PROLOG

PROGRAMMATION EN **LOG**IQUE

프로그래밍 언어. 과제5. Prolog.
홍익대학교 길재은

개요

- Prolog 설치 방법
- Prolog 란?
- Prolog 의 특성
- Prolog 의 데이터형
- Prolog 과제 팁

PROLOG 설치 방법

- swipl : prolog IDE 실행
- halt(). : swipl 종료
- swipl –s <파일이름>.pl
- (윈도우, 맥 설치 가능) http://www.swi-prolog.org/

```
[       @localhost pl]$ swipl -s  hanoi.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- hanoi(3).
1->[1,2]
2->[1,3]
1->[2,3]
3->[1,2]
1->[3,1]
2->[3,2]
1->[1,2]
true.

?- halt().
[       @localhost pl]$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- halt().
```

PROLOG란?

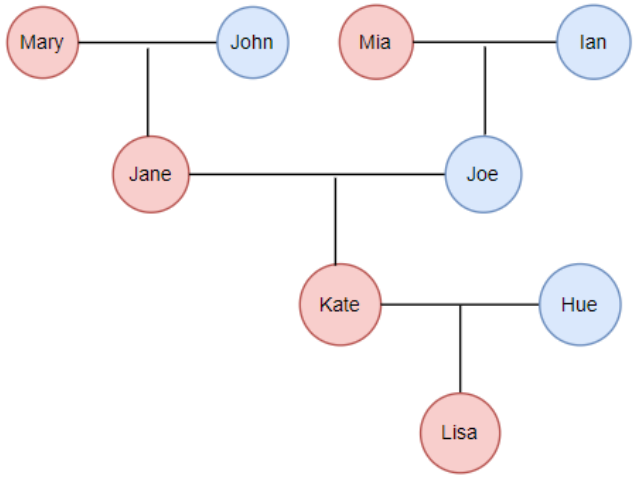
- 논리형 프로그래밍 언어
- 1973년 프랑스 알랭 코메르가(Alan Colmerauer)가 개발한 언어
- **논리식**을 토대로 하여 **오브젝트와 오브젝트 간의 관계에** 관한 문제를 해결하기 위해 사용한다.

PROLOG란?

- **Symbolic Programming** (vs Numeric Computation)
  - 수치 연산을 못하는 것은 아니지만, 취약하다. 언어적 지원이 거의 없다.
  - **심볼 연산 관련 구현에서** 다른 언어라면 알아보기 힘든 다량의 코드가 프롤로그에서는 깔끔하게 표현된다.
- **Declarative Programming** (vs Procedural Programming)
  - 대부분의 절차적 언어와는 달리 프롤로그는 **선언적 패러다임**으로, 같은 문제에 대해 다른 문제 해결방법을 강제한다.
- **Logic Programming** (vs Functional Programming)
  - 말 그대로, **논리를 프로그래밍 언어로** 표현한다.

PROLOG란?

PROLOG란?




```
% fact1          % fact2
female(mary).    parent(mary, jane).
female(jane).    parent(john, jane).
female(kate).    parent(mia, joe).
female(mia).     parent(ian, joe).
female(lisa).    parent(jane, kate).
male(john).      parent(joe, kate).
male(ian).       parent(kate, lisa).
male(joe).       parent(hue, lisa).
male(hue).
```

```
% rule
father(X):-parent(A,X), male(A), write(A).
mother(X):-parent(A,X), female(A), write(A).
grandfather(X):-parent(A,X),parent(B,A), male(B), write(B).
grandmother(X):-parent(A,X),parent(B,A), female(B), write(B).
```

```
?- % query
|    mother(kate).
jane
true .

?- % query
|    grandmother(kate).
mary
true ;
mia
true .
```

데이터 유형

데이터형에는 아톰(Atom), 변수,
숫자(Number) 가 있다.

Atom

```
father(a,b).
father(b,c).
```

- 위의 코드에서 `a`,`b`,`c` 가 다 아톰이라 볼 수 있다.
- 아톰은 데이터, 상수, 변할 수 없는 그냥 정해진 데이터이다.
- 일반적으로 첫글자는 소문자로 시작하거나 ' ' 안에 가두어져있다.
- 첫 글자 이후로는 문자,숫자,_(밑줄) 로 이뤄짐.

데이터 유형

- **변수 R을 직접 출력하는 방법.**

```
?- func(5,R).
R = 15.
```

```
?- func(10,R).
R = 55
```

- **변수 R이 될 수 있는 값을 모두 물어보는 방법.**

변수(Variable)

```
father(a,b).
father(b,c).
grandfather(X,Z):- father(X,Y), father(Y,Z).
```

- prolog 에서는 다른 언어와 달리 변수가 메모리 공간을 의미하지 않는다.
- just 관계를 나타내기 위해 존재한다.
- 예를 들면, X는 Y 이고, Y 는 Z 이면, X 는 Z 이다라는 것을 표현하기 위해 변수가 존재한다.
- 변수는 무조건 첫 글자가 대문자나 _(밑줄)로 시작해야한다.

특수 변수 `_`

- 어떠한 값이 와도 된다는 것을 뜻하는 특수변수 이다.
- (한 구문에 한 변수를 한 번만 쓰면 경고창이 뜨는데, 그 변수 대신 `_` 를 쓰면 된다.)

```
father(john, tom).
male(X):-father(X,_).
```

데이터 유형

숫자(Number)

- 숫자는 숫자로 이뤄져있다.
- 12344

compound term

- functor(params)의 구조

예) `person_friends(zelda,[tom,jim])`.

리스트

- 데이터형에는 아톰(Atom), 변수, 숫자(Number) 가 있다.
- 이것들을 모아 만들어진 것이 리스트(List)이다.
- 리스트는 대괄호([]) 로 표현된다.
- [1, two, 'THREE', A] 이런 것이 바로 리스트 이다.
- 리스트 안에 리스트가 들어갈 수 있다. [1,2,[4,5]]

PROLOG 요소

사실

- 어떤 하나의 사실을 true 로 정의하는 것이다.
  - 예. `fruit(apple).`
- 관계와 객체의 이름은 소문자로 시작한다.
- 관계의 이름을 먼저 쓰고, 객체들은 괄호 안에 쉼표로 구분한다.
- Fact는 항상 `.` 으로 끝난다.
- 예) `John likes Mary` → `likes(john, mary).`

규칙

- 규칙은 다른 사실로부터 하나의 또 다른 사실을 추론 가능하게 한다.
- ```
  Head  :- Body
  ```
  - "Body가 true이면 Head is true" 라 읽는다.
  - 예. `sweet(Var) :- fruit(Var).`
    - Var(변수)가 과일이라면, 달콤하다.
- `P:- Q, R.` % If Q **and** R ,then P.
- `P:- Q;R.` % If Q **or** R, then P.
  ```

PROLOG 요소

질문(Questions)

- 사실들을 저장한 후에 그에 관한 질문을 제기한다.
- 질문은 사실과 유사하나, 그 앞에 특수 기호가 있습니다.
- 예제)

  `Does mary own the book` → ?- `owns(mary, book).`

해당 질문이 참,거짓 판별

- 예. `?-sweet(apple)`
  - true

프롤로그 디버깅

Trace

- trace 는 질의 시에 질의 과정을 추적하는 trace 모드로 들어가는 역할을 하는 명령어이다.
- trace 를 이용하면 질의의 내부 원리를 이용하는데 도움이 된다.
- `?-trace.` 라고 질의하면 trace 모드로 진입한다.
- `?-nodebug.` 라고 치면 trace 모드에서 빠져나올 수 있다.

```
male(tom).
male(john).
female(jane).
female(irene).
female(britney).
person(A):- male(A);female(A).
```

```
?- trace.
true.

[trace]  ?- male(X).
   Call: (10) male(_14488) ? creep
   Exit: (10) male(tom) ? creep
X = tom ;
   Redo: (10) male(_14488) ? creep
   Exit: (10) male(john) ? creep
X = john.

[trace]  ?- nodebug.
true.
```

[trace 모드에서 `?-male(X).` 를 질의한 결과]

속성 설명

- `_14488` : 임시 변수(변수)
- `creep` : 엔터를 눌렀을때 나오는 결과

- Call : 질의에 답을 찾기 위해 시도했다.
- Exit: 질의에 대한 답을 찾았다.
- Redo: 질의에 대한 답을 찾기 위해 다시 해보았다.
- Fail: 답을 찾는데 실패했다.

추적중에 `h` 를 누르면 도움말이 나옵니다.

Prolog 논리연산자

```
1 isIn(In,List):-
2     \+member(In,List)->
3     write('no!');
4     write('yes!').
5
```

- ! ➔ ₩+ not
- != ➔ =₩= A!=B A와 B가 다르다면.
- == ➔ =:= A==B A와 B가 같다면
- A&&B ➔ A,B AND (앞에서부터 순서대로)
- A || B ➔ A;B OR
- if(A==true) B; ➔ A->B (A가 true면 B를 수행)

```
1 comp(A,B):-
2     A>B ->
3     write('A');
4     write('B').
```

```
?- comp(1,5).
B
true.

?- comp(5,1).
A
true.

?-
```

PROLOG 예시

```
1  sum(A):- func(A,R), write(R), nl.
2
3  func(0,0):-!.
4  func(N,R) :-
5    N > 0,
6    N1 is N-1,
7    func(N1,R1),
8    R is R1+N.
```

```
?- sum(10).
55
true
```

```
?- sum(5).
15
true.
```

예시가 알려주는 것!

- 재귀의 구조.
- , (and구문)의 사용법.
- ! 의 사용법.
- is의 사용법.

- 변수 R을 직접 출력하는 방법.

```
?- func(5,R).
R = 15.
```

```
?- func(10,R).
R = 55
```

- 변수 R이 될 수 있는 값을 모두 물어보는 방법.

!

- cut
- 원치않는 역추적을 방지하고, 불필요한 계산을 방지한다.

is

```
A is 3+4.
```

A=7.

write

- `write`[redacted]는 화면에 문자열을 출력하기 위해 사용된다.

```
write('Hello').
```

- `nl` : new line 을 의미한다.

PROLOG LIST 예제 – |

- List에서 앞 쪽 여러개의 원소를 구분하는 방법.
- functor([Head1,Head2,...|Rest])

```
head([A1,A2|B]):-
    write('Head 1= '),
    write(A1), nl,
    write('Head 2= '),
    write(A2), nl,
    write('Rest= '),
    write(B), nl.
```

```
?- head([a,b,c,d,e]).
Head 1=a
Head 2=b
Rest= [c,d,e]
true.

?-
```

merge()

- List합치기
- merge(A_list, B_list, Var).

```
1 ans(A,B):- merge(A,B,C), write(C).
```

```
?- ans([1,2,3],[4,5,6]).
[1, 2, 3, 4, 5, 6]
true
```

```
?- merge([1,2,3],[4,5,6],C).
C = [1, 2, 3, 4, 5, 6]
```

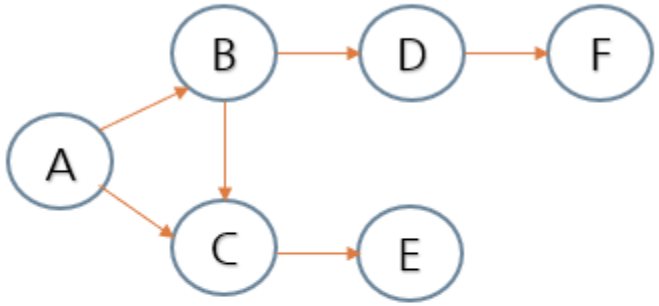
numlist(A, B, N_List)

- List 만들기.
  - numlist(A, B, N_List).
  - A부터 B의 정수값을 가지는 List 생성.

```
?- numlist(1,5,List).
List = [1, 2, 3, 4, 5]
```

findall()

- findall(Var, func(Var), N_list).
  변수 Var이 함수 func() 안에서 가질 수 있는 모든 값을 N_list에 list로 저장




```
1 near(a,b). near(a,c).
2 near(b,c). near(b,d).
3 near(c,e).
4 near(d,f).
5
6 path(X,X).
7 path(X,Y) :- near(X,Z),path(Z,Y).
8
9 fnd() :- findall(X,path(b,X),L),write(L).
```

```
?- path(b,X).
X = b ;
X = c ;
X = e ;
X = d ;
X = f ;
false.

?- fnd.
[b,c,e,d,f]
true.

?-
```

setof()

- setof(Var, func(Var2), N_list).
  - func()를 만족하는 Var들을 Set에 넣는 함수.
  - ex) atom인 foo 중에서 가운데 원소가 c인 첫번째와 마지막 원소의 쌍을 Set에 저장.

```
1 foo(a, b, c).
2 foo(a, b, d).
3 foo(b, c, e).
4 foo(b, c, f).
5 foo(c, c, g).
```

```
?- setof([X,Y],foo(X,c,Y),Set).
Set = [[b, e], [b, f], [c, g]].

?-
```

member()

- member(Var, N_list).
  - Var가 N_list의 원소인지 대답하는 함수.

```
1 isIn(In,List):-
2     \+member(In,List)->
3     write('no!');
4     write('yes!').
5
```

```
?- isIn(a,[a,b,c]).
yes!
true.

?- isIn(k,[a,b,c]).
no!
true.

?-
```

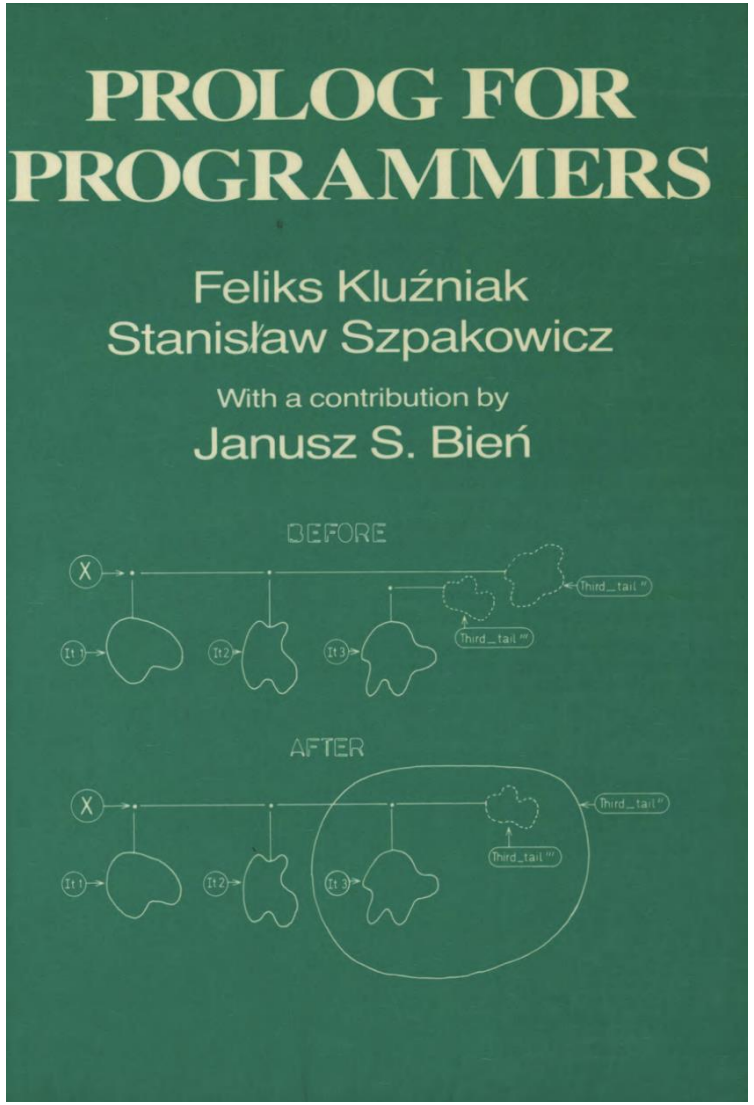
reverse()

- reverse(N_list, Var).
  - N_list의 원소의 순서를 뒤집어서 Var에 리스트로 저장.
  - 코딩 스타일에 따라서 재귀로 짜다보면 리스트를 뒤집어야 하는 경우가 발생할 수도!

```
?- reverse([a,b,c,d,e],A).
A = [e, d, c, b, a].

?-
```

프롤로그 팁



- https://sites.google.com/site/prologforprogrammers/the-book
- 코딩 테크닉을 모를 때 참고.