

Національний університет «Одеська політехніка»
Інститут комп'ютерних систем
Кафедра інформаційних систем

КУРСОВА РОБОТА

з дисципліни «Алгоритмізація та програмування»

Тема «Програмування динамічної структури даних – стек»

Студента __1__ курсу AI-222 групи
Спеціальності 122 – «Комп'ютерні науки»
_____ Лясковського А.А.

(прізвище та ініціали)

Керівник доцент, к.т.н. Бабілунга О.Ю.

_____ (посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала _____

Кількість балів: _____

Оцінка: ECTS _____

Члени комісії

_____ (підпис)	_____ (прізвище та ініціали)
_____ (підпис)	_____ (прізвище та ініціали)
_____ (підпис)	_____ (прізвище та ініціали)

Національний університет «Одеська політехніка»
Інститут комп'ютерних систем
Кафедра інформаційних систем

ЗАВДАННЯ
НА КУРСОВУ РОБОТУ

студенту Лясковському Артему Андрійовичу група АІ-222

1. Тема роботи
«Програмування динамічної структури даних – стек»

2. Термін здачі студентом закінченої роботи 16.06.2023

3. Початкові дані до проекту (роботи)
Варіант 9

Структура стек: назва типу вкладу, клієнт, дата початку, дата закінчення, термін, процентна ставка.

Програма повинна виконувати: додавання елемента; видалення елемента; можливість коригування даних; виведення всіх даних; сортування по полю «процентна ставка»; додавання елемента в відсортований список по полю «термін»; виведення інформації про внесок по клієнту; підрахунок кількості елементів; виведення списку всіх елементів по заданому типу вклад.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які належить розробити)

Вступ. Теоретичні відомості про динамічну структуру даних стеку . Програмна реалізація алгоритмів роботи зі стеко для відділу банківських депозитів. Інструкція користувача. Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Блок-схема алгоритму – 1 аркуш формату А1.

Завдання видано

20.03.23

(підпис викладача)

Завдання прийнято до виконання 20.03.23

(підпис студента)

АНОТАЦІЯ

Розглянуто структуру Deposit, що представляє депозит, включаючи його тип, клієнта, дату початку і закінчення, термін і відсоткову ставку.

Запропоновано клас DepositNode, який є вузлом для зберігання депозитів у стеку. Кожен вузол містить посилання на об'єкт депозиту та посилання на наступний вузол.

Розроблено клас DepositStack, який представляє стек депозитів і має функціональність додавання, видалення, оновлення і друк усіх депозитів.

Також реалізовано функції сортування депозитів за відсотковою ставкою та збереження/завантаження депозитів з файлу.

У програмі здійснюється взаємодія з користувачем, який може виконувати різні операції з депозитами, такі як додавання, видалення, оновлення та виведення на екран.

ABSTRACT

The structure of Deposit, which represents a deposit, including its type, client, start and end date, term and interest rate, is considered.

The DepositNode class is proposed, which is a node for storing deposits in the stack. Each node contains a link to the deposit object and a link to the next node.

A DepositStack class has been developed, which represents a stack of deposits and has the functionality of adding, deleting, updating and printing all deposits.

The functions of sorting deposits by interest rate and saving/loading deposits from a file have also been implemented.

The program interacts with the user, who can perform various deposit operations such as adding, deleting, updating and displaying.

ЗМІСТ

Вступ.....	6
1 Теоретичні відомості про стек.....	8
1.1 Стек даних	8
2 Програмна реалізація стека даних.....	10
2.1 Програмна реалізація депозиту даних.....	10
3 Інструкція користувача	17
3.1 Інструкція користувача	17
Висновки.....	23
Перелік використаних джерел.....	24
Додаток А Код програми	25

ВСТУП

Метою курсової роботи є закріплення і поглиблення знань, одержаних студентами в курсі «Алгоритмізація та програмування», розвиток навичок при виборі представлення початкових даних, вдосконалення техніки використання засобів тестування і налагоджування програми, грамотне оформлення документації на програмну розробку.

Динамічні структури даних широко застосовуються в програмуванні для зберігання та обробки даних, які можуть змінюватися в ході виконання програми. Вони надають можливість ефективно маніпулювати даними, додавати, видаляти або змінювати їх, а також забезпечують гнучкість і оптимальність використання пам'яті.

Різновиди динамічних структур даних включають такі елементи, як списки, стеки, черги, дерева, графи та інші. Кожен з цих типів має свої унікальні особливості і використовується відповідно до конкретних потреб програми.

Особливості програмування динамічних структур даних залежать від підходу до програмування, який використовується. Один з найпоширеніших підходів - процедурний підхід, в якому програма організована навколо набору процедур або функцій, які виконують конкретні завдання. При використанні процедурного підходу для програмування динамічних структур даних, основні особливості включають наступне:

1. Створення структури даних: Початкова ініціалізація динамічних структур даних вимагає виділення пам'яті та ініціалізації відповідних змінних або показчиків.
2. Додавання елементів: При додаванні нових елементів до динамічної структури даних, необхідно виконати операції виділення пам'яті, перенаправлення показчиків і оновлення відповідних значень.
3. Видалення елементів: При видаленні елементів з динамічної структури даних, необхідно звільнити використану пам'ять і оновити зв'язки між елементами.

4. Оновлення елементів: Якщо необхідно змінити значення або властивості елементів динамічної структури даних, потрібно знайти відповідний елемент і оновити його значення.
5. Робота з ітераторами або покажчиками: Для перебору або доступу до елементів динамічної структури даних можуть використовуватися ітератори або покажчики.

Щодо основних етапів створення програмного продукту з використанням процедурного підходу, вони включають наступне:

1. Аналіз вимог: Розуміння потреб і вимог користувача, формулювання функціональних і нефункціональних вимог до програмного продукту.
2. Проектування: Визначення структури програми, обрання необхідних динамічних структур даних, проектування алгоритмів та вибір необхідних покажчиків або змінних.
3. Реалізація: Написання коду програми з використанням процедурного підходу, включаючи створення і обробку динамічних структур даних.
4. Тестування: Перевірка програмного продукту на відповідність вимогам, виявлення та виправлення помилок та недоліків.
5. Впровадження: Розгортання програмного продукту і його використання в реальних умовах.
6. Підтримка та післяреалізаційне супроводження: Підтримка програмного продукту, виправлення помилок, вдосконалення та розширення функціональності.

У цьому підході програма розділяється на невеликі фрагменти (процедури або функції), що спрощує розробку, тестування та підтримку програмного продукту. Процедурний підхід є одним з класичних підходів до програмування та застосовується в багатьох мовах програмування.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО СТЕК

1.1 Стек даних

Стек даних - це динамічна структура даних, яка дозволяє створювати та зберігати колекцію об'єктів впорядкованою послідовністю. Основна ідея стеку полягає в тому, що дані можуть бути додані або вилучені тільки з одного кінця стеку, який називається вершиною (top).

Стек працює за принципом (рисунок 1.1) "першим прийшов - останній вийшов" (LIFO - Last-In-First-Out). Це означає, що останній елемент, який був доданий до стеку, буде першим, який буде вилучений зі стеку. Введення нового елемента до стеку називається операцією "push", а вилучення - операцією "pop".

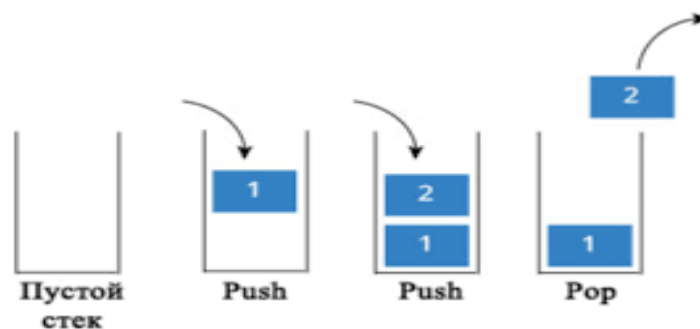


Рисунок 1.1 – Схема додавання елемента в список

Стеки широко використовуються в програмуванні для реалізації різних завдань. Наприклад, вони можуть бути використані для зберігання та керування контекстами виклику функцій (включення та виключення), виконання рекурсивних алгоритмів, роботи зі стековими кадрами в мові асемблера та багатьох інших випадках, де важливо зберігати та керувати послідовністю об'єктів.

Стеки зазвичай використовуються для реалізації алгоритмів, де важливим є збереження порядку обробки даних. Наприклад, при обході дерева в глибину або виконанні рекурсивних викликів функцій стек може використовуватись для

збереження поточного стану обробки. Також стеки використовуються для реалізації системного стеку викликів у багатьох програмах та операційних системах.

2 ПРОГРАМНА РЕАЛІЗАЦІЯ СТЕКА ДАННИХ

2.1 Програмна реалізація депозиту даних

Даний код є програмою для керування депозитами. Вона надає можливість додавати нові депозити, видаляти депозити, оновлювати інформацію про депозити, виводити список всіх депозитів, зберігати депозити в файл, сортувати депозити за відсотковою ставкою, знаходити депозити за ім'ям клієнта та кількістю депозитів за типом депозиту.

Структура `Deposit` використовується для представлення депозиту і містить такі поля:

- `type` (тип депозиту)
- `client` (ім'я клієнта)
- `start_date` (дата початку депозиту)
- `end_date` (дата закінчення депозиту)
- `term` (термін депозиту)
- `interest_rate` (відсоткова ставка)
- Функція `getPositiveNumberInput()` отримує ціле число від користувача, перевіряє, чи введене число є додатнім, та повертає його.

Клас `DepositNode` представляє вузол депозиту в стеку депозитів. Клас має наступні поля:

- `deposit` (вказівник на об'єкт `Deposit`)
- `next` (вказівник на наступний вузол)

Клас `DepositStack` представляє стек депозитів. Клас має наступні методи:

- Конструктор `DepositStack()` - ініціалізує стек без вузлів:
 1. Створюється новий об'єкт типу `DepositStack`.
 2. Ініціалізується порожній список `deposits`, який буде використовуватись для зберігання депозитів у стеку.

Отже, конструктор `DepositStack()` створює новий об'єкт стеку депозитів та ініціалізує порожній список для зберігання депозитів.

– Метод `isEmpty()` - перевіряє, чи стек пустий:

1. Перевіряється кількість депозитів у стеці.
2. Якщо кількість депозитів дорівнює нулю, повертається значення `true`, що вказує на те, що стек депозитів порожній.
3. Якщо кількість депозитів більше нуля, повертається значення `false`, що вказує на те, що стек депозитів не є порожнім.

Отже, метод `isEmpty()` дозволяє перевірити, чи містить стек депозитів хоча б один депозит.

– Метод `push()` - додає новий депозит до стеку:

1. Приймає параметр, який представляє новий депозит, який потрібно додати до стеку.
2. Створює новий об'єкт депозиту з наданими параметрами.
3. Перевіряє, чи є вільні місця у стеці депозитів. Якщо стек заповнений, не можна додати новий депозит і видається повідомлення про переповнення стеку.
4. Якщо є вільне місце у стеці, новий депозит додається до верхнього кінця стеку.
5. Збільшує лічильник депозитів у стеці на одиницю.
6. Повертається повідомлення про успішне додавання депозиту до стеку.

Отже, метод `push()` дозволяє додати новий депозит до стеку, якщо це можливо, і вказує про результат операції.

– Метод `pop()` - видаляє та повертає останній доданий депозит зі стеку:

1. Перевіряє, чи є депозити в стеці. Якщо стек порожній, видається повідомлення про те, що стек депозитів порожній і немає депозитів для видалення.

2. Якщо в стеці є депозити, останній депозит видаляється з верхнього кінця стеку.
3. Зменшує лічильник депозитів у стеці на одиницю.
4. Повертає видалений депозит як результат операції.

Отже, метод `pop()` дозволяє видалити та отримати останній депозит зі стеку, якщо це можливо, і повідомляє про результат операції.

- Метод `printAllDeposits()` - виводить на екран інформацію про всі депозити у стеку:

1. Перевіряє, чи є депозити в стеці. Якщо стек порожній, виводиться повідомлення про те, що стек депозитів порожній і немає депозитів для виведення.
2. Якщо в стеці є депозити, метод починає ітерацію через всі елементи стеку, починаючи з верхнього елемента.
3. Для кожного депозиту у стеці виконується наступне:
4. Виводиться інформація про депозит, така як його номер, назва або опис, розмір і термін депозиту.
5. Переходиться до наступного депозиту в стеці.
6. Після того як усі депозити були виведені, метод завершує свою роботу.

Отже, метод `printAllDeposits()` дозволяє вивести на екран інформацію про всі депозити, що зберігаються в стеці, якщо вони є, або повідомляє про те, що стек депозитів порожній.

- Метод `updateDeposit()` - оновлює інформацію про певний депозит у стеку:

1. Приймає параметром номер депозиту, який потрібно оновити.
2. Перевіряє, чи є депозити в стеці. Якщо стек порожній, виводиться повідомлення про те, що стек депозитів порожній і немає депозитів для оновлення.

3. Якщо в стеці є депозити, метод починає ітерацію через всі елементи стеку, починаючи з верхнього елемента.
4. Для кожного депозиту у стеці виконується наступне:
5. Порівнюється номер депозиту з номером, який потрібно оновити. Якщо номери збігаються, виконується оновлення інформації про депозит.
6. Оновлюється будь-який параметр депозиту, який необхідно змінити (наприклад, розмір, термін, назва).
7. Метод завершує свою роботу після успішного оновлення депозиту.
8. Якщо номер депозиту не знайдений після ітерації через всі депозити в стеці, виводиться повідомлення про те, що депозит з вказаним номером не знайдений у стеці.

Отже, метод `updateDeposit()` дозволяє оновити інформацію про конкретний депозит у стеці, виконуючи заміну або зміну його параметрів.

– Метод `removeDeposit()` - видаляє певний депозит зі стеку:

1. Приймає параметром номер депозиту, який потрібно видалити.
2. Перевіряє, чи є депозити в стеці. Якщо стек порожній, виводиться повідомлення про те, що стек депозитів порожній і немає депозитів для видалення.
3. Якщо в стеці є депозити, метод починає ітерацію через всі елементи стеку, починаючи з верхнього елемента.
4. Для кожного депозиту у стеці виконується наступне:
5. Порівнюється номер депозиту з номером, який потрібно видалити. Якщо номери збігаються, виконується видалення депозиту зі стеку.
6. Після видалення депозиту метод завершує свою роботу.
7. Якщо номер депозиту не знайдений після ітерації через всі депозити в стеці, виводиться повідомлення про те, що депозит з вказаним номером не знайдений у стеці.

Отже, метод `removeDeposit()` дозволяє видалити конкретний депозит зі стеку, зменшуючи розмір стеку та оновлюючи індекси інших депозитів у стеці, якщо необхідно.

– Метод `printDepositByClient()` - виводить інформацію про депозити за ім'ям клієнта:

1. Приймає параметром ім'я клієнта, для якого потрібно вивести депозити.
2. Перевіряє, чи є депозити в стеці. Якщо стек порожній, виводиться повідомлення про те, що стек депозитів порожній і немає депозитів для виведення.
3. Якщо в стеці є депозити, метод починає ітерацію через всі елементи стеку, починаючи з верхнього елементу.
4. Для кожного депозиту у стеці виконується наступне:
5. Порівнюється ім'я клієнта, пов'язаного з депозитом, з вказаним ім'ям клієнта. Якщо імена збігаються, виводиться інформація про депозит на екран.
6. Продовжується ітерація через інші депозити в стеці.
7. Якщо жоден депозит не знайдено з вказаним ім'ям клієнта після ітерації через всі депозити в стеці, виводиться повідомлення про те, що депозити для вказаного клієнта не знайдені у стеці.

Отже, метод `printDepositByClient()` дозволяє знайти та вивести на екран усі депозити, пов'язані з певним клієнтом, якщо такі депозити є у стеці.

– Метод `printDepositByType()` - виводить інформацію про депозити за типом депозиту:

1. Приймає параметром ім'я клієнта, для якого потрібно вивести депозити.

2. Перевіряє, чи є депозити в стеці. Якщо стек порожній, виводиться повідомлення про те, що стек депозитів порожній і немає депозитів для виведення.
3. Якщо в стеці є депозити, метод починає ітерацію через всі елементи стеку, починаючи з верхнього елемента.
4. Для кожного депозиту у стеці виконується наступне:
5. Порівнюється ім'я клієнта, пов'язаного з депозитом, з вказаним ім'ям клієнта. Якщо імена збігаються, виводиться інформація про депозит на екран.
6. Продовжується ітерація через інші депозити в стеці.
7. Якщо жоден депозит не знайдено з вказаним ім'ям клієнта після ітерації через всі депозити в стеці, виводиться повідомлення про те, що депозити для вказаного клієнта не знайдені у стеці.

Отже, метод `printDepositByClient()` дозволяє знайти та вивести на екран усі депозити, пов'язані з певним клієнтом, якщо такі депозити є у стеці.

– Метод `sortDepositsByInterestRate()` - сортує депозити за відсотковою ставкою:

1. Перевіряє, чи є депозити в стеці. Якщо стек порожній або містить лише один елемент, немає потреби в сортуванні, тому метод повертається без змін до викликача.
2. Якщо в стеці є більше одного депозиту, метод починає сортування.
3. Використовуючи алгоритм сортування (наприклад, "сортування бульбашкою" або "сортування вставкою"), метод порівнює відсоткові ставки депозитів і переставляє їх у стеці в порядку зростання ставок.
4. Процес сортування продовжується, поки всі депозити не будуть впорядковані за зростанням відсоткових ставок.
5. Після завершення сортування метод повертає відсортований стек депозитів.

Отже, метод `sortDepositsByInterestRate()` забезпечує сортування депозитів у стеці за зростанням відсоткових ставок, що дозволяє легко отримати доступ до депозитів з найбільш вигідними ставками.

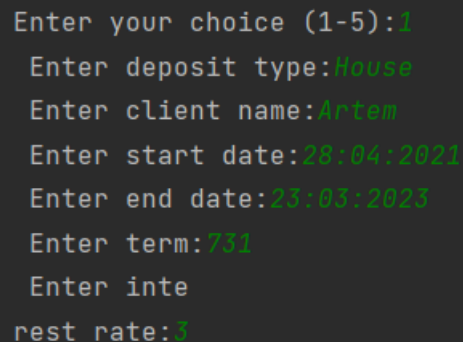
Основна програма створює об'єкт класу `DepositStack` та надає користувачу меню для взаємодії з депозитами. Користувач може вибрати опцію для додавання, видалення, оновлення, виводу або сортування депозитів.

3 ІНСТРУКЦІЯ КОРИСТУВАЧА

3.1 Інструкція користувача

Програма є системою управління депозитами. Нижче наведено опис меню програми та інструкції щодо взаємодії з нею:

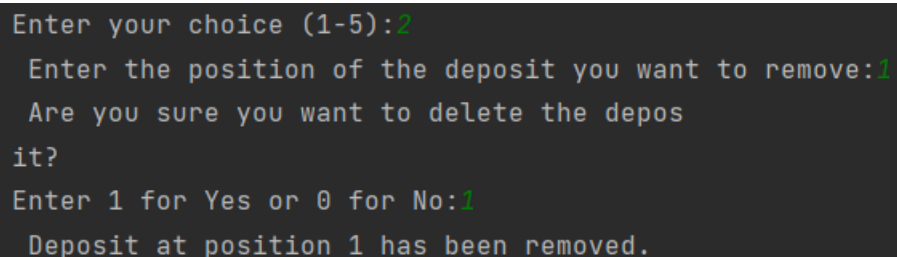
- Додати новий депозит(малюнок 2.2): Користувач може ввести дані про новий депозит, такі як тип депозиту, ім'я клієнта, дату початку та закінчення, термін депозиту та процентну ставку.



```
Enter your choice (1-5):1
Enter deposit type:House
Enter client name:Artem
Enter start date:28:04:2021
Enter end date:23:03:2023
Enter term:731
Enter inte
rest rate:3
```

Малюнок 2.2 – Додавання нового депозиту

- Видалити депозит(малюнок 2.3): Користувач може видалити депозит зі стеку. Він буде попрошений ввести позицію депозиту, який потрібно видалити.



```
Enter your choice (1-5):2
Enter the position of the deposit you want to remove:1
Are you sure you want to delete the depos
it?
Enter 1 for Yes or 0 for No:1
Deposit at position 1 has been removed.
```

Малюнок 2.3– Видалення депозиту

- Оновити існуючий депозит(малюнок 2.4): Користувач може оновити дані будь-якого існуючого депозиту. Він буде попросений ввести позицію депозиту, який потрібно оновити, а потім ввести нові дані про депозит.

```

Enter your choice (1-5):3
Enter the position of the deposit you want to update:1
Enter new deposit type:Dollar
Enter new client
name:Vova
Enter new start date:20:01:2022
Enter new end date:21:01:2023
Enter new term:366
Enter new interest rate:2

```

Малюнок 2.4– Оновлення депозиту

- Вивести всі депозити(малюнок 2.5): Програма виводить всі депозити, що зберігаються в стеці.

```

Enter your choice (1-5):4

1-deposit
Deposit type: ipoteka
Client name: Seny
Start date: 23:04:2001
End date: 24:07:2005
Term: 1000
Interest rate: 1

2-deposit
Deposit type: Deposit
Client name: Artem
Start date: 2
End date: 21
Term: 19
Interest rate: 2

3-deposit
Deposit type: Kredit
Client name: Vova
Start date: 32
End date: 12
Term: 21
Interest rate: 2

4-deposit
Deposit type: df
Client name: rdf
Start date: 43

```

Малюнок 2.5– Виведення всіх депозитів

- Зберегти депозити у файл(малюнок 2.6): Програма зберігає всі депозити зі стеку у текстовий файл з назвою "deposits.txt".

```
Enter your choice (1-5):5
Data is successfully save to file.
```

Малюнок 2.6– Збереження депозитів у фай

- Відсортувати депозити(малюнок 2.7): Програма сортує депозити за процентною ставкою в порядку спадання.

```
0. Quit
Enter your choice (1-5):6
Deposits sorted by interest rate.

1-deposit
Deposit type: avto_kredit
Client name: Vova
Start date: 28:08:2003
End date: 29:09:2004
Term: 324
Interest rate: 5

2-deposit
Deposit type: df
Client name: rdf
Start date: 43
End date: 54
Term: 12
Interest rate: 3

3-deposit
Deposit type: Kredit
Client name: Vova
Start date: 32
End date: 12
Term: 21
Interest rate: 2
```

Малюнок 2.7– Сортування депозитів

- Знайти депозит за ім'ям клієнта(малюнок 2.8): Користувач може ввести ім'я клієнта, і програма виведе всі його депозити.

```
Enter your choice (1-5):7
Enter client name:Artem

1-deposit
Deposit type: Deposit
Client name: Artem
Start date: 2
End date: 21
Term: 19
Interest rate: 2
```

Малюнок 2.8– Пошук депозиту за клієнтом

- Кількість депозитів(малюнок 2.9): Програма виводить загальну кількість депозитів, які зберігаються в стеці.

```
Enter your choice (1-5):8
Number of deposits: 5
```

Малюнок 2.9– Кількість депозитів

- Знайти депозити за типом депозиту(малюнок 2.10): Користувач може ввести тип депозиту, і програма виведе всі депозити з цим типом.

```
Enter your choice (1-5):9
Enter deposit type:Kredit

1-deposit
Deposit type: Kredit
Client name: Vova
Start date: 32
End date: 12
Term: 21
Interest rate: 2
```

Малюнок 2.10– Пошук депозиту за типом

- Сортувати депозити за терміном(малюнок 2.11): Користувач може ввести дані про новий депозит із всіма його атрибутами, і програма вставить цей депозит на відповідну позицію в стеці, враховуючи термін д.

```

Enter your choice (1-5):10
Enter deposit type:Гривня
Enter client name:Паша
Enter start date:29-01-2020
Enter end date:30-01-2022
Enter term:730
Enter inte
rest rate:3
Deposit added successfully.

1-deposit
Deposit type: avto_kredit
Client name: Vova
Start date: 28:08:2003
End date: 29:09:2004
Term: 324
Interest rate: 5

2-deposit
Deposit type: df
Client name: rdf
Start date: 43
End date: 54
Term: 12
Interest rate: 3

3-deposit
Deposit type: Kredit
Client name: Vova
Start date: 32

```

Малюнок 2.11– Сорткування депозиту за терміном

ВИСНОВКИ

Була реалізована база даних для депозитного відділу банку, яке включає в себе опрацювання даних, зберігання цих даних у файлі. Також була реалізована найпростіше сортування даних вкладних депозитів. Курсова робота виконана по всім вимогам до неї на 100%.

При написанні курсової роботи дізнався і закріпив свої навички роботи зі стеком пам'яті. Також поглибив свої знання с функціями і показниками.

На мою думку увагу треба приділити функцій (`getPositiveNumberInput`) вона допомагає у випадку, якщо користувач вів на місце числа букву чи інший символ програма не ламалася і не закривалася, а просила користувача знову вести число. Також треба приділи увагу моїй реалізації стеку даних без використання ООП.

В ході роботи виникли труднощі у реалізації стеку даних без ООП, це було для мене складно і вона забрала найбільше часу і зусиль. А в цілому по роботі складнощів не виникло окрім цього випадку.

Якщо оцінювати у відсотках мій вклад у роботу, то він складає десь 70%. Тому що за час виконання курсової роботи було прочитано багато сайтів де я брав чи копіював деякі рішення, чи брав їх за основу і робив вже свої рішення.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до курсової роботи з дисципліни «Алгоритмізація та програмування» для студентів спеціальності 122 Комп'ютерні науки першого (бакалаврського) рівня вищої освіти /Укл.: О.Ю. Бабілунга, О.В. Іванов, О.С. Манікаєва. – Одеса: «Одеська політехніка», 2022. – 26 с.

2. Stack in C++: веб-сайт. URL:

<https://www.geeksforgeeks.org/stack-in-cpp-stl/>

(дата звернення: 25.05.2023).

3. C++ saving records to file: веб-сайт. URL:

<https://stackoverflow.com/questions/20652939/c-saving-records-to-file>

(дата звернення: 25.05.2023).

4. C++ Pointers: веб-сайт.URL:

<https://www.geeksforgeeks.org/cpp-pointers/>

(дата звернення: 25.05.2023).

5. File Handling through C++ Classes: веб-сайт. URL:

<https://www.geeksforgeeks.org/file-handling-c-classes/>

(дата звернення: 21.06.2023).

6. What is the use of cin.ignore() in C++?: веб-сайт. URL:

<https://www.tutorialspoint.com/what-is-the-use-of-cin-ignore-in-cplusplus>

(дата звернення: 21.06.2023).

ДОДАТОК А

Код програми

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <limits>

using namespace std;

struct Deposit {
    string type;
    string client;
    string start_date;
    string end_date;
    int term{};
    float interest_rate{};
};

int getPositiveNumberInput() {
    int number;
    while (true) {
        if (cin >> number && number > 0) {
            break;
        } else {
            cout << "Invalid input. Must be a positive number: ";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
    }
    return number;
}

class DepositNode {
public:
    DepositNode(Deposit* deposit, DepositNode* next = nullptr) {
        this->deposit = deposit;
        this->next = next;
    }
    Deposit* deposit;
    DepositNode* next;
};

class DepositStack {
public:
    DepositStack() {
        top = nullptr;
    }

    ~DepositStack() {
        DepositNode *temp;
        while (top != nullptr) {
```



```

        temp = top;
        top = top->next;
        delete temp->deposit;
        delete temp;
    }
}

bool isEmpty() const {
    return top == nullptr;
}

void push(Deposit *deposit) {
    DepositNode *newNode = new DepositNode(deposit, top);
    top = newNode;
}

Deposit *pop() {
    if (isEmpty()) {
        return nullptr;
    }
    Deposit *deposit = top->deposit;
    DepositNode *temp = top;
    top = top->next;
    delete temp;
    return deposit;
}

void printAllDeposits() const {
    int q = 1;
    if (isEmpty()) {
        cout << "Stack is empty." << endl;
    } else {
        DepositNode *current = top;
        while (current != nullptr) {
            Deposit *deposit = current->deposit;
            cout << endl << q << "-deposit " << endl;
            cout << "Deposit type: " << deposit->type << endl;
            cout << "Client name: " << deposit->client << endl;
            cout << "Start date: " << deposit->start_date << endl;
            cout << "End date: " << deposit->end_date << endl;
            cout << "Term: " << deposit->term << endl;
            cout << "Interest rate: " << deposit->interest_rate << endl;
            current = current->next;
            q++;
        }
    }
}

bool updateDeposit() const {
    int position;
    cout << "Enter the position of the deposit you want to update: ";
    cin >> position;
}

```

```

DepositNode *current = top;
int currentPosition = 1;
while (current != nullptr && currentPosition < position) {
    current = current->next;
    currentPosition++;
}
if (current == nullptr) {
    cout << "Invalid position." << endl;
    return false;
}
Deposit *deposit = current->deposit;
cout << "Enter new deposit type: ";
cin >> deposit->type;
cout << "Enter new client name: ";
cin >> deposit->client;
cout << "Enter new start date: ";
cin >> deposit->start_date;
cout << "Enter new end date: ";
cin >> deposit->end_date;

cout << "Enter new term: ";
deposit->term = getPositiveNumberInput();

cout << "Enter new interest rate: ";
deposit->interest_rate = getPositiveNumberInput();
return true;
}

bool removeDeposit() {
    if (isEmpty()) {
        cout << "Stack is empty. Cannot remove deposit." << endl;
        return false;
    }
    int position;
    cout << "Enter the position of the deposit you want to remove: ";
    cin >> position;

    if (position < 1 || position > countDeposits()) {
        cout << "Invalid position." << endl;
        return false;
    }
    int confirmation;
    cout << "Are you sure you want to delete the deposit?" << endl;
    cout << "Enter 1 for Yes or 0 for No: ";
    cin >> confirmation;

    if (confirmation != 1) {
        cout << "Deposit removal canceled." << endl;
        return false;
    }

    DepositNode *previous = nullptr;

```

```

DepositNode *current = top;
int currentPosition = 1;

while (current != nullptr && currentPosition < position) {
    previous = current;
    current = current->next;
    currentPosition++;
}

if (current == nullptr) {
    cout << "Invalid position." << endl;
    return false;
}

if (previous == nullptr) {
    top = current->next;
} else {
    previous->next = current->next;
}

delete current->deposit;
delete current;

cout << "Deposit at position " << position << " has been removed." << endl;
return true;
}

DepositNode *top;

void printDepositByClient(const string &clientName) const {
    DepositNode *current = top;
    int a = 1;
    bool found = false;
    while (current != nullptr) {
        Deposit *deposit = current->deposit;
        if (deposit->client == clientName) {
            cout << endl << a << "-deposit " << endl;
            cout << "Deposit type: " << deposit->type << endl;
            cout << "Client name: " << deposit->client << endl;
            cout << "Start date: " << deposit->start_date << endl;
            cout << "End date: " << deposit->end_date << endl;
            cout << "Term: " << deposit->term << endl;
            cout << "Interest rate: " << deposit->interest_rate << endl;
            found = true;
            a++;
        }
        current = current->next;
    }
    if (!found) {
        cout << "No deposit found for client: " << clientName << endl;
    }
}

```

```

void printDepositsByType(const string &depositType) const {
    int z = 1;
    DepositNode *current = top;
    bool found = false;
    while (current != nullptr) {
        Deposit *deposit = current->deposit;
        if (deposit->type == depositType) {
            cout << endl << z << "-deposit " << endl;
            cout << "Deposit type: " << deposit->type << endl;
            cout << "Client name: " << deposit->client << endl;
            cout << "Start date: " << deposit->start_date << endl;
            cout << "End date: " << deposit->end_date << endl;
            cout << "Term: " << deposit->term << endl;
            cout << "Interest rate: " << deposit->interest_rate << endl;
            found = true;
            z++;
        }
        current = current->next;
    }
    if (!found) {
        cout << "No deposits found for type: " << depositType << endl;
    }
}

int countDeposits() const {
    int count = 0;
    DepositNode *current = top;
    while (current != nullptr) {
        count++;
        current = current->next;
    }
    return count;
}
};

void sortinsertDepositByTerm(DepositStack &stack) {

    DepositNode *current = stack.top;
    int count = 0;
    while (current != nullptr) {
        count++;
        current = current->next;
    }
    DepositNode *current1;
    DepositNode *current2 = nullptr;
    bool flag;
    do {
        flag = false;
        current1 = stack.top;
        while (current1->next != current2) {
            if (current1->deposit->term < current1->next->deposit->term) {
                Deposit *temp = current1->deposit;

```

```

        current1->deposit = current1->next->deposit;
        current1->next->deposit = temp;
        flag = true;
    }
    current1 = current1->next;
}
current2 = current1;
} while (flag);
cout << "Deposit added successfully." << endl;
}

void sortDepositsByInterestRate(DepositStack &stack) {
    DepositNode *current = stack.top;
    int count = 0;
    while (current != nullptr) {
        count++;
        current = current->next;
    }
    DepositNode *current1;
    DepositNode *current2 = nullptr;
    bool flag;
    do {
        flag = false;
        current1 = stack.top;
        while (current1->next != current2) {
            if (current1->deposit->interest_rate < current1->next->deposit->interest_rate) {
                Deposit *temp = current1->deposit;
                current1->deposit = current1->next->deposit;
                current1->next->deposit = temp;
                flag = true;
            }
            current1 = current1->next;
        }
        current2 = current1;
    } while (flag);
}

void saveDepositsToFile(DepositStack& stack) {
    ofstream outputFile("deposits.txt");
    if (!outputFile) {
        cout << "Error: could not open file." << endl;
        outputFile.close();
        return;
    }
    DepositNode *current = stack.top;
    while (current != nullptr) {
        Deposit *deposit = current->deposit;
        outputFile << deposit->type << "," << deposit->client << "," << deposit->start_date << ","
        << deposit->end_date
        << "," << deposit->term << "," << deposit->interest_rate << endl;
        current = current->next;
    }
}

```

```

    cout << "Data is successfully save to file." << endl;
    outputFile.close();
}
void loadDepositsFromFile(DepositStack& stack) {
    ifstream inputFile("deposits.txt");
    if (!inputFile) {
        cout << "Error: could not open file." << endl;
        inputFile.close();
        return;
    }
    string line;
    while (getline(inputFile, line)) {
        Deposit* deposit = new Deposit();
        stringstream ss(line);
        getline(ss, deposit->type, ',');
        getline(ss, deposit->client, ',');
        getline(ss, deposit->start_date, ',');
        getline(ss, deposit->end_date, ',');
        string termString, interestRateString;
        getline(ss, termString, ',');
        getline(ss, interestRateString, ',');
        deposit->term = stoi(termString);
        deposit->interest_rate = stof(interestRateString);
        stack.push(deposit);
    }
    cout << "Data uploaded successfully." << endl;
    inputFile.close();
}
int main() {
    DepositStack stack;

    loadDepositsFromFile(stack);

    while (true) {

        cout << endl << "==== Deposit Management System =====" << endl;
        cout << "1. Add a new deposit" << endl;
        cout << "2. Remove deposit" << endl;
        cout << "3. Update an existing deposit" << endl;
        cout << "4. Print all deposits" << endl;
        cout << "5. Save deposits to file" << endl;
        cout << "6. Sort the deposits" << endl;
        cout << "7. Find deposit by client name" << endl;
        cout << "8. Number of deposit" << endl;
        cout << "9. Find deposit by type of deposit" << endl;
        cout << "10.Sort deposit by term" << endl;
        cout << "0. Quit" << endl;
        cout << "Enter your choice (1-5): ";

        int choice;
        cin >> choice;
    }
}

```

```

switch (choice) {
    case 1: {

        Deposit* deposit = new Deposit();
        cout << "Enter deposit type: ";
        cin >> deposit->type;
        cout << "Enter client name: ";
        cin >> deposit->client;
        cout << "Enter start date: ";
        cin >> deposit->start_date;
        cout << "Enter end date: ";
        cin >> deposit->end_date;

        cout << "Enter term: ";
        deposit->term = getPositiveNumberInput();

        cout << "Enter interest rate: ";
        deposit->interest_rate = getPositiveNumberInput();

        stack.push(deposit);
        break;
    }
    case 2: {
        bool success = stack.removeDeposit();
        if (!success) {
            cout << "Failed to remove deposit." << endl;
        }
        break;
    }
    case 3: {

        bool success = stack.updateDeposit();
        if (!success) {
            cout << "Failed to update deposit." << endl;
        }
        break;
    }
    case 4: {
        stack.printAllDeposits();
        break;
    }
    case 5: {

        saveDepositsToFile(stack);
        break;
    }
    case 6: {
        sortDepositsByInterestRate(stack);
        cout << "Deposits sorted by interest rate." << endl;
        stack.printAllDeposits();
        break;
    }
}

```

```

case 7: {
    string clientName;
    cout << "Enter client name: ";
    cin >> clientName;
    stack.printDepositByClient(clientName);
    break;
}
case 8: {
    int numDeposits = stack.countDeposits();
    cout << "Number of deposits: " << numDeposits << endl;
    break;
}
case 9: {
    string depositType;
    cout << "Enter deposit type: ";
    cin >> depositType;
    stack.printDepositsByType(depositType);
    break;
}
case 10: {
    Deposit* deposit = new Deposit();
    cout << "Enter deposit type: ";
    cin >> deposit->type;
    cout << "Enter client name: ";
    cin >> deposit->client;
    cout << "Enter start date: ";
    cin >> deposit->start_date;
    cout << "Enter end date: ";
    cin >> deposit->end_date;

    cout << "Enter term: ";
    deposit->term = getPositiveNumberInput();
    cout << "Enter interest rate: ";
    deposit->interest_rate = getPositiveNumberInput();

    stack.push(deposit);

    sortinsertDepositByTerm(stack);

    stack.printAllDeposits();
    cout << "Data successfully add." << endl;
    break;
}
case 0: {
    cout << "Exiting program." << endl;
    return 0;
}

default: {
    cout << "Invalid choice." << endl;
    break;
}

```


}
}
}