חלק 1:

א. לא נכון, G מקבלת T1 כקלט ולא Number שהוא הטיפוס של a.
ב. לא נכון, F מקבלת T2 כקלט ולא T1 שהוא הטיפוס של x.
ג. לא נכון, הערך המוחזר מהביטוי הוא closure זאת אומרת [T2 -> ()] ולא T2.
ד. נכון, x בתוך הlambda אינו פרמטר חופשי, ובפרט טיפוסו הוא T1 מכיוון שאינו תלוי בסביבה וגם הפרמטר הראשון בביטוי f x y , נתון y מטיפוס T2, ובפרט נקבל lambda חוקית, (f x y) חוקי, טיפוס המוחזר מה-lambda הוא [T2<-T1].

חלק 2:

(a) ((lambda (f x1) (f 1 x1)) + #t)

Stage 1: Rename Bound variables.
((lambda (f x1) (f 1 x1)) + #t) turn to ((lambda (f x) (f 1 x)) + #t)

Stage 2: Assign type variable for every sub expression:

Expression	Variable
((lambda (f x) (f 1 x)) + #t)	T0
(lambda (f x) (f 1 x))	T1
(f 1 x)	T2
f	Tf
x	Tx
+	TaddOp
#t	Ttrue

Stage 3: construct type equations
The equations for the sub-expression are:

Expression	Variable
((lambda (f x) (f 1 x)) + #t)	T1 = [TaddOp * Ttrue -> T0]
((lambda (f x) (f 1 x))	T1 = [Tf * Tx -> T2]
(f 1 x)	Tf = [Tnum1 * Tx -> T2]

The equations for the primitive are:

Expression	Variable
+	TaddOp = [Number * Number -> Number]
#t	Ttrue = Boolean

Stage 4: Solve the equations:

Equation	Substitution
1. T1 = [TaddOp * Ttrue -> T0]	{}
2. T1 = [Tf * Tx -> T2]	
3. Tf = [Tnum1 * Tx -> T2]	
4. TaddOp = [Number * Number -> Number]	
5. Ttrue = Boolean	

Step 1:

`(`T1 = [TaddOp * Ttrue -> T0]`) ∘ Substitution = (`T1 = [TaddOp * Ttrue -> T0]) and is a type-sub.

`Substitution = Substitution ∘ (`T1 = [TaddOp * Ttrue -> T0]).

Equation	Substitution
2. T1 = [Tf * Tx -> T2]	{T1 := [[TaddOp * Ttrue -> T0]}
3. Tf = [Tnum1 * Tx -> T2]	
4. TaddOp = [Number * Number -> Number]	
5. Ttrue = Boolean	

Step 2: T1 = [Tf * Tx -> T2] `∘ Substitution = (`[TaddOp * Ttrue -> T0] = [Tf * Tx -> T2])

There is no type-sub since both sides of the equation are composite we split it into three equations (6,7,8) and remove equation 2.

Equation	Substitution
3. Tf = [Tnum1 * Tx -> T2]	{T1 := [[TaddOp * Ttrue -> T0]}
4. TaddOp = [Number * Number -> Number]	
5. Ttrue = Boolean	
6. Tf = TaddOp	
7. Tx = Ttrue	
8. T2 = T0	

Step 3:

(Tf = [Tnum1 * Tx -> T2]) ○ Substitution = (Tf = [Tnum1 * Tx -> T2]).
Substitution = Substitution ○ (Tf = [Tnum1 * Tx -> T2]).

Equation	Substitution
4. TaddOp = [Number * Number -> Number]	{T1 := [[TaddOp * Ttrue -> T0], Tf := [Tnum1 * Tx -> T2]}
5. Ttrue = Boolean	
6. Tf = TaddOp	
7. Tx = Ttrue	
8. T2 = T0	

Step 4:

(TaddOp = [Number * Number -> Number]) ○ Substitution = (TaddOp = [Number * Number -> Number]). Substitution = Substitution ○ (TaddOp = [Number * Number -> Number]).

Equation	Substitution
5. Ttrue = Boolean	{T1 := [[[Number * Number -> Number] * Ttrue -> T0], Tf := [Tnum1 * Tx -> T2], TaddOp := [Number * Number -> Number]}
6. Tf = TaddOp	
7. Tx = Ttrue	
8. T2 = T0	

Step 5:

(Ttrue = Boolean) ○ Substitution = (Ttrue = Boolean), is a type-sub. Substitution = Substitution ○ (Ttrue = Boolean).

Equation	Substitution
6. Tf = TaddOp	{T1 := [[[Number * Number -> Number] * Boolean -> T0], Tf := [Tnum1 * Tx -> T2], TaddOp := [Number * Number -> Number], Ttrue := Boolean }
7. Tx = Ttrue	
8. T2 = T0	

Step 6:

(Tf = TaddOp) ○ Substitution =[Tnum1 * Tx -> T2] = [Number * Number -> Number]) There is no a sub-type. We split the equation into three equations.

Tnum1 = Number, Tx = Number, T2 = Number.

We remove the equation number 6.

Equation	Substitution
7. Tx = Ttrue	{T1 := [[[Number * Number -> Number] * Boolean -> T0], Tf := [Tnum1 * Tx -> T2], TaddOp := [Number * Number -> Number], Ttrue := Boolean }
8. T2 = T0	
9. Tnum1 = Number	
10. Tx = Number	
11. T2 = Number	

Step 7:

(Tx = Ttrue) ○ Substitution = ([Tx = Boolean]), type-sub. Substitution = Substitution ○ ([Tx = Boolean]).

Equation	Substitution
8. T2 = T0	{T1 := [[[Number * Number -> Number] * Boolean -> T0], Tf := [Tnum1 * Boolean -> T2], Tx = Ttrue, TaddOp := [Number * Number -> Number], Ttrue := Boolean }
9. Tnum1 = Number	
10. Tx = Number	
11. T2 = Number	

Step 8:

(T2 = T0) ○ Substitution = (T2 = T0), type-sub. Substitution = Substitution ○ (T2 = T0).

Equation	Substitution
9. Tnum1 = Number	{T1 := [[[Number * Number -> Number] * Boolean -> T0], Tf := [Tnum1 * Boolean -> T2], Tx = Ttrue, TaddOp := [Number * Number -> Number], Ttrue := Boolean, T2:= T0 }
10. Tx = Number	
11. T2 = Number	

Step 9: (Tnum1 = Number) ○ Substitution = (Number = Number) always true.

Equation	Substitution
10. Tx = Number	{T1 := [[[Number * Number -> Number] * Boolean -> T0], Tf := [Tnum1 * Boolean -> T2], Tx = Ttrue, TaddOp := [Number * Number -> Number], Ttrue := Boolean, T2:= T0,Tnum1 = Number }
11. T2 = Number	

Step 10:

(Tx = Number) ○ Substitution = (Tx = Number), here we have a conflict because we got that Number = Tx = Ttrue = Boolean, therefore we can say that the expression is not well typed. FAILED!

(b) ((lambda (f1 x1) (f1 x1 1)) + *)

Stage 1: Rename Bound variables.

((lambda (f1 x1) (f1 x1 1)) + *) turn to ((lambda (f x) (f x 1)) + *)

Stage 2: Assign type variable for every sub expression:

Expression	Variable
((lambda (f x) (f x 1)) + *)	T0
(lambda (f x) (f x 1))	T1
(f x 1)	T2
f	Tf
x	Tx
1	Tnum1
+	TaddOp
*	TmulOp

Stage 3: construct type equations
The equations for the sub-expression are:

Expression	Variable
((lambda (f x) (f x 1)) + *)	T1 = [TaddOp * TmulOp -> T0]
(lambda (f x) (f x 1))	T1 = [Tf * Tx -> T2]
(f x 1)	Tf = [Tx * Tnum1 -> T2]

The equations for the primitive are:

Expression	Variable
1	Tnum1 = Number
+	TaddOp = [Number * Number -> Number]
*	TmulOp = [Number * Number -> Number]

Stage 4: Solve the equations:

Equation	Substitution
1. T1 = [TaddOp * TmulOp -> T0]	{}
2. T1 = [Tf * Tx -> T2]	
3. Tf = [Tx * Tnum1 -> T2]	
4. TaddOp = [Number * Number -> Number]	
5. Tnum1 = Number	
6. TmulOp = [Number * Number -> Number]	

(T1 = [TaddOp * TmulOp -> T0] ∘ Substitution = (T1 = [TaddOp * TmulOp -> T0]), type-sub.

Substitution = Substitution ∘ (T1 = [TaddOp * TmulOp -> T0]).

Equation	Substitution
2. T1 = [Tf * Tx -> T2]	{ T1 := [TaddOp * TmulOp -> T0] }
3. Tf = [Tx * Tnum1 -> T2]	
4. TaddOp = [Number * Number -> Number]	
5. Tum1 = Number	
6. TmulOp = [Number * Number -> Number]	

Step 2:

(T1 = [Tf * Tx -> T2]) ∘ Substitution = ([Tf * Tx -> T2] = [TaddOp * TmulOp -> T0])

There is no type-sub. We split the equation to

- Tf = TaddOp,
- Tx = TmulOp,
- T2 = T0

We add them to the equations (7,8,9) and remove equation 2.

Equation	Substitution
3. Tf = [Tx * Tnum1 -> T2]	{ T1 := [TaddOp * TmulOp -> T0] }
4. TaddOp = [Number * Number -> Number]	
5. Tum1 = Number	
6. TmulOp = [Number * Number -> Number]	
7. Tf = TaddOp	
8. Tx = TmulOp	
9. T2 = T0	

Step 3:

(Tf = [Tx * Tnum1 -> T2] ○ Substitution = (Tf =[Tx * Tnum1 -> T2]) , type-sub.
Substitution = Substitution ○ (Tf = [Tx * Tnum1 -> T2]).

Equation	Substitution
4. TaddOp = [Number * Number -> Number]	{ T1 := [TaddOp * TmulOp -> T0], Tf := [Tx * Tnum1 -> T2] }
5. Tum1 = Number	
6. TmulOp = [Number * Number -> Number]	
7. Tf = TaddOp	
8. Tx = TmulOp	
9. T2 = T0	

Step 4: (TaddOp = [Number * Number -> Number]) ○ Substitution = (TaddOp = [Number * Number -> Number]), type-sub.

Substitution = Substitution ○ (TaddOp = [Number * Number -> Number]).

Equation	Substitution
5. Tnum1 = Number	{ T1 := [[Number * Number -> Number]* TmulOp -> T0], Tf := [Tx * Tnum1 -> T2], TaddOp := [Number * Number -> Number] }
6. TmulOp = [Number * Number -> Number]	
7. Tf = TaddOp	
8. Tx = TmulOp	
9. T2 = T0	

Step 5: (Tnum1 = Number) ○ Substitution = (Tnum1 = Number), type-sub.

Substitution = Substitution ○(Tnum1 = Number).

Equation	Substitution
6. TmulOp = [Number * Number -> Number]	{ T1 := [[Number * Number -> Number]* TmulOp -> T0], Tf := [Tx * Number -> T2], TaddOp := [Number * Number -> Number], Tnum1 := Number }
7. Tf = TaddOp	
8. Tx = TmulOp	
9. T2 = T0	

Step 6:

(TmulOp = [Number * Number -> Number]) ○ Substitution = (TmulOp = [Number * Number -> Number]) , type-sub. Substitution = Substitution ○ (TmulOp = [Number * Number -> Number]).

Equation	Substitution
7. Tf = TaddOp	{ T1 := [[Number * Number -> Number]* [Number * Number -> Number]-> T0], Tf := [Tx * Number -> T2], TaddOp := [Number * Number -> Number], Tnum1 := Number, TmulOp := [Number * Number -> Number] }

Equation	Substitution
8. Tx = TmulOp	
9. T2 = T0	

Step 7:
(Tf = TaddOp) ○ Substitution = ([Tx*Number -> T2] = [Number * Number -> Number])

There is no type-sub since both sides of the equation are composite, therefore we split the equation to two sub equations (10, 11) and remove equation 6. (no need to add Number = Number)

Equation	Substitution
8. Tx = TmulOp	{ T1 := [[Number * Number -> Number]* [Number * Number -> Number]-> T0], Tf := [Tx * Number -> T2], TaddOp := [Number * Number -> Number], Tnum1 := Number, TmulOp := [Number * Number -> Number], }
9. T2 = T0	
10. Tx = number	
11. T2 = Number	

Step 8:
(Tx = TmulOp) ○ Substitution = (Tx = TmulOp) , type-sub. Substitution = Substitution ○ (Tx = TmulOp).

Equation	Substitution
9. T2 = T0	{ T1 := [[Number * Number -> Number]* [Number * Number -> Number]-> T0], Tf := [[Number * Number -> Number]* Number -> T2], TaddOp := [Number * Number -> Number], Tnum1 := Number, TmulOp := [Number * Number -> Number],

Equation	Substitution
	Tx := [Number * Number -> Number] }
10. Tx = number	
11. T2 = Number	

Step 9:
(T2 = T0) ○ Substitution = (T2 = T0) , type-sub. Substitution = Substitution ○ (T2 = T0).

Equation	Substitution
10. Tx = number	{ T1 := [[Number * Number -> Number]* [Number * Number -> Number]-> T0], Tf := [[Number * Number -> Number]* Number -> T2], TaddOp := [Number * Number -> Number], Tnum1 := Number, TmulOp := [Number * Number -> Number], Tx := [Number * Number -> Number], T0 = T2 }
11. T2 = Number	

Step 10:
(Tx = number) ○ Substitution = ([[Number * Number -> Number] = Number)

We get the conflicting equation:
[Number * Number -> Number] = Number, and we can say that the expression is not well typed.

חלק 3:

Typing rule set!:
For every: type environment _Tenv,
variable reference _x1
expressions _e1 and
type expressions _S1:
If _Tenv |- _e1 : _S1 and
Tenv |- _x1 : _S1_
Then _Tenv |- (set! _x1 _e1) : void

Typing rule lit:
For every: type environment _Tenv,
variable reference _x1
expressions _e1 and
type expressions _S1:
If Tenv | {_e1 : _S1 , _x1 : _S1}
Then _Tenv |- (lit _x1 _e1) : void

Typing rule define-type:
For every: type environment _Tenv,
variable reference _x1
expressions _e1 and
type expressions _S1:
If Tenv o {_x1 : _S1} |- _e1 : _S1
Then _Tenv |- (define-type _x1 _e1) : void

Typing rule type-case:
For every: type environment _Tenv,
variable reference _x1
expressions _e1 and
type expressions _S1:
If   Tenv |- {_x1 : _S1} or Tenv |- {_e1 : _S1}

Then _Tenv |- (type-case _x1 _e1) : void