

# General Computer Science I (320101) Fall 2011

## Practice Problem 1: SML Basics

### Problem 1.1 (Head and Tail)

Program the following elementary list functions in SML and test them on three examples each. Please note that your program must work on every input except the empty list.

1. `head` takes a list  $L$  as input and returns the first element of  $L$
2. `tail` takes a list  $L$  as input and returns the list consisting of all elements of  $L$  in original order except the first one.

**Problem 1.2:** Define the `member` relation which checks whether an integer is member of a list of integers. The solution should be a function of type `int * int list -> bool`, which evaluates to `true` on arguments `n` and `l`, iff `n` is an element of the list `l`.

**Problem 1.3:** Define the `subset` relation. Set  $T$  is a subset of  $S$  iff all elements of  $T$  are also elements of  $S$ . The empty set is subset of any set.

---

**Hint:** Use the `member` function from Problem 1.2

---

**Problem 1.4:** Define functions to zip and unzip lists. `zip` will take two lists as input and create pairs of elements, one from each list, as follows: `zip [1,2,3] [0,2,4] ~> [[1,0], [2,2], [3,4]]`. `unzip` is the inverse function, taking one list of tuples as argument and outputting two separate lists. `unzip [[1,4], [2,5], [3,6]] ~> [1,2,3] [4,5,6]`.

**Problem 1.5:** Define a function `mymax` over SML lists of integers. Here, we take the maximum of an empty list to be the “default value” 0. You might want to consider a notation of type `mymax(x : y : ys)` and work with the first two elements of the list in the step case.

### Problem 1.6 (Factorial)

Write a recursive procedure `fact : int -> real`, that computes the factorial function  $(n)! = 1 \cdot 2 \cdot \dots \cdot n$ , where  $(0)! = 1$ . We want the procedure `fact` to diverge for negative arguments (on an abstract interpreter without resource limitations).

What is the largest number  $n$  you can compute  $(n)!$  for on your system?

**Problem 1.7:** Write three SML functions `nth`, `take`, `drop` that take a list and an integer as arguments, such that

1. `nth(xs, n)` gives the  $n$ -th element of the list `xs`.
2. `take(xs, n)` returns the list of the first  $n$  elements of the list `xs`.
3. `drop(xs, n)` returns the list that is obtained from `xs` by deleting the first  $n$  elements.

In all cases, the functions should raise the exception `Subscript`, if  $n$  is negative or the list `xs` has less than  $n$  elements. We assume that list elements are numbered beginning with 0.

**Problem 1.8:** Write three SML functions `rev`, `take`, `drop` operating on lists (`lst`) and integers (`n`), such that

1. `rev(lst)` returns the list `lst` in reversed order.
2. `last(lst,n)` returns the list of the last `n` elements of the list `lst`.

Make only use of the `::` and `@` built-in operators or those functions you have defined yourself. Note that `n` is always a positive integer. Assume that list elements are numbered beginning with 0.

**Problem 1.9:** Write an SML function that tabulates lists, i.e `tabulate(f,n)` evaluates to the list `[f(0),\ldots,f(n-1)]`. As an example if `f(x)=2*x` and `n=[0,1,2]` then `tabulate(f,n)=[0,2,4]`

**Problem 1.10:** Declare a (abstract) data type for natural numbers and one for lists of natural numbers in SML. Write an SML function that given two natural number `n` and `m` (as a constructor term) creates the list `[n,n+1,...,m-1,m]` if `n ≤ m` and raises an exception otherwise.

**Problem 1.11:** Using the abstract data type of truth functions, give the defining equations for a function `myif` that takes three arguments, such that `myif(X,Y,Z)` behaves like “if `X` then `Y`, else `Z`”.

---

**Hint:** There is a control structure `if...then...else` in SML, of course you are not supposed to use that.

---

**Problem 1.12:** Write three variants of the `member` function in SML, where `member(x,xs)` returns `true`, iff `x` is an element in `xs`.

1. the first variant should not use another function.
2. the second variant should be non-recursive, using the function `myexists` (write that as well) that takes a property `p` and a list `l` as arguments and returns `true`, iff there is an element `a` in `l` such that `p(a)` evaluates to `true`.
3. the third variant should be non-recursive using the `foldl` function.

**Problem 1.13:** Generalize the `mymax` function from Problem 1.5 to general lists with an ordering function (i.e. a function of type `('a * 'a) ->bool`) and a default value of type `'a`, which are passed as arguments. Given a list `l` an ordering relation `ord`, and a default value `d`, calling `gmax(l,ord,d)` evaluates to the `ord`-maximal element of `l`, or `d`, if `l` is empty.

Test this on lists of digits with the ordering relation given by `1 < 3 < 2 < 5 < 7 < 8 < 4 < 9 < 6`.

**Problem 1.14 (Infinite Precision Arithmetics)**

We represent natural numbers of arbitrary length by non-empty lists of digits. Write SML functions for the arithmetical operations of addition, multiplication, integer division and modulo.

**Problem 1.15 (Sorting a list)**

Write a function that takes a list of strings and sorts it in ascending order like in dictionary.

**Problem 1.16:** Write a function `split` that takes a string (a sentence) and returns a list of all pairs of strings one can get by splitting the sentence between any two words. A word is defined as a sequence of symbols between two spaces or a space and nothing. For example,

```
split "We_really_love_SML!";  
val it = [("We", "really_love_SML!"), ("We_really", "love_SML!"),  
("We_really_love", "SML!")]
```

**Problem 1.17 (Ordering Function)**

Write an SML function that takes an `int list` and sorts it in ascending order. For example:

```
sort [7,4,1,3];  
val it = [1,3,4,7];
```

**Problem 1.18:** Write a recursive higher-order SML function `mapcan` that maps a list-valued function  $f$  over a list and appends all the result lists to a single list. What is the SML type of this function (explain).()

Write versions of `map` and `mapcan` that map a binary function over two lists. What are the SML types of these functions (explain).()

**Problem 1.19:** Write a non-recursive variant of the `member` function from Problem 1.2 using the `foldl` function.

**Problem 1.20 (List functions via `foldl`/`foldr`)**

Write the following procedures using `foldl` or `foldr`

1. `length` which computes the length of a list
2. `concat`, which gets a list of lists and concatenates them to a list.
3. `map`, which maps a function over a list
4. `myfilter`, `myexists`, and `myforall` from ??

**Problem 1.21 (Understanding `map`)**

Given the SML higher-order function `fun f x = fn (y) => y::x` and the list `val l = [1,2,3]`.

- Determine the types of `f` and `map (f l)`
- evaluate the expression `map (f l) l`

**Problem 1.22:** Write a function `napply` that takes a function `f:int->int` and an integer `n`, and returns the result of  $n$  applications of  $f$  to itself, starting with number `n` as its argument. What does `napply(fn x=>x+1, n)` numerically compute? Is there any `n` for which `napply(fn x=>x div 2, n)` returns a non-zero value?

**Problem 1.23 (Nary Multiplication)**

By defining a new datatype for  $n$ -tuples of unary natural numbers, implement an  $n$ -ary

multiplications using the function `mul` from Problem 1.25. For  $n = 1$ , an  $n$ -tuple should be constructed by using a constructor named `first`; for  $n > 1$ , further elements should be prepended to the first by using a constructor named `next`. The multiplication function `nmul` should return the product of all elements of a given tuple.

For example,

```
nmul(next(s(s(zero)),
        next(s(s(zero))),
        first(s(s(s(zero))))))
```

should output `s(s(s(s(s(s(s(s(s(s(s(s(zero))))))))))` since  $223 = 12$ .

**Problem 1.24:** Declare a data type `myNat` for unary natural numbers and `NatList` for lists of natural numbers in SML syntax, and define a function that computes the length of a list (as a unary natural number in `mynat`). Furthermore, define a function `nms` that takes two unary natural numbers `n` and `m` and generates a list of length `n` which contains only `ms`, i.e. `nms(s(s(zero)),s(zero))` evaluates to `construct(s(zero),construct(s(zero),elist))`.

### Problem 1.25 (Unary natural numbers)

Define a datatype `nat` of unary natural numbers and implement the functions

- `add = fn : nat * nat -> nat` (adds two numbers)
- `mul = fn : nat * nat -> nat` (multiplies two numbers)

### Problem 1.26 (Binary and Decimal Addition)

Define binary and decimal addition, and multiplication on lists of digits.

**Hint:** This is just a sneaky way of getting you to practice with lists.

For the decimal addition function, we represent natural numbers as lists of digits, e.g. `[5,3,4]` for the number 534. Now the function `badd` works as follows: `badd([2,8],[1,3])` evaluates to `[4,1]`.

The binary addition function is similar, only have it operates on lists of binary digits, i.e. the numbers 1 and 0.

**Problem 1.27:** Program the function  $f$  with  $f(x) = x^2$  on unary natural numbers without using the multiplication function.

### Problem 1.28 (Floating Point Powers)

Write a recursive procedure `power : real * int -> real`, that computes the power  $x^n$  for a real number  $x$  and a natural number  $n$  by floating point operations. What is the result of `power(3.0,100)`, is this really the number  $3^{100}$  (discuss)?

### Problem 1.29 (Your own lists)

Define a data type `mylist` of lists of integers with constructors `mycons` and `mynil`. Write translators `tosml` and `tomy` to and from SML lists, respectively.

**Problem 1.30:** Write a function that takes an odd positive integer and returns a `char list list` which represents a triangle of stars with  $n$  stars in the last row. For example,

```

triangle 5;
val it =
["□", "□", "*", "□", "□"],
["□", "*", "*", "*", "□"],
["*", "*", "*", "*", "*"]

```

### Problem 1.31 (Generating Function)

Write an SML function of type `combine: int -> int list list` that takes an integer  $n$  and returns a list of  $2^n$  lists of length  $n$  which represent all combinations of ones and zeros (as in a truth table).

For example:

```

combine 2;
val it = [[0,0],[0,1],[1,0],[1,1]] : int list list

```

### Problem 1.32 (Permutations)

1. Implement a function with the following type

```
intersperse: 'a -> 'a list -> 'a list list
```

The function takes a List `xs`, an element `y` and calculates all available lists, whereas `y` has been inserted into `xs` at an arbitrary position. Example:

```
intersperse 1 [2,3] = [[1,2,3],[2,1,3],[2,3,1]]
```

2. Implement a function with the following type

```
permutations: 'a list -> 'a list list
```

that gives you all permutations of a list. Example:

```
permutations [1,2,3] = [[1,2,3],[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]]
```

Whether or not you successfully implemented the function `intersperse`, you may use it here.