FIT2102 T4 Group 1

Week 5 – Lambda Calculus

Surname	Firstname	Contribution %	Any issues?
Lee	Jun Kang	25%	
Lee	Kai Yi	25%	
Khor	Kai Wen	25%	
Low	Gabriel	25%	

Exercise 1: I-Combinator

Combinator is the identity I-Combinator with the variable x bound to the parameter. It is used for extracting data from encapsulated types.

x => x

Lambda calculus expression: λx.x

Exercise 2: Alpha equivalence

1. Which lambda expression is alpha equivalent to λx.x:
    a. λx.y
    b. λa.a
    c. λz.x
2. Which lambda expression is alpha equivalent to λxy.yx:
    a. λaz.az
    b. λa(λb.ba)
    c. λaz.ba
3. Which lambda expression is alpha equivalent to λxy.xz:
    a. λxz.xz
    b. λmn.mz
    c. λz.(λx.xz)

Exercise 3: Beta normal form or divergence?

```
1. (λx.x)y
   = (λx [x:=y]. x)
   = y                          => Beta normal form

2. λx.xx
   = x                          => Beta normal form

3. (λz.zz)(λy.yy)
   = (λz [z:=(λy.yy)]. zz)
   = (λy.yy)(λy.yy)             => Divergence (alpha equivalence with start line)

4. (λx.xx)y
   = (λx [x:=y]. xx)
   = yy                         => Beta normal form
```

Exercise 4: Beta reduction

```
1. (λy.zy)a
   = (λy [y:=a]. zy)
   = za

2. (λx.x)(λx.x)
   = (λx [x:=(λx.x)]. x)
   = (λx.x)
```

```
3. (λx.xy)(λx.xx)
   = (λx [x:=(λx.xx)]. xy)
   = (λx.xx)y
   = (λx [x:=y]. xx)
   = yy

4. (λz.z)(λa.aa)(λz.zb)
   = (λz [z:=(λa.aa)]. z)(λz.zb)
   = (λa.aa)(λz.zb)
   = (λa [a:=(λz.zb)]. aa)
   = (λz.zb)(λz.zb)
   = (λz [z:=(λz.zb)]. zb)
   = (λz.zb)b
   = (λz [z:=b]. zb)
   = bb
```

Exercise 5: Eta Conversion

```
1. λx.zx
   = z             => Eta Conversion

2. λx.xz
   = z             => Eta Conversion

3. (λx.bx)(λy.ay)
   = (λx.bx)a      => Eta Conversion
   = ba            => Eta Conversion
```

Exercise 6: Which of the following are combinators?

// A combinator is a lambda expression (function) with no free variables.

```
1. λx.xxx          => combinator
2. λxy.zx          => not a combinator, z is a free variable
3. λxyz.xy(zx)     => combinator
4. λxyz.xy(zxy)    => combinator
```

Exercise 7: Y-Combinator application

The definition of the Y-Combinator is: Y = λf.(λx.f(xx))(λx.f(xx)) where Y is the Y combinator.

```
Y(g) = λf.(λx.f(xx))(λx.f(xx))g
     = (λf [f:=g]. (λx.f(xx))(λx.f(xx)))
     = (λx.g(xx))(λx.g(xx))
     = (λx [x:=(λx.g(xx))]. g(xx))
     = g((λx.g(xx))(λx.g(xx)))
     subs Y(g) = (λx.g(xx))(λx.g(xx)) from line 3,
     = g(Y(g))
```

Exercise 8: Church Encoding

```
-   TRUE = \xy.x
-   FALSE = \xy.y
-   IF = \btf. b t f
-   AND = \xy. IF x y FALSE
-   OR = \xy. IF x TRUE y
-   NOT = \x. IF x FALSE TRUE
```

```
1.  NOT FALSE
    = (\x. IF x FALSE TRUE) FALSE                        - expand NOT
    = (\x [x:=FALSE]. IF x FALSE TRUE)                   - beta reduction
    = IF FALSE FALSE TRUE
    = (\btf. b t f) FALSE FALSE TRUE                     - expand IF
    = (\btf [b:=FALSE, t:=FALSE, f:=TRUE]. b t f)        - beta reduction
    = FALSE FALSE TRUE
    = (\xy.y) FALSE TRUE                                 - expand FALSE
    = (\xy [x:=FALSE, y:=TRUE]. y)                       - beta reduction
    = TRUE

2.  IF (OR TRUE FALSE)
    = (\btf. b t f) (OR TRUE FALSE)                      - expand IF
    = (\btf [b:=OR, t:=TRUE, f:=FALSE]. b t f)           - beta reduction
    = OR TRUE FALSE
    = (\xy. IF x TRUE y) TRUE FALSE                      - expand OR
    = (\xy [x:=TRUE, y:=FALSE]. IF x TRUE y)             - beta reduction
    = IF TRUE TRUE FALSE
    = (\btf. b t f) TRUE TRUE FALSE                      - expand IF
    = (\btf [b:=TRUE, t:=TRUE, f:=FALSE]. b t f)         - beta reduction
    = TRUE TRUE FALSE
    = (\xy.x) TRUE FALSE                                 - expand TRUE
    = (\xy [x:=TRUE, y:=FALSE]. x)                       - beta reduction
    = TRUE

3.  IF (AND TRUE TRUE)
    = (\btf. b t f) (AND TRUE TRUE)                      - expand IF
    = (\btf [b:=AND, t:=TRUE, f:=TRUE]. b t f)           - beta reduction
    = AND TRUE TRUE
    = (\xy. IF x y FALSE) TRUE TRUE                      - expand AND
    = (\xy [x:=TRUE, y:=TRUE]. IF x y FALSE)             - beta reduction
    = IF TRUE TRUE FALSE
    = (\btf. b t f) TRUE TRUE FALSE                      - expand IF
    = (\btf [b:=TRUE, t:=TRUE, f:=FALSE]. b t f)         - beta reduction
    = TRUE TRUE FALSE
    = (\xy.x) TRUE FALSE                                 - expand TRUE
    = (\xy [x:=TRUE, y:=FALSE]. x)                       - beta reduction
    = TRUE
```

Challenge exercise: Y-Combinator in JavaScript

Bigger hint: there's another famous combinator called `Z` which is basically `Y` adapted to work with strict evaluation:

```
Z=λf.(λx.f(λv.xxv))(λx.f(λv.xxv))
```

```
// The Y-Combinator, gives anonymous functions recursive powers.
const Y = f => {
      return (x => f(v => x(x)(v)))(x => f(v => x(x)(v)))
}

// A simple function that recursively calculates 'n!'.
const fac = Y(f => n => n>1 ? n * f(n-1) : 1);

console.log(fac(3)); // Prints 6
console.log(fac(5)); // Prints 120
```