Пролог - увод

Пролог е език, който мисли сам.
Поне това е идеята.

Представяш му някакви факти, останалото сам се сеща.
И е умен. Доста по-умен от всички други езици.

**Защо няма 89% пазарен дял?**

* На Пролог се пише адски трудно. Такива неща като if / while са за женствени мъже.
* Често необходимите факти за решаването на задачата са толкова, колкото и кодът, който я решава на произволен друг език.
* **Бавно** е.

Нека се разберем отсега. Това все още не е езикът на бъдещето. Скоро няма и да стане. Но има хляб в него, има *много* нови идеи. Освен това - ако съвсем всичко друго се провали - най-вероятно с Пролог ще се получи. Точно така, Пролог е *Капитан Планета* на езиците за програмиране.

**Име:**

Идва от **Pro**gramming **Log**ic. Очевидно е кръщаван от същите хора, които са нарекли FORTRAN.

**Организационни въпроси**

**НЕ** ползвай **Strawberry Prolog**. Писано е от българин, май. Но не е готово. Естествено, много работа е свършена по програмата, но има още какво да се желае. В някои случаи резултатът изкаран там не се придържа към стандарта, което може да е проблемно.
**GNU-Prolog** е добра алтернатива.

Всъщност един израз на Пролог може да върне много резултати.
Когато на екрана се покаже един, всяка имплементация има някаква команда(т.е. някакъв клавиш), с която се показва и следващия отговор. Даже може да има да се показват всички отговори.

**Синтаксис**

Много, много лесен.
Ако си мислеше, че Лисп има само пет-шест команди - е, Пролог има и по-малко.

**Изречение**

Всяко нещо тук е *Изречение*. Тоест, завършва с точка. А някой си мисли, че езикът му може да бъде четен като нормален текст. Естествено, това са глупости.
Но, всяка команда завършва с точка. Почти както в Ц++ завършват с **;**.

***"Функция"***

"Функция" е силно казано в случая, но е нещото, което най-много прилича.
Най-точно може да се оппредели като твърдение.
Всъщност, всичко в езика е твърдение от вида:

Това :- Онова.

Да, на някои имплементации може да се пише на Unicode. Както винаги, не е препоръчително.
Горното онзачава, че "Това" е истина, когато "Онова" е истина.
Примерно, фунцкията less\_than\_five се записва така:

less\_than\_five(X) :- X < 5.
?- less\_than\_five(4).
> yes
?- less\_than\_five(5).
> no

Може да има по няколко условия:

f(X) :- X < 5, X > 3.

Това е все едно "X < 5" И "X > 3"

Също така, функция може да има по няколко дефиниции. Примерно:
g(X) :- X < 3.
g(X) :- X > 5.

Това е "X > 5" Или "X < 3". Почти, де. Има малко разлики.

**Променлива**

**Внимание!!!**
Променливите винаги са с **главни букви**! Най-често срещаната грешка.
Всяко нещо с малки букви е константа.

Когато в някоя дефиниция се сложи променлива, тя при извикване се замества с каквото е дадено. Примерно, 5. Но може и да не е 5. Може и да е 6. В моя случай най-често се замества с 3.

less\_than(X, Y) :- X < Y.
?- less\_than(2, 3).
> yes
?- less\_than(6, 5).
> no

**Въпрос**

Въпросите, както вече стана ясно, се задават с

?- нещо(някакви\_данни).

Проверява се дали съответното извикване се оценява като истина и се връща резултатът.

Когато в някой "въпрос" се сложи променлива, като "отговор" се връщат всички възможни стойности, които може да заема променливата, а дефиницията все пак да е вярна.

p(x).
?- p(Y).
> Y = x
>
> yes

Малкото **x** е константа. Y е променлива. За да е вярно горното, трябва Y = x.

**Константа**

Ъъъ, да. Това са нещата с малки букви.

**Факт**

Факт е някакво винаги изпълнено твърдение.
Например, твърдение без условие.

p(5).

p, извикано с 5 е вярно. Сега, какво е p, какво е 5 - това остава за нас да си решим. Компилаторът не го интересува.

p(5) :- true.

е същото нещо, но е ненужно да се пише така.

**Коментар**

Коментар до края на реда се слага с **%**. Всичко след процента не се обработва.

**Лесен пример**

Имаме граф.
В него има ребра.
Записваме ребро между върхове **a** и **b** като:

edge(a,b).

Как ще намерим дали има път от A до B?
(A и B са променливи, ако не е очевидно).

Дефинираме си твърдение "Има път". Какво е условието да има път от A до B?
Ами, или има ребро A->B, или има някакво ребро A->C, **И** има път C->B.

path(A,B) :- edge(A,B).
path(A,B) :- edge(A,C), path(C,B).

Току-що тихомълком си дефинирахме нова променлива, която се ползва в дефиницията. Да, може.
Егати пичовете дето сме, а?

Това работи, впрочем:

edge(a,b).
edge(b,c).
edge(b,d).
edge(d,e).
edge(e,f).
edge(f,c).

?- path(a,c).
> yes

?- path(X,c).
> X = b
> X = f
> X = a
> X = a
> X = b
> X = d
> X = e

(има два пътя от **a** до **c**, впрочем)

?- path(X,a).
> no

Не е много зле.

**Трудна / Лесна задачка**

Намирането на производни като цяло е болезнена работа.
Тук, обаче, е сравнително лесно.
Трябва само да му сложим правилата за диференциране, и ще се сети само.

само че…

**Връщане на резултат**

Твърденията в Пролог връщат само **yes / no**(true / false).
На нас ни трябва…малко повече от това.
Как да връща резултат?
Ами, с грозни хакове - как иначе.

Можем винаги да слагаме първия аргумент да е променлива. И да гледаме каква стойност заема:

% твърдения
d(4, 2).
d(5, 2).
d(3, 2).
d(3, 1).

?- d(X, 2).
> X = 4
> X = 5
> X = 3

?- d(X, 1).
> X = 3

Та, производни.
Ето такива ще са дефинициите:

d(производна\_на\_f, f).

Така в първата променлива ще имаме функцията, която е производна на f.
Т.е. ако го викнем с d(X, някаква\_функция), в X ще е записано каквото трябва.
Ето малко примерни твърдения(ти ще ми кажеш защо са верни):
(На Strawberry Prolog долното се чупи като пич, впрочем).

% Това работи за цели числа.

% За домашно - как да започне да работи за реални :) :)

d(0, X) :- integer(X).

% x е константа. Т.е. буквичка, реално(т.е. не е число).

d(1, x).

% Диференцираме по x, не по y.

d(0, y).

% Точно така, \*\*толкова\*\* умно е.

% G1 е производната на G, F1 - на F.

% Вдясно ги смятаме. Освен това, ако някоя производна върне "no",

% целият резултат ще е "no".

d(F1 + G1, F + G) :- d(F1,F), d(G1,G).

d(F1\*G + G1\*F, F \* G) :- d(F1,F), d(G1,G).

d((F1\*G - G1\*F)/(G\*G), F / G) :- d(F1,F), d(G1,G).

% Ето една уловка - това прави сложна фунцкия, всъщност.

d(cos(F) \* Y, sin(F)) :- d(Y, F).

Ето малко примери, за да стане ясно, че все пак работи:

?- d(X, x\*x).

> X = 1\*x+1\*x

?- d(X, x\*x + 3 \* x + 3 + x\*x\*x).

> X = 1\*x+1\*x+(0\*x+1\*3)+0+((1\*x+1\*x)\*x+1\*(x\*x))

Ей, казах верни - не оптимално съкратени(и това може да се направи, впрочем).
Нещо повече, това много много лесно може да се използва за намиране на интеграли.
Разби ли ти съзнанието, а?
Просто като променлива пъхаме **другото** нещо.

d(1\*x+1\*x+(0\*x+1\*3)+0+((1\*x+1\*x)\*x+1\*(x\*x)), X).

> X = x\*x+3\*x+y+x\*x\*x

Забележи, че говедото се сети за шибаната КОНСТАНТА!!! Егати умния език, честно.

Цялата идея на Пролог е, че вътре вкарваме не код, не алгоритми, а информация.
Която може да бъде преизползвана доста.
Кадърна програма на Пролог ще реши доста проблеми.
Ако се комбинира с C++, или с някой език от по-високо ниво, това би било доооста полезно.