**CSE340 FALL 2021 HOMEWORK 5. Due Monday Thursday December 2nd by 11:59 PM**

**Problem 1 (Static and Dynamic Scoping)** Consider the following program written in a C-like syntax. Assume parameters are passed by value.

int a, b, c, d;

// global variables **initially 0**

int g(int a; int b)

{

print(“%d %d %d %d”, a, b, c, d);

return a + b + c + d;

}

int f(int c)

{

int b;

b = 3;

b = g(a,b);

{ int c;

int d;

d = 4;

c = 8;

g(a,b);

}

g(a,b);

return b;

}

void main()

{

int a;

int b;

a = 4;

b = 5;

a = f(b);

g(b,a);

}

**Static Scoping**

a\_global = 0

b\_global = 0

c\_global = 0

main()

a\_main = 4 -> 3

b\_main = 5

a\_main = f(b\_main) = f(5)

f(5)

c\_f = 5

b\_f = 3

b\_f = g(a\_global\_b\_f) = g(0,3)

g(0,3)

a\_g = 0

b\_g = 3

print

a\_g,  **0**

b\_g, **3**

c\_global, **0**

d\_global **0**

return 0+3+0+0 = 3

b\_f = g(0,3) = **3**

{ d\_local = 4

c\_local = 8

g(a\_global,b\_f) = g(0,3)

a\_g = 0

b\_g = 3

print

a\_g,  **0**

b\_g, **3**

c\_global, **0**

d\_global **0**

return 0+3+0+0 = 3

}

g(a\_global\_b\_f) = g(0,3)

g(0,3)

a\_g = 0

b\_g = 3

print

a\_g,  **0**

b\_g, **3**

c\_global, **0**

d\_global **0**

return 0+3+0+0 = 3

return b\_f = **3**

a\_main = f(5)= **3**

g(b\_main,a\_main) = g(5,3)

g(5,3)

a\_g = 5

b\_g = 3

print

a\_g,  **5**

b\_g,  **3**

c\_global, **0**

d\_global **0**

return 0+3+0+0 = 8

**Output 0 3 0 0**

**0 3 0 0**

**0 3 0 0**

**5 3 0 0**

int a, b, c, d;

// global variables **initially 0**

int g(int a; int b)

{

print(“%d %d %d %d”, a, b, c, d);

return a + b + c + d;

}

int f(int c)

{

int b;

b = 3;

b = g(a,b);

{ int c;

int d;

d = 4;

c = 8;

g(a,b);

}

g(a,b);

return b;

}

void main()

{

int a;

int b;

a = 4;

b = 5;

a = f(b);

g(b,a);

}

**Dynamic Scoping**

a\_global = 0

b\_global = 0

c\_global = 0

main()

a\_main = 4

b\_main = 5

a\_main = f(b\_main) = f(5)

f(5)

c\_f = 5

b\_f = 3 **12**

b\_f = g(a\_main\_b\_f) = g(4,3)

g(0,3)

a\_g = 4

b\_g = 3

print

a\_g,  **4**

b\_g, **3**

c\_f, **5**

d\_global **0**

return 4+3+5+0 = 12

b\_f = g(4,3) = **12**

{ d\_local = 4

c\_local = 8

g(a\_main,b\_f) = g(4,12)

a\_g = 4

b\_g = 12

print

a\_g,  **4**

b\_g, **12**

**c\_local**, **8**

**d\_local** **4**

return 4+12+8+4 = 28

}

g(a\_main\_b\_f) = g(4,12)

g(0,3)

a\_g = 4

b\_g = 12

print

a\_g,  **4**

b\_g, **12**

c\_f, **5**

d\_global **0**

return 4+12+5+0 = 21

return b\_f = **12**

a\_main = f(5)= **12**

g(b\_main,a\_main) = g(5,12)

g(5,12)

a\_g = 5

b\_g = 12

print

a\_g,  **5**

b\_g, **12**

c\_global, **0**

d\_global **0**

return 5+12+0+0 = 17

**Output 4 3 5 0**

**4 12 8 4**

**4 12 5 0**

**5 12 0 0**

**Problem 2 (Hindley Milner Type checking)**

For this problem, you should give the answers and you do not need to show your work if there is no type checking error. **If there is a type checking error, you should show your work**. You can use an online OCaml editor to check your answers, but you should not solely rely on that. If you do, you will not do well on the final.

For each question, just give the type of x or explain why there is a type mismatch.

1. What is the type of x in:

let f1 x = 1 + x ;;

**Answer** Tx = int

1. What is the type of x in:

let f2 a x = x 1 +. a ;;

**Answer**  x 1 is x applied to 1. So x is a function that takes an int argument.

x 1 is an operand of +., so x 1 is float

Tx = int 🡪 float

1. What is the type of x in:

let f3 x y f = f 1 + f x.(y) ;;

**Answer**

* For the expression f 1 , we have f applied to 1, so Tf = int 🡪 T1, where T1 is not constrained
* For the expression f x.(y) , we have f applied to x.(y), so Tf = Type (x.(y)) 🡪 T2 . so x.(y) = int and T1 = T2
* For the expression x.(y), we have Type ( i ) = int and Type (x) = Type (x.(y)) array = int array

Tx = int array

1. What is the type of x in:

let f4 f x i = f 1.0 + x.(i) + f i ;;

**Answer**

* For the expression x.(i) , we have Type ( i ) = int and Tx = T1 array, T1 is not constrained
* For the expression f 1.0 , we have f applied to 1.0 so Tf = float 🡪 T2 , T2 is not constrained
* For the expression f 1 , we have f applied to 1, so Tf = int 🡪 T3
* Tf = int 🡪 T3 and Tf = float 🡪 T2 means that int = float and T2 = T3, type mismatch

1. What is the type of x in:

let f5 x y i = x (if y.(i) then i else i+1) = y.(i);;

**Answer**

* The expression of the function compares two values
  + x (if y.(i) then i else i+1) and
  + y.(i)
* So, Type ( x (if y.(i) then i else i+1) ) = Type ( y.(i) )
* “if y.(i) then i else i+1” is an “if expression, so Type (if y.(i) then i else i+1) = Type (i) = Type (i+1) = int
* x is applied to “if y.(i) then i else i+1” , so x is a function that takes int as argument
* y.(i) is the condition of the “if” , so y.(i) has type bool
* x (if y.(i) then i else i+1) is compared to y.(i), so x (if y.(i) then i else i+1) also has type bool
* so Tx = int 🡪 bool

1. What is the type of x in:

let f6 x y = y x + x (y x) ;;

**Answer**

* Type (y x) = Type (x ( y x )) = int because they are operands to +
* Type (y x) = int , so Type (y) = Type(x) 🡪 int
* Type (x (y x)) = int, so Type (x) = Type (y x) 🡪 int = int 🡪 int
* It follows that Type (y) = Type (x) 🡪 int = (int 🡪 int) 🡪 int

1. What is the type of x in:

let f7 a b c = if b c then a c else b c ;;

**Answer**

* Cannot be answered because there is no x in the question

1. What is the type of x in:

let f8 x b c = if b c then x c else b (x c) ;;

**Answer**

* Type (b c ) = bool , Type ( x c ) = Type ( b ( x c) )
* Type (b c ) = bool, so Type ( b (x c) ) = bool and Type ( x c ) = bool
* The are no constraints on the type of c, so Tx = Tc 🡪 bool and Tb = Tc 🡪 bool
* Type ( b (x c) ) = bool and Type ( x c) = bool, so Tc = bool and Tx = Tb = bool 🡪 bool
* Type ( f8 ) = Tx 🡪 Tb 🡪 Tc 🡪 bool = (bool 🡪 bool) 🡪 (bool 🡪 bool) 🡪 bool 🡪 bool

1. What is the type of x in:

let rec f9 x m = match x with

[] -> x

| h::t -> h\*m:: f8 m t;;

**Answer**

* We answer the question as given. We now from part 8. above that

Type ( f8 ) = (bool 🡪 bool) 🡪 (bool 🡪 bool) 🡪 bool 🡪 bool

* For the expression h\*m, we have Type ( h ) = Type ( m ) = int because \* is integer multiplication
* For the expression h::t, we have Type ( t ) = Type ( h) List = int List
* For the expression f8 m t , we have Type ( f8 ) = Tm 🡪 Tt 🡪 T1 where T1 is not constrained
* But we know that Type ( f8 ) = (bool 🡪 bool) 🡪 (bool 🡪 bool) 🡪 bool 🡪 bool
* It follows that **Tm** = bool 🡪 bool and Tt = bool 🡪 bool and T1 = bool 🡪 bool
* But **Tm** = int, so we have type mismatch

1. what does f9 calculate? You can try different examples using the OCaml command line

**Answer**

f9 has type mismatch, so there is no answer

Note. This problem was supposed to be

let rec f9 x m = match x with

[] -> x

| h::t -> h\*m:: f9 t m ;;

but the final version of the file was not saved. The function was supposed to multiply every element of an int List x with an integer m

1. What is the type of x in:

let rec f10 x l = match l with

[] -> []

| h::t -> x h :: f10 x t;;

**Answer**

This is a generalization of the previous problem

* For the match expression, we have
  + Patterns have the same type: Type ( l ) = Type ( [] ) = Type ( h :: t ) , so Type Type ( l ) = Type ( h :: t ) = T1 List, where T1 is not constrained
  + Expressions have the same type: Type ( [] ) = Type ( x h :: f10 x t ) = T2 List
* For h::t expression, we have Type (h :: t) = Type (t) = Th List, so Th = T1 and Tt = T1 List
* For x h :: f10 x t expression, we have

Type ( x h :: f10 x t ) = Type (f10 x t ) = Type ( x h ) List, so Type ( x h ) = T2

* For expression x h , we have Type ( x ) = Type (h) 🡪 Type (x h) = T1 🡪 T2
* The rest of the type checking results in no type mismatch, so
* Type ( x ) = T1 🡪 T2
* Type ( f10 ) = (T1 🡪 T2) 🡪 T1 List 🡪 T2 List

1. What does f10 f1 [1;2.3.4.5] do? f1 is the function f1 defined above

**Answer**

The function f10 applies a function x, where Type ( x ) = T1 🡪 T2 to every element of a list of type T1 List to obtain a new list of type T2 List.

In this particular case,

1. What is the type of x in:

let rec f11 x l = match (x,l) with

([],\_) -> []

| (\_,[]) -> []

| (h::t,\_) -> List.append (f10 h l) (f11 t l);;

**Answer**

* For the expression f10 h l, Type ( f10 ) = Th 🡪 Tl 🡪 T1, but we know that Type ( f10 ) = (T1 🡪 T2) 🡪 T1 List 🡪 T2 List, so Th = (T1 🡪 T2) and Tl = T1 List
* For the match expression, we have Type ( (x,l) ) = Type ( (h::t,\_) ) , so Tx = Type (h::t)
* For expression h::t, we have Type(h::t) = Type(t) = Th List, so Tx = Th List = (T1 🡪 T2) List
* The rest of the type constraints are consistent and there is no type mismatch

1. What does f11 [ (fun n->n+1) ; (fun n -> n+2) ] [1;2;3] do?

**Answer**

The function f11 takes as argument two lists. One list x is a list of functions and one list l is a list of arguments. The function f11 applies each function in the list x to every element in the list l and concatenate the resulting lists.

In this particular case, the list of functions are two functions:

* (fun n->n+1) takes an integer argument and increments it by 1
* (fun n -> n+2) takes an integer argument and increments it by 2

So, f11 [ (fun n->n+1) ; (fun n -> n+2) ] [1;2;3] will concatenate two lists

* [2;3;4] obtained by applying (fun n->n+1) to every element in [1;2;3] and
* [3;4;5] obtained by applying (fun n->n+2) to every element in [1;2;3]

The result is [2;3;4;3;4;5]