МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ЛІСОТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
Кафедра інженерії програмного забезпечення

КУРСОВА РОБОТА

з об’єктно-орієнтованого програмування
на тему: «Реалізувати програму « Результати спринтерського забігу»

Студента 2-го курсу КН-22 групи
спеціальності *«Комп’ютерні науки»*

_____*Стасів Владислав*_____
(прізвище та ініціали)
Керівник: *к.т.н., доц. Яцишин С.І.*_____
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала _____
Кількість балів: _____Оцінка: ECTS_____

Члени комісії _____ _____
(підпис) (прізвище та ініціали)
_____ _____
(підпис) (прізвище та ініціали)

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

м. Львів – 2024 рік

ЗАВДАННЯ НА КУРСОВУ РОБОТУ

студенту 2-го курсу групи КН-22
Стасіву Владиславу
Варіант 15

1. Визначити клас ЦИФРОВИЙ ЛІЧИЛЬНИК. Лічильник - це змінна з обмеженим діапазоном, який скидається у початковий стан, коли її цілочисельне значення досягає визначеного максимуму.
2. Визначити конструктори (ініціалізації, копіювання, переміщення), деструктор та методи встановлення і виведення значень полів даних. Забезпечити можливість встановлення максимального і мінімального значень, зчитування поточного значення.
3. Перевантажити операцію () для встановлення значень полів даних, операцію ++ для збільшення значення лічильника на 1, - - для зменшення значення лічильника на 1, += дня збільшення значення лічильника на задану величину, -= для зменшення значення лічильника на задану величину, операцію присвоєння об'єктів = та переміщення, потокові операції введення » та виведення « об'єктів.
4. Визначити похідний клас СЕКУНДОМІР з додатковими полями даних, які позначають включення та виключення секундоміра. Визначити операторні методи зчитування та виведення значення заміряного часу.
5. У межах ієрархії класів побудувати поліморфічний кластер на основі віртуального методу виведення поточного значення лічильника або часу вимкнення секундоміра. Продемонструвати механізм пізнього зв'язування.
6. Розробити клас РЕЗУЛЬТАТИ спринтерського забігу, який містить масив об'єктів класу СЕКУНДОМІР. Визначити найменший та найбільший час забігу, середнє значення часу забігу, результати часу для трьох переможців змагання.
7. Для роботи з масивом об'єктів побудувати та використати клас-ітератор.
8. Реалізувати пункти 1-7 завдання мовою C++ та C# з використанням WinForms.
9. Вимоги до оформлення курсової роботи дивитись у методичних вказівках до виконання курсового проектування з ООП (диск Р)
10. Забезпечити читання і запис даних для формування масиву об’єктів згідно завдання із файлу.

Вимоги:

1. Вхідні дані до роботи: літературні джерела, технічна документація щодо розробки програм, ДСТУ з оформлення документації.
2. Зміст пояснювальної записки: Вступ. Специфікація роботи. Програмна документація. Висновки. Додатки.
3. Перелік графічного матеріалу: діаграма класів додатку; схема алгоритму реалізації одного з розроблених методів (за узгодженням з керівником курсової роботи).
   Дата здачі студентом завершеної роботи 10.05.2024 р.

Календарний план
виконання курсової роботи

№ з/п	Назва етапу роботи	Строк виконання	Відмітка про виконання
1	З’ясування загальної постановки завдання; розробка чернетки першого розділу пояснювальної записки		
2	Програмна реалізація, налагодження та тестування додатка; розробка чернетки пояснювальної записки (другий розділ, висновки, список використаних джерел, додатки).		
3	Остаточне налагодження та тестування додатка; розробка чернетки пояснювальної записки (вступ, другий розділ, висновки, список використаних джерел, додатки).		
4	Розробка остаточного варіанта пояснювальної записки (вступ, другий розділ, висновки, список використаних джерел, додатки)...		

Дата видачі завдання: 2.11.2023 р.

Керівник роботи: Яцишин С.І. _____
(підпис)

Студент: Стасів В. _____
(підпис)

Зміст

Вступ **4**

1 Специфікація роботи **5**

1.1 Основи ООП 5

1.2 Створення класів: поля та методи 8

1.3 Перевантаження операторів 9

1.4 Контейнери 10

1.5 Ітератори 11

2. Програмна документація **13**

2.1 Проектування структури програми 13

2.2 Проектування інтерфейсу користувача 19

2.2.1 Консольний інтерфейс 19

2.2.2 Графічний інтерфейс з використанням Windows Forms 20

2.3 Розроблення консольної версії програми на C++ 22

2.4 Розроблення програми з графічним інтерфейсом на C# 26

Висновки **31**

Список використаних джерел **32**

Додатки **33**

Додаток № 1. Діаграма класів 33

Додаток № 2. Код консольної реалізації програми на мові C++ 34

Додаток № 3. Код реалізації програми з графічним інтерфейсом з використанням Windows Forms на мові C# 57

Вступ

Об’єктно-орієнтоване програмування (ООП) – це як своєрідне мистецтво програмування, де програміст виступає як творець, що ліпить з кубиків конструктора чудові образи – об'єкти. Кожен об'єкт має свою унікальну особистість – дані, що відображають його стан, та методи, що відтворюють його взаємодію зі світом. Так само, як художник використовує пензель для нанесення фарби на полотно, програміст використовує класи для створення об'єктів, які втілюють його ідеї.

У світі ООП об'єкти можуть спілкуватися між собою, обмінюватися інформацією та взаємодіяти, що надає програмі неймовірну гнучкість та потужність. Це, мов кисть художника на полотні, дозволяє створювати програми, які є не просто послідовністю команд, а справжніми шедеврами технологічного мистецтва.

Ціль курсової роботи продемонструвати основи ООП на прикладі програми розробки класу РЕЗУЛЬТАТИ спринтерського забігу, який містить масив об'єктів класу СЕКУНДОМІР.

Програму створено для обліку результатів забігів. Він використовує класи *DigitalCounter*, *Stopwatch* та *RaceResults*, які унаслідовані один від одного. Клас *DigitalCounter*: Лічильник з обмеженнями на значення. Має можливість встановлювати та отримувати максимальне та мінімальне значення, а також перевіряти та змінювати поточне значення. Клас *Stopwatch*: Розширення *DigitalCounter*, що додає можливість вимірювання часу зупинок. Містить функціонал для запуску та зупинки секундоміра. Клас *RaceResults*: Розширення *Stopwatch*, що зберігає результати кількох забігів. Має можливість додавати нові забіги, визначати мінімальний, максимальний та середній час забігу, а також визначати переможців. Підтримує зчитування та запис результатів у файл. Користувач може додавати нові забіги, переглядати рекорди, змінювати значення забігу тощо.

Створено два програмних застосунки, що реалізують поставлене завдання. Перший програмний застосунок має консольний інтерфейс і розроблений на мові програмування C++. Другий програмний застосунок має графічний інтерфейс на мові програмування C# з використанням Windows Forms засобами Visual Studio 2022.

1 Специфікація роботи

Для написання коду на об'єктно-орієнтованому програмуванні потрібно знати основні концепції: класи та об'єкти, конструктори та деструктори, інкапсуляцію, наслідування, поліморфізм, перевантаження методів та операторів, віртуальність, контейнерні класи, ітератори, абстракцію та інше. Ці концепції допомагають створювати структурований, повторно використовуваний та легко зрозумілий код. Коротко опишемо ці поняття.

1.1 Основи ООП

Об'єктно-орієнтоване програмування (ООП) - це парадигма програмування, що базується на використанні класів та об'єктів. Клас визначає структуру об'єкта, включаючи його властивості (дані) та методи (функції).

Основні концепції ООП включають інкапсуляцію, що дозволяє об'єднати дані та методи в одному класі для забезпечення доступу через інтерфейс; наслідування, що дозволяє створювати нові класи на основі існуючих; поліморфізм, який дозволяє об'єктам різних класів виконувати однакові дії; та абстракцію, яка дозволяє визначати класи з концентрацією на їхній функціональності без деталей реалізації. Розрізняють абстракції стану та поведінки об’єкта. Стан характеризується переліком та значенням певних ознак. Поведінка визначається набором операцій, які виконуються об’єктом, або над об’єктом.

ООП сприяє покращенню структури програм, полегшує їх розширення та обслуговування, і дозволяє створювати більш масштабовані та повторно використовувані програми.

Основні переваги ООП:

- Інкапсуляція: Приховання деталей реалізації.
- Успадкування: Наслідування функціональності.
- Поліморфізм: Різна поведінка об'єктів через спільний інтерфейс.
- Абстракція: Визначення класів з фокусом на функціональності.
- Модульність: Розділення програми на незалежні модулі (класи).
- Повторне використання коду: Можливість використовувати класи та об'єкти знову.
- Масштабованість: Легка розширюваність та підтримка складних структур.
- Зручність у роботі з великими проектами: Розподіл завдань між командою програмістів.
- Підтримка паралельного програмування: Полегшує паралельне виконання

різних частин програми.

Інкапсуляція – це концепція об'єктно-орієнтованого програмування, що полягає у прихованні внутрішньої реалізації об'єкта від зовнішнього середовища і доступу до даних та методів лише через визначені інтерфейси. Це дозволяє контролювати доступ до даних та методів класу, захищаючи їх від неправильного використання. Інкапсуляція сприяє зменшенню залежностей між різними частинами програми, полегшує розвиток, збереження та відлагодження коду. Вона також дозволяє змінювати внутрішню реалізацію класу без необхідності змінювати зовнішній інтерфейс, що полегшує підтримку та розвиток програмного забезпечення.

Модифікатори доступу:

Модифікатори доступу – це ключові слова в об'єктно-орієнтованому програмуванні, які визначають рівень доступу до членів класу з інших частин програми. Основні модифікатори доступу включають:

public: Член класу, який визначений з модифікатором *public*, доступний з будь-якої частини програми.

private: Член класу, визначений з модифікатором *private*, доступний лише в межах самого класу і недоступний зовнішнім частинам програми.

protected: Член класу з модифікатором *protected* доступний в межах самого класу і в похідних класах (у випадку успадкування), але недоступний в інших частинах програми.

Ці модифікатори допомагають забезпечити контроль доступу до даних та методів класу, покращуючи безпеку і забезпечуючи внутрішню структуру класу.

Успадкування в об'єктно-орієнтованому програмуванні – це механізм, що дозволяє створювати новий клас на основі вже існуючого, використовуючи його властивості та функціональність. Основні аспекти успадкування:

- Класи та підкласи: Клас, який успадковує функціональність, називається підкласом або нащадком, а клас, властивості якого успадковуються, - базовим класом або батьківським класом.
- Переваги: Успадкування дозволяє використовувати властивості та методи базового класу в підкласі, що спрощує розробку програм, дозволяє уникати дублювання коду та сприяє поліпшенню структури

програми.

- Перевизначення методів: Підклас може перевизначити (або перекрити) методи базового класу, що дозволяє змінювати їх поведінку відповідно до вимог підкласу.
- Зв'язок "є одним із": Успадкування відображає зв'язок "є одним із" між класами, де підклас є конкретною реалізацією базового класу.
- Доступ до конструкторів і деструкторів: При успадкуванні може бути доступ до конструкторів і деструкторів базового класу для ініціалізації та звільнення ресурсів відповідно.

Успадкування сприяє впорядкуванню та розширенню коду, забезпечує поліморфізм та дозволяє створювати ієрархії класів для більшої гнучкості в програмуванні.

Поліморфізм в об'єктно-орієнтованому програмуванні – це можливість об'єктів різних класів виконувати однакові дії або методи, але за різних умов. Це дозволяє викликати однаковий метод для об'єктів різних класів та забезпечує гнучкість інтерфейсів. У кінцевому підсумку, поліморфізм сприяє спрощенню та уніфікації коду.

Абстракція в об'єктно-орієнтованому програмуванні – це процес визначення загальних характеристик об'єктів та ігнорування деталей, які не є необхідними для конкретного контексту. Це означає створення моделі, яка представляє сутність з певного ракурсу, приховуючи складність її реалізації. Абстракція дозволяє розглядати об'єкти на вищому рівні абстракції, що спрощує розробку та забезпечує більшу зрозумілість програмного коду. Це робить код більш масштабованим, підтримуваним і легким для розуміння.

Віртуальні методи – це методи в базовому класі, які можуть бути перевизначені в похідних класах. Вони дозволяють використовувати поліморфізм, тобто викликати методи на об'єктах похідних класів через вказівники або посилання базового класу. При виклику віртуального методу відбувається розділення виконання - виклик відбувається відповідно до типу об'єкта, а не типу вказівника чи посилання. Це дозволяє динамічно визначати

поведінку методу під час виконання програми, що є корисним для створення більш гнучких і розширюваних програм.

1.2 Створення класів: поля та методи

Клас - це шаблон, що використовується для створення об’єктів. Він визначає типи даних і функції, які обробляють ці дані. Об’єкти є екземплярами класу, а ініціалізація їх змінних відбувається в конструкторі. Клас може мати кілька конструкторів, тому він може розглядатися як набір планів для створення об’єкта. Важливо розуміти, що клас - це абстрактна концепція, поки не буде створений об’єкт, який є фізичною реалізацією цього класу в пам’яті комп’ютера. Методи і змінні, які входять до складу класу, називаються його членами. При визначенні класу оголошуються дані, які він містить, і функції, які працюють з цими даними. При створенні класу спочатку вказується ключове слово class. Найпростіші класи можуть мати тільки функції або тільки дані, але в реальних програмах класи включають обидва елементи. Дані зберігаються в змінних екземпляра, які визначені класом, а функції знаходяться в методах. Важливо відзначити, що в C++ визначено декілька специфічних типів членів класу: змінні екземпляра, статичні змінні, константи, методи, конструктори, деструктори, події, оператори і властивості.

Поле – це змінна, яка визначена в класі і зберігає дані, що належать кожному екземпляру класу (об'єкту). Поля представляють стан об'єкта і використовуються для зберігання інформації, яка характеризує конкретний екземпляр класу. Вони можуть мати різні типи даних, такі як цілі числа, рядки, логічні значення або навіть інші об'єкти. Поля можуть бути публічними, приватними або захищеними, відповідно до того, чи доступні вони для зовнішнього світу і для нащадків класу. Вони визначаються в тілі класу і можуть бути ініціалізовані в конструкторі класу або присвоєні значення під час виконання програми. Користувачі класу можуть отримувати доступ до полів для читання або запису, використовуючи методи доступу або, якщо поля публічні, за допомогою прямого доступу до них.

Методи – це функції, що визначаються в класі і призначені для виконання певних операцій з даними цього класу. Вони визначають поведінку об'єктів класу

і надають інтерфейс для взаємодії з ними. Методи можуть змінювати стан об'єкта, отримувати доступ до його даних та взаємодіяти з іншими об'єктами. Вони можуть мати параметри, повертати значення або нічого не повертати (void). Методи можуть бути публічними, приватними або захищеними в залежності від того, які дозволи на доступ до них надаються зовнішнім об'єктам і нащадкам класу. Користувачі класу можуть викликати методи для виконання певних операцій або отримання результатів обробки даних класу.

Коротко кажучи, поля в класі представляють дані або стан об'єкта, тоді як методи - це функції, які виконують операції з цими даними. Поля описують характеристики об'єкта, в той час як методи визначають його поведінку. Разом вони створюють клас, який дозволяє моделювати об'єкти з реального світу та взаємодіяти з ними у програмі.

1.3 Перевантаження операторів

Перевантаження операторів в C++ дозволяє визначати власні варіанти роботи стандартних операторів для користувацьких типів даних. Це дає можливість здійснювати зручні та зрозумілі операції з об'єктами класів, які мають семантику, аналогічну вбудованим типам даних. Перевантаження дозволяє використовувати стандартні оператори, такі як +, -, *, тощо, для виконання певних дій з об'єктами класів. Це спрощує роботу з користувацькими типами даних та підвищує читабельність коду. Перевантаження може бути використане для забезпечення конкретного значення для операцій порівняння, які здійснюються між об'єктами класів, а також для створення спеціальних операцій, які відповідають конкретним потребам додатку. Важливо при перевантаженні операторів дотримуватися звичайних правил мови програмування та забезпечувати правильну семантику для визначених операцій.

Перевантаження операторів в C++ відбувається за допомогою створення спеціальних функцій-членів або глобальних функцій, які визначають бажану поведінку для оператора при роботі з об'єктами класів. Ці функції повинні мати специфічні імена, що вказують на відповідність конкретному оператору.

Наприклад, для перегрузки оператора + для додавання двох об'єктів класу, може бути визначена функція з іменем *operator*+, яка приймає два аргументи відповідного типу і повертає результат відповідно до визначеної логіки.

Таким чином, при використанні операторів з об'єктами класів, компілятор викликає відповідні перегружені функції, які визначають бажану поведінку. Це дозволяє використовувати стандартні оператори з користувацькими типами даних, що значно полегшує роботу з ними та збільшує зрозумілість коду.

Операторна функція-член має наступний вигляд:

Тип ім’я_класу::operator#{список-аргументів){...//Операції}

Функція-член оператора зазвичай повертає об'єкт класу, з яким вона працює. Проте тип, який вона повертає, може варіюватися. Символ # замінюється відповідним оператором. Наприклад, якщо оператор ділення / перевантажується в класі, функція-член оператора називається operator/. У випадку перевантаження унарного оператора список аргументів залишається порожнім, а для бінарного оператора - містить один параметр.

1.4 Контейнери

У C++, контейнери - це структури даних, які дозволяють зберігати колекції об'єктів. Вони забезпечують різні способи організації та управління даними, такі як додавання, вилучення та доступ до елементів. Контейнери можуть бути реалізовані у вигляді стеку, черги, вектора, списку, мапи та інших структур. В C++ стандартна бібліотека містить широкий вибір готових контейнерів, а також можливість створення власних контейнерів за допомогою шаблонів. Користувач може вибрати контейнер, що найкраще відповідає конкретним потребам програми з урахуванням швидкодії, об'єму даних та операцій, які необхідно виконати.

Основні контейнери в C++ включають вектор (vector), списки (list), множини (set), асоціативні масиви (map), стеки (stack) та черги (queue).

- Вектор (vector): Динамічний масив, що забезпечує швидкий доступ до елементів та підтримує автоматичне збільшення розміру при необхідності.

- Списки (list): Двоспрямований зв'язаний список, який дозволяє додавати та видаляти елементи в будь-якому місці зі складністю O(1).
- Множини (set): Колекція унікальних елементів, яка автоматично сортується в зростаючому порядку.
- Асоціативні масиви (map): Колекція пар ключ-значення, де кожен ключ є унікальним, та автоматично сортується за ключами.
- Стеки (stack): Контейнер, що працює за принципом "Last In, First Out" (LIFO), де елементи додаються та вилучаються з кінця.
- Черги (queue): Контейнер, що працює за принципом "First In, First Out" (FIFO), де елементи додаються в кінець, а вилучаються з початку.

Кожен з цих контейнерів має свої унікальні особливості та використовується в залежності від конкретних потреб програми.

1.5 Ітератори

Ітератори - це об'єкти, які ведуть себе більш-менш подібно вказівниками. Вони надають можливість виконувати циклічну обробку елементів контейнера - подібно до того, як використовується покажчик для організації циклу по масиву.

Існує п'ять типів ітераторів.

Ітератор	**Тип доступу**
Довільний доступ BiIter	Зберігає і витягує значення. Доступ до елементів - в довільному порядку
Двонаправлений ForIter	Зберігає і витягує значення. Допускає переміщення вперед і назад
Прямий InIter	Зберігає і витягує значення. Переміщення тільки вперед
Вхідний OutIter	Витягує, але не зберігає значення. Переміщення тільки вперед
Вихідний RandIter	Зберігає, але не витягує значення. Переміщення тільки вперед

Ітератори в C++ та C# - це інструменти, які дозволяють отримувати доступ до елементів послідовностей даних, таких як масиви, списки або колекції. Ось короткий огляд ітераторів у кожній з цих мов:

C++:

- Iterator-класи: В C++ ітератори зазвичай реалізовані у вигляді класів, які надають інтерфейс для роботи з колекціями даних. Ці класи

зазвичай мають методи для отримання значень, переходу до наступного або попереднього елементу, а також для перевірки кінця колекції.

- Типи ітераторів: В C++ існують різні типи ітераторів, які відрізняються за своєю функціональністю та можливостями. Найпоширеніші типи ітераторів включають випадковий доступ, прямий доступ, ввод та вивід ітератори.
- Оператори: Ітератори в C++ зазвичай підтримують оператори інкременту (++) та декременту (--), щоб здійснювати переходи між елементами колекції. Крім того, ітератори можуть підтримувати різні арифметичні операції для переміщення на задану кількість елементів вперед або назад.
- Діапазони: Введення концепції діапазонів у C++11 спрощує роботу з ітераторами. Діапазони дозволяють більш зручний та елегантний синтаксис для роботи з колекціями даних та дозволяють використовувати стандартні алгоритми більш ефективно.

C#:

- IEnumerator і IEnumerable: У C# ітератори часто реалізовані за допомогою інтерфейсів IEnumerator та IEnumerable. IEnumerator дозволяє послідовно отримувати доступ до елементів колекції, а IEnumerable дозволяє створювати ітератори для об'єктів, які можна перебирати.
- foreach цикл: У C# для роботи з ітераторами часто використовується foreach цикл, який автоматично керує ітератором та обробляє всі елементи колекції. Це робить роботу з ітераторами більш простою та зручною.
- LINQ: У мові C# ітератори є ключовою складовою Language-Integrated Query (LINQ), яка дозволяє виконувати складні операції фільтрації, сортування, групування та обробки даних виразами запитів, які вбудовані безпосередньо в мову програмування. LINQ дозволяє

працювати з різноманітними джерелами даних, такими як масиви, колекції, бази даних та XML, використовуючи зручний та експресивний синтаксис запитів.

- Parallel LINQ (PLINQ): Додатковим розширенням LINQ є Parallel LINQ (PLINQ), який дозволяє виконувати операції LINQ паралельно, використовуючи всі доступні ядра процесора. Це дозволяє оптимізувати продуктивність програм та прискорювати обробку великих обсягів даних шляхом розпаралелювання обчислень.
- Async LINQ (LINQ to Async): Ще одним розширенням LINQ є Async LINQ, який дозволяє виконувати асинхронні операції обробки даних. Це особливо корисно для роботи з великими обсягами даних або в мережевих додатках, де асинхронність є ключовою функцією для ефективності та продуктивності.
- Ітератори в мовах програмування C++ та C# є потужними засобами для роботи з колекціями даних, дозволяючи здійснювати різноманітні операції з елементами колекцій та спрощуючи розробку програмних рішень з використанням лаконічного та експресивного синтаксису.

2. Програмна документація

2.1 Проектування структури програми

Проектування структури програми ґрунтується на визначенні структур класів, операцій, даних та методів їх опрацювання, семантики потоків повідомлень.

Розв’язання задачі почнемо з виявлення понять класів і їх фундаментальних взаємозв'язків.

Основним завдання є створення класу РЕЗУЛЬТАТИ спринтерського забігу (у коді – `RaceResults`), який містить масив об’єктів класу СЕКУНДОМІР (у коді – `Stopwatch`), який в свою чергу походить від класу ЛІЧИЛЬНИК (у коді – `DigitalCounter`). Отже об’єкт типу `RaceResults` повинен містити колекцію об’єктів типу `Stopwatch`.

Для роботи з даними класами у програмі яка писалася на мові програмування

C++, потрібно було створити власний контейнерний клас. Я вирішив створити власний вектор, який назвав `MyVector`. Даний клас є шаблонним, для того щоб можна було працювати з різними типами даних. Він має три поля: T* array – це вказівник на динамічний масив елементів; size_t capacity – це поточна ємність вектора, тобто максимальна кількість елементів, яку він може зберігати без виділення додаткової пам'яті; size_t Size – це поточний розмір вектора, іншими словами кількість елементів, що зберігаються у векторі. Ось конструктор:

```
MyVector() : array(nullptr), capacity(0), Size(0) {}
```

Цей конструктор ініціалізує вектор з нульовою ємністю і нульовим розміром. array встановлюється в nullptr, оскільки на початку немає виділеної пам'яті для зберігання елементів. Також реалізований деструктор:

```
~MyVector()
{
    delete[] array;
}
```

Деструктор звільняє пам'ять, виділену під динамічний масив, щоб уникнути витоків пам'яті. Пам’ять видаляється після того, як закінчується область видимості функції.

Ось реалізований метод push_back:

```
void push_back(const T& element)
 {
     if (Size >= capacity)
     {
         capacity = (capacity == 0) ? 1 : capacity * 2;
         T* newArray = new T[capacity];
         for (size_t i = 0; i < Size; ++i)
         {
             newArray[i] = array[i];
         }
         delete[] array;
         array = newArray;
     }
     array[Size++] = element;
     }
```

Даний метод додає новий елемент у кінець вектора. Ось як це працює: Спочатку перевіряємо чи є достатньо місця для нового елемента (Size >= capacity). Якщо ні, то:

- Подвоюємо поточну ємність (capacity). Якщо capacity була нульовою, встановлюємо її в 1.
- Створюємо новий масив з новою ємністю.

- Копіюємо старі елементи у новий масив.
- Звільняємо стару пам'ять.
- Присвоюємо вказівнику array адресу нового масиву.

І на останок, при будь якій умові, додаємо новий елемент і збільшуємо розмір вектора (Size++).

Ось перевантажений оператор квадратних дужок []:

```
T& operator[](size_t index)
{
    if (index >= Size)
    {
        throw out_of_range("Індекс вийшов за межі видимості");
    }
    return array[index];
}
```

Цей оператор дозволяє отримати доступ до елемента за заданим індексом. Він також перевіряє, чи не виходить індекс за межі допустимих значень, і в разі порушення кидає виключення out_of_range.

Залишились методи *size*, *empty*, *begin* та *end*. Метод *size* повертає поточний розмір вектора (кількість збережених елементів). Метод *empty* повертає true, якщо вектор порожній (не містить жодного елемента), і false в іншому випадку. Методи begin та end повертають вказівники на початок і кінець вектора відповідно. Вони можуть бути використані наприклад для ітерації по елементах вектора. Я використав їх для пошуку мінімального та максимального значень забігу, та для сортування.

З вектором закінчили, тепер перейдемо до базового класу `DigitalCounter`. У ньому ми маємо три приватних поля: поточне значення, мінімальне та максимальне. Оскільки дані поля є приватними, ми повинні організувати для них гетери та сетери, до яких я повернуся пізніше. Також я описав всі конструктори, які розуміється мають модифікатор доступу public: за замовчуванням `DigitalCounter()`, параметризований `DigitalCounter(int value, int max_value, int min_value)`, а також конструктори копіювання `DigitalCounter(const DigitalCounter& counter)` та переміщення `DigitalCounter(DigitalCounter&& counter) noexcept`. Ось всі решта методів, які є в даному класі (всі вони є публічними):

```
void setMaxValue(int max_value);
```

```
int getMaxValue();
void setMinValue(int min_value);
int getMinValue();
void setValue(int val);
int getValue();
virtual void printValues(); // віртуальний метод виведення інформації
void operator()(int val, int max_val, int min_val);
DigitalCounter operator++();
DigitalCounter operator--();
DigitalCounter operator+=(int val);
DigitalCounter operator-=(int val);
bool operator==(const DigitalCounter& other);
DigitalCounter& operator=(const DigitalCounter& counter);
DigitalCounter& operator=(DigitalCounter&& counter) noexcept;
friend istream& operator>>(istream& is, DigitalCounter& counter);
friend ostream& operator<<(ostream& os, const DigitalCounter counter);
```

Як ви вже помітили, у моїй програмі потрібно було реалізувати перевантаження операторів. Ці перевантажені оператори забезпечують зручну роботу з об'єктами *DigitalCounter*, дозволяючи виконувати звичні операції (додавання, віднімання, інкремент, декремент, порівняння, присвоєння, введення та виведення) з врахуванням обмежень (мінімальне і максимальне значення) та забезпечуючи безпечне управління ресурсами (наприклад, уникнення витоків пам'яті).

Наступним класом про який я розкажу є `Stopwatch`. Він є нащадком класу `DigitalCounter` та наслідується з модифікатором доступу public. Це означає, що всі поля і методи з батьківського класу будуть мати ті ж сам модифікатори доступу й в нащадку, одна до полів з модифікатором доступу private ми доступитися на пряму ніяк не зможемо. В класі `Stopwatch` ми додаємо нове поле типу bool під назвою running. Воно відповідатиме в нас за те включений секундомір чи ні. Для цього класу я також зробив конструктор за замовчування `Stopwatch()` :

`DigitalCounter()` та з параметрами `Stopwatch(int value, int max_value, int min_value, bool running) : DigitalCounter(value, max_value, min_value).`

Ось всі інші методи даного класу (всі вони також з модифікатором доступу public):

```
void start();
void stop();
bool isRunning();
void increaseValue();
void decreaseValue();
void increaseValueBy(int increment);
void decreaseValueBy(int decrement);
void assignValue(int newValue);
void printValues() override; // перевизначення віртуального метода виведення інформації
friend ostream& operator<<(ostream& os, Stopwatch& stopwatch);
```

Серед завдань є одне особливо цікаве: «У межах ієрархії класів побудувати поліморфічний кластер на основі віртуального методу виведення поточного значення лічильника або часу вимкнення секундоміра. Продемонструвати механізм пізнього зв'язування.»

Поліморфічний кластер - це колекція об'єктів різних типів, які спадають від одного базового класу та можуть бути використані у коді як об'єкти базового класу. Основна ідея полягає в тому, що код може працювати з об'єктами різних типів, використовуючи їх спільний інтерфейс. Поліморфізм дозволяє об'єктам різних класів вести себе по-різному, навіть якщо вони використовують однакові методи. При виклику віртуального методу базового класу для об'єкта підкласу викликається конкретна реалізація цього методу для цього підкласу. Це називається пізнім зв'язуванням.

Пізнє зв'язування створюється за допомогою ключового слова virtual, яке вказує на те, що метод базового класу може бути перевизначений в похідних

класах. Під час виклику віртуального методу для об'єкта посилання або вказівника базового класу викликається реалізація цього методу для конкретного типу об'єкта, який знаходиться в пам'яті. Таким чином, поліморфічний кластер разом з пізнім зв'язуванням дозволяє створювати гнучкий та розширюваний код, який може працювати з різними типами об'єктів, використовуючи їх загальний інтерфейс і викликаючи конкретну реалізацію методів для кожного типу об'єкта.

Так ось. В базовому класі `DigitalCounter` я оголосив віртуальну функцію виведення поточного значення лічильника `virtual void printValues()`. В класі `Stopwatch` я перевизначив даний метод та додав туди виведення стану секундоміра. Пізнє зв’язування буде продемонстроване вже в самій програмі.

А ось і останній клас даній програмі: `RaceResults`. У ньому всього одне поле, це вектор об’єктів класу `Stopwatch`. У ньому конструктор не реалізований, так як на це немає ніякої необхідності. При створенні об’єкта класу він сам заповнить його так званим «мусором», який я вже пізніше перезаповню. Ось всі реалізовані методи цього класу:

```
void addRace(Stopwatch race);

Stopwatch& getRace(int index);

int getSize();

int minTime();

int maxTime();

double averageTime();

MyVector<int>& winners();

void readFromFile(const string& filename);

void writeToFile(const string& filename);
```

Також дуже важливо передбачити оброблення помилок, які можуть виникнути під час виконання програми. Обробка помилок – це процес

управління помилками, який дозволяє програмам коректно реагувати на непередбачені ситуації або помилки в ході виконання. Основна мета обробки помилок - забезпечити відновлення програми в нормальний робочий стан або коректне завершення її роботи в разі виникнення помилки. Найкраще для цього підходить конструкція *try-catch*. Він працює дуже просто:

- Код, який може викликати виключення, поміщується всередину блоку try.
- Якщо під час виконання блоку try виникає виключення, програма переходить до блоку catch, який відповідає типу винятку.
- У відповідному блоку catch можна виконати необхідні дії для обробки або виведення інформації про помилку.
- Після виконання блоку catch, програма продовжить виконання з місця, де виникло виключення.

Ця конструкція дозволяє програмі бути більш надійною та стійкою до помилок, оскільки вона може відповідно реагувати на помилки, які можуть виникати під час виконання коду.

2.2 Проектування інтерфейсу користувача

2.2.1 Консольний інтерфейс

Розглянемо спочатку проектування програми з консольним інтерфейсом. У завданні вказано, що мають бути перевантажені оператори потокового вводу і виводу для певних класів, а це можливо лише в для консольних програм на C++.

Для зручності роботи з даною програмою я розробив можливість працювати через пункти меню. Пункти цього меню мають надавати певний функціонал користувачу і водночас забезпечити демонстрацію поставлених перед нами підзадач. Ось як вигладає моє головне меню:

```
cout << "Меню:" << endl;
cout << "1. Додати результати забігу" << endl;
cout << "2. Показати рекорди" << endl;
cout << "3. Вивести інформацію про забіг за його номером" << endl;
cout << "4. Змінити значення забігу" << endl;
cout << "5. Зберегти результати у файл" << endl;
cout << "6. Зчитати результати з файлу" << endl;
cout << "7. Вийти" << endl;
cout << "Виберіть опцію: ";
```

За допомогою першого пункту можна додавати результати забігу в створений в програмі вектор. Для того, щоб додати результат, потрібно просто ввести цілочисельне значення від 0 до 59. Якщо буде введено будь-що інше, тоді виведеться повідомлення про помилку.

Пункт меню № 2 дозволяє переглянути результати спринтерського забігу, точніше переглянути результати трьох переможців.

Натиснувши 3, ви зможете переглянути інформацію про забіг за його номером. У вас буде можливість вибрати: вивести інформацію із значенням стану секундоміра чи без нього. Якщо вибрати ні, то просто виведеться мінімально й максимально можливе значення, та поточне значення забігу.

Пункт № 4 дозволяє змінювати значення вже створених результатів. Ось які є можливості:

```
cout << "1. Збільшити значення забігу на 1" << endl;
cout << "2. Зменшити значення забігу на 1" << endl;
cout << "3. Збільшити значення забігу на вказане число" << endl;
cout << "4. Зменшити значення забігу на вказане число" << endl;
cout << "5. Присвоїти нове значення забігу" << endl;
cout << "Виберіть опцію: ";
```

Після того як ви виберете що ж саме робити, потрібно буде ввести індекс забігу, який ви збираєтесь змінювати. Наголошую, що саме ***індекс***, а не ***номер***.

Пункт № 5-6 надає можливість записувати результати до файлу, або ж зчитувати їх.

І останній, пункт № 7 завершує роботу програми.

Ці пункти меню надають користувачу необхідний функціонал і водночас задовольняють поставлені перед нами завдання про демонстрацію певних механізмів реалізації програмного коду.

2.2.2 Графічний інтерфейс з використанням Windows Forms

Друга версія програми повинна мати графічний інтерфейс, який будемо проекувати з використанням Windows Forms. Програма з графічним інтерфейсом складається з однієї головної форми.

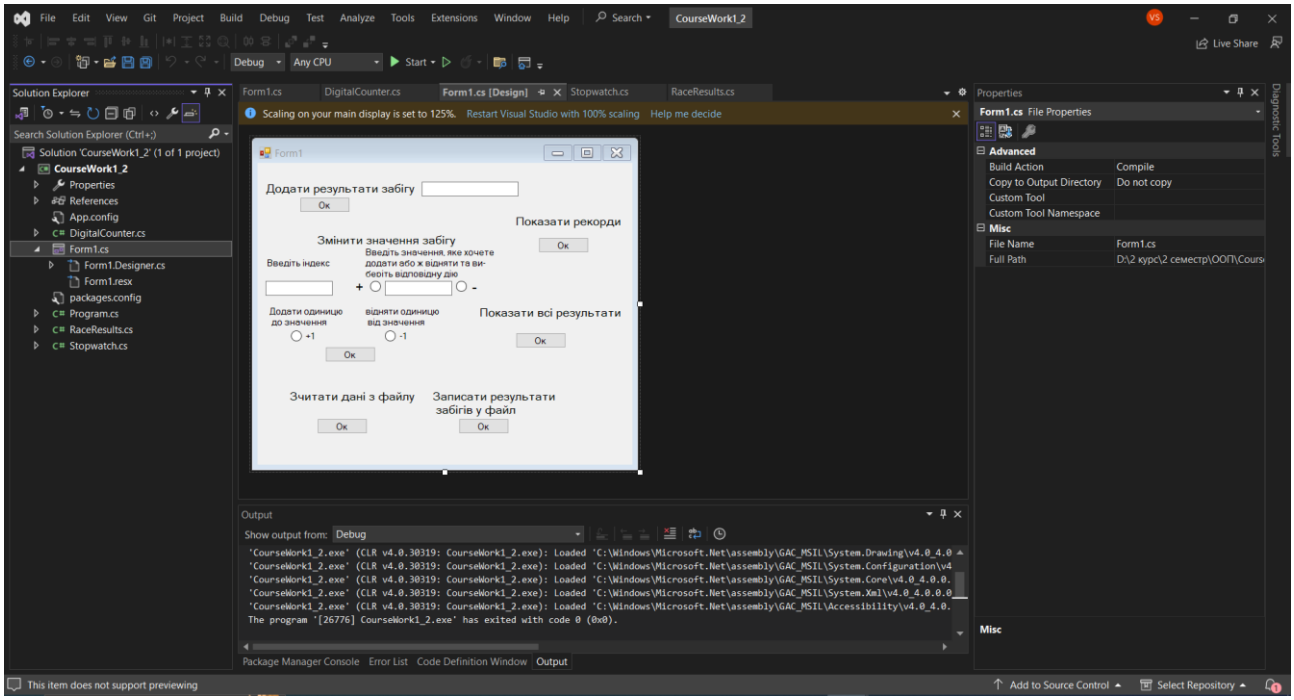
Windows Forms (WinForms) - це бібліотека для створення графічного інтерфейсу користувача (GUI) в додатках на платформі .NET. Ось основні аспекти:

- Форма (Form): Базове вікно додатка, яке містить інтерфейсні елементи.
- Контроли (Controls): Елементи інтерфейсу, такі як кнопки (Button), текстові поля (TextBox), мітки (Label) тощо. Їх можна додавати на форму візуально або програмно.
- Події (Events): Механізм, який дозволяє реагувати на дії користувача, наприклад, натискання кнопки. Обробники подій виконують певний код у відповідь на подію.
- Обробка подій:

Наприклад, для кнопки можна створити обробник події натискання.

- Візуальний дизайнер: Інструмент у Visual Studio, який дозволяє перетягувати контролі на форму та налаштовувати їх властивості.

WinForms надає простий спосіб створення настільних додатків з графічним інтерфейсом, підтримуючи розробку як візуально, так і через код.



На ній у нас є можливість додати результат забігу, змінити значення вибраного за індексом забігу. Значення можна змінити додавши чи віднявши

одиницю за допомогою двох кнопочок, з яких можна вибрати тільки одну. Або ж можна додати чи відняти якесь конкретне значення, ввівши його в текстове поле та вибравши дію. Дію можна вибрати також за допомогою двох кнопок, які знаходяться по обидві сторони від текстового поля.

Також є можливість переглянути рекорди, тобто трьох переможців, або ж подивитися на всі результати забігів. І на останок є можливість записати результати в файл, або ж зчитати дані з файлу. Для цього в коді я використовував сереалізацію та десеріалізацію.

2.3 Розроблення консольної версії програми на C++

Після створення плану структури програми, розробки її екранних форм, елементів діалогу, графічних схем класів та відношень між ними ми проведемо деталізацію складових частин програми. Ці компоненти повинні виконувати розв’язання поставленої задачі та забезпечувати взаємодію з користувачем через спроектований інтерфейс. Для розробки цих складових частин необхідні знання мов програмування, таких як C++ та C#. Також важливо розуміти підходи до проектування програмного забезпечення з графічним інтерфейсом, зокрема використання Windows Forms у середовищі розробки Visual Studio 2022. У результаті роботи було створено дві програми, які вирішують поставлену задачу. Перша версія програми розроблена на мові C++ з консольним інтерфейсом, а друга версія має графічний інтерфейс і реалізована на мові C# з використанням Windows Forms.

Повернемося до програми розробленої за допомогою мови C++. Ось які функції я створив для полегшення роботи з пунктами меню:

Функція для додавання результатів забігу в вектор:

```
void startNewRace(RaceResults& raceResults)
{
   Stopwatch newRace;
   newRace.start();
   cout << "Запишіть результат: " << endl;
```

```
    int result;
    cin >> result;
    try
    {
        if (result < newRace.getMinValue() || result > newRace.getMaxValue())
        {
            throw out_of_range("Невірне значення");
        }
        else if (cin.fail())
        {
            throw invalid_argument("Введено неправильний тип даних");
        }
        else
        {
            newRace.setValue(result);
            newRace.stop();
            raceResults.addRace(newRace);
        }
    }
    catch (const out_of_range& e)
    {
        cerr << "Помилка: " << e.what() << endl;
    }
    catch (const invalid_argument& e)
    {
        cerr << "Помилка: " << e.what() << endl;
    }
}
```

Ось функція для виведення рекорду, тобто перших трьох переможців:

```
void displayRecords(RaceResults& raceResults)
{
    cout << "Рекорди:" << endl;
    cout << "Мінімальний час: " << raceResults.minTime() << endl;
    cout << "Максимальний час: " << raceResults.maxTime() << endl;
    cout << "Середній час: " << raceResults.averageTime() << endl;
    cout << "Перші три переможці: ";
    MyVector<int> winners = raceResults.winners();
    for (int time : winners)
    {
        cout << time << " ";
    }
    cout << endl;
}
```

Функція для зміни значень для результатів:

```
void changeValue(RaceResults& raceResults)
{
    cout << "1. Збільшити значення забігу на 1" << endl;
    cout << "2. Зменшити значення забігу на 1" << endl;
    cout << "3. Збільшити значення забігу на вказане число" << endl;
    cout << "4. Зменшити значення забігу на вказане число" << endl;
    cout << "5. Присвоїти нове значення забігу" << endl;
    cout << "Виберіть опцію: ";
    unsigned choice;
    cin >> choice;
    int index;
    cout << "Введіть індекс забігу, який бажаєте змінити: ";
    cin >> index;
    switch (choice)
    {
    case 1:
        if (index >= 0 && index < raceResults.getSize())
        {
            Stopwatch& race = raceResults.getRace(index);
            race.increaseValue();
        }
        else
        {
            cout << "Невірний індекс." << endl;
        }break;
    case 2:
        if (index >= 0 && index < raceResults.getSize())
        {
            Stopwatch& race = raceResults.getRace(index);
            race.decreaseValue();
        }
        else
        {
            cout << "Невірний індекс." << endl;
        }break;
    case 3:
        if (index >= 0 && index < raceResults.getSize())
        {
            int i;
            cout << "Введіть значення: ";
            cin >> i;
            Stopwatch& race = raceResults.getRace(index);
            race.increaseValueBy(i);
        }
        else
        {
            cout << "Невірний індекс." << endl;
        }break;
    case 4:
        if (index >= 0 && index < raceResults.getSize())
        {
            int i;
            cout << "Введіть значення: ";
```

```
            cin >> i;
            Stopwatch& race = raceResults.getRace(index);
            race.decreaseValueBy(i);
        }
        else
        {
            cout << "Невірний індекс." << endl;
        }break;
    case 5:
        if (index >= 0 && index < raceResults.getSize())
        {
            int i;
            cout << "Введіть значення: ";
            cin >> i;
            Stopwatch& race = raceResults.getRace(index);
            race.assignValue(i);
        }
        else
        {
            cout << "Невірний індекс." << endl;
        }break;
    default:  cout << "Помилка при виборі пункту меню." << endl; break;
    }
}
```

Та функція для виведення інформації про забіг за його індексом:

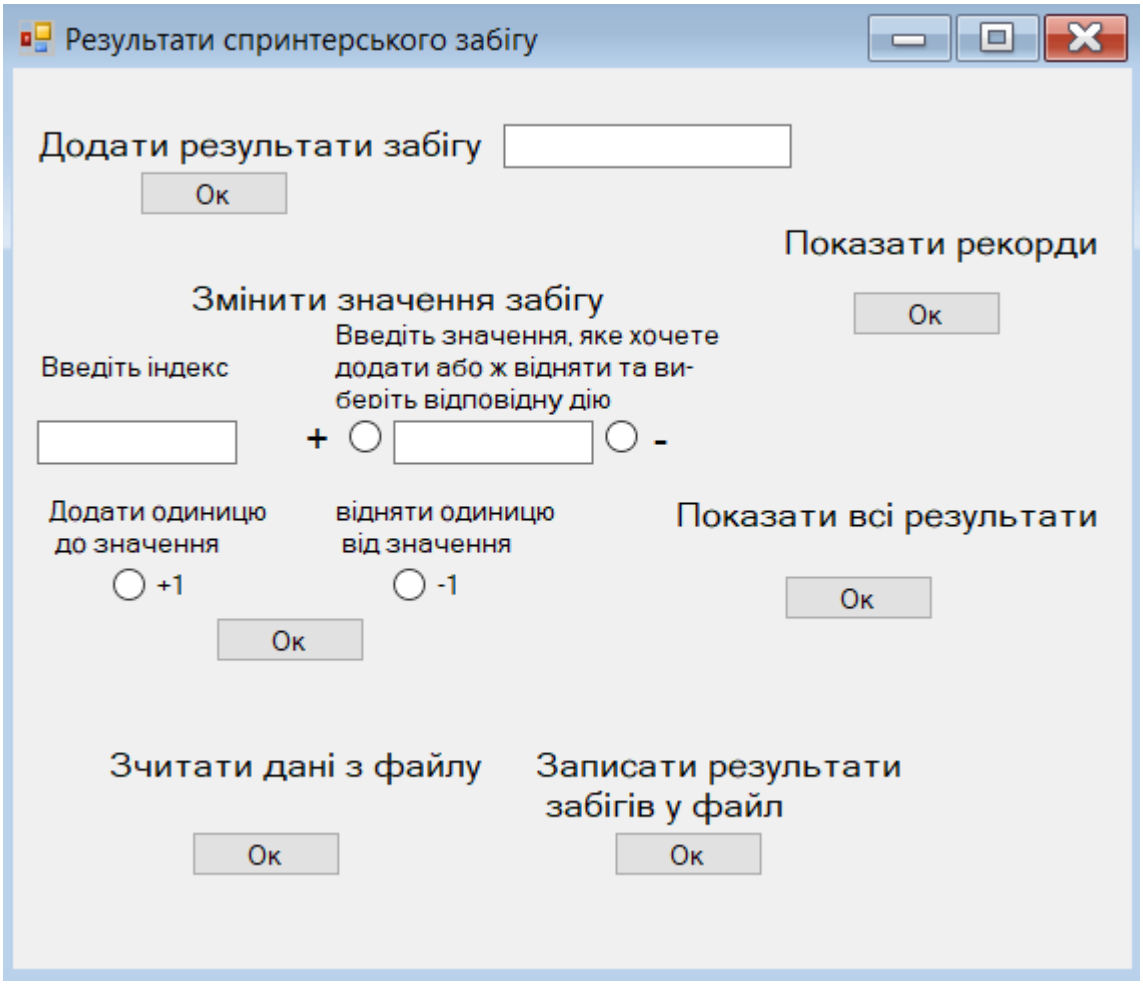
```
void displayCurrentValue(RaceResults& races, size_t index)
{
    index -= 1;
    if (index < races.getSize())
    {
        try
        {
            cout << "Вивести інформацію разом зі станом секундоміра?\n1. Так\n2. Ні" <<
endl;
            unsigned choice;
            cin >> choice;
            if (cin.fail() || (choice < 1 && choice > 2))
            {
                throw exception("Помилка при введенні значення.");
            }
            else if (choice == 1)
            {
                Stopwatch& race = races.getRace(index);
                displayValues(race);
            }
            else if (choice == 2)
            {
                DigitalCounter& race = races.getRace(index);
                DigitalCounter race1 = race;
```

```
            displayValues(race1);
        }
    }
    catch (exception& e)
    {
        cerr << "Помилка: " << e.what() << endl;
    }
  }
  else
  {
    cout << "Невірний індекс." << endl;
  }
}
```

Розглядати методи створених класів ми не будемо, адже їх дуже багато, тому переглянути їх можна буде пізніше у додатку № 2.

2.4 Розроблення програми з графічним інтерфейсом на C#

Давайте розглянемо розробку програми на мові C# з використанням Windows Forms. На рисунку нижче ви можете побачити графічний інтерфейс програми, який був спроектований за допомогою Visual Studio 2022. Кожен елемент керування графічним інтерфейсом може генерувати відповідні події в залежності від дій користувача. Ці згенеровані події можуть бути пов’язані з відповідними методами для їх обробки, які реалізовані у відповідних класах.



Перейдемо до реалізації методів обробки подій, які генерують елементи керування графічного інтерфейсу при зміні свого стану або при роботі користувача з ними. Ці методи, реалізовані в класі форми public partial class Form1 : Form.

Розглянемо метод обробки натиснення кнопки для додавання результату до спринтерського забігу:

```
private void btn_addResult_Click(object sender, EventArgs e)
{
   try
   {
      Stopwatch newRace = new Stopwatch();
      newRace.Start();
      int result;
      if (int.TryParse(resultTextBox.Text, out result))
      {
         if (result >= newRace.Min_value && result <= newRace.Max_value)
         {
            newRace.Value = result;
            newRace.Stop();
            raceResults.AddRace(newRace);
            resultTextBox.Clear();
         }
         else
         {
```

```
            MessageBox.Show($"Некоректне значення результата. Значення повинно бути в межах від
{newRace.Min_value} до {newRace.Max_value}", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
      }
      else
      {
        MessageBox.Show("Некоректне значення результата", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        resultTextBox.Clear();
      }
    }
    catch (Exception ex)
    {
      MessageBox.Show("Сталася помилка: " + ex.Message, "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
      resultTextBox.Clear();
    }
}
```

У ньому оброблені всі можливі помилки та організоване їх виведення для розуміння що пішло не так.

Ось метод оброблення кнопки для зміни значень результатів. У ньому також оброблені помилки, які можуть виникати та активно використовуються перевантажені оператори для зручності роботи із класами:

```
private void btn_change_Click(object sender, EventArgs e)
{
    int index;
    if (int.TryParse(indexTextBox.Text, out index))
    {
      try
      {
        if (index >= 0 && index < raceResults.GetSize())
        {
          Stopwatch race = raceResults.GetRace(index);
          int valueToAdd = 0;
          if (addRadioButton.Checked)
          {
            race++;
            addRadioButton.Checked = false;
          }
          else if (subtractRadioButton.Checked)
          {
            race--;
            subtractRadioButton.Checked = false;
          }
          else
          {
            if (!int.TryParse(valueTextBox.Text, out valueToAdd))
            {
              MessageBox.Show("Некоректне значення для додавання або віднімання", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
              return;
            }
          }
```

```
            if (plusRadiobtn.Checked)
            {
                race += valueToAdd;
                plusRadiobtn.Checked = false;
            }
            else if (minusRadiobtn.Checked)
            {
                race -= valueToAdd;
                minusRadiobtn.Checked = false;
            }
            raceResults.SetRace(index, race);
            indexTextBox.Text = "";
            valueTextBox.Text = "";
        }
        else
        {
            MessageBox.Show("Некоректний індекс забігу", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
    }
    catch (ArgumentOutOfRangeException ex)
    {
        MessageBox.Show("Помилка: " + ex.Message, "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    }
  }
  else
  {
    MessageBox.Show("Некоректний індекс забігу", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
  }
}
```

Наступним буде метод для перегляду топ 3 серед спринтерського забігу:

```
private void btn_showRecord_Click(object sender, EventArgs e)
{
  StringBuilder records = new StringBuilder();
  records.AppendLine("Рекорди:");
  records.AppendLine("Мінімальний час: " + raceResults.MinTime());
  records.AppendLine("Максимальний час: " + raceResults.MaxTime());
  records.AppendLine("Середній час: " + raceResults.AverageTime());
  records.Append("Перші три переможці: ");
  List<int> winners = raceResults.Winners();
  for (int i = 0; i < winners.Count; i++)
  {
    records.Append(winners[i]);
    if (i < winners.Count - 1)
    {
      records.Append(", ");
    }
  }
  MessageBox.Show(records.ToString(), "Рекорди", MessageBoxButtons.OK,
MessageBoxIcon.Information);
}
```

Також ось такий метод, для виведення всієї інформації:

```
private void btn_showAll_Click(object sender, EventArgs e)
{
  raceResults.PrintAllResults();
}
```

Також покажу метод PrintAllResults для кращого розуміння:

```
public void PrintAllResults()
 {
    string message = "Усі значення забігів:\n\n";

    foreach (var race in races)
    {
       message += $"Значення: {race.Value}, Максимальне значення: {race.Max_value}, Мінімальне
значення: {race.Min_value}, Стан: {(race.Running ? "Включено" : "Виключено")}\n";
    }

        MessageBox.Show(message, "Усі результати", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
 }
```

І на останок методи для зчитування інформації з файлу та запису у нього. Використовував я для цього серіалізацію та десеріалізацію:

Ось метод де робиться серіалізація:

```
  public void WriteToFile(string filename)
  {
     string jsonString = JsonSerializer.Serialize(races);
     File.WriteAllText(filename, jsonString);
  }
```

Та сам метод натиснення кнопки:

```
 private void btn_writeToFile_Click(object sender, EventArgs e)
 {
    SaveFileDialog saveFileDialog = new SaveFileDialog();
    try
    {
       if (saveFileDialog.ShowDialog() == DialogResult.OK)
       {
          raceResults.WriteToFile(saveFileDialog.FileName);
          MessageBox.Show("Результати збережено у файл", "Успіх", MessageBoxButtons.OK,
MessageBoxIcon.Information);
       }
    }
    catch (Exception ex)
    {
          MessageBox.Show("Помилка при збереженні у файл: " + ex.Message, "Помилка",
          MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
 }
```

А ось і десереалізація:

```
 public void ReadFromFile(string filename)
 {
    string jsonString = File.ReadAllText(filename);
```

```
    races = JsonSerializer.Deserialize<List<Stopwatch>>(jsonString);
}
```

І метод для натиснення кнопки:

```
private void btn_readFromFile_Click(object sender, EventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog();
    try
    {
        if (openFileDialog.ShowDialog() == DialogResult.OK)
        {
            raceResults.ReadFromFile(openFileDialog.FileName);
            MessageBox.Show("Результати зчитано з файлу", "Успіх", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Помилка при зчитуванні з файлу: " + ex.Message, "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Висновки

У даній курсовій роботі ми розглянули розробку програми на мовах C++ та C# із використанням принципів об’єктно-орієнтованого програмування (ООП). Основними класами, що були розроблені та використовувались у програмі, стали DigitalCounter, Stopwatch та RaceResults.

Клас DigitalCounter забезпечив базову функціональність лічильника з можливістю задавати мінімальні та максимальні значення, а також інтерфейс для збільшення та зменшення значення лічильника з обробкою можливих виключень. Це дозволило створити гнучкий та надійний базовий компонент для побудови більш складних класів.

Клас Stopwatch, наслідуючи DigitalCounter, розширив його функціональність до секундоміра з можливістю запуску та зупинки. Це додало можливість відслідковувати стан секундоміра (працює чи ні), а також додаткові методи для збільшення та зменшення значення, зберігаючи при цьому всі переваги базового класу.

Клас RaceResults дозволив організувати колекцію секундомірів, додавати нові результати, визначати мінімальний, максимальний та середній час перегонів, а також зберігати та завантажувати дані з файлу. Це дало змогу наочно продемонструвати зберігання та обробку результатів змагань.

При розробці даної програми було використано такі основні принципи ООП як наслідування, інкапсуляція, поліморфізм та абстракція. Інкапсуляція

дозволила сховати деталі реалізації класів та надавати чітко визначений інтерфейс для взаємодії з ними. Наслідування забезпечило можливість створювати нові класи на основі існуючих, розширюючи або змінюючи їхню поведінку. Поліморфізм дозволив використовувати базові типи для роботи з об’єктами похідних класів, забезпечуючи гнучкість та зручність при розширенні функціональності. Також у проекті була реалізована обробка помилок з використанням механізмів виключень, що дозволило зробити програму більш надійною та захищеною від некоректного вводу даних.

В цілому, дана курсова робота продемонструвала застосування ООП для розробки практичної програми з використанням C++ та C#. Завдяки цьому, ми змогли створити модульну, розширювану та надійну програму, що відповідає сучасним стандартам програмування.

Список використаних джерел

1. Яцишин С.І. Конспект лекцій з курсу «Об’єктно-орієнтоване програмування», 2024 рік [Електронний ресурс].

Для роботи на C++

1. Б. Страуструп, "Мова програмування C++", 4-е видання, Аддісон-Везлі, 2013.
2. С. Майерс, "Ефективне використання C++. 55 вірних порад покращення ваших програм та проєктів", Вільямс, 2007.
3. Б. Еккель, "Філософія C++. Вступ до стандартного C++", Вільямс, 2006.
4. Офіційна документація по C++ на сайті ISO C++: https://isocpp.org/
5. J. Liberty, "Programming C++", O'Reilly Media, 2002.

6. Офіційна документація по стандартній бібліотеці C++: https://en.cppreference.com/w/

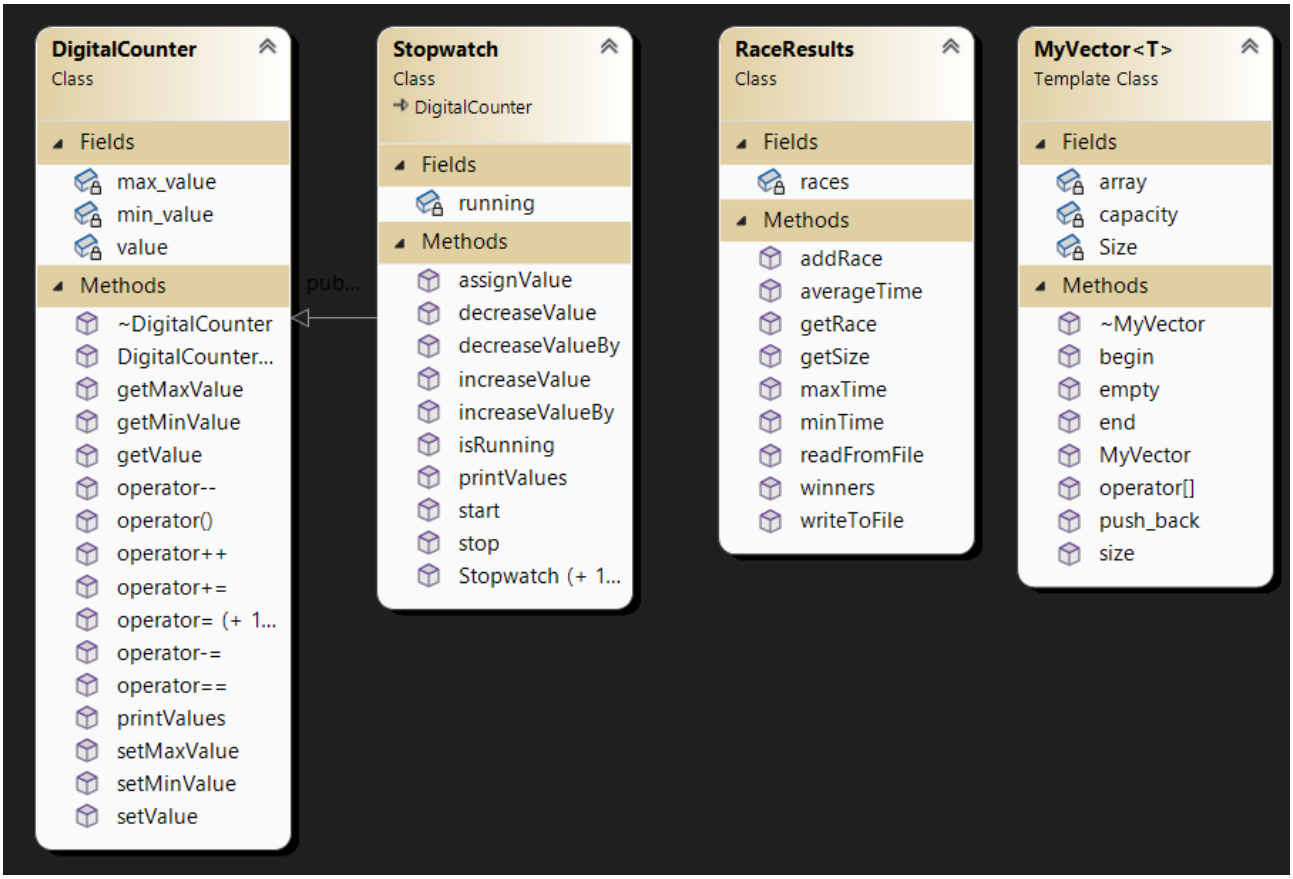
Для реалізації на C#

1. Е. Троелсен, "Мова програмування C# 7 і платформи .NET та .NET Core", Вільямс, 2018.
2. М. Макдональд, "Pro .NET 4.5 Framework", Apress, 2012.
3. А. Фрімен, "Programming .NET Components", O'Reilly Media, 2006.
4. Офіційна документація по C# та .NET на сайті Microsoft Docs: https://docs.microsoft.com/en-us/dotnet/csharp/
5. Дж. Албахарі та Б. Албахарі, "C# 7.0 in a Nutshell: The Definitive Reference", O'Reilly Media, 2017.
6. Т. Дізані та Т. Віллінгем, "Windows Forms 2.0 Programming", Apress, 2006.

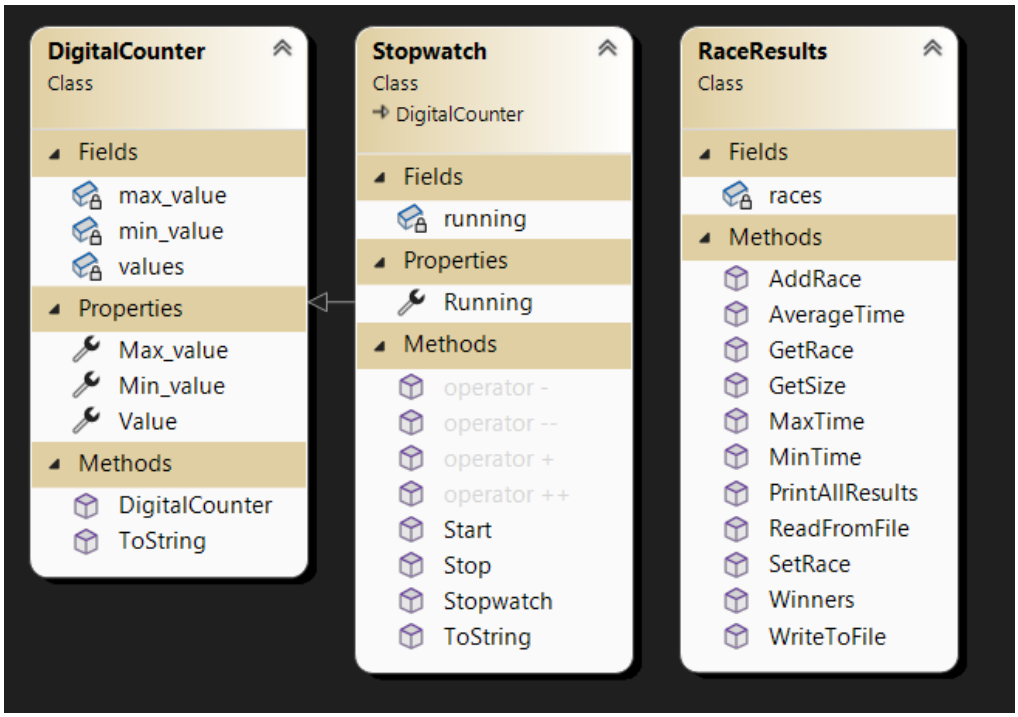
Додатки

Додаток № 1. Діаграма класів

Діаграми класів консольної програми реалізованої на мові C++.



Діаграми класів програмного додатку з графічним інтерфейсом Windows Forms, які реалізовані на C#.



Додаток № 2. Код консольної реалізації програми на мові C++

```
#include <iostream>

#include <Windows.h>
```

```
#include <fstream>
#include <string>
#include <algorithm>

using namespace std;

template <typename T>
class MyVector
{
    T* array;
    size_t capacity;
    size_t Size;

public:
    MyVector() : array(nullptr), capacity(0), Size(0) {}

    ~MyVector()
    {
        delete[] array;
    }

    void push_back(const T& element)
    {
        if (Size >= capacity)
        {
            capacity = (capacity == 0) ? 1 : capacity * 2;
            T* newArray = new T[capacity];
            for (size_t i = 0; i < Size; ++i)
            {
                newArray[i] = array[i];
            }
            delete[] array;
            array = newArray;
```

```
        }
        array[Size++] = element;
    }

    T& operator[](size_t index)
    {
        if (index >= Size)
        {
            throw out_of_range("Індекс вийшов за межі видимості");
        }
        return array[index];
    }

    size_t size() const
    {
        return Size;
    }

    bool empty() const
    {
        return Size == 0;
    }

    T* begin()
    {
        return array;
    }

    T* end()
    {
        return array + Size;
    }
};
```

```
class DigitalCounter
{
    int value;
    int max_value;
    int min_value;
public:
    DigitalCounter()
    {
        value = 0;
        max_value = 59;
        min_value = 0;
    }

    DigitalCounter(int value, int max_value, int min_value)
    {
        this->value = value;
        this->max_value = max_value;
        this->min_value = min_value;
    }

    DigitalCounter(const DigitalCounter& counter)
    {
        value = counter.value;
        max_value = counter.max_value;
        min_value = counter.min_value;
    }

    DigitalCounter(DigitalCounter&& counter) noexcept
    {
        value = counter.value;
        max_value = counter.max_value;
        min_value = counter.min_value;
```

```
        counter.value = 0;
    }

    ~DigitalCounter() {}

    void setMaxValue(int max_value)
    {
        this->max_value = max_value;
    }

    int getMaxValue()
    {
        return max_value;
    }

    void setMinValue(int min_value)
    {
        this->min_value = min_value;
    }

    int getMinValue()
    {
        return min_value;
    }

    void setValue(int val)
    {
        try
        {
            if (val < min_value || val > max_value + 1)
            {
                throw out_of_range("Невірне значення. Значення має бути в межах від " +
to_string(min_value) + " до " + to_string(max_value));
            }
```

```
            else if (cin.fail())
            {
                throw invalid_argument("Введено неправильний тип даних");
            }
            else
            {
                value = val;
                if (value == max_value + 1)
                {
                    value = min_value;
                    cout << "Значення досягло максимуму. Встановлено мінімальне значення: "
<< min_value << endl;
                }
            }
        }
        catch (const out_of_range& e)
        {
            cerr << "Помилка: " << e.what() << endl;
        }
        catch (const invalid_argument& e)
        {
            cerr << "Помилка: " << e.what() << endl;
        }
    }

    int getValue()
    {
        return value;
    }

    virtual void printValues()
    {
        cout << "Поточне значення: " << value << endl;
        cout << "Максимальне значення: " << max_value << endl;
```

```
        cout << "Мінімальне значення: " << min_value << endl;
    }

    void operator()(int val, int max_val, int min_val)
    {
        value = val;
        max_value = max_val;
        min_value = min_val;
    }

    DigitalCounter& operator++()
    {
        try
        {
            if (value > max_value)
            {
                throw string("Помилка: Перевищено максимальне значення");
            }
            ++value;
            if (value == max_value + 1)
            {
                value = min_value;
                cout << "Значення досягло максимуму. Встановлено мінімальне значення: " <<
min_value << endl;
            }
        }
        catch (const string& errorMsg)
        {
            cerr << errorMsg << endl;
        }
        return *this;
    }

    DigitalCounter& operator--()
```

```
    {
        try
        {
            if (value > min_value)
            {
                --value;
            }
            else
            {
                throw string("Помилка: Значення вже досягло мінімального значення.");
            }
        }
        catch (const string& errorMessage)
        {
            cerr << errorMessage << endl;
        }
        return *this;
    }


    DigitalCounter& operator+=(int val)
    {
        try
        {
            if (value + val > max_value + 1)
            {
                throw string("Помилка: Спроба збільшити значення настільки, що воно
перевищить максимально допустиме значення.");
            }
            value += val;
            if (value == max_value + 1)
            {
                value = min_value;
                cout << "Значення досягло максимуму. Встановлено мінімальне значення: " <<
min_value << endl;
```

```
            }
        }
        catch (const string& errorMessage)
        {
            cerr << errorMessage << endl;
        }
        return *this;
    }

    DigitalCounter& operator-=(int val)
    {
        try
        {
            if (value - val >= min_value)
            {
                value -= val;
            }
            else
            {
                throw string("Помилка: Значення виходить за межі діапазону.");
            }
        }
        catch (const string& errorMessage)
        {
            cerr << errorMessage << endl;
        }
        return *this;
    }

    bool operator==(const DigitalCounter& other)
    {
        return value == other.value && max_value == other.max_value && min_value ==
other.min_value;
    }
```

```
DigitalCounter& operator=(const DigitalCounter& counter)
{
    if (this != &counter)
    {
        value = counter.value;
        max_value = counter.max_value;
        min_value = counter.min_value;
    }
    return *this;
}

DigitalCounter& operator=(DigitalCounter&& counter) noexcept
{
    if (this != &counter)
    {
        value = counter.value;
        max_value = counter.max_value;
        min_value = counter.min_value;
        counter.value = 0;
    }
    return *this;
}

friend istream& operator>>(istream& is, DigitalCounter& counter)
{
    is >> counter.value;
    is >> counter.max_value;
    is >> counter.min_value;
    return is;
}

friend ostream& operator<<(ostream& os, const DigitalCounter& counter)
```

```
    {
        os << "Поточне значення: " << counter.value << endl;
        os << "Максимальне значення: " << counter.max_value << endl;
        os << "Мінімальне значення: " << counter.min_value << endl;
        return os;
    }
};

class Stopwatch : public DigitalCounter
{
    bool running;
public:
    Stopwatch() : DigitalCounter()
    {
        running = false;
    }

    Stopwatch(int value, int max_value, int min_value, bool running) :
DigitalCounter(value, max_value, min_value)
    {
        this->running = running;
    }

    void start()
    {
        running = true;
    }

    void stop()
    {
        running = false;
    }

    bool isRunning()
```

```
{
    return running;
}

void increaseValue()
{
    ++(*this);
}

void decreaseValue()
{
    --(*this);
}

void increaseValueBy(int increment)
{
    *this += increment;
}

void decreaseValueBy(int decrement)
{
    *this -= decrement;
}

void assignValue(int newValue)
{
    setValue(newValue);
}

void printValues() override
{
    DigitalCounter::printValues();
    cout << "Стан секундоміра: " << (running ? "Включено" : "Виключено") << endl;
```

```
    }

    friend ostream& operator<<(ostream& os, Stopwatch& stopwatch)
    {
        os << "Поточне значення: " << stopwatch.getValue() << endl;
        os << "Максимальне значення: " << stopwatch.getMaxValue() << endl;
        os << "Мінімальне значення: " << stopwatch.getMinValue() << endl;
        os << "Стан секундоміра: " << (stopwatch.running ? "Включено" : "Виключено") <<
endl;
        return os;
    }
};

class RaceResults
{
    MyVector<Stopwatch> races;
public:
    void addRace(Stopwatch race)
    {
        races.push_back(race);
    }

    Stopwatch& getRace(int index)
    {
        if (index >= 0 && index < races.size())
        {
            return races[index];
        }
        else
        {
            throw out_of_range("Неприпустимий індекс забігу.");
        }
    }
```

```
int getSize()
{
    return races.size();
}

int minTime()
{
    if (races.empty())
    {
        cerr << "Масив перегонів порожній!" << endl;
        return -1;
    }

    auto minRace = min_element(races.begin(), races.end(),
        [](Stopwatch& a, Stopwatch& b)
        {
            return a.getValue() < b.getValue();
        });

    return minRace->getValue();
}

int maxTime()
{
    if (races.empty())
    {
        cerr << "Масив перегонів порожній!" << endl;
        return -1;
    }

    auto maxRace = max_element(races.begin(), races.end(),
        [](Stopwatch& a, Stopwatch& b)
        {
```

```
            return a.getValue() < b.getValue();
        });

    return maxRace->getValue();
}

double averageTime()
{
    if (races.empty())
    {
        cerr << "Масив перегонів порожній!" << endl;
        return -1;
    }

    int sum = 0;
    for (auto& race : races)
    {
        sum += race.getValue();
    }

    return static_cast<double>(sum) / races.size();
}

MyVector<int>& winners()
{
    if (races.empty())
    {
        cerr << "Масив перегонів порожній!" << endl;
        static MyVector<int> emptyVector;
        return emptyVector;
    }

    MyVector<int> times;
```

```
        for (auto& race : races)
        {
            times.push_back(race.getValue());
        }

        sort(times.begin(), times.end());

        static MyVector<int> winners; // Static для збереження значення поза областю видимості функції
        winners = MyVector<int>();
        for (int i = 0; i < 3 && i < times.size(); ++i)
        {
            winners.push_back(times[i]);
        }
        return winners;
    }

    void readFromFile(const string& filename)
    {
        ifstream file(filename);
        if (!file.is_open())
        {
            cerr << "Неможливо відкрити файл " << filename << endl;
            return;
        }

        int value, max_value, min_value;
        bool running;
        while (file >> value >> max_value >> min_value >> running)
        {
            races.push_back(Stopwatch(value, max_value, min_value, running));
        }

        file.close();
```

```
    }

    void writeToFile(const string& filename)
    {
        ofstream file(filename);
        if (!file.is_open())
        {
            cerr << "Неможливо відкрити файл " << filename << " для написання." << endl;
            return;
        }
        file.clear();
        for (auto& race : races)
        {
            file << race.getValue() << " " << race.getMaxValue() << " " <<
race.getMinValue() << " " << race.isRunning() << endl;
        }

        file.close();
    }
};

void startNewRace(RaceResults& raceResults)
{
    Stopwatch newRace;
    newRace.start();
    cout << "Запишіть результат: " << endl;
    int result;
    cin >> result;
    try
    {
        if (result < newRace.getMinValue() || result > newRace.getMaxValue())
        {
            throw out_of_range("Невірне значення");
        }
```

```
        else if (cin.fail())
        {
            throw invalid_argument("Введено неправильний тип даних");
        }
        else
        {
            newRace.setValue(result);
            newRace.stop();
            raceResults.addRace(newRace);
        }
    }
    catch (const out_of_range& e)
    {
        cerr << "Помилка: " << e.what() << endl;
    }
    catch (const invalid_argument& e)
    {
        cerr << "Помилка: " << e.what() << endl;
    }
}

void displayRecords(RaceResults& raceResults)
{
    cout << "Рекорди:" << endl;
    cout << "Мінімальний час: " << raceResults.minTime() << endl;
    cout << "Максимальний час: " << raceResults.maxTime() << endl;
    cout << "Середній час: " << raceResults.averageTime() << endl;
    cout << "Перші три переможці: ";
    MyVector<int> winners = raceResults.winners();
    for (int time : winners)
    {
        cout << time << " ";
    }
```

```
    cout << endl;
}

void displayMenu()
{
    cout << "Меню:" << endl;
    cout << "1. Додати результати забігу" << endl;
    cout << "2. Показати рекорди" << endl;
    cout << "3. Вивести інформацію про забіг за його номером" << endl;
    cout << "4. Змінити значення забігу" << endl;
    cout << "5. Зберегти результати у файл" << endl;
    cout << "6. Зчитати результати з файлу" << endl;
    cout << "7. Вийти" << endl;
    cout << "Виберіть опцію: ";
}

void changeValue(RaceResults& raceResults)
{
    cout << "1. Збільшити значення забігу на 1" << endl;
    cout << "2. Зменшити значення забігу на 1" << endl;
    cout << "3. Збільшити значення забігу на вказане число" << endl;
    cout << "4. Зменшити значення забігу на вказане число" << endl;
    cout << "5. Присвоїти нове значення забігу" << endl;
    cout << "Виберіть опцію: ";
    unsigned choice;
    cin >> choice;
    int index;
    cout << "Введіть індекс забігу, який бажаєте змінити: ";
    cin >> index;
    switch (choice)
    {
    case 1:
        if (index >= 0 && index < raceResults.getSize())
```

```
    {
        Stopwatch& race = raceResults.getRace(index);
        race.increaseValue();
    }
    else
    {
        cout << "Невірний індекс." << endl;
    }break;
case 2:
    if (index >= 0 && index < raceResults.getSize())
    {
        Stopwatch& race = raceResults.getRace(index);
        race.decreaseValue();
    }
    else
    {
        cout << "Невірний індекс." << endl;
    }break;
case 3:
    if (index >= 0 && index < raceResults.getSize())
    {
        int i;
        cout << "Введіть значення: ";
        cin >> i;
        Stopwatch& race = raceResults.getRace(index);
        race.increaseValueBy(i);
    }
    else
    {
        cout << "Невірний індекс." << endl;
    }break;
case 4:
    if (index >= 0 && index < raceResults.getSize())
```

```
        {
            int i;
            cout << "Введіть значення: ";
            cin >> i;
            Stopwatch& race = raceResults.getRace(index);
            race.decreaseValueBy(i);
        }
        else
        {
            cout << "Невірний індекс." << endl;
        }break;
    case 5:
        if (index >= 0 && index < raceResults.getSize())
        {
            int i;
            cout << "Введіть значення: ";
            cin >> i;
            Stopwatch& race = raceResults.getRace(index);
            race.assignValue(i);
        }
        else
        {
            cout << "Невірний індекс." << endl;
        }break;
    default:  cout << "Помилка при виборі пункту меню." << endl; break;
    }
}

void displayValues(DigitalCounter& counter)
{
    counter.printValues();
}
```

```
void displayCurrentValue(RaceResults& races, size_t index)
{
    index -= 1;
    if (index < races.getSize())
    {
        try
        {
            cout << "Вивести інформацію разом зі станом секундоміра?\n1. Так\n2. Ні" <<
endl;
            unsigned choice;
            cin >> choice;
            if (cin.fail() || (choice < 1 && choice > 2))
            {
                throw exception("Помилка при введенні значення.");
            }
            else if (choice == 1)
            {
                Stopwatch& race = races.getRace(index);
                displayValues(race);
            }
            else if (choice == 2)
            {
                DigitalCounter& race = races.getRace(index);
                DigitalCounter race1 = race;
                displayValues(race1);
            }
        }
        catch (exception& e)
        {
            cerr << "Помилка: " << e.what() << endl;
        }
    }
    else
    {
```

```
        cout << "Невірний індекс." << endl;
    }
}

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    RaceResults raceResults;
    unsigned choice;
    bool stopwork = false;
    while (stopwork != true)
    {
        displayMenu();

        if (cin.fail())
        {
            cin.clear();
            cin.ignore(INT_MAX, '\n');
        }
        cin >> choice;
        switch (choice)
        {
        case 1: startNewRace(raceResults); break;
        case 2: displayRecords(raceResults); break;
        case 3: int index;
            cout << "Введіть номер результату, який хочете переглянути: ";
            cin >> index;
            displayCurrentValue(raceResults, index); break;
        case 4: changeValue(raceResults); break;
        case 5: raceResults.writeToFile("RaceResults.txt");
            cout << "Результати збережено у файл " << "RaceResults" << endl; break;
        case 6: raceResults.readFromFile("RaceResults.txt");
```

```
            cout << "Результати зчитано з файлу " << "RaceResults" << endl; break;

        case 7: stopwork = true; break;

        default:cout << "Помилка при виборі пункту меню." << endl; break;

        }

    }

    system("pause");

    return 0;

}
```

Додаток № 3. Код реалізації програми з графічним інтерфейсом з використанням Windows Forms на мові C#

DigitalCounter.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CourseWork1_2
{
    class DigitalCounter
    {
        private int values;
        private int max_value;
        private int min_value;

        public DigitalCounter()
        {
            values = 0;
            max_value = 59;
            min_value = 0;
        }
        public int Value
        {
            get { return values; }
            set
            {
                try
                {
                    if (values < min_value || values > max_value)
                    {
                        throw new ArgumentOutOfRangeException("Невірне значення.
Значення має бути в межах від " + min_value + " до " + max_value);
                    }
                    else values = value;
                }
                catch (ArgumentOutOfRangeException ex)
                {
                    Console.WriteLine("Помилка: " + ex.Message);
                }
            }
        }

        public int Max_value
        {
```

```
            get { return max_value; }
            set { max_value = value; }
        }

        public int Min_value
        {
            get { return min_value; }
            set { min_value = value; }
        }

        public virtual string ToString()
        {
            return "Поточне значення: " + min_value + "\nМаксимальне значення: " +
max_value + "\nМінімальне значення: " + min_value;
        }
    }
}
```

Stopwatch.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CourseWork1_2
{
    class Stopwatch : DigitalCounter
    {
        private bool running;

        public bool Running
        {
            get { return running; }
            set { running = value; }
        }

        public Stopwatch()
        {
            Value = 0;
            Max_value = 59;
            Min_value = 0;
            running = false;
        }

        public static Stopwatch operator +(Stopwatch stopwatch, int valueToAdd)
        {
            stopwatch.Value = (stopwatch.Value + valueToAdd) % 60;
            return stopwatch;
        }

        public static Stopwatch operator -(Stopwatch stopwatch, int valueToSubtract)
        {
            if (valueToSubtract < 0)
    {
                throw new ArgumentOutOfRangeException("valueToSubtract", "Значення для
віднімання не може бути від'ємним.");
            }

            stopwatch.Value = (stopwatch.Value < valueToSubtract) ? 0 :
stopwatch.Value - valueToSubtract; // Якщо значення менше віднімання, встановлюємо
його на 0, інакше віднімаємо без змін
            return stopwatch;
        }

        public static Stopwatch operator ++(Stopwatch stopwatch)
```

```
        {
            stopwatch.Value = (stopwatch.Value + 1) % 60;
            return stopwatch;
        }

        public static Stopwatch operator --(Stopwatch stopwatch)
        {
            stopwatch.Value = (stopwatch.Value == 0) ? 0 : stopwatch.Value - 1; //
Якщо значення 0, залишаємо його, інакше зменшуємо на 1
            return stopwatch;
        }

        public void Start()
        {
            running = true;
        }

        public void Stop()
        {
            running = false;
        }

        public override string ToString()
        {
            return base.ToString() + "\nСтан секундоміра: " + (running ? "Включено" :
"Виключено");
        }
    }
}
```

RaceResults.cs

```
using System;
using System.Text.Json;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace CourseWork1_2
{
    class RaceResults
    {
        private List<Stopwatch> races = new List<Stopwatch>();

        public void AddRace(Stopwatch race)
        {
            races.Add(race);
        }

        public int GetSize()
        {
            return races.Count;
        }

        public int MinTime()
        {
            try
            {
                if (races.Count == 0)
                {
                    throw new InvalidOperationException("Список забігів порожній.");
                }

                return races.Min(race => race.Value);
```

```
            }
            catch (InvalidOperationException ex)
            {
                Console.WriteLine($"Помилка: {ex.Message}");
                return -1;
            }
        }

        public int MaxTime()
        {
            try
            {
                if (races.Count == 0)
                {
                    throw new InvalidOperationException("Список забігів порожній.");
                }

                return races.Max(race => race.Value);
            }
            catch (InvalidOperationException ex)
            {
                Console.WriteLine($"Помилка: {ex.Message}");
                return -1;
            }
        }

        public double AverageTime()
        {
            try
            {
                if (races.Count == 0)
                {
                    throw new InvalidOperationException("Список забігів порожній.");
                }

                return races.Average(race => race.Value);
            }
            catch (InvalidOperationException ex)
            {
                Console.WriteLine($"Помилка: {ex.Message}");
                return -1;
            }
        }

        public List<int> Winners()
        {
            List<int> times = races.Select(race => race.Value).ToList();
            times.Sort();
            return times.Take(3).ToList();
        }

        public void PrintAllResults()
        {
            string message = "Усі значення забігів:\n\n";

            foreach (var race in races)
            {
                message += $"Значення: {race.Value}, Максимальне значення:
{race.Max_value}, Мінімальне значення: {race.Min_value}, Стан: {(race.Running ?
"Включено" : "Виключено")}\n";
            }

            MessageBox.Show(message, "Усі результати", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }
```

```
        public Stopwatch GetRace(int index)
        {
            try
            {
                if (index >= 0 && index < races.Count)
                {
                    return races[index];
                }
                else
                {
                    throw new ArgumentOutOfRangeException("index", "Некоректний індекс
забігу");
                }
            }
            catch (ArgumentOutOfRangeException ex)
            {
                MessageBox.Show("Помилка: " + ex.Message, "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return null;
            }
        }

        public void SetRace(int index, Stopwatch race)
        {
            try
            {
                if (index >= 0 && index < races.Count)
                {
                    races[index] = race;
                }
                else
                {
                    throw new ArgumentOutOfRangeException("index", "Некоректний індекс
забігу");
                }
            }
            catch (ArgumentOutOfRangeException ex)
            {
                MessageBox.Show("Помилка: " + ex.Message, "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        public void ReadFromFile(string filename)
        {
            string jsonString = File.ReadAllText(filename);
            races = JsonSerializer.Deserialize<List<Stopwatch>>(jsonString);
        }

        public void WriteToFile(string filename)
        {
            string jsonString = JsonSerializer.Serialize(races);
            File.WriteAllText(filename, jsonString);
        }
    }
}
```

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace CourseWork1_2
{
    public partial class Form1 : Form
    {
        private RaceResults raceResults = new RaceResults();
        public Form1()
        {
            InitializeComponent();
        }

        private void btn_change_Click(object sender, EventArgs e)
        {
            int index;
            if (int.TryParse(indexTextBox.Text, out index))
            {
                try
                {
                    if (index >= 0 && index < raceResults.GetSize())
                    {
                        Stopwatch race = raceResults.GetRace(index);
                        int valueToAdd = 0;
                        if (addRadioButton.Checked)
                        {
                            race++;
                            addRadioButton.Checked = false;
                        }
                        else if (subtractRadioButton.Checked)
                        {
                            race--;
                            subtractRadioButton.Checked = false;
                        }
                        else
                        {
                            if (!int.TryParse(valueTextBox.Text, out valueToAdd))
                            {
                                MessageBox.Show("Некоректне значення для додавання або
віднімання", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                                return;
                            }
                        }
                        if (plusRadiobtn.Checked)
                        {
                            race += valueToAdd;
                            plusRadiobtn.Checked = false;
                        }
                        else if (minusRadiobtn.Checked)
                        {
                            race -= valueToAdd;
                            minusRadiobtn.Checked = false;
                        }
                        raceResults.SetRace(index, race);
                        indexTextBox.Text = "";
                        valueTextBox.Text = "";
                    }
                    else
                    {
                        MessageBox.Show("Некоректний індекс забігу", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    }
                }
                catch (ArgumentOutOfRangeException ex)
```

```
                {
                    MessageBox.Show("Помилка: " + ex.Message, "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
            else
            {
                MessageBox.Show("Некоректний індекс забігу", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        private void btn_addResult_Click(object sender, EventArgs e)
        {
            try
            {
                Stopwatch newRace = new Stopwatch();
                newRace.Start();
                int result;
                if (int.TryParse(resultTextBox.Text, out result))
                {
                    if (result >= newRace.Min_value && result <= newRace.Max_value)
                    {
                        newRace.Value = result;
                        newRace.Stop();
                        raceResults.AddRace(newRace);
                        resultTextBox.Clear();
                    }
                    else
                    {
                        MessageBox.Show($"Некоректне значення результата. Значення
повинно бути в межах від {newRace.Min_value} до {newRace.Max_value}", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    }
                }
                else
                {
                    MessageBox.Show("Некоректне значення результата", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    resultTextBox.Clear();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Сталася помилка: " + ex.Message, "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                resultTextBox.Clear();
            }
        }

        private void btn_showRecord_Click(object sender, EventArgs e)
        {
            StringBuilder records = new StringBuilder();
            records.AppendLine("Рекорди:");
            records.AppendLine("Мінімальний час: " + raceResults.MinTime());
            records.AppendLine("Максимальний час: " + raceResults.MaxTime());
            records.AppendLine("Середній час: " + raceResults.AverageTime());
            records.Append("Перші три переможці: ");
            List<int> winners = raceResults.Winners();
            for (int i = 0; i < winners.Count; i++)
            {
                records.Append(winners[i]);
                if (i < winners.Count - 1)
                {
                    records.Append(", ");
                }
```

```
            }
            MessageBox.Show(records.ToString(), "Рекорди", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }

        private void btn_readFromFile_Click(object sender, EventArgs e)
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            try
            {
                if (openFileDialog.ShowDialog() == DialogResult.OK)
                {
                    raceResults.ReadFromFile(openFileDialog.FileName);
                    MessageBox.Show("Результати зчитано з файлу", "Успіх",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка при зчитуванні з файлу: " + ex.Message,
"Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        private void btn_writeToFile_Click(object sender, EventArgs e)
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog();
            try
            {
                if (saveFileDialog.ShowDialog() == DialogResult.OK)
                {
                    raceResults.WriteToFile(saveFileDialog.FileName);
                    MessageBox.Show("Результати збережено у файл", "Успіх",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка при збереженні у файл: " + ex.Message,
"Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        private void btn_showAll_Click(object sender, EventArgs e)
        {
           raceResults.PrintAllResults();
        }
    }
}
```