Comp 524—Spring 2009

Exercise 3

Due at 3:30 PM (on paper), March 17, 2009.

1. (27 pts) Ex. 7.2, p. 398. Use a chart to record your answers, as shown below:

	Structural Equiv	Strict Name Equiv	Loose Name Equiv
A and B	yes	yes	yes
A and C	yes	no	yes
A and D	yes	no	no

2. (17 pts) Ex. 7.8, p.400. Show your work, e.g. show the address offset for each field in the first record. Hint: drawing a diagram is helpful.

s		c		t		d		r								i			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

So 20B used for the first record.

Is it just 200B for the whole array of 10 records?

Not so fast!

We have alignment constraints.

Consider the second record:

s		c		t		d		r								i			
20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39

WRONG! The r field is a real and can only start on an address that is a multiple of 8!

28 is not a multiple of 8

Must add some more padding

4B worth actually

s		c		t		d						r								i			
20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43

Now the second record takes up 24 bytes.

And it ends at address 44

So when we get to the third record we'll try to put its field r at address 52 and the alignment problem repeats.

It will recur every time...

So the actual required storage for the whole array is

20B + 9*24B = 236B

Note that if a char follows the array, that character could start at the very next memory location.

But what if an int or real follows?

We may have to pad the end as well.

Counting that padding, the array uses 240B

You may observe that we can also remedy the problem by leaving padding at the end of each record in the array so that each record starts at an address that meets the most restrictive alignment constraints of any of its fields.

In fact, many languages do exactly that.

In that case, we would regard 24B as the size of a record

The total works out the same: 236B or 240B

3. (15 pts) Write an ML function called `isPalin` to test whether or not a string `s` is a palindrome. Hint: you may find some of the built-in functions useful (http://www.cs.unc.edu/~stotts/723/ML/pre.html).

Here is an example of `isPalin` in use:

```
- isPalin "abcba";
val it = true : bool
```

```
fun isPalin x = explode (x) = rev (explode (x));

(* TYPE:  isPalin = fn : string -> bool *)

(* Want to factor out the explode(x)?  Here is an alternate
version using the construct: *)
```

```
fun isPalin x =
    let
       val chArray = explode(x)
    in
       chArray = rev (chArray)
    end;


(* TYPE:  isPalin = fn : string -> bool *)

(* More involved solutions recursively check for the proper
character matches. *)
```

4. (5 pts) What is the type of the following ML expression?

```
[([1,2],("ab",true)),([],("xy",false))]
```

```
(int list * (string * bool)) list
```

5. (18 pts) Write an ML function called `numDiff` for numerical differentiation. Use the following formula:

$$(f(x + \partial) - f(x - \partial)) / 2\partial$$

Three arguments are needed: `f`, `x`, and `delta`. It should use currying and operate on real numbers.

Here is an example of `numDiff` in use:

```
- numDiff (fn x => x * x * x - x - 1.0) 3.0 1E~6;
val it = 26.0000000036 : real
```

What is the type of the function `numDiff`?

```
fun numDiff f x delta =
     (f(x+delta)-f(x-delta))/(2.0*delta);

(* TYPE: fn : (real -> real) -> real -> real -> real *)

(* The version above uses ML's handy syntactic sugar for
currying.  Here is an equivalent but more verbose version
```

```
without the syntactic sugar: *)

fun numDiff f = fn x => fn delta =>
     (f(x+delta)-f(x-delta))/(2.0*delta);

(* TYPE: fn : (real -> real) -> real -> real -> real *)

(* Note that numDiff(f,x,delta) ... is not equivalent to
the functions above.  When the function is called using
that construction a single three-tuple argument is
interpreted. *)
```

6. (18 pts) Write two ML functions called `iDoubleMap and rDoubleMap` that take a function `f` and a list `l` as curried arguments. `iDoubleMap` should operate on integers and `rDoubleMap` should operate on reals. Return a new list by applying 2*f(x) to every element x in `l`. Hint: use the built-in function `map`.

Here is an example of `iDoubleMap` in use:

```
- iDoubleMap (fn x => x * x) [1, 2, 3];
val it = [2,8,18] : int list
```

What are the types of the functions `iDoubleMap` and `rDoubleMap`?

```
fun iDoubleMap f l = map (fn x => 2 * f(x)) l;

(* TYPE: fn : ('a -> int) -> 'a list -> int list *)

(* It is perfectly reasonable to break things down a bit.
Here is an alternate version using the let construct, which
is useful for that purpose: *)

fun iDoubleMap f l =
    let
       fun doubleF x = 2 * f(x)
    in
       map doubleF l
    end;

(* TYPE: fn : ('a -> int) -> 'a list -> int list *)

(* Or can use the map function twice, though probably less
efficient – think about why. The code: *)
```

```
fun iDoubleMap f l = map (fn x => 2*x) (map f l);

(* TYPE: fn : ('a -> int) -> 'a list -> int list *)
```

```
fun rDoubleMap f l = map (fn x => 2.0 * f(x)) l;

(* TYPE: fn : ('a -> real) -> 'a list -> real list *)

(* Note that the only difference between rDoubleMap and
iDoubleMap solutions are the numeric literals used, 2 vs.
2.0, from which ML can infer the function type. *)
```