Lecture notes
Constructing Prolog Program
- Defining facts and rules
- Using connectors
- Constructing query
- Problem representation
- Input & output predicates
- Subroutines

Defining Fact
- Facts – to describe the relationship between objects.
- To represent specific knowledge.
- Example: “Alor Setar is a capital of Kedah”

```
capital_of(alor_setar, kedah).
state(kedah, alor_setar).
is_in(alor_setar, kedah).
```

Defining Rule
- Rule - clause that depend on other facts.
- Example:

```
like(A, B):-
toy(B),
play(A, B).
```
**Defining Rule**

- **Syntax:**
  - Head & body separated by ":-"
  - Body
  - Subgoal separated by ""," or ";;"
  - Full stop

- **Example (Logical statement)**
  - for all X and Y,
  - X is the mother of Y if X is a parent of Y and
  - X is a female.
  - mother(X,Y) :- parent(X,Y), female(X).

- **Example (IF-THEN)**
  - IF A is in B AND B is in C THEN A is in C.

**Defining Rule**

- **Example (IF-THEN)**
  - IF A is clever OR A is smart THEN A is intelligent

**Using Connector**

- Two or more queries or sub goals are connected by the connectors.
- Three main connectors:
  - AND "\&" \\
  - OR "\|" \\
  - NOT "\+" or "NOT"

**Connector - AND**

- **Split with "\&"**

- **Query:**
  - ?- in(city_plaza, alor_star),
  - in(alor_star, kedah).

- **Rule:**
  - intelligent:-
    - clever,
    - smart.
Connection - OR

- Split with ";"

- Query:
  ?:- in(city_plaza, alor_star);
  in(alor_star, kedah).

- Rule:
  intelligent:-
  clever;
  smart.

Connection - NOT

- Start with "\+" or "not"

- Query:
  ?:- \+ in(city_plaza, alor_star).

- Rule:
  dumb:-
  \+ clever.

Establishing Query

- Why needs query?
  - To test relationships especially rules.
  - To obtain knowledge from a system.

Establishing Query

- Connector - OR
  - Split with ";"
  - Query:
    ?:- in(city_plaza, alor_star);
    in(alor_star, kedah).
  - Rule:
    intelligent:-
    clever;
    smart.

- Connector - NOT
  - Start with "\+" or "not"
  - Query:
    ?:- \+ in(city_plaza, alor_star).
  - Rule:
    dumb:-
    \+ clever.

Establishing Query

- Start with "?" and follow by "-" and end with ".".

- Example:
  ?- like(Who, Toy).

- Example:
  ?+ in(city_plaza, alor_star).

- Rule:
  dumb:=-
  \+ clever.

Establishing Query

- List all places in the world.
  ?:- in(X, world).

- Malaysia is in South East.
  ?:- in(malaysia, south_east).

- City Plaza is not in perak.
  ?:- \+ is_in(city_plaza, perak).

Example:

Query that is embedded inside the program file.
Execute automatically during the compiling
Format:

?- start.

*Note: Put the query at the right place.
Representation of problem
- Defining relations:
  - Analyze a problem by considering possible relationships exist
  - Identify possible queries
  - Identify types of relationship (facts or rules)
  - Create meaningful terms that can best describe the relationships between entities in the problem
  - Identify arguments of relations

Representation of problem
- Example:
  - General knowledge
    "If A is in B, then whatever in A is in B as well"
  - Specific knowledge
    "A is in B"
    "C is in B"
    "D is in A"

Representation of problem
- Example:
  - General knowledge
    "If any state is located in a country, then all cities located in that state will be in the same country"
  - Specific knowledge
    "Kedah is in Malaysia"
    "Kelantan is in Malaysia"
    "Johor is in Malaysia"
    "Sintok is in Kedah"
    "Kota Bharu is in Kelantan"
    "Muar is in Johor"

Representation of problem
- Example:
  - General knowledge
    is_in(City, Country):-
    located(City, State),
    located(State, Country).
  - Specific knowledge
    located(kedah, malaysia).
    located(kelantan, malaysia).
    located(johor, malaysia).
    located(sintok, kedah).
    located(kota, bharu, kelantan).
    located(muar, johor).
Representation of problem

How to query?

<table>
<thead>
<tr>
<th>Natural Language</th>
<th>Prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is Muar located in Johor?</td>
<td>located(muar, johor). yes</td>
</tr>
<tr>
<td>Is Sintok located in Kelantan?</td>
<td>located(sintok, kelantan). no</td>
</tr>
<tr>
<td>Which state Sintok is located?</td>
<td>located(sintok, X). X = kedah</td>
</tr>
<tr>
<td>Is Kota Bharu is in Malaysia?</td>
<td>is_in(kota_bharu, malaysia). yes</td>
</tr>
</tbody>
</table>

Example (proposed solution):

A person is a grandfather of someone if he is a father of another person who is the father of that someone

rule:

grandfather(X,Y):-
father(X, T), father(T, Y).

Specific knowledge (facts)

father(george, michael).
ofather(michael, cathy).
ofather(michael, tom).
ofather(cathy, mary).
ofather(tom, david).

General knowledge (rule)

grandfather(X,Y):-
father(X, T), father(T, Y).

Querying the knowledge base

Knowledge base

| father(george, michael).                  |
| father(michael, cathy).                  |
| father(michael, tom).                    |
| mother(cathy, mary).                     |
| father(tom, david).                      |

Console

?- grandfather(X,Y). X = george, Y = cathy.
?- more press "*"
Querying the knowledge base

Examples of queries:

NL: Is Michael Cathy's father?
Prolog: father(michael, cathy).

NL: Who is the father of Cathy?
Prolog: father(X, cathy).

NL: Who is the father of Cathy and mother of Cathy?
Prolog: father(X, cathy), mother(Y, cathy).

NL: Who are Michael's children?
Prolog: father(michael, X).

Exercise

Ann likes every toy she plays with
Doll is a toy
Snoopy is a toy
Ann plays with Snoopy
Sue likes everything Ann likes

Exercise (fact & rules)

facts:
applicant(siti).
applicant(ahmad).
salary(siti, 2000).
salary(ahmad, 1000).
expenses(siti, 4000).
expenses(ahmad, 300).

rules:
status(X, rejected):-
 applicant(X), salary(X, Y), expenses(X, Z), Y =< Z.
status(X, accepted):-
 applicant(X), salary(X, Y), expenses(X, Z), Y > Z.

Exercise (facts & rules)

facts:
toy(doll).
toy(snoopy).
play(ann, snoopy).

rules:
likes(ann, Y):-
toy(Y),
play(ann, Y).
likes(sue, X):-
likes(ann, Y).

Output predicates

- To write or display and format output on console window or screen.
- Commonly use predicates:

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>write/1</td>
<td>write(Term)</td>
<td>write a term to the current output stream</td>
</tr>
<tr>
<td>nl/0</td>
<td>nl</td>
<td>start a new line on the current output stream</td>
</tr>
<tr>
<td>display/1</td>
<td>display(Term)</td>
<td>write a term to the standard output stream in standard prefix notation</td>
</tr>
</tbody>
</table>
Output predicates

Examples:

?- write('TIN2023').
   TIN2023

?- write('TIN2023'), write('Prolog').
   TIN2023\nProlog

?- write('TIN2023'), nl, write('Prolog').
   TIN2023
Prolog

?- display(2+3).
   +(2,3)

Output predicates

Other output predicates:

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>write/1</td>
<td>writeq(Term)</td>
<td>write a quoted term to the current output stream</td>
</tr>
<tr>
<td>write_canonical/1</td>
<td>write_canonical(Term)</td>
<td>write a term to the current output stream in canonical form (combine effects of writeq and display)</td>
</tr>
</tbody>
</table>

Predicate display/1

- Puts all functors in front of their arguments.
- Useful for investigating the internal representation of Prolog terms.
- Example:

  Given X is 2+2, when
  
  7: display(X is 2+2), Prolog will show
  
  is(X,+(2,2))

Writing Formatted Output

fwrite/4 - formatted write of a term

- Writes a simple term Term to the current output stream using the Format, FieldWidth and Modifier flag.
- Syntax:

  fwrite(Format, FieldWidth, Modifier, Term)

  +Format <atom> in the domain {a,b,f,i,n,r,s}.
  +FieldWidth <integer> in the range [-255..255]
  +Modifier <integer> in the range [-255..255]
  +Term <term>

The allowed formats are:

- a: atom
- b: byte-list
- f: floating point number
- i: integer
- n: unsigned integer
- r: arbitrary radix (uses modifier)
- s: string

Limitation of write/1

- displays quoted atoms without quotes.
- cannot easily be read back in using Prolog syntax.
- Example: write('hello there') will display hello there – without quotes.

writeq/1

- Display the term with quotes – can be read back in.
Output predicates

- Examples:
  1. `?- write('TIN2023').`
  2. `?- write('TIN2023'). TIN2023/yes`
  3. `?- write('Course '), write('TIN2023'). Course 'TIN2023'/yes`
  4. `?- display('2' + 3).
     +('2',3)/yes`
  5. `?- write_canonical('2' + 3).
     +(`2`,3)/yes`
  6. `?- display('STIN2023'). STIN2023/yes`
  7. `?- display('Course '). Course 'STIN2023'/yes`

Input of terms

- The input terms must be typed in the same syntax as if it were within a Prolog program.
- Must be followed by a period.

Examples:

1. `?- display(Abc).
   10. ?- display(2+2).
2. `?- display(2+2).
   11. ?- writeq(Abc).
3. `?- writeq(Abc).
   9. ?- writeq(abc).
4. `?- write(2+2).
   8. ?- write(Abc).
5. `?- write(abc), write(cde).
   3. ?- writeq(abc).
6. `?- write(abc), nl, write(cde).
   2. ?- write(abc), nl, write(cde).
7. `?- write('don''t panic').
   7. ?- display('don''t panic').
8. `?- writeq('don''t panic').
   6. ?- writeq('don''t panic').
9. `?- write('don''t panic').
   5. ?- write('don''t panic').
10. `?- write('don''t panic').

Input of terms - Usage Example

% Facts

| capital_of(kota_kinabalu,sabah). |
| capital_of(kuching,sarawak). |
| capital_of(kota_baharu,kelantan). |
| capital_of(kuala_terengganu,terengganu). |
| capital_of(kuantan,pahang). |

% Rule

| go: - |
| read(abc). |
| read(hussain). |
| read(Abc). |
| read(X). |

Query and output example

?- go.
Enter the state name
| ; kelantan. |
Its capital is: kota_baharu
Reading Formatted Data

- **fread/4** - formatted read of a term
- **fread** reads a simple term Term from the current input stream using the Format, FieldWidth and Modifier flag.

**Syntax:**

- Format <atom> in the domain \(\{a, b, f, i, n, r, s\}\).
- FieldWidth <integer> in the range \([-255..255]\).
- Modifier <integer> in the range \([-255..255]\).
- Term <variable>. 

**The allowed formats are:**

- **a** atom (uses modifier)
- **b** byte list (uses modifier)
- **f** floating point number (uses modifier)
- **i** Integer
- **n** unsigned integer
- **r** arbitrary radix (uses modifier)
- **s** string (uses modifier)

**Example:**

```prolog
?- like(ann, X), write(X).
```

Other Character Input/Output

- **Display/print character**

  **Predicate** | **Syntax** | **Description**
  ----------- | ----------- | ------------------
  **put/1**   | \(\text{put}(N)\) (\(N\) is char) | Writes the character whose ASCII code is \(N\) to the current output stream. \(N\) can be an integer in the ASCII range \(0\) to \(255\), or an expression that evaluates to an integer in the ASCII range.
  **putb/1**  | \(\text{putb}(\text{Byte})\) (\(\text{Byte}\) is char) | Output to the screen the ASCII character related to the ASCII value \(\text{Byte}\). If \(\text{Byte}\) is a negative integer then two characters are output to the console window: the first is the null character (0), followed by the character related to the absolute value of \(\text{Byte}\).

**Get/read character**

- **get/1** | \(\text{get}(N)\) (\(N\) is variable or char) | Reads the next non-white space character from the current input stream, and unifies \(N\) with the ASCII value of this character.
- **get0/1** | \(\text{get0}(N)\) (\(N\) is variable or char) | Reads a character from the current input stream, and unifies \(N\) with the ASCII value of this character. When the input file pointer is at the end of a file this \(\text{get0}/1\) returns the value -1.
- **getb/1** | \(\text{getb}(\text{Byte})\) (\(\text{Byte}\) is a variable) | Input a byte from the keyboard or mouse. Mouse keys return -1, -2 and -3 for the pressing of the left, right and both buttons respectively.

**Example:**

```prolog
?- like(ann, X).
```

**In contrast**

- ?- like(ann, X).

  *will force Prolog to look for what Ann like but no output is force to be printed on the screen.*

  By default Prolog will print the value of \(X\) which is instantiated during the matching process.

  Maintaining the value of \(X\) is beneficial when passing a value to other subgoal in the same program.
Computing Vs. Printing

- In other programming language, passing or returning the value is done as follows:
- Example:

```plaintext
Z = add(X, Y);
Z2 = mul(X, Y);
```

```plaintext
int add(int X, int Y)
{
    return X + Y;
}

int mul(int X, int Y)
{
    return X * Y;
}
```

Predicates Vs. Subroutines

- Split the program into separate operations.
- Eg: printing the vegetables in the desired format and backtracking through all alternatives

```prolog
print_veg(X, Y,):-
    add(X, Y, Z),
    write("I like to eat vegetable "),
    write(X), nl,
    fail.

print_vegs:-
    print_veg, fail.
```

Exercise

```prolog
cal(X, Y,):-
    add(X, Y, Z),
    write("I like to eat vegetable "),
    write(X), nl.

print_vegs:-
    print_veg,
    fail.
```

Term and Case Conversion

- `atom_chars/2` converts between an atom and a list of characters
- Example:

```prolog
atom_chars(Atom, CharList )
Atom <variable> or <atom>
CharList <char_list> or <variable>
```

```prolog
?-_atom_chars(eat, CharList ).
CharList = [101,97,116]
Atom = eat
```
Term and Case Conversion

atom_string/2 - convert between an atom and a string

atom_string(Atom, String)

Atom <atom> or <variable>
String <string> or <variable>

Term and Case Conversion

atom_string/2 – example:

?- atom_string(eat, String).
String = `eat`

?- atom_string(Atom, `eat`).
Atom = eat

Term and Case Conversion

number_atom/2 – convert between a number and an atom

number_atom(Number, Atom)

Number <number> or <variable>
Atom <atom> or <variable>

Term and Case Conversion

number_atom/2 – example:

?- number_atom(123, Atom).
Atom = `123`

?- number_atom(Number, `123`).
Number = 123

Term and Case Conversion

number_chars/2 - convert between numbers and a list of characters

number_chars(Number, CharList)

Number <number> or <variable>
CharList <char_list> or <variable>

Term and Case Conversion

number_chars/2 – example:

?- number_chars(123, CharList).
CharList = [49,50,51]

?- number_chars(Number, [49,50,51]).
Number = 123
Term and Case Conversion

- **number_string/2** - convert between a number and a string

  ```
  number_string(Number, String )
  Number <number> or <variable>
  String <string> or <variable>
  ```

- **number_string/2** – example:

  ```
  ?- number_string(123, String ).
  String = `123`
  ?- number_string(Number, `123` ).
  Number = 123
  ```

Term and Case Conversion

- **string_chars/2** - convert between strings and character lists

  ```
  string_chars( String, CharList)
  String <string> or <variable>
  CharList <char_list> or <variable>
  ```

- **string_chars/2** – example:

  ```
  ?- string_chars( `eat`, CharList).
  CharList = [101,97,116]
  ?- string_chars( String, [101,97,116]).
  String = `eat`
  ```

Term and Case Conversion

- **lwrupr/2** - convert between lower and upper case

  ```
  lwrupr(Lower,Upper)
  Lower <atom>, <string> or <variable>
  Upper <atom>, <string> or <variable>
  ```

- **lwrupr/2** – example:

  ```
  ?- lwrupr(eat,Upper).
  Upper = 'EAT'
  ?- lwrupr(Lower,'EAT').
  Lower = eat
  ```
Term and Case Conversion

/.2 - "univ": define the relationship between a term and a list

Term =.. List

  Term  <term> or <variable>
  List  <list> or <variable>

Term and Case Conversion

/.2 – example:

?- eat(ahmad, rice) =.. U.
U = [eat, ahmad, rice]

?- P =.. [eat, ahmad, rice].
P = eat(ahmad, rice)