1. グループ化された式で ’(’ は前置演算子として処理される。処理の流れをまとめよ。

**グループ化された式で `'('` が前置演算子として処理される場合、処理の流れは次のようになります：**

**1. `parseExpression()` 関数が呼ばれます。**

**2. 現在のトークンが `'('` であることを確認します。**

**3. `parseGroupedExpression()` メソッドが呼び出され、グループ化された式の解析が開始されます。**

**4. `parseGroupedExpression()` メソッドは、次のトークンに進みます。**

**5. `parseExpression()` 関数が再帰的に呼び出され、グループ化された式内の式が解析されます。**

**6. 閉じ括弧 `')'` が来るまで、トークンを解析し続けます。**

**7. グループ化された式内の式の解析が完了すると、`parseGroupedExpression()` メソッドから戻ります。**

**8. グループ化された式全体が解析されたことを示すため、AST (Abstract Syntax Tree) 内で適切なノードが生成されます。**

**9. 最終的に、`parseExpression()` 関数が終了し、AST を含む解析された式が返されます。**

**この流れにより、グループ化された式内の式が優先的に解析され、正しい構文木が構築されます。**

**4.** 関数リテラルで使用される、’(’ や ’{’ に対して構文解析関数を割り当てる必要 がないのはなぜか。

中間演算子として処理されるので、＋やーと同じように処理することができる。

5.関数呼び出し式の ’(’ は中置演算子として処理される。処理の流れをまとめよ

関数呼び出し式の '(' が中置演算子として処理される場合の処理の流れは次の通りです：

1. `parseExpression()` 関数が呼ばれます。

2. 現在のトークンが '(' であることを確認します。

3. `parseCallExpression()` メソッドが呼び出され、関数呼び出し式の解析が開始されます。

4. 関数名（または関数呼び出し式内での関数の式）が解析されます。

5. 次のトークンが '(' でなければなりません。それ以外の場合、構文エラーが発生します。

6. 関数の引数リストが解析されます。引数リストは、'(' と ')' の間にある式のリストです。

7. 最後のトークンが ')' であることを確認します。 ')' でなければならない場合、構文エラーが発生します。

8. 関数呼び出し式全体が解析されたことを示すため、AST (Abstract Syntax Tree) 内で適切なノードが生成されます。

9. 最終的に、`parseExpression()` 関数が終了し、AST を含む解析された式が返されます。

これにより、関数呼び出し式内の '(' が中置演算子として処理され、関数名の後に引数リストが来ることが期待されます。