ГУАП

КАФЕДРА № 42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент |  |  |  | В. А. Кузнецов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ДОПОЛНИТЕЛЬНОМ ЗАДАНИИ № 2 |
|  |
| по курсу: |
| ОСНОВЫ ПРОГРАММИРОВАНИЯ |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4326 |  |  |  | Г. С. Томчук |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#__RefHeading___Toc801_316780651)

[2 Полное описание реализованных функций 4](#__RefHeading___Toc807_316780651)

[2.1 readParticipants(const std::string &filename) 4](#__RefHeading___Toc3907_3112333877)

[2.2 readExpenses(const std::string &filename) 4](#__RefHeading___Toc3913_3112333877)

[2.3 calculateTotalExpenses(const std::vector<std::string> &participants, const std::vector<Expense> &expenses) 4](#__RefHeading___Toc3913_31123338771)

[2.4 calculateTotalDebts(const std::vector<std::string> &participants, const std::vector<Expense> &expenses) 4](#__RefHeading___Toc3823_1590905865)

[2.5 calculateBalances(const std::vector<std::string> &participants, const std::vector<Expense> &expenses, const std::map<std::string, double> &totalDebts) 4](#__RefHeading___Toc3821_1590905865)

[2.6 calculateTransactions(const std::map<std::string, double> &balances) 4](#__RefHeading___Toc3819_1590905865)

[2.7 printTotalExpenses(const std::map<std::string, double> &totalExpenses) 4](#__RefHeading___Toc3817_1590905865)

[2.8 printTotalDebts(const std::map<std::string, double> &totalDebts) 4](#__RefHeading___Toc3815_1590905865)

[2.9 printBalances(const std::map<std::string, double> &balances) 5](#__RefHeading___Toc3813_1590905865)

[2.10 printTransactions(const std::vector<std::tuple<std::string, std::string, double>> &transactions) 5](#__RefHeading___Toc3811_1590905865)

[3 Листинг программы 6](#__RefHeading___Toc805_316780651)

[4 Результаты тестирования программы 10](#__RefHeading___Toc803_316780651)

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.

1. Полное описание реализованных функций
   1. readParticipants(const std::string &filename)

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

* 1. readExpenses(const std::string &filename)

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

* 1. 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.

* 1. 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.

* 1. 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.

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

Функция calculateTransactions вычисляет транзакции между участниками для компенсации лишних расходов.

* 1. printTotalExpenses(const std::map<std::string, double> &totalExpenses)

Функция printTotalExpenses выводит на экран общие расходы каждого участника.

* 1. printTotalDebts(const std::map<std::string, double> &totalDebts)

Функция printTotalDebts выводит на экран общие долги каждого участника.

* 1. printBalances(const std::map<std::string, double> &balances)

Функция printBalances выводит на экран балансы каждого участника.

* 1. printTransactions(const std::vector<std::tuple<std::string, std::string, double>> &transactions)

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

1. 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;  
}

1. Результаты тестирования программы

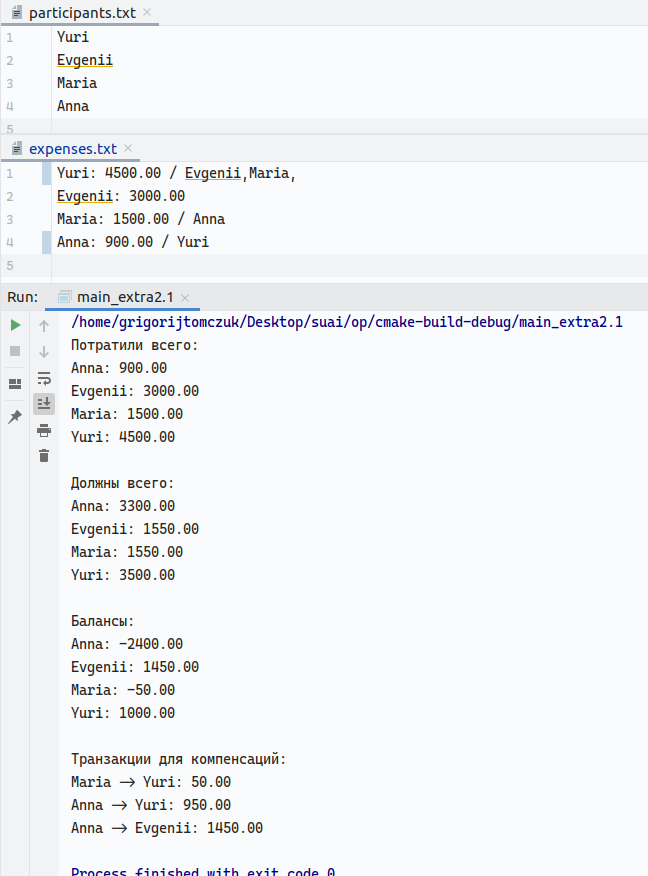


Рисунок 1

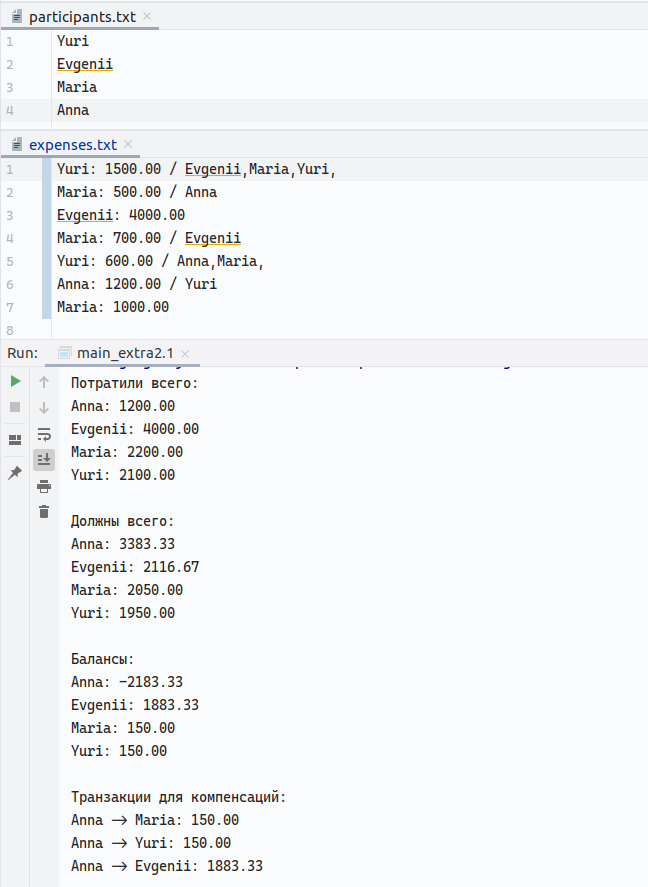


Рисунок 2