Zdroj: http://lide.uhk.cz/fim/ucitel/fshusam2/lekarnicky/zt_old/lekce1.html

Náplň lekce:

- Fakta a Pravidla
- Proměnné
- Prostředí ClipsWin
- Jak je program Clipsem zpracováván ?
- První programy v Clipsu

Základní části programu - Fakta a Pravidla

Při psaní programů se budete zpočátku setkávat s dvěma základními konstrukcemi a to s fakty a pravidly.

Co tyto pojmy znamenají?

FAKTA

Dá se říci, že fakta jsou znalosti o reálných "objektech" reality (my je vlastně v programu identifikujeme (deklarujeme) a dáváme tím najevo, s čím bude náš program pracovat, čeho se bude týkat). Vlastně podle toho jaká fakta jsou v programu obsažena se dá docela dobře usoudit, k čemu je program určen. Daná fakta se Vám zobrazují v Clipsu v okně FACTS, přičemž během činnosti programu můžete různá fakta do báze přidávat nebo rušit. Tato fakta definujete v konstrukci deffacts.

Syntaxe:

```
(deffacts nazev_ struktury
  (relace prvni_objekt)
  (relace druhy_objekt)
...
  (relace n_ty_objekt)
)
```

Pod názvem relace si můžete představit název daného faktu (čeho se fakt bude týkat) např. ovoce, zvirata, filmy. Po názvu relace může následovat jeden či více objektů (tj.seznam). Seznam hodnot může být libovolně dlouhý, ale musí obsahovat min. jednu hodnotu a to název relace. K ní budeme moci později přidávat různé objekty.

Máme zde tři struktury deffacts. Je samozřejmě možné vytvořit jen jednu strukturu deffacts a do ní dát vše. Je ale lepší dát podobné informace k sobě - je to více přehlednější.

```
(deffacts seznam_akcnich_filmu
   (akce James_Bond)
   (akce Sberatel_kosti)
```

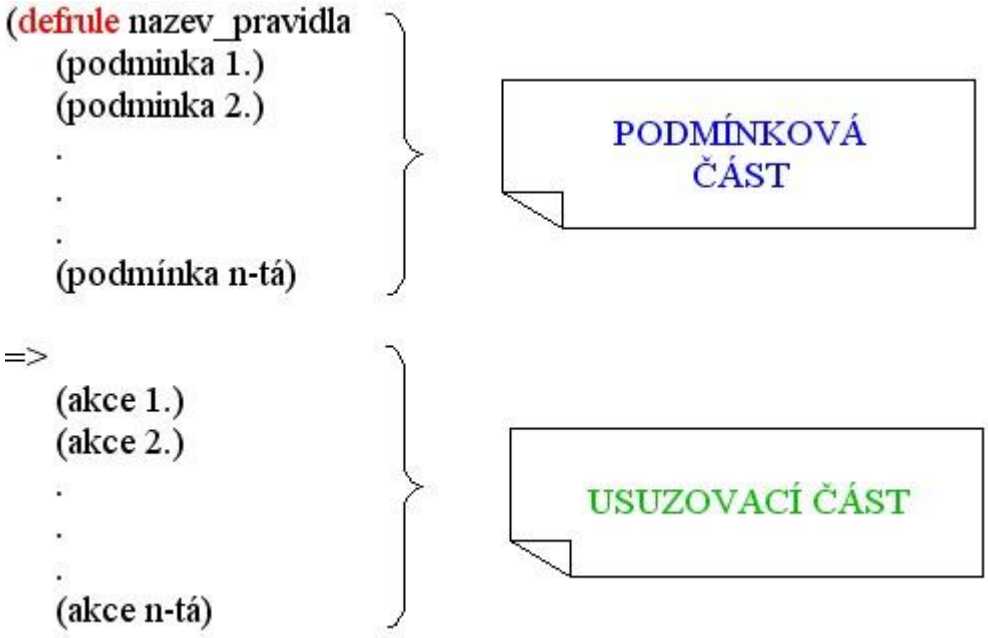
```
    (akce Tomb_Raider)
)

(deffacts seznam_sci-fi_filmu
  (sci-fi Alien)
  (sci-fi StarWars)
  (sci-fi Ja-robot)
)

(deffacts seznam_komedii
  (komedie Slecna_drsnak)
  (komedie Cetnik_a_cetnice)
)
```

atd.

PRAVIDLA

Pravidla jsou konstrukcemi, které již něco provádějí s Vašimi fakty. Provádějí nějakou akci. Například skutečnost, že můžete vypsat např. jen filmy z oblasti sci-fi.

Syntaxe:



Pravidlo je tvořeno dvěma částmi:

1. podmínková část - zde definujete za jakých podmínek má být provedena daná akce - tedy to co je uvedeno v usuzovací / akční části pravidla
2. usuzovací / akční část - to co bude provedeno - ta nějaká akce, jestliže budou splněny všechny podmínky uvedené v podmínkové části pravidla. Zde pozor na slovíčko všechny. Podmínek uvedených v podmínkové části pravidla samozřejmě může být více. Pak mezi nimi platí vztah AND (a zároveň - tzn., že musí být splněna podmínka 1. a zároveň 2. až n-tá) . Stačí, aby alespoň jedna podmínka splněna nebyla, pak se pravidlo vůbec neprovede.

Pravidla, která budou moci být v nejbližší době provedena se Vám zobrazují v okně AGENDA. Je to jakýsi aktuální seznam aktivovaných pravidel, které budou moci být v nejbližší době provedena.

Jinak si fakta a pravidla můžete představit i jinak. Vezměte si například způsob svého uvažování. Během svého života získáváte mnoho a mnoho informací. Ty si ukládáte do své paměti a dále s nimi určitým způsobem pracujete, využíváte je, manipulujete s nimi. Těmito informacemi jsou například informace o matematických plošných objektech např. o obdélníku. Víte jak obdélník vypadá, jaké má charakteristiky např. dané strany. Tyto informace jsou našimi fakty. Znáte i vzorec pro výpočet jeho obsahu. Jestliže tento vzoreček použijete a zjistíte obsah obdélníku, je to jako byste realizovali pravidlo - konali s informacemi, které jste získali, nějakou akci. Možná, že tento příklad není příliš trefný, ale snad svůj účel splnil.

Toto jsou základní konstrukce v Clipsu (existují i další konstrukce např. šablony či moduly, ale s nimi až později).

PROMĚNNÉ

Proměnnou můžeme charakterizovat jako místo, kam si na určitou dobu něco uschováte. Přičemž na toto místo můžete uložit cokoliv a měnit její obsah (něco jako spižírna).

Základní vlastnosti proměnných:

1. v Clipsu se proměnná označuje otazníkem s nějakým pojmenováním např. ?osoba - tato proměnná bude obsahovat třeba konkrétní osobu: Karel
2. proměnná, ale nemusí být nějak pojmenována a to v případě, když nás její obsah nezajímá - takto ?
3. v jiných programovacích jazycích jste asi byli zvyklí proměnnou deklarovat, zde nemusíte
4. lze ji použít jen u konstrukce typu pravidlo (nikoliv u struktury deffacts)
5. není možné, aby byly v usuzovací části pravidla uvedeny proměnné, který by neměly žádný obsah
6. proměnné uvedené v pravidle jsou proměnnými lokálními tzn., že existují jen v rámci tohoto pravidla a ne jiného => stejný název proměnné může být uveden ve více pravidlech

Prostředí ClipsWin

Spuštění prostředí ClipsWin:

1. Vyberte spustitelný soubor s názvem ClipsWin6-21.exe a spusťte ho.
2. Po spuštění se Vám objeví samotné prostředí Clipsu

Popis prostředí:

Toto prostředí je tvořeno jakýmsi souborem různých druhů oken, kde každé z nich má svůj význam.

1/ dialogové okno = příkazové okno: toto okno je jedno z nejdůležitějších. Zde se Vám budou objevovat výsledky vámi napsaného programu, chybová hlášení, již načtené konstrukce, které jste v programu vytvořili, objeví se Vám např. informace o tom, jestli je daný soubor, který jste do prostředí načetli příkazem Load Binary, načten správně apod.

2/ v hlavní nabídce v položce Window si můžete zobrazit další okna. Z nichž nejdůležitější jsou tato:

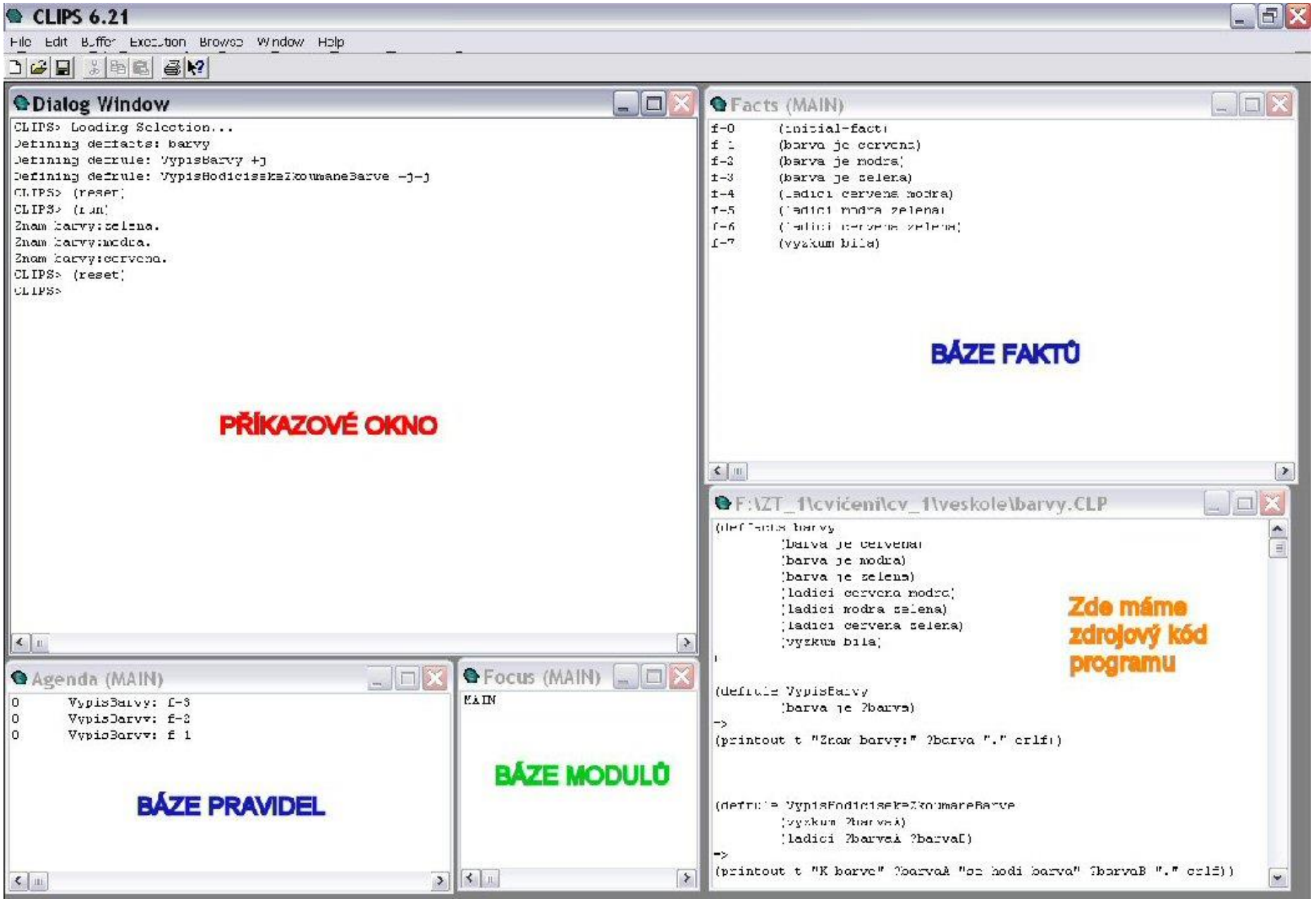
a) Facts
b) Agenda
c) Focus

ad a) Window Facts = v tomto okně budete mít zobrazeny všechna důležitá fakta (informace o "objektech reality", se kterými budete pracovat). To co bude v okně Facts obsaženo ztotožňujeme s BÁZÍ FAKTŮ.

ad b) Window Agenda = toto okno Vám bude zobrazovat pravidla, která jsou tzv. aktivována tj.která se mají v následujícím kroku provést. Může jich být v jednu chvíli i několik - pak mluvíme o tzv. Konfliktní množině pravidel, kdy inferenční mechanismus Clipsu má za úkol např. podle priority pravidla odpovídající pravidlo vybrat. To co bude v okně Agenda ztotožňujeme s BÁZÍ PRAVIDEL.

ad c) Window Focus = okno, které se v sekci Znalostní technologie I. nebude používat, protože spadá do oblasti modulárního vytváření programů. Ale jen tak předběžně se jedná o okno, ve kterém se Vám objeví ty moduly, na které bude v následujícím kroku inferenční mechanismus zaměřen.

Někdy se hodí i volba Window/Clear Dialog Window, kdy si vyčistíte obsah dialogového okna. Pomůže Vám to zpřehlednit jednotlivé prováděné akce programu, který jste spustili.Docela šikovné jsou i volby Window/Cascade-Tile Horizontally-Tile Vertically, které Vám pomohou uspořádat jednotlivá okna prostředí Clips.Pomocí Edit/Set Font si můžete nastavit typ písma a jeho barvu pro zdrojový kód Vašeho programu.


CLIPS 6.21
File Edit Buffer Execution Browse Window Help
Dialog Window
CLIPS> Loading Selection...
Defining deffacts: barvy
Defining defrule: VypisBarvy +j
Defining defrule: VypisHodiciseskeZkoumaneBarve -j-j
CLIPS> (reset)
CLIPS> (run)
Znam barvy:zelena.
Znam barvy:modra.
Znam barvy:cervena.
CLIPS> (reset)
CLIPS>
PŘÍKAZOVÉ OKNO
Facts (MAIN)
f-0 (initial-fact)
f-1 (barva je cervena)
f-2 (barva je modra)
f-3 (barva je zelena)
f-4 (ladici cervena modra)
f-5 (ladici modra zelena)
f-6 (ladici cervena zelena)
f-7 (vyzkum bila)
BÁZE FAKTŮ
F:\ZT_1\cvičení\cv_1\veskole\barvy.CLP
(deffacts barvy
(barva je cervena)
(barva je modra)
(barva je zelena)
(ladici cervena modra)
(ladici modra zelena)
(ladici cervena zelena)
(vyzkum bila)
)
(defrule VypisBarvy
(barva je ?barva)
=>
(printout t "Znam barvy:" ?barva "." crlf))
(defrule VypisHodiciseskeZkoumaneBarve
(vyzkum ?barvaA)
(ladici ?barvaA ?barvaB)
=>
(printout t "K barve" ?barvaA "se hodi barva" ?barvaB "." crlf))
Zde máme zdrojový kód programu
Agenda (MAIN)
0 VypisBarvy: f-3
0 VypisBarvy: f-2
0 VypisBarvy: f-1
BÁZE PRAVIDEL
Focus (MAIN)
MAIN
BÁZE MODULŮ

Otevření a spuštění programu

Předtím než začneme psát programy v Clipsu, tak se podíváme na to, jak program otevřít a spustit. Vytvořte si soubor ovoce_1.clp.

```
;**************************
;
;           POTRAVINY
;
;          (MH 2005)
;**************************

;FAKTA - znalosti, se kterými manipuluji
;pomocí pravidel

(deffacts potraviny
  (ovoce Jablko)
  (ovoce Pomeranc)
  (ovoce Hruska)
  (ovoce Mango)
  (zelenina Mrkvicka)
  (zelenina Celer)
  (zelenina Paprika) )


;PRAVIDLO PRVNÍ - konktrétní akce, díky
;které manipuluji s fakty
;vypisuji ovoce

(defrule vypis_ovoce
  (ovoce ?nejake)
=>
  (printout t "Znam:" " " ?nejake crlf) )


;PRAVIDLO DRUHÉ
;vypisuji zeleninu

(defrule vypis_zeleninu
  (zelenina ?nejaka)
=>
  (printout t "Znam jeste:" " " ?nejaka
crlf) )
```

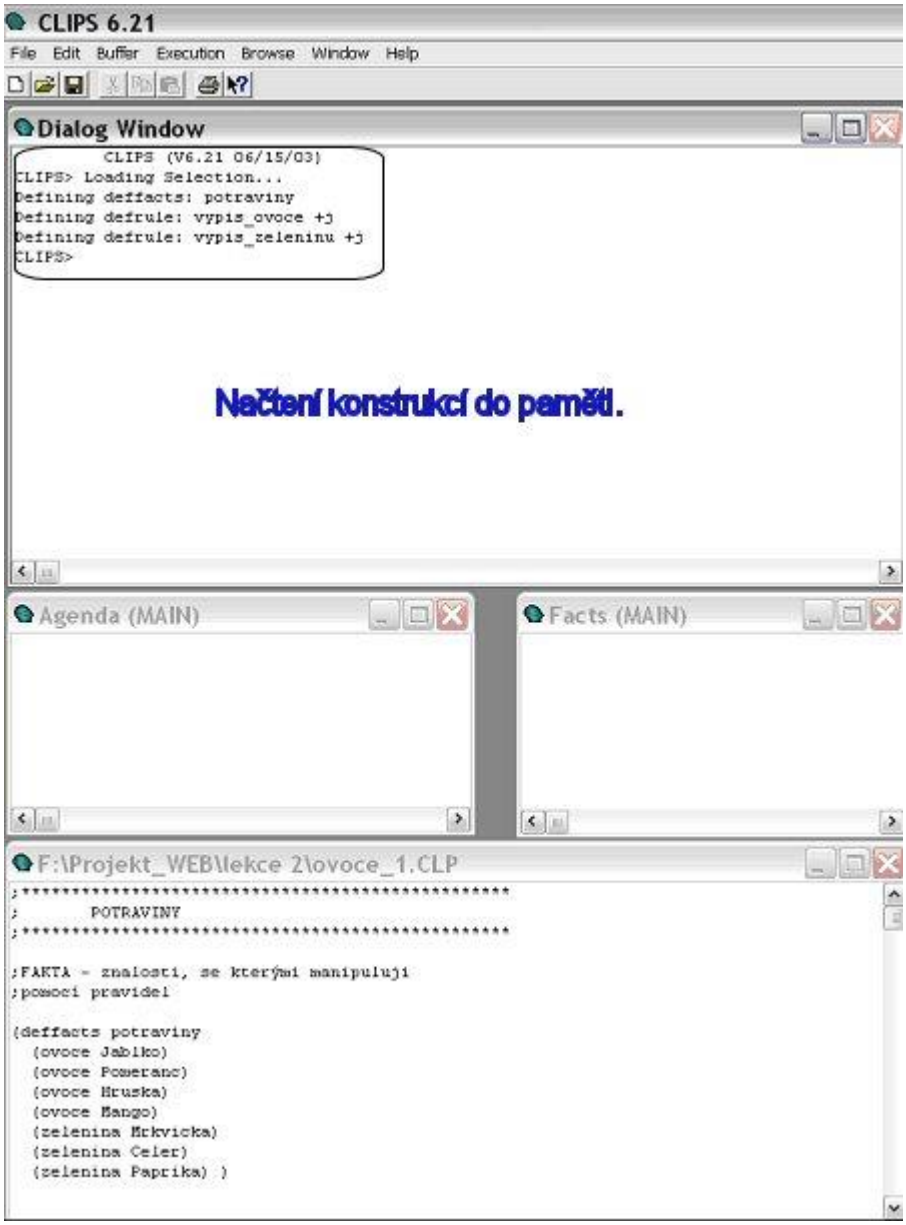
Otevření programu

File/Open/ovoce_1.clp = otevře se Vám nové okno, kde je zobrazen zdrojový kód programu ovoce_1.

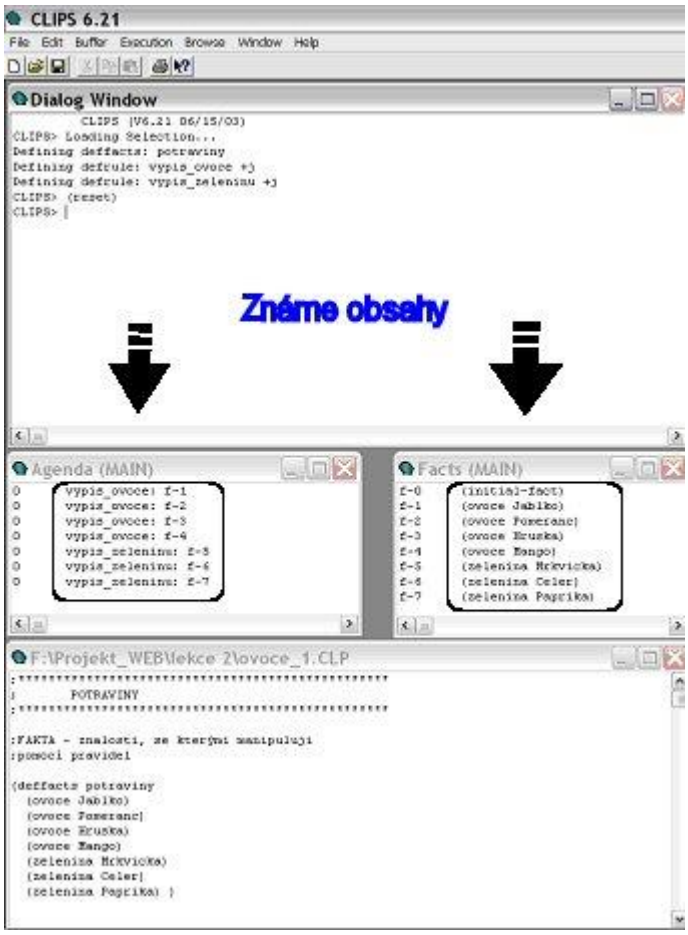
Spuštění programu

Pro jeho spuštění musíte udělat několik kroků - nejde to jedním povelem.

- 1/ do okna, kde máte zdrojový kód programu umístěte kurzor myši (tím jakoby dáte Clipsu najevo s čím chcete pracovat)
- 2/ Buffer/Load Buffer (už z názvu Load Buffer je zřejmé, že touto akcí dojde k nahrání programových konstrukcí do paměti počítače a to jen v případě, že ve Vašem programu nejsou žádné syntaktické chyby). Tedy Clips bude vědět, že se bude pracovat s nějakými konstrukcemi - fakty a pravidly, ale ještě nebude znát jejich přesný obsah - můžete vidět, že se zatím nic neobjevilo v okně Facts ani v okně Agenda.



- 3/ Execution/Reset : jedná se o inicializační činnost programu (zde už Clips ví, co je obsahem jednotlivých konstrukcí - můžete vidět, že okna: Facts (naše báze faktů) a Agenda (báze pravidel) již nejsou prázdná



- 4/ Execution/Run: spuštění programu (resp. spuštění inferenčního mechanismu)

Důležitá poznámka:

Jestliže jste program spustili, on se vykonal a činnost programu byla pak ukončena (zobrazí se: CLIPS>), pak jestliže zdrojový kód programu upravíte např. přidáte nějaké to pravidlo či ho zrušíte, pak se ty úpravy musejí promítnout také do paměti počítače. Aby se tak stalo pak …
Musíte učinit tyto kroky:
a/ klikněte do plochy dialogového okna, aby Clips věděl co jste aktivovali. Pak zadejte z hlavní nabídky Execution/Clear Clips … jednoduše řečeno tím vymažete program z paměti
b/ opakujete kroky 1 - 4 (viz. výše)

Můžete si zkusit nejprve spustit program tak jak je ode mě napsaný. Pak vymazat jedno pravidlo např. vypis_zeleninu a dát jen Execution/Reset a Execution/Run. Uvidíte, že se Vám vypíše jak ovoce tak i zelenina, i když jste pravidlo pro výpis zeleniny smazali. Je to právě tím, že jste neobnovili změny ve Vašem programu.

Jak je program Clipsem zpracováván

Píšeme první program v Clipsu

Veškerou činnost programu, který napíšete řídí tzv. inferenční mechanismus. Podle stavu báze faktů (= báze dat, báze znalostí, to co je uvedené v konstrukci deffacts) provádí příslušná pravidla (to co je v konstrukci defrule).

Provádí tři důležité činnosti:

1. inferenční mechanismus vytvoří jakousi množinu aktivovaných pravidel - to je taková množina pravidel, která budou moci být v nejbližším okamžiku provedena, tedy ta pravidla, která mají splněnu podmínkovou část (jestliže takové pravidlo existuje, může být provedeno to co je definováno v usuzovací - akční části pravidla). Co znamená, že mají splněnu podmínkovou část? Podmínková část pravidla, kterou vytvoříte, se totiž porovnává s bází faktů (tedy s tím co vytvoříte v konstrukci deffacts). A jestliže tato báze faktů obsahuje taková data - fakta, která jsou postačující pro to, aby bylo dané pravidlo vykonáno, tak pak bude moci být pravidlo vykonáno. Takové pravidlo se Vám objeví v okně Agenda. Agenda je tvořena množinou těch aktivovaných pravidel.
2. po vytvoření této množiny aktivovaných pravidel, inferenční mechanismus vybere jedno pravidlo. To které bude vybráno závisí na dané řídící strategii. Tato strategie se volí v Clipsu - Execution/Options/Strategy.
3. nakonec inferenční mechanismus vybrané pravidlo vykoná. V agendě můžete pozorovat, že dané pravidlo z ní zmizí. To co bude pravidlem provedeno závisí na tom, co jste v usuzovací-akční části pravidla napsali. Například můžete přidat nějaký fakt do báze faktů nebo nějaký fakt zrušit.

Tedy inferenční mechanismus je to algoritmus, který zajišťuje vykonávání pravidel na základě stavu báze faktů. Určuje jak a v jakém pořadí budou tato pravidla aplikována. Můžeme postupovat ve směru od počátečního stavu k cílovému. Pak mluvíme o strategii řízené daty (data driven strategy nebo jinak dopředné/přímé řetězení) nebo od cílového stavu směrem k počátečnímu. Pak se jedná o strategii řízenou cílem (goal driven strategy nebo jinak zpětné řetězení).

Píšeme první program v Clipsu

Půjdeme krok po kroku. Od nejjednodušších záležitostí k těm těžším (ale nebojte, ne zas tak moc). Program budeme postupně rozšiřovat.

Úloha č.1:

Zadání:

1. Evidujte různé krevní skupiny v bázi faktů
2. Všechny krevní skupiny vypište
3. Evidujte různé osoby a jejich krevní skupiny
4. Vypište všechny osoby jejichž krev známe
5. Vypište jen ty osoby, které mají krev B
6. Vypište jen ty osoby, které mají krev AB, v případě, že taková krev není evidována, tento případ ošetřete.
7. Evidujte jaké krevní skupiny jsou navzájem kompatibilní (stačí 2 - 3)
8. Zjistěte jaká krevní skupina je ke zkoumané skupině kompatibilní.

Řešení:

```
;*********************************************
;
;                 Krevni skupiny - zaklad
;
;                 (MH 2004)
;*********************************************


(deffacts krevni_skupiny
  (skupina A)
  (skupina B)
  (skupina 0)
  (skupina AB)

)

(defrule vypis_vsechny_skupiny
  (skupina ?nejaka)
=>
  (printout t "Znam skupinu:" " " ?nejaka crlf) )
```

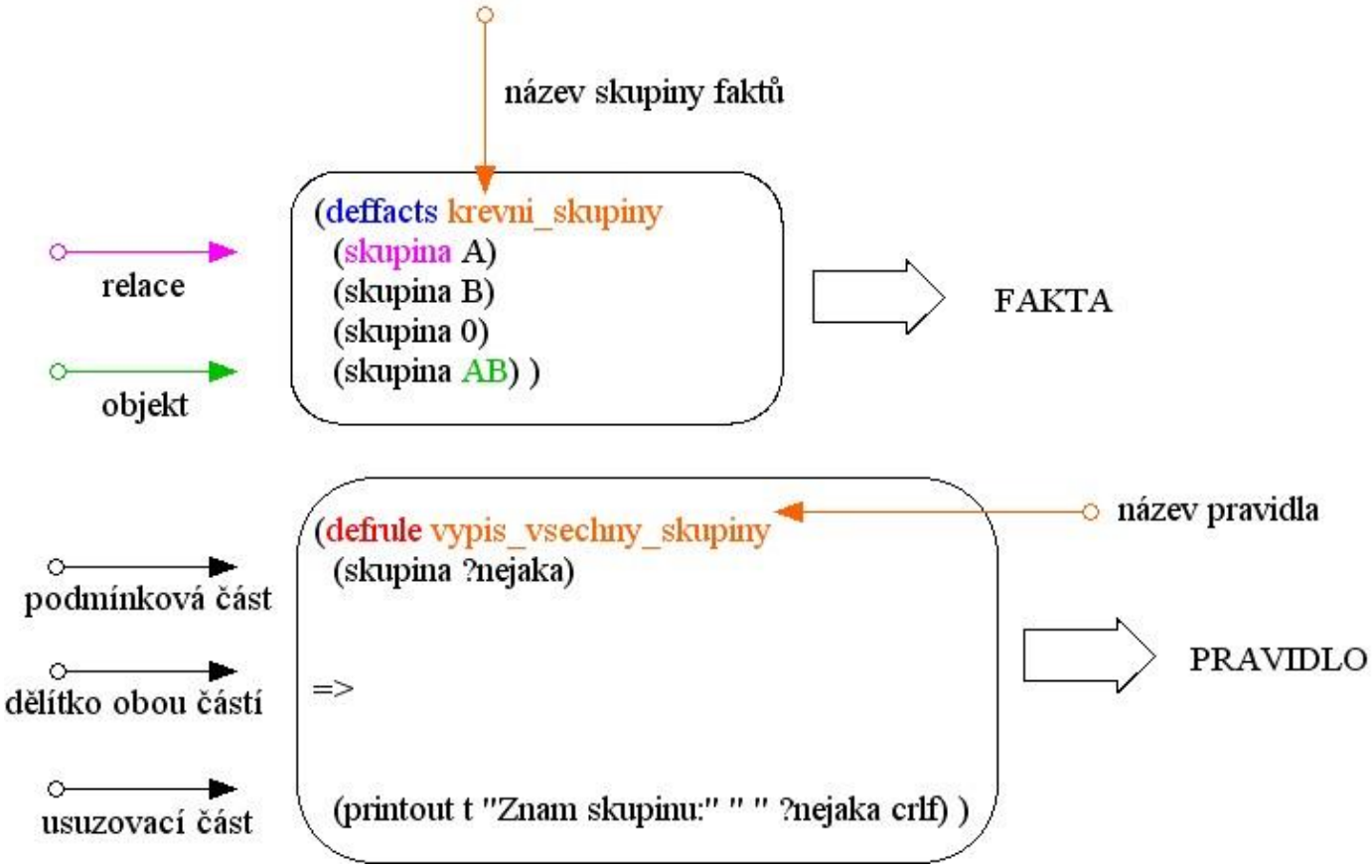
```
;*********************************************
;
;         Krevni skupiny - postupujeme dale
;
;                 (MH 2004)
;*********************************************
;Zadani:
;1/ Evidujte ruzne krevni skupiny v bazi faktu
;2/ Vsechny krevni skupiny vypiste
;3/ Evidujte ruzne osoby a jejich krevni skupiny
;4/ Vypiste vsechny osoby jejichz krev zname
;5/ Vypiste jen ty osoby, ktere maji krev B
;6/ Vypiste jen ty osoby, ktere maji krev AB, v pripade, ze takova krev neni
evidovana, tento pripad osetrete.
;7/ Evidujte jake krevni skupiny jsou navzajem kompatibilni (staci 2 - 3)
;8/ Zjistete, jaka krevni skupina je ke zkoumane skupine kompatibilni.

(deffacts krevni_skupiny
  (skupina A)
  (skupina B)
  (skupina 0)
  (skupina AB)

)

(deffacts osoby_a_krev
  (osoba Jan 0)
  (osoba Eva B)
  (osoba Karel B)
  (osoba Alzbeta 0) )


(deffacts kompatibilni_krevni_skupiny
  (kompatibilita 0 A)
  (kompatibilita B AB)
  (kompatibilita A AB) )

(deffacts zkoumana__krevni_skupina
  (zkoumame A) )
```

```
;uloha 1+2

(defrule vypis_vsechny_skupiny
  (skupina ?nejaka)
=>
  (printout t "Znam skupinu:" " " ?nejaka crlf) )


;uloha 3+4

(defrule vypis_zname_osoby
  (osoba ?osobax ?krev)
=>
  (printout t "Osoba:" ?osobax " " "ma krev:" ?krev crlf) )

;uloha 5

(defrule jen_osoba_s_B
  (osoba ?nejakaosoba B)
=>
  (printout t "Krevni skupinu B ma:" ?nejakaosoba crlf) )

;uloha 6

(defrule jen_osoba_s_AB
  (osoba ?nejakaosubka AB)
=>
  (printout t "Krevni skupinu AB ma:" ?nejakaosubka crlf) )


;uloha 6 osetreni, pro pripad, ze zadna osoba s AB neni evidovana

(defrule osoba_s_AB_neni_evidovana
  (not(osoba ?nejaka AB))
=>
  (printout t "Zadna osoba s krvi AB neni v bazi definovana !!!" crlf) )


;uloha 8

(defrule vypis_kompatibilni_skupiny_1moznost ;1. pozice
  (zkoumame ?neco)
  (kompatibilita ?neco ?y)
=>
  (printout t "Osoba s krevni skupinou:"?neco " " "muze darovat krev osobe s
krevni skupinou:" ?y crlf) )


(defrule vypis_kompatibilni_skupiny_2moznost ;2.pozice
  (zkoumame ?neco)
  (kompatibilita ?x ?neco)
=>
  (printout t "Osobe s krevni skupinou:"?neco " " "muze byt darovana krev:" ?x
crlf) )
```

Vysvětlení:

1/ Evidujte různé krevní skupiny v bázi faktů (soubor: krev_1.clp)

2/ Všechny krevní skupiny vypište (soubor: krev_1.clp)



Pro to, abychom vypsali všechny krevní skupiny stačí jedno pravidlo. Jak vidíte, je zde jen jedna podmínka: (skupina ?nejaka) … tzn. jako byste se ptali: za podmínky, že v bázi faktů je nějaká krevní skupina (?nejaka) evidována, pak tyto všechny skupiny (?nejaka) vypiš. Pro vypsání se používá funkce printout t. Pro odřádkování - crlf.

Jak to tedy funguje?

Výzkum podmínkové části pravidla

```
(skupina ?nejaka)
```

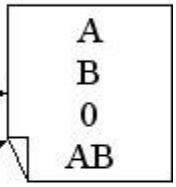
Ve struktuře deffacts máte uvedeny konkrétní skupiny takto:

```
(skupina A)
(skupina B)
(skupina 0)
(skupina AB).
```

Aby inferenční mechanismus věděl s jakými fakty chcete pracovat, pak v podmínkové části pravidla musíte uvést přesné znění dané relace tzn. v našem příkladu skupina (pozor na velká a malá písmena). Pak se inf. mech. podívá právě do struktury deffacts, kde je název dané relace s názvem skupina. Ale protože chcete vypsat konkrétní krevní skupiny, musíte využít v podmínkové části pravidla k relaci

proměnnou, která bude zastupovat konkrétní krevní skupinu (A, B, 0, AB). Z toho vyplývá, že podmínková část pravidla je splněna pro 4 fakty (pro skupinu A, B, 0 a AB).

```
(deffacts krevni_skupiny
  (skupina A)
  (skupina B)
  (skupina 0)
  (skupina AB)

)

(defrule vypis_vsechny_skupiny
(skupina ?nejaka)
=>
  (printout t "Znam skupinu:" " " ?nejaka crlf) )
```




```
Výsledek:

Znam skupinu: A
Znam skupinu: B
Znam skupinu: 0
Znam skupinu: AB
```

Výzkum usuzovací části pravidla

Tedy pravidlo s názvem vypis_vsechny_skupiny se provede 4-krát. Nesmíte zapomenout uvést co chcete vypsat - opět uvedete daný název proměnné, který musí být stejný jako ten název proměnné uvedený v podmínkové části pravidla, aby inf. mech. věděl, co chcete vypisovat. Velmi jednoduše můžete vyzkoumat, jak vše toto funguje, pomocí krokování. Funkcí Reset způsobíte to, že v Agendě uvidíte danou množinu aktivovaných pravidel (tedy těch, které mohou být vykonány). Pak nedáte Run, ale Step (Execution/Step). Tím způsobíte, že se Vám vykoná jen jedno pravidlo z celkových čtyř a v Agendě tím pádem pravidlo z Agendy zmizí. To opakujete do té doby, dokud v Agendě nezbude žádné pravidlo, které by se mohlo vykonat.

Píšeme první program v Clipsu

Obecný popis problému (velmi zjednodušeně):

Inferenční mechanismus si uvědomí, která všechna fakta vyhovují dané podmínce (tedy co vše lze dosadit za proměnnou ?nejaka).

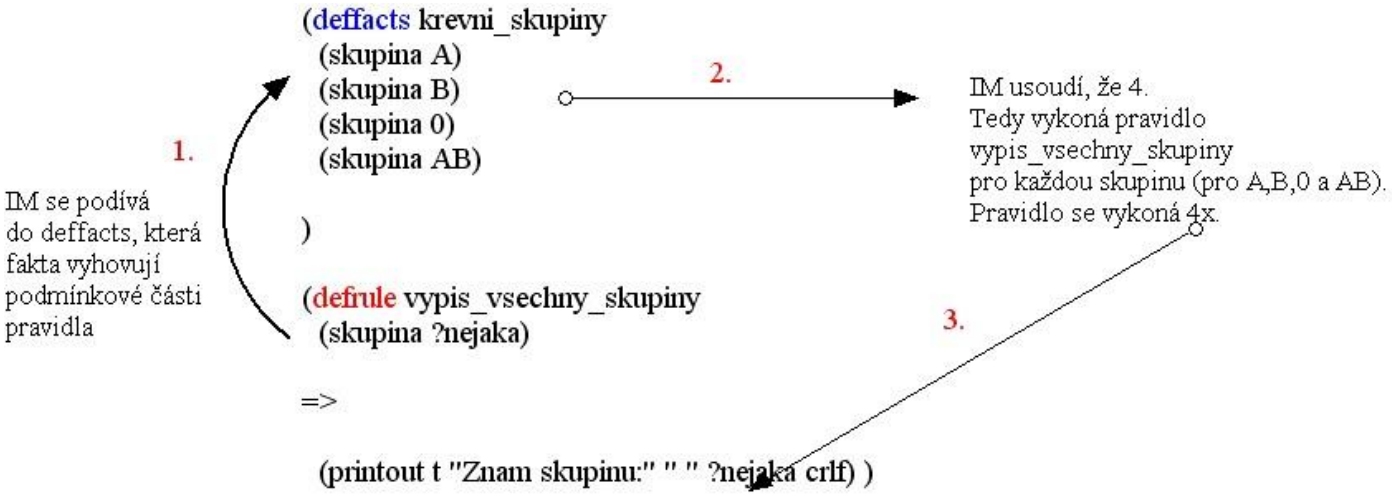
```
(defrule vypis_vsechny_skupiny
   (skupina ?nejaka)
```

Můžete si představit, že si inf. mech. vytvoří jakýsi seznam tvrzení, které podmínkové části pravidla vyhovují.

Imaginární seznam: (tedy podmínkové části pravidla vyhovují tato fakta)

```
   (skupina A)
   (skupina B)
   (skupina 0)
   (skupina AB).
```

Na základě toho si inf. mech. vytvoří množinu aktivovaných pravidel, těch jejichž podmínková část je splněna (to platí pro naše 4 fakty). Tato aktivovaná pravidla přidá do okna Agenda. Pak vybere první fakt z našeho imaginárního seznamu (např. skupina A) a pro něj vykoná usuzovací část pravidla. Dané pravidlo z Agendy zmizí. Dále vezme druhý fakt z imaginárního seznamu (skupina B) a i pro něj vykoná usuzovací část pravidla. A tak to jde do té doby, dokud se Agenda nevyprázdní - tedy již nebude moci být vykonáno žádné pravidlo. Záleží ovšem na strategii inf. mech., v jakém pořadí budou pravidla vykonávána. V našem případě, je zde zatím jen jedno pravidlo.



3/ Evidujte různé osoby a jejich krevní skupiny

Je nutné přidat do naší báze faktů informace o osobách a jejich krevních skupinách.

```
(deffacts osoby_a_krev
  (osoba Jan 0)
  (osoba Eva B)
  (osoba Karel B)
  (osoba Alzbeta 0)
)
```

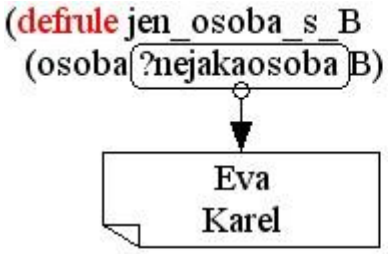
4/ Vypište všechny osoby jejichž krev známe

```
(defrule vypis_zname_osoby
  (osoba ?osobax ?krev)
=>   (printout t "Osoba:" ?osobax " " "ma krev:" ?krev crlf)
)
```

Co pravidlo znamená ?

Za podmínky, že je evidována v bázi faktů nějaká osoba (?osobax) s určitou krví (?krev) pak tuto osobu (viz. usuzovací část pravidla) (?osobax) s její krví (?krev) vypiš. (POZOR na stejné názvy proměnných a stejný název relace osoba - tak jak je to uvedeno v bázi faktů - ve struktuře deffacts).

5/ Vypište jen ty osoby, které mají krev B




```
=>
  (printout t "Krevni skupinu B ma:" ?nejakaosoba crlf) )
```

Za podmínky, že existuje nějaká taková osoba (?nejakaosoba) s krví B, "pak" ji vypiš.

Můžete si představit, že slovíčko "pak" je vlastně oddělovač obou částí pravidla =>. Všimněte si, že v programu v jednom pravidle užíváme název proměnné pro osobu: ?osobax a v jiném pravidle ?nejakaosoba. To je možné. Můžete používat různé názvy proměnných v různých pravidlech, ale v jednom pravidle musí být název proměnné v podmínkové části pro tu danou konkrétní proměnnou stejný jako v usuzovací části.

6/ Vypište jen ty osoby, které mají krev AB, v případě, že taková krev není evidována, tento případ ošetřete.

```
(defrule jen_osoba_s_AB
  (osoba ?nejakaosubka AB)
=>
  (printout t "Krevni skupinu AB ma:" ?nejakaosubka crlf)
)
```

Řešení je skoro stejné jako v úloze 5.

Ošetření:

```
(defrule osoba_s_AB_neni_evidovana
  (not(osoba ?nejaka AB))
=>
  (printout t "Zadna osoba s krvi AB neni v bazi definovana !!!" crlf)
)
```

V případě, že byste situaci neošetřili. Nic by se nestalo, ale je lepší dát uživateli vědět, že nic nebylo nalezeno. Představte, že byste měli jakési menu a z něj volili výpis všech osob s krevní skupinou AB, ale uživateli by se opět zobrazilo menu aniž by něco zjistil.

Využíváme zde podmínku s not. Jednoduše tím říkáme: za podmínky, že v bázi faktů neexistuje (to je to not) nějaká osoba s krevní skupinou AB, pak dejte uživateli vědět, že žádná taková osoba není v bázi definována. Not vlastně obalí celý výraz (osoba ?nejaka AB).

Píšeme první program v Clipsu

7/ Evidujte jaké krevní skupiny jsou navzájem kompatibilní

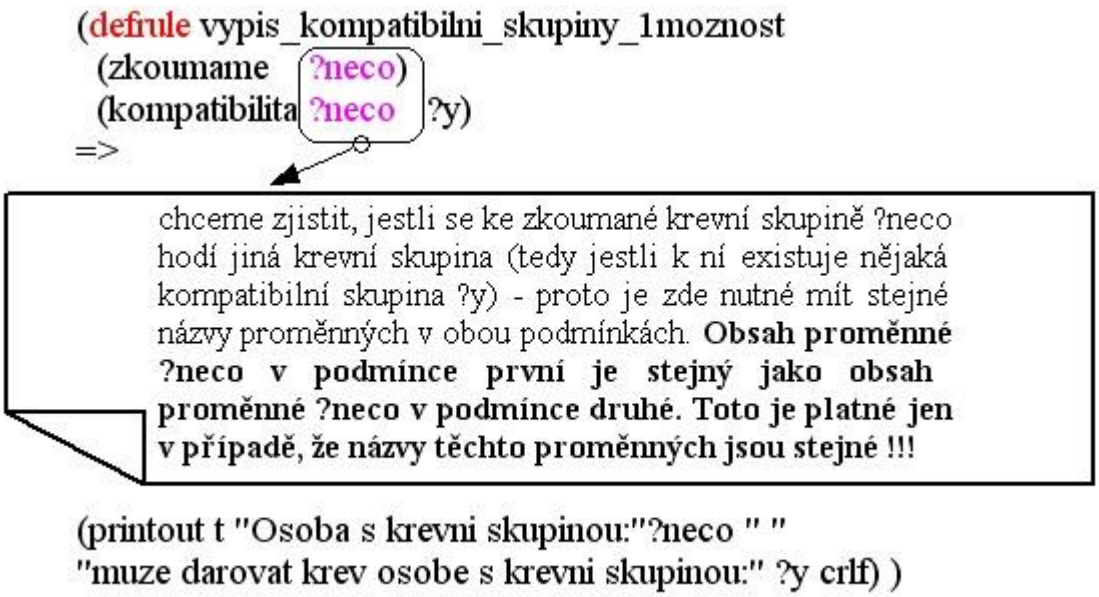
Kompatibilita se opět týká nových faktů, o kterých má náš program vědět. Tedy je vhodné vytvořit novou strukturu deffacts zachycující kompatibilní krevní skupiny. Kompatibilita znamená např. v případě (kompatibilita 0 A), že osoba s krví 0 může darovat osobě s krví A.

```
(deffacts kompatibilni_krevni_skupiny

  (kompatibilita 0 A)

  (kompatibilita B AB)

  (kompatibilita A AB)

)
```

8/ Zjistěte, jaká krevní skupina je ke zkoumané skupině kompatibilní. (máme před sebou určitou krevní skupinu a zjišťujeme, která je k ní kompatibilní)

```
(defrule vypis_kompatibilni_skupiny_1moznost

  (zkoumame ?neco)

  (kompatibilita ?neco ?y)

=>

  (printout t "Osoba s krevni skupinou:"?neco " " "muze darovat krev osobe s krevni skupinou:" ?y crlf)

)
```

Vysvětlení pravidla s kompatibilitou



Za podmínky, že tedy máme před sebou určitou zkoumanou krevní skupinu ?neco viz. podmínka (zkoumame ?neco) a zároveň osoba, která má tuto zkoumanou skupinu ?neco, může darovat osobě se skupinou ?y (viz. podmínka (kompatibilita ?neco ?y)). Všimněte si názvů proměnných v obou podmínkách.

Druhé pravidlo, je pro případ, že daná krevní skupina se nachází ve faktu pro kompatibilitu na druhé pozici (vedle ?x)

```
(defrule vypis_kompatibilni_skupiny_2moznost
  (zkoumame ?neco)
  (kompatibilita ?x ?neco)
=>
  (printout t "Osobe s krevni skupinou:"?neco " " "muze byt darovana krev:" ?x crlf)
)
```

LEKCE 3.

Náplň lekce:

- Práce se seznamy
- Příkazy assert, read, bind
- Prefixový zápis
- Další praktické příklady

Práce se seznamy

Seznam je určitá datová struktura, do které můžete uložit více hodnot najednou. Je souborem určitých hodnot-položek, který je opatřený jménem. Jednotlivé položky seznamu se oddělují jen mezerou. Seznam je struktura jinak nazývána souborem nestrukturovaných faktů s více položkami. Struktury se strukturovanými fakty nazýváme šablony, ale to zatím probírat nebudeme.

Podoba seznamu v Clipsu:

```
(název_seznamu položka_1 položka_2 . . . položka_n)
```

příklad:

```
(ovoce Jablko Hruska Bananek Pomeranc Kiwi)
```

Druhy seznamů:

1. prázdný: tedy i takovýto seznam je seznamem -> (zelenina)
2. jednopoložkový: (zelenina mrkvicka)
3. vícepoložkový: (zelenina celer mrkvicka pazitka)

Využití seznamů:

1. tam, kde nám nezáleží na pořadí položek definovaných v seznamu (tak třeba již zmíněnou zeleninu)
2. tam, kde položky seznamu mají zvláštní význam (adresa Jan Novak Manesova 320 Pardubice)
3. tam, kde chceme zachytit určitou relaci např. (rodic Jan Lucie) (zn.Jan je otcem dcery Lucie)

Příkazy assert, read, bind

Příkaz ASSERT (vlož)

Tento příkaz slouží k vkládání faktů do naší báze faktů.

Syntaxe: (assert (muj_fakt_ktery_vkladam_do_baze))

příklad: (assert (zelenina rajce))

Příkaz READ (čti)

Tento příkaz nám načte údaj/údaje, které zadá uživatel na klávesnici počítače.
(read) - tento příkaz má návratovou hodnotu získanou od uživatele.
Můžete si představit, že obsahem závorek je určitá konkrétní hodnota (rajce)

Příkaz BIND (svaž)

Tento příkaz sváže určitou hodnotu (např. hodnotu získanou od uživatele pomocí příkazu Read) s konkrétní proměnnou. Přiřadí tuto hodnotu konkrétní proměnné, abychom mohli s informací získanou od uživatele dále pracovat (aby hodnota od uživatele měla své místo/své pojmenování v programu - byla někam přiřazena). Může sloužit např. k uložení mezivýsledků při matematických výpočtech do proměnných.

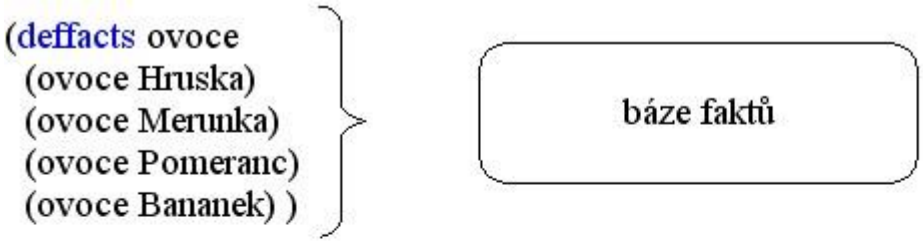
Syntaxe: (bind ?komu ?co)

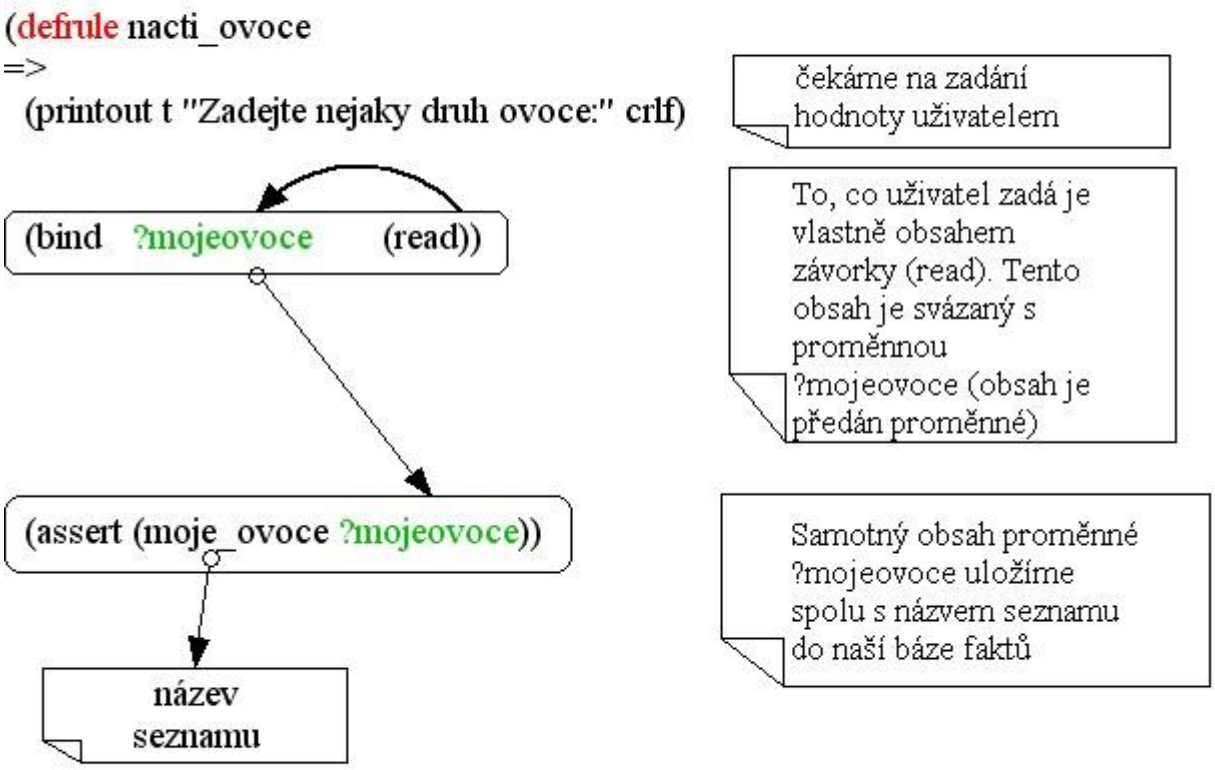
POZOR: Všechny tyto příkazy mohou být použity jen v usuzovací části pravidla.

Praktický příklad použití těchto funkcí:

```
;*********************************************
;
;      Prikazy pro ovoce (bind + assert + read)
;
;                 (MH 2005)
;*********************************************


(deffacts ovoce
  (ovoce Hruska)
  (ovoce Merunka)
  (ovoce Pomeranc)
  (ovoce Bananek) )


(defrule nacti_ovoce
=>
  (printout t "Zadejte nejaky druh ovoce:" crlf)
  (bind ?mojeovoce (read))
  (assert (moje_ovoce ?mojeovoce))
  (printout t "Udaj byl ulozen do baze faktu !!!" crlf) )
```

Vysvětlení obrázkem:





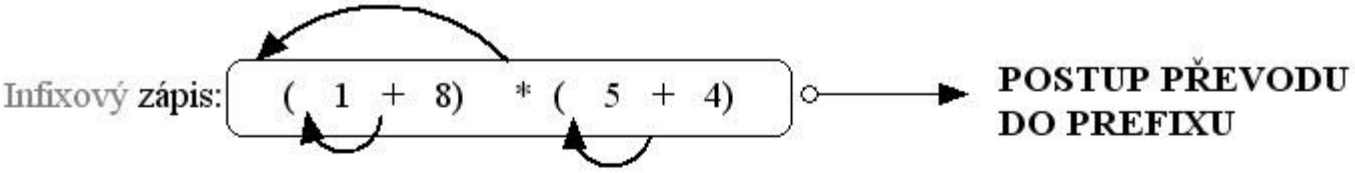
Při spuštění programu je uživatel vyzván, aby zadal nějaký druh ovoce. Po jeho zadání je program ukončen. Po ukončení programu se podívejte do okna Facts, kde Vám přibyl Váš nový fakt s názvem: (moje_ovoce xy).

Program si můžete pozměnit a to tak, že u příkazu assert neuvedete název seznamu moje_ovoce, ale přímo ovoce (tak jak to je u ostatních faktů ve struktuře deffacts). Pak spusťte program a jako svůj fakt zadejte takový, který již v bázi faktů existuje např. Pomeranc. Uvidíte, že do báze faktů žádný nový fakt nepřibude, ale vypíše se jen informace o vložení tohoto faktu do báze. Otázka, která Vás asi napadá je: "Proč se do báze nevložil žádný fakt?". Odpověď je jednoduchá. Clips Vám nedovolí do báze vložit fakt, který již v bázi existuje. Tedy (ovoce Pomeranc) vloží jen 1x, ale (moje_ovoce Pomeranc) je už něco jiného.

Poznámka: Někdy může být celá podmínková část vynechána. Bude provedena jen usuzovací část hned po příkazu Reset.

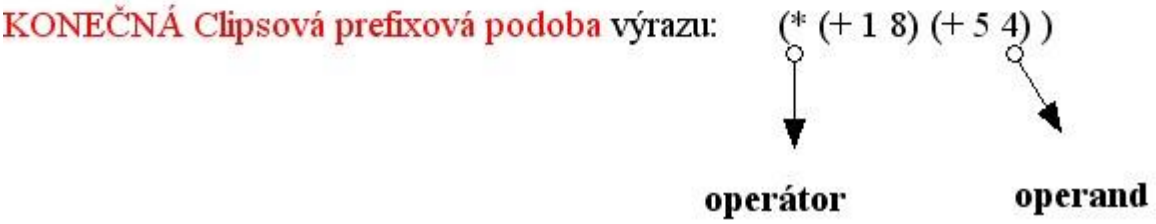
Prefixový zápis

Je takový zápis při kterém se operátory (např. + - * /) vkládají před operandy (různé hodnoty). Klasický matematický zápis je infixový, při kterém operátory vkládáme mezi operandy. Vysvětlení přefixu je na obrázku níže.



Prefixový zápis: * (+ 1 8) (+ 5 4)

V Clipsu je třeba uzavřít takovýto výraz ještě do závorek.



Složitější výraz:

Infixový zápis : ((a + b) or (2 - c < 1))

Prefixový zápis: (or (+ a b) (< (- 2 c) 1))

DALŠÍ PRAKTICKÉ PŘÍKLADY

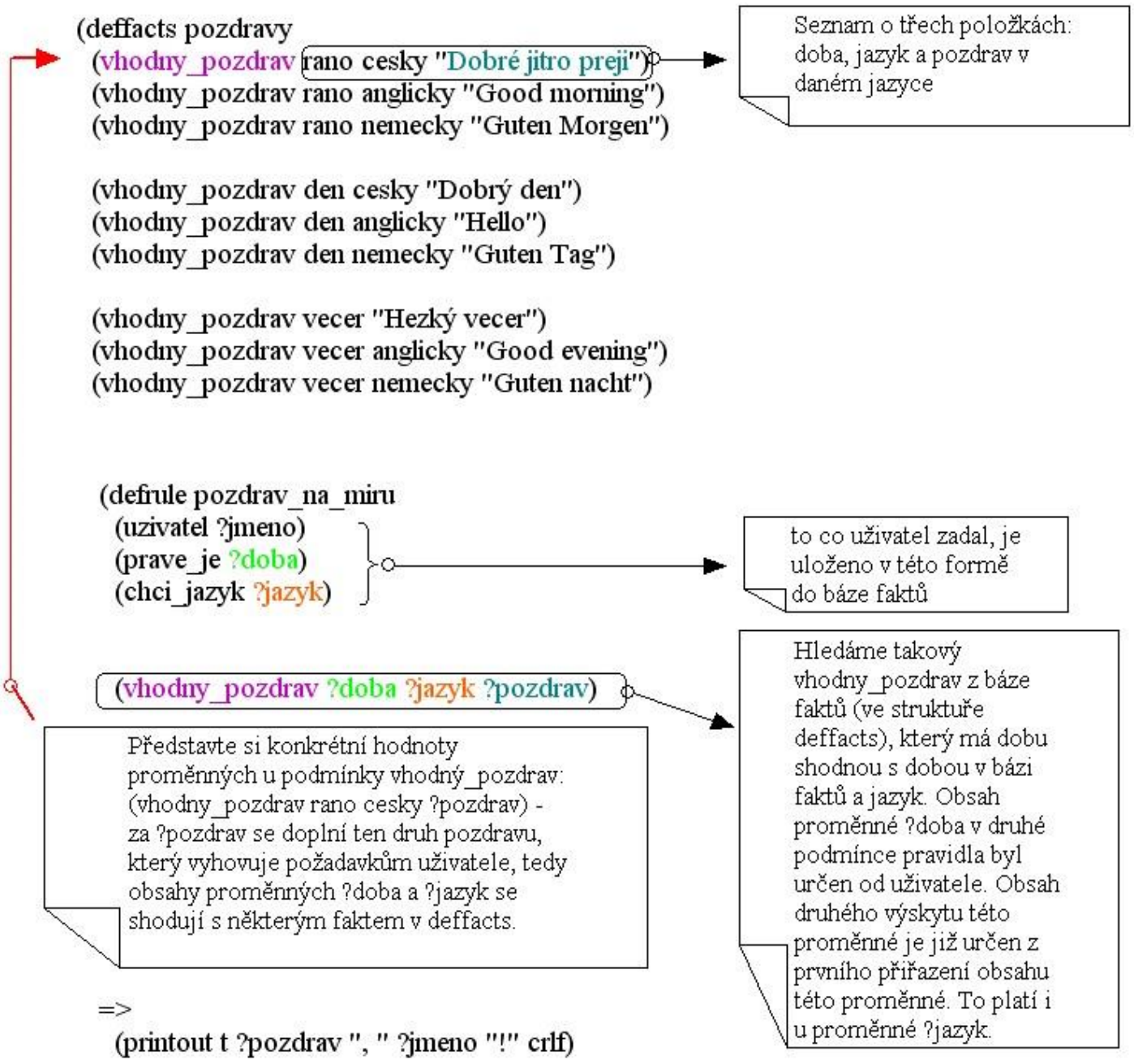
1.Tento program Vás pozdraví Vaším jménem v závislosti na zvolené denní době i jazyce.

```
;*************************************
```

```
;
;      POZDRAVY SITE NA MIRU
;      Tento program Vás pozdraví
;      Vaším jménem v závislosti na
;      zvolené denní dobe i jazyce.
;
;            (MH 2004)
;************************************


;Je treba znát: denní doby +
;jazyky, kterými se bude oslovovat +
;konkrétní oslovení

(deffacts denni_doby
  (denni_doba rano)
  (denni_doba den)
  (denni_doba vecer)
)

(deffacts jazyky
       (pozdrav cesky)
       (pozdrav anglicky)
       (pozdrav nemecky))


(deffacts pozdravy
  (vhodny_pozdrav rano cesky "Dobré jitro preji")
  (vhodny_pozdrav rano anglicky "Good morning")
  (vhodny_pozdrav rano nemecky "Guten Morgen")

  (vhodny_pozdrav den cesky "Dobrý den")
  (vhodny_pozdrav den anglicky "Hello")
  (vhodny_pozdrav den nemecky "Guten Tag")

  (vhodny_pozdrav vecer "Hezký vecer")
  (vhodny_pozdrav vecer anglicky "Good evening")
  (vhodny_pozdrav vecer nemecky "Guten nacht")
)


;************************************************

;PRAVIDLA
;Zde budeme od uživatele požadovat zadání jeho jména, denní doby a jazyka, ve
kterém požaduje oslovení.
;Všechny tyto informace si uložíme príkazem assert do báze faktu, abychom s nimi
mohli dále pracovat -
;vybrat konkrétní oslovení z báze faktù.

(defrule nacteni_dat
=>
  (printout t "-----------------------------" crlf)
  (printout t "Zadejte prosím, Vaše jméno: " crlf)
  (bind ?jmeno_uzivatele (read))
  (assert (uzivatel ?jmeno_uzivatele))

  (printout t "-----------------------------" crlf)
  (printout t "Zadejte prosím denní dobu " crlf)
  (printout t "rano---den---vecer" crlf)
  (printout t "============================" crlf)
```

```
   (bind ?denni_doba (read))
   (assert (prave_je ?denni_doba))

   (printout t "-----------------------------" crlf)
   (printout t "Zadejte jazyk pozdravu" crlf)
   (printout t "cesky--anglicky--nemecky:"crlf)
   (printout t "=============================" crlf)
   (bind ?jazyk (read))
   (assert (chci_jazyk ?jazyk))
)

;Toto pravidlo vypíše už konkrétní pozdrav vzhledem k požadavkum uživatele.

(defrule pozdrav_na_miru
   (uzivatel ?jmeno)
   (prave_je ?doba)
   (chci_jazyk ?jazyk)
   (vhodny_pozdrav ?doba ?jazyk ?pozdrav)

=>
   (printout t ?pozdrav ", " ?jmeno "!" crlf)
)

;Tedy jestliže uživatel zadal sve jmeno (jmeno je v bazi faktu)a zaroven
;Zadal denni dobu a jazyk a zaroven plati
;ze existuje takovy vhodny_pozdrav ?pozdrav, ktery se shoduje s pozadavky
uzivatele
;pak tento pozdrav vytiskni spolu se jmenem uzivatele
```

Vysvětlení graficky:



Poznámky k obrázku:

Jestliže např. uživatel zadá dobu rano (v bazi bude prave_je rano), pak zde existuji 3 varianty na výběr druhu pozdravu (Dobre jitro preji, Good morning, Guten Morgen). Pak jestliže zadáte jazyk cesky, pak zde je jen už jedna varianta pozdravu, která se může vybrat. Znázorněno červenou šipkou. V podstatě dochází ke konkretizování odpovědi.

2.Příklad na procvičení prefixu, assertu, readu, bindu - evidence různých druhů geometrických objektů

```
;*****************************************
;
;      VYPOCTY OBVODU ROVINNYCH UTVARU
;            (MH 2004)
;
```

```
;     Urcovani obvodu OBDELNIKU
;                 TROJUHELNIKU
;                 KRUHU
;
;*****************************************

(deffacts Utvary
      (utvar obdelnik)
      (utvar trojuhelnik)
      (utvar kruh)

)

;---------------<OBDELNÍK>----------------

(defrule Zadani_utvaru
=>
      (printout t "-------------------------------" crlf)
      (printout t "Zadejte nazev rovinneho utvaru:" crlf)
      (printout t "obdelnik---trojuhelnik---kruh" crlf)
      (printout t "-------------------------------" crlf)
      (bind ?utvar (read))
      ;to co uzivatel zada se ulozi do promenne utvar
      (assert (chci_obvod ?utvar))
      ;do baze faktu se ulozi to co je v promenne utvar
)


(defrule VypocetObdelnik
      (chci_obvod obdelnik)
      ;za podminky, ze si uzivatel preje vypocitat
      ;obvod obdelnika pak ...
=>
      (printout t "Zadejte stranu a:" crlf)
      (bind ?stranaA (read))
      ;stranu, kterou uzivatel zada si uloz do promenne
      ;?stranaA
      (assert (obdelnik_strA ?stranaA))
      ;tuto stranu si uloz do baze faktu

      (printout t "Zadejte stranu b:" crlf)
      (bind ?stranaB (read))
      (assert (obdelnik_strB ?stranaB))

      (bind ?vysledek (+ (* 2 ?stranaA) (* 2 ?stranaB))) ;PREFIXOVY ZAPIS !!!
      ;do promenne ?vysledek uloz vysledek ze vzorce
      (assert (obvod_obdel je ?vysledek))
      ;vysledek vypoctu uloz do baze prikazem assert
      (printout t "Obvod obdelniku:" ?vysledek crlf)
      ;to co je v promenne ?vysledek hned vypis

      ;DALSI POZNAMKY
      ;nebylo by nutne vubec ukladat jednotlive strany
      ;i konecny vysledek do baze - stacilo by jen vypsat
      ;ale baze s agendou je mozkem programu, takze
      ;se do ni spravne maji promitnout vsechny zmeny
      ;tim nam dava program najevo, ze provedl operace
      ;s temi a temi fakty a neco mu vyslo
)

;------------<TROJUHELNÍK>------------
```

```
(defrule Vypocet_trojuhelniku
      (chci_obvod trojuhelnik)
=>

      (printout t "Zadejte stranu a:" crlf)
      (bind ?stranaA (read))
      (assert (trojuhelnik_strA ?stranaA))

      (printout t "Zadejte stranu b:" crlf)
      (bind ?stranaB (read))
      (assert (trojuhelnik_strB ?stranaB))

      (printout t "Zadejte stranu c:" crlf)
      (bind ?stranaC (read))
      (assert (trojuhelnik_strC ?stranaC))

      (bind ?vysledek (+ ?stranaA ?stranaB ?stranaC))
      (assert (obvod_troj je ?vysledek))

      (printout t "Obvod trojuhelniku:" ?vysledek crlf)

)

;-------------<KRUH>-------------

(defrule Vypocet_KRUH
      (chci_obvod kruh)
=>
      (printout t "Zadejte prumer kruhu:" crlf)
      (bind ?prumer (read))
      (assert (kruh_prumer ?prumer))
      (bind ?vysledek (* (pi) ?prumer))
      ;POZOR lze zadat i 3.14, ale ne 3,14
      (assert (obvod_kruh je ?vysledek))

      (printout t "Obvod kruhu:" ?vysledek crlf)
)

;-----------<ZADANI UTVARU, KTERY V BAZI NENI>--------

(defrule neznam_utvar
      (chci_obvod ?utvar)
      ;uzivatel sice zadal utvar
      (not(utvar ?utvar))
      ;ale tento utvar neni v bazi evidovany
=>
      (printout t "Neznam tento utvar!" crlf)
      (printout t "Prosim, zadejte ho znovu:" crlf)
      (bind ?utvar (read))
      (assert (chci_obvod ?utvar)))


;Poznamka: pokud nekolikrat zadate stejne spatny nazev utvaru, pak
;vam program skonci - v tomto programu to jeste neni osetreno, protoze prikaz,
ktery
;by se pro to mohl pouzit, se tyka az jedne z dalsich lekci
```

LEKCE 4.

Náplň lekce:

- Určování počtu faktů
- Příkaz test
- Logické a porovnávací funkce
- Příkaz random
- Úlohy k procvičení

Tato lekce je věnována opakování toho co jsme se již dozvěděli v předchozích lekcích. Dále zkusíme určovat počty faktů v bázi faktů a vytvořit hru v Clipsu.

Určování počtu faktů

Začneme úlohou - Určování počtu druhů ovoce

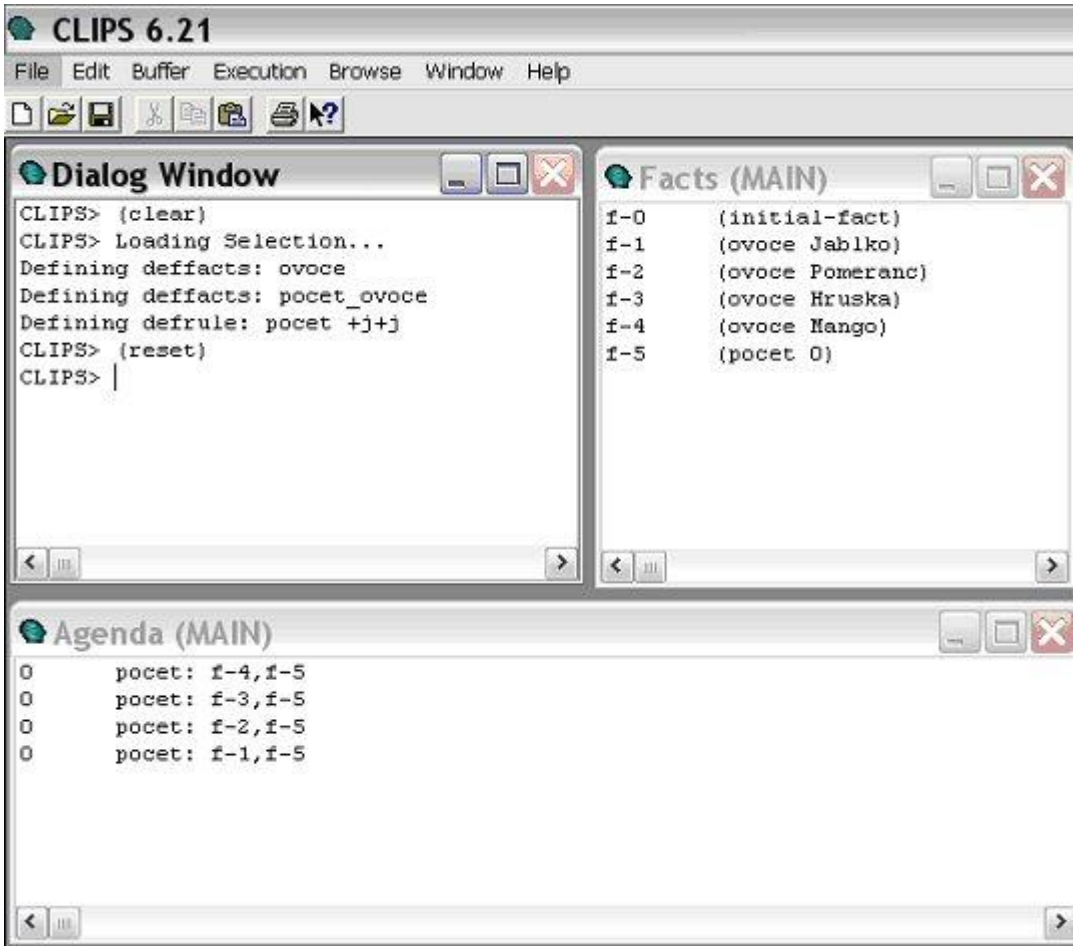
```
;*******************************************;
;                                           ;
;           OVOCE - NEFUNKCNI PROGRAM !!!  ;
;                (MH 2004)                  ;
;                                           ;
;*******************************************;

(deffacts ovoce
  (ovoce Jablko)
  (ovoce Pomeranc)
  (ovoce Hruska)
  (ovoce Mango) )

(deffacts pocet_ovoce
  (pocet 0) )


(defrule pocet
  (ovoce ?x)
  (pocet ?y)
=>
  (bind ?novy_pocet (+ ?y 1))
  (assert (pocet ?novy_pocet))
  (printout t "Pocet druhu ovoce je:" ?novy_pocet crlf)
)
```

Pozor: jedná se o nefunkční program pro vysvětlení problému !

Proč dojde k zacyklení programu ? (popis nefunkční verze programu)

Při odhalování činnosti Clipsu Vám může velmi pomoci krokovací mechanismus, kdy místo spuštění programu příkazem Run, provedete Step.

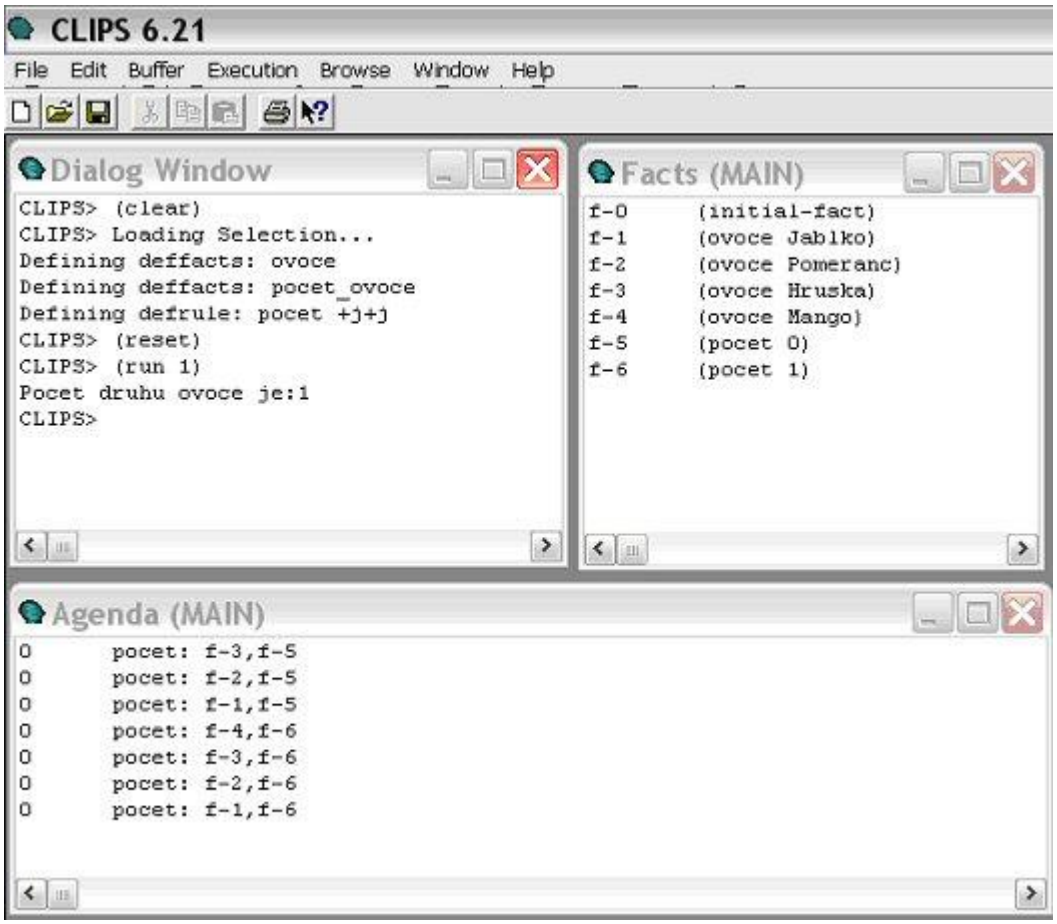
Po spuštění Reset je stav následující:



V agendě vidíme, že se budou provádět 4 pravidla pro určování počtu a to pro každý druh ovoce. To zní dobře. Tak to chceme. Dále v agendě vidíme, že se má pravidlo vykonat v závislosti na druhu ovoce a daném počtu - tedy zatím počtu 0 (f-4, f-5 zn. že dané pravidlo pro své provedení bude potřebovat tato čísla faktů z naší báze - z okna deffacts - f-4 je Mango a f-5 je počet 0.) Ten f-5 je počet 0 a bude použit i pro ostatní pravidla. Jak je patrné z agendy.

Co se stane po prvním kroku?

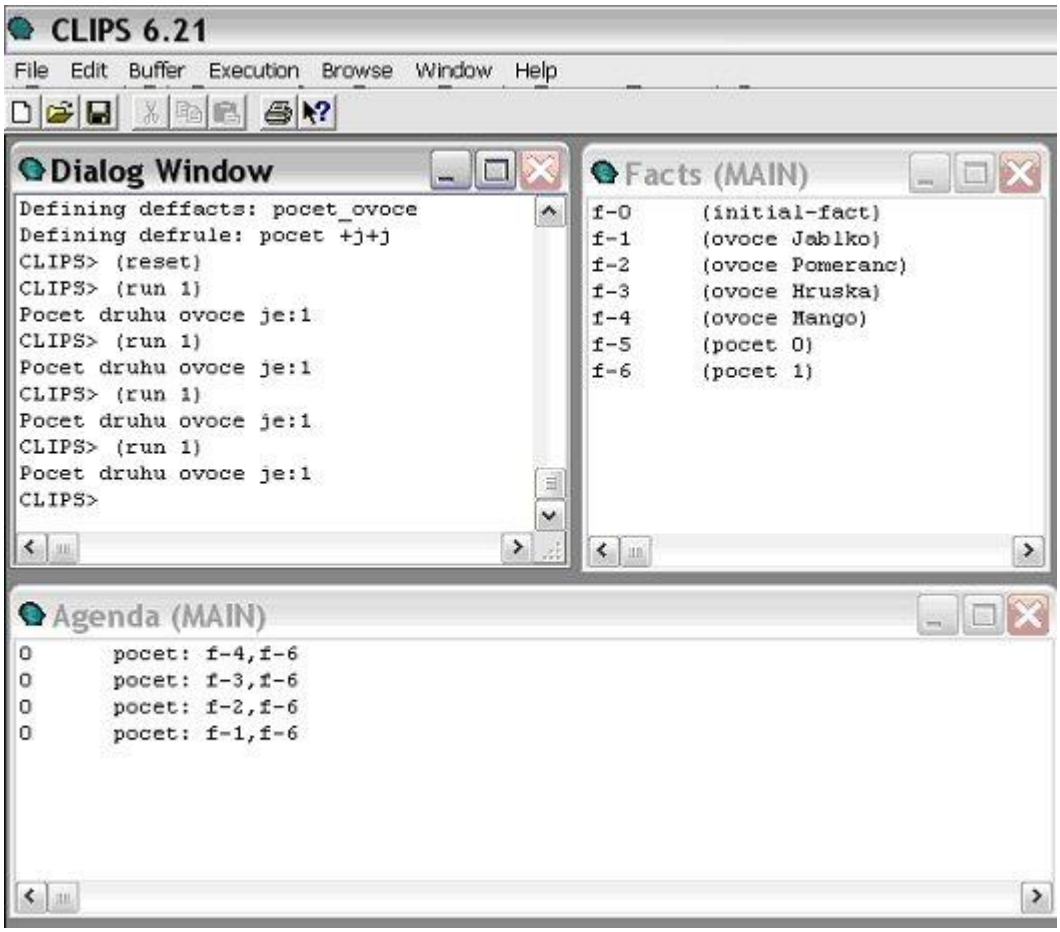
Budou se dít hotové divy, především v agendě. Pravidlo pro určení odpočítání prvního druhu ovoce (např. pro Mango) bylo provedeno, počet se zvýšil o jednotku a to se také uložilo do báze faktů. Tím nám vznikl nový fakt v bázi (pocet 1). Jedno pravidlo nám sice v agendě ubylo, ale další 4 přibyla. Proč? Problém spočívá v dané kombinaci faktů: (ovoce ?x) a (počet ?y).



Jakoby Clips nejprve dokonal to staré tzn. nejprve provede ta pravidla vzhledem k poctu 0 a to ke každému druhu ovoce (kombinace faktů: f-3,f-5;f-2,f-5;f-1,f-5). 4x se nám vypíše počet druhu ovoce je:1. Počet se sice o jedna zvyšuje, ale 4x stále od nuly. V bázi faktů ale nemáme 4x vypsáno (pocet 1), protože nám Clips nedovolí uložit do báze faktů několik naprosto stejných faktů.

Potom co dokoná to staré - vzhledem k počtu 0, pak si opět vezme počet, ale už s číslem 1 a každý druh ovoce (kombinace faktů: f-4, f-6; f-3,f-6;f-2,f-6;f-1,f-6) (f-6 značí ten počet 1). Následně provede zase dané pravidlo 4x (podle počtu druhů ovoce). V dalším kroku si opět vezme počet, ale s číslem 2 atd… Z toho vyplývá, že se nám program zacyklí.

Až po čtyřnásobném provedení daného pravidla vzhledem k jednomu počtu (např. počet 1) se počet zvýší o jedničku - přibude nový fakt v bázi a přibudou 4 nová pravidla v agendě.



Jak problém vyřešit?

Všimněte si, že počty druhů ovoce se vždy zvyšují o jedna - vlastně až donekonečna. Jestliže budeme mít v bázi faktů pocet 0 a pocet 1, jak již bylo řečeno (viz. obr. výše), Clips vykoná nejprve to staré - resp. provede pravidlo pro počet 0 čtyřikrát (pro každé ovoce). Ale my chceme, aby Clips prošel každý druh ovoce právě jednou a ne několikrát (v případě počtu 1, počtu 2, počtu 3, … počtu n) !

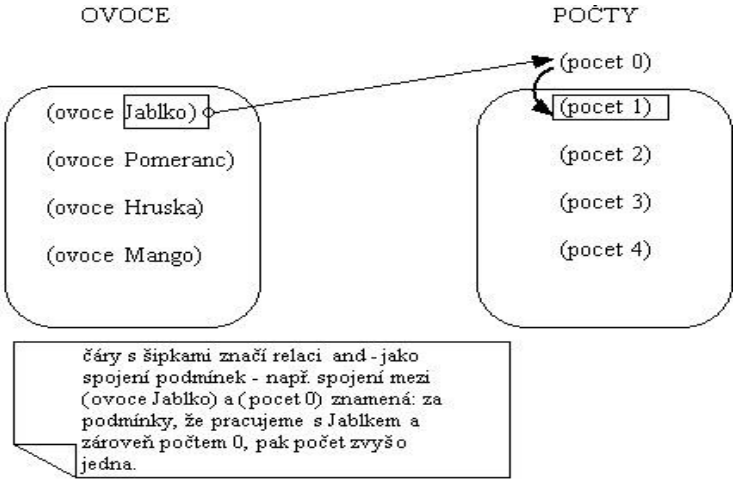
Musíme tedy programu říci, aby to ovoce, které jsme již započítali, dále neuvažoval (nepracoval s nimi). Toto je zajištěno podmínkou: (not (ovoce_pripocitano ?x))

Ale to nestačí, musí nám docházet také k pravidelnému přičítání hodnot. Tedy co druh ovoce to daný počet zvýšený o jednotku. Zároveň také chceme, aby Clips jakoby postupoval od toho počtu, od jakého chceme. Tedy jestliže máme Jablko - tak počet se má zvýšit o jedna. V dalším kroku (při realizaci pravidla pocet podruhé) Jablko už uvažovat nebudeme (zajištěno podmínkou (not (ovoce_pripocitano ?x)) a zároveň chceme postupovat už od toho počtu 1 a ne od 0 (jak by Clips chtěl). Proto musíme přidat do podmínkové části pravidla pocet ještě jednu podmínku, která zajistí, abychom pokračovali od počtu o jednotku větší - tedy u Jablka od počtu 1 (a ne 0). Toto je řešeno podmínkou:

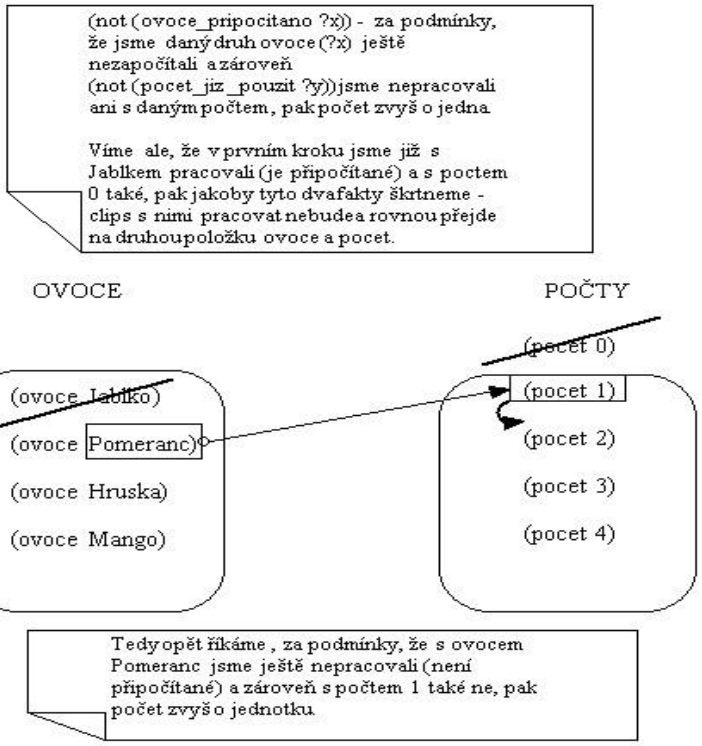
(not (pocet_jiz_pouzit ?y)).

Pokusil jsem se vysvětlit Vám tyto dvě podmínky ještě pomocí obrázku, doufám, že ne moc složitě.

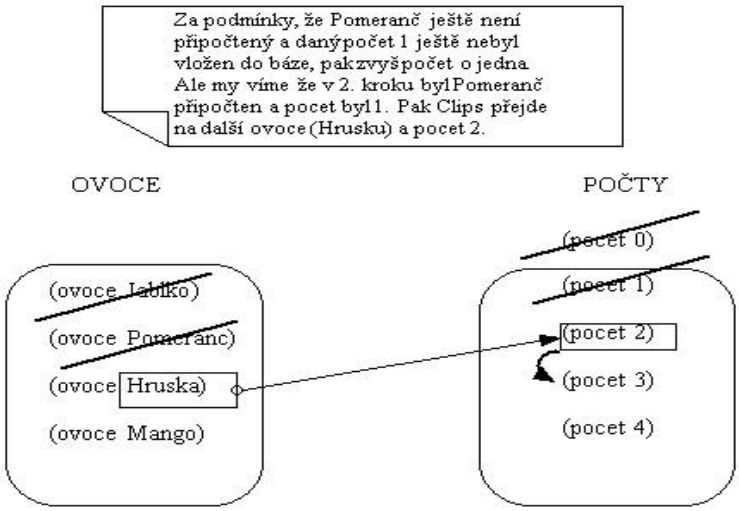
1. KROK



2. KROK



3. KROK



Tento problém lze řešit ještě jinou metodou a to pomocí příkazu retract. Nechci se zde o něm příliš dlouho rozepisovat, ale ve zkratce slouží k vymazání určitého faktu z báze faktů.

Možná řešení úlohy s ovocem (vyřešení zacyklení programu)

Verze č. 1:

Vy vlastně z báze můžete zrušit daný neaktuální počet - v bázi ho vůbec neudržovat

```
;*****************************************
;
;     URCOVANI POCTU DRUHU OVOCE
;          S RETRAKTEM (SINGLE)
;             (MH 2005)
;
;*****************************************


(deffacts ovoce
  (ovoce Jablko)
  (ovoce Pomeranc)
  (ovoce Hruska)
  (ovoce Mango) )

(deffacts pocet_ovoce
  (pocet 0) )


(defrule pocet
  (ovoce ?x)
  (not (ovoce_pripocitano ?x))
  ?pryc<-(pocet ?y)
  ;do promenne ?pryc si ulozite to co chcete smazat, ten fakt, ktery chcete dat z
baze pryc

=>
  (bind ?novy_pocet (+ ?y 1))
  (retract ?pryc)
  ;prikaz zruseni obsahu promenne ?pryc - v ni je ten vas fakt - resp. jeho cislo

  (assert (pocet ?novy_pocet))
  (assert (ovoce_pripocitano ?x))
  (printout t "Pocet druhu ovoce je:" ?novy_pocet crlf))
```

Verze č.2

… nebo neudržovat ani dané ovoce, ani daný počet v bázi faktů. Toto řešení ale není příliš chytré, protože dané ovoce Vám z báze zmizí. Sice si v Clipsu můžete vypsat, co v bázi bylo (tak jako to je řešeno v programu), ale reálnému uvažování člověka se to příliš nepodobá. To je jako byste zapomněli, co jste před chvílí počítali (jestliže se to týká např. ovoce, tak je to méně pravděpodobné).

```
;*******************************************;
;                                           ;
;                  OVOCE a RETRACT   (DOUBLE)    ;
;                   (MH 2005)        ;
;                                          ;
```

```
;*******************************************;

(deffacts ovoce
  (ovoce Jablko)
  (ovoce Pomeranc)
  (ovoce Hruska)
  (ovoce Mango) )

(deffacts pocet_ovoce
  (pocet 0) )


(defrule pocet
  (ovoce ?x)
  (pocet ?y)
?prycO<-(ovoce ?x)
?prycP<-(pocet ?y)
;budete rusit jak ovoce tak pocet

=>
  (bind ?novy_pocet (+ ?y 1))
  (assert (pocet ?novy_pocet))
  (retract ?prycO ?prycP)
  (printout t "V bazi je ovoce:" ?x crlf)
  (printout t "Pocet druhu ovoce je:" ?novy_pocet crlf))
```

Příkaz test
Logické a porovnávací funkce
Příkaz random

Příkaz TEST
Tento příkaz nám slouží k otestování platnosti určitého tvrzení. Můžeme např. otestovat jestli daná proměnná nabývá určité hodnoty.

Tímto způsobem:

(pocet_mistnosti ?pocet)
(test (<> ?pocet 0))
Říkáme: za podmínky, že je určitý počet místností evidován a zároveň tento počet místností (?pocet) není roven (<>) 0, pak se provede akční část pravidla např. počet místností vypíšeme tak jako to je uvedeno v souboru mistnosti_pocty.clp viz. pravidlo vypis_počet_mistnosti. S tím souvisejí i zápisy rovností, nerovností apod.

LOGICKÉ FUNKCE
klíčové slovo význam slova
not boolean not
and boolean and
or boolean or

POROVNÁVACÍ FUNKCE
klíčové slovo význam slova
eq rovnost (libovolných typů)
neq nerovnost (libovolných typů)

= rovnost u číselných typů
<> nerovnost u číselných typů
>= větší nebo rovno
> větší
<= menší nebo rovno
< menší než

Příkaz RANDOM
Tento příkaz Vám umožní vygenerovat náhodnou sadu čísel ve Vámi definovaném rozmezí.

Syntaxe:(random dolni horni) - dolni znamená spodní hranici intervalu a horni logicky horní hranici intervalu.

Tedy když definujete příkaz např. takto: (random 1 50), tak Vám tento příkaz vygeneruje sadu čísel v intervalu od 1 do 50ti.. Obsahem závorky je konkrétní vygenerovaná hodnota, která např. může být uložena do proměnné pomocí příkazu bind (tak jako to je ukázáno v programu s hrou).

Úlohy k procvičení

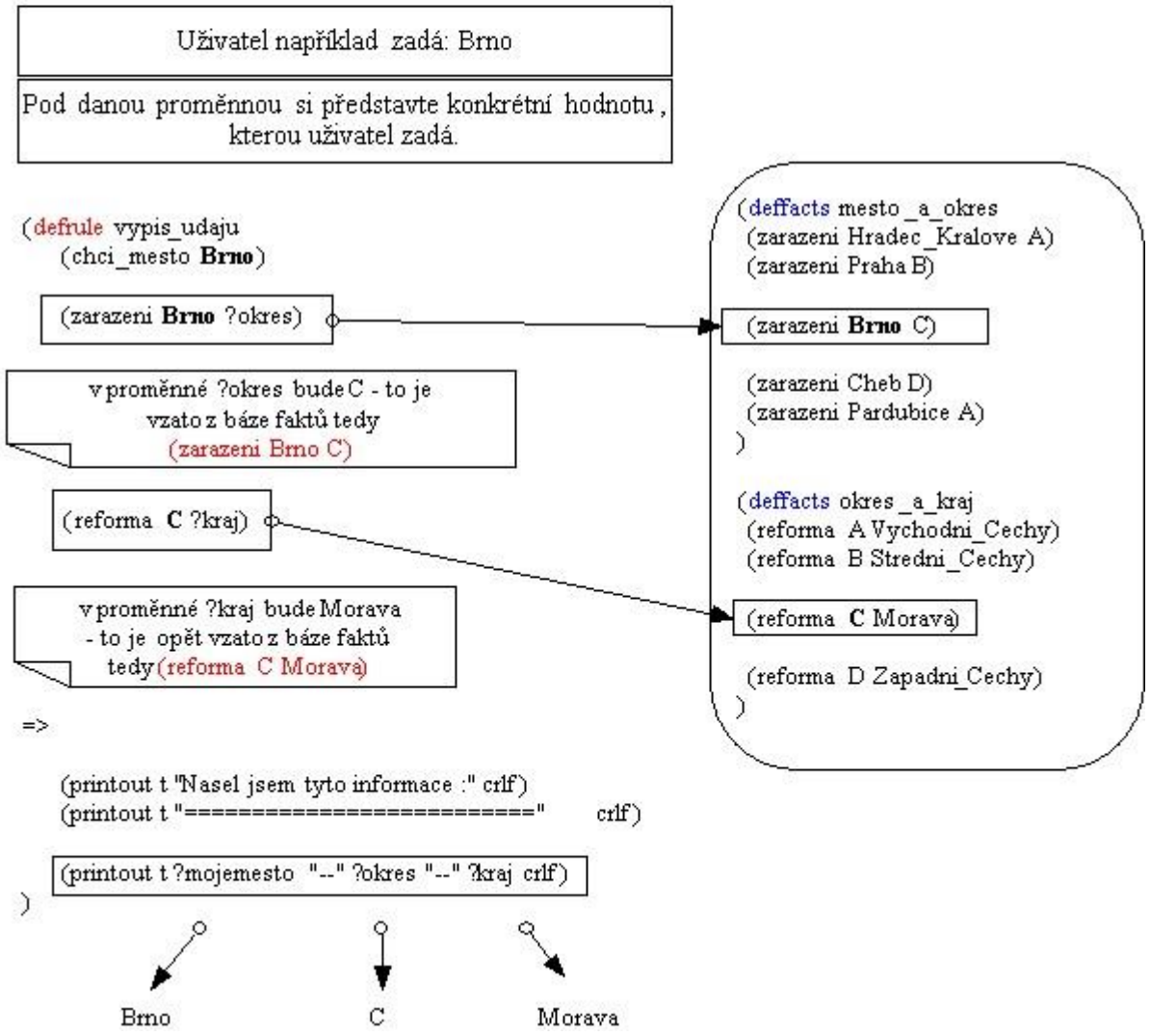
Úloha č.1: Města

Zadání:

1. Do báze faktů uložte údaje ve tvaru (zařazení město okres), kam uložíte údaje o příslušnosti několika měst do okresů. Ve tvaru (reforma okres kraj), kam uložíte údaje o začlenění okresů do krajů.
2. Napište pravidla, pomocí kterých načtete město, které zajímá uživatele a vypíšete, do kterého okresu a kraje město patří.

Vysvětlení funkce pravidla pro výpis údajů

```
(defrule vypis_udaju
    (chci_mesto ?mojemesto )
    (zarazeni ?mojemesto  ?okres)
    (reforma  ?okres ?kraj)
=>
    (printout t "Nasel jsem tyto informace :" crlf)
    (printout t "============================"      crlf)
    (printout t ?mojemesto  "--" ?okres "--" ?kraj crlf)
)
```




```
;****************************************;
;                                        ;
;              MESTA                     ;
;              (MH 2004)                 ;
;                                        ;
;****************************************;
```

```
;Zadani:
;Do baze faktu ulozte udaje
;ve tvaru (zarazení mesto okres), kam ulozite udaje o prislusnosti nekolika mest
do okresu.
;Ve tvaru (reforma okres kraj), kam ulozite udaje o zacleneni okresu do kraju.
;Napiste pravidla, pomoci kterych nactete mesto, ktere zajima uzivatele a
vypisete, do ktereho okresu a kraje mesto patri.

;Poznamka
;pismena A B C D A znamenaji urcity okres


(deffacts mesto_a_okres
      (zarazeni Hradec_Kralove A)
      (zarazeni Praha B)
      (zarazeni Brno C)
      (zarazeni Cheb D)
      (zarazeni Pardubice A)
)

(deffacts okres_a_kraj
      (reforma A Vychodni_Cechy)
      (reforma B Stredni_Cechy)
      (reforma C Morava)
      (reforma D Zapadni_Cechy)
)


;=======================================

(defrule nacteni_mesta
=>
      (printout t "Zadejte nazev hledaneho mesta:" crlf)
      (bind ?mojemesto (read))
      (assert (chci_mesto ?mojemesto))
)

;pravidlo, ktere nacte to mesto, ktere uzivatel zada - chce vyhledat
;prikazem assert je toto mesto ulozeno do baze
;ve forme (chci_mesto ?mojemesto)


(defrule vypis_udaju
      (chci_mesto ?mojemesto)
      ;jestlize uzivatel jiz zadal dane mesto
      ;tedy v bazi faktu je jeho mesto ve forme faktu (chci_mesto ?mojemesto)
      ;vsimnete si, ze v pravidle nacteni_mesta a vypis_udaju uzivam stejny nazev
      ;pro promenou ?mojemesto - jedna se o lokalni promennou ne globalni
      ;tzn. ze dana promenna se stejnym nazvem muze nabyvat ruzneho obsahu ve vice
pravidlech

      (zarazeni ?mojemesto ?okres)
      ;za podminky, ze uzivatelovo mesto (?mojemesto) se shoduje s nejakym mestem
      ;v bazi faktu, pak k nemu chci znat dany okres (?okres)

      (reforma ?okres ?kraj)
      ;a k tomuto okresu, ktery byl zjisten z predchozi podminky chci jeste
zjistit kraj (?kraj)
=>
      ;za podminky, ze jsou splneny vsechny vyse definovane podminky, pak vytiskni
mesto, okres a kraj
```

```
      (printout t "Nasel jsem tyto informace:" crlf)
      (printout t "=========================" crlf)
      (printout t ?mojemesto "--" ?okres "--" ?kraj crlf)
)



(defrule neznam_mesto
      (chci_mesto ?mojemesto)
      ;jestlize uzivatel dane mesto jiz zadal, ale
      ;toto ?mojemesto neni shodne s zadnym jinym mestem ve faktu zarazeni, pak
...
      (not (zarazeni ?mojemesto ?okres))
=>
      ;dej uzivateli vedet o tom, ze zadne mesto neni v databazi mest evidovano a
      ;pozaduj od uzivatele, aby mesto zadal znovu viz. prikaz bind + assert
      (printout t "V me databazi neni toto mesto evidovano!" crlf)
      (printout t "Zadejte jine mesto, prosim:" crlf)
      (bind ?mojemesto (read))
      (assert (chci_mesto ?mojemesto))
)
```

Úloha č.2: Místnosti

V bázi faktů jsou popsány rozměry místností v prostorném domě:

```
(mistnost A 5 5)
(mistnost B 6 6)
(mistnost C 4 7)
(mistnost D 3 4)
(mistnost E 6 5)
(mistnost F 4 3)
```

(Pozn. písmeno označuje místnost, čísla její rozměry v metrech.)

Zadání:

1. napište pravidlo, kterým spočítáte místnosti v domě
2. napište pravidlo, kterým spočítáte čtvercové místnosti v domě
3. napište pravidlo, jímž zjistíte celkovou plochu všech místností

Doporučení: Každé pravidlo si napište a zkuste spustit zvlášť, všechna naráz nespouštějte.

Řešení části 1.

```
;*****************************************;
;                                         ;
;     MISTNOSTI - urceni jejich poctu       ;
;           (MH 2004)                     ;
;                                         ;
;*****************************************;
;tato uloha je resena stejnym zpusobem jako ovoce, o kterych
;se mluvilo drive v lekci 4
;zatim zde neeviduji rozmery, neni to pro tuto ulohu treba

(deffacts mistnosti
```

```
      (mistnost A)
      (mistnost B)
      (mistnost C)
      (mistnost D)
      (mistnost E)
      (mistnost F)

        ;klidne muzete zavest novou strukturu deffacts pro
      ;pocet_mistnosti

      (pocet_mistnosti 0)
)


;toto pravidlo funguje stejne, jako u prikladu s ovocem
(defrule pocet_mistnosti
      (mistnost ?nejaka)
      (pocet_mistnosti ?pocet)
      (not (mam_evidovanou_mistnost ?nejaka))
      (not (predchozi_pocet_mistnosti ?pocet))
=>
      (bind ?novy_pocet (+ ?pocet 1))
      (assert (pocet_mistnosti ?novy_pocet))
      (assert (mam_evidovanou_mistnost ?nejaka))
      (assert (predchozi_pocet_mistnosti ?pocet))
)


(defrule vypis_pocet_mistnosti
      (pocet_mistnosti ?pocet)
      (test (<> ?pocet 0))

=>
      (printout t ?pocet ".mistnost" crlf))

;toto pravidlo sice neni nutne uvadet, ale je to vhodne
;slouzi k tomu, aby se nam vypisoval pocet mistnosti jen v pripade, ze
;jejich pocet je vetsi nez 0 - tedy alespon jedna mistnost je evidovana
;v nasi bazi faktu - to zajistujeme novym prikazem test

;toto pravidlo se vykona jen v tom pripade, kdyz v bazi bude alespon jedna
mistnost
;tedy pocet mistnosti nebude nulovy



(defrule vypis_hlavicku
=>
      (printout t "========================" crlf)
      (printout t "POCET MISTNOSTI:" crlf)
      (printout t "========================" crlf)
)

;neni nutne definovat toto pravidlo, ale vhodne
;vypise Vam jakesi zahlavi - o co se jedna, co resime
```

Řešení části 2. Pro určení počtu čtvercových místností je třeba si uvědomit, jaké charakteristiky má čtvercová místnost. Má stejné rozměry pro stranu a i b. Z toho vyjdeme.

```
(defrule pocet_mistnosti
  (mistnost ?nejaka ?rozmerA ?rozmerB)
  (test(= ?rozmerA ?rozmerB))
```

Vlastně tímto pravidlem říkáme: za podmínky, že je v naší bázi evidována nějaká místnost s rozměry ?rozmerA ?rozmerB, pak pro zjištění toho, jestli se jedná o čtvercovou místnost, použijeme příkaz test. Otestujeme si shodnost rozměrů místnosti (rovnost proměnných ?rozmerA ?rozmerB). Pak to řešíme stejným způsobem jako v předchozím případě.

```
;*********************************************;
;                                             ;
;     MISTNOSTI - urceni poctu ctvercovych m.  ;
;               (MH 2004)                      ;
;                                             ;
;*********************************************;
;tato uloha je resena temer stejnym zpusobem jako ovoce, o kterych
;se mluvilo drive v lekci 4
;zde uz jsou evidovany rozmery mistnosti

(deffacts mistnosti
      (mistnost A 5 5)
      (mistnost B 6 6)
      (mistnost C 4 7)
      (mistnost D 3 4)
      (mistnost E 6 5)
      (mistnost F 4 3)

      (pocet_ctverec_mistnosti 0)
)

(defrule pocet_mistnosti
      (mistnost ?nejaka ?rozmerA ?rozmerB)
      (test(= ?rozmerA ?rozmerB))
      ;za podminky, ze jsou rozmery mistnosti stejne, pak ...
      (pocet_ctverec_mistnosti ?pocet)
      (not (existuje_ctverec_mistnost ?nejaka))
      (not (predchozi_pocet_mistnosti ?pocet))
=>
      (bind ?novy_pocet (+ ?pocet 1))
      (assert (pocet_ctverec_mistnosti ?novy_pocet))
      (assert (existuje_ctverec_mistnost ?nejaka))
      (assert (predchozi_pocet_mistnosti ?pocet))
)


(defrule vypis_pocet_mistnosti
      (pocet_ctverec_mistnosti ?pocet)
      (test (<> ?pocet 0))
=>
      (printout t ?pocet"." "ctvercova místnost" crlf)
)


(defrule vypis_hlavicku
=>
      (printout t "Je evidovano:" crlf)
      (printout t "==========================" crlf)
)
```

Řešení části 3. Pro vyřešení tohoto problému je vhodné si úlohu rozdělit na dvě části.

1. část: nejprve vypočítáme obsah pro každou jednotlivou místnost - tedy máme místnost A s rozměry 5 a 5 - takže první hodnota bude 5x5=25. Pak to samé učiníme pro ostatní místnosti a informaci o výsledcích obsahů u každé místnosti vložíme do báze faktů, abychom s těmito hodnotami mohli dále pracovat. Toto je zajištěno pravidlem: vypocti_obsah_jednotlivych_mistnosti. Vložený fakt bude mít název obsah_mistnosti ?oznaceni ?obsah /název místnosti + její obsah/

2. část: dále pak vytvoříme pravidlo, které nám sečte všechny tyto obsahy jednotlivých místností, přičemž to poslední číslo bude to naše hledané. Princip je velmi podobný příkladu s místnostmi.

Pravidlo vypocti_obsah_jednotlivych_mistnosti, nám do báze faktů umístí tyto fakty:

```
(defrule secti_obsahy
    (obsah_mistnosti ?oznaceni ?obsah)
    (soucty_obsahu ?mezivysledek)
    (not(pricteny_obsah_mistnosti ?oznaceni))
    (not(stary_soucet ?mezivysledek))
=>
    (bind ?aktualniobsah (+ ?mezivysledek ?obsah))
    (assert (soucty_obsahu ?aktualniobsah))
    (assert (pricteny_obsah_mistnosti ?oznaceni))
    (assert (stary_soucet ?mezivysledek))
)
```

```
(obsah_mistnosti A 25)
(obsah_mistnosti B 36)
(obsah_mistnosti C 28)
(obsah_mistnosti D 12)
(obsah_mistnosti E 30)
(obsah_mistnosti F 12)
```

Vždy budeme sčítat daný rozměr místnosti s tím co je uvedeno v proměnné ?mezivysledek (tedy s tím co již bylo přičteno).

Od této hodnoty budeme začínat

(soucty_obsahu 0)

PRINCIP

Velmi zjednodušeně si můžete představit, že budeme sčítat tyto hodnoty: pro místnost A (0+25=25 - tedy v bázi se objeví toto (soucty_obsahu25) - 25 je náš mezivýsledek.

Dále pro místnost B: (25+36=61) - mezivýsledek bude61 a fakt (soucty_obsahu61).
Dále pro místnost C: (61+28=89),
pro D: (89+12=101),
pro E: (101+30=131),
pro F: (131+12=143).

Nezáleží na tom, v jakém pořadí se obsahy sčítají, jen se rozměr pro jednu místnost nesmí přičíst více než jednou - dáno podmínkou (not(pricteny_obsah_mistnosti ?oznaceni)). Konečný výsledek je ale stejný s výše uvedeným postupem.

```
;*********************************************************
;
;      MISTNOSTI - celkova plocha vsech mistnosti
;                  (MH 2004)
;
;*********************************************************


(deffacts mistnosti
      (mistnost A 5 5)
```

```
      (mistnost B 6 6)
      (mistnost C 4 7)
      (mistnost D 3 4)
      (mistnost E 6 5)
      (mistnost F 4 3)

      (soucty_obsahu 0)
)



;Pravidlo pro urceni plochy kazde mistnosti

(defrule vypocti_obsah_jednotlivych_mistnosti
      (mistnost ?oznaceni ?rozmerA ?rozmerB)
=>
      (bind ?obsah (* ?rozmerA ?rozmerB))
      (assert (obsah_mistnosti ?oznaceni ?obsah))
)



;Pravidlo pro secteni vsech obsahu techto mistnosti

(defrule secti_obsahy
      (obsah_mistnosti ?oznaceni ?obsah)
      (soucty_obsahu ?mezivysledek)
      (not(pricteny_obsah_mistnosti ?oznaceni))
      (not(stary_soucet ?mezivysledek))
=>
      (bind ?aktualniobsah (+ ?mezivysledek ?obsah))
      (assert (soucty_obsahu ?aktualniobsah))
      (assert (pricteny_obsah_mistnosti ?oznaceni))
      (assert (stary_soucet ?mezivysledek))
)


;Pravidlo, ktere vypise mezisoucty a take konecny vysledek

(defrule vypis
      (soucty_obsahu ?mezisoucty)
=>
      (printout t ?mezisoucty crlf)
)




(defrule vypis_hlavicky
=>
      (printout t "=====================================" crlf)
      (printout t "PREHLED MEZISOUCTU" crlf)
      (printout t "Posledni hodnota = HLEDANY VYSLEDEK" crlf)
      (printout t "=====================================" crlf)
)
```

Úloha č. 3 - Hra

Naprogramujte jednoduchou hru: počítač vygeneruje náhodné číslo v rozsahu 1 až 10 a uživatel má určitý počet pokusu, aby číslo uhodl. Po každém neúspěšném pokusu program sdělí, zda hledané číslo je větší, nebo menší, než uživatelův tip.

Tato úloha je už docela složitá, ale myslím, že se pochopit dá.

```
;*****************************************
;
;            HRA - UHODNI CISLO
;                 (MH 2004)
;
;*****************************************
;Zadani:
;Naprogramujte jednoduchou hru: pocitac vygeneruje nahodne cislo v rozsahu 1 až 10
a uzivatel má urcity pocet pokusu, aby cislo ;uhodl. Po kazdem neuspesnem pokusu
program sdeli, zda hledane cislo je vetsi, nebo mensi, nez uzivateluv tip.

(deffacts pokusy
      (pocet_pokusu 0) )

;pocet pokusu, ktere ma uzivatel na uhadnuti vygener. cisla


;--------------------------------

(defrule generuj
      (not(vygenerovane ?nejake))
      ;musime zajistit, aby se vygenerovalo jen jedno cislo
      ;tedy za podminky, ze v bazi faktu jeste neni informace o tom, ze
      ;by bylo nejake cislo vygenerovano, pak ho vygeneruj
=>
      (bind ?cislo (random 1 10))
      ;samotne vygenerovani cisla a jeho ulozeni do promenne ?cislo

      (assert (vygenerovane ?cislo)) )
      ;do baze se vlozi info. o vygenerovani cisla, tim se zabrani
      ;spolu s prvni podminkou, ze nedojde k opakovani tohoto pravidla

;pravidlo, ktere mi zajisti vygenerovani jen jednoho cisla a to
;v intervalu od 1 do 10

;--------------------------------
(defrule zadej_cislo
      (vygenerovane ?cislo)
      ;za podminky, ze cislo bylo vygenerovano

      (not(pocet_pokusu 3))
      ;a zaroven za podminky, ze se nevyplytvaly pokusy na uhodnuti cisla
      ;pomerne dulezita podminka - uzivatel ma celkem 3 pokusy na uhodnuti cisla

      (pocet_pokusu ?pocet)
      ;sledovani celk. poctu pokusu a jeho nasledne zvyseni o jedna

      (not (cislo_uhodnuto))
      ;opet dulezita podminka, uzivatel jiz nema byt dotazovan na svuj tip, kdyz
      ;se mu podarilo uhodnout cislo na mene nez 3 pokusy

=>
      (printout t "Zadejte cislo:" crlf)
```

```
      (bind ?mojecislo (read))
      (assert (moje_cislo ?mojecislo))
      ;ulozeni uzivatelova tipu do baze

      (bind ?pridejpokus (+ ?pocet 1))
      ;pocet pokusu musime zvysit o jedna

      (assert (pocet_pokusu ?pridejpokus)) )


;pravidlo se 3x zepta uzivatele na jeho tip (na cislo, ktere mohlo byt
vygenerovano)

;---------------------------------

(defrule vygenerovane_je_vetsi
      (vygenerovane ?gen)
      (moje_cislo ?moje)
      (test (> ?gen ?moje))
=>
      (printout t "Vase cislo je mensi nez ktere jsem vygeneroval" crlf))

;toto pravidlo informuje uzivatele o tom, ze jeho tip (cislo) je mensi nez
vygenerovane

;---------------------------------


(defrule vygenerovane_je_mensi
      (vygenerovane ?gen)
      (moje_cislo ?moje)
      (test (< ?gen ?moje))

=>
      (printout t "Vase cislo je vetsi nez ktere jsem vygeneroval" crlf))

;toto pravidlo informuje uzivatele o tom, ze jeho tip (cislo) je vetsi nez
vygenerovane

;---------------------------------

(defrule rovnost
      (vygenerovane ?gen)
      ;cislo bylo vygenerovano pocitacem

      (moje_cislo ?moje)
      ;uzivatel zadal svuj tip na cislo

      (test (= ?gen ?moje))
      ;porovnani obou cisel (pocitac x clovek)

=>
      (assert (cislo_uhodnuto)) )
      ;do baze se ulozi info. o tom, ze uzivatel jiz cislo uhodl
      ;pravidlo zadej_cislo se nevykona - neni splnena cela podminkova
      ;cast pravidla

;uzivatel se trefi do hodnoty vygenerovaneho cisla

;---------------------------------
```

```
(defrule konec_hry_1
      (pocet_pokusu 3)
      (not (cislo_uhodnuto))
=>
      (printout t " " crlf)
      (printout t "Uz jste VYCERPAL/A pocet pokusu na uhodnuti cisla" crlf)
      (printout t "-----------------------------------------------------" crlf)
      (printout t "KONEC HRY" crlf) )

;existuje prvni moznost ukonceni hry
;kdyz uzivatel na tri pokusy cislo neuhodne

;--------------------------------

(defrule konec_hry_2
      (cislo_uhodnuto)
=>
      (printout t "Gratuluji, strefil/a jste se !!!" crlf)
      (printout t "-----------------------------------------------------" crlf)
      (printout t "KONEC HRY" crlf) )

;a nebo se mu podari do tri pokusu cislo uhodnout
;vsimnete si podminkove casti tohoto pravidla (cislo_uhodnuto)

;Kde se tato podminka vzala? V jakem pravidle doslo k vlozeni tohoto faktu do
baze?
;Pokud rikate, ze je to v pravidle rovnost, pak se pochvalte :-)
;(v akcni casti pravidla (posledni prikaz))


;--------------------------------
(defrule vypis_pocet_pokusu
=>
      (printout t "====================" crlf)
      (printout t "HRA - UHODNI CISLO" crlf)
      (printout t "Mate 3 pokusy !!!" crlf)
      (printout t "====================" crlf))

;jen pravidlo pro vypis hlavicky
```

LEKCE 5.

Náplň lekce:

- Detailnější pohled na seznamy
- Vícehodnotové proměnné
- Vysvětlující příklady pro práci se seznamy
- Základní funkce pro práci se seznamy:length$ a member$
- Retract
- Příklady k procvičení

Detailnější pohled na seznamy

Seznam je určitá datová struktura, která může nabývat libovolného množství položek (může se jednat i o prázdný seznam). Pokud fakt obsahuje jen jednu hodnotu - jedná se o jednoduchý fakt. Jestliže obsahuje více hodnot - jedná se o seznam. Položky seznamu se oddělují mezerami. Záleží na Vás, jestli jednotlivé položky seznamu budou mít určité uspořádání např. seznam bude obsahovat soubor čísel od nejmenšího po největší (seznam_cisel 2 5 9 12 13 18 25) nebo bude seznam obsahovat libovolné položky bez vztahů (ovoce hruska jablko pomeranc kiwi)

Se seznamy resp. práce s nimi je spojen pojem vícehodnotová proměnná.

Vícehodnotové proměnné

V předchozích příkladech jsme pracovali s takovými proměnnými, které mohly nabývat jen jedné hodnoty - např. ?ovoce - tato proměnná mohla nabývat jen hodnoty hruska nebo jablko nebo pomeranc, ale ne všechny tyto hodnoty najednou. Název už Vám mohl napovědět, že jsme dosud pracovali s tzv. jednohodnotovými proměnnými.

Příklady jednohodnotových proměnných:

```
(osoba Karel)
(zvire Pes) …
```

Existují ještě proměnné, které mohou nabývat více hodnot. Jedná se potom o tzv. vícehodnotové proměnné, které se označují takto: $?nazev_promenne. Znaky před nazvem_promenne jsou dolar a otazník. Tento druh seznamu se užívá např., když si nejste jisti kolik hodnot daný seznam má. Co se týká jejich použití v pravidlech, tak se v akční části pravidla používá tento zápis pro vícehodnotovou proměnnou ?nazev_promenne tj. bez dolaru. (doporučuje se ho v akční části nepoužívat, ale když ho zde užijete - nezjistila jsem, že by to byla nějaká výrazná chyba).

Platí:
jednohodnotová proměnná musí vždy obsahovat právě jednu hodnotu. U vícehodnotové proměnné (u seznamu) nemusí být žádná hodnota.

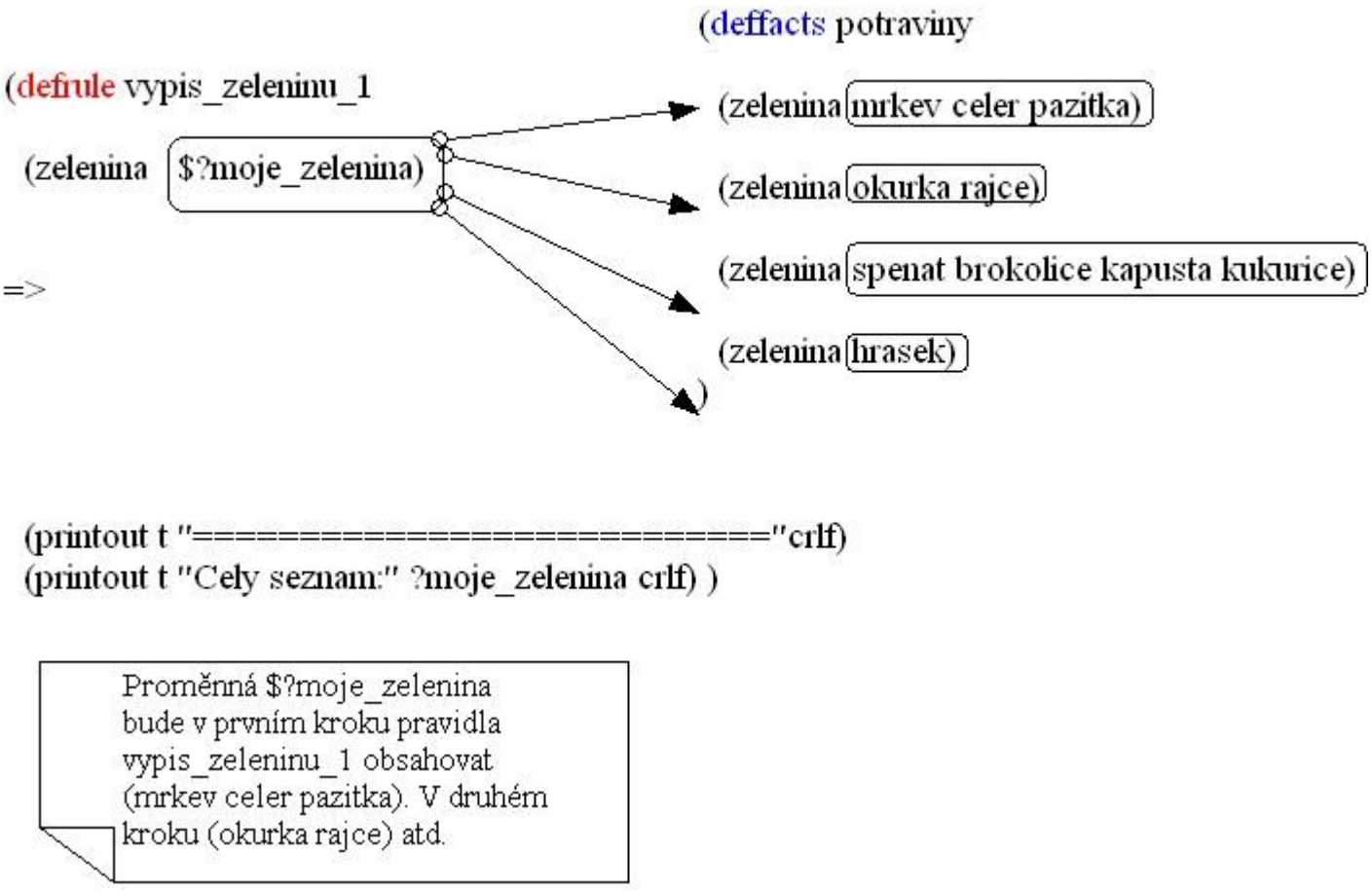
Příklady pro práci se seznamy (způsoby výpisů hodnot):

Příklad 1.

```
;************************************
;
;      ZELENINA - vypis druhu 1.
;            (MH 2005)
;
;************************************


(deffacts potraviny

  (zelenina mrkev celer pazitka)
  (zelenina okurka rajce)
  (zelenina spenat brokolice kapusta kukurice)
  (zelenina hrasek)
)
```

```
(defrule vypis_zeleninu_1
  (zelenina $?moje_zelenina)
=>
  (printout t
"============================================================"crlf)
  (printout t "Cely seznam:" ?moje_zelenina crlf) )
```

Grafické vysvětlení činnosti programu zelenina_1.clp

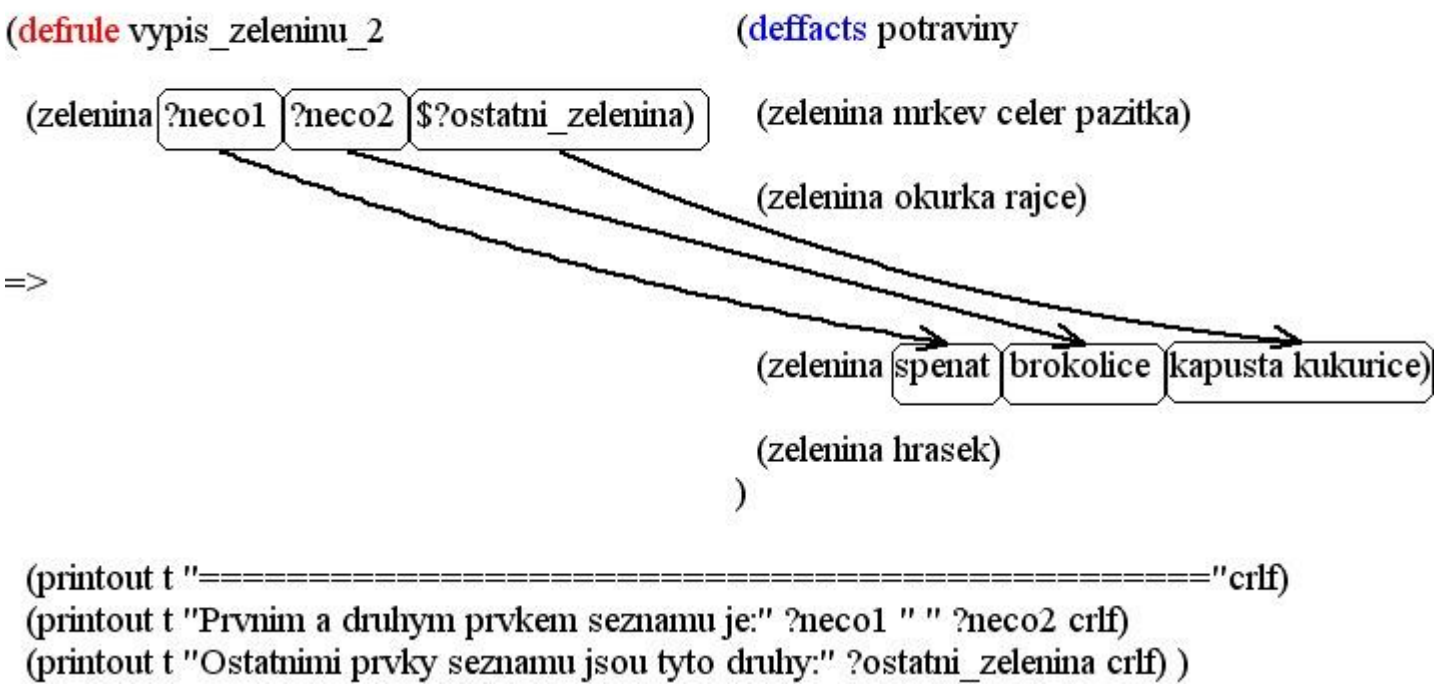


Příklad 2.

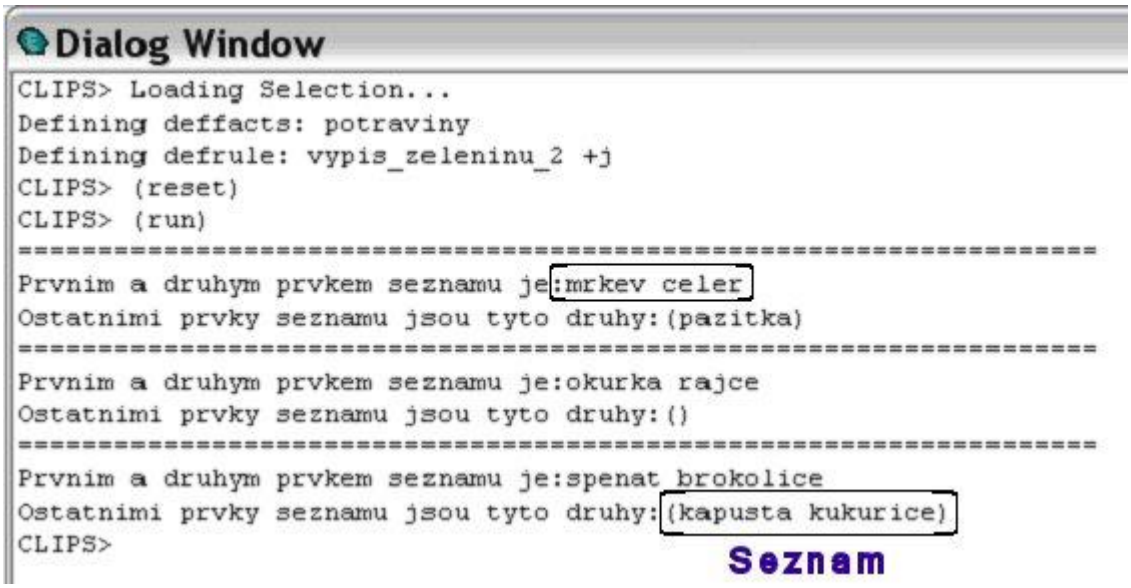
```
;***********************************
;
;      ZELENINA - vypis druhu 2.
;            (MH 2005)
;
;***********************************


(deffacts potraviny

  (zelenina mrkev celer pazitka)
```

```
   (zelenina okurka rajce)
   (zelenina spenat brokolice kapusta kukurice)
   (zelenina hrasek)
)


(defrule vypis_zeleninu_2
   (zelenina ?neco1 ?neco2 $?ostatni_zelenina)
=>
   (printout t
"=============================================================="crlf)
   (printout t "Prvnim a druhym prvkem seznamu je:" ?neco1 " " ?neco2 crlf)
   (printout t "Ostatnimi prvky seznamu jsou tyto druhy:" ?ostatni_zelenina crlf) )
```

Grafické vysvětlení činnosti programu zelenina_2.clp



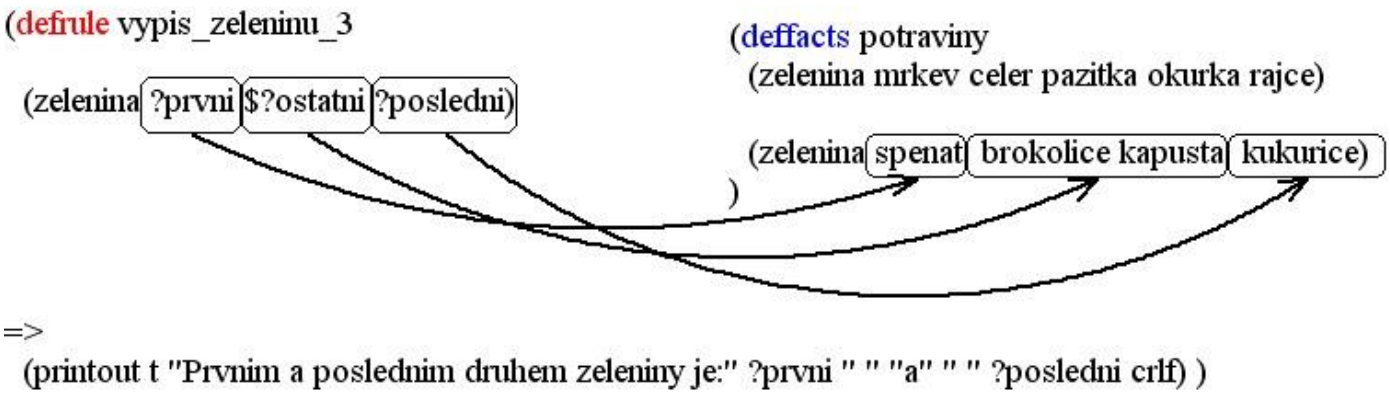
Všimněte si, že výsledkem výpisu v Clipsu je seznam hodnot opatřený závorkami - tak můžete poznat, že jste pracovali s nějakým seznamem. Jestliže pracujete s jednohodnotovou proměnnou - není opatřena závorkami.



Příklad 3.

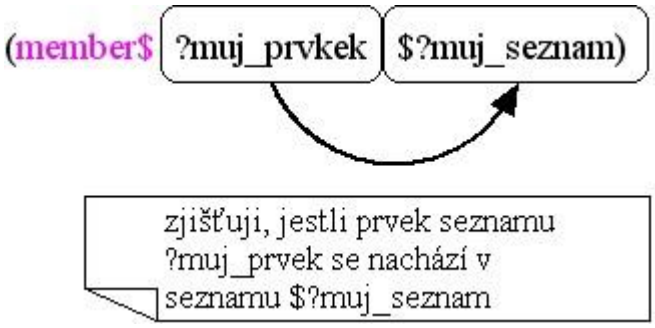
```
;************************************
;
;      ZELENINA - vypis druhu 3.
;            (MH 2005)
;
;************************************


(deffacts potraviny
  (zelenina mrkev celer pazitka okurka rajce)
  (zelenina spenat brokolice kapusta kukurice)
)


(defrule vypis_zeleninu_3
  (zelenina ?prvni $?ostatni ?posledni)
=>
  (printout t "Prvnim a poslednim druhem zeleniny je:" ?prvni " " "a" " "
?posledni crlf) )
```

Grafické vysvětlení činnosti programu zelenina_3.clp



Základní funkce pro práci se seznamy:

1. length$: tato funkce zjistí délku seznamu - tedy počet prvků, které se v seznamu nacházejí.
   syntaxe: (length$ $?muj_seznam) tj. název funkce a název vašeho seznamu, který u této funkce musí být vícehodnotovou proměnnou.
2. member$: tj. "je členem" tzn., že tato funkce ověří, jestli se určitý prvek nachází v určitém seznamu.
   syntaxe: (member$?muj_prvkek $?muj_seznam)



RETRACT

Dalším příkazem, který nemusí být spojen jen s vícehodnotovými proměnnými, ale je spojen s fakty jako takovými je příkaz retract. S tímto příkazem jsme se setkali již dříve (lekce 4.). Slouží nám k odstraňování faktů z báze faktů. Proč bychom vůbec chtěli nějaký fakt odstranit? Například když chceme zabránit opakovanému vykonávání nějakého pravidla, nebo již daný fakt v programu nepotřebujeme. Praxe Vám ukáže, kdy je to vhodné a užitečné.

Vysvětlující příklady příkazu retract:

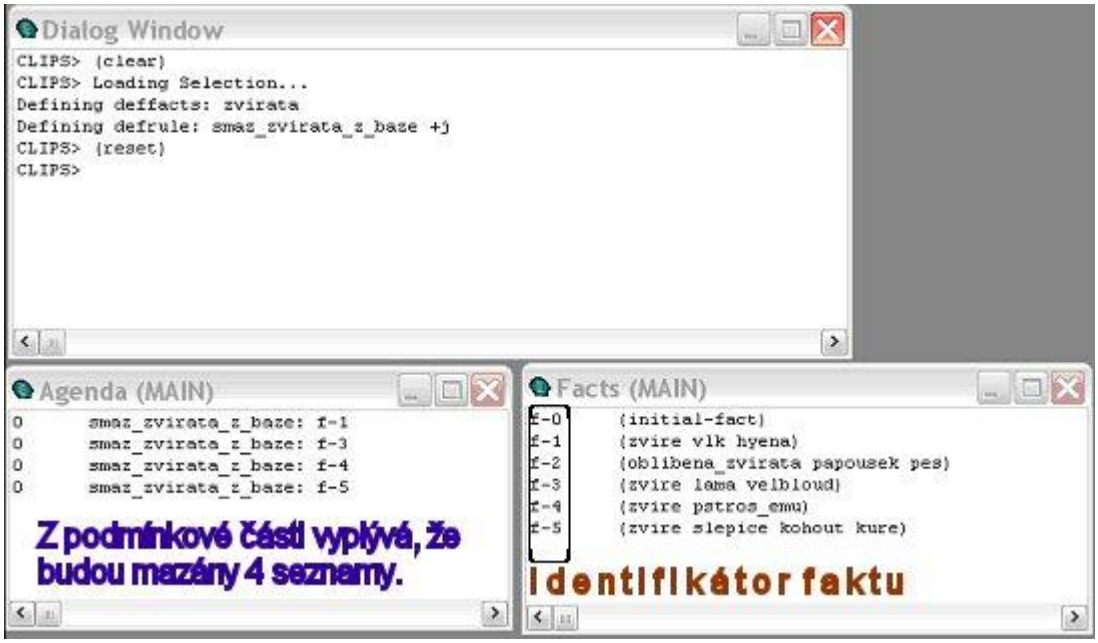
```
;*************************************
;
;      POUZITÍ PRIKAZU RETRACT (1)
```

```
;
;            (MH 2005)
;*********************************

(deffacts zvirata
  (zvire vlk hyena)
  (oblibena_zvirata papousek pes)
  (zvire lama velbloud)
  (zvire pstros_emu)
  (zvire slepice kohout kure) )

;-------------------------------------

(defrule smaz_zvirata_z_baze
  ?pryc_s_nimi<-(zvire $?nejaka_zvirata)
=>
  (retract ?pryc_s_nimi)
  (printout t "----------------------------------------"crlf)
  (printout t "Seznam zvirat uspesne smazan z baze faktu !" crlf)
  (printout t "----------------------------------------"crlf) )
```

Princip retractu:

Každý fakt v naší bázi je jednoznačně identifikován pomocí tzv. identifikátoru. Jaký identifikátor má který fakt můžete zjistit v okně Facts (tak jak je to znázorněno na obrázku níže).

V příkladu retract_1.clp můžeme vidět, že např. seznam s oblíbenými zvířaty má identifikátor: f-2 apod. Kdybychom chtěli smazat seznam s oblíbenými zvířaty v bázi faktů, pak by stačilo zapsat jen do usuzovací části jeden příkaz (retract 2), kde číslo 2 by značilo daný identifikátor, ke kterému tento seznam patří. Kde ale máme jistotu, že seznam s oblíbenými zvířaty bude mít opravdu číslo identifikátoru 2. Nevíme totiž předem, jakou hodnotu identifikátoru bude mít daný fakt. Proto tento postup není vhodný.



Lepším postupem pro smazání určitého faktu je ten následující. Vezměme si první podmínku z příkladu retract_1.clp.

Podmínková část:
```
?pryc_s_nimi<-(zvire $?nejaka_zvirata)
```

Touto metodou si vlastně zjišťujeme jaký identifikátor má fakt (zvire $?nejaka_zvirata.) Toto číslo (identifikátor) si uložíme do proměnné ?pryc_s_nimi. Šipka Vám může naznačit "co jde kam". Identifikátor faktu (zvire $?nejaka_zvirata) si uložíme do proměnné ?pryc_s_nimi.

Poznámka: značka šipky je tvořena znaménkem menší a pomlčkou.

Usuzovací (akční část):
```
(retract ?pryc_s_nimi)
```

Zde odstraňujeme proměnnou ?pryc_s_nimi, která obsahuje náš identifikátor faktu, který chceme smazat a tím odstraníme i fakt samotný.

Vysvětlující příklady

Příklady k procvičení

Vysvětlující příklad na příkaz retract

```
;**********************************
;
;     POUZITÍ PRIKAZU RETRACT (2)
;          CHUDAK LAMA
;
;            (MH 2005)
;**********************************

(deffacts zvirata
  (zvire vlk hyena)
  (oblibena_zvirata papousek pes)
  (zvire lama velbloud)
  (zvire pstros_emu lama)
  (zvire slepice kohout kure) )

;-----------------------------------

(defrule smaz_zvirata_z_baze
  ?pryc_s_lamou<-(zvire $?zacatek lama $?konec)
=>
  (retract ?pryc_s_lamou)
  (printout t "-------------------------------"crlf)
  (printout t "Seznam s Lamou byl odstranen !" crlf)
  (printout t "-------------------------------"crlf) )
```

Tento příklad je obdobný, jen s tím rozdílem, že rušíme ten seznam, který obsahuje konkrétní hodnotu. V našem příkladu odstraňujeme ty seznamy obsahující zvíře lama (předem se omlouvám všem, kteří mají rádi toto zvíře :-)).

`?pryc_s_lamou<-(zvire $?zacatek lama $?konec)` - zde říkáme: zapamatuj si identifikátor toho faktu, kde na jakémkoliv místě v seznamu je slovo lama.

Vysvětlující příklad na příkaz length

```
;*********************************
;
;     POUZITÍ PRIKAZU length
;
;          (MH 2005)
;*********************************

;Poznamka: seznamy se zviraty jsou pro
;jednoznacnost oznaceny velkymi pismeny A-D
;uvadim zde 2 pohodlne zpusoby zapisu prikazu length


(deffacts zvirata
  (zvire A vlk hyena)
  (oblibena_zvirata papousek pes)
  (zvire B lama velbloud)
  (zvire C pstros_emu)
  (zvire D slepice kohout kure) )

;---------------------------------

;Pravidlo pro poctu prvku se seznamem oblibenych zvirat

(defrule urci_delku_seznamu
  (oblibena_zvirata $?seznam)
=>
  (bind ?velikost (length$ $?seznam))
  ;ulozeni delky do promenne (zavorka s prikazem length nam vrati integerovou
hodnotu - cislo = delku seznamu

  (printout t "*****" crlf)
  (printout t "Delka seznamu s oblibenymi zviraty je:" ?velikost " " ?seznam crlf)
)
  ;vypsani obsahu promenne ?velikost = pocet prvku seznamu oblibena_zvirata


;Pravidlo pro vypsani delek ostatnich seznamu

(defrule delky_ostatnich_seznamu
  (zvire ?skupina $?seznam_zvirat)
  ;promenna ?skupina je z duvodu odliseni jednotlivych seznamu

=>
  (printout t "*****" crlf)
  (printout t "Seznam:" ?skupina " " "obsahuje:" (length$ ?seznam_zvirat) " "
"druhu/druhy/druh zvirat." " " ?seznam_zvirat crlf) )
  ;prikaz length je primo v radku s prikazem printout
  ;predstavte si, ze misto zavorky s prikazem length jiz mate konkretni delku
seznamu
```

Vysvětlující příklad na příkaz member

```
;*********************************************
;
;      POUZITÍ PRIKAZU member + ruzne pristupy
;
;                 (MH 2005)
;*********************************************

;Poznamka: budeme zjistovat, jestli existuje nejake zvire (fakt zvire = obecny
seznam zvirat), ktere
;se nachazi v seznamu oblibenych zvirat (fakt oblibene_zvire)
;pri vypisu - hvezdicky oznacuji zpusob hledani prvku v seznamu


(deffacts zvirata
  (zvire pes)
  (zvire kocka)
  (zvire kohout)
  (zvire orangutan)
  (zvire papousek)
  (oblibene_zvire mys simpanz pes loskutak papousek potkan) )

;-----------------------------------------------



;Toto pravidlo pouziva pristupu: provazani pomoci promennych, nikoliv prikazu
member

(defrule hledej_stejne_zpusob_1
  (zvire ?obecne_zvire)
  (oblibene_zvire $?zacatek ?obecne_zvire $?konec)
  ;at se obecne zvire nachazi kdekoliv v seznamu oblibenych zvirat, pak ho vypis

=>
  (printout t "*" crlf)
  (printout t "Z obecnych seznamu zvirat se v oblibenych zviratech nachazi:"
?obecne_zvire crlf) )




;Pravidlo s vyuzitim prikazu test a member

(defrule hledej_stejne_zpusob_2
  (zvire ?obecne_zvire)
  (oblibene_zvire $?oblibene_zvire)
  (test (member$ ?obecne_zvire $?oblibene_zvire))
  ;otestuj, jestli obecne zvire je prvkem seznamu s oblibenymi zviraty

=>
  (printout t "**" crlf)
  (printout t "Z obecnych seznamu zvirat se v oblibenych zviratech nachazi:"
?obecne_zvire crlf) )
```

```
;Slozitejsi podoba vyhledavani polozky v seznamu s prikazem member

(defrule hledej_stejne_zpusob_3
  (oblibene_zvire $?oblibene_zvire)
  (zvire ?obecne_zvire&:(member$ ?obecne_zvire $?oblibene_zvire))
  ;touto podminkou rikame: pro fakt s nazvem zvire má platit:
  ;ze toto obecne zvire ma byt prvkem seznamu oblibenych zvirat
  ;jestlize tato i ostatni podminky jsou splneny, pak tato obecna zvirata vypis

=>
  (printout t "***" crlf)
  (printout t "Z obecnych seznamu zvirat se v oblibenych zviratech nachazi:"
?obecne_zvire crlf) )
```

Příklady k procvičení

Úloha 1.

Definujte seznam několika čísel. Napište pravidla, pomocí kterých:

1. vypíšete celý seznam čísel
2. vypíšete na zvláštní řádky jednotlivá čísla
3. zpracujte seznam tak, že vypíšete vždy jeho první prvek, poté seznam smažete a nahradíte novým seznamem, kde první prvek již nebude obsažen; činnost pravidla skončí v momentě, kdy seznam bude prázdný

Úloha je rozdělena na dvě části:

```
;*********************************************
;
;           Uloha s ciselnym seznamem (1/2)
;
;                 (MH 2004)
;*********************************************
;Zadani:
;Definujte seznam nekolika cisel. Napiste pravidla, pomoci kterych:
;--- vypisete cely seznam cisel
;--- vypisete na zvlastni radky jednotliva cisla
;--- zpracujte seznam tak, ze vypisete vzdy jeho prvni prvek, pote seznam smazete
a nahradite novym seznamem, kde prvni prvek
;    jiz nebude obsazen; cinnost pravidla skonci v momente, kdy seznam bude
prazdny

;Zde je reseni pro prvni a druhy bod

;---------------------------------------------

(deffacts cisla
      (cislo 1 2 8 3 5)
)

(defrule vypis_vse
      (cislo $?seznam)
=>
      (printout t "Cely seznam je:" crlf)
      (printout t ?seznam crlf)
```

```
)


(defrule vypis_samostatne_fakty
      (cislo $?cislo ?nejake $?dalsi_cisla)
      ;da se rici, ze Clips disponuje nejakym ukazovatkem, ktere
      ;se posouva od prvku k prvku viz. obrazek na strankach
=>
      (printout t "Samostatne cislo:" crlf)
      (printout t ?nejake crlf)
)
```

```
;*********************************************
;
;           Uloha s ciselnym seznamem (3)
;
;                (MH 2004)
;*********************************************
;Zadani:
;Definujte seznam nekolika cisel. Napiste pravidla, pomoci kterych:
;--- vypisete cely seznam cisel
;--- vypisete na zvlastni radky jednotliva cisla
;--- zpracujte seznam tak, ze vypisete vzdy jeho prvni prvek, pote seznam smazete
a nahradite novym seznamem, kde prvni prvek
;    jiz nebude obsazen; cinnost pravidla skonci v momente, kdy seznam bude
prazdny

;Zde je reseni pro treti bod

;-----------------------------------------------

(deffacts cisla
      (cislo 1 2 8 3 5)
)

;Pravidlo pro vypisy prvku a vytvareni novych seznamu

(defrule zmena_seznamu
      ?all <- (cislo $?vse)
      ;abychom mohli neco po case vymazat z baze faktu (ze zadani je patrne, ze
      ;budeme muset zrusit cely seznam s cisly), musime
      ;si zapamatovat identifikator tohoto faktu - ten si ulozime
      ;do promenne all

      (cislo ?nejake $?ostatni)
      ;ale zaroven potrebujeme vzdy vypsat prvni prvek seznamu (znat jeho
hodnotu), proto si
      ;sestrojime tuto podminku s promennou nejake a zbytkem seznamu ostatni

=>
      (printout t ?nejake crlf)
      ;vypiseme si vzdy prvni prvek seznamu na monitor

      (retract ?all)
      ;z baze faktu odstranime cely seznam s cisly

      (assert (cislo $?ostatni))
      ;a do baze faktu vlozime novy seznam BEZ PRVNIHO PRVKU !!!
      ;vsimnete si, ze u prikazu assert neni promenna ?nejake !!!
)
```

```
;Toto pravidlo sice neni nutne konstruovat, ale je to vhodne
;Jestlize jiz seznam cisel nebude mit zadny prvek (=>jeho delka bude 0), pak
;uzivateli rekneme, ze je prazdny

(defrule seznam_je_prazdny
      (cislo $?neco)
      (test (=(length$ $?neco)0))
=>
      (printout t "Seznam je prazdny !" crlf)
)

;PO SKONCENI PROGRAMU SE MRKNETE DO OKNA FACTS, KDE JE SICE SEZNAM S NAZVEM CISLO,
ale
;tento seznam JE PRAZDNY a to jsme chteli :-)
```

Pokus o grafické vysvětlení části úlohy (cisla_ab.clp)

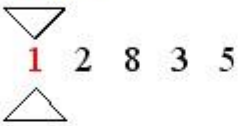
```
(defrule vypis_samostatne_fakty
    (cislo $?cislo ?nejake $?dalsi_cisla)
=>
    (printout t ?nejake crlf)
)
```

```
(deffacts cisla
    (cislo 1 2 8 3 5)
)
```

Přibližný postup Clipsu:

(cislo $?cislo ?nejake $?dalsi_cisla)

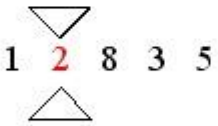
Krok 1.: ukazovátko



stavy proměnných

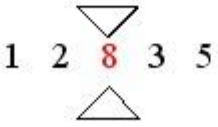
$?cislo stav: nic
?nejake stav: 1
$?dalsi_cisla stav: 2 8 3 5

Krok 2.:



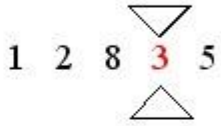
$?cislo stav: 1
?nejake stav: 2
$?dalsi_cisla stav: 8 3 5

Krok 3.:



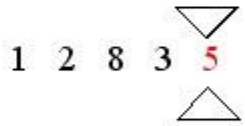
$?cislo stav: 1 2
?nejake stav: 8
$?dalsi_cisla stav: 3 5

Krok 4.:



$?cislo stav: 1 2 8
?nejake stav: 3
$?dalsi_cisla stav: 5

Krok 5.:



$?cislo stav: 1 2 8 3
?nejake stav: 5
$?dalsi_cisla stav: nic

Úloha 2.

Definujte dva seznamy a do každého z nich uložte několik znaků, oddělené mezerami. Napište pravidlo, které vyhledá znaky vyskytující se v obou seznamech (použijte provázání pomocí proměnné).

```
;*********************************************
;
;            Uloha s ciselnym seznamem (3)
;
;                  (MH 2004)
;*********************************************
;Zadani:
;Definujte seznam nekolika cisel. Napiste pravidla, pomoci kterych:
;--- vypisete cely seznam cisel
;--- vypisete na zvlastni radky jednotliva cisla
;--- zpracujte seznam tak, ze vypisete vzdy jeho prvni prvek, pote seznam smazete
a nahradite novym seznamem, kde prvni prvek
;    jiz nebude obsazen; cinnost pravidla skonci v momente, kdy seznam bude
prazdny

;Zde je reseni pro treti bod

;-----------------------------------------------

(deffacts cisla
      (cislo 1 2 8 3 5)
)

;Pravidlo pro vypisy prvku a vytvareni novych seznamu

(defrule zmena_seznamu
      ?all <- (cislo $?vse)
      ;abychom mohli neco po case vymazat z baze faktu (ze zadani je patrne, ze
      ;budeme muset zrusit cely seznam s cisly), musime
      ;si zapamatovat identifikator tohoto faktu - ten si ulozime
      ;do promenne all

      (cislo ?nejake $?ostatni)
      ;ale zaroven potrebujeme vzdy vypsat prvni prvek seznamu (znat jeho
hodnotu), proto si
      ;sestrojime tuto podminku s promennou nejake a zbytkem seznamu ostatni

=>
      (printout t ?nejake crlf)
      ;vypiseme si vzdy prvni prvek seznamu na monitor

      (retract ?all)
      ;z baze faktu odstranime cely seznam s cisly

      (assert (cislo $?ostatni))
      ;a do baze faktu vlozime novy seznam BEZ PRVNIHO PRVKU !!!
      ;vsimnete si, ze u prikazu assert neni promenna ?nejake !!!
)


;Toto pravidlo sice neni nutne konstruovat, ale je to vhodne
;Jestlize jiz seznam cisel nebude mit zadny prvek (=>jeho delka bude 0), pak
;uzivateli rekneme, ze je prazdny

(defrule seznam_je_prazdny
```

```
      (cislo $?neco)
      (test (=(length$ $?neco)0))
=>
      (printout t "Seznam je prazdny !" crlf)
)

;PO SKONCENI PROGRAMU SE MRKNETE DO OKNA FACTS, KDE JE SICE SEZNAM S NAZVEM CISLO,
ale
;tento seznam JE PRAZDNY a to jsme chteli :-)
```

Úloha 3.

Definujte seznam několika barev. Napište pravidlo, které uloží jednotlivé barvy ze seznamu do samostatných faktů. Tyto samostatné fakty budou mít tento tvar: (barvy ?hodnota).

```
;*********************************************
;
;                 Uloha s barvami
;
;               (MH 2004)
;*********************************************
;Zadani:
;Definujte seznam nekolika barev. Napiste pravidlo, ktere ulozi jednotlive barvy
ze seznamu do samostatnych faktu.
;Tyto samostatne fakty budou mit tento tvar: (barva ?hodnota).


(deffacts barvy
      (barvy cerna modra zelena cervena bila))

(defrule uloz_barvy_a_vypis
      (barvy $?nejaka_barva ?mojebarva $?ostatni_barvy)
=>
      (assert (barva ?mojebarva))
      ;nejdulezitejsi radek celeho programu, ktery to vse udela
      ;do baze ulozime fakt s nazvem barva a konkretni barvu
      (printout t " " crlf)
      (printout t ?mojebarva crlf)
)

(defrule cely_seznam
  (barvy $?cely_seznam)
=>
  (printout t "-------------------------------------------------------------"
crlf)
  (printout t "Cely seznam barev je tento:" ?cely_seznam crlf)
  (printout t "-------------------------------------------------------------"
crlf)
)
```

Úloha 4.

Definujte několik seznamů. Všechny seznamy vypište a současně vypište jejich délky. Zkuste napsat pravidla, jimiž vyhledáte nejdelší seznam a nejkratší seznam.

Řešení selským rozumem:

```
;*********************************************
;
;                Seznamy Max a Min
;
;                 (MH 2004)
;*********************************************

;Zadani:Definujte nekolik seznamu. Vsechny seznamy vypiste a soucasne vypiste
jejich delky.
;Pokuste se napsat pravidla, jimiz vyhledate nejdelsi seznam a nejkratsi seznam

;Poznamky: velka pismena abecedy oznacuji nazvy seznamu
;Kroky:     1. zjistime delku pro kazdy seznam
;      2. najdeme z techto hodnot maximalni hodnotu
;      3. maximum vypiseme
;      4. najdeme z techto hodnot minimalni hodnotu
;      5. minimum vypiseme

;-----------------------------------------------


(deffacts seznamy
      (seznam A a b c d e)
      (seznam B a b)
      (seznam C a b c))



;Pravidlo pro zjisteni delky KAZDEHO seznamu

(defrule zjisti_delky_seznamu
      (seznam ?nejaky $?obsah)
=>
      (bind ?delka (length$ $?obsah))
      (assert (delka_seznamu ?nejaky ?delka)))
      ;delku kazdeho seznamu ulozime do baze faktu prikazem assert, abychom
      ;jednotlive delky seznamu mohli jednoduse porovnavat a zjistovat max. ci
min.


;Pravidlo pro nalezeni MAXIMALNI hodnoty seznamu

(defrule hledej_max
      (delka_seznamu A ?neco)
      (delka_seznamu B ?znak)
      (delka_seznamu C ?dalsi)
=>
      (bind ?max (max ?neco ?znak ?dalsi))
      ;max je funkci, ktera nam z hodnot najde maximum

      (assert (maximum je ?max)))
      ;toto maximum take vlozime do baze faktu, i kdyz bychom mohli delku seznamu
rovnou vypsat
      ;to ano, ale jeste chceme znat nazev seznamu, ktery maximum obsahuje


;Toto pravidlo nam umozni vypsat nazev seznamu, ktery
;ma nejvetsi pocet prvku

(defrule vypis_max_seznam
      (maximum je ?max)
```

```
      ;tato podminka nam zarucuje, ze se toto pravidlo vykona az po tom, co bude
maximum nalezeno
      ;tedy informace o maximu bude vlozena do baze faktu (to provede pravidlo
hledej_max prikazem assert)
      ;tedy touto podminkou rikame, jestlize bylo jiz maximum nalezeno a zaroven
...

      (delka_seznamu ?nejaky ?delka)
      ;... zjistena jeho delka (to provedlo pravidlo zjisti_delky_seznamu)

      (test (= ?max ?delka))
      ;Proc je zde tato podminka? Vime uz jaka je maximalni hodnota delky seznamu,
ale chceme i vedet, ktery seznam toto max. obsahuje
      ;Tedy porovname zjistenou max. hodnotu seznamu s delkou seznamu, ktera je
zaznamenana ve faktu delka_seznamu a vypisu
      ;priradime obsah promenne ?nejaky - tj. jmeno seznamu, ktery maximum
obsahuje
=>
      (printout t "Nejdelsi seznam je:" ?nejaky " " "s poctem prvku:" ?delka crlf)
      (printout t "-----------------------------------------------------" crlf)
)



;STEJNE POSTUPUJEME I PRI HLEDANI MINIMA


;Pravidlo pro nalezeni MINIMALNI hodnoty seznamu

(defrule hledej_min
      (delka_seznamu A ?neco)
      (delka_seznamu B ?znak)
      (delka_seznamu C ?dalsi)
=>
      (bind ?min (min ?neco ?znak ?dalsi))
      (assert (minimum je ?min)))


;Toto pravidlo nam umozni vypsat nazev seznamu, ktery
;ma nejmensi pocet prvku

(defrule vypis_min_seznam
      (minimum je ?min)
      (delka_seznamu ?nejaky ?delka)
      (test (= ?min ?delka))
=>
      (printout t "Nejkratsi seznam je:" ?nejaky " " "s poctem prvku:" ?delka
crlf)
      (printout t "-----------------------------------------------------" crlf)
)
```

Řešení bez selského rozumu:

```
;*************************************************
;
;                Seznamy Max a Min (2. moznost)
```

```
;
;                 (MH 2004)
;*************************************************

;Zadani:Definujte nekolik seznamu. Vsechny seznamy vypiste a soucasne vypiste
jejich delky.
;Pokuste se napsat pravidla, jimiz vyhledate nejdelsi seznam a nejkratsi seznam

;Poznamky: Tento zpusob reseni problemu je myslim vice elegantnejsi, kdezto
;ta sama uloha resena v prvnim pripade (soubor: max_a_min_1.clp) se vice podoba
uvazovani pomoci
;selskeho rozumu :-)


(deffacts seznamy
      (seznam A 5 6 4 -1)
      (seznam B 8 1 5 4 9 -2))



;Pravidlo nejprve JEN vypise delky jednotlivych seznamu
;na rozdil od prvni moznosti reseni problemu (soubor: max_a_min_1.clp)
;zde neukladame do baze faktu info. o delkach seznamu
;opet toto pravidlo mam odpoznamkovane - kvuli jeste docela neznamemu
;prikazu declare

;(defrule vypis_seznam+delku
;     (declare(salience 10))
;     (seznam ?x $?seznam)
;=>
;     (printout t "Seznam" " " ?x " " "ma delku:"(length$ ?seznam) crlf))



;Pravidlo vyhleda nejdelsi seznam a  JEN ho vypise
;Mame zde jen dva seznamy, mohly by být tri a vice - opet bychom pouzili fci max
ci min

(defrule hledej_nejdelsi_seznam
      (seznam ?x $?seznam1)
      (seznam ?y $?seznam2)
      (test(>(length$ $?seznam1) (length$ $?seznam2)))
      ;tato podminka vypada velmi slozite, ale nelekejte se ji
      ;jednoduse zjistujeme jestli delka prvniho seznamu je vetsi nez delka
druheho seznamu
      ;vyuzivame zde prefixovy zapis
=>
      (printout t "Nejdelsi seznam je:" ?x crlf))


;Pravidlo vyhleda nejkratsi seznam a JEN ho vypise

(defrule hledej_nejkratsi_seznam
      (seznam ?x $?seznam1)
      (seznam ?y $?seznam2)
      (test(>(length$ $?seznam2) (length$ $?seznam1)))
=>
      (printout t "Nejkratsi seznam je:" ?x crlf))
```

Úloha 5.

Napište pravidlo, jímž číselný seznam setřídíte podle velikosti (od nejmenšího po největší).

```
;*********************************************
;
;             Trideni seznamu vzestupne
;
;               (MH 2004)
;*********************************************
;Zadani: Napiste pravidlo, jimz ciselny seznam setridite podle velikosti
;od nejmensiho po nejvetsi
;Poznamka: vysledek programu zatim uvidite jen v okne Facts
;je zde zminen novy prikaz operujici s prioritou pravidla (declare (salience
...)), ktery
;bude probiran v lekci c. 6


(deffacts cisla
      (cisla 3 7 9 -10 0 4) )


(defrule setrid_seznam
?pryc <-(cisla $?zacatek ?cislo1 $?stred ?cislo2 $?konec)
      ;vzdy budeme porovnavat dve cisla

      (test( > ?cislo1 ?cislo2) )
      ;budeme testovat jestli pozici techto dvou cisel je spravna
      ;tedy jestlize cislo1 je vetsi nez cislo2 pak ...
=>
      (retract ?pryc)
      ;nebudeme chtit zachovat tento spatne serazeny seznam - z baze faktu ho
zrusime

      (assert(cisla $?zacatek ?cislo2 $?stred ?cislo1 $?konec)) )
      ;a do baze vlozime ten spravne serazeny seznam (ale cisla se budou serazovat
postupne -
      ;toto pravidlo se provede tolikrat, kolikrat bude potreba nespravne serazena
cisla prestehovat
      ;na spravnou pozici
      ;v posledni podmince s assertem si vsimnete opacne pozice promennych ?cislo2
a ?cislo1 - to je to co jsme chteli
      ;jestlize nemaji spravnou pozici pak je prohodime


;tohoto pravidla si zatim vsimat nemusite, protoze je zde uveden novy prikaz,
ktery jsme jeste neprobirali
;jedna se o prikaz nastavujici PRIORITU PRAVIDLA (da se tedy ovlivnit, kdy bude
ktere pravidlo aplikovano)
;toto pravidlo je zatim v rezimu poznamek - muzete si ho zatim jen prohlednout -
slouzi pro vypis serazeneho seznamu

;(defrule vypis_setrideny_seznam
;     (declare(salience -10) )
;     (cisla $?seznam)
;=>
;     (printout t "Setrideny seznam:" ?seznam) )
```

Na tomto programu lze dobře poznat to, že při programování v Clipsu (a asi nejen v něm) můžete používat svůj přirozený jazyk a na základě toho formulovat pravidla a fakta.

Co tím myslím?

Máte před sebou určitý problém (např. seřadit čísla vzestupně). Přesně tak jak si slovně vyjadřujete svůj postup pro vyřešení problému - tak tento slovní postup můžete dobře přetransformovat např. do nějaké podmínky. Tedy jestliže máte dvě čísla : 10 5, pak vidíte, že nejsou seřazena dle velikosti (10 má být až po 5). Zní to naprosto jasně. Stačí tyto čísla vyměnit - na pozici desítky dát číslo pět. Ono to sice jasně zní, ale napsat to v programovacím jazyce je o trochu složitější, ale vychází to z přirozeného jazyka.

LEKCE 6.

Náplň lekce:

- Práce s pomocnými fakty
- Priorita pravidla (+ porovnání Pascalu a Clipsu)
- Výzkum podmínkové části pravidla
- Souhrnný příklad

Práce s pomocnými fakty

Pomocné fakty jsou takové fakty, kterými můžete určitým způsobem řídit běh programu (kdy - za jakých podmínek se co provede). Například máme fakta - seznamy, obsahující informace o ovoci. Už víme, jak zajistit výpis daných seznamů. Budeme ale požadovat, aby byl výpis proveden jen v případě, že si tak uživatel přeje. Tedy jestliže sám zadá ano nebo ne.

Tento problém je řešen s poznámkami v následujícím kódu:

```
;*****************************************;
;                                         ;
;         OVOCE s POMOCNÝMI FAKTY         ;
;               (MH 2005)                 ;
;                                         ;
;*****************************************;
;Cil: chceme aby uzivatel byl dotazan, jestli chce vypsat zaznamy z databaze
zahrnujici druhy ovoce
;Kazdou moznou reakci (ano, ne, zadani jine mozne odpovedi) osetrete


(deffacts info
  (ovoce jablko)
  (ovoce hruska)
  (ovoce pomeranc)
  (ovoce kiwi)
  (ovoce mandarinka)
)


;Toto pravidlo bude vykonano jako prvni;ptame se zde
```

```
;uzivatele jestli chce vypsat informace z databaze

(defrule ptame_se_uzivatele
=>
  (printout t "Prejete si vypsat vsechny druhy ovoce, ktere mam v databazi ?"
crlf)
  (bind ?odpoved_uzivatele (read))
  ;odpoved uzivatele si ulozime do promenne a nasledne i do baze faktu, abychom
  ;s informaci o odpovedi mohli dale pracovat

  (assert (volba_uzivatele ?odpoved_uzivatele))
)


(defrule vypisuj_ano
  (volba_uzivatele ano)
  ;jestlize uzivatel zada o vypsani informaci (tedy do baze pravidlem
ptame_se_uzivatele
  ;bylo vlozeno ano a zaroven ...

  (ovoce ?nejake)
  ;v bazi faktu vubec nejake ovoce existuje pak tyto druhy ovoce vypis na monitor

=>
  (printout t "************" crlf)
  (printout t ?nejake crlf)
)

;Poznamka: Proc nam v podminkove casti nestaci jen prvni podminka ?
;Jednoduse proto, ze v prip. ze uzivatel napsal volbu ano, pak asi bude chtit
vypsat ovoce
;a my potrebujeme tyto druhy v podminkove casti zachytit, abychom mohli v
usuzovaci casti
;realizovat jejich vypis


(defrule vypisuj_ne
  (volba_uzivatele ne)
  ;jestlize uzivatel zvolil ne, pak se jen vypise, ze je program ukoncen - zadna
ovoce nechceme vypisovat
  ;tak proto take nepotrebujeme v podminkove casti pravidla podminku: (ovoce
?nejake)

=>
  (printout t "......................" crlf)
  (printout t "  Konec programu !!!" crlf)
)


(defrule spatna_volba
  ?pryc<-(volba_uzivatele ~ano&~ne)
  ;muzete si rici, ze tento zapis znamena: volba_uzivatele neni ano a neni ne
  ;jestlize uzivatel zadal neco jineho nez ano nebo ne do volby, pak
  ;asi nebudeme chtit mit v bazi faktu tento chybny fakt a jednoduse ho
  ;odstranime z baze prikazem retract
  ;pak jeste do baze dame informaci o tom, ze uzivatel zadal neznamou volbu

=>
  (retract ?pryc)
  (printout t "??????????????????????????????" crlf)
  (printout t "Byla zadana neznama volba !" crlf)
```

```
   (assert (volba neznama))
)
```

Priorita pravidla

Tím, že pravidlu nastavíte prioritu docílíte toho, že pravidlo může být vykonáno dříve (priorita vyšší - definujete kladné celé číslo) nebo později (priorita nižší - definujete záporné celé číslo) než pravidla ostatní.

Klady:

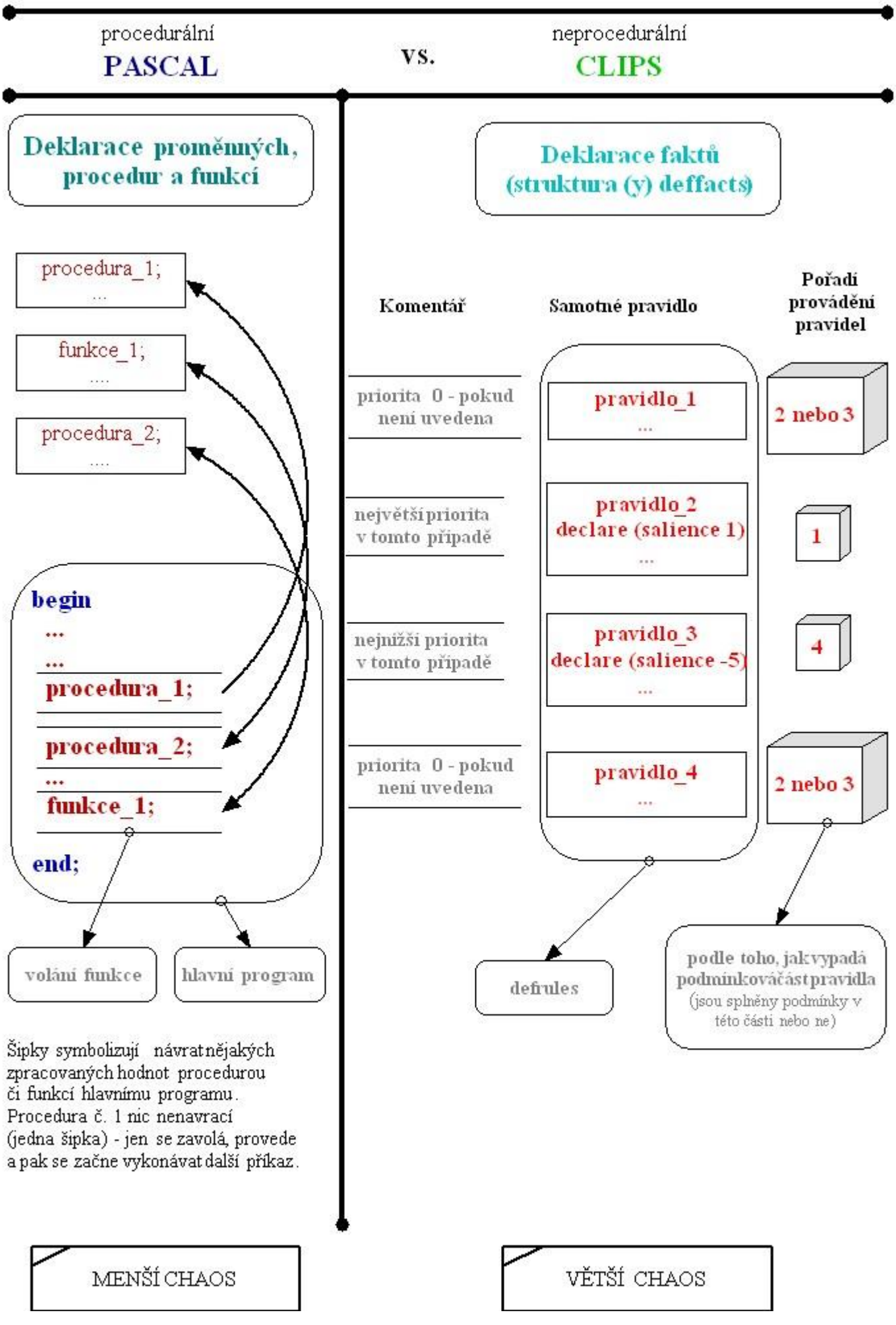
Je to dobrý způsob jak např. docílit toho, aby se jedno pravidlo vykonalo dříve a to druhé počkalo až se první pravidlo vykoná (nebo naopak). Řídíte si běh programu (resp. pořadí vykonávání pravidel - pořadí pravidel jak se budou řadit v Agendě Clipsu) tak jak chcete vy (tedy ne vždy se to podaří na první pokus jak chcete :-().

Zápory:

Když porovnám programování např. v procedurálním jazyku Pascal a programování v Clipsu, tak v Pascalu jsem měla větší přehled o tom, co kdy bude vykonáno. Zkrátka jste měli hlavní program ohraničený beginem a endem. Uvnitř těchto hranic byly nějaké příkazy. Dále také procedury či funkce, které jste volali a věděli jste docela rychle kam se volání procedury či funkce odkazuje (věděli jste kam se dívat - jak to celé probíhá). Jednoduše to bylo v posloupnostech příkazů a ne tak nějak příliš chaoticky napřeskáčku.

V Clipsu je situace jiná. Nemáte zde žádný hlavní program (jakési jádro toho všeho) ze kterého voláte nějaké funkce či procedury. Dostáváte se jaksi mimo hranice konstrukce beginu a endu. Máte zde sadu pravidel, které mají za úkol něco vykonat. Hůře se odhaluje co bude vykonáno dříve a co později. Musíte projít všechna pravidla a zkoumat. Je pravdou, že si pravidla můžete seřadit za sebou (tam kde píšete zdrojový kód programu) podle toho v jakém pořadí budou vykonávána (pořadí nemá vliv na fungování programu - inferenční mechanismus se řídí především tím co je definováno v podmínkové části pravidla tzn. i danou prioritou pokud je v pravidle stanovena).

Pokusila jsem se vytvořit obrázek, který toto popisuje.


procedurální
PASCAL
vs.
neprocedurální
CLIPS
Deklarace proměnných, procedur a funkcí
Deklarace faktů (struktura (y) deffacts)
procedura_1;
...
funkce_1;
....
procedura_2;
....
begin
...
...
procedura_1;
procedura_2;
...
funkce_1;
end;
volání funkce
hlavní program
Šipky symbolizují návrat nějakých zpracovaných hodnot procedurou či funkcí hlavnímu programu. Procedura č. 1 nic nenavrací (jedna šipka) - jen se zavolá, provede a pak se začne vykonávat další příkaz.
Komentář
Samotné pravidlo
Pořadí provádění pravidel
priorita 0 - pokud není uvedena
pravidlo_1
...
2 nebo 3
největší priorita v tomto případě
pravidlo_2
declare (salience 1)
...
1
nejnižší priorita v tomto případě
pravidlo_3
declare (salience -5)
...
4
priorita 0 - pokud není uvedena
pravidlo_4
...
2 nebo 3
defrules
podle toho, jak vypadá podmínková část pravidla (jsou splněny podmínky v této části nebo ne)
MENŠÍ CHAOS
VĚTŠÍ CHAOS

Poznámka: Jednotlivé struktury nejsou přesně syntakticky správné, ale doufám že problém s pořadím provádění pravidel vystihuje. Je to jen má představa, jak to asi funguje a jaké jsou zde rozdíly např. oproti Pascalu. Tento obrázek je i ilustrací toho v jakém pořadí, při daných prioritách v pravidlech, se jednotlivá pravidla vykonají.

Detailní pohled na priority

Deklarace priority pravidla:

Syntaxe: (declare (salience celé číslo kladné nebo záporné))

například:

```
(declare (salience 5))

(declare (salience -50))
```

Kladným číslem definujeme vysokou prioritu a záporným nízkou tzn. že pravidlo s vyšší prioritou než mají ostatní pravidla bude vykonáno dříve. Záporným číslem definujeme nízkou prioritu tzn. že pravidlo s nižší prioritou, než mají ostatní pravidla, bude vykonáno později - až po vykonání těch s vyšší prioritou.

Důležité upozornění č. 1:

Priorita se definuje v podmínkové části pravidla jako první podmínka (na prvním místě) (Clips si to musí v hlavince srovnat jaké pravidlo může odložit a které spěchá a jakoby si je seřadit.).

Důležité upozornění č. 2:

Pokud neuvedete v podmínkové části pravidla žádnou prioritu - pak má pravidlo prioritu nulovou. Jako byste definovali (declare (salience 0))

Důležité upozornění č. 3:

Rozsah v jakém můžete definovat prioritu je <-10.000 ; + 10.000>.

Výzkum podmínkové části pravidla

Jaké druhy podmínek zde mohou být?
Jak jsou tyto podmínky zapsány?

Chceme definovat:

1. podmínku, kde proměnná bude nabývat jakékoliv hodnoty:

(ovoce ?nejake) - pro jednu hodnotu

(ovoce $?nejaka) - pro více hodnot = seznam

2. podmínku, kde proměnná nebude nabývat jedné konkrétní hodnoty
(ovoce ?nejake~hruska) - proměnná ?nejake není rovna hrusce

3. podmínku, kde proměnná nebude nabývat více konkrétních hodnot

(ovoce ?nejake~hruska&~jablicko&~kiwi&~mango) - proměnná ?nejake není rovna hrusce a zároveň není rovna jablicku atd.

4. podmínku s významem nebo
(ovoce cervene|zelene|zlute)

5. podmínku s hodnotou, která souvisí s hodnotami jiných proměnných
(ovoce ?nejake & :(eq ?nejake ?jine)) - říkáme, že pro proměnnou ?nejake platí, že jeho hodnota je rovna hodnotě v proměnné ?jiné.

Pozn. Předpokládám práci s řetězci - proto definuji eq nikoliv znaménko rovnosti = (to platí pro číselné proměnné). Proměnná ?jiné byla dříve definována.

Souhrnný příklad na procvičení:

Zadání:

1. Mějme zadány druhy zvířat a kontinent na kterém žijí. Ty budou součástí naší budoucí databáze:

```
(deffacts databaze
  (zvirata afrika slon zebra zirafa opice antilopa pes)
  (zvirata asie tygr slon opice panda pes)
  (zvirata evropa pes zajic jelen kocka medved rys)
)
```

2. Pokuste se vytvořit pravidla, která budou řešit následující:

1. vypsat celou databázi zvířat i s kontinentem (nejprve název kontinentu a k němu zvířata, která ho mohou obývat)
2. načíst název kontinentu a zjistit, jaká zvířata jej obývají
3. načíst druh zvířete a zjistit, kde žije
4. vložit do databáze nové zvíře (na konec příslušného seznamu)
5. vložit do databáze celý nový záznam (např. pro Austrálii)
6. zjistit, na kterém kontinentě žije kolik druhů zvířat
7. zjistit, na kterém kontinentě žije největší počet zvířecích druhů
8. ukončit program

Celý příklad bude zastřešen menu, ze kterého si bude uživatel vybírat co chce s databází provádět.

Tuto úlohu budeme řešit postupně. Nejprve začneme prvníma dvěma body A+B s ukončením programu. Postupně budeme přidávat další pravidla.

```
;*****************************************;
;                                         ;
;            DATABAZE ZVIRAT              ;
;               (MH 2004)                 ;
;                                         ;
;*****************************************;
;Zadani:
;A.   vypsat celou databazi zvírat i s kontinentem (nejprve nazev kontinentu a k
nemu zvírata, ktera ho mohou obyvat)
```

```
;B.   nacist nazev kontinentu a zjistit, jaka zvírata jej obyvaji
;H.   Ukoncit program

;Poznamky: ke kazdemu pravidlu je mimo poznamek a oznaceni ulohy i cislo
;cislo oznacuje poradi, kdy bude pravidlo vykonano (pokud mame menu, tak to hlavne
zavisi
;na poradi pozadavku uzivatele
;v tomto pripade bude objasneno poradi vykonavani pravidel jen pro jeden ukol
(treba uloha A), ke kteremu
;se vaze nekolik pravidel
;A2. zn. ze toto pravidlo je soucasti ulohy A a bude vykonano jako druhe v poradi
;program neresi zadani chybnych faktu pr. (chci_kontinent asoa)


(deffacts databaze
      (zvirata afrika slon zebra zirafa opice antilopa pes)
      (zvirata asie tygr slon opice panda pes)
      (zvirata evropa pes zajic jelen kocka medved rys)

      (nactena_volba)) ;tento fakt by mohl byt soucasti nove struktury deffacts,
ale i tato varianta je OK


;Pravidlo, ktere nam zobrazi volby pro operace s databazi
;je zde nulova priorita

;1.
(defrule zobraz_menu
?x <- (nactena_volba)
;tato podminka je zde velmi dulezita !!!
;chceme zobrazit menu (mit zobrazene ruzne volby) po nejake akci
;napr. po spusteni programu, po zobrazeni cele databaze atd.
;musime ale zabezpecit, aby se nam menu nezobrazovalo v dobe kdy nechceme napr.
;aby se nam nezobrazovalo jen menu a my bychom si po zobrazeni menu nemohli obsah
databaze zobrazit
;podminkova cast: za podminky, ze v bazi faktu je fakt nactena_volba, pak mi
zobraz menu
;aby se pri druhem kroku menu nezobrazovalo, tak tento fakt z baze retractem
vymazeme
;tim jiz nebude splnena podminkova cast tohoto pravidla (pravidlo se neprovede)
=>
      (retract ?x)
      (printout t "-----------------<NABIDKA>------------------" crlf)
      (printout t "Zobrazeni cele databaze ... a" crlf)
      (printout t "Zadani kontinentu a zobrazeni zvirat ... b" crlf)
      (printout t "Konec programu ... k" crlf)
      (printout t "==========================================" crlf)

      (printout t "Zadejte svoji volbu:")
      ;zadame uzivatele aby zadal jednu z voleb a nebo b ...
      (bind ?volba (read))
      (assert (moje_volba ?volba)))
      ;tuto volbu ukladame do baze faktu, abychom tuto informaci mohli uzit dale
      ;v jinem pravidle


;A2.
(defrule vypis_vsechny_seznamy
      (declare(salience 10))
      ;urcite se ptate proc se nejprve nevykona toto pravidlo, kdyz je zde uvedena
      ;priorita 10 (v pravidle zobraz_menu je priorita 0 - vlastne vubec neni
```

```
priorita deklarovana)
      ;je to proto, ze aby mohlo byt pravidlo vykonano museji byt splneny vsechny
podminky v podminkove
      ;casti pravidla a pro vykonani tohoto pravidla je treba faktu moje_volba
(druha podminka tohoto pravidla)
      ;tento fakt moje_volba bude v bazi az po vykonani pravidla zobraz_menu, kde
ho vkladame assertem do baze faktu
      ;a tedy pak muze byt toto pravidlo vykonano (v bazi jiz je moje_volba
(program tedy uz vi, co uzivatel zada))

      (moje_volba a)
      (zvirata $?vse)
=>
      (printout t "Cela databaze obsahuje tyto udaje:" crlf)
      (printout t ?vse crlf))


;A3.
(defrule zrus_volbu_a
?x <-   (moje_volba a)
=>
      (retract ?x)
      (assert(nactena_volba)))

;Toto posledni pravidlo ulohy A je take velmi dulezite a asi nebude jednoduche ho
objasnit ale pokusim se
;je jasne ze po vypisu databaze (volba a) budete chtit, aby se Vam menu opet
zobrazilo a uzivatel zvolil z menu neco jineho
;jinymi slovy: chcete, aby se nam pravidlo vypis_vsechny_seznamy nevykonavalo
neustale
;tedy musime docilit toho, aby podminkova cast pravidla vypis_vsechny_seznamy
nebyla splnena (pravidlo se nevykonalo)
;toho docilime tak, ze fakt (moje_volba a) z baze faktu odstranime (vlastne
uzivatel jiz nebude chtit treba zobrazit
;znovu celou databazi (to je volba a)), to nevime predem co bude chtit udelat a
tak zrusime jeho volbu a ocekavame volbu novou
;jakobychom uklizeli neporadek - to stare co ted nepotrebujeme, abychom to mohli
nekdy pouzit znovu
;tim take nebude splnena podminkova cast pravidla vypis_vsechny_seznamy, ktera
predpoklada existenci faktu (moje_volba a)
;fakt (moje_volba a) je zrusen, ale to nestaci, preci chceme hned po vypisu
zobrazit menu
;a jaky fakt k tomu potrebujeme ???
;potrebujeme fakt (nactena_volba), ktery je vyzadovan pravidlem zobraz_menu, aby
se pravidlo mohlo provest


;B1.
(defrule zadej_kontinent
      (moje_volba b)
      ;jestlize uzivatel zadal volbu b pak (pravidlo bude vykonano jako prvni) po
nem chceme, aby zadal kontinent, ktery
      ;bude nasledne ulozen do baze faktu
=>
      (printout t "Zadejte nazev kontinentu:" crlf)
      (bind ?nejaky (read))
      (assert (chci_kontinent ?nejaky)))


;B2.
(defrule vypis_zvirata_pro_muj_kontinent
```

```
         (chci_kontinent ?k)
         ;kontinent zadan

       (zvirata ?k $?zvirata)
       ;predpokladame predem, ze program ma nejaka fakta o zviratech predem

=>
       (printout t "Nalezena zvirata:" ?zvirata crlf))


;B3.
(defrule zrus_volbu_b+kontinent
       (declare(salience -1))
       ;je zde i jina moznost zruseni jiz nepouzitelnych faktu nez u prikladu A
       ;potom co budou vypsana zvirata pro dany kontinent tak az pak
       ;zrusime vse potrebne - volbu a kontinent (proto je zde priorita -1

?x <- (moje_volba b)
?y <- (chci_kontinent ?nejaky)

=>
       (retract ?x ?y)
       (assert (nactena_volba))) ;opet chceme nacist menu


;1.
(defrule ukonci_program
       (moje_volba k)
       ;jestlize uzivatel zada volbu k pak vypis ...
=>
       (printout t "Konec programu !" crlf))
       ;neni potreba odstranovat fakt moje_volba
       ;program stejne skonci a po jeho opetovnem spusteni
       ;muzete pracovat s cistym platnem (baze nebude obsahovat zadny fakt
moje_volba)
```

```
;****************************************;
;                                ;
;           DATABAZE ZVIRAT              ;
;              (MH 2004)              ;
;                                  ;
;****************************************;
;Zadani:
;A.   vypsat celou databazi zvírat i s kontinentem (nejprve nazev kontinentu a k
nemu zvírata, ktera ho mohou obyvat)
;B.   nacist nazev kontinentu a zjistit, jaka zvírata jej obyvaji
;H.   Ukoncit program
;C.   nacist druh zvírete a zjistit, kde zije

;Poznamky: ke kazdemu pravidlu je mimo poznamek a oznaceni ulohy i cislo
;cislo oznacuje poradi, kdy bude pravidlo vykonano (pokud mame menu, tak to hlavne
zavisi
;na poradi pozadavku uzivatele
;v tomto pripade bude objasneno poradi vykonavani pravidel jen pro jeden ukol
(treba uloha A), ke kteremu
;se vaze nekolik pravidel
;A2. zn. ze toto pravidlo je soucasti ulohy A a bude vykonano jako druhe v poradi
;program neresi zadani chybnych faktu pr. (chci_kontinent asoa)
```

```
(deffacts databaze
      (zvirata afrika slon zebra zirafa opice antilopa pes)
      (zvirata asie tygr slon opice panda pes)
      (zvirata evropa pes zajic jelen kocka medved rys)

      (nactena_volba)) ;tento fakt by mohl byt soucasti nove struktury deffacts,
ale i tato varianta je OK


;Pravidlo, ktere nam zobrazi volby pro operace s databazi
;je zde nulova priorita

;1.
(defrule zobraz_menu
?x <- (nactena_volba)
;tato podminka je zde velmi dulezita !!!
;chceme zobrazit menu (mit zobrazene ruzne volby) po nejake akci
;napr. po spusteni programu, po zobrazeni cele databaze atd.
;musime ale zabezpecit, aby se nam menu nezobrazovalo v dobe kdy nechceme napr.
;aby se nam nezobrazovalo jen menu a my bychom si po zobrazeni menu nemohli obsah
databaze zobrazit
;podminkova cast: za podminky, ze v bazi faktu je fakt nactena_volba, pak mi
zobraz menu
;aby se pri druhem kroku menu nezobrazovalo, tak tento fakt z baze retractem
vymazeme
;tim jiz nebude splnena podminkova cast tohoto pravidla (pravidlo se neprovede)
=>
      (retract ?x)
      (printout t "-----------------<NABIDKA>-----------------" crlf)
      (printout t "Zobrazeni cele databaze ... a" crlf)
      (printout t "Zadani kontinentu a zobrazeni zvirat ... b" crlf)
      (printout t "Zjisti o zvireti, kde zije ... c" crlf)
      (printout t "Konec programu ... k" crlf)
      (printout t "=============================================" crlf)

      (printout t "Zadejte svoji volbu:")
      ;zadame uzivatele aby zadal jednu z voleb a nebo b ...
      (bind ?volba (read))
      (assert (moje_volba ?volba)))
      ;tuto volbu ukladame do baze faktu, abychom tuto informaci mohli uzit dale
      ;v jinem pravidle



;A2.
(defrule vypis_vsechny_seznamy
      (declare(salience 10))
      ;urcite se ptate proc se nejprve nevykona toto pravidlo, kdyz je zde uvedena
      ;priorita 10 (v pravidle zobraz_menu je priorita 0 - vlastne vubec neni
priorita deklarovana)
      ;je to proto, ze aby mohlo byt pravidlo vykonano museji byt splneny vsechny
podminky v podminkove
      ;casti pravidla a pro vykonani tohoto pravidla je treba faktu moje_volba
(druha podminka tohoto pravidla)
      ;tento fakt moje_volba bude v bazi az po vykonani pravidla zobraz_menu, kde
ho vkladame assertem do baze faktu
      ;a tedy pak muze byt toto pravidlo vykonano (v bazi jiz je moje_volba
(program tedy uz vi, co uzivatel zada))

      (moje_volba a)
      (zvirata $?vse)
```

```
=>
      (printout t "Cela databaze obsahuje tyto udaje:" crlf)
      (printout t ?vse crlf))


;A3.
(defrule zrus_volbu_a
?x <-   (moje_volba a)
=>
      (retract ?x)
      (assert(nactena_volba)))

;Toto posledni pravidlo ulohy A je take velmi dulezite a asi nebude jednoduche ho
objasnit ale pokusim se
;je jasne ze po vypisu databaze (volba a) budete chtit, aby se Vam menu opet
zobrazilo a uzivatel zvolil z menu neco jineho
;jinymi slovy: chcete, aby se nam pravidlo vypis_vsechny_seznamy nevykonavalo
neustale
;tedy musime docilit toho, aby podminkova cast pravidla vypis_vsechny_seznamy
nebyla splnena (pravidlo se nevykonalo)
;toho docilime tak, ze fakt (moje_volba a) z baze faktu odstranime (vlastne
uzivatel jiz nebude chtit treba zobrazit
;znovu celou databazi (to je volba a)), to nevime predem co bude chtit udelat a
tak zrusime jeho volbu a ocekavame volbu novou
;jakobychom uklizeli neporadek - to stare co ted nepotrebujeme, abychom to mohli
nekdy pouzit znovu
;tim take nebude splnena podminkova cast pravidla vypis_vsechny_seznamy, ktera
predpoklada existenci faktu (moje_volba a)
;fakt (moje_volba a) je zrusen, ale to nestaci, preci chceme hned po vypisu
zobrazit menu
;a jaky fakt k tomu potrebujeme ???
;potrebujeme fakt (nactena_volba), ktery je vyzadovan pravidlem zobraz_menu, aby
se pravidlo mohlo provest


;B1.
(defrule zadej_kontinent
      (moje_volba b)
      ;jestlize uzivatel zadal volbu b pak (pravidlo bude vykonano jako prvni) po
nem chceme, aby zadal kontinent, ktery
      ;bude nasledne ulozen do baze faktu
=>
      (printout t "Zadejte nazev kontinentu:" crlf)
      (bind ?nejaky (read))
      (assert (chci_kontinent ?nejaky)))


;B2.
(defrule vypis_zvirata_pro_muj_kontinent
        (chci_kontinent ?k)
        ;kontinent zadan

      (zvirata ?k $?zvirata)
      ;predpokladame predem, ze program ma nejaka fakta o zviratech predem

=>
      (printout t "Nalezena zvirata:" ?zvirata crlf))


;B3.
(defrule zrus_volbu_b+kontinent
```

```
      (declare(salience -1))
      ;je zde i jina moznost zruseni jiz nepouzitelnych faktu nez u prikladu A
      ;potom co budou vypsana zvirata pro dany kontinent tak az pak
      ;zrusime vse potrebne - volbu a kontinent (proto je zde priorita -1

?x <- (moje_volba b)
?y <- (chci_kontinent ?nejaky)

=>
      (retract ?x ?y)
      (assert (nactena_volba))) ;opet chceme nacist menu


;1.
(defrule ukonci_program
      (moje_volba k)
      ;jestlize uzivatel zada volbu k pak vypis ...
=>
      (printout t "Konec programu !" crlf))
      ;neni potreba odstranovat fakt moje_volba
      ;program stejne skonci a po jeho opetovnem spusteni
      ;muzete pracovat s cistym platnem (baze nebude obsahovat zadny fakt
moje_volba)


;C1.
(defrule zadej_zvire
      (moje_volba c)
=>
      (printout t "Zadejte druh zvirete:" crlf)
      (bind ?zvire (read))
      (assert (chci_zvire ?zvire)))

;C2.
(defrule vypis_kontinent_pro_moje_zvire
      (chci_zvire ?x)
      ;uzivatel zadal sve zvire, ktere chce najit

      (zvirata ?kontinent $?zvirata)
      ;pritom budeme vyhledavat v seznamu $?zvirata

      (chci_zvire ?x&:(member$ ?x $?zvirata))
      ;ale chceme, aby nase zvire ?x bylo prvkem tohoto seznamu ($?zvirata)
=>
      (printout t ?x "-" ?kontinent crlf))


;C3.
(defrule zrus_volbu_c_a_moje_zvire
      (declare (salience -10))
?x <- (moje_volba c)
?y <- (chci_zvire ?nejake)

=>
      (retract ?x)
      (retract ?y)
      (assert (nactena_volba)))

;vse nepotrebne k uloze C smazeme z baze faktu
;otazka: Proc zde musi byt deklarovana priorita s -10 a proc ma byt vubec
deklarovana?
```

```
;kdybychom prioritu vubec nedeklarovali pak by hrozilo riziko, ze bychom sice
zadali volbu c a nase zvire, ale
;k vypsani kontinentu by jiz nedoslo, protoze by se mohlo vykonat drive pravidlo
zrus_volbu_c ... a tedy
;data, ktera zadal uzivatel, by byla pryc - aby toto bylo realizovano pozdeji je
treba deklarovat nizsi prioritu nez
;ma pravidlo vypis_kontinent_pro_moje_zvire
```

```
;****************************************;
;                                ;
;           DATABAZE ZVIRAT              ;
;              (MH 2004)               ;
;                                ;
;****************************************;
;Zadani:
;A.   vypsat celou databazi zvírat i s kontinentem (nejprve nazev kontinentu a k
nemu zvírata, ktera ho mohou obyvat)
;B.   nacist nazev kontinentu a zjistit, jaka zvírata jej obyvaji
;H.   Ukoncit program
;C.   nacist druh zvírete a zjistit, kde zije
;D.   vlozit do databaze nove zvire (na konec prislusneho seznamu)

;Poznamky: ke kazdemu pravidlu je mimo poznamek a oznaceni ulohy i cislo
;cislo oznacuje poradi, kdy bude pravidlo vykonano (pokud mame menu, tak to hlavne
zavisi
;na poradi pozadavku uzivatele
;v tomto pripade bude objasneno poradi vykonavani pravidel jen pro jeden ukol
(treba uloha A), ke kteremu
;se vaze nekolik pravidel
;A2. zn. ze toto pravidlo je soucasti ulohy A a bude vykonano jako druhe v poradi
;program neresi zadani chybnych faktu pr. (chci_kontinent asoa)


(deffacts databaze
      (zvirata afrika slon zebra zirafa opice antilopa pes)
      (zvirata asie tygr slon opice panda pes)
      (zvirata evropa pes zajic jelen kocka medved rys)

      (nactena_volba)) ;tento fakt by mohl byt soucasti nove struktury deffacts,
ale i tato varianta je OK


;Pravidlo, ktere nam zobrazi volby pro operace s databazi
;je zde nulova priorita

;1.
(defrule zobraz_menu
?x <- (nactena_volba)
;tato podminka je zde velmi dulezita !!!
;chceme zobrazit menu (mit zobrazene ruzne volby) po nejake akci
;napr. po spusteni programu, po zobrazeni cele databaze atd.
;musime ale zabezpecit, aby se nam menu nezobrazovalo v dobe kdy nechceme napr.
;aby se nam nezobrazovalo jen menu a my bychom si po zobrazeni menu nemohli obsah
databaze zobrazit
;podminkova cast: za podminky, ze v bazi faktu je fakt nactena_volba, pak mi
zobraz menu
;aby se pri druhem kroku menu nezobrazovalo, tak tento fakt z baze retractem
vymazeme
;tim jiz nebude splnena podminkova cast tohoto pravidla (pravidlo se neprovede)
=>
```

```
      (retract ?x)
      (printout t "-----------------<NABIDKA>-----------------" crlf)
      (printout t "Zobrazeni cele databaze ... a" crlf)
      (printout t "Zadani kontinentu a zobrazeni zvirat ... b" crlf)
      (printout t "Zjisti o zvireti, kde zije ... c" crlf)
      (printout t "Zadej nove zvire do seznamu ... d" crlf)
      (printout t "Konec programu ... k" crlf)
      (printout t "=========================================" crlf)

      (printout t "Zadejte svoji volbu:")
      ;zadame uzivatele aby zadal jednu z voleb a nebo b ...
      (bind ?volba (read))
      (assert (moje_volba ?volba)))
      ;tuto volbu ukladame do baze faktu, abychom tuto informaci mohli uzit dale
      ;v jinem pravidle


;A2.
(defrule vypis_vsechny_seznamy
      (declare(salience 10))
      ;urcite se ptate proc se nejprve nevykona toto pravidlo, kdyz je zde uvedena
      ;priorita 10 (v pravidle zobraz_menu je priorita 0 - vlastne vubec neni
priorita deklarovana)
      ;je to proto, ze aby mohlo byt pravidlo vykonano museji byt splneny vsechny
podminky v podminkove
      ;casti pravidla a pro vykonani tohoto pravidla je treba faktu moje_volba
(druha podminka tohoto pravidla)
      ;tento fakt moje_volba bude v bazi az po vykonani pravidla zobraz_menu, kde
ho vkladame assertem do baze faktu
      ;a tedy pak muze byt toto pravidlo vykonano (v bazi jiz je moje_volba
(program tedy uz vi, co uzivatel zada))

      (moje_volba a)
      (zvirata $?vse)
=>
      (printout t "Cela databaze obsahuje tyto udaje:" crlf)
      (printout t ?vse crlf))


;A3.
(defrule zrus_volbu_a
?x <-   (moje_volba a)
=>
      (retract ?x)
      (assert(nactena_volba)))

;Toto posledni pravidlo ulohy A je take velmi dulezite a asi nebude jednoduche ho
objasnit ale pokusim se
;je jasne ze po vypisu databaze (volba a) budete chtit, aby se Vam menu opet
zobrazilo a uzivatel zvolil z menu neco jineho
;jinymi slovy: chcete, aby se nam pravidlo vypis_vsechny_seznamy nevykonavalo
neustale
;tedy musime docilit toho, aby podminkova cast pravidla vypis_vsechny_seznamy
nebyla splnena (pravidlo se nevykonalo)
;toho docilime tak, ze fakt (moje_volba a) z baze faktu odstranime (vlastne
uzivatel jiz nebude chtit treba zobrazit
;znovu celou databazi (to je volba a)), to nevime predem co bude chtit udelat a
tak zrusime jeho volbu a ocekavame volbu novou
;jakobychom uklizeli neporadek - to stare co ted nepotrebujeme, abychom to mohli
nekdy pouzit znovu
;tim take nebude splnena podminkova cast pravidla vypis_vsechny_seznamy, ktera
```

```
predpoklada existenci faktu (moje_volba a)
;fakt (moje_volba a) je zrusen, ale to nestaci, preci chceme hned po vypisu
zobrazit menu
;a jaky fakt k tomu potrebujeme ???
;potrebujeme fakt (nactena_volba), ktery je vyzadovan pravidlem zobraz_menu, aby
se pravidlo mohlo provest


;B1.
(defrule zadej_kontinent
      (moje_volba b)
      ;jestlize uzivatel zadal volbu b pak (pravidlo bude vykonano jako prvni) po
nem chceme, aby zadal kontinent, ktery
      ;bude nasledne ulozen do baze faktu
=>
      (printout t "Zadejte nazev kontinentu:" crlf)
      (bind ?nejaky (read))
      (assert (chci_kontinent ?nejaky)))


;B2.
(defrule vypis_zvirata_pro_muj_kontinent
        (chci_kontinent ?k)
        ;kontinent zadan

      (zvirata ?k $?zvirata)
      ;predpokladame predem, ze program ma nejaka fakta o zviratech predem

=>
      (printout t "Nalezena zvirata:" ?zvirata crlf))


;B3.
(defrule zrus_volbu_b+kontinent
      (declare(salience -1))
      ;je zde i jina moznost zruseni jiz nepouzitelnych faktu nez u prikladu A
      ;potom co budou vypsana zvirata pro dany kontinent tak az pak
      ;zrusime vse potrebne - volbu a kontinent (proto je zde priorita -1

?x <- (moje_volba b)
?y <- (chci_kontinent ?nejaky)

=>
      (retract ?x ?y)
      (assert (nactena_volba))) ;opet chceme nacist menu


;1.
(defrule ukonci_program
      (moje_volba k)
      ;jestlize uzivatel zada volbu k pak vypis ...
=>
      (printout t "Konec programu !" crlf))
      ;neni potreba odstranovat fakt moje_volba
      ;program stejne skonci a po jeho opetovnem spusteni
      ;muzete pracovat s cistym platnem (baze nebude obsahovat zadny fakt
moje_volba)


;C1.
(defrule zadej_zvire
```

```
      (moje_volba c)
=>
      (printout t "Zadejte druh zvirete:" crlf)
      (bind ?zvire (read))
      (assert (chci_zvire ?zvire)))

;C2.
(defrule vypis_kontinent_pro_moje_zvire
      (chci_zvire ?x)
      ;uzivatel zadal sve zvire, ktere chce najit

      (zvirata ?kontinent $?zvirata)
      ;pritom budeme vyhledavat v seznamu $?zvirata

      (chci_zvire ?x&:(member$ ?x $?zvirata))
      ;ale chceme, aby nase zvire ?x bylo prvkem tohoto seznamu ($?zvirata)
=>
      (printout t ?x "-" ?kontinent crlf))


;C3.
(defrule zrus_volbu_c_a_moje_zvire
      (declare (salience -10))
?x <- (moje_volba c)
?y <- (chci_zvire ?nejake)

=>
      (retract ?x)
      (retract ?y)
      (assert (nactena_volba)))

;vse nepotrebne k uloze C smazeme z baze faktu
;otazka: Proc zde musi byt deklarovana priorita s -10 a proc ma byt vubec
deklarovana?
;kdybychom prioritu vubec nedeklarovali pak by hrozilo riziko, ze bychom sice
zadali volbu c a nase zvire, ale
;k vypsani kontinentu by jiz nedoslo, protoze by se mohlo vykonat drive pravidlo
zrus_volbu_c ... a tedy
;data, ktera zadal uzivatel, by byla pryc - aby toto bylo realizovano pozdeji je
treba deklarovat nizsi prioritu nez
;ma pravidlo vypis_kontinent_pro_moje_zvire



;D1.
(defrule zadej_kontinent_pro_nove_zvire
      (moje_volba d)
=>
      (printout t "Zadejte nazev kontinentu kam vase zvire patri:" crlf)
      ;nejprve potrebujeme zvire nekam zaradit a pak ho do seznamu X vlozit
      (bind ?k (read))
      (assert (nove_zvire_kontinent ?k)))

;D2.
(defrule zadej_zvire_pro_seznamX
      (nove_zvire_kontinent ?k)
      ;kontinent zadan
=>
      (printout t "Zadejte nove zvire:" crlf)
      ;definujeme konkretni zvire, ktere ma byt soucasti seznamu X
```

```
      (bind ?x (read))
      (assert (pridej_zvire ?x)))


;D3.
(defrule pridej_zvire_do_seznamu
?a <- (nove_zvire_kontinent ?zeme)
?b <- (pridej_zvire ?x)

?staryseznam <- (zvirata ?zeme $?zvirata)
;rusime jen ten seznam, do ktereho chceme vlozit nove zvire
;tedy ten seznam z baze faktu, jehoz jmeno se shoduje s jmenem
;zadanym uzivatelem (vsimnete si promennych ?zeme v podm. casti pravidla)

;abychom mohli nejaky prvek vlozit do seznamu
;nejprve ho cely zrusime - ale pritom jsou v promennych ?zeme $?zvirata
;puvodni stare informace o seznamu "ulozena"

=>
      (retract ?a ?b ?staryseznam)
      (assert(zvirata ?zeme ?zvirata ?x))
      ;na konec seznamu dame nase zviratko ?x


      (printout t "Zvire bylo pridano do seznamu !" crlf))


;D4.
(defrule zrus_volbu_d_zvire_kontinent
      (declare (salience -10))
?c <- (moje_volba d)

=>
      (retract ?c)
      (assert (nactena_volba)))
```

```
;*****************************************;
;                                 ;
;           DATABAZE ZVIRAT             ;
;              (MH 2004)              ;
;                                   ;
;*****************************************;
;Zadani:
;A.   vypsat celou databazi zvírat i s kontinentem (nejprve nazev kontinentu a k
nemu zvírata, ktera ho mohou obyvat)
;B.   nacist nazev kontinentu a zjistit, jaka zvírata jej obyvaji
;H.   Ukoncit program
;C.   nacist druh zvírete a zjistit, kde zije
;D.   vlozit do databaze nove zvire (na konec prislusneho seznamu)
;E.   vlozit do databaze cely novy zaznam (napr. pro Australii)
;F.   zjistit, na kterem kontinente zije kolik druhu zvirat


;Poznamky: ke kazdemu pravidlu je mimo poznamek a oznaceni ulohy i cislo
;cislo oznacuje poradi, kdy bude pravidlo vykonano (pokud mame menu, tak to hlavne
zavisi
;na poradi pozadavku uzivatele
;v tomto pripade bude objasneno poradi vykonavani pravidel jen pro jeden ukol
(treba uloha A), ke kteremu
;se vaze nekolik pravidel
```

```
;A2. zn. ze toto pravidlo je soucasti ulohy A a bude vykonano jako druhe v poradi
;program neresi zadani chybnych faktu pr. (chci_kontinent asoa)


(deffacts databaze
      (zvirata afrika slon zebra zirafa opice antilopa pes)
      (zvirata asie tygr slon opice panda pes)
      (zvirata evropa pes zajic jelen kocka medved rys)

      (nactena_volba)) ;tento fakt by mohl byt soucasti nove struktury deffacts,
ale i tato varianta je OK


;Pravidlo, ktere nam zobrazi volby pro operace s databazi
;je zde nulova priorita

;1.
(defrule zobraz_menu
?x <- (nactena_volba)
;tato podminka je zde velmi dulezita !!!
;chceme zobrazit menu (mit zobrazene ruzne volby) po nejake akci
;napr. po spusteni programu, po zobrazeni cele databaze atd.
;musime ale zabezpecit, aby se nam menu nezobrazovalo v dobe kdy nechceme napr.
;aby se nam nezobrazovalo jen menu a my bychom si po zobrazeni menu nemohli obsah
databaze zobrazit
;podminkova cast: za podminky, ze v bazi faktu je fakt nactena_volba, pak mi
zobraz menu
;aby se pri druhem kroku menu nezobrazovalo, tak tento fakt z baze retractem
vymazeme
;tim jiz nebude splnena podminkova cast tohoto pravidla (pravidlo se neprovede)
=>
      (retract ?x)
      (printout t "-----------------<NABIDKA>-----------------" crlf)
      (printout t "Zobrazeni cele databaze ... a" crlf)
      (printout t "Zadani kontinentu a zobrazeni zvirat ... b" crlf)
      (printout t "Zjisti o zvireti, kde zije ... c" crlf)
      (printout t "Zadej nove zvire do seznamu ... d" crlf)
      (printout t "Vytvor novy-prazdny seznam ... e" crlf)
      (printout t "Zjisti pocet druhu na kontinente ...f" crlf)
      (printout t "Konec programu ... k" crlf)
      (printout t "=========================================" crlf)

      (printout t "Zadejte svoji volbu:")
      ;zadame uzivatele aby zadal jednu z voleb a nebo b ...
      (bind ?volba (read))
      (assert (moje_volba ?volba)))
      ;tuto volbu ukladame do baze faktu, abychom tuto informaci mohli uzit dale
      ;v jinem pravidle



;A2.
(defrule vypis_vsechny_seznamy
      (declare(salience 10))
      ;urcite se ptate proc se nejprve nevykona toto pravidlo, kdyz je zde uvedena
      ;priorita 10 (v pravidle zobraz_menu je priorita 0 - vlastne vubec neni
priorita deklarovana)
      ;je to proto, ze aby mohlo byt pravidlo vykonano museji byt splneny vsechny
podminky v podminkove
      ;casti pravidla a pro vykonani tohoto pravidla je treba faktu moje_volba
(druha podminka tohoto pravidla)
```

```
      ;tento fakt moje_volba bude v bazi az po vykonani pravidla zobraz_menu, kde
ho vkladame assertem do baze faktu
      ;a tedy pak muze byt toto pravidlo vykonano (v bazi jiz je moje_volba
(program tedy uz vi, co uzivatel zada))

      (moje_volba a)
      (zvirata $?vse)
=>
      (printout t "Cela databaze obsahuje tyto udaje:" crlf)
      (printout t ?vse crlf))


;A3.
(defrule zrus_volbu_a
?x <-   (moje_volba a)
=>
      (retract ?x)
      (assert(nactena_volba)))

;Toto posledni pravidlo ulohy A je take velmi dulezite a asi nebude jednoduche ho
objasnit ale pokusim se
;je jasne ze po vypisu databaze (volba a) budete chtit, aby se Vam menu opet
zobrazilo a uzivatel zvolil z menu neco jineho
;jinymi slovy: chcete, aby se nam pravidlo vypis_vsechny_seznamy nevykonavalo
neustale
;tedy musime docilit toho, aby podminkova cast pravidla vypis_vsechny_seznamy
nebyla splnena (pravidlo se nevykonalo)
;toho docilime tak, ze fakt (moje_volba a) z baze faktu odstranime (vlastne
uzivatel jiz nebude chtit treba zobrazit
;znovu celou databazi (to je volba a)), to nevime predem co bude chtit udelat a
tak zrusime jeho volbu a ocekavame volbu novou
;jakobychom uklizeli neporadek - to stare co ted nepotrebujeme, abychom to mohli
nekdy pouzit znovu
;tim take nebude splnena podminkova cast pravidla vypis_vsechny_seznamy, ktera
predpoklada existenci faktu (moje_volba a)
;fakt (moje_volba a) je zrusen, ale to nestaci, preci chceme hned po vypisu
zobrazit menu
;a jaky fakt k tomu potrebujeme ???
;potrebujeme fakt (nactena_volba), ktery je vyzadovan pravidlem zobraz_menu, aby
se pravidlo mohlo provest


;B1.
(defrule zadej_kontinent
      (moje_volba b)
      ;jestlize uzivatel zadal volbu b pak (pravidlo bude vykonano jako prvni) po
nem chceme, aby zadal kontinent, ktery
      ;bude nasledne ulozen do baze faktu
=>
      (printout t "Zadejte nazev kontinentu:" crlf)
      (bind ?nejaky (read))
      (assert (chci_kontinent ?nejaky)))


;B2.
(defrule vypis_zvirata_pro_muj_kontinent
        (chci_kontinent ?k)
        ;kontinent zadan

      (zvirata ?k $?zvirata)
      ;predpokladame predem, ze program ma nejaka fakta o zviratech predem
```

```
=>
      (printout t "Nalezena zvirata:" ?zvirata crlf))


;B3.
(defrule zrus_volbu_b+kontinent
      (declare(salience -1))
      ;je zde i jina moznost zruseni jiz nepouzitelnych faktu nez u prikladu A
      ;potom co budou vypsana zvirata pro dany kontinent tak az pak
      ;zrusime vse potrebne - volbu a kontinent (proto je zde priorita -1

?x <- (moje_volba b)
?y <- (chci_kontinent ?nejaky)

=>
      (retract ?x ?y)
      (assert (nactena_volba))) ;opet chceme nacist menu


;1.
(defrule ukonci_program
      (moje_volba k)
      ;jestlize uzivatel zada volbu k pak vypis ...
=>
      (printout t "Konec programu !" crlf))
      ;neni potreba odstranovat fakt moje_volba
      ;program stejne skonci a po jeho opetovnem spusteni
      ;muzete pracovat s cistym platnem (baze nebude obsahovat zadny fakt
moje_volba)


;C1.
(defrule zadej_zvire
      (moje_volba c)
=>
      (printout t "Zadejte druh zvirete:" crlf)
      (bind ?zvire (read))
      (assert (chci_zvire ?zvire)))

;C2.
(defrule vypis_kontinent_pro_moje_zvire
      (chci_zvire ?x)
      ;uzivatel zadal sve zvire, ktere chce najit

      (zvirata ?kontinent $?zvirata)
      ;pritom budeme vyhledavat v seznamu $?zvirata

      (chci_zvire ?x&:(member$ ?x $?zvirata))
      ;ale chceme, aby nase zvire ?x bylo prvkem tohoto seznamu ($?zvirata)
=>
      (printout t ?x "-" ?kontinent crlf))


;C3.
(defrule zrus_volbu_c_a_moje_zvire
      (declare (salience -10))
?x <- (moje_volba c)
?y <- (chci_zvire ?nejake)

=>
```

```
      (retract ?x)
      (retract ?y)
      (assert (nactena_volba)))

;vse nepotrebne k uloze C smazeme z baze faktu
;otazka: Proc zde musi byt deklarovana priorita s -10 a proc ma byt vubec
deklarovana?
;kdybychom prioritu vubec nedeklarovali pak by hrozilo riziko, ze bychom sice
zadali volbu c a nase zvire, ale
;k vypsani kontinentu by jiz nedoslo, protoze by se mohlo vykonat drive pravidlo
zrus_volbu_c ... a tedy
;data, ktera zadal uzivatel, by byla pryc - aby toto bylo realizovano pozdeji je
treba deklarovat nizsi prioritu nez
;ma pravidlo vypis_kontinent_pro_moje_zvire



;D1.
(defrule zadej_kontinent_pro_nove_zvire
      (moje_volba d)
=>
      (printout t "Zadejte nazev kontinentu kam vase zvire patri:" crlf)
      ;nejprve potrebujeme zvire nekam zaradit a pak ho do seznamu X vlozit
      (bind ?k (read))
      (assert (nove_zvire_kontinent ?k)))

;D2.
(defrule zadej_zvire_pro_seznamX
      (nove_zvire_kontinent ?k)
      ;kontinent zadan
=>
      (printout t "Zadejte nove zvire:" crlf)
      ;definujeme konkretni zvire, ktere ma byt soucasti seznamu X

      (bind ?x (read))
      (assert (pridej_zvire ?x)))


;D3.
(defrule pridej_zvire_do_seznamu
?a <- (nove_zvire_kontinent ?zeme)
?b <- (pridej_zvire ?x)

?staryseznam <- (zvirata ?zeme $?zvirata)
;rusime jen ten seznam, do ktereho chceme vlozit nove zvire
;tedy ten seznam z baze faktu, jehoz jmeno se shoduje s jmenem
;zadanym uzivatelem (vsimnete si promennych ?zeme v podm. casti pravidla)

;abychom mohli nejaky prvek vlozit do seznamu
;nejprve ho cely zrusime - ale pritom jsou v promennych ?zeme $?zvirata
;puvodni stare informace o seznamu "ulozena"

=>
      (retract ?a ?b ?staryseznam)
      (assert(zvirata ?zeme ?zvirata ?x))
      ;na konec seznamu dame nase zviratko ?x


      (printout t "Zvire bylo pridano do seznamu !" crlf))
```

```
;D4.
(defrule zrus_volbu_d_zvire_kontinent
      (declare (salience -10))
?c <- (moje_volba d)

=>
      (retract ?c)
      (assert (nactena_volba)))


;E1.
(defrule pojmenuj_novy_seznam
      (moje_volba e)
=>
      (printout t "Zadejte nazev noveho seznamu/nazev kontinentu:" crlf)
      (bind ?novy (read))
      (assert (novy_seznam ?novy)))
      ;novy nazev ulozen do baze faktu


;E2.
(defrule vytvor_novy_seznam
?x<-(novy_seznam ?novy)
;seznam jiz pojmenovan

?y <- (moje_volba e)

=>
      (assert(zvirata ?novy))
      ;vlozeni noveho seznamu do baze faktu s danym pojmenovanim

      (retract ?x ?y)
      (assert (nactena_volba))
      (printout t "Seznam byl vytvoren !" crlf))

;je zbytecne vytvaret nove pravidlo pro ruseni prebytecnych faktu


;F1.
(defrule kontinent_a_pocet_zvirat1
      (moje_volba f)
=>
      (printout t "Zadejte nazev kontinentu:"crlf)
      ;musime vedet, pro ktery kontinent bude pocitan pocet

      (bind ?k (read))
      (assert (chci_kontinent ?k)))


;F2.
(defrule kontinent_a_pocet_zvirat2
?smazvolbu <- (moje_volba f)
?smazk <-(chci_kontinent ?k)
;vime o kontinentu jehoz pocet budeme urcovat

      (zvirata ?k $?zvirata)
      ;k tomuto kontinentu existuje nejaky seznam s x polozkami
      ;vsimnete si promennych ?k
=>
      (bind ?x (length$ ?zvirata))
      ;zjistime delku vsech seznamu (pocet zvirat)
```

```
      (printout t "Na kontinente-" ?k "je:" ?x "-zvirat" crlf)
      (retract ?smazk ?smazvolbu)

      (assert (nactena_volba)))
      ;a zobrazime menu
;---------------------------------------------------------
(defrule ukonci_program
      (moje_volba k)
=>
      (printout t "Konec programu !" crlf))
;neni potreba odstranovat fakt moje_volba k
;program stejne skonci a po jeho opetovnem spusteni
;je vse ciste
```

```
;****************************************;
;                                ;
;           DATABAZE ZVIRAT              ;
;              (MH 2004)              ;
;                               ;
;****************************************;
;Zadani:
;A.   vypsat celou databazi zvírat i s kontinentem (nejprve nazev kontinentu a k
nemu zvírata, ktera ho mohou obyvat)
;B.   nacist nazev kontinentu a zjistit, jaka zvírata jej obyvaji
;H.   Ukoncit program
;C.   nacist druh zvírete a zjistit, kde zije
;D.   vlozit do databaze nove zvire (na konec prislusneho seznamu)
;E.   vlozit do databaze cely novy zaznam (napr. pro Australii)
;F.   zjistit, na kterem kontinente zije kolik druhu zvirat
;G.   zjistit, na kterem kontinente zije nejvetsi pocet zvirecich druhu


;Poznamky: ke kazdemu pravidlu je mimo poznamek a oznaceni ulohy i cislo
;cislo oznacuje poradi, kdy bude pravidlo vykonano (pokud mame menu, tak to hlavne
zavisi
;na poradi pozadavku uzivatele
;v tomto pripade bude objasneno poradi vykonavani pravidel jen pro jeden ukol
(treba uloha A), ke kteremu
;se vaze nekolik pravidel
;A2. zn. ze toto pravidlo je soucasti ulohy A a bude vykonano jako druhe v poradi
;program neresi zadani chybnych faktu pr. (chci_kontinent asoa)


(deffacts databaze
      (zvirata afrika slon zebra zirafa opice antilopa pes)
      (zvirata asie tygr slon opice panda pes)
      (zvirata evropa pes zajic jelen kocka medved rys)

      (nactena_volba)) ;tento fakt by mohl byt soucasti nove struktury deffacts,
ale i tato varianta je OK


;Pravidlo, ktere nam zobrazi volby pro operace s databazi
;je zde nulova priorita

;1.
(defrule zobraz_menu
?x <- (nactena_volba)
;tato podminka je zde velmi dulezita !!!
```

```
;chceme zobrazit menu (mit zobrazene ruzne volby) po nejake akci
;napr. po spusteni programu, po zobrazeni cele databaze atd.
;musime ale zabezpecit, aby se nam menu nezobrazovalo v dobe kdy nechceme napr.
;aby se nam nezobrazovalo jen menu a my bychom si po zobrazeni menu nemohli obsah
databaze zobrazit
;podminkova cast: za podminky, ze v bazi faktu je fakt nactena_volba, pak mi
zobraz menu
;aby se pri druhem kroku menu nezobrazovalo, tak tento fakt z baze retractem
vymazeme
;tim jiz nebude splnena podminkova cast tohoto pravidla (pravidlo se neprovede)
=>
      (retract ?x)
      (printout t "-----------------<NABIDKA>-----------------" crlf)
      (printout t "Zobrazeni cele databaze ... a" crlf)
      (printout t "Zadani kontinentu a zobrazeni zvirat ... b" crlf)
      (printout t "Zjisti o zvireti, kde zije ... c" crlf)
      (printout t "Zadej nove zvire do seznamu ... d" crlf)
      (printout t "Vytvor novy-prazdny seznam ... e" crlf)
      (printout t "Zjisti pocet druhu na kontinente ...f" crlf)
      (printout t "Nejvetsi pocet zvirecich druhu ... g" crlf)
      (printout t "Konec programu ... k" crlf)
      (printout t "=========================================" crlf)

      (printout t "Zadejte svoji volbu:")
      ;zadame uzivatele aby zadal jednu z voleb a nebo b ...
      (bind ?volba (read))
      (assert (moje_volba ?volba)))
      ;tuto volbu ukladame do baze faktu, abychom tuto informaci mohli uzit dale
      ;v jinem pravidle


;A2.
(defrule vypis_vsechny_seznamy
      (declare(salience 10))
      ;urcite se ptate proc se nejprve nevykona toto pravidlo, kdyz je zde uvedena
      ;priorita 10 (v pravidle zobraz_menu je priorita 0 - vlastne vubec neni
priorita deklarovana)
      ;je to proto, ze aby mohlo byt pravidlo vykonano museji byt splneny vsechny
podminky v podminkove
      ;casti pravidla a pro vykonani tohoto pravidla je treba faktu moje_volba
(druha podminka tohoto pravidla)
      ;tento fakt moje_volba bude v bazi az po vykonani pravidla zobraz_menu, kde
ho vkladame assertem do baze faktu
      ;a tedy pak muze byt toto pravidlo vykonano (v bazi jiz je moje_volba
(program tedy uz vi, co uzivatel zada))

      (moje_volba a)
      (zvirata $?vse)
=>
      (printout t "Cela databaze obsahuje tyto udaje:" crlf)
      (printout t ?vse crlf))


;A3.
(defrule zrus_volbu_a
?x <-   (moje_volba a)
=>
      (retract ?x)
      (assert(nactena_volba)))

;Toto posledni pravidlo ulohy A je take velmi dulezite a asi nebude jednoduche ho
```

```
objasnit ale pokusim se
;je jasne ze po vypisu databaze (volba a) budete chtit, aby se Vam menu opet
zobrazilo a uzivatel zvolil z menu neco jineho
;jinymi slovy: chcete, aby se nam pravidlo vypis_vsechny_seznamy nevykonavalo
neustale
;tedy musime docilit toho, aby podminkova cast pravidla vypis_vsechny_seznamy
nebyla splnena (pravidlo se nevykonalo)
;toho docilime tak, ze fakt (moje_volba a) z baze faktu odstranime (vlastne
uzivatel jiz nebude chtit treba zobrazit
;znovu celou databazi (to je volba a)), to nevime predem co bude chtit udelat a
tak zrusime jeho volbu a ocekavame volbu novou
;jakobychom uklizeli neporadek - to stare co ted nepotrebujeme, abychom to mohli
nekdy pouzit znovu
;tim take nebude splnena podminkova cast pravidla vypis_vsechny_seznamy, ktera
predpoklada existenci faktu (moje_volba a)
;fakt (moje_volba a) je zrusen, ale to nestaci, preci chceme hned po vypisu
zobrazit menu
;a jaky fakt k tomu potrebujeme ???
;potrebujeme fakt (nactena_volba), ktery je vyzadovan pravidlem zobraz_menu, aby
se pravidlo mohlo provest



;B1.
(defrule zadej_kontinent
      (moje_volba b)
      ;jestlize uzivatel zadal volbu b pak (pravidlo bude vykonano jako prvni) po
nem chceme, aby zadal kontinent, ktery
      ;bude nasledne ulozen do baze faktu
=>
      (printout t "Zadejte nazev kontinentu:" crlf)
      (bind ?nejaky (read))
      (assert (chci_kontinent ?nejaky)))


;B2.
(defrule vypis_zvirata_pro_muj_kontinent
        (chci_kontinent ?k)
        ;kontinent zadan

      (zvirata ?k $?zvirata)
      ;predpokladame predem, ze program ma nejaka fakta o zviratech predem

=>
      (printout t "Nalezena zvirata:" ?zvirata crlf))


;B3.
(defrule zrus_volbu_b+kontinent
      (declare(salience -1))
      ;je zde i jina moznost zruseni jiz nepouzitelnych faktu nez u prikladu A
      ;potom co budou vypsana zvirata pro dany kontinent tak az pak
      ;zrusime vse potrebne - volbu a kontinent (proto je zde priorita -1

?x <- (moje_volba b)
?y <- (chci_kontinent ?nejaky)

=>
      (retract ?x ?y)
      (assert (nactena_volba))) ;opet chceme nacist menu
```

```
;C1.
(defrule zadej_zvire
      (moje_volba c)
=>
      (printout t "Zadejte druh zvirete:" crlf)
      (bind ?zvire (read))
      (assert (chci_zvire ?zvire)))

;C2.
(defrule vypis_kontinent_pro_moje_zvire
      (chci_zvire ?x)
      ;uzivatel zadal sve zvire, ktere chce najit

      (zvirata ?kontinent $?zvirata)
      ;pritom budeme vyhledavat v seznamu $?zvirata

      (chci_zvire ?x&:(member$ ?x $?zvirata))
      ;ale chceme, aby nase zvire ?x bylo prvkem tohoto seznamu ($?zvirata)
=>
      (printout t ?x "-" ?kontinent crlf))


;C3.
(defrule zrus_volbu_c_a_moje_zvire
      (declare (salience -10))
?x <- (moje_volba c)
?y <- (chci_zvire ?nejake)

=>
      (retract ?x)
      (retract ?y)
      (assert (nactena_volba)))

;vse nepotrebne k uloze C smazeme z baze faktu
;otazka: Proc zde musi byt deklarovana priorita s -10 a proc ma byt vubec
deklarovana?
;kdybychom prioritu vubec nedeklarovali pak by hrozilo riziko, ze bychom sice
zadali volbu c a nase zvire, ale
;k vypsani kontinentu by jiz nedoslo, protoze by se mohlo vykonat drive pravidlo
zrus_volbu_c ... a tedy
;data, ktera zadal uzivatel, by byla pryc - aby toto bylo realizovano pozdeji je
treba deklarovat nizsi prioritu nez
;ma pravidlo vypis_kontinent_pro_moje_zvire



;D1.
(defrule zadej_kontinent_pro_nove_zvire
      (moje_volba d)
=>
      (printout t "Zadejte nazev kontinentu kam vase zvire patri:" crlf)
      ;nejprve potrebujeme zvire nekam zaradit a pak ho do seznamu X vlozit
      (bind ?k (read))
      (assert (nove_zvire_kontinent ?k)))

;D2.
(defrule zadej_zvire_pro_seznamX
      (nove_zvire_kontinent ?k)
      ;kontinent zadan
```

```
=>
      (printout t "Zadejte nove zvire:" crlf)
      ;definujeme konkretni zvire, ktere ma byt soucasti seznamu X

      (bind ?x (read))
      (assert (pridej_zvire ?x)))


;D3.
(defrule pridej_zvire_do_seznamu
?a <- (nove_zvire_kontinent ?zeme)
?b <- (pridej_zvire ?x)

?staryseznam <- (zvirata ?zeme $?zvirata)
;rusime jen ten seznam, do ktereho chceme vlozit nove zvire
;tedy ten seznam z baze faktu, jehoz jmeno se shoduje s jmenem
;zadanym uzivatelem (vsimnete si promennych ?zeme v podm. casti pravidla)

;abychom mohli nejaky prvek vlozit do seznamu
;nejprve ho cely zrusime - ale pritom jsou v promennych ?zeme $?zvirata
;puvodni stare informace o seznamu "ulozena"

=>
      (retract ?a ?b ?staryseznam)
      (assert(zvirata ?zeme ?zvirata ?x))
      ;na konec seznamu dame nase zviratko ?x


      (printout t "Zvire bylo pridano do seznamu !" crlf))


;D4.
(defrule zrus_volbu_d_zvire_kontinent
      (declare (salience -10))
?c <- (moje_volba d)

=>
      (retract ?c)
      (assert (nactena_volba)))


;E1.
(defrule pojmenuj_novy_seznam
      (moje_volba e)
=>
      (printout t "Zadejte nazev noveho seznamu/nazev kontinentu:" crlf)
      (bind ?novy (read))
      (assert (novy_seznam ?novy)))
      ;novy nazev ulozen do baze faktu


;E2.
(defrule vytvor_novy_seznam
?x<-(novy_seznam ?novy)
;seznam jiz pojmenovan

?y <- (moje_volba e)

=>
      (assert(zvirata ?novy))
      ;vlozeni noveho seznamu do baze faktu s danym pojmenovanim
```

```
      (retract ?x ?y)
      (assert (nactena_volba))
      (printout t "Seznam byl vytvoren !" crlf))

;je zbytecne vytvaret nove pravidlo pro ruseni prebytecnych faktu



;F1.
(defrule kontinent_a_pocet_zvirat1
      (moje_volba f)
=>
      (printout t "Zadejte nazev kontinentu:"crlf)
      ;musime vedet, pro ktery kontinent bude pocitan pocet

      (bind ?k (read))
      (assert (chci_kontinent ?k)))


;F2.
(defrule kontinent_a_pocet_zvirat2
?smazvolbu <- (moje_volba f)
?smazk <-(chci_kontinent ?k)
;vime o kontinentu jehoz pocet budeme urcovat

      (zvirata ?k $?zvirata)
      ;k tomuto kontinentu existuje nejaky seznam s x polozkami
      ;vsimnete si promennych ?k
=>
      (bind ?x (length$ ?zvirata))
      ;zjistime delku vsech seznamu (pocet zvirat)

      (printout t "Na kontinente-" ?k "je:" ?x "-zvirat" crlf)
      (retract ?smazk ?smazvolbu)

      (assert (nactena_volba)))
      ;a zobrazime menu


;G1.
(defrule nejdelsi_seznam
      (moje_volba g)
      (zvirata ?kontinent1 $?zvirata1)
      (not (zvirata ?kontinent2 $?zvirata2&: (>(length$ $?zvirata2)(length$
$?zvirata1))))
      ;porovnavame dva seznamy, pricemz rikame: za podminky ze neexistuje seznam
druhy takovy, pro ktery plati:
      ;ze jeho delka je vetsi nez delka prvniho seznamu
=>
      (printout t "Nejvetsi pocet zvirat je v kontinente:" ?kontinent1 crlf))

;pravidlo nejdelsi_seznam se realizuje nekolikrat (porovnavame mezi sebou ruzne
seznamy a to nelze udelat jen v jednom kroku)
;proto je nutne zrusit fakt (moje_volba g) az v novem pravidle


;G2.
(defrule zrus_volbu_g
  (declare (salience -10))
?x <- (moje_volba g)
```

```
=>
      (retract ?x)
      (assert (nactena_volba)))



;H1.
(defrule ukonci_program
      (moje_volba k)
=>
      (printout t "Konec programu !" crlf))
;neni potreba odstranovat fakt moje_volba k
;program stejne skonci a po jeho opetovnem spusteni
;je vse ciste
```

LEKCE 7.

Náplň lekce:

Dnešní lekce bude věnována jen opakování toho co jsme se zatím naučili.

Úloha 1.

Pokuste se vytvořit program, který z několika seznamů vytvoří jejich sjednocení. Odstraňte dále z tohoto seznamu duplicitní výskyty téhož prvku. Nakonec seznam vypište na obrazovku.

Řešení několika způsoby:

```
;*****************************************;
;                                         ;
;           OPERACE SE SEZNAMY            ;
;          Sjednoceni a duplicity I.      ;
;              (MH 2004)                  ;
;                                         ;
;*****************************************;
;Zadani:
;1/ Pokuste se vytvorit program, ktery z nekolika seznamu vytvori jejich
sjednoceni.
;2/ Odstrante dale z tohoto seznamu duplicitni vyskyty tehoz prvku.
;3/ Nakonec seznam vypiste na obrazovku

;Princip vytvorení sjednoceni: vytvorime si jeden novy seznam, do ktereho
;vlozime vsechny druhy zvirat ze seznamu A az D


(deffacts seznamy
      (seznam A pes kocka lev)
      (seznam B andulka tygr simpanz koala)
      (seznam C kocka tygr orangutan)
      (seznam D koala pes mys)
)
```

```
;Pravidlo, ktere sjednoti vsechny prvky seznamu do jednoho seznamu
;Toto pravidlo probehne jen 1x (vse se realizuje naraz)

(defrule sjednot_seznamy
?prycX<-(seznam A $?zvirata1)
?prycY<-(seznam B $?zvirata2)
?prycZ<-(seznam C $?zvirata3)
?prycS<-(seznam D $?zvirata4)
=>
      (retract ?prycX ?prycY ?prycZ ?prycS)
      (assert (sjednoceni ?zvirata1 ?zvirata2 ?zvirata3 ?zvirata4)))
      ;vytvoreni noveho seznamu s nazvem sjednoceni, jehoz soucasti
      ;jsou dana zvirata
      ;vsimnete si promennych ?zvirata (resp.$?zvirata1) - seznamy se zviraty sice
z baze odstranite, ale
      ;v usuzovaci casti muzete vyuzit obsahy promennych u rusenych faktu, ktere
jsou soucasti
      ;noveho seznamu (viz. prikaz assert)


(defrule odstran_duplicity
?pryc<-(sjednoceni $?neco1 ?prvek $?neco2 ?prvek $?neco3)
;vychazime jen ze sjednoceneho (noveho) seznamu
;hledame stejne prvky pak si vsimnete nazvu promennych ?prvek - jejich nazvy
;jsou take stejne !!!

=>
      (retract ?pryc)
      ;proc novy seznam (se sjednocen. prvky) rusime?
      ;proto, ze chceme odstranit ze seznamu duplicitni prvky a chceme mit seznam
bez techto prvku duplicitnich
      ;tedy nam vznikne novy seznam bez duplicit viz. prikaz assert bez druhe
promenne s nazvem ?prvek

      (assert (sjednoceni ?neco1 ?prvek ?neco2 ?neco3)))



(defrule vypis_sjednoceny_seznam_bez_duplicit
      (declare(salience -5))
      ;kdybychom zde nedali prioritu, pak by hrozilo riziko, ze se nam sice mohou
odstranit duplicity v prvnim seznamu, ale
      ;pak by Clips hned presel na vykonavani tohoto pravidla a cely proces
odstranovani duplicit by tak byl narusen
      ;Clips by mohl nejprve duplicitu odstranit - pak seznam vypsat - odstranit
duplicitu - a pak vypsat atd.
      ;asi je lepsi nejprve odstranit duplicity ve vsech seznamech a pak seznam
vypsat

      (sjednoceni $?zvirata)
=>
      (printout t "Seznam bez duplicit:" ?zvirata crlf))

;Zkuste si nedat v tomto pravidle prioritu a pritom si menit i strategii
inferencniho mechanismu
;Execution/Options -> Strategy
```

```
;*****************************************;
```

```
;                                    ;
;          OPERACE SE SEZNAMY          ;
;         Sjednoceni a duplicity II.         ;
;             (MH 2004)          ;
;                                  ;
;*****************************************;
;Zadani:
;1/ Pokuste se vytvorit program, ktery z nekolika seznamu vytvori jejich
sjednoceni.
;2/ Odstrante dale z tohoto seznamu duplicitni vyskyty tehoz prvku.
;3/ Nakonec seznam vypiste na obrazovku

;Princip vytvorení sjednoceni: dany seznam sjednoceni je v bazi faktu uveden jako
fakt - tento seznam je zatim prazdny
;postupne bude plnen druhy zvirat
;seznam sjednoceni bude pred naplnenim novych druhu zvirat smazan a pak naplnen
;nejdrive se jakoby seznam pro plneni pripravi (zrusi = retract) a pak plni
(assert)
;Vyhodou tohoto reseni je fakt, ze se Vam z baze neodstrani puvodni seznamy A-C

(deffacts seznamy
      (seznam A lev kocka pes had)
      (seznam B opice tygr)
      (seznam C pes jezevec had)
      (sjednoceni)
)


;Pravidlo pro vytvoreni sjednoceni jinym zpusobem nez varianta I.
;Pravidlo probehne tolikrat kolik je seznamu - narozdil od varianty I.
;tam probehne toto pravidlo jen 1x (vse se realizuje naraz)

(defrule vytvor_sjednoceni
        (seznam ?x $?zvirata)
      ;seznam(y) se zviraty v bazi existuji

?pryc<-(sjednoceni $?polozky)
      ;budeme rusit stary seznam (jeste bez novych zvirat)

      (not(pridany_seznam ?x))
      ;musime konrolovat, se kterym seznamem jsme jiz pracovali
      ;aby druhy zvirat ze seznamu A nebyly soucasti seznamu sjednoceni podruhe
      ;to by se pak mohl i program zacyklit (bezel by neustale dokola bez
zastaveni)

=>
      (retract ?pryc)
      ;pripravujeme seznam sjednoceni pro plneni novymi druhy zvirat

      (assert(sjednoceni ?polozky ?zvirata))
      ;seznam je obohacen o ty puvodni druhy zvirat (?polozky) + ty nove druhy
zvirat (?zvirata)

      (assert(pridany_seznam ?x))
      ;do baze je ulozena informace o seznamu jehoz polozky jsou jiz v seznamu
sjednoceni
      ;nechceme s nim podruhe pracovat
)


;Nasledujici dve pravidla jsou temer beze zmeny (stejne jako u verze I.)
```

```
;Jsou zde upraveny priority z duvodu rozdilnosti pravidla vytvor_sjednoceni
;u obou verzi


(defrule odstran_duplicity
      (declare(salience -10))
?pryc<-(sjednoceni $?neco1 ?zvire $?neco2 ?zvire $?neco3)
=>
      (retract ?pryc)
      (assert (sjednoceni $?neco1 ?zvire $?neco2 $?neco3)))



(defrule vypis_sjednoceny_seznam_bez_duplicit
      (declare(salience -20))
      (sjednoceni $?zvirata)
=>
      (printout t "Seznam bez duplicit:" ?zvirata crlf))
```

```
;****************************************;
;                                        ;
;          OPERACE SE SEZNAMY          ;
;         Sjednoceni a duplicity III.    ;
;             (MH 2004)                  ;
;                                        ;
;****************************************;
;Zadani:
;1/ Pokuste se vytvorit program, ktery z nekolika seznamu vytvori jejich
sjednoceni.
;2/ Odstrante dale z tohoto seznamu duplicitni vyskyty tehoz prvku.
;3/ Nakonec seznam vypiste na obrazovku

;Princip vytvoreni sjednoceni: skoro stejny jako ve verzi II.
;jen zde naraz rusime seznam se sjednocenim i samotny seznam s druhy zvirat, ktera
budou
;soucasti noveho seznamu sjednoceni


(deffacts seznamy
      (seznam A lev kocka pes had)
      (seznam B opice tygr)
      (seznam C pes jezevec had)
      (sjednoceni)
)



(defrule vytvor_sjednoceni
?pryc1<-(sjednoceni $?prvky)
?pryc2<-(seznam ?x $?zvirata)
=>
      (retract ?pryc1 ?pryc2)
      ;vse ruseno naraz

      (assert(sjednoceni ?prvky ?zvirata)))



(defrule odstran_duplicity
```

```
      (declare(salience -10))
?pryc<-(sjednoceni $?neco1 ?zvire $?neco2 ?zvire $?neco3)
=>
      (retract ?pryc)
      (assert (sjednoceni $?neco1 ?zvire $?neco2 $?neco3)))



(defrule vypis_sjednoceny_seznam_bez_duplicit
      (declare(salience -20))
      (sjednoceni $?zvirata)
=>
      (printout t "Seznam bez duplicit:" ?zvirata crlf))
```

Úloha 2.

Tato úloha je malinko obtížnější...

1. pomocí seznamů definujte síť linek MHD třeba takto:

- (linka 1 Hl_nadrazi Tesco Strelnice Lipky Akvarium Nemocnice Heyrovskeho)
- (linka 2 Hl_nadrazi Gocarova Ulrichovo_nam Adal)
- (linka 6 Hl_nadrazi Gocarova Ulrichovo_nam Adal Urad Gymnazium)
- (linka 13 Hl_nadrazi Tesco Central Muzeum Aldis Bedrna)

2. napište pravidlo pro výpis všech zastávek na uživatelem hledané lince
3. napište pravidlo pro vyhledání všech linek, které zastavují na uživatelem zadané zastávce
4. napište pravidlo pro přidání nové linky do báze faktů
5. napište pravidlo pro zrušení určité zastávky - její název musí být vymazán ze všech seznamů
6. napište pravidlo pro vyhledání spojení mezi dvěma zadanými zastávkami
7. zobrazení celé databáze
8. umožněte uživateli program ukončit (vhodným způsobem - ne stisknutím tlačítka Reset nebo něco podobného)
9. obohaťte svůj prográmek nabídkou, ze které si uživatel bude moci vybrat, co chce s databází v dané chvíli dělat

Řešení:

```
;*****************************************;
;                                         ;
;                MHD                        ;
;             (MH 2004)                   ;
;                                         ;
;*****************************************;
;Zadani:
;a) vytvorte strukturu faktu s linkami a zastavkami
;b) napiste pravidlo pro vypis vsech zastavek na uzivatelem hledane lince
;c) napiste pravidlo pro vyhledani vsech linek, ktere zastavuji na uzivatelem
zadane zastavce
;d) napiste pravidlo pro pridani nove linky do baze faktu
;e) napiste pravidlo pro zruseni urcite zastavky - jeji nazev musi byt vymazan ze
vsech seznamu
;f) napiste pravidlo pro vyhledani spojeni mezi dvema zadanymi zastavkami
;g) zobrazení cele databaze
```

```
;h) umoznite uzivateli program ukoncit (vhodnym zpusobem - ne stisknutim tlacitka
Reset nebo neco podobneho)
;ch) vytvorte menu, ze ktereho si bude moci uzivatel vybirat


;a
(deffacts linky
      (linka 1 Hl_nadrazi Tesco Strelnice Lipky Akvarium Nemocnice Heyrovskeho)
      (linka 2 Hl_nadrazi Gocarova Ulrichovo_nam Adal)
      (linka 6 Hl_nadrazi Gocarova Ulrichovo_nam Adal Urad Gymnazium)
      (linka 13 Hl_nadrazi Tesco Central Muzeum Aldis Bedrna)
      (menu nacteno))


;ch. Pravidlo zobrazujici menu

(defrule vypis_menu
?pryc<- (menu nacteno)
=>
      (retract ?pryc)
      (printout t "------------<MENU>------------" crlf)
      (printout t "Zobraz info k lince.....A" crlf)
      (printout t "Zobraz info ke stanici..B" crlf)
      (printout t "Zadejte novou linku.....C" crlf)
      (printout t "Zobrazte vse .........D" crlf)
      (printout t "Zrus zastavku ..........E" crlf)
      (printout t "Vyhledej spojeni pro zastavky ... F" crlf)
      (printout t "Konec programu ....K" crlf)
      (printout t "------------------------------" crlf)
      (printout t "Zadejte svoji volbu:")

      (bind ?volba (read))
      (assert (moje_volba ?volba)))

;-------------------------------------------
;b. Pravidlo pro zadani pozadovane linky

(defrule zadejte_linku
      (moje_volba A)
      ;uzivatel zvolil volbu A
=>
      (printout t "Zadejte cislo linky:" crlf)
      (bind ?x (read))
      (assert (chci_linku ?x)))
      ;info o lince ulozena do baze faktu

;b.
(defrule vypis_info_k_lince
      (moje_volba A)
      (chci_linku ?x)
      (linka ?x $?seznam)
      ;linka ?x (od uzivatele) je evidovana))

?prycV<-(moje_volba A)
?prycL<-(chci_linku ?x)
=>
      (retract ?prycV ?prycL)
      ;odstranujeme tyto informace pro moznost opetovneho zadani volby A a nejake
linky

      (assert (menu nacteno))
```

```
      ;timto prikazem bude splnena podminkova cast v pravidle vypis_menu

      (printout t "Informace k lince c." ?x ":" ?seznam crlf))

;-----------------------------------------

;c. Pravidlo pro zadani stanice

(defrule zadejte_stanici
      (moje_volba B)
=>
      (printout t "Zadejte nazev stanice:" crlf)
      (bind ?y (read))
      (assert (chci_stanici ?y)))


;c.
(defrule vypis_ke_stanici_linky
       (moje_volba B)
       (linka ?l $?seznam)
       (chci_stanici ?y&:(member$ ?y $?seznam))
=>
      (printout t "Ke stanici s nazvem:" ?y "-zajizdi tyto linky:" ?l ";" crlf))

;c.
(defrule zrus_volbuB_a_stanici
      (declare(salience -10))
?prycV<-(moje_volba B)
?prycS<-(chci_stanici ?x)
=>
      (retract ?prycV ?prycS)
      (assert(menu nacteno)))

;c. je treba pro zruseni jiz nepotrebnych informaci nove pravidlo, protoze
;pravidlo vypis_ke_stanici_linky nebude realizovano jen jednou
;az po tom co se vsechny linky k stanici najdou pak se nepotrebne informace smazou
;to je zajisteno def. priority (declare(salience -10))

;-----------------------------------------

;d. Pravidlo pro pridani nove linky

(defrule pridej_novou_linku
?pryc<-(moje_volba C)
=>
      (retract ?pryc)
      (printout t "Zadejte cislo linky:" crlf)
      (bind ?linka (read))
      (assert (linka ?linka))
      (printout t "Linka byla pridana" crlf)
      (assert (menu nacteno)))

;-----------------------------------------

;e. Pravidlo pro zadani nazvu zastavky, kterou chceme zrusit

(defrule zadej_nazev_rusene_zastavky
      (moje_volba E)
=>
      (printout t "Zadejte nazev rusene zastavky:" crlf)
      (bind ?zastavka (read))
```

```
      (assert (rusena_zastavka ?zastavka)))


;e. Pravidlo, ktere pozadovanou zastavku zrusi

(defrule zruseni_zastavky
      (moje_volba E)
      (rusena_zastavka ?zastavka)
?pryc<-(linka ?x $?neco1 ?zastavka $?neco2)
=>
      (retract ?pryc)
      (assert (linka ?x ?neco1 ?neco2)))
      ;do baze faktu je vlozen fakt bez zastavky (?zastavka)


;e. - zruseni vseho nepotrebneho (volba + zrusena zastavka)

(defrule zrus_volbu_a_zastavku
      (declare(salience -10))
?prycV<-(moje_volba E)
?prycZ<-(rusena_zastavka ?zastavka)
=>
      (printout t "Zastavka zrusena!" crlf)
      (retract ?prycV ?prycZ)
      (assert(menu nacteno)))

;----------------------------------------

;f - Pravidlo pro zadani nazvu pocatecni zastavky a konencne zastavky

(defrule zadej_pocatec_a_cil_zastavku
      (moje_volba F)
=>
      (printout t "Zadejte pocatecni zastavku:" crlf)
      (bind ?a (read))
      (assert (chci_pocatek ?a))
      (printout t "Zadejte cilovou zastavku:" crlf)
      (bind ?b (read))
      (assert(chci_cil ?b)))


;f - Pravidlo, ktere urci linku, ktera Vas muze dovezt z pocatku do cile

(defrule urci_spoj
      (moje_volba F)
      (chci_pocatek ?x)
      (chci_cil ?y)
      (linka ?l $?neco1 ?x $?neco2 ?y $?neco3)
=>
      (printout t "Muzete zvolit spoj:" ?l crlf))

;f

(defrule zrus_vse_pro_volbuF
      (declare(salience -5))
?prycV<-(moje_volba F)
?prycP<-(chci_pocatek ?a)
?prycC<-(chci_cil ?b)
=>
      (retract ?prycV ?prycP ?prycC)
      (assert (menu nacteno)))
```

```
;-----------------------------------------
;g. Zobrazeni cele databaze

(defrule zobraz_vse
        (moje_volba D)
      (linka ?nejaka $?stanice)
=>
      (printout t "Znam tyto linky:" ?nejaka ?stanice crlf))


;g.

(defrule zrus_volbuD
      (declare(salience -5))
?pryc<-(moje_volba D)
=>
      (retract ?pryc)
      (assert(menu nacteno)))


;-----------------------------------------
;h.

(defrule konec_programu
      (moje_volba K)
=>
      (printout t "Konec programu!" crlf))
```

LEKCE 8.

Náplň lekce:

- Šablony (deftemplates) - První nahlédnutí
- Vysvětlující příklad pro použití šablony
- Vlastnosti rubrik
- Možnosti pro allowed-values
- Nevyplňování rubrik
- Funkce readline a explode$
- Další příklady k procvičení

Šablony (deftemplates) - první nahlédnutí

Až dosud jsme používali tzv. jednoduché fakty, které obsahovaly název relace s hodnotou nebo hodnotami (= seznam). Fakt mohl obsahovat hodnoty na sobě zcela nezávislé (viz. níže příklad 1.) nebo byly součástí faktu takové hodnoty, které spolu nějak souvisely (viz. níže příklad 2.). Sestavení faktů tak jak vidíme u příkladu 2 je celkem nepraktické. Například při pohledu na fakt nemusí být patrné co znamenenají jeho jednotlivé položky (musíme často hádat co položka znamená ... Alex_Proyas je herec nebo režisér?), obtížněji se s celým faktem zachází (vyhledávání), zápis je více chaotický.

Z těchto a jiných důvodů se zde nabízí lepší řešení problému - VYUŽITÍ ŠABLON.

Příklad 1.

```
(barva cervena zelena zluta cerna modra)
(zvire andulka pes)
(rostlina pampeliska)
```

Příklad 2.

```
(osoba muz Jan_Novak Manesova_390 Hradec_Kralove)
(film sci-fi Ja_robot 2004 Alex_Proyas Will _Smith Bridget_Moynahan
James_Cromwell)
```

Co je šablona ?

Jedná se o programovou strukturu, díky které můžete vytvářet tzv. strukturované fakty. Jinak řečeno svému faktu/faktům dodáte určitou - Vámi předem definovanou strukturu, podle které budou Vaše fakty vytvořeny. Šablonou vlastně vytvoříte jakýsi základ pro Vaše budoucí fakty a tak jim dodáte určitý řád. Můžete si šablonu představit jako formulář. Ten obsahuje určitá pole. Každé pole má nějakou vlastnost/vlastnosti. Máte např. pole jméno, které může např. nabývat max. 20 znaků, pole stav jen několika hodnot svobodný/á, vdaný/á apod.

Poznámka: Tato struktura je podobná struktuře Record - záznam z jiných programovacích jazyků. K položkám faktu budete moci přistupovat podle jména nejen podle pořadí.

VYSVĚTLUJÍCÍ PŘÍKLAD

Vytvoření šablony

Vytvoření šablony je demonstrováno na obrázku obr.1.

Jak vidíte na obr.1, šablona je tvořena svým názvem a položkami (= sloty, rubriky) s názvem. Konkrétní příklad je uveden na obr.2. U každé položky můžeme definovat její další charakteristiky. Bude se jednat o podrobnější specifikaci polí resp. jejich vymezení.

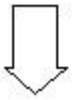
obr. 1 OBECNÁ SYNTAXE ŠABLONY

```
(deftemplate nazev_sablony

  (slot nejaka_polozka)
  (slot nejaka_dalsi_polozka)
  (slot a_jeste_dalsi_polozka)
  ...

  (multislot nekolik_polozek)
)
```

Definujeme skupinu příbuzných polí

obr.2 KONKRÉTNĚ

```
(deftemplate film
  (slot nazev)
  (slot rok)
  (slot rezie)
  (slot zanr)
  (multislot herci)
)
```



Základ pro vytvoření faktů (struktury deffacts)

VLASTNOSTI SLOTŮ (položek, rubrik)

type	SYMBOL, STRING, LEXEME (pozn. LEXEME zahrnuje oba předchozí typy SYMBOL i STRING)
	INTEGER (celé číslo),FLOAT (desetinné číslo), NUMBER (oba předchozí typy)
	BOOLEAN (true, false)
range	rozpětí hodnot
allowed-values	výčet povolených hodnot v příp. definice typu slotu
default	předpokládaná hodnota rubriky, užijeme vždy, když rubrika nemá nějakou hodnotu - ta defaultní se nám do pole sama doplní

Možnosti pro allowed-values

Allowed-symbols: bohaty maly svetly
Allowed-strings: "Michal" "Jana" "Marek"
Allowed-numbers: 1 2 4.5 -2.5 5e-4

Allowed-integers: -120 51
Allowed-floats: 2.5 -5.8 215.12
Allowed-values: "Jan" chudy 120 -5 (dovoleny všechny možné typy hodnot)

Na co si dát pozor?

- na rozdíl mezi symbols a strings
- jestliže užiji range, nemohu užít allowed-values a opačně
- u jednoho slotu nemohu dát více allowed
- vlastnosti u type (např. LEXEME, INTEGER, STRING ...) pište vždy velkými písmeny

Nevyplňování rubrik

Máme několik možností jak nakládat s rubrikami:

1. vyplnit všechny hodnoty slotů

```
(deffacts moje_filmy
     (film (nazev Vetrelec) (rok 1979) (rezie Ridley_Scott) (herci
Sigourney_Weaver John_Hurt Ian_Holm))
)
```

2. vyplnit jen některé hodnoty, pak jsou zde určité odlišnosti:

1. pokud bylo nastaveno default, Clips sám hodnotu u default doplní
2. pokud jsme uvedli typ SYMBOL nebo nic, pak Clips doplní nil
3. pokud INTEGER či FLOAT, pak bude doplněno 0 nebo 0.0 (bez range)
4. pokud jsme uvedli range, pak bude dosazena nejnižší hodnota rozsahu
5. pokud nebylo nic uvedeno u multislotu, pak Clips nechá prázdný seznam

Šablony v PRAXI

Úloha s filmy

Zadání:

1. definujte šablonu pro filmy a vytvořte fakty, které budou na definované šabloně založené
2. napište pravidlo, které Vám vypíše název filmu, jeho žánr a herce, kteří v něm hrají
3. napište pravidlo, které Vám vypíše filmy, které byly natočeny před rokem 1980
4. dále se pokuste vytvořit pravidlo, které vypíše filmy, kde hrála nebojácná herečka Sigourney Weaver
5. ... a pravidlo, kterým zjistíte, ve kterých filmech nehrál herec Lance Henriksen
6. a to těžší nakonec: vytvořte pravidlo, kterým zjistíte filmy, které byly natočeny po roce 1990. Tyto filmy dejte do noveho seznamu a jeho položky potom vypište.

Úloha je blíže demonstrována na obrázku:

```
(deftemplate film
  (slot nazev (type SYMBOL))
  (slot rok (type INTEGER) (range 1924 2005))
  (slot rezie (type SYMBOL))
  (slot zanr (default sci-fi))
  (multislot herci (type SYMBOL))
)

)
```

vlastnost nazev nabývá jen jedné hodnoty

V deffacts není žádná zmínka o žánru; u všech filmů je stejný. Při takto napsaném programu a jeho spuštění bude žánr sci-fi automaticky součástí každé položky (každého filmu).

vlastnost herci nabývá více hodnot

použití šablony film pro vytvoření konkrétních faktů naplnění slotů a multislotů konkrétními hodnotami

```
(deffacts moje_filmy
  (film (nazev Vetrelec)
        (rok 1979)
        (rezie Ridley_Scott)
        (herci Sigourney_Weaver John_Hurt Ian_Holm)
)

  (film (nazev Vetrelci)
        (rok 1986)
        (rezie James_Cameron)
        (herci Sigourney_Weaver Lance_Henriksen
         Michael_Biehn Paul_Reiser)
)

  (film (nazev Ruda_planeta)
        (rok 2000)
        (rezie Anthony_Hoffman)
        (herci Carrie_Anne_Moss Val_Kilmer
        Tom_Sizemore)
)

  (film (nazev Umela_intelligence)
        (rok 2001)
        (rezie Steven_Spielberg)
        (herci Haley_Joel_Osment Jude_Law)
)

  (film (nazev Metropolis)
        (rok 1925)
        (rezie Fritz_Lang)
        (herci Brigitte_Helm Alfred_Abel Gustav_Frolich
         Fritz_Rasp)
)

)
```

název použité šablony

Celá úloha

```
;*****************************************;
;                                         ;
;       FILMY aneb jak na sablony I.      ;
;             (MH 2005)                   ;
;                                         ;
;*****************************************;


;SABLONA film

(deftemplate film
  (slot nazev (type SYMBOL))
  (slot rok (type INTEGER) (range 1924 2005)) ;budeme zaznamenavat filmy od roku
1924 do roku 2005
  (slot rezie (type SYMBOL))
  (slot zanr (default sci-fi)) ;budeme zaznamenavat jen scifinky
  (multislot herci (type SYMBOL))
)

;FAKTY ZALOZENE NA SABLONE film

(deffacts moje_filmy
  (film (nazev Vetrelec) (rok 1979) (rezie Ridley_Scott) (herci Sigourney_Weaver
John_Hurt Ian_Holm))
  (film (nazev Vetrelci) (rok 1986) (rezie James_Cameron) (herci Sigourney_Weaver
Lance_Henriksen Michael_Biehn Paul_Reiser))
  (film (nazev Ruda_planeta) (rok 2000) (rezie Anthony_Hoffman) (herci
Carrie_Anne_Moss Val_Kilmer Tom_Sizemore))
  (film (nazev Umela_inteligence) (rok 2001) (rezie Steven_Spielberg) (herci
Haley_Joel_Osment Jude_Law))
  (film (nazev Metropolis) (rok 1925)(rezie Fritz_Lang) (herci Brigitte_Helm
Alfred_Abel Gustav_Frolich Fritz_Rasp))

)

;Tento seznam se tyka az predposledniho a posledniho pravidla
;Bude naplnen filmy, ktere byly natoceny po roce 1990

(deffacts novy_seznam
  (novy_seznam)
)


;a) Pravidlo vypise nazev filmu, zanr a herce
;V podminkove casti si muzete definovat jen ty polozky, ktere chcete vypsat (neni
nutno je vypisovat vse (od nazvu az po herce),
;kdyz je potom vubec k nicemu nepotrebujete
;kdyby jste chteli vypsat uplne vsechny polozky - tak je v podm. casti zkratka
definujete, pritom neni nutne
;dodrzovat poradi polozek - muzete si nejdriv definovat rezisera, pak rok a pak to
dalsi
;mate zde vetsi volnost nez u nestrukturovanych faktu

(defrule vypis_vse
 (film (nazev ?n) (zanr ?z) (herci $?h))
=>
```

```
  (printout t "V databazi je" " " ?z " " "film:" ?n " " "s herci:" ?h crlf)


)

;b) Pravidlo, ktere vypise filmy natocene pred rokem 1980

(defrule starsi_nez_1980
  (declare (salience -1))
  (film (nazev ?nejaky) (rok ?r&: (< ?r 1980)))
=>
  (printout t " " crlf)
  (printout t "-------< Starsi nez 1980 >-------" crlf)
  (printout t ?nejaky " " "z roku:" ?r crlf)
)

;b) Nebo jinak zapsano

;(defrule starsi_nez_1980
;  (declare (salience -2))
;  (film (nazev ?nejaky) (rok ?r))
;  (test (< ?r 1980))
;=>
;  (printout t " " crlf)
;  (printout t "-------< Starsi nez 1980 >-------" crlf)
;  (printout t ?nejaky " " "z roku:" ?r crlf)
;)



;c) Pravidlo, ktere vypise filmy, ve kterych hrala Sigourney Weaver

(defrule najdi_filmy_pro_SW
  (declare (salience -3))
  (film (nazev ?nejaky) (herci $?seznam&:(member$ Sigourney_Weaver $?seznam)))
=>
  (printout t " " crlf)
  (printout t ""------------------< SW >------------------" " crlf)
  (printout t "Sigourney Weaver hrala ve filmu:" ?nejaky crlf)
)


;d) Pravidlo, ve kterych filmech nehral Lance Henriksen

(defrule neni_Lance
  (declare (salience -4))
  (film (nazev ?nejaky) (herci $?seznam&:(not(member$ Lance_Henriksen $?seznam))))
=>
  (printout t " " crlf)
  (printout t "----< Lance Henriksen neni ve filmu >----" crlf)
  (printout t ?nejaky crlf)
)


;e) Obecne: Vytvoreni pravdla, kde film byl natocen po roce 1990 a tyto filmy
;dejte do zvlastniho seznamu, jehoz polozky potom vypiste

;Nejprve zjisteni filmu po roce 1990 + ulozeni do baze faktu

(defrule filmy_po_1990
  (declare (salience -5))
```

```
   (film (nazev ?nejaky) (rok ?r&: (> ?r 1990)))
=>
   (assert (po_roce_1990 ?nejaky))
)


;e) Dame zjistene filmy do noveho seznamu

(defrule filmy_do_seznamu
   (declare (salience -6))
   ?pryc1<-(po_roce_1990 ?nejake)
   ?pryc2<-(novy_seznam $?neco)
=>
   (retract ?pryc1 ?pryc2)
   (assert (novy_seznam ?neco ?nejake))
)


;e) Filmy z noveho seznamu vypiseme

(defrule vypis_novy_seznam
   (declare (salience -7))
   (novy_seznam $?seznam)
=>
   (printout t " " crlf)
   (printout t "-------------------------------------------------" crlf)
   (printout t "Filmy po roce 1990:" ?seznam crlf)
)
```

Funkce readline a explode$

Další příklady k procvičení

Funkce readline a explode$

Funkce readline je funkcí, která nám umožní zadávat v programu více hodnot najednou. Například máte zadávat informace k nějaké osobě. Je pravděpodobné, že např. město, kde osoba bydlí, bude mít více slov (Hradec Králové). Všechny informace o městu budete chtít mít zachycené. Proto zde existuje funkce readline, která Vám to umožní.

Demonstrace použití funkce readline a explode je v příkladu readline_explode

```
;*****************************************;
;                                         ;
;  Vysvetleni funkce explode$ a readline  ;
;             (MH 2005)                   ;
;                                         ;
;*****************************************;


(deftemplate osoba
```

```
   (slot jmeno (type SYMBOL))
   (slot prijmeni (type SYMBOL))
   (slot vek (type INTEGER))
   (multislot ulice (type SYMBOL))
   (multislot mesto (type SYMBOL))
   (multislot zajmy (type SYMBOL))
)

(defrule zadavani_informaci
=>
   (printout t "Zadejte jmeno osoby:" crlf)
   (bind ?jmeno (read))
   (printout t "Zadejte prijmeni osoby:" crlf)
   (bind ?prijmeni (read))
   (printout t "Zadejte vek osoby:" crlf)
   (bind ?vek (read))
   (printout t "Zadejte ulici, kde osoba bydli:" crlf)
   (bind ?ulice (readline))
   (printout t "Zadejte mesto, kde osoba bydli:" crlf)
   (bind ?mesto (readline))
   (printout t "------------Detaily osoby:------------" crlf)
   (printout t "Zadejte zajmy teto osoby:" crlf)
   (bind ?zajmy (readline))

   ;bez explode:(assert (osoba (jmeno ?jmeno) (prijmeni ?prijmeni) (vek ?vek)
(ulice $?ulice) (mesto $?mesto) (zajmy $?zajmy)))
   (assert (osoba (jmeno ?jmeno) (prijmeni ?prijmeni) (vek ?vek) (ulice (explode$
?ulice)) (mesto (explode$ ?mesto))
   (zajmy (explode$ ?zajmy))))
)


(defrule vypis_co_bylo_zadano
   (osoba (jmeno ?jmeno) (prijmeni ?prijmeni) (vek ?vek) (ulice $?ulice) (mesto
$?mesto) (zajmy $?zajmy))
=>
   (printout t "Byly zadany informace k teto osobe:" crlf)
   (printout t "Jmeno:" ?jmeno crlf)
   (printout t "Prijmeni:" ?prijmeni crlf)
   (printout t "Vek:" ?vek crlf)
   (printout t "Ulice:" ?ulice crlf)
   (printout t "Mesto:" ?mesto crlf)
   (printout t "Zajmy:" ?zajmy crlf)
)
```

Zkuste si zadat více slov u položky jméno nebo příjmení (tyto položky jsou definovány jako slot - jednopoložkové). Uvidíte, že při zadání např. jména Jan Novák, se Vám vypíše jen slovo Jan.

Funkce explode$

Tato funkce nám umožňuje rozkouskovat víceslovnou položku na samostatné prvky. Kdybyste v našem příkladu readline_explode.clp použili jen readline (tedy byste mohli zadávat víceslovné názvy), pak by i vícepoložková hodnota byla chápána jako hodnota jednopoložková (všimněte si, že při nevyužití explode$ bude např. Ulice: "Manesova 320" v uvozovkách a chápána jako jeden prvek). Proto, abychom z ulice: "Manesova 320" vytvořily dvě položky: Manesova 320, pak využijeme funkci explode$.

Další úlohy k procvičení

Úloha 1.: Země naší planety

1. Na základě informací získaných z tabulky o zemích vytvořte patřičnou šablonu a k ní fakty
2. Dále napište pravidla, pomocí kterých bude moci uživatel vkládat do báze faktů další obdobné záznamy

Název státu	Počet obyvatel	Rozloha	Hospodářství
Kena	23300000	583000	kava kukurice caj
Kamerun	1300000	570000	dobytek nikl
Uganda	16591000	237000	ryby kava dobytek

```
;*****************************************;
;                                         ;
;             ZEME NASI PLATNETY          ;
;               (MH 2004)                 ;
;                                         ;
;*****************************************;
;Zadani:
;vytvorit sablonu
;na ni zalozit fakty
;pravidlo pro zadavani udaju o zemich
;vypis vsech zemi


(deftemplate stat
      (multislot nazev (type SYMBOL))
      (slot pocet_obyv (type NUMBER))
      (slot rozloha (type NUMBER))
      (multislot hospodarstvi (type SYMBOL)))

(deffacts zeme_info
      (stat (nazev Kena) (pocet_obyv 23300000) (rozloha 583000) (hospodarstvi kava
kukurice caj))
      (stat (nazev Kamerun) (pocet_obyv 1300000) (rozloha 570000) (hospodarstvi
dobytek nikl))
      (stat (nazev Uganda) (pocet_obyv 16591000) (hospodarstvi ryby kava dobytek)
(rozloha 237000)))


;Pravidlo pro zadavani udaju o zemich

(defrule zadej_udaje_k_novemu_statu
=>
      (printout t "Zadejte nazev statu:" crlf)
      (bind ?novyS (read))
      (printout t "Zadejte pocet obyvatel statu:" crlf)
      (bind ?novyPO (read))
      (printout t "Zadejte rozlohu statu:" crlf)
      (bind ?novaR (read))
      (printout t "Zadejte hospodarstvi:" crlf)
      (bind ?noveH (readline))
      (assert (stat (nazev ?novyS) (pocet_obyv ?novyPO) (rozloha ?novaR)
(hospodarstvi (explode$ ?noveH))))
      ;vse je nakonec ulozeno do baze faktu
```

```
)

(defrule vypis_vseho
  (declare (salience -1))
  (stat (nazev $?nejaky))
=>
  (printout t "Znam zemi:" ?nejaky crlf)
)
```

Úloha 2.: Zvířectvo

1. Vraťte se k příkladu se zvířaty (lekce 6.) a přepracujte program tak, aby byla fakta se zvířaty založena na šabloně, přitom šablona bude obsahovat vlastnost zachycující počet končetin zvířete.
2. Vypište všechna zvířata, která v databázi jsou
3. Dále vypište jen ta zvířata, která mají jen dvě končetiny

```
;*****************************************;
;                                         ;
;     ZVIRECTVO aneb jak na sablony II.   ;
;              (MH 2004)                  ;
;                                           ;
;*****************************************;


;Sablona pro zvirata

(deftemplate zvire
      (multislot nazev (type SYMBOL))
      (multislot kontinent (type SYMBOL))
      (slot pocet_nohou (type INTEGER)(range 0 4))
)


;Fakta zalozena na sablone zvire

(deffacts zvirena
      (zvire (nazev kocka domaci)(kontinent Evropa Amerika Asie)(pocet_nohou 4))
      (zvire (nazev simpanz maly) (kontinent Evropa Amerika Australie)
(pocet_nohou 2))
      (zvire (nazev kakadu bily) (kontinent Australie Amerika) (pocet_nohou 2))
)


;Nekdy chceme vypsat urcitou informaci jen jednou pro vice dat

(defrule vypis_hlavicku1
      (declare(salience 10))
=>
        (printout t "================================" crlf)
      (printout t "Znam tato zvirata:" crlf)
      (printout t "================================" crlf))


;Pravidlo pro vypis informaci
```

```
(defrule vypis_vse
      (declare (salience 9))
      (zvire (nazev $?nejaky) (kontinent $?k) (pocet_nohou ?x))
=>
      (printout t ?nejaky " " ?k " " ?x crlf))


;Opet hlavicka

(defrule vypis_hlavicku2
=>
      (printout t "--------------------------------" crlf)
      (printout t "Znam dvounohe tyto:" crlf)
      (printout t "--------------------------------" crlf))



;Chceme vypsat jen dvounoha zvirata

(defrule vypis_se_2_k
        (declare (salience -1))
      (zvire(pocet_nohou ?x) (nazev $?y))
      (test(= ?x 2))
=>
      (printout t "Nalezen zaznam pro zvire:" ?y "-pocet koncetin" " " ?x crlf))
```

Úloha 3.: Databáze pro pokročilé (databáze knih)

Pokuste se vytvořit program, který bude umět pracovat s jednoduchou databází knih:

1. definujte šablonu kniha tak, aby sdružovala tyto údaje: název knihy, příjemní a jméno autora, rok vydání, nakladatelství, typ textu, cenu
2. vložte do báze faktů několik faktů vytvořených podle šablony kniha
3. napište pravidlo pro načtení kritéria, podle kterého chce uživatel hledat knihu
4. napište pravidla pro hledání knihy podle kritérií: název knihy, příjmení autora, rok vydání a cena

```
;*****************************************;
;                                 ;
;               KNIHY                   ;
;             (MH 2004)           ;
;                                   ;
;*****************************************;
;Zadani:
;a) definujte sablonu kniha tak, aby sdruzovala tyto udaje: nazev knihy, prijmeni
a jmeno autora, rok vydani, nakladatelstvi, typ textu, cenu
;b) vlozte do baze faktu nekolik faktu vytvorenych podle sablony kniha
;c) napiste pravidlo pro nacteni kriteria, podle ktereho chce uzivatel hledat
knihu
;d) napiste pravidla pro hledani knihy podle kriterii: nazev knihy, prijmeni
autora, rok vydani a cena


;SABLONA ----------------------------------------
(deftemplate kniha
```

```
      (multislot nazev (type SYMBOL))                 ;viceslovne
      (multislot jmeno_prijmeni (type SYMBOL))        ;viceslovne
      (slot rok_vydani (type INTEGER) (range 0 2004))
      (multislot nakladatelstvi (type SYMBOL)) ;viceslovne
      (multislot typ_textu (type SYMBOL))      ;viceslovne
      (slot cena (type INTEGER) (range 1 5000)))


;FAKTA ------------------------------------------
(deffacts pomocne_fakty
      (menu_nacti))
;pomocny fakt pro opetovne zobrazeni menu
;baze faktu s knihami bude uzivatelem postupne plnena



;ZOBRAZENÍ MENU ---------------------------------
(defrule zobraz_menu
?pryc<-(menu_nacti)
=>
      (retract ?pryc)
      (printout t "-----------<MENU>-----------" crlf)
      (printout t "Vlozeni nove knihy ........1" crlf)
      (printout t "Zadejte parametr pro vyhledavani ...2" crlf)
      (printout t "Ukonceni programu .........k" crlf)
      (printout t "----------------------------" crlf)
      (printout t " " crlf)
      (printout t "Zadejte jednu z moznosti:")
      (assert (moje_volba (read))))



;ZADÁNÍ PARAMETRÙ PRO NOVOU KNIHU---------------------------------
(defrule zadej_novou_knihu
?pryc<-(moje_volba 1)
=>
      (retract ?pryc)
      (printout t "Zadejte nazev knihy:" crlf)
      (bind ?n (readline))
      ;funkce pro zadani vice slov najednou (pr. Ja robot)

      (printout t "Zadejte jmeno a prijmeni autora:" crlf)
      (bind ?jmapr (readline))
      (printout t "Zadejte rok vydani knihy do roku 2004:" crlf)
      (bind ?vydani(read))
      (printout t "Zadejte nakladatelstvi:" crlf)
      (bind ?nakl (readline))
      (printout t "Zadejte typ textu:" crlf)
      (bind ?txt (readline))
      (printout t "Zadejte cenu knihy:" crlf)
      (bind ?cena (read))

      (assert(kniha(nazev(explode$ ?n)) (jmeno_prijmeni(explode$ ?jmapr))
(rok_vydani ?vydani) (nakladatelstvi (explode$ ?nakl)) (typ_textu (explode$ ?txt))
(cena ?cena)))
      (assert(menu_nacti)))

;funkce explode$ nam vicehodnotovou polozku, ktera by neuzitim explode$ mela
podobu jednohodnotove polozky, rozdeli na samostatne polozky
```

```
;PODLE CEHO CHCI VYHLEDAT ------------------------------
(defrule zadej_parametr_pro_vyhledavani
      (moje_volba 2)
=>
      (printout t "Zadejte parametr, podle ktereho chcete vyhledat knihu:" crlf)
      (printout t "Zde jsou moznosti: nazev/prijmeni/rok_vydani/cena" crlf)
      (assert(chci_hledat_podle (read))))



;VYHLEDÁVÁNÍ PODLE NAZVU----------------------------
(defrule zadejte_nazev_hledane_knihy
      (chci_hledat_podle nazev)
=>
      (printout t "Zadejte nazev knihy:" crlf)
      (bind ?n (readline))
      (assert (chci_nazev_knihy (explode$ ?n))))


(defrule hledej_podle_nazvu_knihy
      (chci_nazev_knihy $?neco)
      (kniha (nazev $?zacatek $?neco $?konec) (jmeno_prijmeni $?a) (rok_vydani ?b)
(nakladatelstvi $?c) (typ_textu $?d) (cena ?e))
      ;muze se stat, ze nazev polozky je viceslovny napr. Nase zahrada, pak musime
uvazovat o seznamu $?neco a je take mozne, ze
      ;uzivatel nezadal cely nazev knihy (cely je treba: Nase zahrada a rostliny),
pak bychom mohli zajistit vyhledavani tzv.
      ;seznamu v seznamu viz. (nazev $?zacatek $?neco $?konec) tzn. i pri nezadani
celeho nazvu knihy, bude nejaky titul nalezen, pokud je evidovan
=>
      (printout t "Kniha" " " $?neco " " "nalezena s temito udaji:" crlf)
      (printout t ?a"-"?b"-"?c"-"?d"-"?e crlf))


;ZRUŠ VYHLEDÁVÁNÍ -----------------------------------
;potom, co jsme nasli, rusime vse nepotrebne, pro pripad opetovneho zadani stejne
volby a dalsich faktu
(defrule zrus_hledani_podle_nazvu
      (declare(salience -10))
?pryc1<-(moje_volba 2)
?pryc2<-(chci_hledat_podle nazev)
?pryc3<-(chci_nazev_knihy $?x)
=>
      (retract ?pryc1 ?pryc2 ?pryc3)
      (assert(menu_nacti)))




;VYHLEDÁVÁNÍ PODLE PRIJMENI
(defrule zadejte_prijmeni_hledaneho
      (chci_hledat_podle prijmeni)
=>
      (printout t "Zadejte prijmeni autora:" crlf)
      (bind ?p (readline))
      (assert (chci_prijmeni (explode$ ?p))))


(defrule hledej_podle_prijmeni_autora
      (chci_prijmeni $?p)
```

```
      (kniha (nazev $?neco) (jmeno_prijmeni $?neco1 $?p $?neco2)(rok_vydani ?b)
(nakladatelstvi $?c) (typ_textu $?d) (cena ?e))
=>
      (printout t "Pro prijmeni" " " ?p "byly nalezeny knihy/a:" crlf)
      (printout t ?neco"-"?b"-"?c"-"?d"-"?e crlf))


;ZRUŠ VYHLEDÁVÁNÍ PODLE PRIJMENI AUTORA -------------
(defrule zrus_vyhledavani_podle_prijmeni_autora
      (declare(salience -10))
?pryc1<-(moje_volba 2)
?pryc2<-(chci_prijmeni $?p)
?pryc3<-(chci_hledat_podle prijmeni)
=>
      (retract ?pryc1 ?pryc2 ?pryc3)
      (assert(menu_nacti)))




;VYHLEDÁVÁNÍ PODLE ROKU VYDÁNÍ -------------------
(defrule zadejte_rok_vydani_knihy
      (chci_hledat_podle rok_vydani)
=>
      (printout t "Zadejte rok vydani knihy:" crlf)
      (bind ?r (read))
      (assert (chci_rok ?r)))

(defrule hledej_podle_roku_vydani
      (chci_rok ?x)
      (kniha (nazev $?neco) (jmeno_prijmeni $?a) (rok_vydani ?x) (nakladatelstvi
$?c) (typ_textu $?d) (cena ?e))
=>
      (printout t "Pro rok" " " ?x " " "nalezeny tyto udaje:" crlf)
      (printout t ?neco"-"?a"-"?c"-"?d"-"?e crlf))


;ZRUS VYHLEDÁVÁNÍ PODLE ROKU VYDÁNÍ -------------------
(defrule zrus_vyhledavani_podle_roku_vydani
      (declare(salience -10))
?pryc1<-(moje_volba 2)
?pryc2<-(chci_rok ?x)
?pryc3<-(chci_hledat_podle rok_vydani)
=>
      (retract ?pryc1 ?pryc2 ?pryc3)
      (assert(menu_nacti)))




;D - VYHLEDÁVÁNÍ PODLE CENY ---------------------------
(defrule zadejte_cenu_knihy
      (chci_hledat_podle cena)
=>
      (printout t "Zadejte cenu knihy:" crlf)
      (bind ?c (read))
      (assert (chci_cenu ?c)))


(defrule hledej_podle_ceny
      (chci_cenu ?x)
```

```
      (kniha (nazev $?neco) (jmeno_prijmeni $?a) (rok_vydani ?b) (nakladatelstvi
$?c) (typ_textu $?d) (cena ?x))
=>
      (printout t "Pro cenu" " " ?x " " "nalezeny tyto udaje:" crlf)
      (printout t ?neco"-"?a"-"?b"-"?c"-"?d crlf))


;ZRUS VYHLEDÁVÁNÍ PODLE CENY ------------------------
(defrule zrus_vyhledavani_podle_ceny
      (declare(salience -10))
?pryc1<-(moje_volba 2)
?pryc2<-(chci_cenu ?x)
?pryc3<-(chci_hledat_podle cena)
=>
      (retract ?pryc1 ?pryc2 ?pryc3)
      (assert(menu_nacti)))




;UKONCI PROGRAM -------------------------------------
(defrule konec_programu
      (moje_volba k)
=>
      (printout t "Program ukoncen!" crlf))
```

LEKCE 9.

Náplň lekce:

- Procvičení práce se strukturovanými fakty

Úlohy

Úloha 1.

Napište program, který pomůže s výběrem vhodné rostliny k pěstování. Uživatel zadá požadované charakteristiky a program doporučí seznam rostlin, které tyto charakteristiky (každou z nich) mají. Následující tabulka obsahuje charakteristiky 7 rostlin.

Název rostliny	Chlad	Stín	Vlhká půda	Kyselá půda	Město	Květináč	Nenáročná
aukuba	ne	ano	ne	ne	ne	ano	ano
azalka	ano	ano	ano	ano	ne	ano	ne
gardenie	ne	ano	ne	ano	ne	ano	ne
hortenzie	ano	ano	ano	ne	ano	ano	ne
jalovec	ano	ne	ne	ano	ano	ne	ano
oleandr	ne	ne	ano	ne	ano	ano	ano

zimolez ano ne ano ne ano ano ano

```
;****************************************;
;                                ;
;                ROSTLINY              ;
;             (MH 2004)          ;
;                                  ;
;****************************************;
;Zadani:
;1. Napiste program, ktery pomuze s vyberem vhodne rostliny k pestovani.
; Uzivatel zada pozadovane charakteristiky a program doporuci
; seznam rostlin, ktere tyto charakteristiky (kazdou z nich) maji.


;SABLONA, na zaklade ktere budou rostliny definovany (informace o nich budou mit
urcitou strukturu)
(deftemplate rostlina ;NÁZEV RELACE !
      (slot nazev (type SYMBOL) (allowed-symbols aukuba azalka gardenie hortenzie
jalovec oleandr zimolez))
      (slot chlad (type SYMBOL) (allowed-symbols ano ne))
      (slot stin (type SYMBOL) (allowed-symbols ano ne))
      (slot vlhka_puda (type SYMBOL) (allowed-symbols ano ne))
      (slot kysela_puda (type SYMBOL) (allowed-symbols ano ne))
      (slot mesto (type SYMBOL) (allowed-symbols ano ne))
      (slot kvetinac (type SYMBOL) (allowed-symbols ano ne))
      (slot nenarocna (type SYMBOL) (allowed-symbols ano ne)))


;SABLONA hledane rostliny
(deftemplate hledana_rostlina ;NÁZEV RELACE !
      (slot chlad (type SYMBOL) (allowed-symbols ano ne))
      (slot stin (type SYMBOL) (allowed-symbols ano ne))
      (slot vlhka_puda (type SYMBOL) (allowed-symbols ano ne))
      (slot kysela_puda (type SYMBOL) (allowed-symbols ano ne))
      (slot mesto (type SYMBOL) (allowed-symbols ano ne))
      (slot kvetinac (type SYMBOL) (allowed-symbols ano ne))
      (slot nenarocna (type SYMBOL) (allowed-symbols ano ne)))


;BAZE FAKTU
(deffacts info_rostliny
      (rostlina (nazev aukuba) (chlad ne) (stin ano) (vlhka_puda ne) (kysela_puda
ne) (mesto ne) (kvetinac ano) (nenarocna ano))
      (rostlina (nazev azalka) (chlad ano) (stin ano) (vlhka_puda ano)
(kysela_puda ano) (mesto ne) (kvetinac ano) (nenarocna ne))
      (rostlina (nazev gardenie) (chlad ne) (stin ano) (vlhka_puda ne)
(kysela_puda ano) (mesto ne) (kvetinac ano) (nenarocna ne))
      (rostlina (nazev hortenzie) (chlad ano) (stin ano) (vlhka_puda
ano)(kysela_puda ne) (mesto ano) (kvetinac ano) (nenarocna ne))
      (rostlina (nazev jalovec) (chlad ano) (stin ne) (vlhka_puda ne) (kysela_puda
ano) (mesto ano) (kvetinac ne) (nenarocna ano))
      (rostlina (nazev oleandr) (chlad ne) (stin ne) (vlhka_puda ano) (kysela_puda
ne) (mesto ano) (kvetinac ano) (nenarocna ano))
      (rostlina (nazev zimolez) (chlad ano) (stin ne) (vlhka_puda ano)
(kysela_puda ne) (mesto ano) (kvetinac ano) (nenarocna ano)))

(deffacts pomocny_fakt
      (dalsi_hledani))
```

```
(defrule zadejte_charakteristiky
?pryc<-(dalsi_hledani)
=>
      (retract ?pryc)
      (printout t " " crlf)
      (printout t "Zadejte následující požadavky na rostliny...ANO/NE" crlf)
      (printout t "================================================" crlf)
      (printout t "CHLAD:")
      (bind ?o1 (read))
      (printout t "+")
      (printout t "STIN:")
      (bind ?o2 (read))
      (printout t "+")
      (printout t "VLHKA PUDA:")
      (bind ?o3 (read))
      (printout t "+")
      (printout t "KYSELA PUDA:")
      (bind ?o4 (read))
      (printout t "+")
      (printout t "MESTO:")
      (bind ?o5 (read))
      (printout t "+")
      (printout t "KVETINAC:")
      (bind ?o6 (read))
      (printout t "+")
      (printout t "NENAROCNA:")
      (bind ?o7 (read))

      (assert (hledana_rostlina (chlad ?o1) (stin ?o2) (vlhka_puda ?o3)
(kysela_puda ?o4) (mesto ?o5) (kvetinac ?o6) (nenarocna ?o7))))
      ;pozadavky uzivatele na rostinu budou ulozeny do baze faktu (info o hledane
rostline definovany sablonou hledana_rostlina)


(defrule zjisti_vhodnou_rostlinu
      (hledana_rostlina (chlad ?o1) (stin ?o2) (vlhka_puda ?o3) (kysela_puda ?o4)
(mesto ?o5) (kvetinac ?o6) (nenarocna ?o7))
      ;vime tedy, ze uzivatel jiz zadal pozadavky na rostlinu

      (rostlina (nazev ?x)(chlad ?o1) (stin ?o2) (vlhka_puda ?o3) (kysela_puda
?o4) (mesto ?o5) (kvetinac ?o6) (nenarocna ?o7))
      ;zde porovnavame obsahy promennych u pozadavku uzivatele s promennymi, ktere
jsou soucasti faktu v nasi bazi
      ;vsimnete si stejnych nazvu promennych v prvni a druhe podmince !!!
=>
      (printout t " " crlf)
      (printout t "NALEZENA rostlina:" ?x crlf))



(defrule rostlina_nenalezena
      (hledana_rostlina (chlad ?o1) (stin ?o2) (vlhka_puda ?o3) (kysela_puda ?o4)
(mesto ?o5) (kvetinac ?o6) (nenarocna ?o7))
      (not(rostlina (nazev ?x)(chlad ?o1) (stin ?o2) (vlhka_puda ?o3) (kysela_puda
?o4) (mesto ?o5) (kvetinac ?o6) (nenarocna ?o7)))
=>
      (printout t " " crlf)
      (printout t "Vasim pozadavkum NEVYHOVUJE zadna rostlina !" crlf))


;Pro pripad, ze by uzivatel chtel dale hledat rostlinu, zrusime to stare (tu
```

```
minule hledanou rostlinu)
(defrule zrus_hledanou
      (declare (salience -1))
?pryc<-(hledana_rostlina (chlad ?o1) (stin ?o2) (vlhka_puda ?o3) (kysela_puda ?o4)
(mesto ?o5) (kvetinac ?o6) (nenarocna ?o7))
=>
      (retract ?pryc))


(defrule chcete_hledat_dalsi
      (declare(salience -2))
      (not (hledana_rostlina (chlad ?o1) (stin ?o2) (vlhka_puda ?o3) (kysela_puda
?o4) (mesto ?o5) (kvetinac ?o6) (nenarocna ?o7)))
      ;hledani dalsi rostliny bude mozne jen, kdyz budeme mit tzv. ciste platno
(zrusena minula hledana rostlina - hledame znovu)
=>
      (printout t "---------------------------------------" crlf)
      (printout t "Chcete hladat dalsi rostliny ano/ne ?" crlf)
      (bind ?neco (read))
      (assert (hledat_dalsi ?neco)))



(defrule hledat_dalsi_ano
?pryc<-(hledat_dalsi ano)
=>
      (retract ?pryc)
      (assert (dalsi_hledani)))



(defrule hledat_dalsi_ne
?pryc<-(hledat_dalsi ne)
=>
      (retract ?pryc)
      (printout t "*************************************** " crlf)
      (printout t "KONEC PROGRAMU !!!" crlf))
```

Úloha 2.:

Napište program, který bude evidovat planety naší sluneční soustavy.

Potřebné údaje o planetách jsou zachyceny v tabulce.

Název	Rovníkový průměr /km/	Min. teplota	Max. teplota	Počet družic	Složení atmosféry
Merkur	4878	-183	450	0	žádná
Venuše	12104	-100	480	0	mlha, mraky, oxid uhličitý
Země	12756	-100	50	1	dusík, kyslík
Mars	6794	-130	5	2	oxid uhličitý, prach
Jupiter	142800	-200	5	16	vodík, čpavek, sirník amonitý, vodní led, vodní kapky
Saturn	120000	-220	-10	22	prach, čpavek, sirník amonitý,vodní led

Uran	50800	-300	-100	15	vodík, hélium,metan
Neptun	48600	-300	-50	8	mraky
Pluto	2300	-350	-200	1	voda, metan, led

1. napište pravidlo pro výpis všech evidovaných planet (stačí jen název planety)
2. napište pravidlo, kterým zjistíte planety bez atmosféry
3. napište pravidlo, kterým zjistíte jestli existuje nějaká planeta bez družice. Tyto planety uložte do samostatného seznamu. Každou z bezdružicových planet vypište na samostatný řádek (ne celý seznam najednou)
4. pokuste se také napsat pravidlo, kterým zjistíte planety, u kterých je rozsah teplot v tomto intervalu <-100;+100> stupňů Celsia
5. ... a nakonec vytvořte pravidlo, které Vám pomůže určit, která planeta má v atmosféře vodík, ale bez metanu

```
;*****************************************;
;                                         ;
;                  PLANETY                ;
;              (MH 2005)                  ;
;                                         ;
;*****************************************;
;Zadani:
;a)   napište pravidlo pro výpis všech evidovaných planet (staèí jen název
planety)
;b)   napište pravidlo, kterým zjistíte planety bez atmosféry
;c)   napište pravidlo, kterým zjistíte jestli existuje nejaká planeta bez
družice. Tyto planety uložte do samostatného seznamu. ;Každou z bezdružicových
planet vypište na samostatný rádek (ne celý seznam najednou)
;d)   pokuste se také napsat pravidlo, kterým zjistíte planety, u kterých je
rozsah teplot v tomto intervalu <-100;+100> stupòù Celsia
;e)   a nakonec vytvoøte pravidlo, které Vám pomuže urcit, která planeta má v
atmosfére vodík, ale bez metanu



(deftemplate planeta
  (slot nazev (type SYMBOL) (allowed-symbols Merkur Venuse Zeme Mars Jupiter
Saturn Uran Neptun Pluto))
  (slot rovnik_prumer (type INTEGER) (range 1000 160000))
  (slot teplota_min (type INTEGER))
  (slot teplota_max (type INTEGER))
  (slot druzice (type INTEGER))
  (multislot atmosfera (type SYMBOL))
)

(deffacts info_o_planetach
  (planeta (nazev Merkur) (rovnik_prumer 4878) (teplota_min -183) (teplota_max
450) (druzice 0) (atmosfera zadna))
  (planeta (nazev Venuse) (rovnik_prumer 12104) (teplota_min -100) (teplota_max
480) (druzice 0) (atmosfera mlha mraky oxid_ uhlicity))
  (planeta (nazev Zeme) (rovnik_prumer 12756) (teplota_min -100) (teplota_max 50)
(druzice 1) (atmosfera dusik kyslik))
  (planeta (nazev Mars) (rovnik_prumer 6794) (teplota_min -130) (teplota_max 5)
(druzice 2) (atmosfera oxid_uhlicity prach))
  (planeta (nazev Jupiter) (rovnik_prumer 142800) (teplota_min -200) (teplota_max
5) (druzice 16) (atmosfera vodik cpavek, sirnik_amonity vodni_led vodni_kapky))
  (planeta (nazev Saturn) (rovnik_prumer 120000) (teplota_min -220) (teplota_max -
10) (druzice 22) (atmosfera prach cpavek sirnik_amonity vodni_led))
```

```
  (planeta (nazev Uran) (rovnik_prumer 50800) (teplota_min -300) (teplota_max -
100) (druzice 15) (atmosfera vodik helium metan))
  (planeta (nazev Neptun) (rovnik_prumer 48600) (teplota_min -300) (teplota_max -
50) (druzice 8) (atmosfera mraky))
  (planeta (nazev Pluto) (rovnik_prumer 2300) (teplota_min -150) (teplota_max -38)
(druzice 1) (atmosfera voda metan led))
)


;stanoveno pro ulohu c.
(deffacts seznam
  (bez_druzic)
)



(defrule vypis_hlavicku
=>
  (printout t ">>>PLANETY<<< " crlf)
  (printout t " " crlf)
  (printout t "*" crlf)
  (printout t "**" crlf)
  (printout t "***" crlf)
  (printout t " " crlf)
)


(defrule vypis_zname_planety
  (planeta (nazev ?nejaka))
=>
  (printout t "---------------------------------------" crlf)
  (printout t "Znama planeta:" ?nejaka crlf)
)


(defrule planety_bez_atmosfery
  (planeta (nazev ?nejaka) (atmosfera zadna))
=>
  (printout t "---------------------------------------" crlf)
  (printout t "Planeta bez atmosfery:" ?nejaka crlf)
)


(defrule planety_bez_druzice
  (planeta (nazev ?nejaka) (druzice 0))
  ?pryc<-(bez_druzic $?seznam)
  (not (bezdruzicova_pridana ?nejaka))
  ;zabranujeme zacykleni programu, tak ze davame Clipsu najevo, ktere planety jiz
byly
  ;do seznamu pridany, aby nemel tendeci vkladat do seznamu stejne planety
=>
  (retract ?pryc)
  (assert (bez_druzic $?seznam ?nejaka))
  (assert (bezdruzicova_pridana ?nejaka))
)


(defrule vypis_bezdruzicovych_planet
  (declare (salience -1))
  (bez_druzic $?zacatek ?planeta $?konec)
  (not (bezdruzicova_vypsana ?planeta))
```

```
  ;zajisteni vypisu na radky
=>
  (printout t "-------------------------------------" crlf)
  (printout t "Bezdruzicova planeta je:" ?planeta crlf)
  (assert (bezdruzicova_vypsana ?planeta))
)


(defrule planety_s_teplotnim_intervalem
  (declare (salience -2))
  (planeta (nazev ?planetaX) (teplota_min ?tmin&: (>= ?tmin -100)) (teplota_max
?tmax&: (<= ?tmax 100)))
=>
  (printout t "-------------------------------------" crlf)
  (printout t "Planeta" " " ?planetaX " " "ma rozsah teplot v intervalu od -100 do
+100 stupnu Celsia." crlf)
)

(defrule planety_s_vodikem_a_ne_metanem
  (declare (salience -3))
  (planeta (nazev ?nejaka) (atmosfera $?zacatek1 vodik $?konec1))
  (not(planeta (nazev ?nejaka) (atmosfera $?zacatek2 metan $?konec2)))
=>
  (printout t "-------------------------------------" crlf)
  (printout t "Planeta s vodikem, ale bez metanu:" ?nejaka crlf)
  (printout t "-------------------------------------" crlf)
)
```

LEKCE 10.

Náplň lekce:

- Příkaz SUBSETP
- Úloha k procvičení

Příkaz SUBSETP

Tento příkaz slouží pro další možnou manipulaci se seznamy. Dovoluje nám vyhledávat tzv. seznam v seznamu.

Syntaxe: (subsetp $?seznam_1 $?seznam_2)

Pokud má funkce subsetp vrátit hodnotu TRUE, pak všechny hodnoty seznamu seznam_1 mají být obsaženy v seznamu seznam_2. Tedy seznam_1 je jakousi podmnožinou seznamu seznam_2.

Rozdíl mezi member$ a subsetp

U funkce member$ vyhledáváme jednuhodnotovou proměnnou ve vícehodnotové:

?moje_zvire = papousek
$?zvirata = pes kocka kolibrik papousek

```
(member$ ?moje_zvire $?zvirata)
```

U funkce subsetp vyhledáváme vícehodnotovou proměnnou ve vícehodnotové:

$?moje_filmy = Robots Shrek Kruh Valiant
$?filmy_videopujcovny = Valka_svetu Shrek Doba_ledova Mise_na_Mars Kruh

```
(subsetp $?moje_filmy $?filmy_videopujcovny)
```

Příklady:

1. MÁME DEFINOVÁNY TYTO DVA SEZNAMY:

... a je definován příkaz subsetp takto:

(subsetp $?seznam_1 $?seznam_2) => TRUE

Všechny prvky seznamu_1 jsou obsaženy v seznamu_2 - proto TRUE

2. DÁLE MÁME DEFINOVÁNY TYTO DVA SEZNAMY:

... a je definován příkaz subsetp takto:

(subsetp $?seznam_1 $?seznam_2) => FALSE

Všechny prvky seznamu_1 nejsou obsaženy v seznamu_2 - proto FALSE

Úloha k procvičení - Baktérie

Zadání:

Baktérie je možné klasifikovat pomocí několika charakteristik, jako je základní tvar (kulička, tyčinka, vlákno, spirála), výsledek laboratorního testu (pozitivní, negativní) a jestli potřebují ke svému přežití kyslík (aerobní=potřebují nebo anaerobní=nepotřebují). Napište program, který identifikuje bakterii na základě informací v níže uvedené tabulce. Program se uživatele zeptá na tvar, výsledek laboratorního testu a potřebnost kyslíku pro bakterii.

Typ	Tvar	Laboratorní test	Nároky na kyslík
actinomyceta	tyčinka NEBO vlákno	pozitivní	aerobní
kok	kulička	pozitivní	aerobní A anaerobni
corynebakterie	tyčinka	pozitivní	aerobní
endospora	tyčinka	pozitivní NEBO negativní	aerobní A anaerobní
střební baktérie	tyčinka	negativní	aerobní
plísňová bakterie	kulička	žádný	aerobní
mycoplazma	kulička	žádný	aerobní
pseudomonáda	tyčinka	negativní	aerobní
rickettsia	kulička NEBO tyčinka	negativní	aerobní
zapouzdřilka	vlákno	negativní	aerobní
spirila	spirála	negativní	aerobní
spirocheta>	spirála	negativní	anaerobní
vibria	tyčinka	negativní	aerobní

Poznámky

1. tvar: shoda ve tvaru mezi pozorovanou a některou konkrétní bakterií nastane tehdy, jestliže se všechny hodnoty tvaru pozorované bakterie nachází ve výčtu možných hodnot tvaru známé baktérie.

Uživatel zadá: (tycinka) - pak může jít o actinomycetu, corynebakterii, rickettsiu, endosporu atd.
Uživatel zadá: (tycinka vlakno) - pak může jít o actinomycetu, ale ne o corynebakterii (ta může nabývat jen tvaru tyčinka, ale ne vlákna !)

2. laboratorní test: výsledek laboratorního testu je vždy JEDNOZNAČNÝ, proto u pozorované baktérie nabývá vlastnost lab. test. vždy přávě jednu hodnotu (proto slot a ne multislot). Jedná se o podobné zacházení s hodnotami jako v případě tvaru. Výsledek lab. testu může být pozitivní nebo negativní (ale ne oba zároveň). Tedy shoda ve výsledku lab. testu nastane tehdy, jestliže (jediná !) hodnota výsledku lab. testu u pozorované baktérie se nachází ve výčtu možných hodnot této vlastnosti u některé konkrétní baktérie.

Uživatel zadá: (negativní) - pak může jít o endosporu, střevní baktérii, pseudomonádu atd.
Uživatel zadá: (pozitivní negativní) - pak může jít o endosporu, ale ne o koka či spirilu (ty mohou nabývat jen jedné hodnoty)

3. nároky na kyslík: tato vlastnost může nabývat více hodnot NAJEDNOU. Shoda nastane tehdy, jestliže hodnoty této vlastnosti u pozorované baktérie přesně odpovídají hodnotám této vlastnosti nějaké konkrétní baktérie (tj. žádná hodnota nechybí ani nepřebývá !)

Uživatel zadá: (aerobní) - pak může jít o actinomycetu, corynebakterii, ale ne o koka (ten může nabývat i hodnoty anaerobní)
Uživatel zadá: (aerobní anaerobní) - pak může jít o koka či endosporu, ale už ne o mycoplazmu atd.

Detail: Pravidlo vyhledej_bakterii

```
(defrule vyhledej_bakterii
   (hledana_bakterie (tvar $?tvarmuj) (lab_test ?testmuj) (kyslik $?kyslikmuj))
   (bakterie (nazev ?jmeno) (tvar $?t & :(subsetp $?tvarmuj $?t)) (lab_test $?lt &
:(member$ ?testmuj $?lt))
(kyslik $?k & :(subsetp $?kyslikmuj $?k) & :(subsetp $?k $?kyslikmuj)))
=>
   (printout t "Vasim pozadavkum vyhovuje bakterie:" ?jmeno crlf))
```

1. podmínka: soubor požadavků uživatele
2. podmínka: to co uživatel požaduje je porovnáváno s bází faktů (s fakty o bakteriích)

Vysvětlení jednotlivých částí 2. podmínky

```
***(tvar $?t & :(subsetp $?tvarmuj $?t)) (lab_test $?lt & :(member$ ?testmuj
$?lt))
```

Tato část druhé podmínky využívá příkazu jak subsetp tak member. Musíme se řídit tím, jestli je pro daný tvar možné definovat logickou spojku "NEBO" či "A". Ve sloupci pro tvar vidíme, že se jedná o spojku "NEBO". Tedy budeme vyhledávat seznam v seznamu (příkaz subsetp). Totéž platí i pro laboratorní test s tím rozdílem, že test je definován jako slot - kvůli jednoznačnosti lab. testů.

```
***(kyslik $?k & :(subsetp $?kyslikmuj $?k) & :(subsetp $?k $?kyslikmuj)
```

Část 2. podmínky, týkající se kyslíku, je složitější. Pro kyslík u některých baktérií platí logická spojka "A".

`(kyslik $?k & :(subsetp $?kyslikmuj $?k)`: musíme také zajistit, aby se nám vyhledával seznam v seznamu.

Např. uživatel zadá: (aerobní) - pak má být vybrána např. baktérie actinomyceta (ta vybrána bude - to zajistíme právě touto částí podmínky).

Jak ale zajistit, aby nebyla vybrána např. baktérie kok? To je docíleno druhou části kódu: `(subsetp $?k $?kyslikmuj)`. Zde na to jdeme opačně než u první části 2. podmínky pro kyslík. Seznam s informacemi o kyslíku, který je napevno definován v bázi faktů, musíme porovnat se seznamem od uživatele (s jeho požadavky na baktérii). U baktérie kok to bude takto: vezmeme si seznam pro kyslik - z báze faktů (zde je aerobni A anaerobni) a tento seznam ma byt podmnozinou a to seznamu od uzivatele. Ale při zadání vlastnosti aerobni tomu díky této části podmínky není. (aerobni anaerobni) --> není podmnožinou (aerobni). Navíc je zde vlastnost anaerobni. Proto nebude vybrána baktérie s názvem kok.

```
;*********************************************
;
;            Bakterie - funkce subsetp
;
```

```
;               (MH 2004)
;*********************************************
;Zadani: Podle pozadavku uzivatele naleznete prislusny
;druh bakterie


(deftemplate bakterie
      (slot nazev (type SYMBOL))
      (multislot tvar (type SYMBOL))
      (multislot lab_test(type SYMBOL))
      (multislot kyslik (type SYMBOL)))


(deftemplate hledana_bakterie
      (multislot tvar (type SYMBOL))
      (slot lab_test(type SYMBOL))
      (multislot kyslik (type SYMBOL)))

;slot u lab_testu je z toho duvodu, ze uzivatel
;ktery hleda info k bakterii vi presne, jestli byl
;u ni test poz. ci neg. (je zde jen jedna moznost)
;ale pritom nektera bakterie sama muze nabyvat ruznych
;vysledku (poz. ci neg. - ruzne testy) - proto multislot u
;sablony bakterie


(deffacts info_bakterie
      (bakterie (nazev actinomyceta)
              (tvar tycinka vlakno)
              (lab_test pozitivni)
              (kyslik aerobni))
      (bakterie (nazev kok)
              (tvar kulicka)
              (lab_test pozitivni)
              (kyslik aerobni anaerobni))
      (bakterie (nazev corynebakterie)
              (tvar tycinka)
              (lab_test pozitivni)
              (kyslik aerobni))
      (bakterie (nazev endospora)
              (tvar tycinka)
              (lab_test pozitivni negativni)
              (kyslik aerobni anaerobni))
      (bakterie (nazev zapouzdrilka)
              (tvar vlakno)
              (lab_test negativni)
              (kyslik aerobni))
      (bakterie (nazev spirocheta)
              (tvar spirala)
              (lab_test negativni)
              (kyslik anaerobni))
      (bakterie (nazev rickettsia)
              (tvar kulicka tycinka)
              (lab_test negativni)
              (kyslik aerobni))
      (bakterie (nazev plisnova_bakterie)
              (tvar kulicka)
              (lab_test zadny)
              (kyslik aerobni)))

(defrule zadejte_nazev_bakterie
```

```
=>
      (printout t "Zadejte tvar bakterie / maximalne 2 tvary !" crlf)
      (bind ?x (readline))
      (printout t "Zadejte vysledek laboratorniho testu --
pozitivni/negativni/zadny:" crlf)
      (bind ?y (read))
      (printout t "Zadejte pro bakterii naroky na kyslik -- aerobni/anaerobni:"
crlf)
      (bind ?z (readline))
      (assert (hledana_bakterie (tvar (explode$ ?x)) (lab_test ?y) (kyslik
(explode$ ?z)))))


(defrule vyhledej_bakterii
      (hledana_bakterie (tvar $?tvarmuj) (lab_test ?testmuj) (kyslik $?kyslikmuj))
      (bakterie (nazev ?jmeno) (tvar $?t&:(subsetp $?tvarmuj $?t)) (lab_test
$?lt&:(member$ ?testmuj $?lt))
                        (kyslik $?k&:(subsetp $?kyslikmuj $?k)&:(subsetp $?k
$?kyslikmuj)))

=>
      (printout t "Vasim pozadavkum vyhovuje bakterie:" ?jmeno crlf))

(defrule nenalezena_bakterie
      (hledana_bakterie (tvar $?tvarmuj) (lab_test ?testmuj) (kyslik $?kyslikmuj))
      (not(bakterie (nazev ?jmeno) (tvar $?t&:(subsetp $?tvarmuj $?t)) (lab_test
$?lt&:(member$ ?testmuj $?lt))
                        (kyslik $?k&:(subsetp $?kyslikmuj $?k)&:(subsetp $?k
$?kyslikmuj))))

=>
      (printout t "Nenalezena zadna odpovidajici bakterie !" crlf))
```