**CSE340 FALL 2020 HOMEWORK 5**

**Due Monday 30 November 2020 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);

return b;

}

void main()

{

int a;

int b;

a = 4;

b = 5;

a = f(b);

g(b,a);

}

**Solution**

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

I am going to assume that global variables are initialized to 0. If you did not assume that in your solution and simply assumed that the initial value is unknown, that is fine.

The execution is equivalent to the following execution

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

return b\_f = 3

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 => 11

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\_global+x\_global+d\_g = 4+3+5+3 = 15

b\_f = 15

return b\_f = 15

a\_main = 15

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

g(5,11): 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\_global+x\_global+d\_g = 5+5+0+15 = 25

**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 = struct {

b: int;

a: pointer to T8;

};

T8 = struct {

a: T0;

b: pointer to T7;

};

T9 = struct {

x: T7 \* T8 -> T7; // function of T7 and T8 that returns T7

y: T11 \* T12 -> T8;

};

T10 = struct {

x: T8 \* T7 -> T8; // function of T7 and T8 that returns T8

y: T12 \* T11 -> T7;

};

T11 = array [4][5] of T9;

T12 = array [4][5] of T10

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

1. **T0:** not equivalent to any other type. The only other “simple” type is T1 which is real and is not equivalent to T0
2. **T2:** can only be equivalent to another pointer. It is equivalent to T4 (pointer to T0) because T0 is structurally equivalent to int.
3. **T4:** equivalent to T2
4. **T6:** is a structure with only one field. There is no other structure with one field, so T6 is not equivalent to any other type in the list.
5. **T8:** T8 can only be equivalent to T7 because it is the only other type whose first field is equivalent to int. Doing the iterative process and assuming T7 and T8 are equivalent, when we compare them, we find int equivalent to T0 and pointer to T7 is equivalent to pointer to T8, so T7 and T8 are equivalent.
6. **T10:** T10 is equivalent to T9. This result and the result for T12 and T11 are obtained through the iterative process.
7. **T12:** is equivalent to T11

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

**VARS**

x : pointer to int;  
 y : pointer to int;  
 z : T3;  
 p, q : T9 -> pointer to int;

r : T9 -> T3;

s : T9;

t : T10;

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

1. x = y is valid under structural equivalence only
2. y = z not valid because pointer to int and pointer to real (T3) are not structurally

equivalent

1. p = q since p and q are part of the same declaration, this is valid under internal name

equivalence and structural equivalence

1. q = r this is not valid because the type of q is a function that returns pointer to int and

the type of r is a function that returns T3 which is pointer to real. Since the

return types are not equivalent under structural equivalence, this assignment is not valid

1. z = p(s) there are two parts to this statement: (1) function call and (2) assigning return

value to z. (1) is valid under name equivalence because s has type T9 and p()

expects an argument of type T9. (2) is not valid because z has type T3 and p() returns pointer to int which is not equivalent to T3.

We conclude that the statement is not valid

1. x = p(t) there are two parts to this statement: (1) function call and (2) assigning return

value to x. (1) is valid under structural equivalence because t has type T9 and p() expects an argument of type T10 which is structurally equivalent to T9. (2) the assignment is valid under structural equivalence because the return type is pointer to int and x has type pointer to int.

We conclude that the assignment is valid under structural equivalence.

1. z = r(t) there are two parts to this statement: (1) function call and (2) assigning return

value to z. (1) is valid under structural equivalence only because t has type T10

and r() expects an argument of type T9 which is structurally equivalent to T10.

(2) is valid under name equivalence because z has type T3 and r() returns a value

of type T3.

We conclude that the assignment is valid under structural equivalence.

1. z = r(s) There are two parts to this statement. (1) function call and (2) assigning return

value to z. (1) valid under name equivalence because s has type T9 and r()

expects an argument of type T9. (2) is also valid under name equivalence

because r() returns a value of type T3 and z has type T3.

We conclude that the whole statement is valid under name equivalence (and

therefore under internal name and structural 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.

let (1)

1. let f1 x = 1 ;;

1

f1 x

visiting node (1) : Tf1 = Tx ➝ int where Tx is unconstrained

1. let f2 x = x +. 1.0 ;;

f2 x

let (1)

(2)

+.

1.0

x

visiting node (1) : Tf2 = Tx ➝ T(2)

visiting node (2) : T(2)  = Tx = T1.0  = float

So : Tf2 = float ➝ float

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

(2)

.()

x

.()

let (1)

f3 x i

(3)

i

x

visiting node (1) : Tf3 = Tx ➝ Ti ➝ T(2)

visiting node (2) : Tx = T(2)  array

T(3)  = int

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

Ti = int

also, we conclude that T(2)  = int because Tx = T(2)  array and Tx = int array

So, Tf3 = int array ➝ int ➝ int

(3)

i

x

(2)

.()

y

.()

let (1)

f4 x y i

1. let f4 x y i = y.(x.(i)) ;;

visiting node (1) : Tf4 = Tx ➝ Ty ➝ Ti ➝ T(2)

visiting node (2) : Ty = T(2)  array

T(3)  = int

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

Ti = int

So, Tf4 = int array ➝ T(2)  array ➝ int ➝ T(2)  where T(2)  is unconstrained

let (1)

1. let f5 x y = x y ;;

(2)

f5 x y

apply

y

x

visiting node (1) : Tf5 = Tx ➝ Ty ➝ T(2)

visiting node (2) : Tx = Ty ➝ T(2)

So, Tf5 = (Ty ➝ T(2))➝ Ty ➝ T(2) for some unconstrained type T(2)

(2)

x

apply

f5 x y

let (1)

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

visiting node (1) : Tf6 = Tx ➝ Ty ➝ T(2)

visiting node (3) : T(3 = Ty = int

(3)

+

visiting node (2) : Tx = T(3) ➝ T(2) = int ➝ T(2)

1

So, Tf6 = (int➝ T(2))➝ int➝ T(2)

y

So, we can write Tf6 = (int➝ T) ➝ int➝ T for some unconstrainted type T

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

let rec f7 l1 l2 = match (l1,l2) with

([],\_) -> l2

| (\_,[]) -> l1

| (h1::t1,h2::t2) -> f6 h1 h2 :: f7 t1 t2 ;;

let

(1)

match

f7 l1 l2

(2)

(8)

(5)

(4)

(3)

::

l1

l2

()

()

()

()

(10)

(9)

::

\_

\_

[]

(7)

(6)

::

[]

l2

l1

apply

apply

t2

h2

h1

t1

t1

t2

f7

h2

h1

f6

Visiting node (1) : T(2) = T(2) = T(4) = T(5)

T(1) = Tl2 = Tl1 = T(8) = T

Visiting node (2) : T(2) = Tl1  \* Tl2 = T \* T

Visiting node (5) : T(5) = T(6)  \* T(7) = T(2) = T \* T, so T(6)  = T(7) = T

Visiting node (6) : T(6)  = Tt1 = Th1 list = T

Visiting node (7) : T(7)  = Tt2 = Th2 list = T

so Th1 list = Th1 list and Th1 = Th2 = Th and Tl1 = Tl2 = Th list

Tt1 = Tt2 = Th list

Visiting nodes (3) and (4) makes no difference

Visiting node (9) : Tf6 = Th1 ➝ Th2 ➝ T(9)  = Th ➝ Th ➝ T(9)

but we know that Tf6 = (int➝ T’)➝ int➝ T’for some unconstrained type T’

(I use T’ because T is used above = Th list)

Th ➝ Th ➝ T(9) ≣ (int➝ T’)➝ int gives us (int➝ T’)≣ int which is not possible

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

([],\_) -> l2

| (\_,[]) -> l1

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

Tf8 = int list ➝ int list ➝ int 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 is equivalent to the following. The shorter of the two lists is extended by appending 0’s to obtain a list that has the same length as the longer list. Then, f8 creates a new list where the i’th element of the result is obtained by adding the i’th element of the extended list to the i’th element of the other list.