Import i export činjenica

- Činjenice koje ubacujemo u listu činjenica CLIPS dodeljuje onom modulu u kome se nalazi deftemplate po kome su činjenice formirane.

```
CLIPS> (deftemplate detekcija::greska (slot komponenta))
CLIPS> (assert (greska (komponenta A)))
==> f-1     (greska (komponenta A))
<Fact-1>

CLIPS> (deftemplate izolacija::moguce-otkazivanje
                                          (slot komponenta)))
CLIPS> (assert (moguce-otkazivanje (komponenta B)))
==> f-2     (moguce-otkazivanje (komponenta B))
<Fact-2>

CLIPS> (set-current-module detekcija)
izolacija
CLIPS> (facts)
f-1     (greska (komponenta A))
For a total of 1 fact.

CLIPS>(facts *)
f-0     (initial-fact)
f-1     (greska (komponenta A))
f-2     (moguce-otkazivanje (komponenta B))
For a total of 3 facts.
```

Import i export činjenica

- Deftemplate konstrukcije mogu biti deljene između modula.
- Defrule i deffacts konstrukcije ne mogu!!!

Import i export činjenica

- Modul u kome je definisan određeni deftemplate “poseduje” sve činjenice napravljene po tom deftemplate-u.

- Modul može exportovati dati deftemplate, a ostali moduli moraju importovati taj deftemplate da bi sve činjenice napravljene po njemu postale vidljive u svim modulima.

Import i export činjenica

- Moduli koji treba da exportuju deftemplate konstrukcije moraju u svojoj definiciji imati **`export`** atribut. On može biti u nekom od sledećih formata:
  - **`(export ?ALL)`**
  - **`(export ?NONE)`**
  - **`(export deftemplate ?ALL)`**
  - **`(export deftemplate ?NONE)`**
  - **`(export deftemplate <naziv-deftemplate>+)`**

Import i export činjenica

- Moduli koji treba da importuju deftemplate konstrukcije moraju u svojoj definiciji imati **`import`** atribut. On može biti u nekom od sledećih formata:

  - **`(import <naziv-modula> ?ALL)`**
  - **`(import <naziv-modula> ?NONE)`**
  - **`(import <naziv-modula> deftemplate ?ALL)`**
  - **`(import <naziv-modula> deftemplate ?NONE)`**
  - **`(import <naziv-modula> deftemplate <naziv-deftemplate>+)`**

Import i export činjenica

- Konstrukcija deftemplate mora biti definisana pre nego što se navede uz import atribut.

- Pre definisanja deftemplate-a u nekom modulu, taj modul mora biti definisan, a samim tim i njegova export lista.

- Ova ograničenja obezbeđuju da dva modula ne mogu međusobno da importuju deftemplate (ako modul A importuje, od modula B, modul B ne može importovati od modula A)

- Kada je jednom definisan modul se ne može predefinisati.

- MAIN modul se može predefinisati jednom, kako bi se uključile export i import naredbe, naročito ybog exporta *initial-fact*-a.

Primer

```
(defmodule detekcija
  (export deftemplate greska))

(deftemplate detekcija::greska
  (slot komponenta))

(defmodule izolacija
  (export deftemplate moguce-otkazivanje))

(deftemplate izolacija::moguce-otkazivanje
   (slot komponenta))

(defmodule recovery
   (import detekcija deftemplate greska)
   (import izolacija deftemplate moguce-otkazivanje))
```

```
(deffacts detekcija::greske
   (greska (komponenta A)))

(deffacts izolacija::greske
   (moguce-otkazivanje (komponenta B)))

(deffacts recovery::greske
   (greska (komponenta C))
   (moguce-otkazivanje (komponenta D)))
```

```
CLIPS> (reset)

CLIPS> (facts detekcija)
f-1     (greska (komponenta A))
f-3     (greska (komponenta C))
For a total of 2 facts.

CLIPS> (facts izolacija)
f-2     (moguce-otkazivanje (komponenta B))
f-4     (moguce-otkazivanje (komponenta D))
For a total of 2 facts.

CLIPS> (facts recovery)
f-1     (greska (komponenta A))
f-2     (moguce-otkazivanje (komponenta B))
f-3     (greska (komponenta C))
f-4     (moguce-otkazivanje (komponenta D))
For a total of 4 facts.

CLIPS>
```

Moduli i kontrola toka programa

- Svaki definisani modul ima svoju agendu
- Redosled izvršavanja pravila se kontroliše odabirom odgovarajuće agende

Sva sledeća pravila će se aktivirati ako su ubačene činjenice kao u prethodnom primeru

```
(defrule detekcija::pravilo1
   (greska (komponenta A|C))=>)

(defrule izolacija::pravilo2
   (moguce-otkazivanje (komponenta B|D))=>)

(defrule recovery::pravilo3
   (greska (komponenta A|C))
   (moguce-otkazivanje (komponenta B|D))=>)
```

```
CLIPS> (agenda *)
MAIN:
detekcija:
   0      pravilo1: f-3
   0      pravilo1: f-1
izolacija:
   0      pravilo2: f-4
   0      pravilo2: f-2
recovery:
   0      pravilo3: f-1,f-4
   0      pravilo3: f-3,f-4
   0      pravilo3: f-3,f-2
   0      pravilo3: f-1,f-2
For a total of 8 activations.

CLIPS>(run)
CLIPS>
```

Naredba **`focus`**

- Trenutni fokus određuje čiju agendu će run naredba koristiti tokom izvršavanja programa.
- **`(reset)`** i **`(clear)`** automatski postavljaju fokus na MAIN modul.
- Trenutni fokus se ne menja promenom trenutnog modula.
- Za promenu važećeg fokusa koristi se naredba

`(focus <naziv-modula>+)`

- Upotrebom fokus naredbe se praktično vrši ubacivanje modula, čiji je naziv u argumentu naredbe naveden, na stek fokusa.

- Kada agenda modula koji je trenutno u fokusu postane prazna on se automatski skida sa steka fokusa i prelazi se na agendu modula koji je tada na vrhu steka.

```
CLIPS> (focus izolacija)
==> Focus izolacija from detekcija
TRUE

CLIPS> (focus recovery)
==> Focus recovery from izolacija
TRUE

CLIPS> (list-focus-stack)
recovery
izolacija
detekcija
MAIN
```

```
CLIPS> (run)

FIRE    1 pravilo3: f-1,f-4
FIRE    2 pravilo3: f-3,f-4
FIRE    3 pravilo3: f-3,f-2
FIRE    4 pravilo3: f-1,f-2
<== Focus recovery to izolacija
FIRE    5 pravilo2: f-4
FIRE    6 pravilo2: f-2
<== Focus izolacija to detekcija
FIRE    7 pravilo1: f-3
FIRE    8 pravilo1: f-1
<== Focus detekcija to MAIN
<== Focus MAIN

CLIPS>
```

```
CLIPS> (focus detekcija izolacija recovery)
==> Focus recovery
==> Focus izolacija from recovery
==> Focus detekcija from izolacija
TRUE
CLIPS> (list-focus-stack)
detekcija
izolacija
recovery

CLIPS> (focus recovery)
==> Focus recovery from detekcija
TRUE
CLIPS> (list-focus-stack)
recovery
detekcija
izolacija
recovery
```

Naredbe za rad sa fokus stekom

- **`(get-focus-stack)`**
- **`(get-focus)`**
- **`(pop-focus)`**
- **`(clear-focus-stack)`**

Naredba `return`

- Naredba return se upotrebljava za trenutni prekid desne strane pravila i uklanjanje važećeg fokusa sa fokus steka.

```
CLIPS> (defmodule MAIN
         (export deftemplate initial-fact))

CLIPS> (defmodule detekcija
         (import MAIN deftemplate initial-fact))

CLIPS> (defrule MAIN::start
          =>
         (focus detekcija))

CLIPS> (defrule detekcija::primer1
         =>
          (return)
          (printout t "PRAZNO" crlf))

CLIPS> (defrule detekcija::primer2
         =>
         (return)
         (printout t "PRAZNO" crlf))
```

```
CLIPS> (reset)
<== Focus MAIN
==> Focus MAIN

CLIPS> (run)
FIRE    1 start: f-0
==> Focus detekcija from MAIN
FIRE    2 primer1: f-0
<== Focus detekcija to MAIN
<== Focus MAIN
CLIPS>
```

Auto-focus

- Moguće je automatsko fokusiranje na neki modul kada su određena pravila iz tog modula aktivirana.

- Ovo se postiže navođenjem
  `(declare (auto-focus TRUE))` u pravilu

- Koristi se kod pravila koja detektuju narušavanje ograničenja, jer kad god se naruše ograničenja modul u kome je kontrolno pravilo prelazi u fokus, čime se odmah preduzima akcija, bez čekanja da se eksplicitno pređe u fazu uočavanja prekršaja

```
(defmodule detekcija)
(defmodule izolacija)
(defmodule recovery)

(deffacts MAIN::kontrolne-informacije
  (redosled-faza detekcija izolacija recovery))

(defrule MAIN::promena-faze
  ?lista <- (redosled-faza ?naredna-faza
$?ostale-faze)
   =>
  (focus ?naredna-faza)
  (retract ?lista)
  (assert (redosled-faza ?ostale-faze ?naredna-
faza)))
```

```
CLIPS> (reset)
CLIPS> (run 5)
FIRE    1 promena-faze: f-1
==> Focus detekcija from MAIN
<== Focus detekcija to MAIN
FIRE    2 promena-faze: f-2
==> Focus izolacija from MAIN
<== Focus izolacija to MAIN
FIRE    3 promena-faze: f-3
==> Focus recovery from MAIN
<== Focus recovery to MAIN
FIRE    4 promena-faze: f-4
==> Focus detekcija from MAIN
<== Focus detekcija to MAIN
FIRE    5 promena-faze: f-5
==> Focus izolacija from MAIN
<== Focus izolacija to MAIN
CLIPS>
```

Proceduralne funkcije u CLIPSu

- if...then...else
- while
- loop-for-count
- return
- break
- switch

If...then...else funkcija

```
(if <izraz>
        then <akcija_1>
                :
             <akcija_m>
      [else <akcija1>
                :
             <akcija_n>])
```

PRIMER 1:

```
(defrule nastavi-ili-ne
 ?faza <- (faza nastaviti-ili-ne)
  =>
 (retract ?faza)
 (printout t “Nastavljate? “)
 (bind ?odg (read))
 (if (or (eq ?odg da)(eq ?odg DA))
  then(assert (faza nastavak))
  else(assert (faza prekid))
 )
)
```

PRIMER 1:

```
CLIPS> (assert (faza nastaviti-ili-ne))
==> f-1     (faza nastaviti-ili-ne)
==> Activation 0      nastavi-ili-ne: f-1
<Fact-1>

CLIPS> (run)
<== f-1     (faza nastaviti-ili-ne)
Nastavljate? da
==> f-2     (faza nastavak)

CLIPS>
```

While

```
(while <izraz> [do]
                <akcija_1>
                   :
                <akcija_m>)
```

PRIMER 2:

```
(defrule nastavi-ili-ne
 ?faza <- (faza nastaviti-ili-ne)
  =>
 (retract ?faza)
 (printout t “Nastavljate? “)
 (bind ?odg (read))
 (while (and (neq ?odg da)(neq ?odg ne)) do
      (printout t “Nastavljate? “)
      (bind ?odg (read)))
 (if (eq ?odg da)
  then(assert (faza nastavak))
  else(assert (faza prekid))
 )
)
```

PRIMER 2:

```
CLIPS> (assert (faza nastaviti-ili-ne))
==> f-1     (faza nastaviti-ili-ne)
==> Activation 0      nastavi-ili-ne: f-1
<Fact-1>

CLIPS> (run)
<== f-1     (faza nastaviti-ili-ne)
Nastavljate? bla
Nastavljate? Bla-bla
Nastavljate? da
==> f-2     (faza nastavak)
CLIPS>
```

Loop-for-count

```
(loop-for-count <opseg> [do]
        <akcija_1>
            :
        <akcija_m>)
```

<opseg> se definiše kao:

<krajnja-vrednost> ili
(<promenljiva> <krajnja-vrednost>) ili
(<promenljiva> <polazna-vrednost> <krajnja-vrednost>)

PRIMER 3:

```
CLIPS>
(loop-for-count 2
     (printout t "Hello world" crlf))
Hello world
Hello world
FALSE
```

Funkcija vraća FALSE osim kada se njeno izvršenje prekine naredbom RETURN.

PRIMER 4:

```
CLIPS>
(loop-for-count (?i 2 4) do
   (loop-for-count (?j 3) do
   (printout t ?i " " )
   (loop-for-count 3 do
      (printout t “.”))
   (printout t “ “ ?j crlf)))

2 ... 1
2 ... 2
2 ... 3
3 ... 1
3 ... 2
3 ... 3
4 ... 1
4 ... 2
4 ... 3
FALSE
CLIPS>
```

Switch

```
(switch <test-izraz>
(case <uporedni-izraz> then <akcija>*)*
[(default <akcija>*)])
```

- Kod switch funkcije se prvo utvđuje vrednost test izraza, a potom se redom proverava da li neki od uporednih izraza ima vrednost jedanku vrednosti test izraza. U slučaju da ima, izvršavaju se akcija navedene kao posledica odgovarajućeg uporednog izraza. U slučaju da nijedan izraz nema traženu vrednost izvršavaju se default akcije, koje su opcione.

PRIMER 7:

```
CLIPS>
(defrule vrednost-karte
  (karta ?naziv-karte)
    =>
  (switch ?naziv-karte
    (case as then (bind ?vrednost 11))
    (case zandar then (bind ?vrednost 12))
    (case dama then (bind ?vrednost 13))
    (case kralj then (bind ?vrednost 14))
    (default (bind ?vrednost 0)))
  (printout t ?vrednost)))
CLIPS> (assert (karta dama))
CLIPS> (run)
13
```

Break

- **`break`** funkcija momentalno prekida *while* ili l*oop-for-count* petlju u kojoj je sadržana. Nema argumenata.

Halt

- **Halt** funkcija se upotrebljava na desnoj strani pravila kako bi se zaustavilo dalje okidanje pravila.
- Nema argumenata. Nema povratnih vrednosti.
- Nakon pozivanja **halt** funkcije agenda ostaje netaknuta, a izvršavanje programa se može nastaviti komandom **run.**

Deffunction konstrukcija

- **deffunction** konstrukcija omogućava korisniku da definiše nove funkcije u CLIPS okruženju korišćenjem CLIPS sintakse.

Sintaksa

```
(deffunction <naziv-funkcije> [komentar]
(<regularni-parametar>*
 [(<wildcard-parametar>])
 <akcija>*)
```

- Kada se pozove funkcija definisana upotrebom deffunction konstrukcije, njene akcije se izvršavaju u onom redosledu kojim su navedene
- Vrednost koju funkcija vraća je rezultat poslednje izvršene akcije.
- Ako funkcija nema ni jednu akciju vratiće simbol FALSE.
- Ako prilikom izvršavanja funkije dođe do greške, sve akcije koje nisu još uvek izvršene se stopiraju i funkcija vraća simbol FALSE.

PRIMER:

```
(deffunction hipotenuza
  (?a ?b)
  (sqrt(+(** ?a 2)(** ?b 2))))

(defrule rule1
  (katete ?x ?y)
  =>
  (bind ?c (hipotenuza ?x ?y))
  (printout t "Hipotenuza je " ?c crlf))
```

PRIMER:

```
CLIPS> (assert (katete 3 4))
==> f-0     (katete 3 4)
==> Activation 0      rule1: f-0
<Fact-0>

CLIPS> (run)
FIRE    1 rule1: f-0
Hipotenuza je 5.0
```

Return

- **return** funkcija momentalno prekida izvršavanje trenutno aktivne funkcije.

- Ako nema argumente ne vraća vrednost, u suprotnom vraća vrednost svog argumenta.

PRIMER 5:

```
CLIPS>
(deffunction sgn (?broj)
(if (> ?broj 0) then
(return 1))
(if (< ?broj 0) then
(return -1))
 0)

CLIPS> (sgn 5)
1
CLIPS> (sgn -10)
-1
CLIPS> (sgn 0)
0
```

Zadatak

- Napisati funkciju koja vraća TRUE ako je njen argument prost broj, a u suprotnom vraća FALSE.

Defglobal konstrukcija

- Konstrukcijom **`defglobal`** mogu se definisati globalne promenljive koje zadržavaju svoje vrednosti i van opsega konstrukcija.
- Sintaksa:

```
(defglobal [<naziv-modula>]
           <deklaracija-gl-promenljive>*)
```

`<deklaracija-gl-promenljive>` je

`?*<naziv-promenljive>*=<izraz>`

PRIMER:

```
CLIPS> (defglobal
           ?*x* = 3
           ?*y* = ?*x*
           ?*z* = (+ ?*x* ?*y*)
           ?*q* = (create$ a b c))

CLIPS> (show-defglobals)
?*x* = 3
?*y* = 3
?*z* = 6
?*q* = (a b c)
```

```
CLIPS> (ppdefglobal y)
(defglobal MAIN ?*y* = ?*x*)

CLIPS> (list-defglobals)
x
y
z
q
For a total of 4 defglobals.

CLIPS> (get-defglobal-list)
(x y z q)

CLIPS> (show-defglobals)
?*x* = 3
?*y* = 3
?*z* = 6
?*q* = (a b c)
```

```
CLIPS> (bind ?*y* 5)
5

CLIPS> (show-defglobals)
?*x* = 3
?*y* = 5
?*z* = 6
?*q* = (a b c)

CLIPS> (undefglobal y)

CLIPS> (list-defglobals)
x
z
q
For a total of 3 defglobals.

CLIPS>
```

```
CLIPS> (bind ?*x* 12)
12
CLIPS> ?*x*
12
CLIPS> (reset)

CLIPS> ?*x*
3
CLIPS> (watch globals)
CLIPS> (bind ?*x* 7)
:== ?*x* ==> 7 <== 3
7
CLIPS> (reset)
:== ?*x* ==> 3 <== 7

CLIPS>
```

- Globalne promenljive se ne mogu upotrebljavati kao parametri za deffunction.

- Globalne promenljive se mogu upotrebljavati na desnoj strani pravila, a na levoj strani pravila se ne mogu upotrebljavati na način na koji upotrebljavamo lokalne promenljive. Dakle, neispravno je:

```
(defrule example
  (fact ?*x*)
   =>)
```

Ispravno je upotrebiti globalne promenljive na levoj strani pravila u okviru poziva funkcije :

```
(defrule example
   (fact ?y&:(> ?y ?*x*))
    =>)
```

Ovo pravilo će se aktivirati ako postoji činjenica čije je prvo polje **`fact`**, a vrednost drugog polja je veća od vrednosti globalne promenljive x.

Definisane funkcije i globalne promenljive se mogu koristiti u više modula korišćenjem export i import naredbi:

- **(export defglobal ?ALL)**
- **(export defglobal ?NONE)**
- **(export defglobal <naziv-defglobal>+)**

- **(export deffunction ?ALL)**
- **(export deffunction ?NONE)**
- **(export deffunction <naziv-deffunction>+)**

- **(import <naziv-modula> defglobal ?ALL)**
- **(import <naziv-modula> defglobal ?NONE)**
- **(import <naziv-modula> defglobal <naziv-defglobal>+)**

- **(import <naziv-modula> deffunction ?ALL)**
- **(import <naziv-modula> deffunction ?NONE)**
- **(import <naziv-modula> deffunction <naziv-deffunction>+)**

Neke funkcije CLIPS-a

Multifield funkcije

- `(create$ <izraz>*)` kreira multifield vrednost spajanjem svih svojih argumenta.

- `(nth$ <integer-izraz> <multifield-izraz>)` vraća određeno polje iz multifield vrednosti.

```
CLIPS> (nth$ 3 (create$ a b c d e f g))
c
```

- **(member$ <izraz> <multifield-izraz>)** ako je prvi argument single field i sadrži se u drugom argumentu onda vraća integer koji određuje poziciju prvog argumenta u drugom. Ako je prvi argument multifield i sadrži se u drugom argumentu onda vraća multifield vrednost koja se sastoji od dva integera – početne i krajnje pozicije prvog argumenta u drugom. U suprotnom vraća FALSE.

```
CLIPS> (member$ belo (create$ 31 "text" belo))
3
CLIPS> (member$ 4 (create$ 3 8.7 zuto))
FALSE
```

- **(subsetp <multifield-izraz> <multifield-izraz>)**

vraća TRUE ako se ceo prvi argument sadrži u drugom argumentu, u suprotnom vraća FALSE.

```
CLIPS> (defrule p1
         (f1 $?x)
         (f2 $?y)
         (test (subsetp ?x ?y))
           =>)
CLIPS> (assert (fact1 e a c))
<Fact-0>
CLIPS> (assert (fact2 a b c d e f))
==> Activation 0      p1: f-0,f-1
<Fact-1>
```

- **(delete$ <multifield-izraz>**
  <begin-integer-izraz>
  <end-integer-izraz>)

briše sva polja prvog argumenta i to od pozicije označene drugim do pozicije označene trećim argumentom.

```
CLIPS> (delete$ (create$ a b c d e) 2 4)
(a e)
```

- **(explode$ <string-izraz>)** vraća multifield vrednost napravljenu od stringa datog u argumentu.

```
CLIPS> (explode$ "ekspertni sistem")
(ekspertni sistem)
```

- **(imlode$ <multifield-izraz>)** vraća string napravljen od multifield vrednosti date u argumentu.

```
CLIPS> (implode$ (create$ ekspertni sistem))
"ekspertni sistem"
```

- **(subseq$ <multifield-izraz>**
  <begin-integer-izraz>
  <end-integer-izraz>)

vraća multifield vrednost koja je formirana izvlačenjem polja prvog argumenta i to od pozicije označene drugim do pozicije označene trećim argumentom.

```
CLIPS> (deftemplate osoba
         (multislot ime))
CLIPS>
(defrule p
  (osoba (ime $?i))
   =>
  (printout t "Prezime je " (subseq$ ?i 2 2) crlf))
CLIPS> (assert (osoba (ime Pera Peric)))
CLIPS> (run)
Prezime je (Peric)
```

- **(replace$ <multifield-izraz>**
  <begin-integer-izraz>
  <end-integer-izraz>
  <single-ili-multifield-izraz>+)

  vraća multifield vrednost koja je formirana zamenjivanjem polja prvog argumenta i to od pozicije označene drugim do pozicije označene trećim argumentom poljima ostalih argumenata.

```
CLIPS> (replace$ (create$ a b c d) 2 3 x y q r s)
(a x y q r s d)
```

- **(insert$ <multifield-izraz>**
  <integer-izraz>
  <single-ili-multifield-izraz>+) vraća multifield vrednost koja je formirana ubacivanjem polja trećeg argumenta u prvi argument i to od pozicije označene drugim argumentom.

```
CLIPS> (insert$ (create$ a b c d) 4 y z)
(a b c y z d)
```

- `(first$ <multifield-izraz>)` vraća prvo polje multifield vrednosti argumenta.
- `(rest$ <multifield-izraz>)` vraća sva osim prvog polja multifield vrednosti argumenta.
- `(length$ <multifield-izraz>)` vraća integer koji označava broj polja multifield vrednosti argumenta.

- **(delete-member$ <multifield-izraz> <izraz>+)**

  iz prvog argumenta briše sva pojavljivanja vrednosti naznačenih ostalim argumentima.

```
CLIPS>
(delete-member$ (create$ a b a c b) a b)
(c)
```

- **(replace-member$ <multifield-izraz>**
  <izraz-kojim-se-menja>
  <izraz-koji-se-menja>+)

  vrednošću drugog argumenta zamenjuje u prvom argumentu sva pojavljivanja vrednosti naznačenih ostalim argumentima.

```
CLIPS>
(replace-member$ (create$ a b a c d b) x a b)
(x x x c d x)
```