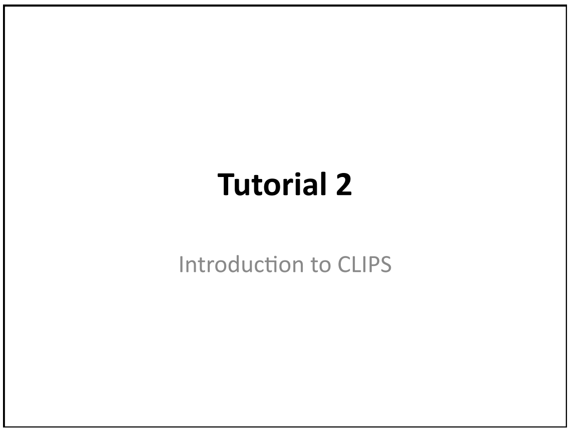
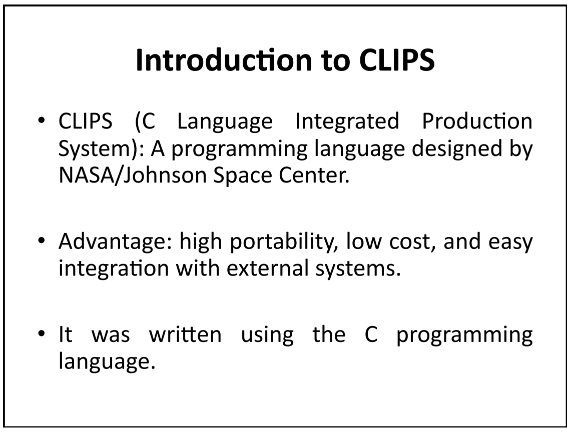


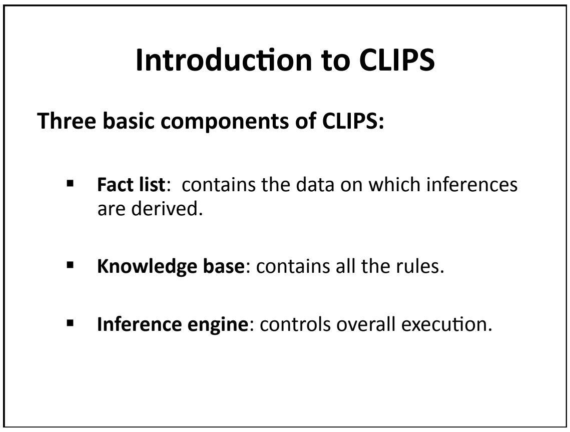
Tutorial 2

Introduction to CLIPS



Introduction to CLIPS

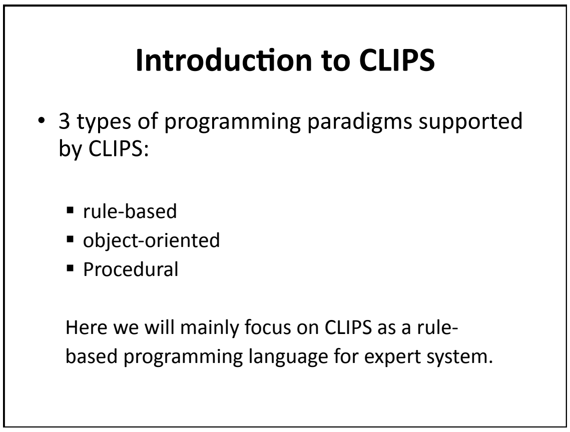
- CLIPS (C Language Integrated Production System): A programming language designed by NASA/Johnson Space Center.
- Advantage: high portability, low cost, and easy integration with external systems.
- It was written using the C programming language.



Introduction to CLIPS

Three basic components of CLIPS:

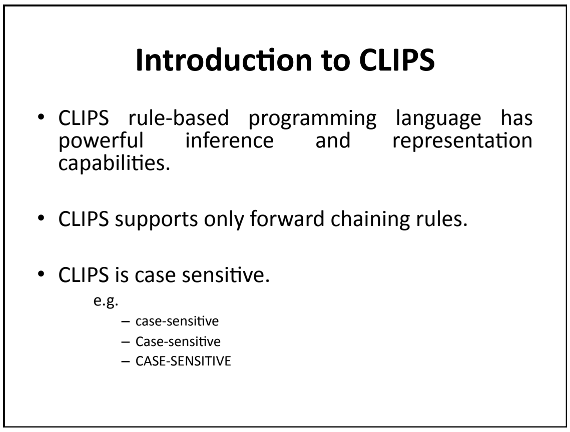
- **Fact list**: contains the data on which inferences are derived.
- **Knowledge base**: contains all the rules.
- **Inference engine**: controls overall execution.



Introduction to CLIPS

- 3 types of programming paradigms supported by CLIPS:
  - rule-based
  - object-oriented
  - Procedural

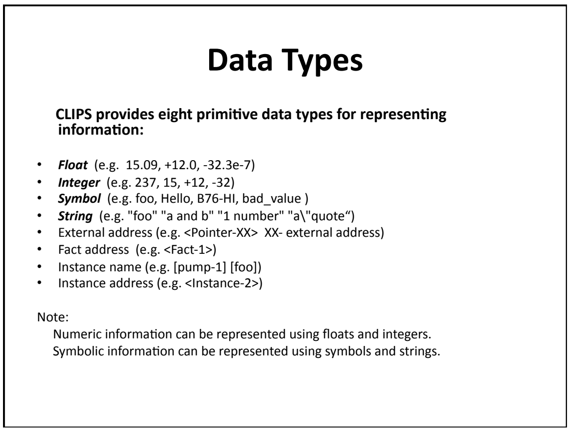
  Here we will mainly focus on CLIPS as a rule-based programming language for expert system.



Introduction to CLIPS

- CLIPS rule-based programming language has powerful inference and representation capabilities.
- CLIPS supports only forward chaining rules.
- CLIPS is case sensitive.

  e.g.
  - case-sensitive
  - Case-sensitive
  - CASE-SENSITIVE



Data Types

CLIPS provides eight primitive data types for representing information:

- ***Float*** (e.g. 15.09, +12.0, -32.3e-7)
- ***Integer*** (e.g. 237, 15, +12, -32)
- ***Symbol*** (e.g. foo, Hello, B76-HI, bad_value)
- ***String*** (e.g. "foo" "a and b" "1 number" "a\"quote“)
- External address (e.g. <Pointer-XX> XX- external address)
- Fact address (e.g. <Fact-1>)
- Instance name (e.g. [pump-1] [foo])
- Instance address (e.g. <Instance-2>)

Note:
Numeric information can be represented using floats and integers.
Symbolic information can be represented using symbols and strings.

Facts

There are three types of facts:

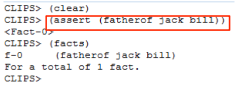
- Ordered facts
- Non-ordered facts
- Initial facts

Ordered Facts

Ordered facts : consist of a symbol followed by a sequence of zero or more fields separated by spaces and delimited by parentheses. And function **assert** adds facts to CLIPS's fact-list.

e.g. (father- of jack bill)

states that bill is the father of jack



```
CLIPS> (clear)
CLIPS> (assert (fatherof jack bill))
<Fact-0>
CLIPS> (facts)
f-0     (fatherof jack bill)
For a total of 1 fact.
CLIPS>
```

Defining a set of Facts

Use the **deffacts** construction to define a set of facts. Defining a set of facts is not the same as asserting that they are TRUE. In order to do this you use the **reset** function.

e.g.

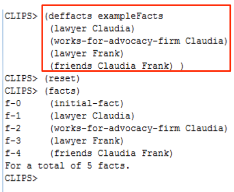
(**deffacts** exampleFacts

(lawyer Claudia)

(works-for-advocacy-firm Claudia)

(lawyer Frank)

(friends Claudia Frank))



```
CLIPS> (deffacts exampleFacts
        (lawyer Claudia)
        (works-for-advocacy-firm Claudia)
        (lawyer Frank)
        (friends Claudia Frank) )
CLIPS> (reset)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (lawyer Claudia)
f-2     (works-for-advocacy-firm Claudia)
f-3     (lawyer Frank)
f-4     (friends Claudia Frank)
For a total of 5 facts.
CLIPS>
```

Initial Facts

- The deffacts construct allows a set of *a priori* or initial knowledge to be specified as a collection of facts.
- When the CLIPS environment is reset (using the reset command) every fact specified within a deffacts construct in the CLIPS knowledge base is added to the factlist which includes the initial fact (e.g. fact-0 initial fact).

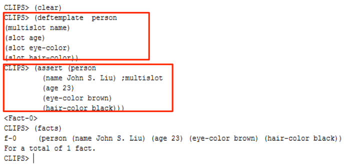
Non-ordered Facts

Non-ordered (or deftemplate) facts: provide the user with the ability to abstract the structure of a fact by assigning names to each field in the fact. The **deftemplate construct** is used to create a template which can then be used to access fields by name.

e.g.

(**deftemplate** person
(multislot name)
(slot age)
(slot eye-color)
(slot hair-color))

(**assert** (person
    (name John S. Liu) ;multislot
    (age 23)
    (eye-color brown)
    (hair-color black)))



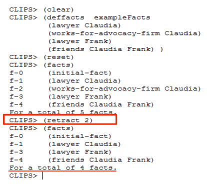
```
CLIPS> (clear)
CLIPS> (deftemplate  person
(multislot name)
(slot age)
(slot eye-color)
(slot hair-color))
CLIPS> (assert (person
        (name John S. Liu) ;multislot
        (age 23)
        (eye-color brown)
        (hair-color black)))
<Fact-0>
CLIPS> (facts)
f-0     (person (name John S. Liu) (age 23) (eye-color brown) (hair-color black))
For a total of 1 fact.
CLIPS> |
```

Remove Facts

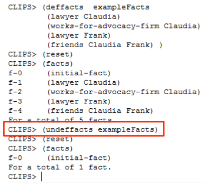
- Use command **retract** to remove one fact

  e.g. (retract 0) ; "0" is fact index in fact-list
- Use command **undeffacts** to remove a set of facts

  e.g. (undeffacts exampleFacts)



```
CLIPS> (clear)
CLIPS> (deffacts  exampleFacts
        (lawyer Claudia)
        (works-for-advocacy-firm Claudia)
        (lawyer Frank)
        (friends Claudia Frank) )
CLIPS> (reset)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (lawyer Claudia)
f-2     (works-for-advocacy-firm Claudia)
f-3     (lawyer Frank)
f-4     (friends Claudia Frank)
For a total of 5 facts.
CLIPS> (retract 2)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (lawyer Claudia)
f-3     (lawyer Frank)
f-4     (friends Claudia Frank)
For a total of 4 facts.
CLIPS> |
```



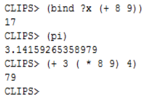
```
CLIPS> (deffacts  exampleFacts
        (lawyer Claudia)
        (works-for-advocacy-firm Claudia)
        (lawyer Frank)
        (friends Claudia Frank) )
CLIPS> (reset)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (lawyer Claudia)
f-2     (works-for-advocacy-firm Claudia)
f-3     (lawyer Frank)
f-4     (friends Claudia Frank)
For a total of 5 facts.
CLIPS> (undeffacts exampleFacts)
CLIPS> (reset)
CLIPS> (facts)
f-0     (initial-fact)
For a total of 1 fact.
CLIPS> |
```

Functions

A function in CLIPS is a piece of executable code identified by a specific name which returns a useful value or performs a useful side effect (such as displaying information).

e.g. math functions : (+ 3 (* 8 9) 4), (pi), (sqrt 9)

procedural functions : (bind ?x (+ 8 9)), If..Then..Else



```
CLIPS> (bind ?x (+ 8 9))
17
CLIPS> (pi)
3.14159265358979
CLIPS> (+ 3 ( * 8 9) 4)
79
CLIPS>
```

Constructs

- Defining a construct is intended to alter the CLIPS environment by adding to the CLIPS knowledge base. However, a function call leaves the CLIPS environment unchanged. (with exceptions of *reset* or *clear*).
- Unlike function calls, constructs never have a return value.
- All constructs in CLIPS are surrounded by parentheses.

e.g. ***deffacts***
deftemplate
defrule

Defrule Construct

- In rule-based expert system, rules defined using the **defrule** construct.
- If no conditional elements (CE) are on the LHS, the initial- fact is automatically used. If no actions are on the RHS, the rule can be activated and fired but nothing will happen.

Syntax

```
    (defrule <rule-name> [<comment>]
            <conditional-element>* ; Left-Hand Side (LHS)
        =>
             <action>*) ; Right-Hand Side (RHS)
```

e.g.

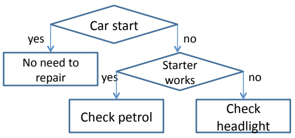
```
   (defrule example-rule "This is an example of a simple rule"
             (refrigerator light on)
             (refrigerator door open)
        =>
             (assert (refrigerator food spoiled)))
```

Comments

- As with any programming language, it is highly beneficial to comment CLIPS code. All constructs (with the exception of defglobal) allow a comment directly following the construct name.
- Comments can be placed in parentheses (“ ”). Everything within parentheses will be ignored by CLIPS.
- Comments also can be placed within CLIPS code by using a semicolon (;). Everything from the semicolon until the next return character will be ignored by CLIPS.

Exercise

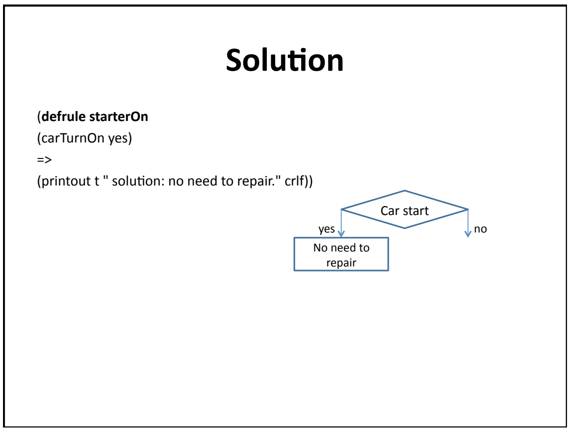
How to implement this decision tree in CLIPS



Solution

```
(defrule carDiagnosis
=>
(printout t "does car start? (1-yes, 0-no)" crlf)
(bind ?x (read))
(if (= ?x 1)
    then (assert (carTurnOn yes))
    else (assert (carTurnOn no))))
```

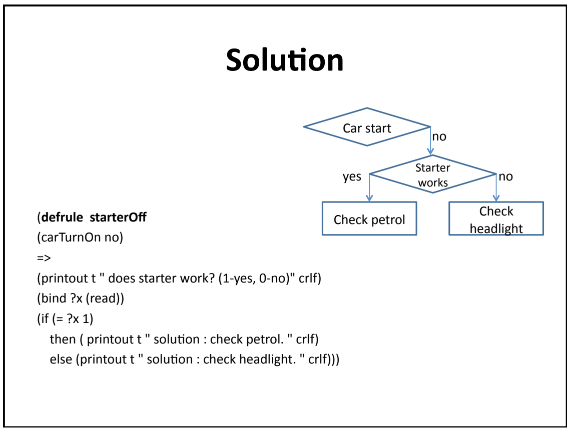
Car start
yes
no



Solution

```
(defrule starterOn
(carTurnOn yes)
=>
(printout t " solution: no need to repair." crlf))
```

Car start
yes
no
No need to repair



Solution

Car start
no
Starter works
yes
no
Check petrol
Check headlight

```
(defrule  starterOff
(carTurnOn no)
=>
(printout t " does starter work? (1-yes, 0-no)" crlf)
(bind ?x (read))
(if (= ?x 1)
   then ( printout t " solution : check petrol. " crlf)
   else (printout t " solution : check headlight. " crlf)))
```