Kang Hong Bo

32684673

FIT2102 Assignment 2

Part 1

Exercise 1 BNF Grammar :

<letter> ::= "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"

<lambda> ::= "λ" <letter> "."

<repeatL> ::= “(” <lambda> <check>

<termexc> ::= <letter>+ <check> “)”

<termbase> ::= <letter>+

<termnew> ::= “(” <letter>+ “)”

<check> ::= <repeatL> | <termexc> |<termbase>|<termnew>

letter: is used to indicate which the variable name

lambda: make a builder of `lam`

repeatL: check if there is and repeat expression

termexc: create a term by checking the remaining variable (the variable behind .)

termbase: a base statement which can just read only variable and stop for recursion

termnew: add a new term behind the previous terms.

In this exercise 1 I used do block for passing and checking for the input string.

For the repeating variable I used foldr to traverse all the variable in a list and assign term function to the variable which is converted before using list1.

Besides I also used lambda to shorten the do block and the check is a recursive loop which will continue to traverse the string and have a parser builder as return

Update of BNF for exercise 2

<checks> :: = stermexc ||| termnew |||termbase |||newLamb

< stermexc > :: = [<letter>+] <checks>

< newLamb > ::= “(“ noBrac “)”

<lambdaS> ::= noBrac | gotBrac

<noBrac> ::= “λ” [<letter>+] “.” <checks>

<gotBrac> ::= “(” <noBrac> “.” <checks>

In Part 2 I alter the previous check to checks which is used for short lambda since is some different structure. Thers is a lambdaS which will indicate which kind of lambda expression will be input first no bracket or got bracket. The noBrac has the same usage as lambda but is different cause of the repeating variable at the expression. And the repeating variable used the same function as above the foldr.

Part 2

I made some Bnf For the usage in the question 1

Part 3

Exercise 2

<digit> :: = ‘0’|’1’|’2’|’3’|’4’|’5’|’6’|’7’|’8’|’9’

<numberFact>::= [<digit>+]

<fact> ::= <numberFact> “!”

For this question I made a factorial function. Which will take a input of a digit that is in range 0 to 9 and a ‘!’ and will return a result of the factorial.

In this function I used guardian to set the base case of the factorial and pattern match for the recursion of the value.