

ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

доцент

должность, уч. степень, звание

подпись, дата

В. А. Кузнецов

инициалы, фамилия

ОТЧЕТ О ДОПОЛНИТЕЛЬНОМ ЗАДАНИИ № 2

по курсу:

ОСНОВЫ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №

4326

подпись, дата

Г. С. Томчук

инициалы, фамилия

Санкт-Петербург 2024

СОДЕРЖАНИЕ

1 Постановка задачи.....	3
2 Полное описание реализованных функций.....	4
2.1 readParticipants(const std::string &filename).....	4
2.2 readExpenses(const std::string &filename).....	4
2.3 calculateTotalExpenses(const std::vector<std::string> &participants, const std::vector<Expense> &expenses).....	4
2.4 calculateTotalDebts(const std::vector<std::string> &participants, const std::vector<Expense> &expenses).....	4
2.5 calculateBalances(const std::vector<std::string> &participants, const std::vector<Expense> &expenses, const std::map<std::string, double> &totalDebts)	4
2.6 calculateTransactions(const std::map<std::string, double> &balances).....	4
2.7 printTotalExpenses(const std::map<std::string, double> &totalExpenses).....	4
2.8 printTotalDebts(const std::map<std::string, double> &totalDebts).....	4
2.9 printBalances(const std::map<std::string, double> &balances).....	5
2.10 printTransactions(const std::vector<std::tuple<std::string, std::string, double>> &transactions).....	5
3 Листинг программы.....	6
4 Результаты тестирования программы.....	10

1 Постановка задачи

Исходные данные задачи

1. Дано количество участников организации и перечень их имен. Имена не повторяются.

2. Каждый участник осуществляет некоторые покупки и указывает свои расходы в формате:

Имя : Сумма расходов / Имена участников, не включенных в трату, через запятую

Yuri : 4502.43 / Evgenii, Maria

3. Имена участников, не включенных в трату, являются не обязательным полем. По умолчанию трата распределяется равномерно среди всех участников.

4. Перечень текстовых строк с расходами (пункт 2) также является исходными данными. Способ хранения этих данных: файл.

Задачи для решения

1. Построить структурированную таблицу с суммарными расходами каждого участника. Рассчитать для каждого участника сумму, которую он должен был внести в соответствии с общими затратами.

2. Построить структурированную таблицу транзакций между участниками для того, чтобы компенсировать каждому участнику лишние расходы.

2 Полное описание реализованных функций

2.1 readParticipants(const std::string &filename)

Функция readParticipants считывает список участников организации из файла и возвращает вектор их имен.

2.2 readExpenses(const std::string &filename)

Функция readExpenses считывает расходы участников из файла и возвращает вектор структур Expense, содержащий информацию о каждом расходе.

2.3 calculateTotalExpenses(const std::vector<std::string> &participants, const std::vector<Expense> &expenses) (ПУНКТ 1)

Функция calculateTotalExpenses вычисляет общие расходы каждого участника путем суммирования всех его расходов.

2.4 calculateTotalDebts(const std::vector<std::string> &participants, const std::vector<Expense> &expenses) (ПУНКТ 1)

Функция calculateTotalDebts вычисляет общий долг каждого участника, учитывая его расходы и распределение платежей между участниками.

2.5 calculateBalances(const std::vector<std::string> &participants, const std::vector<Expense> &expenses, const std::map<std::string, double> &totalDebts) (ПУНКТ 2)

Функция calculateBalances вычисляет балансы каждого участника, учитывая их общие расходы и долги.

2.6 calculateTransactions(const std::map<std::string, double> &balances) (ПУНКТ 2)

Функция calculateTransactions вычисляет транзакции между участниками для компенсации лишних расходов.

2.7 printTotalExpenses(const std::map<std::string, double> &totalExpenses)

Функция printTotalExpenses выводит на экран общие расходы каждого участника.

2.8 printTotalDebts(const std::map<std::string, double> &totalDebts)

Функция printTotalDebts выводит на экран общие долги каждого

участника.

2.9 printBalances(const std::map<std::string, double> &balances)

Функция printBalances выводит на экран балансы каждого участника.

2.10 printTransactions(const std::vector<std::tuple<std::string, std::string, double>> &transactions)

Функция printTransactions выводит на экран транзакции между участниками для компенсации лишних расходов.

3 Листинг программы

Листинг 1

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <map>
#include <set>
#include <iomanip>
#include <stdexcept>
#include <tuple>

struct Expense {
    std::string name;
    double amount;
    std::set<std::string> excluded;
};

std::vector<std::string> readParticipants(const std::string &filename) {
    std::ifstream file(filename);
    std::vector<std::string> participants;
    std::string name;
    while (std::getline(file, name)) {
        participants.push_back(name);
    }
    return participants;
}

std::vector<Expense> readExpenses(const std::string &filename) {
    std::ifstream file(filename);
    std::vector<Expense> expenses;
    std::string line;
    while (std::getline(file, line)) {
        std::istringstream iss(line);
        Expense expense;
        std::getline(iss, expense.name, ':');
        iss >> expense.amount;
        std::string excluded;
        if (iss >> excluded) {
            iss >> excluded;
            std::istringstream excl(excluded);
            std::string name;
            while (std::getline(excl, name, ',')) {
                expense.excluded.insert(name);
            }
        }
        expenses.push_back(expense);
    }
    return expenses;
}

std::map<std::string, double>
calculateTotalExpenses(const std::vector<std::string> &participants, const
std::vector<Expense> &expenses) {
    std::map<std::string, double> totalExpenses;
    for (const auto &expense : expenses) {
        totalExpenses[expense.name] += expense.amount;
    }
}
```

```

    }
    return totalExpenses;
}

std::map<std::string, double>
calculateTotalDebts(const std::vector<std::string> &participants, const
std::vector<Expense> &expenses) {
    std::map<std::string, double> totalDebt;
    for (const auto &participant : participants) {
        totalDebt[participant] = 0.0;
    }

    for (const auto &expense : expenses) {
        double sharedAmount = expense.amount / (participants.size() -
expense.excluded.size());
        for (const auto &participant : participants) {
            if (expense.excluded.find(participant) == expense.excluded.end())
{
                totalDebt[participant] += sharedAmount;
            }
        }
    }

    return totalDebt;
}

std::map<std::string, double>
calculateBalances(const std::vector<std::string> &participants, const
std::vector<Expense> &expenses,
                 const std::map<std::string, double> &totalDebts) {
    std::map<std::string, double> balances;

    for (const auto &participant : participants) {
        balances[participant] = -totalDebts.at(participant);
    }

    for (const auto &expense : expenses) {
        balances[expense.name] += expense.amount;
    }

    return balances;
}

std::vector<std::tuple<std::string, std::string, double>>
calculateTransactions(const std::map<std::string, double> &balances) {
    std::vector<std::tuple<std::string, std::string, double>> transactions;
    std::multimap<double, std::string> creditors;
    std::multimap<double, std::string> debtors;

    for (const auto&[name, balance] : balances) {
        if (balance > 0) {
            creditors.emplace(balance, name);
        } else if (balance < 0) {
            debtors.emplace(-balance, name);
        }
    }

    while (!creditors.empty() && !debtors.empty()) {

```

```

    auto credit = creditors.begin();
    auto debit = debtors.begin();

    double amount = std::min(credit->first, debit->first);
    transactions.emplace_back(debit->second, credit->second, amount);

    if (credit->first > amount) {
        creditors.emplace(credit->first - amount, credit->second);
    }
    creditors.erase(credit);

    if (debit->first > amount) {
        debtors.emplace(debit->first - amount, debit->second);
    }
    debtors.erase(debit);
}

return transactions;
}

void printTotalExpenses(const std::map<std::string, double> &totalExpenses) {
    std::cout << "Потратили всего:\n";
    for (const auto &entry : totalExpenses) {
        std::cout << entry.first << ": " << std::fixed << std::setprecision(2)
    << entry.second << "\n";
    }
}

void printTotalDebts(const std::map<std::string, double> &totalDebts) {
    std::cout << "\nДолжны всего:\n";
    for (const auto &entry : totalDebts) {
        std::cout << entry.first << ": " << std::fixed << std::setprecision(2)
    << entry.second << "\n";
    }
}

void printBalances(const std::map<std::string, double> &balances) {
    std::cout << "\nБалансы:\n";
    for (const auto &entry : balances) {
        std::cout << entry.first << ": " << std::fixed << std::setprecision(2)
    << entry.second << "\n";
    }
}

void printTransactions(const std::vector<std::tuple<std::string, std::string,
double>> &transactions) {
    std::cout << "\nТранзакции для компенсаций:\n";
    for (const auto&[debtor, creditor, amount] : transactions) {
        std::cout << debtor << " → " << creditor << ": " << std::fixed <<
std::setprecision(2) << amount << "\n";
    }
}

int main() {
    std::vector<std::string> participants = readParticipants(
        "/home/grigorijtomczuk/Desktop/suai/op/extra2/participants.txt");
    std::vector<Expense> expenses =
readExpenses("/home/grigorijtomczuk/Desktop/suai/op/extra2/expenses.txt");

```



```

    std::map<std::string, double> totalExpenses =
calculateTotalExpenses(participants, expenses);
    printTotalExpenses(totalExpenses);

    std::map<std::string, double> totalDebts =
calculateTotalDebts(participants, expenses);
    printTotalDebts(totalDebts);

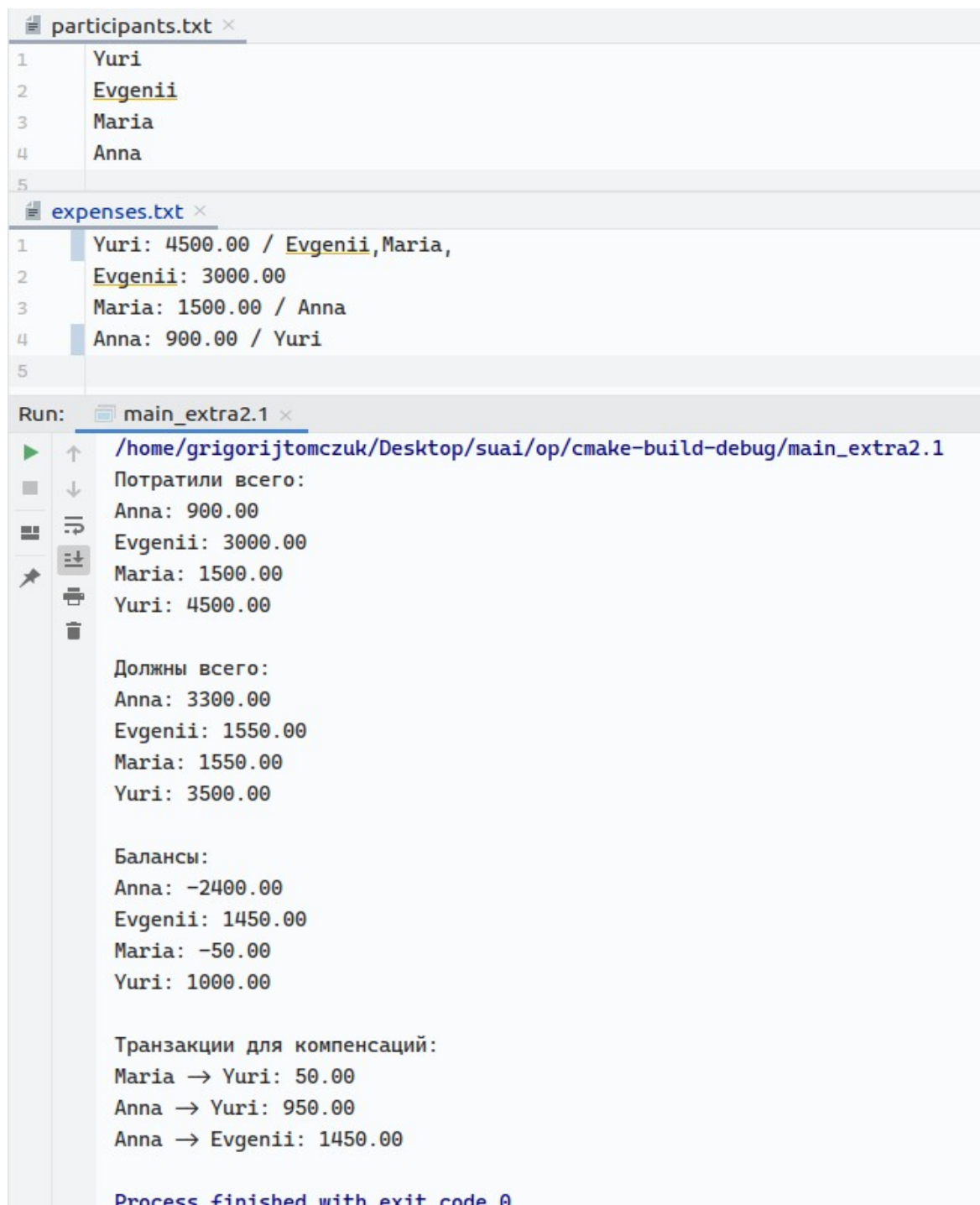
    std::map<std::string, double> balances = calculateBalances(participants,
expenses, totalDebts);
    printBalances(balances);

    std::vector<std::tuple<std::string, std::string, double>> transactions =
calculateTransactions(balances);
    printTransactions(transactions);

    return 0;
}

```

4 Результаты тестирования программы



The screenshot displays a code editor with three tabs: `participants.txt`, `expenses.txt`, and `Run: main_extra2.1`. The `participants.txt` tab shows a list of names: Yuri, Evgenii, Maria, and Anna. The `expenses.txt` tab shows transactions: Yuri: 4500.00 / Evgenii, Maria; Evgenii: 3000.00; Maria: 1500.00 / Anna; and Anna: 900.00 / Yuri. The `Run: main_extra2.1` tab shows the program's output, which includes the total expenses, balances, and transactions for compensation.

```
participants.txt ×
1 Yuri
2 Evgenii
3 Maria
4 Anna
5

expenses.txt ×
1 Yuri: 4500.00 / Evgenii, Maria,
2 Evgenii: 3000.00
3 Maria: 1500.00 / Anna
4 Anna: 900.00 / Yuri
5

Run: main_extra2.1 ×
/home/grigorijtomczuk/Desktop/suai/op/cmake-build-debug/main_extra2.1
Потратили всего:
Anna: 900.00
Evgenii: 3000.00
Maria: 1500.00
Yuri: 4500.00

Должны всего:
Anna: 3300.00
Evgenii: 1550.00
Maria: 1550.00
Yuri: 3500.00

Балансы:
Anna: -2400.00
Evgenii: 1450.00
Maria: -50.00
Yuri: 1000.00

Транзакции для компенсаций:
Maria → Yuri: 50.00
Anna → Yuri: 950.00
Anna → Evgenii: 1450.00

Process finished with exit code 0
```

Рисунок 1

```
participants.txt ×
1 Yuri
2 Evgenii
3 Maria
4 Anna

expenses.txt ×
1 Yuri: 1500.00 / Evgenii, Maria, Yuri,
2 Maria: 500.00 / Anna
3 Evgenii: 4000.00
4 Maria: 700.00 / Evgenii
5 Yuri: 600.00 / Anna, Maria,
6 Anna: 1200.00 / Yuri
7 Maria: 1000.00
8

Run: main_extra2.1 ×
↑ Потратили всего:
↓ Anna: 1200.00
⇅ Evgenii: 4000.00
⇅ Maria: 2200.00
⇅ Yuri: 2100.00
↓
Должны всего:
Anna: 3383.33
Evgenii: 2116.67
Maria: 2050.00
Yuri: 1950.00

Балансы:
Anna: -2183.33
Evgenii: 1883.33
Maria: 150.00
Yuri: 150.00

Транзакции для компенсаций:
Anna → Maria: 150.00
Anna → Yuri: 150.00
Anna → Evgenii: 1883.33
```

Рисунок 2