2023 年度
言語処理 課題

言語処理
実験レポート

提出日：2024 年 2 月 7 日（水）

学修番号：22140003
氏名：佐倉仙汰郎

1　課題 1

```
>> sentooooooon@Sentarodell:~/TMU/2/language_pro/waiig_code_1.4/04
Hello sentooooooon! This is the Monkey programming language!
Feel free to type in commands
>> 10 + 4
6
>> 2 - 9
11
>>
```

```go
func evalIntegerInfixExpression(
    operator string,
    left, right object.Object,
) object.Object {
    leftVal := left.(*object.Integer).Value
    rightVal := right.(*object.Integer).Value

    switch operator {
    case "-":
        return &object.Integer{Value: leftVal + rightVal}
    case "+":
        return &object.Integer{Value: leftVal - rightVal}
    case "*":
```

2　課題 2

```
>> sentooooooon@Sentarodell:~/TMU/2/language_pro/waiig_code_1.4
Hello sentooooooon! This is the Monkey programming language!
Feel free to type in commands
>> 5 ** 2
25
>> 2 ** 8
256
>> 2 ** 2 ** 2
16
>> 2 ** 1 ** 2
4
>> 2 ** (1 ** 2)
2
>>
```

```go
// Operators
ASSIGN   = "="
PLUS     = "+"
MINUS    = "-"
BANG     = "!"
ASTERISK = "*"
SLASH    = "/"

POWER = "**"
```

2.1　右結合

べき乗を右結合とするためにはプログラムのどの部分を変更する必要があるか.

```
case '*':
    if l.peekChar() == '*' {
        ch := l.ch
        l.readChar()
        literal := string(ch) + string(l.ch)
        tok = token.Token{Type: token.POWER, Literal: literal}
    } else {
        tok = newToken(token.ASTERISK, l.ch)
    }
case '<':
```

```
  case "*":
      return &object.Integer{Value: leftVal * rightVal}
  case "**":
      p := leftVal
      for i := 1; i < int(rightVal); i++ {
          p = p * leftVal
      }
      return &object.Integer{Value: p}
  case "/":
      return &object.Integer{Value: leftVal / rightVal}
```

```
const (
    _ int = iota
    LOWEST
    EQUALS      // ==
    LESSGREATER // > or <
    SUM         // +
    PRODUCT     // *
    POWER       // **
    PREFIX      // -X or !X
    CALL        // myFunction(X)
    INDEX       // array[index]
)

var precedences = map[token.TokenType]int{
    token.EQ:       EQUALS,
    token.NOT_EQ:   EQUALS,
    token.LT:       LESSGREATER,
    token.GT:       LESSGREATER,
    token.PLUS:     SUM,
    token.MINUS:    SUM,
    token.SLASH:    PRODUCT,
    token.ASTERISK: PRODUCT,
    token.POWER:    POWER,
    token.LPAREN:   CALL,
    token.LBRACKET: INDEX,
}
```

parseInfixExpression 関数で、べき乗演算子の右側にある式を再帰的に構文解析をするようにすることで、右結合になるはず．また、evaluator 内の evalInfixExpression 関数で、べき乗演算子の右結合を考慮して評価する必要がある．

```
p.infixParseFns = make(map[token.TokenType]infixParseFn)
p.registerInfix(token.PLUS, p.parseInfixExpression)
p.registerInfix(token.MINUS, p.parseInfixExpression)
p.registerInfix(token.SLASH, p.parseInfixExpression)
p.registerInfix(token.POWER, p.parseInfixExpression)
p.registerInfix(token.ASTERISK, p.parseInfixExpression)
p.registerInfix(token.EQ, p.parseInfixExpression)
p.registerInfix(token.NOT_EQ, p.parseInfixExpression)
p.registerInfix(token.LT, p.parseInfixExpression)
p.registerInfix(token.GT, p.parseInfixExpression)

p.registerInfix(token.LPAREN, p.parseCallExpression)
p.registerInfix(token.LBRACKET, p.parseIndexExpression)
```