(Weekly exercises in INF3110/4110 week 37 8-13.09.2008 *)*

(ML-programming *)*

(**Exercise 1:***

Characters have type "char" and constants have the form #c, where c is a character. The functions "ord" and "chr" convert between characters and character codes. In most implementations if 0 <= k <= 255, then char(k) returns the character with code k. Conversely, ord(c) returns the (ASCII) integer code of character c. The following function convert the number 5 into the character '5':

- fun digit i = chr(i + ord #"0");
val digit = fn : int -> char

Try the following:

- ord (5);
- ord #"0";
- digit 5;
- digit 7;
**)*

```
fun digit i = chr(i + ord #"0");

=> val digit = fn : int -> char


ord (5);

=> stdIn:11.1-11.8 Error: operator and operand don't agree [literal]
    operator domain: char
    operand:         int
    in expression:
      ord 5


ord #"0";

=> val it = 48 : int


digit 5;

=> val it = #"5" : char


digit 7;

=> val it = #"7" : char
```

```
(*
The functions "str" and "String.sub" convert between characters and strings. If s is a string then String.sub(s, n) returns
the nth character in s, counting from zero. We can define the digit function in a different way:

- fun digit i = String.sub("0123456789", i);
val digit = fn : int -> char

Try the following:

- str (digit 5);
- digit 5;
*)


fun digit i = String.sub("0123456789", i);

=> val digit = fn : int -> char


str (digit 5);

=> val it = "5" : string


digit 5;

=> val it = #"5" : char




(*
For each version of "digit", try to guess what would be the answer when called with the following parameters (try next
in the computer to check your guess):

- digit ~1;
- digit 10;

Play with other combinations of the functions in order to understand them better.
*)


(* The first version returns #":" and #"/", which are the characters just outside the range of digits in the ASCII
character set: *)

digit 10;

=> val it = #":" : char


digit ~1;

=> val it = #"/" : char
```

```
(* The second version reports an error, by raising exception Subscript: *)

digit ~1;

=> uncaught exception Subscript [subscript out of bounds]
   raised at: stdIn:14.6-15.7


digit 10;

=> uncaught exception Subscript [subscript out of bounds]
   raised at: stdIn:14.6-15.7
```

(* ***Exercise 2:***

Write the function

fun abs (v) = ... ;

that finds the absolute value of v (i.e. -v if v is negative).

Example:

- abs 17;

val it = 17 : int

- abs ~19;

val it = 19 : int

)

```
fun abs v = if v<0 then ~v else v;

abs 17;
abs ~19;
```

(* ***Exercise 3:***

Second degree polynomials $ax^2+bx+c=0$ have solutions

(-b+Math.sqrt(b^2-4ac))/2a and (-b-Math.sqrt(b^2-4ac))/2a

where «Math.sqrt» is the square root function in ML.

Write the function

fun annengrad (a, b, c) = ... ;

that returns both solutions.

Avoid computing the square root more than once.

)

```
fun annengrad (a, b, c) =
    let val d = Math.sqrt(b*b - 4.0*a*c)
    in
      ((~b+d)/(2.0*a), (~b-d)/(2.0*a))
    end;

annengrad(1.0, 2.0, ~3.0);
annengrad(1.0, 0.0, ~49.0);
```

```
(* Exercise 4:
  Write the function
        fun power (a, b) = ... ;
  that computes the exponentiation a^b (i.e. a multiplied by itself b times).
    We may assume that a and b are integers and that b>=0.
    Example:
        - power(2, 23);
        val it = 8388608 : int
        - power(~25, 2);
        val it = 625 : int
*)

fun power (_,0) = 1
  | power (a,b) = a*power(a,b-1);

power(2, 23);
power(~25, 2);




(* Exercise 5:
   (Note that "teller" and "nevner" is Norwegian for numerator and denominator.)

   A fraction can be represented by the record
        {teller=..., nevner=...}
   Write the function
        fun add ({teller=t1,nevner=n1}, {teller=t2,nevner=n2}) = ... ;
   that evaluates the sum of the two fractions.
   (Remember the rules for fraction additions:
        a/b + c/d = (a*d + c*b)/(b*d)
   .)
   Example:
        - add({teller=1,nevner=2}, {teller=3,nevner=4});
        val it = {nevner=8,teller=10} : {nevner:int, teller:int}
*)

fun add ({teller=t1,nevner=n1},
         {teller=t2,nevner=n2}) = {teller=(t1*n2 + t2*n1), nevner=(n1*n2)};

add({teller=1,nevner=2},
    {teller=3,nevner=4});
```

```
(* Exercise 6:
   a) Write the function
         fun max2 (a, b) = ... ;
      that returns the largest of the two integers a and b
      Example:
         - max2(3, ~17);
         val it = 3 : int

   b) Write the function
         fun max3 (a, b, c) = ... ;
      that returns the largest of the three integers a, b and c.
      Example:
         - max3(4, ~5, 19);
         val it = 19 : int

   c) Write the function
         fun max ... = ... ;
      that returns the largest integer in the list of integers given as a parameter.
      Example:
         - max [7];
         val it = 7 : int
         - max [5, ~4, ~17, 28, 13];
         val it = 28 : int
*)

fun max2 (a, b) = if a > b then a else b;
fun max3 (a, b, c) = max2(a, max2(b,c));
fun max nil = 0 (* this branch should never be taken it is added to avoid the compiler warning about nonexhaustiveness *)
  | max [n] = n
  | max (n::ns) = max2(n,max(ns));


max2(3, ~17);
max3(4, ~5, 19);
max [7];
max [~7];
max [5, ~4, ~17, 28, 13];
```

```
(* Exercise 7:
  a) Write the function
        fun first ... = ... ;
    that returns the first element in the string list given as parameter.
    (There exists a standard operator "hd" which does this but do not use it.)
    Example:
        - val tv = ["Det", "bodde", "en", "underlig"];
        val tv = ["Det","bodde","en","underlig"] : string list
        - first tv;
        val it = "Det" : string
    The answer should be "" (an empty string) if the list is empty.

  b) Write the function
        fun last ... = ... ;
    that returns the last element in the string list given as parameter.
    Example:
        - last tv;
        val it = "underlig" : string
    The answer should be "" (an empty string) if the list is empty.


  c) Write the function
        fun nth (n,x) = ... ;
    that retrieves element number n from the list x.
    The answer should be "" (an empty string) if there is no element number n.
    Example:
        - nth(1, tv);
        val it = "Det" : string
        - nth(3, tv);
        val it = "en" : string
        - nth(5, tv);
        val it = "" : string
 *)

fun first nil    = ""
  | first (s::_) = s;

fun last nil      = ""
  | last (s::nil) = s
  | last (_::ss)  = last(ss);

fun nth (_, nil)   = ""
  | nth (1, s::_)  = s
  | nth (n, _::ss) = nth(n-1,ss);

val tv = ["Det", "bodde", "en", "underlig"];
first tv;
last tv;
nth(1, tv);
nth(3, tv);
nth(5, tv);
```

```
(* Exercise 8:
   ("smaa" and "store" is Norwegian for small and large respectively)
   a) Write the function
         fun smaa (v, x) = ...;
      that returns a list of the "small" integers in the list x,
      i.e. the ones less than or equal to v.
      Example:
         - smaa(5, [4, 2, 8, ~2, 5, 12]);
         val it = [4,2,~2,5] : int list

   b) Write the function
         fun store (v, x) = ...;
      that works like a) but gives the "large" numbers, i.e. those that are larger than v.
      Example:
         - store(5, [4, 2, 8, ~2, 5, 12]);
         val it = [8,12] : int list

   c) Write the function
         fun quicksort (x) = ... ;
      that sorts the integer list x using the quicksort algorithm.
      (Quicksort amounts to selecting one element from the data and
         then split the other values in two groups: "small"
         (i.e. less than or equal to the chosen element) and "large".
         The two groups are sorted separately and then merged
         afterwards.)
      Example:
         - quicksort [5, 4, 2, 8, ~2, 5, 12];
         val it = [~2,2,4,5,5,8,12] : int list
*)

fun smaa (_, nil)   = []
  | smaa (v, n::ns) = if n<=v then n::smaa(v,ns) else smaa(v,ns);

fun store (_, nil)   = []
  | store (v, n::ns) = if n>v then n::store(v,ns) else store(v,ns);

fun quicksort (nil)   = []
  | quicksort (n::ns) = quicksort(smaa(n,ns))
                        @ [n]
                        @ quicksort(store(n,ns));

smaa(5, [4, 2, 8, ~2, 5, 12]);
store(5, [4, 2, 8, ~2, 5, 12]);
quicksort [5, 4, 2, 8, ~2, 5, 12];
```