I/O and Exceptions

x

Exercise

Write a function that flips alternate elements of a list.

If $a = [a_1, \ldots, a_n]$, the result should be $a = [a_2, a_1, a_4, , a_3 \ldots, a_n]$. If n is odd, a_n should remain at the end

Solution

```
fun flip [] = []
  | flip [x] = [x]
  | flip (x::y::r) = (y::x::flip(r));
```

Exercise

Write a function that sorts a list of numbers using Quicksort

Hint: Use an auxiliary function to partition with respect to a pivot a

[

Solution

```
fun quick [] = []
    | quick [x] = [x]
    | quick (a::bs) =
        let fun partition (left,right,[]) =
              (quick left) @ (a::quick right)
                | partition (left,right,x::xs) =
                   if x<=a then partition(x::left, right,xs)
                           else partition (left,x::right,xs)
        in partition([],[],bs) end;
```

Mutual recursion

- Function takes list L and produces list of alternate elements of L
  - Function `take` that takes the first element and then calls
  - Function `skip` that drops the first element and then calls `take`
- First attempt

```
fun skip(L) =
   if L = nil then nil
   else take(tl(L));
```

- This doesn't work
- We should use the keyword `and`

Mutual recursion

```
fun take(L) =
   if L = nil then nil
   else hd(L)::skip(tl(L))
and
skip(L) =
   if L = nil then nil
   else take(tl(L));


take[1,2,3,4];
```

Exercise

Using mutual recursion, write a function to test if a number is even

Solution

```
fun even 0 = true
  | even n = odd (n-1)
and odd 0 = false
  | odd n = even (n-1);
```

Exercise

Test if a list of integers alternates between 1 and 2, not ending with a 1.

Solution

```
fun match xs =
let fun s_need_one xs =
case xs of
[] => true
      | 1::xs' => s_need_two xs'
      | _ => false
and s_need_two xs =
case xs of
[] => false
      | 2::xs' => s_need_one xs'
      | _ => false
in
s_need_one xs
end;
```

Printing

- `print(x)` prints a *string*
- What is the type of `print`?

Printing

```
print;
val it = fn: string -> unit

print ("ab");
abval it = (): unit

print ("ab\n");
ab
val it = (): unit


fun testZero(0) = print("zero\n")
  | testZero(_) = print("not zero\n");
val testZero = fn: int -> unit

testZero(2);
not zero
val it = (): unit
```

Printing non-strings

Characters

```
val c = #"a";
val c = #"a": char

str;
val it = fn: char -> string

print (str(c));
aval it = (): unit
```

Other conversions

```
val x = 1.0E50;
val x = 1E50: real

print(Real.toString(x));
1E50val it = (): unit

print(Int.toString(123));
123val it = (): unit

print(Bool.toString(true));
trueval it = (): unit
```

Statements. We can write compound statements like

```
(print(x);
print(y);  )
```

The type of a compound statement is that of the last statement

Exercise

Write a function that prints a list of integers

Solution

```
fun printList(nil) = ()
| printList(x::xs) = (
     print(Int.toString(x));
     print("\n");
     printList(xs)
);
val printList = fn: int list -> unit

printList [1,2,3];
1
2
3
val it = (): unit
```

Exercise

Write a function to compute $\binom{n}{m}$, while printing n, m and the result.

Solution

```
fun factorial 0 = 1
  | factorial n = n * factorial(n-1);

fun comb n m =
   (
   print ("n is ");
   print(Int.toString(n));
   print ("\n");
   print ("m is ");
   print(Int.toString(m));
   print ("\n");
   print ("Result is ");
   print (Int.toString (factorial(n) div (factorial(m) * factorial(n-m))));
   print ("\n")
);

comb 5 2;
n is 5
m is 2
Result is 10
val it = (): unit
```

Exercise

Given n, print 2^n X's

Solution

```
fun makelist 1 = "X"
| makelist n = makelist (n-1) ^ makelist (n-1);
val makelist = fn: int -> string

fun printXs n = print(makelist n);
val printXs = fn: int -> unit

printXs 4;
XXXXXXXXval it = (): unit
```

Input

File

```
cat test

12
ab
```

Open the file

```
val infile = TextIO.openIn ("test");
val infile = ?: TextIO.instream
```

Instreams

```
TextIO.endOfStream (infile);
val it = false: bool

TextIO.inputN (infile,4);
val it = "12\na": string

TextIO.inputN (infile,1);
val it = "b": string

TextIO.inputN (infile,1);
val it = "\n": string

TextIO.endOfStream (infile);
val it = true: bool
```

Reading lines of a file

```
val infile = TextIO.openIn ("test");
val infile = ?: TextIO.instream

TextIO.inputLine (infile);
val it = SOME "12\n": string option

TextIO.inputLine (infile);
val it = SOME "ab\n": string option

TextIO.inputLine (infile);
val it = NONE: string option
```

Reading complete files

```
val infile = TextIO.openIn ("test");
val infile = ?: TextIO.instream

val s = TextIO.input (infile);
val s = "12\nab\n": string
```

Read a single character

`TextIO.input1`

Reads a single character. Type

```
val it = fn: TextIO.instream -> char option
```

Lookahead

```
TextIO.lookahead;
val it = fn: TextIO.instream -> char option
```

Reads the next character, but leaves it in the input stream

Are there n characters left?

```
TextIO.canInput(f,n);
```

Are there at least n characters available on instream f?

Exercise

Write expressions to

1. Open a file `zap` for reading
2. Close the file whose instream is `in1`
3. Read 5 characters from the instream `in2`
4. Read a line of text from the instream `in3`
5. Read the entire file from instream `in4`
6. Find the first character waiting on the `in1`, without consuming it

Solution

1. `val IN = TextIO.openIn("zap")`
2. `TextIO.closeIn(in1);`
3. `TextIO.inputN(in2,5);`
4. `TextIO.inputLine(in3);`
5. `TextIO.input(in4);`
6. `TextIO.lookahead(in1);`

Exercise

Assume that we have a file with the following contents

```
abd
de
f
```

What does each command return, if issued repeatedly

```
val x = input (infile);
val x = input1 (infile);
val x = inputN (infile,2);
val x = inputN (infile,5);
val x = inputLine (infile);
val x = lookahead (infile);
```

Solution

1. First time `abc\nde\nf\n`, Subsequent times, the empty string
2. `a`, `b`, `c`, `\n`, `d`, `e`, `\n`, `f`, then empty string
3. `ab`, `c\n`, `de`, `\nf`, enpty string
4. `abc\nd`, `e\nf` then empty string
5. `abc\n`, `de\n`, `f\n`, then empty string
6. Always `a`

Exercise

What are the types of the following expressions

1. `SOME()`
2. `SOME 123`
3. `SOME NONE`
4. `fun f() = SOME true;`
5. `fun f(NONE) = 0 | f(SOME i) = 1;`

Solution

1. `unit option`
2. `int option`
3. `'a option option`
4. `fn:unit -> bool option`
5. `fn : 'a option -> int`

Exercise

Read a file of characters, treating it as a sequence of words (consecutive, non-white space characters). Return a list of the words in the file

Solution

```
(* test if a character is white space *)
     fun white(" ") = true
     |   white("\t") = true
     |   white("\n") = true
     |   white(_) = false;

     fun getWord(file) = (* read one word *)
     if TextIO.endOfStream(file) then ""
     else
     let
     val c = TextIO.inputN(file,1)
     in
     if white(c) then ""
     else c^getWord(file)
     end;
```

Solution, continued

```
fun getList1(file) = (* read all words from an instream *)
if TextIO.endOfStream(file) then nil
else getWord(file) :: getList1(file);

(* read all words from a file given the file name *)
fun getList(filename) = getList1(TextIO.openIn(filename));
```

Exceptions

```
5 div 0;
Exception- Div raised

hd (nil: int list);
Exception- Empty raised

tl (nil: real list);
Exception- Empty raised

chr (500);
Exception- Chr raised
```

User-defined exceptions

```
exception Foo;
exception Foo

raise Foo;
Exception- Foo raised
```

Another example

```
exception BadN;

exception BadM

fun comb(n,m)=
   if n<0 then raise BadN
   else if m<0 orelse m>n then raise BadM
   else if m=0 orelse m=n then 1
   else comb(n-1,m) + comb (n-1,m-1);
val comb = fn: int * int -> int

comb(5,2);
val it = 10: int

comb(~1,0);
Exception- BadN raised

comb(5,6);
Exception- BadM raised
```

Exceptions with parameters

```
exception Foo of string;

raise Foo ("bar");
Exception- Foo "bar" raised

raise Foo(5);
poly: : error: Type error in function application.


raise Foo;
poly: : error: Exception to be raised must have type exn.
```

Handling exceptions

```
exception OutOfRange of int * int;

fun comb1(n,m)=
   if n <= 0 then raise OutOfRange (n,m)
   else if m<0 orelse m>n then raise OutOfRange (n,m)
   else if m=0 orelse m=n then 1
   else comb1 (n-1,m) + comb1 (n-1,m-1);
val comb1 = fn: int * int -> int
```

Exceptions, continued

```
fun comb (n,m) = comb1 (n,m) handle
   OutOfRange (0,0) => 1 |
   OutOfRange (n,m) => (
          print ("out of range: n=");
          print (Int.toString(n));
          print (" m=");
          print (Int.toString(m));
          print ("\n");
          0
     );
val comb = fn: int * int -> int

comb (4,2);
val it = 6: int

comb (3,4);
out of range: n=3 m=4
val it = 0: int

comb (0,0);
val it = 1: int
```

Exercise

Write a program to return the third element of a list of integers

Solution

```
exception shortList of int list;

fun returnThird1 L =
   if length(L) < 3 then raise shortList (L)
   else hd(tl(tl(L)));
val returnThird1 = fn: int list -> int


fun returnThird L = returnThird1 L handle
    shortList L => (
       print ("List too short\n");
       0
         );
val returnThird = fn: int list -> int


returnThird [1,2,3,4];
val it = 3: int
returnThird [1,2];
List too short
val it = 0: int
```

Exercise

Write a factorial function that produces 1 when its argument is 0, 0 for a negative argument, with an error message

Solution

```
exception Negative of int;

     fun fact1(0) = 1
     |   fact1(n) =
             if n>0 then n*fact1(n-1)
             else raise Negative(n);
val fact1 = fn: int -> int

     fun fact(n) = fact1(n) handle Negative(n) => (
         print("Warning: negative argument ");
         print(Int.toString(n));
         print(" found\n");
         0
     );
val fact = fn: int -> int
```

Solution, continued

```
fact 5;
val it = 120: int

fact 0;
val it = 1: int

fact ~2;
Warning: negative argument ~2 found
val it = 0: int
```