

Reading

- Wednesday 6 April: Sethi, Section 5.4
Friday 8 April: Sethi, Sections 5.2, 5.5
Monday 11 April: Sethi, Sections 5.5, 5.6, 12.1, 12.2
Wednesday 13 April: Sethi, Sections 12.3, 12.5, 12.6

Exercises

Due date: Tuesday 12 April. Please leave your paper in the Taylor Hall homework box (it's located in the breezeway between 2.132 and 2.136) **by 4pm**.

- [10] Define a Haskell enumeration `Month` with 12 nullary constructors `Jan`, `Feb`, etc. Use the modifier **deriving** to make `Month` an instance of the six classes to which belongs the type `Day` shown in slide 230. Using Hugs, demonstrate that `Month` belongs to those six classes.
- [20] Define a Haskell algebraic type `Date` which supports three formats:
 - month, day, year
 - day, month, year
 - year, month, dayusing a different constructor for each format. To represent months, use the type `Month` you defined in Exercise 1.
Make `Date` an instance of `Eq` and `Ord`, so that dates can be compared using operators such as `(==)` and `(<)`. Also make it an instance of `Show`. Demonstrate your functions by running them on some examples. Import the library module `List.hs` into your demonstration script by including the directive

```
import List
```

and demonstrate that the `sort` function in that module works on a list of your `Dates`.
- [20] Repeat Exercise 2 in C++, defining `Date` as a `struct` or a `class`, and using C++'s operator-overloading feature. You may use an array of `Dates` in place of Exercise 2's list, and you may either use a library sorting function or write one of your own.
- [10] Consider the following code fragment:

```
var a, b, x: integer;
function f ( x: integer ) : integer;
begin
    return a + x
end;
function g ( a, b: integer) : integer;
begin
    var x : integer = 8;
    return f(a) * f (b);
end
begin
    a := 3;
    b := 4;
    x := 7;
    b := f(12) + g(14, 16);
end
```

What is the value assigned to `b`, assuming that functions' free variables are bound

- statically (i.e., lexically)?
 - dynamically?
- [10] Suppose you didn't know whether, in Haskell, free variables are bound statically or dynamically. Write a small program to determine which it is. The program should indicate Haskell's policy by printing "STATIC" or "DYNAMIC".
 - [10] Repeat Exercise 5 in C.

7. [15] Given the following program fragment:

```
var r: integer;
procedure inc( x: integer ) : integer;
begin
    x := x + 1;
    return x
end inc;
procedure f( a, b : integer );
begin
    r := inc( a ) + inc( b )
end f;
begin
    r := 10; f( r, r )
end
```

What is the final value of *r* for each of the following parameter-binding mechanisms?

- a. call by value
 - b. call by reference
 - c. call by value-result
8. [10] Given the following program fragment:

```
var a, b: integer;
procedure double( y : integer ) : integer
begin
    y := y + y;
    return y;
end
procedure triple( x : integer ) : integer
begin
    return x * x * x;
end
begin
    a := 3;
    b := triple( double(a) );
end
```

What is the final value of *b* for each of the following parameter-binding mechanisms?

- a. call by value
- b. call by name

9. [20] Call-by-name supports a technique known as Jensen's device, which is illustrated by the following program.

```
function sum( int index, low, high, x ) : int
{
    int S = 0;
    index := low;
    while( index <= high )
    {
        S := S + x;
        index++
    };
    return S;
};
function square (int k) : int
{
    return k*k
};
print ( sum(i, 1, 5, square( i )) );
```

- a. [5] Assuming parameters are bound to arguments *by name*, what is printed by the program?
- b. [5] Describe Jensen's device in a few words, and explain how it depends on by-name parameter binding.
- c. [10] Use Jensen's device to write an existential-quantification procedure exists, which mimics the \exists operator of predicate calculus.
10. [10] In the following code,

```
var a, b, x: int;
#define f( x ) (a + (x))
function g ( a, b: int ) : int;
{
    var x : int = 8;
    return f(a) * f(b);
};
procedure main ( )
{
    a := 2;
    b := 4;
    x := 5;
    b := f(3) + g(4,6);
}
```

- a. Show the effect of macro expansion of the three invocations of *f* by showing the resulting program text
- b. Find the values corresponding to the three invocations.
11. [20] Write a function `max(int a[], int n)` in C++ which takes an `int`-array argument containing `n` elements and returns a *reference* to a largest element of the array. Demonstrate it by running a program that uses `max` to increment a largest element of an array of `int`.

CS 345 Richards

Assignment **8**

Authors:

1. _____

2. _____

3. _____

4. _____