**CSE340 SPRING 2021 HOMEWORK 5**

**Due Monday Friday April 23 by 11:59 PM**

1. The homework has 3 problems
2. Your answers must be typed.
3. On Gradescope, you should submit the answers to separate question separately.

**Problem 1 (Static and Dynamic Scoping)**

Consider the following program written in a C-like syntax (well not quite C-like but should not be hard to understand). Assume parameters are passed by value.

a, b, x : int; // global variables

int g(a : int; d: int)

{

print **a**; print **b**; print **x**; print **d**;

return a + b+ x + d;

}

int f(x : int)

{

b : int;

b = 3;

b = g(a,b);

{ int x;

int b;

b = 4;

x = 8;

g(a,b);

}

return b;

}

void main()

{

int a;

int b;

a = 4;

b = 5;

a = f(b);

g(b,a);

}

1. What is the output of this program if static scoping is used

a\_global = 0

b\_global = 0

x\_global = 0

a\_main = 4

b\_main = 5

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

f(5): x\_f = 5

b\_f = 3

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

g(0,3): a\_g = 0

d\_g = 3

print a\_g 0 printed

print b\_global 0 printed

print x\_global 0 printed

print d\_g 3 printed

return a\_g + b\_global + x\_global + d\_g = 0 + 0 + 0 + 3 = 3

b\_f = 3

Scope S: x\_S = 8 // local scope

b\_S = 4

g(a\_global, b\_S) = g(0, 4)

g(0,4): a\_g = 0

d\_g = 4

print a\_g 0 printed

print b\_global 0 printed

print x\_global 0 printed

print d\_g 4 printed

return a\_g + b\_global + x\_global + d\_g = 0 + 0 + 0 + 4 = 4

return b\_f

a\_main = 3

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

g(5,3): a\_g = 5

d\_g = 3

print a\_g 5 printed

print b\_global 0 printed

print x\_global 0 printed

print d\_g 3 printed

return a\_g + b\_global + x\_global + d\_g = 5 + 0 + 0 + 3 = 8

2. What is the output of this program if dynamic scoping is used

a\_global = 0

b\_global = 0

x\_global = 0

a\_main = 4 => 15

b\_main = 5

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

f(5): x\_f = 5

b\_f = 3 => 15

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

g(4,3): a\_g = 4

d\_g = 3

print a\_g 4 printed

print b\_f 3 printed

print x\_f 5 printed

print d\_g 3 printed

return a\_g + b\_f + x\_f + d\_g = 4 + 3 + 5 + 3 = 15 // 15 returned

b\_f = 15

Scope S: x\_S = 8 // local scope

b\_S = 4

g(a\_main, b\_S) = g(4, 4)

g(4,4): a\_g = 4

d\_g = 4

print a\_g 4 printed

print b\_S 4 printed

print x\_S 8 printed

print d\_g 4 printed

return a\_g + b\_f + x\_f + d\_g = 4 + 4 + 8 + 4 = 20

// 20 returned

return b\_f = 15

a\_main = 15

g(b\_main, a\_main) = g(5, 15)

g(5,15): a\_g = 5

d\_g = 15

print a\_g 5 printed

print b\_main 5 printed

print x\_global 0 printed

print d\_g 15 printed

return a\_g + b\_f + x\_f + d\_g = 5 + 4 + 8 + 15 = 20

**Problem 2 (Structural, Name and Internal Name Equivalence)**

Consider the following type declarations

**TYPE**

T0 = int;

T1 = real;

T2  = pointer to int;

T3 = pointer to real;

T4  = pointer to T0;

T5  = pointer to T1;

T6  = struct {

a: int;

};

T7 = pointer to T6;

T8 = struct {

a: pointer to T8;

b: int;

};

T9 = struct {

a: T0;

b: pointer to T10;

};

T10 = struct {

a: int;

b: pointer to T9;

};

T11 = struct {

a: int;

b: pointer to T11;

};

T12 = struct {

x: T9 \* T10 -> T11; // function of T9 and T10 that returns T11

y: T11 \* T12 -> T8;

};

T13 = struct {

x: T10 \* T9 -> T11;

y: T12 \* T11 -> T8;

};

T14 = array [4][5] of T9;

T15 = array [4][5] of T10;

T16 = array [5][4] of T9;

For each of the following types, list the types that are equivalent to them, assuming structural equivalence:

1. T0 : None
2. T2 : T4
3. T4 : T2
4. T6 : None
5. T8 : None
6. T10 : T9 and T11
7. T12 : T13
8. T14 : T15
9. T16 : None

Consider the following variable declarations (this is a continuation of the previous declarations

**VARS**

u : T0;

v : int;

x : pointer to T10;  
 y : pointer to T10;  
 z : T3;  
 p, q : T9 -> T9; // function of T9 that returns T9

r : T9 -> T10;

s : T9;

t : T10;

For each of the following assignment statements write if the statement is valid under structural equivalence (SE), internal name equivalence (IE), or name equivalence (NE). Write all that apply.

1. x = y
   * **Answer**: SE
2. y = z
   * **Answer**: // nothing applies
3. p = q
   * **Answer**: SE, IE // part of same declaration but no type name
4. u = p(s).a;
   * s has type T9
   * p expects argument of type T9
   * so the call p(s) is valid under NE
   * p(s) returns type T9
   * p(s).a has type T0
   * u has type T0
   * u = p(s).a is valid under NE because both sides have type T0
   * Conclusion: the whole assignment is valid under NE
   * **Answer**: SE, IE, NE
5. v = p(t).a;
   * t has type T10
   * p expects argument of type T9
   * T10 and T9 are structurally equivalent
   * So the call p(t) is valid under SE
   * p(t) returns a value of type T9
   * p(t).a has type T0
   * v has type int
   * the assignment v = p(t).a is valid under SE because LHS is int and RHS is T10 and int is structurally equivalent to T0
   * Conclusion: the whole assignment is valid under SE // all parts are valid under SE
   * **Answer**: SE
6. z = p(s)
   * s has type T9
   * p expects argument of type T9
   * so the call p(s) is valid under NE
   * p(s) returns type T9
   * z has type T3
   * T3 and T9 are not SE, not IE, and not NE
   * **Answer**: // nothing applies
7. x = p(t).b
   * we have seen above that p(t) is valid under SE
   * p(t).b has type pointer to T10
   * x has type pointer to T10
   * the assignment x = p(t).b is valid under SE
   * Conclusion: the whole assignment is valid under SE because all parts are valid under structural equivalence
   * **Answer**: SE
8. y = r(t).b
   * r expects a value of type T9
   * t has type T10
   * T9 and T10 are structurally equivalent
   * so the call r(t) is valid under structural equivalence
   * r(t) returns a value of type T10
   * so r(t).b has type pointer to T9
   * y has type pointer to T10
   * so the assignment of the return value to y is valid under structural equivalence
   * Conclusion: the whole assignment is valid under SE because all parts are valid under structural equivalence
   * **Answer**: SE

For example, the answer for 1, would be the following (no need for an explanation)

1. SE

Remember that if an assignment is valid under name equivalence, it is also valid under internal name equivalence.

**Problem 3 (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. 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 of the following determine the type of the function

1. let f1 x = 1+x ;;

let (0)

f1 x

+ (1)

1 x

visiting (1): Tx = T(1) = type of 1 = int

visiting (0): Tf1 = T(1) -> Tx = int -> int

**Answer**: Tx = int

Tf1 = int -> int

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

let (0)

+. (1)

x a

f2 a x

visiting (1): Tx = Ta  = float

visiting (0): Tf2 = float -> float -> float

**Answer** Tf2 = float -> float -> float

1. let f3 x y i = i.(x.(y)) ;;

.() (1)

let (0)

f3 x y i

i .() (2)

x y

visiting (2): Tx = T(2) array Ty  = int

visiting (1): Ti = T(1) array T(2) = int

So, Tx = int array

visiting (0): Tf3 = Tx -> Ty -> Ti -> T(1)

**Answer** Ty = int

Tx = int array

Ti = T array

Tf3 = int array -> int -> T array -> T where T is unconstrained

1. Explain why the following declaration results in a type checking error:

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

let (0)

f4 x i +. (1)

.() (2) 1.0

**Answer**

x i

x .() (3)

visiting (2): Tx = T(2) array T(3) = int

visiting (3): Tx = T(3) array = int array

Ti = int

so T(2) = int

visiting (1): T(2) = type of 1.0 = T(1) = float

but T(2) = int, so there is type mismatch

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

y i

x .() (3)

let (0)

f5 x y i + (1)

apply (2) .() (4)

y i

**Answer**

visiting (1): T(2) = T(4) = T(1) = int

visiting (4): Ty = T(4) array = int array

Ti = int

visiting (3): Ty = T(3) array

Ti = int

but Ty = int array, so T(3) = int

visiting (2): Tx = T(3) -> T(2) = int -> int

visiting (0): Tf5 = Tx -> Ty -> Ti -> T(1) , so

Tf5 = (int -> int) -> int array -> int -> int

**Summary**:

Ti = int

Ty  = int array

Tx = int -> int

Tf5 = (int -> int) -> int array -> int -> int

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

y apply (4)

x y

apply (2) apply (3)

x y

f6 x y + (1)

let (0)

visiting (1): T(2) = T(3) = T(1) = int

visiting (2): Tx = Ty -> T(2)

so, Tx = Ty -> int

visiting (2): Tx = Ty -> T(4)

but, Tx = Ty -> int, so T(4) = int

visiting (3): Ty = T(4) -> T(3)

so Ty = int -> int

Tx = Ty -> int = (int -> int) -> int

visiting (0): Tf6 = Tx -> Ty -> T(1) , so

Tf6 = ((int -> int) -> int) -> (int -> int) -> int

**Summary**

Tx = ((int -> int) -> int

Ty  = int -> int

Tf6 = ((int -> int) -> int) -> (int -> int) -> int

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

let (0)

f7 a b c if (1)

b apply (2) apply (3)

a b a c

visiting (1): Tb = bool

T(1) = T(2) = T(3) = T

visiting (2): Ta = Tb -> T(2)

so, Ta = bool -> T

visiting (3): Ta = Tc -> T(3)

so Ta = Tc -> T

but, Ta = bool -> T, so Tc = bool

visiting (0): Tf7 = Ta -> Tb -> Tc ->T(1) , so

Tf7 = (bool -> T) -> bool -> bool -> T

**Summary**

Ta = bool -> T

Tb = bool

Tc  = bool

Tf7 = (bool -> T) -> bool -> bool -> T where T is unconstrained

1. let rec f8 l1 l2 = match (l1,l2) with

([],\_) -> l2

| (\_,[]) -> l1

| (h1::t1,h2::t2) -> h1:: h2:: f8 t1 t2 ;;

No derivation provided. Look at examples from the typed notes for derivations with examples with lists.

**Answer**

Tl1 = T list

Tl2 = T list

Tf8 = T list -> T list -> T list

1. what does f8 calculate? You can try different examples using the OCaml command line

Function f8 takes two lists l1 and l2 as input. f8 then creates a new list with alternating elements from l1 and l2. When either of the list is exhausted, the remainder of the other list is used.