Teaching Module

Pengaturcaraan Logik

Wan Hussain Wan Ishak
September
Session 2011/2012  (A111)

School of Computing
UUM College of Arts and Sciences
Universiti Utara Malaysia
(P) 04-9284786   (F) 04-9284753
(E) hussain@uum.edu.my
(U) http://staf.uum.edu.my/hussain
# Table of content

1.0 **Overview of Prolog**
   1.1 Basic Data Structures, Facts, Query, Variables, Rules
   1.2 Defining relation by facts and rules
   1.3 Representation of problem
   1.4 Pattern Matching and Backtracking
   1.5 Declarative and Procedural
   1.6 (1-19)

2.0 **Constructing Prolog Programs**
   2.1 Output predicates
   2.2 Predicates vs. subroutines
   2.3 Input of terms
   2.4 Manipulating knowledge base
   2.5 File handling
   2.6 Character input and output
   2.7 (20-34)

3.0 **Computation and Data Structures**
   3.1 Arithmetic Operation, constructing expressions
   3.2 Linguistic Operator
   3.3 List
   3.4 Character strings and structures
   3.5 (35-51)

4.0 **Expressing Procedural Algorithms**
   4.1 Conditional Execution
   4.2 cut (!) operator
   4.3 “if-then-else” structure and repetition through backtracking
   4.4 (52-57)

5.0 **Input/Output and Other Built-in Predicates**
   5.1 Opening/Closing File
   5.2 Setting Input & Output Stream
   5.3 Reading & Writing Formatted Data
   5.4 Term and Case Convention
   5.5 (58-65)

6.0 **Windows Programming**
   6.1 Designing GUI using Prolog
   6.2 Object Classes for dialogs
   6.3 Message passing mechanism
   6.4 Message handling mechanism
   6.5 Related Built-in Predicates
   6.6 (66-82)
Chapter 1
Overview of Prolog

1.0 Overview

Introduction to logic

- Logic is a form of knowledge representation.
- It is a scientific study on reasoning process and production system (also rule) that support the reasoning process.

Premise
- Fact
- Sentence
- Observation
- Information

Logical Process

Inference
- New fact
- Action
- Conclusion

The idea of Prolog

- A programming language.
- “PROgramming in LOGic” → PROLOG
- Very versatile language – can do or implement all kinds of algorithms.
- Prolog program consist of
  1) a set of facts
  2) a set of conditions
- The computer can figure out for itself how to deduce the solution from the facts given.
- Invented by Alain Colmerauer and his colleagues at the University of Aix-Marseille, France in 1972.
- Powerful language for AI and non-numerical programming in general.
- Commercially used in expert systems, intelligent databases, and natural language processing programs.

Varieties of Prolog

- Many Prolog versions are available.
- Standard applied are the same – different in syntax, built-in function, and operating system compatibility.
Example:

<table>
<thead>
<tr>
<th>Arity Prolog</th>
<th>ALS Prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quintus Prolog</td>
<td>SWI Prolog</td>
</tr>
<tr>
<td>Amzi Prolog</td>
<td>LPA Prolog</td>
</tr>
</tbody>
</table>

Prolog vs Lisp

- **Similarities:**
  - both easy to perform complex computations on complex data.
  - both allocate memory dynamically – programmer does not have to declare the size of data structures before creating them.
  - both can examine and modify itself.

- **What difference?**
  - Prolog has an automated reasoning procedure – an INFERENCE ENGINE.
  - Programs that perform logical reasoning are much easier to write in Prolog.

How Prolog works

- **Process - Procedural Interpretation of Logic.**
  - Knowledge is represented in terms of procedure definitions – clauses.
  - Reasoning – a process of calling the right procedures.

- **Example:**

  
  [1] For any X, if X is in Kedah, then X is in Malaysia
  [2] Alor Star is in Kedah

  - **Note:**
    - [1] is a RULE – enables us to infer other info.
    - [2] is a fact – does not depend on other

- **To know whether Alor Star is in Malaysia** – [1] and [2] can be chained together.

  In Prolog:

  \[
  \text{in\_Malaysia(X):- in\_Kedah(X).} \quad \leftarrow \text{rule}
  \]

  \[
  \text{in\_Kedah(alor\_star).} \quad \leftarrow \text{fact}
  \]

  - **Note:**
    - in\_malaysia and
    - in\_Kedah are PREDICATES
1.1 Basic Data Structures, Facts, Query, Variables, Rules

**Terminology**

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Predicate</strong></td>
<td>Consists of name, bracket, and arguments</td>
<td><code>country(malaysia).</code></td>
</tr>
<tr>
<td><strong>Argument</strong></td>
<td>Element in predicate (written in bracket)</td>
<td><code>country(malaysia)</code> → argument is ‘malaysia’.</td>
</tr>
<tr>
<td><strong>Fact</strong></td>
<td>Consist of predicate with or without argument.</td>
<td><code>state(kedah).</code> <code>wan_hussain.</code></td>
</tr>
<tr>
<td><strong>Rule</strong></td>
<td>Predicate that depend on other predicates/facts or information.</td>
<td><code>is_in(X,Y):-</code>  <code>state(X), country(Y,X).</code></td>
</tr>
<tr>
<td><strong>Arity</strong></td>
<td>Referring to the number of arguments in predicate.</td>
<td><code>negara(malaysia).</code> → negara/1 <code>bapa(ahmad, karim).</code> → bapa/2 <code>lapar.</code> → lapar/0</td>
</tr>
</tbody>
</table>

Two distinct predicates can have the same name if they have different arities.

Example: `makan(ahmad,nasi).` `makan(ahmad).`

Predicate is identified by its name, a slash, and its arity.
Syntax

The fundamental units of Prolog syntax are:
- Atoms,
- Numbers,
- Structures, and
- Variables

Atom

- Used as names of individuals and predicates.
- Begins with a lowercase letter
- Can contain letters, digits, and the underscore mark (_)

Number

- Comprises integer and real number.

Example:

<table>
<thead>
<tr>
<th>Atom</th>
<th>Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>kedah</td>
</tr>
<tr>
<td>fatihah1982</td>
<td>fatihah1982</td>
</tr>
<tr>
<td>muhamad_shahrul_aiman_rashid</td>
<td>muhamad_shahrul_aiman_rashid</td>
</tr>
<tr>
<td>‘Malaysia’</td>
<td>‘Malaysia’</td>
</tr>
<tr>
<td>‘17638’</td>
<td>‘17638’</td>
</tr>
</tbody>
</table>

Example:

<table>
<thead>
<tr>
<th>Integer</th>
<th>Real number</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1.254</td>
</tr>
<tr>
<td>12</td>
<td>0.124</td>
</tr>
<tr>
<td>0</td>
<td>0.000009</td>
</tr>
<tr>
<td>100</td>
<td></td>
</tr>
<tr>
<td>789</td>
<td></td>
</tr>
</tbody>
</table>
Structure

- Comprises of several components – atom, bracket (open and close), and argument (inside the bracket).

Example:

```
is_on(book, table).
country(malaysia).
success.
'Country'(malaysia).
book(title(artificial_intelligence),
writer(hisham)).
```

book1(artificial_intelligence, hisham).

- The atom at the beginning is called FUNCTOR of the structure. Example: book1

```
book2(title(artificial_intelligence), writer(hisham)).
```

- If some of the arguments are also structures, the functor at the beginning of the whole thing is called the PRINCIPAL FUNCTOR. Example: book2.

Variable

- Contains of a string of letters, digit, and underscore.
- Begin with capital letters or the underscore mark.

Example:

```
A
Pembolehubah1
Nama_Pelajar
_name_Kolej
Id_17638
```

- A special variable - anonymous variable “underscore (_) character”.
- Don't care how this variable is instantiated - don't care which term it's bound to, as long as it's bound to something.
- Example:

```
?- is_in(X, _).
X = sintok
X = jitra
X = kuantan
X = muar

is_in(sintok, kedah).
is_in(jitra, kedah).
is_in(kuantan, pahang).
is_in(muar, johor).
```
1.1 What are these and is it legal?

- country(indonesia).
- Book(software_engineering) _on__table
- 123574_nom
- eat(ahmad,nasi,ayam).
- ‘Country’(‘Malaysia’).
- 5,000.04
- 78854

1.2 What is wrong with this one?

capital_of(alar_setar, kedah).

1.2 Defining Relation by Facts and Rules

Defining Fact

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

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

Syntax

- Name of predicate and object must be an ATOM.
- The relation is written before the objects.
- End with “.”

relationship(arg_1, arg_2, arg_N).
Defining Rule

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

  like(A, B):-
    toy(B),
    play(A, B).

  % Facts
  toy(bear).
  toy(snoopy).
  toy(car).
  play(ann, snoopy).
  play(ann, comel).
  play(ann, bear).

- Example (IF-THEN)

  IF A is in B AND B is in C THEN A is in C.

- In Prolog

  is_in(A, C):-
    in(A, B),
    in(B, C).

- Syntax:

  goal :-
    Head & body
    Head & body
    separated by “:-”
    separated by “:-”
    Body
    Subgoal separated by “,” or “;”
    Full stop

- Example (converting IF-THEN to Prolog)

  IF A is clever
  OR A is smart
  THEN A is intelligent

  intelligent(A):-
    clever(A);
    smart(A).

- Example (converting logical statement to Prolog)

  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).
More Example

Recursion

A procedure that calling itself to perform the tasks inside its tasks until the stopping condition is reached.

Must have at least two clauses:
- Basic clause – to stop the recursion.
- Recursive clause – the one that call and reference to itself.

Example
Recursive Rule

Example (step-by-step)

(1)

? – is_in(city_plaza, world).

in(city_plaza, alor_star).
in(alor_star, kedah).
in(kedah, malaysia).
in(malaysia, south_east).
in(south_east, asia).
in(asia, world).

Working memory:

X = city_plaza
Y = world

* Fact in(city_plaza, world) not exist
Establishing Query

- **Why needs query?**
  - To test relationships especially rules.
  - To obtain knowledge from a system.
Start with “?” and follow by “=” and end with “.”.
Example:

? - like(Wo, To).

Example with Prolog:

- List all places in the world.
  - ? - is_in(X, world).
- Malaysia is in South East.
  - ? - in(malaysia, south_east).
- City Plaza is not in perak.
  - ? - \+ is_in(city_plaza, perak).

Using Connector

- Two or more queries or sub goals are connected by the connectors.
- Three main connectors:
  - **AND** “,”
  - **OR** “;”
  - **NOT** “\+” or “NOT”

<table>
<thead>
<tr>
<th>Connector</th>
<th>Description &amp; Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AND</strong></td>
<td>Split with “,”</td>
</tr>
<tr>
<td></td>
<td>Query:</td>
</tr>
<tr>
<td></td>
<td>`- in(city_plaza, alor_star),</td>
</tr>
<tr>
<td></td>
<td>in(alor_star, kedah).</td>
</tr>
<tr>
<td></td>
<td>Rule:</td>
</tr>
<tr>
<td></td>
<td>intelligent:-</td>
</tr>
<tr>
<td></td>
<td>clever,</td>
</tr>
<tr>
<td></td>
<td>smart.</td>
</tr>
</tbody>
</table>

| **OR**    | Split with “;”       |
|           | Query:               |
|           | `- in(city_plaza, alor_star); |
|           | in(alor_star, kedah). |
|           | Rule:                |
|           | intelligent:-       |
|           |   clever;            |
|           |   smart.            |
### 1.3 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
- **Simplified the problem** – use table, diagram or chart.

<table>
<thead>
<tr>
<th>NOT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Start with “+” or “not”</td>
<td></td>
</tr>
<tr>
<td>Query:</td>
<td>+ in(city_plaza, alor_star).</td>
</tr>
<tr>
<td>Rule:</td>
<td>dumb:- + clever.</td>
</tr>
</tbody>
</table>

- **Identify general and specific knowledge and the relationship.**
- **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”
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”

How to query?

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

Example:

George is Michael’s father
Michael is Cathy’s father
Joanna is Cathy’s mother
Michael is Tom’s father
Joanna is Tom’s mother
Cathy is Mary’s mother
Tom is David’s father

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

George is Michael’s father
Michael is Cathy’s father
Joanna is Cathy’s mother
Michael is Tom’s father
Joanna is Tom’s mother
Cathy is Mary’s mother
Tom is David’s father

facts:
father(george, michael).
father(michael, cathy).
mother(joanna, cathy).
father(michael, tom).
mother(joanna, tom).
mother(cathy, mary).
father(tom, david).

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

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

In Prolog:

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

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

Specific knowledge (facts)

General knowledge (rule)
Querying the knowledge base

Knowledge base

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

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

Console

?- father(X, michael).
X=george

?- mother(cathy, Y).
Y=mary

?- grandfather(X, Y).
X = george,
Y = cathy;
...more press “;”

Problem Representation 1

Ann’s Doll

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

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

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

  likes(sue, X):-
    likes(ann, Y).
Problem Representation 2

Loan application problem

<table>
<thead>
<tr>
<th>Applicant</th>
<th>Salary</th>
<th>Expenses</th>
<th>Loan Application status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Siti</td>
<td>2000</td>
<td>4000</td>
<td>REJECTED</td>
</tr>
<tr>
<td>Ahmad</td>
<td>1000</td>
<td>300</td>
<td>ACCEPTED</td>
</tr>
</tbody>
</table>

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.

1.3 Pattern Matching and Backtracking

Pattern Matching

- Pattern matching can be regard as the key success in Prolog.
- Matching is a process that takes as input two terms and checks whether they match.
- The matching operator is “=“.
- Example:

  a = a.
  state(kedah) = state(kedah).
Two objects/terms are match if:

- they are IDENTICAL, or
- the variables in both terms can be INSTANTIATED to objects.

**IDENTICAL object** – objects are the same.

**IDENTICAL predicates** - the predicate properties must be the same, i.e;

- The predicate name,
- Number of argument/arity
- The sequence/order of arguments in the predicate

**Example:**

```
2 arity       2 arity       → is_in/2

is_in(kedah,malaysia) = is_in(kedah,malaysia)

Same predicate name

Same sequence of arguments
```

In Prolog matching process is also called UNIFICATION.

**Unification between:**

- Query with the fact
- Query with head of a rule

**Variable in:**

- One side       \[\text{foo(a)} = \text{foo(A)}\].
- Both side      \[\text{foo(A)} = \text{foo(A)}\].
- Mix (both contains variable & non-variable)
  \[\text{foo(A,b)} = \text{foo(a,b)}\].
  \[\text{foo(A,b)} = \text{foo(A,B)}\].

To become identical – the variables will be instantiated – INSTANTIATION.
- Assign value to a variable in order to achieve a match.

**Example:**

| foo(a)=foo(A). | A=a  |
| foo(A)=foo(A). | A=A  |
| foo(A,b)=foo(a,B). | A=a, B=b |
| foo(A,b)=foo(A,B). | A=A, B=b |

### Pattern Matching and Unification

- `binatang(comel)`.  
- `binatang(tompok)`.  
- `binatang(hitam)`.  
- `binatang(boboy)`.  

- `makan(comel,ikan)`.  
- `makan(tompok,ikan)`.  
- `makan(boboy,jagung)`.  
- `makan(hitam,nasi)`.  

- `mengiau(cindai)`.  
- `mengiau(tompok)`.  
- `mengiau(hitam)`.  
- `mengiau(boboy)`.  

- `kucing(X) :-`  
  - `binatang(X)`,  
  - `makan(X,ikan)`,  
  - `mengiau(X)`.

### Discussion

1.3

- `binatang(X) = binatang(comel)`

- `makan(X,ikan) = makan(comel,ikan)`

- `mengiau(comel)`  
  - ERROR - *Predicate not defined*

### Exercise

1.3

- `point(A,B) = point(1,2)`.  
- `point(A,B) = point(X,Y,Z)`.  
- `plus(2,2) = 4`.  
- `+(2,D) = +(E,2)`.  
- `triangle(point(-1,0), P2, P3) = triangle(P1, point(1,0), point(0,Y))`.  

---

Page | 18
Backtracking

- Prolog will automatically backtrack – for satisfying a goal.
- Useful – relieves the programmer of the burden of programming backtracking explicitly.
- In console, Prolog will backtrack automatically after we press “;”: This is inconvenience to some problem.
- Example:

```
?- sayur(X), write(`saya suka makan sayur `), write(X), nl.
saya suka makan sayur sawi
X = sawi ;
saya suka makan sayur kangkung
X = kangkung ;
---
% facts
sayur(sawi).
sayur(kangkung).
sayur(bendi).
sayur(kacang).
sayur(petola).
```

- To force backtracking use fail/0.
- Example:

```
?- sayur(X), write(`saya suka makan sayur `), write(X), nl, fail.
saya suka makan sayur sawi
saya suka makan sayur kangkung
saya suka makan sayur bendi
saya suka makan sayur kacang
saya suka makan sayur petola
```

1.4 Declarative and Procedural Meaning

- **Declarative meaning**
  - concerned only with the **relations** defined in the program.
  - determines **what** will be the output of the program.
- **Procedural meaning**
  - How the output is obtained – how the relations are actually evaluated by Prolog.
  - Necessary when writing programs that interact with user.
Chapter 2
Constructing Prolog Programs

2.0 Overview

Learn how to write data to output stream and read data from input stream.

2.1 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>

- Examples:

```prolog
?- write('TIN2023').
TIN2023yes
?- write('TIN2023'), write('Prolog').
TIN2023Prologyes
?- write('TIN2023'), nl, write('Prolog').
TIN2023Prologyes
?- display(2+3).
+(2,3)yes
```

- Predicate display/1
  - Puts all functors in front of their arguments.
  - Useful for investigating the internal representation of Prolog terms.
  - Example:

```prolog
Given X is 2+2, when
?- display(X is 2+2), Prolog will show
is(X,+(2,2))
```
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.

Other output predicates:

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>writeq/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>

Examples:

?- write(`TIN2023`).
TIN2023yes
?- writeq(`TIN2023`).
`TIN2023`yes
?- write(`Course `), writeq(`TIN2023`).
Course `TIN2023`yes
?- display(`2` + 3).
+(2,3)yes
?- write_canonical(`2` + 3).
+(`2`,3)yes
Output Predicates – What will be the output?

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

2.1 Computing Vs. Printing

- Using output predicates such as write, and display will force Prolog to print the result or output on screen.

  example:

  \(?-\) like(ann, X), write(X).

- will force Prolog to look for what ann like and print it on the screen.

- 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.

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

```
Exercise

2.1

```

```
cal(X, Y, Z):-
    add(X, Y, Z),
    write(Z),
    mul(X, Y, Z2),
    write(Z2).

add(X, Y, Z):-
    Z is X + Y.

mul(X, Y, Z):-
    Z is X * Y.
```

2.2 Predicates vs. subroutines

- The rule defines a subroutine – all subgoals can be execute through single query.
- Writing all subgoals in one rule in inefficient in Prolog.
- Example:

  ```
  print_veg:-
      veg(X),
      write('I like to eat vegetable '),
      write(X), nl,
      fail.
  ```

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

```
print_veg:-
    veg(X),
    write('I like to eat vegetable '),
    write(X), nl.
print_vegs:-
    print_veg,
    fail.
```

### 2.3 Input of terms

- To get input from user or input streams.
- Built-in predicate `read/1`
- Syntax:

  ```prolog
  read(Term).
  ```

Example:

```
| ?- read(X).
|: stin2023.
X = stin2023
| ?- read(X).
|: 'STIN2023 Prolog'.
X = 'STIN2023 Prolog'
| ?- read(X).
|: stin2023 prolog.
* Syntax Error
```

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

1) ```prolog
    ?- read(X).
    |: abc.
    X = abc
```

2) ```prolog
    ?- read(hussain).
    |: hussain.
    Yes
```
3)  \( ?- \) read(X).
    \[ \text{;} \text{;} Y. \]
    X = _

4)  \( ?- \) read(X).
    \[ \text{;} \text{;} \text{;} abc \]
    \[ \text{;} \text{;} X = abc \]

5)  \( ?- \) read(X).
    \[ \text{;} \text{;} \text{;} a \text{;} b. \]
    * Syntax Error

Usage examples:

\% Facts

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).

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

Query and output example

? - go.
Enter the state name
| : kelantan.
Its capital is: kota_baharu
2.4 Manipulating knowledge base

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.
- **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.
- Example using dynamic

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

Or

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

- Example using assert

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

- 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

- 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.

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

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assert/1</td>
<td>assert(Clause)</td>
<td>add a clause at the end of the clauses associated with its predicate name</td>
</tr>
<tr>
<td>assert/2</td>
<td>assert(Clause, Position)</td>
<td>assert the clause at the given position</td>
</tr>
<tr>
<td>asserta/1</td>
<td>asserta(Clause)</td>
<td>add a clause at the beginning of the clauses associated with its predicate name</td>
</tr>
<tr>
<td>assertz/1</td>
<td>assertz(Clause)</td>
<td>add a clause at the end of the clauses associated with its predicate name</td>
</tr>
</tbody>
</table>

Example: assert/1

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

**Working memory**

toy(car).
toy(bunny).
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) |

Example: asserta/1 and assertz/1

?- asserta(toy(snoopy)).
  yes
?- assertz(toy(lorry)).
  yes

Working memory

| toy(snoopy) |
| toy(car) |
| toy(bunny) |
| toy(lorry) |

Example: insert new rule

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

*assert/1 only accept one argument only


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

Working memory

| cucu(A, B) :- anak(A, C), anak(C, B) |

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

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>retract/1</td>
<td>retract(Clause)</td>
<td>delete a clause that matches the given clause</td>
</tr>
<tr>
<td>retract/2</td>
<td>retract(Clause, Position)</td>
<td>retract a clause at a specified position</td>
</tr>
<tr>
<td>retractall/1</td>
<td>retractall(Head)</td>
<td>delete all clauses that match the given clause head</td>
</tr>
</tbody>
</table>
Example: retract/1

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

Example: retract/2

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

Example: retractall/1

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

Example: retract rule

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

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.
<table>
<thead>
<tr>
<th>Predicate</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abolish/1</td>
<td>abolish(Preds)</td>
<td>delete all predicates specified by the given argument</td>
</tr>
<tr>
<td>abolish/2</td>
<td>abolish(Functor, Arity)</td>
<td>delete the predicate specified by the given functor and arity</td>
</tr>
</tbody>
</table>

- Example: abolish/1
  ```prolog
  ?- abolish(toy/1).
  yes
  ```

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

- Example – abolish more than one clauses
  ```prolog
  ?- abolish ([toy/1,like/2]).
  yes
  ```

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

- Example: abolish/2
  ```prolog
  ?- abolish(toy,1).
  yes
  ```

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

- To see the contents of the knowledge base in memory use listing/0 or listing/1.
<table>
<thead>
<tr>
<th>Predicate</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>listing/0</td>
<td>listing</td>
<td>list all the dynamic clauses in the workspace to the current output stream</td>
</tr>
<tr>
<td>listing/1</td>
<td>listing(Ttolist)</td>
<td>list the specified dynamic predicates to the current output stream</td>
</tr>
</tbody>
</table>

Example: listing/0 and listing/1

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

2.5 File handling

File Handling - Reading data

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

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

Example:

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

- 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.
- 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.
- Example:

  ?- see('data.pl').
  yes
  ?- read(X), read(Y).
  X = kedah
  Y = alor_star
  ?- seen.
  yes

  File data.pl
  kedah
  alor_star

File Handling – Writing Data

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

    
    tell(Stream). ← Causes Stream to become the current output stream.

    ...

    ...

    told. ← Close the Stream connection.

- Example:

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

- 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.
Example:

?- tell('out.pl').
  yes
?- write('kedah'), nl, write('alor_star').
  yes
?- told.
  yes

About Consult and Reconsult?

When to use consult and reconsult?

main.pl

```
:- consult('file.pl').
start:-
  ...
  ...
  reconsult('file.pl'),
  ...
```

file.pl

```
.....
.....
.....
.....
.....
```

2.6 Character input and output

Get/read character

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get/1</td>
<td>get(N) (N is variable or char)</td>
<td>Reads the next non-white space character from the current input stream, and unifies N with the ASCII value of this character.</td>
</tr>
<tr>
<td>get0/1</td>
<td>get0(N) (N is variable or char)</td>
<td>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 get0/1 returns the value -1.</td>
</tr>
<tr>
<td>getb/1</td>
<td>getb(Byte) (Byte is a variable)</td>
<td>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.</td>
</tr>
<tr>
<td>Predicate</td>
<td>Syntax</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| put/1     | 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    | putb(Byte)  
(Byte is char) | Output to the screen the ASCII character related to the ASCII value Byte. If 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 Byte. |
Chapter 3
Computation and Data Structures

3.0 Overview

Prolog also supports arithmetic operation and list manipulation. List is one of powerful data structures in Prolog.

3.1 Arithmetic Operation

- Arithmetic – how we do calculation in Prolog.
- Use predicate “is/2” not “=/2”.
- is/2 takes an arithmetic expression on its right, evaluates it, and unifies the result with its argument on the left.

- Format:

  <argument> is <arithmetic expression>.

- Example:

  X is 2 + 1.
  X is 9.
  A = 1, X is A + 5.
  2 is 1 + 1.

- Wrong expression:

  2 + 1 is 3. ← Type error - not in the right format.
  2 is 1 + X ← Instantiation error – value of X is unknown.

- Different with “is/2” and “=/2”:
  - is/2 evaluate its right argument and unifies it with its argument on the left – do calculation.
  - =/2 is to test if both arguments on the left and right are unify – do unification.
Common arithmetic operators (infix operators):

<table>
<thead>
<tr>
<th>Operator &amp; Symbol</th>
<th>Example &amp; Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>X is 3 + 2.</td>
</tr>
<tr>
<td>-</td>
<td>X is 3 - 2.</td>
</tr>
<tr>
<td>*</td>
<td>X is 3 * 2.</td>
</tr>
<tr>
<td>/</td>
<td>X is 3 / 2.</td>
</tr>
<tr>
<td>//</td>
<td>X is 3 // 2.</td>
</tr>
</tbody>
</table>

Other arithmetic functions supported in LPA Prolog

**is/2 arithmetic functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X^Y</td>
<td>X to the power of Y.</td>
</tr>
<tr>
<td>abs(X)</td>
<td>the absolute value of X. e.g. abs(-3.5) returns 3.5.</td>
</tr>
<tr>
<td>acos(X)</td>
<td>the arc cosine of X in degrees.</td>
</tr>
<tr>
<td>asin(X)</td>
<td>the arc sine of X in degrees.</td>
</tr>
<tr>
<td>atan(X)</td>
<td>the arc tangent of X in degrees.</td>
</tr>
<tr>
<td>cos(X)</td>
<td>the cosine of X degrees.</td>
</tr>
<tr>
<td>sin(X)</td>
<td>the sine of X degrees.</td>
</tr>
<tr>
<td>tan(X)</td>
<td>the tangent of X degrees.</td>
</tr>
<tr>
<td>log(X)</td>
<td>the base 10 logarithm of X.</td>
</tr>
<tr>
<td>max(X,Y)</td>
<td>the maximum value of X and Y. e.g. max(3.5,4) returns 4.</td>
</tr>
<tr>
<td>min(X,Y)</td>
<td>the minimum value of X and Y. e.g. min(3.5,4) returns 3.5.</td>
</tr>
<tr>
<td>rand(X)</td>
<td>a random floating point number between zero and X.</td>
</tr>
<tr>
<td>sign(X)</td>
<td>-1 if X is negative, 0 if X is 0, or 1 if X is positive. e.g. sign(-3.5) returns -1.</td>
</tr>
<tr>
<td>sqrt(X)</td>
<td>the square root of X.</td>
</tr>
</tbody>
</table>

The precedence of operators:

Example:

\[ X \text{ is } 1 - 2 \times 5^2 / 8 + 6 \]

Equivalent to

\[ X \text{ is } (1 - (((2 \times (5^2)) / 8) + 6)) \]

Where ^ is performed first, then * and /, and finally + and -. 

Page | 36
Using Arithmetic Operators

- Example:

```prolog
?- papar(5).
1
21
321
4321
```

1. `printOut(N):-`
2. `printOut(1, N).
3.
4. `printOut(N, N).
5.
6. `printOut(X, N):-`
7. `printOut_num(X),`
8. `nl,`
9. `NewX is X + 1,`
10. `printOut(NewX, N).
11.
12. `printOut_num(0).
13.
14. `printOut_num(X):-`
15. `write(X),`
16. `NewX is X - 1,`
17. `printOut_num(NewX).
```

Constructing Expression

- Different between Prolog an other programming languages is:
  - Other programming language evaluate arithmetic expression s whenever they occur
  - Prolog evaluates them only in specific places.

- List of built-In Predicates that evaluate expressions.

<table>
<thead>
<tr>
<th>R is Ex</th>
<th>Evaluates Ex and unifies result with R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex1 =:= Ex2</td>
<td>test if the results of two expressions are equal</td>
</tr>
<tr>
<td>Ex1 == Ex2</td>
<td>test if the results of two expressions are not equal</td>
</tr>
<tr>
<td>Ex1 &gt; Ex2</td>
<td>test if the result of one expression is greater than another</td>
</tr>
<tr>
<td>Ex1 &lt; Ex2</td>
<td>test if the result of one expression is less than another</td>
</tr>
<tr>
<td>Ex1 &gt;= Ex2</td>
<td>test if one expression is greater than or equal to another</td>
</tr>
<tr>
<td>Ex1 &lt;= Ex2</td>
<td>test if one expression is less than or equal to another</td>
</tr>
</tbody>
</table>
Exercise 3.1

Explain which of the following succeed, fail, or raise error conditions, and why:

1. ?- 5 is 2 + 3.
2. ?- 5 =:= 2+3.
3. ?- 5 = 2 + 3.
4. ?- 5 is 2 + What.
5. ?- 4+1 = 2+3.
6. ?- 4+1 is 2+3.
7. ?- 4+1 =:= 2+3.
8. ?- What is 2+3.
9. ?- What =:= 2+3.
10. ?- What is 5.
11. ?- What = 5.
12. ?- What =\= 2+3.

Practical Calculation

- When using expressions in Prolog, we are mixing the styles of encoding knowledge.
- Normal syntax:
  
  ```prolog
  X is 2 + 3.
  ```
- Can be written as:
  
  ```prolog
  ?- sum(2, 3, X).
  ```
- Example:
  
  ```prolog
  doublevalue(X, Z):-
  Z is X * X.
  ```
  
  ```prolog
  areaOfSquare(P, L, Area):-
  Area is P * L.
  ```

3.2 Linguistic Operator

- Data structure in Prolog:
  
  ```prolog
  functor(arg1, arg2, ..., argN).
  ```
- Functor can become operator.
  
  ```prolog
  eat(ali, rice).
  ```
  
  ```prolog
  ali eat rice.
  ```
Types of operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infix</td>
<td>3 + 4</td>
</tr>
<tr>
<td>Prefix</td>
<td>+(3,4)</td>
</tr>
<tr>
<td>Postfix</td>
<td>8 factorial</td>
</tr>
</tbody>
</table>

Operator can be defined using op/3.

- **op/3** - Declares an operator with a given type and precedence.
- **Syntax:**
  
  \[
  \text{op(Precedence, Type, Name)}
  \]

- **Note:**
  The Name argument is an atom that is the name of the operator. The *Precedence* is an integer between 0 and 1199, and whose type is *Type*. *Name* may also be a list of atoms, in which case all of the atoms are declared to be operators of the specified precedence and type.

Linguistic Operator – Precedence Value

- The value between 0 – 1199.
- The precedence value can influence the structure meaning:
  - low precedence meaning high priority.
  - Example: operator with 33 is given higher priority compare to 35.
  - operator with 33 will be first executed.

Example:

```prolog
| ?- op(35,xfx,is_in).
  yes |
| ?- op(33,fx,room).
  yes |
| ?- display(apple is in room kitchen).
  is_in(apple, room(kitchen)) |
| ?- op(36,fx,room).
  yes |
```

Defining operator 'is_in' and 'room'

Use display/1
To view the standard syntax

Change the precedence value

Error
Wrongly use operator

```prolog
| ?- display(apple is in room kitchen).
  is_in(apple, room(kitchen)). |
| ! Error 42 : Syntax Error |
| ! Goal    : ered(_11848,_11850) |
| ?- display(room kitchen is in house).
  room(is_in(kitchen,house)) |
```
Example

?- op(35, fx, is_in).
yes

?- op(33, fx, room).
yes

?- op(36, fx, room).
yes

?- display(apple is in room kitchen).
is_in(apple, room(kitchen))

?- display(room kitchen is in house).
room(is_in(kitchen, house))

Linguistic Operator – Defining Type

- If the precedence value is the same Prolog will read either from left to right or right to left based on the type.

Type for operator

<table>
<thead>
<tr>
<th>Types</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>fx</td>
<td>non-associative prefix operator ← prefix</td>
</tr>
<tr>
<td>fy</td>
<td>right associative prefix operator ← prefix</td>
</tr>
<tr>
<td>xf</td>
<td>non-associative postfix operator ← postfix</td>
</tr>
<tr>
<td>yf</td>
<td>left associative postfix operator ← postfix</td>
</tr>
<tr>
<td>xfx</td>
<td>non-associative infix operator ← infix</td>
</tr>
<tr>
<td>xfy</td>
<td>right associative infix operator ← infix</td>
</tr>
<tr>
<td>yfx</td>
<td>left associative infix operator ← infix</td>
</tr>
</tbody>
</table>

Example cth.pl

% Linguistic operators
:- op(800, fx, if),
    op(700, xfx, then),
    op(300, xfy, or),
    op(200, xfy, and).

% Facts
binatang(comel).
binatang(tompok).
makan(comel, ikan).
Makan(tompok, ikan).
mengiau(comel).
mengiau(tompok).

% Rules
if binatang(X) and makan(X, ikan) then kucing(X).
Examine

% Rules
if binatang(X) and makan(X,ikan) then kucing(X).

in standard syntax

\[
\text{if}(\text{then}(\text{and}(\text{binatang}(_{13214}),\text{makan}_{13214}(\_13214,\_13214)),
\text{kucing}(_{13214})))
\]

## 3.3 List

- List is one of the most important Prolog data structures.
- A list is an ordered sequence of zero or more terms written between square brackets and separated by commas.

\[
[a,b,c,d]
\]

- The elements of a list can be any kind of Prolog terms, including other lists.
- Example:

\[
[1,2,3,4] \quad [a1,a2,a3]
\]
\[
{sawi, kangkung} \quad [\text{kucing(comel)}, \text{kucing(hitam)}]
\]
\[
[[\text{satu,dua}],\text{[tiga,empat]}]
\]

- The empty list is written:

\[
[]
\]

- Please note one element list \([a]\) is not equivalent to the atom a.

\[
\]
\[
\text{no}
\]
List can be constructed or decomposed through unification.

<table>
<thead>
<tr>
<th>Unify with</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>([a,b,c] = X)</td>
<td>(X = [a,b,c])</td>
</tr>
<tr>
<td>([X,b,Z] = [a,Y,c])</td>
<td>(X = a, Y = b, Z = c)</td>
</tr>
<tr>
<td>([a,b],c = [X,Y])</td>
<td>(X = [a,b], Y = c)</td>
</tr>
<tr>
<td>([a(b),c(X)] = [Z,c(a)])</td>
<td>(X = a, Z = a(b))</td>
</tr>
</tbody>
</table>

More examples

<table>
<thead>
<tr>
<th>Unify with</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>([X</td>
<td>Y] = [a,b,c,d])</td>
</tr>
<tr>
<td>([X</td>
<td>Y] = [a])</td>
</tr>
<tr>
<td>([X,Y</td>
<td>Z] = [a,b,c])</td>
</tr>
<tr>
<td>([X,Y,Z</td>
<td>A] = [a,b,c])</td>
</tr>
<tr>
<td>([X,Y</td>
<td>Z] = [a</td>
</tr>
</tbody>
</table>

The list can be divided into head and tail by the symbol ‘\(|\)’. 

\[H | T\]

The first element is the head and the rest are the tail.

Example:

\([a | [b,c,d,e]]\)

The tail of a list is always a list, the head of a list is an element.

Every nonempty list has a head and a tail.

\([a,b,c,d] = [a | [b,c,d]]\)
\([a] = [a | []]\)
List Manipulation

- Built-in predicates

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>append/3</td>
<td>join or split lists</td>
</tr>
<tr>
<td>length/2</td>
<td>get the length of a Prolog list</td>
</tr>
<tr>
<td>member/2</td>
<td>get or check a member of a list</td>
</tr>
<tr>
<td>member/3</td>
<td>get or check a member of a list and its position</td>
</tr>
<tr>
<td>remove/3</td>
<td>remove an element from a list</td>
</tr>
<tr>
<td>removeall/3</td>
<td>remove all occurrences of an item from a list</td>
</tr>
<tr>
<td>reverse/2</td>
<td>check or get the reverse of a list</td>
</tr>
</tbody>
</table>

- append/3

Syntax:

append(First, Second, Whole)

Example:

?- append([a,b], [c,d], Whole).
Whole = [a,b,c,d]
?- append([a,b], Second, [a,b,c,d]).
Second = [c,d]
?- append(First, [c,d], [a,b,c,d]).
First = [a,b]

- length/2

Syntax:

length(Term, Length)

Example:

?- length([a,b,c,d], Length).
Length = 4
member/2

Syntax:

member(Element, List)

Example:

?- member(a, [a,b,c,d]).
   yes
?- member(Element, [a,b,c,d]).
   Element = a ;
   Element = b ;
   Element = c ;
   Element = d ;

member/3

Syntax:

member(Element, List, Position)

Example:

?- member(c, [a,b,c,d], Position). \(\leftarrow\) checking the element position
   Position = 3 ;

?- member(Element, [a,b,c,d], 2). \(\leftarrow\) what element at given position
   Element = b

?- member(Element, [a,b,c,d], Position). \(\leftarrow\) get element and its position
   Element = a ,
   Position = 1 ;
   Element = b ,
   Position = 2 ;
   Element = c ,
   Position = 3 ;
   Element = d ,
   Position = 4 ;
remove/3

Syntax:

\[ \text{remove}(\text{Element}, \text{List}, \text{Remainder}) \]

Example:

?- remove(b, [a,b,c,d], Remainder). \quad \leftarrow \text{takeout an element}
Remainder = [a,c,d] ;

?- remove(Element, [a,b,c,d], [a,b,d]). \quad \leftarrow \text{what is missing}
Element = c ;

?- remove(Element, [a,b,c,d], Remainder). \quad \leftarrow \text{what element can be removed}
Element = a ,
Remainder = [b,c,d] ;
Element = b ,
Remainder = [a,c,d] ;
Element = c ,
Remainder = [a,b,d] ;
Element = d ,
Remainder = [a,b,c] ;

removeall/3

Syntax:

\[ \text{removeall}(\text{Item}, \text{List}, \text{Remainder}) \]

Example:

?- removeall(a, [a,b,a,b,a], Remainder).
Remainder = [b,b]

?- removeall(Item, [a,b,a,b,a], Remainder).
Item = a ,
Remainder = [b,b]

?- removeall(Item, [b,a,b,a], Remainder).
Item = b ,
Remainder = [a,a]
Reverse/2

Syntax:
   reverse(List, Revlist)

Example:
   ?- reverse([a,b,c,d], Revlist).
   Revlist = [d,c,b,a]

   ?- reverse(List, [d,c,b,a]).
   List = [a,b,c,d] ;
   Error 4, Heap Space Full, Trying ewrite/1

What is the output?

   ?- append([ab],[b,c,d], X).

   ?- reverse([b,c,d], R), append([ab], R, X).

   ?- reverse([b,c,d], R), member(F, R, 1), remove(F, R, B), append([ab], B, X).

Exercise

3.2

List – Constructing Predicates

- How to define or construct the predicates?
- How they are working/How they manipulate the list?

- append/3
- length/2
- member/2
- member/3
- remove/3
- removeall/3
- reverse/2
append/3 → addon/3

Syntax:
addon(First, Second, Whole)

Definition:
addon([],X,X). ← stop the recursion
addon([H|T1],X,[H|T2]):- addon(T1, X, T2).

?- addon([a,b],[c],W).  W = [a,b,c]
Ep. 1 (2)
(2.1) addon([a|b],[c],[a|T2]):- [a|b,c].
      addon([b],[c],T2).  T2 = [b,c]
Ep. 2 (2)
(2.1) addon([b|[I], [c], [b|T2]):- [b|c].
      addon([], [c], T2).  T2 = [c]
Ep. 3 (1)
addon([], [c], [c]).

length/2 → noOfTerms/2

Syntax:
nOOfTerms(Term, Length)

Definition:
nOOfTerms([],0). ← stop the recursion
nOOfTerms([H|Tail],K):- nOOfTerms(Tail,J),
K is J + 1.
member/2 \rightarrow is\_in/2

**Syntax:**

\[
\text{is\_in} \ (\text{Element}, \text{List})
\]

**Definition:**

\[
\text{is\_in} \ (X, [X \mid T]) \leftarrow \text{check if the Element is Head of the list}
\]

\[
\text{is\_in} \ (X, [H \mid T]) :- \text{if not, traverse the rest of the list (Tail)}
\]

\[
\text{is\_in} (X, T).
\]
Example (list all elements in a list):

\[
\text{is\_in}(X, [a, b, c]) \Rightarrow \\
X = a \Rightarrow \text{Return } X = a \\
X = b \Rightarrow \text{Return } X = b \\
X = c \Rightarrow \text{Return } X = c
\]

Example (find an element in the list):

\[
\text{is\_in}(c, [a, b, c]) \Rightarrow \\
c \not= a \Rightarrow \text{is\_in}(c, [b, c]) \Rightarrow c \not= b \Rightarrow \text{is\_in}(c, [c]) \Rightarrow c = c \Rightarrow \text{Return yes}
\]

- remove/3 $\rightarrow$ take_out/3

Syntax:

\[
\text{take\_out}(\text{Element, List, Remainder})
\]

Definition:

\[
\begin{align*}
\text{take\_out} & (X, [X|R], R). & \leftarrow \text{element found} \\
\text{take\_out} & (X, [F|R], [F|S]):- \\
& \text{take\_out}(X, R, S). & \leftarrow \text{traverse the list, at the} \\
& & \text{same time create new list}
\end{align*}
\]
reverse/2 \(\rightarrow\) overturn/2

**Syntax:**
overturn (List, Revlist)

**Definition:**
overturn (List, RevList):-
    overturn2(List, [], RevList).
overturn2([[H | T], Z, W]:-
    overturn2(T, [H | Z], W).
overturn2([], X, X).

\(\downarrow\) the element position

### 3.4 Character Strings and Structures

- Three ways to represent a string of characters in Prolog:
  - As an atom – atoms are compact but hard to take apart or manipulate.
  - As a list of ASCII codes – can use standard list processing techniques on them.
  - As a list of one-character atoms - can use standard list processing techniques on them.

- Use write/1 with double quotes.

  **Example:**
  
  ```prolog
  ?- write("abc").
  [97,98,99]
  ```

- Use put/1 or putb/1.

  **Example:**
  
  ```prolog
  write_str([H | T]):-
    put(H), write_str(T).
  write_str([]).
  ```

- Convert between an atom or number and a byte list - name/2.

**Syntax:**

name(Atomic, List)

**Example:**

```prolog
?- name(makan, List).
List = [109,97,107,97,110]

?- name(Atomic, [109,97,107,97,110]).
Atomic = makan
```
Example (Prolog codes):

```
splitNprint(Ayat):-
    name(Ayat,AyatList),
    write_str(AyatList).
write_str([H|T]):-
    put(H), write_str(T).
write_str([]).
```

Output:

```
?- splitNprint(ahmad).
ahmad
```

Write a code/rule to print out as follows:

```
?- print_splits("prolog").
prolog
prolog
prolog
prolog
prolog
prolog
yes
```

Exercise 3.3

Write a code/rule to check whether the word end with “s” or not:

```
?- end_with_s("flowers").
yes
?- end_with_s("car").
no
?- end_with_s("cars").
yes
```

Exercise 3.4
Chapter 4
Expressing Procedural Algorithm

4.0 Overview

- Prolog combines procedural and non-procedural programming techniques.
- Prolog’s control strategy – based on simple depth-first search.

4.1 Conditional Execution

- Prolog procedures can have multiple definitions (clauses) – each applying under different conditions.
- Conditional execution (if or case statements) – expressed with alternative definitions of procedures.
- Example - Java switch/case statement:

```java
public static void printNum(int X) {
    switch(X) {
    case 1:
        System.out.println("One");
    case 2:
        System.out.println("Two");
    case 3:
        System.out.println("Three");
    }
}
```

- In Prolog, `printNum` has three definitions:

  ```prolog
  printNum(1):- write('One').
  printNum(2):- write('Two').
  printNum(3):- write('Three').
  ```

- Common mistakes - inefficient:

  ```prolog
  printNum(X):- X=1, write('One').
  printNum(X):- X=2, write('Two').
  printNum(X):- X=3, write('Three').
  ```

- Gives correct results but waste time - Execute each clause, perform test, and backtrack out.
Effective programming in Prolog:
- Make each logical unit of the program into a separate procedure.
- Each if or case statement should become a procedure call – decisions are made by procedure-calling process – choosing the right clause.

Example

In Pascal

```pascal
procedure a(X:integer);
begin
  b;
  if X=0 then c else d;
  e
end;
```

In Prolog

```prolog
a(X):- b,
  cd(X),
  e.
```

```
4.2 Backtracking (revision)
```

- Prolog will automatically backtrack – for satisfying a goal.
- In console, Prolog will backtrack automatically after we press “;”.
- To force backtracking use fail/0.

4.3 Controlling Backtracking

- Uncontrolled backtracking may cause inefficiency in a program.
- Control using ‘cut’ facility.
- The symbol is ‘!’.
- Function – prevent backtracking.
- Useful – relieves the programmer of the burden of programming backtracking explicitly.
- Example (in Pascal):

```pascal
procedure writename(X:integer);
begin
  case X of
  1: write(‘one’);
  2: write(‘two’);
  3: write(‘three’);
  else
    write(‘out of range’)
  end
end;
```
- Converted into Prolog as:

```prolog
writename(1):- write('one').
writename(2):- write('two').
writename(3):- write('three').
writename(X):- X < 1, write('out of range').
writename(X):- X > 3, write('out of range').
```

- Prolog version - Correct but lack of *conciseness*.
- The last two `writename/1` are inefficient.
- A better version:

```prolog
writename(1):- write('one').
writename(2):- write('two').
writename(3):- write('three').
writename(_):- write('out of range').
```

- Problem with this version
  - For example, the query `?- writename(1)` will matches the first and last clauses.
  - It will produce 2 answers/outputs: “one” and “out of range”
  - Why? - Anonymous variable is used in the last clause. This variable will match with any value.
  - This happen because of backtracking.

- How to control or prevent backtracking?
  - Use a special operator called cut. The symbol is ‘!’
  - Cut “!” operator will tell the computer to ignore other alternatives – stop backtracking.

- Example: (Prolog)

```prolog
b:- c, d, !, e, f.
b:- g, h.
```

- Better version of `writename/1` using cuts

```prolog
writename(1):- !, write('one').
writename(2):- !, write('two').
writename(3):- !, write('three').
writename(_):- write('out of range').
```
Using Cuts “!”

Given that:

max(X,Y,Max).

Where Max = X if X is greater than or equal to Y, and Max = Y if X is less than Y.

max(X,Y,X):- X>=Y.
max(X,Y,Y):- X<Y.

These rules are mutually exclusive.

- If the first one succeeds then the second one will fail.
- If the first one fails then the second must succeed.

More economical formulation instead of:

In Prolog:

if X>=Y then Max = X
if X<Y   then Max = Y

max(X,Y,X):
- X>=Y.
max(X,Y,Y):
- X<Y.

we may consider:

In Prolog:

if X>=Y then Max = X,
else Max=Y

max(X,Y,X):- X>=Y, !.
max(X,Y,Y).

4.1 Making Goal Deterministic without Cuts

- Instead of creating deterministic predicates, we can define nondeterministic predicates in the ordinary manner and then block backtracking when we call them.
- Special built-in predicate once/1.
- To define once/1 as:

  once(Goal):- call(Goal), !.
Example:

?- food(A).
   A = rice ;
   A = ice_cream ;
   A = banana
   
   All alternatives were taken out

?- once(food(A)).
   A = rice
   
   Only the first fact return.

Facts:

food(rice).
food(ice_cream).
food(banana).

4.5 The “IF-THEN-ELSE” Structure

- Can be implemented in Prolog as:

\[
\text{Goal1} \rightarrow \text{Goal2} ; \text{Goal3}
\]

“if Goal1 then Goal2 else Goal 3”

- These symbols are used “\(\rightarrow\)” and “\(;\)”.
- Meaning:
  Test whether Goal1 succeeds, and if so, execute Goal2, otherwise execute Goal3.
- Example (simple if-then-else):

  \[
  \text{writeNum(X)}:- \ X=1 \rightarrow \text{write(one)} ; \text{write(‘Not one’)}.
  \]

  Meaning:
  If X = 1 the ‘One’ will be written, if not (else) ‘Not one’ will be written.

- Example (nested if-then-else):

  \[
  \text{writeNum(X)}:-
  \ 
  \ 
  \ 
  ( \ X=1 \rightarrow \text{write(one)}
  ; \ X=2 \rightarrow \text{write(two)}
  ; \ X=3 \rightarrow \text{write(three)}
  ; \text{write(‘out of range’)}
  ) \].

- If-then-else structure – for making decision without calling procedures.
- Discouraged
  - Looks like ordinary structured programming
  - Prolog clauses are supposed to be logical formulas.
4.6 Goal Always Succeed or Always Fail

In order to control the program flow, there is a need:

- to guarantee that a goal will succeed - regardless of the results.
- to guarantee that a goal will always fail.

**Always succeed:**
- Used `true/0`.
- Example:

```prolog
?- eat(ahmad,fish); true.
```

**Always fail:**
- Used `fail/0`.
- Example:

```prolog
writeNum(X):- X > 0, write('more than 0'), fail.
```
Chapter 5
Input/Output and Other Built-In Predicates

5.0 Overview
- In Prolog, besides see/1…seen/0 and tell/1…told/0, fail can also be handled using open/2 and close with close/0.
- fread/4 and fwrite/4 to read formatted input and output.

5.1 Opening/Closing File
- Open file using predicate open/2.
  - Format
    
    open(File, Mode).
  
  - Mode – read, write and 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.
  
  - 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.
  
- 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.

### 5.2 Setting Input/Output Stream

#### 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 |

- **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
  ```

#### Checking Input Stream

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

  ```
  seeing(Stream)
  ```
Example:

?- output(0).
   yes
?- output(1).
   yes
?- output(2).
   yes
?- output('file.pl').
   yes

?- seeing(X).
   X = user
?- seeing(X).
   X = 1
?- seeing(X).
   X = 2
?- seeing(X).
   X = 'file.pl'

Setting Output Stream

- Using output/1 - set output to the screen, a file or a string
- Format
  
  output(Stream).

- Code for input/1

<table>
<thead>
<tr>
<th>Console</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOS</td>
<td>1</td>
</tr>
<tr>
<td>Terminal</td>
<td>2</td>
</tr>
<tr>
<td>File</td>
<td>File Name</td>
</tr>
</tbody>
</table>

- Examples:

  ?- output(0).
     yes  ← set to write output to screen
  ?- output(1).
     yes  ← set to write output to DOS
  ?- output(2).
     yes  ← set to write output to terminal
  ?- output('output.pl').
     yes  ← set to write output to 'output.pl'
Checking Output Stream

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

- Syntax:
  
telling(Stream)

- Example:

```prolog
?- output(0).
yes
?- output(1).
yes
?- output(2).
yes
?- output('file.pl').
yes
```

5.3 Reading & Writing Formatted Data

Reading Formatted Data

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

- Syntax:

```prolog
read(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 <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 please refer to LPA Technical Reference pg: 102-109

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 (uses modifier)
    i integer
    n unsigned integer
    r arbitrary radix (uses modifier)
    s string (uses modifier)

Example please refer to LPA Technical Reference pg: 111-118

5.4 Term and Case Conversion

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

  atom_chars(Atom, CharList)

  Atom <variable> or <atom>
  CharList <char_list> or <variable>
- example:

  ?- atom_chars(eat, CharList ).
  CharList = [101,97,116]

  Atom = eat

- atom_string/2 - convert between an atom and a string

  atom_string(Atom, String)

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

- example:

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

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

- number_atom/2 - convert between a number and an atom

  number_atom(Number, Atom)

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

- example:

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

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

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

  number_chars(Number, CharList)

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

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

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

number_string/2 - convert between a number and a string

number_string(Number, String )

Number <number> or <variable>
String <string> or <variable>

example:

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

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

string_chars/2 - convert between strings and character lists

string_chars( String, CharList)

String <string> or <variable>
CharList <char_list> or <variable>

example:

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

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

lwrupr/2 - convert between lower and upper case

lwrupr(Lower,Upper)

Lower <atom>, <string> or <variable>
Upper <atom>, <string> or <variable>
example:

?- lwrupr(eat,Upper).
Upper = 'EAT'

?- lwrupr(Lower,'EAT').
Lower = eat

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

Term =.. List

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

example:

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

?- P =.. [eat,ahmad,rice].
P = eat(ahmad,rice)
Chapter 6
Window Programming in Prolog

6.0 Overview

- Dialog Editor – toolkit to develop interactive GUI.
- A dialog consists of:
  - Dialog window with GUI elements and its controls.
  - Dialog message handler – to interpret the control messages.

- Types of dialog:
  - Modal dialog
  - Modeless dialog

- Modal dialog
  - Once active, it wait user to response. Other windows will be inactive mode.
  - Example: use to accept input or for verification purposes (eg: Ok, Yes, No, Cancel, etc.).
6.1 Designing GUI using Prolog

- Starting editor (Run > Dialog Editor)
- Click and draw

- Changing properties

- Double Click
Preview Dialog

Export Source Code

Source Code

Click Edit

Click Export
6.2 Object Classes for Dialogs

- Two types of classes:
  - **Window class** - A framework for a window (skeleton) in which dialogs can be developed.
  - **Control class** - Objects embedded on a window and to control the behaviour of the dialog.

- Window classes
  - Attribute for the window class is determined by user.
Control classes
- Button class
- Edit Class
- Listbox Class
- Comobbox Class
- Static Class
- Grafix Class
- Scrollbar class

Button class
- **Edit class**

![Edit class diagram](image)

- **Listbox class**

![Listbox class diagram](image)

- **Combobox class**

![Combobox class diagram](image)
### 6.3 Message Passing Mechanism

- Message passing – to coordinate windows functions.
- Information passes between processes – windows messages.
- Windows messages: contains – commands, parameters or notification of actions.

### 6.4 Message Handling Mechanism

- Message is passes to Win Prolog to perform user specific actions.
- Message handler will catch the message and do the processing.
- Each dialog has its own handler.
- How message handling work?
  1. A message interrupts the execution of Prolog program.
  2. Program control is passed to a Message Handler – a specified action.
  3. Prolog code associated with the Message Handler is executed.
  4. Once completed, the control is passed back to the ain program (return to the dialog/window).

To define a relation between window and its handlers.
- **Window Handlers** - `window_handler/2`.
  - Get or set the current message handler for the given window

  ```prolog
  window_handler(Window, Handler)
  ```

  Window – name of the window created.
  Handler – state a name as the handler.

- The handler name will become the name for predicate handler/4.

  ```prolog
  handler(Win, Msg, Data, Result)
  ```

  Win – name of the window
  Data – suitable data
  Msg – message receive
  Result – variable

- General message names

  ```prolog
  `msg_menu` `msg_sysmenu` `msg_close` `msg_focus`  
  `msg_fuzzy` `msg_change` `msg_button` `msg_select`  
  `msg_double` `msg_size` `msg_move` `msg_horz`  
  `msg_vert` `msg_paint` `msg_leftdown` `msg_leftdouble`  
  `msg_leftup` `msg_rightdown` `msg_rightdouble` `msg_rightup`  
  `msg_char` `msg_key` `msg_drop` `msg_mousedown`  ```

Page | 73
Example

```
test:-
    create_example_dialog, 
    window_handler(example, example_handler), 
    show_dialog(example).

(example_handler((example,1000), msg_button,_,Result):-
    wtext((example,8000), N1),
    wtext((example,8001), N2),
    number_string(Number1, N1),
    number_string(Number2, N2),
    Ht is Number1 + Number2,
    number_string(Ht,H),
    wtext((example,8002),H).
```

Example – source code for a dialog

```
create_example_dialog:-
    _S1 = [dlg_ownedbyprolog,ws_sysmenu,ws_caption],
    _S2 = [ws_child,ws_visible,ss_center],
    _S3 = [ws_child,ws_visible,ws_tabstop,ws_border,es_center,es_multiline,es_autohscroll,es_autovscroll],
    _S4 = [ws_child,ws_visible,ws_tabstop,bs_pushbutton],
    wdccreate( example, 'Example Dialog', 322, 199, 272, 161, _S1 ),
    wccreate( (example,10000), static, 'Masukkan dua nilai dan tekan butang dibawah', 20, 20, 230, 20, _S2 ),
    wccreate( (example,8002), edit, '0', 20, 110, 230, 20, _S3 ),
    wccreate( (example,8000), edit, '0', 20, 50, 60, 20, _S3 ),
    wccreate( (example,8001), edit, '0', 190, 50, 60, 20, _S3 ),
    wccreate( (example,1000), button, 'TAMBAH', 20, 80, 50, 20, _S4 ),
    wccreate( (example,1001), button, 'TOLAK', 80, 80, 50, 20, _S4 ),
    wccreate( (example,1002), button, 'DARAB', 140, 80, 50, 20, _S4 ),
    wccreate( (example,1003), button, 'BAHAGI', 200, 80, 50, 20, _S4 ).
```

Example - GUI
6.5 Related Built-in Predicates

Creating Window

- **wdcreate/7** - create a dialog window

```
wdcreate( Window, Title, Left, Top, Width, Height, Style)
```

Create a dialog window with the given Window, Title, Left - Top corner, Width - Height dimensions, and Style. The Window argument must be an atom. Style is a list of logical window styles which are combined to create the 32-bit integer which is passed directly to Windows. This predicate can only use the generic window styles. Note: at present all dialogs must include the style 'ws_popup' to allow the dialog to function correctly stand-alone.

- **wcreate/8** - create a window

```
wcreate( Window, Class, Title, Left, Top, Width, Height, Style)
```

The predicate wcreate/8 creates a window with the given Window name, Class, Title, Left - Top corner coordinates, Width - Height dimensions and Style. The window will be created in one of several styles, depending upon the given handle and style: if the handle is an atom, a top level window is created; if the handle is a conjunction of the form (window,id), then a control window with the given ID is created within the given window. The Class argument defines the type of window to be created; it may be either a predefined Windows class or an LPA defined window class. The Style argument is a 32-bit integer that specifies an available window style which is passed directly to Windows. For top level windows (or MDI children), the text argument forms the window title (style permitting) of the window; for control windows, the text is the label of the control (where appropriate).

- **wccreate/8** - create a control window

```
wccreate( Window, Class, Title, Left, Top, Width, Height, Style)
```

Create a control Window with the given Class, Title, Left - Top corner coordinates, Width - Height dimensions, and Style. The Window argument is of the form (Parent,ID), where Parent is the handle of a top-level window and ID is the handle of the control window. The Class argument is one of the predefined control window classes. The Style argument is a list of logical window styles which are combined to create the 32-bit integer which is passed directly to Windows. This predicate can combine any of the generic window styles with the styles for the given class. Note: you should always include the ws_child style in the Style list.
wtcreate/6 - create a text window

wtcreate( Name, Title, Left, Top, Width, Height)

Create a "text" window with the given Name, Title, Left - Top corner coordinates, Width and Height dimensions. Name should be an atom which is used from then on to refer to the window. Text windows contain an "edit" field that is automatically resized according to the resizing of the window.

Example:

wucreate/6 - create a user MDI window

wucreate( Name, Title, Left, Top, Width, Height)

Create a "user" MDI window with the given Name, Title, Left - Top corner coordinates, Width and Height dimensions. Name should be an atom which is used from then on to refer to the window. User windows are created with a system menu, a hide button, a maximize button and are re-sizeable. They do not contain any other controls. Note: if you put any control items in a user MDI window you must write your own code to handle the re-sizing of the window.

Example:
Button Class

- `wbtnsel/2` - get or set selection state of a button

```
wbtnsel(Window, Status)
```

Get or set the selection status of the given radio or checkbox "button". The Window argument is the handle of the button. The Status argument is a button status value.

<table>
<thead>
<tr>
<th>Value</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>radio button deselected or checkbox unchecked</td>
</tr>
<tr>
<td>1</td>
<td>radio button selected or checkbox checked</td>
</tr>
</tbody>
</table>

Example: `wbbtnsel((example,100), Status)`

Listbox Class

- `wlbxadd/3` - add an item to a list box

```
wlbxadd(Window, Position, String)
```

Add a String to the "listbox" control Window at the given Position. If the position is given as -1, the item is inserted or appended to the list box depending upon the list box style. Entries in a listbox are numbered from 0.

Example: `wlbxadd((user_dialog,4000), 0, `item1`).`

- `wlbxdel/2` - delete an item from a list box

```
wlbxdel(Window, Position)
```

Delete the item at the given Position in the given "listbox" control Window. Entries in a listbox are numbered from 0.

Example: `wlbxdel((user_dialog,4000), 0)`.
- **wlbxfnd/4** - find a string in a list box

  \[\text{wlbxfnd(Window, Start, String, Position)}\]

  Return the Position of a partial match String in the given "listbox" control Window, starting search one place after the given Start. Entries in a listbox are numbered from 0. If String is the empty string `` then wlbxfnd/4 will return the Position of the entry following the Start.

  example: wlbxfnd((example,4000), 0, String, Position).

- **wlbxget/3** - get an item from a list box

  \[\text{wlbxget(Window, Position, String)}\]

  Get the String at the given Position in the given "listbox" control Window. Entries in a listbox are numbered from 0. This predicate will fail if the listbox has a number of entries that is less than or equal to the given position.

  example: wlbxget((example,4000), 2, String).

- **wlbxsel/3** - get or set selection in a list box

  \[\text{wlbxsel(Window, Position, State)}\]

  Set or get the selection State of the item at the given Position in the given "listbox" control Window. If the Position is given as -1, and the listbox is a multi-choice list box, the selection state is applied to all items. Entries in a listbox are numbered from 0. The State argument is a variable or a listbox selection state value.

  example: wlbxsel((example,4000), 2, 1).

**Editbox Class**

- **wedtsel/3** - get or set selection in an "edit" control window

  \[\text{wedtsel( Window, First, Second)}\]

  Sets the text selection area in the given "edit" control Window to start and finish at the given First and Second values, or returns the existing values. This predicate causes a direct side effect on the window, whose cursor moves to the position specified. Note that the start and finish positions can be given in either order: the flashing caret is positioned at the end specified by the First value. Windows does not
provide the caret position when retrieving the selection: the smallest value is always returned in the First parameter.

example: wedtsel((example,800), 11, 15).

wedtfnd/6 - find a text string in an "edit" control window

wedtfnd( Window, Start, End, String, StartMatch, EndMatch)

Search the given "edit" control Window for the given text String within the given Start and End points. The start and finish of the first matching string is returned as a pair of integers, StartMatch and EndMatch. As a special case, the search text may be specified as an empty string. In this case, the start and finish of the next space-delimited token is returned. No side effects are caused by this predicate, which is used for the gathering of information only. The returned parameters may be passed directly into wedtsel/3 if it is desired to move the selection to the found string.

wedtlm/4 - get offsets a line in an "edit" control window

wedtlm( Window, Offset, Start, Finish)

Returns the Start and Finish of the line of text containing the given character offset in the given "edit" control Window, or tests the given values for correctness. The offsets returned include everything on the given line, but not the carriage return/line feed. No side effects are caused by this predicate, which is used for the gathering of information only.

wedtpxy/4 - convert between linear offset and x, y coordinates in "edit" windows

wedtpxy( Window, Offset, X, Y)

Returns the X and Y coordinates that are the equivalent of a given character Offset, or returns the character Offset of the given X and Y values, or tests the given values for correctness. The values are computed for the given "edit" Window. No side effects are caused by this predicate, which is used for the gathering of information only.
- **wcount/4** - get char, word and line counts for the given window

```
wcount( Window, Characters, Words, Rows)
```

Returns the number of Characters, Words and Rows in the given Window. No side effects are caused by this predicate, which is used for the gathering of information only.

- **wtext/2** - get or set the window text

```
wtext(Window, Text)
```

Replace the text of the given Window to the given Text, or get the current Text. For top level and MDI child windows, the text is the window title (style permitting); for "button" and "static" control windows it is the window label, and for "edit" control windows and the "edit" control components of "combobox" windows, it is the entire window contents. Note that, unlike wedtxt/2, this predicate works with all types of window, but instead of replacing the current selection it replaces the entire text.

```
example: wtext((example,8002),`item1`).
```

**Message Box**

- **msgbox/4** - display the message box

```
msgbox(Title, Message, Style, Button)
```

Display a standard Windows message box with a given title, message and style returning the users response to the dialog. The Title argument is a string that sets the message box's window caption. The Message argument is either an atom or string that is the message to be shown to the user. The Style argument is a message box style value that dictates which combination of predefined buttons, icons and modality is used in the message box. The Button argument is a variable that gets bound to an integer indicating which button was used to terminate the dialog.

The predicate succeeds whichever button is clicked, or when RETURN is pressed.

Example:
message_box/3 - create a message box and return a response

message_box(Buttons, Message, Response).

Display a Message in a window with the specified Buttons (in the domain {ok, okcancel, yesno, yesnocancel}) and return the selected button in Response. The returned selected button may be one of the following: "ok", "cancel", "yes" or "no".

Example:

About Box

abtbox/3 - display the about box

abtbox(Title, Message, Font)

Displays the "about" dialog box with the given title (window caption) and message, using the given font.

The predicate succeeds if the 'OK' button is clicked or the <return> key is pressed, or fails if the dialog is closed with 'Close' system menu option or the <escape> key is pressed.

Example:
**abtbox/3 styles**

<table>
<thead>
<tr>
<th>Value</th>
<th>Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>fixed IBM PC font with large Prolog bitmap</td>
</tr>
<tr>
<td>1</td>
<td>proportional Windows font with large Prolog bitmap</td>
</tr>
<tr>
<td>2</td>
<td>fixed IBM PC font without bitmap</td>
</tr>
<tr>
<td>3</td>
<td>proportional Windows font without bitmap</td>
</tr>
</tbody>
</table>

Styles 4-7 have the same attributes as above except that the window displayed is wider. In this format a smaller LPA bitmap is displayed on the left hand side of the window and the text is displayed on the right.