clМІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**КУРСОВА РОБОТА**

з дисципліни «Об’єктно-орієнтоване програмування»

Варіант №--

|  |  |  |
| --- | --- | --- |
| **Керівник :** |  | **Виконав:** |
| Допущено до захисту |  |
| І\_\_\_І \_\_\_\_\_\_\_\_\_ 2023 \_\_\_\_\_\_\_\_\_\_\_  підпис |  |
| Захищено з оцінкою  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_  оцінка підпис  І\_\_\_І \_\_\_\_\_\_\_\_\_ 2023 |  |

Київ-2023

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Факультет |  | | | | |
|  | (назва факультету, інституту) | | | | |
| Кафедра |  | | | | |
|  | (назва кафедри) | | | | |
| Дисципліна |  | | | | |
|  | (назва) | | | | |
| Курс | ***2*** | Група |  | Семестр | ***4*** |

**ЗАВДАННЯ**

**на курсовий проект (роботу) студента**

|  |
| --- |
|  |
| (прізвище, ім’я, по батькові) |

.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. Тема роботи: | | ***Розробка програмного забезпечення з використанням*** | | | |
| ***об’єктно- орієнтованого підходу.*** | | | | | |
|  | | | | | |
|  | |  | | | |
| 2. Строк здачі студентом закінченого проекту (роботи) | | | | | ***19.05.2023*** |
|  | |  | | | |
| 3. Вихідні дані до проекту (роботи): | | | | ***Варіант №--*** | |
|  | | | | | |
|  | | | | | |
|  | |  | | | |
| 4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають | | | | | |
| розробці): | ***1. Визначення класів та програмування меню користувача*** | | | | |
| ***(ієрархія класів програми: абстрактний клас –Завдання в системі управління*** | | | | | |
| ***проектом з розробки програмного забезпечення, класи-нащадки - дефект,*** | | | | | |
| ***задача для розробки). 2. Створення об’єктів та використання контейнерів*** | | | | | |
| ***(тип контейнера – черга з пріоритетом). 3. Організація роботи з даними*** | | | | | |
| ***через файл. 4. Пошук даних у контейнері (запит для пошуку – підрахувати*** | | | | | |
| ***загальну кількість дефектів, знайдених в заданому компоненті*** | | | | | |
| ***програмного забезпечення).*** | | | | | |
|  | |  | | | |
| 5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): | | | | | |
| ***діаграми класів, послідовності та об’єктів в нотації UML*** | | | | | |
|  | | | | | |
|  | |  | | | |
| 6. Дата видачі завдання: | | | ***15.02.2023*** | | |

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| **№п/п** | **Назва етапів курсового проекту (роботи) та питань, які мають бути розроблені відповідно до завдання** | **Термін виконання етапу** | **Позначки керівника  про виконання  завдань** |
| 1. | Отримання завдання на курсову роботу | 15.02.2023 |  |
| 2. | Огляд технічної літератури за темою роботи | 24.02.2023 |  |
| 3. | Розробка першої частини курсової роботи (визначення класів та програмування меню користувача) | 10.03.2023 |  |
| 4. | Перший контроль за процесом виконання курсової роботи, консультація у викладача | 17.03.2023 |  |
| 5. | Розробка другої частини курсової роботи (створення об’єктів та використання контейнерів) | 31.03.2023 |  |
| 6. | Розробка третьої частини курсової роботи (робота з даними через файл) | 14.04.2023 |  |
| 7. | Другий контроль за процесом виконання курсової роботи, консультація у викладача | 21.04.2023 |  |
| 8. | Розробка четвертої частини курсової роботи (пошук даних у контейнері) | 05.05.2023 |  |
| 9. | Оформлення пояснювальної записки | 19.05.2023 |  |
| 10. | Захист курсової роботи | 26.05.2023 |  |

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(підпис)

**Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(підпис) (прізвище ім'я, по батькові)

**«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 р.**

**Анотація**

Розробка програмного забезпечення з використанням об’єктно-орієнтованого підходу.

Курсова робота з дисципліни «Об’єктно-орієнтоване програмування» присвячена питанню створення ієрархії класів, застосування спадкування та поліморфізму, розробки контейнера для зберігання даних множини об’єктів, які створюються користувачем. У курсовій роботі було виконано визначення класів (ієрархія класів програми: базовий клас – завдання в системі управління проектом з розробки програмного забезпечення, класи-нащадки – дефект, задача для розробки), розроблено меню користувача, створено множину об’єктів та розроблено контейнер для її зберігання (тип контейнера – Черга з пріоритетом), виконано серіалізацію даних елементів контейнера у файл та створення вмісту контейнера через десеріалізацію даних файлу, реалізовано пошук даних у контейнері (запит для пошуку – підрахувати загальну кількість дефектів, знайдених в заданому компоненті програмного забезпечення).

**Структура і обсяг роботи.** Курсова робота складається із вступу, трьох розділів, висновків, списку використаної літератури з 6 джерел і 5 додатків. Загальний обсяг курсової роботи становить 81 сторінок, основного тексту (без додатків) – 50 сторінок, ілюстрацій – 31, таблиць – 32.

**Annotation**

Software development using object oriented approach.

Coursework on the Object Oriented Programming course is devoted to the issue of creating a hierarchy of classes, applying inheritance and polymorphism, developing storage container for the collection of objects created by the user. In the coursework class definitions were performed (hierarchy of program classes: base class – tasks in the software development project management system, derived classes – a bug, a task for development), user menu was designed, set of objects was created and its storage container was developed (type of container – deque with limited output), serialization of container data was done and container elements were created via deserialization of file data, data search in the container was performed (search query – count the total number of defects found in a given software component).

**The structure and the amount of work.** Coursework consists of an introduction, three partitions, conclusions, list of used literature with 6 references, and 5 applications. The total volume of coursework is 81 pages, main text (without applications) – 50 pages, illustrations – 31, tables – 32.

**ЗМІСТ**

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 6](#_Toc135391008)

[ВСТУП 7](#_Toc135391009)

[РОЗДІЛ 1 СПЕЦИФІКАЦІЯ ВИМОГ 8](#_Toc135391010)

[1.1 Функціональні вимоги 8](#_Toc135391011)

[1.2 Нефункціональні вимоги 14](#_Toc135391012)

[РОЗДІЛ 2 СТРУКТУРА ТА ЛОГІКА РОБОТИ 15](#_Toc135391013)

[2.1 Логічна структура 15](#_Toc135391014)

[2.2 Фізична структура 24](#_Toc135391015)

[2.3 Логіка роботи 29](#_Toc135391016)

[РОЗДІЛ 3 ТЕСТУВАННЯ 30](#_Toc135391017)

[3.1 Сценарії тестування вимог 30](#_Toc135391018)

[3.2 Тестові дані 36](#_Toc135391019)

[3.3 Результати роботи ПЗ 38](#_Toc135391020)

[3.4 Модульне тестування 48](#_Toc135391021)

[ВИСНОВКИ 49](#_Toc135391022)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 50](#_Toc135391023)

[Додаток А 51](#_Toc135391024)

[Додаток Б 52](#_Toc135391025)

[Додаток В 54](#_Toc135391026)

[Додаток Г 55](#_Toc135391027)

[Додаток Д 58](#_Toc135391028)

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Скорочення:

ПЗ – програмний застосунок;

Терміни:

ООП (об’єктно-орієнтовне програмування) – це одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою.

Клас – це тип даних, що визначається користувачем і являє собою модель реального об’єкта у вигляді даних та функцій для роботи з ними.

Об’єкт – це екземпляр класу.

Поля – змінні, визначені всередині класу, які містять в собі дані, властиві об’єктам даного класу.

Методи – це функції, визначені всередині класу для роботи з полями.

Конструктор – це особливий тип методу класу, який використовуються для ініціалізації змінних-членів класу значеннями, які надані за замовчуванням або користувачем.

Деструктор – це спеціальний тип методу класу, який призначений для видалення раніше створеного об’єкта класу.

UML (англ. Unified Modeling Language) – це уніфікована графічна мова моделювання, яка використовується для опису, візуалізації, проектування та документування об’єктно-орієнтованих систем.

CLI (англ. Command Line Interface) – текстовий інтерфейс користувача.

# ВСТУП

Важливим аспектом створення якісного та надійного сучасного програмного забезпечення є відповідальний підхід фахівця до його розробки. Це означає, що розробник повинен мати глибоке розуміння методів аналізу, проектування, реалізації та тестування програмних систем, використовуючи при цьому сучасні підходи та технології. Один з таких підходів, який вже успішно застосовується в програмуванні – це об'єктно-орієнтований підхід. Використання об'єктно-орієнтованого підходу дозволяє розробникам створювати більш модульне, гнучке та легко зрозуміле програмне забезпечення, що спрощує процес його розробки та підтримки.

Курсова робота спрямована на закріплення знань з дисципліни «Об’єктно-орієнтоване програмування», а саме поглиблення теоретичний знань та отримання практичних навичок щодо об’єктно-орієнтованого підходу при розробці програмного забезпечення, під час проектування складових програмного забезпечення мовою моделювання UML та – написання коду для програмного застосунку мовою програмування С++ в інтегрованому середовищі розробки Microsoft Visual Studio.

Основні завдання:

* Проаналізувати вимоги
* Виконати проєктування архітектури мовою UML
* Розробити ПЗ мовою C++
* Протестувати ПЗ

# РОЗДІЛ 1 СПЕЦИФІКАЦІЯ ВИМОГ

1.1. Функціональні вимоги

1.1.1. Створення об’єкта типу «задача для розробки» та додавання його до контейнера

Опис варіанта використання даної функціональної вимоги наведено у Таблиці 1.1. Діаграма варіанту використання даної функціональної вимоги в нотації UML наведена на Рис.1.1.

*Таблиця 1.1*

**Створення об’єкта типу «задача для розробки»**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Назва | Мета | Дійові особи | Тип | Опис |
| rq\_01 | Створення об’єкта типу «задача для розробки» | Створити об’єкт типу «задача для розробки» та додати його до контейнера | Користувач | Основний | Користувач заповнює всі необхідні поля об’єкта, після чого об’єкт додається до контейнера |

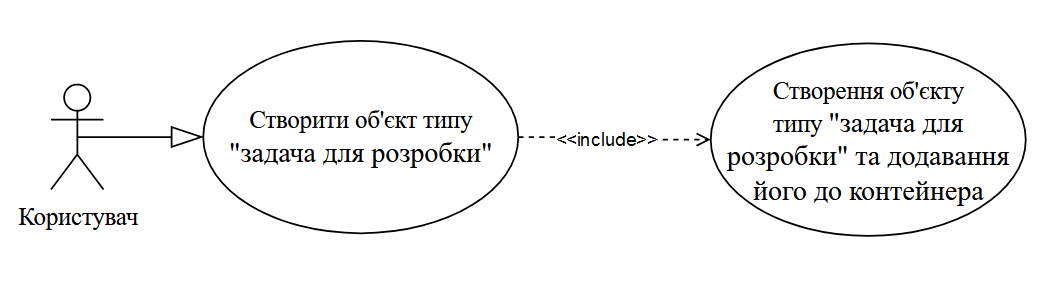


Рис.1.1. Створення об’єкта типу «задача для розробки»

1.1.2. Створення об’єкта типу «дефект» та додавання його до контейнера

Опис варіанта використання даної функціональної вимоги наведено у Таблиці 1.2. Діаграма варіанту використання даної функціональної вимоги в нотації UML наведена на Рис.1.2.

*Таблиця 1.2*

**Створення об’єкта «дефект»**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Назва | Мета | Дійові особи | Тип | Опис |
| rq\_02 | Створення об’єкта «дефект» | Створити об’єкт «дефект» та додати його до контейнера | Користувач | Основний | Користувач заповнює всі необхідні поля об’єкта, після чого об’єкт додається до контейнера |

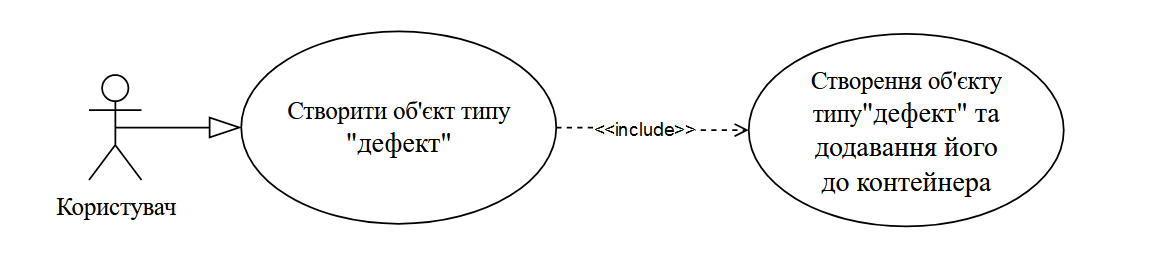


Рис.1.2. Створення об’єкта типу «дефект»

1.1.3. Виведення на екран вмісту контейнера

Опис варіанта використання даної функціональної вимоги наведено у Таблиці 1.3. Діаграма варіанту використання даної функціональної вимоги в нотації UML наведена на Рис.1.3.

*Таблиця 1.3*

**Виведення вмісту контейнера**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Назва | Мета | Дійові особи | Тип | Опис |
| rq\_03 | Вивід вмісту контейнера | Вивести вміст контейнера у командний інтерфейс користувача | Користувач | Основний | Програма виводить дані всіх об’єктів, що містяться в контейнері у командний інтерфейс користувача |

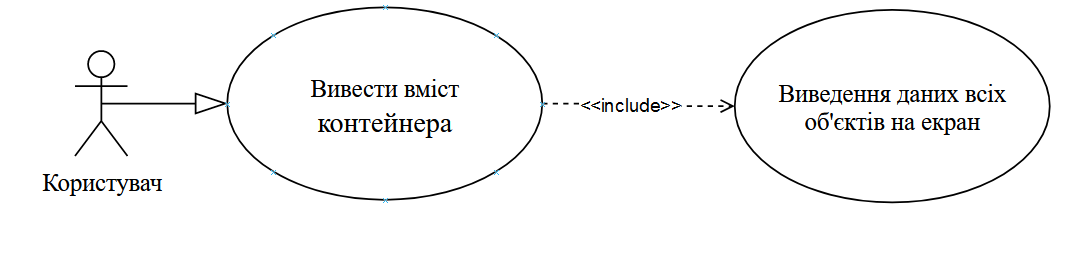


Рис.1.3. Виведення вмісту контейнера

1.1.4. Очищення контейнеру

Опис варіанта використання даної функціональної вимоги наведено у Таблиці 1.4. Діаграма варіанту використання даної функціональної вимоги в нотації UML наведена на Рис.1.4.

*Таблиця 1.4*

**Очищення контейнеру**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Назва | Мета | Дійові особи | Тип | Опис |
| rq\_04 | Очистити контейнер | Видалити усі елементи, що зберігаються у контейнері | Користувач | Основний | Програма видаляє усі елементи, що зберігаються у контейнері та виводить повідомлення про очищення |

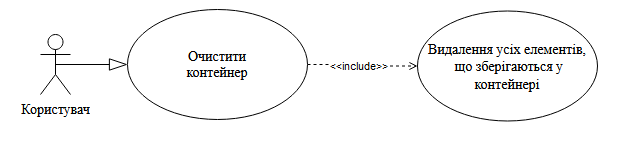


Рис.1.4. Видалення об’єкта з контейнера

1.1.5. Збереження даних всіх об’єктів контейнера у файл

Опис варіанта використання даної функціональної вимоги наведено у Таблиці 1.5. Діаграма варіанту використання даної функціональної вимоги в нотації UML наведена на Рис.1.5.

*Таблиця 1.5*

**Збереження об’єктів контейнера у файл**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Назва | Мета | Дійові особи | Тип | Опис |
| rq\_05 | Запис даних об’єктів у файл | Зберегти дані об’єктів, які зберігаються у контейнері у файл | Користувач | Основний | Користувач вводить назву файлу, куди необхідно зберегти дані, після чого всі дані об’єктів записуються в цей файл |

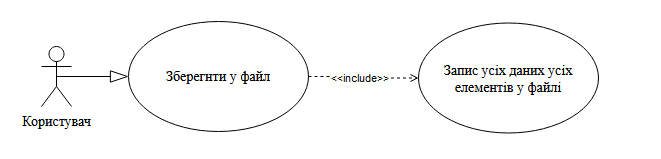


Рис.1.5. Збереження об’єктів контейнера у файл

1.1.6. Зчитування даних об’єктів з файлу

Опис варіанта використання даної функціональної вимоги наведено у Таблиці 1.6. Діаграма варіанту використання даної функціональної вимоги в нотації UML наведена на Рис.1.6.

*Таблиця 1.6*

**Зчитування даних об’єктів з файлу**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Назва | Мета | Дійові особи | Тип | Опис |
| rq\_06 | Зчитування даних об’єктів з файлу | Завантажити дані об’єктів до контейнера з файлу | Користувач | Основний | Користувач вводить назву файлу з якого зчитуються дані об’єктів, які додаються до контейнера |

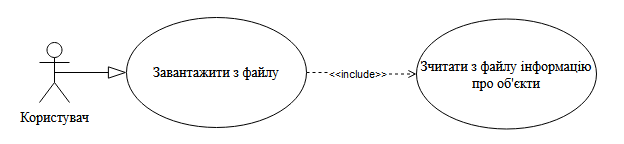


Рис.1.6. Завантаження даних об’єктів з файлу

1.1.7. Підрахунок дефектів у певному компоненті

Опис варіанта використання даної функціональної вимоги наведено у Таблиці 1.7. Діаграма варіанту використання даної функціональної вимоги в нотації UML наведена на Рис.1.7.

*Таблиця 1.7*

**Виконання запиту на підрахунок дефектів у певному компоненті програми**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Назва | Мета | Дійові особи | Тип | Опис |
| rq\_07 | Виконання запиту на підрахунок дефектів у певному компоненті програми | Підрахувати кількість об’єктів класу дефект у певному компоненті | Користувач | Основний | Користувач вводить назву компоненту, в якому рахується кількість дефектів |

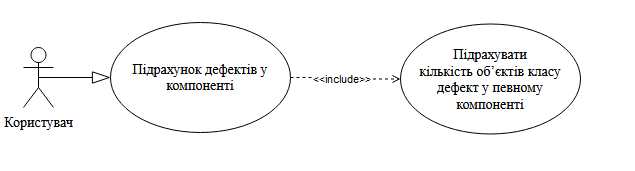


Рис.1.7. Виконання запиту на пошук об’єкта

1.1.8. Сортування

Опис варіанта використання даної функціональної вимоги наведено у Таблиці 1.8. Діаграма варіанту використання даної функціональної вимоги в нотації UML наведена на Рис.1.8.

*Таблиця 1.8*

**Сортування черги з пріоритетом**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Назва | Мета | Дійові особи | Тип | Опис |
| rq\_08 | Сортування | Відсортувати елементи черги за певним правилом | Користувач | Основний | Користувачу дається вибір за яким правилом сортувати чергу і відсортувати її за цим вибором |

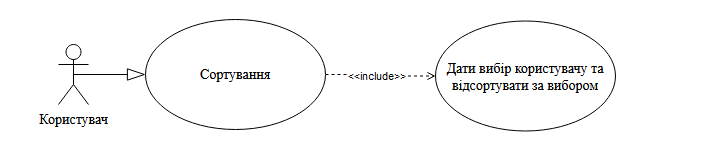


Рис.1.8. Сортування

1.1.9. Вихід з програми

Опис варіанта використання даної функціональної вимоги наведено у Таблиці 1.9. Діаграма варіанту використання даної функціональної вимоги в нотації UML наведена на Рис.1.9.

*Таблиця 1.9*

**Вихід з програми**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Назва | Мета | Дійові особи | Тип | Опис |
| rq\_09 | Вихід з програми | Завершити роботу ПЗ | Користувач | Основний | Відбувається очищення всіх полів об’єктів, видалення контейнера та завершення роботи програми |

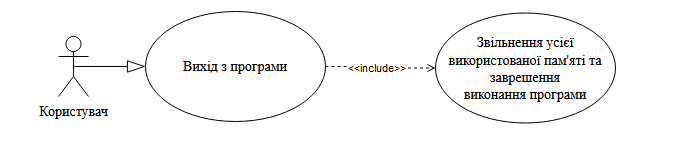


Рис.1.9. Вихід з програми

1.2. Нефункціональні вимоги

nfrq\_01: Перевірка вибору пункту меню користувачем. У випадку некоректного вводу, виведеться повідомлення про помилку та можливість повторити спробу обрати пункт меню.

nfrq \_02: Після введення значень з клавіатури проводиться перевірка їх правильності. У випадку, якщо користувач ввів некоректні дані, система виведе повідомлення про помилку і просить повторити спробу введення інформації..

nfrq \_03: Проводиться перевірка даних, отриманих з файлу. У випадку, некоректності отриманих даних, виведеться повідомлення.

# РОЗДІЛ 2 СТРУКТУРА ТА ЛОГІКА РОБОТИ

## 2.1. Логічна структура

Діаграма класів наведена на окремому аркуші (Див. Додаток А)

2.1.1. Клас «Assignment»

Клас «Assignment» - це абстрактний базовий клас, призначений для визначення сутності «Завдання». Опис захищеного інтерфейсу класу «Assignment» наведено в Таблиці 2.1. Опис відкритого інтерфейсу класу «Assignment» наведено в Таблиці 2.2.

Таблиця 2.1

**Поля захищеного інтерфейсу класу Assignment**

|  |  |  |  |
| --- | --- | --- | --- |
| № з\п | Назва | Тип | Призначення |
| 1 | id | int | ID завдання |
| 2 | priority | int | Пріоритет завдання |
| 3 | component | string | Назва компоненту |

Таблиця 2.2

**Методи відкритого інтерфейсу класу Assignment**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № з\п | Сигнатура | Вхідні параметри | Повернене значення | Призначення |
| 1 | Assignment() | – | — | Конструктор за замовченням |
| 2 | Assignment(int \_id, int \_priority, std::string \_component) | \_id – ID завдання; \_priority – пріоритет завдання; \_component –  назва компоненту | — | Конструктор з параметрами |

*Продовж. Таблиці 2.2*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | virtual ~Assignment() | — | — | Віртуальний деструктор |
| 4 | void setId(int \_id) | \_id – ID завдання | — | Сеттер, що присвоює значення полю id |
| 5 | void setPriority(int \_priority) | \_ priority –  пріоритет завдання | — | Сеттер, що присвоює значення полю priority |
| 6 | void setComponent(  std::string \_component) | \_component –  назва компоненту | — | Сеттер, що присвоює значення полю component |
| 7 | int getId() const | — | id – ID завдання | Геттер, що повертає значення поля id |
| 8 | int getPriority() const | — | priority –  пріоритет завдання | Геттер, що повертає значення поля  number |
| 9 | std::string getComponent() const | — | component – назва компонен-ту | Геттер, що повертає значення поля  component |
| 10 | virtual void createFromKeyboard() | — | — | Віртуальний метод, що заповнює дані об’єкту через ввід користувача |

*Продовж. Таблиці 2.2*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | virtual void show() const | — | — | Віртуальний метод, що виводить дані об’єкту в потік потік CLI |

2.1.2. Клас «Task»

Клас «Task» - це клас-нащадок, призначений для визначення сутності «Завдання для розробки». Опис захищеного інтерфейсу класу «Task» наведено в Таблиці 2.3. Опис відкритого інтерфейсу класу «Task» наведено в Таблиці 2.4.

Таблиця 2.3

**Поля захищеного інтерфейсу класу Task**

|  |  |  |  |
| --- | --- | --- | --- |
| № з\п | Назва | Тип | Призначення |
| 1 | taskDesc | string | Опис завдання |

Таблиця 2.4

**Методи відкритого інтерфейсу класу Task**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № з\п | Сигнатура | Вхідні параметри | Повернене значення | Призначення |
| 1 | Task () | – | — | Конструктор за замовченням |
| 2 | Task (int \_id, int \_priority, std::string \_component, std::string \_taskDesc) | \_id – ID завдання; \_priority – пріоритет завдання; \_component –  назва компоненту; \_taskDesc – опис завдання | — | Конструктор з параметрами |

*Продовж. Таблиці 2.4*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | ~Task() | — | — | Деструктор |
| 4 | void setTaskDesc(std::string \_taskDesc) | \_taskDesc – опис завдання | — | Сеттер, що присвоює значення полю taskDesc |
| 5 | std::string getTaskDesc() const | — | taskDesc – опис завдання | Геттер, що повертає значення поля taskDesc |
| 6 | void createFromKeyboard() override | — | — | Перевизначений метод, що заповнює дані об’єкту через ввід користувача |
| 7 | void show() const override | — | — | Перевизначений метод, що виводить дані об’єкту в потік потік CLI |
| 8 | friend std::ostream& operator<<(std::ostream& out, const Task& task) | out – адреса вихідного потоку;  task – об’єкт типу Task | out – адреса вихідного потоку | Метод виводу інформації у файловий потік, через перевантаження оператору |
| 9 | friend std::ifstream& operator>>(std::ifstream& in, Task\* task) | in – адреса  вхідного потоку;  task – об’єкт типу Task | in – адреса  вхідного потоку | Метод зчитування інформації з файлового потоку, через перевантаження оператору |

2.1.3. Клас «Bug»

Клас «Bug» - це клас-нащадок, призначений для визначення сутності «Дефект». Опис захищеного інтерфейсу класу «Bug» наведено в Таблиці 2.5. Опис відкритого інтерфейсу класу «Bug» наведено в Таблиці 2.6.

Таблиця 2.5

**Поля захищеного інтерфейсу класу Bug**

|  |  |  |  |
| --- | --- | --- | --- |
| № з\п | Назва | Тип | Призначення |
| 1 | expRes | string | Очікувані результати |
| 2 | obtRes | string | Отримані результати |

Таблиця 2.6

**Методи відкритого інтерфейсу класу Bug**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № з\п | Сигнатура | Вхідні параметри | Повернене значення | Призначення |
| 1 | Bug () | – | — | Конструктор за замовченням |
| 2 | Bug (int \_id, int \_priority, std::string \_component, std::string \_bagDesc, std::string \_obtRes) | \_id – ID завдання; \_priority – пріоритет завдання; \_component –  назва компоненту; \_expRes – очікувані результати; \_obtRes – отримані результати | — | Конструктор з параметрами |
| 3 | ~ Bug () | — | — | Деструктор |

*Продовж. Таблиці 2.6*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | void setExpRes(std::string \_expRes) | \_expRes – очікувані результати | — | Сеттер, що присвоює значення полю expRes |
| 5 | std::string getExpRes() const | — | expRes – очікувані результати | Геттер, що повертає значення поля expRes |
| 6 | void setObtRes(std::string \_obtRes) | \_obtRes – отримані результати | — | Сеттер, що присвоює значення полю obtRes |
| 7 | std::string getObtRes() const | \_— | obtRes – отримані результати | Геттер, що повертає значення поля obtRes |
| 8 | void createFromKeyboard() override | — | — | Перевизначений метод, що заповнює дані об’єкту через ввід користувача |
| 9 | void show() const override | — | — | Перевизначений метод, що виводить дані об’єкту в потік потік CLI |
| 10 | friend std::ostream& operator<<(std::ostream& out, const Bug & bug) | out – адреса вихідного потоку;  task – об’єкт типу Task | out – адреса вихідного потоку | Метод виводу інформації у файловий потік, через перевантаження оператору |

*Продовж. Таблиці 2.6*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | friend std::ifstream& operator>>(std::ifstream& in, Bug\* bug) | in – адреса  вхідного потоку;  task – об’єкт типу Task | in – адреса  вхідного потоку | Метод зчитування інформації з файлового потоку, через перевантаження оператору |

2.1.4. Структура «Node»

Структура «Node» необхідна для представлення сутності об’єктів, що знаходяться в контейнері. Опис полів структури наведено в Таблиці 2.7.

Таблиця 2.7

**Поля структури Node**

|  |  |  |  |
| --- | --- | --- | --- |
| № з\п | Назва | Тип | Призначення |
| 1 | priority | int | Значення пріоритету |
| 2 | el | Assignment \* | Покажчик на об’єкт завдання |
| 3 | next | Node \* | Покажчик на наступний елемент |

2.1.5. Клас «QueueP»

Клас «QueueP» - це клас-контейнер, що зберігає об’єкти типів «Task» та «Bug». Опис закритого інтерфейсу класу «QueueP» наведено в Таблиці 2.8. Опис відкритого інтерфейсу класу «QueueP» наведено в Таблиці 2.9.

Таблиця 2.8

**Поля закритого інтерфейсу класу QueueP**

|  |  |  |  |
| --- | --- | --- | --- |
| № з\п | Назва | Тип | Призначення |
| 1 | size | int | Розмір черги з пріоритетом |
| 2 | head | Node\* | Покажчик на перший елемент |

Таблиця 2.9

**Методи відкритого інтерфейсу класу QueueP**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № з\п | Сигнатура | Вхідні параметри | Повернене значення | Призначення |
| 1 | QueueP() | — | — | Конструктор за замовчуванням |
| 2 | ~QueueP() | — | — | Деструктор |
| 3 | int getSize() const | — | size - розмір черги з пріоритетом | Геттер, що повертає значення поля size |
| 4 | Node\* getHead() const | — | head - покажчик на перший елемент | Геттер, що повертає посилання на елемент поля head |
| 5 | void push(Assignment\* elem, int priority) | elem – об’єкт, що додається в контейнер; priority – приорітет елементу | — | Метод додавання елемента в чергу з пріоритетом |

*Продовж. Таблиці 2.9*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 6 | void pop | — | — | Метод видалення одного елемента |
|  | void clear() | — | — | Метод очищення черги |
| 7 | bool empty() const | — | Булеве значення | Метод перевірки черги на порожність |
| 8 | void show() const | — | — | Метод, що виводить дані об’єкту в потік потік CLI |
| 9 | void sort() | — | — | Метод, що сортує чергу |
| 10 | QueueP\* operator=() | — | — | Перевантажен-ня оператора привласнення |
| 11 | void readFromFile () | — | — | Метод для зчитування елементів з файлу у чергу |
| 12 | writeToFile() | — | — | Метод для запису черги у файл |

*Продовж. Таблиці 2.9*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 13 | doReq() | — | — | Метод для отримання кількості елементів типу дефект з черги, що знаходиться в певному компоненті |

## 2.2. Фізична структура

Опис у вигляді таблиці розподілу класів, змінних, функцій та інших компонентних структур по файлах наведено у Таблиці 2.10.

Таблиця 2.10

**Опис фізичної структури**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № з\п | Назва файлу | Компоненти | | Зовнішні компоненти | |
| назва | опис | назва | опис |
| 1 | main.cpp | Run | Функція, необхідна для організації роботи програми | QueueP.h | Хедер з визначенням черги з пріоритетом |
| main | Функція, необхідна для запуску програми |

*Продовж. Таблиці 2.10*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | Assignment.h | Assignment | Визначення абстрактного класу Assignment | iostream | Бібліотека для роботи з потоковим вводом/ виводом даних у CLI |
| fstream | Бібліотека для роботи з файловим потоком вводу/виводу |
| exception | Бібліотека для роботи з винятками |
| string | Бібліотека для роботи з класом std::string |
| ios | Бібліотека для отримання розміру потоку |
| limits | Бібліотека для отримання числових обмежень |
| sys/stat.h | Бібліотека для перевірки існування файлу (stat) |

*Продовж. Таблиці 2.10*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | Assignment.cpp | Assignment | Реалізація методів абстрактного класу Assignment | Assignment.h | Хедер з визначенням абстрактного класу Assignment |
| Assignment |
| ~ Assignment |
| setId |
| setPriority |
| setComponent |
| getId |
| getPriority |
| getComponent |
| createFromKeyboard |
| show |
| 4 | Task.h | Task | Визначення дочірнього класу Task | Assignment.h | Хедер з визначенням абстрактного класу Assignment |

*Продовж. Таблиці 2.10*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | Task.cpp | Task | Реалізація методів дочірнього класу Task | Task.h | Хедер з визначенням дочірнього класу Task |
| Task |
| ~ Task |
| setTaskDesc |
| getTaskDesc |
| createFromKeyboard |
| show |
| operator<< |
| operator>> |
| 6 | Bug.h | Bug | Визначення дочірнього класу Bug | Assignment.h | Хедер з визначенням абстрактного класу Assignment |

*Продовж. Таблиці 2.10*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 7 | Bug.cpp | Bug | Реалізація методів дочірнього класу Bug | Bug.h | Хедер з визначенням дочірнього класу Bug |
| Bug |
| ~ Bug |
| setExpRes |
| setObtRes |
| getExpRes |
| getObtRes |
| createFromKeyboard |
| show |
| operator<< |
| operator>> |
| 8 | QueueP.h | Node | Визначення структури «Node» | Bug.h | Хедер з визначенням дочірнього класу Bug |
| QueueP | Визначення класу контейнера QueueP | Task.h | Хедер з визначенням дочірнього класу Task |

*Продовж. Таблиці 2.10*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 9 | QueueP.cpp | QueueP | Реалізація методів класу контейнера QueueP | QueueP.h | Хедер з визначенням класу контейнера |
| ~ QueueP |
| getSize |
| getHead |
| push |
| pop |
| clear |
| empty |
| show |
| sort |
| operator= |
| readFromFile |
| writeToFile |
| doReq |

## 2.3. Логіка роботи

Діаграма послідовності в нотації UML для визначення часової послідовності подій, що відбувається під час роботи з ПЗ, наведена на окремому аркуші (Див. Додаток Б).

# РОЗДІЛ 3 ТЕСТУВАННЯ

## 3.1. Сценарії тестування вимог

### 3.1.1. Створення об’єкту типу Завдання для розробки

Сценарій тестування даної функціональної вимоги наведено у Таблиці 3.1.

Таблиця 3.1

**Сценарій тестування вимоги створення об’єкту Завдання для розробки**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Назва  варіанта тестування | ID вимоги | Передумова | Тестові кроки | Очікуваний результат | Післяумова |
| t\_01 | Створення об’єкта Завдання для розробки | rq\_01 | Виклик меню користувача | 1. Вибрати в меню пункт 1.  3. Заповнити всі поля об’єкта. | Створюється об’єкт та додається в контейнер. | Виклик меню користувача |

### 3.1.2. Створення об’єкту типу Дефект

Сценарій тестування даної функціональної вимоги наведено у Таблиці 3.2.

Таблиця 3.2

**Сценарій тестування вимоги створення об’єкту Дефект**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Назва  варіанта тестування | ID вимоги | Передумова | Тестові кроки | Очікуваний результат | Післяумова |
| t\_02 | Створення об’єкта Дефект | rq\_02 | Виклик меню користувача | 1. Вибрати в меню пункт 2.  3. Заповнити всі поля. | Створюється об’єкт та додається в контейнер. | Виклик меню користувача |

### 3.1.3. Виведення на екран вмісту контейнера

Сценарій тестування даної функціональної вимоги наведено у Таблиці 3.3.

Таблиця 3.3

**Сценарій тестування вимоги виведення вмісту контейнера**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Назва варіанта тестування | ID вимоги | Передумова | Тестові кроки | Очікуваний результат | Післяумова |
| t\_03 | Вивід вмісту контейнера | rq\_03 | Виклик меню користувача | 1. Вибрати в меню пункт 3. | На екран виводяться дані наявних у контейнері об’єктів | Виклик меню користувача |

### 3.1.4. Очищення контейнера

Сценарій тестування даної функціональної вимоги наведено у Таблиці 3.4.

Таблиця 3.4

**Сценарій тестування вимоги видалення об’єкта з контейнера**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Назва варіанта тестування | ID вимоги | Передумова | Тестові кроки | Очікуваний результат | Післяумова |
| t\_04 | Очищення контейнера | rq\_04 | Виклик меню користувача | 1. Вибрати в меню пункт 4. | Якщо контейнер не є пустий, то з нього видаляються усі об’єкти.  Виводиться повідомлення про видалення. | Виклик меню користувача |

### 3.1.5. Збереження даних всіх об’єктів контейнера у файл

Сценарій тестування даної функціональної вимоги наведено у Таблиці 3.5.

Таблиця 3.5

**Сценарій тестування вимоги збереження об’єктів контейнера у файл**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Назва  варіанта тестування | ID вимоги | Передумова | Тестові кроки | Очікуваний результат | Післяумова |
| t\_05 | Запис даних об’єктів у файл | rq\_05 | Виклик меню користувача | 1. Вибрати в меню пункт 5.  2. Ввести назву файлу, куди зберегти дані. | Збереження наявних у контейнері даних у файл.  Виводиться повідомлення про успішність запису даних. | Виклик меню користувача |

### 3.1.6. Зчитування даних об’єктів з файлу

Сценарій тестування даної функціональної вимоги наведено у Таблиці 3.6.

Таблиця 3.6

**Сценарій тестування вимоги зчитування даних об’єктів з файлу**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Назва варіанта тестування | ID вимоги | Перед-умова | Тестові кроки | Очікуваний результат | Післяумова |
| t\_06 | Зчитування даних об’єктів з файлу | rq\_06 | Виклик меню користувача | 1. Вибрати в меню пункт 6.  2. Ввести назву файлу, де збережені дані | Якщо файл існує і дані коректні, проводиться їх зчитування з файлу.  Виводиться повідомлення про успішне чи неуспішне зчитування з файлу. | Виклик меню користувача |

### 3.1.7. Виконання запиту на підрахунок дефектів у певному компоненті програми

Сценарій тестування даної фукціональної вимоги наведено у Таблиці 3.7.

Таблиця 3.7

**Сценарій тестування вимоги виконання запиту на підрахунок дефектів у певному компоненті програми**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Назва  варіанта тестування | ID вимоги | Передумова | Тестові кроки | Очікуваний результат | Післяумова |
| t\_07 | Виконання запиту на підрахунок дефектів у певному компоненті програми | rq\_07 | Виклик меню користувача | 1. Вибрати в меню пункт 7.  2. Ввести назву необхідного компоненту. | Виведеться кількість дефектів, наявних у конкретному компоненті програми. | Виклик меню користувача |

### 3.1.8. Виконання сортування

Сценарій тестування даної функціональної вимоги наведено у Таблиці 3.8.

Таблиця 3.8

**Сценарій тестування вимоги виконання запиту на підрахунок дефектів у певному компоненті програми**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Назва  варіанта тестування | ID вимоги | Передумова | Тестові кроки | Очікуваний результат | Післяумова |
| t\_08 | Виконати сортування контейнера | rq\_08 | Виклик меню користувача | 1. Вибрати в меню пункт 8.  2. Вибрати необхідний режим сортування. | Якщо контейнер не порожній, то він відсортується необхідним чином.  Виведеться повідомлення про результат сортування. | Виклик меню користувача |

### 3.1.9. Вихід з програми

Сценарій тестування даної функціональної вимоги наведено у Таблиці 3.9.

Таблиця 3.9

**Сценарій тестування вимоги вихід з програми**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Назва варіанта тестування | ID вимоги | Передумова | Тестові кроки | Очікуваний результат | Післяумова |
| t\_09 | Вихід з програми | rq\_09 | Виклик меню користувача | 1. Вибрати в меню пункт 9. | Виводиться повідомлення про завершення роботи з програмою.  Проводиться звільнення всіх полів об’єктів, видалення контейнера. | Програма завершує своє виконання |

### 3.1.10. Хибний вибір пункту меню

Сценарій тестування даної нефункціональної вимоги наведено у Таблиці 3.10.

Таблиця 3.10

**Сценарій тестування вимоги перевірки вводу пункту меню**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Назва  варіанта тестування | ID вимоги | Передумова | Тестові кроки | Очікуваний результат | Післяумова |
| t\_10 | Перевірка вибору хибного пункту меню | nfrq\_01 | Виклик меню користувача | Обрати неіснуючий пункт меню, введенням або числа, що лежить за межами проміжку 0, 9 або букви | Виводиться повідомлення про помилку. | Виклик меню користувача. |

### 3.1.11. Неправильне введення даних

Сценарій тестування даної нефункціональної вимоги наведено у Таблиці 3.11.

Таблиця 3.11

**Сценарій тестування вимоги перевірки введених даних**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Назва варіанта тестування | ID вимоги | Перед-умова | Тестові кроки | Очікуваний результат | Післяумова |
| t\_11 | Перевірка введених даних | nfrq\_02 | Виклик меню користувача. Обрати пункт меню | Ввести некоректні дані, ввівши літеру або символ у полі, де має бути число. | Виводиться повідомлення про помилку та запит на повторне введення даних. | Виклик меню користувача. |

### 3.1.12. Некоректний вміст файлу

Сценарій тестування даної нефункціональної вимоги наведено у Таблиці 3.12.

Таблиця 3.12

**Сценарій тестування вимоги перевірки даних, зчитаних з файлу**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Назва варіанта тестування | ID вимоги | Передумова | Тестові кроки | Очікуваний результат | Післяумова |
| to\_12 | Перевірка вмісту файлу | nfr\_03 | Мати файл з невідповідними даними.  Виклик меню користувача. | Обрати пункт 6. | Виводиться повідомлення про помилку | Виклик меню користувача |

## 3.2. Тестові дані

### 3.2.1. Діаграма об’єктів

Діаграма об’єктів у нотації UML наведена на окремому аркуші (Див. Додаток В).

### 3.2.2. Вміст файлу

Так як дані у вигляді, в якому вони у файлі є не зрозумілі (Рис.3.1), то інформацію, що зберігається у файлі наведено у Таблиці 3.13.

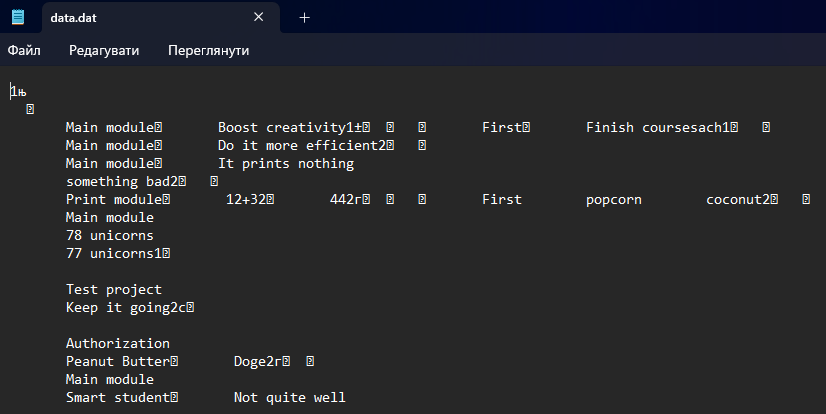


Рис.3.1. Вміст файлу

Таблиця 3.13

**Дані, що зберігаються у файлі**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Type | ID | Priority | Component | Task description | Expected result | Obtained result |
| Task | 1 | 2 | Main module | Do it more efficient. | ­— | — |
| Bug | 2 | 3 | Print function | — | 12+32 | 44 |
| Bug | 3 | 2 | Main module | — | It prints something | something bad |
| Bug | 4 | 5 | Main module | — | 78 unicorns | 77 unicorns |
| Task | 5 | 10 | Test project | Keep it going | — | — |
| Task | 3228 | 1 | Main module | Boost creativity | — | — |
| Task | 8113 | 1 | First | Finish coursesach | — | — |
| Bug | 6115 | 15 | Main module | — | Smart student | Not quite well |
| Bug | 1507 | 3 | First | — | popcorn | coconut |
| Bug | 1635 | 10 | Authorization | — | Peanut Butter | Doge |

## 3.3. Результати роботи ПЗ

Копії з екрану роботи програми під час виконання кожного з усіх пунктів меню користувача наведені на Рис.3.2. –3.21.

При запуску програми, користувача зустрічає CLI, в якому він може обрати 1 з 9 можливих пунктів (Рис.3.2.). Якщо ж користувач забажає обрати щось поза межами можливого вибору, йому виведеться повідомлення про те, що зробив неправильний вибір (Рис.3.3.).

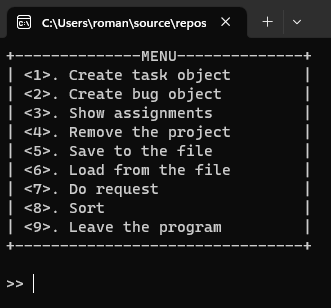


Рис.3.2. Меню користувача

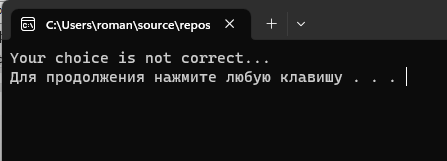


Рис 3.3. Обрання неправильного пункту в меню користувача

Перший та другий пункти – створення об’єкту типу Task та Bug, відповідно. При спробі ввести некоректні дані у поле, буде виведено повідомлення користувачу та прохання ввести коректні дані (Рис.3.4.).

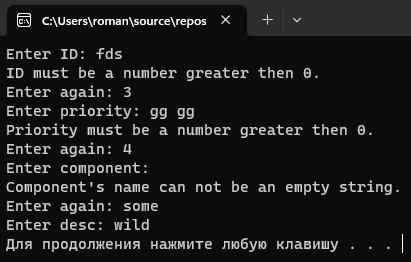
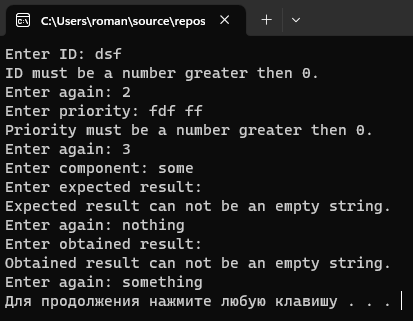
 

Рис 3.4. Приклад створення об’єктів

Третій пункт меню виведе вище створені об’єкти у CLI (Рис.3.5.). Четвертий пункт видалить усі існуючі об’єкти (Рис.3.6.). Тоді, при використанні третього пункту, буде виведено повідомлення, що контейнер порожній (Рис.3.7.).

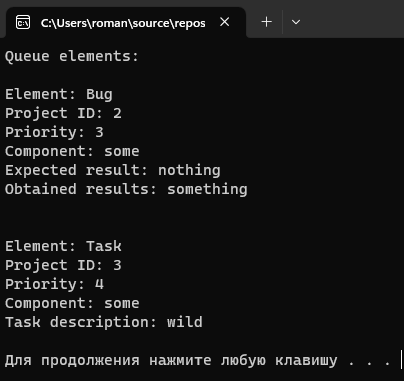


Рис 3.5. Вивід існуючих об’єктів

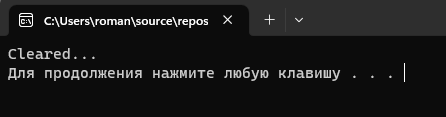


Рис 3.6. Очищення контейнеру

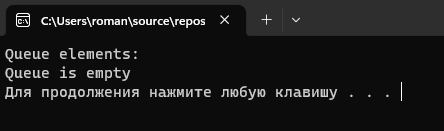


Рис 3.7. Вивід порожнього контейнеру

П’ятий пункт поки пропустимо та перейдемо до шостого. Було завчасно заготовано файл, який зберігає дані п’яти об’єктів («data.dat»). При завантаженні перевіряється файлове розширення (Рис.3.8.) та коректність даних (Рис.3.9.). Вміст файлу з некоректними даними наведено у Рис.3.10.

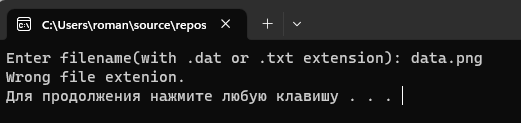
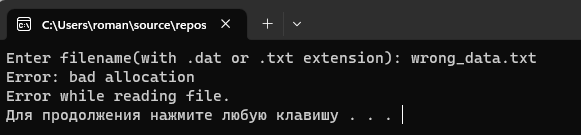


Рис 3.8. Некоректне розширення файлу

 Рис 3.9. Некоректні дані у файлі

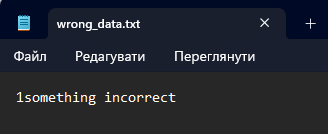


Рис.3.10. Вміст файлу з некоректними даними

У випадку, коли користувача вводить файл з коректним розширенням та у файлі зберігаються коректні дані, буде виведено повідомлення про успішне читання файлу (Рис.3.11).

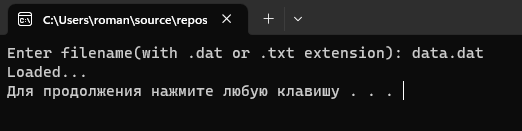


Рис.3.11. Успішне зчитування файлу

Переглянемо завантажені дані завдяки третьому пункту меню (декілька перших елементів) (Рис.3.12.) та збережемо їх у новому файлі («new\_data.dat»), завдяки п’ятому пункту меню. Також спробуємо використати некоректне розширення файлу (Рис.3.13.) та зберегти дані у існуючий файл (Рис.3.14.).

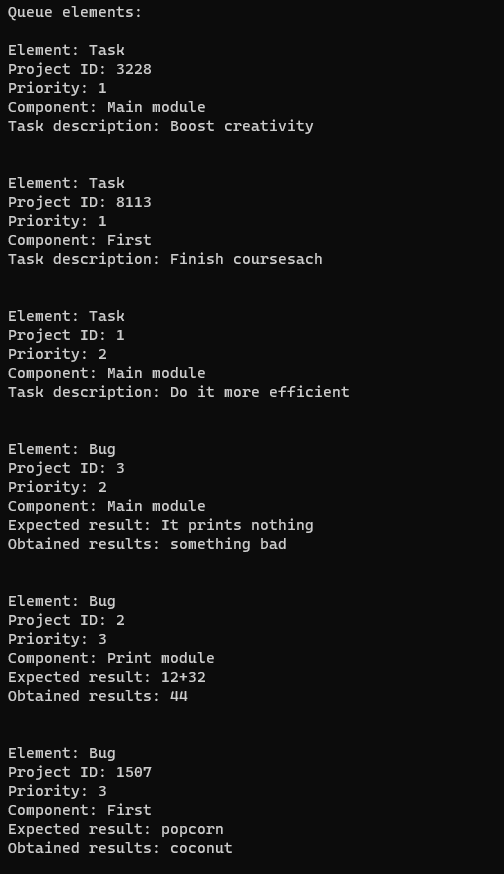


Рис.3.12. Відображення даних

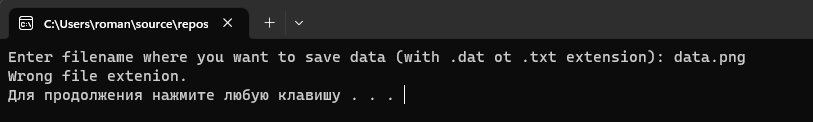


Рис.3.13. Спроба використати некоректне розширення файлу

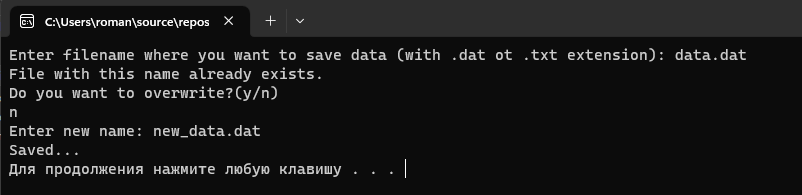


Рис.3.14. Спроба зберегти дані в існуючий файл

Тепер спробуємо на цих даних використати сьомий пункт меню, який має брати назву компоненту та підраховувати баги в ньому (Рис.3.15.). Можна побачити, що регістр літер назви не впливає на результат.

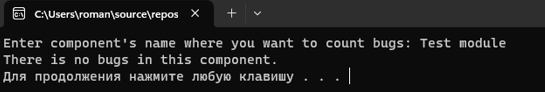
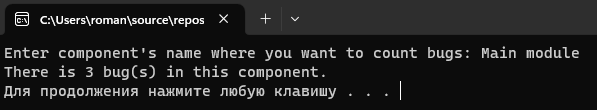
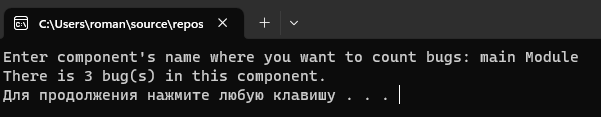


Рис.3.15. Використання 7 пункту меню

Переходимо до восьмого пункту меню – сортування. Наявні 4 типи застування: По пріоритету за зростанням, по пріоритету за спаданням, по ID за зростанням, по ID за спаданням (Рис.3.16.). Результати роботи сортування зображені на Рис.3.17. та Рис.3.18. Після сортування виводиться повідомлення про успішне сортування (Рис.3.19.) Також передбачена можливість вийти з контекстного меню без застосування сортування (Рис.3.20).

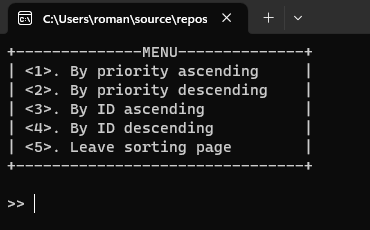


Рис.3.16. Варіанти сортування

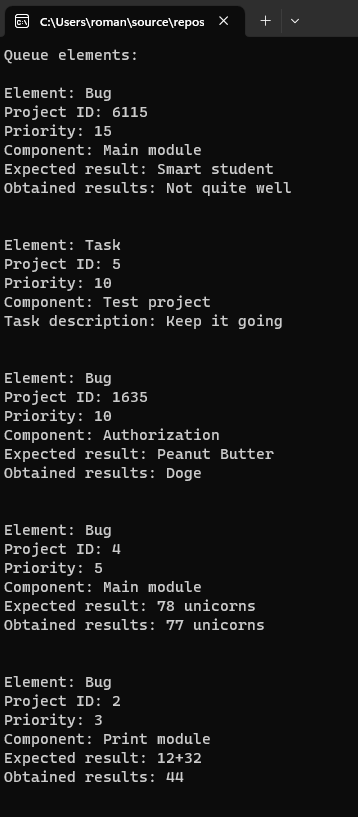
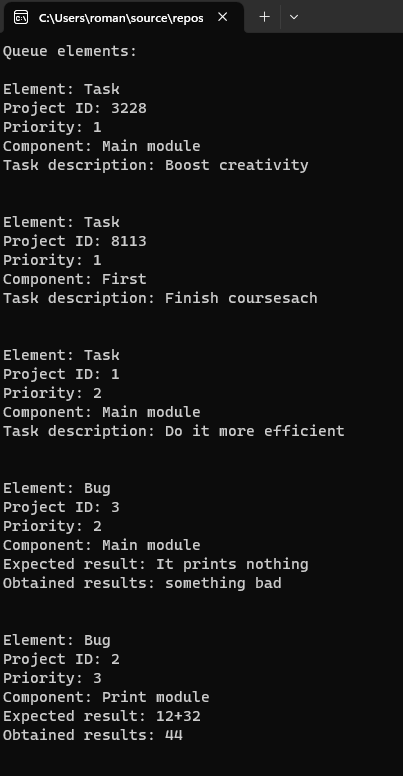


Рис.3.17. Сортування за пріоритетом

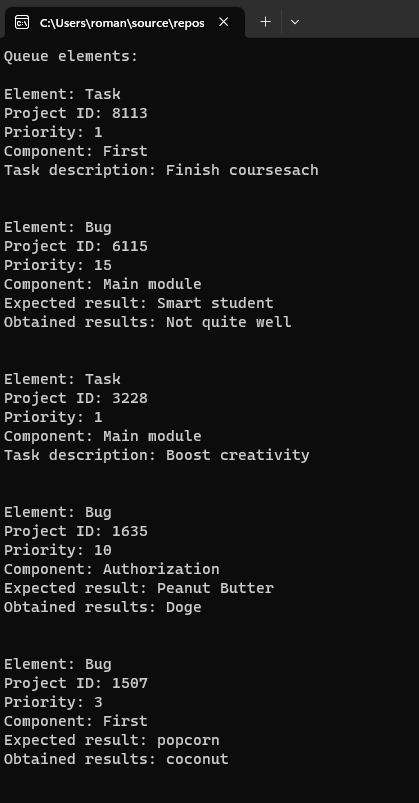
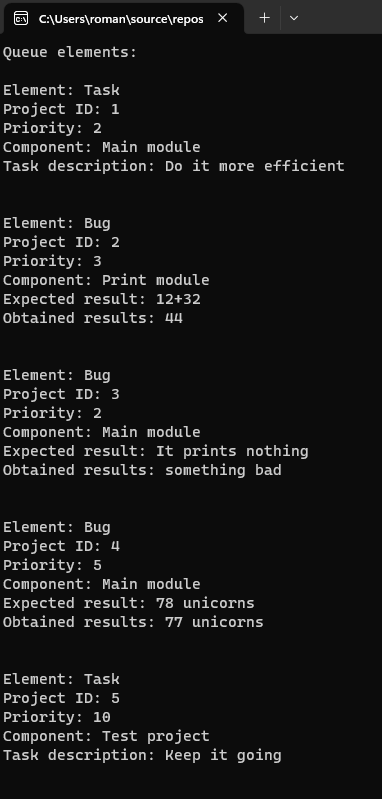


Рис.3.17. Сортування за ID

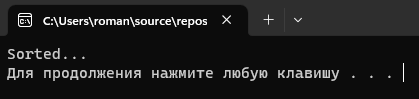


Рис.3.19. Повідомлення про успішне сортування

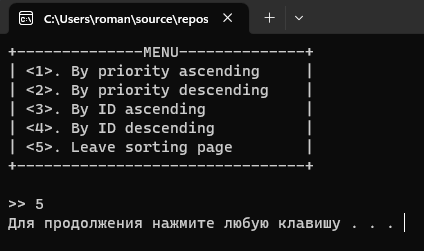


Рис.3.20. Можливість вийти з меню, не застосувавши сортування

Дев’ятий пункт завершує роботу ПЗ. Звільняється зайнята пам’ять та виводиться повідомлення про завершення (Рис.3.21.).

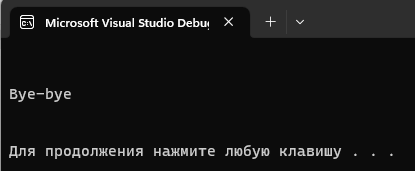


Рис.3.21. Завершення роботи ПЗ

## 3.4. Модульне тестування

Код модульного тестування за допомогою платформи Microsoft CppUnitTest наведено на окремому аркуші (Див. Додаток Г).

Результати тестування наведено на Рис.3.22.

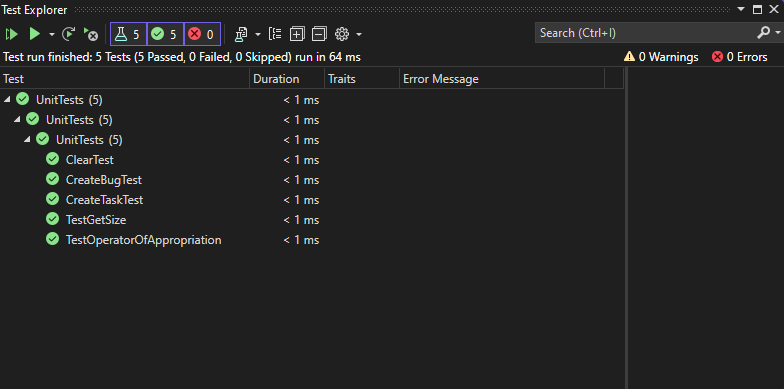


Рис.3.22. Результати тестування

# ВИСНОВКИ

У даній курсовій роботі було покращено та застосовано на практиці знання та навички розробки ПЗ з використанням об’єктно-орієнтованого підходу. Під час виконання було пройдено такі кроки виконання, як постановка задачі, аналіз вимог, проектування архітектури ПЗ мовою UML та написання коду. Під час написання коду було реалізовано ієрархію класів, використано спадкування класів та поліморфізм. Також час пішов на створення класу-контейнеру, що мав зберігати об’єкти дочірніх класів та робити їх серіалізацію і десеріалізацію. Також клас-контейнер мав сортувати елементи та виконувати конкретний, поставлений у вимогах, запит – підрахунок кількості дефектів у певному компоненті, де компонент – поле класу.

ПЗ створювався на мові програмування С++, що дозволило зробити застосунок швидким та ефективним, з точки зору ресурсозатратності. Також ця мова надає всі необхідні можливості для використання об’єктно-орієнтованого підходу.

Також було реалізоване тестування ПЗ, завдяки фреймворку модульного тестування Microsoft CppUnitTest.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Алгоритмізація та програмування мовою високого рівня С++ : комп’ютерний практикум / С. М. Алхімова. - Київ : КПІ ім. Ігоря Сікорського, Вид-во «Політехніка», 2018. - 156 с.
2. Об’єктно-орієнтоване програмування : підручник. У 2-х ч. Ч. 2. Об’єктно-орієнтований підхід до розробки програмного забезпечення / С. М. Алхімова. — Київ : КПІ ім. Ігоря Сікорського, Вид-во «Політехніка», 2019 — 192 с.
3. Уроки програмування на С++ [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: <https://acode.com.ua/uroki-po-cpp/>.
4. C++ reference [Електронний ресурс] – Режим доступу до ресурсу: <https://en.cppreference.com/w/cpp>.
5. C++ Programming Language [Електронний ресурс] // geeksforgeeks – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/c-plus-plus/>.
6. Learn C++ [Електронний ресурс] – Режим доступу до ресурсу: <https://www.learncpp.com/>.

# Додаток А

**UML-діаграма класів**

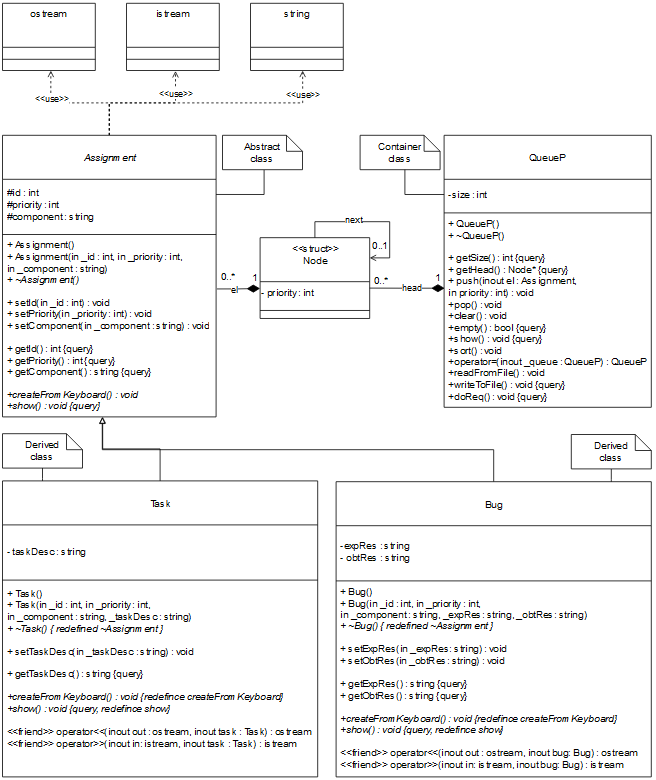


Рис. А.1. Діаграма класів в нотації UML

# Додаток Б

**UML-діаграма послідовностей**

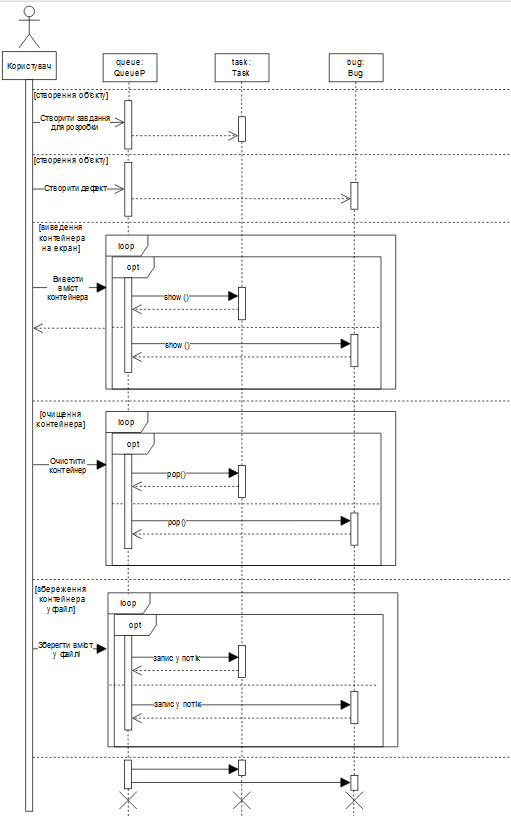


Рис.Б.1 Діаграма послідовностей

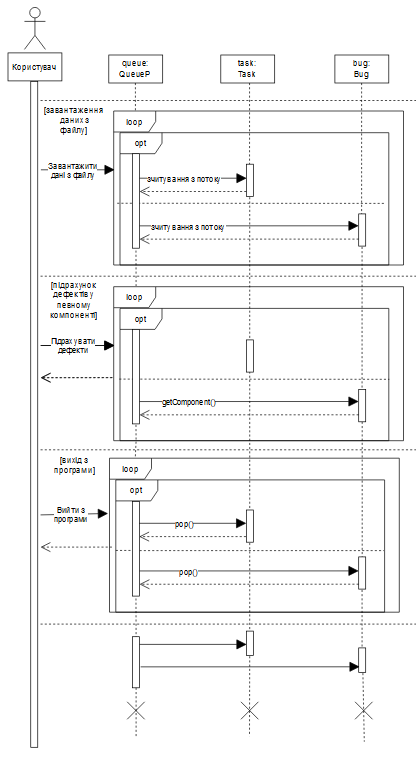


Рис. Б.2 Діаграма послідовностей

# Додаток В

**UML-діаграма об’єктів**

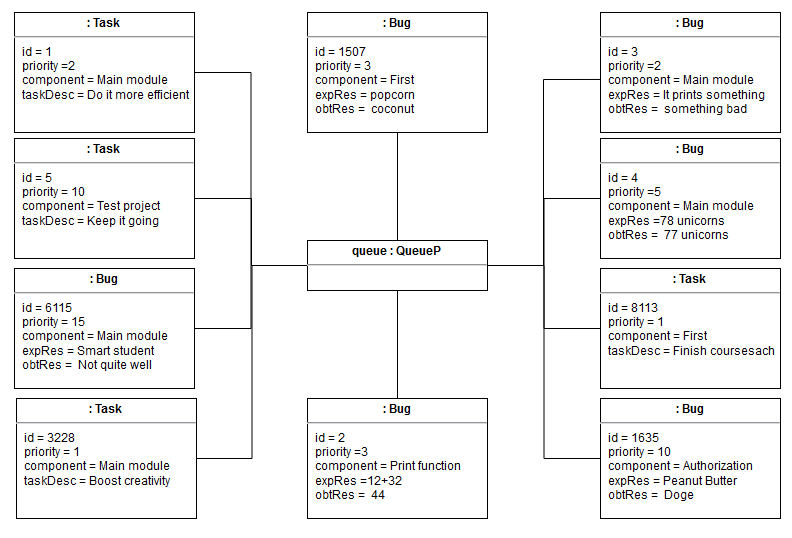


Рис.В.1 Діаграма об’єктів

# Додаток Г

**Код модульних тестів CppUnitTest**

/\* БС-14 Романюк.М.Д.

\* Реалізація модульних тестів на фреймовре CppUnitTest\*/

#include "pch.h"

#include "CppUnitTest.h"

#include "..\Project1\Assignment.h"

#include "..\Project1\Bug.h"

#include "..\Project1\QueueP.h"

#include "..\Project1\Task.h"

#include "..\Project1\Assignment.cpp"

#include "..\Project1\Bug.cpp"

#include "..\Project1\QueueP.cpp"

#include "..\Project1\Task.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTests

{

TEST\_CLASS(UnitTests)

{

public:

//----------------------------------------------------------------

// Тестування вимоги створення об'єкта Bug

// та додавання його до контейнера

TEST\_METHOD(CreateBugTest)

{

QueueP queue;

// контейнер має бути порожнім

Assert::IsTrue(queue.empty());

Assignment\* ass = new Bug(1, 2, "NewFunc", "It is good", "It is bad");

queue.push(ass, ass->getPriority());

// після додавання об'єкта контейнер не має бути порожнім

Assert::IsTrue(queue.empty() == true);

}

//----------------------------------------------------------------

// Тестування вимоги створення об'єкта Task

// та додавання його до контейнера

TEST\_METHOD(CreateTaskTest)

{

QueueP queue;

// контейнер має бути порожнім

Assert::IsTrue(queue.empty());

Assignment\* ass = new Task(2, 3, "OldFunc", "You have to change it to new format");

queue.push(ass, ass->getPriority());

// після додавання об'єкта контейнер не має бути порожнім

Assert::IsTrue(queue.empty() == false);

}

//----------------------------------------------------------------

// Тестування функціональної вимоги очищення контейнера

TEST\_METHOD(ClearTest)

{

QueueP queue;

Assignment\* ass1 = new Task(2, 3, "OldFunc", "You have to change it to new format");

Assignment\* ass2 = new Bug(1, 2, "NewFunc", "It is good", "It is bad");

// в контейнер додається 2 об'єкти

queue.push(ass1, ass1->getPriority());

queue.push(ass2, ass2->getPriority());

// після додавання об'єктів контейнер не має бути порожнім

Assert::IsTrue(!queue.empty());

queue.clear();

// після очищення контейнер має бути порожнім

Assert::IsTrue(queue.empty());

}

// -------------------------------------------------------------------------------------------------

// Тестування нефункціональної вимоги отримання розміру контейнера, позитивний сценарій

TEST\_METHOD(TestGetSize)

{

QueueP queue;

Assignment\* ass1 = new Task(2, 3, "OldFunc", "You have to change it to new format");

Assignment\* ass2 = new Bug(1, 2, "NewFunc", "It is good", "It is bad");

queue.push(ass1, ass1->getPriority());

queue.push(ass2, ass2->getPriority());

Assert::IsTrue(queue.getSize() == 2);

}

// -------------------------------------------------------------------------------------------------

// Тестування нефункціональної вимоги отримання розміру контейнера, позитивний сценарій

TEST\_METHOD(TestOperatorOfAppropriation)

{

QueueP queue;

QueueP \*new\_queue;

Assignment\* ass1 = new Task(2, 3, "OldFunc", "You have to change it to new format");

Assignment\* ass2 = new Bug(1, 2, "NewFunc", "It is good", "It is bad");

queue.push(ass1, ass1->getPriority());

queue.push(ass2, ass2->getPriority());

new\_queue = &queue;

Assert::IsTrue(&queue == new\_queue);

}

};

}

# Додаток Д

**Код програми**

### Файл “Assignment.h”

/\* БС-14 Романюк М.Д.

\* Визначення класу Assingment у файлі Assingment.h \*/

#if !defined(ASSIGNMENT\_H\_)

#define ASSIGNMENT\_H\_

#include <iostream> // використовуєтсья для роботи з потоками вводу/виводу

#include <fstream> // використовується для файлових потоків

#include <exception> // використовується для роботи з винятками

#include <string> // використовується для роботи з класом std::string

#include <ios> // використовується для отримання розміру потоку

#include <limits> // використовується для отримання числових обмежень

#include <sys/stat.h> // використовується для перевірки існування файлу

// Абстрактний клас

class Assignment

{

protected:

// ID завдання

int id;

// Пріоритет завдання

int priority;

// Назва компоненту проєкту

std::string component;

public:

// Конструктор за замовчуванням

Assignment();

// Парметризований конструктор

Assignment(int \_id, int \_priority, std::string \_component);

// Віртуальний деструктор

virtual ~Assignment();

// Сеттер ID

void setId(int \_id);

// Сеттер пріоритету

void setPriority(int \_priority);

// Сеттер назви компоненту проєкту

void setComponent(std::string \_component);

// Геттер ID

int getId() const;

// Геттер пріоритету

int getPriority() const;

// Геттер назви компоненту проєкту

std::string getComponent() const;

// Абстрактний метод для створення завдання з клавіатури

virtual void createFromKeyboard() = 0;

// Віртуальний метод виводу завдання в потік

virtual void show() const;

};

#endif

### Файл “Assignment.cpp”

\* БС-14 Романюк М.Д.

\* Реалізація методів класу Assingment у файлі Assingment.cpp \*/

#include "Assignment.h"

// Конструктор за замовчуванням

Assignment::Assignment()

: id(0)

, priority(0)

, component("")

{}

// Парметризований конструктор

Assignment::Assignment(int \_id, int \_priority, std::string \_component)

: id(\_id)

, priority(\_priority)

, component(\_component)

{}

// Сеттер ID

void Assignment::setId(int \_id) { id = \_id; }

// Сеттер пріоритету

void Assignment::setPriority(int \_priority) { priority = \_priority; }

// Сеттер назви компоненту проєкту

void Assignment::setComponent(std::string \_component) { component = \_component; }

// Геттер ID

int Assignment::getId() const { return id; }

// Геттер пріоритету

int Assignment::getPriority() const { return priority; }

// Геттер назви компоненту проєкту

std::string Assignment::getComponent() const { return component; }

// Віртуальний метод виводу завдання в потік

void Assignment::show() const

{

std::cout << "Project ID: " << id

<< "\nPriority: " << priority

<< "\nComponent: " << component << "\n";

}

// Віртуальний деструктор

Assignment::~Assignment()

{}

### Файл “Task.h”

/\* БС-14 Романюк М.Д.

\* Визначення класу Task у файлі Task.h \*/

#if !defined(TASK\_H\_)

#define TASK\_H\_

#include "Assignment.h"

class Task final: public Assignment

{

private:

// Опис завдання

std::string taskDesc;

public:

// Конструктор за замовчуванням

Task();

// Парметризований конструктор

Task(int \_id, int \_priority, std::string \_component, std::string \_taskDesc);

// Деструктор

~Task() override;

// Сеттер опису завдання

void setTaskDesc(std::string \_taskDesc);

// Геттер опису завдання

std::string getTaskDesc() const;

// Перевизначений метод для створення завдання з клавіатури

void createFromKeyboard() override;

// Перевизначений метод для виводу завдання в потік

void show() const override;

// Перевантажена дружня функція для серіалізації об'єкта завдання до потоку

friend std::ostream& operator<<(std::ostream& out, const Task& task);

// Перевантажена дружня функція для десеріалізації об'єкта завдання з вхідного потоку

friend std::ifstream& operator>>(std::ifstream& in, Task\* task);

};

#endif

### Файл “Task.cpp”

/\* БС-14 Романюк М.Д.

\* Реалізація методів класу Task у файлі Task.cpp\*/

#include "Task.h"

// -------------------------------------------------------------------------------------------------

// Конструктор за замовчуванням

Task::Task()

: Assignment()

, taskDesc("")

{}

// -------------------------------------------------------------------------------------------------

// Парметризований конструктор

Task::Task(int \_id, int \_priority, std::string \_component, std::string \_taskDesc)

: Assignment(\_id, \_priority, \_component)

, taskDesc(\_taskDesc)

{}

// -------------------------------------------------------------------------------------------------

// Сеттер опису завдання

void Task::setTaskDesc(std::string \_taskDesc) { taskDesc = \_taskDesc; }

// -------------------------------------------------------------------------------------------------

// Геттер опису завдання

std::string Task::getTaskDesc() const { return taskDesc; }

// -------------------------------------------------------------------------------------------------

// Перевизначений метод для створення завдання з клавіатури

void Task::createFromKeyboard()

{

// Взяття значення ID з вводу користувача

std::cout << "Enter ID: ";

std::string inp;

std::cin >> inp;

id = atoi(inp.c\_str());

// Перевірка на коректність вводу

while (id < 1)

{

std::cout << "ID must be a number greater then 0.\nEnter again: ";

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cin >> inp;

id = atoi(inp.c\_str());

}

// Взяття значення пріоритету з вводу користувача

std::cout << "Enter priority: ";

std::cin >> inp;

priority = atoi(inp.c\_str());

// Перевірка на коректність вводу

while (priority < 1)

{

std::cout << "Priority must be a number greater then 0.\nEnter again: ";

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cin >> inp;

priority = atoi(inp.c\_str());

}

// Взяття значення назви компоненту з вводу користувача

std::cout << "Enter component: ";

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::getline(std::cin, component, '\n');

// Перевірка на коректність вводу

while (component.empty())

{

std::cout << "Component's name can not be an empty string.\nEnter again: ";

std::getline(std::cin, component, '\n');

}

// Взяття значення опису завдання з вводу користувача

std::cout << "Enter desc: ";

std::getline(std::cin, taskDesc, '\n');

// Перевірка на коректність вводу

while (taskDesc.empty())

{

std::cout << "Task description can not be an empty string.\nEnter again: ";

std::getline(std::cin, taskDesc, '\n');

}

}

// -------------------------------------------------------------------------------------------------

// Перевизначений метод для виводу завдання в потік

void Task::show() const

{

Assignment::show();

std::cout << "Task description: " << taskDesc << "\n\n";

}

// -------------------------------------------------------------------------------------------------

// Перевантажена дружня функція для серіалізації об'єкта завдання до потоку

std::ostream& operator<<(std::ostream& out, const Task& task)

{

// Запис ID та пріоритет до вихідного файлового потоку

out.write((char\*)&task.id, sizeof(task.id));

out.write((char\*)&task.priority, sizeof(task.priority));

// Запис розміру рядка назви компоненту та сам рядок до вихідного потоку

size\_t componentSize = task.component.size();

out.write((char\*)&componentSize, sizeof(componentSize));

out.write(task.component.data(), componentSize);

// Запис розміру рядка опису завдання та сам рядок до вихідного потоку

size\_t taskDescSize = task.taskDesc.size();

out.write((char\*)&taskDescSize, sizeof(taskDescSize));

out.write(task.taskDesc.data(), taskDescSize);

// Повернення вихідного потоку

return out;

}

// -------------------------------------------------------------------------------------------------

// Перевантажена дружня функція для десеріалізації об'єкта завдання з вхідного потоку

std::ifstream& operator>>(std::ifstream& in, Task\* task)

{

try {

// Зчитування ID з вхідного потоку

in.read((char\*)&task->id, sizeof(task->id));

if (in.fail()) {

throw std::runtime\_error("Error reading task ID");

}

// Зчитування пріоритету з вхідного потоку

in.read((char\*)&task->priority, sizeof(task->priority));

if (in.fail()) {

throw std::runtime\_error("Error reading task priority");

}

// Зчитування розміру рядка назви компонента

size\_t componentSize;

in.read((char\*)&componentSize, sizeof(componentSize));

if (in.fail()) {

throw std::runtime\_error("Error reading component size");

}

// Зчитування рядка назви компонента

task->component.resize(componentSize);

in.read((char\*)task->component.data(), componentSize);

if (in.fail()) {

throw std::runtime\_error("Error reading component's name data");

}

// Зчитування розміру рядка опису завдання

size\_t taskDescSize;

in.read((char\*)&taskDescSize, sizeof(taskDescSize));

if (in.fail()) {

throw std::runtime\_error("Error reading task description size");

}

// Зчитування рядка опису завдання

task->taskDesc.resize(taskDescSize);

in.read((char\*)task->taskDesc.data(), taskDescSize);

if (in.fail()) {

throw std::runtime\_error("Error reading task description data");

}

} // кінець try

// Обробка усіх можливих винятків під час десеріалізації об'єкта з вхідного потоку

catch (std::exception& e) {

std::cout << "Error: " << e.what() << std::endl;

in.setstate(std::ios::failbit);

}

// Повернення вхідного потоку

return in;

} // кінець friend std::ifstream& operator>>(std::ifstream& in, Task\* task)

// -------------------------------------------------------------------------------------------------

// Деструктор

Task::~Task()

{}

### Файл “Bug.h”

/\* БС-14 Романюк М.Д.

\* Визначення класу Bug у файлі Bug.h \*/

#if !defined(bug\_h\_)

#define bug\_h\_

#include "Assignment.h"

class Bug final : public Assignment

{

private:

// Очікувані результати

std::string expRes;

// Отримані результати

std::string obtRes;

public:

// Конструктор за замовчуванням

Bug();

// Парметризований конструктор

Bug(int \_id, int \_priority, std::string \_component, std::string \_expRes, std::string \_obtRes);

// Деструктор

~Bug() override;

// Сеттер очікуваних результатів

void setExpRes(std::string \_expRes);

// Сеттер отриманих результатів

void setObtRes(std::string \_obtRes);

// Геттер очікуваних результатів

std::string getExpRes() const;

// Геттер отриманих результатів

std::string getObtRes() const;

// Перевизначений метод для створення дефекту з клавіатури

void createFromKeyboard() override;

// Перевантажена дружня функція для серіалізації об'єкта дефекту до потоку

void show() const;

// Перевантажена дружня функція для серіалізації об'єкта дефекту до потоку

friend std::ostream& operator<<(std::ostream& out, const Bug& bug);

// Перевантажена дружня функція для десеріалізації об'єкта дефекту з вхідного потоку

friend std::istream& operator>>(std::istream& in, Bug\* bug);

};

#endif

### Файл “Bug.cpp”

/\* БС-14 Романюк М.Д.

\* Реалізація методів класу Bug у файлі Bug.cpp \*/

#include "Bug.h"

// -------------------------------------------------------------------------------------------------

// Конструктор за замовчуванням

Bug::Bug()

: Assignment()

, expRes("")

, obtRes("")

{

}

// -------------------------------------------------------------------------------------------------

// Парметризований конструктор

Bug::Bug(int \_id, int \_priority, std::string \_component, std::string \_expRes, std::string \_givenRes)

: Assignment(\_id, \_priority, \_component)

, expRes(\_expRes)

, obtRes(\_givenRes)

{

}

// -------------------------------------------------------------------------------------------------

// Сеттер очікуваних результатів

void Bug::setExpRes(std::string \_expRes) { expRes = \_expRes; }

// -------------------------------------------------------------------------------------------------

// Сеттер отриманих результатів

void Bug::setObtRes(std::string \_givenRes) { obtRes = \_givenRes; }

// -------------------------------------------------------------------------------------------------

// Геттерочікуваних результатів

std::string Bug::getExpRes() const { return expRes; }

// -------------------------------------------------------------------------------------------------

// Геттер отриманих результатів

std::string Bug::getObtRes() const { return obtRes; }

// -------------------------------------------------------------------------------------------------

// Перевизначений метод для створення дефекту з клавіатури

void Bug::createFromKeyboard()

{

// Взяття значення ID з вводу користувача

std::cout << "Enter ID: ";

std::string inp;

std::cin >> inp;

id = atoi(inp.c\_str());

// Перевірка на коректність вводу

while (id < 1)

{

std::cout << "ID must be a number greater then 0.\nEnter again: ";

std::cin >> inp;

id = atoi(inp.c\_str());

}

// Взяття значення пріоритету з вводу користувача

std::cout << "Enter priority: ";

std::cin >> inp;

priority = atoi(inp.c\_str());

// Перевірка на коректність вводу

while (priority < 1)

{

std::cout << "Priority must be a number greater then 0.\nEnter again: ";

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cin >> inp;

priority = atoi(inp.c\_str());

}

// Взяття значення назви компоненту з вводу користувача

std::cout << "Enter component: ";

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::getline(std::cin, component, '\n');

// Перевірка на коректність вводу

while (component.empty())

{

std::cout << "Component's name can not be an empty string.\nEnter again: ";

std::getline(std::cin, component, '\n');

}

// Взяття значення очікуваних результатів

std::cout << "Enter expected result: ";

std::getline(std::cin, expRes, '\n');

// Перевірка на коректність вводу

while (expRes.empty())

{

std::cout << "Expected result can not be an empty string.\nEnter again: ";

std::getline(std::cin, expRes, '\n');

}

// Взяття значення отриманих результатів

std::cout << "Enter obtained result: ";

std::getline(std::cin, obtRes, '\n');

// Перевірка на коректність вводу

while (obtRes.empty())

{

std::cout << "Obtained result can not be an empty string.\nEnter again: ";

std::getline(std::cin, obtRes, '\n');

}

}

// -------------------------------------------------------------------------------------------------

// Перевизначений метод для виводу завдання в потік

void Bug::show() const

{

Assignment::show();

std::cout << "Expected result: " << expRes

<< "\nObtained results: " << obtRes << "\n\n";

}

// Перевантажена дружня функція для серіалізації об'єкта дефекту до потоку

std::ostream& operator<<(std::ostream& out, const Bug& bug)

{

// Запис ID та пріоритет до вихідного файлового потоку

out.write((char\*)&bug.id, sizeof(bug.id));

out.write((char\*)&bug.priority, sizeof(bug.priority));

// Запис розміру рядка назви компоненту та сам рядок назви компоненту до вихідного потоку

size\_t componentSize = bug.component.size();

out.write((char\*)&componentSize, sizeof(componentSize));

out.write(bug.component.data(), componentSize);

// Запис розміру рядка очікуваних результатів та сам рядок до вихідного потоку

size\_t expResSize = bug.expRes.size();

out.write((char\*)&expResSize, sizeof(expResSize));

out.write(bug.expRes.data(), expResSize);

// Запис розміру рядка отриманих результатів та сам рядок до вихідного потоку

size\_t obtResSize = bug.obtRes.size();

out.write((char\*)&obtResSize, sizeof(obtResSize));

out.write(bug.obtRes.data(), obtResSize);

// Повернення вихідного потоку

return out;

}

// -------------------------------------------------------------------------------------------------

// Перевантажена дружня функція для десеріалізації об'єкта дефекту з вхідного потоку

std::istream& operator>>(std::istream& in, Bug\* bug)

{

try {

// Зчитування ID з вхідного потоку

in.read((char\*)&bug->id, sizeof(bug->id));

if (in.fail()) {

throw std::runtime\_error("Error reading bug ID");

}

// Зчитування пріоритету з вхідного потоку

in.read((char\*)&bug->priority, sizeof(bug->priority));

if (in.fail()) {

throw std::runtime\_error("Error reading bug priority");

}

// Зчитування розміру рядка назви компонента

size\_t componentSize;

in.read((char\*)&componentSize, sizeof(componentSize));

if (in.fail()) {

throw std::runtime\_error("Error reading component size");

}

// Зчитування розміру рядка очікуваного результату назви компонента

bug->component.resize(componentSize);

in.read((char\*)bug->component.data(), componentSize);

if (in.fail()) {

throw std::runtime\_error("Error reading component data");

}

// Зчитування розміру рядка очікуваного результату

size\_t expResSize;

in.read((char\*)&expResSize, sizeof(expResSize));

if (in.fail()) {

throw std::runtime\_error("Error reading expected result size");

}

// Зчитування рядка очікуваного результату

bug->expRes.resize(expResSize);

in.read((char\*)bug->expRes.data(), expResSize);

if (in.fail()) {

throw std::runtime\_error("Error reading expected result data");

}

// Зчитування розміру рядка отриманого результату

size\_t obtResSize = 0;

in.read((char\*)&obtResSize, sizeof(obtResSize));

if (in.fail()) {

throw std::runtime\_error("Error reading obtained result size");

}

// Зчитування рядка отриманого результату

bug->obtRes.resize(obtResSize);

in.read((char\*)bug->obtRes.data(), obtResSize);

if (in.fail()) {

throw std::runtime\_error("Error reading obtained result data");

}

} // кінець try

// Обробка усіх можливих винятків під час десеріалізації об'єкта з вхідного потоку

catch (std::exception& e) {

std::cerr << "Error: " << e.what() << std::endl;

in.setstate(std::ios::failbit);

}

// Повернення вхідного потоку

return in;

} // кінець friend std::istream& operator>>(std::istream& in, Bug\* bug)

// -------------------------------------------------------------------------------------------------

// Деструктор

Bug::~Bug()

{

}

### Файл “QueueP.h”

/\* БС-14 Романюк М.Д.

\* Визначення класу QueueP у файлі QueueP.h \*/

#if !defined(QUEUEP\_H\_)

#define QUEUEP\_H\_

#include "Task.h"

#include "Bug.h"

// Структура, використана в реалізації черги з пріоритетом

struct Node

{

// Пріоритет елемента

int priority;

// Елемент

Assignment\* el;

// Посилання на наступний вузол

Node\* next;

};

class QueueP

{

private:

// Розмір черги з пріоритетом

int size;

// Голова черги з пріоритетом

Node\* head;

public:

// Конструктор за замовченням

QueueP();

// Деструктор

~QueueP();

// Геттер розміру черги з пріоритетом

int getSize() const;

// Геттер посилання на голову черги з пріоритетом

Node\* getHead() const;

// Метод додавання елементу за пріоритетом

void push(Assignment\* elem, int priority);

// Метод видалення одного елемента з голови черги з пріоритетом

void pop();

// Метод очищення черги з пріоритетом

void clear();

// Метод перевірки черги з пріоритетом на порожність

bool empty() const;

// Метод виводу черги з пріоритетом в потік

void show() const;

// Метод сортування черги з пріоритетом

void sort();

// Перевантаження оператора привласнення

QueueP\* operator=(const QueueP& \_queue);

// Метод для зчитування елементів з файлу у чергу з пріоритетом

void readFromFile();

// Метод для запису черги з пріоритетом у файл

void writeToFile() const;

// Метод для отримання кількості елементів типу дефект з черги з пріоритетом, що знаходиться в певному компоненті

void doReq() const;

};

#endif

### Файл “QueueP.cpp”

/\* БС-14 Романюк М.Д.

\* Реалізація методів класу QueueP у файлі QueueP.cpp \*/

#include "QueueP.h"

// -------------------------------------------------------------------------------------------------

// Конструктор за замовченням

QueueP::QueueP()

: size(0)

, head(nullptr)

{}

// -------------------------------------------------------------------------------------------------

// Деструктор

QueueP::~QueueP() { clear(); }

// -------------------------------------------------------------------------------------------------

// Геттер розміру черги з пріоритетом

int QueueP::getSize() const { return size; }

// -------------------------------------------------------------------------------------------------

// Геттер посилання на голову черги з пріоритетом

Node\* QueueP::getHead() const { return head; }

// -------------------------------------------------------------------------------------------------

// Метод додавання елементу за пріоритетом

void QueueP::push(Assignment\* \_el, int \_priority)

{

// Створення тимчасових посилань на вузли

Node\* new\_node, \*tmp;

// Створення вузла, що буде доданий в клас-контейнер

new\_node = new Node;

new\_node->el = \_el;

new\_node->priority = \_priority;

// Перевірка пріоритетності задля вірного розташування

if (head == nullptr || \_priority < head->priority)

{

new\_node->next = head;

head = new\_node;

}

else

{

tmp = head;

while (tmp->next != nullptr && tmp->next->priority <= \_priority)

{

tmp = tmp->next;

}

new\_node->next = tmp->next;

tmp->next = new\_node;

}

// Збільшення розміру

size++;

}

// -------------------------------------------------------------------------------------------------

// Метод видалення одного елемента з голови черги з пріоритетом

void QueueP::pop()

{

// Перевірка на порожність

if (head == nullptr)

return;

// Створення тимчасового вузла

Node\* tmp = head;

// Видалення елемента, завдяки тимчасовому вузлу.

head = head->next;

delete tmp;

size--;

}

// -------------------------------------------------------------------------------------------------

// Метод очищення черги з пріоритетом

void QueueP::clear()

{

while (!empty())

pop();

}

// -------------------------------------------------------------------------------------------------

// Метод перевірки черги з пріоритетом на порожність

bool QueueP::empty() const { return (size == 0); }

// -------------------------------------------------------------------------------------------------

// Метод виводу черги з пріоритетом в потік

void QueueP::show() const

{

// Перевірка на порожність

if (head == nullptr)

{

std::cout << "Queue is empty\n";

return;

}

// Створення тимчасового вузла

Node\* tmp;

tmp = head;

// Поелементний прохід по черзі

while (tmp != nullptr)

{

std::cout << "\nElement: ";

// Перевірка типу елемента

if (typeid(\*tmp->el) == typeid(Task))

std::cout << "Task\n";

else

std::cout << "Bug\n";

// Виведення sajhvfws] елемента d gjnsr

tmp->el->show();

tmp = tmp->next;

}

}

// -------------------------------------------------------------------------------------------------

// Метод сортування черги з пріоритетом

void QueueP::sort()

{

if (!size)

{

std::cout << "Queue is empty.\n";

return;

}

int i;

std::string s;

// Створення тимчасової черги з пріоритетом

QueueP\* new\_q = new QueueP();

// Виведення меню

do

{

system("cls");

std::cout

<< "+--------------MENU--------------+\n"

<< "| <1>. By priority ascending |\n"

<< "| <2>. By priority descending |\n"

<< "| <3>. By ID ascending |\n"

<< "| <4>. By ID descending |\n"

<< "| <5>. Leave sorting page |\n"

<< "+--------------------------------+\n"

<< "\n>> ";

std::getline(std::cin, s, '\n');

i = atoi(s.c\_str());

// Створення тимчасовго вузла

Node\* tmp = head;

// Обробка варіантів сортування

switch (i)

{

// За пріоритетом по зростанню

case 1:

{

while (tmp != nullptr)

{

new\_q->push(tmp->el, tmp->priority);

tmp = tmp->next;

}

\*this = \*new\_q;

// Видалення тимчасової черги з пріоритетом

new\_q->clear();

delete new\_q;

std::cout << "Sorted...\n";

return;

}

// За пріоритетом по спаданню

case 2:

{

while (tmp != nullptr)

{

new\_q->push(tmp->el, tmp->priority\*(-1));

tmp = tmp->next;

}

\*this = \*new\_q;

// Видалення тимчасової черги з пріоритетом

new\_q->clear();

delete new\_q;

std::cout << "Sorted...\n";

return;

}

// За індексом по зростанню

case 3:

{

while (tmp != nullptr)

{

new\_q->push(tmp->el, tmp->el->getId());

tmp = tmp->next;

}

\*this = \*new\_q;

// Видалення тимчасової черги з пріоритетом

new\_q->clear();

delete new\_q;

std::cout << "Sorted...\n";

return;

}

// За індексом по спаданню

case 4:

{

while (tmp != nullptr)

{

new\_q->push(tmp->el, tmp->el->getId()\*(-1));

tmp = tmp->next;

}

\*this = \*new\_q;

// Видалення тимчасової черги з пріоритетом

new\_q->clear();

delete new\_q;

std::cout << "Sorted...\n";

return;

}

case 5:

{

// Видалення тимчасової черги з пріоритетом

new\_q->clear();

delete new\_q;

return;

}

default:

{

system("cls");

std::cout << "Your choice is not correct...\n";

}

}

} while (1);

}

// -------------------------------------------------------------------------------------------------

// Перевантаження оператора привласнення

QueueP\* QueueP::operator=(const QueueP& \_queue)

{

// Перевірка на самоприсвоєння

if (this == &\_queue)

return this;

// Очищення поточої черги з пріоритетом

clear();

Node\* curr = \_queue.head;

// Додавання копій елементів з \_queue до поточої черги з пріоритетом

while (curr != nullptr) {

push(curr->el, curr->priority);

curr = curr->next;

}

// Повернення поточної черги з пріоритетом

return this;

}

// -------------------------------------------------------------------------------------------------

// Функція для отримання розширення файлу

std::string GetFileExtension(const std::string& fileName)

{

if (fileName.find\_last\_of(".") != std::string::npos)

return fileName.substr(fileName.find\_last\_of(".") + 1);

return "";

}

// -------------------------------------------------------------------------------------------------

// Метод для зчитування елементів з файлу у чергу з пріоритетом

void QueueP::readFromFile()

{

// Взяття у користувача назви файлу

std::cout << "Enter filename(with .dat or .txt extension): ";

std::string filename;

std::getline(std::cin, filename, '\n');

// Перевірка на правильність розширення файлу

std::string fileExt = GetFileExtension(filename);

if (!(fileExt == "dat" || fileExt == "txt"))

{

std::cout << "Wrong file extenion.\n";

return;

}

std::ifstream in;

in.open(filename, std::ios::binary);

// Перевірка на успішність відкриття файлу

if (!in.is\_open())

{

std::cout << "Error opening file!\n";

return;

}

// Поелементне виведення об'єктів з файлу

int k, count = 0;

while (in >> k)

{

// Перевірка типу об'єкта

if (k == 1)

{

// Десеріалізація об'єкту тупи завдання з потоку

Task\* task = new Task();

// Перевірка на успішність зчитування

if (!(in >> task))

{

std::cout << "Error while reading file.\n";

in.close();

return;

}

// Додавання об'єкту до черги з пріоритетом

push(task, task->getPriority());

}

else if (k == 2)

{

// Десеріалізація об'єкту тупи дефект з потоку

Bug\* bug = new Bug();

// Перевірка на успішність зчитування

if (!(in >> bug))

{

std::cout << "Error while reading file.\n";

in.close();

return;

}

// Додавання об'єкту до черги з пріоритетом

push(bug, bug->getPriority());

}

else

{

// Обробка випадку, коли у файлі записаний не передбачений специфікатор об'єкту

std::cout << "Error while reading file. Unknown type specifier.\n";

in.close();

return;

}

} // кінець while (in >> k)

std::cout << "Loaded...\n";

in.close();

}

// -------------------------------------------------------------------------------------------------

// Функція перевірки існування файлу

bool chechExistance(const std::string& name) {

struct stat buffer;

return (stat(name.c\_str(), &buffer) == 0);

}

// -------------------------------------------------------------------------------------------------

// Метод для запису черги з пріоритетом у файл

void QueueP::writeToFile() const

{

// Взяття у користувача назви файлу

std::cout << "Enter filename where you want to save data (with .dat ot .txt extension): ";

std::string filename;

std::getline(std::cin, filename, '\n');

// Перевірка на правильність розширення файлу

std::string fileExt = GetFileExtension(filename);

if (!(fileExt == "dat" || fileExt == "txt"))

{

std::cout << "Wrong file extenion.\n";

return;

}

while (chechExistance(filename))

{

std::string ch;

std::cout

<< "File with this name already exists.\n"

<< "Do you want to overwrite?(y/n)\n";

std::getline(std::cin, ch, '\n');

while (!(ch == "y" || ch == "n"))

{

std::cout << "Incorrect choice. Try again: ";

std::getline(std::cin, ch, '\n');

}

if (ch == "n")

{

std::cout << "Enter new name: ";

std::getline(std::cin, filename, '\n');

}

else

break;

}

std::ofstream out;

out.open(filename, std::ios::binary);

// Перевірка на успішність відкриття файлу

if (!out.is\_open())

{

std::cout << "Error opening file!\n";

out.close();

return;

}

// Поелементний вивід об'єктів у файл

Node\* tmp;

tmp = head;

while (tmp != nullptr)

{

if (typeid(\*(tmp->el)) == typeid(Task))

{

// Перевірка на успішність виводу об'єкта завдання

if (!(out << 1 << (\*((Task\*)(tmp->el)))))

std::cout << "Error writing data to the file.\n";

}

else if (typeid(\*(tmp->el)) == typeid(Bug))

{

// Перевірка на успішність виводу об'єкта дефекту

if (!(out << 2 << (\*((Bug\*)(tmp->el)))))

std::cout << "Error writing data to the file.\n";

}

tmp = tmp->next;

}

std::cout << "Saved...\n";

out.close();

}

// -------------------------------------------------------------------------------------------------

// Функція переводу строки в нижній регістр

std::string toLowerCase(const std::string& str)

{

std::string result = str;

// Прохід по елементам строки

for (char& c : result)

{

if (c >= 'A' && c <= 'Z')

{

c += 'a' - 'A';

}

}

return result;

}

// -------------------------------------------------------------------------------------------------

// Метод для отримання кількості елементів типу дефект з черги з пріоритетом,

// що знаходиться в певному компоненті

void QueueP::doReq() const

{

std::string component, tmp\_str;

// Взяття у користувача назви компоненту

std::cout << "Enter component's name where you want to count bugs: ";

std::getline(std::cin, component, '\n');

// Перевірка вводу

if (component.empty())

{

std::cout << "String can not be empty.\n";

return;

}

// Переведення назви в нижній регістр задля запобігання проблеми різниці регістрів

component = toLowerCase(component);

int count = 0;

int len = getSize();

// Поелементний прохід по черзі

Node\* tmp;

tmp = head;

while (tmp != nullptr)

{

// Перевірка типу об'єкта в черзі

if (typeid(\*(tmp->el)) == typeid(Bug))

{

tmp\_str = tmp->el->getComponent();

tmp\_str = toLowerCase(tmp\_str);

// Перевірка на однаковість

if (tmp\_str == component)

count++;

}

tmp = tmp->next;

}

if (count)

{

std::cout << "There is " << count << " bug(s) in this component.\n";

}

else

std::cout << "There is no bugs in this component.\n";

}

### Файл “main.cpp”

/\* БС-14 Романюк М.Д.

\* Розробка програмного забезпечення з використанням об’єктно-орієнтованого підходу.\*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "QueueP.h"

// -------------------------------------------------------------------------------------------------

// Функція меню користувача

void Run()

{

// Створення об'єкту типу черги з пріоритетом

QueueP\* queue = new QueueP();

int i;

std::string s;

// Виведення меню

do

{

system("cls");

std::cout

<< "+--------------MENU--------------+\n"

<< "| <1>. Create task object |\n"

<< "| <2>. Create bug object |\n"

<< "| <3>. Show assignments |\n"

<< "| <4>. Remove the project |\n"

<< "| <5>. Save to the file |\n"

<< "| <6>. Load from the file |\n"

<< "| <7>. Do request |\n"

<< "| <8>. Sort |\n"

<< "| <9>. Leave the program |\n"

<< "+--------------------------------+\n"

<< "\n>> ";

std::getline(std::cin, s, '\n');

i = atoi(s.c\_str());

// Обробка варіантів меню

switch (i)

{

case 1:

{

system("cls");

// Створення нового об'єкту

Assignment\* ptr = new Task();

ptr->createFromKeyboard();

// Додавання об'єкту до черги з пріоритетом

queue->push(ptr, ptr->getPriority());

system("pause");

break;

}

case 2:

{

system("cls");

// Створення нового об'єкту

Assignment\* ptr = new Bug();

ptr->createFromKeyboard();

// Додавання об'єкту до черги з пріоритетом

queue->push(ptr, ptr->getPriority());

system("pause");

break;

}

case 3:

{

system("cls");

std::cout << "Queue elements: \n";

queue->show();

system("pause");

break;

}

case 4:

{

system("cls");

queue->clear();

std::cout << "Cleared...\n";

system("pause");

break;

}

case 5:

{

system("cls");

queue->writeToFile();

system("pause");

break;

}

case 6:

{

system("cls");

queue->readFromFile();

system("pause");

break;

}

case 7:

{

system("cls");

queue->doReq();

system("pause");

break;

}

case 8:

{

system("cls");

queue->sort();

system("pause");

break;

}

case 9:

{

system("cls");

std::cout << "\n\nBye-bye\n\n\n";

system("pause");

return;

}

default:

{

system("cls");

std::cout << "Your choice is not correct...\n";

system("pause");

break;

}

} // кінець switch (i)

} while (1);

} // кінець void Run()

// -------------------------------------------------------------------------------------------------

// Основна функція

int main()

{

Run();

return 0;

}