// Тема 11: программы для работы с файлами. Вариант: "Личный бюджет".

// Файл будет называться budget.txt. В каждой строке: дата;тип;категория;сумма;комментарий

// Тип: IN (доход) или OUT (расход). Сделано максимально просто.

#include <iostream> // подключаю ввод/вывод в консоль

#include <fstream> // тут работа с файлами (ifstream/ofstream)

#include <string> // строковый тип std::string

#include <vector> // динамический массив (вектор)

#include <map> // ассоциативный массив (словарь категория -> сумма)

#include <limits> // чтобы чистить ввод (numeric\_limits)

#include <algorithm> // для transform

#include <cctype> // для toupper, isspace

using namespace std; // чтобы не писать std:: перед каждым словом

const string FILE\_NAME = "budget.txt"; // имя файла с данными

// --- Вспомогательные утилиты для нормализации ввода ---

// Убираю пробелы по краям

static inline string trim(const string& s) {

size\_t a = 0, b = s.size();

while (a < b && isspace(static\_cast<unsigned char>(s[a]))) ++a;

while (b > a && isspace(static\_cast<unsigned char>(s[b - 1]))) --b;

return s.substr(a, b - a);

}

// Перевожу строку в ВЕРХНИЙ регистр (для сравнения IN/OUT)

static inline string toUpper(string s) {

transform(s.begin(), s.end(), s.begin(),

[](unsigned char c){ return static\_cast<char>(toupper(c)); });

return s;

}

// Структура одной записи бюджета (как маленькая «табличная строка»)

struct Record {

string date; // дата: например 2025-10-16

string kind; // тип: IN или OUT

string category; // категория: Еда/Зарплата/Транспорт и т.п.

double amount; // сумма (положительная)

string note; // произвольный комментарий

};

// Простой разбор строки по разделителю ';'

vector<string> split(const string& s, char sep) { // s — исходная строка, sep — чем режем

vector<string> res; // сюда складываю куски

string cur; // текущий набираемый кусок

for (char c : s) { // прохожусь по всем символам строки

if (c == sep) { // если встретился разделитель

res.push\_back(cur); // добавляю накопленное

cur.clear(); // и начинаю новый кусок

}

else cur.push\_back(c); // иначе просто дописываю символ

}

res.push\_back(cur); // докидываю последний кусок

return res; // возвращаю массив строк

}

// Преобразую запись Record в строку формата "дата;тип;категория;сумма;коммент"

string toLine(const Record& r) {

return r.date + ";" + r.kind + ";" + r.category + ";" +

to\_string(r.amount) + ";" + r.note; // to\_string — перевожу число в текст

}

// Обратная операция: из строки делаю Record. Если что-то не так — верну false.

bool fromLine(const string& line, Record& out) {

auto parts = split(line, ';'); // режу строку на кусочки

if (parts.size() < 5) return false; // если кусков меньше 5 — формат сломан

out.date = trim(parts[0]); // раскладываю по полям

out.kind = toUpper(trim(parts[1])); // НОРМАЛИЗУЮ ТИП -> IN/OUT

out.category = trim(parts[2]);

try {

out.amount = stod(parts[3]); // stod — превращает текст в число (double)

}

catch (...) { return false; } // если не получилось — бросаю false

out.note = parts[4];

return true; // всё хорошо

}

// Пункт меню: добавить одну запись и сохранить её в файл

void addRecord() {

Record r; // создаю пустую запись

cout << "Дата (напр. 2025-10-16): "; // прошу ввести дату

cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // чищу буфер ввода (после ввода числа)

getline(cin, r.date); // читаю строку целиком (с пробелами)

r.date = trim(r.date);

cout << "Тип (IN=доход, OUT=расход): "; // тип операции

getline(cin, r.kind);

r.kind = toUpper(trim(r.kind)); // НОРМАЛИЗАЦИЯ РЕГИСТРА

if (r.kind != "IN" && r.kind != "OUT") { // простая валидация

cout << "Тип должен быть IN или OUT. Запись отменена.\n";

return;

}

cout << "Категория (еда/транспорт/зарплата/...): "; // категория

getline(cin, r.category);

r.category = trim(r.category);

cout << "Сумма: "; // сумма текстом (чтобы потом проверить)

string s;

getline(cin, s);

try { r.amount = stod(s); } // пробую превратить в число

catch (...) { // если не число — вежливо отказываюсь

cout << "Сумма странная, записывать не буду.\n";

return;

}

cout << "Комментарий: "; // комментарий к записи

getline(cin, r.note);

ofstream fout(FILE\_NAME, ios::app); // открываю файл на дозапись (append)

if (!fout.is\_open()) { // если вдруг не открылся

cout << "Не могу открыть файл для записи :(\n";

return;

}

fout << toLine(r) << "\n"; // пишу строку и перевод строки

fout.close(); // закрываю файл (на всякий случай)

cout << "Записал.\n"; // подтверждение для пользователя

}

// Читаю весь файл и возвращаю все записи как вектор

vector<Record> readAll() {

vector<Record> v; // сюда сложу всё, что прочитал

ifstream fin(FILE\_NAME); // открываю файл на чтение

if (!fin.is\_open()) return v; // если файла нет — верну пустой список

string line; // переменная для строки из файла

while (getline(fin, line)) { // читаю построчно

if (line.empty()) continue; // пустые строки пропускаю

Record r; // сюда попробую распарсить

if (fromLine(line, r)) v.push\_back(r); // если формат ок — кладу в вектор

// иначе просто тихо пропускаю битую строку

}

return v; // отдаю все записи

}

// Показать всё, что есть в файле, в аккуратном виде

void showAll() {

auto v = readAll(); // читаю весь файл

if (v.empty()) { // если ничего нет

cout << "Пока записей нет.\n";

return;

}

cout << "=== Все операции ===\n"; // заголовок

int i = 1; // счётчик строк

for (auto& r : v) { // пробегаюсь по всем записям

cout << i++ << ") " // номер

<< r.date << " [" << r.kind << "] "// дата и тип в квадратных скобках

<< r.category << " : " // категория

<< r.amount << " // " // сумма

<< r.note << "\n"; // комментарий

}

}

// Аналитика: суммы доходов/расходов, по категориям, максимальный расход, поиск по дате

void analyze() {

auto v = readAll(); // читаю все записи

cout << "=== Аналитика по файлу '" << FILE\_NAME << "' ===\n";

cout << "Всего операций: " << v.size() << "\n";

double totalIn = 0.0, totalOut = 0.0; // общие суммы доходов/расходов

map<string, double> byCat; // словарь «категория -> сумма расходов»

double maxOut = 0.0; // самый большой расход

string maxOutDescr = "(нет)"; // описание для максимального расхода

for (auto& r : v) { // пробегаюсь по всем записям

// ВАЖНО: r.kind уже нормализован к верхнему регистру в fromLine/addRecord

if (r.kind == "IN") totalIn += r.amount; // доход — прибавляю к totalIn

else if (r.kind == "OUT") { // расход — в отдельную ветку

totalOut += r.amount; // суммирую общий расход

byCat[r.category] += r.amount; // накапливаю по категории

if (r.amount > maxOut) { // проверяю максимум

maxOut = r.amount; // обновляю сумму максимума

maxOutDescr = // собираю текст-описание максимального расхода

r.date + " " + r.category + " (" + r.note + ")";

}

}

}

cout << "Сумма доходов: " << totalIn << "\n"; // печатаю итоги

cout << "Сумма расходов: " << totalOut << "\n";

cout << "Баланс (IN-OUT): " << (totalIn - totalOut) << "\n"; // баланс = доходы - расходы

if (!byCat.empty()) { // если есть расходы по категориям

cout << "\nРасходы по категориям:\n";

for (auto& kv : byCat) { // kv.first — категория, kv.second — сумма

cout << " - " << kv.first << ": " << kv.second << "\n";

}

}

cout << "Самый большой расход: " << maxOut // печатаю максимум

<< " -> " << maxOutDescr << "\n";

// Небольшой поиск по дате: можно ввести целиком 2025-10-16 или кусок 2025-10

cout << "\nПоиск по дате (введите часть даты или Enter, чтобы пропустить): ";

string needle; // сюда прочитаю, что ищем

cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // чищу буфер перед getline

getline(cin, needle); // читаю строку поиска

needle = trim(needle);

if (!needle.empty()) { // если что-то ввели

int cnt = 0; // счётчик найденных строк

for (auto& r : v) { // бегу по всем записям

if (r.date.find(needle) != string::npos) { // если подстрока нашлась в дате

cout << r.date << " [" << r.kind << "] " // печатаю найденную запись

<< r.category << " : " << r.amount

<< " // " << r.note << "\n";

cnt++;

}

}

if (cnt == 0) // если ничего не нашли

cout << "Ничего не нашёл по дате \"" << needle << "\"\n";

}

}

// Убедиться, что файл существует: если нет — создать пустой

void ensureFile() {

ifstream fin(FILE\_NAME); // пробую открыть на чтение

if (!fin.good()) { // если не получилось (нет файла)

ofstream fout(FILE\_NAME); // просто создаю пустой файл

// здесь ничего не пишу — файл сам появится при закрытии

}

}

// Точка входа в программу

int main() {

setlocale(0, ""); // чтобы русские буквы нормально печатались

ensureFile(); // создаю файл, если его ещё нет

while (true) { // бесконечный цикл меню, пока не выберут «0»

cout << "\n==== Личный бюджет ====\n"; // шапка меню

cout << "1. Добавить операцию (запись)\n"; // пункт 1

cout << "2. Показать все (чтение)\n"; // пункт 2

cout << "3. Анализ (доход/расход/категории/поиск)\n"; // пункт 3

cout << "0. Выход\n"; // выход

cout << "Ваш выбор: "; // приглашение ввода

int c; // сюда читаю номер пункта

if (!(cin >> c)) { // если ввели не число

cin.clear(); // сбрасываю ошибку

cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // вычищаю мусор из ввода

cout << "Нужно число.\н"; // подсказка пользователю

continue; // заново показываю меню

}

if (c == 1) addRecord(); // 1 — добавление записи

else if (c == 2) showAll(); // 2 — показать все записи

else if (c == 3) analyze(); // 3 — аналитика

else if (c == 0) { // 0 — выход из программы

cout << "Пока!\n";

break; // выходим из while(true)

}

else cout << "Нет такого пункта.\n"; // любое другое число — ошибка меню

}

return 0; // нормальное завершение программы

}