КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Кафедра інженерії програмного забезпечення та кібербезпеки

|  |  |
| --- | --- |
|  | *Захищено на кафедрі інженерії програмного*  *забезпечення та кібербезпеки*  *«14» травня 2021 р.*  *з оцінкою 98 балів (A)*  *Підпис членів комісії:*  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |

**КУРСОВА РОБОТА**

**з дисципліни**

**«ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»**

**НА ТЕМУ:**

**«Розробка програмного продукту «Hex-редактор» мовою програмування C#»**

**Виконав:** студент факультету інформаційних технологій

2 курсу 12 групи

**Фєфєлов Максим Андрійович**

**Науковий керівник:**

асистент кафедри інженерії програмного забезпечення та кібербезпеки

**Гнатченко Дмитро Дмитрович**

**КИЇВ 2021**

Київський національний торговельно-економічний університет

Кафедра інженерії програмного забезпечення та кібербезпеки

**КАРТКА**

**завдання та контролю за ходом виконання курсової роботи**

1. Прізвище та ініціали студента Фєфєлов М.А.
2. Факультет Факультет інформаційних технологій (ФІТ)
3. Курс, група 2 курс 12 група
4. Форма навчання денна
5. Номер залікової книжки №34/19-297
6. Номер та назва теми курсової роботи Розробка програмного продукту «Hex-редактор» мовою програмування C#
7. Дата погодження теми з науковим керівником 08.04.2021
8. Дата затвердження плану роботи 08.04.2021
9. Дата подання виконаної роботи на кафедрі 11.05.2021
10. Дата захисту роботи 14.05.2021
11. Тему, план і строки прийняв до виконання

(підпис студента)

1. Науковий керівник Гнатченко Д.Д.

(прізвище та підпис)

«14» травня 2021 р.

**ЗМІСТ**

[ВСТУП 2](#_Toc71224730)

[РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ 4](#_Toc71224731)

[1.1. Передумови виникнення об’єктно-орієнтованого програмування 4](#_Toc71224732)

[1.2. Основні поняття та принципи об’єктно-орієнтованого програмування 5](#_Toc71224733)

[1.3. Вплив об’єктно-орієнтованого програмування на розвиток програмного забезпечення 7](#_Toc71224734)

[Висновки до Розділу 1 8](#_Toc71224735)

[РОЗДІЛ 2. АНАЛІЗ ІНШИХ ПРОГРАМНИХ ПРОДУКТІВ І ВИБІР ІНСТРУМЕНТІВ РОЗРОБКИ 9](#_Toc71224736)

[2.1. Порівняльна характеристика різних шістнадцяткових редакторів 9](#_Toc71224737)

[2.2. Вибір середовища та інструментів розробки 10](#_Toc71224738)

[Висновки до Розділу 2 11](#_Toc71224739)

[РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 12](#_Toc71224740)

[3.1. Визначення функціональності програми 12](#_Toc71224741)

[3.2. Проектування програмного забезпечення 12](#_Toc71224742)

[3.3. Створення керуючого компонента 13](#_Toc71224743)

[3.4. Створення користувацького інтерфейсу 15](#_Toc71224744)

[3.5. Розробка логіки редактора 17](#_Toc71224745)

[Висновки до Розділу 3 19](#_Toc71224746)

[ВИСНОВКИ 20](#_Toc71224747)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 21](#_Toc71224748)

[ДОДАТКИ 22](#_Toc71224749)

# ВСТУП

З розвитком електронно-обчислювальних машин (ЕОМ) збільшується також й об’єм даних, які ці пристрої оброблюють. Виникає необхідність в упорядковуванні цих даних у конкретні одиниці — файли. Спочатку файлами називали папки, куди складали перфокарти, а згодом їх замінили логічні структурні одиниці, що містять певний набір даних. Через деякий час ми відмовилися від перфокарт і перфострічок, і користуємося магнітними стрічками та дискетами для зберігання цих файлів, а згодом — жорсткими дисками та іншими сучасними носіями інформації.

Оскільки з часом створюються різні ЕОМ і різні програми до них, що оперують різними даними (текст, набір чисел тощо), виникають також і різні формати файлів. Іноді виникає необхідність у перегляді цих файлів безпосередньо у вигляді набору звичайних чисел (у випадку з двійковими ЕОМ машинні команди та дані є наборами чисел, які в свою чергу складаються з нулів та одиниць), наприклад, при вивченні структури невідомого формату файлу. Не виключено, що їх потрібно ще і відредагувати. Якщо безпосередні дані на перфокарті або перфострічці можна переглянути «на око», то у випадку з даними на більш сучасних носіях це вочевидь не вдасться. Виникає потреба у спеціальному засобі, що дозволяє переглядати або редагувати будь-який файл незалежно від його формату на рівні окремих чисел, з яких він складається. Вважається, що для цього треба застосувати об’єктно-орієнтоване програмування (ООП).

*Завдання* курсової роботи: дослідження проблеми редагування файлів довільного або невідомого формату, вирішення проблеми за допомогою розробки спеціального програмного забезпечення (ПЗ), що вміє редагувати будь-який двійковий файл.

*Метою* даної роботи є розробка програмного застосунку мовою C# для перегляду та редагування комп’ютерних файлів на рівні окремих байтів з використанням елементів ООП.

*Об’єктом дослідження* є редагування комп’ютерних файлів довільного або невідомого формату.

*Предметом* *дослідження* є програма-редактор двійкових файлів.

*Під час виконання проекту* буде розроблений програмний продукт мовою C# для платформи .NET Framework 4.8 з використанням середовища розробки Microsoft Visual Studio 2019.

# РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

## 1.1. Передумови виникнення об’єктно-орієнтованого програмування

Однією з найперших мов програмування, яка запровадила багато сучасних понять об’єктно-орієнтованого програмування (ООП), є мова «Симула» (англ. Simula), яка була створена в Норвезькому Комп’ютерному центрі Оле-Юганом Далем і Крістеном Нюгордом у 1962 році. У ній з’явилися такі поняття, як: об’єкти, класи, наслідування, віртуальні функції. Також у неї був механізм збирання сміття (англ. garbage collection).[1]

На основі «Симули» в 1972 році компанія Xerox PARC розроблює Smalltalk. Основною особливістю цієї мови є те, що будь-що, навіть примітивні типи даних, є об’єктами. Крім того, кожен об’єкт взаємодіє між собою через повідомлення: один об’єкт надсилає повідомлення, інший — отримує його, оброблює та надсилає наступному об’єкту.[2]

Концепція мов Simula та Smalltalk була використана при розробці мови Objective-C, яка з’явилась у 1984 році, Б’ярном Страуструпом для створення мови C++ у 1985 році та компанією Apple для розробки Object Pascal у 1986 році.

ООП набуває великої популярності в кінці 1980-х — на початку 1990-х років. З’являються нові мови, які впливають на розвиток одна одної; значна їх частина є об’єктно-орієнтованою: Python (1991), Java (1995), C# (2000), Visual Basic .NET (2001), Go (2009) та інші. Усі ці перелічені мови різні за можливостями та наявністю інших парадигм програмування, але їх усіх об’єднує схожа концепція та термінологія: об’єктна орієнтованість.

Таким чином, саме «Симула» дала поштовх до розвитку ООП в цілому, та відповідно вплинула на розвиток майбутніх об’єктно-орієнтованих мов.[2]

## 1.2. Основні поняття та принципи об’єктно-орієнтованого програмування

Об’єктно-орієнтоване програмування — нова парадигма програмування, що базується на взаємодії між собою певних умовних одиниць — об’єктів, що містять дані та програмний код. Це дозволяє розробнику створювати програми у вигляді сукупності цих умовних предметів, задаючи їхні властивості та поведінку.

Об’єктно-орієнтоване програмування має власну, досить специфічну термінологію. Насамперед варто зазначити деякі основні поняття, що використовуються в ООП:

* **Класи.** Представляють собою опис загальної структури та функцій деякої сукупності об’єктів. Наприклад, посудину для зберігання рідини можна описати класом «Тара», що містить поля «Об’єм» і «Кількість рідини» та методи «Залити рідину» та «Вилити рідину». Таким чином, клас є своєрідним «кресленням» об’єкта певного типу.
* **Об’єкти.** Об’єктами називають власне об’єкти, які відносяться до певних класів.[3] Клас задає певний тип об’єкту, а сам об’єкт заданого типу створюється згідно з певним класом.
* **Функції-члени класів.** Функціями-членами класів, або **методами**, називають ті функції, що визначені в межах будь-яких класів.
* **Інтерфейс.** Визначає спосіб обміну інформацією між функціями, класами, модулями і т.д. Інтерфейс функції реалізується, наприклад, через її параметри або значення, що вона повертає.
* **Об’єктно-орієнтований.** В об’єктно-орієнтованому програмуванні задачі вирішуються шляхом створення певних класів і подальшої роботи з ними. Більшу частину витрат при програмуванні займає розробка відповідного класу для представлення об’єктів. Програма, що використовує розроблений клас, значно спрощується, оскільки при описі класу визначаються його методи, що точно обмежують коректні дії програми. Внутрішня реалізація класу при цьому може бути невідома програмісту, який використовує даний клас у своїй програмі. Перевагою є також можливість повторного використовування коду у вигляді класів, побудованому на принципі «орієнтації на об’єкти» людського мислення, яке оперує деякими даними в цілому, а не елементарними даними, з чого ці об’єкти складаються.[3]

Об’єктно-орієнтоване програмування базується на чотирьох принципах:

* **Абстракція.** Означає опис зовнішніх властивостей об’єкта без урахування його внутрішньої організації та конкретної реалізації. Дане поняття найчастіше застосовується при створенні класів та означає, що клас представляється у вигляді «чорної скрині».[3] Таким чином, оскільки програмісту не потрібно знати точну реалізацію та внутрішню структуру класу, це полегшує його використання. Поєднується з інкапсуляцією.
* **Інкапсуляція.** Термін означає, що реальні об’єкти у програмі розглядаються як ті, що складаються з деяких елементів (даних) і їх функцій (методів). Елементи та функції об’єкта описуються класом, який представляє об’єкт. З концепцією інкапсуляції тісно пов’язане поняття абстракції та створення оптимального інтерфейсу.[3]
* **Наслідування.** Класи можуть бути ієрархічно організовані. Даний термін означає, що в похідному класі можуть бути збережені деякі властивості (структура та методи) базового класу.[3] Розглянемо той самий клас «Тара»: від нього можна створювати похідні класи «Чашка», «Пляшка», «Діжка», «Цистерна» тощо.
* **Поліморфізм.** Дозволяє виконувати однойменні (та однотипні) операції з об’єктами різноманітних класів. Наприклад, набір операцій для роботи з об’єктом «Коло» відрізняється від набору для роботи з об’єктом «Прямокутник»:[3] методи для знаходження периметру та площі цих геометричних фігур мають різну реалізацію. В об’єктно-орієнтованому програмуванні можна використовувати для різних типів об’єктів функцію з тією самою назвою, причому для кожного типу об’єктів буде викликатися саме функція, яка його оброблює. Це можливо завдяки ієрархії класів і механізму віртуальних функцій. Перевантаження функцій також може розглядатися як прояв поліморфізму.[3]

## 1.3. Вплив об’єктно-орієнтованого програмування на розвиток програмного забезпечення

Досліджуючи історію ООП, можна дійти висновку, що воно з’явилося в сучасному вигляді завдяки мовам Simula та Smalltalk[2]. Дана парадигма набула популярності, коли розробники мов програмування виявили інтерес до об’єктної орієнтованості, вбачаючи в ній деякий потенціал, і застосували її у своїх розробках. Та ще більшої популярності набуло ООП, коли розробники програмного забезпечення також зацікавились цією парадигмою та почали її широко застосовувати.

За індексом TIOBE (див. рис. 1), станом на 2021 рік, якщо не враховувати мову C, об’єктно-орієнтовані мови програмування є найпопулярнішими у світі.[4]

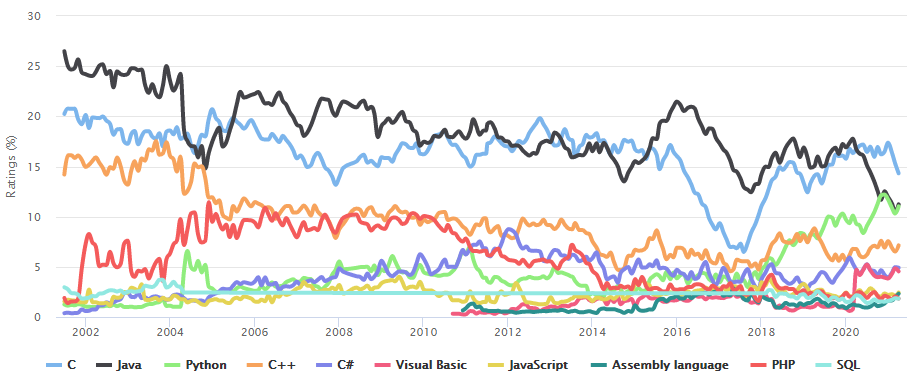


Рисунок 1. Графік рейтингу 10 найпопулярніших мов програмування за версією TIOBE.

Причина тривалої популярності мови C, яка не є об’єктно-орієнтованою, у тому, що в той час, як ООП спрямоване на швидку та зручну розробку програмного забезпечення, мова C спрямована на швидкодію виконання комп’ютером готової програми.

## Висновки до Розділу 1

Дана парадигма є справжнім нововведенням у програмуванні. У той час, як структуроване та функціональне програмування вже зробило розробку програмного забезпечення зручнішим у деякій мірі, об’єктно-орієнтований підхід також дозволяє:

* повторно використовувати вже існуючий програмний код;
* легко розроблювати типи даних, використовуючи концепції наслідування та поліморфізму;
* зменшити можливість помилкових дій завдяки інкапсуляції;
* покращити читабельність програми.

# РОЗДІЛ 2. АНАЛІЗ ІНШИХ ПРОГРАМНИХ ПРОДУКТІВ І ВИБІР ІНСТРУМЕНТІВ РОЗРОБКИ

## 2.1. Порівняльна характеристика різних шістнадцяткових редакторів

Вже існують різні Hex-редактори, і кожен із них відрізняється певними можливостями. Варто розглянути деякі з них.

**HHD Hex Editor Neo** є комерційним редактором двійкових файлів для Windows.[5] Окрім базової функціональності будь-якого звичайного Hex-редактора, головними особливостями цієї програми є:

* висока швидкодія при роботі з файлами будь-якого розміру;
* необмежений розмір файлу;
* багатопоточна обробка даних;
* представлення даних у вигляді різних форматів (десяткові та вісімкові числа, дійсні числа);
* миттєвий пошук конкретних значень у файлі, у тому числі та з заміною;
* необмежений буфер скасування (undo) змін;
* дуже гнучкий користувацький інтерфейс.

Програма поширюється за моделлю «Freemium», що означає, що не обов’язково платити за базову функціональність програмного забезпечення. Прикладами додаткових функцій Hex Editor Neo є:

* редагування дискових накопичувачів на рівні окремих секторів;
* шифрування та розшифрування даних;
* логічні та арифметичні операції;
* обчислення контрольних сум CRC-32, MD5, SHA-1 тощо;
* порівняння файлів.

**HxD** — безкоштовний (freeware) редактор.[6] Деякі можливості:

* редагування оперативної пам’яті;
* редагування дискових накопичувачів на рівні окремих секторів;
* висока швидкодія при роботі з великими файлами;
* обчислення контрольних сум;
* необмежений undo-буфер.

Спільним недоліком цих двох програм є незвично великий повний розмір програм (близько 10 МБ), враховуючи відносну простоту самого hex-редактора. Причинами невиправданого збільшення програм у розмірі можуть бути:

* навмисне чи ненавмисне «роздуття» програмного коду (code bloat) розробниками;
* наявність графічних даних інтерфейсу;
* неоптимальна генерація коду компіляторами.

Наявність переважно таких «роздутих» програмних продуктів є ще одним привидом до створення власного варіанту шістнадцяткового редактора.

## 2.2. Вибір середовища та інструментів розробки

Існує багато способів створення даного програмного застосунку. Є багато як об’єктно-орієнтованих мов програмування, які можна застосувати при написанні програмного коду, так і середовищ розробки.

Цільовою платформою буде MS Windows, оскільки ця операційна система використовується на більшості комп’ютерів у світі. Для написання програми можна застосувати інтерфейс Win32 та мову програмування C++, а можна — платформу .NET Framework та C#.

C++ є об’єктно-орієнтованою мовою програмування, що є розширенням мови C. Оскільки Windows API був з самого початку розроблений мовою C, деякий час ця мова була основною для розробки програмного забезпечення, що використовує цей інтерфейс. З появою підтримки 32-розрядних програм Windows API почав підтримувати C++, та відтоді більшість програм для Windows створюються даною мовою.

Мова C# була створена у 2000 році для проекту .NET, який ще був у розробці. Платформа .NET Framework націлена на Windows; на відміну від Win32 та C++, .NET Framework і C# із самого початку були створені об’єктно-орієнтованими. Зараз C# використовується не тільки для .NET, а також і для інших програмних платформ на кшталт Unity, Universal Windows Platform тощо.

У даному випадку буде застосовуватися .NET Framework і мова C#, оскільки дані програмна платформа та мова є легшими у використанні.

Конкретно буде застосовуватися .NET Framework 4.8 задля сумісності з Windows 7. Це остання версія .NET Framework, оскільки Microsoft наразі розроблює .NET / .NET Core, націлену на підтримку різними операційними системами.

## Висновки до Розділу 2

Згідно з результатами порівняння існуючих на даний момент Hex-редакторів, приймається рішення в написанні компактної програми (виконуваний файл повинен займати не більше 1 МБ) без графічного інтерфейсу.

Використання об’єктно-орієнтованої мови C# дозволяє створити застосунок у вигляді сукупності заданих об’єктів, що взаємодіють між собою. А завдяки мові C# функціональність платформи .NET Framework реалізована у вигляді сукупності певних класів, а не лише як набір звичайних функцій у C.

# РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1. Визначення функціональності програми

Насамперед потрібно визначити, з якою метою програмний продукт створюватиметься, і для чого його застосовувати. Завдання включає в себе безпосередньо створення програмного застосунку, що має змогу створювати, відкривати, редагувати та зберігати файли. Програма працюватиме в консолі, оскільки дані, з якими працює програма, представляються виключно в текстовому вигляді. Застосунок повинен вміти:

* відкривати, створювати та зберігати файли;
* відображати вміст файлу у вигляді таблиці шістнадцяткових чисел, кожне з яких показує значення кожного байту файлу;
* відображати інтерпретацію цих байтів як текст у 8-бітних стандартах кодування (ASCII, CP866, Windows-1251, UTF-8 тощо);
* показувати поточний розмір файлу;
* працювати в режимі заміщення (англ. overwrite) або вставки (англ. insert);
* видаляти з файлу байти;
* шукати зазначену користувачем послідовність байтів або символів;
* переходити на конкретну адресу;
* скасовувати (англ. undo) та повертати (англ. redo) внесені користувачем зміни.

На виході повинна вийти компактна програма з мінімально-можливим набором функцій.

## 3.2. Проектування програмного забезпечення

Після дослідження проблеми та визначення способу її вирішення першим етапом розробки програмного забезпечення є проектування та розробка безпосередньо програмного продукту.

У випадку з hex-редактором визначаються певні основні компоненти програми, кожен із яких відповідає за певну функціональність:

* Керуючий компонент: є точкою входження при запуску програми. Після ініціалізації зчитує стан вікна програми та користувацький ввід та надсилає ці дані до компоненту користувацького інтерфейсу.
* Користувацький інтерфейс. Відповідає за відображення змісту файлу, над яким працює програма, та меню функцій. Оброблює користувацький ввід.
* Логіка перегляду та редагування документів. Словом «документ» можна описати будь-який файл у контексті його редагування. Клас, що описує такий документ у програмі, безпосередньо містить логіку редактора, зокрема зчитування та запис даних.

У програмі буде відбуватися розмежування функціональності по різних класах. Таким чином, кожен із цих компонентів абстрагується один від одного, дозволяючи вносити зміни в один із цих компонентів, не змінюючи інший.

## 3.3. Створення керуючого компонента

Керуючим компонентом є статичний клас Program. Він містить метод Main, який власне є вхідною точкою програми. При запуску програми цей метод задає початковий стан програми та консолі: задає розмір вікна та потім викликає метод ініціалізації користувацького інтерфейсу. Після ініціалізації програма входить у цикл, із якого, щоб вийти, потрібно, щоб користувацький інтерфейс установив відповідний флаг.

Тіло цього циклу виділене в окремий метод. Ця функція у свою чергу перевіряє, якщо змінився розмір вікна, та наявність користувацького вводу. В обох випадках надсилаються відповідні події: WindowSizeChanged і KeyPress.

У .NET і мові C# події засновані на моделі делегатів. Коли код підписується на ці події, він зазначає в них посилання на методи для виконання певних дій при виникненні цих сигналів. Таким чином, коли програма викликає подію, виконуються методи, які підписані на неї.[10]

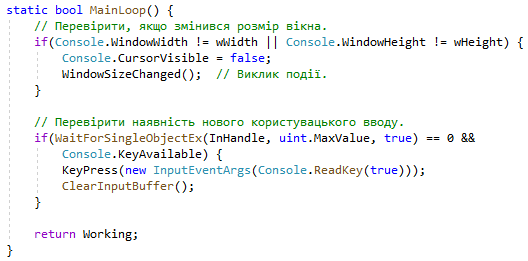


Рисунок 2. Метод, виділений з тіла головного циклу програми

Варто звернути увагу на метод WaitForSingleObjectEx. .NET Framework, як уже зазначено, створена для роботи лише в Windows, тому вона містить багато класів і методів, які насправді є просто «красиво поданими» функціями Win32. Але .NET вміє не все, що вміє Windows API загалом. Тому для використання деяких можливостей цього інтерфейсу потрібно застосувати атрибут DllImportAttribute, що дозволяє добавляти в програму функції з файлів DLL.[7]

WaitForSingleObjectEx чекає, поки зазначений за вказівником об’єкт не отримає сигнал, або поки не вийде час очікування.[8] Таким чином, у даній програмі ця функція застосовується для очікування зміни стану вікна або нових даних з клавіатури.

А щоб отримати вказівник, що вказує на ввід з клавіатури (stdin), треба застосувати GetStdHandle. Ця функція використовується, щоб отримати вказівник на дескриптор стандартного вводу/виводу.[9]

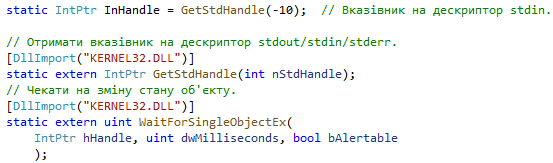


Рисунок 3. Поле зі вказівником і методи, що посилаються на одну з бібліотек Win32

## 3.4. Створення користувацького інтерфейсу

Клас UI відповідає за користувацький інтерфейс загалом. Він оброблює події від керуючого компонента про ввід з клавіатури та зміну розміру вікна. Щоб мати можливість виконувати певні дії при виникненні цих подій, метод Initialize призначає відповідні методи.

За обробку вводу з клавіатури безпосередньо відповідає метод ProcessInput, який в залежності від натисненої клавіші викликає інші процедури класу UI; за наявності вікна опцій (об’єкта класу OptionBox) натомість перенаправляє дані про натиснені клавіші до цього об’єкту. Якщо було також затиснуто клавішу Ctrl, викликається окремий метод для обробки гарячих клавіш. Варто зазначити детальну поведінку деяких із них.

Комбінація Ctrl+G дозволяє користувачеві перейти на певну адресу файлу. Справа від «матриці» з шістнадцяткових чисел з’являється діалогове вікно, у якому треба ввести нову адресу. При введенні символу з клавіатури він вставляється в кінець адреси, а сама адреса зміщується на один символ вліво. Символи, що не вміщується в 9-розрядне шістнадцяткове число, відкидаються.

При натисненні клавіш Ctrl+O або Ctrl+S викликається діалогове вікно відповідно відкриття або збереження файлу. Якщо при спробі виконати зазначену дію над файлом виникає помилка, це вікно з’являється знову.

Ctrl+Z і Ctrl+Y відповідно є функціями скасування (undo) та повернення (redo) змін у файлі. При кожному натисканні цих комбінацій курсор переміщується туди, де було внесено зміну.

Клас UI має метод ShowBytes для відображення самих даних файлу, над яким працює програма. Сам ж користувацький інтерфейс не має цих даних; він звертається до редакторської логіки, що буде описана в наступному підрозділі.

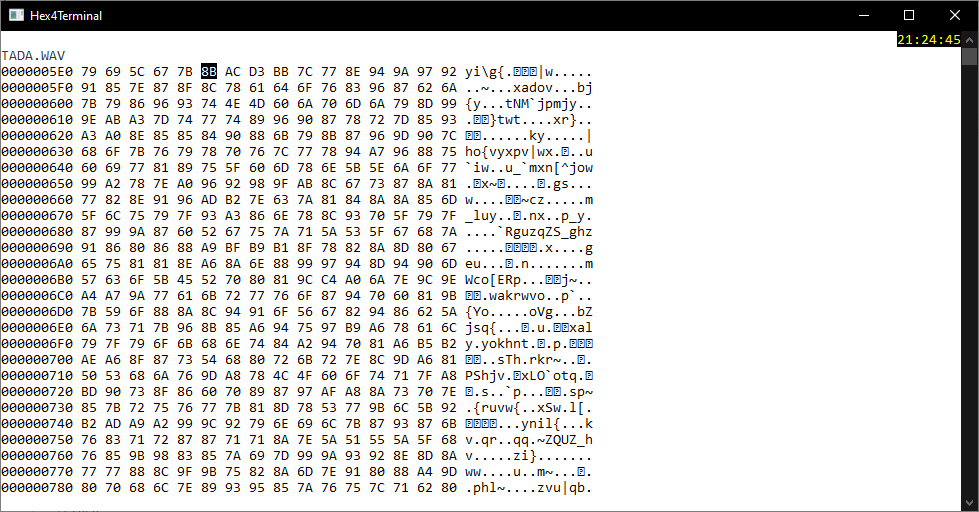


Рисунок 4. Знімок вікна програми Hex4Terminal (з метою ілюстрації кольорову схему інтерфейсу змінено)

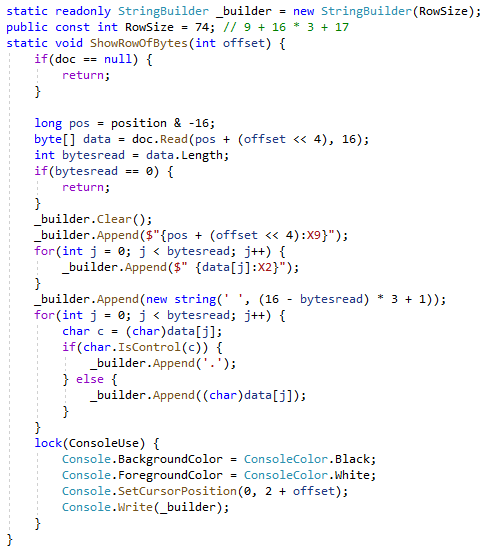


Рисунок 5. Деякі з полів класу та метод для відображення одного рядка байтів на екрані

## 3.5. Розробка логіки редактора

Клас Document містить саму логіку редагування файлів. Він лише вміє:

* зчитувати байти файлу з зазначеної адреси;
* вставляти, видаляти або перезаписувати певну кількість даних у зазначеній адресі;
* відкривати та зберігати файли.

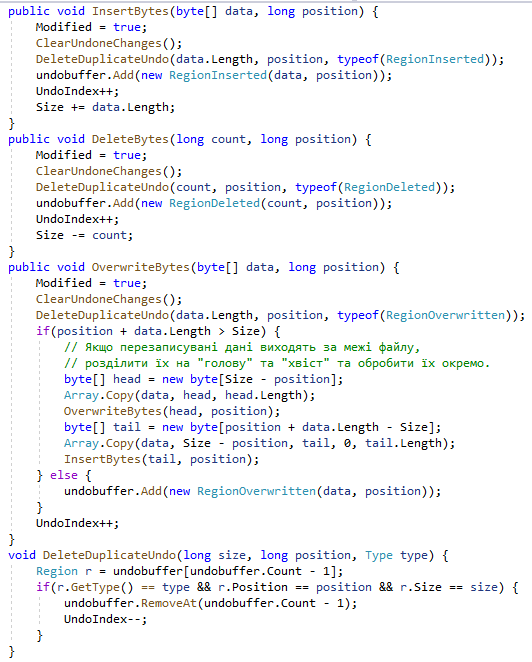


Рисунок 6. Методи внесення змін до файлу

У редакторській логіці реалізовано необмежений буфер скасування змін. Доступ до нього ззовні реалізовано за допомогою методів Undo та Redo. Внутрішня ж логіка полягає в зберіганні внесених змін в об’єктах типу Region («Ділянка») — ділянках файлу, де були внесені зміни (сам Region є абстрактним класом, від якого походять класи RegionInserted, RegionDeleted і RegionOverwritten), та в проходженні через ці об’єкти при зчитуванні даних з файлу. Для зчитування даних через undo-буфер застосовується рекурсивний метод ReadFromUndo.

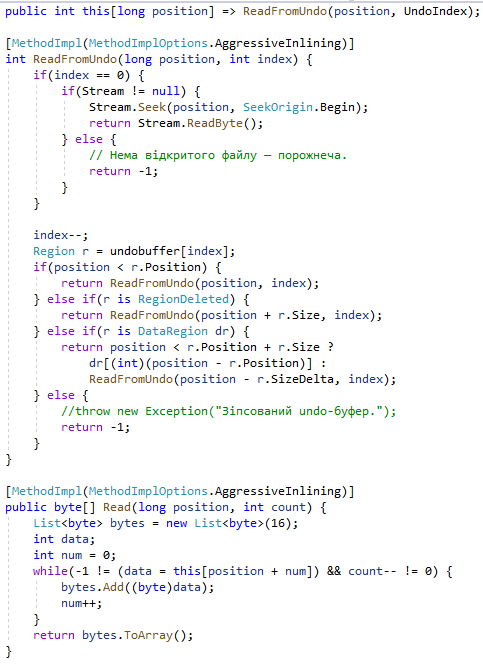


Рисунок 7. Реалізація індексатора та методу для зчитування даних через undo-буфер

Таким чином, редакторська логіка містить мінімальну функціональність для перегляду та редагування файлів. Інші ж, додаткові функції для полегшення роботи в редакторі, реалізовуються в користувацькому інтерфейсі.

## Висновки до Розділу 3

Код програми показує гарний приклад застосування об’єктно-орієнтованого програмування: відокремлення функціональних частин по класах, застосування наслідування й абстрактних класів, обмін інформацією через події.

На виході ми отримуємо програму розміром не більше 100 КБ, але з набором майже всіх потрібних базових функцій будь-якого іншого шістнадцяткового редактора, зокрема:

* відображення та редагування файлів розміром до 64 ГБ;
* підтримка гарячих клавіш;
* перехід до певної адреси;
* вставлення та видалення байтів;
* необмежений undo-буфер.

Дана програма добре підійде для тих цілей, якщо треба швидко відредагувати якийсь із файлів.

# ВИСНОВКИ

Під час виконання даної роботи було досліджене питання редагування файлів довільного або невідомого формату. Вирішенням цього питання став редактор двійкових файлів, що вміє редагувати будь-який файл та в розробці якого було застосоване ООП.

Як показує практика, об’єктно-орієнтоване програмування справді є дуже корисним нововведенням у розробці програмного забезпечення. Тепер розробник може опиратися на «об’єктне» мислення, створюючи в коді умовні моделі речей із реального життя, що взаємодіють між собою.

Завдяки розмежуванню функціональності на frontend- (користувацький інтерфейс) і backend-частини, шістнадцятковий редактор можна незалежно розширювати з метою покращення зовнішнього вигляду інтерфейсу та засобів взаємодії та збільшення швидкодії редакторської логіки.

Оскільки програма націлена на якнайменший розмір кінцевого виконавчого файлу, вона не використовує графічний інтерфейс, а натомість використовує вікно консолі. Єдиний момент, коли застосовується графічний інтерфейс, — вибір файлу на комп’ютері для відкривання або зберігання.

Програма застосовує лише .NET Framework 4.8 і Win32, що надає перевагу в сумісності з застарілими та майбутніми версіями Windows.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ярослав Скленар. Ознайомлення з ООП у мові Simula. [Електронний ресурс] URL: <http://staff.um.edu.mt/jskl1/talk.html> (дата звернення 30.04.2021)
2. Hansen Hsu. Introducing the Smalltalk Zoo. [Електронний ресурс] URL: <https://computerhistory.org/blog/introducing-the-smalltalk-zoo-48-years-of-smalltalk-history-at-chm/> (дата звернення 30.04.2021)
3. Дерк Луїс. C та C++. Довідник // оригінальне видання Markt&Technik, 1996 // видання російською Восточная Книжная Компания, 1997. – ISBN 5-89350-079-2, ISBN 3-8272-5066-8 (с. 21-23)
4. Індекс TIOBE. [Електронний ресурс] URL: <https://www.tiobe.com/tiobe-index/> (дата звернення 21.04.2021)
5. Офіційний сайт HHD Software. Free Hex Editor Neo. [Електронний ресурс] URL: <https://www.hhdsoftware.com/free-hex-editor> (дата звернення 30.04.2021)
6. Офіційний сайт mh-nexus. HxD. [Електронний ресурс] URL: <https://mh-nexus.de/en/hxd/> (дата звернення 30.04.2021)
7. Офіційна технічна документація Microsoft. Клас DllImportAttribute. [Електронний ресурс] URL: <https://docs.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.dllimportattribute> (дата звернення 30.04.2021)
8. Офіційна технічна документація Microsoft. Функція WaitForSingleObjectEx. [Електронний ресурс] URL: <https://docs.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-waitforsingleobjectex> (дата звернення 30.04.2021)
9. Офіційна технічна документація Microsoft. Функція GetStdHandle. [Електронний ресурс] URL: <https://docs.microsoft.com/en-us/windows/console/getstdhandle> (дата звернення 30.04.2021)
10. Офіційна технічна документація Microsoft. Обробка та виклик подій. [Електронний ресурс] URL: <https://docs.microsoft.com/en-us/dotnet/standard/events/> (дата звернення 06.05.2021)

# ДОДАТКИ

Додаток А.

Вміст файлу Document.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Runtime.CompilerServices;

namespace Hex4Terminal {

class Document {

FileStream Stream;

public string Name {

get; private set;

}

public long Size {

get; private set;

}

public bool Modified {

get; private set;

}

readonly List<Region> undobuffer = new List<Region>(8);

public int UndoIndex {

// Якщо це значення дорівнює 0, це означає,

// що файл ще не змінювався з моменту відкриття.

get; private set;

}

public int UndoCount {

get => undobuffer.Count;

}

public Document() {

// Створити новий файл.

Stream = null;

Name = "untitled";

}

public Document(string path) {

// Відкрити файл на комп'ютері.

Stream = File.OpenRead(path);

Size = Stream.Length;

Name = Path.GetFileName(Stream.Name);

}

~Document() {

if(Stream != null) {

Stream.Dispose();

}

}

public int this[long position] =>

ReadFromUndo(position, undobuffer.Count - 1);

[MethodImpl(MethodImplOptions.AggressiveInlining)]

int ReadFromUndo(long position, int index) {

if(index == -1) {

if(Stream != null) {

Stream.Seek(position, SeekOrigin.Begin);

return Stream.ReadByte();

} else {

// Нема відкритого файлу — порожнеча.

return -1;

}

}

Region r = undobuffer[index];

if(position < r.Position) {

return ReadFromUndo(position, index - 1);

} else if(r is RegionDeleted) {

return ReadFromUndo(position + r.Size, index - 1);

} else if(r is DataRegion dr) {

return position < r.Position + r.Size ?

dr[(int)(position - r.Position)] :

ReadFromUndo(position - r.SizeDelta, index - 1);

} else {

//throw new Exception("Зіпсований undo-буфер.");

return -1;

}

}

[MethodImpl(MethodImplOptions.AggressiveInlining)]

public byte[] Read(long position, int count) {

List<byte> bytes = new List<byte>(16);

int data;

int num = 0;

while(-1 != (data = this[position + num]) && count-- != 0) {

bytes.Add((byte)data);

num++;

}

return bytes.ToArray();

}

public void InsertBytes(byte[] data, long position) {

Modified = true;

ClearUndoneChanges();

undobuffer.Add(new RegionInserted(data, position));

UndoIndex++;

Size += data.Length;

}

public void DeleteBytes(long count, long position) {

Modified = true;

ClearUndoneChanges();

undobuffer.Add(new RegionDeleted(count, position));

UndoIndex++;

Size -= count;

}

public void OverwriteBytes(byte[] data, long position) {

Modified = true;

ClearUndoneChanges();

if(position + data.Length > Size) {

// Якщо перезаписувані дані виходять за межі файлу,

// розділити їх на

// "голову" та "хвіст" та обробити їх окремо.

byte[] head = new byte[Size - position];

Array.Copy(data, head, head.Length);

OverwriteBytes(head, position);

byte[] tail = new byte[position + data.Length - Size];

Array.Copy(data, Size - position, tail, 0, tail.Length);

InsertBytes(tail, position);

} else {

undobuffer.Add(new RegionOverwritten(data, position));

}

UndoIndex++;

}

[MethodImpl(MethodImplOptions.AggressiveInlining)]

void ClearUndoneChanges() {

if(UndoCount < undobuffer.Count) {

undobuffer.RemoveRange(UndoIndex, undobuffer.Count - UndoIndex);

}

}

// Скасувати останню зміну та повернути позицію, де вона була.

public long Undo() {

Modified = true;

return UndoIndex > 0

? undobuffer[--UndoIndex].Position

: throw new InvalidOperationException("Нема чого скасовувати!");

}

// Повернути скасовану зміну.

public long Redo() {

if(UndoIndex < undobuffer.Count) {

Modified = true;

Region r = undobuffer[UndoIndex++];

return r is DataRegion ? r.Position + r.Size : r.Position;

}

throw new InvalidOperationException("Нема чого повертати!");

}

public void Save() {

Modified = false;

int num = 0;

string name = Path.GetFileNameWithoutExtension(Stream.Name);

string ext = Path.GetExtension(Stream.Name);

string path = new FileInfo(Stream.Name).DirectoryName;

while(File.Exists($@"{path}\{name}{num}{ext}")) {

num++;

}

string bakname = $@"{path}\{name}{num}{ext}";

FileStream file = File.Create(bakname);

for(long i = 0; i < Size; i++) {

file.WriteByte((byte)this[i]);

}

file.Flush(true);

file.Dispose();

Stream.Dispose();

string newname = $@"{path}\{name}{ext}";

File.Move(bakname, newname);

Stream = File.OpenRead(newname);

UndoIndex = 0;

ClearUndoneChanges();

}

public void Save(string path) {

Modified = false;

FileStream file = File.Create(path);

for(long i = 0; i < Size; i++) {

file.WriteByte((byte)this[i]);

}

file.Flush(true);

file.Dispose();

}

}

}

Додаток Б.

Вміст файлів Region.cs, DataRegion.cs, RegionDeleted.cs RegionInserted.cs і RegionOverwritten.cs.

namespace Hex4Terminal {

abstract class Region {

public long Size {

// Розмір зміненої ділянки файлу.

get; protected set;

}

public long SizeDelta {

// На стільки змінюється розмір файлу після внесення зміни.

get; protected set;

}

public long Position {

// Де було змінено файл.

get; protected set;

}

}

}

namespace Hex4Terminal {

abstract class DataRegion: Region {

protected byte[] data;

public byte this[int index] {

get => data[index];

}

}

}

namespace Hex4Terminal {

class RegionDeleted: Region {

public RegionDeleted(long size, long position) {

Size = size;

SizeDelta = -size;

Position = position;

}

}

}

namespace Hex4Terminal {

class RegionInserted: DataRegion {

public RegionInserted(byte[] data, long position) {

this.data = data;

Size = data.Length;

SizeDelta = Size;

Position = position;

}

}

}

namespace Hex4Terminal {

class RegionOverwritten: DataRegion {

public RegionOverwritten(byte[] data, long position) {

this.data = data;

Size = data.Length;

SizeDelta = 0;

Position = position;

}

}

}

Додаток В.

Вміст файлу Program.cs.

using System;

using System.Text;

using System.Threading;

using System.Runtime.InteropServices;

namespace Hex4Terminal {

static class Program {

[STAThread]

static void Main(string[] args) {

Console.OutputEncoding = Encoding.UTF8;

Console.InputEncoding = Encoding.Unicode;

Console.Title = "Hex4Terminal";

Console.CursorVisible = false;

Console.TreatControlCAsInput = true;

// Задати розмір екрану з шириною 120 символів.

Console.WindowWidth = 120;

Console.BufferWidth = 120;

Console.WindowHeight = 30;

wHeight = 30;

wWidth = 120;

if(Console.BufferHeight < 50) {

Console.BufferHeight = 999;

}

InHandle = GetStdHandle(-10);

Console.BackgroundColor = ConsoleColor.Black;

Console.Clear();

Thread clock = new Thread(ClockLoop);

try {

if(args.Length > 0) {

UI.Initialize(args[0]);

} else {

UI.Initialize();

}

clock.Priority = ThreadPriority.Lowest;

clock.Start();

while(MainLoop()) { }

} finally {

// Якщо щось станеться, все одно завершити програму коректно.

if(clock.IsAlive) {

clock.Priority = ThreadPriority.Highest;

clock.Interrupt();

clock.Join();

}

Console.ResetColor();

Console.Clear();

Console.CursorVisible = true;

Console.TreatControlCAsInput = false;

}

}

static int wWidth;

static int wHeight;

static IntPtr InHandle = GetStdHandle(-10); // Вказівник на дескриптор stdin.

// Отримати вказівник на дескриптор stdout/stdin/stderr.

[DllImport("KERNEL32.DLL")]

static extern IntPtr GetStdHandle(int nStdHandle);

// Чекати на зміну стану об'єкту.

[DllImport("KERNEL32.DLL")]

static extern uint WaitForSingleObjectEx(

IntPtr hHandle, uint dwMilliseconds, bool bAlertable

);

static bool MainLoop() {

// Перевірити, якщо змінився розмір вікна.

if(Console.WindowWidth != wWidth || Console.WindowHeight != wHeight) {

Console.CursorVisible = false;

WindowSizeChanged(); // Виклик події.

}

// Перевірити наявність нового користувацького вводу.

if(WaitForSingleObjectEx(InHandle, uint.MaxValue, true) == 0 &&

Console.KeyAvailable) {

KeyPress(new InputEventArgs(Console.ReadKey(true)));

ClearInputBuffer();

}

return Working;

}

static void ClearInputBuffer() {

while(Console.KeyAvailable) {

Console.ReadKey(true);

}

}

static void ClockLoop() {

// Викликати подію кожні півсекунди.

// У випадку будь-якого винятка припинити роботу.

try {

do {

int ms = DateTime.Now.Millisecond;

if(ms < 500) {

Thread.Sleep(500 - ms);

} else {

Thread.Sleep(1000 - ms);

}

ClockTick();

} while(Working);

} catch { }

}

public static volatile bool Working = true;

public delegate void InputEventHandler(InputEventArgs e);

public static event InputEventHandler KeyPress = Dummy;

public static event Action WindowSizeChanged = Dummy;

public static event Action ClockTick = Dummy;

static void Dummy() { }

static void Dummy(object o) { }

}

}

Додаток Г.

Вміст файлу UI.cs

using System;

using System.Text;

using System.IO;

using Microsoft.Win32;

namespace Hex4Terminal {

static class UI {

public static object ConsoleUse = new object();

static Document doc = null;

static readonly OpenFileDialog oFileDialog = new OpenFileDialog();

static readonly SaveFileDialog sFileDialog = new SaveFileDialog();

public static void Initialize() {

InitInternal();

}

public static void Initialize(string filepath) {

doc = new Document(filepath);

string folder = Path.GetDirectoryName(Path.GetFullPath(filepath));

oFileDialog.InitialDirectory = folder;

sFileDialog.InitialDirectory = folder;

sFileDialog.Filter = "Усі файли|\*.\*";

Console.SetCursorPosition(0, 1);

Console.ForegroundColor = ConsoleColor.DarkGray;

Console.WriteLine(Path.GetFileName(filepath));

ShowBytes();

HighlightCursor();

InitInternal();

}

static void InitInternal() {

Program.KeyPress += ProcessInput;

Program.ClockTick += UpdateClock;

Program.WindowSizeChanged += RedrawScreen;

}

static readonly StringBuilder \_builder = new StringBuilder(RowSize);

public const int RowSize = 74; // 9 + 16 \* 3 + 17

static void ShowRowOfBytes(int offset) {

if(doc == null) {

return;

}

long pos = position & -16;

byte[] data = doc.Read(pos + (offset << 4), 16);

int bytesread = data.Length;

if(bytesread == 0) {

return;

}

\_builder.Clear();

\_builder.Append($"{pos + (offset << 4):X9}");

for(int j = 0; j < bytesread; j++) {

\_builder.Append($" {data[j]:X2}");

}

\_builder.Append(new string(' ', (16 - bytesread) \* 3 + 1));

for(int j = 0; j < bytesread; j++) {

char c = (char)data[j];

if(char.IsControl(c)) {

\_builder.Append('.');

} else {

\_builder.Append((char)data[j]);

}

}

lock(ConsoleUse) {

//Console.BackgroundColor = ConsoleColor.Black;

//Console.ForegroundColor = ConsoleColor.White;

Console.BackgroundColor = ConsoleColor.White;

Console.ForegroundColor = ConsoleColor.Black;

Console.SetCursorPosition(0, 2 + offset);

Console.Write(\_builder);

}

}

static void ShowBytes() {

for(int i = 0; i < Console.WindowHeight - 3; i++) {

ShowRowOfBytes(i);

}

}

static void ShowLowerBytes() {

ShowRowOfBytes(Console.WindowHeight - 4);

}

static void ShowUpperBytes() {

ShowRowOfBytes(0);

}

public static OptionBox optionBox;

static void CommandKey(ConsoleKeyInfo cki) {

switch(cki.Key) {

case ConsoleKey.G:

optionBox = new GoToBox();

break;

case ConsoleKey.O:

for(; ; ) {

try {

OpenFile();

break;

} catch { }

}

break;

case ConsoleKey.S:

for(; ; ) {

try {

SaveFile();

break;

} catch { }

}

break;

case ConsoleKey.Y:

if(doc.UndoIndex < doc.UndoCount) {

position = doc.Redo();

BlankBytes();

ShowBytes();

}

break;

case ConsoleKey.Z:

if(doc.UndoIndex > 0) {

position = doc.Undo();

BlankBytes();

ShowBytes();

}

break;

}

}

static void ProcessInput(InputEventArgs e) {

if(optionBox != null) {

// Якщо наявне вікно опцій, перенаправити

// туди ввід з клавіатури.

optionBox.UserInput(e.Key);

goto InputDone;

}

UnhighlightCursor();

ConsoleKeyInfo cki = e.Key;

if((cki.Modifiers & ConsoleModifiers.Control) != 0) {

CommandKey(cki);

goto InputDone;

}

switch(cki.Key) {

case ConsoleKey.Escape:

Program.Working = false;

break;

case ConsoleKey.PageUp:

PageUpScroll();

break;

case ConsoleKey.PageDown:

PageDownScroll();

break;

case ConsoleKey.End:

ScrollToEnd();

break;

case ConsoleKey.Home:

ScrollToStart();

break;

case ConsoleKey.LeftArrow:

MoveCursorLeft();

break;

case ConsoleKey.UpArrow:

ScrollUp();

break;

case ConsoleKey.RightArrow:

MoveCursorRight();

break;

case ConsoleKey.DownArrow:

ScrollDown();

break;

case ConsoleKey.Insert:

doc.InsertBytes(new byte[] { 0 }, position++);

ShowBytes();

break;

case ConsoleKey.Delete:

doc.DeleteBytes(1, position);

ShowBytes();

break;

case ConsoleKey.D0:

case ConsoleKey.D1:

case ConsoleKey.D2:

case ConsoleKey.D3:

case ConsoleKey.D4:

case ConsoleKey.D5:

case ConsoleKey.D6:

case ConsoleKey.D7:

case ConsoleKey.D8:

case ConsoleKey.D9:

doc.OverwriteBytes(new byte[] {

(byte)((doc[position] << 4) +

cki.Key - ConsoleKey.D0)

},

position);

ShowUpperBytes();

break;

case ConsoleKey.A:

case ConsoleKey.B:

case ConsoleKey.C:

case ConsoleKey.D:

case ConsoleKey.E:

case ConsoleKey.F:

doc.OverwriteBytes(new byte[] {

(byte)((doc[position] << 4) +

cki.Key - ConsoleKey.A + 10)

},

position);

ShowUpperBytes();

break;

case ConsoleKey.NumPad0:

case ConsoleKey.NumPad1:

case ConsoleKey.NumPad2:

case ConsoleKey.NumPad3:

case ConsoleKey.NumPad4:

case ConsoleKey.NumPad5:

case ConsoleKey.NumPad6:

case ConsoleKey.NumPad7:

case ConsoleKey.NumPad8:

case ConsoleKey.NumPad9:

doc.OverwriteBytes(new byte[] {

(byte)((doc[position] << 4) +

cki.Key - ConsoleKey.NumPad0)

},

position);

ShowUpperBytes();

break;

}

InputDone:

if(optionBox == null) {

HighlightCursor();

}

}

public static void GoTo(long pos) {

position = pos > doc.Size ? doc.Size : pos;

BlankBytes();

ShowBytes();

}

public static void OpenFile() {

if(oFileDialog.ShowDialog().GetValueOrDefault(false)) {

doc = new Document(oFileDialog.FileName);

BlankBytes();

position = 0;

ShowBytes();

}

}

public static void SaveFile() {

if(sFileDialog.ShowDialog().GetValueOrDefault(false)) {

doc.Save(sFileDialog.FileName);

}

}

static long position = 0;

//static int selectionsize = 1;

//static int cursordata;

static void HighlightCursor() {

int offset = (int)position & 15;

int data = doc[position];

lock(ConsoleUse) {

Console.SetCursorPosition(10 + offset \* 3, 2);

//Console.BackgroundColor = ConsoleColor.White;

//Console.ForegroundColor = ConsoleColor.Black;

Console.BackgroundColor = ConsoleColor.Black;

Console.ForegroundColor = ConsoleColor.White;

if(data == -1) {

Console.Write(" ");

} else {

Console.Write($"{data:X2}");

}

}

}

static void UnhighlightCursor() {

int offset = (int)position & 15;

int data = doc[position];

lock(ConsoleUse) {

Console.SetCursorPosition(10 + offset \* 3, 2);

//Console.BackgroundColor = ConsoleColor.Black;

//Console.ForegroundColor = ConsoleColor.White;

Console.BackgroundColor = ConsoleColor.White;

Console.ForegroundColor = ConsoleColor.Black;

if(data == -1) {

Console.Write(" ");

} else {

Console.Write($"{data:X2}");

}

}

}

static void MoveCursorLeft() {

if(position != 0) {

long oldpos = position;

position--;

if((oldpos & -16) != (position & -16)) {

position += 16;

ScrollUp();

}

}

}

static void MoveCursorRight() {

if(position < doc.Size) {

long oldpos = position;

position++;

if((oldpos & -16) != (position & -16)) {

position -= 16;

ScrollDown();

}

}

}

static void ScrollDown() {

if(position + 16 < doc.Size) {

position += 16;

lock(ConsoleUse) {

Console.MoveBufferArea(

0, 3, RowSize, Console.WindowHeight - 4, 0, 2);

}

ShowLowerBytes();

}

}

static void ScrollUp() {

if(position > 15) {

position -= 16;

lock(ConsoleUse) {

Console.MoveBufferArea(

0, 2, RowSize, Console.WindowHeight - 4, 0, 3);

}

ShowUpperBytes();

}

}

static void ScrollToStart() {

if(position != 0) {

BlankBytes();

position = 0;

ShowBytes();

}

}

static void ScrollToEnd() {

if(position != (doc.Size & -16)) {

BlankBytes();

position = doc.Size & -16;

ShowBytes();

}

}

static void PageUpScroll() {

if(position - ((Console.WindowHeight - 4) << 4) < 0) {

ScrollToStart();

return;

}

BlankBytes();

position -= (Console.WindowHeight - 4) << 4;

ShowBytes();

}

static void PageDownScroll() {

if(position + ((Console.WindowHeight - 4) << 4) >= doc.Size) {

ScrollToEnd();

return;

}

BlankBytes();

position += (Console.WindowHeight - 4) << 4;

ShowBytes();

}

static void BlankBytes() {

lock(ConsoleUse) {

Console.BackgroundColor = ConsoleColor.Black;

Console.ForegroundColor = ConsoleColor.White;

for(int i = 0; i < Console.WindowHeight - 3; i++) {

Console.SetCursorPosition(0, 2 + i);

Console.Write(new string(' ', RowSize));

}

}

}

static void UpdateClock() {

lock(ConsoleUse) {

Console.SetCursorPosition(Console.WindowWidth - 8, 0);

Console.BackgroundColor = ConsoleColor.Black;

Console.ForegroundColor = ConsoleColor.Yellow;

DateTime time = DateTime.Now;

if(time.Millisecond < 500) {

Console.Write($"{time:HH:mm:ss}");

} else {

Console.Write($"{time:HH mm ss}");

}

}

}

public static void RedrawScreen() {

Console.Clear();

ShowBytes();

}

}

}

Додаток Ґ.

Вміст файлів OptionBox.cs і GoToBox.cs.

using System;

namespace Hex4Terminal {

abstract class OptionBox {

public abstract void UserInput(ConsoleKeyInfo cki);

}

}

using System;

namespace Hex4Terminal {

class GoToBox: OptionBox {

public override void UserInput(ConsoleKeyInfo cki) {

switch(cki.Key) {

case ConsoleKey.Enter:

UI.GoTo(num);

goto case ConsoleKey.Escape;

case ConsoleKey.Escape:

UI.optionBox = null;

ClearBox();

break;

case ConsoleKey.D0:

case ConsoleKey.D1:

case ConsoleKey.D2:

case ConsoleKey.D3:

case ConsoleKey.D4:

case ConsoleKey.D5:

case ConsoleKey.D6:

case ConsoleKey.D7:

case ConsoleKey.D8:

case ConsoleKey.D9:

num = ((num << 4) & 0xF\_FFFF\_FFFF)

+ (cki.Key - ConsoleKey.D0);

UpdateDisplay();

break;

case ConsoleKey.A:

case ConsoleKey.B:

case ConsoleKey.C:

case ConsoleKey.D:

case ConsoleKey.E:

case ConsoleKey.F:

num = ((num << 4) & 0xF\_FFFF\_FFFF)

+ (cki.Key - ConsoleKey.A + 10);

UpdateDisplay();

break;

case ConsoleKey.NumPad0:

case ConsoleKey.NumPad1:

case ConsoleKey.NumPad2:

case ConsoleKey.NumPad3:

case ConsoleKey.NumPad4:

case ConsoleKey.NumPad5:

case ConsoleKey.NumPad6:

case ConsoleKey.NumPad7:

case ConsoleKey.NumPad8:

case ConsoleKey.NumPad9:

num = ((num << 4) & 0xF\_FFFF\_FFFF)

+ (cki.Key - ConsoleKey.NumPad0);

UpdateDisplay();

break;

}

}

static long num = 0;

public GoToBox() {

lock(UI.ConsoleUse) {

Console.BackgroundColor = ConsoleColor.Black;

Console.ForegroundColor = ConsoleColor.Green;

Console.SetCursorPosition(UI.RowSize, 3);

Console.Write("Введiть адресу переходу");

}

UpdateDisplay();

}

void ClearBox() {

lock(UI.ConsoleUse) {

Console.BackgroundColor = ConsoleColor.Black;

Console.SetCursorPosition(UI.RowSize, 3);

Console.Write(" ");

Console.SetCursorPosition(UI.RowSize, 4);

Console.Write(" ");

}

}

void UpdateDisplay() {

lock(UI.ConsoleUse) {

Console.BackgroundColor = ConsoleColor.DarkBlue;

Console.ForegroundColor = ConsoleColor.DarkYellow;

Console.SetCursorPosition(UI.RowSize, 4);

Console.Write($"{num:X9}h");

}

}

}

}

Додаток Д.

Вміст файлу InputEventArgs.cs.

using System;

namespace Hex4Terminal {

class InputEventArgs: EventArgs {

public ConsoleKeyInfo Key {

get;

}

public InputEventArgs(ConsoleKeyInfo key) {

Key = key;

}

}

}

**Київський національний торговельно-економічний університет**

# Рецензія на курсову роботу (проект) і результат захисту

Студента **Фєфєлов Максим Андрійович**

2 курсу 12 групи факультету інформаційних технологій

Курсова робота (проект) з «Об’єктно-орієнтоване програмування»

(назва навчальної дисципліни)

Тема Розробка програмного продукту «Hex-редактор» мовою програмування C#

Реєстраційний № \_\_\_\_, дата одержання 11.05.2020р.

Науковий керівник асист. Гнатченко Д.Д.

(вчене звання, прізвище, ініціали)

## Зміст рецензії

У курсовій роботі студента Фєфєлов М.А. на тему «Розробка програмного продукту «Hex-редактор» мовою програмування C#» зазначено актуальність дослідження, мету, об’єкт, предмет та задачі дослідження.

У теоретичній частини курсової роботи студент розглянув питання наявності необхідної функціональності програми. Також було розроблено принцип роботи даного застосунку. Автором детально описані та охарактеризовані користувацький інтерфейс і внутрішню логіку програми. У третьому розділі подано детальний опис розробки додатку в середовищі Visual Studio. В цілому, курсова робота студента Фєфєлов М.А. на тему «Розробка програмного продукту «Hex-редактор» мовою програмування C#» оформлена у відповідності до Вимог, може бути допущена до захисту та заслуговує на позитивну оцінку.

Допущено до захисту “11” травня 2021 р.

Захист планується о 10:30 “14” травня 2021 р.

(час)

кафедра інженерії програмного забезпечення та кібербезпеки

(місце роботи комісії)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис наукового керівника)

Курсова робота захищена “14” травня 2021 р.

з оцінкою «відмінно» 98 балів (A)

(за шкалою КНТЕУ, національною шкалою та шкалою ЄКТС)

**Комісія:**

1. \_\_\_\_\_\_\_\_\_\_\_\_\_ Бебешко Б.Т.

(підпис) (прізвище, ініціали)

1. \_\_\_\_\_\_\_\_\_\_\_\_\_ Жирова Т.О.

(підпис) (прізвище, ініціали)

1. \_\_\_\_\_\_\_\_\_\_\_\_\_ Хорольська К.В.

(підпис) (прізвище, ініціали)