

1. [15] Define in Haskell an algebraic type `Weight` to represent weights as floating-point values in either pounds or kilograms. Make `Weight` an instance of the type classes to which it must belong in order for lists of type `[Weight]` to be sorted and printed. Note: One kilogram equals 2.2 pounds.

```
> data Weight = Pound Float | Kilo Float
> deriving (Show)

> instance Eq Weight where
>   w1 == w2 = kilos w1 == kilos w2

> instance Ord Weight where
>   w1 <= w2 = kilos w1 <= kilos w2

> kilos :: Weight -> Float
> kilos (Pound w) = w / 2.2
> kilos (Kilo w) = w
```

2. [10] Show the steps in the evaluation of the following C expression. Assume that operands are evaluated *right-to-left* whenever that's allowed by the definition of C. Assume that `x`'s initial value is 2, and include its final value in your answer.

```
(x++ + ++x) && (++x ? x++ : --x)
(x++ + ++x) && (++x ? x++ : --x) where x = 2
~ if (x++ + ++x) == 0 then 0 else if (++x ? x++ : --x) == 0 then 0 else 1 where x = 2
~ if (x++ + 3) == 0 then 0 else if (++x ? x++ : --x) == 0 then 0 else 1 where x = 3
~ if (3 + 3) == 0 then 0 else if (++x ? x++ : --x) == 0 then 0 else 1 where x = 4
~ if 6 == 0 then 0 else if (++x ? x++ : --x) == 0 then 0 else 1 where x = 4
~ if false then 0 else if (++x ? x++ : --x) == 0 then 0 else 1 where x = 4
~ if (++x ? x++ : --x) == 0 then 0 else 1 where x = 4
~ if (5 ? x++ : --x) == 0 then 0 else 1 where x = 5
~ if x++ == 0 then 0 else 1 where x = 5
~ if 5 == 0 then 0 else 1 where x = 6
~ 1 where x = 6
```

3. [10] Suppose `S` is an array of `char` that implements sets of `Month`, where `Month` is defined (in C or C++) by

```
enum Month { Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec };
```

using the characteristic-vector implementation discussed in class. Assume that

```
sizeof(char) = 1
sizeof(Month) = 1
```

- a. Suppose `S` is defined by

```
char S[n];
```

what is the minimum value of `n`?

The size of the universe is 12, so $n = \lceil 12/8 \rceil = 2$.

- b. Write an expression that has the effect of deleting `Nov` from the set implemented by `S`.

```
S[1] &= ~(1<<5);
```

4. [10] When the following program

```
integer f( integer n )
{
  return n*3;
}
integer a = 4;
integer g( a : integer )
{
  return 4 + f(a);
}
procedure main()
{
  integer f( z : integer )
  {
    return z*10;
  }
  integer a = 2;
  print g(a);
}
```

runs, what is printed, assuming that free names are bound

- a. statically?

10

- b. dynamically?

24

5. [10] When the following program

```
integer x = 4;
procedure f( integer y )
{
  y := y + 10; x := x + 10;
}
procedure main()
{
  f( x ); print x;
}
```

runs, what is printed, assuming that arguments are bound to parameters

- a. by reference?

24

- b. by value-result?

14

6. [10] When the following program

```
integer x = 4;
integer f( integer a, b )
{
  a := a + b; return b;
}
integer g( integer y )
{
  y := y*3; return y+2;
}
procedure main()
{
  print f( x, 2 * g(x) );
}
```

runs, what is printed, assuming that arguments and operands are evaluated left to right, and that arguments are bound to parameters

a. by value?

```
f( x, 2 * g(x) ) where x = 4
~ f( 4, 2 * g(4) )
~ f( 4, 2 * (4*3+2) )
~ f( 4, 2 * 14 )
~ f( 4, 28 )
~ 28
```

b. by name?

```
f( x, 2 * g(x) ) where x = 4
~ x := x + 2 * g(x); return 2 * g(x) where x = 4
~ x := 4 + 2 * g(x); return 2 * g(x) where x = 4
~ x := 4 + 2 * (x := x*3; return x+2); return 2 * g(x) where x = 4
~ x := 4 + 2 * (x := 4*3; return x+2); return 2 * g(x) where x = 4
~ x := 4 + 2 * (x := 12; return x+2); return 2 * g(x) where x = 4
~ x := 4 + 2 * (return x+2); return 2 * g(x) where x = 12
~ x := 4 + 2 * (return 12+2); return 2 * g(x) where x = 12
~ x := 4 + 2 * 14; return 2 * g(x) where x = 12
~ x := 4 + 28; return 2 * g(x) where x = 12
~ x := 4 + 28; return 2 * g(x) where x = 12
~ x := 32; return 2 * g(x) where x = 12
~ return 2 * (x := x*3; return x+2) where x = 32
~ return 2 * (x := 32*3; return x+2) where x = 32
~ return 2 * (x := 96; return x+2) where x = 32
~ return 2 * (return x+2) where x = 96
~ return 2 * (return 96+2) where x = 96
~ return 2 * 98 where x = 96
~ return 196 where x = 96
~ 196 where x = 96
```

7. [10] Consider this C program:

```

void g( int );
void f( int k )
{
    while( k>0 )
    {
        g(k);
        k--;
    }
}
void g( int k )
{
    while( k>0 )
    {
        k--;
        f(k);
    }
}
int main()
{
    f( 2 );
    return 0;
}

```

Fill in the following table to show the successive states of the stack of activation records of f and g. Label each activation record with its function's name and its parameter's value. The first two columns are filled in to get you started. If you run out of space, don't worry— you may stop when you reach the end of the table.

f 2	f 2	f 2	f 2	f 2	f 2	f 2	f 2	f 2	f 2	f 2	f 2	f 2	f 2	f 2		
	g 2	g 2	g 2	g 2	g 2	g 2	g 2	g 2	g 2			g 1	g 1	g 1		
		f 1	f 1	f 1	f 1	f 1		f 0					f 0			
			g 1	g 1	g 1											
				f 0												

8. [10] Redefine the following function as an equivalent tail-recursive function whose body contains no assignment operations. Do not use any auxiliary functions.

```

int f( int x, int k, int n )
{
    while( n > 0 )
    {
        x = k*x; n = n-1;
    }
    return x;
}

```

The effect of the loop is to rebind the parameters x and n to values k*x and n-1, respectively. A straightforward translation obtains the same rebinding by means of a recursive function call:

```

int f( int x, int k, int n )
{
    if( n > 0 )
        return f( k*x, k, n-1 );
    else
        return x;
}

```