CSE340 SPRING 2020
Homework 5
DUE Monday 13 April 2020

PLEASE READ THE FOLLOWING CAREFULLY

- Your answers for all problems must be types
- On Gradescope, you should submit the answers to separate problems separately.

PROBLEM 1

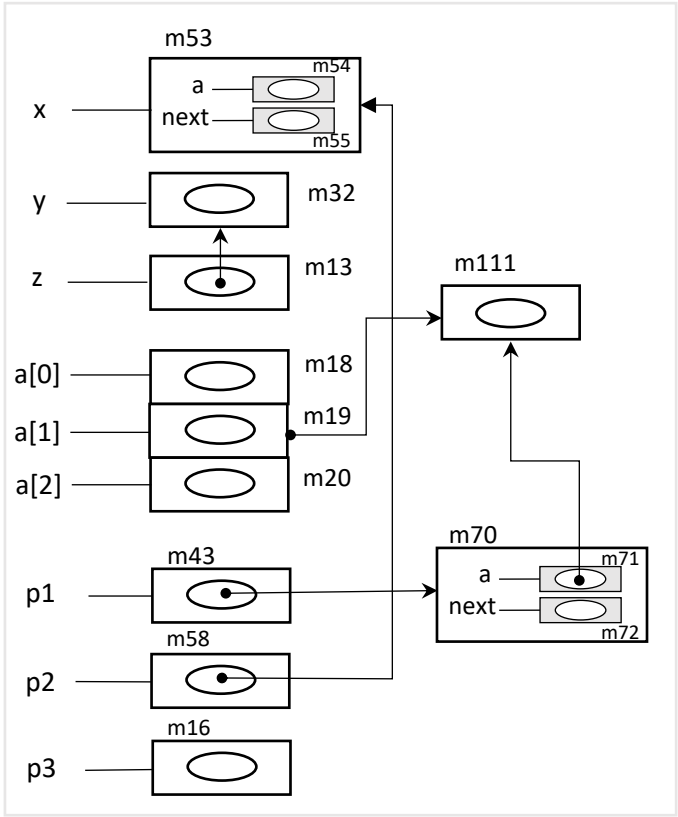
Assume stack memory allocation for nested scopes is used (which means that memory for variables in a scope is allocated on the stack and that it is deallocated when the scope is exited). Consider the following code below and the box-circle diagram to the right which illustrates the situation at **point 1.**

```
struct T {
    int *a;
    struct T* next;
};

int *y;
int **z;
int **w;
struct T   x;
struct T*  p1;
struct T*  p2;
struct T**  p3;

void f()
{ // the following malloc() call allocates m102
  // (which is not shown in the diagram because
  // f() is called after point 1)
  y = (int *) malloc(sizeof(int));
  x.a = y;
}

main()
{  p1 = (struct T*) malloc(sizeof(struct T));
   p2 = &x ;
   { int* a[3];
     a[1] = (int *) malloc(sizeof(int));
     (*p1).a = a[1];
     z = &y ;
     w = &a[1];            // point 1
   }
   f();                    // point 2
   free(y);
   (*p1).next = &x;        // point 3
   // point 4
}
```



Question 1. What is the location associated with *p1 at point 1

m70

Question 2. What is the location associated with *z at point 2?

m32

Question 3. What is the location associated with *(x.a) at point 2?

m102

Question 4. What is the location associated with *((*p1).next) at point 2?

undefined. At point 2, (*p1).next is a wild pointer.

Question 5. What are the dangling references, if any, at point 2?

(*p1).next is not a dangling reference at poitnt 2, it is a wild pointer there is only one dangling reference: w which points to a[1] which has been deallocated.

Question 6. What are the locations that are garbage, if any, at point 3?

Garbage must be locations that have been allocated and that are not accessible by the program even though they have not been deallocated. Let us look at all the allocated memory at point 3

- m102 has been allocated in f() and deallocated with free(y).
- m70 has been allocated and has not been deallocated. It is still accessible as *p1 because p1's value did not change
- m111 has been allocated and has not been deallocated. It is still accessible as *((*p1).a)

Conclusion. No location is garbage at point 3.

Question 7. What are the dangling references, if any, at point 3?

Dangling references at references to deallocated memory. Let is consider all the deallocated memory at point 3

- a[] the memory associated with array a is deallocated after we exit the scope in which a is defined. After the scope is exited, w becomes a dangling reference because it is still pointing to the location associated with a[1]
- m102 is deallocated when free(y) is executed. After the free(y), y becomes a dangling reference because it is still pointing to deallocated memory. Also, x.a is a dangling reference because it points to m102. Also *z which is an alias of y is a dangling reference

Question 8. Assume that the following is executed at point 3 (this applies only to this question):

```
p3 = &p1;
*p3 = &(*((*p1).next));
```

This will results in news arrows, if any, from where to where?

- p3 = &p1 will result in a new arrow from m16 to m43
- *p3 = &(*((*p1).next)).
    - *p3 is an alias of p1
    - &(*((*p1).next)): the value of (*p1).next is &x, so *((*p1).next) is an alias of x and &(*((*p1).next)) is the address of x. So, there will be an arrow from m43 (location associated with p1) to m53 (location associated with x)

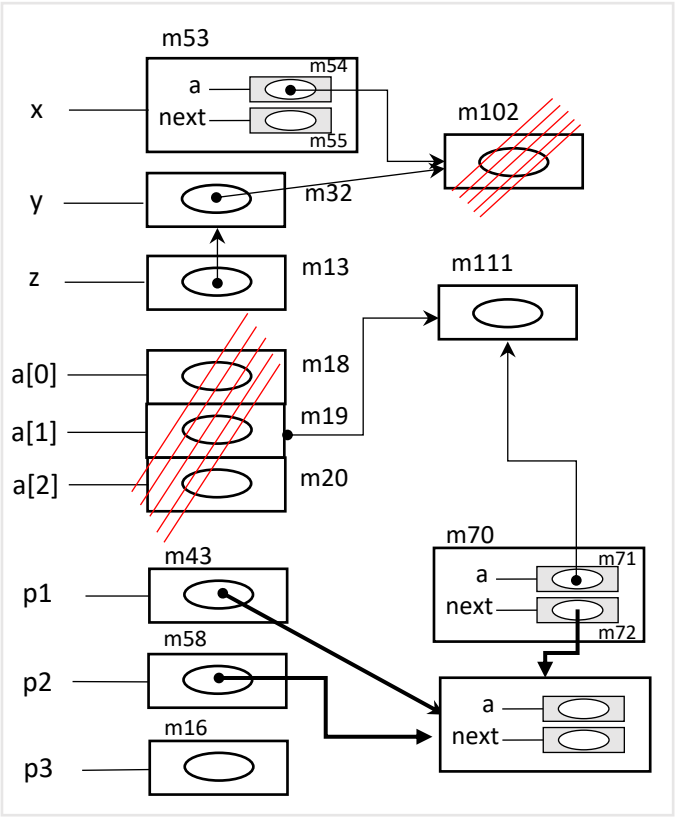
Question 9. Assume the following is executed at point 4:

```
p2 = (struct T*) malloc(sizeof(struct T));
(*p1).next = p2;
p1 = p2;
```

what location become garbage due to the execution of the code?

Let us consider all the locations that have been allocated in the program including those allocated in the code of the question

- m102 is already freed so it is not garbage
- locations associated with array a[] have already been deallocated so they are not garbage
- all locations of variables, such as x,y,z,p1, p2, p3 are not deallocated
- m70 is not longer reachable and not deallocated, so it is garbage
- m111 is not longer reachable and is not deallocated, so it is garbage



Questions 9

Question 10. If we execute free(p2) after the code above, what are the new dangling references, if any, that result from that?

If we execute free(p2), this makes both p1 and p2 dangling references. Note that the arrow from m72 is not a dangling reference because it has no name you can refer to with

Question 11. What is an alias of x at point 1 (the alias should have a variable other than x. Something like *&x does not count)

At point 1, an alias of x is *p2

Question 12. What is an alias of a[0] at point 1. Tthe alias should have a variable other than a.

At point 1, the value of w is the address of a[1]. An alias of a[0] is *(w-1). To understand this, the value of w is the address of the location pointed to by w, which is the location associated with a[1]. w-1 is the value of w – sizeof(int) which is the address of the location associated with a[0]

PROBLEM 2: Lambda Calculus

Question 1. Write a non-recursive lambda expression to compute the n'th Fibonacci number. The Fibonacci numbers are defined as follows

$F_1 = 1$
$F_2 = 1$
$F_n = F_{n-1} + F_{n-2}$ if $n > 1$

Answer

nextf = λp. pair (snd p) (plus (fst p) (snd p))

the function nextf takes a pair that consists of two Fibonacci numbers F_i and F_{i+1} and returns the pair consisting of F_{i+1} and F_{i+2}

The Fibonacci function can be given as

```
initf = pair 0 1
Fib = λn. fst (n nextf init)
```

Question 2. Write a recursive lambda expression to compute the n'th Fibonacci number.

Answer

```
g = λfib.λn. (EQUAL n 1)
                    1
                    (EQUAL n 2)
                           1
                           (plus (fib (prd n)) (fib (prd(prd n))))
```

Fib = fix g

The function requires that the argument n is a Church's numeral

PROBLEM 2: Lambda Calculus

Question 3. Write a non-recursive lambda expression to calculate the sum of the first n squares:

$$1^2 + 2^2 + 3^2 + 4^2 + \ldots + n^2$$

You should not use a closed-form formula for the sum

Answer

```
sq = λn. times n n
inits = pair 0 1
nexts = λp. pair  (plus(fst p) (sq (snd p))) (succ (snd p))
```

The function nexts take a pair of the s and i+1 where s is the sum of squares of the first i integers and returns a pair of s′ and i+2 where s′ is the sum of the squares of the first i+1 integers.

```
sum = λn. fst ( n nexts inits)
```

Question 4. Write a recursive lambda expression to calculate the sum of the first n squares:

$$1^2 + 2^2 + 3^2 + 4^2 + \ldots + n^2$$

You should not use a closed-form formula for the sum

Answer

```
sq = λn. times n n
g = λsm.λn. (EQUAL n 0)
                    0
                    (plus (sq n) (sum (prd n)))
```

Sum = fix g

The function requires that the argument n is a Church's numeral

PROBLEM 3: Type Systems

This problem refers to the type system of the Go Programming Language. You can find the specification at https://golang.org/ref/spec. In particular you should consult the section on types: https://golang.org/ref/spec#Properties_of_types_and_values.

I expect you to consult the specification to answer the questions below.

Question 1. What is the term used in the Go language to refer to type compatibility?

assignability

Question 2. What term is used in the Go language to refer to type equivalence?

identity

Question 3. Can a function type with a variable number of parameter be identical to a function type with a fixed number of parameters?

No because two functions are identical if both have the same fixed number of parameters or both are variadic (have variable number of parameters)

Question 4. If two types are structurally equivalent according to the definition we covered in class, would the two types be identical according to the Go language? Explain or give a counterexample.

Note necessarily. In fact, if we declare

```
type A int
type B int
```

A and B are not identical in the Go language even though they are structurally equivalent in our definition

Question 5. If two types are identical according to the Go language, are the two types structurally equivalent according to the definition we covered in class? Explain or give a counterexample.

Most types that are identical in the Go language, they are structurally equivalent according to our definition. In fact, for types that are not defined (named types), the rules are very similar and even more restrictive than what we had in our definition (for example, the definition requires corresponding structure fields to have the same names and identical types not only identical types). One possible exception is slice type which is basically a variable size array for which there is not corresponding concept in what we did in class. Two slices are identical if their element types are identical. If one think of slices the same way one thinks of pointers, then that makes sense as far as structural equivalence.

PROBLEM 3: Type Systems

The following are examples given in the Go language spec

```
type (
        A0 = []string
        A1 = A0
        A2 = struct{ a, b int }
        A3 = int
        A4 = func(A3, float64) *A0
        A5 = func(x int, _ float64) *[]string
)

type (
        B0 A0
        B1 []string
        B2 struct{ a, b int }
        B3 struct{ a, c int }
        B4 func(int, float64) *B0
        B5 func(x int, y float64) *A1
)

type    C0 = B0
```

For the following questions, you should give explanations that are more specific than what is given in the specification document.

Question 6. Are A2 and B2 in the example above identical? Why?

A2 and B2 are not identical because B2 is a defined type and is different from every other type

Question 7. Are struct {a, b int} and struct {a, c int} identical? Why?

No, because the field b and c which correspond to each other have different names

Question 8. Are struct {a, b int} and B2 in the example above identical? Why?

No, because B2 is a defined type which is different from every other type

Question 9. What is the return type of the function type A5 in the example above?

pointer to a slice of string

Question 10. Explain why A4 and A5 are identical.

A4 and A5 are aliases not defined types. A0 is also an alias for []string. Otherwise, the two types are identical