**Міністерство освіти і науки України**

**Національний університет “Львівська політехніка”**

Кафедра ЕОМ



**Курсова робота**

з предмету: «Програмування, частина 2 (Об’єктно-орієнтованепрограмування)»

на тему:

**«Базові принципи об’єктно-орієнтованогопрограмування»**

Індивідуальне завдання: «Служба таксі»

**Виконав:**

ст. гр. КІ-15

Петрущак Ярема

**Прийняв:**

Викладач

Козак Н. Б.

Львів – 2020

Анотація

У цій курсовій роботі було виконано написання програмного забезпечення на мові програмування С++ з використанням основних принципів об’єктно-орієнтованого програмування, таких як одинарне та множинне наслідування, абстрагування, інкапсуляція. Програма являє собою спрощену модель функціонування служби таксі та включає у себе роботу з базою даних, а саме: запис, зчитування, видалення та редагування вибраних даних. Програма має кілька рівнів доступу, які спрощують її використання, оскільки можливо одразу визначити, який саме користувач увійшов у систему та які операції він може виконувати. Інтерфейс програмного забезпечення написаний українською мовою.

Зміст

[Анотація 2](#_Toc10508192)

[Вступ 4](#_Toc10508193)

[1. Визначення поняття ООП, його основні принципи та обґрунтування його технологій 5](#_Toc10508194)

[1.1 ООП як основа сучасного програмування 5](#_Toc10508195)

[1.2 Визначення ООП 6](#_Toc10508196)

[1.3 Фундаментальні поняття ООП 7](#_Toc10508197)

[1.4 Переваги обраного методу 8](#_Toc10508198)

[2. Аналіз та розробка алгоритму згідно індивідуального завдання 9](#_Toc10508199)

[2.1 Опис алгоритму 9](#_Toc10508200)

[2.2 Аналіз особливостей алгоритму 10](#_Toc10508201)

[2.3 Опис алгоритму у псевдокоді 10](#_Toc10508202)

[2.4 Блок-схема алгоритму 11](#_Toc10508203)

[3. Програмна реалізація алгоритму згідно індивідуального завдання 13](#_Toc10508204)

[3.1 Опис послідовної версії програмної реалізації алгоритму 13](#_Toc10508205)

[3.1.1 Опис структур даних програми 13](#_Toc10508206)

[3.1.2 Діаграма класів 13](#_Toc10508207)

[3.1.3 Опис реалізації основних функцій/методів 15](#_Toc10508208)

[4. Відлагодження та тестування реалізованої програми згідно індивідуального завдання 17](#_Toc10508209)

[4.1 Опис структури програми 17](#_Toc10508210)

[4.2 Відлагодження та тестування окремих частин програми 18](#_Toc10508211)

[4.3 Відлагодження та тестування програми в цілому 18](#_Toc10508212)

[5. Аналіз продуктивності (часу роботи) та структурної складності реалізованої програм згідно індивідуального завдання 19](#_Toc10508213)

[5.1 Аналіз структурної складності виконання програми 19](#_Toc10508214)

[5.2 Аналіз часу виконання програми в залежності від об'єму вхідних даних 21](#_Toc10508215)

[Висновки 24](#_Toc10508216)

[Використані джерела: 25](#_Toc10508217)

[Додатки 26](#_Toc10508218)

Вступ

Мова програмування Сi була розроблена в 1972 р. Деннисом Рiтчi, який є одним з авторів операційної системи UNIX. Мова програмування Сi пізніше була використана для програмування цієї системи, а також багатої бібліотеки обслуговуючих програм, оскільки, являлася універсальною мовою загального призначення, мова Сi зручна для програмування системних задач. Поява мікрокомп’ютерів закріпила позиції мови Сi. Було створено близько 30 його нових компіляторів, а після проведення Американським національним інститутом стандартів (American National Standards Institute) робіт по стандартизації в області програмування почали розроблятися компілятори, що відповідали опублікованому весною 1986 р. проекту стандарту.

Першим компілятором по стандарту ANSI являлась система Турбо Сi версii 1.0 фiрми Borland International. Ця система, що складається з компілятора мови Сi, пов'язаного з ним редактора, компоновщика i бібліотек, забезпечила користувачам зручну інтегровану операційну оболонку, а також суттєво полегшувала професійне програмування, в якому визначальними параметрами є висока швидкість компіляції, висока якість згенерованого коду та невеликий об’єм оперативної пам’ять. Мова С серед сучасних мов програмування є однією із найбільш поширених. Вона універсальна, але найбільш ефективне її використання в задачах системного програмування - розробки трансляторів, операційних систем, інструментальних засобів. Мова С добре зарекомендувала себе ефективністю, лаконічністю запису алгоритмів, логічною стрункістю програм. У багатьох випадках програми, написані на мові С, можна порівняти по швидкості з програмами, написаними на мові Асемблера, при цьому вони більш наглядні i прості у супроводі.

Основними перевагами мови С вважаться висока передача написаних на ній програм між комп'ютерами з різною архітектурою, між різними операційними середовищами. Транслятори мови С діють практично на всіх персональних комп'ютерах, які використовуються в наш час. Перечислимо окремі особливості мови С:

-в мові С реалізовані окремі операції низького рівня (а саме операції над бітами). Окремі з таких операцій напряму відповідають машинним командам;

- мова С підтримує механізм показників на змінні i функції;

-базові типи даних мови С відображають тi ж об’єкти, з якими потрібно мати справу програмі на мові Асемблера,- байти, машинні слова, символи, строки;

- як в ніякій іншій мові програмування високого рівня в мові С накладаються лише незначні обмеження на перетворення типів даних;

- не дивлячись на ефективність і потужність конструкції мови С, він відносно малий по об'єму, але в системне оточення мови С входить бібліотека стандартних функцій, в який реалізовані встроєні оператори введення - виведення, динамічного розподілу пам’ять, управління процесами i т.д.

1. Визначення поняття ООП, його основні принципи та обґрунтування його технологій
   1. ООП як основа сучасного програмування

Раніше програми розроблялися із застосуванням процедурного підходу, що полягав у поділі проекту на окремі більш дрібні складові – функції. Із розростанням проектів та ускладненням завдань з’явилась необхідність у більш раціональному підході у написанні програмного забезпечення та ефективному поділі поставленого завдання таким чином, щоб не виникало проблем при розробці окремих складових проекту. Для подолання цієї проблеми було розроблено структурне програмування. Структурне програмування, на відміну від свого попередника — стихійного програмування, має певні правила та засоби, які суттєво вдосконалили процес написання програм. Однак через обмеженість структурного програмування дуже важко писати програми ще більшої складності, яких вимагає сучасний ринок праці. Принципово новим підходом стало впровадження об’єктно-орієнтованого програмування. Першою мовою програмування, в якій були запропоновані базові поняття, які згодом об’єдналися у парадигму, була Симула, але термін “об'єктна орієнтованість” не можна було застосувати в контексті використання цієї мови[1]. У 1967 році були запропоновані революційні ідеї: об'єкти, класи, віртуальні методи і ін., проте це не було сприйнято сучасниками як щось революційне та здатне змінити підхід до розробки програмного забезпечення. Погляд на програмування «під іншим кутом» (відмінним від процедурного) запропонували Алан Кей і Ден Інгаллс в мові Smalltalk. Тут поняття класу стало осново твірною ідеєю для всіх інших конструкцій мови (тобто клас в Смолток є примітивом, за допомогою якого описані більш складні конструкції). Саме ця мова стала першою широко поширеною об’єктно орієнтованою мовою програмування. Станом на сьогодні кількість прикладних мов програмування (список мов), що реалізують об'єктно-орієнтовану парадигму, є найбільшою порівняно з іншими парадигмами.

Отже, впровадження ООП стало вагомим внеском у розвитку сучасного програмування раціональним шляхом мінімізації коду та покращенням ефективності.

* 1. Визначення ООП

Об'єктно-орієнтоване програмування (ООП) - методологія програмування, заснована на представленні програми у вигляді сукупності об'єктів, кожен з яких є екземпляром певного класу, а класи можуть утворюють ієрархію спадкування. Лука Карделла і Мартін Абаді побудували теоретичне обґрунтування ООП і класифікацію на основі цього обґрунтування. Вони відзначають, що виділені ними поняття і категорії разом зустрічаються далеко не у всіх об’єктно-орієнтованих мовах, більшість мов підтримують лише підмножини теорії, а часом і своєрідні відхилення від неї.

Ключові поняття:

• квантифікація змінних типу (універсальна, екзистенційна, обмежена);

• під типізація (англ. subtyping - відносини «супертип-підтип»);

• включення (англ. subsumption) - окремий випадок під типізації;

• об'єкт;

• об'єктний тип (фактично, спеціальна форма «запису», в число полів якої входять

• функції першого класу);

• клас.

Клас – це сутність, що описує абстрактну поведінку. Об'єктні типи будуються на основі класу за допомогою додавання різних приватних полів і методів. Об'єкт (тобто значення об'єктного типу, традиційно називається «екземпляром класу») породжується конструктором на основі початкових параметрів. [2]

* 1. Фундаментальні поняття ООП

До фундаментальних понять (принципів), використаних у переважній більшості визначень об'єктно-орієнтованого програмування належать:

* Абстракція даних – виділення лише значущої інформації і виключення її з розгляду незначною. В ООП розглядають лише абстракцію даних (нерідко називаючи її просто «абстракцією»), маючи на увазі набір значущих характеристик об'єкта, доступних решті частини програми.
* Інкапсуляція – властивість системи, що дозволяє об'єднати дані і методи, що працюють з ними, в класі. Одні мови (наприклад, С ++, Java або Ruby) ототожнюють інкапсуляцію з приховуванням, але інші (Smalltalk, Eiffel, OCaml) розрізняють ці поняття.
* Спадкування – властивість, що дозволяє описати новий клас на основі вже існуючого, що передбачає часткове або абсолютне отримання властивостей чи методів батьківського класу. Клас, від якого відбувається спадкування, називається базовим, батьківським або суперкласом. Новий клас – нащадком, спадкоємцем, дочірнім або похідним класом.
* Поліморфізм підтипів – властивість, що дозволяє використовувати об'єкти з однаковим інтерфейсом без інформації про тип і внутрішню структуру об'єкта. Інший вид поліморфізму – параметричний.
* Клас – абстрактні характеристики деякої сутності, включаючи характеристики самої сутності (її атрибути або властивості) та дії, які вона здатна виконувати (її поведінки, методи або можливості). Класи вносять модульність та структурованість в об'єктно-орієнтовану програму. Як правило, клас має бути зрозумілим для осіб, що не мають відношення до програмування, проте знаються на предметній області, що, у свою чергу, значить, що клас повинен мати значення в контексті. Також, код реалізації класу має бути досить самодостатнім. Властивості та методи класу, разом називаються членами даного класу.
* Об’єкт – сутність в адресному просторі обчислювальної системи, що з'являється при створенні екземпляра класу (наприклад, після запуску результатів компіляції і зв'язування вихідного коду на виконання).
  1. Переваги обраного методу

Оптимальність та раціональність вибору того чи іншого методу є однією із фундаментальних основ при розробленні програмного забезпечення для розв’язання конкретної задачі. Від вибору підходу при створенні програми залежить час виконання програми, ресурсозатратність і т.д. Тому грамотне та доцільне рішення може суттєво спростити процес розроблення та вдосконалення програмного забезпечення в майбутньому. Чим більшою стає програма, тим гостріше постає проблема про важливість чітко окресленого поділу на компоненти, абстрагуючись від незначних деталей і в той же час проектуючи складові так, щоб вони могли описати сутності, на базі яких буде передбачено створення об’єктів та описано правила їх взаємодії.[3] Однією із основних переваг використання саме об’єктно-орієнтованого підходу є можливість створювати розширювані та модульні системи, що й дає перевагу над традиційними методами програмування. Розширюваність передбачає, що існуючу систему можна доповнити новими компонентами, причому без внесення до неї будь-яких змін. Компоненти можуть бути додані на етапі виконання.

1. Аналіз та розробка алгоритму згідно індивідуального завдання
   1. Опис алгоритму



Діаграма прецедентів

Актори:

1. Користувач

* Зареєструватися
* Увійти в акаунт
* Викликати таксі
* Переглянути статус всіх викликів користувача
* Скасувати виклик

1. Адміністратор

* Зареєструватися
* Увійти в акаунт
* Викликати таксі
* Скасувати виклик
* Видалення акаунтів
* Перегляд статусу усіх викликів

Як видно з діаграми адміністратор має усі можливості користувача, проте він здатний переглядати дані викликів усіх користувачів та видаляти акаунти.

* 1. Аналіз особливостей алгоритму

1. Функція «Зареєструватися» — доступна лише користувачу. Вона дозволяє створити новий акаунт.
2. Функція «Увійти в акаунт» — доступна і користувачу і адміністратору. Дозволяє увійти у існуючий акаунт.
3. Функція «Викликати таксі» — доступна і користувачу і адміністратору. Дозволяє викликати таксі (створити виклик).
4. Функція «Переглянути статус викликів» — доступна лише користувачу. Дозволяє переглянути статус усіх викликів поточного користувача.
5. Функція «Скасувати виклик» — доступна і користувачу і адміністратору. Дозволяє скасувати вибраний виклик.
6. Функція «Видалення акаунтів» — доступна тільки адміністратору. Дозволяє видалити вибраний акаунт.
7. Функція «Перегляд статусу усіх викликів» — доступна лише адміністратору. Дозволяє переглянути статус викликів усіх користувачів.
   1. Опис алгоритму у псевдокоді

Програма містить такі основні класи:

1. Клас Output — це клас якій відповідає за взаємодію програми з користувачем.
2. Клас Users — клас, який відповідає за зчитування і запис інформації про користувачів в файл.
3. Клас Administrator — клас успадковує методи класу User. Містить додаткові методи для адміністрування.
4. Клас Taxi — клас, який відповідає за створення і скасування викликів, збереження і зчитування інформації про виклики у файл.
   1. Блок-схема алгоритму



Під час запуску програма зчитує з файлу базу даних користувачів. Після цього відбудеться перехід на головне меню, де необхідно вибрати “Вхід в акаунт” або “Реєстрація”. При виборі пункту “Реєстрація” відбудеться перехід у меню реєстрації, де потрібно ввести ім’я користувача і пароль. Після цього відбудеться перехід в головне меню.

При виборі пункту “Вхід в акаунт” відбудеться перехід в меню авторизації, у якому необхідно ввести ім’я користувача і пароль. Якщо пара логін:пароль відсутня у базі даних виведеться повідомлення про неправильність імені користувача або пароля. У випадку коли пара логін:пароль існує у базі даних, відбудеться перехід у меню користувача. Разом з цим загружається база даних з інформацією про виклики.

Меню користувача складається з чотирьох пунктів:

1. Викликати таксі
2. Переглянути статус виклику
3. Скасувати виклик
4. Повернутися в головне меню

Після вибору першого пункту необхідно ввести поточне місцерозташування і пункт прибуття.

Якщо вибрати пункт “Переглянути статус виклику” буде виведений список викликів даного користувача. У списку відображаються наступні дані:

* Номер виклику
* Час до прибуття
* Ціна подорожі
* Поточне місцерозташування
* Пункт прибуття
* Статус

Для скасування виклику необхідно вибрати відповідний пункт в меню користувача і ввести номер виклику який необхідно скасувати.

Якщо вибрати останній пункт відбудеться перехід в головне меню.

1. Програмна реалізація алгоритму згідно індивідуального завдання
   1. Опис послідовної версії програмної реалізації алгоритму
      1. Опис структур даних програми

База даних реалізована за допомогою двох файлів: users.txt і orders.txt.

Users.txt зберігає інформацію про дані авторизації користувачів у форматі логін**:**пароль.

Orders.txt зберігає інформацію про виклик таксі у форматі номер\_виклику**;**час\_до\_прибуття**;**ціна**;**поточне\_місцерозташування**;**пункт\_прибуття**;**статус.

В програмі використовується такі структури: userInfo, orderInfo.

UserInfo містить дані типу string (username, password).

OrderInfo містить дані типу int (orderNum, timeToArrived, price) та string (customerName, destination, curDestination, status).

* + 1. Діаграма класів



Клас users є батьківським класом, який відповідає за роботу з базою даних користувачів.

Клас taxi є батьківським класом, який відповідає за роботу з викликами таксі.

Клас admin є спадкоємцем класів users і taxi, є батьківськім класом для output.

Клас output є спадкоємцем класу admin.

* + 1. Опис реалізації основних функцій/методів

**Клас users:**

Метод saveU();

Метод записує у файл дані про користувачів. Не повертає жодних значень.

Метод loadU();

Метод загружає в пам’ять дані про користувачів. Не повертає жодних значень.

**Клас taxi:**

Метод call();

Метод дозволяє створити новий виклик таксі. Не повертає жодних значень.

Метод getStat();

Метод виводить на екран інформацію про всі виклики поточного користувача. Не повертає жодних даних.

Метод load();

Метод загружає в пам’ять дані про усі виклики усіх користувачів. Не повертає жодних значень.

Метод save();

Метод записує у файл дані про усі виклики усіх користувачів. Не повертає жодних значень.

Метод cancel();

Метод дозволяє скасувати вибраний виклик. Не повертає жодних значень.

Метод sort();

Метод сортує усі виклики в порядку спадання номеру виклику. Не повертає жодних значень.

**Клас admin:**

Метод delUser();

Метод дозволяє видалити вибраного користувача. Не повертає жодних значень.

Метод getAllStat();

Метод виводить на екран інформацію про всі виклики усіх користувача. Не повертає жодних даних.

Метод showAccounts();

Метод дозволяє переглянути список усіх користувачів. Не повертає жодних значень.

**Клас output:**

Метод auth();

Метод відповідає за перевірку наявності акаунта в базі даних. Повертає результат перевірки.

Метод menu();

Метод виводить на екран меню користувача. Не повертає жодних значень.

Метод mainMenu();

Метод виводить на екран вступне меню. Не повертає жодних значень.

Метод newUser();

Даний метод дозволяє створювати новий акаунт користувача. Не повертає жодних значень.

1. Відлагодження та тестування реалізованої програми згідно індивідуального завдання
   1. Опис структури програми

Для відлагодження програми було прийнято рішення використовувати Visual Studio Unit Testing Framework

Visual Studio Unit Testing Framework - інструмент модульного тестування, вбудований в середу розробки Visual Studio 2005 і вище. Щоб визначити, що клас тестується, необхідно позначити його атрибутом [Test\_Class]. Якщо клас позначений цим атрибутом, то він може містити в собі тестові методи. Зазвичай тестовий клас називають так само, як і тестований, тільки з префіксом Test.

В тестуючому класі можуть міститися тестуючі методи і зазвичай для всіх методів тестованого класу, які повертають значення, створюється окремий тестуючий метод. Тестуючі методи зазвичай називають, так само як і тестований, тільки з префіксом Test.

Крім тестуючих методів в якому тестуючому класі можуть бути методи ініціалізації і очищення. Метод ініціалізації позначається атрибутом [TestInitialize] і дозволяє форматувати необхідні змінні перед виконанням методу-тесту. Метод очищення позначається атрибутом [TestCleanup] і дозволяє очистити результати виконання тесту, наприклад, очистити файл, видалити зайві записи з бази даних, привласнити змінним значення за замовчуванням.

* 1. Відлагодження та тестування окремих частин програми

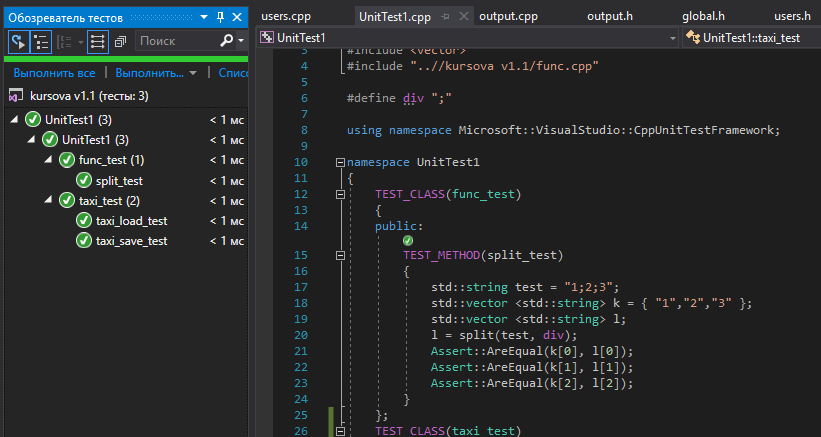
Для роботи програми необхідне середовище Visual Studio 2019, у якому ця програма і розроблялась. Для коректної роботи необхідні такі додаткові файли:

* <iostream> — для виводу інформації в консоль.
* <iomanip> — для форматування виводу.
* <Windows.h> — для коректного відображення українських літер.
* <vector> — необхідна для більш зручного оперування динамічними масивами.
* <string> — потрібна для роботи з рядками.
* <fstream> — для запису і зчитування інформації з файлів.

Також в корінній папці програми повинні міститися такі файли:

* curNum.txt — для зберігання номера останнього виклику.
* orders.txt — база даних усіх викликів.
* users.txt — база даних користувачів.
  1. Відлагодження та тестування програми в цілому

В даному тесті перевіряється функція яка ділить рядок на окремі рядки за певним розділювачем і повертає vector типу string. Також перевіряється функція запису/зчитування інформації з файлу.



1. Аналіз продуктивності (часу роботи) та структурної складності реалізованої програм згідно індивідуального завдання
   1. Аналіз структурної складності виконання програми

У контексті теоретичного аналізу центральним для інформатики є поняття складності. Неформально складність - це "інформаційна опірність середовища", тобто те, що заважає вирішувати завдання, принципово забороняє автоматизувати ті чи інші речі. Крім неформального визначення складності, теоретики інформатики придумали кілька набагато більше добре формалізованих видів складності, які фокусуються на більш вузьких властивостях завдань.

Перший такий вид складності - це **ентропія,** або кількість байт, потрібних для того, щоб описати задачу. Треба сказати, що назви збігаються не даремно, цей вид складності дивно схожий на термодинамічну ентропію як кількість можливих станів. Поняття інформаційної ентропії, яке використовує в своїй роботі К. Шеннон, в рамках конструктивістських ідей розглядається як "складність" структури, що лежить в основі системи (у тому числі і комунікаційної).

Другий вид складності - це "заплутаність" **структур даних.**

Чисельність характеристики для нього немає, але для програміста він найпомітніше. Справа в тому, що більшість алгоритмів "побудовані" у вигляді маніпулювання якимись структурами даних. Часто ускладнення в програмі відбуваються за рахунок не самих алгоритмів, а за рахунок ускладнення структур, що представляють дані . Крім того, ієрархічним ускладнення схильні не тільки структури даних, а й алгоритми роботи з ними. Віртуальна машина, використовувана програмістом при написанні програм на деякій мові високого рівня, є ієрархічною системою віртуальних машин . Важливим з точки зору розвитку комп'ютерних технологій стає цей перехід до мов "високого рівня". З цього моменту програмування дійсно стає діяльністю зі створення складних систем.

Третій вид складності - **колмогорівска.**

Вона являє собою математично оформлений аналог "заплутаність". Наявність або відсутність великих обсягів роботи і складності моделі не вичерпують принципових проблем, з якими стикаються люди при створенні автономних систем. Є ще якась досить неочевидна характеристика, яка говорить нам, наскільки взагалі задача, яку ми намагаємося вирішити, піддасться алгоритмізації. Ця характеристика має числову величину і називається **колмогорівскою складністю.**

Дуже цікаво, що колмогоровской складність має зв'язок з ентропією Шеннона, згаданої на початку глави. І та і інша в деякому розумінні допомагають формалізувати задачу "зниження рівня невизначеності".

Якщо з цих позицій підійти до поняття обчислення з точки зору математики, то "складність" в сенсі алгорітмізіруемості - це складність щодо оптимального мови, вимірюють довжину найкоротшого опису цього обчислення. Чим складніше об'єкт, тим більше інформації він містить. Ця "довжина опису", що задає модель обчислень, яка описується як довжина довічних слів (технічно - кінцеві послідовності бітів) і називається " **колмогорівскою складністю** ".

Тобто на інженерному рівні складність і буде описом способу конструювання, хоча на рівні теорії вичислювальних функцій вона задається функціонально: "... спосіб опису є обчислювана функція з множин довічних слів".

У певному сенсі, в математиці колмогорівской складності доведено, що "для випадкових даних" колмогорівской складність (довжина програми, що моделює ці дані) буде приблизно дорівнює ентропії Шеннона цих даних. Проте показано, що для невипадкових даних колмогорівской складність мала, що, власне, і дозволяє програмістам писати програми, що моделюють різні системи.

Четвертий різновид складності - **обчислювальна.**

Складність обчислювальних процесів — це поняття теорії складності обчислень, оцінка ресурсів (зазвичай часу) необхідних для виконання алгоритму. Для оцінки алгоритмів існує багато критеріїв. Найбільшу увагу приділяють порядку росту необхідних для розв'язання задачі часу та розміру пам'яті при збільшенні розміру вхідних даних.

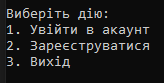
П’ятий різновид складності – **часова.**

Часова складність алгоритму в комп'ютерних науках є обчислювальною складністю алгоритму, яка описує час потрібний для виконання алгоритму. Вона зазвичай визначається шляхом підрахунку кількості елементарних операцій, виконуваних алгоритмом, при цьому вважають, що кожна елементарна операція виконується за фіксовану кількість часу. Таким чином, кількість часу і кількість елементарних операцій, необхідних для виконання алгоритму, відрізняються постійним множником.

* 1. Аналіз часу виконання програми в залежності від об'єму вхідних даних

Всі операції всередині програми виконуються миттєво, немає затримки між переходами по меню та виведенням інформації на консоль. Читання даних з файлу та виконання операцій з числами виконується бездоганно.

Під час запуску програми користувача зустрічає консольний інтерфейс. Який дає змогу зареєструвати нового користувача або увійти у існуючий акаунт.



Меню входу в систему

Спочатку зареєструємо новий акаунт. Потрібно ввести ім’я користувача і придумати пароль.



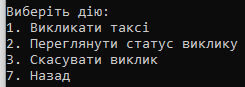
Реєстрація нового користувача

Тепер увійдем в створений акаунт.



Вхід в акаунт

Ми увійшли у головне меню користувача.



Меню користувача

Виберемо пункт викликати таксі. Необхідно ввести поточне місце розташування і пункт прибуття.



Переглянемо статус виклику.



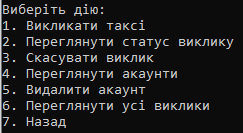
Статус викликів поточного користувача

Тепер увійдемо у акаунт адміністратора.

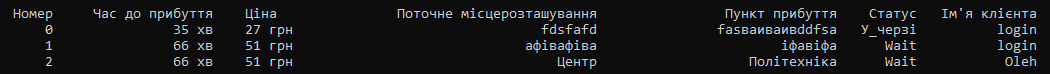


Вхід в акаунт адміністратора

В головному меню адміністратора доступно значно більше функцій ніж у меню користувача.



Головне меню адміністратора



Список усіх викликів усіх користувачів

Висновки

В результаті виконання курсової роботи, я вивчив базові принципи об’єктно – орієнтованого програмування та закріпив знання на практиці. Створив програму, яка симулює роботу служби таксі. На практиці зміг застосувати основні принципи мови програмування С++: інкапсуляцію, поліморфізм та наслідування. У загальному в мене вийшло 4 класи ,21 з яких є батьківським і 2 інші наслідують ці класи. Можна зробити висновки, що С++ дає великі можливості і набагато спрощує роботу для програміста, оскільки за допомогою гілки класів я створив систему для виклику таксі, зумів створити набір операцій за допомогою яких можна створити базу даних користувачів і викликів.

Використані джерела:

1. Бадд Т. Об'єктно-орієнтоване програмування в дії / Перекл. з англ. - Спб .: Питер, 1997. - 464 с.
2. Павловская Т.А. С/С++. Программирование на языке высокого уровня. – СПб.: Питер, 2003. – 461с.
3. Г. Буч. Об'єктно-орієнтований аналіз та проектування з прикладами додатків на С ++, 2-е вид. / Пер. з англ. - М .: «Видавництво Біном», СПб: «Невський діалог», 1998 р - 560 с.

Додатки

**Global.h**

#ifndef \_GLOBAL\_H\_

#define \_GLOBAL\_H\_

#include <iostream>

#include <iomanip>

#include <Windows.h>

#include <vector>

#include <string>

#include <fstream>

#include <iterator>

using namespace std;

#define udiv ":"

#define div ";"

extern int curNum;

struct userInfo

{

string username, password;

};

struct orderInfo

{

int orderNum, timeToArrived, price;

string customerName, destination, curDestination, status;

};

extern userInfo curUser;

#endif

**Func.h**

std::vector <std::string> split(std::string str, std::string divider);

**Func.cpp**

#include "global.h"

#include "func.h"

using namespace std;

vector <string> split(string str, string divider)

{

vector <string> res;

int start = 0, end, delta = divider.length(), i = 0;

while ((end = str.find(divider, start)) != string::npos)

{

res.push\_back(str.substr(start, end - start));

start = end + delta;

}

res.push\_back(str.substr(start));

return res;

}

**Users.h**

#include "global.h"

class users

{

private:

public:

std::vector <userInfo> usersData;

users();

~users();

void saveU();

void loadU();

};

**Users.cpp**

#include "users.h"

#include "taxi.h"

#include "func.h"

using namespace std;

userInfo curUser;

users::users()

{

}

users::~users()

{

}

void users::saveU()

{

ofstream file("users.txt");

for (unsigned int i = 0; i < usersData.size() - 1; i++)

{

file << usersData[i].username << udiv << usersData[i].password << endl;

}

file.close();

}

void users::loadU()

{

int i = 0;

usersData.clear();

string temp;

vector <string> buff;

ifstream file("users.txt");

while (file.eof() != 1)

{

file >> temp;

buff = split(temp, udiv);

usersData.resize(usersData.size() + 1);

usersData[i].username = buff[0];

usersData[i].password = buff[1];

i++;

}

};

**Taxi.h**

#include "global.h"

class taxi

{

private:

public:

std::vector <orderInfo> ordersData;

std::vector <orderInfo> curUserOrders;

taxi();

~taxi();

void call();

void getStat();

void load();

void save();

void cancel();

void sort();

};

**Taxi.cpp**

#include "taxi.h"

#include "func.h"

using namespace std;

int curNum = 0;

taxi::taxi()

{

ifstream file("curNum.txt");

string temp;

file >> temp;

curNum = stoi(temp);

file.close();

}

taxi::~taxi()

{

}

void taxi::call()

{

orderInfo newOrder;

newOrder.orderNum = curNum;

newOrder.status = "Wait";

newOrder.price = rand() % (99 - 10 + 1) + 10;

newOrder.timeToArrived = newOrder.price \* 1.3;

cout << "Введіть ваше місцерозташування: ";

cin >> newOrder.curDestination;

cout << "Введіть пункт прибуття: ";

cin >> newOrder.destination;

newOrder.customerName = curUser.username;

ofstream file("orders.txt", ios\_base::app);

file << newOrder.orderNum << div << newOrder.timeToArrived << div << newOrder.price << div << newOrder.curDestination << div << newOrder.destination << div << newOrder.customerName << div << newOrder.status << endl;

file.close();

curNum++;

ofstream fileNum("curNum.txt");

fileNum << curNum;

fileNum.close();

}

void taxi::getStat()

{

load();

locale::global(locale(""));

unsigned int j = 0;

for (unsigned int i = 0; i < ordersData.size(); i++)

{

if (ordersData[i].customerName == curUser.username)

{

curUserOrders.resize(curUserOrders.size() + 1);

curUserOrders[j] = ordersData[i];

j++;

}

}

cout << setw(10) << "Номер" << setw(20) << "Час до прибуття" << setw(8) << "Ціна" << setw(40) << "Поточне місцерозташування" << setw(30) << "Пункт прибуття" << setw(10) << "Статус" << endl;

for (unsigned int i = 0; i < curUserOrders.size() - 1; i++)

{

cout << setw(10) << curUserOrders[i].orderNum << setw(17) << curUserOrders[i].timeToArrived << " хв" << setw(6) << curUserOrders[i].price << " грн" << setw(38) << curUserOrders[i].curDestination << setw(30) << curUserOrders[i].destination << setw(10) << curUserOrders[i].status << endl;

}

curUserOrders.clear();

ordersData.clear();

}

void taxi::load()

{

int i = 0;

ordersData.clear();

string temp;

vector <string> buff;

ifstream file("orders.txt");

while (file.eof() != 1)

{

file >> temp;

buff = split(temp, div);

ordersData.resize(ordersData.size() + 1);

ordersData[i].orderNum = stoi(buff[0]);

ordersData[i].timeToArrived = stoi(buff[1]);

ordersData[i].price = stoi(buff[2]);

ordersData[i].curDestination = buff[3];

ordersData[i].destination = buff[4];

ordersData[i].customerName = buff[5];

ordersData[i].status = buff[6];

i++;

}

}

void taxi::save()

{

ofstream file("orders.txt");

for (unsigned int i = 0; i < ordersData.size() - 1; i++)

{

file << ordersData[i].orderNum << div << ordersData[i].timeToArrived << div << ordersData[i].price << div << ordersData[i].curDestination << div << ordersData[i].destination << div << ordersData[i].customerName << div << ordersData[i].status << endl;

}

file.close();

}

void taxi::cancel()

{

unsigned int num;

getStat();

cout << "Введіть номер виклику ";

cin >> num;

load();

for (unsigned int i = 0; i < ordersData.size(); i++)

{

if (ordersData[i].orderNum == num)

{

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

save();

}

}

ordersData.clear();

sort();

}

void taxi::sort()

{

load();

unsigned int i;

for (i = 0; i < ordersData.size(); i++)

{

ordersData[i].orderNum = i;

}

save();

ofstream fileNum("curNum.txt");

curNum = i - 1;

fileNum << i - 1;

fileNum.close();

ordersData.clear();

}

**Admin.h**

#include "users.h"

#include "taxi.h"

class admin : public users, public taxi

{

public:

admin();

~admin();

void delUser();

void getAllStat();

void save();

void showAccounts();

};

**Admin.cpp**

#include "admin.h"

admin::admin()

{

}

admin::~admin()

{

}

void admin::delUser()

{

unsigned int num;

showAccounts();

cout << "Введіть номер ";

cin >> num;

loadU();

usersData.erase(usersData.begin() + num);

saveU();

usersData.clear();

}

void admin::showAccounts()

{

loadU();

cout << setw(6) << "Номер" << setw(18) << "Ім'я користувача" << endl;

for (unsigned int i = 0; i < usersData.size() - 1; i++)

{

cout << i << " " << usersData[i].username << endl;

}

usersData.clear();

}

void admin::getAllStat()

{

load();

unsigned int j = 0;

cout << setw(10) << "Номер" << setw(20) << "Час до прибуття" << setw(8) << "Ціна" << setw(40) << "Поточне місцерозташування" << setw(30) << "Пункт прибуття" << setw(10) << "Статус" << setw(15) << "Ім'я клієнта" << endl;

for (unsigned int i = 0; i < ordersData.size() - 1; i++)

{

cout << setw(10) << ordersData[i].orderNum << setw(17) << ordersData[i].timeToArrived << " хв" << setw(6) << ordersData[i].price << " грн" << setw(38) << ordersData[i].curDestination << setw(30) << ordersData[i].destination << setw(10) << ordersData[i].status << setw(15) << ordersData[i].customerName << endl;

}

ordersData.clear();

}

**Output.h**

#include "admin.h"

class output : public admin

{

public:

output();

~output();

void auth();

void menu();

void mainMenu();

void newUser();

};

**Output.cpp**

#include "output.h"

output::output()

{

}

output::~output()

{

}

void output::auth()

{

userInfo authUser;

bool correct = 0;

cout << "Введіть ім'я користувача: ";

cin >> authUser.username;

cout << "Введіть пароль: ";

cin >> authUser.password;

loadU();

for (unsigned int i = 0; i < usersData.size(); i++)

{

if (usersData[i].username == authUser.username && usersData[i].password == authUser.password)

{

curUser.username = usersData[i].username;

curUser.password = usersData[i].password;

correct = 1;

menu();

break;

}

}

if (correct == 0)

{

cout << "Неправильне ім'я користувача або пароль";

mainMenu();

}

usersData.clear();

menu();

};

void output::menu()

{

taxi taxiOr;

cout << endl << "Виберіть дію:" << endl;

cout << "1. Викликати таксі" << endl;

cout << "2. Переглянути статус виклику" << endl;

cout << "3. Скасувати виклик" << endl;

if (curUser.username == "admin")

{

cout << "4. Переглянути акаунти" << endl;

cout << "5. Видалити акаунт" << endl;

cout << "6. Переглянути усі виклики" << endl;

}

cout << "7. Назад" << endl;

int ch;

cin >> ch;

if (curUser.username != "admin" && ch > 3 && ch < 7)

{

ch = 0;

}

switch (ch)

{

case 1:

{

taxiOr.call();

menu();

}

case 2:

{

taxiOr.getStat();

break;

}

case 3:

{

taxiOr.cancel();

break;

}

case 4:

{

showAccounts();

break;

}

case 5:

{

delUser();

break;

}

case 6:

{

getAllStat();

menu();

break;

}

case 7:

{

mainMenu();

break;

}

default:

menu();

}

};

void output::mainMenu()

{

cout << endl << "Виберіть дію:" << endl;

cout << "1. Увійти в акаунт" << endl;

cout << "2. Зареєструватися" << endl;

cout << "3. Вихід" << endl;

int ch;

cin >> ch;

switch (ch)

{

case 1:

{

auth();

break;

}

case 2:

{

newUser();

break;

}

case 3:

{

exit(0);

break;

}

default:

mainMenu();

}

}

void output::newUser()

{

userInfo addUser;

cout << "Введіть ім'я користувача: ";

cin >> addUser.username;

cout << "Придумайте пароль: ";

cin >> addUser.password;

ofstream file("users.txt", ios\_base::app);

file << addUser.username << udiv << addUser.password << endl;

file.close();

mainMenu();

};

**Main.cpp**

#include "output.h"

using namespace std;

output user;

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

user.mainMenu();

return 0;

}