CS109A Notes for Lecture 1/29/96

Exceptions

An *exception* is the only thing that a function can return other than a value of its return-type (range-type).

- Generally indicates an unexpected argument for the function.
- Declare as

```
exception EmptyList;
```

exception EmptyList

A function can *raise* an exception in lieu of returning a value.

Example:

```
fun minmax([x:int]) = (x,x)
|   minmax(nil) = raise EmptyList
|   minmax(x::xs) =
    let
        val (low,high) = minmax(xs);
    in
        if x<low then (x,high)
        else if x>high then (low,x)
        else (low,high)
    end;
```

*val minmax = fn : int list → int * int*

Getting Caught in an Exception

If you cause an exception to be raised, and there is nothing to "handle" it, your program stops with an "uncaught exception."

```
minmax(nil);
```

uncaught exception EmptyList

- Many built-in functions have a corresponding exception of the same name but with capitalized first letter.

```
1 div 0;
```

uncaught exception Div

- See p. 211–212 EMLP for built-in exceptions.

Handling Exceptions

Catch exceptions with an expression of the form

<expression> **handle** <match>

- A *match* is one or more clauses of the form <pattern> `=>` <expression> separated by vertical bars, as in anonymous functions.
  - □ In the context of a "handle" match, the patterns must be exceptions.
- If the expression before `handle` returns:
  - a) A non-exception: return this value.
  - b) An exception: see if it matches one of the patterns and return the corresponding expression.

Example: Function `safeMinmax` calls `minmax`.

- If `minmax` does not raise an exception, `safeMinmax` produces a pair with first component `"OK"` and second component the pair produced by `minmax`.
- If `minmax` raises `EmptyList`, `safeMinmax` produces a tuple with `"Empty List"` as the first component and 2nd component (0,0).
- Note that an exception is of type `exn`, while `"Empty List"` is of type string.

```
fun safeMinmax(L) = ("OK", minmax(L))
    handle EmptyList => ("Empty List", (0,0));
```
*val safeMinmax = fn : int list → string * (int * int)*

```
safeMinmax([0,0,0]);
```
*val it = ("OK", (0, 0)) : string * (int * int)*

```
safeMinmax([]);
```
*val it = ("Empty List", (0, 0)) : string * (int * int)*

Simple Printing

`print(x)` works for elementary types: int, string, boolean, real.

- Must be disambiguated so ML can figure out the type of x.

Example:

```
fun hw() = print("hello world\n");
```
val hw = fn : unit → unit

- Note that the parameter of `hw` is the unit, not an empty tuple.

- The range type of `hw` is also "unit," characteristic of functions like `print` that don't really produce a return value.

```
hw();
```
hello world
val it = () : unit

Statement Lists

- General tool, important to print a sequence of items.

- An expression can be a list of expressions, surrounded by parentheses and separated by semicolons.

  - The value of this expression is the value of the last.

Example:

```
fun foo(i) = (
        print("The value ");
        print(i:int);
        print(" was given\n")
    )
```
val foo = fn : int → unit

```
foo(123);
```
The value 123 was given
```
val it = () : unit
```

- Again, distinguish between the thing printed and the value of the function call; the latter is the unit.

Input

- First, you need to open a file and get a "token" of type `instream` by using function `open_in`.

```
val file = open_in("foo");
```
val file = - : instream

- Next, you may read n characters from the file `foo` by calling the function `input`.

```
val s = input(file, 3);
```
val s = "abc" : string

assuming that the first three characters in file `foo` are `"abc"`.

- Note you refer to the file by its instream token, not the file name.

Example: Here is a function to read a sequence of digits ended by the newline character and return the integer value.

```
fun ri(f, i) =
    let
        val c = input(f, 1);
    in
        if c = "\n" then i
        else ri(f, 10*i+ord(c)-ord("0"))
    end;
```
*val ri = fn : instream * int → int*

- Key tricks:
  - □ Second argument is the value of the integer read so far.
  - □ To incorporate another digit, multiply the value of what was read by 10 and add the difference between the ASCII codes for the digit read and `"0"`.

```
fun readInt(f) = ri(f,0);
```
val readInt = fn : instream → int

- `readInt` starts `ri` off properly, with 0 read so far.

```
readInt(open_in("test"));
```
val it = 1234 : int

- Note that the instream token is hidden; it is returned by `open_in` and passed immediately to `readInt`.