Build an algorithm to parse and process the dynamic **expression** based on the **dictionary** provided. The **expression** basically consists of Operators, Variables, and Parentheses. Parentheses have higher precedence (i.e the subexpression in the parentheses should be executed first)

A or (B and C)

- 🟥 Variables
- ⬛ Operators
- 🟦 Parentheses

A **dictionary** is a key-value pair with variables(in the expression) as the key and with a respective value. Dictionary might be provided with additional variables (not present in the expression) or even provided with missing variables.

Basically, the algorithm should support two operators - AND (concatenates the two variables) & OR (provides first variable value if it is available. Otherwise, go for the second variable)

The algorithm should parse the given expression and apply the respective dictionary values on the expression and provides the respective computed results.

SAMPLE INPUT AND OUTPUT

	INPUTS		**OUTPUT**
Sno	**Expression**	**Dictionary**	**Expected Results**
1	A and B	{'A':'Hello', 'B': 'World', 'C': 'Buddy'}	HelloWorld
2	A and C	{'A':'Hello', 'B': 'World', 'C': 'Buddy'}	HelloBuddy
3	D or B	{'A':'Hello', 'B': 'World', 'C': 'Buddy'}	World
4	A or B	{'A':'Hello', 'B': 'World', 'C': 'Buddy'}	Hello
5	A and B and C	{'A':'Hello', 'B': 'World', 'C': 'Buddy'}	HelloWorldBuddy
6	A and (B or C)	{'A':'Hello', 'B': 'World', 'C': 'Buddy'}	HelloWorld
7	A and (C or D)	{'A':'Hello', 'B': 'World', 'C': 'Buddy'}	HelloBuddy
8	A and (B or C) and D	{'A':'Hello', 'C': 'Buddy', ‘D’: ‘Welcome’}	HelloBuddyWelcome

NOTE: *The algorithm must satisfy the above sample use cases and is designed to be completely dynamic - processes new expressions too without altering the code.*