Задача A. ABBC

В этой задаче требовалось требуется написать регулярное выражение и подставить его в уже готовый шаблон. Нас интересуют подстроки, которые начинаются с символа «a», содержат внутри «bb» и заканчиваются на «c», возьмем эти три маленькие строчки и с помощью регулярного выражения подставим любое количество символов между ними.

```
REGEX_MASK = r'a+bb+c+'
```

Далее заменяем все такие подстроки на «QQQ»

```
REGEX_SUB = r'QQQ'
```

Задача B. EuroEnglish

Похожая задача, но у всех замен появляется более сложная логика, поэтому нельзя будет решить задачу только используя шаблон. В целом логика довольно простая: в первый год заменяем «ci» на «si», «ck» на «k» в итоге получится огромные блоки регулярных выражений. Самое главное не ошибится при их составлении и тд.

```
REGEX_MASK = r'(\b[Aa]\b)|(\b[Aa]n\b)|(\b[Tt]he\b)'
REGEX_SUB = r''
REGEX_MASK_1 = r'(ci)'
REGEX_SUB_1 = r'si'
REGEX_MASK_2 = r'(Ci)'
REGEX_SUB_2 = r'Si'
REGEX_MASK_3 = r'(ce)'
REGEX_SUB_3 = r'se'
REGEX_MASK_4 = r'(Ce)'
REGEX_SUB_4 = r'Se'
REGEX_MASK_5 = r'(ck)'
REGEX_SUB_5 = r'k'
REGEX_MASK_6 = r'(Ck)'
REGEX_SUB_6 = r'K'
REGEX_MASK_7 = r'c'
REGEX_SUB_7 = r'k'
REGEX_MASK_8 = r'C'
REGEX_SUB_8 = r'K'
REGEX_MASK_9 = r'ee'
REGEX_SUB_9 = r'i'
REGEX_MASK_10 = r'Ee'
REGEX_SUB_10 = r'I'
REGEX_MASK_11 = r'oo'
REGEX_SUB_11 = r'u'
REGEX_MASK_12 = r'Oo'
REGEX_SUB_12 = r'U'
REGEX_MASK_13 = r'([a-z])\1'
REGEX_SUB_13 = r'\1'
REGEX_MASK_14 = r'([A-Z])\1'
REGEX_SUB_14 = r'\1'
REGEX_MASK_15 = r'([a-z]+|[A-Z]+)e\b'
REGEX_SUB_15 = r'\1'
REGEX_MASK_16 = r'\s+'
REGEX_SUB_16 = r' '
asd = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
asd_1 = 'abcdefghijklmnopqrstuvwxyz'
for i in range(len(asd)):
   if asd[i] + asd_1[i] in new_st:
       new_st = new_st.replace(asd[i] + asd_1[i], asd[i])
```

```
        new_st = re.sub(REGEX_MASK_15, REGEX_SUB_15, new_st)
        st = re.sub(REGEX_MASK_16, REGEX_SUB_16, new_st)
```

Задача C. Поворот на 90

В этой задаче нам нужно поменять местами аргументы в функции. В этой задаче будем опять использовать шаблон. Считаем два целочисленных аргумента

```
REGEX_MASK = r'\\circle{\((\d+),(\d+)\)'
```

И подставим их в другом порядке.

```
REGEX_SUB = r'\\circle{(\2,\1)'
```

Задача D. Индексы

В задаче нужно сделать замену индексации из latex разметки в обычную. Название массива не меняется, что сильно упрощает решение задачи, мы можем использовать уже знакомый нам шаблон и просто написать регулярное выражение.

```
REGEX_MASK = r'\$v_(([a-zA-Z0-9])|{([a-zA-Z0-9]+)})\$'
REGEX_SUB = r'v[\2\3]'
```

Задача E. Химия

В задаче нужно проверять две данные формулы на равенство. Давайте научимся приводить формулу к общему виду и потом сравнивать две строки тривиально, сами строки будем хранить как словари, в которых будут содержаться химические элементы и их количество. Задача сводится к разбору случаев, легче всего использовать функциальную реализацию, которая поможет проще написать код (иметь отдельную функцию для париснга цифр, букв и тд). Сначала научимся парсить числа,

```
def parse_digits(text: str, ptr: int) -> [int, int]:

    ptr_begin = ptr
    cycles = 0

    while (ptr < len(text)) and (text[ptr].isdigit()):
        ptr += 1
        cycles += 1

    if cycles == 0:
        return 1, ptr

    return int(text[ptr_begin:ptr]), ptr
```

Затем напишем метод для париснга букв

```
def parse_chemical_element(text: str, ptr: int) -> [dict, int]:
    all_vocabulary = {}

    ptr_begin = ptr
    ptr += 1
    while (ptr < len(text)) and (text[ptr].islower()):
        ptr += 1

    all_vocabulary[
        text[ptr_begin:ptr]
    ] = 1

    return all_vocabulary, ptr
```

Используя эти две функции мы уже можем парсить химические элементы

```
def parse_sequence(text: str, ptr: int) -> [dict, int]:
    all_vocabulary = {}
```

```
    while ptr < len(text):
        if text[ptr] == ")":
            break

        vocabulary, ptr = parse_elements(text, ptr)
        coef, ptr = parse_digits(text, ptr)

        for elem, value in vocabulary.items():
            if all_vocabulary.get(elem) is None:
                all_vocabulary[elem] = 0
            all_vocabulary[elem] += value * coef
    return all_vocabulary, ptr


def parse_elements(text: str, ptr: int) -> [dict, int]:
    all_vocabulary = {}

    if text[ptr] == "(":
        ptr += 1
        vocabulary, ptr = parse_sequence(text, ptr)
        ptr += 1
    else:
        vocabulary, ptr = parse_chemical_element(text, ptr)

    for elem, value in vocabulary.items():
        if all_vocabulary.get(elem) is None:
            all_vocabulary[elem] = 0
        all_vocabulary[elem] += value

    return all_vocabulary, ptr
```

И в итоге собирать итоговый словарь из строки

```
def parse_formula(formula: str) -> dict:
    all_vocabulary = {}

    for sequence in formula.split("+"):
        coef, ptr = parse_digits(sequence, 0)
        vocabulary, ptr = parse_sequence(sequence, ptr)

        for elem, value in vocabulary.items():
            if all_vocabulary.get(elem) is None:
                all_vocabulary[elem] = 0
            all_vocabulary[elem] += value * coef

    return all_vocabulary
```

Далее нужно считать входные данные и сравнить словари, что довольно просто

```
left_line = input()
left = parse_formula(left_line)
N = int(input())

for i in range(N):
    result = "=="
    right_line = input()
```

```
right = parse_formula(right_line)

if len(left) > len(right):
    checker = left
    other = right
else:
    checker = right
    other = left

for atom, value in checker.items():
    if other.get(atom) != value:
        result = "!="
        break

print(left_line + result + right_line)
```