BAGIAN I: Fakta, Rule, dan Query

1. Fakta

a. pria(X) : X adalah pria
b. wanita(X) : X adalah wanita
c. usia(X,Y) : X berusia Y
d. menikah(X,Y) : X menikah dengan Y
e. anak(X,Y) : X adalah anak Y

```
/* pria(X)      : X adalah pria*/
pria(panji).
pria(qika).
pria(barok).
pria(francesco).
pria(eriq).
pria(shelby).
pria(rifia).
pria(aqua).
pria(mauang).
pria(jamet).

/*wanita(X)    : X adalah wanita*/
wanita(frieren).
wanita(mary).
wanita(myers).
wanita(yennefer).
wanita(gizelle).
wanita(rhiannon).
wanita(eve).
wanita(roxy).
wanita(ruby).
wanita(aihashino).
wanita(suzy).

/*usia(X,Y)    : X berusia Y*/
usia(panji, 78).
usia(frieren, 75).
usia(mary, 51).
usia(qika, 55).
usia(myers, 40).
usia(shelby, 25).
usia(yennefer, 30).
usia(rifia, 35).
usia(gizelle, 20).
usia(rhiannon, 5).
```

```
usia(eve, 10).
usia(barok, 75).
usia(roxy, 73).
usia(aqua, 53).
usia(ruby, 48).
usia(eriq, 50).
usia(aihashino, 32).
usia(mauang, 15).
usia(francesco, 23).
usia(suzy, 23).
usia(jamet, 3).

/*menikah(X,Y) : X menikah dengan Y*/
menikah(panji, frieren).
menikah(frieren, panji).
menikah(qika, mary).
menikah(mary, qika).
menikah(qika, myers).
menikah(myers, qika).
menikah(rifia, yennefer).
menikah(yennefer, rifia).
menikah(rifia, aihashino).
menikah(aihashino, rifia).
menikah(barok, roxy).
menikah(roxy, barok).
menikah(eriq, ruby).
menikah(ruby, eriq).
menikah(francesco, suzy).
menikah(suzy, francesco).

/*anak(X,Y)    : X adalah anak Y*/
anak(qika, panji).
anak(qika, frieren).
anak(shelby, qika).
anak(yennefer, qika).
anak(gizelle, qika).
anak(shelby, mary).
anak(yennefer, mary).
anak(gizelle, myers).
anak(rhiannon, shelby).
anak(eve, rifia).
anak(eve, yennefer).
anak(aqua, barok).
anak(ruby, barok).
anak(aqua, roxy).
anak(ruby, roxy).
```

```
anak(aihashino, aqua).
anak(francesco, eriq).
anak(francesco, ruby).
anak(mauang, rifia).
anak(mauang, aihashino).
anak(jamet, francesco).
anak(jamet, suzy).
```

2. Rule

a. saudara(X,Y) : X adalah saudara kandung maupun saudara tiri dari Y
b. saudaratiri(X,Y) : X adalah saudara tiri dari Y
c. kakak(X,Y) : X adalah kakak dari Y
d. keponakan(X,Y) : X adalah keponakan dari Y
e. mertua(X,Y) : X adalah mertua dari Y
f. nenek(X,Y) : X adalah nenek dari Y
g. keturunan(X,Y) : X adalah keturunan dari Y (anak, cucu, dan seterusnya)
h. lajang(X) : X adalah orang yang tidak menikah
i. anakbungsu(X) : X adalah anak paling muda
j. anaksulung(X) : X adalah anak paling tua
k. yatimpiatu(X) : X adalah orang yang orang tuanya tidak terdefinisi

```
/*saudara(X,Y)        : X adalah saudara kandung maupun tiri dari
Y*/
saudara(X, Y) :-
    anak(X, Z),
    anak(Y, Q),
    Z == Q,
    X \== Y.

/*saudaratiri(X,Y)    : X adalah saudara tiri dari Y*/
saudaratiri(X,Y) :-
    anak(X, Z),
    anak(Y, Q),
    X \== Y,
    wanita(Z),
    wanita(Q),
    Z \== Q,
    menikah(Z, P),
    menikah(P, Q).

/*kakak(X,Y)          : X adalah kakak dari Y (kakak kandung maupun
tiri)*/
kakak(X, Y) :-
    saudara(X, Y),
```

```
    usia(X, A),
    usia(Y, B),
    A > B.

/*keponakan(X,Y)      : X adalah keponakan dari Y*/
keponakan(X,Y) :-
    saudara(Y, Z),
    anak(X, Z).

/*mertua(X,Y)         : X adalah mertua dari Y*/
mertua(X, Y) :-
    anak(Z, X),
    menikah(Z, Y).

/*nenek(X,Y)              : X adalah nenek dari Y*/
nenek(X, Y) :-
    anak(Z, X),
    anak(Y, Z),
    wanita(X).

/*keturunan(X,Y)          : X adalah keturunan dari Y (anak, cucu,
dan seterusnya)*/
keturunan(X, Y) :-
    anak(X, Y).
keturunan(X, Y) :-
    anak(X, Z),
    keturunan(Z, Y).

/*lajang(X)                   : X adalah orang yang tidak menikah*/
lajang(X) :-
    pria(X),
    \+ menikah(X, _).
lajang(X) :-
    wanita(X),
    \+ menikah(X, _).

/*anakbungsu(X)           : X adalah anak paling muda*/
anakbungsu(X) :-
    anak(X, _Parent),
    anak(_Y, _Parent),
    _Y \= X,
    \+ (anak(_Z, _Parent), _Z \= X, usia(_Z, _AgeZ), usia(X,
_AgeX), _AgeZ < _AgeX).

/*anaksulung(X)           : X adalah anak paling tua*/
anaksulung(X) :-
    anak(X, _Parent),
    anak(_Y, _Parent),
    _Y \= X,
    \+ (anak(_Z, _Parent), _Z \= X, usia(_Z, _AgeZ), usia(X,
_AgeX), _AgeZ > _AgeX).
```

```
/*yatimpiatu(X)           : X adalah orang yang orang tuanya tidak
terdefinisi*/
yatimpiatu(X) :-
    usia(X,_),
    \+ anak(X, _).
```

3. Query

a. Ayah dari Rhiannon

```
| ?- pria(Ayah), anak(rhiannon, Ayah).
Ayah = shelby ?
yes
```

b. Kakek buyut dan nenek buyut dari Mauang

```
| ?- nenek(NenekBuyut, _Y), anak(mauang, _Y), menikah(NenekBuyut,
KakekBuyut).
KakekBuyut = barok
NenekBuyut = roxy ?
yes
```

c. Paman dari orang tua Jamet

```
| ?- keponakan(_Y, Paman), anak(jamet, _Y).
Paman = aqua ?
```

d. Orang yang memiliki lebih dari satu istri

```
| ?- menikah(_Pasangan1, Orang), menikah(Orang, _Pasangan2),
_Pasangan1 \= _Pasangan2, wanita(_Pasangan1), wanita(_Pasangan2),
pria(Orang).
Orang = qika ? ;
Orang = rifia ?
yes
```

e. Anak paling muda dari Qika

```
| ?- anakbungsu(Bungsu), anak(Bungsu, qika).
Bungsu = gizelle ?
yes
```

f. Orang tua yang memiliki tepat satu anak

```
| ?- anak(_X, Y), \+ (saudara(_X, _Z), anak(_Z, Y)).
Y = panji ? ;
Y = frieren ? ;
Y = myers ? ;
Y = shelby ? ;
Y = yennefer ? ;
Y = aqua ? ;
Y = eriq ? ;
Y = ruby ? ;
Y = aihashino ? ;
Y = francesco ? ;
Y = suzy
yes
```

g. Pria yang memiliki mertua berusia lebih dari 50 tahun

```
| ?- mertua(_X, Y), pria(Y), usia(_X, _Z), _Z > 50.
Y = rifia ? ;
Y = eriq ?
yes
```

h. Orang yatim piatu dengan usia termuda

```
| ?- yatimpiatu(X), usia(X, Usia), \+ (yatimpiatu(Y), Y \= X,
usia(Y, UsiaY), UsiaY < Usia).
Usia = 23
X = suzy ?
yes
```

i. Orang diatas 20 tahun yang statusnya menikah dengan anak sulung

```
| ?- usia(X, _Y), _Y > 20, menikah(X, _Z), anaksulung(_Z).
X = rifia ?
yes
```

j. Sepupu dari saudara tiri anak bungsu Aihoshino

```
/* Cukup gunakan predikat anak saja karena Aihoshino hanya memiliki
1 anak */
| ?- anak(_Anak, aihashino), saudaratiri(_SaudaraTiri, _Anak),
anak(_SaudaraTiri, _Ortu), keponakan(X, _Ortu).
X = rhiannon ?
yes
```

BAGIAN II : Rekurens

1. Exponent

exponent(A, B, X)

```
exponent(_, 0, 1).

exponent(A, 1, A).

/* Rekursi */
exponent(A,B, Result):-
    B > 0,
    B1 is B - 1,
    exponent(A, B1, Result1),
    Result is A * Result1.
```

2. Growth

growth(I, G, H, T, X)

```
is_prime(2).
is_prime(3).
is_prime(N) :-
    N > 3,
    N mod 2 =\= 0,
    \+ has_factor(N, 3).

has_factor(N, F) :-
    N mod F =:= 0.
has_factor(N, F) :-
    F * F < N,
    F2 is F + 2,
    has_factor(N, F2).
```

```
growth(I, G, H, 0, I).

growth(I, G, H, T, X) :-
    T > 0,
    T1 is T - 1,
    growth(I, G, H, T1, PrevX),
    ( is_prime(T) -> X is PrevX + G ; X is PrevX - H ).
```

3. Si Imut Anak Nakal

harvestFruits(N, Fruits, TreeNumber, FinalFruits)

```
harvestFruits(N, Fruits, TreeNumber, FinalFruits) :-
    TreeNumber =:= N + 1,
    FinalFruits is Fruits.

harvestFruits(_, Fruits, _, FinalFruits) :-
    Fruits =< 0,
    write('Si Imut pulang sambil menangis :('), nl,
    FinalFruits is 0.

harvestFruits(N, Fruits, TreeNumber, FinalFruits) :-
    Fruits > 0,
    fruitEffect(TreeNumber, Effect),
    NewFruits is Fruits + Effect,
    NextTree is TreeNumber + 1,
    harvestFruits(N, NewFruits, NextTree, FinalFruits).

fruitEffect(TreeNumber, Effect) :-
    (TreeNumber mod 3 =:= 0 -> Effect1 is 2; Effect1 is 0),
    (TreeNumber mod 4 =:= 0 -> Effect2 is -5; Effect2 is 0),
    (TreeNumber mod 5 =:= 0 -> Effect3 is 3; Effect3 is 0),
    (is_prime(TreeNumber) -> Effect4 is -10; Effect4 is 0),
    Effect is Effect1 + Effect2 + Effect3 + Effect4.
```

4. KPK

KPK

```
mod(A, B, X) :-
    A >= 0,
    B > 0,
    A < B,
    X is A.
mod(A, B, X) :-
```

```
    A >= 0,
    B > 0,
    A >= B,
    A1 is A - B,
    mod(A1, B, X).

gcd(A, 0, A) :-
    A >= 0, !.

gcd(A, B, X) :-
    B > 0,
    mod(A, B, R),
    gcd(B, R, X).

kpk(A, B, X) :-
    gcd(A, B, GCD),
    (
        GCD = 0 -> X = 0;
        X is (A * B) / GCD
    ).
```

5. Factorial

factorial(N, X)

```
factorial(0, 1).
factorial(N, X) :-
    N > 0,
    N1 is N - 1,
    factorial(N1, X1),
    X is N * X1.
```

6. Make Pattern

makePattern(N)

```
makePattern(N) :-
    generate_square(N, 1, N).

generate_square(N, CurrentDepth, MaxDepth) :-
    CurrentDepth =< MaxDepth, !,
    print_line(N, CurrentDepth, MaxDepth),
    nl,
    NextDepth is CurrentDepth + 1,
```

```
    generate_square(N, NextDepth, MaxDepth).

print_line(N, CurrentDepth, MaxDepth) :-
    print_element(N, CurrentDepth, MaxDepth, 1).

print_element(N, _, _, Counter) :-
    Counter > N, !.
print_element(N, CurrentDepth, MaxDepth, Counter) :-
          MinDepth   is   min(CurrentDepth,   min(MaxDepth   -
CurrentDepth + 1, min(Counter, N - Counter + 1))),
    write(MinDepth),
    write(' '),
    NextCounter is Counter + 1,
    print_element(N, CurrentDepth, MaxDepth, NextCounter).
```

BAGIAN III : List

1. List Statistic

min(List, Min)

```
/* Basis */
min([Head|Tail], Head) :-
    Tail = [], !.

/* Rekurens */
min([Head|Tail], Min) :-
    min(Tail, CurrMin),
    Head < CurrMin, !,
    Min is Head.

min([Head|Tail], Min) :-
    min(Tail, CurrMin),
    Head >= CurrMin, !,
    Min is CurrMin.
```

max(List, Max)

```
/* Basis */
max([Head|Tail], Head) :-
    Tail = [], !.

/* Rekurens */
max([Head|Tail], Max) :-
    max(Tail, CurrMax),
    Head > CurrMax, !,
    Max is Head.

max([Head|Tail], Max) :-
    max(Tail, CurrMax),
    Head =< CurrMax, !,
    Max is CurrMax.
```

range(List, Range)

```
range(List, Range) :-
    max(List, Max),
    min(List, Min),
    Range is Max - Min.
```

count(List, Count)

```
/* Basis */
count([], 0) :- !.

/* Rekurens */
```

```
count([Head|Tail], Count) :-
    count(Tail, CurrCount),
    Count is CurrCount + 1.
```

sum(List, Sum)

```
/* Basis */
sum([], 0) :- !.

/* Rekurens */
sum([Head|Tail], Sum) :-
    sum(Tail, CurrSum),
    Sum is CurrSum + Head.
```

2. List Manipulation

a. mergeSort

mergeSort(ListA, ListB, Result)

```
mergeSort([], ListB, ListB).
mergeSort(ListA, [], ListA).

mergeSort([HeadA | TailA], [HeadB | TailB], [HeadA |
ResultTail]) :-
    HeadA =< HeadB,
    mergeSort(TailA, [HeadB | TailB], ResultTail).

mergeSort([HeadA | TailA], [HeadB | TailB], [HeadB |
ResultTail]) :-
    HeadA > HeadB,
    mergeSort([HeadA | TailA], TailB, ResultTail).
```

b. filterArray

filterArray(List, Element1, Element2, Result)

```
filterArray([], _, _, []).

filterArray([Head|Tail], Element1, Element2,
[Head|ResultTail]) :-
    Head > Element1,
    Head mod Element2 =:= 0,
    filterArray(Tail, Element1, Element2, ResultTail).

filterArray([Head|Tail], Element1, Element2, Result) :-
    (Head =< Element1; Head mod Element2 =\= 0),
```

```
    filterArray(Tail, Element1, Element2, Result).
```

c. reverse

reverse (List, Result)

```
reverse_custom([], []).
reverse_custom(List, Result) :-
    reverse_helper(List, [], Result).

reverse_helper([], TempRes, TempRes).
reverse_helper([Head|Tail], TempRes, Result) :-
    reverse_helper(Tail, [Head|TempRes], Result).
```

d. cekPalindrom

cekPalindrom(List)

```
cekPalindrom([]).
cekPalindrom([_]).

cekPalindrom([H|T]) :-
    lastElement(T, Last, T2),
    H =:= Last,
    cekPalindrom([T2]).

lastElement([_, Y], Y, []) :- !.
lastElement([H|T], Last, [H|T2]):-
    lastElement(T, Last, T2).
```

e. rotate

rotate(List, N, Result)

```
rotate([], _, []).

rotate(List, 0, List).

rotate(List, N, Result) :-
    N > 0,
    normalize_rotation(List, N, N1),
    rotate_left(List, N1, Result).

rotate(List, N, Result) :-
    N < 0,
    length_manual(List, Length),
    N1 is Length + (N mod Length),
```

```
    rotate(List, N1, Result).

normalize_rotation(List, N, N1) :-
    length_manual(List, Length),
    N1 is N mod Length.

rotate_left([Head|Tail], N, Result) :-
    N > 0,
    append_tail(Tail, Head, Rotated),
    N1 is N - 1,
    rotate_left(Rotated, N1, Result).

rotate_left(List, 0, List).

append_tail([], X, [X]).
append_tail([H|T], X, [H|R]) :- append_tail(T, X, R).

length_manual([], 0).
length_manual([_|T], Length) :-
    length_manual(T, SubLength),
    Length is SubLength + 1.
```

f. Mapping

prosesMahasiswa(Name, Grades, Result)

```
/* Konversi nilai ke indeks */
set_index([], []).
set_index([Grade|Tail], [Index|IndexTail]) :-
    (   Grade >= 80
    ->  Index = 'A'
    ;   Grade >= 70
    ->  Index = 'B'
    ;   Grade >= 60
    ->  Index = 'C'
    ;   Grade >= 50
    ->  Index = 'D'
    ;   Index = 'E'
    ),
    set_index(Tail, IndexTail).

/* Fungsi AVG untuk mencari rata-rata nilai */
calculate_final_grade([], 0).
calculate_final_grade(List, Average) :-
    sum_of_list(List, Sum),
    calculate_length(List, Length),
    Length > 0,
    Average is Sum / Length.

/* Fungsi untuk mencari panjang array of nilai */
```

```
calculate_length([], 0).
calculate_length([_|Tail], Length) :-
    calculate_length(Tail, TailLength),
    Length is TailLength + 1.

/* Fungsi sum yang telah dibuat pada soal sebelumnya */
sum_of_list([], 0).
sum_of_list([Head|Tail], Sum) :-
    sum_of_list(Tail, TailSum),
    Sum is Head + TailSum.

/* Proses mahasiswa */
prosesMahasiswa(Name, Grades, [Name, Index, FinalGrade,
Status]) :-
    set_index(Grades, Index),
    calculate_final_grade(Grades, FinalGrade),
    (   FinalGrade >= 80
    ->  Status = 'Pass'
    ;   Status = 'Fail'
    ).
```

Bonus

```
:- dynamic(game_started/0).
:- dynamic(player_score/1).

startingScore(0).
startingMessage('Unta putih dan unta hitammu diambil oleh
pedagang karavan. Kumpulkan koin sebanyak-banyaknya !!!').
exitMessage('Terima kasih telah memainkan simulasi ini! Sampai
jumpa lagi.').
notStartedMessage('Permainan belum dimulai. Gunakan "start"
untuk memulai.').

start :-
    (   game_started
    ->  write('Permainan sudah dimulai. Gunakan "exit" untuk
keluar dan memulai ulang.'), nl
    ;   retractall(player_score(_)),
        retractall(game_started),
        startingScore(Score),
        assertz(player_score(Score)),
        assertz(game_started),
        startingMessage(Message),
        write(Message), nl
    ).

move(Direction) :-
    (   game_started
    ->  write('Kamu bergerak ke arah '), write(Direction),
write('.'), nl,
        random(0, 101, TrapChance),
        (   TrapChance < 50
        ->  write('Oh tidak! Kamu terkena jebakan!'), nl,
            update_score(-10)
        ;   write('Jalannya aman.'), nl,
            write('Kamu mendapatkan 10 koin.'), nl,
            update_score(10)
        ),
        display_status
    ;   fail
    ).

update_score(Points) :-
    retract(player_score(OldScore)),
    NewScore is OldScore + Points,
    assertz(player_score(NewScore)).
```

```
display_status :-
    (   game_started
    ->  player_score(Score),
        write('Koin Saat Ini: '), write(Score), nl
    ;   fail
    ).

exit :-
    (   game_started
    ->  exitMessage(Message),
        write(Message), nl,
        retractall(player_score(_)),
        retractall(game_started)
    ;   write('Permainan belum dimulai. Gunakan "start" untuk
memulai.'), nl,
        fail
    ).
```