Списъци

**Представяне на списъци в prolog**

Темата на упражнението бяха списъците (както може да сте се досетили от хилядите заглавия, които сочат именно това) и някои основни операции над тях. Минималистичният синтаксис на **prolog** никак не ограничава работата с абстракции, стига те да се представят подходящо.
Ta, списъците се разделят на глава и опашка (най-общо - първи елемент и всичко останало), както и някакво означение за край на списък/празен списък, подобно стратегията за работа със списъци при функционалното програмиране. Което на **prolog** за списъка **[a, b]**, например, би следвало да изглежда горе-долу така:

list (a, list (b, f)).

- предикат, в който с **f** се означава празния списък.

Разбира се, че с горното само ни гъбаркаха и всъщност в езика има **специален** синтаксис за списъци, който поне не ни отегчава с кръгли скоби:

[ ] % означаващо празния списък

[ a | [ b | [ ] ] ] % подробно означение за списъка [ a, b ]

[ a, | L ] % означение за списъка [ a, <всичко останало>]

[ a, b ] % прегледно означение за списъка [ a, b ],

 % когато го подаваме като аргумент (в този му вид е трудно да се работи с него при дефинирането на правила)

**Извличане на елемент на списък**

**Извличане на първи елемент**

…е тривиално:

first(X, [X|\_]).

като долната черта е анонимен аргумент - подчертаваме (буквално) това, че не ни пука какво има там. Мнемониката е голяма работа при програмирането, дори има езици, в които тя е всичко най-трудно за написване, но това е една друга приказка.

**Извличане на произволен елемент**

Важното е, че макар и да не ни помага в списък за пазаруване на произволна майка, за кратки списъци работи извличане на елемент по следния начин:

element\_at(\_, \_, \_, ..., X|\_).

А ето и не-дебилен начин за това да се извлече K-тия елемент от списък (с поне K елемента):

element\_at(H, [H|T], 1).

element\_at(X, [H|T], Y) :- Y1 is Y - 1, element\_at(X, T, Y1).

Ако искаме да сме по-прецизни:

element\_at(H, [H|T], 1).

element\_at(X, [H|T], Y) :- integer(Y), Y > 1, Y1 is Y - 1, element\_at(X, T, Y1).

**Извличане на последен елемент**

Логиката е следната - ако списъкът се състои от един елемент, взимаме го, инак прилагаме същото изискване към опашката му, т. е.:

last(X, [X]).

last(X, [\_|T]):- last(X, T).

**Принадлежност към списък**

Проверка дали даден елемент се среща в списък:

has\_item(X, [X|T]).

has\_item(X, [H|T]) :- has\_item(X, T).

Забележка: Това връща не резултат в променлива, а **true**, когато е изпълнено. Може да се направи и с променлива, естествено.

**Конкатениране на списъци**

Ако някой от списъците, които искаме да конкатенираме, е празен, резултатът е другият списък, иначе резултатът има главата на първия списък и опашка - конкатенацията на остатъка от първия списък с втория. Съвсем като функционалното програмиране, с разликата, че не "връщаме" резултат, а насилваме **prolog** да ни помни фактите:

my\_append([], [Y|Ys], [Y|Ys]).

my\_append([X|Xs], [], [X|Xs]).

my\_append([X|Xs], [Y|Ys], [X|Res1]) :- my\_append(Xs, [Y|Ys], Res1).

**Подсписъци**

**Префикс**

Тук имаме една проверка за коректност дали това, в което търсим даден префикс, въобще е списък. А какво е списък - празният списък и всичко, което изглежда като списък, звучи като списък и мирише на списък:

is\_list([]).

is\_list([\_|\_]).

Тогава просто имаме предвид, че магическият празен списък е префикс на всеки списък и дефинираме:

preffix([], L).

preffix([X|P], [X|L]):- preffix(P, L).

Суфиксът се проверява по подобен-не-записах-какъв начин. Повярвайте.

**Подсписък**

За проверката дали един списък е подсписък на друг, асистентът позволи да еволюираме и да използваме нещо, което вече сме дефинирали - конкатенацията на списъци. Идеята е, че ако единият списък действително е подсписък на другия, то *префикс* + подсписъка + *суфикс* == списък. Казваме, че S е подсписък на L, ако съществува някакво парче X, което като конкатенираме с някакъв суфикс B, да дава целия списък L. Към X пък поставяме изискването да е конкатенация на някакъв префикс А с нашия евентуален подсписък L. Абе:

sublist(S, L):- append(X, B, L), append(A, S, X).

**Премахване на елемент**

Имаме конкретен елемент, който не го щем в списъка. Като срещнем първия такъв - махаме го, без да го питаме:

remove(X, [X|Xs], Xs ).

remove(X, [Y|Ys], [Y|Zs]):- remove(X, Ys, Zs).

**Пермутации**

Тази толкова основна операция, за която нямахте търпение да видите как става, нали. Само дето е във вариант на проверка дали един списък е пермутация на друг, а не на генериране на такава. Ми на. Та как става - хващаме главата на единия списък и я премахваме от другия (използваме вече дефинираното правило remove), повтаряме с остатъка от двата списъка, докато не падне и последната глава и ламята издъхне в кърви:

perm([], []).

perm([X|Xs], L):- remove(X, L, Res), perm(Xs, Res).

Алтернативно, това:

perm([], []).

perm(L1, L2):- sort(L1, SortedL1), sort(L2, SortedL2), SL1 == Sl2.

(Подсказка: Асимптотично по-бързо е)

**Сортиране**

Проверка дали списък е сортиран:

is\_sorted([]).

is\_sorted([X]).

is\_sorted([A, B|L]):- A < B, is\_sorted([B|L]).

Колкото до самия акт на сортиране, някой по-малко честен човек би казал "о, колко лесно и елегантно е сортирането в Пролог". Не е лесно. И далеч не е бързо.
Както и да е, дами и господа - merge sort:

merge\_sort([],[]).

merge\_sort([X],[X]).

merge\_sort(List,Sorted):- halve(List,L1,L2), % разделя списъка на 2

 merge\_sort(L1,Sorted1),merge\_sort(L2,Sorted2), % и сортира всяка част отделно, после ги събира

 merge(Sorted1,Sorted2,Sorted).

% Помощни процедури

% Това прави от 1 списък - два

halve([], [], []).

halve([H], [H], []).

halve([H1|[H2|T]], [H1|L1], [H2|L2]) :- halve(T, L1, L2).

% Това обединява два сортирани списъка в един - пак сортиран.

% резултатът е в последния параметър.

merge([],L,L).

merge(L,[],L).

merge([X|T1],[Y|T2],[X|T]):-X=<Y,merge(T1,[Y|T2],T).

merge([X|T1],[Y|T2],[Y|T]):-X>Y,merge([X|T1],T2,T).

**Дължина на списък**

len(0, []).

len(X, [H|T]) :- len(Y, T), X is Y + 1.

Толкоз. Избягваме да изразяваме събирането направо някъде из скобите, че най-много да ни застигне някое проклятие.
Ако още не е станало очевидно, този език е анти-C++.

**Специален елемент (минимален, максимален, най-малък, …)**

От математическа гледна точка имало разлика между минимален и най-малък елемент, казват. Dude, we don't care. Минимален елемент на списък може да се намери като първо дефинираме как да разпознаем по-малкия от два елемента:

% помощната дефиниция

min2(A, B, A):- A < B.

min2(A, B, B):- A >= B.

%същинската дефиниция

min(X, [X]).

min(X, [H|T]):- min(N, T), min2(H, N, X).