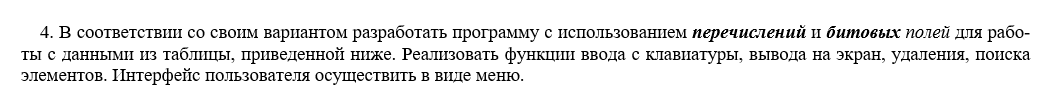
**Варыянт 3**





#include <Windows.h>

#include <iostream>

#include <string>

#include <vector>

#include <unordered\_map>

using namespace std;

struct StudentClass {

unsigned classDigit : 4; // 4 біта для лічбы класа (0-15)

int classSymbol : 8; // 8 біт для сімвала класа (ASCII)

};

enum Lessons { // Пералік для прадметаў

Mathematics = 1,

BelarusianLanguage,

RussianLanguage,

Literature,

History,

Geography,

Physics,

Chemistry,

Biology,

ForeignLanguage,

PhysicalEducation,

Music,

Art,

Technology

};

struct Student {

// Імя

string firstName = "";

// Прозвішча

string lastName = "";

// Імя па бацьку

string middleName = "";

// Ствараем аб'ект grade структуры StudentClass для прадстаўлення класа

struct StudentClass grade;

// Функцыя для ўстаноўкі значэнняў палёў grade структуры StudentClass (для зручнасці ўстаноўкі значэнняў адным дзеяннем)

void setGrade(int digit, char symbol) {

grade.classDigit = digit;

grade.classSymbol = symbol;

}

double GPA = 0; // Сярэдні бал

int size = 0; // Колькасць элементаў (прадметаў, адзнак) у масівах

int\* marks; // Паказальнік на масіў адзнак

Lessons\* lessons; // Паказальнік на масіў прадметаў

// Канструктар для ініцыялізацыі паказальнікаў

Student(int numMarksAndClasses) : size(numMarksAndClasses) {

marks = new int[size];

lessons = new Lessons[size];

}

// Канструктар капіравання

// Паколькі я выкарыстоўваю вектар для захоўвання спісу навучэнцаў, то мне неабходна вызначыць канструктар

// капіравання. Гэта абумоўлена тым, што пры даданні элемента ў вектар з дапамогай push\_back()

// адбываецца капіраванне перададзенага аб'екта. Калі канструктар капіравання не вызначаны,

// будзе выкарыстоўвацца канструктар па змаўчанні, які створыць толькі павярхоўную копію (масівы

// будуць уяўляць сабой спасылкі на арыгінальныя масівы, а не іх незалежныя копіі).

Student(const Student& other) :

firstName(other.firstName),

lastName(other.lastName),

middleName(other.middleName),

grade(other.grade),

GPA(other.GPA),

size(other.size) {

// Вылучэнне новай памяці для капіравання масіваў marks і lessons

marks = new int[size];

lessons = new Lessons[size];

// Капіраванне дадзеных з іншага аб'екта

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

marks[i] = other.marks[i];

lessons[i] = other.lessons[i];

}

}

// Аператар прысвойвання

// Пры выдаленні элементаў з вектара ўсе элементы справа ад выдаляемага зрушваюцца налева

// Пры гэтым выконваецца аперацыя прысвойвання, і каб пазбегнуць праблем з дынамічнай памяццю,

// неабходна відавочна вызначыць аператар прысвойвання

Student& operator=(const Student& other) {

if (this != &other) { // Прадухіляем самапрысвойванне. this - паказальнік на перададзены аб'ект (у які капіруем, які стаіць злева)

// Вызваляем памяць аб'екта, ў які капіруем (бо колькасць прадметаў можа не супадаць з аб'ектам з якога капіруем)

delete[] marks;

delete[] lessons;

// Капіруем дадзеныя з іншага аб'екта

firstName = other.firstName;

lastName = other.lastName;

middleName = other.middleName;

grade = other.grade;

GPA = other.GPA;

size = other.size;

// Вылучаем новую памяць і капіруем масівы

marks = new int[size];

lessons = new Lessons[size];

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

marks[i] = other.marks[i];

lessons[i] = other.lessons[i];

}

}

// Вяртаем копію аб'екта

return \*this;

}

// Дэструктар для вызвалення памяці пры выдаленні аб'ектаў

~Student() {

delete[] lessons;

delete[] marks;

}

// Вылічэнне сярэдняга бала

void calculateGPA() {

double sum = 0;

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

sum += marks[i];

}

GPA = sum / size;

}

};

// Вектар для захоўвання аб'ектаў структуры Student, які ўяўляе сабой спіс вучняў.

vector<Student> students;

// std::unordered\_map - гэта кантэйнер, які прадстаўляе асацыятыўнае адлюстраванне,

// гэта значыць калекцыю пар ключ-значэнне, дзе кожнаму ключу адпавядае значэнне.

// У дадзеным выпадку, ключамі з'яўляюцца элементы пераліку Lessons, а значэннямі - радкі з назвамі прадметаў.

// Выкарыстоўваецца для палягчэння вываду назваў прадметаў

std::unordered\_map<Lessons, std::string> subjects = {

{Mathematics, "Матэматыка"},

{BelarusianLanguage, "Беларуская мова"},

{RussianLanguage, "Руская мова"},

{Literature, "Літаратура"},

{History, "Гісторыя"},

{Geography, "Геаграфія"},

{Physics, "Фізіка"},

{Chemistry, "Хімія"},

{Biology, "Біялогія"},

{ForeignLanguage, "Замежная мова"},

{PhysicalEducation, "Фізічная культура"},

{Music, "Музыка"},

{Art, "Выяўленчае мастацтва"},

{Technology, "Праца"}

};

void addStudent() {

// Атрымліваем ад карыстальніка ўсе звесткі і захоўваем

std::cin.ignore(); // Пасля ўвода варыянта дзеяння ў галоўнай функцыі застаецца ў буферы сімвал новага радка. пазбаўляемся ад яго

cout << "Увядзіце Імя: ";

string firstName;

std::getline(cin, firstName); // Атрымліваем радок

cout << "Увядзіце Прозвішча: ";

string lastName;

std::getline(cin, lastName);

cout << "Увядзеце Імя па бацьку: ";

string middleName;

std::getline(cin, middleName);

cout << "Увядзіце нумар класа (лічба + літара): ";

int classDigit; //Лічба класа

cin >> classDigit;

char classSymbol; // Літара класа

cin >> classSymbol;

cout << "Увядзіце колькасць прадметаў, якія вывучае вучань: ";

int size;

cin >> size;

Student student(size); // ствараем новы аб'ект структуры Student, ініцыялізуючы яго колькасцю прадметаў (будзе выкліканы канструктар)

student.firstName = firstName; // Усталеўваем імя

student.middleName = middleName; // Усталеўваем прозвішча

student.lastName = lastName; // Усталеўваем імя па бацьку

student.setGrade(classDigit, classSymbol); // выклікаем метад setGrade для усталявання класа

std::cout << "Матэматыка - " << 1 << " | "

<< "Беларуская мова - " << 2 << " | "

<< "Руская мова - " << 3 << " | "

<< "Літаратура - " << 4 << " | "

<< "Гісторыя - " << 5 << " | "

<< "Геаграфія - " << 6 << " | "

<< "Фізіка - " << 7 << " | "

<< "Хімія - " << 8 << " | "

<< "Біялогія - " << 9 << " | "

<< "Замежная мова - " << 10 << " | "

<< "Фізічная культура - " << 11 << " | "

<< "Музыка - " << 12 << " | "

<< "Выяўленчае мастацтва - " << 13 << " | "

<< "Праца - " << 14 << std::endl;

for (int i = 0; i < size; i++) { // Увод прадметаў праз нумар прадмету са спісу вышэй

printf("Увядзіце нумар прадмета %d: ", i + 1);

int subject;

cin >> subject;

student.lessons[i] = Lessons(subject); // Канвертуем лік у элемент пераліку

cout << "Увядзіце адзнаку за гэты прадмет: ";

cin >> student.marks[i];

}

student.calculateGPA(); // Выклікаем метад для вылічэння адзнакі

students.push\_back(student); // Захоўваем аб'ект у вектары за дапамогай push\_back()

}

void displayStudents() {

int counter = 1;

for (int j = 0; j < students.size(); j++) { // .size() вяртае бягучую колькасць элементаў у вектары

Student student = students[j]; // перабіраем усіх вучняў

printf("\n-----Вучань нумар %d-----\n", counter++);

//Выводзім дадзеныя вучня

cout << "ПІБ: " + student.lastName + ' ' + student.firstName + ' ' + student.middleName << endl;

cout << "Клас: " << student.grade.classDigit << (char)student.grade.classSymbol << endl;

cout << "Прадметы + адзнака: \n";

for (int i = 0; i < student.size; i++) { // Праз цыкл перабіраю ўсе прадметы і іх адзнакі

cout << subjects[student.lessons[i]] << ' ' << student.marks[i] << endl; // вось тут карыстаюсь unordered\_map для атрымання назвы прадмета ў выгляде радка, а не лічбы

}

cout << "Сярэдні бал: " << student.GPA;

}

}

void deleteStudent() {

cin.ignore();

cout << "Увядзіце прозвішча вучня, якога трэба выдаліць: ";

string lastName;

getline(cin, lastName);

int foundIndex = -1;

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

if (lastName == students[i].lastName) { // Пошук па прозвішчы

foundIndex = i;

break;

}

}

if (foundIndex != -1) { // -1 - пачатковае знач. індэкса, калі яно не змянілася, то вучань не быў знойдзены

students.erase(students.begin() + foundIndex); // з дапамогай erase выдаляю элемент вектара. метад прымае паказальнік на элемент які трэба выдаліць. знахожу яго з дапамогай .begin(), які вяртае паказальнік на пачатак вектара, дадаючы да яго індэкс знойдзенага элемента

cout << "Вучань з прозвішчам " + lastName + " выдалены\n";

}

else {

cout << "Вучань з прозвішчам " + lastName + " не знойдзены\n";

}

}

void search() {

cin.ignore();

cout << "Увядзіце прозвішча вучня, якога трэба выдаліць: ";

string lastName;

getline(cin, lastName);

int foundIndex = -1;

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

if (lastName == students[i].lastName) {

foundIndex = i;

break;

}

}

if (foundIndex != -1) {

cout << "Вучань з прозвішчам " + lastName + " знойдзены:\n";

Student student = students[foundIndex];

cout << "ПІБ: " + student.lastName + ' ' + student.firstName + ' ' + student.middleName << endl;

cout << "Клас: " << student.grade.classDigit << (char)student.grade.classSymbol << endl;

cout << "Прадметы + адзнака: \n";

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

cout << student.lessons[i] << ' ' << student.marks[i] << endl;

}

cout << "Сярэдні бал: " << student.GPA;

}

else {

cout << "Вучань не знойдзены\n";

}

}

int main() {

SetConsoleOutputCP(1251);

int choice = -1;

while (choice!=0) {

cout << "\n1 - увод вучня\n";

cout << "2 - вывад вучняў\n";

cout << "3 - пошук вучня\n";

cout << "4 - выдаліць вучня\n";

cout << "0 - выхад\n";

cout << "Увядзіце ваш выбар: ";

cin >> choice;

switch (choice) {

case 1: {

addStudent();

break;

}

case 2: {

displayStudents();

break;

}

case 3: {

search();

break;

}

case 4: {

deleteStudent();

break;

}

default: {

cout << "Няправільны выбар";

break;

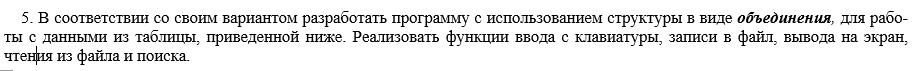
}

}

}

return 0;

}



#include <Windows.h>

#include <iostream>

#include <string>

#include <cstring>

#include <vector>

#include <fstream>

#include <limits>

using namespace std;

// Не прыдумаў нічога лепшага для выкарыстання аб'яднання

// Рашыў што будзе альбо дата здачы, альбо тэрмін, да якога трэба вярнуць заклад. У наступных варыянтах штось больш цікавае

union ReturnDate {

// Гэтае поле, якое паказвае дату здачы прадмета ў заклад

char itemPawningDate[11]; // 2000.00.00

// Дата, да якой трэба вярнуць доўг

char repaymentDeadline[11];

};

// Пералік для ўстаноўкі тыпу даты ў аб'яднанні

enum DateType {

ItemPawningDate,

RepaymentDeadline

};

struct Pawnshop {

//анкетныя дадзеныя кліента

std::string clientSurveyData;

//Найменне тавару

std::string itemName;

//ацэначны кошт

double estimatedValue;

//сума, выдадзеная пад заклад

double loanValue;

//дата

union ReturnDate date;

DateType currentDateType; // Поле для адсочвання тыпу даты

};

std::vector<Pawnshop> pawnshopData; // Вектар для захоўвання звестак ламбарда

//Функцыя дадання кліента

void addClient();

//Функцыя пошуку

void search();

//Функцыя запісу ў файл

void writeToFile(const std::string& fileName);

//Функцыя чытання з файла

void readFromFile(const std::string& fileName);

//Функцыя вываду ўсіх кліентаў

void displayAll();

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

//спачатку счытваем двух кліентаў

addClient();

addClient();

//выводзім іх

displayAll();

//счытваем дадзеныя з файла

readFromFile("out.txt");

//запісваем у файл

writeToFile("out.txt");

//зноў выводзім

cout << "\n\nПасля чытання з файла\n\n";

displayAll();

//шукаем сярод іх

search();

return 0;

}

void writeToFile(const std::string& fileName) {

//адкрываем файл для запісу і правяраем паспяховасць акрыцця

ofstream file(fileName);

if (!file.is\_open() || file.fail()) {

cerr << "Памылка пры адкрыцці файла для запісу\n";

return;

}

for (Pawnshop item : pawnshopData) { //Перабіраем элементы вектара з дапамогай ітэратара, і запісваем значэнні ўсіх палёў аб'екта item структуры Pawnshop у файл праз коску

file << item.clientSurveyData << ",";

file << item.itemName << ',';

file << item.estimatedValue << ',';

file << item.loanValue << ',';

file << item.currentDateType << ':' << item.date.itemPawningDate << '!'; // для апошняга поля - дата, запісваем тып даты і саму дату праз ":". тут можна запісваць любое поле, як itemPawningDate, так і repaymentDeadline, паколькі тып дадзеных і іх памер аднолькавы

}

file.close();

}

void readFromFile(const std::string& fileName) {

ifstream file(fileName); //адкрываем файл для чытання

if (!file.is\_open() || file.fail()) {

cerr << "Памылка пры адкрыцці файла для чытання\n";

return;

}

//першае поле ў файле - анкетныя дадзеныя кліента

string clientSurveyData;

while (getline(file, clientSurveyData, ',')) { //счытваем дадзеныя з файла ў радок clientSurveyData пакуль не сустрэнем сімвал ",". пакуль атрымоўваецца іх палічыць, то ў файле ёсць дадзеныя

Pawnshop item; //ствараем аб'ект структуры, які прадстаўляе бягучага кліента

item.clientSurveyData = clientSurveyData;

getline(file, item.itemName, ','); //счытваем наступныя дадзеныя(назва тавара) адразу ў адпавядае поле

file >> item.estimatedValue; // паколькі тып дадзеных поля estimatedValue - double, то сустрэўшы коску ўвод ліку спыняецца

file.ignore(); // Прапускаем коску пасля ацэначнага кошту

file >> item.loanValue;

file.ignore();

char currentDateType; // сімвал, які прадстаўляе тып даты (DateType)

file.get(currentDateType); //счытваем гэты сімвал

if (currentDateType == '0') { // усталёўваем ў структуры адпаведны тып даты

item.currentDateType = ItemPawningDate;

}

else {

item.currentDateType = RepaymentDeadline;

}

file.ignore(); //прапускаем двукроп'е

std::string fileInput;

getline(file, fileInput, '!');

strncpy\_s(item.date.itemPawningDate, sizeof(item.date.itemPawningDate), fileInput.c\_str(), \_TRUNCATE); // strncpy\_s капіюе радок. будзе скапіравана ў item.date.itemPawningDate sizeof(item.date.itemPawningDate) байт з fileInput. c\_str() вяртае паказальнік на масіў сімвалаў, які прадстаўляе радок. макрас \_TRUNCATE кажа, што калі зыходны радок fileInput больш, чым буфер item.date.itemPawningDate, то адбудзецца абразанне гэтага радка

pawnshopData.push\_back(item); //дадаем элемент у вектар

}

}

void search() {

string itemName;

cout << "Увядзіце назву тавару для пошуку: ";

getline(cin, itemName);

for (Pawnshop item : pawnshopData) { //з дапамогай ітэратара праходзім па вектары

if (item.itemName == itemName) { // калі назва тавару роўна уведзенаму для пошуку, выводзім інфармацыю аб ім

cout << "Анкетныя дадзеныя кліента: " << item.clientSurveyData << endl;

cout << "Назва тавару: " << item.itemName << endl;

cout << "Ацэначны кошт: " << item.estimatedValue << endl;

cout << "Сума, выдадзеная пад заклад: " << item.loanValue << endl;

// Правяраем, які тып даты ўсталяваны

if (item.currentDateType == ItemPawningDate) {

cout << "Дата здачы ў заклад:" << item.date.itemPawningDate << endl;

}

else if (item.currentDateType == RepaymentDeadline) {

cout << "Тэрмін вяртання закладу:" << item.date.repaymentDeadline << endl;

}

return; //выходзім з функцыі

}

}

cout << "Не знойдзена\n";//будзе выканана калі не атрымалася знайсці

}

void addClient() {

Pawnshop client;

cout << "Анкетныя дадзеныя кліента: ";

//cin.ignore(); //прапускаем сімвал новага радка

getline(cin, client.clientSurveyData); // счытваем радок адразу ў поле

cout << "Назва тавару: ";

getline(cin, client.itemName);

std::cout << "Ацэначны кошт: ";

cin >> client.estimatedValue;

std::cout << "Сума, выдадзеная пад заклад: ";

cin >> client.loanValue;

cout << "Усталёўваны дату выдачы закладу або дату, да якой трэба вярнуць доўг?(0/1): ";

int dateType;

cin >> dateType;

//Усталёўваны тып дадзеных праз прывядзенне тыпаў

client.currentDateType = (DateType)dateType;

cout << "Вядзіце дату: ";

cin >> client.date.itemPawningDate;

cin.ignore();

//дадаем кліента ва вектар усіх кліентаў

pawnshopData.push\_back(client);

}

void displayAll() {

int counter = 1;

for (Pawnshop item : pawnshopData) {

//Выводзім інфармацыю аб кожным кліенце

printf\_s("\n-----Кліент нумар %d-----\n", counter++);

cout << "Анкетныя дадзеныя кліента: " << item.clientSurveyData << endl;

cout << "Назва тавару: " << item.itemName << endl;

cout << "Ацэначны кошт: " << item.estimatedValue << endl;

cout << "Сума, выдадзеная пад заклад: " << item.loanValue << endl;

// Правяраем, якая дата ўстаноўлена

if (item.currentDateType == ItemPawningDate) {

cout << "Дата здачы ў заклад:" << item.date.itemPawningDate << endl;

}

else if (item.currentDateType == RepaymentDeadline) {

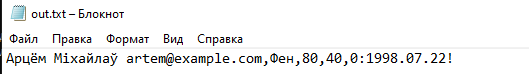
cout << "Тэрмін вяртання закладу:" << item.date.repaymentDeadline << endl;

}

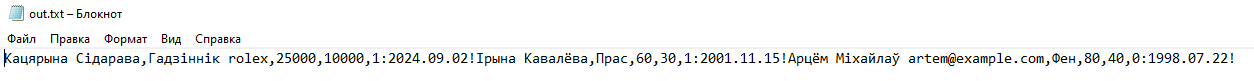
}

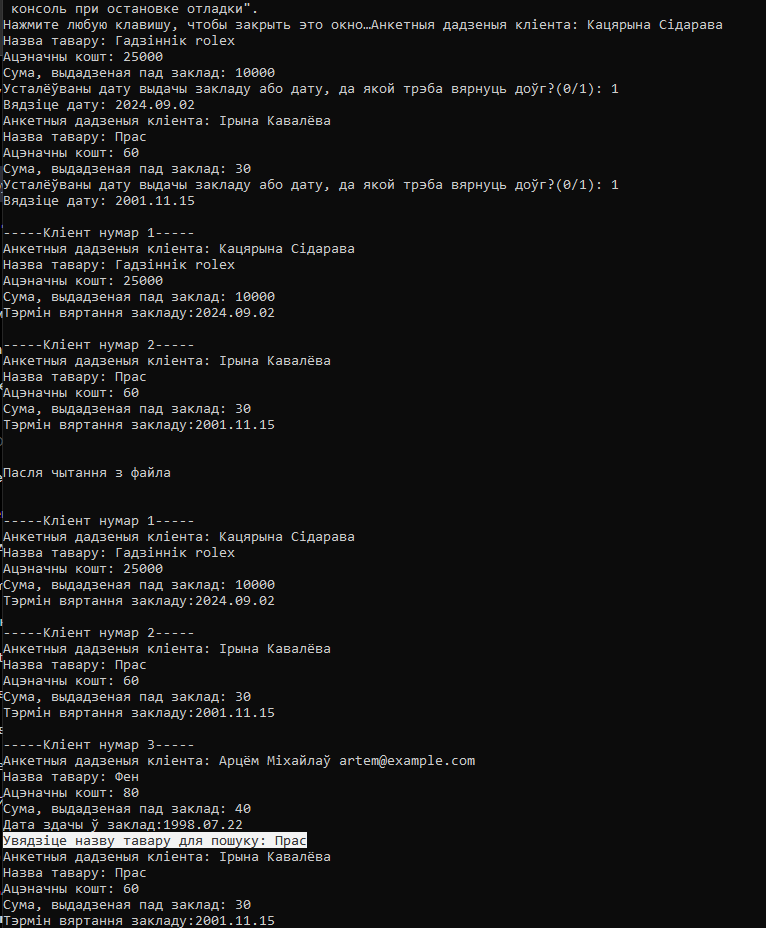
}

Перад запускам

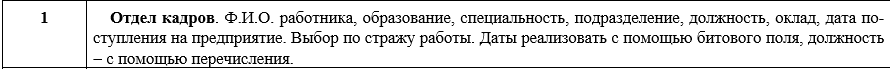


Пасля





**Варыянт 1**



#include <iostream>

#include <vector>

#include <string>

#include "Windows.h"

// Пералік для пасады

enum Position {

MANAGER,

ENGINEER,

ACCOUNTANT,

OTHER

};

// Бітавое поле для даты

struct Date {

unsigned day : 5; // 0-31

unsigned month : 4; // 0-15

unsigned year : 12; // 0-4095

};

// Структура для супрацоўніка

struct Employee {

std::string full\_name;

std::string education;

std::string specialty;

std::string department;

Position position;

double salary;

Date hire\_date;

};

// Функцыя для ўводу супрацоўніка з клавіятуры

Employee inputEmployee() {

Employee employee;

std::cout << "Увядзіце імя: ";

std::getline(std::cin, employee.full\_name); //Атрымліваем поўнае імя ў выгдядзе радка і адразу запісваем у поле

std::cout << "Увядзіце адукацыю: ";

std::getline(std::cin, employee.education);

std::cout << "Увядзіце спецыяльнасць: ";

std::getline(std::cin, employee.specialty);

std::cout << "Увядзіце падраздзяленне: ";

std::getline(std::cin, employee.department);

std::cout << "Увядзіце пасаду (0 - Мэнэджар, 1 - Інжынер, 2 - Бухгалтар, 3 - Іншае): ";

// Атрымліваем нумар пасады і канвертуем у тып пераліку Position

int positionInput;

std::cin >> positionInput;

employee.position = (Position)positionInput;

std::cout << "Увядзіце аклад: ";

std::cin >> employee.salary;

int input;

std::cout << "Увядзіце дзень паступлення на прадпрыемства: ";

std::cin >> input;

employee.hire\_date.day = (unsigned)input; // паколькі проста ўвесці нельга, то спачатку захоўваем у зменную, а потым канвертуем і захоўваем у полі

std::cout << "Увядзіце месяц паступлення на прадпрыемства: ";

std::cin >> input;

employee.hire\_date.month = (unsigned)input;

std::cout << "Увядзіце год паступлення на прадпрыемства: ";

std::cin >> input;

employee.hire\_date.year = (unsigned)input;

// Чысцім буфер уводу, бо пасля ўводу лікаў там застаўся сімвал новага радка

std::cin.ignore();

return employee;

}

// Функцыя для вываду супрацоўніка

void printEmployee(const Employee& employee) {

//выводзім значэнне кожнага поля

std::cout << "Імя: " << employee.full\_name << std::endl;

std::cout << "Адукацыя: " << employee.education << std::endl;

std::cout << "Спецыяльнасць: " << employee.specialty << std::endl;

std::cout << "Падраздзяленне: " << employee.department << std::endl;

std::cout << "Пасада: ";

switch (employee.position) { //для пасады, прадстаўленнай пералікам, выводзім з дапамогай switch

case MANAGER:

std::cout << "Мэнэджар";

break;

case ENGINEER:

std::cout << "Інжынер";

break;

case ACCOUNTANT:

std::cout << "Бухгалтар";

break;

case OTHER:

std::cout << "Іншае";

break;

}

std::cout << std::endl;

std::cout << "Аклад: " << employee.salary << " руб." << std::endl;

std::cout << "Дата паступлення на прадпрыемства: "

<< employee.hire\_date.day << "." << employee.hire\_date.month << "." << employee.hire\_date.year << std::endl;

std::cout << "------------------------------------------- --" << std::endl;

}

// Функцыя для вываду ўсіх супрацоўнікаў у вектары

void printAllEmployees(const std::vector<Employee>& employees) {

std::cout << "Спіс усіх супрацоўнікаў:" << std::endl;

for (const auto& employee : employees) { // з дапамогай ітэратара праходзім па кожнаму элементу ў вектары

printEmployee(employee); // і для кожнага элемента вызываем функцыю вываду

}

}

// Функцыя для выдалення супрацоўніка па імені

void deleteEmployeeByName(std::vector<Employee>& employees, const std::string& name) {

bool flag = false;

auto it = employees.begin(); // auto каб ты дадзеных вызначыў кампілятар самастойна. ствараем паказальнік it на пачатак вектара з дапамогай begin(), які і вяртае паказальнік на пачатак вектара

for (; it != employees.end(); it++) { // end() верне пакальнік на канец вектара. пакуль паказальнік не дайшоў да канца - павялічваем яго знач. на 1

if (it->full\_name == name) { // калі імя супрацоўніка, на якога паказвае паказальнік роўна пераданаму для пошука, то выходзім з цыкла, захоўваючы значэнне паказальніка, які паказвае на элемент, які трэба выдаліць

flag = true; //ставім сцяг на true, каб перадаць, што быў знойдзены супрацоўнік

break;

}

}

if(flag)employees.erase(it); // erase выдаляе элемент вектара, на які паказвае паказальнік

}

// Функцыя для пошуку супрацоўнікаў па стажы працы ў месяцах

std::vector<Employee> findEmployeesByExperience(const std::vector<Employee>& employees, int experienceInMonths) {

std::vector<Employee> result;

// Бягучая дата фіксавана (2024.02.20)

Date currentDate{ 20, 2, 2024 };

for (const auto& employee : employees) { // канстантны ітэратар па вектары

// Разлічваем стаж у месяцах адносна бягучай даты

int experience = (currentDate.year - employee.hire\_date.year) \* 12 +

(currentDate.month - employee.hire\_date.month);

if (experience >= experienceInMonths) {

result.push\_back(employee); //захоўваем у вектары ўсіх супрацоўнікаў, стаж якіх больш уведзенага для пошука

}

}

return result; //вяртаем вектар супрацоўнікаў

}

int main() {

std::vector<Employee> employees; //вектар усіх супрацоўніках

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int choice;

do {

std::cout << "\n----- Меню -----" << std::endl;

std::cout << "1. Дадаць супрацоўніка" << std::endl;

std::cout << "2. Вывесці ўсіх супрацоўнікаў" << std::endl;

std::cout << "3. Выдаліць супрацоўніка па імені" << std::endl;

std::cout << "4. Знайсці супрацоўнікаў па стажы працы" << std::endl;

std::cout << "0. Выхад" << std::endl;

std::cout << "Абярыце опцыю: ";

std::cin >> choice;

switch (choice) {

case 1:

std::cin.ignore(); //пасля ўводу опцыі застаўся сімвал новага радка ў буферы, ігнаруем яго

employees.push\_back(inputEmployee()); //захоўваем новага супрацоўніка

break;

case 2:

printAllEmployees(employees);

break;

case 3:

{

std::string nameToDelete;

std::cout << "Увядзіце імя супрацоўніка для выдалення: ";

std::cin.ignore();

std::getline(std::cin, nameToDelete); //атрымлівем радок, які ўтрымлівае імя супрацоўніка, якога трэба выдаліць

deleteEmployeeByName(employees, nameToDelete);

}

break;

case 4:

{

int experienceToFind;

std::cout << "Увядзіце стаж працы ў месяцах для пошуку: ";

std::cin >> experienceToFind; //стаж працы ў месяцах для пошуку

std::vector<Employee> result = findEmployeesByExperience(employees, experienceToFind); //ствараем вектар супрацоўнікаў, якія меюць уведзены, або большы стаж працы і ініцыялізуем яго вынікам функцыі пошуку

std::cout << "Супрацоўнікі з досведам працы больш " << experienceToFind << " месяцаў:" << std::endl;

printAllEmployees(result); //перадаем гэты вектар у функцыю вывада

}

break;

case 0:

std::cout << "Выхад з праграмы." << std::endl;

break;

default:

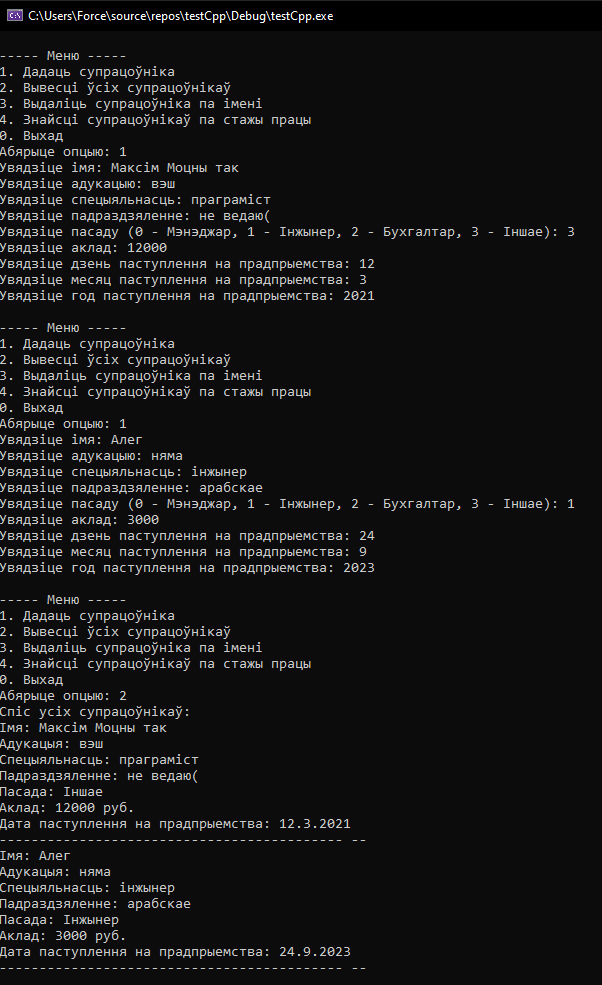
std::cout << "Некарэктны ўвод. Калі ласка, абярыце існуючую опцыю." << std::endl;

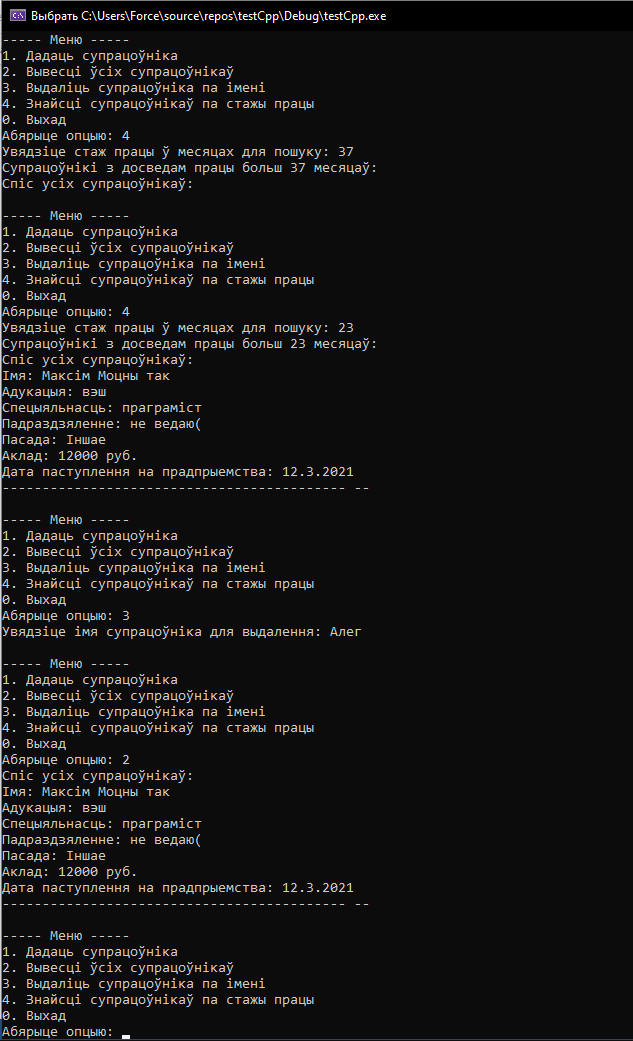
}

} while (choice!= 0);

return 0;

}







#include <iostream>

#include <string>

#include <cstring>

#include "Windows.h"

#include <fstream>

using namespace std;

//структура для інфармацыі

struct TeacherInfo {

char lastName[50];

char examName[50];

char examDate[20];

};

//аб'яднанне

union Exam {

TeacherInfo teacher; //ствараем аб'ект структуры

char data[120]; //агульны масіў, які будзе захоўваць усе дадзеныя ў адной зменнай

};

// функцыя параўнання радкоў

int myStrcmp(const char\* str1, const char\* str2) { //прымаем два паказальніка на радкі для параўнання

while (\*str1 != '\0' && \*str2 != '\0' && \*str1 == \*str2) { // пакуль абодва радкі не скончыліся (!='\0') і пакуль яны посимвольно роўныя

//павялічваем значэння паказальнікаў

str1++;

str2++;

}

//калі паказальнікі роўныя пасля выхаду з цыклу, то значыць абодва радкі скончыліся('\0') і яны роўныя

return \*str1 == \*str2;

}

//макраканстанта

#define MAX\_TEACHERS 5

//адлюстраванне інфармацыі аб выкладчыку

void displayTeacherData(const Exam& teacher) {

cout << "Прозвішча выкладчыка: " << teacher.teacher.lastName << endl; //выкарыстоўваю палі структуры для вываду дадзеных

cout << "Назва іспыту: " << teacher.teacher.examName << endl;

cout << "Дата іспыту: " << teacher.teacher.examDate << endl << endl;

}

// Функцыя для пошуку выкладчыка па прозвішчы

void searchByLastName(const Exam teachers[], const char\* targetLastName, int size) { //прымае масіў настаўнікаў, імя для пошуку, колькасць элементаў у масіве

for (int i = 0; i < size; i++)

{

if (myStrcmp(teachers[i].teacher.lastName, targetLastName)) { //калі ўведзенае імя для пошуку і імя бягучага выкладчыка супадаюць

cout << "\nВыкладчык знойдзены:" << endl;

displayTeacherData(teachers[i]); //выводзім

return; //выходзім з функцыі

}

}

cout << "\nВыкладчык з прозвішчам " << targetLastName << " не знойдзены." << endl;

}

//увод выкладчыка

Exam inputTeacherData() {

Exam teacher;

cout << "Увядзіце прозвішча выкладчыка: ";

cin.getline(teacher.data, 50); //атрымліваем прозвішча выкладчыка праз getline, запісваем праз поле аб'яднання, абмяжоўваючы максімальны памер уводу 50 сімваламі

cout << "Увядзіце назву іспыту: ";

cin.getline(teacher.data + 50, 50); //дадаем 50, бо гэта будзе з 51 элемента па 100. паколькі памяць агульная ў аб'яднанні для data і TeacherInfo, то гэта раўнасільна запісу ў teacher.teacher.examName

cout << "Увядзіце дату іспыту: ";

cin.getline(teacher.data + 100, 20);

return teacher;

}

void writeToFile(Exam teachers[], const int& size, const char\* fileName) {

ofstream file(fileName); //адкрываем файл для запісу

//правяраем на паспяховасць адкрыцця

if (!file.is\_open() || file.fail()) {

cerr << "Памылка адкрыцця файла для запісу\n";

return;

}

//праз індэксы праходзімся па ўсіх элементах у масіве

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

file << teachers[i].teacher.lastName << ',' << teachers[i].teacher.examName << ',' << teachers[i].teacher.examDate << '\n'; //запісваем у файл праз коску ўсе палі структуры, завяршаем сімвалам новага радка

}

file.close();

cout << "\nДадзеныя паспяхова запісаны ў файл\n\n";

}

//Прыклад самай простай уласнай рэалізацыі strcpy\_s, у кодзе не выкарыстоўваецца

void stringTochararr(char\* dest, string& src) {

int i = 0;

for (; src[i] != '\0'; i++) { //праз src[i]!= '\0' правяраем, што радок, з якога які капіруецца, не скончыўся

dest[i] = src[i]; // які капіруецца знакі з src у dest

}

dest[i] = '\0'; // апошні элемент сімвал канца радка

}

void readFromFile(Exam teachers[], int& size, const char\* fileName) {

ifstream file(fileName); // адкрываем файл для чытання

if (!file.is\_open() || file.fail()) {

cerr << "Памылка адкрыцця файла для чытання\n";

return;

}

string input; //першы радок, які будзе прачытаны

while (MAX\_TEACHERS > size && getline(file, input, ',')) { // пакуль ёсць месца ў масіве і атрымалася прачытаць дадзеныя з файла

strcpy\_s(teachers[size].teacher.lastName, input.c\_str()); //strcpy\_s прымае паказальнік на масіў, у які будзем капіраваць знакі(teachers[size].teacher.lastName), і паказальнік адкуль будзем капіраваць. метад std::string c\_str верне сімвалальны паказальнік на пачатак радка input

getline(file, input, ','); //атрымліваем сімвалы з файла і запісваем у радок input, пакуль не сустрэнецца коска

strcpy\_s(teachers[size].teacher.examName, input.c\_str());

getline(file, input);

strcpy\_s(teachers[size].teacher.examDate, input.c\_str());

size++;

}

file.close();

cout << "\nДадзеныя з файла прачытаныя\n\n";

}

int main() {

int currentTeachers = 0; //бягучая колькасць выкладчыкаў

Exam examData[MAX\_TEACHERS]; //масіў выкладчыкаў

int choice;

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

do { //выводзім меню пакуль не ўведзена 0

cout << "\nМеню:" << endl;

cout << "1. Увесці дадзеныя аб выкладчыку" << endl;

cout << "2. Вывесці дадзеныя аб выкладчыку" << endl;

cout << "3. Пошук выкладчыка па прозвішчы" << endl;

cout << "4. Запіс усіх выкладчыкаў у файл" << endl;

cout << "5. Чытанне выкладчыкаў з файла" << endl;

cout << "0. Выйсці" << endl;

cout << "Абярыце опцыю: ";

cin >> choice;

cin.ignore(); // чысцім буфер уводу пасля ўводу ліку, каб можна было прачытаць радок, выкарыстаючы getline, без праблем

switch (choice) {

case 1:

// Увод дадзеных аб выкладчыку

cout << "\nУвядзіце даныя аб выкладчыку:" << endl;

examData[currentTeachers++] = inputTeacherData(); //захоўваем вынік выканання inputTeacherData (новы выкладчык) у масіве і павялічваем іх лік

break;

case 2: {

// Выснова дадзеных аб выкладчыку

int counter = 1;

cout << '\n';

for (int i = 0; i < currentTeachers; i++) { //для кожнага выкладчыка выклікаем функцыю вываду інфармацыі аб ім на экран

cout << "выкладчык " << counter++ << ": \n";

displayTeacherData(examData[i]);

}

break;

}

case 3:

// Пошук выкладчыка па прозвішчы

char targetLastName[50]; //прозвішча для пошуку

cout << "\nУвядзіце прозвішча выкладчыка для пошуку: ";

cin.getline(targetLastName, 50); //уводзім прозвішча

searchByLastName(examData, targetLastName, currentTeachers);

break;

case 4:

writeToFile(examData, currentTeachers, "teachers.txt");

break;

case 5:

readFromFile(examData, currentTeachers, "teachers.txt");

break;

case 0:

cout << "\nВыхад з праграмы." << endl;

break;

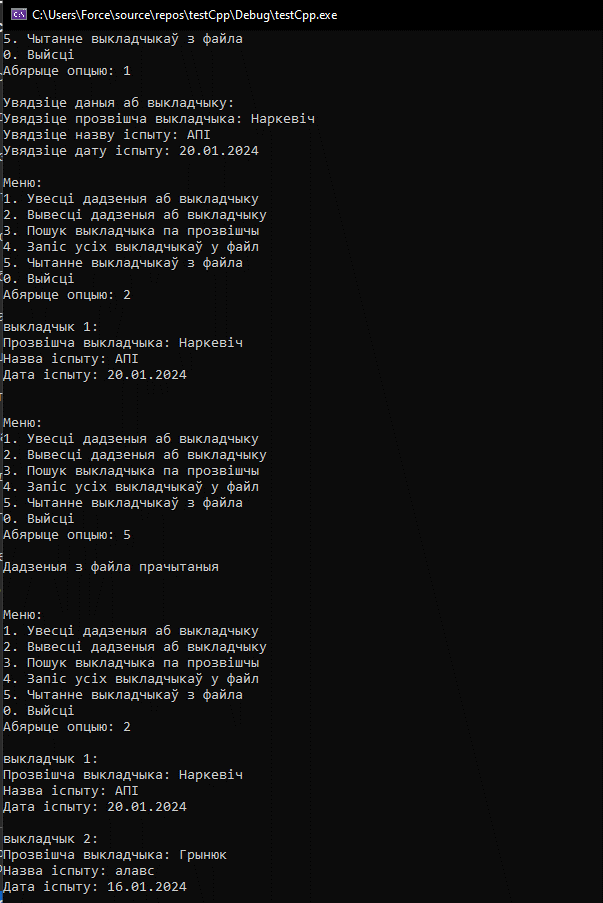
default:

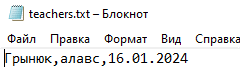
cout << "Некарэктны выбар. Калі ласка, абярыце зноў." << endl;

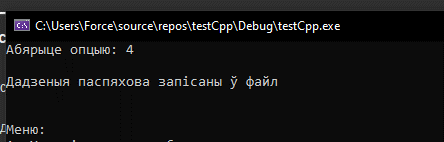
}

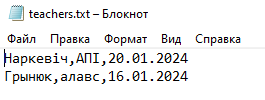
} while (choice!= 0);

return 0;

}







**Варыянт 2**



#include <iostream>

#include <vector>

#include <string>

#include <Windows.h>

using namespace std;

// Пералік для прадстаўлення полу

enum Gender {

Male,

Female

};

// Структура для прадстаўлення даты з выкарыстаннем бітавых палёў

struct Date {

unsigned day : 5; // 0-31

unsigned month : 4; // 0-15

unsigned year : 12; // 0-4095

};

// Структура для прадстаўлення гараджаніна

struct Citizen {

std::string fullName;

Date birthDate;

std::string address;

Gender gender;

};

// Функцыя для ўводу гараджаніна з клавіятуры

Citizen inputCitizen() {

Citizen newCitizen;

cout << "Увядзіце П.І.І.: ";

std::getline(cin, newCitizen.fullName); //атрымліваем радок і адразу запісваем у поле newCitizen.fullName

int dataInput; //прамежкавае захоўванне ўводу

cout << "Увядзіце дзень нараджэння (ГГГГ.ММ.ДД): ";

// год нараджэння

cin >> dataInput;

newCitizen.birthDate.year = static\_cast<unsigned>(dataInput); //static\_cast<unsigned> - перавод дадзеных у круглых дужках у тып дадзеных unsigned

cin.get(); // прапускаем сімвал "."

//Месяц

cin >> dataInput;

newCitizen.birthDate.month = static\_cast<unsigned>(dataInput);

cin.get();

//Дзень

cin >> dataInput;

newCitizen.birthDate.day = static\_cast<unsigned>(dataInput);

cin.ignore(); // Ачыстка буфера пасля ўводу лікаў ад сімвала новага радка

cout << "Увядзіце адрас: ";

std::getline(cin, newCitizen.address);

cout << "Увядзіце пол (0 - мужчынскі, 1 - жаночы): ";

int genderInput;

cin >> genderInput;

cin.ignore();// Очистка буфера после ввода числа

// Пераўтварэнне ўведзенага ліку ў пералік Gender

newCitizen.gender = static\_cast<Gender>(genderInput);

return newCitizen;

}

// Функцыя для вываду гараджаніна на экран

void printCitizen(const Citizen& citizen) {

cout << "П.І.І.: " << citizen.fullName << std::endl;

cout << "Дата нараджэння: " << citizen.birthDate.day << "." << citizen.birthDate.month << "." << citizen.birthDate.year << std::endl;

cout << "Адрас: " << citizen.address << std::endl;

cout << "Пол: " << (citizen.gender == Male ? "Мужчынскі" : "Жаночы") << std::endl;

cout << "------------------------" << std::endl;

}

// Функцыя для выдалення грамадзян па годзе нараджэння з выкарыстаннем erase()

void removeCitizensByYear(std::vector<Citizen>& citizens, unsigned targetYear) {

std::vector<Citizen>::iterator it = citizens.begin(); //ствараем ітэратар на пачатак вектара, які захоўвае ў сабе грамадзян

while (it != citizens.end()) { //пакуль мы не дайшлі да канца вектара

if (it->birthDate.year == targetYear) { //калі год роўны уведзенаму для пошуку

it = citizens.erase(it); // выдаляемы элемент выкарыстоўваючы erase, перадаючы ітэратар, які паказвае на элемент, які трэба выдаліць, абнаўляем ітэратар

}

else {

++it; //павялічваем значэнне ітэратара

}

}

}

// Функцыя для пошуку і вываду грамадзян па годзе нараджэння

void findAndPrintCitizensByYear(const std::vector<Citizen>& citizens, unsigned targetYear) {

cout << "Грамадзяне з годам нараджэння" << targetYear << ":" << std::endl;

for (const auto& citizen : citizens) { //канстантны ітэратар па элементах вектара citizens

if (citizen.birthDate.year == targetYear) { //калі год роўны уведзенаму для пошуку

printCitizen(citizen); //выводзім

}

}

}

// Функцыя для адлюстравання меню

int displayMenu() {

int choice;

cout << "\nМеню:" << std::endl;

cout << "1. Увесці грамадзяніна" << std::endl;

cout << "2. Вывесці ўсіх грамадзян" << std::endl;

cout << "3. Выдаліць грамадзян па годзе нараджэння" << std::endl;

cout << "4. Знайсці і вывесці грамадзян па годзе нараджэння" << std::endl;

cout << "0. Выйсці" << std::endl;

cout << "Абярыце опцыю: ";

cin >> choice;

cin.ignore(); // Ачыстка буфера пасля ўводу ліку

return choice;

}

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

std::vector<Citizen> citizens; // Вектар для захоўвання грамадзян

int menuChoice;

do {

menuChoice = displayMenu(); //выводзім меню і атрымліваем у функцыі выбар варыянту дзеяння карыстальнікам

switch (menuChoice) {

case 1:

citizens.push\_back(inputCitizen()); // дадаем у вектар уведзенага грамадзяніна выкарыстоўваючы pus\_back

break;

case 2:

// Выснова інфармацыі аб усіх грамадзянах

cout << "\nІнфармацыя пра ўсіх грамадзян:" << std::endl << std::endl;

for (const auto& citizen : citizens) {

printCitizen(citizen);

}

break;

case 3:

// Выдаленне грамадзян па годзе нараджэння

unsigned targetYearToRemove;

cout << "Увядзіце год нараджэння для выдалення: ";

cin >> targetYearToRemove;

cin.ignore(); // Ачыстка буфера пасля ўводу ліку

removeCitizensByYear(citizens, targetYearToRemove);

break;

case 4:

// Пошук і вывад грамадзян па годзе нараджэння

unsigned targetYearToFind;

cout << "Увядзіце год нараджэння для пошуку: ";

cin >> targetYearToFind;

cin.ignore(); // Ачыстка буфера пасля ўводу ліку

findAndPrintCitizensByYear(citizens, targetYearToFind);

break;

case 0:

cout << "Выхад...\n";

break;

default:

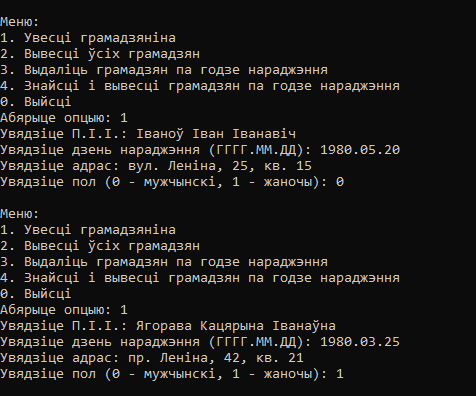
cout << "няправільны выбар\n";

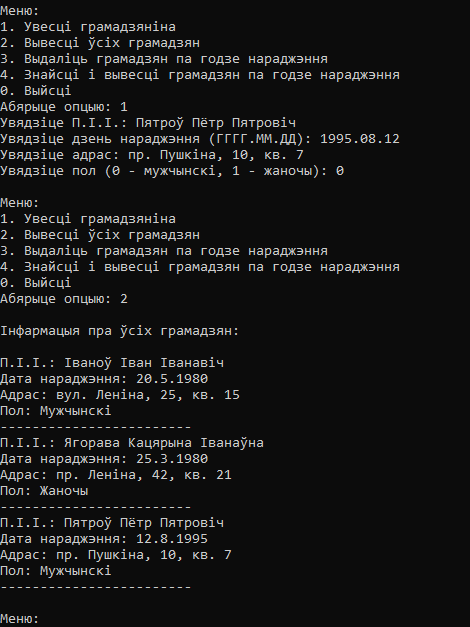
break;

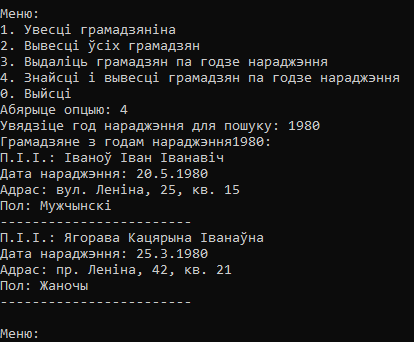
}

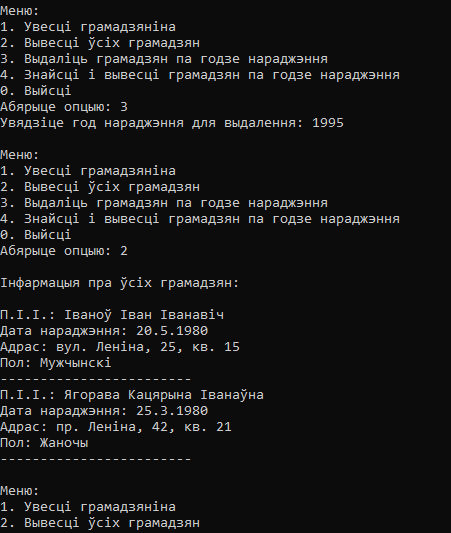
} while (menuChoice != 0);

}











#include <iostream>

#include <string>

#include <vector>

#include <Windows.h>

#include <fstream>

using namespace std;

// Вызначэнне структуры для даных цягніка

struct TrainData {

char trainNumber[10];

char destination[50];

char travelDays[10];

char arrivalTime[20];

char departureTime[20];

};

// Вызначэнне аб'яднання для розных прадстаўленняў даных

union Station {

TrainData trainInfo;

char allData[]; // дазваляе атрымаць доступ да ўсіх палёў структуры TrainData

};

Station inputTrain() {

Station train;

cout << "Увядзіце нумар цягніка: ";

cin.getline(train.trainInfo.trainNumber, 10); // Атрымліваем радок і адразу запісваем у поле, абмяжоўваючы даўжыню 10 сімваламі

cout << "Увядзіце пункт прызначэння цягніка: ";

cin.getline(train.trainInfo.destination, 50);

cout << "Увядзіце час у шляху: ";

cin.getline(train.trainInfo.travelDays, 10);

cout << "Увядзіце час і дату прыбыцця: ";

cin.getline(train.trainInfo.arrivalTime, 20);

cout << "Увядзіце час і дату адпраўлення: ";

cin.getline(train.trainInfo.departureTime, 20);

return train;

}

void display(const Station& station) {

cout << endl;

cout << "Нумар цягніка: " << station.trainInfo.trainNumber << endl;

cout << "Пункт прызначэння цягніка: " << station.trainInfo.destination << endl;

cout << "Час у шляху: " << station.trainInfo.travelDays << endl;

cout << "Час і дата прыбыцця: " << station.trainInfo.arrivalTime << endl;

cout << "Час і дата адпраўлення: " << station.trainInfo.departureTime << endl;

}

// Функцыя для капіравання з аднаго масіва сімвалаў у іншы

void my\_strcpy\_s(char\* destination, size\_t dest\_size, const char\* source) {

if (destination == nullptr || source == nullptr || dest\_size == 0) {

// Апрацоўка некарэктных аргументаў

std::cerr << "Некарэктныя аргументы ў my\_strcpy\_s\n";

return;

}

size\_t i; // size\_t - тып дадзеных, які выкарыстоўваецца для прадстаўлення памераў аб'ектаў у памяці

for (i = 0; i < dest\_size - 1 && source[i] != '\0'; ++i) { //dest\_size - 1 так як яшчэ трэба запісаць сімвал канца радка \0. source[i] != '\0' будзе азначаць, што радок крыніцы скончыўся

destination[i] = source[i];

}

// Даданне завяршальнага знака '\0'

destination[i] = '\0';

}

//функцыя для параўнання двух радкоў

int myStrcmp(const char\* str1, const char\* str2) { //для параўнання бяром два паказальнікі на радок

while (\*str1 != '\0' && \*str2 != '\0' && \*str1 == \*str2) { // пакуль абодва радкі не скончацца (!='\0') і пакуль яны не стануць пасімвальна роўнымі

//павялічваем значэнні паказальнікаў

str1++;

str2++;

}

// Калі пасля выхаду з цыкла абодва радкі скончыліся (==\0), то яны роўныя

if (\*str1 == '\0' && \*str2 == '\0') {

return 0;

}

// Вяртаем 1, калі str1 даўжэй str2, інакш -1

return (str1 > str2) ? 1 : -1;

}

void writeToFile(std::vector<Station>& stations, const char\* fileName) {

ofstream file(fileName); //адкрываем файл для запісу

//правяраем паспяховасць адкрыцця файла

if (!file.is\_open() || file.fail() || file.bad()) {

std::cerr << "Не атрымалася адкрыць файл для запісу\n";

return;

}

//праходзім ітэратэрам па вектары з цягнікамі

for (const Station& station : stations) {

//для кожнага поля структуры TrainData у аб'яднанні Station запісваем змесціва ў файл, выкарыстаючы агульны масіў allData, падзяляя коскамі

file << station.allData << ',' << (station.allData + 10) << ',' << (station.allData + 60) << ',' << (station.allData + 70) << ',' << (station.allData + 90) << '\n';

}

file.close();

}

void readFromFile(std::vector<Station>& stations, const char\* fileName) {

ifstream file(fileName); // адкрываем файл для чытання

// Праверка на паспяховае адкрыццё файла

if (!file.is\_open() || file.fail() || file.bad()) {

std::cerr << "Памылка: Не атрымалася адкрыць файл\n";

return;

}

char trainNumber[10]; // Буфер для счытвання ўнутры ўмовы цыклу

while (file.getline(trainNumber, 10, ',')) { //счытваем з файла 10 сімвалаў(ці менш), або пакуль не сустрэнем коску, у масіў trainNumber

Station station;

// Захаванне ў аб'яднанні

my\_strcpy\_s(station.allData, sizeof(station.trainInfo.trainNumber), trainNumber);

// Паколькі выкарыстоўваецца аб'яднанне, палі структуры TrainData можна запісаць,

// выкарыстоўваючы масіў allData, дадаючы да яго пачатковыя індэксы і задаючы памеры,

// адпаведныя памерам і індэксам палёў структуры

file.getline((station.allData + 10), 50, ','); // Счытванне з файла 50 сімвалаў або да коскі, запіс у station.allData пачынаючы з індэкса 10

file.getline((station.allData + 60), 10, ',');

file.getline((station.allData + 70), 20, ',');

file.getline((station.allData + 90), 20);

stations.push\_back(station); // захоўваем у вектары аб'ект аб'яднання выкарыстоўваючы push\_back

}

file.close();

}

void search(std::vector<Station>& stations, const char\* dest) {

printf\_s("Рэйсы з пунктам прызначэння %s: \n", dest);

for (const Station& station : stations) {

if (myStrcmp(dest, station.trainInfo.destination) == 0) { //калі пункт прызначэння бягучага элемента вектара роўны мэтаваму

display(station); //выводзім

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

}

}

cout << '\n';

}

int main() {

std::vector<Station> stations;

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

int choice = -1;

do

{

cout << "1. Увод інфармацыі\n";

cout << "2. Вывад інфармацыі пра ўсе цягнікі\n";

cout << "3. Захаванне ў файле\n";

cout << "4. Чытанне з файла\n";

cout << "5. Пошук\n";

cout << "0. Выхад\n";

cout << "Абярыце варыянт: ";

cin >> choice;

cin.ignore();

switch (choice) {

case 1: {

stations.push\_back(inputTrain());

break;

}

case 2: {

for (const Station& station : stations) {

display(station);

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

}

break;

}

case 3: {

writeToFile(stations, "out.txt");

break;

}

case 4: {

readFromFile(stations, "out.txt");

break;

}

case 5:

{

cout << "Увядзіце пункт прызначэння для пошуку: ";

char distination[50];

cin.getline(distination, 50);

search(stations, distination);

break;

}

default: {

std::cout << "Няправільны выбар";

break;

}

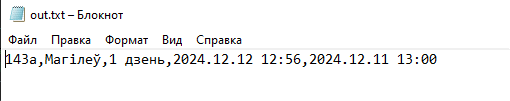
}

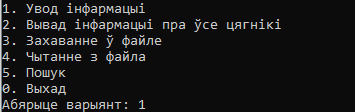
} while (choice!= 0);

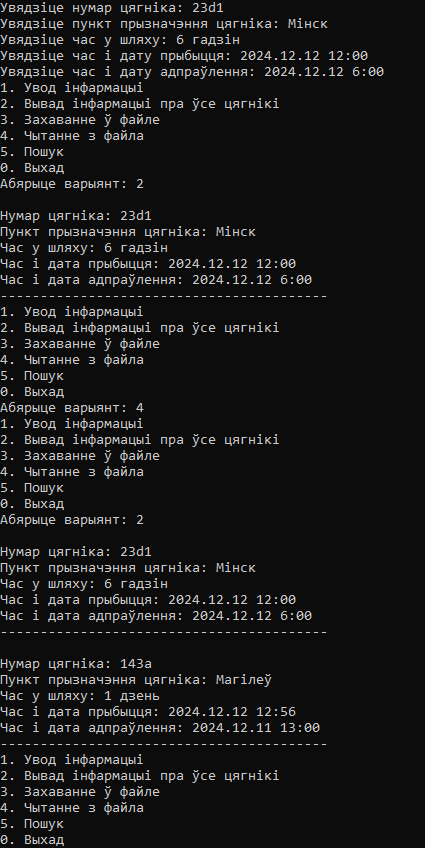
return 0;

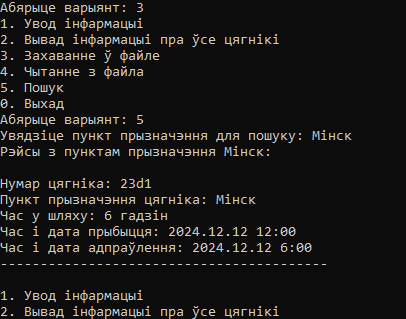
}

Пачатковы стан

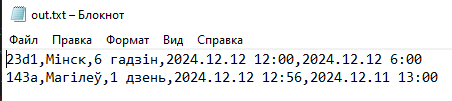




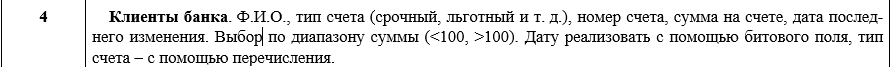




Пасля запісу



**Варыянт 4**



#include <iostream>

#include <string>

#include <Windows.h>

enum AccountType {

SAVINGS, // Ашчадны

PREFERENTIAL, // Льготны

};

struct Date {

unsigned day : 5; // 0-31

unsigned month : 4; // 0-15

unsigned year : 12; // 0-4095

};

struct BankClient {

std::string fullName; // Прозвішча, імя кліента банка

AccountType accountType; // Тып акаўнта

int accountNumber = 0; // Нумар рахунку

long long accountBalance = 0; // Баланс

Date lastModifiedDate; // Бітавое поле для даты апошняй змены інфармацыі

};

const int MAX\_CLIENTS = 15; // максімальная колькасць кліентаў

BankClient clients[MAX\_CLIENTS]; // масіў для захоўвання кліентаў

int currentClientsCount = 0; //лічыльнік кліентаў

// Увод інфармацыі аб кліенце

void inputClientsData();

// Выснова інфармацыі аб усіх кліентах

void displayAllClientsData();

// Выдаленне дадзеных аб кліенце

void deleteClient(int accountNumberToDelete);

// Пошук кліента

void searchClient();

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

int choice;

do { // вывад меню

std::cout << "\nБанкаўская сістэма кіравання кліентамі\n";

std::cout << "1. Увод дадзеных аб кліентах\n";

std::cout << "2. Вывесці дадзеныя кліента\n";

std::cout << "3. Выдаліць кліента\n";

std::cout << "4. Пошук кліента\n";

std::cout << "0. Выхад\n";

std::cout << "Увядзіце свой выбар: ";

std::cin >> choice; std::cin.ignore(); // Адчышчаем буфер ад знака новага радка пасля ўводу ліку

switch (choice) {

case 1:

inputClientsData();

break;

case 2:

displayAllClientsData();

break;

case 3: {

int accountNumberToDelete;

std::cout << "Увядзіце нумар рахунку кліента, якога неабходна выдаліць: ";

std::cin >> accountNumberToDelete;

deleteClient(accountNumberToDelete);

break;

}

case 4:

searchClient();

break;

case 0:

std::cout << "Выхад з праграмы....\n";

break;

default:

std::cout << "Некарэктны выбар\n";

}

} while (choice!= 0);

return 0;

}

void inputClientsData() {

if (currentClientsCount >= MAX\_CLIENTS) { // Правяраем, што яшчэ ёсць месца ў масіве для кліентаў

std::cerr << "Максімальная колькасць кліентаў ужо дасягнута\n";

return;

}

// Ствараем новы аб'ект структуры для прадстаўлення бягучага кліента

BankClient newClient;

std::cout << "Увядзіце Прозвішча, імя: ";

std::getline(std::cin, newClient.fullName);// выкарыстоўваем getline для атрымання радка і запісы ў поле структуры

std::cout << "Увядзіце тып акаўнта (0 - Ашчадны, 1 - Льготны): ";

int accountTypeInput; // Паколькі адразу лічыць у тыпе пераліку нельга

std::cin >> accountTypeInput;

newClient.accountType = static\_cast<AccountType>(accountTypeInput); // Канвертоўны з int у тып дадзеных пераліку AccountType, выкарыстоўваючы static\_cast і запісвае ў структуру

std::cout << "Увядзіце нумар рахунку: ";

std::cin >> newClient.accountNumber;

std::cout << "Увядзіце баланс: ";

std::cin >> newClient.accountBalance;

std::cout << "Увядзіце дату апошняй змены (дзень месяц год, праз прабел): ";

unsigned day, month, year; // гэтак жа нельга запісаць у бітавае поле адразу

std::cin >> day >> month >> year; // спачатку захоўваем у зменныя

newClient.lastModifiedDate.day = day; // потым дадаем іх значэння адпаведным палям

newClient.lastModifiedDate.month = month;

newClient.lastModifiedDate.year = year;

clients[currentClientsCount++] = newClient; // захоўваем кліента

}

void displayClientInfo(const BankClient& client) {

// прымаем аб'ект структуры, які прадстаўляе кліента і выводны змесціва ўсіх яго палёў

std::cout << "ПІБ: " << client.fullName << "\n";

switch (client.accountType)

{

case SAVINGS:

std::cout << "Тып рахунку: Ашчадны\n";

break;

case PREFERENTIAL:

std::cout << "Тып рахунку: Льготны\n";

break;

default:

std::cout << "Тып рахунку: Не вядома\n";

break;

}

std::cout << "Нумар рахунку: " << client.accountNumber << "\n";

std::cout << "Баланс: " << client.accountBalance << "\n";

std::cout << "Дата апошняй змены: " << client.lastModifiedDate.day

<< " " << client.lastModifiedDate.month

<< " " << client.lastModifiedDate.year << "\n\n";

}

void displayAllClientsData() {

// Перабіраем усіх кліентаў у масіве і для кожнага выклікаем функцыю вываду

for (int i = 0; i < currentClientsCount; i++) {

displayClientInfo(clients[i]);

}

}

void searchClient() {

char comparisonOperator;

long long Balance;

std::cout << "Увядзіце крытэр(<, >, =) і суму параўнання(напрыклад, <100, >100, =100): ";

std::cin >> comparisonOperator; // для char лічыцца толькі адзін сімвал

std::cin >> Balance; //астатнія для ліку

switch (comparisonOperator) {

case '<': {

for (int i = 0; i < currentClientsCount; i++) { // Перабіраем усіх кліентаў у масіве і для кожнага параўноўваем баланс з уведзеным балансам

if (clients[i].accountBalance < Balance) { // калі ўмова выконваецца, то выводны інфармацыю аб кліенце

std::cout << "Знойдзены кліент з балансам менш " << Balance << ":\n";

displayClientInfo(clients[i]); //Выводзім звесткі пра кліента

}

}

break;

}

case '>': {

for (int i = 0; i < currentClientsCount; i++) {

if (clients[i].accountBalance > Balance) {

std::cout << "Знойдзены кліент з балансам больш " << Balance << ":\n";

displayClientInfo(clients[i]);

}

}

break;

}

case '=': {

for (int i = 0; i < currentClientsCount; i++) {

if (clients[i].accountBalance == Balance) {

std::cout << "Знойдзены кліент з дакладным балансам " << Balance << ":\n";

displayClientInfo(clients[i]);

}

}

break;

}

default:

std::cout << "Некарэктны аператар параўнання\n";

}

}

void deleteClient(int accountNumberToDelete) {

int foundIndex = -1;

// спачатку знаходзім індэкс кліента для выдалення ў масіве па нумары рахунку

for (int i = 0; i < currentClientsCount; i++) {

if (clients[i].accountNumber == accountNumberToDelete) {

foundIndex = i;

break;

}

}

// калі знайшлі, то выдаляем яго шляхам зруху ўсіх астатніх кліентаў, якія стаяць справа ад яго, налева на 1

if (foundIndex != -1) {

for (int i = foundIndex; i < currentClientsCount - 1; i++) {

clients[i] = clients[i + 1];

}

--currentClientsCount; // пасля памяншэння лічыльніка, пры ўводзе след. кліента, выдалены будзе перазапісаны

std::cout << "Кліент з нумарам рахунку " << accountNumberToDelete << " быў выдалены.\n";

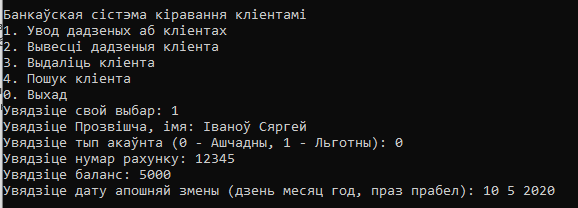
}

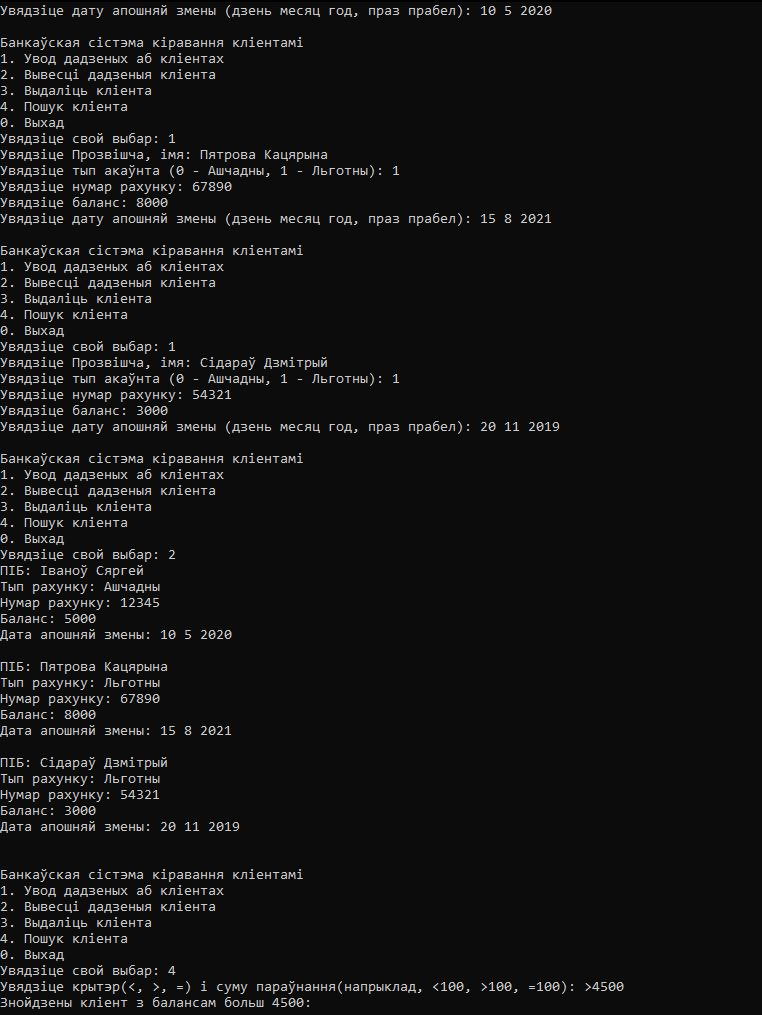
else {

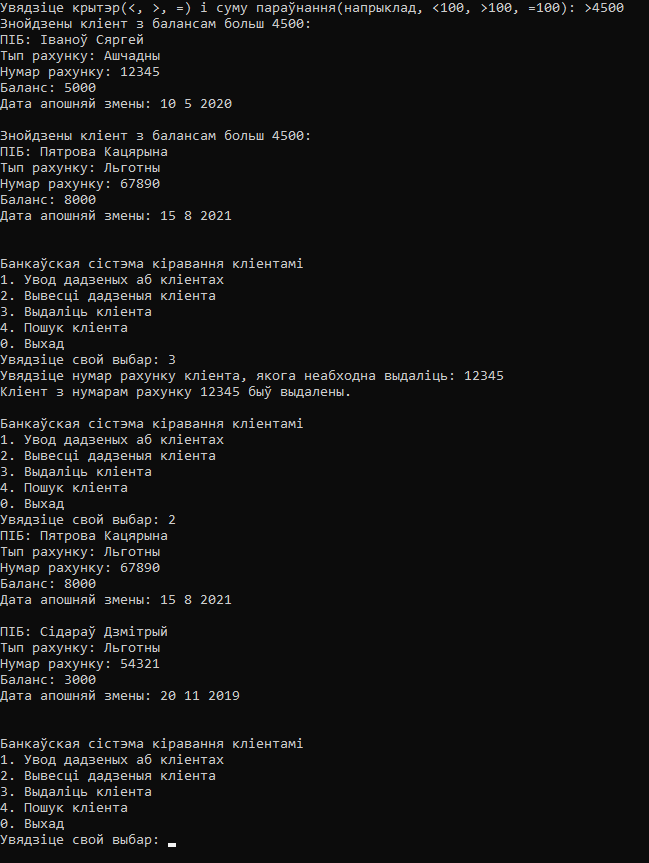
std::cout << "Кліент з нумарам рахунку " << accountNumberToDelete << " не знойдзены.\n";

}

}









#include <iostream>

#include <fstream>

#include <string>

#include <Windows.h>

using namespace std;

struct ClientInfo {

char fullName[50]; // Прозвішча, імя кліента банка

char accountType[40]; // Тып акаўнта

char lastModifiedDate[20]; // Дата апошняй змены інфармацыі

int accountNumber = 0; // Нумар рахунку

long long accountBalance = 0; // Баланс

};

union BankClient

{

ClientInfo clientInfo;

char allData[122]; // Сума памераў усіх палёў у структуры

BankClient() : clientInfo() {}; // Канструктар па змаўчанні

};

const int MAX\_CLIENTS = 15;

BankClient clients[MAX\_CLIENTS];

int currentClientsCount = 0;

// Уводзім інфармацыю аб кліенце

void inputClientsData();

// Выснова інфармацыі аб усіх кліентах

void displayClientsData();

// Шукаем кліента

void searchClient();

// Чытанне дадзеных з файла

void readFromFile();

// Запіс дадзеных у файл

void writeToFile();

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

int choice;

do {

cout << "\nБанкаўская сістэма кіравання кліентамі\n";

cout << "1. Увод дадзеных аб кліенце\n";

cout << "2. Паказваць дадзеныя кліента\n";

cout << "3. Пошук кліентаў\n";

cout << "4. Захаваць дадзеныя ў файл\n";

cout << "5. Прачытаць дадзеныя з файла\n";

cout << "0. Выхад\n";

cout << "Увядзіце свой выбар: ";

cin >> choice;

cin.ignore(); // Выдаляем сімвал новага радка з буфера пасля ўводу ліку, каб getline не атрымліваў пусты радок замест уводу карыстальніка

switch (choice) {

case 1:

inputClientsData();

break;

case 2:

displayClientsData();

break;

case 3:

searchClient();

break;

case 4:

writeToFile();

break;

case 5:

readFromFile();

break;

case 0:

cout << "выхад з праграмы....\n";

break;

default:

cout << "некарэктны выбар\n";

}

} while (choice != 0);

return 0;

}

void inputClientsData() {

if (currentClientsCount >= MAX\_CLIENTS) {

std::cerr << "максімальная колькасць кліентаў ужо дасягнута\n";

return;

}

BankClient newClient; // ствараем новы аб'ект аб'яднання

// Атрымліваем ад карыстальніка ўсе звесткі і захоўваем

cout << "Увядзіце Прозвішча, імя: ";

cin.getline(newClient.clientInfo.fullName, sizeof(newClient.clientInfo.fullName)); // Як І ў прошлым заданні счытваем радок у поле, але колькасць сімвалаў для запісу вызначаем з дапамогай sizeof

cout << "Уведзяце тып акаўнта: ";

cin.getline(newClient.clientInfo.accountType, sizeof(newClient.clientInfo.accountType));

cout << "Увядзіце нумар рахунку: ";

cin >> newClient.clientInfo.accountNumber;

cout << "Увядзіце баланс: ";

cin >> newClient.clientInfo.accountBalance;

cout << "Увядзіце дату апошняй змены: ";

cin.ignore();// Выдаляем сімвал новага радка з буфера пасля ўводу ліку

cin.getline(newClient.clientInfo.lastModifiedDate, sizeof(newClient.clientInfo.lastModifiedDate));

// Павялічваем колькасць кліентаў пры выкарыстанні зменнай у індэксе пры захаванні звестак аб кліенце

clients[currentClientsCount++] = newClient;

}

void searchClient() {

int accountNumberToSearch;

cout << " вядзіце нумар рахунку для пошуку: ";

cin >> accountNumberToSearch;

// Праход па масіве кліентаў

for (int i = 0; i < currentClientsCount; i++) {

// Праверка, ці супадае нумар акаўнта з шуканым

if (clients[i].clientInfo.accountNumber == accountNumberToSearch) {

// Выснова інфармацыі аб знойдзеным кліенце

cout << "знойдзены кліент:\n ";

cout << "ПІБ: " << clients[i].clientInfo.fullName << "\n";

cout << "тып рахунку: " << clients[i].clientInfo.accountType << "\n";

cout << "нумар рахунку: " << clients[i].clientInfo.accountNumber << "\n";

cout << "Баланс: " << clients[i].clientInfo.accountBalance << "\n";

cout << "дата апошняй мадыфікацыі: " << clients[i].clientInfo.lastModifiedDate << "\n\n";

return; // кліент знойдзены, завяршаем функцыю

}

}

// Калі цыкл завяршыўся без вяртання, кліент не знойдзены

cout << "кліент з нумарам рахунку " << accountNumberToSearch << " не знойдзены.\n";

}

void displayClientsData() {

// Праход па масіве кліентаў для вываду інфармацыі аб кожным

for (int i = 0; i < currentClientsCount; i++) {

// Выснова інфармацыі аб кліенце

cout << "\nКліент " << i + 1 << ": \n";

cout << "ПІБ: " << clients[i].clientInfo.fullName << "\n";

cout << "тып рахунку: " << clients[i].clientInfo.accountType << "\n";

cout << "нумар рахунку: " << clients[i].clientInfo.accountNumber << "\n";

cout << "Баланс: " << clients[i].clientInfo.accountBalance << "\n";

cout << "дата апошняй мадыфікацыі: " << clients[i].clientInfo.lastModifiedDate << "\n\n";

}

}

void readFromFile() {

std::ifstream inFile("bank\_clients.txt"); //адкрыццё файла для чытання

if (!inFile.is\_open()) { // Праверка на паспяховасць адкрыцця файла

std::cerr << "Памылка адкрыцця файла для чытання.\n";

return;

}

while (currentClientsCount < MAX\_CLIENTS) { // Пакуль у масіве кліентаў ёсць месца

BankClient newClient;

if (!inFile.getline(newClient.allData, sizeof(newClient.clientInfo.fullName), ',')) break; // Пакуль атрымліваецца атрымаць радок з файлу

inFile.getline(newClient.allData + sizeof(newClient.clientInfo.fullName), sizeof(newClient.clientInfo.accountType), ','); //запісваем радок у полі newClient.clientInfo.accountType, карыстаючысь агульным масівам newClient.allData і даўляючы да яго пачатковый індэксы

(inFile >> newClient.clientInfo.accountNumber);

inFile.ignore(); // прапускаем сімвал ',' у файле

(inFile >> newClient.clientInfo.accountBalance);

inFile.ignore(); // прапускаем сімвал ',' у файле

inFile.getline(newClient.allData + sizeof(newClient.clientInfo.fullName) + sizeof(newClient.clientInfo.accountType), sizeof(newClient.clientInfo.lastModifiedDate));

clients[currentClientsCount++] = newClient;

}

cout << "Дадзеныя паспяхова прачытаныя з файла.\n";

inFile.close();

}

void writeToFile() {

// Адкрыццё файла для запісу

std::ofstream outFile("bank\_clients.txt");

// Праверка паспяховасці адкрыцця файла

if (!outFile) {

std::cerr << "Памылка адкрыцця файла для запісу.\n";

return;

}

// Запіс дадзеных кожнага кліента ў файл, падзяляючы палі коскі

for (int i = 0; i < currentClientsCount; ++i) {

outFile << clients[i].clientInfo.fullName << ","

<< clients[i].clientInfo.accountType << ","

<< clients[i].clientInfo.accountNumber << ","

<< clients[i].clientInfo.accountBalance << ","

<< clients[i].clientInfo.lastModifiedDate << "\n";

}

// Вывад паведамлення аб паспяховым запісе дадзеных у файл

cout << "Даныя паспяхова запісаны ў файл.\n";

// Закрыццё файла пасля завяршэння запісу

outFile.close();

}

