ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

Факултет по изчислителна техника и автоматизация

Катедра „СИТ“

**СЕМЕСТРИАЛНА ДОМАШНА РАБОТА**

по дисциплината „Базово програмиране”

на тема: „Информационна система влогове”

Вариант 283

|  |  |
| --- | --- |
| Изготвил: Али-Джан Садък | Проверил: |
| Специалност: СИТ |  |
| Група: 5 а |  |
| Факултетен номер: 23621730 |  |

2024

Съдържание

1. Задание на проекта
2. Анализ на решението

1.Константи, изброими типове и Структури за данните от програмата

2.Реализация на условие А.

3.Реализация на условие B(а).

Реализация на условие B(b).

4.Реализация на условие C.

5.Реализация на условие D(а).

6.Реализация на условие D(b).

7.Реализация на условие E.

8.Реализация на условие F(a).

8.Реализация на условие F(b).

9.Реализация на условие G(a).

9.Реализация на условие G(b).

10. Реализация на условие H(a,b,c).

11. Реализация на условие I(a).

12.Реализация на условие I(b).

12. Реализация на условие J.

1. Примерно действие на програмата
2. Приложение
3. Задание

**Информационна система Влогове**

Да се напише компютърна програма, реализираща информационна система, която поддържа финансови влогове. Програмата съхранява и обработва данни за вложителите (номер на сметка, име, списък с влогове, година на откриване на сметка) в банка. Максималният брой вложители е 50. Всеки вложител може да има от 1 до 3 влога (валута, стойност, година на създаване) с различни валути.

**Базова задача**

**A.** Меню за избор на функциите от програмата

**Функции от програмата са:**

**B.** Добавяне на вложители (7-8 седмица)

a. Добавяне на нов вложител с един влог. Избира се валутата на влога (BGN (лев), USD (долар), EURO (евро));

b. Добавяне на списък от вложители. Въвежда се цяло число ***n*** и след него ***n*** на брой вложители с един влог. Избира се валутата на влога (BGN (лев), USD (долар), EURO (евро)).

**C.** Извеждане на екрана: a. Извеждане на всички вложители в оформен вид

**D.** Търсене и извеждане на екрана:

a. Извеждане на вложители с влог BGN

b. Извеждане на вложители с максимален влог в EURO

**E.** Подреждане на основния масив с вложители, без да се извежда на екрана:

a. Подреждане на вложителите по нарастващ ред на номер на сметка.

**F.** Управление на файл

a. Извеждане на масива с книги във файл (двоичен)

b. Въвеждане на масива с книги от файл (двоичен)

**Допълнение първо (+ базова задача)**

**G.** Справки за влоговете с под меню (11-12 седмица)

a. Извлича от базовия масив вложителите от дадена година на откриване на сметка и ги сортира по номер на сметка:

b. Извлича от базовия масив вложителите, които имат само един влог и ги сортира по стойност на този влог.

**Допълнение второ (+ базова задача)**

**H.** Добавяне на нов влог: (9-10 седмица)

a. Всеки вложител (по номер на вложител) с по-малко от 3 влога може да му се създаде нов влог за валута, по която няма влог с минимална стойност 500;

b. Ако не съществува вложител с този номер, да се изведе подходящо съобщение; c. При опит за добавяне на влог на вложител с 3 влога се изписва подходящо съобщение;

**I.** Изтегляне на влог (10-11 седмица)

a. Въвеждат се номер на вложител, избира се влога от който ще се тегли и се изтегля цялата или част от сумата по влога.

b. Въвежда се номер на вложител и се извличат всичките му влогове, а профилът му се закрива;

**Допълнение трето (+ базова задача)**

**J.** Данните в програмата да могат да се запазват във файл (текстови) между две стартирания на програмата.

1. Анализ на решението

Структури за данните в програмата

|  |  |  |
| --- | --- | --- |
| Структура | Обяснение | Примерени стойности |
| struct Investor {  int accountNumber;  char name[30];  Deposit deposits[3];  int numDeposits;  int openingYear;  }; | int accountNumber - номер на банкова сметка  char name[30] - имена на вложител  Deposit deposits[3] - масив за влогове  int numDeposits - съхранява броя на влоговете  int openingYear - година на откриване на сметка | 4444  Martin Cvetkov  2022  3 |
| struct Deposit {  double value;  int creationYear;  char currency[20];  }; | double value - стойност на влога  int creationYear - година на създаване на влога  char currency[20] - валута на влога | BGN  30000  2022  EURO  4000  2022  EURO  5000  2022 |

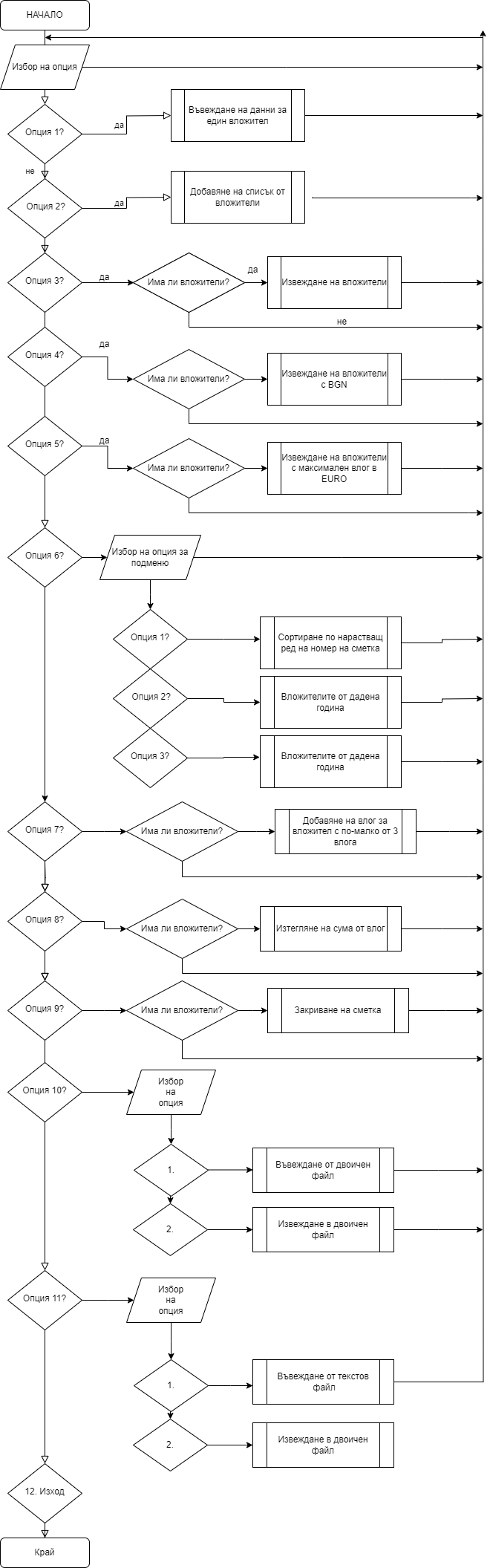
Реализация на условие A

Алгоритъм Меню с избор на функциите от програмата

Алгоритъмът за Меню с избор на функциите извежда на потребителя основните функционалности, чрез избор на опция от основно меню. Главният алгоритъм се реализира чрез функцията int main(), в която се състои и алгоритъма на основното меню.

Блок схема на алгоритъм

*Фиг 1 . блок схема на алгоритъм меню с избор на функциите .*



2.3 Функция, с която е реализиран алгоритъма

int main();

2.3.1. Входни данни на функцията

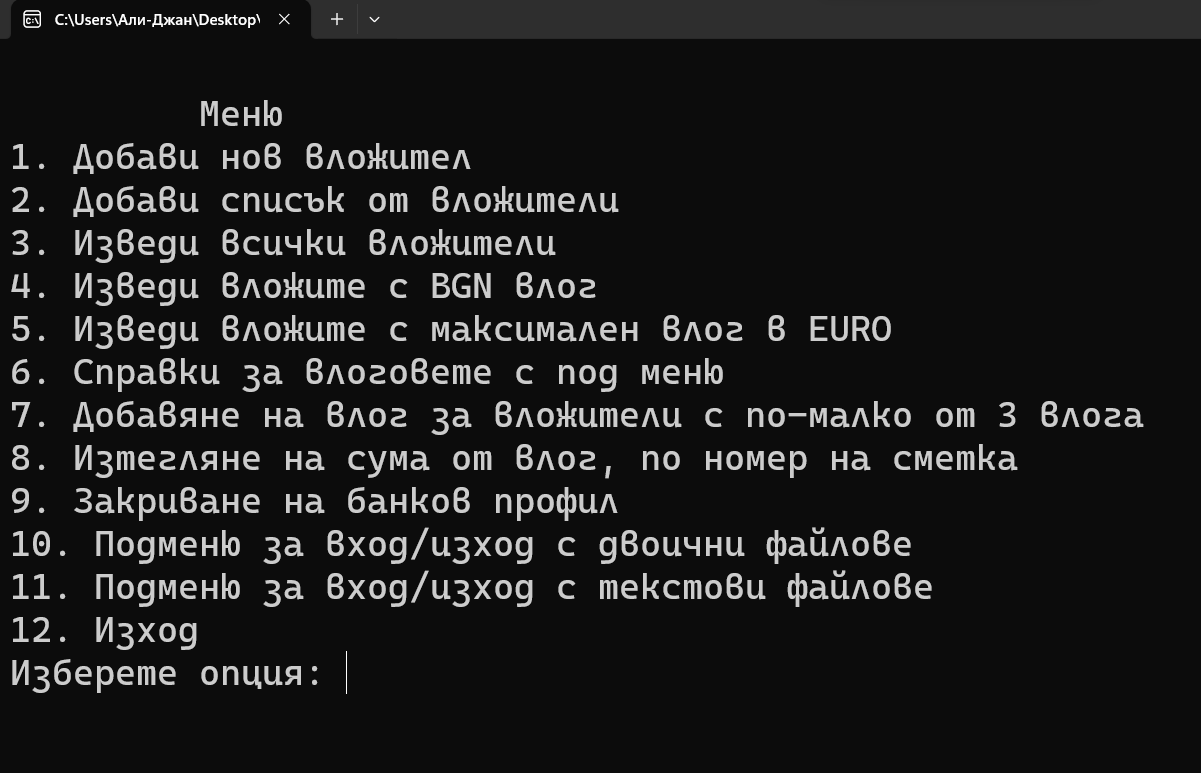
Няма входни данни .

2.3.2 Изходни данни на функцията или данни, които се

извеждат

Главната функция организира основен do while цикъл чрез меню техника. Спрямо избраната опция. функцията извиква съотвените оператори и функции за изпълнение на съответните действия. След приключване на работа функцията връща код 0 за успешно изпълнение.

*Фиг 2. Снимка на менюто*



Реализация на условие B.а

А. Добавяне на нов вложител

Потребителят въвежда номер на сметка, име, година на откриване на сметка, брой влогове( в структурата Deposit), валута, стойност на влога и година на създаване на влога.

Блок схема на алгоритъма

*Фиг 3. Блок схема на условие B.a*



Функция, с която е реализиран алгоритъма

void addInvestor(Investor investors[], int& numInvestors);

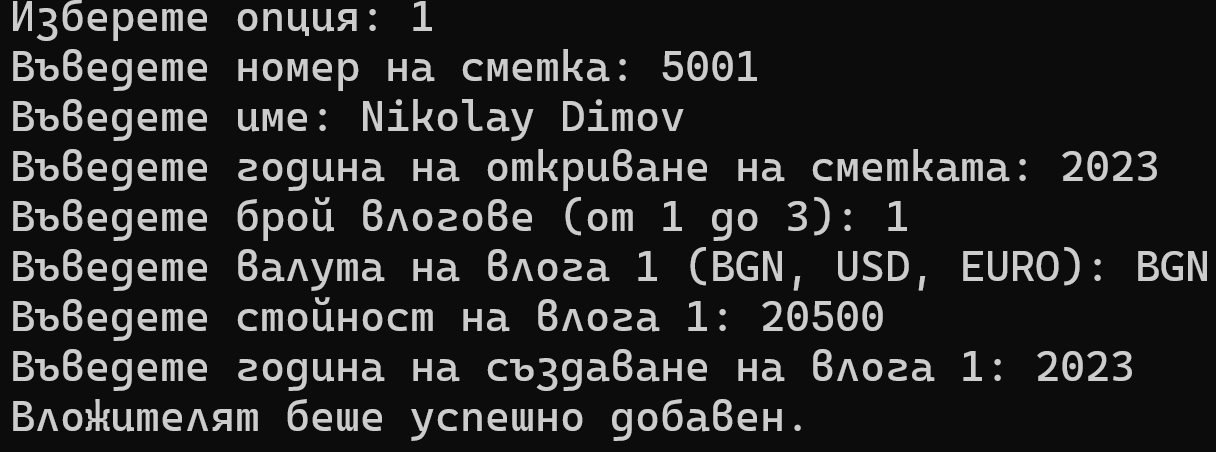
#### *Входни данни на функцията*

Investor& newInvestor - Променлива от тип структура Investor, в която се съхраняват данните за един вложител

#### *Изходни данни на функцията или данни, които се извеждат*

Няма.Функцията връща данните за студента, през параметъра Investor& newInvestor .

*Фиг 4. Снимка на фуннкцията -* void addInvestor(Investor investors[], int& numInvestors);

**

Реализация на условие B.b

Функция за добавяне на списък от вложители

Въвежда се цяло число за броя на вложителите и се извиква предишната функция за добавяне на вложител - void addInvestor(Investor investors[], int& numInvestors);

Блок схема на алгоритъма

*Фиг 5. Блок схема на условие B.b*



Функция, с която е реализиран алгоритъма

void addInvestorsList(Investor investors[], int& numInvestors);

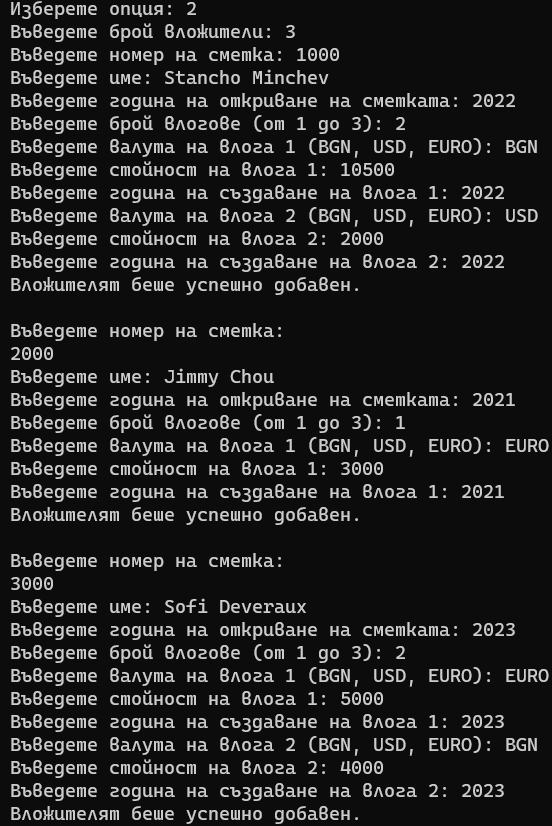
#### *Входни данни на функцията*

Int n, брой вложители.

#### *Изходни данни на функцията или данни, които се извеждат*

Няма.Функцията връща данните за студента, през параметъра Investor& newInvestor .

*Фиг 6. Снимка на фуннкцията -* void addInvestorsList(Investor investors[], int& numInvestors);



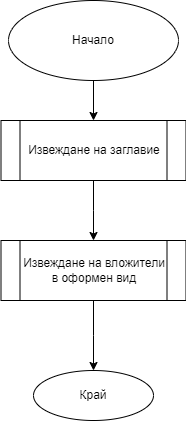
Реализация на условие C

Анализ на алгоритъма, който трябва да се реализира

Функцията void allInvestors(Investor investors[], int& numInvestors), извежда всички вложители, с техните влогове в оформен вид (ако има вложители).

Блок схема на алгоритъма

*Фиг 7. Блок схема на условие C*



Функция с която е реализиран алгоритъма

void allInvestors(Investor investors[], int& numInvestors)

#### *Входни данни на функцията*

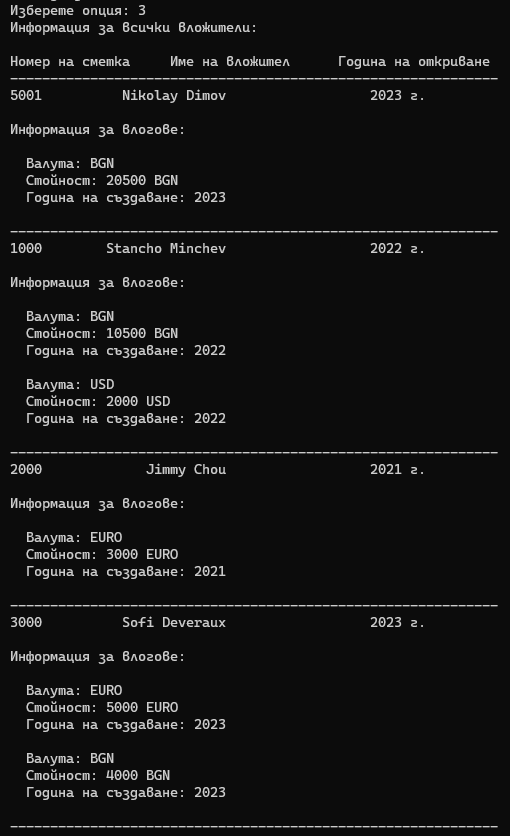
Investor investors[] - масива с инвеститори

int& numInvestors - брой инвеститори

#### *Изходни данни на функцията или данни, които се извеждат*

Няма изходни данни. Функцията отпечатва вложителите с техните данни в оформен вид.

*Фиг 8. Снимка на фуннкцията -* void allInvestors(Investor investors[], int& numInvestors)



Реализация на условие D(a)

Анализ на алгоритъма, който трябва да се реализира

Итерира се през масива и се проверява(bool hasBGNDeposit) дали вложителят има влог с валута в BGN, След това се отпечатват данните(номера на сметката, името, годината на откриване) за вложителя в оформен вид.

Блок схема на алгоритъма

*Фиг 8. Блок схема на функцията*



Функция с която е реализиран алгоритъма

void investorsWithBGNDeposit(Investor investors[], int& numInvestors)

#### *Входни данни на функцията*

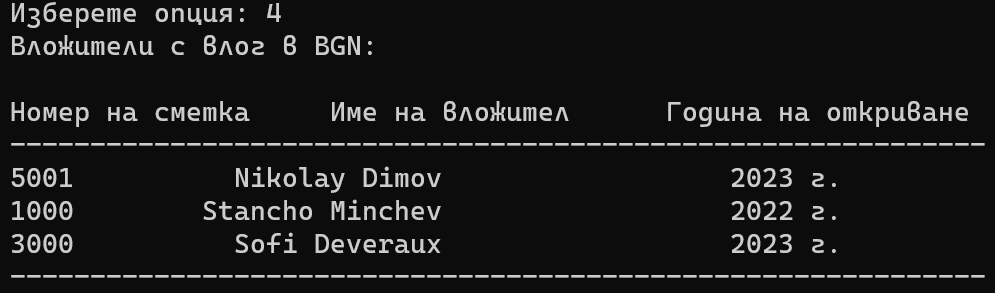
Investor investors[] - масива с инвеститори

int& numInvestors - брой инвеститори

#### *Изходни данни на функцията или данни, които се извеждат*

Няма. Отпечатват се инвеститорите с влогове в BGN.

*Фиг 9. Снимка на функцията*



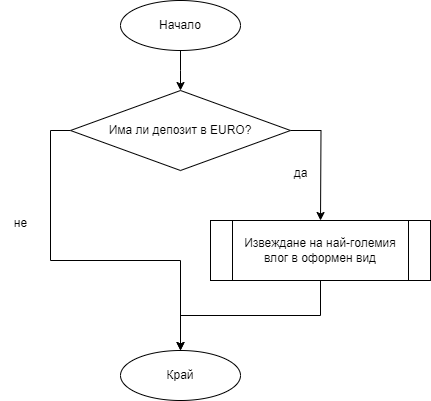
Реализация на условие D(b)

Анализ на алгоритъма, който трябва да се реализира

Итерира се през масива и се проверява дали вложителят има влог с валута в EURO, cлед това се търси, кой е влогът с най-голяма стойност(ако повече от 1) и се отпечатват данните(номера на сметката, името, годината на откриване, стойност на влога в EURO) за вложителя в оформен вид.

Блок схема на алгоритъма

*Фиг 10. Блок схема на функцията*



Функция с която е реализиран алгоритъма

void investorsWithMaxEURO(Investor investors[], int& numInvestors)

#### *Входни данни на функцията*

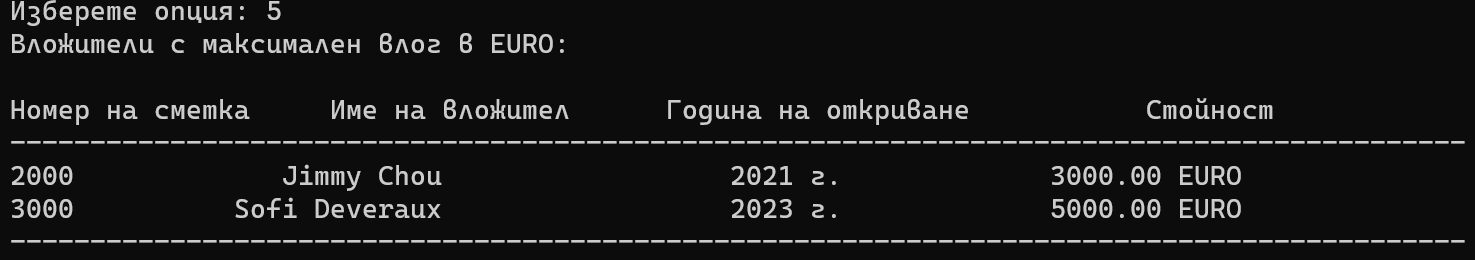
Investor investors[] - масива с инвеститори

int& numInvestors - брой инвеститори

#### *Изходни данни на функцията или данни, които се извеждат*

Няма, отпечатват се вложителите с най-голям влог в EURO

*Фиг 11. Снимка на примерно отпечатване*



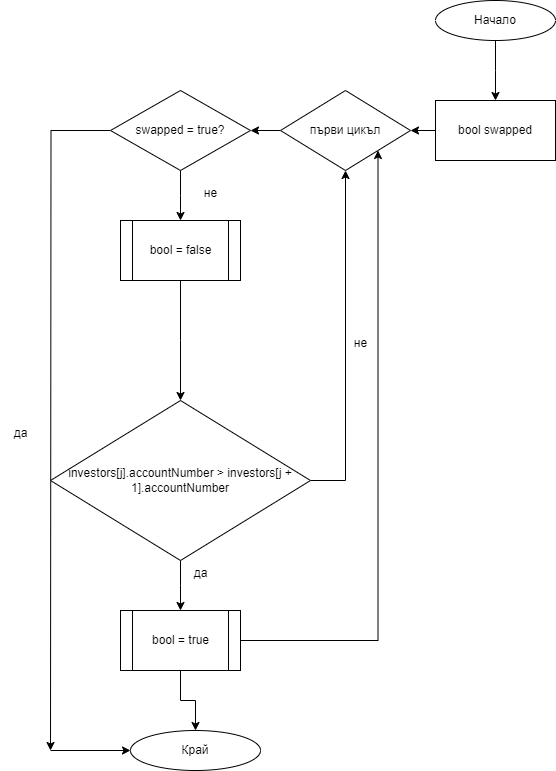
Реализация на условие E

Анализ на алгоритъма, който трябва да се реализира

Подреждане на вложителите по нарастващ ред на номер на сметка. Използван е оптимизиран bubble sort, с bool swapped;Използва се вложен цикъл, в първия от който, се проверява дали swapped е false, ако е цикъла се прекратява, противен случай се променя на false, вторият цикъл се използва bubble sort за пренареждане на вложителите по нарастващ ред на номера на сметка. Във вторият цикъл последван с проверка за номера на сметките се създава буферна променлива от тип Investor temp, която помага аз осъществяването на сортировката. При направена размяна в цикъла променливата swapped се връща в true.

Блок схема на алгоритъма

*Фиг 12. Блок схема на функцията*



Функция с която е реализиран алгоритъма

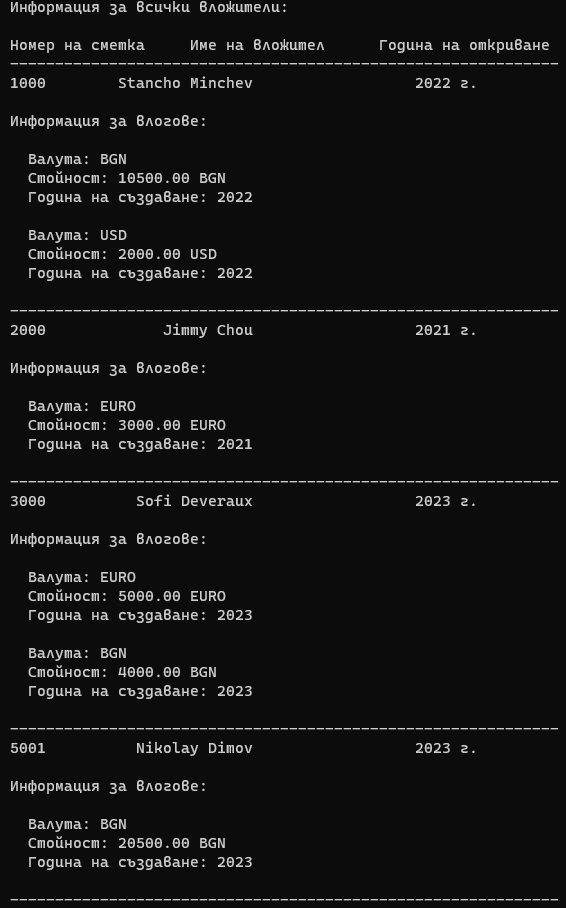
void rearrangeByAccNumber(Investor investors[], int& numInvestors)

#### *Входни данни на функцията*

Investor investors[] - масива с инвеститори

int& numInvestors - брой инвеститори

*Фиг 13. Снимка на функцията*



#### *Изходни данни на функцията или данни, които се извеждат*

Няма. Функцията сортира инвеститорите по нарастващ ред на номер на сметка

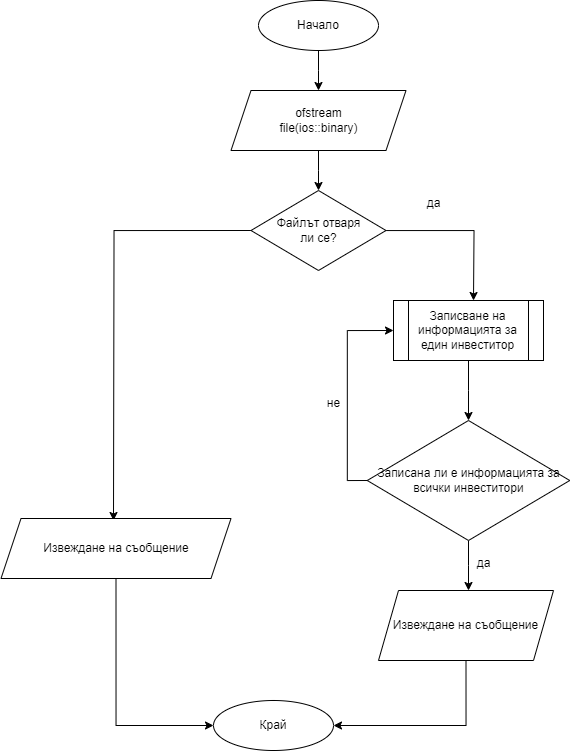
Реализация на условие F(a)

Анализ на алгоритъма, който трябва да се реализира

Данните за инвеститорите се записват в двоичен файл чрез избор от главното меню. Данните за всеки инвеститор се съхраняват на отделен ред .

Блок схема на алгоритъма

*Фиг 14. Блок схема на функцията*



Функция с която е реализиран алгоритъма

void BinaryOutputInvestors(Investor investors[], int& numInvestors, const string& filename)

#### *Входни данни на функцията*

Investor investors[],

int& numInvestors,

const string& filename

#### *Изходни данни на функцията или данни, които се извеждат*

Изходните данни се предават в двоичен файл, четат се ред по ред четенето е реализирано чрез функцията void BinaryOutputInvestors(Investor investors[], int& numInvestors, const string& filename). Данните с инвеститори се четат от двоичен файл чрез избор от подменю.

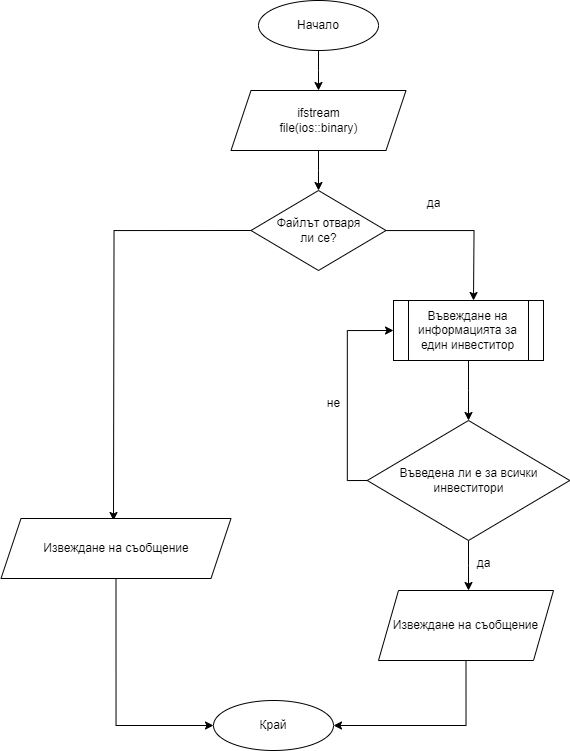
Реализация на условие F(b)

Анализ на алгоритъма, който трябва да се реализира

Данните за инвеститорите се въвеждат от двоичен файл чрез избор от главното меню.

Блок схема на алгоритъма

*Фиг 15. Блок схема на функцията*



Функция с която е реализиран алгоритъма

void BinaryInputInvestors(Investor investors[], int& numInvestors, const string& filename)

#### *Входни данни на функцията*

Investor investors[],

int& numInvestors,

const string& filename

#### *Изходни данни на функцията или данни, които се извеждат*

Няма. Данните се въвеждат от външен двоичен файл и се записват в масива.

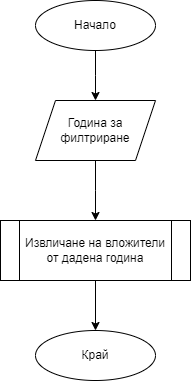
11. Реализация на условие G(a)

11.1. Анализ на алгоритъма, който трябва да се реализира

Извлича от базовия масив вложителите от дадена година на откриване на сметка и ги сортира по номер на сметка.

11.2. Блок схема на алгоритъма

*Фиг 16. Блок схема на функцията*



11.3. Функция с която е реализиран алгоритъма

void investorsByOpeningYearAndSort(Investor investors[], int& numInvestors);

Помощни функции:

rearrangeByAccNumber(filteredInvestors, numFilteredInvestors);

allInvestors(filteredInvestors, numFilteredInvestors);

11.3.1. Входни данни на функциите

Investor investors[],

int& numInvestors,

filteredInvestors,

NumFilteredInvestors

11.3.2. Изходни данни на функциите

Извеждат се инвеститорите в оформен вид, от дадената година за филтриране, посочена от потребителя.

*Фиг 17. Снимка на функцията*



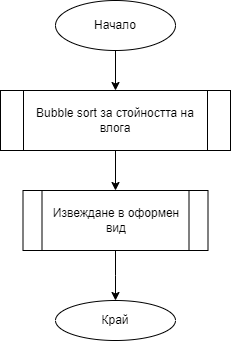
12. Реализация на условие G(b)

12.1. Анализ на алгоритъма, който трябва да се реализира

Извлича от базовия масив вложителите, които имат само един влог и ги сортира по стойност на този влог.

12.2. Блок схема на алгоритъма

*Фиг 18. Блок схема на функцията*



12.3. Функция с която е реализиран алгоритъма

investorsWithSingleDepositAndSort(Investor investors[], int& numInvestors)

12.3.1 Входни данни на функцията

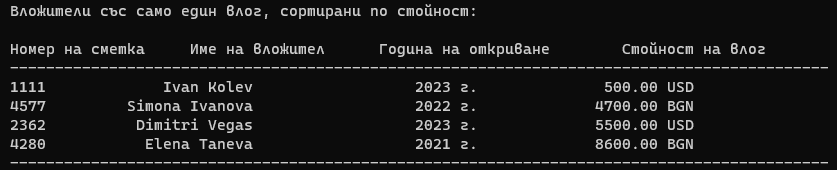
Investor investors[],

int& numInvestors

12.3.2 Изходни данни на функцията

Извеждат се инвеститорите в оформен вид, които само 1 влог и се сортират по стойност на този влог.

*Фиг 19. Снимка на функцията*



13. Реализация на условие H(a, b, c)

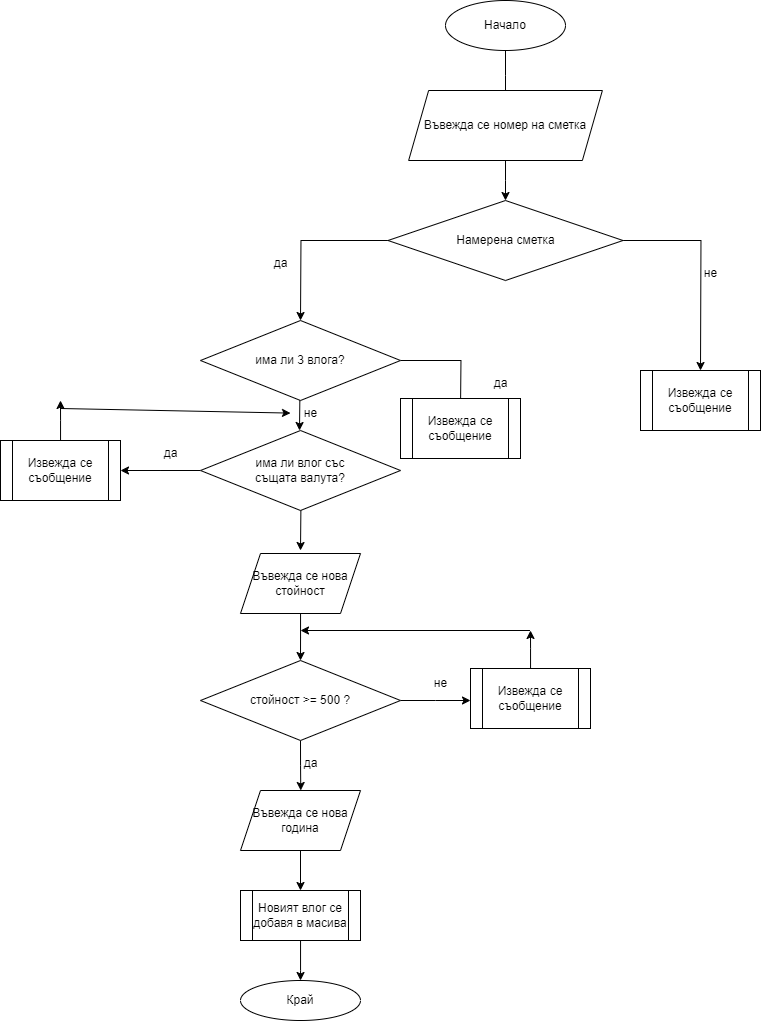
13.1 Анализ на алгоритъма, който трябва да се реализира

Всеки вложител (по номер на вложител) с по-малко от 3 влога може да му се създаде нов влог за валута, по която няма влог с минимална стойност 500;

Ако не съществува вложител с този номер, се извежда подходящо съобщение; При опит за добавяне на влог на вложител с 3 влога се изписва подходящо съобщение;

13.2. Блок схема на алгоритъма

*Фиг 20. Блок схема на функцията*



13.3 Функция с която е реализиран алгоритъма

void addNewDeposit(Investor& investor)

void addNewDepositForAccountNumber(Investor investors[], int numInvestors)

13.3.1 Входни данни на функцията

double newValue;

int newYear;

char newCurrency[20];

bool hasSameCurrency;

Deposit newDeposit;

strcpy\_s(newDeposit.currency, newCurrency);

newDeposit.value = newValue;

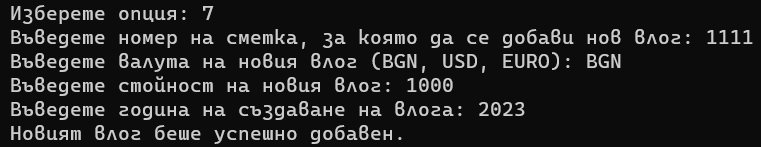
newDeposit.creationYear = newYear;

investor.deposits[investor.numDeposits++] = newDeposit;

13.3.2 Изходни данни на функцията

Няма, функцията добавя влог по даден номер на сметка, което може да се види нагледно с помощта на функцията за извеждане на всички инвеститори.

*Фиг 21. Снимка на функцията*



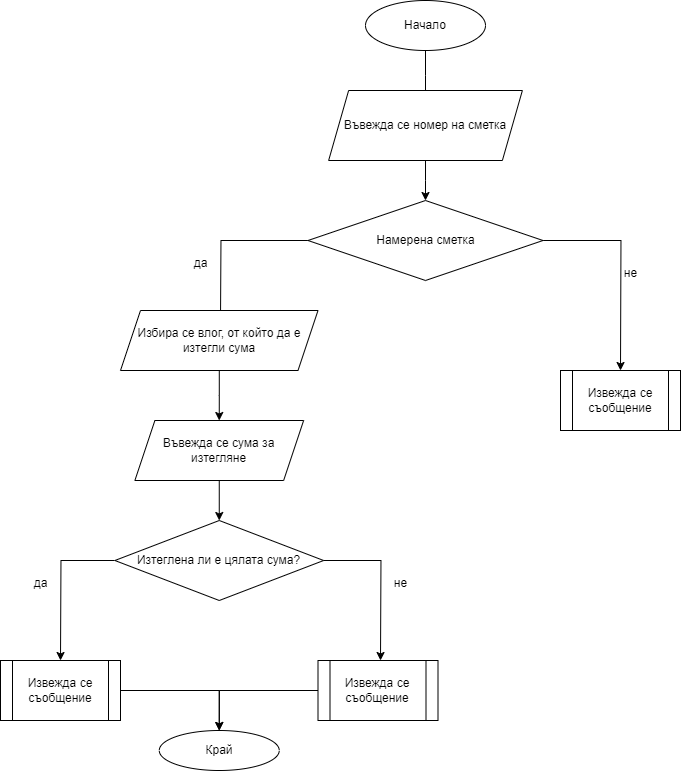
14. Реализация на условие I(a)

14.1. Анализ на алгоритъма, който трябва да се реализира

Въвеждат се номер на вложител, избира се влога от който ще се тегли и се изтегля цялата или част от сумата по влога.

14.2. Блок схема на алгоритъма

*Фиг 22. Блок схема на функцията*



14.3 Функция с която е реализиран алгоритъма

void depositWithdrawl(Investor investors[], int& numInvestors)

14.3.1 Входни данни на функцията

int accNumber;

Investor investors[],

int& numInvestors

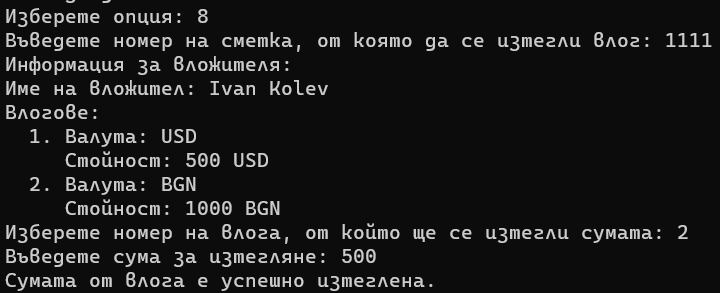
int selectedDeposit;

double withdrawalAmount;

14.3.2 Изходни данни на функцията

Няма, функцията търси вложителите(по номер на сметка), извежда информация за влоговете и се избира влог от който да се изтегли цялата сума или част от нея.

*Фиг 23. Снимка на функцията*



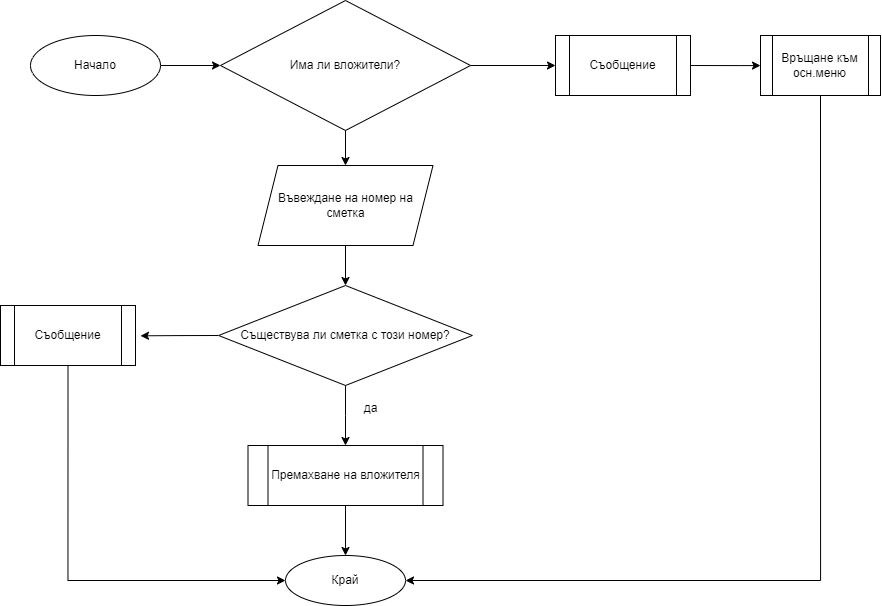
15. Реализация на условие I(b)

15.1 Алгоритъм за премахване на вложител.

Алгоритъма се изпълнява чрез избирането на определена опция от главното меню. Ако няма съществуващи вложители се извежда съобщение. Иначе клиента трябва да въведете номера на сметката, която желае да премахне. Това се случва чрез 2 функции, една от която е помощна. Тя намира индекса на сметката, която трябва да бъде премахната. Основната включва диалога, който се извежда към клиента и цикъл, който началото му е индекса на елемента, който ще бъде премахнат до броя на вложители в масива -1, защото ще бъде премахната сметка. Ако няма такъв индекс помощната функция връща невъзможен индекс, който да участва в масива и той е -1. След премахването на сметката броят на вложителите се намаля с 1. Ако клиента е въвел сметка, която я няма в масива се извежда съобщение за липсата и.

15.2 Блок схема на алгоритъма.

*Фиг 24. Блок схема на функцията*



15.3 Функции с които е реализиран алгоритъма

int findIndexOfInvestor(Investor investors[], int& numInvestors,int& accNumber)

void closeInvestorProfile(Investor investors[], int& numInvestors)

15.3.1 Входни данни на функцията

Investor investors[]

int& numInvestors

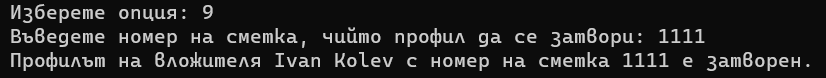
int accNumber

int investorIndexToRemove

15.3.2 Изходни данни на функцията

Помощната функция има единствено изходни данни. Тя връща индекса на елемента, който трябва да бъде премахнат от масива, ако няма такъв връща -1. Основната функция няма изходни данни.

*Фиг 25. Снимка на функцията*



16. Автоматично запазване и попълване на данните след стартирането и затварянето на програмата.

16.1 Алгоритъм за запазване и попълване на данните след стартирането и затварянето

Попълването е осъществено чрез функцията BinaryInputInvestors(investors, numInvestors, binaryFileName), като се поставя преди осъществяването на основното меню.

Запазването е направено като се използва функцията BinaryOutputInvestors(investors, numInvestors, binaryFileName), като се поставя след главното меню и преди изпълянването на главната функция int main().

Упътване за употреба

Представете какво трябва да се въведе за правилното използване на програмата, какво се извежда в резултат на нея. Какви са очакваните резултати и какво ще се случи, ако се въведат невалидни данни. Изтрийте този текст, след като се запознаете с него.

Впишете съответната част от проекта

Очаквани входни данни

Моля, опишете.

Очакван резултат от изпълнението на конкретния фрагмент от проекта

Моля, опишете.

Впишете съответната част от проекта

Очаквани входни данни

Моля, опишете.

Очакван резултат от изпълнението на конкретния фрагмент от проекта

Моля, опишете.

Примерно действие на програмата

Условие A

Снимка на изгледа с примерни входни данни

Моля, поставете снимката тук.

Снимка на изгледа с примерни изходни данни

Моля, поставете снимката тук.

Условие B

Снимка на изгледа с примерни входни данни

Моля, поставете снимката тук.

Снимка на изгледа с примерни изходни данни

Моля, поставете снимката тук.

Условие C

Снимка на изгледа с примерни входни данни

Моля, поставете снимката тук.

Снимка на изгледа с примерни изходни данни

Моля, поставете снимката тук.

Условие D

Снимка на изгледа с примерни входни данни

Моля, поставете снимката тук.

Снимка на изгледа с примерни изходни данни

Моля, поставете снимката тук.

Условие E

Снимка на изгледа с примерни входни данни

Моля, поставете снимката тук.

Снимка на изгледа с примерни изходни данни

Моля, поставете снимката тук.

Условие F

Снимка на изгледа с примерни входни данни

Моля, поставете снимката тук.

Снимка на изгледа с примерни изходни данни

Моля, поставете снимката тук.

Допълнение първо

Снимка на изгледа с примерни входни данни

Моля, поставете снимката тук.

Снимка на изгледа с примерни изходни данни

Моля, поставете снимката тук.

Допълнение второ

Снимка на изгледа с примерни входни данни

Моля, поставете снимката тук.

Снимка на изгледа с примерни изходни данни

Моля, поставете снимката тук.

Допълнение трето

Снимка на изгледа с примерни входни данни

Моля, поставете снимката тук.

Снимка на изгледа с примерни изходни данни

Моля, поставете снимката тук.

Приложение

#include <iostream>

#include <string>

#include <iomanip>

#include <fstream>

#include <cstring>

using namespace std;

// Структура за влог

struct Deposit {

double value;

int creationYear;

char currency[20];

};

// Структура за вложител

struct Investor {

int accountNumber;

char name[30];

Deposit deposits[3]; // масив от структура

int numDeposits; // брой влогове

int openingYear;

};

// Функция за добавяне на един вложител с влогове

void addInvestor(Investor investors[], int& numInvestors) {

Investor& newInvestor = investors[numInvestors];

cout << "Въведете номер на сметка: ";

cin >> newInvestor.accountNumber;

cout << "Въведете име: ";

cin.ignore(); // чистене на потока

cin.getline(newInvestor.name, 30); // getline за c-string

cout << "Въведете година на откриване на сметката: ";

cin >> newInvestor.openingYear;

do {

cout << "Въведете брой влогове (от 1 до 3): ";

cin >> newInvestor.numDeposits;

if (newInvestor.numDeposits < 1 || newInvestor.numDeposits > 3) {

cout << "Броят на влоговете трябва да е между 1 и 3. Моля, въведете отново.\n";

}

} while (newInvestor.numDeposits < 1 || newInvestor.numDeposits > 3);

for (int i = 0; i < newInvestor.numDeposits; ++i) {

while (true) {

cout << "Въведете валута на влога " << i + 1 << " (BGN, USD, EURO): ";

cin.ignore();

cin.getline(newInvestor.deposits[i].currency, 20);

if (strcmp(newInvestor.deposits[i].currency, "BGN") == 0 ||

strcmp(newInvestor.deposits[i].currency, "USD") == 0 ||

strcmp(newInvestor.deposits[i].currency, "EURO") == 0) {

break; // излиза се от цикъла, ако е валидна валутата

}

cout << "Невалидна стойност. Въведете валидна валута.\n";

}

cout << "Въведете стойност на влога " << i + 1 << ": ";

cin >> newInvestor.deposits[i].value;

cout << "Въведете година на създаване на влога " << i + 1 << ": ";

cin >> newInvestor.deposits[i].creationYear;

}

cout << "Вложителят беше успешно добавен.\n\n";

numInvestors++;

}

// Добавяне на списък от вложители

void addInvestorsList(Investor investors[], int& numInvestors) {

int n;

do {

cout << "Въведете брой вложители: ";

cin >> n;

if (n > 50) {

cout << "Достигнат е максималният брой вложители! Моля, въведете отново.\n";

}

else if (n <= 0) {

cout << "Грешка! Броят на вложителите трябва да бъде по-голям от 0. Моля, въведете отново.\n";

}

} while (n > 50 || n <= 0);

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

addInvestor(investors, numInvestors);

}

}

// Илюлстарция в оформен вид

void allInvestors(Investor investors[], int& numInvestors) {

cout << "Информация за всички вложители:\n\n";

cout << setw(10) << "Номер на сметка" << setw(20) << "Име на вложител" << setw(25) << "Година на откриване" << "\n";

cout << setw(62) << setfill('-') << "\n" << setfill(' ');

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

const Investor& currentInvestor = investors[i];

cout << setw(4) << currentInvestor.accountNumber;

cout << setw(23) << currentInvestor.name;

cout << setw(22) << currentInvestor.openingYear << " г.\n\n";

cout << "Информация за влогове: \n\n";

for (int j = 0; j < currentInvestor.numDeposits; ++j) {

const Deposit& currentDeposit = currentInvestor.deposits[j];

cout << " Валута: " << currentDeposit.currency << "\n";

cout << " Стойност: " << currentDeposit.value << " " << currentDeposit.currency << endl;

cout << " Година на създаване: " << currentDeposit.creationYear << "\n\n";

}

cout << setw(62) << setfill('-') << "\n" << setfill(' ');

}

}

// Вложители с влог в BGN

void investorsWithBGNDeposit(Investor investors[], int& numInvestors) {

cout << "Вложители с влог в BGN:\n\n";

cout << setw(10) << "Номер на сметка" << setw(20) << "Име на вложител" << setw(25) << "Година на откриване" << "\n";

cout << setw(62) << setfill('-') << "\n" << setfill(' ');

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

const Investor& currentInvestor = investors[i];

// proverka dali depozita e v BGN

bool hasBGNDeposit = false;

for (int j = 0; j < currentInvestor.numDeposits; ++j) {

if (strcmp(currentInvestor.deposits[j].currency, "BGN") == 0) {

hasBGNDeposit = true;

break;

}

}

if (hasBGNDeposit) {

cout << setw(4) << currentInvestor.accountNumber;

cout << setw(23) << currentInvestor.name;

cout << setw(22) << currentInvestor.openingYear << " г.\n";

}

}

cout << setw(62) << setfill('-') << "\n" << setfill(' ');

}

// Вложители с максимален влог в EURO

void investorsWithMaxEURO(Investor investors[], int& numInvestors) {

cout << "Вложители с максимален влог в EURO:\n\n";

cout << setw(10) << "Номер на сметка" << setw(20) << "Име на вложител" << setw(25) << "Година на откриване" << setw(20) << "Стойност\n";

cout << setw(92) << setfill('-') << "\n" << setfill(' ');

// Намиране на максималния влог в EURO за всеки вложител

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

const Investor& currentInvestor = investors[i];

double maxDeposit = 0.0;

// Намиране на максималния влог в EURO за текущия вложител

for (int j = 0; j < currentInvestor.numDeposits; ++j) {

if (strcmp(currentInvestor.deposits[j].currency, "EURO") == 0 && currentInvestor.deposits[j].value > maxDeposit) {

maxDeposit = currentInvestor.deposits[j].value;

}

}

// Извеждане на вложителите с техните най-големи влози в EURO

if (maxDeposit > 0.0) {

cout << setw(4) << currentInvestor.accountNumber;

cout << setw(23) << currentInvestor.name;

cout << setw(22) << currentInvestor.openingYear << " г.";

cout << fixed << setprecision(2) << setw(20) << maxDeposit << " EURO" << endl;

}

}

cout << setw(92) << setfill('-') << "\n" << setfill(' ');

}

// По нарастващ ред на номера на сметка // bubble sorting

void rearrangeByAccNumber(Investor investors[], int& numInvestors) {

bool swapped;

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

swapped = false;

for (int j = 0; j < numInvestors - i - 1; j++) {

if (investors[j].accountNumber > investors[j + 1].accountNumber) {

Investor temp = investors[j];

investors[j] = investors[j + 1];

investors[j + 1] = temp;

swapped = true;

}

}

if (!swapped) {

break;

}

}

}

// Извлича от базовия масив вложителите от дадена година на откриване на сметка и ги сортира по номер на сметка:

void investorsByOpeningYearAndSort(Investor investors[], int& numInvestors) {

int targetYear;

cout << "Въведете година за филтриране: ";

cin >> targetYear;

// масив за съхраняване на вложители от дадена година

Investor filteredInvestors[50];

int numFilteredInvestors = 0;

// извличане на вложителите от дадената година

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

if (investors[i].openingYear == targetYear) {

filteredInvestors[numFilteredInvestors++] = investors[i];

}

}

// сортиране по номер на сметка

rearrangeByAccNumber(filteredInvestors, numFilteredInvestors);

// извеждане

allInvestors(filteredInvestors, numFilteredInvestors);

}

// Извлича от базовия масив вложителите, които имат само един влог и ги сортира по стойност на този влог:

void investorsWithSingleDepositAndSort(Investor investors[], int& numInvestors) {

cout << "Вложители със само един влог, сортирани по стойност:\n\n";

cout << setw(10) << "Номер на сметка" << setw(20) << "Име на вложител" << setw(25) << "Година на откриване" << setw(25) << "Стойност на влог\n";

cout << setw(92) << setfill('-') << "\n" << setfill(' ');

// bubble sort по стойност на влога

bool swapped;

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

swapped = false;

for (int j = 0; j < numInvestors - i - 1; ++j) {

Deposit& deposit1 = investors[j].deposits[0];

Deposit& deposit2 = investors[j + 1].deposits[0];

if (deposit1.value > deposit2.value) {

// swap с буферна клетка, ако стойността е по-висока

Investor temp = investors[j];

investors[j] = investors[j + 1];

investors[j + 1] = temp;

swapped = true;

}

}

// ако няма размяна, масивът е сортиран, излизане от цикъла преждевременно

if (!swapped) {

break;

}

}

// дисплей

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

Investor& currentInvestor = investors[i];

// проверка дали вложителя има 1 влог

if (currentInvestor.numDeposits == 1) {

Deposit& singleDeposit = currentInvestor.deposits[0];

cout << setw(4) << currentInvestor.accountNumber;

cout << setw(23) << currentInvestor.name;

cout << setw(22) << currentInvestor.openingYear << " г.";

cout << fixed << setprecision(2) << setw(20) << singleDeposit.value << " " << singleDeposit.currency << endl;

}

}

cout << setw(92) << setfill('-') << "\n" << setfill(' ');

}

// Добавяне на влог за вложител с по-малко от 3 влога (следващите две функции са по това условие)

void addNewDeposit(Investor& investor) {

double newValue;

int newYear;

char newCurrency[20];

if (investor.numDeposits >= 3) {

cout << "Вложителят вече има 3 влога и не може да добави повече.\n";

return;

}

bool hasSameCurrency = false;

// Проверка дали вложителят вече има влог с тази валута

while (true) {

cout << "Въведете валута на новия влог (BGN, USD, EURO): ";

cin.ignore();

cin.getline(newCurrency, 20);

// Проверка дали вложителят вече има влог с тази валута

for (int i = 0; i < investor.numDeposits; i++) {

if (strcmp(investor.deposits[i].currency, newCurrency) == 0) {

cout << "Вложителят има влог с тази валута. ";

hasSameCurrency = true;

break;

}

}

if (!hasSameCurrency) {

break;

}

hasSameCurrency = false;

cout << "Моля, въведете различна валута.\n";

}

do {

cout << "Въведете стойност на новия влог: ";

cin >> newValue;

// Валидация за стойността

if (newValue >= 500) {

break;

}

cout << "Стойността на новия влог трябва да бъде поне 500.\n";

} while (true);

cout << "Въведете година на създаване на влога: ";

cin >> newYear;

Deposit newDeposit;

strcpy\_s(newDeposit.currency, newCurrency);

newDeposit.value = newValue;

newDeposit.creationYear = newYear;

// Добавя се новия влог в масива

investor.deposits[investor.numDeposits++] = newDeposit;

cout << "Новият влог беше успешно добавен.\n";

}

void addNewDepositForAccountNumber(Investor investors[], int numInvestors) {

while (true) {

cout << "Въведете номер на сметка, за която да се добави нов влог: ";

int accNumber;

cin >> accNumber;

bool found = false;

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

if (investors[i].accountNumber == accNumber) {

addNewDeposit(investors[i]);

found = true;

break;

}

}

if (found) {

// ако е намерен, изход от цикъла

break;

}

else {

cout << "Несъществува вложител с такъв номер. Моля, опитайте отново.\n\n";

// изчистване на стрийма

cin.clear();

cin.ignore();

}

}

}

// Справки с подменю

void submenuByAccNumber(Investor investors[], int& numInvestors) {

int subChoice;

do {

cout << "\nПодменю 6\n";

cout << "1. - Изведи вложителите по нарастващ ред на номера на сметката\n";

cout << "2. - Вложителите от дадена година на откриване на сметка\n";

cout << "3. - Вложителите, които имат само един влог\n";

cout << "4. - Връщане към основното меню\n";

cin >> subChoice;

if (subChoice > 4 && subChoice < 1) {

cout << "Изберете цяло число от 1 до 4.\n\n";

}

switch (subChoice) {

case 1:

rearrangeByAccNumber(investors, numInvestors);

allInvestors(investors, numInvestors);

break;

case 2:

// Извлича от базовия масив вложителите от дадена година на откриване на сметка и ги сортира по номер на сметка:

investorsByOpeningYearAndSort(investors, numInvestors);

break;

case 3:

// Извлича от базовия масив вложителите, които имат само един влог и ги сортира по стойност на този влог:

investorsWithSingleDepositAndSort(investors, numInvestors);

break;

case 4:

cout << "Връщане към основното меню.\n";

break;

}

} while (subChoice != 4);

}

// Изтегляне на влог

void depositWithdrawl(Investor investors[], int& numInvestors) {

while (true) {

cout << "Въведете номер на сметка, от която да се изтегли влог: ";

int accNumber;

cin >> accNumber;

bool found = false;

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

if (investors[i].accountNumber == accNumber) {

cout << "Информация за вложителя:\n";

cout << "Име на вложител: " << investors[i].name << endl;

cout << "Влогове:\n";

for (int j = 0; j < investors[i].numDeposits; ++j) {

const Deposit& currentDeposit = investors[i].deposits[j];

cout << " " << j + 1 << ". Валута: " << currentDeposit.currency << "\n";

cout << " Стойност: " << currentDeposit.value << " " << currentDeposit.currency << endl;

}

cout << "Изберете номер на влога, от който ще се изтегли сумата: ";

int selectedDeposit;

cin >> selectedDeposit;

if (selectedDeposit >= 1 && selectedDeposit <= investors[i].numDeposits) {

selectedDeposit--; // Превръща се в масивен индекс

cout << "Въведете сума за изтегляне: ";

double withdrawalAmount;

cin >> withdrawalAmount;

if (withdrawalAmount > 0 && withdrawalAmount <= investors[i].deposits[selectedDeposit].value) {

// Изтегляне на сумата

investors[i].deposits[selectedDeposit].value -= withdrawalAmount;

cout << "Сумата от влога е успешно изтеглена.\n";

if (investors[i].deposits[selectedDeposit].value == 0) {

cout << "Изтеглена е цялата сума. Няма налична стойност за теглене" << endl;

}

}

else {

cout << "Грешка! Въведената сума за изтегляне е по-голяма от наличната в сметката.\n";

}

}

else {

cout << "Невалиден номер на влог. Моля, опитайте отново.\n";

}

found = true;

break;

}

}

if (found) {

// ако е намерен, изход от цикъла

break;

}

else {

cout << "Несъществува вложител с такъв номер. Моля, опитайте отново.\n\n";

// изчистване на стрийма

cin.clear();

cin.ignore();

}

}

}

// Затваряне на профил

int findIndexOfInvestor(Investor investors[], int& numInvestors,int& accNumber) {

// итерира се през масива и като се открие съответствие в номерата на сметките

// се връща индекса на инвеститора със съответната сметка

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

if (investors[i].accountNumber == accNumber) {

return i;

}

}

return -1;

}

void closeInvestorProfile(Investor investors[], int& numInvestors) {

// премахване на профил на инвеститор от масива

if (numInvestors <= 0) {

cout << "Няма добавени вложители.\n\n";

}

else {

cout << "Въведете номер на сметка, чийто профил да се затвори: ";

int accNumber;

cin >> accNumber;

int investorIndexToRemove = findIndexOfInvestor(investors, numInvestors, accNumber);

if (investorIndexToRemove != -1) {

cout << "Профилът на вложителя " << investors[investorIndexToRemove].name

<< " с номер на сметка " << accNumber << " е затворен.\n";

for (int i = investorIndexToRemove; i < numInvestors - 1; ++i) {

investors[i] = investors[i + 1];

}

--numInvestors;

}

else {

cout << "Несъществува вложител с такъв номер. Моля, опитайте отново.\n\n";

// Изчистване на стрийма

cin.clear();

cin.ignore();

}

}

}

// Извеждане на масива с книги във файл(двоичен)

void BinaryOutputInvestors(Investor investors[], int& numInvestors, const string& filename) {

ofstream file(filename, ios::binary);

if (file.is\_open()) {

file.write((const char\*)(&numInvestors), sizeof(numInvestors));

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

file.write((const char\*)(&investors[i]), sizeof(Investor));

}

cout << "Информацията беше записана в двоичен файл.\n";

file.close();

}

else {

cout << "Файлът не може да бъде намерен или вече съществува.\n";

}

}

// Въвеждане на масива с книги от файл(двоичен)

void BinaryInputInvestors(Investor investors[], int& numInvestors, const string& filename) {

ifstream file(filename, ios::binary);

if (file.is\_open()) {

file.read((char\*)&numInvestors, sizeof(numInvestors));

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

file.read((char\*)&investors[i], sizeof(Investor));

}

cout << "Информацията беше прочетена от файла.\n";

file.close();

}

else {

cout << "Файлът не може да бъде намерен.\n";

}

}

// Данните в програмата да се четат от файл (текстов).

int loadInvestorsFromFile(const char fileName[], Investor investors[], int& numInvestors)

{

ifstream file(fileName, ios::app);

if (!file)

{

cout << endl << "Файлът не може да бъде отворен!";

return -1;

}

int i = 0;

while (!file.eof())

{

file >> investors[i].accountNumber;

file.ignore();

file.getline(investors[i].name, sizeof(investors[i].name));

file >> investors[i].openingYear;

file >> investors[i].numDeposits;

for (int j = 0; j < investors[i].numDeposits; ++j)

{

file.ignore();

file.getline(investors[i].deposits[j].currency, sizeof(investors[i].deposits[j].currency));

file >> investors[i].deposits[j].value;

file >> investors[i].deposits[j].creationYear;

}

i++;

}

file.close();

cout << "Готово";

return 0;

}

// Данните в програмата да се отпечатват във файл (текстов).

int saveInvestorsToFile(const char fileName[], Investor investors[], int& numInvestors)

{

ofstream file(fileName, ios::app); /\*Данните в програмата да могат да се запазват във файл(текстови) между две стартирания на програмата.\*/

if (!file)

{

cout << endl << "Файлът не може да бъде отворен!";

return -1;

}

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

{

file << investors[i].accountNumber << '\n';

file << investors[i].name << '\n';

file << investors[i].openingYear << '\n';

file << investors[i].numDeposits << '\n';

for (int j = 0; j < investors[i].numDeposits; ++j)

{

file << investors[i].deposits[j].currency << '\n';

file << investors[i].deposits[j].value << '\n';

file << investors[i].deposits[j].creationYear << '\n';

}

file << investors[i].openingYear << '\n';

}

file.close();

cout << "Готово";

return 0;

}

// Маин функция с меню

int main() {

setlocale(LC\_ALL, "Bulgarian");

const string binaryFileName = "binary\_file.dat";

Investor investors[50];

int numInvestors = 0;

int choice;

do {

cout << "\n Меню\n";

cout << "1. Добави нов вложител\n";

cout << "2. Добави списък от вложители\n";

cout << "3. Изведи всички вложители\n";

cout << "4. Изведи вложите с BGN влог\n";

cout << "5. Изведи вложите с максимален влог в EURO\n";

cout << "6. Справки за влоговете с под меню\n";

cout << "7. Добавяне на влог за вложители с по-малко от 3 влога\n";

cout << "8. Изтегляне на сума от влог, по номер на сметка\n";

cout << "9. Закриване на банков профил\n";

cout << "10. Подменю за вход/изход с двоични файлове\n";

cout << "11. Подменю за вход/изход с текстови файлове\n";

cout << "12. Изход\n";

cout << "Изберете опция: ";

cin >> choice;

if (choice > 12 && choice < 1) {

cout << "Изберете цяло число от 1 до 12.\n\n";

}

switch (choice) {

case 1:

addInvestor(investors, numInvestors);

break;

case 2:

addInvestorsList(investors, numInvestors);

break;

case 3:

allInvestors(investors, numInvestors);

break;

case 4:

investorsWithBGNDeposit(investors, numInvestors);

break;

case 5:

investorsWithMaxEURO(investors, numInvestors);

break;

case 6:

// подменю

submenuByAccNumber(investors, numInvestors);

break;

case 7:

addNewDepositForAccountNumber(investors, numInvestors);

break;

case 8:

depositWithdrawl(investors, numInvestors);

break;

case 9:

closeInvestorProfile(investors, numInvestors);

break;

case 10: {

int submenuChoice;

do {

cout << "\nПод Меню\n";

cout << "1. Въвеждане на масива с книги от файл(двоичен)\n";

cout << "2. Извеждане на масива с книги във файл(двоичен)\n";

cout << "3. Връщане към основното меню\n";

cout << "Изберете опция: ";

cin >> submenuChoice;

if (submenuChoice > 3 && submenuChoice < 1) {

cout << "Изберете цяло число от 1 до 3.\n\n";

}

switch (submenuChoice) {

case 1:

BinaryInputInvestors(investors, numInvestors, binaryFileName);

break;

case 2:

BinaryOutputInvestors(investors, numInvestors, binaryFileName);

break;

case 3:

cout << "Връщане към основното меню.\n";

break;

}

} while (submenuChoice != 3);

break;

}

case 11: {

int submenuChoice;

do {

cout << "\nПод Меню\n";

cout << "1. Въвеждане на масива с книги от файл(текстов)\n";

cout << "2. Извеждане на масива с книги във файл(текстов)\n";

cout << "3. Връщане към основното меню\n";

cout << "Изберете опция: ";

cin >> submenuChoice;

if (submenuChoice > 3 && submenuChoice < 1) {

cout << "Изберете цяло число от 1 до 3.\n\n";

}

switch (submenuChoice) {

case 1:

loadInvestorsFromFile("info.txt", investors, numInvestors);

break;

case 2:

saveInvestorsToFile("info.txt", investors, numInvestors);

break;

case 3:

cout << "Връщане към основното меню.\n";

break;

}

} while (submenuChoice != 3);

break;

}

case 12:

cout << "Излизане от програмата.\n";

return 0;

}

} while (choice != 12);

return 0;

}