# سیستم های خبره
# محیط برنامه نویسی سیستم های خبره CLIPS

دکتر سیامک سرمدی

## اجرای دستورات

- خط دستور محیط نرم افزار به شکل زیر است.

```
CLIPS>
```

- برای خارج شدن از محیط از طریق دستور از عبارت زیر استفاده کرده و یا از منوی فایل exit را انتخاب می کنیم. توجه داشته باشید که دستورات CLIPS معمولا با پرانتز احاطه شده ند.

```
CLIPS> (exit)
```

- برای اجرای مثالهای CLIPS، ابتدا از منوی فایل، فایل مورد نظر را از دایرکتوری مثالها load نموده و سپس حقایق قدیمی را از محیط CLIPS با استفاده از منوی command->reset و یا با دستور (reset) خالی می کنیم و سپس با منوی command -> run یا دستور (run) برنامه را اجرا می کنیم.

```
CLIPS> (reset)
CLIPS> (run)
```

## حقایق (facts)

- برای قرار دادن داده در لیست حقایق (Facts list) از دستور assert استفاده می نمائیم:

```
CLIPS> (assert (duck))
<Fact-1>
```

- به شرطی که محیط کاری قبل از اجرای این دستور ریست شده باشد، حقیقت فوق با شماره 1 در لیست حقایق (حافظه کاری یا working memory) ذخیره می شود. شماره 0 به حقیقت اولیه (حقیقتی خالی) اختصاص دارد. برای مشاهده حقایق موجود در حافظه کاری از دستور (facts) استفاده کنید.

```
CLIPS> (facts)
f-0     (initial-fact)
f-1     (duck)
For a total of 2 facts.
CLIPS>
```

## حقایق (facts)

- مشخصه های f0 و f1 مشخصه حقایق نامیده شده و عدد استفاده شده در هر مشخصه "اندیس" (index) آن حقیقت نامیده می شود.

- در صورتیکه یک حقیقت را بصورت تکراری وارد نمائید CLIPS گزارش عدم موفقیت در اضافه کردن آن حقیقت را خواهد داد که به شکل کلمه FALSE است.

```
CLIPS> (assert (duck))
FALSE
CLIPS> (facts)
f-0     (initial-fact)
f-1     (duck)
For a total of 2 facts.
CLIPS>
```

- البته می توان هر تعداد حقیقت متفاوت را در حافظه کاری قرار داد.

## حقایق (facts)

- مثلا با دستور زیر مجموعا 3 حقیقت خواهیم داشت.

```
CLIPS> (assert (cat))
<Fact-2>
CLIPS> (facts)
f-0     (initial-fact)
f-1     (duck)
f-2     (cat)
For a total of 3 facts.
CLIPS>
```

- می توان حافظه کاری (حقایق) را با دستور clear خالی کرد. البته دستور clear علاوه بر حافظه کاری قواعد را هم از حافظه پایگاه دانش خالی می کند.

```
CLIPS> (clear)
CLIPS> (facts)
f-0     (initial-fact)
For a total of 1 fact.
CLIPS>
```

## حقایق (facts)

- امکان اضافه کردن چند حقیقت بطور همزمان نیز وجود دارد.

```
CLIPS> (clear)
CLIPS> (assert (a) (b) (c))
<Fact-3>
CLIPS> (facts)
f-0     (initial-fact)
f-1     (a)
f-2     (b)
f-3     (c)
For a total of 4 facts.
CLIPS>
```

## حقایق (facts)

- برای مشاهده حقایق از یک ایندکس به بعد شماره شروع را با دستور (facts) می دهیم.

```
CLIPS> (facts 1)
f-1     (a)
f-2     (b)
f-3     (c)
For a total of 3 facts.
CLIPS> (facts 2)
f-2     (b)
f-3     (c)
For a total of 2 facts.
CLIPS>
```

## حقایق (facts)

- فرض کنید حقایق زیر را (7 حقیقت) داریم:

f-0: (initial-fact), f-1: (a), f-2: (b), f-3: (c), f-4: (d), f-5: (e), f-6: (f)

- اگر تعداد زیادی حقیقت داشته باشیم محدوده ای از اندیکس ها را به شکل زیر می بینیم :

```
CLIPS> (facts 2 4)
f-2     (b)
f-3     (c)
f-4     (d)
For a total of 3 facts.
CLIPS>
```

- یا برای دیدن تعداد حقیقت مشخص در یک محدوده (یک حقیقت در محدوده 2 و 4):

```
CLIPS> (facts 2 4 1)
f-2     (b)
For a total of 1 fact.
```

## حقایق (facts)

- حقایقی که تا به حال تعریف کردیم حقایق تک فیلدی (single field facts) هستند. و در ضمن تنها فیلد این حقایق، بدون نام (unnamed) هستند.
- اگر بخواهیم حقیقتی را با دو فیلد بی نام بسازیم به شکل زیر عمل می کنیم:

```
CLIPS> (assert (brian duck))
<Fact-1>
CLIPS> (facts)
f-0     (initial-fact)
f-1     (brian duck)
For a total of 2 facts.
CLIPS>
```

- توجه کنید که حقیقت 1، دوفیلد با مقادیر brian وduck دارد که که هر دو بی نام هستند. ترتیب فیلدها درون حقایق مهم است. مثلا حقیقت فوق ممکن است به این معنی باشد که brian ، duck را شکار می کند.

## حقایق (facts)

- معمولا بهتر است در صورتیکه بیش از یک فیلد در یک حقیقت داریم (مثل ali hassan)، رابطه بین آنها را در ابتدای حقیقت با یک فیلد مشخص کنیم (parent).

```
CLIPS> (assert (parent ali hasan))
<Fact-2>
CLIPS> (facts 2)
f-2     (parent ali hasan)
For a total of 1 fact.
CLIPS>
```

- هرکدام از fact های فوق را می توان به عنوان یک لیست از فیلد ها در نظر گرفت. یک لیست می تواند دارای ترتیب (ordered) و یا بدون ترتیب (unordered) در نظر گرفته شود.

## حقایق (facts)

- اگر بخواهیم فیلدی با مقدار خالی ایجاد کنیم از کلمه nil استفاده می کنیم.

```
CLIPS> (assert (duck nil))
<Fact-3>
CLIPS> (facts 3)
f-3     (duck nil)
For a total of 1 fact.
CLIPS>
```

- توجه داشته باشید که یک فیلد خالی با عدم وجود فیلد متفاوت است. برای مثال حقیقت فوق یک حقیقت 2 فیلدی است. در حالیکه اگر کلمه nil حذف می شد، یک حقیقت تک فیلدی ساخته می شد.

- هر فیلد می تواند نوع های مختلفی داشته باشد ، از جمله: float, integer, symbol, string, external-address, fact-address, instance-name و instance-address . در فیلدهای بی نام نوع فیلد بطور اتوماتیک بر حسب مقدار داده شده تنظیم می شود. درمثالهای بالا نوع فیلدها symbol خواهد بود.

## حقایق (facts)

- تعریف حقایق بصورت تو درتو مجاز نیست. مثلا تعریف زیر قابل قبول نمیباشد.

```
(duck (shot Brian Gary Rey))
```

- البته بعدا خواهیم دید که با استفاده از تعریف الگوها (deftemplate) این تعریف می تواند صحیح باشد (به شرطی که shot بعنوان نام فیلد تعریف گردد).

- Clips یک زبان حساس به نوع حروف (بزرگ و کوچک) است. علاوه بر این بعضی علائم کاربرد خاص دارند:

```
$ ? ; ~ > | & ( ) "
```

- سه علامت & | ~ را نباید در نامگذاری علامت ها (symbols) مثلا حقایق استفاده کرد.

## فیلد ها

- **فیلد های علامت (symbol):** علامت، نوعی فیلد است که با یک یا چند کارکتر ASCII غیر از مقادیری که مستثنی شدند ساخته شده. مانند حقایقی که در ابتدا ساخته شدند.

(cat)

(brian duck)

(duck-shot brian duck)

- **فیلد های رشته ای:** هر رشته ای که بین " " قرار بگیرد رشته محسوب می شود. توجه کنید که خود quote ها هم جزو رشته هستند.

- **فیلد عددی:** می تواند یا اعشاری و یا صحیح باشد. نوع فیلد با توجه به مقدار داده شده به آن تعیین می شود. در نسخه های قبل از v6 امکان ساخت حقایقی که فقط از یک فیلد عددی تشکیل شده باشند ممکن بود. ولی از نسخه 6 به بعد حداقل اولین فیلد یک حقیقت که شامل اعداد است باید از نوع Symbol باشد.

(x 1.5) یا (z 65)

## فیلد های عددی

- مثالهای زیر حقایقی با فیلد های عددی را می سازند.

```
CLIPS> (clear)
CLIPS> (assert (number 1))
<Fact-1>
CLIPS> (assert (x 1.5))
<Fact-2>
CLIPS> (assert (y -1))
<Fact-3>
CLIPS> (assert (z 65))
<Fact-4>
CLIPS> (assert (distance 3.5e5))
<Fact-5>
```

```
CLIPS> (assert (coordinates 1 2 3))
<Fact-6>
CLIPS> (assert (coordinates 1 3 2))
<Fact-7>
CLIPS> (facts)
f-0     (initial-fact)
f-1     (number 1)
f-2     (x 1.5)
f-3     (y -1)
f-4     (z 65)
f-5     (distance 350000.0)
...
```

- در مثال قبل حقایقی با چند فیلد ساخته شدند. همانطورکه قبلا اشاره شد بطور پیش فرض ترتیب فیلد های یک حقیقت اهمیت دارد. به مثال زیر توجه کنید:

```
CLIPS> (assert (coordinates 1 2 3))
<Fact-1>
CLIPS> (assert (coordinates 1 3 2))
<Fact-2>
CLIPS> (facts)
f-0     (initial-fact)
f-1     (coordinates 1 2 3)
f-2     (coordinates 1 3 2)
For a total of 3 facts.
CLIPS>
```

- در مثال فوق دو واقعیت تقریبا شبیه هم هستند، ولی ترتیب متفاوت فیلد های عددی 2 و 3 باعث شده که دو حقیقت متمایز داشته باشیم (اگر این دو از نظر CLIPS یکی بودند، تنها یکی از آنها ذخیره می شد).

- **جدا کردن فیلد ها در یک حقیقت:** فیلد های یک حقیقت را معمولا با فاصله جدا سازی می نمایند. برای افزایش خوانایی میتوان از خط جدید و فاصله های بیشتر استفاده کرد. اما توجه داشته باشید که خط جدید در میان یک رشته نباید صورت بگیرد (رشته شامل quote ها نیز می باشد) و تنها در فاصله بین فیلدها مجاز است.

```
CLIPS> (clear)
CLIPS> (assert (The
                duck
                says
                "Quack"))
<Fact-1>
CLIPS> (facts)
f-0     (initial-fact)
f-1     (The duck says "Quack")
For a total of 2 facts.
CLIPS>
```

- بنابراین حقیقت چند خطی فوق معادل حقیقت یک خطی زیر است:

```
CLIPS> (assert (The duck says "Quack"))
```

- **حذف حقیقت ها:** حذف حقایق با دستور retract (با ذکر اندیس یک حقیقت) انجام می گیرد

```
CLIPS> (clear)
CLIPS> (assert (x y z))
<Fact-1>
CLIPS> (assert (l m n))
<Fact-2>
CLIPS> (assert (a b c))
<Fact-3>
CLIPS> (facts)
f-0     (initial-fact)
f-1     (x y z)
f-2     (l m n)
f-3     (a b c)
For a total of 4 facts.
CLIPS> (retract 2)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (x y z)
f-3     (a b c)
For a total of 3 facts.
CLIPS>
```

- در صورت لزوم می توان چندین حقیقت را با ذکر شماره آنها در دستور retract حذف کرد. همچنین می توان از * به معنی همه حقایق استفاده کرد.

```
CLIPS> (retract 1 3 5 6)
CLIPS> (retract *)
```

- **مشاهده دائمی تغییرات حقایق:** یکی از امکانات اشکال زدائی CLIPS نمایش مداوم تغییرات حقایق موجود در حافظه کاری است (بدون آنکه مجبور شویم دائما از دستور facts استفاده کنیم). با دستور (watch facts) هرگونه تغییر در حقایق لیست می شود:

```
CLIPS> (clear)
CLIPS> (watch facts)
CLIPS> (assert (cat))
==> f-1     (cat)
<Fact-1>
CLIPS> (assert (dog))
==> f-2     (dog)
<Fact-2>
CLIPS> (retract 1)
<== f-1     (cat)
CLIPS> (facts)
f-0     (initial-fact)
f-2     (dog)
For a total of 2 facts.
CLIPS>
```

- در واقع برای مشاهده اضافه و حذف شدن موارد دیگر نیز می توان از watch استفاده کرد:

```
(watch facts)
(watch rules)
(watch deffunctions)
...
(watch all)      ; watch everything
```

- برای متوقف کردن watch از دستور unwatch استفاده می کنیم:

```
(unwatch facts)
(unwatch all)
```

## قواعد CLIPS

## ساختن قواعد

- فرض کنید می خواهیم قاعده مقابل را در clips پیاده سازی کنیم:

IF the animal is a duck
THEN the sound made is quack

- دستورات زیر ابتدا حقیقتی را ایجاد کرده و سپس یک قاعده (با نام duck) ایجاد می کنند که قسمت شرط آن با حقیقت ایجاد شده ارضاء (satisfy) می شود. نتیجه اجرای قاعده، ایجاد یک حقیقت جدید در حافظه کاری است.

```
CLIPS> (clear)
CLIPS> (assert  (animal-is duck))
<Fact-1>
CLIPS> (defrule duck
   (animal-is duck)
=>
   (assert (sound-is quack)))
CLIPS>
```

- تعریف قاعده فوق را طبیعتا می توان در یک خط هم انجام داد، ولی برای خوانایی بیشتر معمول است که قواعد در چند خط تعریف شوند.

```
CLIPS> (defrule duck (animal-is duck) => (assert (sound-is quack)))
```

- می توان برای هر قاعده میتوان توضیحات اضافه کرد. توضیح به شکل یک رشته (با quote) بعد از اسم قاعده اضافه می گردد. علاوه بر این با علامت سمیکولون برای هر خط از برنامه CLIPS نیز توضیحات اضافه کرد.

```
(defrule duck "This is the duck quack rule"      ; Rule header
   (animal-is duck)                              ; Pattern
=>                                               ; THEN arrow
   (assert (sound-is quack)))                    ; Action
```

- توجه کنید که هر قاعده تنها یک نام (منحصر به فرد) و یک توضیح می تواند داشته باشد.

- هر قاعده می تواند چندین الگو (شرط) و چندین نتیجه یا عمل (action) داشته باشد:

```
(defrule rule_name "optional_comment"
   (pattern_1)
   (pattern_2)
   …
   (pattern_N)
=>
   (action_1)
   (action_2)
   …
   (action_M))
```

- توجه کنید که پرانتز آخر مربوط به بستن defrule است. عمل برای مثال، می تواند ایجاد یا حذف یک حقیقت (assert یا retract) باشد.

- **فعال سازی و اجرای قواعد:** هرگاه قسمت شرط یک قاعده با حقایق ارضا (match یا satisfy) شود، آن قاعده فعال (enable) شده و در لیست کاری (agenda) قرار می گیرد.

- تنها یکی از قواعد موجود در agenda، آتش یا اجرا می شوند. CLIPS بطور اتوماتیک تصمیم می گیرد که کدام قاعده فعال شده را انتخاب (select) و آتش کند. لیست قواعد فعال شده داخل agenda، بترتیب افزایش اولویت (salience) نگهداری می شوند. در صورتیکه agenda خالی باشد اجرای برنامه متوقف می شود.

- اولویت قواعد: دستور (agenda) محتوای لیست کاری را نشان می دهد.

```
CLIPS> (assert (animal-is duck))
==> f-1     (animal-is duck)
<Fact-1>
CLIPS> (defrule duck (animal-is duck) => (assert (sound-is quack)))
CLIPS> (rules)
duck
For a total of 1 defrule.
CLIPS> (agenda)
0      duck: f-1
For a total of 1 activation.
CLIPS>
```

- شماره 0 قبل از قاعده در لیست agenda اولویت قاعده فعال شده را نشان می دهد. مقدار پیش فرض این اولویت 0 بوده ولی می توان مقداری بین 10000- و 10000 به آن داد.

- **اجرا:** برای اجرای قواعد از دستور (run) استفاده می شود. برای مثال اجرای مثال قبل به شکل زیر انجام می گیرد.

```
CLIPS> (run)
==> f-2     (sound-is quack)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (animal-is duck)
f-2     (sound-is quack)
For a total of 3 facts.
CLIPS>
```

- بنابراین در اثر اجرای برنامه یک حقیقت جدید به حافظه کاری اضافه شده است.

- همچنین، پس از اجرای قاعده، قواعد موجود در لیست agenda از لیست مذکور حذف می شوند. بنابراین اگر دوباره دستور (run) را اجرا کنیم اتفاقی نخواهد افتاد (چون قاعده ای فعال نیست که بخواهد آتش شود).

- اگر بخواهیم اجرای دوباره نتیجه ای بدهد، یکی از این دو حالت باید پیش آید:
- 1- باید یک حقیقت جدید که از قبل نبوده اضافه شود. در اینصورت دوباره عمل matching (با حقایق جدید وارد شده) انجام می شود.
- 2- یکی از حقایق موجود retract و دوباره assert شود. در این حالت در واقع یک قاعده حذف و یک قاعده برابر با آن اضافه شده است.

- اگر قرار بود بعد از اجرا قواعد همچنان فعال بمانند، قاعده ها برای همیشه بطور مداوم آتش می شدند که مطلوب نبود (مانند مغز انسان که نورون ها نباید بعد از یکبار آتش شدن به آتش شدن ادامه دهند).

- در ضمن قواعد داخل agenda ممکن است بر اثر تغییر قواعد و یا حقایق حذف گردند (مثلا به علت این بر اثر اجرای یک قاعده و آثار آن، که شرایط ارضاء دیگر برقرار نباشد، و یا تغییر دستی آنها).

- مثال زیر این مراحل را نشان می دهد:

```
CLIPS> (clear)
CLIPS> (defrule duck (animal-is duck)  => (assert (sound-is quack)))
CLIPS> (watch facts)
CLIPS> (watch activations)
CLIPS> (assert (animal-is duck))
==> f-1     (animal-is duck)                    ; output of "watch facts"
==> Activation 0      duck: f-1                 ; output of "watch activations"
<Fact-1>
CLIPS> (agenda)
0      duck: f-1
For a total of 1 activation.
CLIPS> (run)
==> f-2     (sound-is quack)                    ; new fact created from rule
CLIPS> (agenda)                                 ; nothing in agenda after run
CLIPS> (facts)
f-0     (initial-fact)
f-1     (animal-is duck)
f-2     (sound-is quack)
For a total of 3 facts.
CLIPS> (run)
```

## مشاهده قواعد

- برای مشاهده متن قواعد موجود در KB از دستور ppdefrule ( به معنی pretty print) استفاده می کنیم (همانطور که قبلا دیدیم، برای دیدن لیست قواعد از دستور rules استفاده می کنیم).

```
CLIPS> (ppdefrule duck)
(defrule MAIN::duck
   (animal-is duck)
   =>
   (assert (sound-is quack)))
CLIPS>
```

## چاپ خروجی

- قسمت عمل (action) قواعد علاوه بر تغییر در حقایق می تواند برای چاپ خروجی بکار رود. برای اینکار از دستور printout در قسمت عمل یک قاعده استفاده می شود. برای چاپ خروجی در خط جدید از پارامتر crlf در دستور استفاده می شود. همچنین به پارامتر t قبل از رشته قابل چاپ توجه داشته باشید. این پارامتر به دستور می گوید که نتیجه را به "ترمینال خروجی" (همان پنجره برنامه) ارسال کند. نتایج این دستور قابل ارسال به دیسک هم هستند.

```
CLIPS> (defrule duck
  (animal-is duck)
=>
  (printout t "quack" crlf))
==> Activation 0      duck: f-1
CLIPS> (run)
quack
CLIPS>
```

## ذخیره قواعد

- برای ذخیره قواعد از دستور save استفاده می کنیم.

```
CLIPS> (save "duck.clp")
TRUE
CLIPS>
```

- برای خواندن قواعد از فایل، از دستور load استفاده می کنیم.

```
CLIPS> (clear)
CLIPS> (load "duck.clp")
Defining defrule: duck +j+j
TRUE
CLIPS> (rules)
duck
For a total of 1 defrule.
CLIPS>
```

## ذخیره و بازخوانی حقایق

- برای ذخیره و بازخوانی حقایق نیز می توان از دستورات save-facts و load-facts استفاده کرد:

```
CLIPS> (clear)
==> f-0     (initial-fact)
CLIPS> (assert (cat) (dog) (fish))
==> f-1     (cat)
==> f-2     (dog)
==> f-3     (fish)
<Fact-3>
CLIPS> (save-facts "facts1")
TRUE
CLIPS> (clear)
==> f-0     (initial-fact)
CLIPS> (load-facts "facts1")
==> f-1     (cat)
==> f-2     (dog)
==> f-3     (fish)
TRUE
CLIPS>
```

## اجرای دستورات از فایل

- **اجرای دستورات از فایل:** با استفاده از دستور batch می توان دستورات را از یک فایل اجرا کرد. مثلا اگر محتوای فایل testbatch.txt دستورات زیر باشد:

```
(clear)
(assert (cat))
```

با دستور زیر می توان فایل را اجرا کرد:

```
CLIPS> (batch "test.txt")
```

- **اجرای دستورات سیستم عامل:** با استفاده از دستور system می توان دستورات سیستم عامل را از clips اجرا کرد.

```
CLIPS> (system "notepad")
```

- توجه: کلیه اعمالی که بر روی فایلها انجام می گیرند در دایرکتوری برنامه عمل می کنند و برای عملکرد ساده و صحیح باید دسترسی کامل به دایرکتوری برنامه به کاربر سیستم عامل باید داده شود.

## تعیین اولویت برای قواعد

- برای اینکار بعد از پارامتر نام قاعده با استفاده از دستور declare salience اولویت قاعده را تعیین می کنیم. توجه کنید که استفاده بی مورد یا زیاد این دستور باعث کنترلی شدن ساختار برنامه می شود.

```
CLIPS> (defrule testrule1 (declare (salience 1))
(testfact1) (testfact2) => (printout t "testrule1 firing." crlf))
CLIPS> (defrule testrule2
(testfact1) => (printout t "testrule2 firing." crlf))
CLIPS> (assert (testfact1) (testfact2))
==> f-1     (testfact)
<Fact-1>
CLIPS> (agenda)
1      testrule1: f-1
0      testrule2: f-1
For a total of 2 activations.
CLIPS> (run)
testrule1 firing.
testrule2 firing.
CLIPS>
```

## اضافه کردن جزئیات بیشتر

- **استراتژی انتخاب:** نحوه انتخاب از میان قواعد فعال شده و یا ترتیب آتش شدن قواعد فعال شده در CLIPS با استراتژی های مختلفی که قابل انتخاب هستند انجام می گیرد. استراتژی پیش فرض استراتژی عمق است. ولی استراتژی های سطح، LEX، MEA، complexity، simplicity و random هم قابل استفاده هستند. در هنگام اجرای برنامه تهیه شده توسط شخص یا گروه دیگر، باید مطمئن شد که استراتژی انتخاب، با استراتژی مورد استفاده توسط تهیه کننده سیستم تطابق دارد وگرنه نتیج متفاوت یا حتی غلط ممکن است بدست بیاید.

- **استراتژی عمق:** استراتژی پیش فرض CLIPS ، قواعد فعال شده جدید بعد از قواعد با اولویت بالاتر ولی قبل از قواعد با اولویت برابر که قبلا فعال شده اند و همچنین قواعد با اولویت پایینتر قرار می گیرند.

## تعریف گروهی حقایق

- در هنگام برنامه نویسی، وارد کردن تعدادی حقیقت از کیبورد و در خط دستور در هر اجرا، عملی وقت گیر است. برای اینکه مجبور به اینکار نشویم یا از batch فایل ها و یا از deffacts می توان استفاده کرد.

- روش batch فایل: فایلی با محتوای زیر و با نام batch1.txt درست کنید (یک خط خالی هم در آخر فایل قرار دهید).

```
(assert (a) (b) (c))
(assert (is_animal cat))
```

- حالا با دستور زیر فایل را اجرا کنید (مسیر فایل را به مسیر دلخواه تغییر دهید):

```
CIPS> (batch "c:/tmp/batch1.txt")
CLIPS> (batch "c:/tmp/batch1.txt")
TRUE
CLIPS> (assert (a) (b) (c))
<Fact-3>
CLIPS> (assert (is_animal cat))
<Fact-4>
CLIPS>
```

## تعریف گروهی حقایق با deffacts

- در این روش چند قاعده در یک دستور قرار می گیرند.توجه کنید که حقایق تعریف شده توسط این دستور بلافاصله ایجاد نمی شوند. بلکه بعد از یک reset اجرا می گردند. بنابراین بعد از reset حقایق جدید تعریف شده و گروهی از قاعد را بلافاصله فعال می کنند.

```
CLIPS> (reset)
CLIPS> (facts)
f-0     (initial-fact)
For a total of 1 fact.
CLIPS> (deffacts animals "defining a few animals"
       (is_animal cat)
       (is_animal dog)
       (is_animal duck))
CLIPS> (facts)
f-0     (initial-fact)
For a total of 1 fact.
```

```
CLIPS> (reset)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (is_animal cat)
f-2     (is_animal dog)
f-3     (is_animal duck)
For a total of 4 facts.
CLIPS>
```

**تفاوت دستور clear ودستور reset:** دستور reset قواعد و deffacts ها را حذف نمی کند، ولی دستور clear کلیه قواعد و دستورات (از جمله deffacts) ها را حذف می کند. دستور reset اعمال زیر را انجام می دهد:

1. حقایق را از لیست حقایق حذف می کند. در نتیجه ممکن است قواعدی هم از لیست agenda حذف گرند.
2. حقیقت اولیه را دوباره تعریف می کند.
3. حقایقی را که توسط دستورات deffacts تعریف شده اند دوباره تعریف می کند.
4. بعدا خواهیم دید که دستور ریست همانند حقایق، شیء اولیه را هم ایجاد کرده و با جملات definstances، اشیاء تعریف شده را هم باز سازی می کند.

اصولا دستور reset باید قبل از اجرای برنامه (و قبل از تعریف احتمالی دستی قواعد) اجرا شود، بخصوص اگر برنامه یکبار قبلا اجرا شده باشد.

- با دستور ppdeffacts می توان متن یک دستور deffacts را دید. برای اینکار دانستن نام دستور deffacts لازم است:

```
CLIPS> (deffacts animals "defining a few animals"
         (is_animal cat)
         (is_animal dog)
         (is_animal duck))
CLIPS> (ppdeffacts animals)
(deffacts MAIN::animals "defining a few animals"
   (is_animal cat)
   (is_animal dog)
   (is_animal duck))
```

```
CLIPS> (list-deffacts)
initial-fact
animals
For a total of 2 deffacts.
CLIPS> (undeffacts animals)
CLIPS> (ppdeffacts animals)
[PRNTUTIL1] Unable to find deffacts animals.
CLIPS>
```

- مشاهده می کنید که با دستور undeffacts هم می توان تعریفات deffacts را حذف نمود (در صورت reset کردن هم طبیعتا حقیقتی ساخته نمی شود.) با دستور (list-deffacts)هم فقط نام deffacts ها لیست می شود.

- یادآوری میشود که با دستورات watch می توان تغییرات پشت پرده CLIPS را رصد کرد. برای مثال (watch activations) ورود قواعد به agenda و (watch complications) بار گزاری قواعد را گزارش می کنند.
همچنین (watch statistics) آمار و اطلاعاتی را که برای بهینه سازی کارآیی سیستم مفید هستند ارائه می کنند.

- درصورتیکه می خواهید یک لاگ از آنچه روی ترمینال ظاهر می شود ضبط کنید از دستورات زیر استفاده کنید:

```
CLIPS> (dribble-on "c:/tmp/drible1.txt")
TRUE
CLIPS>
```

- برای متوقف کردن لاگ از دستور زیر استفاده کنید (وگرنه فایل قابل باز کردن در برنامه دیگر نخواهد بود).

```
CLIPS> (dribble-off)
```

**اجرای کنترل شده قواعد:** دستور مقابل تنها یک قاعده را اجرا (آتش) می کند.

```
CLIPS> (run 1)
```

با استفاده از دستور فوق می توان اجرای قواعد را یک به یک انجام داد و نتایج حاصله را بررسی کرد. همچنین می توان قواعد را تا قاعده خاصی به شکل زیر اجرا کرد:

```
CLIPS> (run 5)
```

دستور فوق، حداکثر 5 قاعده را اجرا کرده و می ایستد.

```
CLIPS> (defrule rule1 (fact1) (fact2) => (printout t "rule1" crlf))
CLIPS> (defrule rule2 (fact2)         => (printout t "rule2" crlf))
CLIPS> (assert (fact1) (fact2))
<Fact-2>
CLIPS> (agenda)
0     rule1: f-1,f-2
0     rule2: f-2
For a total of 2 activations.
```

```
CLIPS> (run 1)
rule1
CLIPS> (run 1)
rule2
CLIPS>
```

## نقاط breakpoint

مانند زبانهای برنامه نویسی CLIPS نیز امکان ایجاد نقاط ایست را دارد. دستور زیر یک نقطه ایست برروی قاعده rule2 ایجاد می کند:

```
CLIPS> (set-break rule2)
```

فرض کنید دو قاعده قبلی هنوز در حافظه هستند.

```
CLIPS> (reset)
CLIPS> (set-break rule2)
CLIPS> (assert (fact1) (fact2))
<Fact-2>
CLIPS> (agenda)
0     rule1: f-1,f-2
0     rule2: f-2
For a total of 2 activations.
```

```
CLIPS> (run)
rule1
Breaking on rule rule2.
CLIPS> (run)
rule2
CLIPS>
```

گاهی اوقات انتظار دارید که یک قاعده خاص فعال شده باشد ولی می بینید که نشده. برای آنکه تست کرده وببینید کدامیک از شروط یک قاعده در حال حاضر ارضاء شده اند می توانید از دستور matches استفاده کنید.

```
CLIPS> (defrule rule1 (fact1) (fact2) => (printout t "rule1" crlf))
CLIPS> (defrule rule2 (fact2)         => (printout t "rule2" crlf))
CLIPS> (assert (fact2))
<Fact-1>
CLIPS> (agenda)
0      rule2: f-1
For a total of 1 activation.
CLIPS> (matches rule1)
Matches for Pattern 1
 None
Matches for Pattern 2
f-1
Partial matches for CEs 1 – 2              ;CE means conditional
element
 None
Activations
 None
```

```
CLIPS> (matches rule2)
Matches for Pattern 1
f-1
Activations
f-1
CLIPS>
```

## توابع رشته ها

تعدادی تابع درونساخت برای کارکردن روی رشته ها در CLIPS وجود دارند.

| | |
|---|---|
| str-index | Returns a string index of first occurrence of a substring |
| sub-string | Returns a substring from a string. |
| str-compare | Performs a  string compare |
| str-length | Returns the string length which is the length of a string: |
| str-cat | string concatenation |

برای مثال از دستور str-compare به شکل های زیر می توان استفاده کرد:

```
CLIPS> (str-compare "test" "test1")
-1
CLIPS> (assert (testresult (str-compare "test" "test1")))
<Fact-2>
CLIPS> (facts)
f-0     (initial-fact)
f-1     (fact2)
f-2     (testresult -1)
For a total of 3 facts.
CLIPS>
```

مثالی دیگر برای استفاده از تابع str-cat:

```
CLIPS> (str-cat "test1" "test2")
"test1test2"
CLIPS> (assert (field1 (str-cat "str1" "str2")))
<Fact-1>
CLIPS> (facts)
f-0     (initial-fact)
f-1     (field1 "str1str2")
For a total of 4 facts.
CLIPS>
```

از تابع sym-cat هم برای اتصال سیمبول ها استفاده می شود:

```
CLIPS> (sym-cat cat dog)
catdog
CLIPS> (assert (field1 (sym-cat cat dog)))
```

توجه: در assert ها توابع نمی توانند فیلد اول باشند (وگرنه syntax error می گیرید).

# متغییر ها (variables)

در CLIPS هنگامی که یک حقیقت ایجاد گردید دیگر نمی توان فیلد های آنرا تغییر داد. تنها راه تغییر حقیقت، حذف حقیقت قبلی (retract) و اضافه کردن حقیقتی جدید (با مقادیر فیلدهای جدید) است.

برخلاف حقایق، می توان در CLIPS از متغییرها استفاده نمود که مقدار آنها قابل تغییر است. متغییر ها با نامی که با یک علامت سوال شروع شده مشخص می گردند.

?x ?color ?age

توجه داشته باشید که به متغییر ها قبل از استفاده (خواندن) باید یک مقدار اولیه نسبت داد. وگرنه پیام خطا دریافت می کنیم.

```
CLIPS> (defrule test (fact1) => (printout t ?x crlf))
[PRCCODE3] Undefined variable x referenced in RHS of defrule.
ERROR:
(defrule MAIN::test
  (fact1)
=>
   (printout t ?x crlf))
CLIPS>
```

پیام خطا به این علت داده می شود که یک مقدار(value) به متغیر bound نشده. توجه داشته باشید که استفاده از متغیر در قسمت RHS یا نتیجه بدون اینکه قبلا مقداری به متغییر داده شده باشد مجاز نیست، ولی استفاده از متغییر بدون مقدار در قسمت شرط مجاز است.

یک کاربرد معمول متغییر ها هنگامی است که در قسمت condition یک قاعده یک مقدار را تست کرده و سپس بخواهیم همان مقدار را در قسمت نتیجه آن مقدار را استفاده کنیم. مثال:

```
CLIPS> (defrule sound  (make-sound ?sound)  =>  (assert
(sound-is ?sound)))
CLIPS> (assert (make-sound hahaa))
<Fact-1>
CLIPS> (agenda)
0      make-sound: f-1
For a total of 1 activation.
CLIPS> (run)
```

```
CLIPS> (facts)
f-0     (initial-fact)
f-1     (make-sound hahaa)
f-2     (sound-is hahaa)
For a total of 3 facts.
CLIPS>
```

توجه داشته باشید که در متغییر فوق (?sound)، تنها داخل همان قاعده معتبر است (متغیر محلی همان قاعده است) و در بیرون قاعده شناخته شده نیست.

همچنین امکان استفاده چند باره از متغییر وجود دارد.

```
CLIPS> (defrule sound  (make-sound ?sound)  =>  (printout t ?sound " " ?sound crlf))
CLIPS> (assert (make-sound hihii))
<Fact-1>
CLIPS> (run)
hihii hihii
CLIPS>
```

درصورتیکه از متغییر مقدار نگرفته بصورت مستقیم (نه در یک قاعده) استفاده کنیم، خطایی به شکل زیر دریافت می کنیم:

```
CLIPS> (printout t ?x crlf)
[EVALUATN1] Variable x is unbound
CLIPS>
```

مثالی دیگر از استفاده از متغییر ها در چاپ نتایج خروجی به شرح زیر است:

```
CLIPS> (defrule parentship
          (parent ?par ?child)
           =>
         (printout t ?par " is " ?child "'s parent." crlf))
CLIPS> (assert (parent ali hasan))
<Fact-2>
CLIPS> (run)
ali is hasan's parent.
CLIPS>
```

وقتی یک متغییر با مقدم یک قاعده تطبیق یافت، مقدار تطبیق یافته، در تمام قسمت های مقدم و تالی اعمال خواهد شد (مگر آنکه مقدار آن در تالی تغییر داده شود). در مثال زیر متغییر ?num می تواند مقادیر 0 یا 1 را بگیرد. در صورتیکه مقدار 0 بگیرد، حقایق 1 و 2 با هم مقدم قاعده را ارضاء می کنند. و اگر مقدار 1 بگیرند حقایق 3 و 4 مقدم قاعده را ارضاء خواهند کرد.

```
CLIPS> (facts)
f-0     (initial-fact)
f-1     (number-1 0)
f-2     (number-2 0)
f-3     (number-1 1)
f-4     (number-2 1)
For a total of 5 facts.
CLIPS> (defrule bound
         (number-1 ?num)
         (number-2 ?num)
         =>)
```

```
CLIPS> (agenda)
0      bound: f-3,f-4
0      bound: f-1,f-2
For a total of 2 activations.
CLIPS>
```

## دریافت از ورودی

دریافت مقادیر از ورودی با استفاده از دستور read انجام می گردد. برای مثال اگر لازم است فیلدی از یک حقیقت از ورودی دریافت شود به شکل زیر عمل می کنیم:

```
CLIPS> (assert (book (read)))
mybook
<Fact-1>
CLIPS> (facts)
f-0     (initial-fact)
f-1     (book mybook)
For a total of 2 facts.
CLIPS>
```

یک برنامه نمونه با قواعد ، متغیرها، ایجاد حقایق و چاپ خروجی.

```
CLIPS> (defrule get-names
          =>
          (printout t "Who's the parent? ")
          (assert (parent (read)))
          (printout t "Who's the child? ")
          (assert (child (read))))

CLIPS> (defrule build-parentship
          (parent ?tparent)
          (child   ?tchild)
           =>
          (assert (parent ?tparent ?tchild)))
```

```
CLIPS> (defrule parentship
          (parent ?rparent ?rchild)
           =>
          (printout t ?rparent " is " ?rchild "'s parent." crlf))
CLIPS> (run)
Who's the parent? Ali
Who's the child? Hasan
Ali is Hasan's parent.
CLIPS>
```

## استفاده از wildcard

از علامت ? برای نشان دادن یک فیلد و از علامت $? برای نشاندادن صفر یا چند فیلد می توان استفاده کرد.

```
CLIPS> (assert (book1) (book1 chapter1) (book1 chapter1
chapter2) (book1 chapter1 chapter2 chapter3))
<Fact-4>
CLIPS> (defrule selection (book1 ?)  => (printout t "rule match"
crlf))
CLIPS> (run)
rule match
CLIPS>
```

```
CLIPS> (clear)
CLIPS> (assert (book1) (book1 chapter1) (book1 chapter1
chapter2) (book1 chapter1 chapter2 chapter3))
<Fact-4>
CLIPS> (defrule selection (book1 $?)  => (printout t "rule match"
crlf))
CLIPS> (run)
rule match
rule match
rule match
rule match
CLIPS>
```

## حذف حقایق با استفاده از قواعد

برای استفاده از دستور retract به منظور حذف یک حقیقت در قسمت نتیجه یک قاعده ، ابتدا باید شماره مشخصه حقیقت را بدست آورد. شماره حقیقت به یک متغییر داده شده و آن متغییر در دستور retract استفاده می شود.

```
CLIPS> (assert (book1 chapter1) (book1 chapter1 chapter2)
(book1 chapter1 chapter2 chapter3))
<Fact-3>
CLIPS> (facts)
f-0     (initial-fact)
f-1     (book1 chapter1)
f-2     (book1 chapter1 chapter2)
f-3     (book1 chapter1 chapter2 chapter3)
For a total of 3 facts.
CLIPS> (defrule selection
          ?bookfact <- (book1 ?)
         =>
          (printout t "Removing fact : " ?bookfact crlf)
          (retract ?bookfact))
```

```
CLIPS> (run)
Removing fact : <Fact-1>
CLIPS> (facts)
f-0     (initial-fact)
f-2     (book1 chapter1 chapter2)
f-3     (book1 chapter1 chapter2 chapter3)
For a total of 3 facts.
CLIPS>
```

## استفاده از وایلدکارد $?  برای متغیرهای چندتایی

توجه کنید که در هنگام تطبیق دادن در مقدم (LHS)، از متغیر چندتایی استفاده شده ولی در قسمت تالی (RHS) از متغیر معمولی که تنها یکی از محتویات متغیر چندتایی را نشان خواهد داد.

```
CLIPS> (assert (book1 chapter1)
               (book1 chapter1 chapter2)
               (book1 chapter1 chapter2 chapter3))
<Fact-3>
CLIPS> (defrule selection
          (book1 $?chapters)
           =>
          (printout t "Book1 : " ?chapters crlf))
CLIPS> (run)
Book1 : (chapter1 chapter2 chapter3)
Book1 : (chapter1 chapter2)
Book1 : (chapter1)
CLIPS>
```

## مقدار دهی به متغییر ها در تالی

در قسمت های گذشته یاد گرفتیم که چگونه در مقدم یک قاعده به یک متغییر مقدار داده (با تطبیق مقدم با حقایق) و در قسمت نتیجه از آن استفاده کنیم. در صورتیکه بخواهیم در قسمت تالی یک قاعده به متغیری را تعریف کرده یا به آن مقدار بدهیم، باید از دستور bind استفاده کنیم.

```
CLIPS> (defrule testrule
          =>
           (bind ?myvar 10)
           (printout t ?myvar crlf))
CLIPS> (run)
10
CLIPS>
```

## استفاده از ارتباط دهنده های منطقی

ارتباط دهنده ها و عملگر های منطقی در قسمت شرط قابل استفاده هستند. اگر چند حقیقت در قسمت مقدم لیست گردند، همه آنها برای فعال شدن قاعده باید ارضاء شوند (حقایق موجود در مقدم همگی وجود داشته باشند). بنابراین حقایق لیست شده در مقدم قواعد ، بصورت پیش فرض، باهم AND شده اند.

نحوه استفاده از اپراتورهای NOT و OR به شکل زیر است:

```
CLIPS> (clear)
CLIPS> (defrule test
          (not (book1))
           =>
          (printout t "hello" crlf))
CLIPS> (agenda)
0      test: *
For a total of 1 activation.
CLIPS> (run)
hello
CLIPS>
```

قاعده فوق با عدم وجود یک حقیقت ارضاء شده است.

برای عملگر or به شکل زیر عمل می کنیم:

```
CLIPS> (defrule test
          (or (book1) (book2))
             =>
           (printout t "hello" crlf))
CLIPS> (assert (book1))
<Fact-1>
CLIPS> (agenda)
0      test: f-1
For a total of 1 activation.
CLIPS>
```

## استفاده از دستور if then else

مثال زیر برنامه ای را نشان می دهد که از دستور if برای پردازش ورودی استفاده می کند. توجه کنید که چون قاعده مقدم ندارد، مقدم TRUE فرض شده و در نتیجه قاعده به محض تعریف و یا بعد از هر reset فعال خواهد شد.

```
CLIPS> (defrule get-inp
            =>
          (printout t "Are you older than 20? ")
          (bind ?answer (read))
          (if (eq ?answer yes)
              then
                (assert (person old))
              else
                (assert (person teenage))))
CLIPS> (run)
Are you older than 20? yes
```

```
CLIPS> (facts)
f-0     (initial-fact)
f-1     (person old)
For a total of 2 facts.
CLIPS>
```

## تعریف توابع

مانند زبانهای دیگر CLIPS نیز امکان ایجاد توابع یا رویه ها را فراهم می کند. شکل کلی توابع در CLIPS به شکل زیر است:

```
(deffunction <function-name> [optional comment]
  (?arg1 ?arg2 ...?argM)
  (<action1>
   <action2>
    ...
   <actionK>)                    ;only last action returns value
```

## تعریف توابع

**مثال:**

```
CLIPS> (deffunction ask-question (?question $?allowed-values)
          (printout t ?question)
          (bind ?answer (read))
           ?answer)
CLIPS> (defrule ask-his-name
          =>
          (bind ?response (ask-question "What is your name? "))
          (assert (name ?response)))
CLIPS> (run)
What is your name? ali
CLIPS> (facts)
f-0     (initial-fact)
f-1     (name ali)
For a total of 2 facts.
```

قاعده فوق باعث ایجاد یکی از حقایق (person old) یا (person teenage) خواهد شد. در ادامه می توان قواعدی اضافه کرد که بر اساس این حقایق فعال و اجرا شوند. مثلا قواعد زیر برای حالت (person teenage) هستند.

```
(defrule teenage-heart-condition
   (person teenage)
   =>
   (printout t "Do you feel pain in your heart (yes/no)? ")
   (bind ?answer (read))
   (if (eq ?answer yes)
      then
       (assert (heart in-pain))
      else
       (assert (heart no-pain))))
```

```
(defrule teenage-heart-in-pain
   (person teenage)
   (heart in-pain)
   =>
   (printout t "Are you in love (yes/no)? ")
   (bind ?answer (read))
   (if (eq ?answer yes)
      then
       (printout t "You will be fine ..." crlf)
      else
       (printout t "See a doctor ..." crlf)))
```

و دو قاعده زیر نیز در صورتیکه شخص بالای 20 سال باشد اجرا خواهند شد:

```
(defrule old-heart-condition
   (person old)
   =>
   (printout t "Do you feel pain in your heart (yes/no)? ")
   (bind ?answer (read))
   (if (eq ?answer yes)
      then
       (assert (heart in-pain))
      else
       (assert (heart no-pain))))
```

```
(defrule old-heart-in-pain
   (person old)
   (heart in-pain)
   =>
   (printout t "Do you take medications (yes/no)? ")
   (bind ?answer (read))
   (if (eq ?answer yes)
      then
       (printout t "Take your medicine ..." crlf)
      else
       (printout t "See a doctor ..." crlf)))
```

اگر بخواهیم در ازای همه مقادیر غیر از مقدار خاصی قاعده ای اجرا شود به شکل زیر عمل می کنیم:

```
CLIPS> (defrule walk
  (light ~green)
=>
  (printout t "Don't walk" crlf))
CLIPS> (assert (light green))
<Fact-1>
CLIPS> (agenda)
CLIPS> (assert (light yellow))
<Fact-2>
CLIPS> (agenda)
0     walk: f-2
For a total of 1 activation.
CLIPS>
```

در قاعده فوق اگر حقیقتی موجود باشد که فیلد اول آن light و فیلد دوم آن هرچیز دیگری بغیر از green باشد، قاعده فوق اجرا می شود.

اگر بخواهیم در صورتیکه فیلد خاصی از قواعد، مقادیر مشخصی را داشتند، قاعده اجرا شود، باید به شکل زیر عمل کنیم.

```
CLIPS>  (defrule cautious
  (light yellow|blinking)
=>
  (printout t "Be cautious" crlf))
CLIPS> (assert (light yellow))
<Fact-1>
CLIPS> (agenda)
0     cautious: f-1
For a total of 1 activation.
CLIPS> (assert (light blinking))
<Fact-2>
CLIPS> (agenda)
0     cautious: f-2
0     cautious: f-1
For a total of 2 activations.
```

در قاعده فوق اگر حقیقتی داشته باشیم که فیلد اول آن light و فیلد دوم آن یکی از مقادیر yellow یا blinking باشد، در آنصورت قاعده فعال می شود. دومقدار مجاز فیلد با علامت | از هم جدا شده اند.