**Форма № Н-6.01**

ЗАТВЕРДЖЕНО

Наказ Міністерства освіти і науки,

молоді та спорту України

29 березня 2012 року № 384

**Чернівецький національний університет імені Юрія Федьковича**

**Факультет математики та інформатики**

(повна назва факультету)

**Кафедра прикладної математики та інформаційних технологій**

(повна назва кафедри)

**КУРСОВА РОБОТА**

на тему : Пошук у тексті засобами лексичного аналізатора

Студента 3 курсу 322 групи

напряму підготовки

6.040301 – Прикладна математика

\_\_\_\_\_\_\_Поклітар В.Д.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

(прізвище та ініціали)

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

(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)

Чернівці–2016

**АНОТАЦІЯ**

В курсовій роботі була створена програма, яка генерує шаблон та текст за шаблон і перевіряється якість виконаної студентами лабораторної роботи, пов’язаної із розпізнаванням ланцюжків по заданих шаблонах в текстових файлах.

Зміст

[**КУРСОВА РОБОТА** 1](#_Toc449106430)

[РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА 5](#_Toc449106431)

[1.1. Регулярний вираз 5](#_Toc449106432)

[1.2. Клас RegEx 5](#_Toc449106433)

[1.3.Синтаксис 6](#_Toc449106434)

[1.4. Дії 13](#_Toc449106435)

[1.4.1. Пошук 14](#_Toc449106436)

[1.4.2. Заміна 15](#_Toc449106437)

[1.4.3. Порівняння 16](#_Toc449106438)

[1.4.4. Поділ рядка 17](#_Toc449106439)

[РОЗДІЛ 2.ПРАКТИЧНА ЧАСТИНА 18](#_Toc449106440)

[2.1. Інструкція користувача 18](#_Toc449106441)

[2.2. Опис деяких процедур/функцій 23](#_Toc449106442)

[Висновки 24](#_Toc449106443)

[Перелік посилань 25](#_Toc449106444)

[Додаток 1. 26](#_Toc449106445)

[Додаток 2. 33](#_Toc449106446)

**ВСТУП**

Одним з найпотужніших інструментів пошуку в текстах є регулярні вирази.

Регулярні вирази підтримуються багатьма програмами (редакторами, системними утилітами, ядрами баз даних і так далі), але їх можливості повною мірою виявляються в мовах програмування, зокрема Java, Jscript, Visual Basic, Vbscript, Javascript, C, C++, C#, Perl, Python, Ruby, PHP, awk і тому подібне. Регулярні вирази займають центральне місце в багатьох програмах, написаних на цих мовах.

Підтримка регулярних виразів в таких різнорідних застосуваннях пояснюється тим, що вони володіють виключно багатими можливостями. На низькому рівні регулярний вираз описує якийсь фрагмент тексту. Їм можна скористатися для перевірки даних, введених користувачем, або, наприклад, для фільтрації великих об'ємів даних. На більш високому рівні регулярні вирази дозволяють управляти даними.

За допомогою регулярних виразів ви можете проводити пошук, заміну, порівняння підрядків, використовуючи шаблони. Вони складаються із звичайних символів і так званих метасимволів (metacharacters) - символів, що управляють.

# РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА

## 1.1.Лексичний аналіз

**Лексичний розбір** - це процес перетворення послідовності символів в послідовність токенів (груп символів що відповідають певним шаблонам), та визначення їх типів. Програма, чи функція що виконує лексичний аналіз, називається лексичним аналізатором, чи сканером. Часто сканер є звичайною функцією що використовується парсером (синтаксичним аналізатором), для отримання наступного токена з потоку вхідних символів в процесі компіляції. Також часто лексичний аналіз використовують для підсвітки синтаксису певних мов.

Лексичний аналізатор є скінченним автоматом, перехід в певні стани якого викликає функції, які зазвичай повертають тип лексеми, і саму лексему (токен)

Позначки часто визначаються за допомогою регулярних виразів, що зрозумілі генераторам лексичних аналізаторам. Лексичний аналізатор (чи створений автоматично приладдям на кшталт lex або вручну) читає потік символів, визначає лексеми в потоці, і обробляє їх.

## 1.1. Регулярний вираз

**Регулярний вираз —** це рядок, що описує або збігається з множиною рядків, відповідно до набору спеціальних синтаксичних правил. Вони використовуються в багатьох текстових редакторах та допоміжних інструментах для пошуку та зміни тексту на основі заданих шаблонів. Багато мов програмування підтримують регулярні вирази для роботи з рядками. Наприклад, Perl та Tcl мають потужний механізм для роботи, вбудований безпосередньо в їх синтаксис. Завдяки набору утиліт (включаючи редактор sed та фільтр grep), що входили до складу дистрибутивів Юнікс регулярні вирази стали відомими та поширеними.

Регулярний вираз (часто називається шаблон) базуються на теорії автоматів та теорії формальних мов. Ці розділи теоретичної кібернетики займаються дослідженням моделей обчислення (автомати) та способами описання та класифікації формальних мов.

Регулярний вираз є послідовністю, що описує множину рядків. Ці послідовності використовуються для того, аби дати точне описання множини, не перелічуючи всі її елементи. Це потужна технологія, призначена для роботи з рядками. Широко застосовується в скриптових мовах програмування – Perl, Jscript, VBScript.

## 1.2. Клас RegEx

Для роботи з регулярними виразами в VB. NET використовується клас RegEx, що знаходиться в просторі імен System.Text.RegularExpressions. За допомогою цього класу ви можете проводити наступні дії:

* Пошук підрядків за шаблоном
* Заміна підрядків за шаблоном
* Порівняння рядки з шаблоном
* Поділ рядка на підрядка з використанням шаблонів.

## 1.3.Синтаксис

* **Звичайні символи (літерали) і спеціальні символи (метасимволи)**

Найпростішим регулярним виразом, з якого формуються складні, є звичайний символ. Більшість символів у регулярному виразі представляють самі себе за винятком спеціальних символів (метасимволів) [ ] \ ^ $ . | ? \* + ( ) { }, яким може передувати символ \ (зворотна коса риска), котра робить метасимволи «екранованими», «захищеними» для представлення, відображення їх самих як символів тексту. Можна екранувати деяку послідовність символів, розмістивши її між \Q і \E .

|  |  |
| --- | --- |
| **Приклад** | **Відповідність** |
| a\.? | a. або a |
| a\\\\b | a\\b |
| a\[F\] | a[F] |
| \Q+-\*/\E | +-\*/ |

Аналогічно можуть бути представлені інші спеціальні символи (набір символів, що вимагають екранування, може відрізнятися залежно від конкретної реалізації). Частина символів, які в тій або іншій реалізації не вимагають екранування (наприклад, кутові дужки < >), можуть бути екрановані з міркувань зручності читання.

Залежно від інтерпретатора регулярних виразів, метасимволи «?», «+», «{», «|», «(», та «)» можуть втрачати своє спеціальне значення, замість цього слід вживати «\?», «\+», «\{», «\|», «\(», та «\)».

* **Позиція всередині рядка**

Наступні символи дозволяють позиціонувати регулярний вираз щодо елементів тексту: початку й кінця рядка, меж слова.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Представлення** | **Позиція** | **Приклад** | | **Відповідність** |
| ^ | Початок рядка | ^a | aaa aaa | |
| $ | Кінець рядка | a$ | aaa aaa | |
| \b | Межа слова | a\b | aaa aaa | |
| \ba | aaa aaa | |
| \B | Не межа слова | \Ba\B | aaa aaa | |
| \G | Попередній успішний пошук | \Ga | aaa aaa (пошук зупинився на 4-й позиції — там, де не знайшлося a) | |

* **Символьні класи (набори символів)**

Набір символів у квадратних дужках [ ] іменується символьним класом і дозволяє вказати інтерпретаторові регулярних виразів, що на даному місці в рядку може стояти один із перерахованих символів. Зокрема, [абв] задає можливість появи в тексті одного із трьох зазначених символів, а [1234567890] задає відповідність одній із цифр. Можливе зазначення діапазонів символів: наприклад, [А-Яа-я] відповідає всім літерам.

Для часто використовуваних символьних класів існують короткі позначення.

|  |  |
| --- | --- |
| **Символ** | **Опис** |
| \d | Відповідає цифрі. Еквівалентно [0-9] |
| \D | Відповідає нецифровому символу. Еквівалентно [^0-9] |
| \s | Відповідає будь-якому пробільному символу. Еквівалентно [ \f\n\r\t\v] |
| \S | Відповідає будь-якому непробільному символу. Еквівалентно [^ \f\n\r\t\v] |
| \w | Відповідає будь-якому літерному символу, цифровому й знаку підкреслення. Еквівалентно [[:word:]] |
| \W | Відповідає будь-якому символу, крім літерного символу, цифрового або підкреслення. Еквівалентно [^[:word:]] |

* **Квантифікація (пошук послідовностей)**

Квантифікатор після символу, символьного класу або групи визначає, скільки разів попередній вираз може зустрічатися. Варто враховувати, що квантифікатор може стосуватися більш ніж до одного символу в регулярному виразі, тільки якщо це символьний клас або група.

|  |  |  |  |
| --- | --- | --- | --- |
| **Представлення** | **Кількість повторень** | **Приклад** | **Відповідність** |
| {*n*} | Рівно *n* разів | colou{3}r | colouuur |
| {*m*,*n*} | Від *m* до *n* включно | colou{2,4}r | colouur, colouuur, colouuuur |
| {*m*,} | Не менше *m* | colou{2,}r | colouur, colouuur, colouuuur і т.д. |
| {,*n*} | Не більше *n* | colou{,3}r | color, colour, colouur, colouuur |

Часто використовується послідовність .\* для позначення будь-якої кількості будь-яких символів між двома частинами регулярного виразу.

Символьні класи в поєднанні із квантифікаторами дозволяють установлювати відповідності з реальними текстами. Наприклад, колонками цифр, телефонами, поштовими адресами, елементами HTML-розмітки й ін.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Представлення** | **Кількість повторень** | **Еквівалент** | **Приклад** | **Відповідність** |
| \* | Нуль або більше | {0,} | colou\*r | color, colour, colouur і т.д. |
| + | Одне або більше | {1,} | colou+r | colour, colouur і т.д. (але не color) |
| ? | Нуль або одне | {0,1} | colou?r | color, colour |

Якщо символи { } не утворюють квантифікатор, їхнє спеціальне значення ігнорується. Якщо символи { } не утворюють квантифікатор, їхнє спеціальне значення ігнорується.

* **Керуючі символи**

|  |  |  |  |
| --- | --- | --- | --- |
| **Представлення** | **Символ** | **Позначення** | **Розшифровка** |
| \t | Табуляція | HT | Horizontal tabulation |
| \v | Вертикальна табуляція | VT | Vertical tabulation |
| \r | Повернення каретки | CR | Carriage return |
| \n | Переведення рядка | LF | Line feed |
| \f | Кінець сторінки | FF | Form feed |
| \a | Дзвінок | BEL | Bell character |
| \e | Escape-Символ | ESC | Escape character |
| \b | Забій  Повинен знаходитися всередині квадратних дужок (інакше інтерпретується як межа слова). | BS | Backspace |
| \cA ... \cZ | Ctrl+A ... Ctrl+Z  Наприклад, послідовність \cM\cJ відповідає керуючим символам CR LF.  Еквівалентно \x01 ... \x1A. |  |  |

* **Жадібна й ледача квантифікація**

У деяких реалізаціях квантифікаторам у регулярних виразах відповідає максимально довгий рядок із можливих (квантифікатори є жадібними, англ. greedy). Це може стати значною проблемою. Наприклад, часто очікують, що вираз (<.\*>) знайде в тексті теги HTML. Однак, якщо в тексті є більше одного HTML-тегу, то цьому виразу відповідає цілком рядок, що містить множину тегів.

<p><b>Коледж</b> - Вас вітає <i>!!!</i>Ура)</p>.

Цю проблему можна вирішити двома способами.

1. Ураховувати символи, що не відповідають бажаному взірцю (<[^>]\*> для вищеописаного випадку).
2. Визначити квантифікатор як нежадібний (ледачий, англ. lazy) — більшість реалізацій дозволяють це зробити, додавши після нього знак питання.

Щоб виділити окремі теги, можна застосувати ледачу версію цього виразу: (<.\*?>) Їй відповідає не весь показаний вище рядок, а окремі теги (виділені кольором):

<p><b>Коледж</b> - Вас вітає <i>!!!</i> Ура)</p>.

|  |  |
| --- | --- |
| **Жадібний** | **Ледачий** |
| \* | \*? |
| + | +? |
| {*n*,} | {*n*,}? |

* **Групування**

Круглі дужки використовуються для визначення області дії й пріоритету операцій. Шаблон усередині групи обробляється як єдине ціле й може бути квантифікованим.

Одне із застосувань групування — повторне використання раніше знайдених груп символів (*підрядків*, *блоків*, *позначених підвиразів*). При обробці виразу підрядки, що знайдені за шаблоном усередині групи, зберігаються в окремій області пам'яті й отримують номер, починаючи з одиниці. Кожному підрядку відповідає пара дужок у регулярному виразі. Квантифікація групи не впливає на збережений результат, тобто зберігається лише перше входження. Зазвичай підтримується до 9 нумерованих підрядків із номерами від 1 до 9, але деякі інтерпретатори дозволяють працювати з більшою кількістю. Згодом у межах даного регулярного виразу можна використати позначення від \1 до \9 для перевірки на збіг із раніше знайденим підрядком.

Наприклад, регулярний вираз (До|ля)-\1 знайде рядок До-До або ля-ля, але пропустить рядок До-ля.

## 1.4. Дії використанні в програмі за допомогою регулярних виразів

Для твору дій з регулярними виразами необхідно створити екземпляр класу RegEx. Для цього використовується стандартний конструктор New. Він перевантажений і має дві комбінації параметрів. Ви можете задати тільки шаблон (змінна типу String), який буде використовуватися в подальшому, або шаблон і параметри об’єкта. Параметри задаються константами з перерахування RegExOptions.

Нижче наведені константи та інформація про них.

|  |  |
| --- | --- |
| **Константа** | **Опис** |
| Compiled | Вказує, що регулярний вираз буде скомпільований в збірку. Детальніше про компіляцію регулярного виразу читайте нижче. |
| ECMAScript | Встановлює ECMAScript-сумісність вираження. Може використовуватися разом з константами IgnoreCase, Multiline і Compiled. При використанні ECMAScript з іншими константами буде ініційовано виняток. |
| IgnoreCase | Визначає, що при проведенні операцій з виразами, регістр не має значення. |
| IgnorePatternWhitespace | Усуває прогалини з шаблонів і дозволяє використовувати коментарі, відмічені знаком “#”. |
| Multiline | Якщо цей прапор встановлений, то метасимволи “^” і “$” відзначають початок і кінець кожного рядка (рядки розділені знаком переведення рядка), а не початок і кінець усього висловлювання. |
| None | Вказує, що не встановлена ​​жодна з опцій. |
| RightToLeft | Встановлює, що пошук буде проводитися з права на ліво, а не навпаки. |
| SingleLine | Вказує, що операції будуть проводитися в однорядковому режимі. Ця константа змінює значення метасимвола “.”. |

### 1.4.1. Поділ рядка

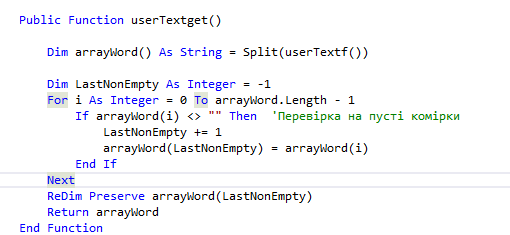
Поділ рядка з використанням регулярних виразів дуже схожий зі звичайним поділом рядка функцією Split. Але якщо в Split як роздільник використовувалася рядок, то тут роздільником є ​​регулярний вираз.

Для поділу рядка використовується перевантажений метод Split класу RegEx. Він може приймати такі ж комбінації параметрів, як і методи Matches і IsMatch. Split повертає масив типу String, який містить рядки, отримані з початкового рядка. Масив індексується з нуля.

У даній програмі за допомогою цієї функції робиться поділ за метасимволом у шаблоні, в процедурі rez().



А також у функції userTextget()

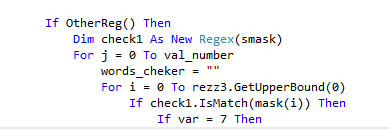


Для зчитування тексту, із файла студента, в масив arrayWord().с

### 1.4.2. Порівняння

Порівняння здійснюється методом IsMatch. Він перевантажений і може приймати такі ж параметри, як і метод Matches. Повертає значення типу Boolean. Метод повертає True, якщо рядок збігається з шаблоном, і False в протилежному випадку.Метод повертає змінну типу String – рядок, в якій були зроблені заміни.

Використовується у процедурі rez(), для порівняння слова із маскою для подальшої роботи із цим словом.

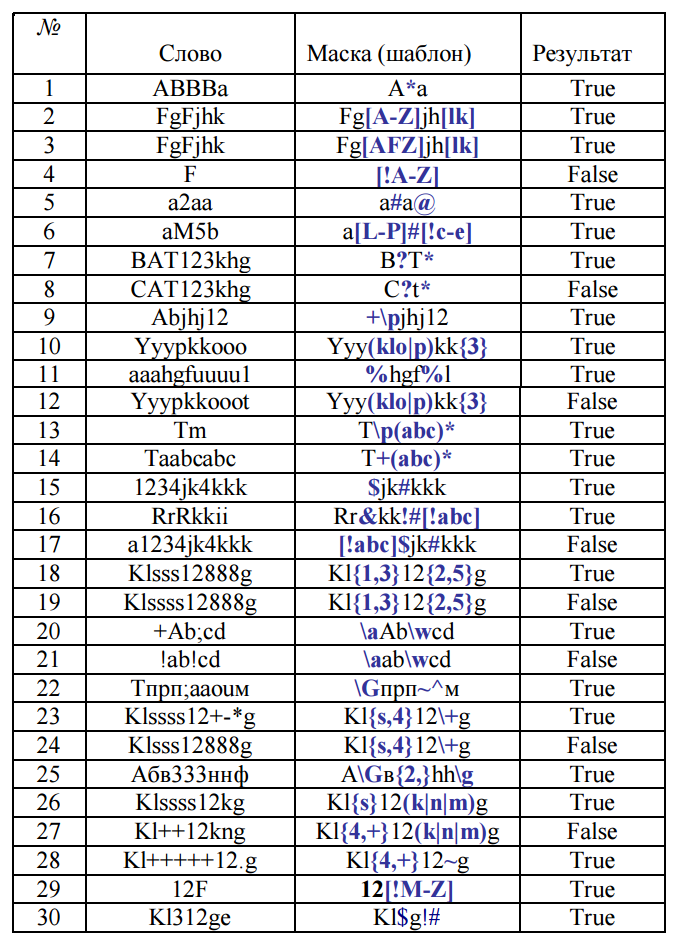


## 1.5 Умова лабораторної роботи

**Завдання:**

Вхідний текст зберігається у файлі і складається із слів, розділених пробілами. Ввести з клавіатури шаблон з метасимволами (метасимвол задається в конкретному варіанті). У вхідному тексті знайти слова (ланцюжки) за заданою маскою. Сформувати вихідний файл, перетворивши вхідний текст визначеним у конкретному варіанті способом. Метасимволи у шаблоні можуть повторюватись, але вони всі для конкретного варіанту одно типові.

Нижче наведено таблицю, в якій показано відповідність деякого введеного слова шаблону (у варіантах завдань є посилання на потрібні приклади). Метасимволи у шаблонах виділені жирним шрифтом. Іноді метасимволи у шаблонах повторюються і бувають різно типовими. Для конкретного варіанту в умовах задач задається метасимвол тільки одного типу.



### 1.5.1 Варіанти завдань:

1. Метасимвол – [ланцюжок символів]. Зміст метасимволу – один з

перерахованих символів повинен бути присутнім в слові в потрібній

позиції (приклади 2, 3).

2. Метасимвол – [діапазон символів]. Зміст метасимволу – один з

діапазону символів повинен бути присутнім в слові в потрібній

позиції (приклади 2, 6).

3. Метасимвол – @. Зміст метасимволу @– одна довільна мала

латинська буква повинна бути присутньою в слові в потрібній позиції

(приклад 5)

4. Метасимвол – [!діапазон символів]. Зміст метасимволу – один з

символів, який не попадає в діапазон, повинен бути присутнім в слові

в потрібній позиції (приклади 4, 6, 29).

5. Метасимвол – !#. Зміст метасимволу !#– один довільний символ,

який не є цифрою повинен бути присутнім в слові в потрібній позиції

(приклад 16, 30).

6. Метасимвол – [!ланцюжок символів]. Зміст метасимволу – один з

символів, який не співпадає з перерахованими символами, повинен

бути присутнім в слові в потрібній позиції (приклади 16, 17).

7. Метасимвол – (ланцюжок символів 1 / ланцюжок символів 2).

Зміст метасимволу – один з ланцюжків символів повинен бути

присутнім в слові в потрібній позиції (приклади 10, 12).

8. Метасимвол – \*. Зміст метасимволу – ланцюжок довільних

символів (приклади 1,7,8).

9. Метасимвол – ?. Зміст метасимволу – один довільний символ (або

ні одного) повинен бути присутнім в слові в потрібній позиції

(приклади 7, 8).

10. Метасимвол – $. Зміст метасимволу – ланцюжок довільних цифр

повинен бути присутнім в слові в потрібній позиції (приклад 15, 30).

11. Метасимвол – +. Зміст метасимволу + – одна довільна голосна

латинська літера повинна бути присутньою в слові в потрібній

позиції (приклади 9, 14).

12. Метасимвол – {цифра}. Зміст метасимволу – ланцюжок

однакових символів заданої довжини (цифра) повинен бути

присутнім в слові в потрібній позиції. (приклади 10, 12).

13. Метасимвол – (ланцюжок символів)\*. Зміст метасимволу – один,

ні одного або декілька ланцюжків символів повинно бути присутнім в

слові в потрібній позиції (приклади 13,14).

14. Метасимвол – %. Зміст метасимволу – ланцюжок однакових

символів повинен бути присутнім в слові в потрібній позиції.

(приклад 11).

15. Метасимволи – {цифра1, цифра2}. Зміст метасимволів –

ланцюжок однакових символів заданої довжини повинен бути

присутнім в слові в потрібній позиції. (приклади 18,19).

16. Метасимвол – \w. Зміст метасимволу – один алфавітно-цифровий

символ або символ “;” .

17. Метасимвол – \a. Зміст метасимволу – знак арифметичної операції

(приклади 20, 21).

18. Метасимвол – \g. Зміст метасимволу – одна довільна глуха буква

повинна бути присутньою в слові в потрібній позиції (приклад 25).

19. Метасимвол – \G. Зміст метасимволу – одна довільна дзвінка

буква повинна бути присутньою в слові в потрібній позиції (приклади

22, 25).

20. Метасимвол – &. Зміст метасимволу – велика латинська літера

повинна бути присутньою в слові в потрібній позиції (приклад 16).

21. Метасимвол – #. Зміст метасимволу – одна довільна цифра

повинна бути присутньою в слові в потрібній позиції (приклади 15,

16).

22. Метасимвол – ~. Зміст метасимволу – один з розділових знаків

повинен бути присутнім в ланцюжку в потрібній позиції (приклад 22,

28). Сформувати новий текст, замінивши в знайдених ланцюжках

кожну латинську букву на крапку.

23. Метасимвол – ^. Зміст метасимволу – ланцюжок довільних

голосних букв (приклад 22).

24. Метасимвол – {буква, цифра}. Зміст метасимволу – ланцюжок

однакових букв довжиною цифра повинен бути присутнім в слові в

потрібній позиції (приклади 23, 24).

25. Метасимвол – \+. Зміст метасимволу – ланцюжок знаків

арифметичних операцій повинен бути присутнім в слові в потрібній

позиції (приклади 23, 24).

26. Метасимвол – \p. Зміст метасимволу \p – одна довільна

приголосна латинська літера повинна бути присутньою в слові в

потрібній позиції (приклад 9, 13).

27. Метасимвол – {буква}. Зміст метасимволу – ланцюжок однакових

(більше одної) букв (буква) повинен бути присутнім в слові в

потрібній позиції (приклад 26).

28. Метасимвол – (символ / символ / символ). Зміст метасимволу –

один з трьох символів повинен бути присутнім в слові в потрібній

позиції (приклади 26, 27).

29. Метасимвол – {цифра,знак}. Зміст метасимволу – ланцюжок

однакових знаків довжиною не меншою за цифра повинен бути

присутнім в слові в потрібній позиції (приклади 27, 28).

30. Метасимвол – {цифра}. Зміст метасимволу – ланцюжок

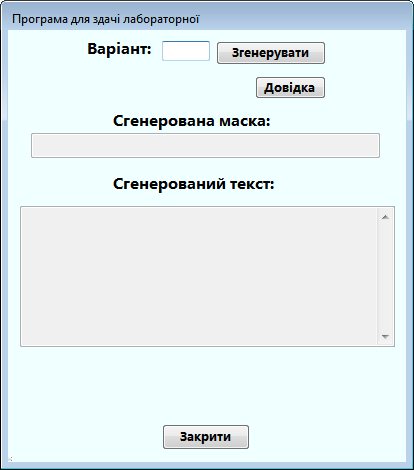
однакових символів заданої довжини повинен бути присутнім в слові

в потрібній позиції (приклад 25).

# РОЗДІЛ 2.ПРАКТИЧНА ЧАСТИНА

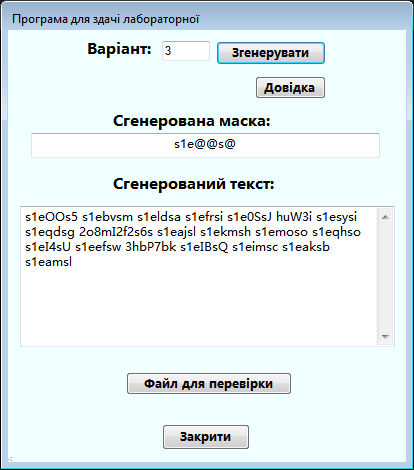
## ***2.1. Інструкція користувача***

Після запуску програми, на екрані з’являється перша форма.

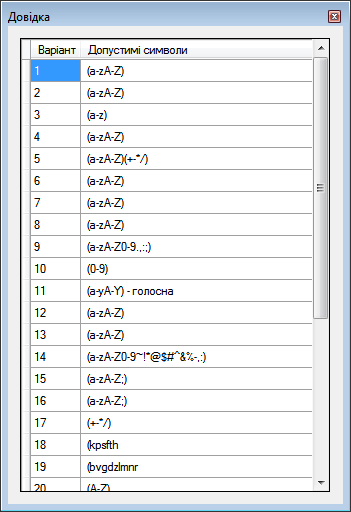


Тут в текстовому полі треба ввести свій варіант із методички,і натиснути «Згенерувати».

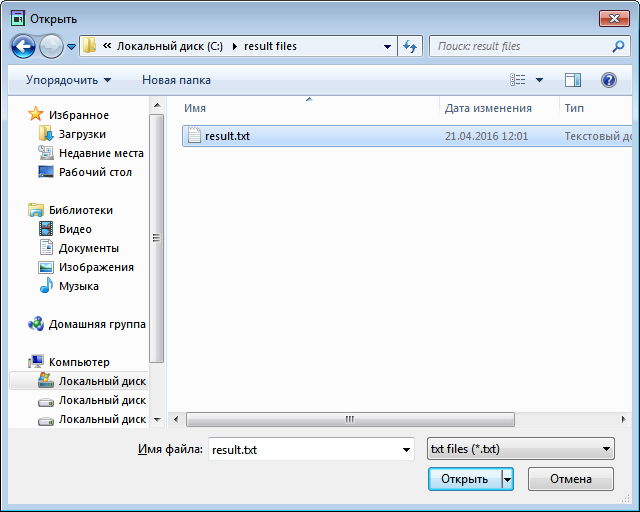
Після натискання відбувається генерація маски та тексту згідно цієї маски.



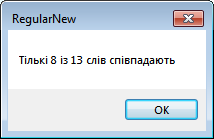
Також є присутньою кнопка «**Довідка**», яка відкриває іншу форму з детальним описом символів які підпадають під даний варіант.

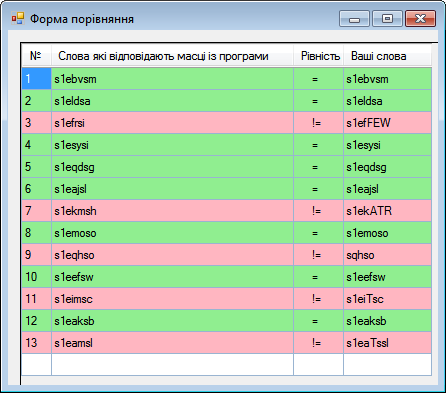


Потім треба натисканням кнопки «Файл для перевірки», вибрати текстовий файл із словами – результатом, виконання програми студента.



Якщо обрано файл для перевірки то відкривається форма з порівнянням слів ,та виводиться і визначається їх рівність.





## 2.2. Опис деяких процедур/функцій

* Процедура chekvariant()

Sub chekvariant()

Перевіряє інформацію введеній в текстове поле для введення варіанту і записує у відповідну змінну.

* Функція GetReg ()

Public Function GetReg(ByVal reg As String, Optional tor As Boolean = True) As String

Генерує маску враховуючи метасимвол закріплений за варіантом .

Повертає згенеровану маску

* Функція userTextf()

Function userTextf() As String

Зчитує із файлу студента слова.

Повертає масив слів.

* Процедура rez()

Public Sub rez()

Проводить заміну метасимволів,і генерує масиви слів які відповідають і не відповідають заданому варіанту.

* Функція GetText()

Public Function GetText()

Створює текст із масивів слів.

# Висновки

У курсовій роботі за допомогою VB .Net було створено програму для перевірки якості лабораторних робіт студентів.

# Перелік посилань

1. Сопронюк Т.М.  Системне програмування. Частина І. Елементи теорії формальних мов: Навчальний посібник у двох частинах. – Чернівці: ЧНУ, 2008. – 84 c
2. Internet: <http://regexr.com/>
3. Internet: <https://msdn.microsoft.com/ru-ru/>
4. Internet: <https://uk.wikipedia.org/>

# Додаток 1

Зміст RegForm.vb (частина)

Imports System.Text.RegularExpressions

Imports System.IO

Public Class RegForm

Public smask As String = ""

Private Sub Button1\_Click(sender As Object, e As EventArgs) Handles btnclose.Click

Close()

End Sub

Function truevar() As Boolean

If OtherReg() Then

Exit Function

ElseIf (var = 9) Or (var = 8) Or (var = 19) Or (var = 10) Or (var = 11) Or (var = 18) Or (var = 23) Or (var = 16) Or (var = 17) Or (var = 26) Then

Return True

Else

Return False

End If

End Function

''' <summary>

''' Заміна для деяких варіантів

''' </summary>

''' <param name="r">Варіант</param>

Sub zaminasmask2(r As Integer)

Dim d As Boolean = True

Select Case d

Case r = 2

smask = "\[[a-z]-[a-z]]"

Case r = 15

smask = "\{[0-9],[0-9]}"

Case r = 30

smask = "\{[0-9]}"

Case r = 12

smask = "\{[0-9]}"

Case r = 29

smask = "\{[0-9],[-/\*+]}"

Case r = 28

smask = "\([@#$%^&\*~!]\/[@#$%^&\*~!]\/[@#$%^&\*~!]\)"

Case r = 27

smask = "\{[a-zA-Z]}"

Case r = 24

smask = "\{[a-zA-Z],[0-9]}"

Case r = 1

smask = "\[[A-Za-z]+\]"

Case r = 6

smask = "\[![A-Za-z]+\]"

Case r = 4

smask = "\[![a-z]-[a-z]\]"

Case r = 7

smask = "\([A-Za-z]+\/[A-Za-z]+\)"

Case r = 13

smask = "\([A-Za-z]+\)\\*"

End Select

End Sub

Public Function truereg(ByVal wword As String, ByVal reg As String)

If truevar() Then

smask = "\" + reg

Else

If OtherReg() Then

Else

If (var = 25) Then

smask = "\\" + reg

Else

smask = reg

End If

End If

End If

Dim rreg As New Regex(smask)

Dim mm As MatchCollection

mm = rreg.Matches(wword)

Return mm.Count

End Function

Public Function GetRandom(ByVal Min As Integer, ByVal Max As Integer) As Integer

Static Generator As Random = New Random()

Return Generator.Next(Min, Max)

End Function

Sub chekvariant()

Dim fa As Boolean = True

Select Case fa

Case txtvar.Text = 20

ee = "&"

var = CInt(txtvar.Text)

Case txtvar.Text = 26

ee = "\p"

var = CInt(txtvar.Text)

Case txtvar.Text = 18

ee = "\g"

var = CInt(txtvar.Text)

Case txtvar.Text = 11

ee = "+"

var = CInt(txtvar.Text)

Case txtvar.Text = 3

ee = "@"

var = CInt(txtvar.Text)

Case txtvar.Text = 5

ee = "!#"

var = CInt(txtvar.Text)

Case txtvar.Text = 21

ee = "#"

var = CInt(txtvar.Text)

Case txtvar.Text = 22

ee = "~"

var = CInt(txtvar.Text)

Case txtvar.Text = 16

ee = "\w"

var = CInt(txtvar.Text)

Case txtvar.Text = 17

ee = "\a"

var = CInt(txtvar.Text)

Case txtvar.Text = 23

ee = "^"

var = CInt(txtvar.Text)

Case txtvar.Text = 10

ee = "$"

var = CInt(txtvar.Text)

Case txtvar.Text = 14

ee = "%"

var = CInt(txtvar.Text)

Case txtvar.Text = 19

ee = "\G"

var = CInt(txtvar.Text)

Case txtvar.Text = 25

ee = "\+"

var = CInt(txtvar.Text)

Case txtvar.Text = 2

ee = "\[a-z]"

var = CInt(txtvar.Text)

Case txtvar.Text = 15

ee = "\{0,8}"

var = CInt(txtvar.Text)

Case txtvar.Text = 30

ee = "{3}"

var = CInt(txtvar.Text)

Case txtvar.Text = 12 '--однокові варіанти 12-30

ee = "{121}"

var = 12

Case txtvar.Text = 30 '--однокові варіанти 12-30

ee = "{30}"

var = 30

Case txtvar.Text = 29

ee = "{2,+}"

var = CInt(txtvar.Text)

Case txtvar.Text = 28

ee = "\([-+/\*]\/[-+/\*]\/[-+/\*]\)"

var = CInt(txtvar.Text)

Case txtvar.Text = 27

ee = "\{[a-zA-Z]}"

var = CInt(txtvar.Text)

Case txtvar.Text = 24

ee = "\{[a-z],[0-9]}"

var = CInt(txtvar.Text)

Case txtvar.Text = 1

ee = "\{[a-z]+"

var = CInt(txtvar.Text)

Case txtvar.Text = 4

ee = "\[!a-z]"

var = CInt(txtvar.Text)

Case txtvar.Text = 6

ee = "\[[A-Za-z]+\]"

var = CInt(txtvar.Text)

Case txtvar.Text = 7

ee = "\([a-z]+\/[a-z]+\)"

var = CInt(txtvar.Text)

Case txtvar.Text = 8

ee = "\*"

var = CInt(txtvar.Text)

Case txtvar.Text = 9

ee = "?"

var = CInt(txtvar.Text)

Case txtvar.Text = 13

ee = "\([a-z]+\)\\*"

var = CInt(txtvar.Text)

End Select

End Sub

''' <summary>

''' Масив малих букв

''' </summary>

Dim alphabet() As String = {"a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z"}

''' <summary>

''' Масив великих букв

''' </summary>

Dim alphabet2() As String = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"}

''' <summary>

''' Масив усіх букв

''' </summary>

Dim alphabet3() As String = {"a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z",

"A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"}

Dim numerik() As Integer = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Dim symbol() As String = {".", ",", ";", ":"}

Dim arf() As String = {"+", "-", "\*", "/"}

Dim symbol2() As String = {"~", "!", "\*", "@", "$", "#", "^", "&", "%", "-", ",", ":"} '

# Додаток 2

Зміст CompareForm.vb

Public Class CompareForm

Public Sub CompareForm\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

Dim userarray() As String = RegForm.userTextget()

Dim mas() As String = RegForm.arrgood(RegForm.txtgettext.Text.Split)

Dim k As Integer = 0

Dim j As Integer = 0

If userarray.Length < RegForm.val\_number + 1 Then

MsgBox("Недостатньо слів у вашому тексті")

End If

If userarray.Length > RegForm.val\_number + 1 Then

MsgBox("Забагато слів у вашому тексті")

End If

For Each word As String In mas

For i = 0 To RegForm.val\_number

If word = RegForm.val\_words(i) Then

j += 1

If k < userarray.Length Then

If (word = userarray(k)) Then

datacompare.Rows.Add(j, word, "=", userarray(k))

k += 1

Exit For

Else

datacompare.Rows.Add(j, word, "!=", userarray(k))

k += 1

Exit For

End If

Else

datacompare.Rows.Add(j, word, "???", " - - -")

Exit For

End If

End If

Next

Next

If datacompare.Columns.GetColumnsWidth(DataGridViewElementStates.Visible) < 700 Then

datacompare.Width = datacompare.Columns.GetColumnsWidth(DataGridViewElementStates.Visible) + 18

Width = datacompare.Width + 18

Else

Width = datacompare.Width + 18

End If

End Sub

End Class