**📄 Пояснительная записка**

**по курсовой работе ,по дисциплине:**  
«Программирование и алгоритмизация»

**Тема:**  
Создание программы для работы с файлами (чтение, запись, анализ)  
**Вариант:** «Личный бюджет»

**1. Цель работы**

Целью курсовой работы является разработка программы на языке **C++**, которая позволяет пользователю вести учёт личных доходов и расходов, сохраняя данные в текстовый файл.  
В программе реализованы основные операции:

* добавление записей (запись в файл);
* просмотр всех операций (чтение из файла);
* анализ данных (подсчёт итогов, поиск, статистика).

**2. Назначение программы**

Программа «Личный бюджет» предназначена для ведения простого финансового учёта.  
Пользователь может записывать свои доходы и расходы в файл budget.txt, указывая дату, тип операции (IN — доход, OUT — расход), категорию и сумму.

Данные сохраняются между запусками программы, то есть пользователь может постепенно накапливать записи и анализировать общие суммы и категории расходов.

**3. Структура программы**

Программа состоит из следующих основных частей:

1. **Подключение библиотек и объявление структуры Record**, которая описывает одну запись (дату, тип, категорию, сумму и комментарий).
2. **Функции для чтения и записи данных в файл** — через стандартные потоки ifstream и ofstream.
3. **Меню взаимодействия с пользователем**, позволяющее выбирать действия.

**4. Описание основных функций**

**4.1 Структура данных**

struct Record {

string date; // дата операции

string kind; // IN или OUT

string category; // категория расхода/дохода

double amount; // сумма

string note; // комментарий

};

Данная структура используется для хранения одной записи бюджета. Каждая строка в файле соответствует одному объекту Record.

**4.2 Функция addRecord() — добавление новой записи**

void addRecord() {

Record r;

cout << "Дата (напр. 2025-10-16): ";

getline(cin, r.date);

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

getline(cin, r.kind);

cout << "Категория: ";

getline(cin, r.category);

cout << "Сумма: ";

string s; getline(cin, s);

r.amount = stod(s);

cout << "Комментарий: ";

getline(cin, r.note);

ofstream fout(FILE\_NAME, ios::app);

fout << toLine(r) << "\n";

fout.close();

}

Эта функция обеспечивает **запись данных** в текстовый файл budget.txt.  
Каждая новая строка добавляется в конец файла, чтобы сохранить историю всех операций.

**4.3 Функция readAll() — чтение данных из файла**

vector<Record> readAll() {

vector<Record> v;

ifstream fin(FILE\_NAME);

string line;

while (getline(fin, line)) {

if (line.empty()) continue;

Record r;

if (fromLine(line, r)) v.push\_back(r);

}

return v;

}

Функция открывает файл для чтения и построчно загружает все операции в вектор.  
Если файл отсутствует, создаётся пустой (ensureFile()), чтобы избежать ошибок.

**4.4 Функция showAll() — вывод всех записей**

void showAll() {

auto v = readAll();

if (v.empty()) { cout << "Пока записей нет.\n"; return; }

cout << "=== Все операции ===\n";

int i = 1;

for (auto& r : v) {

cout << i++ << ") " << r.date << " [" << r.kind << "] "

<< r.category << " : " << r.amount << " // " << r.note << "\n";

}

}

Отвечает за **чтение** и отображение всех операций.  
Каждая строка выводится с номером, датой, типом, категорией и суммой.

**4.5 Функция analyze() — анализ и поиск**

void analyze() {

auto v = readAll();

double totalIn = 0.0, totalOut = 0.0;

map<string, double> byCat;

double maxOut = 0.0; string maxOutDescr = "(нет)";

for (auto& r : v) {

if (r.kind == "IN") totalIn += r.amount;

else if (r.kind == "OUT") {

totalOut += r.amount;

byCat[r.category] += r.amount;

if (r.amount > maxOut) {

maxOut = r.amount;

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

}

}

}

cout << "Сумма доходов: " << totalIn << "\n";

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

cout << "Баланс: " << (totalIn - totalOut) << "\n";

}

Эта функция выполняет **анализ данных**, подсчитывая общую сумму доходов и расходов, определяя баланс и категорию с наибольшим расходом.  
Также предусмотрен **поиск по дате**, позволяющий отфильтровать операции за конкретный день или месяц.

**4.6 Главное меню**

int main() {

setlocale(0, "");

ensureFile();

while (true) {

cout << "\n==== Личный бюджет ====\n";

cout << "1. Добавить операцию\n";

cout << "2. Показать все\n";

cout << "3. Анализ\n";

cout << "0. Выход\n";

cout << "Ваш выбор: ";

int c;

if (!(cin >> c)) { ... }

if (c == 1) addRecord();

else if (c == 2) showAll();

else if (c == 3) analyze();

else if (c == 0) break;

}

}

Главная функция реализует **меню пользователя**.  
В зависимости от выбора выполняются нужные действия: запись, чтение, анализ или выход.

**5. Формат данных файла**

Файл budget.txt содержит строки вида:

2025-10-15;IN;Зарплата;25000;Основная работа

2025-10-16;OUT;Еда;500;Обед в кафе

2025-10-17;OUT;Транспорт;100;Автобус

Разделитель между полями — ;.  
Такой формат легко читается и позволяет при необходимости обработать файл в Excel.

**6. Пример работы программы**

После запуска отображается меню:

==== Личный бюджет ====

1. Добавить операцию (запись)

2. Показать все (чтение)

3. Анализ (доход/расход/категории/поиск)

0. Выход

Ваш выбор:

Пример вывода анализа:

=== Аналитика по файлу 'budget.txt' ===

Всего операций: 3

Сумма доходов: 25000

Сумма расходов: 600

Баланс (IN-OUT): 24400

Расходы по категориям:

- Еда: 500

- Транспорт: 100

Самый большой расход: 500 -> 2025-10-16 Еда (Обед в кафе)

**7. Вывод**

В ходе выполнения курсовой работы была разработана программа **«Личный бюджет»** на языке **C++**, демонстрирующая практическое применение **работы с файлами**.  
Реализованы функции **чтения**, **записи**, **анализа**, а также **поиск данных** по дате.  
Программа успешно компилируется, корректно работает с файлами и может быть использована для учёта личных финансов.