MEDII DE PROGRAMARE IN INTELIGENTA ARTIFICIALA
Jess – definirea functiilor

1. Functii aritmetice built-in in Jess.
2. Definirea de utilizator a noi functii. Comanda deffunction
3. Functii recursive in Jess
4. Exercitii

1. Tabel cu cele mai utilizate functii aritmetice Jess.

Functii aritmetice built-in				
(+) (-) (*) (\ ) (**) (mod)	(=) (>) (<) (<=) (>=) (<>)	(and) (or) (not)	(random) (min) (max)	(sin) (cos) (tan) (sqrt)

2. Desi Jess are peste 200 de functii built-in este nevoie totusi ca utilizatorii sa-si poata defini propriile functii. Pentru a pastra uniformitatea limbajului, definitia unei noi functii se face tot prin intermediul unei liste. O astfel de lista are pe prima pozitie cuvantul **deffunction** pentru a indica faptul ca urmeaza o definire de functie. Numele functiei ce se defineste este indicat in cel de-al doilea parametru urmat de lista argumentelor formale ale functiei:

   (deffunction nume_functie (<lista_parametrii>) [<comentarii>] <bloc_instructiuni>)

   Exemplu de functie construita cu deffunction:

   Jess> (deffunction varsta (?an)

   ”Calculeaza varsta unei persoane”

   (bind ?varsta (- 2011 ?an))

   (return ?varsta))

Din momentul evaluarii unei definitii de functie de catre interpretorul Jess, functie definita devine *cunoscuta* si poate fi astfel apelata prin intermediul numelui atribuit la fel ca si functiile predefinite:

```
Jess> (varsta 1978)

32
```

Acesta este un exemplu de functie care nu utilizeaza functii de control. Un astfel de exemplu este dat mai jos:

```
Jess> (deffunction maxim (?a ?b)
        ”Calculeaza maximul dintre cei doi parametrii”
        (if (> ?a ?b) then
            (return ?a)
                else
            (return ?b)))
Jess> (maxim 20 17)
20
```

Functia de mai sus se poate scrie si mai simplu astfel:

```
Jess> (deffunction maxim (?a ?b)
        ”Calculeaza maximul dintre cei doi parametrii”
        (if (> ?a ?b) then
            ?a
                else
            ?b))
```

deoarece in mod implicit, orice functie in Jess returneaza valoarea ultimei expresii evaluate. In cazul in care am dori sa calculam maximul dintr-o lista de numere intregi pozitive, functia **maxim** devine:

```
Jess> (deffunction maxim ($?args)
          (bind ?max (nth$ 1 ?args))
          (foreach ?n ?args
                (if (> ?n ?max) then
```

(bind ?max ?n)))

?max)

TRUE

Jess> (maxim 10 100 77 6 43)

6

3. Iata un exemplu de functie recursiva definita folosind **deffunction** care implementeaza functia fibonacci.

$$fib(n) = \begin{cases} 1, n = 0 \ sau \ n = 1 \\ fib(n-1) + fib(n-2), altfel \end{cases}$$

Jess> (deffunction fib (?f)

(if (or (= ?f 1) (=? f 0)) then

1

else

(+ (fib (- ?f 1)) (fib (- ?f 2))))

TRUE

Jess> (fib 5)

8

4. **Concatenarea a doua liste**. Vom defini o functie care concateneaza doua liste simple (elementele celei de-a doua liste sunt adaugate la prima lista).

(deffunction concat (?list1 ?list2)

(return (create$?list1 ?list2))

Apelul acestei functii va avea urmatorul rezultat:

Jess> (bind ?list1 (create$ 1 2 3))

Jess> (bind ?list2 (create$ 2 3 7))

Jess> (concat ?list1 ?list2)

(1 2 3 2 3 7)

5. Exercitii:

   a. Definiti o functie care returneaza lista obtinuta din diferenta dintre lista din primul argument si lista din cel de-al doilea:

(deffunction list_diff (?list1 ?list2)

b. **Produsul scalar a doua liste**. Conditia pentru realizarea produsului este ca cele doua liste sa aiba aceeasi lungime. De exemplu produsul scalar pentru listele (1 2) si (3 4) trebuie sa returneze valoarea $1\times3 + 2\times4 = 11$.

(deffunction scalar_prod (?list1 ?list2)