

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

2 **(* ML-programming *)**

3

4 **(*Exercise 1:**

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

9

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

11 val digit = fn : int -> char

12

13 Try the following:

14

15 - ord (5);

16 - ord #"0";

17 - digit 5;

18 - digit 7;

19 *)

20

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

22 => **val** digit = **fn : int -> char**

23

24 ord (5);

25 => stdIn:11.1-11.8 Error: operator and operand don't agree [literal]

26 operator domain: **char**

27 operand: **int**

28 in expression:

29 ord 5

30

31 ord #"0";

32 => **val** it = 48 : **int**

33

34 digit 5;

35 => **val** it = #"5" : **char**

36

37 digit 7;

38 => **val** it = #"7" : **char**

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56 (*
57 *The functions "str" and "String.sub" convert between characters and strings. If s is a string then String.sub(s, n) returns*
58 *the nth character in s, counting from zero. We can define the digit function in a different way:*

59
60 - fun digit i = String.sub("0123456789", i);
61 val digit = fn : int -> char

62
63 *Try the following:*

64
65 - str (digit 5);
66 - digit 5;
67 *)

68
69
70 **fun** digit i = String.sub("0123456789", i);

71
72 **=> val** digit = fn : int -> char

73
74
75 str (digit 5);

76
77 **=> val** it = "5" : string

78
79
80 digit 5;

81
82 **=> val** it = #"5" : char

83
84
85
86
87 (*
88 *For each version of "digit", try to guess what would be the answer when called with the following parameters (try next*
89 *in the computer to check your guess):*

90
91 - digit ~1;
92 - digit 10;

93
94 *Play with other combinations of the functions in order to understand them better.*
95 *)

96
97
98 (* *The first version returns #":" and #"/", which are the characters just outside the range of digits in the ASCII*
99 *character set: *)*

100
101 digit 10;
102
103 **=> val** it = #":" : char

104
105
106 digit ~1;

107
108 **=> val** it = #"/" : char

109
110
111

```

112 (* The second version reports an error, by raising exception Subscript: *)
113
114 digit ~1;
115
116 => uncaught exception Subscript [subscript out of bounds]
117     raised at: stdIn:14.6-15.7
118
119
120 digit 10;
121
122 => uncaught exception Subscript [subscript out of bounds]
123     raised at: stdIn:14.6-15.7
124
125
126
127
128 (* Exercise 2:
129
130     Write the function
131         fun abs (v) = ... ;
132     that finds the absolute value of v (i.e. -v if v is negative).
133     Example:
134         - abs 17;
135         val it = 17 : int
136         - abs ~19;
137         val it = 19 : int
138 *)
139
140 fun abs v = if v<0 then ~v else v;
141
142 abs 17;
143 abs ~19;
144
145
146
147
148 (* Exercise 3:
149     Second degree polynomials  $ax^2+bx+c=0$  have solutions
150      $(-b+\text{Math.sqrt}(b^2-4ac))/2a$  and  $(-b-\text{Math.sqrt}(b^2-4ac))/2a$ 
151     where «Math.sqrt» is the square root function in ML.
152
153     Write the function
154         fun annegrad (a, b, c) = ... ;
155     that returns both solutions.
156     Avoid computing the square root more than once.
157 *)
158
159 fun annegrad (a, b, c) =
160     let val d = Math.sqrt(b*b - 4.0*a*c)
161     in
162         ((-b+d)/(2.0*a), (-b-d)/(2.0*a))
163     end;
164
165 annegrad(1.0, 2.0, ~3.0);
166 annegrad(1.0, 0.0, ~49.0);
167
168

```

169 (* **Exercise 4:**
 170 Write the function
 171 `fun power (a, b) = ... ;`
 172 that computes the exponentiation a^b (i.e. a multiplied by itself b times).
 173 We may assume that a and b are integers and that $b \geq 0$.
 174 Example:
 175 `- power(2, 23);`
 176 `val it = 8388608 : int`
 177 `- power(~25, 2);`
 178 `val it = 625 : int`
 179 *)

```
180 fun power (_, 0) = 1
181 | power (a, b) = a * power (a, b - 1);
183 power (2, 23);
184 power (~25, 2);
```

186
187
188
189

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

192
 193 A fraction can be represented by the record
 194 `{teller=..., nevner=...}`
 195 Write the function
 196 `fun add ({teller=t1,nevner=n1}, {teller=t2,nevner=n2}) = ... ;`
 197 that evaluates the sum of the two fractions.
 198 (Remember the rules for fraction additions:
 199 $a/b + c/d = (a*d + c*b)/(b*d)$
 200 .)

201 Example:
 202 `- add({teller=1,nevner=2}, {teller=3,nevner=4});`
 203 `val it = {nevner=8,teller=10} : {nevner:int, teller:int}`
 204 *)

```
205 fun add ({teller=t1,nevner=n1},
206         {teller=t2,nevner=n2}) = {teller=(t1*n2 + t2*n1), nevner=(n1*n2)};
208 add({teller=1,nevner=2},
209     {teller=3,nevner=4});
```

211
212
213
214
215
216
217
218
219
220
221
222
223
224

```

225 (*Exercise 6:
226 a) Write the function
227     fun max2 (a, b) = ... ;
228     that returns the largest of the two integers a and b
229     Example:
230     - max2(3, ~17);
231     val it = 3 : int
232
233 b) Write the function
234     fun max3 (a, b, c) = ... ;
235     that returns the largest of the three integers a, b and c.
236     Example:
237     - max3(4, ~5, 19);
238     val it = 19 : int
239
240 c) Write the function
241     fun max ... = ... ;
242     that returns the largest integer in the list of integers given as a parameter.
243     Example:
244     - max [7];
245     val it = 7 : int
246     - max [5, ~4, ~17, 28, 13];
247     val it = 28 : int
248 *)
249
250 fun max2 (a, b) = if a > b then a else b;
251 fun max3 (a, b, c) = max2(a, max2(b,c));
252 fun max nil = 0 (* this branch should never be taken it is added to avoid the compiler warning about nonexhaustiveness *)
253   | max [n] = n
254   | max (n::ns) = max2(n,max(ns));
255
256 max2(3, ~17);
257 max3(4, ~5, 19);
258 max [7];
259 max [~7];
260 max [5, ~4, ~17, 28, 13];
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281

```

```

282 (*Exercise 7:
283 a) Write the function
284     fun first ... = ... ;
285     that returns the first element in the string list given as parameter.
286     (There exists a standard operator "hd" which does this but do not use it.)
287     Example:
288     - val tv = ["Det", "bodde", "en", "underlig"];
289     val tv = ["Det","bodde","en","underlig"] : string list
290     - first tv;
291     val it = "Det" : string
292     The answer should be "" (an empty string) if the list is empty.
293
294 b) Write the function
295     fun last ... = ... ;
296     that returns the last element in the string list given as parameter.
297     Example:
298     - last tv;
299     val it = "underlig" : string
300     The answer should be "" (an empty string) if the list is empty.
301
302
303 c) Write the function
304     fun nth (n,x) = ... ;
305     that retrieves element number n from the list x.
306     The answer should be "" (an empty string) if there is no element number n.
307     Example:
308     - nth(1, tv);
309     val it = "Det" : string
310     - nth(3, tv);
311     val it = "en" : string
312     - nth(5, tv);
313     val it = "" : string
314 *)
315
316 fun first nil      = ""
317   | first (s::_) = s;
318
319 fun last nil      = ""
320   | last (s::nil) = s
321   | last (_::ss)  = last(ss);
322
323 fun nth (_, nil)  = ""
324   | nth (1, s::_) = s
325   | nth (n, _::ss) = nth(n-1,ss);
326
327 val tv = ["Det", "bodde", "en", "underlig"];
328 first tv;
329 last tv;
330 nth(1, tv);
331 nth(3, tv);
332 nth(5, tv);
333
334
335
336

```

337 (***Exercise 8:**
338 (*"smaa" and "store" is Norwegian for small and large respectively*)

339 a) Write the function

340 `fun smaa (v, x) = ...;`

341 *that returns a list of the "small" integers in the list x,*

342 *i.e. the ones less than or equal to v.*

343 *Example:*

344 `- smaa(5, [4, 2, 8, ~2, 5, 12]);`

345 `val it = [4,2,~2,5] : int list`

346

347 b) Write the function

348 `fun store (v, x) = ...;`

349 *that works like a) but gives the "large" numbers, i.e. those that are larger than v.*

350 *Example:*

351 `- store(5, [4, 2, 8, ~2, 5, 12]);`

352 `val it = [8,12] : int list`

353

354 c) Write the function

355 `fun quicksort (x) = ... ;`

356 *that sorts the integer list x using the quicksort algorithm.*

357 (*Quicksort amounts to selecting one element from the data and*

358 *then split the other values in two groups: "small"*

359 *(i.e. less than or equal to the chosen element) and "large".*

360 *The two groups are sorted separately and then merged*

361 *afterwards.)*

362 *Example:*

363 `- quicksort [5, 4, 2, 8, ~2, 5, 12];`

364 `val it = [~2,2,4,5,5,8,12] : int list`

365 *)

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

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

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

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

381