Exercise 1. Use Church Booleans to complete the functions:

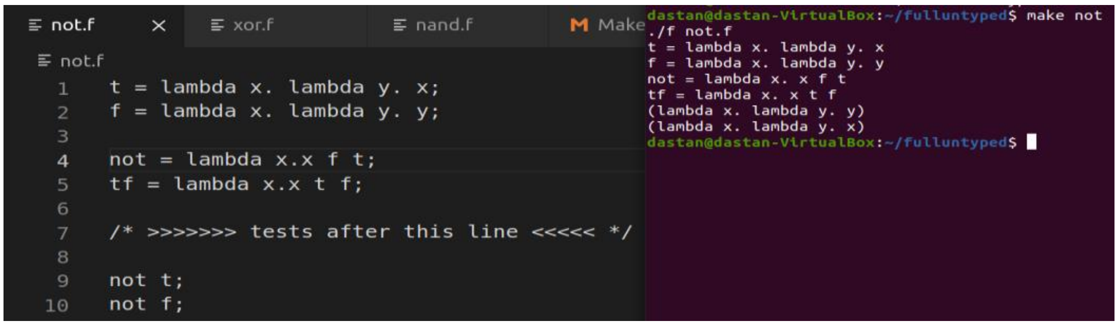
<u>NOT = λx. x FT</u>

It is clear that:

- **"If TRUE then TRUE else FALSE = TRUE"**
- **"if FALSE the TRUE else FALSE = FALSE"**

But our NOT function must do it vice versa. Therefor we just change the positions of TRUE and FALSE. It will give us the right solution for our NOT function:

- **"If TRUE then FALSE else TRUE = FALSE"**
- **"if FALSE the FALSE else TRUE =TRUE"**



```
not.f
t = lambda x. lambda y. x;
f = lambda x. lambda y. y;

not = lambda x.x f t;
tf = lambda x.x t f;

/* >>>>>>> tests after this line <<<<< */

not t;
not f;
```

```
dastan@dastan-VirtualBox:~/fulluntyped$ make not
./f not.f
t = lambda x. lambda y. x
f = lambda x. lambda y. y
not = lambda x. x f t
tf = lambda x. x t f
(lambda x. lambda y. y)
(lambda x. lambda y. x)
dastan@dastan-VirtualBox:~/fulluntyped$
```

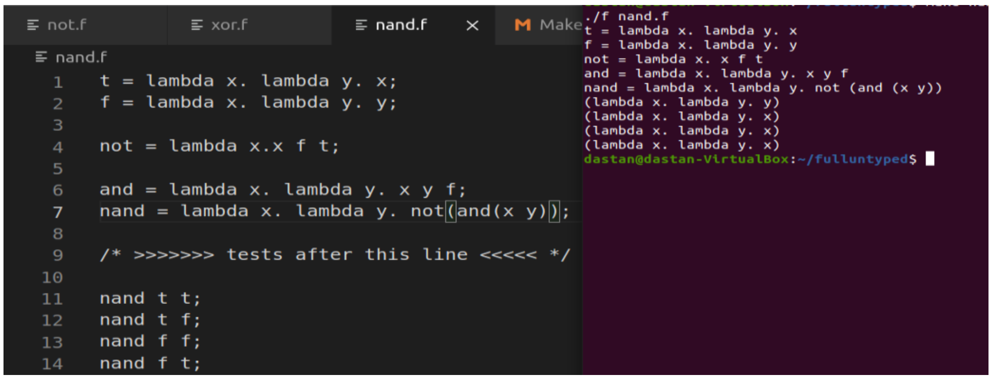
<u>NAND = λx. λy. x (NOT y) T</u>

For this function we must apply two variables, **T** OR **F**(Like (λx. λy. x (NOT y)T)TF).

We know that NAND operation must give us next table which is vice versa for AND:

A	B	A **NAND** B
0	0	1
0	1	1
1	0	1
1	1	0

We know that AND functions Church Boolean representation is λx. λy. xy F. First of all, we must mention that in NAND's output is mainly TRUE. Therefor We will change F to T. But in this way, if we apply FF for xy, we will take T. If apply FT we will also take T. But if we apply TF, then we will take F what is incorrect. We understand that the 3 test with TF is will not work. Therefor we will add NOT operation for y and will have the final function.



```
nand.f
t = lambda x. lambda y. x;
f = lambda x. lambda y. y;

not = lambda x.x f t;

and = lambda x. lambda y. x y f;
nand = lambda x. lambda y. not(and(x y));

/* >>>>>>> tests after this line <<<<< */

nand t t;
nand t f;
nand f f;
nand f t;
```

```
./f nand.f
t = lambda x. lambda y. x
f = lambda x. lambda y. y
not = lambda x. x f t
and = lambda x. lambda y. x y f
nand = lambda x. lambda y. not (and (x y))
(lambda x. lambda y. y)
(lambda x. lambda y. x)
(lambda x. lambda y. x)
(lambda x. lambda y. x)
dastan@dastan-VirtualBox:~/fulluntyped$
```

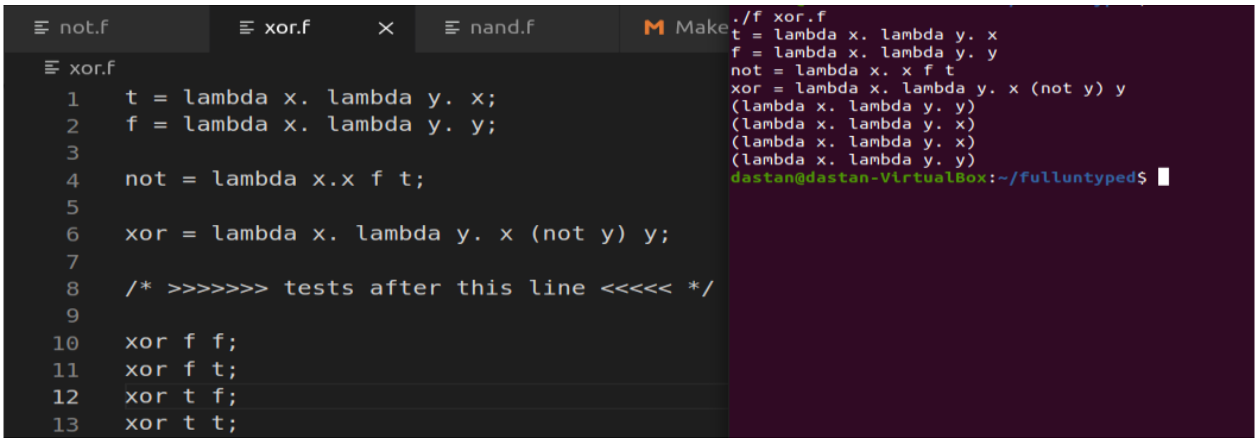
<u>XOR = λx. λy. x (NOT y) y</u>

For this function we must apply two variables, **T** OR **F**(Like (λx. λy. x(NOT y)y)TF).

We know that XOR operation must give us next table:

A	B	A **XOR** B
0	0	0
0	1	1
1	0	1
1	1	0

We know that OR functions Church Boolean representation is λx. λy. xTy. First of all, we must mention that XOR operations output is very same with OR operations output. IF x will be T then output must be false. Therefor, let us try change T from OR operation to F. Then we will take λx. λy. xFy, which will work when we apply TT. But if we apply TF, we will have TFF which will give us F which is incorrect. We can make a conclusion that in both situations, if we, instead of F will put NOTy, we will have 0 as output when apply TT. Also, if we apply TF, we will have TTF, which will give us T, which is correct.



```
t = lambda x. lambda y. x;
f = lambda x. lambda y. y;

not = lambda x.x f t;

xor = lambda x. lambda y. x (not y) y;

/* >>>>>>> tests after this line <<<<< */

xor f f;
xor f t;
xor t f;
xor t t;
```

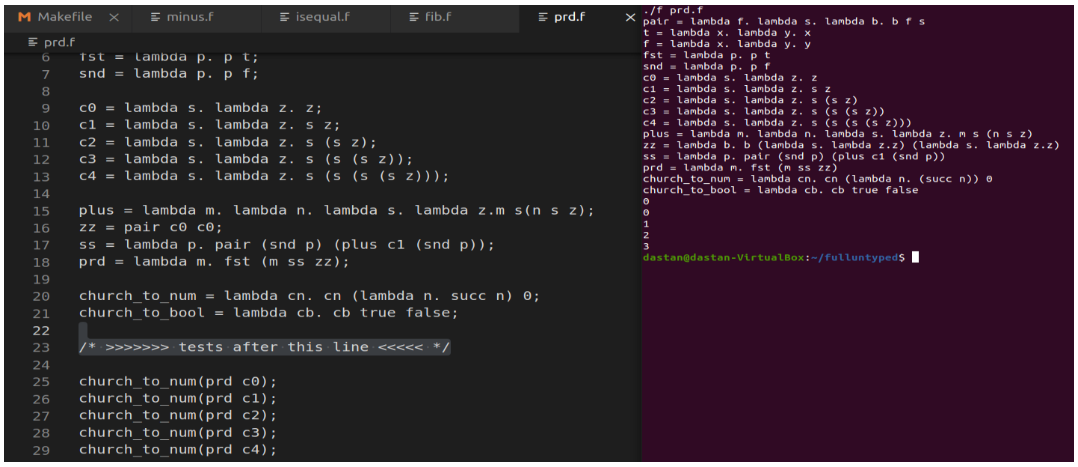
```
./f xor.f
t = lambda x. lambda y. x
f = lambda x. lambda y. y
not = lambda x. x f t
xor = lambda x. lambda y. x (not y) y
(lambda x. lambda y. y)
(lambda x. lambda y. x)
(lambda x. lambda y. x)
(lambda x. lambda y. y)
dastan@dastan-VirtualBox:~/fulluntyped$
```

Exercise 2. using Churh Numerals:

<u>Pred =(ƛn. n Next-pair(0, 0)) ƛxy. x</u>

There, in this Church Numerals Predecessor operation, Next-pair operation means (ƛp. pxy z. z (ƛp. pxy) ƛxy. y S(ƛp. pxy) ƛxy. y). There we see operation which will show us next pair of numbers. Where first of the pair always will be smaller for one than the second number of pair. Because we just do a successor operation for the second number of the previous pair and past it as the second pairs second number. While, as a first number of new pair we will past the second number of the previous pair without successor operation.

Then, in our main predecessor function we see how we give a number n as a how many times this next pair operation must be completed. Then we just take the first number from new pair. For example, if we will give number 5. Then, at the end we will have a pair (4, 5). And first number will be 4. Which is predecessor of 5.

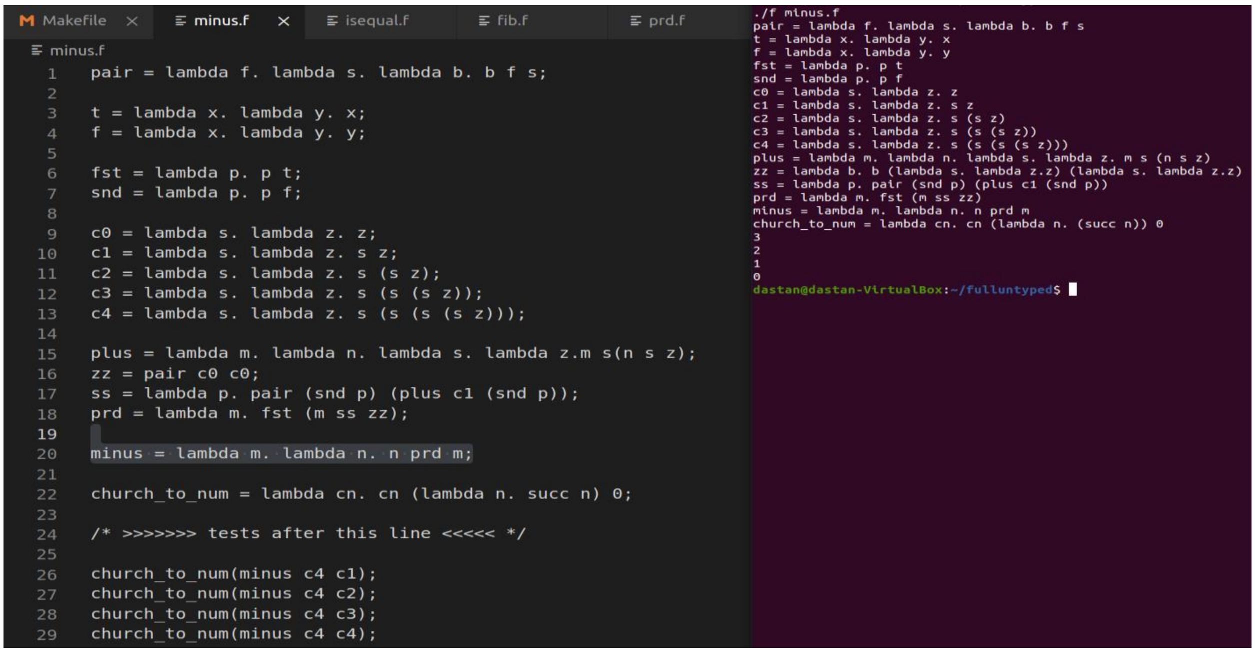


```
fst = lambda p. p t;
snd = lambda p. p f;

c0 = lambda s. lambda z. z;
c1 = lambda s. lambda z. s z;
c2 = lambda s. lambda z. s (s z);
c3 = lambda s. lambda z. s (s (s z));
c4 = lambda s. lambda z. s (s (s (s z)));

plus = lambda m. lambda n. lambda s. lambda z.m s(n s z);
zz = pair c0 c0;
ss = lambda p. pair (snd p) (plus c1 (snd p));
prd = lambda m. fst (m ss zz);

church_to_num = lambda cn. cn (lambda n. succ n) 0;
church_to_bool = lambda cb. cb true false;

/* >>>>>>> tests after this line <<<<< */

church_to_num(prd c0);
church_to_num(prd c1);
church_to_num(prd c2);
church_to_num(prd c3);
church_to_num(prd c4);
```

```
./f prd.f
pair = lambda f. lambda s. lambda b. b f s
t = lambda x. lambda y. x
f = lambda x. lambda y. y
fst = lambda p. p t
snd = lambda p. p f
c0 = lambda s. lambda z. z
c1 = lambda s. lambda z. s z
c2 = lambda s. lambda z. s (s z)
c3 = lambda s. lambda z. s (s (s z))
c4 = lambda s. lambda z. s (s (s (s z)))
plus = lambda m. lambda n. lambda s. lambda z. m s (n s z)
zz = lambda b. b (lambda s. lambda z.z) (lambda s. lambda z.z)
ss = lambda p. pair (snd p) (plus c1 (snd p))
prd = lambda m. fst (m ss zz)
church_to_num = lambda cn. cn (lambda n. (succ n)) 0
church_to_bool = lambda cb. cb true false
0
0
1
2
3
dastan@dastan-VirtualBox:~/fulluntyped$
```

<u>Minus =(λn. λm. nm (n P m))</u>

This operation is very simple if predecessor operation is understanded well. We just apply n times predecessor function to m. If n will be greater, then answer will be zero. If n and m will be equal then answer also be zero. And only if n will be less than m, answer of subtraction function will be non-zero. Therefor it will be useful to also mention iszero function which will be done with a function which will apply a one number and will return T or F. iszero = λa. a FALSE NOT FALSE. If we will give any number instead of 0, we will take "FALSE NOT FALSE" which will return us FALSE. But if we will use 0, then we will take "NOT FALSE", which will give us TRUE. And by using subtraction function in iszero, we can easily identify is number zero or not.

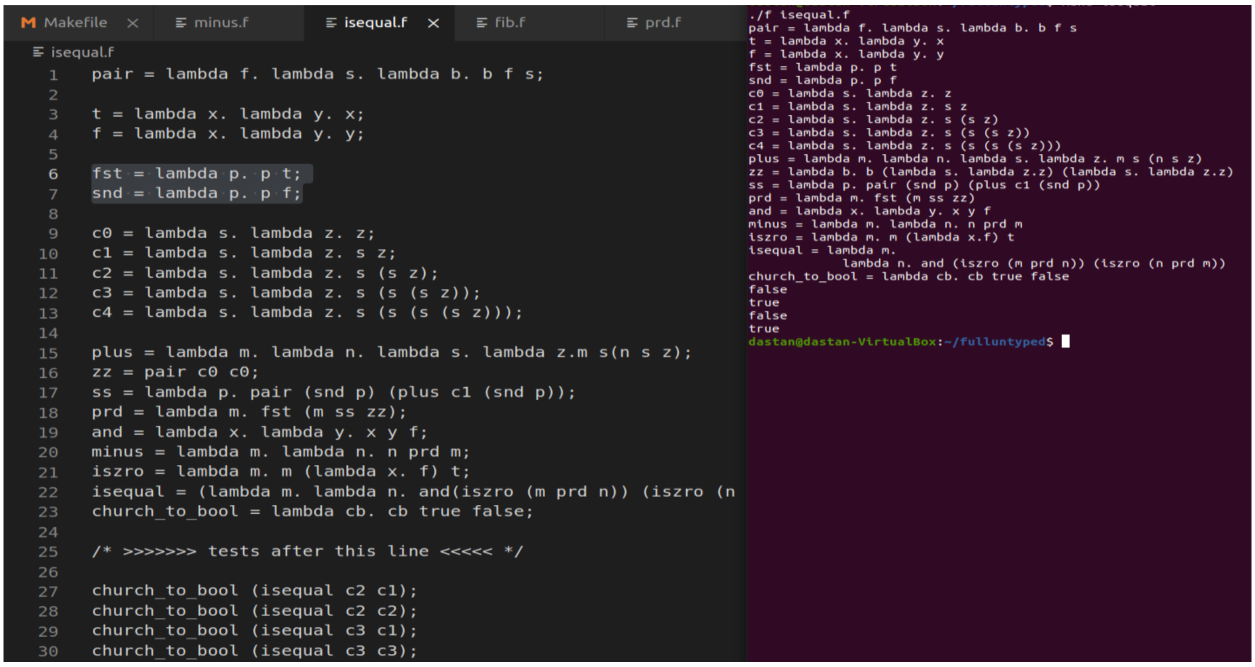


```
pair = lambda f. lambda s. lambda b. b f s;

t = lambda x. lambda y. x;
f = lambda x. lambda y. y;

fst = lambda p. p t;
snd = lambda p. p f;

c0 = lambda s. lambda z. z;
c1 = lambda s. lambda z. s z;
c2 = lambda s. lambda z. s (s z);
c3 = lambda s. lambda z. s (s (s z));
c4 = lambda s. lambda z. s (s (s (s z)));

plus = lambda m. lambda n. lambda s. lambda z.m s(n s z);
zz = pair c0 c0;
ss = lambda p. pair (snd p) (plus c1 (snd p));
prd = lambda m. fst (m ss zz);

minus = lambda m. lambda n. n prd m;

church_to_num = lambda cn. cn (lambda n. succ n) 0;

/* >>>>>>> tests after this line <<<<< */

church_to_num(minus c4 c1);
church_to_num(minus c4 c2);
church_to_num(minus c4 c3);
church_to_num(minus c4 c4);
```

```
./f minus.f
pair = lambda f. lambda s. lambda b. b f s
t = lambda x. lambda y. x
f = lambda x. lambda y. y
fst = lambda p. p t
snd = lambda p. p f
c0 = lambda s. lambda z. z
c1 = lambda s. lambda z. s z
c2 = lambda s. lambda z. s (s z)
c3 = lambda s. lambda z. s (s (s z))
c4 = lambda s. lambda z. s (s (s (s z)))
plus = lambda m. lambda n. lambda s. lambda z. m s (n s z)
zz = lambda b. b (lambda s. lambda z.z) (lambda s. lambda z.z)
ss = lambda p. pair (snd p) (plus c1 (snd p))
prd = lambda m. fst (m ss zz)
minus = lambda m. lambda n. n prd m
church_to_num = lambda cn. cn (lambda n. (succ n)) 0
3
2
1
0
dastan@dastan-VirtualBox:~/fulluntyped$ 
```

<u>Isequal =(ƛn. ƛm. AND(iszero(n P m) iszero(m P n)))</u>

This operation will use iszero operation which we mentioned in subtraction operation. Also, we see that we use AND operation which is ƛx. ƛy. xy F. By the knowledge from explanation of iszero, we know that for example, if we will have two numbers 4 and 5, and will apply it to iszero (4 P 5) and iszero (5 P 4). We will have FALSE for first and TRUE for second. Because 5 times done predecessor for 4 will give us 0. The reason is that there are no negative numbers applied. So only when we will have the same numbers n and m, we will have TRUE and TRUE. Consequently, if we have 3 cases. Where AND will accept next: 1)“TRUE, TRUE”, 2)“TRUE, FALSE” 3)“FALSE, TRUE”. Applying 1 case to AND will give us TTF, which is TRUE. 2 case will give us TFT, which is FALSE. Last 3 case will give us FTF, which is FALSE. Therefore, out Isequal operation is correct.



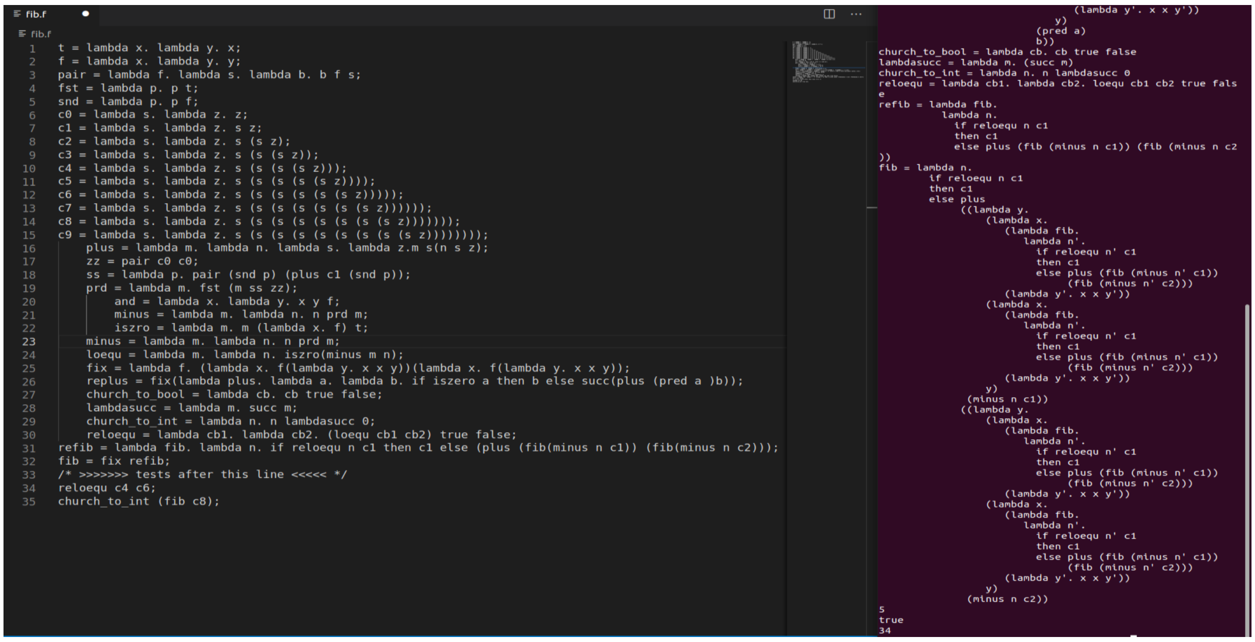
```
pair = lambda f. lambda s. lambda b. b f s;

t = lambda x. lambda y. x;
f = lambda x. lambda y. y;

fst = lambda p. p t;
snd = lambda p. p f;

c0 = lambda s. lambda z. z;
c1 = lambda s. lambda z. s z;
c2 = lambda s. lambda z. s (s z);
c3 = lambda s. lambda z. s (s (s z));
c4 = lambda s. lambda z. s (s (s (s z)));

plus = lambda m. lambda n. lambda s. lambda z.m s(n s z);
zz = pair c0 c0;
ss = lambda p. pair (snd p) (plus c1 (snd p));
prd = lambda m. fst (m ss zz);
and = lambda x. lambda y. x y f;
minus = lambda m. lambda n. n prd m;
iszro = lambda m. m (lambda x. f) t;
isequal = (lambda m. lambda n. and(iszro (m prd n)) (iszro (n
church_to_bool = lambda cb. cb true false;

/* >>>>>>> tests after this line <<<<< */

church_to_bool (isequal c2 c1);
church_to_bool (isequal c2 c2);
church_to_bool (isequal c3 c1);
church_to_bool (isequal c3 c3);
```

```
./f isequal.f
pair = lambda f. lambda s. lambda b. b f s
t = lambda x. lambda y. x
f = lambda x. lambda y. y
fst = lambda p. p t
snd = lambda p. p f
c0 = lambda s. lambda z. z
c1 = lambda s. lambda z. s z
c2 = lambda s. lambda z. s (s z)
c3 = lambda s. lambda z. s (s (s z))
c4 = lambda s. lambda z. s (s (s (s z)))
plus = lambda m. lambda n. lambda s. lambda z. m s (n s z)
zz = lambda b. b (lambda s. lambda z.z) (lambda s. lambda z.z)
ss = lambda p. pair (snd p) (plus c1 (snd p))
prd = lambda m. fst (m ss zz)
and = lambda x. lambda y. x y f
minus = lambda m. lambda n. n prd m
iszro = lambda m. m (lambda x.f) t
isequal = lambda m.
            lambda n. and (iszro (m prd n)) (iszro (n prd m))
church_to_bool = lambda cb. cb true false
false
true
false
true
dastan@dastan-VirtualBox:~/fulluntyped$
```

Exercise 3. Implement function fib(n) that returns n-th Fibonacci number using

a) Y-combinator => Y (f. ƛn. If (n <= 1) (ƛy. (f (n - 1)) + (f (n - 2))) (ƛy. n))
Where Y combinator is ƛf. (ƛx. f (x x)) (ƛx. f (x x)).
b) Z-combinator => Z (f. ƛn. If (n <= 1) (ƛy. (f (n - 1)) + (f (n - 2))) (ƛy. n))
Where Z combinator is ƛf. (ƛx. f (ƛy. x x y)) (ƛx. f (ƛy. x x y)).

This two implementations might be seen that very same to each other. While the main point of difference is that Y combinator or in other words call-by-name fixed point combinator is completely useless in call by value setting. Because every time while we will compile Y combinator, it will not give us a value by default. For example: Y foo => foo (Y foo) => foo (foo (Y foo)) and so on. Therefor, there was implemented Z combinator which is also called call-by-value Y combinator. But I will call it like Z combinator. Real languages usually do call by value. So, Z combinator is useful when the first of all must be calculated the value of the given function. And only after value calculating, it will go on.



```
t = lambda x. lambda y. x;
f = lambda x. lambda y. y;
pair = lambda f. lambda s. lambda b. b f s;
fst = lambda p. p t;
snd = lambda p. p f;
c0 = lambda s. lambda z. z;
c1 = lambda s. lambda z. s z;
c2 = lambda s. lambda z. s (s z);
c3 = lambda s. lambda z. s (s (s z));
c4 = lambda s. lambda z. s (s (s (s z)));
c5 = lambda s. lambda z. s (s (s (s (s z))));
c6 = lambda s. lambda z. s (s (s (s (s (s z)))));
c7 = lambda s. lambda z. s (s (s (s (s (s (s z))))));
c8 = lambda s. lambda z. s (s (s (s (s (s (s (s z)))))));
c9 = lambda s. lambda z. s (s (s (s (s (s (s (s (s z))))))));
    plus = lambda m. lambda n. lambda s. lambda z.m s(n s z);
    zz = pair c0 c0;
    ss = lambda p. pair (snd p) (plus c1 (snd p));
    prd = lambda m. fst (m ss zz);
        and = lambda x. lambda y. x y f;
        minus = lambda m. lambda n. n prd m;
        iszro = lambda m. m (lambda x. f) t;
    minus = lambda m. lambda n. n prd m;
    loequ = lambda m. lambda n. iszro(minus m n);
    fix = lambda f. (lambda x. f(lambda y. x x y))(lambda x. f(lambda y. x x y));
    replus = fix(lambda plus. lambda a. lambda b. if iszero a then b else succ(plus (pred a )b));
    church_to_bool = lambda cb. cb true false;
    lambdasucc = lambda m. succ m;
    church_to_int = lambda n. n lambdasucc 0;
    reloequ = lambda cb1. lambda cb2. (loequ cb1 cb2) true false;
refib = lambda fib. lambda n. if reloequ n c1 then c1 else (plus (fib(minus n c1)) (fib(minus n c2)));
fib = fix refib;
/* >>>>>>> tests after this line <<<<< */
reloequ c4 c6;
church_to_int (fib c8);
```

```
                               (lambda y'. x x y'))
                            y)
                         (pred a)
                         b))
church_to_bool = lambda cb. cb true false
lambdasucc = lambda m. (succ m)
church_to_int = lambda n. n lambdasucc 0
reloequ = lambda cb1. lambda cb2. loequ cb1 cb2 true fals
e
refib = lambda fib.
          lambda n.
            if reloequ n c1
            then c1
            else plus (fib (minus n c1)) (fib (minus n c2
))
fib = lambda n.
        if reloequ n c1
        then c1
        else plus
             ((lambda y.
                 (lambda x.
                    (lambda fib.
                       lambda n'.
                         if reloequ n' c1
                         then c1
                         else plus (fib (minus n' c1))
                              (fib (minus n' c2)))
                    (lambda y'. x x y'))
                 (lambda x.
                    (lambda fib.
                       lambda n'.
                         if reloequ n' c1
                         then c1
                         else plus (fib (minus n' c1))
                              (fib (minus n' c2)))
                    (lambda y'. x x y'))
                 y)
              (minus n c1))
             ((lambda y.
                 (lambda x.
                    (lambda fib.
                       lambda n'.
                         if reloequ n' c1
                         then c1
                         else plus (fib (minus n' c1))
                              (fib (minus n' c2)))
                    (lambda y'. x x y'))
                 (lambda x.
                    (lambda fib.
                       lambda n'.
                         if reloequ n' c1
                         then c1
                         else plus (fib (minus n' c1))
                              (fib (minus n' c2)))
                    (lambda y'. x x y'))
                 y)
              (minus n c2))
5
true
34
```