Artificial Intelligence

Section 3

Clips

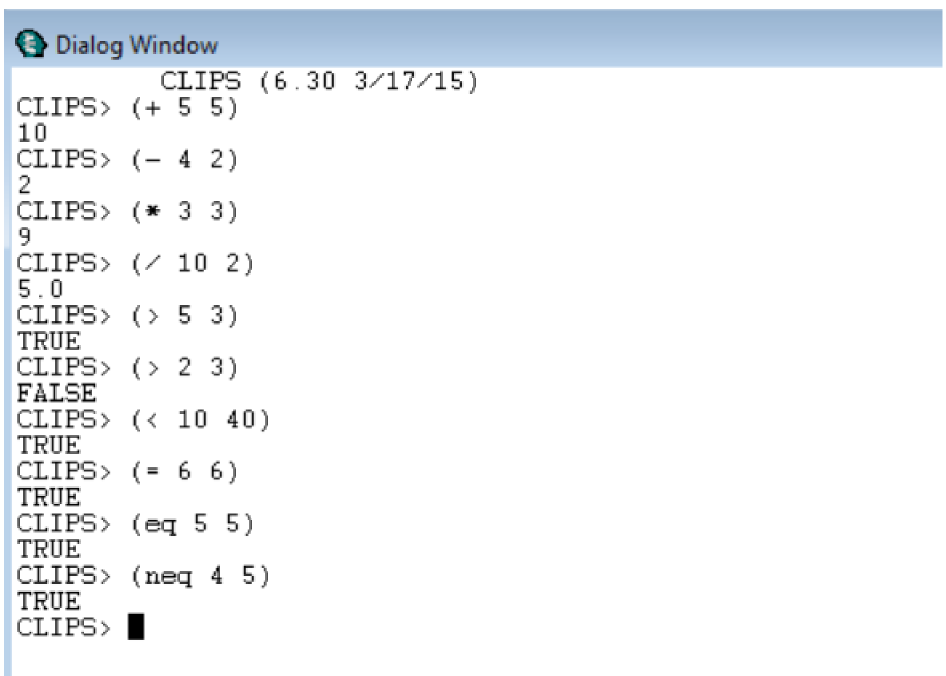
i. What is clips?

1. **CLIPS** (**C Language Integrated Production System**) is a public-domain software tool for building Expert Systems.
2. **CLIPS is a multiparadigm programming language** that provides support for (Rule-based – Object-oriented – Procedural programming).
3. **CLIPS** supports only forward-chaining rules.
4. **CLIPS** is case sensitive.

Core Components of a CLIPS Expert System:

- **Knowledge Base:** Contains the rules and objects that define the system's expertise.
- **Fact-List:** A global memory where data and information are stored as facts.
- **Inference Engine:** The component that controls the execution of rules, deciding which rules to activate and when.

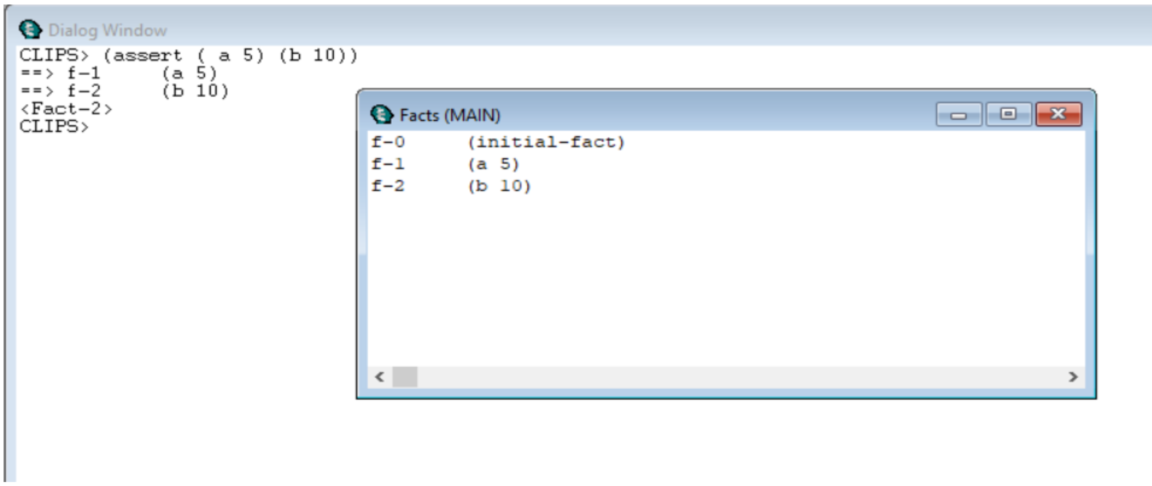
ii. Operations in clips

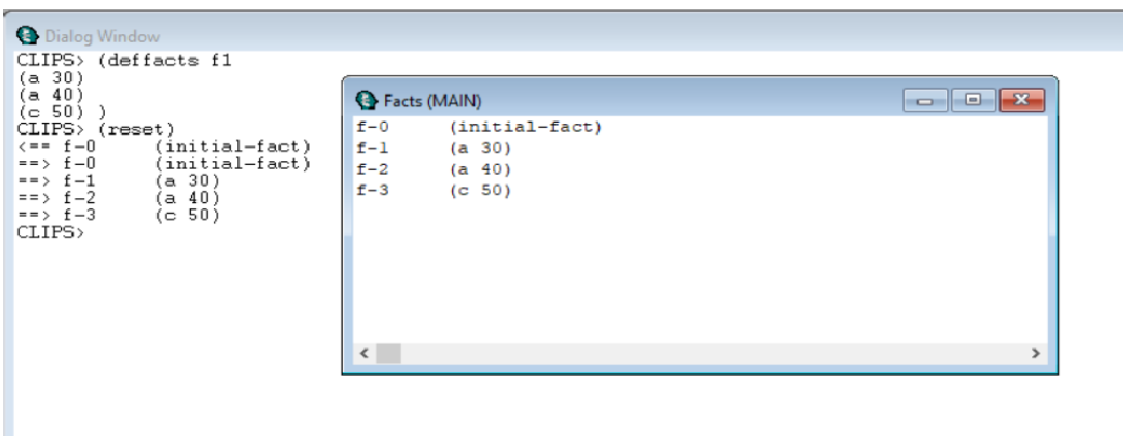


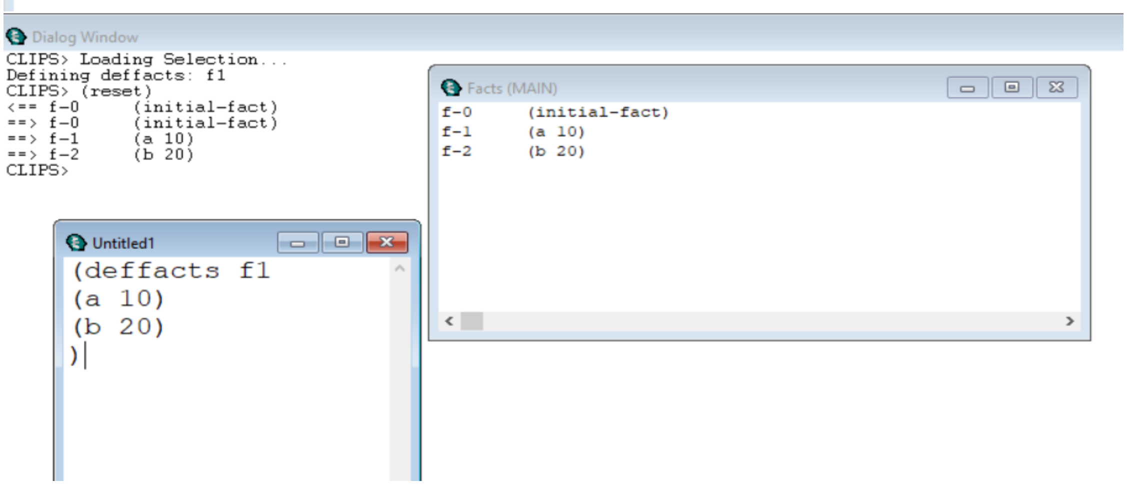

```
         CLIPS (6.30 3/17/15)
CLIPS> (+ 5 5)
10
CLIPS> (- 4 2)
2
CLIPS> (* 3 3)
9
CLIPS> (/ 10 2)
5.0
CLIPS> (> 5 3)
TRUE
CLIPS> (> 2 3)
FALSE
CLIPS> (< 10 40)
TRUE
CLIPS> (= 6 6)
TRUE
CLIPS> (eq 5 5)
TRUE
CLIPS> (neq 4 5)
TRUE
CLIPS>
```

iii. Facts in clips:

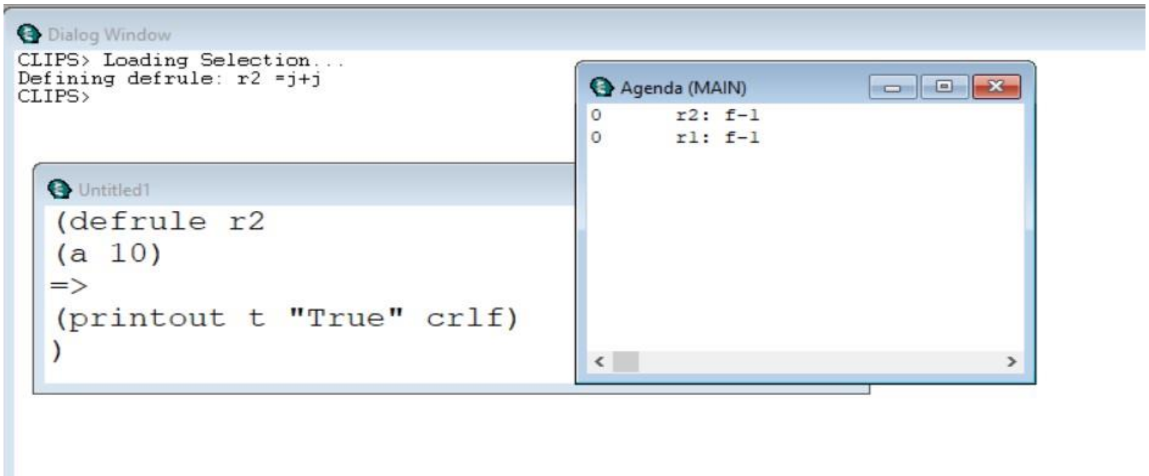
1) Ordered Facts:
   a) Using assert ()
   b) Using deffacts ()

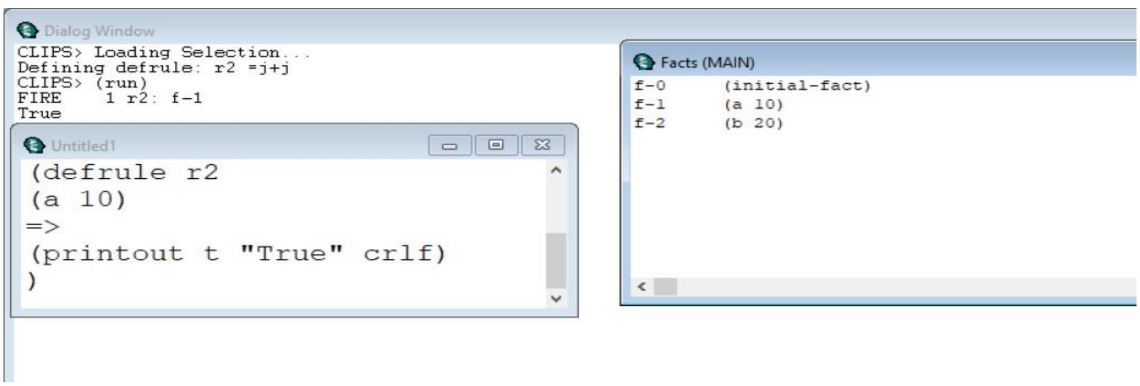




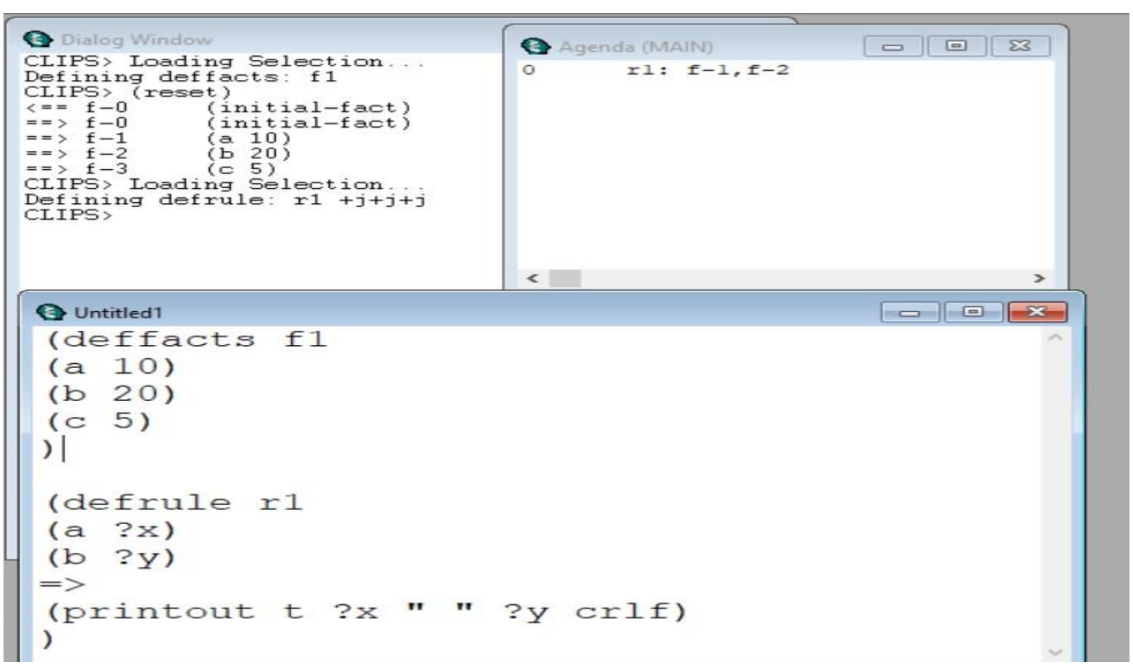


iv. Q3. Create rules in clips:





v. Print values of facts using rules:



Dialog Window

```
CLIPS> Loading Selection...
Defining deffacts: f1
CLIPS> (reset)
<== f-0     (initial-fact)
==> f-0     (initial-fact)
==> f-1     (a 10)
==> f-2     (b 20)
==> f-3     (c 5)
CLIPS> Loading Selection...
Defining defrule: r1 +j+j+j
CLIPS> (run)
FIRE    1 r1: f-1,f-2
10 20
CLIPS>
```

Facts (MAIN)

```
f-0     (initial-fact)
f-1     (a 10)
f-2     (b 20)
f-3     (c 5)
```

Untitled1

```
(deffacts f1
(a 10)
(b 20)
(c 5)
)

(defrule r1
(a ?x)
(b ?y)
=>
(printout t ?x " " ?y crlf)
)
```

Dialog Window

```
CLIPS> (run)
FIRE    1 r2: f-6,f-4,f-5
5 50 5
FIRE    2 r2: f-6,f-3,f-5
5 90 5
FIRE    3 r2: f-2,f-3,f-5
50 90 5
FIRE    4 r2: f-1,f-3,f-5
10 90 5
FIRE    5 r2: f-2,f-4,f-5
50 50 5
FIRE    6 r2: f-1,f-4,f-5
10 50 5
FIRE    7 r1: f-6
Yes
CLIPS>
```

Facts (MAIN)

```
f-0     (initial-fact)
f-1     (a 10)
f-2     (a 50)
f-3     (b 90)
f-4     (b 50)
f-5     (c 5)
f-6     (a 5)
```

Untitled2

```
(defrule r1
(a 5)
=>
(printout t "Yes" crlf) )

(defrule r2
(a ?x)
(b ?y)
(c ?z)
=>
(printout t ?x " " ?y " " ?z " " crlf)
)
```

vi. Add value to variable using bind:

```
CLIPS> (bind ?x 20)
20
CLIPS> ?x
20
CLIPS> (bind ?y (create$ red green blue))
(red green blue)
CLIPS> ?y
(red green blue)
```

vii. Add value using global variable:

```
CLIPS> (defglobal ?*a* = 10)
CLIPS> ?*a*
10
CLIPS> (defglobal ?*b* = (create$ 5 10 15))
CLIPS> ?*b*
(5 10 15)
CLIPS> (printout t "Expert System" crlf)
Expert System
CLIPS>
```

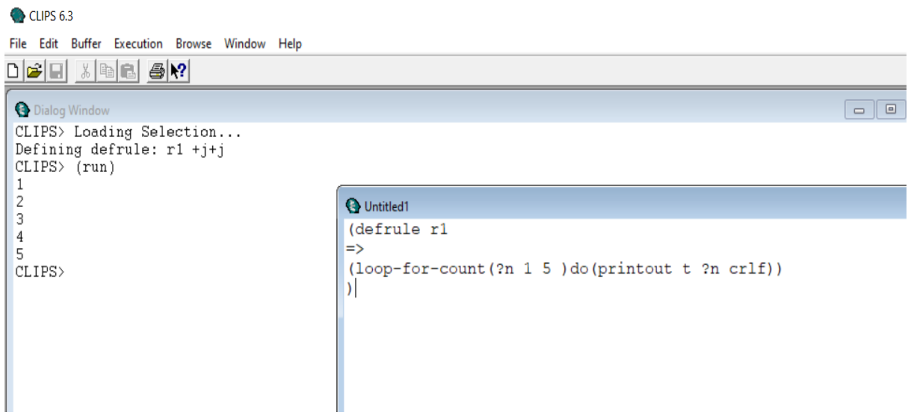
viii. sum two numbers:

```
CLIPS> Loading Selection...
Defining defrule: sum +j+j
CLIPS> (run)
FIRE    1 sum: *
15
CLIPS>
```

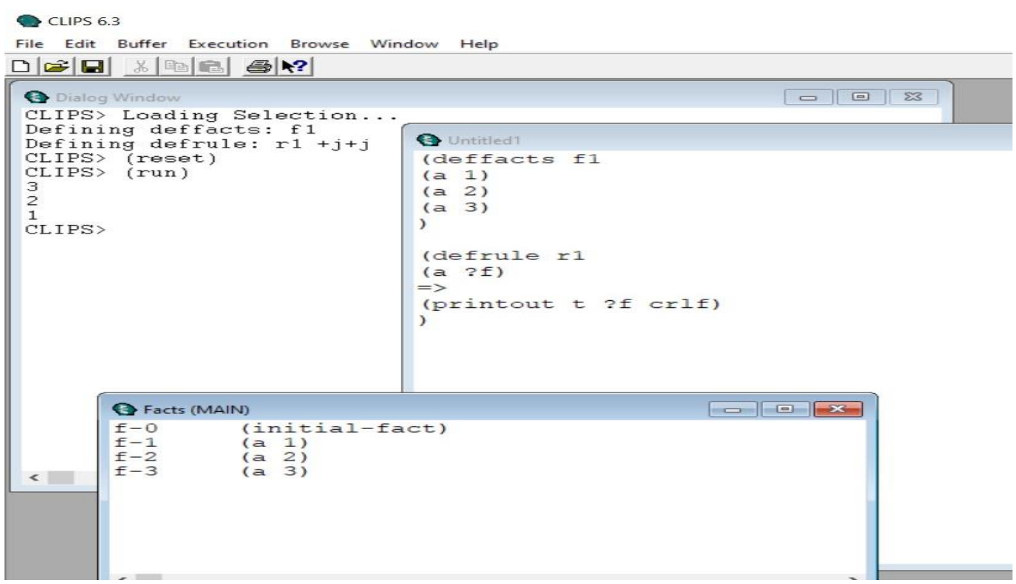
```
(defrule sum
=>
(bind ?x 5)
(bind ?y 10)
(printout t (+ ?x ?y) crlf)
)
```

ix. Using rules to create loop:

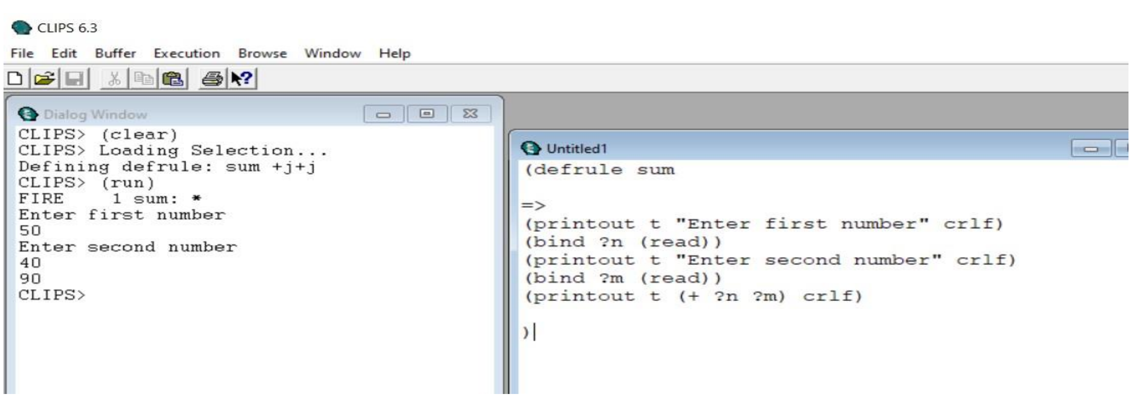
We use built-in function called (loop-for-count)



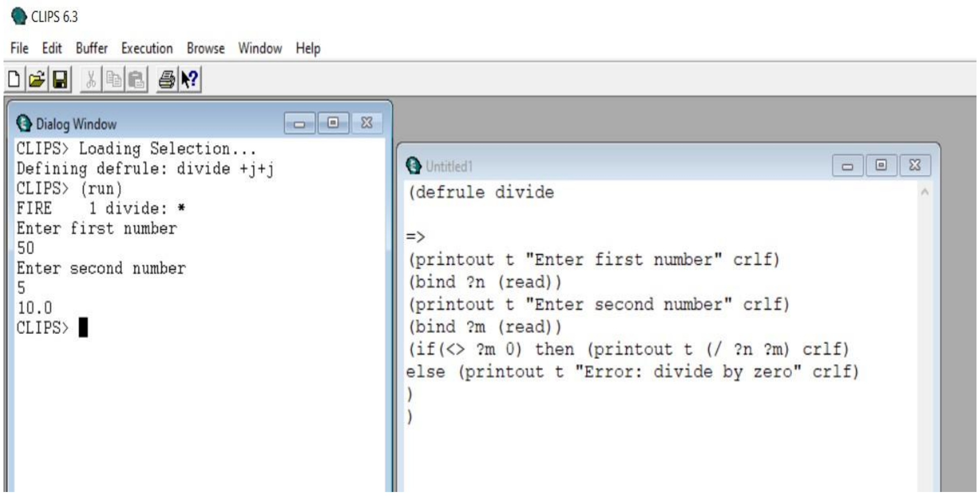
x. Using rules to show facts:



xi. Using rules to summation two numbers:



xii. Dividing Two numbers using Rule:



xiii. Multi-field functions in clips:

- Creating multi-field values **(create$)**
  Ex: (create$ red green blue)
- Specifying multi-field values **(nth$)**
  Ex: (nth$ 2 (create$ 30 40 50))
  40
  Ex: (nth$ 4 (create$ 30 40 50))
  nil
- Finding an element multi-field value **(member$)**
  Ex: (member$ yellow (create$ red green blue))
  FALSE
  Ex: (member$ green (create$ red green blue))
  2

xiv. Predicate functions:

- **lexemep**=> check if the argument is the symbol or string return true, otherwise return false.

  Ex: (lexemep Asmaa) =>true

  Ex: (lexemep "Mohamed") =>true

  Ex: (lexemep 10) =>false

- **symbolp**=> check if the argument is the symbol return true,

  otherwise it will return false.

  Ex: (symbolp Ahmed) => true

  Ex: (symbolp “Ahmed”) => false

- **wordp** => check if the argument is the symbol return true, otherwise return false.

  Ex: (wordp Asmaa) =>true

- **evenp** => check if the argument is even return true, otherwise return false.
  Ex: (evenp 6) => true
  Ex: (evenp 5) => false

- **oddp** => check if the argument is odd return true, otherwise return false.
  Ex: (oddp 3) => true
  Ex: (oddp 4) => false

- **Numberp** =>return the symbol true if its argument is a float or integer, otherwise return false
  Ex: (numberp 10) => true
  Ex: (numberp -5) => true
  Ex: (numberp Cairo) => false

- **floatp** => return the symbol true if its argument is a float, otherwise return false

  Ex: (floatp 5.3) => true
  Ex: (floatp -4.6) => true
  Ex: (floatp "hello") => false

- **integerp** => return the symbol true if its argument is integer, otherwise return false

  Ex: (integerp 5) => true
  Ex: (floatp "hello") => false

- **multifieldp** => function returns the symbol true if its argument is a multifield value, otherwise return false.
  Ex: (multifieldp M) => false
  Ex: (multifieldp (create$ a b c d)) => true
  Ex: (sequencep (create$10 20 30 40)) => true

xv. comparing equality and not equality

Ex: (eq M N) => false
Ex: (eq A A) => true
Ex: (eq 3 3 3) => true
Ex: (neq 10 20) => true
Ex: (neq 5 5) => false
Ex: (neq A B) => true
Ex: (neq A a) => true