Підготував студент 2-го курсу

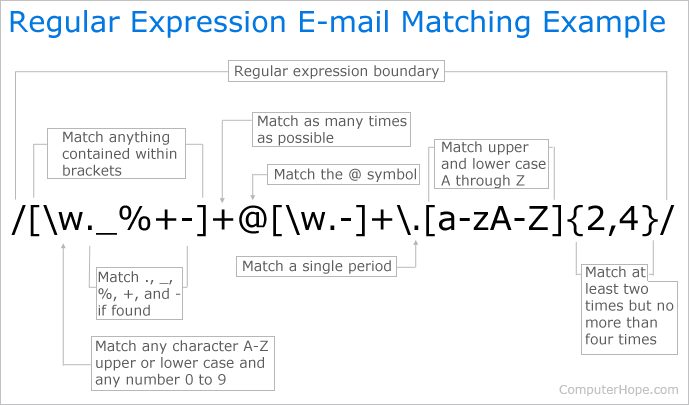
Табала Матвій

Комп’ютерна математика 2

Завдання 12. Регулярні вирази.

Основи роботи з регулярними виразами.

Перед тим, як розбирати саме мій проект, пройдемося по загальним фактам про регулярні вирази. Дізнаємося, що це таке, як воно працює і чи слід його постійно використовувати.



Регулярні вирази - це інструменти зіставлення рядків, які використовуються для ідентифікації узагальнених підпослідовностей символів у рядку.

Регулярні вирази в C ++ є потужними інструментами для обробки текстових даних. Ви можете використовувати певні шаблони, які ви визначили, для пошуку, модифікації та перевірки тексту.

Ці вирази базуються на спеціальній синтаксичній структурі, яка визначає шаблони символів, які ви хочете шукати в тексті. Ви можете використовувати метасимволи, квантори, групи та інші шаблони для пошуку певних комбінацій символів.

Наприклад, якщо ви хочете знайти в тексті всі слова, що починаються на "А", ви можете використовувати регулярний вираз **‘ /A\w +/ ‘**. Ось кілька основних концепцій:

Метасимволи:

* **‘ \w ‘** означає будь-яку букву або цифру.
* **‘ \d ‘** представляє будь-яке число.
* **‘ \s ’** не містить пробілів, вкладок, розривів рядків тощо.
* **‘ . ’** позначає будь-який символ, крім символу нового рядка \n.

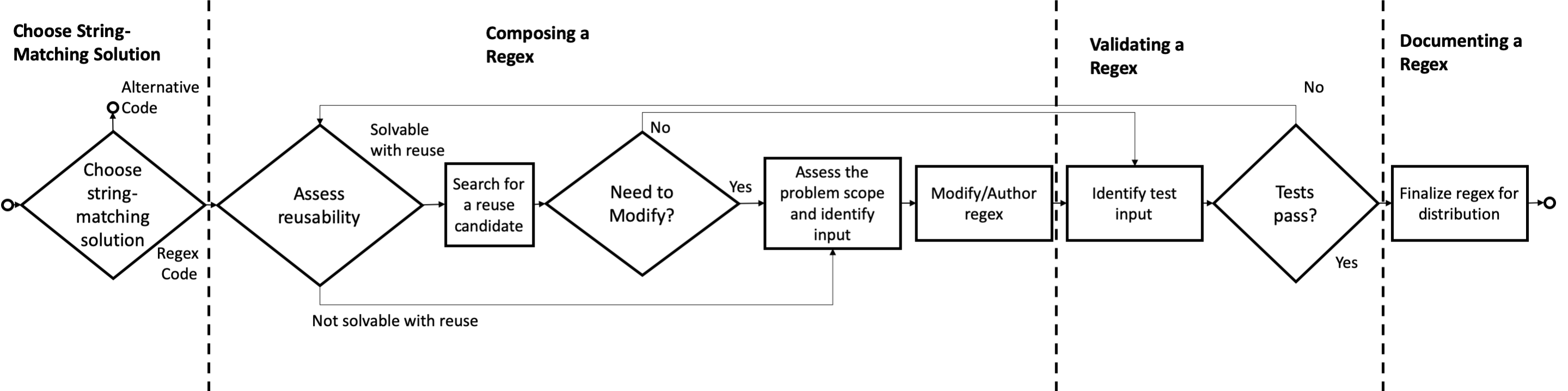
Квантор:

* **‘ + ‘** показує 1 або більше повторів попереднього символу або групи.
* **‘ \* ‘** означає нуль або більше повторень попереднього символу або групи.
* **‘ ? ‘** означає нуль або 1 повторення попереднього символу або групи.

Група:

Він використовується для групування шаблонів. Наприклад, (ABC) + відповідає будь-якій послідовності "ABC", яка повторюється 1 або більше разів.

Коли ви пишете регулярний вираз на C++, ви компілюєте цей вираз, використовуючи такі класи, як std::regex у бібліотеці. Потім ви можете застосувати його до рядка, щоб знайти відповідність, змінити певний шаблон або виконати інші операції з текстом.



(На малюнку зображені чотири етапи програмування в regex)

Регулярні вирази є дуже потужним інструментом, але їх важко зрозуміти на ранніх стадіях через їх складність та гнучкість. І навіть у таких випадках, якщо ви можете використовувати функції для обробки термінів, вам не завжди потрібно їх використовувати, особливо для дуже складних завдань, де можна використати інші та більш прості методи та бібліотеки.

Бібліотеки ‘string’ та ‘string.h’ у мові програмування C++ є корисними інструментами для роботи з рядками тексту. Іноді вони можуть бути перевагою над використанням регулярних виразів у випадках, коли потрібно виконати простіші операції з обробки тексту. Наприклад:

* Прості патерни. Якщо вам потрібно здійснити простий пошук або заміну конкретного слова чи символу у тексті, використання методів рядків (наприклад, find, replace для об'єктів типу string) може бути простішим та зрозумілішим, ніж писати регулярний вираз.
* Виконання базових операцій. Для виконання базових операцій, таких як отримання підрядка, об'єднання рядків, отримання довжини рядка тощо, бібліотеки string надають зручні та ефективні методи, які можна використовувати без необхідності розробки складних регулярних виразів.

Моя робота

Тепер розглянемо мою роботу, як приклад роботи з регулярними прикладами. Моє завдання мало 3 підпункти: а, б та в. У першому потрібно було розв’язати задачі з бібліотекою <string.h>, другому – <string>, третьому – з використанням <regex>.

Я писав код по порядку, тому йшов від пункту 1 до пункту 4.

**Почнемо з першого пункту а)**

Потрібно було знайти усі гіперпосилання та вивести їх разом з попереднім словом перед ними, незалежно від пунктуацій та інших символів.

Для цього я створив функцію, яка повертає булеве значення (true or false) – isSeparator. Далі створив функцію ‘findHyperlinks’, яка, за умови вибору користувачем вводу з файлу, відкриє цей файл та прочитає його і перевірить наявність яких-небудь гіперпосилання. Або ж просто перевірить введену користувачем строку та перевірить її. Ця інформація згодом буде записана у файл “RegExpA”, “RegExpB”, “RegExpC”, залежно від того, який варіант програми буде запущено. Перевірка на наявність будь-яких гіперпосилань зроблена за методом розбиття слів на токени у циклі та пошуку ключового слова ‘http’, якого буде достатньо для цього. Далі ми виводимо цей рядок у консоль і записуємо у файл одночасно, далі рухаємось до наступного токену.

Наступним кроком, була перевірка на правильну дату. Я створив функцію ‘isValidDate’, яка правильно перевіряла формат запису дати у файлі (консолі). Вона визначає, чи на першому місці стоїть рік, чи число дня. Повертає також булеве значення true (правда) або false (хиба). Також було створено масив з назвами місяців, щоб правильно визначати місяці і таким чином оптимізувати швидкість виконання програми, замість перевірки великої кількості умов.

Аналогічно, було створено функцію для переводу дати до єдиного формату ‘convertDates’ яка за минулим принципом проходиться по рядках за допомогою токенів, перевіряє наявність крапок та слешів (‘/’) і прибирає усі інші символи, окрім цих самих.

Далі, завдяки ‘isValidDate’ перевіряємо валідність і виводимо коректно дату і у заданому за умовою форматі.

Також створюємо ‘isNumber’, яка повертає булеве значення. Аналогічним чином проходимось по рядкам і переводимо число у єдиний формат за умовою через функцію ‘convertNumbers’.

Згодом, була створена функція ‘isValidTime’, яка на цей раз, перевіряє правильність запису часу у файлі або консолі. Точніше, чи це буде формат запису де є і години, і хвилини, і секунди або просто години і хвилини.

Знову ж таки, за попередніми прикладами, ми проходимось по рядкам, дістаємо локальний час, замінюємо будь-які дати на поточні, разом із часом. Також там присутня велика кількість умови для правильного виводу часу, бо при наявності одноцифрових значень, такі як 00, 01, 02, … , 09, програма виводить лише одну цифру. Це виправляється таким чином, але впевнений, що є багато кращих варіантів як це виправити. Важливо, що у випадку, якщо рядок містить дату, ми переходимо на новий.

В кінці кожної функції ми закриваємо файли і на цьому все.

**Другий пункт б)**

Для другої програми з пункту б) був використаний схожий підхід, але з використанням бібліотеки <sstream>, яка має дуже компактний запис і дозволяє проходитись по рядкам файлу і зчитувати їх.

Які тут є різниці в порівнянні з пунктом а):

* Замість масиву місяців, я використав словник unordered\_map, який дозволяє зручно виставляти відповідність між номером місяця та його назви.
* Замість великої кількості умов, щоб перевірити чи одноцифровий рік, місяць, день, я використав функціонал бібліотеки string, як наприклад тут:



* Для виводу часу у формату hh:mm, hh:mm:ss, я використав функції setw, setfill, щоб скоротити запис.

На мою думку, завдяки певним бібліотекам, як <string>, <sstream>, <unordered\_map>, зменшилась кількість умов та додаткових перевірок на конкретний формат запису, а також простота зіставлення номеру місяця і його назви завдяки словнику.

**Третій пункт с)**

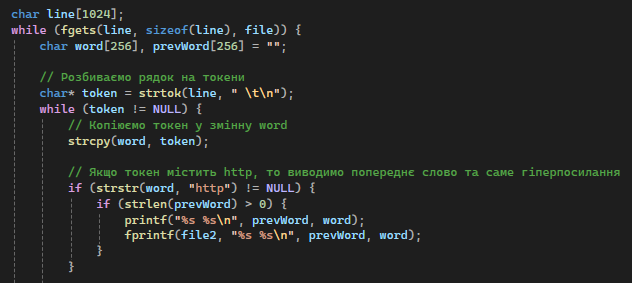
Для третьої програми вже використовувалися регулярні вирази.

В якійсь мірі, вони зменшили складність пошуку рядків, але при цьому, збільшили перевірку деяких умов (не всіх). З ним стало легко знаходити дати і гіперпосилання. Завдяки лише одному рядку регулярних виразів і одного циклу while, з легкістю можна знайти гіперпосилання.

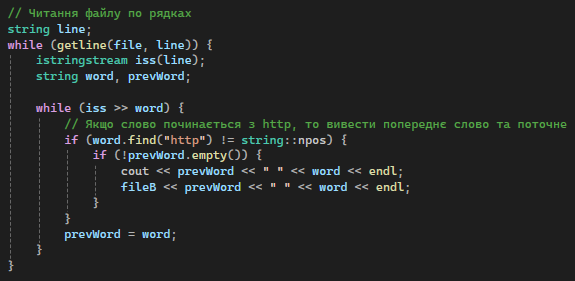
У порівнянні з варіантом а) та б) зменшалась кількість додаткових циклів для проходження по рядках, але повернулась умова з пункту а) для правильного друкування поточного часу для одноцифрових значень. Якщо б не ця гігантська умова, можна було б сказати, що розмір коду і його ефективність у рази збільшилось в порівнянні з попередніми варіантами коду, особливо для першого пункту а).

Ось приклад для проходження по рядкам і пошуку гіперпосилань:

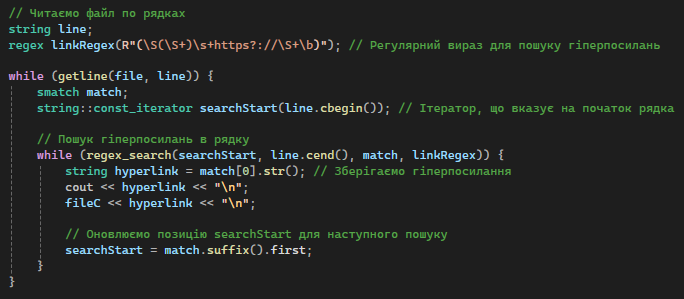
Для а)



Для б)



Для в)



На мою думку, ця робота гарно показує, що одну і ту саму задачу можна виконати багатьма способами. Багато користувачів нехтують використанням <regex>, бо він складний у написанні та використанні і доволі рідко використовується через свою складність. Набагато простіше зробити якісь цикли, умови або булеві функції, для пошуку певних елементів. Крім того, наскільки мені відомо, з точки зору безпеки, багато користувачів зазначають великий відсоток уражень у своїх кодах. Але загалом, мені було зручніше писати код пункту б) без використання регулярних виразів, бо усе набагато простіше і доступніше, і не треба постійно шукати в документації вирази для правильного пошуку в regex.