# Programming Language Design Final Report

## Rational Arithmetic Evaluation

107502570 資工系 4B 陳錕詮

### Run Code

```
runhaskell Main.hs
```

### Main func

接受輸入，並且丟去運算，然後印出結果。

```
main :: IO ()
main = do
  putStrLn "Input Arithmetic String : "
  arithmetic_str <- getLine
  print (rn2string (calculate (token2tree (string2token (arithmetic_str) ) ) ) )
```

## 運算流程

1. 將字串 token 化
2. parse token ，將其轉成 ast-tree
3. 用tree 做運算，算出結果
4. 印出結果

# Overview

## 1. string2token

將字串轉成不同種類得 token

```
string2token :: String -> [Token]
string2token [] = []
string2token (c : cs)
    | c == '(' = T_Left_Paren : string2token cs
    | c == ')' = T_Right_Paren : string2token cs
    | elem c "+-*/" = T_Operator (char2operator c) : string2token cs
    | c == '%'  = T_Percent : string2token cs
    | isDigit c = char2integer c cs
    | isSpace c = string2token cs
    | otherwise = error "Wrong Input Format"
```

## 2. token2tree & token2exp & token2term & token2factor

將 token 根據不同的 token 種類 parse 並且建成 tree 。

先以token2tree為根去 parse ，在遇到 加法＆減法  運算時 進入 token2exp ，在遇到  乘法＆除法  運算時 進入 token2term ，最後token2factor 會去 parse rational number 是否符合格式。

```
token2tree :: [Token] -> Tree
token2tree ts = let (tree, ts1) = token2exp ts
    in
        if null ts1
        then tree
        else error "Wrong Input Format"

token2exp :: [Token] -> (Tree, [Token])
token2exp ts =
    let (left_tree, ts1) = token2term ts
    in
        case get_head ts1 of
            (T_Operator op) | elem op [Plus, Minus] ->
                let (right_tree, ts2) = token2exp (get_tail ts1)
                in (P_M_Node op left_tree right_tree, ts2)
            _ -> (left_tree, ts1)

token2term :: [Token] -> (Tree, [Token])
token2term ts =
    let (left_tree, ts1) = token2factor ts
    in
        case get_head ts1 of
```

```
            (T_Operator op) | elem op [Mul, Div] ->
                let (right_tree, ts2) = token2term (get_tail ts1)
                in (M_D_Node op left_tree right_tree, ts2)
            _ -> (left_tree, ts1)

token2factor :: [Token] -> (Tree, [Token])
token2factor ts =
    case get_head ts of
        T_Left_Paren -> -- (
            let ts1 = get_tail ts
            in case get_head ts1 of
                (T_Integer x) -> -- Int
                    let ts2 = get_tail ts1
                    in case get_head ts2 of
                        T_Percent -> -- %
                            let ts3 = get_tail ts2
                            in case get_head ts3 of
                                (T_Integer y) ->
                                    let ts4 = get_tail ts3
                                    in case get_head ts4 of
                                        T_Right_Paren -> (R_N_Node x y, get_tail ts4) -- )
                                        _ -> error "Wrong Input Format"
                                _ -> error "Wrong Input Format"
                        _ -> error "Wrong Input Format"
                _ ->
                    let (exp_tree, ts2) = token2exp (get_tail ts1)
                    in
                       if get_head ts2 == T_Right_Paren
                       then (exp_tree, get_tail ts2)
                       else error "Wrong Input Format"
        _ -> error "Wrong Input Format"
```

## 3. calculate

以 tree 作為輸入，計算出結果。以遞迴的方式不斷的遞迴節點，根據不同節點得運算符號，進行運算，並且在遇到分母為 0 的時候，輸出錯誤 “divide by zero” 警告，將最終結果以 rational number 形式的節點回傳。

```
calculate :: Tree -> Tree
calculate (R_N_Node x y) = R_N_Node x y
calculate (P_M_Node op left_tree right_tree) =
    let (R_N_Node lx ly)  = calculate left_tree
        (R_N_Node rx ry) = calculate right_tree
    in
        if ly == 0 || ry == 0
        then error "divide by zero"
        else
```

```
            case op of
                Plus ->
                    let a = lx * ry + rx * ly
                        b = ly * ry
                        d = gcd a b
                    in R_N_Node (div a d) (div b d)
                Minus ->
                    let a = lx * ry - rx * ly
                        b = ly * ry
                        d = gcd a b
                    in R_N_Node (div a d) (div b d)
                _ -> error "Wrong Input Format"

calculate (M_D_Node op left_tree right_tree) =
    let (R_N_Node lx ly)  = calculate left_tree
        (R_N_Node rx ry) = calculate right_tree
    in
        if ly == 0 || ry == 0
        then error "divide by zero"
        else
            case op of
                Mul ->
                    let
                        a = (lx * rx)
                        b = (ly * ry)
                        d = gcd a b
                    in R_N_Node (div a d) (div b d)
                Div ->
                    let
                        a = (lx * ry)
                        b = (rx * ly)
                        d = gcd a b
                    in
                        if b == 0
                        then error "divide by zero"
                        else R_N_Node (div a d) (div b d)
                _ -> error "Wrong Input Format"
```

## 4. rn2string

將運算完成的 rational number 節點轉成字串，如果有負號會在分子加個括號，最後將結果以 stdout 輸出。

```
rn2string :: Tree -> String
rn2string node =
    case node of
        R_N_Node a b ->
            if a >= 0
```

```
            then show a ++ " % " ++ show b
            else "(" ++ show a ++ ") % " ++ show b
        _ -> error "Wrong Input Format"
```

# Data Structures

## 1. Token

token 種類分為運算符（Operator）, 整數 (T_Integer) , 左括號（T_Left_Paren）, 右括號（T_Right_Paren）, 有理數百分比符號 (T_Percent) , 結束符號 (T_EOF)

運算符（Operator）又分為 Plus | Minus | Mul | Div 分別代表：加減乘除

```
data Token = T_Operator Operator
    | T_Integer Int
    | T_Left_Paren
    | T_Right_Paren
    | T_Percent     -- %
    | T_EOF
    deriving (Show, Eq)

data Operator = Plus | Minus | Mul | Div
    deriving (Show, Eq)
```

## 2. Tree

tree 的節點分為 P_M_Node 代表 加減 (plus minus )的節點 , M_D_Node 代表 乘除 (mul div )的節點，這兩個樹節點都帶有運算符和左右子樹。最後的節點是有理數節點 R_N_Node，這個節點帶有兩個整數，分別為分子與分母。

```
data Tree = P_M_Node Operator Tree Tree
    | M_D_Node Operator Tree Tree
    | R_N_Node Int Int
    deriving Show
```