Генератори ( и някои други работи )

**Генератор на естествени числа**

Идеята: взимаме число за базов случай на "естественост", в случая тук - скандалната нула ( знаем, че Тинчев я признава и сме спокойни ), и със стъпка единица си генерираме, докато ни писне ( на нас или на **prolog** интерпретатора ) поредни естествени числа.

%n(<променлива>).

n(0).

n(X) :- n(X1), X is X1 + 1.

Две забележки по кода:  
1. предикатите от втория ред са в тази поредност и не в обратната, защото е необходима нещо като дефениция на X1, след като я използваме пряко, за да определим X.  
2. определяме Х като Х1 + 1, а не Х1 като Х - 1, не защото не познаваме операциите, а защото за Х нищо не знаем.

**Една примерна задача**

Да се генерират числа по-малки от 1000, които са сума на два точни квадрата на естествени числа.  
Използваме, тук, а и в доста други задачи, помощнен предикат between, който проверява принадлежност на число към интервал. Приближаваме лявата му граница към дясната и проверяваме дали числото ни е в получения по-тесен интервал ( което в случая означава че в някой момент числото ни е станало лява граница на подинтервала ( туй е базовият случай ) ).

%between (<число>, <лява граница на интервала>, <дясна граница на интервала>).

between(A, A, B) :- A =< B.

between (X, A, B) :- A < B, A1 is A + 1, between (X, A1, B).

Забележка по кода: моят интерпретатор не желаеше да приеме оператор '<=' в по-разпространения му вид, а именно '<=', но се съгласи на '=<', вие знаехте ли?

Самото генериране на числа ще стане като симулираме два вложени цикъла. Едното число ще варира от 0 до някакво ограничение, другото - от първото число до ограничението. Спрямо условието си слагаме за примерно ограничение числото 32, тъй като квадратът му е най-близо до 1000. И сега вече генерирането на сумата от квадратите им се побира на един стандартен ред:

%sum1000 (<променлива>).

sum1000(X) :- between(Y, 0, 32), between(Z, Y, 32), X is Y\*Y + Z\*Z, X < 1000.

Поставяме си строгото ограничение Х < 1000, защото предишното ни ограничение бе за целите на циклите, а самият резултат може да надхвърли 1000 въпреки него.

**Числа на Фибоначи**

Генерираме ги, използвайки итеративния вариант на познатия ни алгоритъм:  
a, b - числа на Фибоначи  
b, a + b - следващите числа на Фибоначи  
Нещо интересено, можем да правим предикати с едно и също име, **prolog** ги различава по броя на аргументите/термовете. Така и правим. С единия предикат генерираме по две числа на Фибоначи, а с другия - взимаме само първото от тях.

%fib(<променлива>).

fib(1, 1).

fib(B, C) :- fib(A, B), C is A + B.

fib(X) :- fib (X, \_).

**Една почти сигурна задача за писмен изпит**

Написан е някакъв код, трябва да кажем какво извежда изпълнението му. Решението е дълго и, за да си помагаме, си строим дърво или стек. Картинка/и ще опитам да сложа по-късно. Примерният код на задачата бе:

q(X) :- Y is X - 3, write(Y), nl.

q(X) :- X < 0, s(X).

s(X) :- Y is X + 4, write(Y), nl, q(Y).

s(X) :- write(X), nl, Y is X + 3, q(Y).

?- q(-7), write('='), nl, fail.

Подсказка: **prolog** изпълнява дефиниции, както са си написани отгоре-надолу ( в случаи на две дефиниции на един предикат, както имаме тук ). Разбира се, тръгва по едната и се залутва по завоите и препратките й и, чак като свърши цялата работа по нея, минава към следващата.