

```

1  % -----
2  % Logische Programmierung, ueb 3, 18.11.2002
3  % (c)Marc Dörflinger, Roger, Philip Iezzi
4  % -----
5
6
7  % -----
8  % my_length(List,N)
9  %
10 % calculate the length of a list
11 % -----
12
13 /*
14  * Alternative 1
15  * -> no accumulator needed
16  * -> not tail recursive
17  */
18
19 my_length([],0).
20 my_length([_|T],N) :-
21     my_length(T,X),
22     N is X + 1.
23
24
25 /*
26  * Alternative 2
27  * -> accumulator
28  * -> tail recursive
29  */
30
31 my_accLength(L,N) :-
32     my_accLength(L,0,N).
33 my_accLength([],A,A).
34 my_accLength([_|T],A,N) :-
35     A1 is A + 1,
36     my_accLength(T,A1,N).
37
38
39
40 % -----
41 % my_remove(Element,List,Remainder)
42 %
43 % Remove the first occurrence of an
44 % element in a list.
45 % -----
46
47 /*
48  * Alternative 1
49  * (Maya's solution)
50  */
51
52 my_remove(_, [], []).
53
54 my_remove(E, [E|T], T).
55
56 my_remove(E, [H|T], [H|RestRemainder]) :-
57     \+ E = H,
58     my_remove(E, T, RestRemainder).
59
60
61 /*
62  * Alternative 2
63  * (using append)
64  *
65  * This returns all possible solution where
66  * any occurrence of the element has been removed
67  * from the list.
68  */
69
70 my_remove2(E, X, X) :-
71     \+member(E, X). % Element is no member of the list
72

```

```

73 % Split list into pre- and post-part next to element
74
75 my_remove2(E, List, Remainder) :-
76     append(Pre, [E|Post], List),
77     append(Pre, Post, Remainder).
78
79
80 /*
81  * Alternative 3
82  * (without using append)
83  *
84  * UGLY SOLUTION - using cut!
85  */
86
87 my_remove3(E,L,R) :-
88     my_remove3(E,0,L,R).
89
90 my_remove3(_, _, [], []).
91
92 my_remove3(H, 0, [H|T], R) :- !,
93     my_remove3(H, 1, T, R).
94
95 my_remove3(E, Done, [H|T], [H|R]) :-
96     my_remove3(E, Done, T, R).
97
98
99
100
101 % -----
102 % my_remove_all(Element,List,Remainder)
103 %
104 % Remove all occurrences of an
105 % element in a list.
106 % -----
107
108
109 /*
110  * Alternative 1
111  * (without using my_remove)
112  */
113
114 my_remove_all(_, [], []).
115
116 my_remove_all(H, [H|T], R) :-
117     my_remove_all(H, T, R).
118
119 my_remove_all(E, [H|T], [H|R]) :-
120     \+ E = H, % or: E \== H
121     my_remove_all(E, T, R).
122
123
124 /*
125  * Alternative 2
126  * (using my_remove)
127  *
128  * UGLY SOLUTION - using cut!
129  */
130
131 my_remove_all2(_, [], []) :- !.
132
133 my_remove_all2(E, X, X) :-
134     \+member(E, X),
135     !.
136
137 my_remove_all2(E, L, R) :-
138     my_remove(E, L, X),
139     my_remove_all2(E, X, R).
140
141
142 % -----
143 % my_sort(List, SortedList)
144 %

```

```

145 % sort a list of unsorted integers
146 % -> insertion sort  o(n^2)
147 % -----
148
149 % my_sort requires insert
150
151 my insert(E, [], [E]).
152 my_insert(E, [X|Xs], [X|L]) :-
153     E > X,
154     my insert(E, Xs, L).
155 my_insert(E, [X|Xs], [E,X|Xs]) :-
156     E =< X.
157
158
159 % Wenn die erste Liste List leer ist,
160 % gibt es nichts zu sortieren, dann ist auch
161 % die zweite Liste SortedList leer.
162 my_sort([], []).
163
164 % Wenn die Liste List nicht leer ist, sortiere
165 % den Rumpf der Liste List und füge das Kopfelement
166 % in die sortierte Liste des Rumpfes ein. (insertion sort)
167 my_sort([H|T], SL) :-
168     my sort(T, X),
169     my_insert(H, X, SL).
170
171
172
173
174 % -----
175 % my_flatten(List, FlattenedList)
176 %
177 % flatten a list that contains sub-
178 % lists
179 % -----
180
181 % Wenn die Liste leer ist, dann gibt's nichts zu 'glätten'
182 my_flatten([], []).
183
184 % Wenn wir ein Nicht-Listen-Element haben, dann machen
185 % wir daraus eine Liste mit nur diesem Element.
186 my_flatten(X, [X]) :-
187     \+is_list(X).
188
189 % Wenn wir eine 'normale' Liste haben, dann rufen wir
190 % für den Head und den Tail rekursiv die Methode nochmals
191 % auf, auf fügen die daraus resultierenden Listen zusammen.
192 my_flatten([H|T], L3) :-
193     my flatten(H, L1),
194     my flatten(T, L2),
195     append(L1, L2, L3).
196

```