**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА**

**КУРСОВА РОБОТА**

На тему Регулярні вирази

Смакула Владислав Ярославович

Київ

КНУ 2023

ЗМІСТ

* ПОБУДОВА ЗАДАЧІ
* РЕАЛІЗАЦІЯ
* ТЕСТУВАННЯ
* ПЕРЕХРЕСНЕ ТЕСТУВАННЯ
* ПРИМІТКА
* ВИСНОВОК
* ДЖЕРЕЛА

# ПОБУДОВА ЗАДАЧІ

Як і в будь-якому хорошому проекті перед тим, як щось робити потрібно описати проект, як очікується його структура і чи вона взагалі можлива. Поскільки за умовою задачі потрібно виконати дані пункти за допомогою трьох різних методів, було вирішено поділити проект на 3 папки:

C\_RegularExpressions/

C++\_RegularExpressions/

C++Re\_RegularExpressions/

Кожна з цих папок відповідно має в собі реалізацію вищезазначених пунктів з використанням відповідних методів. Поскільки функції в даних папках будуть виконувати однаковий функціонал, то тести можна винести в окрему папку: ValidResponses/, в ній зберігаються всі відповіді на тести, для того, щоб потім можна було б порівняти їх і визначити чи функція яку ми створили відпрацювала коректно. Також винесемо функцію для перехресного тестування методів реалізованих на C та C++ в окрему папку для зручності. Пізніше буде добавлена папка Helpers в яку будуть винесені додаткові методи які будуть використовуватися іншими функціями.

# РЕАЛІЗАЦІЯ

Для початку розглянемо реалізовані методи. Документацію по інпуту\аутпуту функцій можна знайти в .h файлах проекту. Першопочатково були реалізовані функції на C, такі як: getHyperlinksFromFile(), getDatesFromFile(), replaceAllDatesToCurrent() та unifyAllNumbersInFile() дані функції по суті служать мостом між внутрішньою логікою і файлами. Весь їх функціонал складається з того, що вони будуть зчитувати дані з файлу і викликати для кожного зчитаного слова функцію яка саме буде займатися парсингом потрібних даних. В папках C++\_RegularExpressions/(префікс Cpp) і C++Re\_RegularExpressions/(префікс Re) дані функції будуть трохи поміняні під синтаксис C++, але їхній функціонал не поміняється, вони продовжать служити лише проміжною функцію для зчитування даних і повернення помилок в разі невдалого відпрацювання. Єдина функція яка вирізняється з вище названих це replaceAllDatesToCurrent() оскільки в собі вона ще має логіку з отриманням сьогоднішньої дати і записуванням її в форматі "%Y %B %d". Цю логіку було вирішено залишити без створення окремої функції, оскільки в собі вона не несе комплексного функціоналу. Тепер розглянемо функції, які саме займаються парсингом.

## parseInlineLink()

Дана функція займається перевіркою, чи дане слово є гіперпосиланням, формат: [word](link). Дана функція має досить схожу структуру на С та C++, але при використанні регулярного виразу спрощується, оскільки нам не потрібно перевіряти окремо відкриваючі і закриваючі дужки, бо ми можемо все це помістити в один регулярний вираз: "(\[(.+)\]\(([^)]+)\))" який виконує той самий функціонал, але спрощує читабельність коду.

## parseInlineDate()

Дана функція займається перевіркою чи є надане слово в одному з форматів. При реалізації на С та C++, дана логіка є досить простою, адже вона має в собі звичайний вираз з використанням OR який провіряє чи підходить дане слово під формат, а оскільки OR спиняється коли він стає True то ми спинимося, коли слово буде підходити за форматом. З використанням regex даний функціонал знову ж таки спрощується, адже тепер всі ці дати ми можемо об’єднати буквально в два регулярні вирази(ми використовуємо не один, оскільки для деяких форматів номери груп будуть відрізнятися). Також при використанні регулярних виразів було вирішено винести форматування в окрему функцію для уникнення повторення коду. Дана функція буде використовуватися для getDatesFromFile() і replaceAllDatesToCurrent() оскільки в обох випадках нам потрібно перевірити чи є дане слово датою.

## parseInlineNumber()

Дана функція спочатку займається тим, що заміняє , на . для уніфікації(при реалізації за допомогою регулярних виразів ми можемо позбутися цієї логіки, оскільки все що від нас буде потрібно це в регулярний вираз добавити [\.,] що буде шукати . або ,), після чого провіряє, чи можливо конвертувати дане слово в тип double, при успіху повертає строку в відповідному форматі.

# ТЕСТУВАННЯ

Ми створили функції, але тепер нам потрібно їх протестувати, для цього спочатку для кожного з них зробимо .h файл, щоб ми могли викликати дані функції з інших файлів. В них добавимо getHyperlinksFromFile(), getDatesFromFile(), replaceAllDatesToCurrent() та unifyAllNumbersInFile(). Також для коду на C добавимо extern "C" для того, щоб потім ми могли імпортувати ці функції з .cpp файлу при перехресному тестуванні. Далі створимо \_test файли. В трьох реалізаціях їх структура по суті аналогічна. В нас є autoFileTest() для автоматичного тестування, manualFileTest() для ручного та RegularExpressionsTest() для саме виклику тестування:

## RegularExpressionsTest()

Дана функція служить лише входом, який питає в юзера який тип тестування він хоче викликати, після чого викликає відповідну функцію

## autoFileTest()

Дана функція слугує для автоматичного тестування. Вона створює папку “test” в якій вона створює вхідні тестові файли за допомогою copyFile() функції з Helpers.h. Після чого викликає по черзі функції getHyperlinksFromFile(), getDatesFromFile(), replaceAllDatesToCurrent() та unifyAllNumbersInFile(), та викликає для кожного з них функцію processTest() яка відповідно викликає compareFiles() яка порівнює їх вихідні дані з даними які зберігаються в ValidResponses/.

## manualFileTest()

Дана функція запитує в юзера назви файлів для тестування функцій, після чого провіряє чи існує такий файл за допомогою методу fileExists() з Helpers.h і якщо файл не існує, то викликає функцію getUserInputAndWriteToFile() яка запитує в юзера чи він хоче ввести дані файлу з консолі і при позитивній відповіді зчитує дані до поки юзер не введе ~stop. Після чого в не залежності від того існує чи не існує вхідний файл викликається відповідна функція. Таким чином ми залишаємо логіку того, що дані для функцій мають зчитуватися з файлів, але і задовольняєм умову того, що юзер має мати можливість вводити тестові дані з консолі.

# ПЕРЕХРЕСНЕ ТЕСТУВАННЯ

За умовою курсової роботи нам потрібно перевірити чи функції на C та C++ відпрацьовують однаково. Для цього для початку ми створити .h файли для функції для тестування, щоб ми могли їх імпортувати і викликати автоматичне тестування, а оскільки при такому типі тестування ми зберігаємо всі вихідні дані, то все що нам потрібно зробити, це викликати RegularExpressionsTest() з кожної з цих папок, після чого викликати processTest() який порівняє чи вихідні дані цих функцій є однаковими. Вищеописана логіка якраз таки і була використана в функції CrossTest() з RegularExpressions\_cross\_test.cpp. Також для неї було створено .h файл для того, щоб пізніше її можна було імпортувати і викликати з іншого місця.

# ПРИМІТКА

В даному тексті було описано лише загальний принцип побудови проекту, для повного ознайомлення варто оглянути вихідний код, оскільки в ньому в конкретиці описані вхідні\вихідні дані функції і принцип їх роботи.

# ВИСНОВОК

При виконанні даної курсової роботи мені вдалося познайомитися з використанням регулярних виразів на C мовах і на практиці побачити те, наскільки вони не просто спрощують синтаксис, а і логіку програми в загальному, адже комплексну логіку можна помістити в досить компактні і зрозумілі вирази. Хоча в процесі виконання курсової роботи я використовував базові методи регулярних виразів, це надало мені можливість детально ознайомитися із синтаксисом їх використання. Я зрозумів, що вони можуть бути не лише корисними, але і у деяких випадках навіть критичними для ефективної обробки даних у програмах.

# ДЖЕРЕЛА

<https://regexr.com/>

<https://learn.microsoft.com/en-us/>

<https://stackoverflow.com/>

<https://www.geeksforgeeks.org/>