
Reading

1. Read Mitchell, Chapter 8.1—8.2.
2. Read Mitchell, Chapter 14.1, 14.4 (up through page 461)

Problems

1. (10 points) Exceptions

Consider the following functions, written in ML:

```
exception Excpt of int;  
fun twice(f,x) = f(f(x)) handle Excpt(x) => x;  
fun pred(x) = if x = 0 then raise Excpt(x) else x-1;  
fun dumb(x) = raise Excpt(x);  
fun smart(x) = (1 + pred(x)) handle Excpt(x) => 1;
```

What is the result of evaluating each of the following expressions?

- (a) `twice(pred,1)`;
- (b) `twice(dumb,1)`;
- (c) `twice(smart,0)`;

In each case, be sure to describe which exception gets raised and where.

2. (10 points) Exceptions in ML

The function `stringToNum` defined below uses two auxiliary functions to convert a string of digits into a non-negative integer.

```
(* Convert one character to a numeric digit. *)  
fun charToNum c = ord c - ord #"0";  
  
fun calcList (nil,n) = n  
  | calcList (fst::rest,n) =  
      calcList(rest,10 * n + charToNum fst);  
  
(* Convert a string of digits to a number. The explode function  
   converts a string to a list of characters. *)  
fun stringToNum s = calcList(explode s, 0);
```

For instance, `stringToNum "3405"` returns the integer 3405. (The function `explode` converts a string into a list of characters, and `ord` returns the ASCII integer value for a character.)

Unfortunately, `calcList` returns a spurious result if the string contains any non-digits. For instance, `stringToNum "3a05"` returns 7905, while `stringToNum " 405"` returns ~15595. This occurs because `charToNum` will convert any character, not just digits. We can attempt to fix this by having `charToNum` raise an exception if it is applied to a non-digit.

- (c) Revise the definition of `charToNum` to raise an exception, and then modify the function `stringToNum` so that it handles the exception, returning `~1` if there is a non-digit in the string. You should make no changes to `calcList`.
- (b) Implement ML functions to provide the same behavior (including returning `~1` if the string includes a non-digit) as in the first part, but without using exceptions. While you may change any function, try to preserve as much of the structure of the original program as possible.
- (c) Which implementation do you prefer? Why?

Please use `turnin` to submit the code for parts (a) and (b), as a single file or in two separate ML files. Also include a printout with your problem set.

3. (10 points) Exceptions and Recursion

Mitchell, Problem 8.4

4. (15 points) Tail Recursion

- (a) The dot product of two vectors $[a_1, \dots, a_n]$ and $[b_1, \dots, b_n]$ is the sum $a_1b_1 + a_2b_2 + \dots + a_nb_n$. For example,

$$[1, 2, 3] \cdot [-1, 5, 3] = 1 \cdot -1 + 2 \cdot 5 + 3 \cdot 3 = 18$$

Implement the function

```
dotprod: int list -> int list -> int
```

to compute the dot product of two vectors represented as lists. You should write this using tail-recursion, so your `dotprod` function will probably just be a wrapper function that calls a second function that does all the work. If passed lists of different length, your function should raise a `DotProd` exception. You will need to declare this type of exception, but you need not catch it.

```
- dotprod [1,2,3] [~1,5,3];
val it = 18 : int
- dotprod [~1,3,9] [0,0,11];
val it = 99 : int
- dotprod [] [];
val it = 0 : int
- dotprod [1,2,3] [4,5];
uncaught exception DotProd
```

- (b) The numbers in the Fibonacci sequence are defined as:

$$\begin{aligned} F(0) &= 0 \\ F(1) &= 1 \\ F(n) &= F(n-1) + F(n-2) \end{aligned}$$

Thus, the sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, etc.

The following defines a function that returns the n -th Fibonacci number.

```
fun slow_fib(0) = 0
  | slow_fib(1) = 1
  | slow_fib(n) = slow_fib(n-1) + slow_fib(n-2);
```

Unfortunately, computing `slow_fib(n)` requires $O(2^n)$ time.

Define a tail recursive function `fast_fib` that can compute $F(n)$ in $O(n)$ time by using tail recursion. (As above, `fast_fib` will most likely be a wrapper that calls a tail-recursive function.) The tail-recursive function should have only one recursive call in its definition.

```

- fast_fib 0
val it = 0 : int
- fast_fib 1
val it = 1 : int
- fast_fib 5
val it = 5 : int
- fast_fib 10
val it = 55 : int

```

(Hint: When converting `sumSquares` to tail-recursive form, we introduced one auxiliary parameter to accumulate the result of the single recursive call in that function. How many auxiliary parameters do you think we will need for `fibtail`?)

Please use `turnin` to submit the code for this problem, and also include a printout with your problem set.

5. (10 points) Race Conditions and Atomicity

The `DoubleCounter` class defined below has methods `incrementBoth` and `getDifference`. Assume that `DoubleCounter` will be used in multi-threaded applications.

```

class DoubleCounter {
    protected int x = 0;
    protected int y = 0;

    public int getDifference() {
        return x - y;
    }

    public void incrementBoth() {
        x++;
        y++;
    }
}

```

There is a potential data race between `incrementBoth` and `getDifference` if `getDifference` is called between the increment of `x` and the increment of `y`. You may assume that `x++` and `y++` execute atomically (although this is not always guaranteed...).

- (a) What are the possible return values of `getDifference` if there are 2 threads?
- (b) What are the possible return values of `getDifference` if there are n threads?
- (c) Data races can be prevented by inserting synchronization primitives. One option is to declare

```

public synchronized int getDifference() {...}
public int incrementBoth() {...}

```

This will prevent two threads from executing method `getDifference` at the same time. Is this enough to ensure that `getDifference` always returns 0? Explain briefly.

- (d) Is the following declaration

```

public int getDifference() {...}
public synchronized int incrementBoth() {...}

```

sufficient to ensure that `getDifference` always returns 0? Explain briefly.

- (e) What are the possible values of `getDifference` if the following declarations are used?

```

public synchronized int getDifference() {...}
public synchronized int incrementBoth() {...}

```

6. (10 points) Bounded Buffers

This question asks about the Java implementation of a bounded buffer given in class. Its code is available on the handouts page for your reference.

- (a) What is the purpose of `while (elementCount == elementData.length) wait()` in `insert`?
- (b) What does `notifyAll()` do in this code?
- (c) Describe one way that the buffer would fail to work properly if all synchronization code is removed from `insert`.
- (d) Suppose a programmer wants to alter this implementation so that one thread can call `insert` at the same time as another calls `delete`. This causes a problem in some situation but not in others. Assume that some locking may be done at entry to `insert` and `delete` to make sure the concurrent-execution test is satisfied. You may also assume that increment or decrement of an integer variable is atomic and that only one call to `insert` and one call to `delete` may be executed at any given time. What test involving `start` and `end` can be used to decide whether `insert` and `delete` can proceed concurrently?
- (e) The changes in the previous part will improve performance of the buffer. Describe one reason that leads to this performance advantage. Despite this win, some programmers may choose to use the original method anyway. Describe one reason why they might make this choice.

7. (30 points) Sieve of Eratosthenes

The ML function, `primesto n`, given below, can be used to calculate the list of all primes less than or equal to `n` by using the classic “Sieve of Eratosthenes”.

```
(*
 * Sieve of Eratosthenes: Remove all multiples of first element from list,
 * then repeat sieving with the tail of the list. If start with list [2..n]
 * then will find all primes from 2 up to and including n.
 *)
fun sieve [] = []
  | sieve (fst::rest) =
    let fun filter p [] = []
        | filter p (h::tail) = if (h mod p) = 0 then filter p tail
                                else h::(filter p tail);
        val nurest = filter fst rest
    in
      fst::(sieve nurest)
    end;

(*
 * Returns list of integers from i to j
 *)
fun fromto i j = if j < i then [] else i::(fromto (i+1) j);

(*
 * Return list of primes from 2 to n
 *)
fun primesto n = sieve(fromto 2 n);
```

Notice that each time through the sieve we first filter all of the multiples of the first element from the tail of the list, and then perform the sieve on the reduced tail. In ML, one must wait until the entire tail has been filtered before you can start the sieve on the resulting list. However, one

could use parallelism to have one process start sieving the result before the entire tail had been completely filtered by the original process.

Here is a good way to think of this concurrent algorithm, which uses the `Java Buffer` class.

The main program should begin by creating a `Buffer` object, with perhaps 5 slots. It should then successively insert the numbers from 2 to n into the `Buffer`, followed by `-1` to signal that there are no more input numbers.

After the creation of the `Buffer` object, but before starting to put the numbers into it, the program should create a `Sieve` object (using the `Sieve` class described below) and pass it the `Buffer` object as a parameter to `Sieve`'s constructor. The `Sieve` object should then begin running in a separate thread while the main program inserts the numbers in the buffer.

After the `Sieve` object has been constructed and the `Buffer` object has been stored in an instance variable, `in`, its `run` method should get the first item from `in`:

- If that number is negative then the `run` method should terminate.
- Otherwise, it should print out the number (using `System.out.println`) and then create a new `Buffer` object, `out`. A new `Sieve` should be created with `Buffer out` and started running in a new thread. Meanwhile the current `Sieve` object should start filtering the elements from the `in` buffer. That is, the `run` method should successively grab numbers from the `in` buffer. If the number is divisible by the first number that was obtained from `in`, it is discarded. Otherwise, it is added to the `out` buffer. This reading and filtering continues until a negative number is read. When the negative number is read, that number is put into the `out` buffer and then the `run` method terminates.

In this way, the program will eventually have created a total of $p+1$ `Sieve` objects (all running in separate threads), where p is the number of primes between 2 and n . The instances of `Sieve` will be working in a pipeline, using the buffers to pass numbers from one `Sieve` object to the next.

Write this program in Java using the `Buffer` class from lecture. Each of the buffers used should be able to hold at most 5 items.

Include a printout of your `Sieve` class with your homework, and use `turnin` to submit it electronically as well. You need not print out or submit the other Java files.

8. (30 points) Five 5's (Bonus Question)

This is a "classic" problem that illustrates many of the strengths of functional programming.

Consider the well-formed arithmetic expressions using the numeral 5. These are expressions formed by taking the integer literal 5, the four arithmetic operators "+", "-", "*", and "/", and properly placed parentheses, such as "5", "5+5", "(5 + 5) * (5 - 5 / 5)", and so on. (Such expressions correspond to binary trees in which the internal nodes are operators and every leaf is a 5.) Write a ML program to answer one or more of the following questions:

- (a) What is the smallest positive integer than cannot be computed by an expression involving exactly five 5's?
- (b) What is the largest prime number that can computed by an expression involving exactly five 5's?
- (c) Exhibit an expression that evaluates to that prime number.

You should start by defining a datatype to represent arithmetic expressions involving 5 and the arithmetic operations, as well as an `eval` function to evaluate them. This question involves only integer arithmetic, so be sure to use `div` for division. You should then write code to generate all expressions containing a fixed number of 5's.

Answering the questions will then involve an exhaustive search (for all numbers that can be computed with a fixed number of 5's), so good programming techniques are important. Avoid

any unnecessarily time-consuming or memory-consuming operations. You may also have to do something special to avoid division by zero inside `eval`, perhaps with exception handlers.

(For those who have taken or are taking 256: while there are many possible ways to solve this problem, it is one very well-suited for dynamic programming...)

Submit your code with `turnin`, and please include a printout with your problem set.