逆波兰式

```C++
"x1 x1 x1 * x2 + *" -> "(x1 * ((x1 * x1) + x2))"
```

1. python 实现

```Python
def rpn_to_infix(rpn):
    stack = []
    for token in rpn.split():
        if token.isalnum():  # 检查是否为操作数
            stack.append(token)
        else:  # 操作符
            if len(stack) < 2:
                raise ValueError("Invalid RPN expression")
            b = stack.pop()
            a = stack.pop()
            stack.append(f"({a} {token} {b})")
    return stack[0]
```

2. c++实现

C++

```cpp
#include <iostream>
#include <stack>
#include <sstream>
#include <vector>
#include <string>

std::string rpnToInfix(const std::string& rpn) {
    std::stack<std::string> stack;
    std::istringstream iss(rpn);
    std::string token;

    while (iss >> token) {
        if (token == "+" || token == "-" || token == "*" || token == "/") {
            if (stack.size() < 2) {
                throw std::runtime_error("Invalid RPN expression");
            }
            std::string b = stack.top(); stack.pop();
            std::string a = stack.top(); stack.pop();
            std::string infix = "(" + a + " " + token + " " + b + ")";
            stack.push(infix);
        } else {
            stack.push(token);
        }
    }

    if (stack.size() != 1) {
        throw std::runtime_error("Invalid RPN expression");
    }
    return stack.top();
}

int main() {
    std::string rpn = "x1 x1 x1 * x2 + *";
    try {
        std::string infix = rpnToInfix(rpn);
        std::cout << "Infix expression: " << infix << std::endl;
    } catch (const std::runtime_error& e) {
        std::cerr << "Error: " << e.what() << std::endl;
    }
    return 0;
}
```