1106. Parsing A Boolean Expression

A **boolean expression** is an expression that evaluates to either `true` or `false`. It can be in one of the following shapes:

- `'t'` that evaluates to `true`.
- `'f'` that evaluates to `false`.
- `'!(subExpr)'` that evaluates to **the logical NOT** of the inner expression `subExpr`.
- `'&(subExpr1, subExpr2, ..., subExprn)'` that evaluates to **the logical AND** of the inner expressions `subExpr1, subExpr2, ..., subExprn` where `n >= 1`.
- `'|(subExpr1, subExpr2, ..., subExprn)'` that evaluates to **the logical OR** of the inner expressions `subExpr1, subExpr2, ..., subExprn` where `n >= 1`.

Given a string `expression` that represents a **boolean expression**, return *the evaluation of that expression*.

It is **guaranteed** that the given expression is valid and follows the given rules.

Example 1:

```
Input: expression = "&(|(f))"
Output: false
Explanation:
First, evaluate |(f) --> f. The expression is now "&(f)".
Then, evaluate &(f) --> f. The expression is now "f".
Finally, return false.
```

Example 2:

```
Input: expression = "|(f,f,f,t)"
Output: true
Explanation: The evaluation of (false OR false OR false OR true) is true.
```

Example 3:

```
Input: expression = "!(&(f,t))"
Output: true
Explanation:
First, evaluate &(f,t) --> (false AND true) --> false --> f. The expression is now
"!(f)".
Then, evaluate !(f) --> NOT false --> true. We return true.
```

Constraints:

- `1 <= expression.length <= 2 * 104`
- expression[i] is one following characters: `'('`, `')'`, `'&'`, `'|'`, `'!'`, `'t'`, `'f'`, and `','`.

1106. Parsing A Boolean Expression

```
/*
   Recursion
   Time complexity: O(n)
   Space complexity: O(n)
*/

class Solution {
   public:
      bool parseBoolExpr(std::string expression){
         // int& i: all intances of the function should share the same i pointer
         auto solve=[&](std::string& exp,int& i,auto& self)->bool{
            char c=exp[i++]; // Get current character

            if(c=='t') return true;
            if(c=='f') return false;

            // Handle "!(...)"
            if(c=='!'){
               i++; // Skip '('
               bool ans=!self(exp,i,self);
               i++; // Skip ')'
               return ans;
            }

            // Handle "&(...)" and "|(...)"
            i++; // Skip '('
            std::vector<bool> operands;
            while(exp[i]!=')'){
               if(exp[i]!=',') operands.push_back(self(exp,i,self));
               else i++; // Skip ','
            }
            i++; // Skip ')'
```

```
            // Evaluate "|(...)"
            if(c=='|'){
                for(bool v: operands){
                    if (v) return true;
                }
                return false;
            }

            // Evaluate "&(...)"
            if(c=='&'){
                for(bool v: operands){
                    if (!v) return false;
                }
                return true;
            }

            return true; // Never reached
        };

        int i=0;
        return solve(expression,i,solve);
    }
};
```