**Rule: X op Y → XYop**

Vd: X + Y → XY+

## **Ví dụ**

Input:

3

(a+(b\*c))

((a+b)\*(z+x))

((a+t)\*((b+(a+c))^(c+d)))

(a+(b\*c)) → **abc\*+**

(b\*c) → bc\*

((a+b)\*(z+x)) → **ab+zx+\***

((a+t)**\***((b+(a+c))^(c+d))) → **at+bac++cd+^\***

(a+t) → at+

((b+(a+c))**^**(c+d)) → bac++cd+^

(b+(a+c)) → bac++

## **Nhận xét:**

* Dấu ‘(‘ báo hiệu cho một biểu thức con mới. Dấu ‘)’ đánh dấu sự kết thúc của một biểu thức con liền kề với nó. → Giống với hoạt động của **stack**.
* Từ a + b chuyển sang ab+ → Xuất ra các toán hạng trước, rồi đến toán tử gắn với nó.

⇒ Hướng giải: duyệt qua chuỗi ký tự:

* Nếu gặp ‘(‘ thì mình bỏ qua.
* Nếu gặp toán tử (+, -, \*, /, ^) thì mình bỏ vào stack.
* Nếu gặp chữ (toán hạng) thì mình sẽ xuất ra.
* Nếu gặp ‘)’, mình sẽ xuất toán tử ở đầu stack, pop nó ra khỏi stack.

**Giải thuật:**

B1: Đọc vào xâu

B2: Khởi tạo 1 empty stack

B3: Duyệt xâu từ trái sang phải:

* Nếu gặp ‘(‘ thì mình bỏ qua.
* Nếu gặp toán tử (+, -, \*, /, ^) thì mình bỏ vào stack.
* Nếu gặp chữ (toán hạng) thì mình sẽ xuất ra.
* Nếu gặp ‘)’, mình sẽ xuất toán tử ở đầu stack, pop nó ra khỏi stack.

**Mã giả**

read(test)

for i in range(test): O(T)

read(expression)

s = stack() // Khởi tạo 1 stack rỗng

for char in expression: O(len(expression)

if char.isalpha() // char là 1 toán hạng (a, b, c…)

Print(char)

else if char == ‘)’:

print(s.top())

s.pop()

else if char != ‘(‘:

s.push(char)

print()

Độ phức tạp: O(T\*len(expression))