Dynamic Vs. Static Predicates

- Predicate or clauses assert during the running, directly into the working memory is called **dynamic predicate**.
- This predicate can be manipulate i.e. can be asserted and removed.

The Eminent Management University

Dynamic Vs. Static Predicates

- **Predicate static** is those written in the program file and cannot be change during the running.
- Using the same clause as dynamic clause will result an error – predicate protected.
- Beside using assert's predicate family, dynamic predicate can also be created using predicate dynamic/1.

The Eminent Management University

Dynamic Vs. Static Predicates

- Example using dynamic

```
?- dynamic(toy/2).
yes
```

Or

```
?- dynamic([toy/2, like/2]).
yes
```

The Eminent Management University

Dynamic Vs. Static Predicates

- Example using assert

```
?- assert(toy(car)).
yes

?- assert(toy(bunny)).
yes

?- toy(X).
X = car;
X = bunny
yes
```


Manipulating Knowledge Base

- To add new fact or rule into the knowledge base using built-in assert's predicate family.

Predicate	Syntax	Description
assert/1	assert(Clause)	add a clause at the end of the clauses associated with its predicate name
assert/2	assert(Clause, Position)	assert the clause at the given position
asserta/1	asserta(Clause)	add a clause at the beginning of the clauses associated with its predicate name
assertz/1	assertz(Clause)	add a clause at the end of the clauses associated with its predicate name

Dynamic Vs. Static Predicates

- Simple manipulation

```
?- assert(toy(ball)).
yes

?- retract(toy(car)).
yes

?- toy(X).
X = bunny ;
X = ball
```

Program file

```
:- dynamic(toy/1).

toy(car).
toy(bunny).
```


Manipulating Knowledge Base

- Example: assert/1

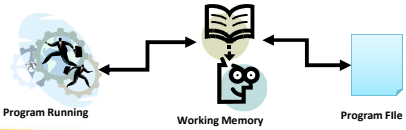
```
?- assert(toy(car)).
yes

?- assert(toy(bunny)).
yes
```

Working memory

```
toy(car).
toy(bunny).
```


Manipulating Knowledge Base

- Much of the power of Prolog comes from the ability of programs to modify themselves.
- The modification is made under working memory and the program file can also be updated.



Manipulating Knowledge Base

- Example: assert/2

```
?- assert(toy(snoopy), 0).
yes

?- assert(toy(lorry), 1).
yes

?- assert(toy(bell), 2).
yes
```

Working memory

```
toy(lorry).
toy(bell).
toy(car).
toy(bunny).
toy(snoopy).
```

Manipulating Knowledge Base

- Example: asserta/1 and assertz/1

```
?- asserta(toy(snoopy)).
yes

?- assertz(toy(lorry)).
yes
```

Working memory

```
toy(snoopy).
toy(car).
toy(bunny).
toy(lorry).
```


Manipulating Knowledge Base

- Example: retract/1

```
?- retract(toy(car)).
yes

?- retract(toy(X)).
X = snoopy;
X = bunny;
X = lorry;
yes
```

Working memory

```
toy(car).
toy(snoopy).
toy(bunny).
toy(lorry).
```


Manipulating Knowledge Base

- Example: insert new rule

```
?- assert((cucu(X,Y):- anak(X,T), anak(T,Y))).
```

*assert/1 only accept one argumen only

```
(
cucu(X,Y):-
   anak(X,T),
   anak(T,Y)
)
```

Working memory

```
cucu( A, B ) :-
  anak( A, C ),
  anak( C, B ).
```


Manipulating Knowledge Base

- Example: retract/2

```
?- retract(toy(X), 2).
X = snoopy;
X = bunny;
X = lorry;
yes
```

Working memory

```
toy(car).
toy(snoopy).
toy(bunny).
toy(lorry).
```


Manipulating Knowledge Base

- Fact or rule in the knowledge base can be remove using built-in retract's predicate family.

Predicate	Syntax	Description
retract/1	retract(Clause)	delete a clause that matches the given clause
retract/2	retract(Clause, Position)	retract a clause at a specified position
retractall/1	retractall(Head)	delete all clauses that match the given clause head

Manipulating Knowledge Base

- Example: retractall/1

```
?- retractall(toy(X)).
X = _
yes
```

Working memory

```
toy(car).
toy(snoopy).
toy(bunny).
toy(lorry).
```

Manipulating Knowledge Base

- Example: retract rule

```
?- retract((cucu(X,Y):- anak(X,T), anak(T,Y))).
yes
```

Working memory

```
cucu( A, B ) :-
  anak( A, C ),
  anak( C, B ).
```

The Eminent Management University

Manipulating Knowledge Base

- Example: abolish/1

```
?- abolish(toy/1).
yes
```

Working memory

```
toy(car).
toy(snoopy).
toy(bunny).
toy(lorry).
```

**Note that, no need to write the complete clause.*

The Eminent Management University

Manipulating Knowledge Base

- Beside retract, facts or rules can also be removed using built-in predicate abolish.
- Two version or abolish i.e. abolish/1 and abolish/2.
- Abolish delete all predicates specified by the given argument.

The Eminent Management University

Manipulating Knowledge Base

- Example – abolish more then one clauses

```
?- abolish ([toy/1,like/2]).
yes
```

Working memory

```
toy(car).
toy(snoopy).

like(ann, car).
like(sue, snoopy).

like(ahmad, car, bear).
```

**write both predicate name with its arity in a list i.e. in [].*

The Eminent Management University

Manipulating Knowledge Base

Predicate	Syntax	Description
abolish/1	abolish(Preds)	delete all predicates specified by the given argument
abolish/2	abolish(Functor, Arity)	delete the predicate specified by the given functor and arity

The Eminent Management University

Manipulating Knowledge Base

- Example: abolish/2

```
?- abolish(toy,1).
yes
```

Working memory

```
toy(car).
toy(snoopy).
toy(bunny).
toy(lorry).
```

**Note that, no need to write the complete clause.*

The Eminent Management University

Manipulating Knowledge Base

- To see the contents of the knowledge base in memory use listing/0 or listing/1.

Predicate	Syntax	Description
listing/0	listing	list all the dynamic clauses in the workspace to the current output stream
listing/1	listing(Tolist)	list the specified dynamic predicates to the current output stream

File Handling - Reading data

- Stream (i.e file name) must be an atom.
- If Stream is the name of an open input stream, it is made the current input stream.
- Otherwise the specified file is opened (for read access only) and made the current input stream.

Manipulating Knowledge Base

- Example: listing/0 and listing/1

```
?- listing.
...
...
yes

?- listing(toy/1).
    /* toy/1 */
    toy( test2 ).
    toy( test ).
```


File Handling - Reading data

- If it is not possible to open a file called Stream, an error will be generated.
- see/1 will not create a new file.
- Streams are not automatically closed. You should close them with seen/0 or close/1.

File Handling - Reading data

- Simple file operation.
- Read data from a file – see/1, seen/0
  - set the current input stream
  - Format:

```
see(Stream).      ← Causes Stream to become the current input stream.
...
...
seen.             ← Close the Stream connection.
```

Example:

```
see('data.pl').
...
...
seen.
```


File Handling - Reading data

- Example:

File data.pl

```
kedah
alor_star
```

```
?- see('data.pl').
yes

?- read(X), read(Y).
X = kedah
Y = alor_star

?- seen.
yes
```

File Handling - Writing Data

- Writing data to a file – telll/1, told/0
  - set the current output stream
  - Format:

```
tell(Stream).
...
...
told.
```

← Causes Stream to become the current output stream.

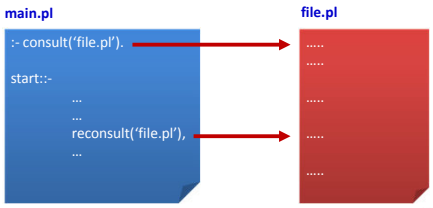
← Close the Stream connection.

Example:

```
tell('out.pl').
...
...
told.
```


Consult and Reconsult?

- When to use consult and reconsult?



File Handling - Writing Data

- Stream must be an atom.
- If Stream is the name of an open stream (i.e. a reserved stream, or a window, or an open file), then it is made the current output stream.
- Otherwise a file called Stream is created (using create/1) and made the current output stream.
- Streams are not automatically closed. You should close them with told/0 or close/1.

Using open/2

- Open file using predicate open/2.
- Format

```
open(File, Mode).
```

Mode:

```
read
write
append
```

- open/2 opens the file specified by File with the given access mode Mode.
- File can also be an atom of the form: Path/File where Path defines the path where the file File will be found.

File Handling - Writing Data

- Example:

```
?- tell('out.pl').
yes

?- write('kedah'), nl, write('alor_star').
yes

?- told.
yes
```

File out.pl

```
kedah
alor_star
```


Using open/2

- Using open/2

open('data.pl', read).	**Open 'data.pl' for reading.**
open('data.pl', write).	**Open 'data.pl' for writing.**
open('data.pl', append).	**Open 'data.pl' for adding.**

- Mode
  - 'read' – file is opened and the file pointer is set to the beginning of the file.
  - 'write' – file is created and the file pointer starts at the beginning of the file. If file is already in existence that corresponds to the file specified by File will be overwritten.
  - 'append', and the file is opened - the file pointer is set to the end of the file, if the file does not exist then 'append' mode acts like 'write' mode.

Using close/1

- Close file with close/1.
- Format:

close(File).

Example:

close('file.pl').

- If File refers to the name of an open file, the file buffer is written to disk and the file closed.

The Eminent Management University

Setting Input Stream

- Using input/1 - set input from a file, device or string
- Format

input(Stream).

- Code for input/1:

Console	0
DOS	1
Terminal	2
File	*File Name*

The Eminent Management University

Program that learn

- Example - the usage dynamic predicate, consult and reconsult.
- Program that learn – modifies its own knowledge base.

The Eminent Management University

Setting Input Stream

- Examples:

```
?- input(0).          ← set to read input from console
yes
?- input(1).          ← set to read input from DOS
yes
?- input(2).          ← set to read input from terminal
yes
?- input('data.pl')   ← set to read input from 'data.pl'
yes
```

The Eminent Management University

Program that learn

```
:- dynamic(capital_of/2).
:- consult('statefacts.pl').

go:-
        write('Enter the state name'), nl,
        read(State),
        (capital_of(City,State);ask(State,City)),
        write('Its capital is: '),  write(City),  nl.

ask(State,X):-
        write('Not in knowledge base!'),  nl,
        write('What is the capital of '), write(State),
        write('?'),  read(X),  nl,
        assert(capital_of(X,State)),
        write('Thank you... my knowledge base is updated'),  nl,
        writetofile.

writetofile:-
        open('statefacts.pl',write),
        tell('statefacts.pl'),
        listing(capital_of/2),
        told.
```

```
/* Facts - capital_of/2 */

capital_of( bandar_melaka, melaka ).
capital_of( johor_baharu, johor ).
capital_of( kuantan, pahang ).
capital_of( kuala_terengganu, terengganu ).
capital_of( kota_baharu, kelantan ).
capital_of( kuching, sarawak ).
capital_of( kota_kinabalu, sabah ).
capital_of( alor_star, kedah ).
```

The Eminent Management University

Checking Input Stream

- Predicate: seeing/1
  - return the current input stream
- Syntax:

seeing(Stream)

The Eminent Management University

Checking Input Stream

- Example:

?- output(0). yes	?- seeing(X). X = user
?- output(1). yes	?- seeing (X). X = 1
?- output(2). yes	?- seeing (X). X = 2
?- output('file.pl'). Yes	?- seeing (X). X = 'file.pl'

Checking Output Stream

- Predicate: telling/1
  - return the current output stream
- Syntax:

telling(Stream)

Setting Output Stream

- Using output/1 - set output to the screen, a file or a string
- Format

output(Stream).

- Code for input/1

Console	0
DOS	1
Terminal	2
File	*File Name*

Checking Output Stream

- Example:

?- output(0). yes	?- telling(X). X = user
?- output(1). yes	?- telling(X). X = 1
?- output(2). yes	?- telling(X). X = 2
?- output('file.pl'). Yes	?- telling(X). X = 'file.pl'

Setting Output Stream

- Examples:

?- output(0). ← set to write output to screen
yes

?- output(1). ← set to write output to DOS
yes

?- output(2). ← set to write output to terminal
yes

?- output('output.pl') ←set to write output to 'output.pl'
yes