به نام خداوند بخشنده مهربان

آموزش نرم افزار CLIPS

درس:

سیستم های خبره

دانشگاه:

آزاد اسلامی واحد پرند

استاد:

خانم مهندس آزاده فتحی

سال تحصیلی:

۱۳۹۰

تهیه کنندگان:

علی رضا علی اله

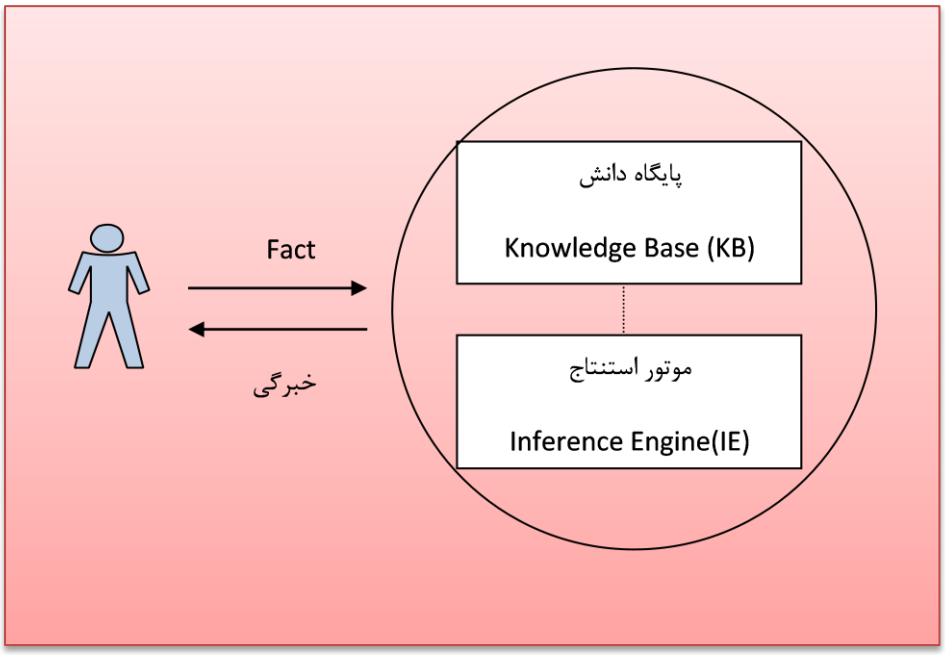
محسن عبیری

به نام خدا

مقدمه

سیستم خبره یک برنامه کامپیوتری هوشمند است که از دانش و روش های استنتاج برای حل مسائلی استفاده می کند که به دلیل مشکل بودن نیاز به تجربه و مهارت انسان دارد.این سیستم(شکل داخل دایره) یک واقعیت (**Fact**) را از بیرون از سیستم دریافت می کند و با توجه به آن واقعیت با پاسخ و راه حل مناسب(خبرگی) را به عنوان خروجی می دهد.این سیستم در حالت کلی از ۲ قسمت تشکیل شده است.

- پایگاه دانش(**Knowledge Base(KB)**)
- موتور استنتاج(**Inference Engine(IE)**)



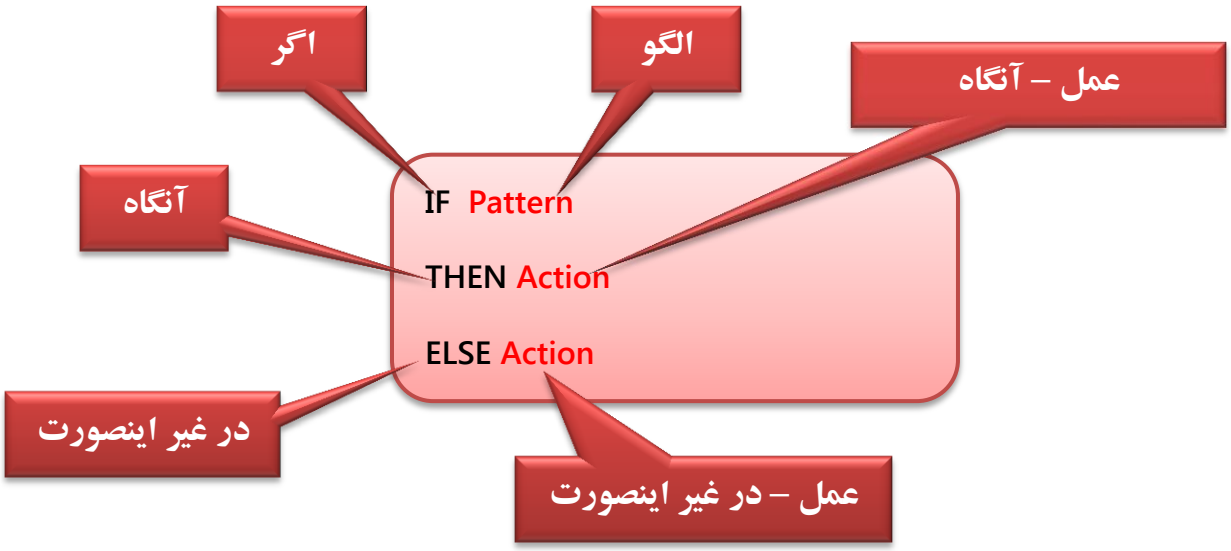
واقعیت(Fact)

واقعیات در همان اطلاعاتی است که به عنوان ورودی به سیستم خبره داده می شود.

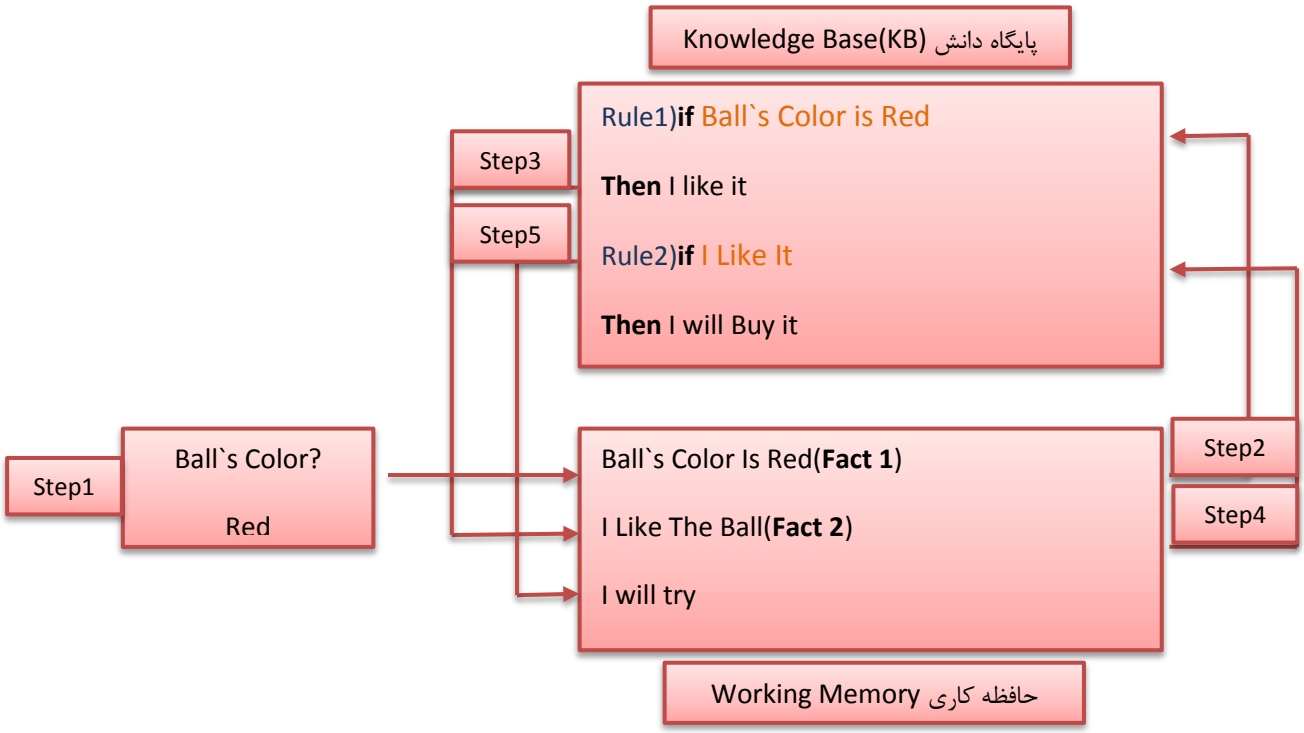
قواعد(Rule)

ساختار دانشی است که یعنی اطلاعات دانسته به اطلاعات دیگری می تواند به معلومات قبلی اضافه و یا دانسته فرض شوند.ساختار این قاعده یک یا چند پیش فرض در قسمت **"اگر"** را با نتیجه گیری ها در قسمت **"آنگاه"** مرتبط می سازد یک قاعده می تواند در **"غیر اینصورت"** هم داشته باشد.

در واقع قاعده ساختاری است که با برقرار شدن قسمت **الگو"Pattern"** قسمت **عمل "Action"** اجرا می شود که اگر این **Action** خودش معادل یک الگو(**Pattern**) دیگر شود عمل مربوط به آن اجرا می شود و به همین ترتیب تا آخر ادامه پیدا می کند.



مثال:



در این مثال واقعیات(**Facts**) در حافظه کاری و قواعد(**Rules**) در پایگاه دانش قرار دارند.قاعده **Rule1** اجـرا مـی شـود و قسمت الگوی آن با **Fact 1** تطبیق دارد یعنی(**Ball`s Color Is Red**) پس چون قسمت الگو با **Fact 1** تطبیـق داشـت قسمت عمل **Rule1** یعنی **I like It** انجام می شود.سپس **Rule2** اجرا می شود چون الگوی آن با **Fact 2** که این قسـمت توسط عمل **Rule1** یعنی(**I like it**) ساخته شده است تطبیق دارد قسمت عمـل **Rule2** یعنـی **I will Buy It** اجـرا مـی شود.

نرم افزار کلیپس(CLIPS)

از این نرم افزار برای ساخت سیستم های خبره استفاده میشود.

برخی از ویژگیهای نرم افزار به شرح زیر می باشد.

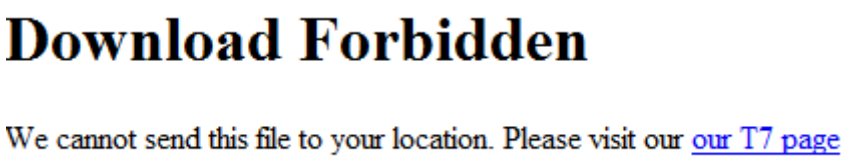
- از سر نام های عبارات **C Language Implementation Production System** گرفته شده است
- برای اولین با توسط **NASA** در واحد فناوری نرم افزار(**Software Technology Branch**) تولید شده است .
- این نرم افزار رایگان بوده و در حال حاضر نسخه ۶.۳ آن موجود می باشد.
- این نرم افزار بر روی سیستم عامل های متفاوت مانند ویندوز، لینوکس، مک نصب می گردد.
- این نرم افزار با استفاده از زبان **C** ساخته شده است.
- این نرم افزار چند کاره می باشد یعنی می توانید بصورت مدل های زیر با آن برنامه نویسی کنید.
  - بر مبنای قواعد(**Rule-based**)
  - بر مبنای شی گرایی(**Object-oriented**)
  - بر مبنای روال ها(**Procedural programming**)
- نرم افزار کلیپس فقط **قواعد زنجیره ای پیشرو** را پشتیبانی می کند.

دستیابی به نرم افزار

این نرم افزار یک برنامه رایگان می باشد که هم اکنون نسخه ۶.۳ آن موجود می باشد می توانید این نرم افزار را برای سیستم عامل های موجود یعنی ویندوز، لینوکس و مک تهیه نمایید به همین منظور می توانید به سایت زیر مراجعه کرده و آنرا دانلود نمایید.

http://clipsrules.sourceforge.net/

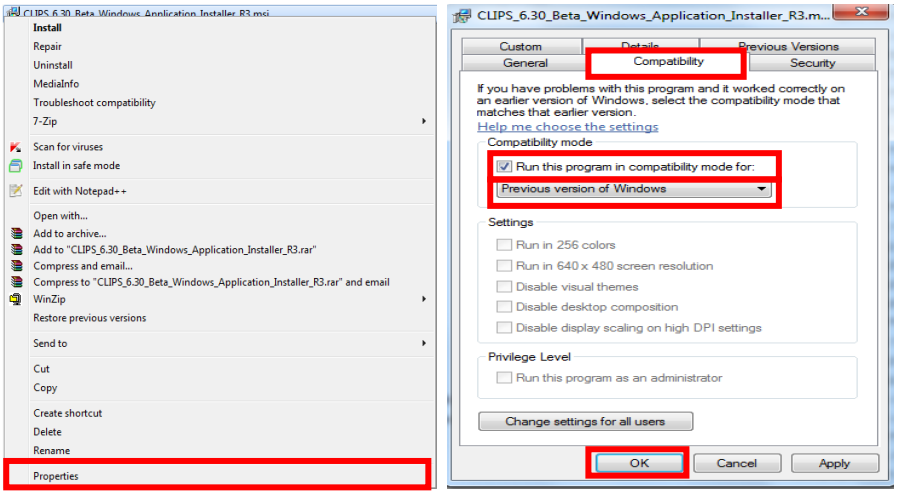
نکته:این سایت با چک نمودن آی پی شما متوجه می شود که در خواست دانلود از سمت کشور ایران می باشد و اجازه دانلود به شما نمی دهد به همین منظور یا باید از **VPN** استفاده نمایید یا از سایت های دیگر که نسخه کلیپس در آنها وجود دارد ،دانلود نمایید.



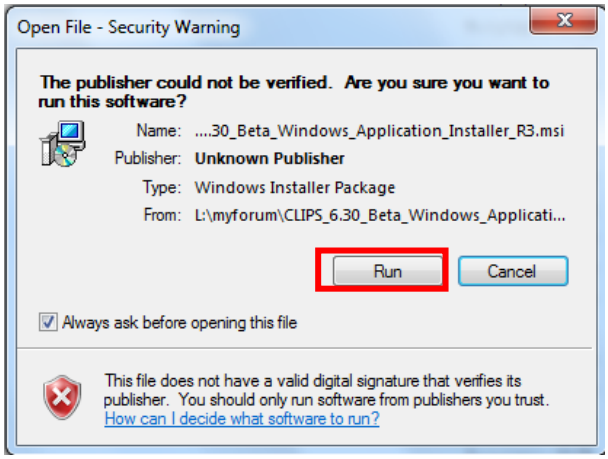
نصب نرم افزار کلیپس

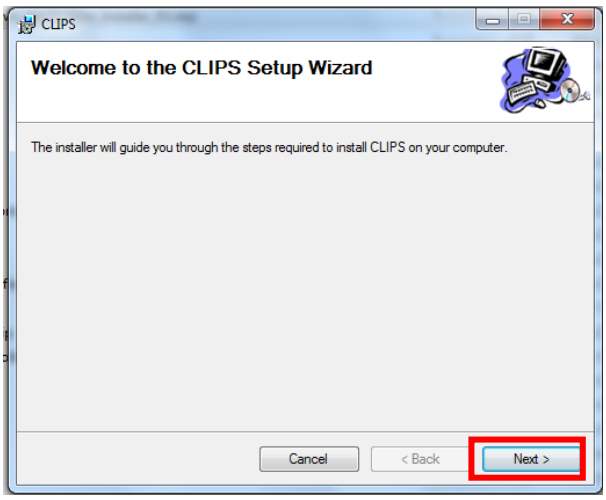
این نرم افزار مانند دیگر نرم افزارهای استاندارد ویندوز نصب می گردد کافی است فایل **Setup** برنامه(**CLIPS_6.30_Beta_Windows_Application_Installer_R3.msi**) را اجرا نمایید.همچنین این برنامه با ویندوز سون هیچ مشکلی ندارد.

نکته:در صورت نصب این برنامه یا دیگر نرم افزار های ۳۲ بیتی روی ویندوز ۶۴ بیتی(**x64**) که اکثر لپ تاپ های امروزی دارای این نوع سیستم عامل هستند کافی است روی فایل **Setup** برنامه راست کلیک نمایید و سپس گزینه **properties** را انتخاب نمایید و از صفحه موجود تب **Compatibility** را انتخاب کرده سپس گزینه **Run this program in Compatibility Mode For** را فعال نمایید.سپس گزینه **Ok** را انتخاب نمایید و مراحل نصب را پیگیری نمایید.



پس از اجرای فایل **setup** صفحه زیر نمایش داده می شود که روی گزینه **Run** کلیک می نماییم.



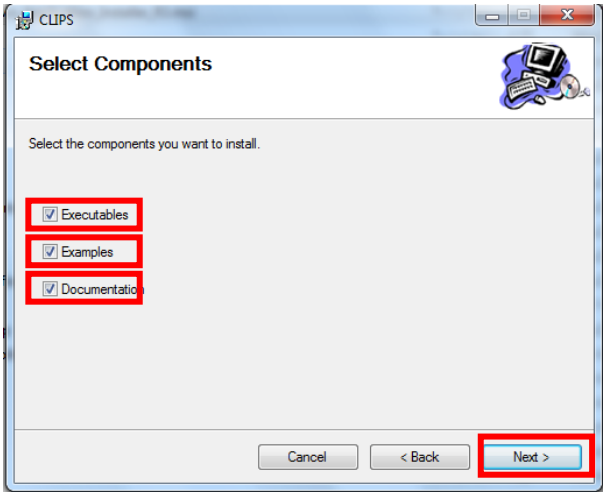


سپس روی دکمه **Next** کلیک می نماییم.سپس صفحه ای ظاهر می شود با ۳ گزینه به شرح زیر:

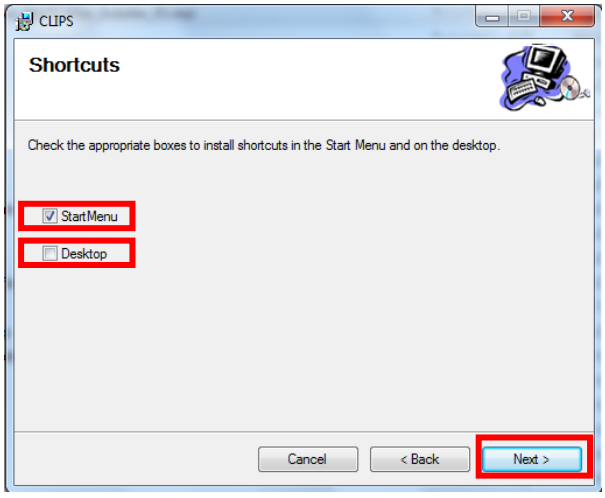
Executables: شامل فایل هایی اجرایی برنامه.

Examples: نمونه مثال های آماده برنامه کلیپس(مانند سیستم تعمیر خودرو).

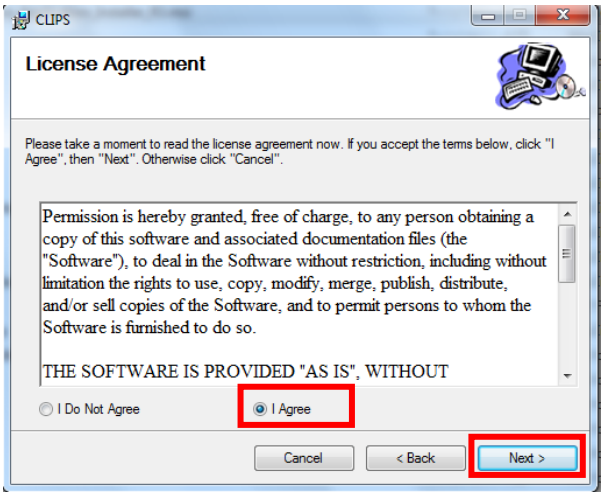
Documentations:مستندات و آموزش نرم افزار در قالب فایل **PDF** به زبان انگلیسی.



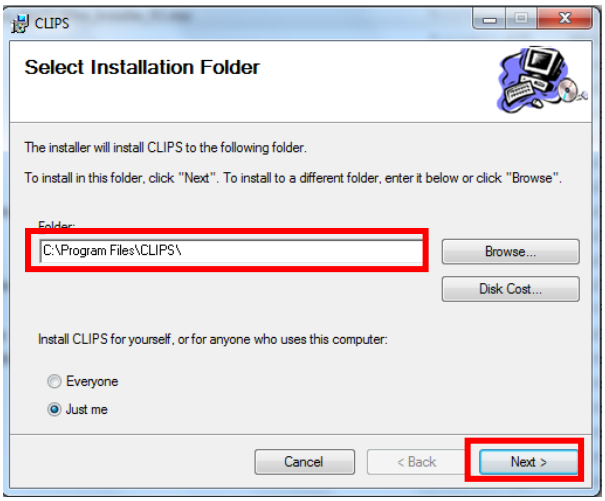
که بصورت پیش فرض هر ۳ گزینه انتخاب شده است.سپس روی دکمه **Next** کلیک می نماییم.در مرحله بعد محل کپی شدن شورت کات برنامه را مشخص می کنیم.



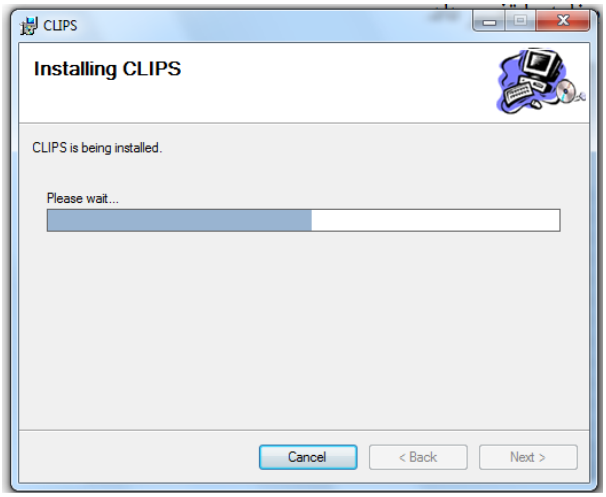
سپس در صفحه بعد توافق نامه را تایید نموده(**I Agree**) و روی دکمه **Next** کلیک نمایید.



در صفحه بعد می تواند مسیر نصب برنامه را تغییر داد.



در نهایت روی دکمه **Next** کلیک نمایید تا برنامه نصب شروع شود.

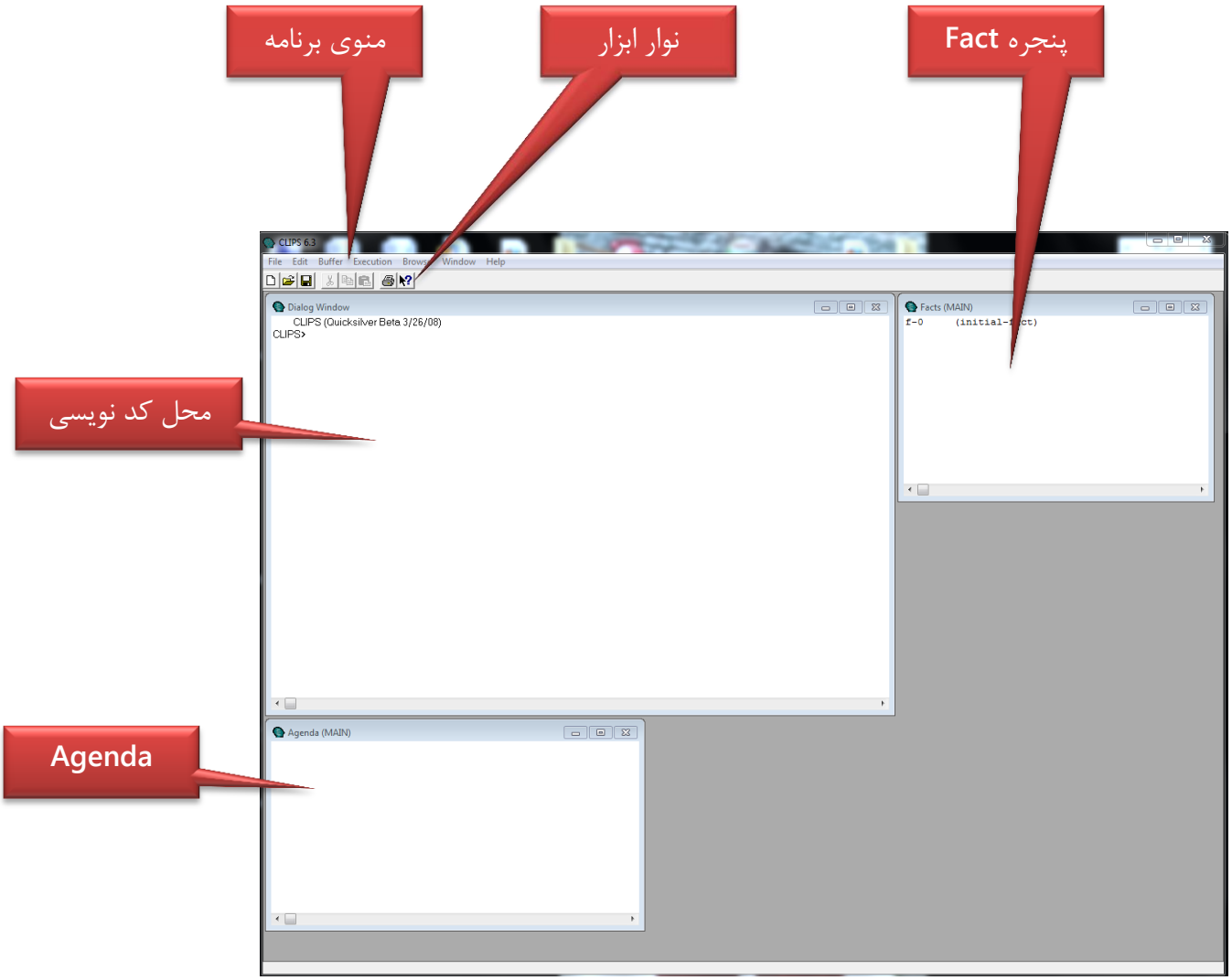


اجرای نرم افزار

برنامه کلیپس شامل ۲ کنسول برای نوشتن و اجرای نمودن برنامه ها میباشد.یک کنسول تحت داس می باشد که محیط شبیه سیستم عامل داس (Command Prompt) دارد و تمام محیط تعاملی بصورتی دستوری می باشد.

و دارای یک محیط ویژوالی مانند دیگر نرم افزار های تحت ویندوز می باشد .برای اجرای نرم افزار فایل CLIPSWin.exe را اجرای می نماییم.

محیط نرم افزار



منوی های نرم افزار

منوی File

- **New**:ایجاد یک فایل جدید.
- **Save**:ذخیره فایل.
- **Open**:باز نمودن فایلی که قبلا ایجاد و یا ذخیره شده است.
- **Load**:فراخوانی و بارگذاری برنامه
- **Close**: بستن فایل جاری
- **Print**:چاپ نمودن دستورات.
- **Exit**:خارج شدن و بستن نرم افزار.

منوی Edit

- **Undo**:لغو دستورات انجام شده.
- **Cut**:برش زدن متن انتخاب شده.
- **Copy**:کپی نمودن متن انتخاب شده.
- **Paste**:تکمیل کننده دستورات **Copy** و **Cut**.
- **Delete**:حذف نمودن متن انتخاب شده.
- **Select All**:انتخاب کل صفحه.
- **Comment**:اعمال توضیحات.
- **UnComment**:خارج نمودن از حالت توضیحات.

منوی **Execution**

- **Reset**:معادل دستور **Reset** میباشد.(در بخش های بعدی توضیح داده خواهد شد.)
- **Run**: معادل دستور **Run** میباشد.(در بخش های بعدی توضیح داده خواهد شد.)
- **Step**: معادل دستور **Run 1** میباشد.(در بخش های بعدی توضیح داده خواهد شد.)
- **Clear**: معادل دستور **Clear** میباشد.(در بخش های بعدی توضیح داده خواهد شد.)

منوی Broswe

دستورات این منو برای مدیریت ساختارها می باشد.

- **Defrule Manager**:از این قسمت برای مدیریت(حذف و مشاهده) قواعد(**Rule**) استفاده می شود.
- **Deffacts Manager**: از این قسمت برای مدیریت(حذف و مشاهده) واقعیات(**Facts**) استفاده می شود.
- **DefTemplate Manager**: از این قسمت برای مدیریت(حذف و مشاهده) الگو(**Template**) استفاده می شود.
- **DefFunction Manager**: از این قسمت برای مدیریت(حذف و مشاهده) توابع(**Function**) استفاده می شود.

منوی Window

دستورات این منو برای مشاهده و مدیریت پنجره های نرم افزار بکار می رود.

- **Facts Window**:پنجره لیست **Fact** ها را نمایش می دهد.
- **Agenda Window**:پنجره مربوط به **Agenda** که قواعد فعال در آن قرار دارد را نمایش می دهد.

نشانه گذاری ها:

بعضی از نشانه گذاری ها در گرامر دستورات (**Syntax**) استفاده می شود که در این قسمت مقداری از آنها برای آگاهی بیشتر از **Syntax** ها توضیح داده می شود.

(example) : یعنی ساختار یعنی در داخل پرانتز هر دستور می تواند قرار بگیرد.

([1] example) :دستوراتی که بین [] قرار دارند یعنی اختیاری هستند در استفاده آنها اجباری نیست.

(example <INTEGER>) :دستوراتی که به این سبک نوشته می شود یعنی دستور یا مقداری جاگزین عبارت بین <> می شود.

***<INTEGER>** :زمانی که علامت * کنار دستوری وجود دارد یعنی تعداد ۰ یا بیشتر از این دستور میتوان استفاده کرد.

+<INTEGER>: زمانی که علامت + کنار دستوری وجود دارد یعنی تعداد ۱ یا بیشتر از این دستور میتوان استفاده کرد.

all | none | some : زمانی که بین دستورات علامت | باشد بدین معنا است که میتوان از هر کدام استفاده کرد معنی **OR** می دهد.

مولفه های پایه نرم افزار

- فیلد(**Field**)
- واقعیات(**Fact**)
- قواعد(**Rule**)
- الگو(**Template**)
- کلاس(**Class**)

فیلدها

برای ساخت پایگاه دانش باید اطلاعات از ورودی(صفحه کلید یا فایل) خوانده شود سپس دستورات اجرا شـوند.در طی فرایند اجرای برنامه، نرم افزار کلیپس نشانه ها(**Symbol**)،کاراکتر هایی که یک معنی می دهد را با یگدیکر ترکیب و گروه بندی می کند و یک توکن می سازد.در واقع فیلد یک نوع خاصی از توکن می باشد.

فیلد عددی

دو نوع فیلد عددی وجود دارد.

- عدد صحیح(**Integer**): مانند ۲۳۸ ، -۳۲
- عدد ممیز شناور(**Float**): مانند ۲۷۳E۳، ۱۵.۰۲ ، ۲۵.E۷-(**E** همان نماد علمی می باشد).

فیلد نشانه ای

- فیلد های نشانه ای حاوی کاراکتر های اسکی می باشند.
- این فیلد ها دارای محدودیت هایی می باشند.
- نشانه ها در کلیپس حساس به حروف(**Case Sensitive**) هستند.
- نشانه ها نمی توانند با کاراکتر های ؟ و ؟$ شروع شوند چون این ۲ کاراکتر برای معرفی متغیر ها رزرو شده اند.
- نمونه هایی از نشانه ها :**foo ، B76-HI، %-=+@**

محدودیت فیلد نشانه ای

فیلد های نشانه ای نمی توانند شامل سمبل های زیر باشند.

- کاراکتر های اسکی چاپ نشدنی(مانند کاراکتر فاصله(**Space**))
- دابل کوتیشین " "

- پرانتز با و بسته ( )
- امپرسند &
- خط عمودی |
- علامت کوچکتر <
- علامت تیلدا ~
- سمی کولن ;

فیلد رشته ای

فیلد های رشته ای در بین ۲ دابل کوتیشن " " قرار می گیرند.تنها محدودیت در این فیلد ها استفاده از بک اسلش \ می باشد.که برای چاپ بک اسلش از \\ و برای چاپ " از "\ استفاده می شود.در نمونه هایی آورده شده است.

- **"foo"**
- **"a and b"**
- **"a\"quote"**
- **"1 number"**

فیلد آدرس

این فیلد ها آدرس داده ها را نگه داری می کنند مانند زبان C.

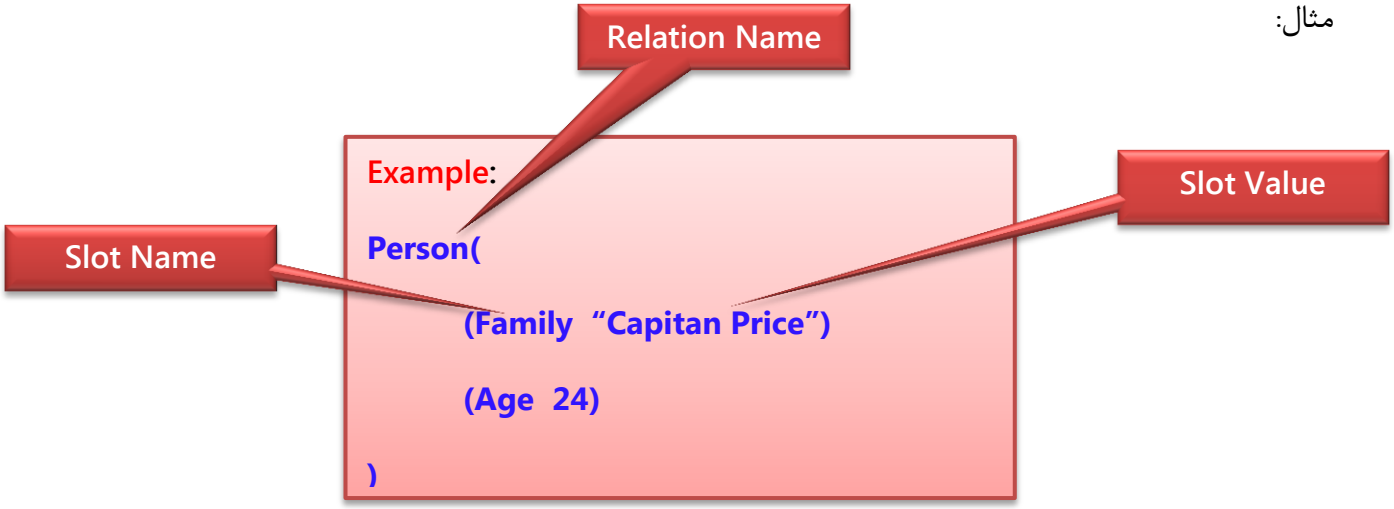
واقعیات(Fact)

Fact ها در واقع واقعیت هایی هستند که کاربربه عنوان ورودی به سیستم خبره به می دهد و از سیستم خبره ،خبرگی دریافت می کنند.کلیپس برای حل مسئله ها نیاز به اطلاعات و داده ها دارد.پایه واحد های داده که در قواعد(**Rule**) استفاده می شود همان **Fact** می باشد.هر **Fact** از قسمت های زیر تشکیل شده است.

- نام واقعیت(**Relation name**)
- تعداد صفر یا بیشتر اسلات(**Slot name**) به همراه مقدار(**Slot value**)

```
Syntax:
Relation Name(
        (SlotName1  SlotValue1)
        (SlotName2  SlotValue2)
)
```

مثال:



ساختار Deftemplate:

با استفاده از این دستور می توان ساختار **fact** ها را بصورت گروهی ایجاد نمود.باید دقت نمود که این دستور فقط ساختاراولیه **Fact** را می سازد و هیچ **Fact** را ایجاد نمی کند و باید با دستور **Assert** آن ساختار ایجاد شده توسط **Deftemplate** را مقدار دهی نماییم.

```
Syntax:
( deftemplate <relation-name> [<optional-comment>]
      <slot-definition>* )
   <slot-definition>
      (slot <slot-name>) | (multislot <slot-name>)
)
```

<relation-name>: نام الگو می باشد.

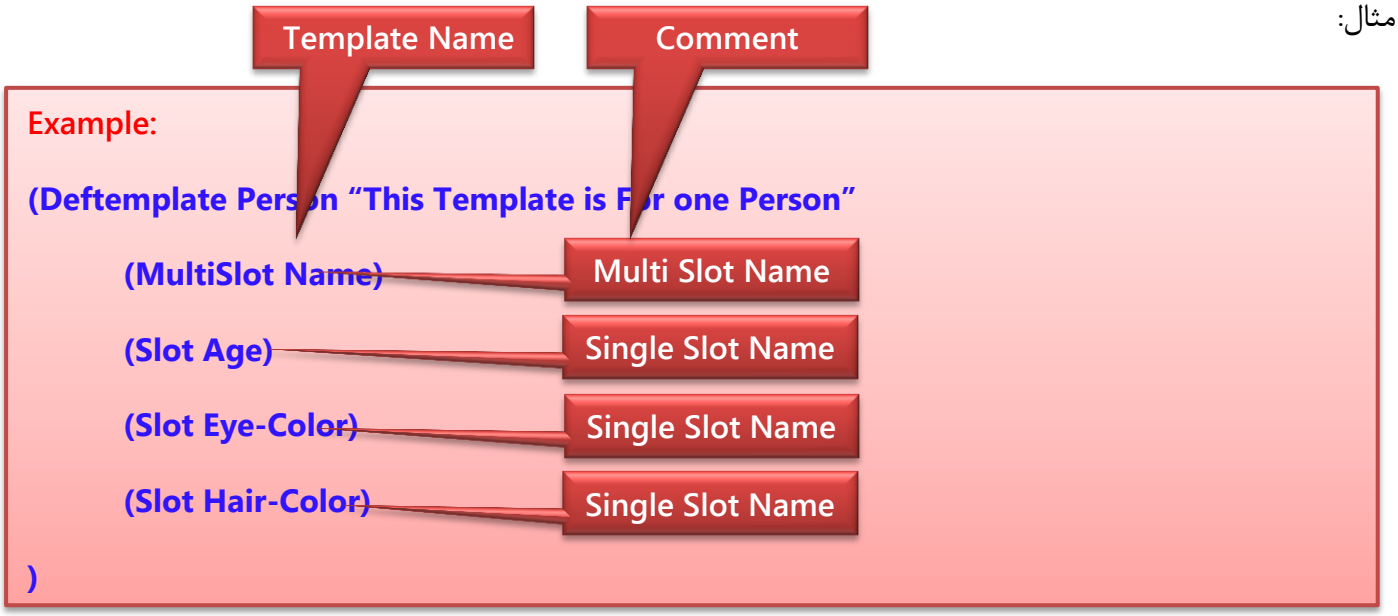
<optional-comment>: می توان در این قسمت توضیحاتی برای الگو قرار داد.این قسمت اختیاری می باشد.

(slot <slot-name>): این قسمت اجازه میدهد یک فیلد با یک مقدار ذخیره گردد.

(multislot <slot-name>): این فیلد اجازه می دهد چندین مقدار در یک اسلات ذخیره شود.

در مثال زیر یک فقط ساختار ایجاد شده است و تا زمان مقدار دهی با دستور **Assert** این الگو به **Fact** تبدیل نمی شود.

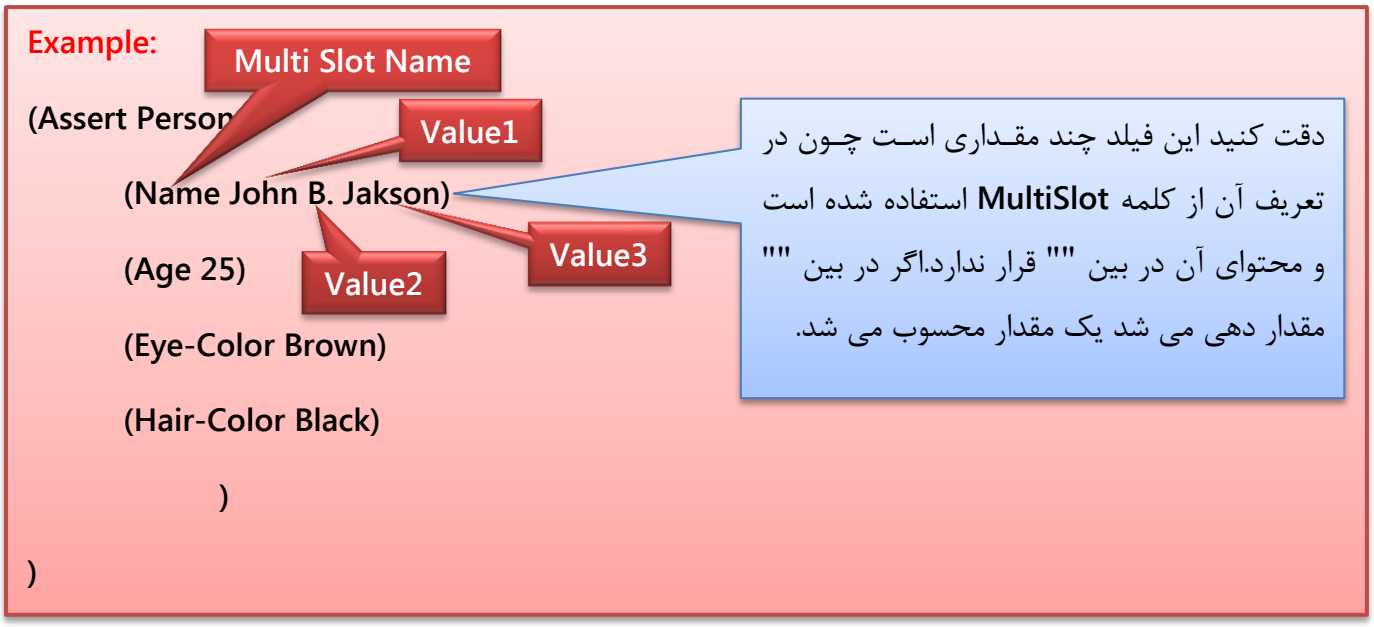
مثال:



دستور Assert

مقداردهی ساختار **Template** با دستور **Assert** انجام می شود.

```
Syntax:
(assert <Fact>)
```



در مثال فوق یک **fact** با نام **Person** که قبلا با دستور **DefTemplate** ساختار آن ایجاد شده است مقدار دهی می شود و در قسمت **Fact-list** اضافه می گردد.این **Fact** بصورت گروهی ایجاد شده است یعنی **Fact** با یک نام و چندین اسلات.

Person	
Name	John B. Jakson
Age	25
Eye-Color	Brown
Hair-Color	Black

با دستور **Assert** می تواند **Fact** هایی بصورت جداگانه نیز تعریف کرد که در این صورت نیـازی بـه تعریـف **Template** نمی باشد.

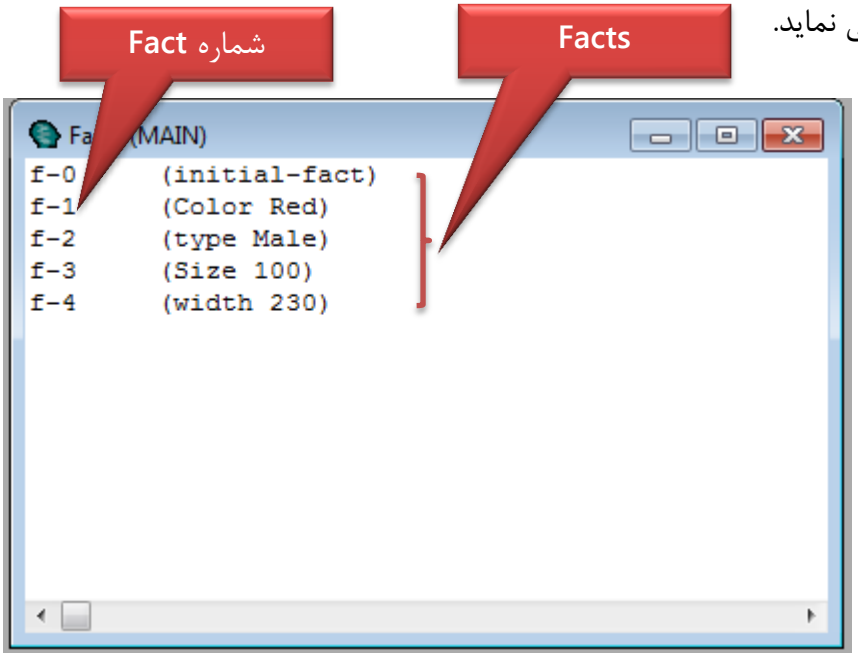
مثال.

```
Example:

(Assert(Color Red))
```

مثال فوق یک **Fact** با نام **Color** با مقدار **Red** به **Fact-List** اضافه می کند.

```
Example:

(Assert(Color Red)

        (Type Male)

        (Size 100)

        (width 230)

)
```

مثال فوق چهار**fact** به ترتیب به نام های **Color** و **Type** و **Size** و **width** بـا مقـادیر **Red** و**Male** و ۱۰۰ و ۲۳۰ بـه طور همزمان اضافه می نماید.



محدودیت ها برای اسلات

با اعمال محدودیت ها می توان **Slot** های یک **template** را مجبور به وارد کردن داده های با قالب خاص نمود و یا مقداری پیش فرض برای اسلات ها در نظر گرفت.

Syntax:

| <constraint-attribute> ::= <type-attribute>

| <allowed-constant-attribute>

| <range-attribute>

<cardinality-attribute>

<default-attribute>

(<type-attribute> ::= (type <type-specification>)

<type-specification> ::= <allowed-type>+ | ?VARIABLE>

| allowed-type> ::= SYMBOL | STRING | LEXEME>

| INTEGER | FLOAT | NUMBER

| INSTANCE-NAME | INSTANCE-ADDRESS

| INSTANCE | FACT-ADDRESS

EXTERNAL-ADDRESS

<allowed-constant-attribute>

::= (allowedsymbols <symbol-list>) |

(allowedstrings <string-list>) |

(allowed-lexemes <lexeme-list>) |

(allowedintegers <integer-list>) |

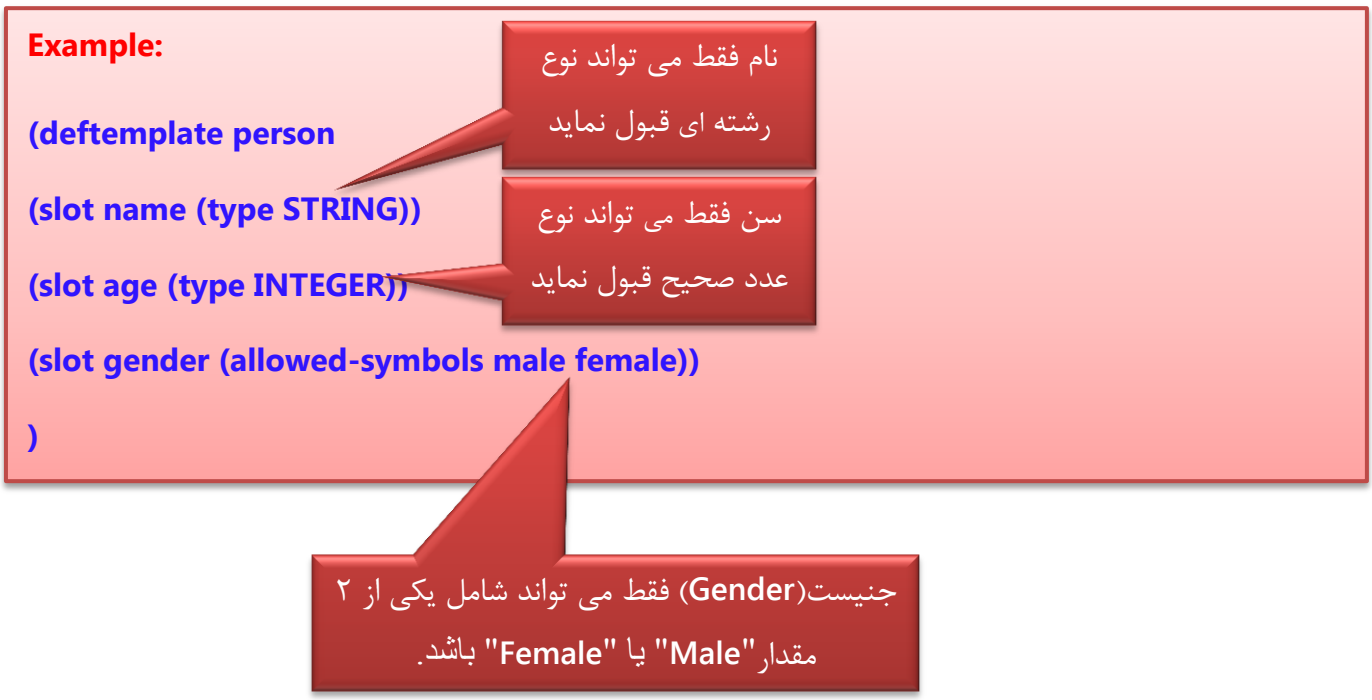
(allowedfloats <float-list>) |

(allowednumbers <number-list>) |

(allowed-instance-names <instance-list>) |

(allowedvalues <value-list>)

مثال:



به عنوان مثال در صورتی که سعی کنید مقداری بر خلاف محدودیت تعریف شده وارد نمایید با خطایی از سوی نرم افـزار کلیپس مواجه خواهید شد.

متغیر ها(Variable)

در نرم افزار کلیپس میتوان ۲ نوع متغیر تعریف نمود.

- متغیر تک مقداری(Single Value)
- متغیر چند مقداری(Multi Values)

Example:

?X

$?Name

ویلکارد ها(Wildcard)

ویلکاردها علامت هایی هستند که بجای هر کاراکتر یا کاراکتر خاصی بکار می روند.مانند * که در هنگام جستجو به معنای "هر" بکار می رود.

Syntax:

Example:

```
(assert
 (data 1 blue)
(data 1 blue red)
(data 1 blue red 6.9)
)
 (defrule find-data
(data ?x $?y ?z)
=>
(printout t "?x = " ?x crlf
          "?y = " ?y crlf
          "?z = " ?z crlf
          "------" crlf)
)
(run)
?x = 1
?y = (blue red)
?z = 6.9
------
?x = 1
?y = (blue)
?z = red
------
?x = 1
?y = ()
?z = blue
------
```

بدست آوردن شماره ایندکس Fact

برای انجام بعضی از عملیات ها نیاز به شماره ایندکس **Fact** می باشد به همین دلیل از روش زیر شماره ایندکس را بدست می آوریم.باید بدانیم که شماره ایندکس منظور همان شماره ای است که در پنجره **Fact-List** در سمت چپ **Fact** نشان داده می شود و این شماره را خود نرم افزار کلیپس بصورت اتوماتیک به **Fact** ها اختصاص می دهد.

```
Syntax:
<variable-symbol> <- <pattern-CE>
```

<variable-symbol>: نام یک متغیر است که شماره ایندکس در آن قرار می گیرد.

<pattern-CE>: نام **Fact** است که می خواهیم شماره ایندکس آنرا بدست بیاوریم.

-> : علامت انتساب می باشد.

مثال.

```
f-0 (initial-fact)
f-1 (data 1 blue)
f-2 (data 1 blue red)
f-3 (data 1 blue red 6.9)
```

در لیست بالا تعداد چهار **Fact** وجود دارد شماره ایندکس ۰ یعنی **f-0** که توسط خود نرم افزار کلیپس رزرو می شود و در واقع یک مقدار دهی اولیه انجام می دهد حتی اگر هیچ **Fact** تعریف نشده باشد این شماره به عنوان **f-0 (initial-fact)** توسط خود نرم افزار استفاده می شود و **Fact** ها از شماره ۱ به بعد شماره گذاری می شوند.

```
f-0 (initial-fact)
f-1 (data 1 blue)
f-2 (data 1 blue red)
f-3 (data 1 blue red 6.9)
```

نکته:این دستور در بین دستور **Defrule** یا دیگر دستورات کاربرد دارد.

```
Example:
?FactNumner1 <- (data 1 blue)
```

با توجه به مثال بالا مقدار ۱ در متغیر **FactNumner1** قرار میگیرد

حذف fact با دستور Retract

با استفاده از این دستور می توان یک **Fact** را حذف نمود.باید دقت نمود که این دستور با داشتن شماره ایندکس **Fact** می تواند آنرا حذف نماید که از روش قبل شماره ایندکس را بدست می آوریم.

Syntax:

```
(retract <retract-specifier>+ | *)
```

<retract-specifier>:شماره ایندکس **Fact** ای را که قرار است حذف نماییم، مشخص می کنیم.

Syntax:

```
(defrule Rule2
      ?FactNumner1 <- (Color red)
=>
     (retract  ?FactNumber1)
)
```

در این مثال یک قاعده با نام **Rule2** تعریف می شود و در قسمت **pattern** تعریف کردیم کـه اگـر **fact** بـه نـام **Color** دارای مقدار **red** وجود داشت شماره ایندکس آنرا در متغیری به نام **FactNumber1** که خودمان تعریف کرده این قـرار می دهد و در قسمت **Action** یا همان **Then** اگر آن **Fact** وجود داشته باشد با دستور **Retract** آن **Fact** یعنـی **Color** را حذف می کند.

نکته:در ادامه ساختار **Rule** توضیح داده خواهد شد.

- زمانی که یک **Fact** حذف می گردد شماره آن **Fact** دیگر به **Fact** های جدید که ایجاد می شود اختصـاص داده نمی شود و جای آن شماره خالی می باشد.
- با دستور (* **Retract**) همه **Fact** ها حذف می گردد.این دستور معادل دستور(**Clear**) می باشد.

تغییر Fact ایجاد شده با Template توسط دستور Modify

با استفاده از دستور **Modify** می توان مقدار **Fact** هایی که قبلا با دستور **Deftemplat** ایجاد شده را تغییر داد.

Syntax:

(modify <fact-index> <slot-name> <slot-value>)

<fact-index>:شماره ایندکس **Fact** مورد نظر را مشخص می نماید.

<slot-name>:نام اسلاتی که می خواهیم مقدار آنرا تغییر دهیم.

<slot-value>:مقدار برای اسلاید که می خواهیم مقدار آنرا تغییر دهیم.

Example:

(Modify 4 (Name Mohsen))

مثال فوق **Template** با شماره ایندکس ۴ محتوای اسلات **Name** را به مقدار **Mohsen** تغییر می دهد.

- دستور **Modify** برای تغییر نیاز به شماره ایندکس دارد که در قسمت های قبل روش بدست آوردن آن توضیح داده شده است.
- دستور **Modify** در برای اعمال تغییرات در واقع اول **Fact** را حذف می کند و سپس آنرا با مقدار جدید ایجاد می نماید باید دقت داشت که با اینکار یک شماره ایندکس جدید به **Fact** اختصاص میدهد.

کپی نمودن Fact ایجاد شده با Template توسط دستور Duplicate

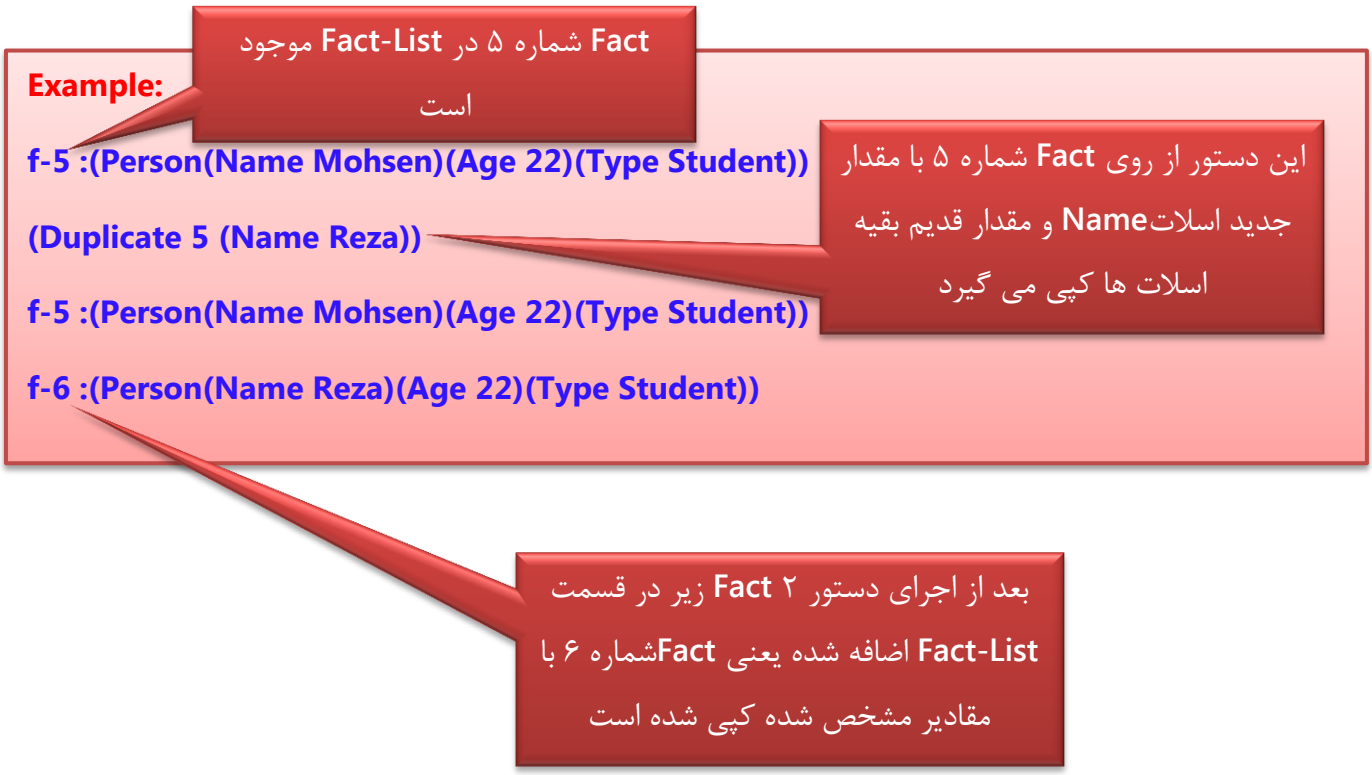
با این دستور از روی **Fact** هایی که قبلا با دستور **DefTemplate** ایجاد شده کپی گرفته می شود.

Syntax:

(duplicate <fact-Index> (<Slot-Name Slot-Value>)*)

<fact-Index>: شماره **fact** ای که می خواهیم اسلات های آنرا تغییر دهیم.

(<Slot-Name Slot-Value>)*: نام اسلات به همراه مقدار جدید آن را مشخص می نماید.



دستور Save-Fact جهت ذخیره Factها

با استفاده از این دستور می توان کلیه **Fact** های موجود (**Fact** ها در پنجره **Fact**) را در داخل فایل ذخیره نمود.

Syntax:

Save-Facts <FileName>

<FileName>: مسیر ذخیره فایل به همراه نام و پسوند فایل را مشخص می نماید.

Example:

(Save-Facts "C:\MyFacts.txt")

مثال زیر کلیه **Fact** های موجود را در فایلی به نام **MyFacts** با پسوند **txt** در درایو **C** ذخیره می کند.

دستور Load-Facts جهت باز نمودن Fact ها

با استفاده از این دستور می توان **Fact** هایی که قبلا توسط دستور **Save-Facts** در فایل ذخیره نموده ایم را بـه لیست **Fact-list** اضافه نماییم.

Syntax:

Load-Facts <FileName>

<FileName>: مسیر ذخیره فایل به همراه نام و پسوند فایل را مشخص می نماید.

Example:

(Save-Facts "C:\MyFacts.txt")

دستور فوق **Fact** هایی که قبلا درفایل مذکور(**MyFacts.txt**) ذخیره شده است را باز نموده و به پنجره **Facts** اضافه می نماید.

تعریف قواعد(Rules)

با این دستور می توان قواعد را تعریف نماییم. نرم افزار کلیپس یک برنامه **Rule-Based** نیز می باشد یعنی بر مبنای قواعد نیزکار می کند.

Syntax:

```
(defrule <rule-name> [<optional comment>]
   <patterns>*
   =>
   <actions>*
  )
```

<rule-name>:این قسمت شامل نام قاعده یا **Rule** می باشد.

<optional comment>:در این قسمت می توان توضیحاتی در مورد آن قاعده نوشت .توضیحات اختیاری می باشد.

<patterns>:این قسمت الگو را مشخص می نماید یعنی در واقع همان قسمت الگو عبارت شرطی می باشد.

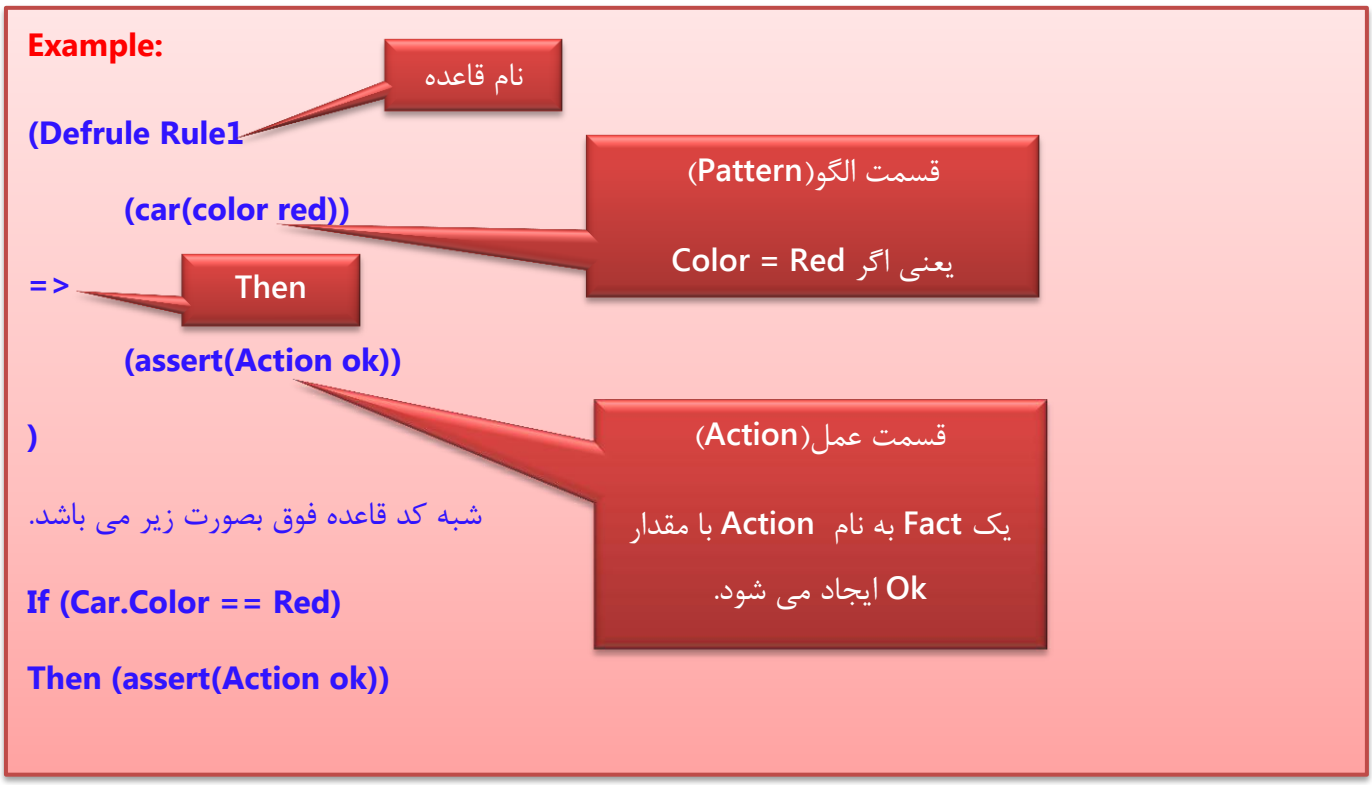
<actions>:این قسمت اعمال و دستوراتی می باشد که در صورت برقرار شدن(**True**) قسمت **Pattern** اجرا می شود.

نکته:علامت * در **Syntax** ها به معنی تعداد ۰ یا بیشتر می باشد*<patterns> یعنی چندین الگو می توان تعریف نمود.

Syntax:

```
If pattern then Action
```

مثال



یعنی اگر **fact** با نام **car** مقدار فیلد **Color** آن **Red** بود سپس یک **Fact** با دستور **Assert** به نام **Action** ایجاد کن و مقدار آنرا **ok** قرار بده.

نکته:علامت => معادل کلمه کلیدی **Then** در شبه کد می باشد

ایجاد Fact گروهی با دستور Deffacts

با استفاده از این دستور می توان **Fact** هایی بصورت گروهی تعریف کرد و به عنوان دانش اولیه با پایگاه دانش اضافه می شود.

Syntax:

```
(deffacts <deffacts-name> [<comment>]
  <pattern>*)
```

<deffacts-name>:نام **fact** را مشخص می کند.

نکته:**Fact** هایی که با دستور **Deffact** تعریف می شوند در پنجره **Facts** نمایش داده نمی شوند.

[<comment>]:توضیحاتی برای **Fact** می باشد که نوشتن این توضیحات اختیاری می باشد.

<pattern>:نام الگو یا همان **Fact** ها را مشخص مینماید.

مثال

```
Example:
(deffacts BaseFact1
        (Color Red)
        (type Stuff)
        (Height 100)
)
```

مثال فوق **Fact** با نام **BaseFact1** شامل فیلد های اولیه **Color,type,Height** را به عنوان دانش اولیـه بـه پایگـاه داده اضافه می نماید.

چاپ رشته / متغیر با دستور Printout

```
Syntax:
(printout <logical-name>  <print-items>*)
```

<logical-name>:در این قسمت می توان نوع خروجی را مشخص نمود بـا کـاراکتر **t** (**Terminal**)مـانیتور بـه عنـوان خروجی در نظر گرفته می شود.

<print-items>:در این قسمت متن مورد نظر با آیتم هایی که قرار است چاپ شوند را مشخص می نماییم.

نکته:عبارت **crlf** تعیین می کند که بعد از چاپ نمودن آیتم ها مکان نما به خط بعدی رود.این کلمـه معـادل **\n** در زبـان **c++** میباشد.

```
Example:
(printout t "Hello WelCome to Clips Software" crlf))

(printout t "The Value of X is " ?x crlf))
```

نمایش لیست دستورات

با استفاده از دستورات زیر می توان لیست بعضی از ساختارهایی که تا بحال یاد گرفته ایم را مشاهده نماییم.

(list-defrules): این دستور لیست قواعد موجود را نمایش می دهد.

(list-deftemplates):این دستور لیست الگو های موجود را نمایش می دهد.

(list-deffacts):این دستور لیست **Fact** هایی راکه با دستور **Deffacts** تعریف کرده ایم را نمایش می دهد.

(Facts):این دستورات لیست تمام **fact**(**Fact** های موجود در پنجره **Facts** که با دستورات **assert** یا **Deftemplate** اضافه نموده ایم) را نمایش می دهد.

نمایش ساختار دستورات

با استفاده از دستورات زیر می توان ساختار یعنی همان سورس کد بعضی از دستوراتی که تا بحال یاد گرفته ایم را نمایش می دهد.

نمایش ساختار قواعد(ppdefrule)

با این دستور سورس و ساختار قواعد را به همان صورت که یا دستور **Defrule** تعریف نموده ایم نمایش می دهد.

Syntax:

(ppdefrule <defrule-name>)

<defrule-name>:نام قاعده مورد نظر را مشخص می کند.

Example:

(ppdefrule rule1)

دستور فوق ساختار قاعده**rule1** را نمایش می دهد.

یادآوری:قواعد را با دستور **Defrule** تعریف می نماییم.

نمایش ساختار الگو(ppdefTemplate)

با این دستور سورس و ساختار الگوهایی که با دستور **Deftemplate** تعریف نموده ایم را نمایش می دهد.

Syntax:

(ppdeftemplate <deftemplate-name>)

<deftemplate-name>:نام الگوی مورد نظر را مشخص می کند.

Example:

(ppDeftemplate person)

دستور فوق ساختار و سورس الگویی به نام **person** را نمایش می دهد.

یادآوری:الگو ها را با دستور **DefTemplate** تعریف می نماییم

نمایش ساختار Fact ها(ppDeffacts)

با این دستور سورس و ساختار الگوهایی که با دستور **DefFacts** تعریف نموده ایم را نمایش می دهد.

Syntax:

(ppdeffacts <deffacts-name>)

<deffacts-name>:نام **Fact** مورد نظر را مشخص می کند.

Example:

(ppdefFacts Animal)

دستور فوق ساختار و سورس **Fact** به نام **Ainmal** را نمایش می دهد.

یادآوری: **Fact** ها را با دستور **Deffacts** تعریف می نماییم.باید توجه داشت **Fact** هایی که با این دستور تعریف می شوند به عنوان دانش اولیه پایگاه دانش در نظر گرفته می شوند.

نمایش ساختار fact ها(ppFact)

با این دستور سورس و ساختار **fact** هایی که با دستور **Assert** تعریف نموده ایم را مشخص می نماید.

Syntax:

(ppfact <Fact-Index>)

<Fact-Index>: شماره ایندکس **fact** را که می خواهیم ساختارش را مشاهده نماییم مشخص می نماییم.

Example:

(ppfact 3)

دستور فوق ساختار و سورس **fact** شماره ۳ را نمایش می دهد.

نکته:برای بدست آوردن شماره ایندکس می توانید از تکنیکی که در قسمت های قبل توضیح داده شده استفاده نمایید.

حذف نمودن ساختار ها

با استفاده از دستورات زیر می توانید ساختاری هایی که تا بحال آموخته ایم را حذف نمایید.

حذف قواعد با دستور (undefrule)

با این دستور قواعدی را که با دستور **Defrule** تعریف و ایجاد نموده ایم را حذف می نماید.

Syntax:

(undefrule <defrule-name>)

<defrule-name>:نام قاعده مورد نظر را مشخص می نماید.

Example:

(undefrule Rule1)

دستور فوق قاعده با نام **Rule1** که قبلا توسط دستور **Defrule** تعریف نموده ایم را حذف می کند.

حذف الگوها با دستور (undeftemplate)

با این دستور الگوهایی را که با دستور **DefTemplate** تعریف و ایجاد نموده ایم را حذف می نماید

Syntax:

(undeftemplate <deftemplate-name>)

<deftemplate-name>:نام الگو مورد نظر را مشخص می کند.

Example:

(UndefTemplate Person)

دستور فوق الگویی را که با نام **Person** را که قبلا توسط دستور **Deftemplate** تعریف نموده ایم را حذف می کند.

حذف Fact با دستور(undeffacts)

با این دستور **fact** هایی را که با دستور **Deffacts** تعریف و ایجاد نموده ایم را حذف می کند.

Syntax:

(undeffacts <deffacts-name>)

<deffacts-name>:نام **fact** مورد نظر را مشخص می کند.

Example:

(undeffacts Animal)

دستور فوق یک **fact** را با نام**Animal** که قبلا توسط دستور **Deffacts** تعریف نموده ایم را حذف می کند.

حذف Fact با دستور retract

این دستور در قسمت های قبل توضیح داده شده است.به یاد داشته باشید که این دستور برای حذف **Fact** هایی که با دستور **Assert** ایجاد نموده ایم می باشد و همچنین برای حذف آن **Fact** بجای نام **Fact** نیاز به ایندکس **Fact** دارد.

دستورات اساسی

(clear):این دستور تمام **Fact** ها و **Rule** ها را از حافظه کاری حذف می کند معادل این است یکبار از نرم افزار خارج شویم و دوباره وارد شویم(**Shutdown + Restart**).

(reset):این دستور اطلاعات مربوط به **Fact** ها را حذف میکند و **Agenda** را ریست می کند.

(**Agenda**): قواعد فعال را نمایش می دهد.

(Refresh): این دستور قواعد را بروز رسانی می نماید.

(Run [<Limit>]):با این دستور قواعد فعال در حافظه کاری اجرا می شود که در قسمت <Limit> تعداد قواعـدی را که می خواهیم اجرا شود را مشخص می کنیم.مشخص نمودن تعداد اختیاری می باشد.در صورتی که تعداد را مشـخص ننماییم یعنی(Run) در این صورت کلیه دستورات فعال در پنجره Agenda اجرا می شوند.

بعضی توابع سودمند

تابع save

با این تابع مجموعه از ساختارهایی که وجود دارند(Fact و قواعد و الگوها و ...) را در فایلی ذخیره می نماید.

Syntax:

(save <file-name>)

<file-name> : مسیر و نام فایل به همراه پسوند آن را مشخص می نماید.

Example:

(save "c:\ball.txt")

تابع Load

توسط این دستور می توان ساختار هایی که قبلا ذخیره نوده ایم(Save) را باز نماید و آماده اجرا کند.

Syntax:

```
(load <file-name>)
```

<file-name> : مسیر و نام فایل به همراه پسوند آن را مشخص می نماید.

Example:

(load "c:\ball.txt")

تابع Open

با استفاده از این تابع می توان فایل برنامه کلیپس که قبلا ذخیره نموده ایم را باز نماییم.

Syntax:

(open <file-name> <logical-name> [<mode>])

<file-name>: این قسمت شامل نام فایل به همراه آدرس کامل و پسوند فایل می باشد.حتما اگر در مسیر علامت بک اسش وجود دارد باید آنرا مشخص نماییم.

نکته:برای مشخص نمودن علامت \ باید این علامت را ۲ بار پشت سر هم تایپ نماییم تا برای کلیپس معنی \ این بدهد یعنی \\ را تایپ می کنیم.

<logical-name>: این قسمت یک نام منطقی می باشد که قبلا در برنامه جاری نباید استفاده شده باشد.

[<mode>]:این قسمت نحوه دستیابی به فایل را مشخص می نماید.این قسمت اختیاری می باشد.

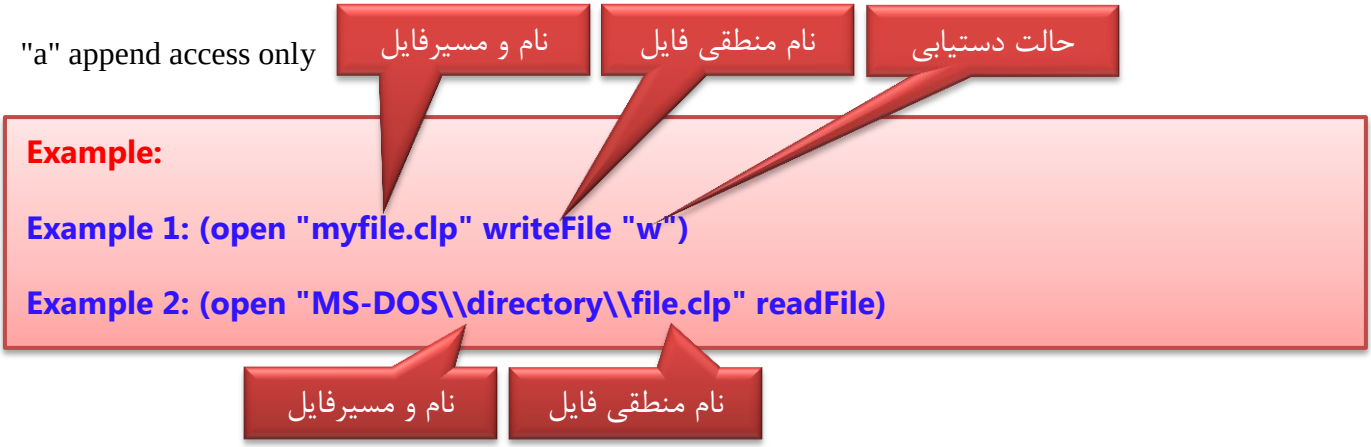
انواع حالت های Mode

"r" read access only

"w" write access only

"r+" read and write access

"a" append access only



در صورتی که تابع **Open** با موفقیت اجرا شود مقدار **True** و در غیر اینصورت مقدار **False** را بر می گرداند.

تابع Close

این تابع بر عکس تابع **Open** کار میکند و فایلی را که قبلا با تابع **Open** باز شده است را می بندد.

Syntax:

(close [<logical-name>])

[<logical-name>]: نام منطقی فایل است که هنگام **Open** کردن به آن اختصاص دادیم.

- اگر تابع **Close** بدون پارامتر فراخوانی شود تمام فایل های باز را می بندد.
- در صورتی که تابع **Open** با موفقیت اجرا شود مقدار **True** و در غیر اینصورت مقدار **False** را بر می گرداند.

تابع Read

با استفاده از این تابع می توان مقداری را از صفحه کلید دریافت نمود یا محتویات یک فایل را خواند.

Syntax:

(read [<logical-name>])

[<logical-name>]: نام منطقی فایل است که هنگام **Open** کردن به آن اختصاص دادیم.این پارامتر اختیاری است و اگر فایل را مشخص نکنیم مقداری را از صفحه کلید دریافت می کند.

Example:

```
CLIPS> (open "data.txt" mydata)
TRUE
CLIPS> (read mydata)
red
CLIPS> (read mydata)
green
CLIPS> (read mydata)
EOF
CLIPS> (close)
TRUE
CLIPS>
```

تابع Bind

با استفاده از این تابع یک مقدار را به یک متغیر منتقل می نماییم.

Syntax:

(bind <variable> <expression>*)

<variable>:نام یک متغیر را مشخص می نماید.

<expression>:یک عبارت را که قرار است در متغیر قرار گیرد مشخص می نماید.

Example:

(bind ?x "Hello")

مقدار Hello در متغیر X قرار می گیرد

(bind ?num1 (read))

مقداری را که کاربر از صفحه کلید وارد کند در متغیر num1 قرار می دهد.

مولفه های شرطی

- پیوند منطقی AND,OR,NOT برای ترکیب Pattern ها
- عملگرهای شرطی(&,|,~) بروی محتوای فیلدها
- عمگر شرطی If بر روی متغیر ها
- عملگر Test

Example:

not ~ **شامل این مقدار نمی باشد**

(name ~Simpson)

(age ~40)

or | **می تواند شامل یکی از این مقادیر باشد.**

(name Simpson|Oswald)

(age 30|40|50)

and & **باید شامل همه مقادیر باشد**

(name ?name&~Simpson)

(name ?name&Harvey)

expr. : **یک عبارت را بصورت محدودیت مشخص می شود**

سن باید بزرگتر از ۲۰

(age ?age&:(> ?age 20))

اولین حرف متغیر نام باید برابر S باشد

(name ?name&:(eq (sub-string 1 1 ?name) "S")

عملگر منطقی AND,OR,NOT برای ترکیب Pattern ها

از عملگرهای شرطی در ترکیب **Pattern** ها در بین **Rule** ها استفاده می شود.

عملگر OR

این پیوند مانند عملگر "**یا منطقی**" می باشد یعنی حداقل یکی از حالات اتفاق بیفتد.

Syntax:

(or <conditional-element>+)

(or (pattern1) (pattern2) ...)

Pattern1 OR Pattern2

<conditional-element>: عملوند های شرطی را مشخص می نماید

عملگر AND

این عملگر مانند عملگر "**و منطقی**" می باشد و هنگامی استفاده می گردد که همه حالات باید اتفاق بیفتند.این عملگر بصورت پیش فرض انتخاب شده است.یعنی اگر بین **Pattern** ها عملگری مشخص نکنیم پیش فرض **AND** میباشد.

Syntax:

(And <conditional-element>+)

(And (pattern1) (pattern2) ...)

Pattern1 AND Pattern2

<conditional-element>: عملوند های شرطی را مشخص می نماید.

Example:

```
(defrule system-fault
(error-status unknown)
(temp high)
=>
(printout t "The system has a fault." crlf))
```

عملگر NOT

این عملگر مانند **"نقیض منطقی"** می باشد معنی نقیض می دهد.

```
Syntax:

(not <conditional-element>)
```

<conditional-element>: عملوند های شرطی را مشخص می نماید.

```
Example:

(defrule high-flow-rate
(temp high)
(valve open)
(not (error-status confirmed))
=>
(printout t "Recommend closing of valve due to high temp"crlf))
```

عمگر شرطی IF-Else

این دستور همانند دستور **IF** در دیگر زبان های برنامه نویسی می باشد.

```
Syntax:

(if <expression>
then
<action>*
[else
<action>*])
```

<expression> : عبارت مورد ارزیابی را مشخص می نماید که خروجی این قسمت یک عبارت منطقی **True** یـا **False** میباشد در صورتی که عبارت برقرار باشد خروجی **True** و **Action** مربوط به **Then** انجـام مـی شـود در غیـر اینصـورت **Action** مربوط به **Else** اجرا می شود.قسمت **Else** اختیاری می باشد.

<action>:عملی که قصد اجرا شدنش را در قسمت **Then** یا قسمت **Else** داریم مشخص می نماید.

Example:

```
(defrule closed-valves
(temp high)
(valve ?v closed)
=>
(if (= ?v 6)
then
(printout t "The special valve " ?v " is closed!" crlf)
(assert (perform special operation))
else
(printout t "Valve " ?v " is normally closed" crlf)))
```

دستور Switch

این دستور معادل دستور **Switch** در زبان برنامه نویسی **C** می باشد که در این حالت از بین چند حالت یک حالت انتخاب می شود و دستور مربوط به آن اجرا می شود.

Syntax:

```
(switch <test-expression>
<case-statement>
<case-statement>+
[<default-statement>])
<case-statement> ::=
(case <comparison-expression> then <action>*)
<default-statement> ::= (default <action>*)
```

<test-expression>: عبارتی که قرار است مورد ارزیابی قرار گیرد را مشخص می نماید.

<case-statement>:حالت هایی که در صورت منطبق بودن یکی از حالات با عبارت مورد ارزیابی اجرا میشود.

[<default-statement>]:مقدار پیش فرض که در صورت عدم تطبیق هیچ یک از حالات با عبارت مورد ارزیابی اجرا می شود.این قسمت اختیاری می باشد.

Example:

```
CLIPS> (defglobal ?*x* = 0)
CLIPS> (defglobal ?*y* = 1)
CLIPS>
(deffunction foo (?val)
(switch ?val
(case ?*x* then x1)
(case ?*y* then y1)
(default none)))
CLIPS> (foo 0)
X1
CLIPS> (foo 1)
Y1
CLIPS> (foo 2)
none
CLIPS>
```

تعریف متغیر *x* بصورت عمومی و با مقدار اولیه ۰

تعریف متغیر *y* بصورت عمومی و با مقدار اولیه ۱

تعریف تابعی به نام foo در ادامه تعریف تابع توضیح داده خواهد شد

متغیر ?val مورد ارزیابی قرار میگیرد

اگر ?val برابر مقدار متغیر *x*? بود(یعنی ۰) عبارت x1 را Return می کند

اگر ?val برابر مقدار متغیر *y*? بود(یعنی 1) عبارت y1 را Return می کند

مقدار پیش فرض را مشخص میکند اگر هیچ کدام از مقادیر منطبق نباشد مقدار none برگشت داده می شود.

دستور Test

این دستور جز دستورات شرطی می باشد.

Syntax:

(test <function-call>)

<function-call>:این قسمت عبارت مورد ارزیابی را مشخص می نماید.

Example:

```
(Defrule Rule2
        (Test( >= ?b 2))
=>
        (printout t "The value Of 2 is 2" crlf)
)
```

دستورات تکرار

- تعداد تکرار مشخص باشد(For)
- تعداد تکرار نا مشخص باشد (While)

تعداد تکرار مشخص باشد(For)

از این حلقه زمانی استفاده می گردد که تعداد تکرار مشخص باشد.این دستور معادل دستور **For** در زبـان **C** مـی باشد.

Syntax1:

(loop-for-count <range-spec> [do] <action>*)

<range-spec> ::= <end-index> |

Syntax1:

(<loop-variable> [<start-index> <end-index>])

<start-index> ::= <integer-expression>

<end-index> ::= <integer-expression>

<range-spec>: تعداد تکرار حلقه را مشخص می کند.

<loop-variable>:متغیر که مقدار شماره حلقه را نگه داری می کند.

<start-index>:مقدار شروع حلقه که از نوع صحیح عددی می باشد.

<end-index>: مقدار پایان حلقه که از نوع صحیح عددی می باشد.

Example 1:

```
(loop-for-count 4 (printout t “Clips ”))
```

Clips Clips Clips Clips

Example 2:

```
(loop-for-count (?cnt1 2 4) do
(loop-for-count (?cnt2 1 3) do
(printout t ?cnt1 " " ?cnt2 crlf)))
2 1
2 2
2 3
3 1
3 2
3 3
4 1
4 2
4 3
```

تعداد تکرار نا مشخص باشد (While)

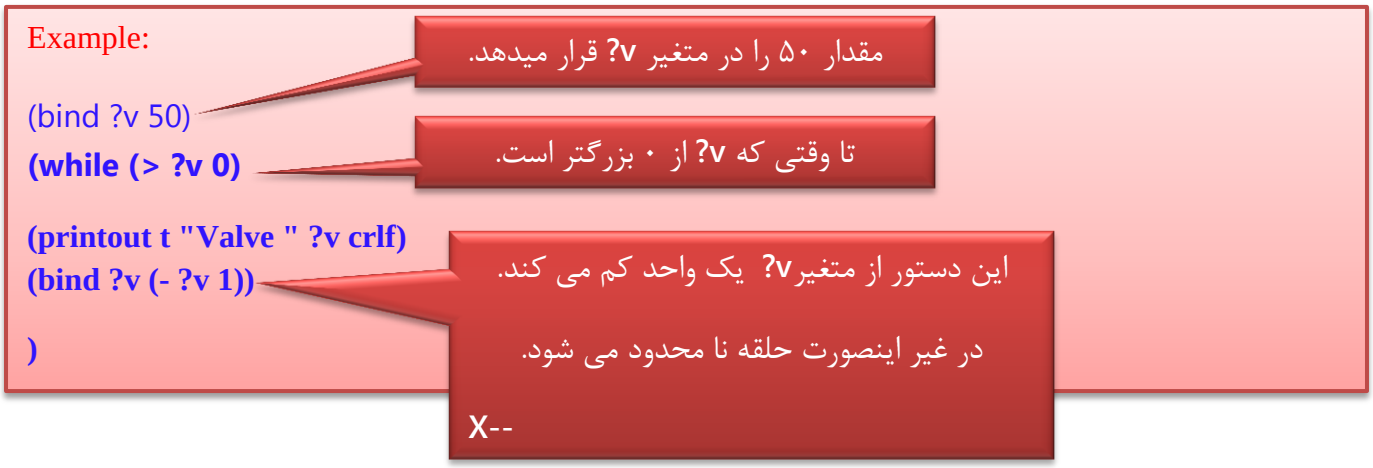
این حلقه زمانی استفاده می گردد که تعداد تکرار مشخص نباشد.معادل دستور**While** در زبان **C** می باشد.

Syntax1:

```
(while <expression> [do]
<action>*)
```

<expression>: عبارت مورد ارزیابی یاهمان شرط حلقه می باشد.

<action>:عملی که باید در بین حلقه انجام شود را مشخص می نماید.



توابع در کلیپس

توابع در کلیپس با دستور **Deffunction** تعریف می گردد و معادل توابع در زبان برنامه نویسی **C** میباشد.

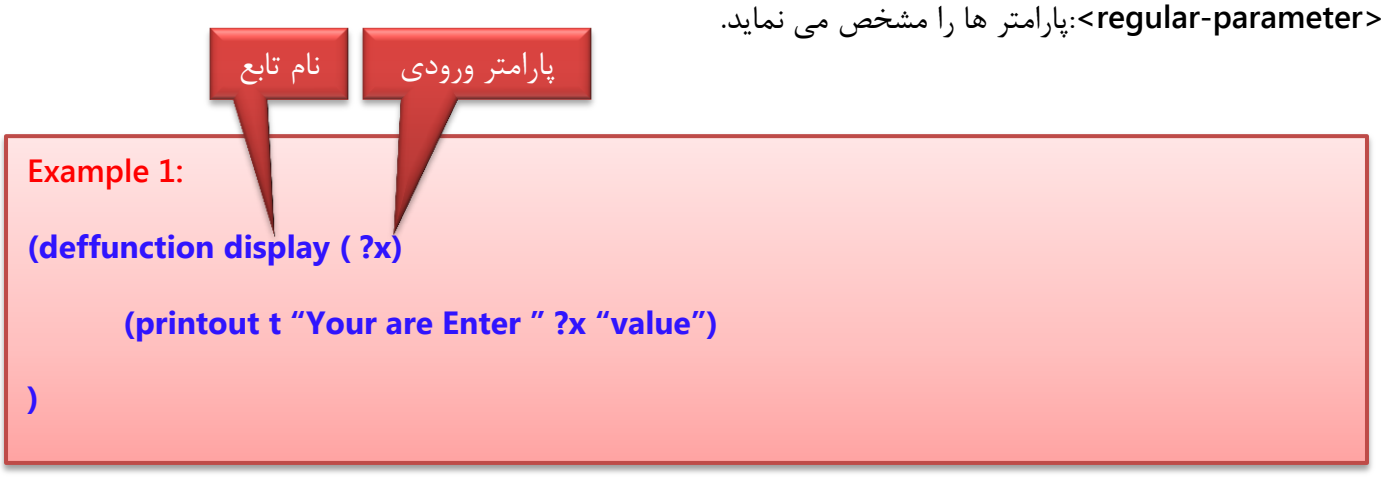
Syntax:

```
(deffunction <name> [<comment>]
(<regular-parameter>* [<wildcard-parameter>])
<action>*)
```

<name>:نام تابع را مشخص می نماید.

[<comment>]:توضیحات را مشخص می نماید.توضیحات اختیاری می باشد.

<regular-parameter>:پارامتر ها را مشخص می نماید.



Example 2:

نام تابع

پارامتر ورودی

```
(deffunction Add ( ?x ?y)
        (printout t "Add is" ( + ?x ?y))
)
```

X+y

محاسبات عددی

محاسبات در کلیپس بصورت پیشوندی(**Prefix**) انجام می گردد.

Syntax:

(<operator> <operand>*)

<operator>:عملگر های ریاضی را مشخص می نماید مانند + ، - ، *، /

<operand>:عملوند ها را مشخص میکند.

Example:

(+ 2 3)

2+3 بصورت پیشوندی است

(+ 2 (* 4 6))

ابتدا مقدار(۴ * ۶) = ۲۴ محاسبه می شود سپس جواب آن با ۲ جمع می گردد.جواب نهایی برابر است با ۲۶

(* ?x 2)

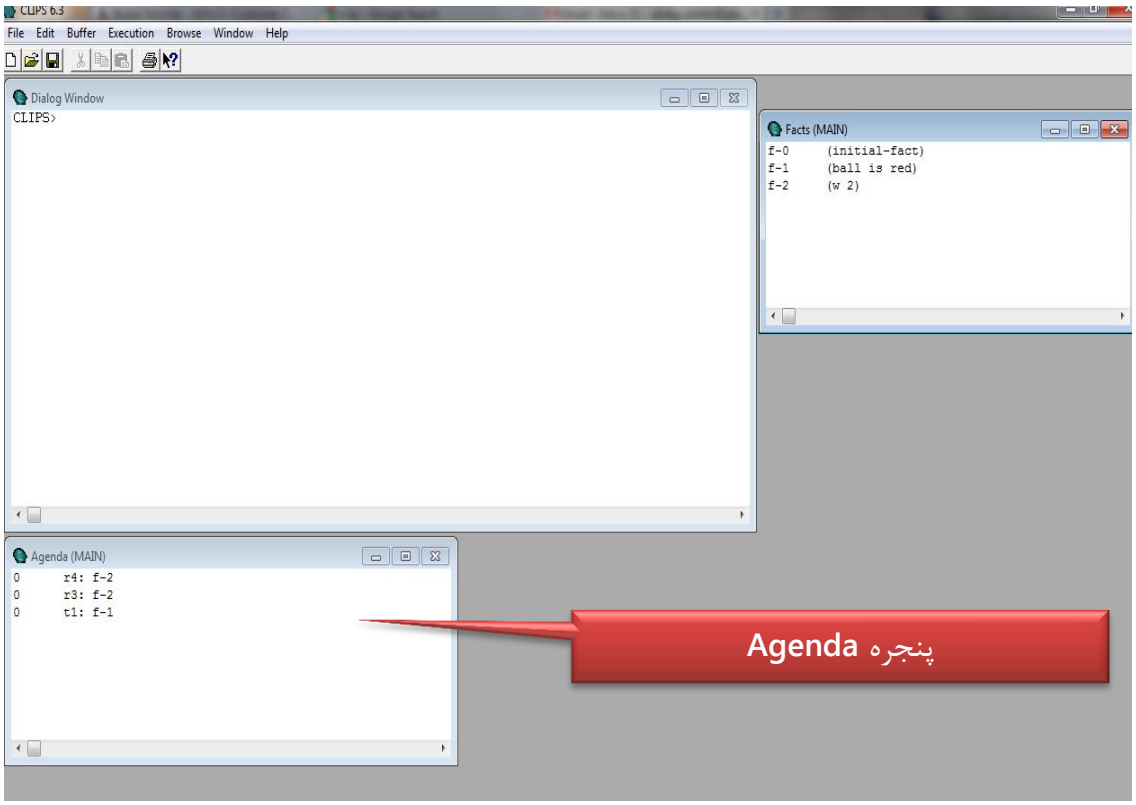
محتوای x و ۲ را با هم ضرب میکند.

پنجره agenda

هنگامی که قسمت الگوی **Rule** ها با وقایع موجود در **fact-list** به اصطلاح **match** شود ،این قاعده ارضاء شده و عنوان یک **Rule** فعال در این پنجره قرار می گیرد توجه داشته باشید که تنها این قاعده فعال شده است ، اما قسمت **Action** آنها اجرا نشده است، هنگامی که با دستور **Run** برنامه اجرا شود قسمت **Action** آنها اجرا شده و در اصلاح **Rule** موجود **fire** می شود ،حال اگر با **fire** شدن **Rule** های موجود، **fact**های جدیدی به لیست ما اضافه شود که با الگوی **rule**های دیگر **match** شود آنها هم فعال و اجرا می شود،البته این موضوع بستگی به مقداری است که در هنگام اجرای دستور **run** داده ایم به عنوان مثال اگر ما دستور **(Run 3)** را اجرا کنیم تنها ۳ قاعده شلیک می شود ، و بقیه **rule** ها در این پنجره فعال باقی می مانند ، ولی اگر دستور **(run)** را بدون مقدار وارد کنیم تمامی **Rule** های فعال **Fire** ، می شوند .

دستور(**refresh <Rule-name>**) باعث می شود که قاعده ای که **fire** شده است و از پنجره **agenda** خارج شده است دوباره به حالت فعال برگردد و از حالت **fire** خارج شود.

دستور (**reset**) همانند دستور **clear** است با این تفاوت که **fact** هایی که به صورت گروهی به عنوان دانش اولیه با دستور **Deffacts** تعریف کرده ایم به برنامه اضافه می شوند



پست الکترونیک تهیه کنندگان

Alireza.Alallah@Gmail.Com

MohsenAbiri@Gmail.com