d %d %d\n", a[0], a[1], a[2], a[3], a[4]);
    return 0;
}
```

---

3) Цикл + массив: печать всех элементов

```
// arr_print.c
#include <stdio.h>

int main(void) {
    int a[5] = {10, 20, 30, 40, 50};
    int n = 5;
    for (int i = 0; i < n; i++) {
        printf("a[%d] = %d\n", i, a[i]);
    }
    return 0;
}
```

---

4) Читаем N чисел от пользователя в массив

```
// arr_input.c
#include <stdio.h>

int main(void) {
    const int MAX = 100;
    int a[MAX];
    int n;

    printf("Сколько чисел (1..100)? ");
    scanf("%d", &n);
    if (n < 1 || n > MAX) { printf("Некорректное N\n"); return 1; }

    for (int i = 0; i < n; i++) {
        printf("a[%d] = ", i);
        scanf("%d", &a[i]);
    }

    printf("Вы ввели: ");
    for (int i = 0; i < n; i++) printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

---

5) Сумма и среднее

```
// arr_sum_avg.c
#include <stdio.h>

int main(void) {
    const int MAX = 100;
    int a[MAX], n;

    printf("N: ");
    scanf("%d", &n);
    if (n < 1 || n > MAX) { printf("Некорректное N\n"); return 1; }

    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
```

```
    long long sum = 0;
    for (int i = 0; i < n; i++) sum += a[i];

    double avg = (double)sum / n;
    printf("Sum = %lld\n", sum);
    printf("Avg = %.2f\n", avg);
    return 0;
}
```

---

6) Минимум и максимум (и их позиции)

```
// arr_min_max.c
#include <stdio.h>

int main(void) {
    const int MAX = 100;
    int a[MAX], n;

    printf("N: ");
    scanf("%d", &n);
    if (n < 1 || n > MAX) { printf("Некорректное N\n"); return 1; }

    for (int i = 0; i < n; i++) scanf("%d", &a[i]);

    int min = a[0], max = a[0];
    int min_pos = 0, max_pos = 0;

    for (int i = 1; i < n; i++) {
        if (a[i] < min) { min = a[i]; min_pos = i; }
        if (a[i] > max) { max = a[i]; max_pos = i; }
    }

    printf("Min = %d at index %d\n", min, min_pos);
    printf("Max = %d at index %d\n", max, max_pos);
    return 0;
}
```

---

7) Линейный поиск: найти число в массиве

```c
// arr_search.c
#include <stdio.h>

int main(void) {
    const int MAX = 100;
    int a[MAX], n, key;

    printf("N: ");
    scanf("%d", &n);
    if (n < 1 || n > MAX) { printf("Некорректное N\n"); return 1; }

    for (int i = 0; i < n; i++) scanf("%d", &a[i]);

    printf("Ищем число: ");
    scanf("%d", &key);

    int pos = -1;
    for (int i = 0; i < n; i++) {
        if (a[i] == key) { pos = i; break; }
    }
    printf("Позиция: %d\n", pos); // -1 если не найдено
    return 0;
}
```

---

8) Разворот массива

Печать наоборот:

```c
// arr_reverse_print.c
#include <stdio.h>

int main(void) {
    int a[5] = {10, 20, 30, 40, 50};
    int n = 5;
    for (int i = n - 1; i >= 0; i--) printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

Ин-плейс разворот:

```c
// arr_reverse_inplace.c
#include <stdio.h>

int main(void) {
    int a[6] = {1, 2, 3, 4, 5, 6};
    int n = 6;

    for (int i = 0; i < n/2; i++) {
        int tmp = a[i];
        a[i] = a[n-1-i];
        a[n-1-i] = tmp;
    }

    for (int i = 0; i < n; i++) printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

---

9) Частые ошибки

1. **Выход за границы**: цикл i < n, не i <= n.
2. **Не проверили n**: всегда 1 ≤ n ≤ MAX.
3. **Неинициализированные элементы**: не используйте, если не записали.
4. **Среднее как целое**: делите через double.
5. **min/max**: начните с a[0], цикл — с i = 1.

---

10) Практика (по возрастанию)

- **A. Ввод/вывод:** спросите n (1..10), затем n целых; выведите их в одну строку.
- **B. Статистика:** посчитайте сумму, Avg=%.2f, минимум и максимум.
- **C. Счётчики:** посчитайте, сколько чисел чётные и нечётные.
- **D. Поиск:** спросите key; выведите индекс **первого** вхождения или -1.
- **E. Разворот:** выведите элементы в обратном порядке (сначала без изменения массива, затем — ин-плейс).

---

11) Мини-экзамен Дня 6 — «Анализ оценок»

Задание: программа **scores**.

Ввод: n (1..100), затем n оценок 0..100.

Вывод:

1. Count = n
2. Min = ... и Max = ...
3. Avg = ... (два знака после точки)
4. спросить Key: и вывести Index = ... (первое вхождение или -1)
5. Reversed: — оценки в обратном порядке в одну строку

```
// main.c
#include <stdio.h>

int main(void) {
    const int MAX = 100;
    int a[MAX], n;

    printf("N: ");
    scanf("%d", &n);
    if (n < 1 || n > MAX) { printf("Некорректное N\n"); return 1; }

    for (int i = 0; i < n; i++) scanf("%d", &a[i]);

    printf("Count = %d\n", n);

    int min = a[0], max = a[0];
    for (int i = 1; i < n; i++) {
        if (a[i] < min) min = a[i];
        if (a[i] > max) max = a[i];
    }
    printf("Min = %d\n", min);
    printf("Max = %d\n", max);

    long long sum = 0;
    for (int i = 0; i < n; i++) sum += a[i];
    double avg = (double)sum / n;
    printf("Avg = %.2f\n", avg);

    int key;
    printf("Key: ");
    scanf("%d", &key);
    int pos = -1;
```

```
    for (int i = 0; i < n; i++) {
        if (a[i] == key) { pos = i; break; }
    }
    printf("Index = %d\n", pos);

    printf("Reversed: ");
    for (int i = n - 1; i >= 0; i--) {
        printf("%d", a[i]);
        if (i > 0) printf(" ");
    }
    printf("\n");
    return 0;
}
```

Сборка и запуск:

```
gcc main.c -o scores
./scores
```

Критерии зачёта: корректные Min/Max/Avg, верный индекс поиска, правильный разворот.

---

12) Бонус: простая сортировка (пузырёк)

```
// arr_bubble_sort.c
#include <stdio.h>

int main(void) {
    int a[6] = {5, 1, 4, 2, 8, 0};
    int n = 6;

    for (int pass = 0; pass < n - 1; pass++) {
        for (int i = 0; i < n - 1 - pass; i++) {
            if (a[i] > a[i+1]) {
                int tmp = a[i]; a[i] = a[i+1]; a[i+1] = tmp;
            }
        }
    }

    for (int i = 0; i < n; i++) printf("%d ", a[i]);
    printf("\n");
```

```
    return 0;
}
```

День 7 — Строки: читаем, печатаем, считаем длину и меняем символы

Цель дня

К концу дня вы:

- Поймёте, что такое **строка в C** (массив char, оканчивается нулём '\0').
- Научитесь **безопасно** читать строку с клавиатуры (fgets) и печатать её.
- Сможете посчитать **длину строки**, количество **пробелов/букв/цифр**.
- Научитесь **удалять '\n'** в конце строки и **менять регистр букв**.
- Сдадите мини-экзамен: программа **инверсии регистра** всей строки.

---

1) Готовим папку

```
mkdir -p ~/c-course/day07
cd ~/c-course/day07
```

---

2) Что такое строка в C (очень просто)

- Строка в C — это **массив символов char**, в конце которого стоит **нулевой байт** '\0'.
- Пример литерала: "Hello" — это символы H e l l o и в конце **невидимый** '\0'.

> Важно: строка заканчивается **первым** '\0'. Если его нет — функции «убежат» читать чужую память (плохо!).

Мини-пример:

```
// str_print.c
#include <stdio.h>

int main(void) {
```

```
    char s[] = "Hello";       // массив из {'H','e','l','l','o','\0'}
    printf("%s\n", s);        // %s печатает до '\0'
    return 0;
}
```

---

3) Как читать строку безопасно

Не используйте gets (её нет) и не используйте пока scanf("%s", ...) — он **режет по пробелу** и может переполнить буфер.

Правильно — fgets:

```
// str_input.c
#include <stdio.h>

int main(void) {
    char buf[64];                                  // максимум 63 видимых
символа + '\0'
    printf("Введите строку (до 63 символов): ");
    if (fgets(buf, sizeof buf, stdin) == NULL) { // NULL — ошибка/EOF
        return 1;
    }
    printf("Вы ввели (с \\n если он влез): \"%s\"\n", buf);
    return 0;
}
```

fgets **сохраняет** '\n', если он поместился. Обычно его удобно убрать.

Функция «убрать финальный \n»:

```
// chomp.c
#include <stdio.h>

void chomp(char *s) {
    for (int i = 0; s[i] != '\0'; i++) {
        if (s[i] == '\n') { s[i] = '\0'; break; }
    }
}

int main(void) {
    char buf[64];
```

```
    printf("Введите строку: ");
    if (!fgets(buf, sizeof buf, stdin)) return 1;
    chomp(buf);
    printf("Без \\n: \"%s\"\n", buf);
    return 0;
}
```

---

4) Длина строки (считаем сами)

Мы пока не пользуемся библиотечной strlen. Напишем свою простую версию.

```
// my_strlen.c
#include <stdio.h>

int my_strlen(const char *s) {
    int n = 0;
    while (s[n] != '\0') {
        n++;
    }
    return n;
}

int main(void) {
    char buf[64];
    printf("Введите строку: ");
    if (!fgets(buf, sizeof buf, stdin)) return 1;
    // уберём \n
    for (int i = 0; buf[i]; i++) { if (buf[i] == '\n') { buf[i] = '\0';
break; } }
    printf("Длина: %d\n", my_strlen(buf));
    return 0;
}
```

---

5) Считаем пробелы, буквы, цифры

Подключим <ctype.h> — там есть полезные проверки: isspace, isalpha, isdigit, и преобразования tolower, toupper.

```c
// str_count.c
#include <stdio.h>
#include <ctype.h>   // isspace, isalpha, isdigit

int main(void) {
    char s[128];
    printf("Введите строку: ");
    if (!fgets(s, sizeof s, stdin)) return 1;

    int spaces = 0, letters = 0, digits = 0, others = 0;

    for (int i = 0; s[i] != '\0'; i++) {
        unsigned char c = (unsigned char)s[i]; // безопаснее для ctype
        if (c == '\n') continue;               // игнорируем перевод
строки
        if (isspace(c)) spaces++;
        else if (isalpha(c)) letters++;
        else if (isdigit(c)) digits++;
        else others++;
    }

    printf("Пробелы: %d\n", spaces);
    printf("Буквы:   %d\n", letters);
    printf("Цифры:   %d\n", digits);
    printf("Иные:    %d\n", others);
    return 0;
}
```

---

6) Меняем регистр букв (a↔A)

Сделаем функцию, которая меняет **каждую латинскую букву**: маленькая → большая, большая → маленькая. Остальное не трогаем.

```c
// toggle_case.c
#include <stdio.h>
#include <ctype.h>

void toggle_case(char *s) {
    for (int i = 0; s[i] != '\0'; i++) {
        unsigned char c = (unsigned char)s[i];
```

```
        if (isalpha(c)) {
            if (islower(c)) s[i] = (char)toupper(c);
            else            s[i] = (char)tolower(c);
        }
    }
}

int main(void) {
    char s[128];
    printf("Введите строку: ");
    if (!fgets(s, sizeof s, stdin)) return 1;
    // убирать \n не обязательно — печатать будем как есть
    toggle_case(s);
    printf("%s", s);  // s уже содержит \n, если влез
    return 0;
}
```

---

7) Склеиваем строки: имя + фамилия

Самый простой безопасный способ — **snprintf** (форматированная запись в буфер фиксированного размера).

```
// full_name.c
#include <stdio.h>

int main(void) {
    char first[32], last[32];
    char full[80];

    printf("Имя: ");
    if (!fgets(first, sizeof first, stdin)) return 1;
    printf("Фамилия: ");
    if (!fgets(last, sizeof last, stdin)) return 1;

    // уберём \n в обоих
    for (int i = 0; first[i]; i++) if (first[i] == '\n') { first[i] =
'\0'; break; }
    for (int i = 0; last[i];  i++) if (last[i]  == '\n') { last[i]  =
'\0'; break; }

    // full = "<Имя> <Фамилия>"
```

```
    // snprintf не переполнит буфер: он обрежет и поставит '\0'
    snprintf(full, sizeof full, "%s %s", first, last);

    printf("Полное имя: \"%s\"\n", full);
    return 0;
}
```

> Альтернатива «вручную» — копировать символы по одному в full, следя за границей и '\0'. Мы ещё вернёмся к этому позже.

---

8) Практика (по возрастанию)

A. Ввод/вывод

1. Прочитайте строку в буфер 64 байта, **уберите \n**, напечатайте её в кавычках.
2. Напечатайте её **длину** (своей функцией my_strlen).

B. Подсчёты

1. Посчитайте **количество пробелов**.
2. Посчитайте **количество слов**: слово — это последовательность **не-пробельных** символов; используйте isspace. Подсказка: ищите переход «было пробел/начало → стал не пробел».

C. Преобразования

1. Сделайте **все буквы маленькими** (tolower).
2. Сделайте **первые буквы слов заглавными** (начало строки или после пробела → toupper).

D. Склейка

1. Прочитайте имя и фамилию (через fgets) и соберите full = "Фамилия, Имя".

---

9) Мини-экзамен Дня 7 — «Инвертор регистра»

Задание: программа **invert_case**.

Читает **одну строку** (до 256 символов), меняет регистр **каждой латинской буквы**:

- a..z → A..Z

- A..Z → a..z Остальные символы (пробелы, цифры, знаки) — **без изменений**. Выводит результат **ровно один раз**.

Шаблон (main.c):

```
#include <stdio.h>
#include <ctype.h>

void invert_case(char *s) {
    for (int i = 0; s[i] != '\0'; i++) {
        unsigned char c = (unsigned char)s[i];
        if (isalpha(c)) {
            if (islower(c)) s[i] = (char)toupper(c);
            else            s[i] = (char)tolower(c);
        }
    }
}

int main(void) {
    char s[256];
    if (!fgets(s, sizeof s, stdin)) return 1;
    invert_case(s);
    printf("%s", s);     // печатаем как есть
    return 0;
}
```

Сборка и запуск:

```
gcc main.c -o invert_case
./invert_case
```

Проверьте:

```
Ввод:  Hello, World! 123
Вывод: hELLO, wORLD! 123
```

Критерии зачёта:

- Строка читается целиком (с пробелами).
- Регистр у латинских букв меняется правильно, остальные символы — без изменений.
- Нет вылета за границы буфера (используем fgets с фиксированным размером).

10) Частые ошибки и как их избежать

- **scanf("%s", buf) «съедает» только до пробела** и может переполнить буфер — лучше fgets.
- **Забыли убрать \n** — сравнения строк и печать «ломаются» (вы видите лишний переход строки).
- **Нет завершающего '\0'** — при ручной склейке всегда ставьте ноль-терминатор.
- **ctype и знаки >127** — приводите символ к unsigned char перед isalpha/toupper и т.п.
- **UTF-8 и «русские буквы»** — сегодня работаем с **латиницей**; кириллица в UTF-8 состоит из нескольких байт и требует других техник (позже).

День 7 — Строки: читаем, печатаем, считаем длину и меняем символы

Цель дня

К концу дня вы:

- Поймёте, что такое **строка в C** (массив char, оканчивается нулём '\0').
- Научитесь **безопасно** читать строку с клавиатуры (fgets) и печатать её.
- Сможете посчитать **длину строки**, количество **пробелов/букв/цифр**.
- Научитесь **удалять '\n'** в конце строки и **менять регистр букв**.
- Сдадите мини-экзамен: программа **инверсии регистра** всей строки.

---

1) Готовим папку

```
mkdir -p ~/c-course/day07
cd ~/c-course/day07
```

---

2) Что такое строка в C (очень просто)

- Строка в C — это **массив символов char**, в конце которого стоит **нулевой байт** '\0'.
- Пример литерала: "Hello" — это символы H e l l o и в конце **невидимый** '\0'.

> Важно: строка заканчивается **первым** '\0'. Если его нет — функции «убежат» читать чужую память (плохо!).

Мини-пример:

```
// str_print.c
#include <stdio.h>

int main(void) {
    char s[] = "Hello";       // массив из {'H','e','l','l','o','\0'}
    printf("%s\n", s);        // %s печатает до '\0'
    return 0;
}
```

---

3) Как читать строку безопасно

Не используйте gets (её нет) и не используйте пока scanf("%s", ...) — он **режет по пробелу** и может переполнить буфер.

Правильно — fgets:

```
// str_input.c
#include <stdio.h>

int main(void) {
    char buf[64];                                  // максимум 63 видимых
символа + '\0'
    printf("Введите строку (до 63 символов): ");
    if (fgets(buf, sizeof buf, stdin) == NULL) { // NULL — ошибка/EOF
        return 1;
    }
    printf("Вы ввели (с \\n если он влез): \"%s\"\n", buf);
    return 0;
}
```

fgets **сохраняет** '\n', если он поместился. Обычно его удобно убрать.

Функция «убрать финальный \n»:

```
// chomp.c
#include <stdio.h>
```

```
void chomp(char *s) {
    for (int i = 0; s[i] != '\0'; i++) {
        if (s[i] == '\n') { s[i] = '\0'; break; }
    }
}

int main(void) {
    char buf[64];
    printf("Введите строку: ");
    if (!fgets(buf, sizeof buf, stdin)) return 1;
    chomp(buf);
    printf("Без \\n: \"%s\"\n", buf);
    return 0;
}
```

---

4) Длина строки (считаем сами)

Мы пока не пользуемся библиотечной strlen. Напишем свою простую версию.

```
// my_strlen.c
#include <stdio.h>

int my_strlen(const char *s) {
    int n = 0;
    while (s[n] != '\0') {
        n++;
    }
    return n;
}

int main(void) {
    char buf[64];
    printf("Введите строку: ");
    if (!fgets(buf, sizeof buf, stdin)) return 1;
    // уберём \n
    for (int i = 0; buf[i]; i++) { if (buf[i] == '\n') { buf[i] = '\0';
break; } }
    printf("Длина: %d\n", my_strlen(buf));
```

```
    return 0;
}
```

---

5) Считаем пробелы, буквы, цифры

Подключим <ctype.h> — там есть полезные проверки: isspace, isalpha, isdigit, и преобразования tolower, toupper.

```
// str_count.c
#include <stdio.h>
#include <ctype.h>   // isspace, isalpha, isdigit

int main(void) {
    char s[128];
    printf("Введите строку: ");
    if (!fgets(s, sizeof s, stdin)) return 1;

    int spaces = 0, letters = 0, digits = 0, others = 0;

    for (int i = 0; s[i] != '\0'; i++) {
        unsigned char c = (unsigned char)s[i]; // безопаснее для ctype
        if (c == '\n') continue;               // игнорируем перевод
строки
        if (isspace(c)) spaces++;
        else if (isalpha(c)) letters++;
        else if (isdigit(c)) digits++;
        else others++;
    }

    printf("Пробелы: %d\n", spaces);
    printf("Буквы:   %d\n", letters);
    printf("Цифры:   %d\n", digits);
    printf("Иные:    %d\n", others);
    return 0;
}
```

---

6) Меняем регистр букв (a↔A)

Сделаем функцию, которая меняет **каждую латинскую букву**: маленькая → большая, большая → маленькая. Остальное не трогаем.

```
// toggle_case.c
#include <stdio.h>
#include <ctype.h>

void toggle_case(char *s) {
    for (int i = 0; s[i] != '\0'; i++) {
        unsigned char c = (unsigned char)s[i];
        if (isalpha(c)) {
            if (islower(c)) s[i] = (char)toupper(c);
            else            s[i] = (char)tolower(c);
        }
    }
}

int main(void) {
    char s[128];
    printf("Введите строку: ");
    if (!fgets(s, sizeof s, stdin)) return 1;
    // убирать \n не обязательно — печатать будем как есть
    toggle_case(s);
    printf("%s", s);  // s уже содержит \n, если влез
    return 0;
}
```

---

7) Склеиваем строки: имя + фамилия

Самый простой безопасный способ — **snprintf** (форматированная запись в буфер фиксированного размера).

```
// full_name.c
#include <stdio.h>

int main(void) {
    char first[32], last[32];
    char full[80];

    printf("Имя: ");
    if (!fgets(first, sizeof first, stdin)) return 1;
```

```
    printf("Фамилия: ");
    if (!fgets(last, sizeof last, stdin)) return 1;

    // уберём \n в обоих
    for (int i = 0; first[i]; i++) if (first[i] == '\n') { first[i] =
'\0'; break; }
    for (int i = 0; last[i];  i++) if (last[i]  == '\n') { last[i]  =
'\0'; break; }

    // full = "<Имя> <Фамилия>"
    // snprintf не переполнит буфер: он обрежет и поставит '\0'
    snprintf(full, sizeof full, "%s %s", first, last);

    printf("Полное имя: \"%s\"\n", full);
    return 0;
}
```

> Альтернатива «вручную» — копировать символы по одному в full, следя за границей и '\0'. Мы ещё вернёмся к этому позже.

---

8) Практика (по возрастанию)

A. Ввод/вывод

1. Прочитайте строку в буфер 64 байта, **уберите \n**, напечатайте её в кавычках.
2. Напечатайте её **длину** (своей функцией my_strlen).

B. Подсчёты

1. Посчитайте **количество пробелов**.
2. Посчитайте **количество слов**: слово — это последовательность **не-пробельных** символов; используйте isspace. Подсказка: ищите переход «было пробел/начало → стал не пробел».

C. Преобразования

1. Сделайте **все буквы маленькими** (tolower).
2. Сделайте **первые буквы слов заглавными** (начало строки или после пробела → toupper).

D. Склейка

1. Прочитайте имя и фамилию (через fgets) и соберите full = "Фамилия, Имя".

9) Мини-экзамен Дня 7 — «Инвертор регистра»

Задание: программа **invert_case**.

Читает **одну строку** (до 256 символов), меняет регистр **каждой латинской буквы**:

- a..z → A..Z
- A..Z → a..z Остальные символы (пробелы, цифры, знаки) — **без изменений**. Выводит результат **ровно один раз**.

Шаблон (main.c):

```
#include <stdio.h>
#include <ctype.h>

void invert_case(char *s) {
    for (int i = 0; s[i] != '\0'; i++) {
        unsigned char c = (unsigned char)s[i];
        if (isalpha(c)) {
            if (islower(c)) s[i] = (char)toupper(c);
            else            s[i] = (char)tolower(c);
        }
    }
}

int main(void) {
    char s[256];
    if (!fgets(s, sizeof s, stdin)) return 1;
    invert_case(s);
    printf("%s", s);     // печатаем как есть
    return 0;
}
```

Сборка и запуск:

```
gcc main.c -o invert_case
./invert_case
```

Проверьте:

```
Ввод:  Hello, World! 123
```

```
Вывод: hELLO, wORLD! 123
```

Критерии зачёта:

- Строка читается целиком (с пробелами).
- Регистр у латинских букв меняется правильно, остальные символы — без изменений.
- Нет вылета за границы буфера (используем fgets с фиксированным размером).

---

10) Частые ошибки и как их избежать

- **scanf("%s", buf) «съедает» только до пробела** и может переполнить буфер — лучше fgets.
- **Забыли убрать \n** — сравнения строк и печать «ломаются» (вы видите лишний переход строки).
- **Нет завершающего '\0'** — при ручной склейке всегда ставьте ноль-терминатор.
- **ctype и знаки >127** — приводите символ к unsigned char перед isalpha/toupper и т.п.
- **UTF-8 и «русские буквы»** — сегодня работаем с **латиницей**; кириллица в UTF-8 состоит из нескольких байт и требует других техник (позже).

День 9 — Структуры: свои мини-«карточки данных»

Цель дня

К концу дня вы:

- Поймёте, что такое **структура** — набор полей разных типов под одним именем.
- Научитесь объявлять структуры, создавать переменные и обращаться к полям (.) и через указатель (->).
- Напишете функции для **ввода/печати** одной структуры и работы с **массивом структур**.
- Сдадите мини-экзамен: маленький «справочник студентов» (ввод N записей, таблица, лучший по баллу, поиск).

---

1) Готовим папку

```
mkdir -p ~/c-course/day09
cd ~/c-course/day09
```

---

2) Что такое структура (очень просто)

Структура — это «карточка» с полями. Пример карточки студента:

```
// struct_basics.c
#include <stdio.h>

struct Student {
    char  name[32];  // имя (до 31 видимого символа)
    int   age;       // возраст
    double score;    // балл (например, 0..100)
};

int main(void) {
    struct Student s;          // создаём переменную-студента
    // Заполняем поля:
    // Строковое поле пока простым копированием посимвольно делать не
будем.
    // Для демонстрации присвоим напрямую числовые поля:
    s.age = 20;
    s.score = 88.5;

    // Печать полей:
    printf("age=%d, score=%.1f\n", s.age, s.score);
    return 0;
}
```

> К полям обращаемся через точку: s.age, s.score.

---

3) Удобные сокращения:

typedef

и инициализация

Чтобы не писать каждый раз struct Student, используем typedef:

```c
// typedef_demo.c
#include <stdio.h>

typedef struct {
    char  name[32];
    int   age;
    double score;
} Student;

int main(void) {
    // Инициализация при объявлении:
    Student a = { "Alice", 19, 91.0 };

    // Именованная инициализация (порядок не важен):
    Student b = { .score = 76.5, .age = 20, .name = "Bob" };

    printf("%s (%d) -> %.1f\n", a.name, a.age, a.score);
    printf("%s (%d) -> %.1f\n", b.name, b.age, b.score);
    return 0;
}
```

---

4) Указатели на структуры: оператор ->

```c
// arrow_demo.c
#include <stdio.h>

typedef struct {
    char  name[32];
    int   age;
    double score;
} Student;

int main(void) {
    Student s = { "Carol", 21, 85.0 };
    Student *p = &s;        // p указывает на s

    // Два равных способа:
    (*p).age = 22;          // через разыменование
```

```
    p->score = 90.0;        // «стрелочка» — удобное сокращение

    printf("%s (%d) -> %.1f\n", s.name, s.age, s.score);
    return 0;
}
```

---

5) Ввод/вывод одной структуры (аккуратный ввод строки)

Сделаем две маленькие функции: **читать** студента и **печать** студента.

```
// io_one_student.c
#include <stdio.h>
#include <string.h>  // для strlen/strcmp при желании

typedef struct {
    char  name[32];
    int   age;
    double score;
} Student;

static void chomp(char *s) {
    // убрать финальный '\n', если есть
    size_t n = strlen(s);
    if (n && s[n-1] == '\n') s[n-1] = '\0';
}

void read_student(Student *st) {
    // Имя (может содержать пробелы) — берём fgets
    printf("Name: ");
    if (!fgets(st->name, sizeof st->name, stdin)) {
        st->name[0] = '\0';
    } else {
        chomp(st->name);
    }

    // Возраст
    printf("Age: ");
    scanf("%d", &st->age);

    // Балл
```

```
    printf("Score: ");
    scanf("%lf", &st->score);

    // После scanf в буфере остаётся '\n' — очистим до конца строки
    int c;
    while ((c = getchar()) != '\n' && c != EOF) { /* flush */ }
}

void print_student(const Student *st) {
    printf("%-31s | %3d | %6.2f\n", st->name, st->age, st->score);
}

int main(void) {
    Student s;
    read_student(&s);
    printf("\n%-31s | %s | %s\n", "Name", "Age", "Score");
    print_student(&s);
    return 0;
}
```

---

6) Массив структур: список студентов

Заполним массив в цикле и распечатаем **таблицу**.

```
// array_of_structs.c
#include <stdio.h>
#include <string.h>

typedef struct {
    char  name[32];
    int   age;
    double score;
} Student;

static void chomp(char *s) {
    size_t n = strlen(s);
    if (n && s[n-1] == '\n') s[n-1] = '\0';
}

void read_student(Student *st) {
    printf("Name: ");
```

```
    if (!fgets(st->name, sizeof st->name, stdin)) st->name[0] = '\0';
    else chomp(st->name);

    printf("Age: ");
    scanf("%d", &st->age);
    printf("Score: ");
    scanf("%lf", &st->score);
    int c; while ((c = getchar()) != '\n' && c != EOF) {}
}

void print_header(void) {
    printf("%-31s | %3s | %6s\n", "Name", "Age", "Score");
    printf("-----------------------------------------------\n");
}

void print_student(const Student *st) {
    printf("%-31s | %3d | %6.2f\n", st->name, st->age, st->score);
}

int main(void) {
    const int N = 3;      // для примера возьмём 3
    Student g[N];

    for (int i = 0; i < N; i++) {
        printf("\n#%d\n", i);
        read_student(&g[i]);
    }

    printf("\n");
    print_header();
    for (int i = 0; i < N; i++) print_student(&g[i]);
    return 0;
}
```

---

7) Полезные функции над массивом структур

Найдём **лучшего по баллу**, посчитаем **средний балл**, поднимем балл всем на дельту.

```
// ops_on_array.c
#include <stdio.h>
#include <string.h>
```

```c
typedef struct {
    char  name[32];
    int   age;
    double score;
} Student;

int best_index(const Student *a, int n) {
    int best = 0;
    for (int i = 1; i < n; i++) {
        if (a[i].score > a[best].score) best = i;
    }
    return best;
}

double average_score(const Student *a, int n) {
    double sum = 0.0;
    for (int i = 0; i < n; i++) sum += a[i].score;
    return sum / n;
}

void add_bonus(Student *a, int n, double delta) {
    for (int i = 0; i < n; i++) a[i].score += delta;
}

int main(void) {
    Student g[3] = {
        { "Ann", 19, 90.0 },
        { "Bob", 20, 75.5 },
        { "Cat", 18, 88.0 }
    };
    printf("Avg = %.2f\n", average_score(g, 3));
    printf("Best = %s\n", g[best_index(g, 3)].name);
    add_bonus(g, 3, 1.0);
    printf("After bonus, best score = %.1f\n", g[best_index(g, 3)].score);
    return 0;
}
```

---

8) Поиск по имени (точное совпадение)

Минимально используем strcmp из <string.h>: возвращает 0, если строки **равны**.

```c
// find_by_name.c
#include <stdio.h>
#include <string.h>

typedef struct { char name[32]; int age; double score; } Student;

int find_by_name(const Student *a, int n, const char *key) {
    for (int i = 0; i < n; i++) {
        if (strcmp(a[i].name, key) == 0) return i;
    }
    return -1;
}

int main(void) {
    Student g[3] = { {"Ann",19,90}, {"Bob",20,76}, {"Cat",18,88} };
    printf("%d\n", find_by_name(g, 3, "Bob"));  // 1
    printf("%d\n", find_by_name(g, 3, "Eve"));  // -1
    return 0;
}
```

---

9) Практика (по возрастанию)

A. Одна карточка

1. Объявите Student s = {"Ivan Petrov", 21, 72.5} и распечатайте в формате:

   Ivan Petrov | 21 | 72.50.

B. Ввод одной карточки

1. Напишите read_student и print_student (как выше). Протестируйте.

C. Список из N студентов

1. Спросите n (1..10).
2. Прочитайте n студентов.
3. Распечатайте таблицу.

D. Аналитика

1. Найдите best_index.
2. Посчитайте средний балл и выведите Avg = ...

3. Реализуйте add_bonus(g, n, 2.0) и выведите обновлённую таблицу.

E. Поиск

1. Спросите ключ name (строка с пробелами через fgets).
2. Выведите индекс первого совпадения или -1.

---

10) Мини-экзамен Дня 9 — «Мини-справочник студентов»

Задание: программа **students**.

Ввод:

- n (1..50)
- затем n студентов: имя (строка до 31 символа, может содержать пробелы), возраст (int), балл (double).

Вывод:

1. Таблица:

```
Name                            | Age |  Score
------------------------------------------------
...
```

1.
2. Count = n
3. Avg = XX.XX
4. Best = <имя лучшего>
5. Поиск: спросить строку Find name: и вывести Index = k (или -1).

Шаблон (main.c):

```
#include <stdio.h>
#include <string.h>

typedef struct {
    char  name[32];
    int   age;
    double score;
} Student;
```

```
static void chomp(char *s) {
    size_t n = strlen(s);
    if (n && s[n-1] == '\n') s[n-1] = '\0';
}

void read_student(Student *st) {
    printf("Name: ");
    if (!fgets(st->name, sizeof st->name, stdin)) st->name[0] = '\0';
    else chomp(st->name);

    printf("Age: ");
    scanf("%d", &st->age);
    printf("Score: ");
    scanf("%lf", &st->score);

    int c; while ((c = getchar()) != '\n' && c != EOF) {} // очистить
хвост строки
}

void print_header(void) {
    printf("%-31s | %3s | %6s\n", "Name", "Age", "Score");
    printf("-----------------------------------------------\n");
}
void print_student(const Student *st) {
    printf("%-31s | %3d | %6.2f\n", st->name, st->age, st->score);
}

int best_index(const Student *a, int n) {
    int best = 0;
    for (int i = 1; i < n; i++) if (a[i].score > a[best].score) best = i;
    return best;
}
double average_score(const Student *a, int n) {
    double sum = 0.0;
    for (int i = 0; i < n; i++) sum += a[i].score;
    return sum / n;
}
int find_by_name(const Student *a, int n, const char *key) {
    for (int i = 0; i < n; i++) if (strcmp(a[i].name, key) == 0) return i;
    return -1;
}
```

```
int main(void) {
    const int MAX = 50;
    Student g[MAX];
    int n;

    printf("N: ");
    scanf("%d", &n);
    int c; while ((c = getchar()) != '\n' && c != EOF) {} // очистить
после scanf
    if (n < 1 || n > MAX) { printf("Bad N\n"); return 1; }

    for (int i = 0; i < n; i++) {
        printf("\n#%d\n", i);
        read_student(&g[i]);
    }

    printf("\n");
    print_header();
    for (int i = 0; i < n; i++) print_student(&g[i]);

    printf("Count = %d\n", n);
    printf("Avg = %.2f\n", average_score(g, n));
    printf("Best = %s\n", g[best_index(g, n)].name);

    char key[32];
    printf("Find name: ");
    if (!fgets(key, sizeof key, stdin)) return 0;
    chomp(key);
    printf("Index = %d\n", find_by_name(g, n, key));
    return 0;
}
```

Сборка и запуск:

```
gcc -std=c17 -Wall -Wextra -Werror -O0 -g main.c -o students
./students
```

Критерии зачёта:

- Корректный ввод n и n карточек (имя с пробелами).
- Таблица печатается аккуратно, средний балл и лучший считаны верно.
- Поиск по имени работает (точное совпадение).

11) Частые ошибки и как их избежать

- **Смешали scanf и fgets и «поймали» лишний \n:** после scanf чистите буфер: int c; while ((c=getchar())!='\n' && c!=EOF) {}
- **Переполнение имени:** буфер name[32] → помещается **до 31** видимого символа (+ '\0'). fgets сам обрежет, но лучше предупреждать пользователя.
- **Сравнение строк через ==:** так сравниваются адреса. Для содержимого используйте strcmp(...) == 0.
- **Забыли '\0' при ручной сборке строки:** при использовании snprintf это безопаснее — он сам ставит нулевой байт.
- **Неправильный формат печати:** %d — int, %lf в scanf для double, а в printf для double — %f.

День 10 — Файлы: читаем и пишем на диск (stdio)

Цель дня

К концу дня вы:

- Поймёте, что такое **файл** и как с ним работать из C: fopen, fprintf, fscanf, fgets, fputs, fclose.
- Научитесь **писать** в файл, **читать** файл построчно и **добавлять** строки.
- Сделаете маленькие утилиты: «нумератор строк», «сумма чисел из файла», «копия файла».
- Сдадите мини-экзамен: **cp_simple** — копирование текстового файла построчно с подсчётом строк.

---

1) Готовим папку

```
mkdir -p ~/c-course/day10
cd ~/c-course/day10
```

Полезные команды терминала:

```
ls -l         # список файлов
```

```
cat file.txt  # показать содержимое
```

---

2) Открываем файл и ПИШЕМ в него

- FILE *f = fopen("имя", "режим");
- Режимы:
  - "w" — запись (создаст файл или обрежет существующий до 0).
  - "a" — **добавление** в конец.
  - "r" — чтение (файл должен существовать).

Простой пример записи строк:

```
// write_lines.c
#include <stdio.h>

int main(void) {
    FILE *f = fopen("notes.txt", "w");   // создаст/очистит файл
    if (!f) {                            // проверка на ошибку
        perror("fopen");
        return 1;
    }

    fprintf(f, "Первая строка\n");
    fprintf(f, "Вторая строка\n");
    fprintf(f, "Третья строка\n");

    fclose(f);                           // обязательно закрываем
    printf("Готово: создан notes.txt\n");
    return 0;
}
```

Сборка и запуск:

```
gcc write_lines.c -o write_lines
./write_lines
ls -l
cat notes.txt
```

---

3) Открываем файл и ДОБАВЛЯЕМ в конец

```
// append_line.c
#include <stdio.h>

int main(void) {
    FILE *f = fopen("notes.txt", "a");   // добавление
    if (!f) { perror("fopen"); return 1; }

    fprintf(f, "Новая строка (append)\n");
    fclose(f);
    printf("Добавлено в notes.txt\n");
    return 0;
}
```

---

4) Читаем файл ПОСТРОЧНО:

fgets

fgets(buf, size, f) читает **до перевода строки** или до заполнения буфера.

Если строка длиннее буфера — придёт кусок (это нормально для наших задач).

```
// read_lines.c
#include <stdio.h>

int main(void) {
    FILE *f = fopen("notes.txt", "r");
    if (!f) { perror("fopen"); return 1; }

    char line[256];
    int num = 1;

    while (fgets(line, sizeof line, f)) {     // пока удаётся читать
строку
        printf("%3d | %s", num, line);        // line уже содержит '\n'
        num++;
    }

    fclose(f);
```

```
    return 0;
}
```

---

5) Читаем ЧИСЛА из файла:

fscanf

или

fgets + sscanf

Вариант A:

fscanf

(по одному числу на строке)

Создадим файл с числами:

```
cat > nums.txt << 'EOF'
10
20
-5
15
EOF
```

Сумма и среднее:

```
// sum_nums.c
#include <stdio.h>

int main(void) {
    FILE *f = fopen("nums.txt", "r");
    if (!f) { perror("fopen"); return 1; }

    long long sum = 0;
    int count = 0, x;

    // читаем "пока удаётся прочитать int"
    while (fscanf(f, "%d", &x) == 1) {
        sum += x;
```

```
        count++;
    }

    fclose(f);

    if (count == 0) {
        printf("Файл пуст или нет чисел\n");
        return 0;
    }

    double avg = (double)sum / count;
    printf("Count = %d\nSum = %lld\nAvg = %.2f\n", count, sum, avg);
    return 0;
}
```

Вариант B (чуть аккуратнее):

fgets

+

sscanf

Читаем строку текстом, потом парсим её как число (полезно, если в строке могут быть лишние пробелы/комментарии).

```
// sum_nums_sscanf.c
#include <stdio.h>

int main(void) {
    FILE *f = fopen("nums.txt", "r");
    if (!f) { perror("fopen"); return 1; }

    char line[128];
    long long sum = 0;
    int count = 0, x;

    while (fgets(line, sizeof line, f)) {
        if (sscanf(line, "%d", &x) == 1) {  // смогли вытащить число
            sum += x;
            count++;
        }
    }
```

```
    fclose(f);

    if (count == 0) { puts("Нет чисел"); return 0; }
    printf("Count=%d Sum=%lld Avg=%.2f\n", count, sum, (double)sum/count);
    return 0;
}
```

---

6) Делаем КОПИЮ файла построчно

Идея: открыть исходник на чтение ("r"), открыть назначение на запись ("w"), читать fgets, писать fputs.

```
// copy_file.c
#include <stdio.h>

int main(void) {
    char src[256], dst[256];

    printf("Источник: ");
    if (!fgets(src, sizeof src, stdin)) return 1;
    // убрать '\n'
    for (int i = 0; src[i]; i++) { if (src[i] == '\n') { src[i] = '\0';
break; } }

    printf("Назначение: ");
    if (!fgets(dst, sizeof dst, stdin)) return 1;
    for (int i = 0; dst[i]; i++) { if (dst[i] == '\n') { dst[i] = '\0';
break; } }

    FILE *in  = fopen(src, "r");
    if (!in)  { perror("fopen src"); return 1; }
    FILE *out = fopen(dst, "w");
    if (!out) { perror("fopen dst"); fclose(in); return 1; }

    char buf[512];
    int lines = 0;
    while (fgets(buf, sizeof buf, in)) {
        fputs(buf, out);
        lines++;
    }
```

```
    fclose(in);
    fclose(out);

    printf("Скопировано строк: %d\n", lines);
    return 0;
}
```

Проверьте:

```
./copy_file
# введите: notes.txt  затем  notes_copy.txt
cat notes_copy.txt
```

---

7) Полезно знать (простыми словами)

- **Текущая папка** — куда пишете/читайте по относительному пути ("notes.txt"). Посмотреть: pwd.
- **Проверка ошибок важна**: если fopen вернул NULL, файла нет или нет прав. perror("fopen") выведет причину.
- Всегда делайте fclose(f) — так вы «смываете буфер» и отпускаете файл.
- fprintf ≈ printf, но в **файл**: первый аргумент — FILE*.
- fscanf ≈ scanf, но из **файла**.
- Для построчного чтения удобнее fgets (читает строку целиком).

---

8) Практика (по возрастанию)

A. Запись

1. Спросите имя файла (до 100 символов, без пробелов — можно scanf("%100s", name)), откройте в "w", спросите у пользователя три строки и запишите их через fprintf. Закройте.

B. Нумерация

1. Программа numlines: спрашивает имя файла, печатает содержимое с **номерами строк** слева (как в read_lines.c).

C. Сумма

1. Программа sumfile: файл с числами (по одному в строке) → Count/Sum/Avg, как в sum_nums.c.

D. Добавление

1. Программа append_stamp: добавляет строку вида --- appended --- в конец указанного файла.

E. Копия

1. Программа copy_file (как выше): копия построчно + вывод количества строк.

---

9) Мини-экзамен Дня 10 — cp_simple

Задание: программа **копирования текстового файла** построчно с подсчётом строк.

Требования:

- Спросить Source: и Dest: (имена файлов).
- Открыть исходник в "r", назначение в "w".
- Скопировать **все строки** (fgets → fputs).
- В конце напечатать ровно:

```
Lines: N
```

- где N — количество скопированных строк.
- При ошибке открытия — печатать через perror и завершать с кодом 1.

Шаблон (скопируйте в main.c):

```
#include <stdio.h>

static void chomp(char *s) { for (int i = 0; s[i]; i++) if (s[i] == '\n')
{ s[i] = '\0'; break; } }

int main(void) {
    char src[256], dst[256];

    printf("Source: ");
    if (!fgets(src, sizeof src, stdin)) return 1;
    chomp(src);
```

```
    printf("Dest: ");
    if (!fgets(dst, sizeof dst, stdin)) return 1;
    chomp(dst);

    FILE *in  = fopen(src, "r");
    if (!in) { perror("fopen src"); return 1; }

    FILE *out = fopen(dst, "w");
    if (!out) { perror("fopen dst"); fclose(in); return 1; }

    char buf[512];
    int lines = 0;
    while (fgets(buf, sizeof buf, in)) {
        fputs(buf, out);
        lines++;
    }

    fclose(in);
    fclose(out);

    printf("Lines: %d\n", lines);
    return 0;
}
```

Сборка и проверка:

```
gcc -std=c17 -Wall -Wextra -Werror -O0 -g main.c -o cp_simple
./cp_simple
# Source: notes.txt
# Dest:   notes_copy.txt
cat notes_copy.txt
```

Критерии зачёта:

- Копирует любой текстовый файл (в разумных пределах длины строки).
- Правильно считает количество строк.
- Обрабатывает ошибки открытия (нет файла / нет прав).

---

10) Частые ошибки и как их избежать

- **Забыли fclose** — данные могут не записаться полностью.
- **Пишете в файл без проверки fopen** — всегда проверяйте if (!f).
- **Путаете scanf/printf и fscanf/fprintf** — для файла нужны варианты с буквой f.
- **Игнорируете \n** — если склеиваете строки, не забудьте перевод строки.
- **Считываете строки scanf("%s")** — он обрывает по пробелу; для строк с пробелами используйте fgets.
