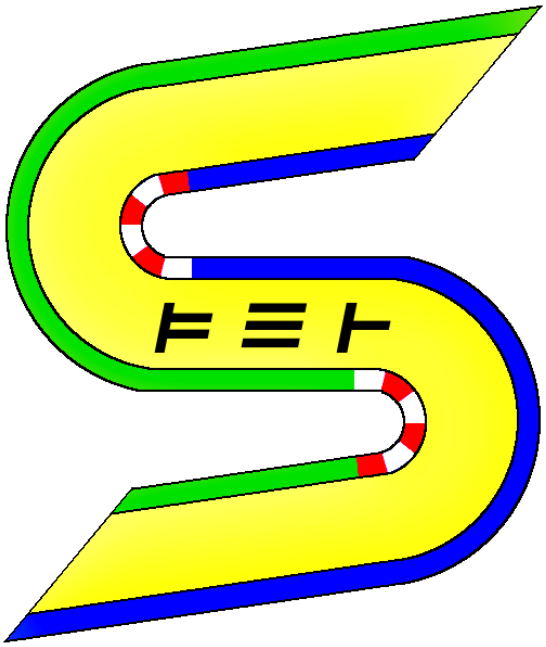
GNU Prolog Handbook

IF1221

Logika Komputasional



Disusun oleh:

Asisten Gaib'22

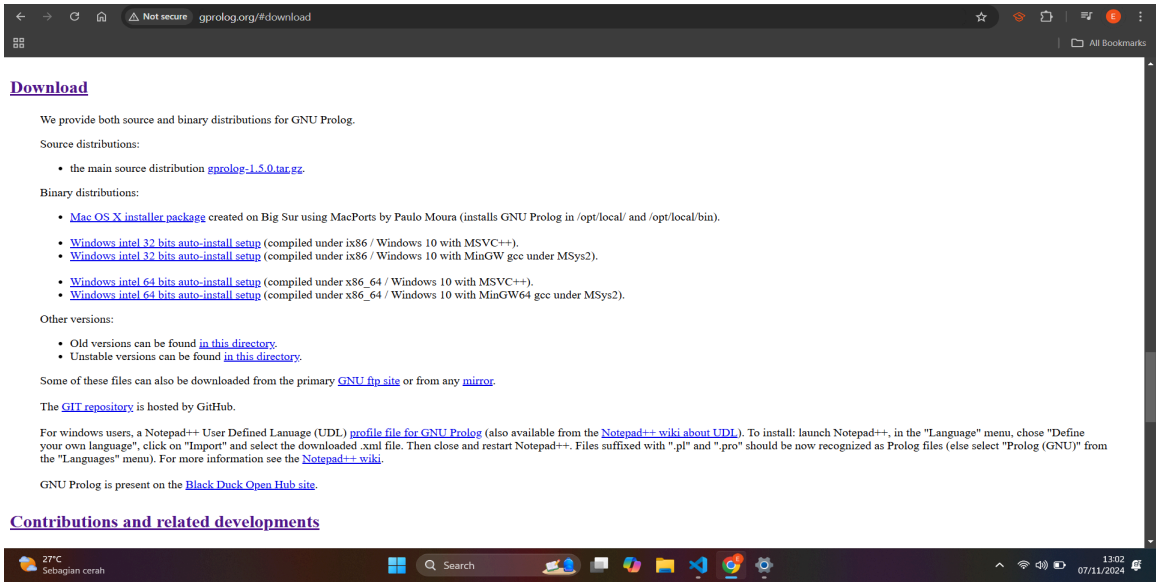
Institut Teknologi Bandung

2024

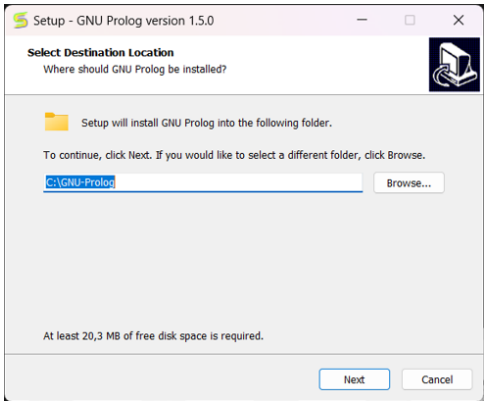
DAFTAR ISI

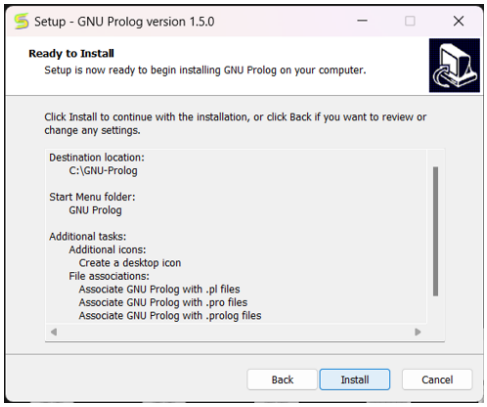
1. SETUP DAN INSTALASI

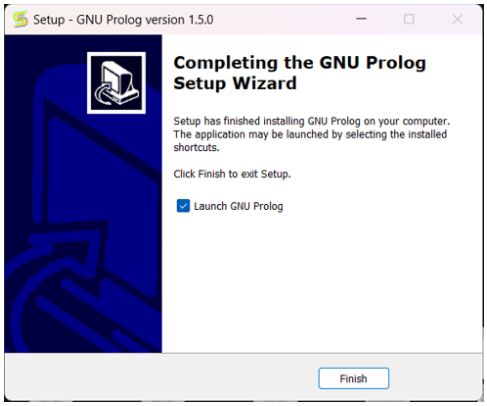
1. Buka web prolog bagian download atau klik di sini. Download yang di bagian binary distribution (sesuaikan saja dengan laptop kalian



2. Install (kaya install biasa ajah wkwk)



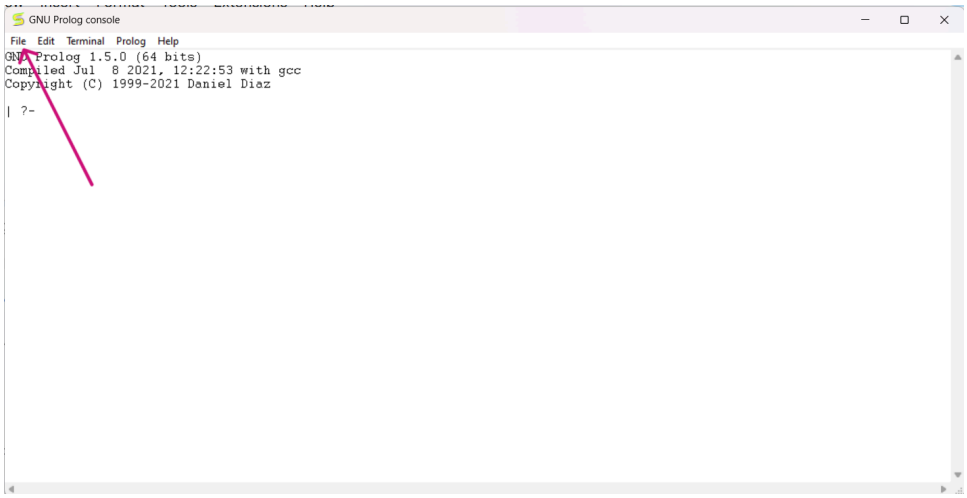


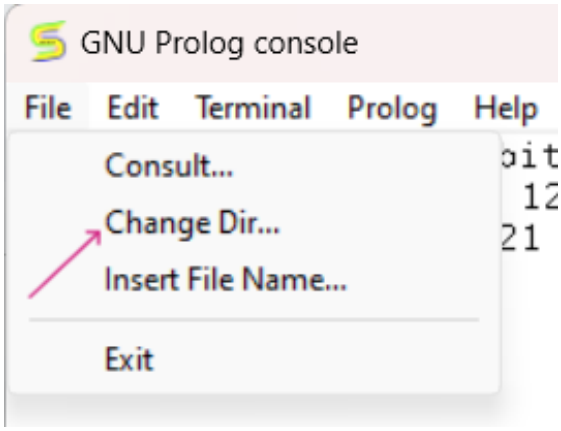


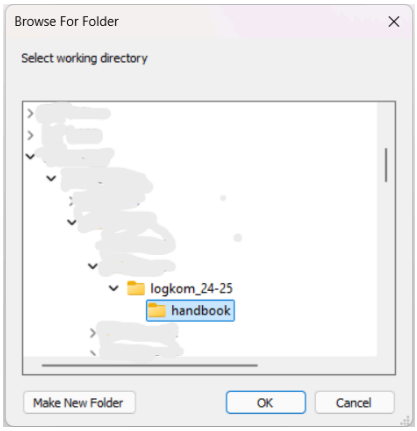
And you're done…

2. HOW TO RUN

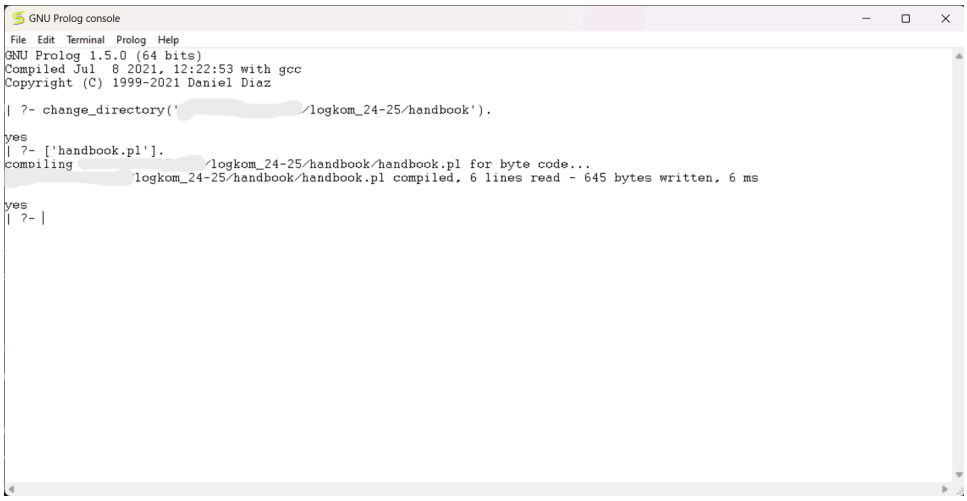
- Buat file prolog (.pl).
- Ubah directory ke folder tujuan: file → change dir → pilih folder.







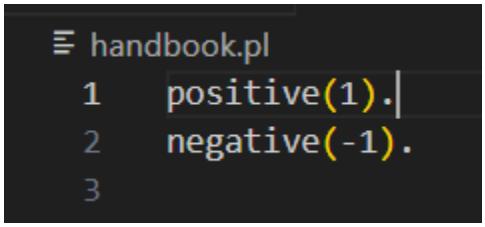
- Compile dengan command: [‘{nama file}’]. (jangan lupa titiknya!)



Di contoh ini, nama filenya yaitu handbook.pl

And just like that…

Contoh file handbook.pl :



Hasil eksekusi di terminal:

```
yes
| ?- positive(X).

X = 1

yes
| ?- negative(X).

X = -1

yes
| ?-
```

Gunakan huruf kapital sebagai variabel di terminal!

3. FACT

Fakta adalah pernyataan yang diasumsikan benar. Gunakan huruf kecil untuk fact (misal positive(1). Bukan Positive(1)).

Contoh code :

```
positive(1).
```

Penjelasan :

1 adalah positive.

4. RULES

Di Prolog, **rules** (atau aturan) digunakan buat mengambil kesimpulan dari satu atau lebih kondisi. Format dasarnya seperti ini :

```
kesimpulan :- kondisi
```

Maksudnya, kesimpulan bisa dianggap benar kalau semua kondisi di sebelah kanan terpenuhi.

4.1.Basic Rules

Sederhananya basic rules itu aturan dasar yang menghubungkan fakta-fakta secara langsung tanpa perlu rekursi atau logika yang rumit.

Contoh Rule :

```
ibu(ana, budi).
orang_tua(X, Y) :- ibu(X, Y).
```

Penjelasan :

Cukup simple, rule ini menyatakan bahwa X adalah orang tua Y jika X adalah ibu Y. Jadi, karena *ana* adalah ibu *budi*, maka *ana* otomatis dianggap sebagai orang tua dari *budi*. Kalau dijalankan di prolog :

```
?- orang_tua(ana, budi).
```

Outputnya harusnya `yes.`

Contoh Rule :

```
hewan(kucing).

memelihara(ali, X) :- hewan(X).
```

Penjelasan :

Rule ini menyatakan bahwa Ali memelihara X jika X adalah hewan. Karena kucing adalah hewan, maka bisa disimpulkan bahwa Ali memelihara kucing.

Contoh Rule :

```
teman(budi, cici).

sahabat(X, Y) :- teman(X, Y).
```

Penjelasan :

Cukup mudah dipahami.

4.2.Recursive

Rekursi adalah konsep di mana predikat memanggil dirinya sendiri untuk menyelesaikan masalah hingga mencapai basis rekursi (kondisi penghentian). Biasanya digunakan untuk memproses list atau perhitungan berulang.

Contoh code : Menghitung Panjang List

```
list_length([], 0).

list_length([_|Tail], Length) :-

    list_length(Tail, TailLength), Length is TailLength + 1.
```

Penjelasan :

Misal, kita ingin menghitung panjang sebuah list. Jika list kosong ([]), panjangnya sudah pasti 0 ini adalah **basis rekursi**. Namun, jika list memiliki elemen, kita abaikan elemen pertama (head) dan fokus menghitung panjang elemen sisanya (tail). Proses ini dilakukan secara rekursif hingga list menjadi kosong. Setiap kali kembali dari rekursi, kita tambahkan 1 ke hasil panjang tail untuk memperhitungkan elemen pertama yang diabaikan sebelumnya.

Contoh cara menjalankan di Query:

```
| ?- list_length([a, b, c], X).

X = 3

yes
```

Contoh code : Mencari Elemen Terakhir List

```
last_element([X], X).

last_element([_|Tail], Last) :-

    last_element(Tail, Last).
```

Penjelasan :

Tujuan kita adalah menemukan elemen terakhir dalam list. Jika list hanya memiliki satu elemen, maka itulah elemen terakhirnya, ini adalah **basis rekursi**. Jika list memiliki lebih dari satu elemen, kita abaikan elemen pertama (head) dan fokus mencari elemen terakhir di bagian sisanya (tail). Proses ini dilakukan berulang kali hingga list menyisakan satu elemen yang kemudian dianggap sebagai elemen terakhir

Contoh cara menjalankan di Query:

```
| ?- last_element([1, 2, 3, 4], X).

X = 4 ?

yes
```

Contoh code : Menjumlahkan Elemen List

```
jumlah_list([], 0).

jumlah_list([Head|Tail], Sum) :-
    jumlah_list(Tail, TailSum),
    Sum is Head + TailSum.
```

Penjelasan :

Di sini, kita ingin menjumlahkan semua elemen dalam list. Jika list kosong ([]), hasilnya adalah 0 , ini adalah **basis rekursi**. Jika list memiliki elemen, kita ambil elemen pertama (head), lalu hitung jumlah elemen sisanya (tail) secara rekursif. Setelah proses rekursif selesai, kita tambahkan elemen pertama yang sebelumnya diabaikan ke hasil jumlah tail.

Contoh cara menjalankan di Query:

```
| ?- jumlah_list([1, 2, 3, 4], X).

X = 10

yes
```

4.3.Array

Tidak seperti bahasa pemrograman lainnya, Prolog tidak punya struktur array bawaan, tetapi kita dapat merepresentasikan array menggunakan *list*. *List* di Prolog digunakan untuk menyimpan elemen-elemen berurutan dan dapat diakses melalui pencocokan pola (*pattern matching*). List didefinisikan dengan menempatkan elemen di dalam tanda kurung siku dan dipisahkan oleh koma.

Dalam Prolog, array terdiri dari elemen-elemen yang dapat dipecah menjadi dua bagian: *head* dan *tail*. Konsep ini adalah dasar untuk memanipulasi dan menavigasi array di Prolog yang dinotasikan dengan `[H|T]` dengan `H` (*Head*) adalah elemen pertama array dan `T` (*Tail*) adalah *list* yang berisi semua elemen setelah *head*.

Contoh code : Deklarasi Array

```
empty_array([]).

array_with_element([1,2,3,4,5]).
```

```
mixed_array([1, 'hello', 3.14, atom, [nested, list]]).
```

Penjelasan :

Dalam Prolog, tentunya array juga didefinisikan dalam sebuah fakta yang menerima sebuah array (dalam bentuk *list*) sebagai argumen.

array_with_element mendefinisikan fakta array dengan elemen [1,2,3,4,5]. Dalam bahasa pemrograman Python, ini sama seperti kita meng-*assign* [1,2,3,4,5] ke array_with_element. Array dalam Prolog juga bisa diisi dengan lebih dari satu tipe data atau bahkan bisa berisi array lain.

Berikut beberapa operasi dasar array yang bisa kita lakukan di Prolog:

Contoh code : GET INDEX

```
get_index([_|Tail, Index, Element):-
    Index > 0,
    NewIndex is index - 1,
    get_index(Tail, NewIndex, Element).
get_index([Element|_], 0, Element).
```

Penjelasan :

Kode ini digunakan untuk mengambil elemen dari array pada indeks tertentu. `get_index` menerima sebuah array, indeks yang diinginkan, dan elemen yang akan diambil. Jika indeks lebih besar dari 0, array diproses secara rekursif hingga mencapai elemen pada indeks tersebut.

Contoh code : GET ELEMENT

```
get_element([Element|_], 0, Element).
get_element([_|Tail], Index, Element) :-
    Index > 0,
    NewIndex is Index - 1,
    get_element(Tail, NewIndex, Element).
```

Penjelasan :

Mirip dengan contoh sebelumnya, kode ini mengakses elemen berdasarkan indeks yang diberikan. Pencocokan dilakukan dengan menurunkan indeks secara rekursif sampai indeks menjadi nol, lalu elemen pada indeks tersebut diambil.

Contoh code : APPEND ELEMENT

```
append_element(List, Element, NewList) :-
    append(List, [Element], NewList.
```

Penjelasan :

Kode ini digunakan untuk menambahkan elemen ke akhir array. `append` adalah bawaan dari Prolog yang menggabungkan dua array. Dalam kasus ini, elemen baru dimasukkan ke dalam array asli, menghasilkan array baru dengan elemen tersebut di akhir.

Contoh code : LENGTH

```
get_length(List, Length) :-
    length(List, Length).
```

Penjelasan :

`length` menghitung jumlah elemen dalam daftar dan mengembalikannya sebagai panjang. Ini berguna untuk mengetahui ukuran array (list).

Contoh code : UPDATE ELEMENT

```
update_element([_|Tail], 0, NewElement, [NewElement|Tail]).

update_element([Head|Tail], Index, NewElement,
[Head|UpdatedTail]) :-
    Index > 0,
    NewIndex is Index - 1,
    update_element(Tail, NewIndex, NewElement, UpdatedTail).
```

Penjelasan :

`update_element` memperbarui elemen dalam array. Jika indeks adalah 0, elemen baru menggantikan elemen lama. Jika tidak, array diproses secara rekursif sampai mencapai indeks yang diinginkan.

Contoh code : DELETE ELEMENT

```
delete_element([_|Tail], 0, Tail).

delete_element([Head|Tail], Index, [Head|UpdatedTail]) :-
    Index > 0,
    NewIndex is Index - 1,
    delete_element(Tail, NewIndex, UpdatedTail).
```

Penjelasan :

`delete_element` menghapus elemen dari array. Jika indeks adalah 0, elemen dihapus, dan sisa daftar dikembalikan. Jika tidak, array diproses secara rekursif hingga indeks tercapai.

Contoh code : REVERSE

```
reverse_list(List, Reversed) :-
    reverse(List, Reversed).
```

Penjelasan :

`reverse` membalik urutan elemen dalam daftar. Berguna jika Anda perlu mengubah urutan array.

5. DYNAMIC PREDICATE

Dynamic predicate dalam Prolog adalah **predikat yang dapat dimodifikasi** selama program dijalankan. Dengan menggunakan dynamic predicates, kita dapat menambah, mengubah, atau menghapus data dari basis fakta tanpa perlu menghentikan eksekusi program.

Note: Pada mata kuliah IF1221, dynamic predicate yang digunakan hanya beberapa saja, untuk sisanya silahkan di-*explore* sendiri.

5.1.Inisiasi

Untuk menginisialisasi dynamic predicate dalam Prolog, pemanggilan harus dilakukan dengan memanggil `:- dynamic nama_predikat/jumlah arity`.

Contoh code :

```
:- dynamic(orang/1). %orang(Nama)
:- dynamic(orangumur/2). %orang(Nama, Umur)
```

Penjelasan :

`orang/1`: Predicate ini hanya memiliki satu parameter (arity 1) dan digunakan untuk menyimpan informasi mengenai nama orang di dalam basis fakta. Contoh `orang(panji).`

`orangumur/2`: Predicate ini memiliki dua parameter (arity 2) dan digunakan untuk menyimpan informasi mengenai nama dan umur orang dalam satu fakta. Contoh `orangumur(panji, 20).`

5.2.Asserta

Asserta digunakan untuk menambahkan fakta atau aturan ke awal basis fakta.

Contoh code :

```
asserta(orang(eriq)).
```

Penjelasan :

Dalam hal ini, eriq akan menjadi fakta yang muncul di awal daftar orang.

5.3.Assertz

Assertz digunakan untuk menambahkan fakta atau aturan ke akhir basis fakta.

Contoh code :

```
assertz(orang(barok)).
```

Penjelasan :

Dalam hal ini, barok akan ditambahkan sebagai fakta di akhir daftar orang.

5.4.Retract

Retract digunakan untuk menghapus fakta atau aturan yang ada dari basis fakta.

Contoh code :

```
retract(orang(eriq)).
```

Penjelasan :

Jika eriq ada dalam daftar orang, maka fakta tersebut akan dihapus dari basis fakta.

5.5.Retractall

Retractall digunakan untuk menghapus semua instans dari suatu fakta atau aturan yang ada dalam basis fakta.

Contoh code :

```
retractall(orang(_)).
```

Penjelasan :

Dalam hal ini, perintah ini akan menghapus semua fakta orang dari basis fakta, tanpa mempertimbangkan siapa pun yang terdaftar.

5.6.Contoh Program

Untuk memahami dynamic predicate lebih lanjut, diberikan contoh program yang menggunakan dynamic predicate.

Contoh code :

```
% Mendefinisikan dynamic predicate
:- dynamic(orang/1).


% Fakta awal
orang(qika).
orang(barok).


% Menampilkan semua orang
daftar_orang :-
    findall(Nama, orang(Nama), Daftar),
    (
        Daftar = [] -> write('Tidak ada orang di daftar.'),
nl;
        write('Daftar Orang: '), write(Daftar), nl
    ).


% Menambahkan orang ke akhir daftar
tambah_orang_z(Nama) :-
    assertz(orang(Nama)),
    write(Nama), write(' telah ditambahkan ke daftar dengan
assertz.'), nl.


% Menambahkan orang ke awal daftar
tambah_orang_a(Nama) :-
```

```
    asserta(orang(Nama)),
    write(Nama), write(' telah ditambahkan ke daftar dengan
asserta.'), nl.


% Menghapus orang dari daftar
hapus_orang(Nama) :-
    retract(orang(Nama)),
    write(Nama), write(' telah dihapus dari daftar.'), nl.


% Menghapus semua orang
hapus_semua_orang :-
    retractall(orang(_)),
    write('Semua orang telah dihapus.'), nl.
```

Penjelasan :

Silahkan

```
daftar_orang.
tambah_orang_z(eve).
daftar_orang.
tambah_orang_a(francesco).
daftar_orang.
hapus_orang(panji).
daftar_orang.
hapus_orang(francesco).
daftar_orang.
hapus_semua_orang.
daftar_orang.
```

dijalankan dan amati yang terjadi.

6. BASIC

6.1.Write

Basically ini kayak `print(foo)` di python

Contoh code :

```
write('Hello').         % Prints without newline

nl.                     % Prints just a newline
```

Penjelasan :

Keknya ga perlu penjelasan lah ya

Atau kalau mau pake formatted print bisa juga

Contoh code :

```
% Basic format string with placeholder ~w

format('~w~n', ['Ohayou Sekai Good Morning World!']).
```

Output:

```
| ?- format('~w~n', ['Ohayou Sekai Good Morning World!']).
Ohayou Sekai Good Morning World!



% Multiple placeholders

format('~w is ~w years old~n', ['Mas Amba', 25]).
```

Output:

```
| ?- format('~w is ~w years old~n', ['Mas Amba', 25]).
Mas Amba is 25 years old
```

REMINDER: Variable akan di print sesuai urutan

```
format('~w dan ~w adalah waifu gwehj~n', ['Makise
Kurisu','Hatsune Miku' ]).
```

Output:

```
| ?- format('~w dan ~w adalah waifu gwehj~n', ['Makise Kurisu','Hatsune Miku' ]).
Makise Kurisu dan Hatsune Miku adalah waifu gwehj
```

Jika urutannya ditukar:

```
| ?- format('~w dan ~w adalah waifu gwehj~n', ['Hatsune Miku','Makise Kurisu' ]).
Hatsune Miku dan Makise Kurisu adalah waifu gwehj
```

```
% Common format specifiers:

format('Number: ~d~n', [42]).        % ~d for integers

format('Float: ~f~n', [3.14]).       % ~f for floats
```

Penjelasan :

Perhatikan *variable* apa yang mau di print, cek tipenya apa. Kalau *integer*, pakai *placeholder* `~d`, kalau *string* pake `~w`, kalau *float* pake `~f`, kalau mau kasih *newline* pake `~n`, and jangan lupa urutan *variable* dalam array yang akan di print itu berpengaruh.

6.2.Read

Buat baca input dari user, kita bisa make `read/1`

Contoh code :

```
getName :-

    write('What is your name?'),

    read(Name),

    format('Ohayou ~w, nice to meet you!~n', [Name]).
```

Penjelasan :

Ya kira-kira gitulah ya

6.3.Assign {=, is}

Ada dua operator assignment di Prolog, yaitu = dan `is`. Perbedaannya = digunakan untuk *assignment* biasa, sedangkan `is` digunakan untuk perhitungan aritmatika dan RHS (right-hand side)-nya harus ada angka.

Contoh code :

```
X = 5.                  % X gets value 5
[Head|Tail] = [1,2,3] % Head gets 1, Tail gets [2,3]


X is 2 + 3.           % X gets 5
Y is X * 2.           % Y gets 10


X is Y.          % Error if Y isn't bound to a number
X = 2 + 3.       % Will unify X with the expression '2+3', not
5
```

Penjelasan :

Sederhananya, `is` untuk *assignment* yang melibatkan angka, sedangkan sisanya pake = aja.

```
| ?- X = 2+5.


X = 2+5


| ?- X is 2+5.


X = 5
```

6.4.Repeat

Di prolog sendiri sebenarnya ada *looping* kayak *while loop* di bahasa pemrograman lainnya. Kita make `repeat` *instead of* `while`.

Contoh code :

```
simple_loop :-
    repeat, % Repeat the following block
    write('Enter a number (0 to exit): '),
    read(X),
    (X = 0 -> !           % If X is 0, cut (stop backtracking)
    ;  write('You entered: '), write(X), nl,
       fail              % Force backtrack to repeat
    ).
```

Namun, terkadang menggunakan rekursif biasa jauh lebih *clean*:

```
simple_loop :-
   write('Enter number (0 to exit): '),
   read(X),
   (X = 0 -> true                % If X=0, just succeed
(true)
   ; write('Got: '), write(X), nl,% Else print and recurse
   simple_loop).
```

Penjelasan :

Kalau make `repeat`, jangan lupa kasih `fail`, kalau ga ada `fail` ga bakalan nge-*loop*, sama jangan lupa kasih `cut`(!) biar ga *infinite loop.*

6.5.Condition (IF ELSE)

Ada dua cara untuk menggunakan percabangan di Prolog, yaitu dengan -> atau dengan menggunakan *multiple rules.*

Contoh code :

```
% Basic structure: (if -> then ; else)
grade(Score) :-
    (Score >= 90 -> write('A')
    ; Score >= 80 -> write('B')
    ; Score >= 70 -> write('C')
    ; Score >= 60 -> write('D')
    ;                write('F')
    ).

% Using rule
grade2(Score) :- Score >= 90, write('A').
grade2(Score) :- Score >= 80, Score < 90, write('B').
grade2(Score) :- Score >= 70, Score < 80, write('C').
grade2(Score) :- Score >= 60, Score < 70, write('D').
grade2(Score) :- Score < 60, write('F').

% Using cut to make if-else mutually exclusive
categorize(Age) :-
    Age >= 18, !,        % Cut prevents backtracking
    write('Adult').
```

```
categorize(Age) :-

    write('Minor').
```

Penjelasan :

- `->` jauh lebih *compact,*
- *Multiple rules* membutuhkan *explicit range* yang jelas, pada contoh terakhir jika tanpa `cut`:

```
| ?- categorize(25).
Adult

true ? ;
Minor

yes
```

Dengan menggunakan `cut`:

```
| ?- categorize(25).
Adult

yes
```

Pada contoh kedua, kita tidak perlu menggunakan `cut` karena sudah ada *range of value* yang jelas dan tiap input hanya akan sesuai dengan salah satu *rule* saja sehingga tidak mungkin terjadi backtracking.

- `->` dan `cut` mencegah *backtracking*, tetapi penggunaan *multiple rules* tanpa `cut` atau tanpa *range* yang jelas bisa menyebabkan *backtracking*.

7. INCLUDE (IMPORT)

Include pada GNU Prolog memiliki kegunaan yang mirip dengan import pada Python. Namun, tidak seperti import yang bisa memuat isi spesifik dari suatu file, include memuat semua isi dari file yang dimuat seakan-akan kode file ditulis di file yang memuat.

Syntax:

```
:- include('filename.pl').
```

Contoh :

Kode di File Square.pl:

```
square(X, Y) :- Y is X * X.
```

Kode di main.pl:

```
:- include('utilities.pl').

main :-
    square(4, Result),
    write('The square of 4 is: '), write(Result), nl.
```

Penjelasan :

Misal, kita ingin menggunakan fungsi **square/2** yang ada di file **Square.pl** di **main.pl**. Untuk itu, kita bisa menggunakan **include/1** yang berfungsi untuk menyertakan kode dari file **Square.pl** ke dalam file **main.pl**. Dengan **:- include('Square.pl')**., semua definisi predikat dari file **Square.pl** akan dimasukkan dan dikenali dalam **main.pl**. Sehingga, ketika kita memanggil square(4, Result) di main.pl, Prolog akan mencari definisi **square/2** di file yang telah disertakan. Pada contoh ini, **main/0** akan mencetak hasil kuadrat dari 4, yaitu 16.

8. LAIN-LAIN

8.1.Findall

Format

findall(X, Condition, Result)

X: Variabel yang mewakili elemen-elemen solusi yang ingin dikumpulkan.

Condition: Kondisi atau tujuan yang harus dipenuhi agar `X` dianggap sebagai solusi yang valid.

Jika kondisi ini bernilai `true`, maka nilai dari `X` akan dikumpulkan.

Result: List yang berisi semua elemen `X` yang sesuai dengan `Condition`.

Cara Kerja

1. `findall(X, Condition, Result)` bekerja dengan cara mencari semua kemungkinan nilai `X` yang memenuhi `Condition` dan mengembalikan hasilnya dalam `Result` berupa sebuah list. Jika tidak ada nilai `X` yang memenuhi `Condition`, `Result` akan berupa list kosong `[]`.
2. Urutan nilai pada `Result` terurut berdasarkan solusi yang ditemukan duluan.
3. Semua solusi duplikat akan dimasukkan pada `Result.`
4. Jika solusi berjumlah tidak terbatas, eksekusi tidak akan berhenti.

Contoh code :

```
?- findall(X, member(X, [1, 2, 3, 4, 5]), Result).
  Result = [1, 2, 3, 4, 5]?
  yes
```

Penjelasan :

Pada contoh ini, `findall/3` mengumpulkan semua elemen `X` yang menjadi anggota dari list `[1, 2, 3, 4, 5]`, dan hasilnya adalah `Result = [1, 2, 3, 4, 5]`.

Contoh code :

```
?- findall(X, (member(X, [1, 2, 3, 4, 5]), X > 3), Result).
  Result = [4, 5]?
  yes
```

Penjelasan :

Di sini, `findall/3` mengumpulkan semua nilai `X` dari list `[1, 2, 3, 4, 5]` yang memenuhi kondisi `X > 3`.

Contoh code :

```
?- findall(X/Y, (member(X, [1, 2, 3, 4, 5]), Y is X * X),
```

```
Result).
  Result = [1/1, 2/4, 3/9, 4/16]?
  yes
```

8.2.Random

Dalam Prolog, untuk mendapatkan nilai acak dapat dilakukan menggunakan *library* seperti `random`. Dengan fungsi seperti `random/1` atau `random/3`, kita dapat membuat solusi yang membutuhkan nilai acak dalam batas tertentu.

random(X).

X adalah variabel yang mewakili elemen solusi yang mungkin didapat. `random/1` bekerja dengan cara membangkitkan angka acak di antara 0 dan 1.

random(Batas Bawah, Batas Atas, Z).

Z adalah variabel yang mewakili elemen solusi yang mungkin didapat. Hasil eksekusi dari kueri ini akan menghasilkan sebuah angka integer yang nilainya berada di antara **Batas Bawah** dan **Batas Atas**.

Contoh code :

```
| ?- random(X).
X = 0.24288677051663399
yes
```

Penjelasan :

Dalam contoh ini, nilai X yang dihasilkan adalah 0.24288677051663399. Angka ini adalah hasil acak yang dipilih dari rentang 0.0 hingga kurang dari 1.0. Nilai X tidak akan pernah mencapai 1.0 , tetapi mendekati 1.0.

Contoh code :

```
| ?- random(0,10,Z).


Z = 1


yes
```

Penjelasan :

Dalam contoh ini, nilai Y yang dihasilkan adalah 1. Angka ini adalah hasil integer acak yang dipilih dari rentang 0 yang merupakan batas bawah dan 10 yang merupakan batas atas.

REFERENSI

Diaz, D. (n.d.). The GNU Prolog web site. http://www.gprolog.org/