**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

**Кафедра автоматизації проектування  
енергетичних процесів і систем**

***КУРСОВА РОБОТА***

**з дисципліни: «Об’єктно-орієнтоване програмування - 1»**

**тема:**

**«Створення моделей об’єктів реального світу на мові C++»**

|  |  |
| --- | --- |
| **Керівник – Карпенко С.Г.** | **Виконав Артамонов О.Ю.** |
| **Допущено до захисту** | **Студент 2-го курсу** |
| **\_\_ \_\_\_\_\_\_\_\_ 2017 р.** | **Групи ТВ-61** |
| **Захищено з оцінкою** | **залікова книжка № ТВ-6121** |
| **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  |

***2017***

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

Кафедра автоматизації проектування  
енергетичних процесів і систем

**КУРСОВА РОБОТА**

з дисципліни «Об’єктно-орієнтоване програмування»

(назва дисципліни)

на тему: «Створення моделей об’єктів реального світу на мові C++»

Студента групи ТВ-61

напряму підготовки ***121* «Інженерія програмного забезпечення»**

спеціальності «**Програмне забезпечення розподілених систем»**

Артамонова О.Ю.

(прізвище та ініціали)

Керівник доцент, канд. фіз.-мат. наук Карпенко С.Г.

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

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_ доц., к. ф.-м. н. Карпенко С.Г.

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_ доц., к.е.н. Левченко Л.О.

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_ доц., к.т.н. Шаповалова С.І.

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали

Київ - 2017 рік

**Національний технічний університет України**

**“Київський політехнічний інститут імені Шгоря Сікорського”**

Факультет ТЕПЛОЕНЕРГЕТИЧНИЙ ФАКУЛЬТЕТ

( повна назва)

Кафедра автоматизації проектування енергетичних процесів та систем

(повна назва)

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки ***121  «Інженерія програмного забезпечення»***

Спеціальність *«****Програмне забезпечення розподілених систем»***

З А В Д А Н Н Я

**НА КУРСОВУ РОБОТУ СТУДЕНТУ**

Артамонов Олексій Юрійович

(прізвище, ім’я, по батькові)

1. Тема роботи «Створення моделей об’єктів реального світу на мові C++»

керівник курсової роботи –

Карпенко Станіслав Григорович, канд. фіз.-мат. наук, доцент

( прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи – 22 грудня 2017 р.

3. Вихідні дані до проекту (роботи): мова C++, модель об’єкту реального світу

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) – Розробити та обґрунтувати ієрархічну модель класів для опису об’єкту реального світу, використовуючи інструментальні можливості мови C#, зокрема статичні поля, властивості, перевантажені операції, інтерфейси, індексатори, виключення та депутати.

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

графічне зображення ієрархічної моделі класів з урахуванням як успадковування, так й включення; зображення масиву основних даних реальних об’єктів у вигляді таблиці.

7. Дата видачі завдання – 30 жовтня 2017 р.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів виконання курсової роботи | Строк виконання етапів роботи | Примітка |
|  | Побудова ієрархічної моделі класів | 17.11.2017 |  |
|  | Розробка та застосування інструментальних засобів обробки даних | 08.12.2017 |  |
|  | Написання розрахунково-пояснювальної записки | 22.12.2017 |  |

**Студент \_\_\_\_\_\_\_\_\_ Артамонов О.Ю.**

**Керівник курсової роботи \_\_\_\_\_\_\_\_\_ Карпенко С.Г.**

# АНОТАЦІЯ

Ця програма створена для запису та маніпуляції данними деякого ресторану. Вона дозволяє користувачу заповнювати дані про закази, наприклад дату заказу, кількість страв у ньому та ін. Також програма дозоволяє переглядати їх у табличному вигляді, зберігати данні на диск у текстовий та бінарний файли. У програмі також присутній пошук данних по числовим та символьним полям.

У записці розповідається про основні поняття ООП, можливості мови програмування C++, обґрунтовується вибір студента стосовно типів зв’язків між класами, типів властивостей, виключних місць та делегатів статистичних функцій.

This program is designed to record and manipulate the data of some restaurant. It allows the user to fill in the details of orders, such as the order date, the number of dishes in it, etc. Also, the program allows you to view them in tabular form, save data to disk in text and binary files. The program also has a search for numerical and symbol fields.

The note describes the basic notions of the OOP, the possibilities of the C ++ programming language, and justifies the choice of the student in relation to the types of relationships between classes, types of properties, exclusive places, and delegates of statistical functions.

# ЗМІСТ

[ЗМІСТ 4](#_Toc419224434)

[ВСТУП 5](#_Toc419224435)

[1.ТЕОРЕТИЧНІ ВІДОМОСТІ 8](#_Toc419224436)

[1.1.Принципи Об’єктно-орієнтованого програмування 8](#_Toc419224437)

[1.2.Середовище MVS 10](#_Toc419224438)

[1.3.Перевантаження операцій 1](#_Toc419224440)1

[1.4.Робота з файлами 1](#_Toc419224441)1

1.5.Шаблони 12

[1.6.Вибір виключних ситуацій таа іх обробка 1](#_Toc419224442)4

[2.ОБҐРУНТУВАННЯ ІЄРАРХІЇ КЛАСІВ 1](#_Toc419224443)5

[2.1.Успадкування 1](#_Toc419224444)6

[2.2.Агрегація 1](#_Toc419224445)7

2.3. Вибір конкретних значень обмежень для полів різних класів 17

[2.4 Структура програми](#_Toc419224446) 17

[3.ОБҐРУНТУВАННЯ ВИБОРУ 2](#_Toc419224447)0

[3.1.Вибір типу властивостей 2](#_Toc419224448)0

[3.2.Вибір перевантаження операцій 2](#_Toc419224449)0

[3.3.Вибір виключних місць 2](#_Toc419224450)0

[3.4.Вибір типу делегата для статистичних функцій 2](#_Toc419224451)1

[ВИСНОВКИ 2](#_Toc419224452)2

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 2](#_Toc419224453)3

[ДОДАТОК 1. ТЕКСТ ПРОГРАМИ 2](#_Toc419224454)4

[ДОДАТОК 2. ОПИС ПРОГРАМИ 5](#_Toc419224455)6

# ВСТУП

Природне прагнення розробників програм - скоротити час розробки, полегшити повторне використання налагоджених модулів і знизити витрати на супровід і модифікацію програм.

Для досягнення цих цілей в галузі створення програмних комплексів використовують методи і підходи управління процесом розробки. На різних етапах розвитку програмної інженерії використовувалися різні технології програмування - імперативне програмування; модульне програмування; структурне програмування; програмування, кероване даними; програмування, кероване подіями; функціональне програмування; логічне програмування і т.п. Тепер неможливо взяти участь в дискусії, присвяченій програмування, якщо не використовувати термін "об'єктно-орієнтоване програмування".

Розвиток технології і мов програмування

На зорі появи обчислювальних машин програмування, як область знання, знаходилося в зародковому стані. Перші програми створювалися за допомогою перемикачів на панелі комп'ютера. Очевидно, що такий спосіб підходив тільки для невеликих програм. Потім програми стали писати на мові машинних команд, а винахід асемблера дозволило писати вже порівняно довгі програми. Наступний крок був зроблений в 1950 році, коли був створений перший мова програмування високого рівня Фортран.

Тепер програмісти могли створювати програми довжиною до декількох тисяч рядків довжиною. Однак мова програмування, легко зрозумілий в простих програмах, коли справа стосувалася великих програм, ставав нечитабельним (і некерованим). Позбавлення від таких неструктурованих програм прийшло після винаходу на початку 70-х років мов структурного програмування (Алгол, Паскаль і С). Структурне програмування має на увазі точно позначені керуючі структури, програмні блоки відсутність (або мінімальне використання) операторів GOTO, автономні підпрограми, в яких підтримується рекурсія і локальні змінні. З появою структурного програмування з'явилася можливість розбиття програми на складові її елементи. Тепер вже один програміст був в змозі створити і підтримувати програму в кілька десятків тисяч рядків диною.

Хоча структурне програмування і принесло видатні результати, навіть воно виявилося неспроможним, коли програма досягала певної довжини. Щоб писати більш складну програму, необхідний був новий підхід до програмування. В результаті були розроблені принципи об'єктно-орієнтованого програмування, яке акумулювало кращі ідеї, втілені в структурному програмуванні, в поєднанні з потужними новими концепціями, що дозволяють оптимально організувати ваші програми. ООП дозволяє розкласти проблему на пов'язані між собою завдання. Кожна проблема стає самостійним об'єктом, який містить свої власні коди і дані, які відносяться до цього об'єкту. В цьому випадку вихідна задача в цілому спрощується, і програміст отримує можливість оперувати з набагато більшими за обсягом програмами.

С ++ - це спроба вирішення розробниками мови З завдань об'єктно-орієнтованого програмування. С ++ був розроблений співробітником дослідницької лабораторії компаніі «AT & T» Бьерном Страуструпом (Bjarn Stroustrup) в 1980 році. У своїх історичних зауваженнях Страуструп пояснює, чому в якості базової мови був обраний З:

- багатоцільовий, лаконічний і щодо низького рівня;

- відповідає більшості завдань системного програмування;

- "йде скрізь і на всьому";

- придатний в середовищі програмування UNIX.

Перша назва "С з класами" в 1983 році, за пропозицією Ріка Массітті (Rick Mascitti), було змінено на С ++. В цьому ж році С ++ був вперше застосований за межами дослідницької групи. З 1980 року С ++ зазнав дві суттєві зміни: в 1985 і 1990 роках. Перший робочий проект мови С ++ стандарту ANSI (American National Standarts Institute) був представлений в січні 1994 року.

14 листопада 1997 року Міжнародна організація стандартизації (International Standarts Organization, ISO) затвердила стандарт С ++. Бйорн Страуструп високо оцінив новий стандарт, зазначивши, що описана в ньому реалізація набагато ближче до ідеалу, ніж первісна версія мови. Очікується, що фірми-розробники приведуть свої продукти відповідно до стандарту, що має радикально поліпшити переносимість написаних на С ++ програм.

Повертаючись до теми курсової роботи, слід сказати, що конструювання ієрархії об’єктів та виокремлення сутностей загалом є найважливішим етапом конструювання програмного забезпечення на об’єктно-орієнтованій мові. Сама можливість закарбувати об’єкт реального світу в програмному коді полегшує її конструювання та розбиття на окремі модулі. При чому не важливо, чи ця сутність є конкретним предметом чи абстрактним поняттям, бо для таких випадків об’єктно-орієнтовані мови оснащенні спеціальними поняттями та відповідними синтаксичними конструкціями, як, наприклад, модифікатор abstract, що забороняє створювати об’єкти класу, помічені цим словом. На даному етапі відбувається вибір типу відношень між об’єктами різних класів та область видимості певних їх частин. З’являються поняття успадкування та агрегація, що у свою чергу ділиться на чисту агрегацію та композицію.

Після побудови ієрархії написаний код ще не є готовим продуктом – він позбавлений інструментів, завдяки яким відбувається маніпулювання об’єктами цієї ієрархії. При цьому важливо закінчити модифікування ієрархії до початку створення інструментарію. Далі програміст зіштовхується із проблемою взаємодії програми із користувачем – зрозумілий інтерфейс, захист від помилкових даних. Тут на допомогу приходить обробка виключних ситуацій, що є особливістю об’єктно-орієнтованих мов.

Отже, курсова робота охоплює як основні аспекти конструювання програмного забезпечення на об’єктно-орієнтованих мовах, так і особливі для мови C++ синтаксичні конструкції і принципи, серед яких обробка виключних ситуацій та інші.

# 1.ТЕОРЕТИЧНІ ВІДОМОСТІ

### 1.1 Принципи Об’єктно-орієнтованого програмування

ООП будується на багатьох парадигмах, але виділяють чотири основних:

* Абстракція;
* Інкапсуляція;
* Успадкування;
* Поліморфізм.

Абстракція – надання об’єкту характеристик, що відрізняють його від усіх подібних об’єктів, чітко визначаючи його концептуальні границі. Основна ідея у тому, щоб відділити спосіб використання складових об’єктів даних від деталей їх реалізації. У кожному конкретному випадку застосовується той чи інший підхід: інкапсуляція, поліморфізм або успадкування. Наприклад, за необхідності звернутися до прихованих даних об’єкта, слід користуватись інкапсуляцією, створивши, так звану, функцію доступу або властивість. Абстракція даних – популярна техніка програмування, яка загалом визначається неправильно. Фундаментальна ідея полягає в поділі несуттєвих деталей реалізації підпрограми і характеристик істотних для коректного її використання. Такий поділ може бути виражений через спеціальний « інтерфейс », що зосереджує опис всіх можливих застосувань програми.

Інкапсуляція - властивість мови програмування , що дозволяє користувачеві не замислюватися про складність реалізації використовуваного програмного компонента ( що у нього всередині? ), а взаємодіяти з ним за допомогою наданого інтерфейсу (публічних методів і членів) , а також об'єднати і захистити життєво важливі для компонента дані. При цьому користувачеві надається тільки специфікація (інтерфейс ) об'єкта . Користувач може взаємодіяти з об'єктом тільки через цей інтерфейс . Реалізується за допомогою ключового слова: public. Користувач не може використовувати закриті дані і методи. Реалізується за допомогою ключових слів: private, protected. Приховування реалізації доцільно застосовувати в наступних випадках: гранична локалізація змін при необхідності таких змін, прогнозованість змін (які зміни в коді треба зробити для заданого зміни функціональності) і прогнозованість наслідків змін.

Успадкування - один з найважливіших механізмів об'єктно-орієнтованого програмування, що дозволяє описати новий клас на основі вже існуючого ( батьківського), при цьому властивості і функціональність батьківського класу запозичуються новим класом . Іншими словами , клас - спадкоємець реалізує специфікацію вже існуючого класу (базовий клас). Це дозволяє звертатися до об'єктів класу - спадкоємця так само, як до об'єктів базового класу. Просте успадкування : клас , від якого відбулося успадкування , називається базовим або батьківським (англ. Base class ). Класи, які походять від базового, називаються нащадками, спадкоємцями або похідними класами (англ. Derived class ). Абстрактний клас - це клас, що містить хоча б один абстрактний метод, він описаний в програмі, має поля, методи і не може використовуватися для безпосереднього створення об'єкта. Тобто від абстрактного класу можна тільки наслідувати. Об'єкти створюються тільки на основі похідних класів, успадкованих від абстрактного. Наприклад, абстрактним класом може бути базовий клас «співробітник вузу», від якого успадковуються класи «аспірант», «професор» і т.д. Так як похідні класи мають спільні поля і функції (наприклад, поле «рік народження »), то ці члени класу можуть бути описані в базовому класі . У програмі створюються об'єкти на основі класів «аспірант», «професор» , але немає сенсу створювати об'єкт на основі класу «співробітник вузу».

Множинне спадкування

При множинному успадкуванні у класу може бути більше одного предка. У цьому випадку клас успадковує методи всіх предків. Переваги такого підходу в більшій гнучкості. Множинне успадкування реалізовано в C ++. Множинне успадкування підтримується в мові UML. Але множинне успадкування - потенційне джерело помилок, які можуть виникнути через наявність однакових імен методів у предках. У мовах, які позиціонуються як спадкоємці C ++ (Java, C # та ін.), від множинного успадкування було вирішено відмовитися на користь інтерфейсів. Практично завжди можна обійтися без використання даного механізму.

Поліморфізм - можливість об'єктів з однаковою специфікацією мати різну реалізацію. Мова програмування підтримує поліморфізм , якщо класи з однаковою специфікацією можуть мати різну реалізацію - наприклад , реалізація класу може бути змінена в процесі спадкування. Коротко зміст поліморфізму можна виразити фразою: « Один інтерфейс, безліч реалізацій ». Поліморфізм дозволяє писати більш абстрактні програми і підвищити коефіцієнт повторного використання коду. Загальні властивості об'єктів об'єднуються в систему, яку можуть називати по-різному - інтерфейс, клас. Спільність має зовнішній і внутрішній вираз: зовнішня спільність проявляється як однаковий набір методів з однаковими іменами і сигнатурами ( ім'ям методів і типами аргументів і їх кількістю ) ; внутрішня спільність - однакова функціональність методів .

### 1.2 Середовище MVS

Microsoft Visual Studio - це гарне середовище розробки додатків під ОС Windows. Microsoft Visual Studio надає користувачеві, при створенні проекту велика кількість різних типів проектів, що безумовно плутає навіть бувалого програміста, не те що початківця. Але велика кількість типів проектів дає гарне уявлення про можливості даного інструменту. MVS – це дійсно висококласна IDE, якою користуються більшість професіоналів для розробки додатків під ОС Windows.

Для програмування під Windows, немає нічого кращого, ніж Visual Studio – це досить потужна IDE. Її редактор підсвічує синтаксис і виконує форматування коду, що в свою чергу в разі робить код більш зрозумілишим. Більш того, редактор MVS автоматично завершує деякі структури коду, нам варто почати друкувати, наприклад, оператор вибору switch, редактор сам його завершить.

### 1.3 Перевантаження операцій

У багатьох мовах програмування використовуються оператори: як мінімум, привласнення (=,: = або схожі) і арифметичні оператори (+, -, \* і /). У більшості мов зі статичної типізацією ці оператори прив'язані до типів. Наприклад, в Java складання з оператором + можливо лише для цілих чисел, чисел з плаваючою комою і рядків. Якщо ми визначимо свої класи для математичних об'єктів, наприклад, для матриць, ми можемо реалізувати метод їх складання, але викликати його можна лише чимось на зразок цього: a = b.add(c).

У C ++ цього обмеження немає - ми можемо перевантажити практично будь-який відомий оператор. Можливостей не злічити: можна вибрати будь-яку комбінацію типів операндів, єдиним обмеженням є необхідність того, щоб був присутній як мінімум один операнд призначеного для користувача типу. Тобто визначити новий оператор над вбудованими типами або переписати існуючий.

Перевантажені оператори повинні діяти так само, як і їхні базові версії. Природно, допустимі винятки, але лише в тих випадках, коли вони супроводжуються зрозумілими поясненнями. Наочним прикладом є оператори << і >> стандартної бібліотеки iostream, які явно ведуть себе не як звичайні оператори бітового зсуву.

### 1.4 Робота з файлами

Файли дозволяють користувачеві зчитувати великі обсяги даних безпосередньо з диска, не вводячи їх з клавіатури. Існують два основних типи файлів: текстові і двійкові. Текстовими називаються файли, що складаються з будь-яких символів. Вони організовуються по рядках, кожна з яких закінчується символом «кінця рядка». Кінець самого файлу позначається символом «кінця файлу». При запису інформації в текстовий файл, переглянути який можна за допомогою будь-якого текстового редактора, всі дані перетворюються до символьного типу і зберігаються в символьному вигляді.

У довічних файлах інформація зчитується і записується у вигляді блоків певного розміру, в яких можуть зберігатися дані будь-якого виду і структури.

Для роботи з файлами використовуються спеціальні типи даних, звані потоками. Потік ifstream служить для роботи з файлами в режимі читання, а ofstream в режимі запису. Для роботи з файлами в режимі як записи, так і читання служить потік fstream.

У програмах на C ++ при роботі з текстовими файлами необхідно підключати бібліотеки iostream і fstream.

Для того щоб записувати дані в текстовий файл, необхідно:

1. описати змінну типу ofstream.

2. відкрити файл за допомогою функції open.

3. вивести інформацію в файл.

4. обов'язково закрити файл.

Для зчитування даних з текстового файлу, необхідно:

1. описати змінну типу ifstream.

2. відкрити файл за допомогою функції open.

3. зчитати інформацію з файлу, при зчитуванні кожної порції даних необхідно перевіряти, чи досягнуто кінець файлу.

4. закрити файл.

### 1.5 Шаблони

Оголошення і визначення шаблону функції починається ключовим словом template, за яким слід укладений в кутові дужки і розділений комами непорожній список параметрів шаблону. Ця частина оголошення або визначення зазвичай називається заголовком шаблону.

Кожен параметр шаблону складається з службового слова class, за яким йде ідентифікатор. У контексті оголошення шаблону функції службове слово class не несе ніякої особливого смислового навантаження. Справа в тому, що аналогічна конструкція використовується також і для оголошення шаблону класу, де, як скоро побачимо, ключове слово class грає свою особливу роль. У заголовку шаблону імена параметрів шаблону повинні бути унікальні.

Слідом за заголовком шаблону розташовується прототип або визначення функції - все залежить від контексту програми. Як відомо, у прототипу і визначення функції також є власний заголовок. Цей заголовок складається з специфікатор повертається (цілком можливо, що специфікатором повертається може виявитися ідентифікатор зі списку параметрів шаблону), ім'я функції і список параметрів. Всі до одного ідентифікатори з заголовка шаблону повинні входити в список параметрів функції. У цьому списку вони грають роль специфікаторів типу. Оголошення параметрів, у яких в якості специфікатор типу використовується ідентифікатор зі списку параметрів шаблону, називається шаблонним параметром. Поряд з шаблонними параметрами в список параметрів функції можуть також входити параметри основних і похідних типів.

Шаблон функції служить інструкцією для транслятора. З цієї інструкції транслятор може самостійно побудувати визначення нової функції.

Параметри шаблону в шаблонних параметрах функції позначають місця майбутньої підстановки, яку здійснює транслятор в процесі побудови функції. Область дії параметрів шаблону обмежується шаблоном. Тому в різних шаблонах дозволено використання одних і тих же ідентифікаторів-імен параметрів шаблону.

У моїй программі функції які призначені для підрахунку статистичних даних не шаблонні, бо використовування шаблонів не є практичним у цьому випадку, але функції для запису та зчитування данних до файлів - шаблонні, бо там це має сенс, тобто я можу записати та зчитати масив обьектів будь-якого типу.

Приклад шаблону функції:

**template<typename T>**

**void Binary\_Write(T &obj) {**

**ofstream Fb("Binary.dat", ios::binary);**

**if (!Fb.is\_open())**

**throw Bad\_File\_Write\_Exeption("Ошибка! Бинарный файл не был открыт для записи!");**

**else {**

**Fb.write((char\*)(&obj), sizeof(T));**

**cout << "\nБинарный файл успешно создан!" << endl;**

**}**

**Fb.close();**

**}**

### 1.6 Виключні ситуації та їх обробка

Механізм обробки виняткових ситуацій в мові C++ заснований на трьох ключових словах: try, catch і throw. Фрагменти програми, що підлягають контролю, містять блок try. Якщо в ході виконання програми в блоці try виникає виняткова ситуація, вона генерується за допомогою ключового слова throw. Потім виняткова ситуація перехоплюється блоком catch і обробляється. Уточнимо цей опис. Код, що підлягає контролю, повинен виконуватися усередині блоку try. (Функції, що викликаються усередині блоку try, також можуть генерувати виняткові ситуації.) Виняткові ситуації перехоплюються оператором catch, який слідує безпосередньо за блоком try.

Розмір блоку try може варіюватися. Він може містити як декілька операторів, так і цілу програму (в цьому випадку функція main() цілком поміщається в блок try). Виникла виняткова ситуація перехоплюється відповідним оператором catch, який виконує її обробку. З одним блоком try може бути зв'язане декілька операторів catch. Вибір правильного оператора catch визначається типом виняткової ситуації. З декількох варіантів вибирається оператор catch, тип аргументу якого співпадає з виниклою винятковою ситуацією (решта варіантів ігнорується). В процесі перехоплення виняткової ситуації аргументу привласнюється якесь значення. Аргумент може бути об'єктом вбудованого типу або класу. Якщо фрагмент не генерує ніяких виняткових ситуацій (тобто в блоці try помилки не виникають), не виконується жоден оператор catch.

Оператор throw генерує вказану виняткову ситуацію. Якщо в програмі передбачено її перехоплення, оператор throw повинен виконуватися або усередині блоку try, або усередині функції, явно або що неявно викликається усередині блоку try. Якщо генерується виняткова ситуація, для якої не передбачена обробка, програма може припинити своє виконання. В цьому випадку викликається стандартна функція terminate (), яка за умовчанням викликає функцію abort ().

# 2.ОБҐРУНТУВАННЯ ІЄРАРХІЇ КЛАСІВ C:\Users\PC\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Untitled-DiagramMMMMM.PNG

Вибір між агрегацією та успадкуванням не є складним, якщо знати просте правило: наслідування та агрегація висловлюють певний вид зв’язку між класами об’єктів. Зв’язок успадкування пояснюється англійським дієсловом «Is a», або українським «є». Тобто при успадкуванні одного класу іншим не змінюється суть класу, що успадковується. Навпаки, цей клас розширюється, уточнюється, набуває більш конкретних властивостей. Агрегація ж, у свою чергу, висловлюється дієсловом «Has a», або українським «містить», тобто клас, що агрегує інший клас, хоче натякнути, що по своїй суті він не є класом, що агрегував у собі, але він його містить як частину себе. Наприклад, якщо маємо клас Man та Brain, то підставляючи одне з двох дієслів, ми розуміємо, що чоловік не є мізками, але він має, або містить їх. У свою чергу, якщо ми маємо класи Human та Man, то встановиться зв’язок успадкування, бо чоловік є людиною, а не містить її у собі.

У таблиці нище описано завдання курсової роботи варіанта 1.

Таблиця 3.2. Ієрархія 1варіанта

|  |  |
| --- | --- |
| Варіант | Структура об’єкту  перелік класів |
|  | Дата {Рік, Місяць, День}  Персона {Прізвище, Ім’я, Дата (народження)};  Кельнер {Персона, Категорія};  Тип страви {перша, друга, десерт};  Страва {Назва, Ціна, Тип страви};  Замовлення {Список страв, Кельнер, Дата (замовлення)};  Денний баланс {Список замовлень страв}.  Середнє значення кількості страв в замовленні та середнє значення вартості замовлень. |

### 2.1.Успадкування

Дивлячись на завдання, видно, що між запропонованими класами можливо провести зв’язок «Is a». Клас Кельнер може успадкувати клас Персона, наділивши його новою характеристикою «Категорія».

### 2.2.Агрегація

Агрегація(композиція) використовується , коли до класів можна застосувати словосполучення Has a(має). В такому разі, один об’єкт є частиною іншого. Відповідно до 6 варіанту, я обрав агрегацію у класах Person(Особа має дату народження), Book(книга має дату отримання в бібліотеці), Student(студент має список книг), StudyGroup(Навчальна група має список студентів та список книг).

При чому, важливо розрізняти чисту агрегацію та композицію. Композиція – це вид агрегації, що говорить нам про те, що об’єкт агрегованого класу існує тільки всередині об’єкту класу, що агрегує. В даних випадках використовується композиція.

### 2.3 Вибір конкретних значень обмежень для полів різних класів

### Для кожного класу створений метод Set(), який дає змогу змінювати поля класів. Наприклад у класі Person цей метод дає змогу змінювати поля ім’я та прізвище. Але є такі ситуації коли користувач або випадково або навмисне ввів хибні дані які не віднсяться до призначення цього поля. Для обробки таких ситуацій були введені обмеження для введення даних до полів. Тому при введенні хибних даних таких як +. -, %, (, … або цифр користувач отримує повідомлення про помилку та пропозицію ввести дані знову. Також для класа Date є обмеження для вводу днів, місяців та років, таких як те що не можна вводити менше 1 та більше 31 днів і так далі.

### 2.4 Структура програми

Дата представляє із себе класData. Який містить в собі поля рік, місяць, день. Також він реалізує методи Show та Set. Даний клас перебуває у відношенні композиція з класом Person, Order.

Персона представляє собою клас Person, який містить в собі поля iм’я, прізвище та дата народження. Поля char\* мають властивості з обмеженням для вводу цифр, різних знаків та кількості вводу літер. Даний клас також реалізовує методи Show та Set.

Кельнер представляє собою клас Celner, який містить в собі поля категорія, дата народження(об’єкт класа Data) і також наслідується від класу Person, оскільки кожен Кельнер є персоною та містить такі ж властивості як ім’я та фамілія. Властивості обмежують вводу цифр, різних знаків та кількості вводу літер до поля «категорія». Даний клас також реалізовує методи Show та Set.

Тип страви представлений класом TypeOFDish. Тип страви містить поле типу(перша, друга, десерт) і він також реалізовує методи Show та Set. Поле char\* має властивость з обмеженням для вводу цифр, різних знаків та кількості вводу літер.

Страва представляє собою клас Dish. Даний клас містить в собі поля тип страви(об’єкт класа TypeOFDish), iм’я та ціну цієї страви. Тип страви перебуває у відношенні композиція з класом Страва. Також даний клас реалізує методи Show та Set.

Замовлення представляє собою клас Order, який містить в собі список страв(масив об’єктів класа Dish), Кельнера та дату замовлення (об’єкт класа Data). Дефолтний конструктор цього класа створює масив вказівників на об’єкти класу Dish випадкового розміру. Також для цього класа у мене перевантажена операція індексації яка повертає елемент масиву страв под переданому індексу. У цьому класі ще реалізовані методи CompareNum, CompareStr які порівнюють поля при пошуку та Show та Set.

Останнім у даній ієрархії є клас DailyBalance, який містить у собі список замовлень. У майбутньому ієрархія класів може дещо змінитися, оскільки написані в майбутньому інструменти для роботи з класами можуть вимагати певних нових можливостей від цих класів. Аналогічна ситуація відноситься і до властивостей полів, а особливо рівня їх доступу. На даний момент ієрархія побудована так, щоб максимально правдиво змоделювати ситуацію з життя.

# 3.ОБҐРУНТУВАННЯ ВИБОРУ

### **3.1.Вибір типу властивостей**

Майже скрізь було використано властивості із обмеженим Set, публічними Get та публічним Show. В деяких місцях було використанні автоматичні властивості, або властивості тільки для пошуку. За допомогою обмеження властивостей було досягнено заповнення о’бєктів коректною інформацією. Наприклад, обмеження цифр в даті.

### **3.2.Вибір перевантаження операцій**

Для перевантаження операцій я вирішив перевантажити оператор [ ] для об’єктів замовлення та денний баланс. При передачі індексу він не повинен бути менше нуля та більше розміру масиву у цьому класі. Також для перевантаження операцій я використав оператор << та >> для запису та зчитування з файлу відповідно.

### 3.3.Вибір виключних місць

При зчитуванні або записі в файл було використана обробка виключних ситуацій, на раз якщо файл з введеним ім’ям не буде можливо створити. Також обробка виключних ситуацій була реалізована при передаці у конструктори деяких параметрів. В деяких випадках ( при зчитуванні, наприклад, номера пункту меню, обраного користувачем, можна обійтись простим оператором switch. Таким чином можна обробляти лише задовільні дані, а незадовільні ігнорувати. Необхідною є обробка виключних ситуацій при роботі з файлами – тут ні від користувача, ні від програміста може нічого не залежати: нема необхідного файлу чи каталогу, відсутній доступ до диску і т.д. Такі місця обов’язково потрібно заключати в блоки try –catch.

# ВИСНОВКИ

Під час написання курсової роботи були досліджені можливості мови C++. Освоєно найважливіші парадигми об’єктно-орієнтованого програмування: інкапсуляція, наслідування, абстракція, поліморфізм. Було використано різноманітні конструкції мови С++, включаючи перевантаження операторів, обробку виключних ситуацій, шаблони типів, роботу з файлами та інше. Були набуті навики з побудови архітектури програмного забезпечення використовуючи засоби мови UML. Також було покращено вміння конструювати додатки, налагоджувати їх, вміння орієнтуватися у відносно великому для студента даного курсу об’ємі коду, краще ознайомлення із можливостями Visual Studio 2017 та інше.

В решті решт, програмний продукт функціонально відповідає вимогам завдання, тобто в програмі присутня можливість вводу даних, виведення у табличному вигляді, запис на диск, зчитування з диску (текстові та бінарні файли), пошук даних. Було оброблено максимальну кількість виключних ситуацій, перевантажено оператори для об’єктів створених класів та розроблено якомога інтуїтивно зрозуміле меню.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Роберт Лафоре — Объектно-ориентированное программирование в С++
2. Герберт Шилдт — C++ базовый курс
3. «Программирование. Принципы и практика с использованием C++», Бьерн Страуструп, 2011
4. <http://cppreference.com>
5. <http://cplusplus.com>
6. <http://en.wikipedia.org>

# ДОДАТОК 1. ТЕКСТ ПРОГРАМИ

**exeption.hpp:**

**#pragma once**

**//базовий класс виключень**

**class MyException {**

**const char \*message = nullptr;**

**public:**

**MyException(const char \*message = "Unknown exception") : message(message) {}**

**const char\* what() const {**

**return message;**

**}**

**};**

**//виключення: "Не вірний аргумент!"**

**struct Wrong\_Argument\_Exeption : MyException {**

**Wrong\_Argument\_Exeption(const char \*message = "Wrong argument exception") : MyException(message) {}**

**};**

**//виключення: "Вихід за межі!"**

**struct Out\_Of\_Range\_Exception : MyException {**

**Out\_Of\_Range\_Exception(const char \*message = "Out of range exception") : MyException(message) {}**

**};**

**//виключення: "Поганий файл!"**

**struct Bad\_File\_Exeption : MyException {**

**Bad\_File\_Exeption(const char \*message = "Bad file exception") : MyException(message) {}**

**};**

**//виключення: "Поганий файл для запису!"**

**struct Bad\_File\_Write\_Exeption : MyException {**

**Bad\_File\_Write\_Exeption(const char \*message = "Bad file write exception") : MyException(message) {}**

**};**

**//виключення: "Поганий файл для зчитування!"**

**struct Bad\_File\_Read\_Exeption : MyException {**

**Bad\_File\_Read\_Exeption(const char \*message = "Bad file read exception") : MyException(message) {}**

**};**

**Date.h:**

**#pragma once**

**#include <iostream>**

**#include <iomanip>**

**#include "exeption.hpp"**

**using namespace std;**

**class Date {/\*(Рік, Місяць, День)\*/**

**private:**

**int day,**

**month,**

**year;**

**public:**

**//конструктор за замовчуванням**

**Date() {**

**day = 1 + rand() % 31;**

**month = 1 + rand() % 12;**

**year = 1980 + rand() % (2017 - 1980);**

**}**

**//конструктор ініціалізації**

**Date(int a\_day, int a\_month, int a\_year) {**

**if (day < 0 || day > 31 || month < 0 || month > 12 || year > 2017 || year < 1980)**

**throw Wrong\_Argument\_Exeption();**

**this->day = a\_day;**

**this->month = a\_month;**

**this->year = a\_year;**

**}**

**//конструктор копіювання**

**Date(const Date &copy) {**

**day = copy.day;**

**month = copy.month;**

**year = copy.year;**

**}**

**//перевантажений оператор виведення в потік**

**friend ostream& operator << (ostream &os, const Date &o\_date) {**

**os << setw(10) << o\_date.day;**

**os << setw(3) << o\_date.month;**

**os << setw(5) << o\_date.year;**

**return os;**

**}**

**//перевантажений оператор введення з потоку**

**friend istream& operator >> (istream &is, Date &o\_date) {**

**is >> o\_date.day;**

**is >> o\_date.month;**

**is >> o\_date.year;**

**return is;**

**}**

**void Set(); //метод змінення полів**

**void Show();//метод який виводить на консоль таблицю**

**int getDay() {**

**return day;**

**}**

**int getMonth() {**

**return month;**

**}**

**int getYear() {**

**return year;**

**}**

**//деструктор**

**~Date() {}**

**};**

**//реалізація методу змінення полів**

**void Date::Set() {**

**bool flag = true;**

**cout << " Введите год: ";**

**while (flag) {**

**cin.clear();**

**rewind(stdin);**

**cin >> year;**

**if (!cin || year < 1980 || year > 2017 )**

**cout << " Неверное значение, введите снова: ";**

**else flag = false;**

**}**

**flag = true;**

**cout << " Введите месяц: ";**

**while (flag) {**

**cin.clear();**

**rewind(stdin);**

**cin >> month;**

**if (!cin || month < 1 || month > 12 || !cin.good())**

**cout << " Неверное значение, месяц должен быть от 1 до 12: ";**

**else flag = false;**

**}**

**flag = true;**

**cout << " Введите день: ";**

**if (month == 2 && year % 4 == 0) {**

**while (flag) {**

**cin.clear();**

**rewind(stdin);**

**cin >> day;**

**if (!cin || day < 1 || day > 29)**

**cout << " Неверное значение, в этом месяце 29 дней: ";**

**else flag = false;**

**}**

**}**

**else if (month == 2 && year % 2 == 1) {**

**while (flag) {**

**cin.clear();**

**rewind(stdin);**

**cin >> day;**

**if (!cin || day < 1 || day > 28)**

**cout << " Неверное значение, в этом месяце 28 дней: ";**

**else flag = false;**

**}**

**}**

**if (month % 2 == 0 && month != 2) {**

**while (flag) {**

**cin.clear();**

**rewind(stdin);**

**cin >> day;**

**if (!cin || day < 1 || day > 30)**

**cout << " Неверное значение, в этом месяце 30 дней: ";**

**else flag = false;**

**}**

**}**

**else {**

**while (flag) {**

**cin.clear();**

**rewind(stdin);**

**cin >> day;**

**if (!cin || day < 1 || day > 31) {**

**cout << " Неверное значение, в месяце 30 дней: ";**

**}**

**else flag = false;**

**}**

**}**

**}**

**//реалізація виводу таблички на консоль**

**void Date::Show() {**

**cout << " " << setw(2) << setfill('0') << day << "." << setw(2) << setfill('0') << month << "." << setw(4) << year << "|";**

**}**

**Person.h:**

**#pragma once**

**#pragma warning(disable : 4996)**

**#include "Date.h"**

**#include <iostream>**

**using namespace std;**

**//масив імен за замовчуванням**

**char \*d\_names[] = { "Алексей", "Денис", "Юрий", "Иван", "Илья", "Андрей", "Богдан", "Владислав", "Анатолий", "Александр", "Максим", "Евгений", "Виктор", "Павел" };**

**//масив прізвищ за замовчуванням**

**char \*d\_surnames[] = { "Гарник", "Трухан", "Данько", "Засека", "Артамонов", "Мартыненко", "Скороход", "Юрченко", "Бочок", "Шматко", "Бевза", "Войтович", "Колотуша", "Ташу" };**

**class Person {/\*(Прізвище, Ім’я, Дата (народження))\*/**

**private:**

**Date birthday\_date;**

**char \*name,**

**\*surname;**

**public:**

**//конструктор зазамовчуванням**

**Person() : birthday\_date() {**

**int i = rand() % 14;**

**name = new char[strlen(d\_names[i]) + 1];**

**strcpy(name, d\_names[i]);**

**i = rand() % 14;**

**surname = new char[strlen(d\_surnames[i]) + 1];**

**strcpy(surname, d\_surnames[i]);**

**}**

**//конструктор ініціалізації**

**Person(int a\_day, int a\_month, int a\_year, char \*a\_name, char \*a\_surname) : birthday\_date(a\_day, a\_month, a\_year) { //конструктор ініціалізації**

**this->name = new char[strlen(a\_name) + 1];**

**strcpy(this->name, a\_name);**

**this->surname = new char[strlen(a\_surname) + 1];**

**strcpy(this->surname, a\_surname);**

**}**

**//конструктор копіювання**

**Person(const Person &copy) {**

**this->birthday\_date = copy.birthday\_date;**

**name = new char[strlen(copy.name) + 1];**

**strcpy(name, copy.name);**

**surname = new char[strlen(copy.surname) + 1];**

**strcpy(surname, copy.surname);**

**}**

**//перевантажений оператор виведення в потік**

**friend ostream& operator << (ostream &os, const Person &o\_person) {**

**os << setw(15) << o\_person.name << endl;**

**os << setw(15) << o\_person.surname << endl;**

**return os;**

**}**

**//перевантажений оператор введення з потоку**

**friend istream& operator >> (istream &is, Person &o\_person) {**

**is >> o\_person.name;**

**is >> o\_person.surname;**

**return is;**

**}**

**virtual void Set();//метод змінення полів**

**virtual void Show();//метод який виводить на консоль таблицю**

**char \*getName() {**

**return name;**

**}**

**char\* getSurname() {**

**return surname;**

**}**

**Date getBDate() {**

**return birthday\_date;**

**}**

**//віртуальний деструктор**

**virtual ~Person() {**

**if (name)**

**delete[] name;**

**if (surname)**

**delete[] surname;**

**}**

**};**

**//реалізація методу змінення полів**

**void Person::Set() {**

**int i;**

**bool flg = true;**

**name = new char[30];**

**cout << " Введите имя: ";**

**while (flg) {**

**rewind(stdin);**

**cin.clear();**

**cin >> name;**

**for (i = 0; name[i]; i++)**

**if (name[i] >= '!' && name[i] <= '@' || name[i] >= '[' && name[i] <= '`' || name[i] == '~' || name[i] == ',' || name[i] == ' ')**

**break;**

**if (name[i])**

**cout << " Введены неверные символы! Введите имя еще раз: ";**

**else flg = false;**

**}**

**flg = true;**

**surname = new char[30];**

**cout << " Введите фамилию: ";**

**while (flg) {**

**rewind(stdin);**

**cin.clear();**

**cin >> surname;**

**for (i = 0; surname[i]; i++)**

**if (surname[i] >= '!' && surname[i] <= '@' || surname[i] >= '[' && surname[i] <= '`' || surname[i] == '~' || surname[i] == ',' || name[i] == ' ')**

**break;**

**if (surname[i])**

**cout << " Введены неверные символы! Введите фамилию еще раз: ";**

**else flg = false;**

**}**

**cout << " Введите дату рождения" << endl;**

**birthday\_date.Set();**

**}**

**//реалізація виводу таблички на консоль**

**void Person::Show() {**

**cout << setw(10) << setfill(' ') << name << " ";**

**cout << setw(10) << surname << "|";**

**birthday\_date.Show();**

**}**

**Celner.h:**

**#pragma once**

**#include "Person.h"**

**#include <iostream>**

**using namespace std;**

**//масив категорій кельнерів**

**char \*d\_category[] = { "Первая", "Вторая", "Третья", "Четвертая", "Пятая" };**

**class Celner : public Person {/\*(Персона, Категорія)\*/**

**private:**

**char \*category;**

**public:**

**//конструктор за замовчуванням**

**Celner() : Person() {**

**int i = rand() % 5;**

**category = new char[strlen(d\_category[i]) + 1];**

**strcpy(category, d\_category[i]);**

**}**

**//конструктор ініціалізації**

**Celner(int a\_day, int a\_month, int a\_year, char \*a\_name, char \*a\_surname, char \*a\_category) : Person(a\_day, a\_month, a\_year, a\_name, a\_surname) { //конструктор ініціалізації**

**this->category = new char[strlen(a\_category) + 1];**

**strcpy(this->category, a\_category);**

**}**

**//конструктор копіювання**

**Celner(const Celner &copy) : Person(copy) {**

**category = new char[strlen(copy.category) + 1];**

**strcpy(category, copy.category);**

**}**

**//перевантажений оператор виведення в потік**

**friend ostream& operator << (ostream &os, Celner &o\_celner) {**

**os << o\_celner.getName();**

**os << o\_celner.getSurname();**

**os << o\_celner.getBDate();**

**os << o\_celner.category;**

**return os;**

**}**

**//перевантажений оператор введення з потоку**

**friend istream& operator >> (istream &os, Celner &o\_celner) {**

**os >> o\_celner.getName();**

**os >> o\_celner.getSurname();**

**os >> o\_celner.getBDate();**

**os >> o\_celner.category;**

**return os;**

**}**

**void Set();//метод змінення полів**

**void Show();//метод який виводить на консоль таблицю**

**char \* getCategory() {**

**return category;**

**}**

**//деструктор**

**~Celner() {**

**if (category)**

**delete[] category;**

**}**

**};**

**//реалізація методу змінення полів**

**void Celner::Set() {**

**cout << " Введите данные о кельнере" << endl;**

**Person::Set();**

**int i;**

**bool flg = true;**

**category = new char[30];**

**cout << " Введите категорию кельнера: ";**

**while (flg) {**

**rewind(stdin);**

**cin.clear();**

**cin >> category;**

**for (i = 0; category[i]; i++)**

**if (category[i] >= '!' && category[i] <= '@' || category[i] >= '[' && category[i] <= '`' || category[i] == '~' || category[i] == ',')**

**break;**

**if (category[i])**

**cout << " Введены неверные символы! Введите категорию еще раз: ";**

**else flg = false;**

**}**

**}**

**//реалізація виводу таблички на консоль**

**void Celner::Show() {**

**Person::Show();**

**cout << setw(14) << setfill(' ') << category << "|";**

**}**

**Type\_Of\_Dish.h:**

**#pragma once**

**#include <iostream>**

**char \*d\_type[] = { "Первое", "Второе", "Десерт" };**

**using namespace std;**

**class TypeOfDish {/\*(перша, друга, десерт)\*/**

**private:**

**char \*type;**

**public:**

**//конструктор за замовчуванням**

**TypeOfDish() {**

**int i = rand() % 3;**

**type = new char[strlen(d\_type[i]) + 1];**

**strcpy(type, d\_type[i]);**

**}**

**//конструктор ініціалізації**

**TypeOfDish(char \*a\_type) {**

**this->type = new char[strlen(a\_type) + 1];**

**strcpy(this->type, a\_type);**

**}**

**//конструктор копіювання**

**TypeOfDish(const TypeOfDish &copy) {**

**type = copy.type;**

**}**

**//перевантажений оператор виведення в потік**

**friend ostream& operator << (ostream &os, const TypeOfDish &o\_type\_of\_dish) {**

**os << setw(15) << o\_type\_of\_dish.type;**

**return os;**

**}**

**//перевантажений оператор введення з потоку**

**friend istream& operator >> (istream &is, TypeOfDish &o\_type\_of\_dish) {**

**is >> o\_type\_of\_dish.type;**

**return is;**

**}**

**void Set();//метод змінення полів**

**void Show();//метод який виводить на консоль таблицю**

**char \*getType() {**

**return type;**

**}**

**//деструктор**

**~TypeOfDish() {**

**if (type)**

**delete[] type;**

**}**

**};**

**//реалізація методу змінення полів**

**void TypeOfDish::Set() {**

**bool flg = true;**

**type = new char[10];**

**cout << " Выберите тип блюда" << endl;**

**cout << "\n 1. Первое"<< endl;**

**cout << "\n 2. Второе" << endl;**

**cout << "\n 3. Десерт" << endl;**

**int a;**

**cout << "\n Ваш выбор: ";**

**while (flg == true) {**

**rewind(stdin);**

**cin.clear();**

**cin >> a;**

**switch (a) {**

**case 1:**

**type = d\_type[0];**

**flg = false;**

**break;**

**case 2:**

**type = d\_type[1];**

**flg = false;**

**break;**

**case 3:**

**type = d\_type[2];**

**flg = false;**

**break;**

**default:**

**cout << "Такого типа блюда нету. Введите еще раз." << endl;**

**}**

**}**

**}**

**//реалізація виводу таблички на консоль**

**void TypeOfDish::Show() {**

**cout << setw(9) << type << "|";**

**}**

**Dish.h:**

**#pragma once**

**#include "Type\_of\_dish.h"**

**#include <iostream>**

**using namespace std;**

**//масив назв страв за замовчуванням**

**char \*d\_nameOfDish[] = { "Бендерики", "Бифштекс", "Борщ", "Винегрет", "Гриль", "Драники", "Жаркое", "Жульен", "Запеканка", "Зразы", "Капустняк", "Кебаб", "Лагман", "Лазанья", "Лапша", "Мафины", "Налистники", "Оливье", "Паштет", "Харчо", "Хачапури", "Шаурма", "Шашлык", "Яичница" };**

**class Dish {/\*(Назва, Ціна, Тип страви)\*/**

**private:**

**TypeOfDish type\_of\_dish;**

**char \*nameOfDish;**

**int price;**

**public:**

**//конструктор за замовчуванням**

**Dish() : type\_of\_dish() {**

**int i = rand() % 24;**

**nameOfDish = new char[strlen(d\_nameOfDish[i]) + 1];**

**strcpy(nameOfDish, d\_nameOfDish[i]);**

**price = rand() % 10001;**

**}**

**//конструктор ініціалізації**

**Dish(char \*a\_type, char \*a\_nameOfDish, int a\_price) : type\_of\_dish(a\_type){**

**if (a\_price < 0)**

**throw Wrong\_Argument\_Exeption();**

**this->nameOfDish = a\_nameOfDish;**

**this->price = a\_price;**

**}**

**//конструктор копіювання**

**Dish(const Dish &copy) : type\_of\_dish(copy.type\_of\_dish) {**

**nameOfDish = new char[strlen(copy.nameOfDish) + 1];**

**strcpy(nameOfDish, copy.nameOfDish);**

**price = copy.price;**

**}**

**//перевантажений оператор виведення в потік**

**friend ostream& operator << (ostream &os, const Dish &o\_dish) {**

**os << setw(15) << o\_dish.type\_of\_dish;**

**os << setw(15) << o\_dish.nameOfDish;**

**os << setw(15) << o\_dish.price;**

**return os;**

**}**

**//перевантажений оператор введення з потоку**

**friend istream& operator >> (istream &is, Dish &o\_dish) {**

**is >> o\_dish.type\_of\_dish;**

**is >> o\_dish.nameOfDish;**

**is >> o\_dish.price;**

**return is;**

**}**

**void Set();//метод змінення полів**

**void Show();//метод який виводить на консоль таблицю**

**int getPrice() {**

**return price;**

**}**

**char \*getNameOfDish() {**

**return nameOfDish;**

**}**

**char \*getTypeOfDish() {**

**return type\_of\_dish.getType();**

**}**

**~Dish() {**

**if (nameOfDish)**

**delete[] nameOfDish;**

**}**

**};**

**//реалізація методу змінення полів**

**void Dish::Set() {**

**int i;**

**bool flg = true;**

**nameOfDish = new char[30];**

**cout << " Введите название блюда: ";**

**while (flg) {**

**rewind(stdin);**

**cin.clear();**

**cin >> nameOfDish;**

**for (i = 0; nameOfDish[i]; i++)**

**if (nameOfDish[i] >= '!' && nameOfDish[i] <= '@' || nameOfDish[i] >= '[' && nameOfDish[i] <= '`' || nameOfDish[i] == '~' || nameOfDish[i] == ',' || nameOfDish[i] == ' ')**

**break;**

**if (nameOfDish[i])**

**cout << " Введены неверные символы! Введите название еще раз: ";**

**else flg = false;**

**}**

**type\_of\_dish.Set();**

**flg = true;**

**cout << " Введите цену блюда: ";**

**while (flg) {**

**cin.clear();**

**rewind(stdin);**

**cin >> price;**

**if (!cin || price < 0 || price > 10000)**

**cout << " Неверное значение, введите снова: ";**

**else flg = false;**

**}**

**}**

**//реалізація виводу таблички на консоль**

**void Dish::Show() {**

**cout << setw(11) << nameOfDish << "|";**

**type\_of\_dish.Show();**

**cout << setw(9) << price << "|" << endl;**

**}**

**Order.h:**

**#pragma once**

**#include "Celner.h"**

**#include "Dish.h"**

**#include "exeption.hpp"**

**#include <iostream>**

**using namespace std;**

**class Order : public Celner{/\*(Список страв, Кельнер, Дата (замовлення))\*/**

**private:**

**Dish \*\*listOfDishes;**

**Date order\_date;**

**int size;**

**public:**

**//конструктор за замовчуванням**

**Order() : Celner(), order\_date() {**

**size = 1 + rand() % 5;**

**listOfDishes = new Dish\*[size];**

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

**listOfDishes[i] = new Dish;**

**}**

**//конструктор за замовчуванням в якому є можливість задати кількість страв**

**Order(int n) : Celner(), order\_date() {**

**if (n < 0)**

**throw Wrong\_Argument\_Exeption();**

**size = n;**

**listOfDishes = new Dish\*[size];**

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

**listOfDishes[i] = new Dish;**

**}**

**//конструктор ініціалізації**

**Order(Dish \*\*a\_listOfDishes, int count\_of\_dishes, int year\_of\_order, int month\_of\_order, int day\_of\_order, int day\_of\_celner, int month\_of\_celner, int year\_of\_celner, char \*celner\_name, char \*celner\_surname, char \*celner\_category) :**

**Celner(day\_of\_celner, month\_of\_celner, year\_of\_celner, celner\_name, celner\_surname, celner\_category), order\_date(year\_of\_order, month\_of\_order, day\_of\_order) {**

**this->size = count\_of\_dishes;**

**listOfDishes = new Dish\*[size];**

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

**listOfDishes[i] = new Dish(\*a\_listOfDishes[i]);**

**}**

**//конструктор копіювання**

**Order(const Order &copy) {**

**size = copy.size;**

**listOfDishes = new Dish\*[size];**

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

**listOfDishes[i] = new Dish(\*copy.listOfDishes[i]);**

**}**

**//перевантажений оператор індексації**

**Dish &operator [](int index) {**

**if (index < 0 || index >= size)**

**throw Out\_Of\_Range\_Exception("Ошибка! Обращение к несуществующему элементу массива.");**

**else**

**return \*listOfDishes[index];**

**}**

**//перевантажений оператор виведення в потік**

**friend ostream& operator << (ostream &os, Order &o\_order) {**

**os << setw(15) << o\_order.order\_date;**

**os << setw(15) << o\_order.getName();**

**os << setw(15) << o\_order.getSurname();**

**os << o\_order.getBDate();**

**os << setw(15) << o\_order.getCategory();**

**for (int i = 0; i < o\_order.size; i++)**

**os << \*o\_order.listOfDishes[i] << endl;**

**return os;**

**}**

**//перевантажений оператор введення з потоку**

**friend istream& operator >> (istream &is, Order &o\_order) {**

**is >> o\_order.order\_date;**

**is >> o\_order.getName();**

**is >> o\_order.getSurname();**

**is >> o\_order.getCategory();**

**is >> o\_order.getCategory();**

**for (int i = 0; i < o\_order.size; i++)**

**is >> \*o\_order.listOfDishes[i];**

**return is;**

**}**

**void Set();//метод змінення полів**

**void Show();//метод який виводить на консоль таблицю**

**bool CompareNum(int);//метод порівнювання чисел який створений для пошуку по числовому полю**

**void CompareStr(char\*);//метод порівнювання символьних полів який створений для пошуку по символьному полю**

**Dish \*\*getArr() {**

**return listOfDishes;**

**}**

**int getSize() {**

**return size;**

**}**

**//деструктор**

**~Order() {**

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

**delete[] listOfDishes[i];**

**}**

**};**

**//реалізація методу змінення полів**

**void Order::Set() {**

**Celner::Set();**

**cout << " Введите дату заказа" << endl;**

**order\_date.Set();**

**cout << " Введите количество блюд: ";**

**bool flg = true;**

**while (flg) {**

**cin.clear();**

**rewind(stdin);**

**cin >> size;**

**if (!cin || size <= 0)**

**cout << " Ошибка, введите снова: ";**

**else flg = false;**

**}**

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

**cout << "\n Блюдо №" << i + 1 << endl;**

**listOfDishes[i]->Set();**

**}**

**}**

**//реалізація виводу таблички на консоль**

**void Order::Show() {**

**order\_date.Show();**

**Celner::Show();**

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

**if (i > 0)**

**cout << "|" << setw(86) << i + 1 << "|";**

**else**

**cout << setw(9) << i + 1 << "|";**

**listOfDishes[i]->Show();**

**}**

**}**

**//реалізація методу порівнювання чисел**

**bool Order::CompareNum(int tmp) {**

**if (tmp == order\_date.getDay() || tmp == order\_date.getMonth() || tmp == order\_date.getYear()) {**

**return true;**

**}**

**if (tmp == Celner::getBDate().getDay() || tmp == Celner::getBDate().getMonth() || tmp == Celner::getBDate().getYear()) {**

**Celner::Show();**

**}**

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

**if (tmp == listOfDishes[i]->getPrice()) {**

**listOfDishes[i]->Show();**

**}**

**}**

**}**

**//реалізація методу порівнювання символьних полів**

**void Order::CompareStr(char\* tmp) {**

**if (strcmp(tmp, Celner::getName()) == 0 || strcmp(tmp, Celner::getSurname()) == 0 || strcmp(tmp, Celner::getCategory()) == 0) {**

**Celner::Show();**

**cout << endl;**

**}**

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

**if (strcmp(tmp, listOfDishes[i]->getNameOfDish()) == 0 || strcmp(tmp, listOfDishes[i]->getTypeOfDish()) == 0)**

**listOfDishes[i]->Show();**

**}**

**}**

**Daily\_Balance.h:**

**#pragma once**

**#include "Order.h"**

**#include "exeption.hpp"**

**#include <iostream>**

**#include <iomanip>**

**using namespace std;**

**class DailyBalance {/\*(Список замовлень страв)\*/**

**private:**

**Order \*\*ordList;**

**int size;**

**public:**

**//конструктор за замовчуванням**

**DailyBalance() {**

**size = 1 + rand() % 5;**

**ordList = new Order\*[size];**

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

**ordList[i] = new Order();**

**}**

**//конструктор за замовчуванням в якому є можливість задати кількість замовлень**

**DailyBalance(int n) {**

**if (n < 0)**

**throw Wrong\_Argument\_Exeption();**

**size = n;**

**ordList = new Order\*[size];**

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

**ordList[i] = new Order;**

**}**

**//конструктор за замовчуванням в якому є можливість задати кількість замовлень та страв у цих замовленнях**

**DailyBalance(int n, int n1) {**

**if (n < 0 || n1 < 0)**

**throw Wrong\_Argument\_Exeption();**

**size = n;**

**ordList = new Order\*[size];**

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

**ordList[i] = new Order(n1);**

**}**

**//конструктор ініціалізації**

**DailyBalance(int count\_of\_orders, Dish \*\*a\_listOfDishes, int count\_of\_dishes, int year\_of\_order, int month\_of\_order, int day\_of\_order, int day\_of\_celner, int month\_of\_celner, int year\_of\_celner, char \*celner\_name, char \*celner\_surname, char \*celner\_category) {**

**this->size = count\_of\_orders;**

**ordList = new Order\*[size];**

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

**ordList[i] = new Order(a\_listOfDishes, count\_of\_dishes, year\_of\_order, month\_of\_order, day\_of\_order, day\_of\_celner, month\_of\_celner, year\_of\_celner, celner\_name, celner\_surname, celner\_category);**

**}**

**//конструктор копіювання**

**DailyBalance(const DailyBalance &copy) {**

**size = copy.size;**

**ordList = new Order\*[size];**

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

**ordList[i] = new Order(\*copy.ordList[i]);**

**}**

**//перевантажений оператор індексації**

**Order &operator [](int index) {**

**if (index < 0 || index >= size)**

**throw Out\_Of\_Range\_Exception("Ошибка! Обращение к несуществующему элементу массива.");**

**else**

**return \*ordList[index];**

**}**

**//перевантажений оператор виведення в потік**

**friend ostream& operator << (ostream &os, const DailyBalance &o\_daily\_balance) {**

**for (int i = 0; i < o\_daily\_balance.size; i++)**

**os << \*o\_daily\_balance.ordList[i] << endl;**

**return os;**

**}**

**//перевантажений оператор введення з потоку**

**friend istream& operator >> (istream &is, DailyBalance &o\_daily\_balance) {**

**for (int i = 0; i < o\_daily\_balance.size; i++)**

**is >> \*o\_daily\_balance.ordList[i];**

**return is;**

**}**

**void Set();//метод змінення полів**

**void Show();//метод який виводить на консоль таблицю**

**Order \*\*getArr() {**

**return ordList;**

**}**

**int getSize() {**

**return size;**

**}**

**//деструктор**

**~DailyBalance() {**

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

**delete[] ordList[i];**

**}**

**};**

**//реалізація методу змінення полів**

**void DailyBalance::Set() {**

**cout << "Введите количество заказов: ";**

**bool flg = true;**

**while (flg) {**

**cin.clear();**

**rewind(stdin);**

**cin >> size;**

**if (!cin || size <= 0)**

**cout << "Ошибка, введите снова: ";**

**else flg = false;**

**}**

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

**cout << "\nЗаказ №" << i + 1 << endl;**

**ordList[i]->Set();**

**}**

**}**

**//реалізація виводу таблички на консоль**

**void DailyBalance::Show() {**

**cout << "------------------------------------------------------------------------------------------------------------------------" << endl;**

**cout << "|" << setw(10) << "№ Заказа|" << setw(15) << "Дата заказа|" << setw(30) << "Кельнер" << setw(22) << "|";**

**cout << setw(23) << "Блюда" << setw(19) << "|" << endl;**

**cout << "------------------------------------------------------------------------------------------------------------------------" << endl;**

**cout << "|" << setw(10) << "-|" << setw(15) << "-|" << setw(22) << "Имя Фамилия|" << setw(15) << "Дата рождения|" << setw(15) << "Категория|";**

**cout << setw(10) << "№ Блюда|" << setw(12) << "Название|" << setw(10) << "Тип|" << setw(10) << "Цена(грн)|" << endl;**

**cout << "------------------------------------------------------------------------------------------------------------------------" << endl;**

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

**cout << "|" << setw(9) << i + 1 << "|";**

**ordList[i]->Show();**

**cout << "------------------------------------------------------------------------------------------------------------------------" << endl;**

**}**

**}**

**main.cpp:**

**#include <locale.h>**

**#include <conio.h>**

**#include <iostream>**

**#include "Daily\_balance.h"**

**#include <fstream>**

**#include <string>**

**#include <windows.h>**

**#include "exeption.hpp"**

**using namespace std;**

**//у звіті обгрунтовано чому наступні дві функції не є шаблонними**

**//функція знаходження статистичних даних(середня кількість став у замовленні)**

**void Average\_Counts\_Of\_Dishes(DailyBalance &db) {**

**float sum = 0;**

**for (int i = 0; i < db.getSize(); i++)**

**sum += db[i].getSize();**

**sum /= db.getSize();**

**cout << "Среднее значение количества блюд в заказе: " << sum << endl;**

**}**

**//функція знаходження статистичних даних(середня вартість замовлень)**

**void Average\_Price(DailyBalance &db) {**

**float sum = 0;**

**for (int i = 0; i < db.getSize(); i++)**

**for (int j = 0; j < db[i].getSize(); j++)**

**sum += db[i][j].getPrice();**

**sum /= db.getSize();**

**cout << "Среднее значение стоимости заказов: " << sum << " грн." << endl;**

**}**

**//функція пошуку за числовим та текстовим полем**

**void Find(DailyBalance &db) {**

**int n;**

**int num\_tmp;**

**char \*str\_tmp = new char[30];**

**while (true) {**

**cout << "\n\t1. Поиск по числовому полю" << endl;**

**cout << "\t2. Поиск по строковому полю" << endl;**

**cout << "\t3. Выход в меню" << endl;**

**cout << "\nВаш выбор: ";**

**rewind(stdin);**

**cin.clear();**

**cin >> n;**

**switch (n) {**

**case 1:**

**cout << "\n Введите число: ";**

**cin >> num\_tmp;**

**for (int i = 0; i < db.getSize(); i++) {**

**if (db[i].CompareNum(num\_tmp) == true) {**

**db[i].Show();**

**}**

**}**

**break;**

**case 2:**

**cout << "\n Введите слово: ";**

**cin >> str\_tmp;**

**for (int i = 0; i < db.getSize(); i++)**

**db[i].CompareStr(str\_tmp);**

**break;**

**case 3:**

**return;**

**break;**

**default:**

**cout << "\nТакого пункта нет в меню. Введите еще раз." << endl;**

**break;**

**}**

**}**

**delete[] str\_tmp;**

**}**

**template<typename T>//шаблонна функція запису до текстового файлу масиву обєктів**

**void Text\_Write(T \*\*arr, int n) {**

**ofstream Ft("Text.txt");**

**if (!Ft.is\_open())**

**throw Bad\_File\_Write\_Exeption("Ошибка! Текстовый файл не был открыт для записи!");**

**else {**

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

**Ft << \*arr[i] << endl;**

**cout << "\nТекстовый файл успешно создан!" << endl;**

**}**

**Ft.close();**

**}**

**template<typename T>//шаблонна функція зчитування з текстового файлу масиву обєктів до іншого массиву обєктів**

**void Text\_Read(T \*\*arr, int n) {**

**ifstream Ft("Text.txt");**

**if (!Ft.is\_open())**

**throw Bad\_File\_Read\_Exeption("Ошибка! Текстовый файл не был открыт для считывания!");**

**else {**

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

**Ft >> \*arr[i];**

**arr[i]->Show();**

**}**

**cout << "\nТекстовый файл успешно считан!" << endl;**

**}**

**Ft.close();**

**}**

**template<typename T>//шаблонна функція запису до бінарного файлу масиву обєктів**

**void Binary\_Write(T \*\*arr, int n) {**

**ofstream Fb("Binary.dat", ios::binary);**

**if (!Fb.is\_open())**

**throw Bad\_File\_Write\_Exeption("Ошибка! Бинарный файл не был открыт для записи!");**

**else {**

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

**Fb.write((char\*)&arr[i], sizeof(&arr[i]));**

**cout << "\nБинарный файл успешно создан!" << endl;**

**}**

**Fb.close();**

**}**

**template<typename T>//шаблонна функція зчитування з бінарного файлу масиву обєктів до іншого массиву обєктів**

**void Binary\_Read(T \*\*arr, int n) {**

**ifstream Fb("Binary.dat", ios::binary);**

**if (!Fb.is\_open())**

**throw Bad\_File\_Read\_Exeption("Ошибка! Бинарный файл не был открыт для считывания!");**

**else {**

**cout << "\nБинарный файл успешно считан!" << endl;**

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

**Fb.read((char\*)&arr[i], sizeof(T));**

**arr[i]->Show();**

**}**

**}**

**Fb.close();**

**}**

**void Menu() {**

**int n = 0;**

**DailyBalance db(10, 10);**

**DailyBalance db1(10, 10);**

**DailyBalance db2(10, 10);**

**while (true){**

**cout << "\nМеню" << endl;**

**cout << "\t1. Ввести данные" << endl;**

**cout << "\t2. Вывести данные в таблице" << endl;**

**cout << "\t3. Запись данных на диск" << endl;**

**cout << "\t4. Считывание данных" << endl;**

**cout << "\t5. Поиск" << endl;**

**cout << "\t6. Выход" << endl;**

**cout << "\nВаш выбор: ";**

**cin >> n;**

**switch (n){**

**case 1:**

**db.Set();**

**break;**

**case 2:**

**db.Show();**

**Average\_Counts\_Of\_Dishes(db);**

**Average\_Price(db);**

**break;**

**case 3:**

**Binary\_Write(db.getArr(), db.getSize());**

**Text\_Write(db.getArr(), db.getSize());**

**break;**

**case 4:**

**Binary\_Read(db1.getArr(), db1.getSize());**

**Text\_Read(db2.getArr(), db2.getSize());**

**break;**

**case 5:**

**Find(db);**

**break;**

**case 6:**

**exit(0);**

**break;**

**default:**

**cout << "\nТакого пункта нет в меню. Введите еще раз." << endl;**

**break;**

**}**

**}**

**}**

**int main() {**

**srand(time(NULL));**

**setlocale(LC\_ALL, "RUS");**

**SetConsoleCP(1251);**

**SetConsoleOutputCP(1251);**

**try {**

**Menu();**

**}**

**catch (MyException &ex) { //контролюючий блок який відловлює всі виключення(структури) які наслідуються від Базового классу виключень**

**cout << ex.what(); //показує яке саме виключення було схоплене**

**}**

**\_getch();**

**return 0;**

**}**

ДОДАТОК 2. ОПИС ПРОГРАМИ

Програмне меню пропонує користувачеві 6 пунктів, серед яких введення даних, виведення даних у табличному вигляді, запис даних до файлів та зчитування даних з файлів, пошук даних, вихід. Всередині відповідних розділів пропонується робота з масивами конкретного типу даних, тобто зчитування або пошук за назвою і або за розміром.





