Министерство образования РФ

Пермский национальный исследовательский политехнический университет

Кафедра ИТАС

ЛАБОРАТОРНАЯ РАБОТА № 10

ПО ИНФОРМАТИКЕ ЗА I СЕМЕСТР

Вариант 5

|  |
| --- |
| Выполнил студент:  Отинов Иван Юрьевич  Группа РИС-20-1бз  Шифр 20-ЭТФ-635  Кафедра ИТАС:  Полякова Ольга Андреевна |

ПЕРМЬ 2020

# Лабораторная работа №10 "Хранение данных на внешних носителях”

**Вариант №5**

**Задача:**

1. Написать приложение для работы с простой базой данных, хранящей информацию об объекте на внешнем носителе. Приложение должно выполнять следующие функции:
2. Создание базы данных, содержащей записи указанного формата.
3. Просмотр базы данных.
4. Удаление элементов из базы данных (по ключу/ по номеру).
5. Корректировка элементов в базе данных (по ключу / по номеру).
6. Добавление элементов в базу данных (в начало / в конец/ с заданным номером).
7. Выполнение задания, указанного в варианте.

**Вариант задания:**

5. Типизированный файл содержит данные о клиентах банка, получивших кредит (ФИО клиента, сумма кредита, вид кредита, срок, на который выдан кредит). Клиент может взять несколько кредитов.

* 1. Программа должна
     1. Добавлять, удалять, корректировать, позволять просматривать записи файлов.
  2. Выполнять:
     1. вывод списка клиентов, получивших указанный пользователем вид кредита;
     2. определение клиентов, взявших самый большой суммарный кредит.

**Текст программы**

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

using namespace std;

struct Credit {

string name;

string kindOfCredit;

int creditSum;

int term;

int flag;

};

void createDb(char\* databaseName);

void loadDb(char\* databaseName, vector<Credit> &credits);

void insertRecord(vector<Credit>& credits);

Credit \*deleteRecord(vector<Credit>& credits, string name);

void updateRecord(vector<Credit>& credits, string name);

void printDb(vector<Credit>& credits);

void cancelDelete(vector<Credit>& books, Credit\* credit);

void findByKindOfCredit(vector<Credit>& credits, string kindOfCredit);

void findMaxCredit(vector<Credit>& credits);

void save(vector<Credit>& credits, char\* fileName);

int main() {

char databaseName[] = "D:\\db.txt";

vector <Credit> credits;

Credit\* lastDeleted = NULL;

int dbWasLoaded = 0;

int action = -1;

while (action) {

cout << "Menu:\n 1.Load database\n 2.Insert\n 3.Delete"

<< "\n 4.Cancel last delete\n 5.Update\n 6.Print\n 7.Find max credit"

<< "\n 8.Find by kind of credit\n 9.Save\n 0.Exit";

cout << "\n\nChoose you action: ";

cin >> action;

string helpStr, name1, name2, name3;

switch (action) {

case 0:

break;

case 1:

if (dbWasLoaded) {

cout << "\nDatabase was already loaded\n\n";

break;

}

dbWasLoaded = 1;

loadDb(databaseName, credits);

break;

case 2:

insertRecord(credits);

break;

case 3:

cout << "\nInput name1: ";

cin >> name1;

cout << "Input name2: ";

cin >> name2;

cout << "Input name3: ";

cin >> name3;

lastDeleted = deleteRecord(credits, name1 + " " + name2 + " " + name3);

break;

case 4:

cancelDelete(credits, lastDeleted);

break;

case 5:

cout << "\nInput name1: ";

cin >> name1;

cout << "Input name2: ";

cin >> name2;

cout << "Input name3: ";

cin >> name3;

updateRecord(credits, name1 + " " + name2 + " " + name3);

break;

case 6:

printDb(credits);

break;

case 7:

findMaxCredit(credits);

break;

case 8:

cout << "\nInput kind of credit: ";

cin >> helpStr;

findByKindOfCredit(credits, helpStr);

break;

case 9:

save(credits, databaseName);

break;

default:

cout << "\nInvalid action";

break;

}

}

}

void loadDb(char\* databaseName, vector<Credit> &credits) {

ifstream in(databaseName);

string line, name, kindOfCredit, creditSum, term;

int count = 0;

if (in.is\_open()) {

while (getline(in, line)) {

if (count == 0) {

name = line;

count++;

} else if (count == 1) {

creditSum = line;

count++;

} else if (count == 2) {

kindOfCredit = line;

count++;

} else {

term = line;

Credit\* credit = new Credit();

credit->name = name;

credit->kindOfCredit = kindOfCredit;

credit->creditSum = stoi(creditSum);

credit->term = stoi(term);

credit->flag = 0;

credits.push\_back(\*credit);

count = 0;

}

}

}

cout << "\nDb was loaded\n\n";

in.close();

}

void printDb(vector<Credit>& credits) {

cout << "\ndb.txt";

cout << "\n---------------------------------------------";

for (int i = 0; i < credits.size(); i++) {

cout << "\nCredit { " << credits.at(i).name << " : " << credits.at(i).creditSum <<

" : " << credits.at(i).kindOfCredit << " : " << credits.at(i).term << " }";

}

cout << "\n---------------------------------------------\n\n";

}

void insertRecord(vector<Credit>& credits) {

Credit\* credit = new Credit();

string name1, name2, name3;

cout << "\nCreate new record\nInput name1: ";

cin >> name1;

cout << "Input name2: ";

cin >> name2;

cout << "Input name3: ";

cin >> name3;

credit->name = name1 + " " + name2 + " " + name3;

cout << "Input creditSum: ";

cin >> credit->creditSum;

cout << "Input kindOfCredit: ";

cin >> credit->kindOfCredit;

cout << "Input term: ";

cin >> credit->term;

credit->flag = 1;

cout << "\nRecord was added\n\n";

credits.push\_back(\*credit);

}

Credit \*deleteRecord(vector<Credit>& credits, string name) {

Credit\* toDelete = new Credit();

for (int i = 0; i < credits.size(); i++) {

if (credits.at(i).name == name) {

toDelete->name = credits.at(i).name;

toDelete->creditSum = credits.at(i).creditSum;

toDelete->kindOfCredit = credits.at(i).kindOfCredit;

toDelete->term = credits.at(i).term;

cout << "\nDelete credit: { " << credits.at(i).name << " : " << credits.at(i).creditSum <<

" : " << credits.at(i).kindOfCredit << " : " << credits.at(i).term << " }\n\n";

credits.erase(credits.begin() + i);

}

}

return toDelete;

}

void cancelDelete(vector<Credit>& books, Credit\* credit) {

books.push\_back(\*credit);

cout << "\n\nCancel last delete\n\n";

}

void updateRecord(vector<Credit>& credits, string name) {

Credit\* credit = NULL;

for (int i = 0; i < credits.size(); i++) {

if (credits.at(i).name == name) {

credit = &credits.at(i);

}

}

if (credit == NULL) {

cout << "Nothing found!";

return;

}

string name1, name2, name3;

cout << "Update credit: { " << credit->name << " : " << credit->creditSum <<

" : " << credit->kindOfCredit << " : " << credit->term << " }";

cout << "\nReplace name1 -> ";

cin >> name1;

cout << "\nReplace name2 -> ";

cin >> name2;

cout << "\nReplace name3 -> ";

cin >> name3;

credit->name = name1 + " " + name2 + " " + name3;

cout << "Replace creditSum -> ";

cin >> credit->creditSum;

cout << "Replace kindOfCredit -> ";

cin >> credit->kindOfCredit;

cout << "Replace term -> ";

cin >> credit->term;

cout << "\n";

}

void findByKindOfCredit(vector<Credit>& credits, string kindOfCredit) {

for (int i = 0; i < credits.size(); i++) {

if (credits.at(i).kindOfCredit == kindOfCredit) {

cout << "\nCredit { " << credits.at(i).name << " : " << credits.at(i).creditSum <<

" : " << credits.at(i).kindOfCredit << " : " << credits.at(i).term << " }\n\n";

}

}

}

void findMaxCredit(vector<Credit>& credits) {

cout << "\nFind max credit...";

int maxCredit = 0;

int index = 0;

for (int i = 0; i < credits.size(); i++) {

if (credits.at(i).creditSum > maxCredit) {

index = i;

}

}

cout << "\nCredit { " << credits.at(index).name << " : " << credits.at(index).creditSum <<

" : " << credits.at(index).kindOfCredit << " : " << credits.at(index).term << " }\n\n";

}

void save(vector<Credit>& credits, char\* fileName) {

ofstream out(fileName, ios::trunc);

if (out.is\_open()) {

for (int i = 0; i < credits.size(); i++) {

out << credits.at(i).name << endl;

out << credits.at(i).creditSum << endl;

out << credits.at(i).kindOfCredit << endl;

out << credits.at(i).term << endl;

}

}

cout << "\nDatabase was saved\n\n";

out.close();

}

**Результаты работы программы :**

