Homework 4

Task 1

1. What is the output for test “if succ(0) then true else (if true the succ(2) else succ(3));” in untyped implementation? Why? Why term in brackets won’t reduce?

```
ubuntu@ubuntu2004:~/Documents/plt/fulluntyped$ cat a.txt
if succ(0) then true else (if true then succ(2) else succ(3));
ubuntu@ubuntu2004:~/Documents/plt/fulluntyped$ ./f a.txt
(if 1 then true else if true then 3 else 4)
```

   It will be “if 1 then true else (if true the succ(2) else succ(3));” because by the rules the guard is computed first, and then there are no rules when the guard is not “true” or “false”

2. Perform same for typed implementation and explain result

```
ubuntu@ubuntu2004:~/Documents/plt/fullsimple$ cat a.txt
if succ(0) then true else (if true then succ(2) else succ(3));
ubuntu@ubuntu2004:~/Documents/plt/fullsimple$ ./f a.txt
/home/ubuntu/Documents/plt/fullsimple/a.txt:1.0:
guard of conditional not a boolean
```

3. What evaluation strategy is implemented? Reason it, show how you understood it.
   Typed lambda calculus uses the same main rules from the previous chapters, thus it uses the same evaluation strategy which is *call-by-value.*
4. eval function is implemented as repeatedly calling eval1 of term. How it knows when it reached the normal form (value or stuck state)?
   eval1 works in such a way that it raises exception *NoRuleApplies* when the term is in normal form. This way *eval* knows when *eval1* has finished. The left screenshot shows eval function, the right one shows the end of the eval1 function

```
let rec eval ctx t =
  try let t' = eval1 ctx t
      in eval ctx t'
  with NoRuleApplies -> t
```

```
    let t1' = eval1 ctx t1 in
    TmLet(fi, x, t1', t2)
| _ ->
    raise NoRuleApplies
```

5. Exc 8.3.5
   Technically, we cand do this, but we should not, because it will cause *leq* and *eq,* which relies on *leq*, functions to break

```
minus = lambda a. lambda b. b predok a;

is0 = lambda n. n (lambda x. fls) tru;
leq = lambda a. lambda b. is0 (minus a b);
isequal = lambda a. lambda b.c_to_bool (and (leq a b) (leq b a));
```

6. Exc 8.3.6.
   Expansion property does not hold
   “If true then 1 else false” – this statement is completely fine and will be reduced to *1*, but the *if term* is not of any type, however *1* is of *Nat* type.

Task 2

```
ubuntu@ubuntu2004:~/Documents/plt/hw4$ cat test.f
succ true;
pred false;
iszero false;
if 1 then true else false;ubuntu@ubuntu2004:~/Documents/plt/hw4$ ./f test.f
You can only get successor of a number! >:(
You can only get predecessor of a number! >:(
You can only know if a number is zero! >:(
The if guard must be boolean! >:(
ubuntu@ubuntu2004:~/Documents/plt/hw4$
```

What I have changed:

Core.ml

```
15  let rec isbadnat t = match t with
16      TmWrong(_,_) -> true
17    | TmTrue(_) -> true
18    | TmFalse(_) -> true
19    | _ -> false
20
21  let rec isbadbool ctx t = match t with
22      TmWrong(_,_) -> true
23    | t when isnumericval ctx t -> true
24    | _ -> false
```

```
42    | TmIf(_, badbool, t2, t3) when (isbadbool ctx badbool) ->
43        TmWrong(dummyinfo,"The if guard must be boolean! >:(")
```

```
84    | TmSucc(_, badnat) when (isbadnat badnat) ->
85        TmWrong(dummyinfo,"You can only get successor of a number! >:(")
86    | TmSucc(fi,t1) ->
87        let t1' = eval1 ctx t1 in
88        TmSucc(fi, t1')
89    | TmPred(_, badnat) when (isbadnat badnat) ->
90        TmWrong(dummyinfo,"You can only get predecessor of a number! >:(")
91    | TmPred(_,TmZero(_)) ->
92        TmZero(dummyinfo)
93    | TmPred(_,TmSsucc(_,nv1)) when (isnumericval ctx nv1) ->
94        nv1
95    | TmPred(fi,t1) ->
96        let t1' = eval1 ctx t1 in
97        TmPred(fi, t1')
98    | TmIsZero(_, badnat) when (isbadnat badnat) ->
99        TmWrong(dummyinfo,"You can only know if a number is zero! >:(")
```

Syntax.ml

```
25    | TmWrong of info * string
26
```

```
81    | TmFalse(fi) as t -> t
82    | TmWrong(fi,msg) as t -> t
```

```
157   | TmWrong(fi,_) -> fi
```

```
233 and printtm_ATerm outer ctx t = match t with
234     | TmWrong(fi,msg) -> pr msg
235     | TmTrue(_) -> pr "true"
```

Syntax.mli

```
24    | TmWrong of info * string
25
```