Реферат до семестрової роботи

Семирог-Орлика Єгора

Тема: Реалізація функціоналу банківської системи мовою програмування С

Вступ

Сучасні інформаційні системи вимагають інтеграції різноманітних функціоналів, що підтримують ефективне управління фінансовими процесами. У даному рефераті розглядається приклад створення банківської системи мовою програмування С, яка дозволяє управляти клієнтами, депозитами, цінними паперами, а також зберігати й обробляти дані у форматі CSV.

Опис програми

Програма складається з основного файлу main.c, заголовкового файлу Bank.h, а також файла реалізації функцій Bank.c. Вона організована у вигляді модульної системи для розділення логіки обробки даних і взаємодії з користувачем.

Основні структури даних:

- **Client**: Зберігає дані про клієнта банку (ім’я, баланс, відсоткову ставку депозиту,

```c
/* Base structure of Client */
typedef struct Client {
    char name[50];          ///< Ім'я клієнта
    double balance;         ///< Баланс клієнта
    double deposit_rate;    ///< Процентна ставка депозиту
    int deposit_duration;   ///< Термін депозиту (місяців)
} Client;
```

  термін депозиту).

- **Security**: Описує цінний папір (назва, ціна, кількість, тип).

```
/* Base structure of Security */
typedef struct Security {
    char name[50];          ///< Назва цінного паперу
    double price;           ///< Ціна цінного паперу
    int quantity;           ///< Кількість цінного паперу
    char type[20];          ///< Тип цінного паперу (наприклад, акція, облігація)
} Security;
```

- **Bank**: Головна структура для управління банком (готівка, масив клієнтів, масив

```
/* Base structure of Bank */
typedef struct Bank {
    double cash;            ///< Готівка банку
    Client* clients;        ///< Масив клієнтів
    int client_count;       ///< Кількість клієнтів
    Security* securities;   ///< Масив цінних паперів
    int security_count;     ///< Кількість цінних паперів
} Bank;
```

цінних паперів).

Функціонал програми

1. **Ініціалізація банку (init_bank)**
   Функція створює початкову структуру банку з заданою сумою готівки, нульовою

```
// Ініціалізація банку з початковою сумою готівки
Bank init_bank(double initial_cash) {
    Bank bank;
    bank.cash = initial_cash;          // "Початковий баланс банку"
    bank.client_count = 0;             // "Кількість клієнтів"
    bank.clients = NULL;               // "Масив клієнтів (поки що порожній)"
    bank.security_count = 0;           // "Кількість цінних паперів"
    bank.securities = NULL;            // "Масив цінних паперів (поки що порожній)"
    return bank;
}
```

   кількістю клієнтів і цінних паперів.

2. **Додавання клієнта (add_client)**
   Ця функція дозволяє додати нового клієнта з початковим депозитом, відсотковою ставкою та строком депозиту. При цьому відбувається:

- Динамічне виділення пам’яті для нового запису клієнта.
- Збільшення загальної кількості клієнтів банку.

- Зарахування депозиту на баланс банку.

```
// Додавання нового клієнта до банку
void add_client(Bank* bank, char* name, double initial_deposit, double deposit_rate, int duration) {
    // Виділення додаткової пам'яті для нового клієнта

    bank->clients = realloc(bank->clients, (bank->client_count + 1) * sizeof(Client));
    // "Збереження інформації про клієнта"

    strcpy(bank->clients[bank->client_count].name, name);
    bank->clients[bank->client_count].balance = initial_deposit;
    bank->clients[bank->client_count].deposit_rate = deposit_rate;
    bank->clients[bank->client_count].deposit_duration = duration;
    bank->client_count++;          // "Збільшення кількості клієнтів"
    bank->cash += initial_deposit; // "Збільшення готівки банку"
```

3. **Додавання цінного паперу (add_security)**
   Функція дозволяє додати новий вид цінного паперу з його назвою, ціною, кількістю та типом (наприклад, акція чи облігація).

```
// Додавання нового цінного паперу до банку
void add_security(Bank* bank, char* name, double price, int quantity, char* type) {
    // "Виділення додаткової пам'яті для нового цінного паперу"
    bank->securities = realloc(bank->securities, (bank->security_count + 1) * sizeof(Security));
    // "Збереження інформації про цінний папір"
    strcpy(bank->securities[bank->security_count].name, name);
    bank->securities[bank->security_count].price = price;
    bank->securities[bank->security_count].quantity = quantity;
    strcpy(bank->securities[bank->security_count].type, type);
    bank->security_count++;        // "Збільшення кількості цінних паперів"
```

4. **Оновлення цін цінних паперів (update_securities)**
   Ціни на цінні папери змінюються випадковим чином у межах від -10% до +10%. Це дозволяє моделювати динаміку ринку.

```
// Оновлення цін цінних паперів у банку
void update_securities(Bank* bank) {
    for (int i = 0; i < bank->security_count; i++) {
        // "Випадкова зміна ціни в межах від -10% до +10%"
        double change = (rand() % 2001 - 1000) / 10000.0;
        bank->securities[i].price += bank->securities[i].price * change;
    }
}
```

5. **Купівля цінного паперу (buy_security)**
   Банк може придбати певну кількість цінних паперів за умови наявності достатньої готівки. Якщо угода успішна, кількість паперів у банку зменшується, а баланс банку знижується.

```
// покупка цінного паперу банком
void buy_security(Bank* bank, char* name, int quantity) {
    for (int i = 0; i < bank->security_count; i++) {
        if (strcmp(bank->securities[i].name, name) == 0 && bank->securities[i].quantity >= quantity) {
            double cost = bank->securities[i].price * quantity;      // "Загальна вартість покупки"
            if (bank->cash >= cost) {                                // "Перевірка, чи вистачає готівки"
                bank->securities[i].quantity -= quantity;            // "Зменшення кількості доступних паперів"
                bank->cash -= cost;                                  // "Зменшення готівки банку"
                printf("\nБанк купив %d одиниць %s.\n", quantity, name);
                return;
            }
        }
    }
    printf("\nЦінний папір %s недоступний або недостатня кількість.\n", name);
```

6. **Продаж цінного паперу (sell_security)**
   Функція дозволяє банку продати певну кількість паперів. У результаті кількість

```
// продаж цінного паперу банком
void sell_security(Bank* bank, char* name, int quantity) {
    for (int i = 0; i < bank->security_count; i++) {
        if (strcmp(bank->securities[i].name, name) == 0) {
            bank->securities[i].quantity += quantity; // "Збільшення кількості паперів у банку"
            bank->cash += bank->securities[i].price * quantity; // "Збільшення готівки банку"
            printf("\nБанк продав %d одиниць %s.\n", quantity, name);
            return;
        }
    }
    printf("\nЦінний папір %s не знайдено.\n", name);
```

   паперів збільшується, а баланс банку поповнюється.

7. **Оновлення депозитів клієнтів (update_client_deposits)**
   Ця функція розраховує щомісячний дохід від депозиту для кожного клієнта та додає його до балансу клієнта.

```
// оновлення депозитів клієнтів
void update_client_deposits(Bank* bank) {
    for (int i = 0; i < bank->client_count; i++) {
        // "Розрахунок місячної процентної ставки"
        double monthly_rate = bank->clients[i].deposit_rate / 12.0;
        // "Додавання відсотків до балансу клієнта"
        bank->clients[i].balance += bank->clients[i].balance * monthly_rate
    }
```

8. **Збереження даних у CSV-файл (save_to_csv)**
   Функція записує інформацію про клієнтів банку у текстовий файл формату CSV. Це дає змогу зберігати дані для подальшого використання.

```
void save_to_csv(Bank* bank, const char* filename) {
    FILE* file = fopen(filename, "w");
    if (file == NULL) {
        printf("\nНе вдалося відкрити файл %s для запису.\n", filename);
        return;
    }

    // "Запис заголовків таблиці"
    fprintf(file, "Name | Balance | DepositRate | DepositDuration |\n");
    for (int i = 0; i < bank->client_count; i++) {
        // "Запис даних клієнтів"
        fprintf(file, "%s | %.2f |%.2f | %d |\n", bank->clients[i].name, bank->clients[i].balance
                bank->clients[i].deposit_rate, bank->clients[i].deposit_duration);
    }
    fclose(file);
```

9. **Завантаження даних із CSV-файлу (load_from_csv)**
   Функція дозволяє завантажити дані клієнтів із файлу CSV, динамічно додаючи їх до існуючої бази.

```
// Завантаження даних банку з CSV-файлу
void load_from_csv(Bank* bank, const char* filename) {
    FILE* file = fopen(filename, "r");
    if (file == NULL) {
        printf("\nНе вдалося відкрити файл %s для читання.\n", filename);
        return;
    }

    char buffer[100];
    while (fgets(buffer, sizeof(buffer), file)) {
        if (strlen(buffer) < 5) continue; // "Пропустити порожні рядки"

        char name[50];
        double balance, rate;
        int duration;

        // "Зчитування даних про клієнта"
        if (sscanf(buffer, "%49[^,],%lf,%lf,%d", name, &balance, &rate, &duration) == 4) {
            add_client(bank, name, balance, rate, duration);
        }
    }

    fclose(file);
}
```

10. **Виведення стану банку (print_bank_state)**
    Відображає поточний стан банку, включаючи баланс, список клієнтів і доступні цінні папери.

```c
// виведення стану банку в консоль
void print_bank_state(Bank* bank) {
    printf("\n========== Стан банку ==========\n");
    printf("Готівка банку: %.2f\n", bank->cash);

    printf("\n========== Цінні папери ==========\n");
    if (bank->security_count == 0) {
        printf("Інформація про цінні папери відсутня.\n");
    } else {
        for (int i = 0; i < bank->security_count; i++) {
            printf("Тип: %s | Назва: %s | Ціна: %.2f | Кількість: %d\n",
                   bank->securities[i].type, bank->securities[i].name,
                   bank->securities[i].price, bank->securities[i].quantity);
        }
    }

    printf("\n========== Клієнти ==========\n");
    if (bank->client_count == 0) {
        printf("Інформація про клієнтів відсутня.\n");
    } else {
        for (int i = 0; i < bank->client_count; i++) {
            printf("Клієнт: %s | Баланс: %.2f | Процентна ставка: %.2f | Тривалість депозиту: %d місяців\n",
                   bank->clients[i].name, bank->clients[i].balance,
                   bank->clients[i].deposit_rate, bank->clients[i].deposit_duration);
        }
    }
}
```

11. **Очищення пам’яті (free_bank)**
    Функція звільняє динамічно виділену пам’ять для масивів клієнтів і цінних паперів,

```c
// Звільнення пам'яті, виділеної для банку
void free_bank(Bank* bank) {
    if (bank->clients) free(bank->clients);       // "Звільнення пам'яті для клієнтів"
    if (bank->securities) free(bank->securities); // "Звільнення пам'яті для цінних паперів"
}
```

    запобігаючи витоку пам’яті.

Інтерфейс користувача

У головному файлі main.c реалізовано меню, яке дозволяє користувачу виконувати наступні операції:

- Додавати клієнтів та цінні папери.
- Оновлювати депозити та ціни.
- Здійснювати купівлю й продаж цінних паперів.
- Зберігати та завантажувати дані.

Меню побудоване у вигляді циклу do-while, що забезпечує повторне виконання операцій до вибору пункту "Вихід".

Висновок

Розглянута програма демонструє базову реалізацію банківської системи, яка включає роботу з клієнтами, депозитами та цінними паперами. Її основною перевагою є використання модульного підходу, динамічного управління пам’яттю та підтримка збереження й завантаження даних із зовнішніх файлів. У майбутньому її можна розширити, додавши складніші фінансові операції або інтеграцію з мережею банківських установ.