CLIPS icon Untuk memulai CLIPS, double-click pada file CLIPSWin.exe.  Akan muncul sebuah window yg di dalamnya kosong dan hanya command prompt CLIPS>. Di sinilah dimana kita menuliskan perintah. Sebagai contoh untuk keluar dari CLIPS, tuliskan **(exit)** seperti halnya aplikasi2 windows lain. Perhatikan bahwa perintah2 CLIPS selalu berada di dalam kurung. Berikut beberapa perintah penting:

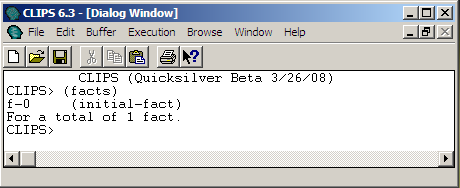
|  |  |
| --- | --- |
| **(exit)** | Keluar dari (Shuts down) CLIPS |
| **(clear)** | Menghapus semua rules dan facts dari memory. Ini sama dengan shutting down dan restarting CLIPS. |
| **(reset)** | Menghapus info facts dari memory (tetapi rules tidak) |
| **(run)** | Menjalankan program CLIPS |

**Fakta**

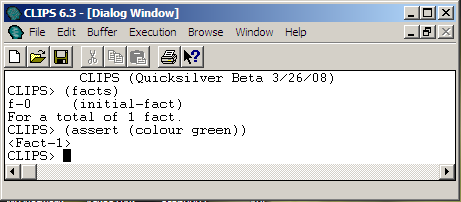
Pada hal yang paling mendasar, CLIPS beroperasi dengan memelihara daftar fakta dan rules (yang beroperasi pada data tsb). Suatu fakta adalah sepotong informasi misalnya (colour green) atau (orangtua\_dari Susan Johan). Fakta dibentuk dengan menggunakan perintah assert.

Coba lakukan:

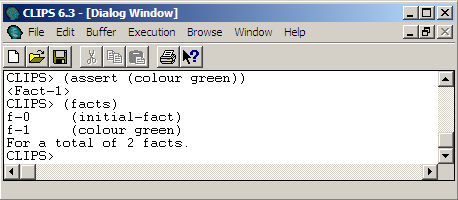
* Masuk (click double CLIPSWin.exe)
* (exit)
* Masuk (click double CLIPSWin.exe)
* Menampilkan fakta dengan perintah (facts)



* Menambah data colour green

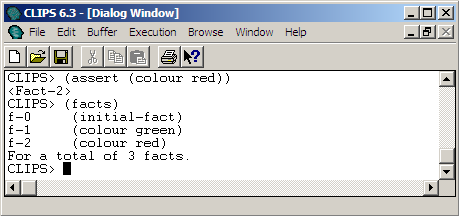


* Menampilkan fakta

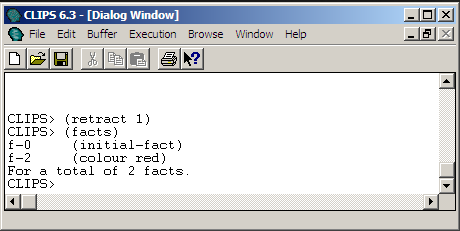


Untuk menghapus fakta gunakan perintah retract dengan menyebutkan index ke berapa fakta tersebut, dan bukan fakta itu sendiri. Coba lakukan hal berikut:

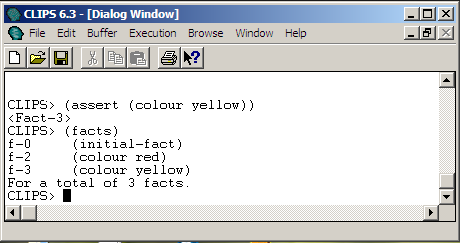
* Tambahkan colour red, tampilkan fakta.



* Hapus colour green, tampilkan fakta



* Tambahkan colour yellow, tampilkan fakta



Perhatikan bahwa fakta baru tidak menggantikan tempat dari fakta yang terhapus sebelumnya, pada contoh ini colour yellow berada di tempat fakta-3 dan bukan di fakta-1. Dengan perkataan lain *fact-indices are not reused*.

**Rules**

Fakta jika hanya seperti itu hanya digunakan/berguna terbatas. Untuk lebih diberdayakan, diperlukan rules untuk mengembangkan program dengan kemampuan fungsi. Secara umum, suatu rule diekspresikan dalam bentuk:

‘If something is true then do some action’

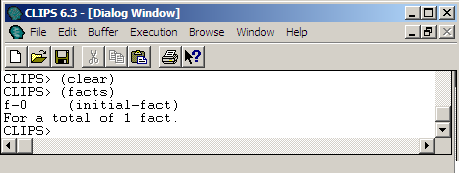
Rules seperti ini dikenal sebagai “production rules”. CLIPS itu sendiri kependekan dari C Language Integrated Production System. Berikut adalah ilustrasi bagaimana membentuk rules pada CLIPS.

|  |  |
| --- | --- |
| (defrule duck  (animal-is duck)  =>  (assert (sound-is quack))) | Rule terdiri dari 3 bagian:   * Pertama: defrule duck mendefinisikan suatu rule bernama duck * Kedua: bagian If dari rule (sebelah kiri tanda panah) * Ketiga: bagian then (sebelah kanan tanda panah) |

Rule duck di atas berarti bila terdapat sebuah fakta (animal-is duck) pada database fakta, maka tambahkan sebuah fakta (sound-is quack) ke dalam fakta database tersebut.

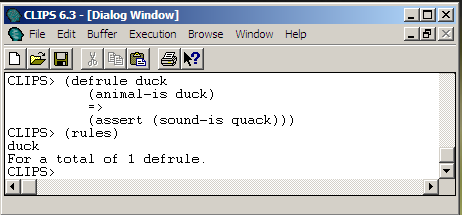
Lakukan hal berikut:

* Clear the system



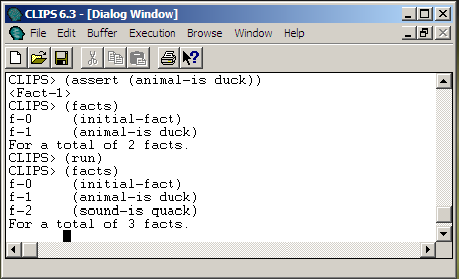
Seperti disebutkan sebelumnya, perintah (clear) akan menghapus fakta dari memori. Lanjutkan lakukan hal berikut:

* Ketikkan persis rule duck tersebut
* Tampilkan daftar rules.



Sejauh ini belum ada fakta sehubungan dengan rule duck ini.

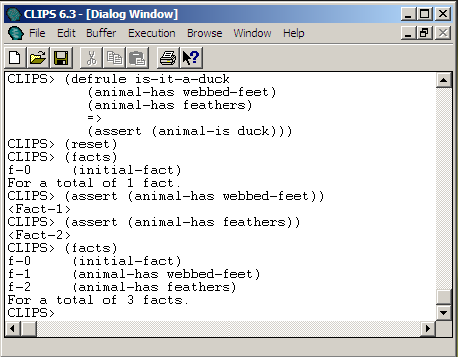
* Masukkan (assert (animal-is duck))
* List ada fakta apa.
* Run program kita (hanya terdiri satu rule dan satu fakta).
* List ada fakta apa saja.



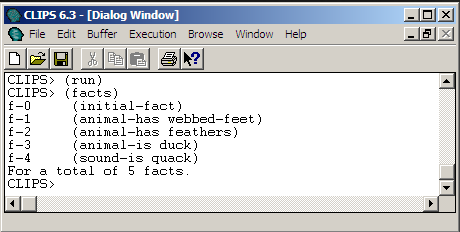
* Tampak bahwa ada fakta baru sebagai hasil di-infer oleh rule yaitu (sound is quack).

Dicobakan kembali dengan:

* Menambah rule baru
* Hapus fakta dari memori (karena akan menggunakan fakta lain).
* Bentuk fakta lain tsb (animal-has webbed-feet) dan (animal-has feathers) dan tampilkan hasilnya



* Jalankan program ini (ada dua rules dan dua fakta) dan perlihatkan hasilnya



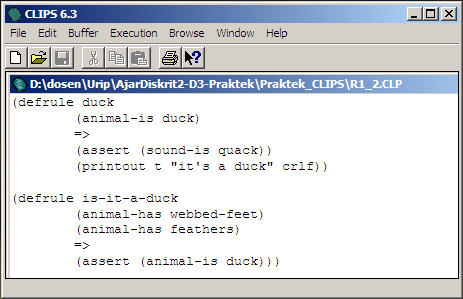
* Hasilnya akan bertambah dua fakta sebagai hasil inference dari rules yang berlaku.
* Cara kerjanya adalah:
  + Pertama kali rule is-it-a-duck dijalankan menghasilkan fakta (animal-is duck)
  + Selanjutnya fakta baru ini memicu rule duck karena (animal-is duck) tersedia, dan menghasilkan (sound is quack).

**Penggunaan Editor**

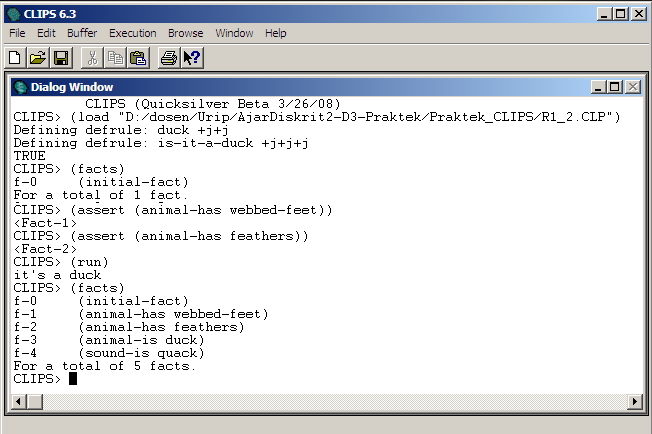
Adalah tidak efisien untuk mengetikkan kembali semua rules setiap kali kita ingin menjalankan program (setelah exit). Terdapat kemampuan CLIPS membaca rules dari file (oleh karena itu rules dapat disimpan) sehingga ketidakefisienan dapat diatasi.

Untuk menggunakan editor, lakukan:

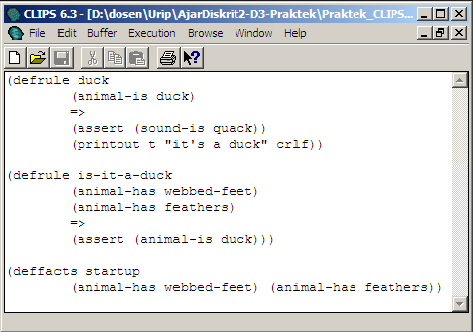
* Pada menu File pilih new
* Ketikkan rules yang ingin dibuat dan simpan hasilnya dan beri nama
* Sebagai ilustrasi dua rules yang lalu disimpan dalam R1\_2.CLP dan tampak sbb:



* Load file dgn nama tersebut
* Buat data (animal-has webbed-feet) dan (animal-has feathers) dan jalankan.

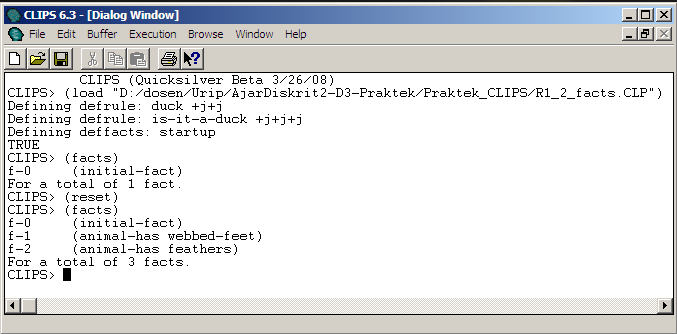


Jika rules dapat disimpan dalam file, demikian juga dengan fakta. Berikut adalah fakta2 persis seperti contoh sebelumnya, namun didefinisikan menggunakan deffacts dan disimpan dalam file .CLP bersama dengan rules.



Selanjutnya untuk menjalankan:

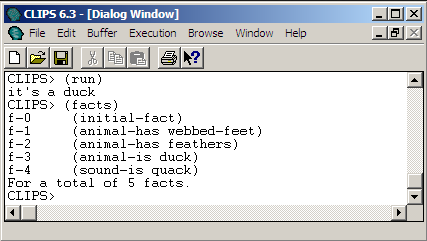
* Load file .CLP yang ingin dijalankan
* Coba perlihatkan fakta



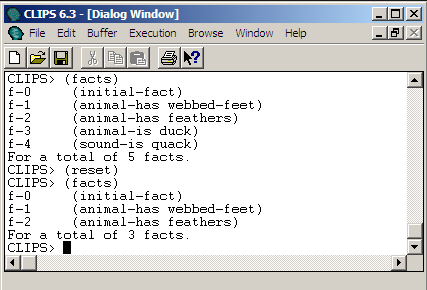
Terlihat ketika kita langsung ingin melihat fakta dengan mengetik (facts) hasilnya adalah masih kosong (ditandai dengan hanya ada fact-0). Setelah kita isukan (reset) diikuti dengan (facts) baru tampak deffacts berefek.

Selanjutnya

* Jalankan
* Coba perlihatkan fakta hasil menjalankan, sama seperti yang diharapkan



Ketika fakta dihapus dari memori dengan (reset) hasilnya adalah tidak kosong namun sama seperti yang didefinisikan pada deffacts.



**Templates**

Walaupun kita dapat menjalankan banyak dengan fakta-fakta sederhana seperti yang telah kita jalankan, dalam baanyak hal sering diinginkan menyambungkan informasi kecil bersama. Misalkan suatu aplikasi tentang menentukan general fitness dari sejumlah orang. Sejumlah indicator akan digunakan sehingga setiap orang mempunyai nilai berbeda. Informasi tentang dua orang yang berbeda dapat diekspresikan sebagai berikut:

(umur Andri 20) (umur Brenda 23)

(berat Andri 70) (berat Brenda 55)

(tinggi Andri 175) (tinggi Brenda 165)

(tekanan-darah Andri 130 80) (tekanan-darah 120 65)

Ekspresi ini bisa saja digunakan, namun hal ini melibatkan banyak fakta terpisah dengan tidak ada yang menghubungkan mereka bersama-sama selain dari nama personnya. Cara yang lebih baik adalah dengan menggunakan struktur yang didefinisikan menggunakan deftemplate.

(deftemplate personal-data

(slot nama)

(slot umur)

(slot berat)

(slot tinggi)

(multislot tekanan-darah)

)

Setiap kali suatu fakta dengan tipe ini dibentuk, ia mengandung slot-slot sesuai definisi dan dapat diakses melalui namanya. Untuk mengisikan datanya digunakan perintah misalkan seperti berikut:

(assert (personal-data

(nama Andri) (umur 20) (berat 70) (tinggi 175) (tekanan-darah 130 80)))

Atau pada deffacts:

(deffacts people

(personal-data

(nama Andri) (umur 20) (berat 70) (tinggi 175) (tekanan-darah 130 80))

(personal-data

(nama Brenda) (umur 23) (berat 55) (tinggi 165) (tekanan-darah 120 65)))

Tidak perlu untuk secara komplit menambah data, jadi berikut ini juga valid dimana selain hanya sebagian atribut juga urutan tidak penting:

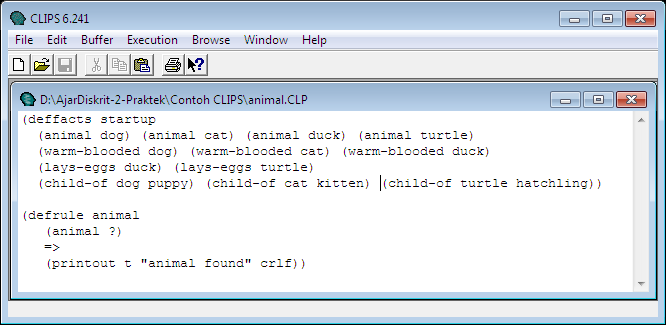
(assert (personal-data

(umur 23) (berat 70) (nama Brenda)))

**Matching Things**

Sejauh ini pola2 yg digunakan utk memadankan (matching) rules terhadap facts adalah sangat sederhana dan terbatas. Masing2 pola dipadankan satu fact spesifik. Dengan menggunakan wildcards, adalah mungkin memadankan rules dengan banyak facts, mengeksekusi secara berulang. Sebagai contoh:

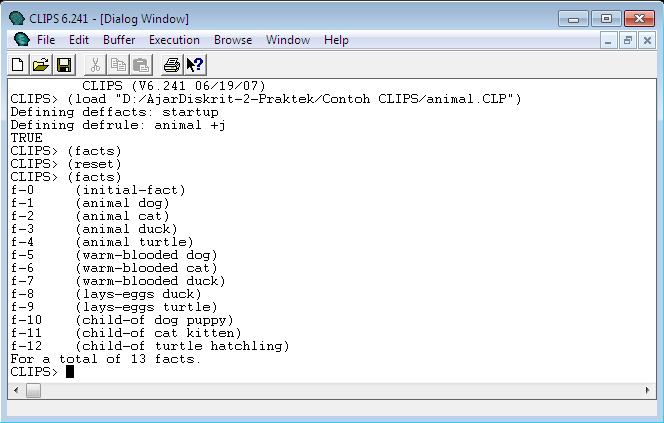
Diberikan defrule dan deffacts tentang animal.



Jalankan program ini namun untuk review yang lalu jalankan sbb:

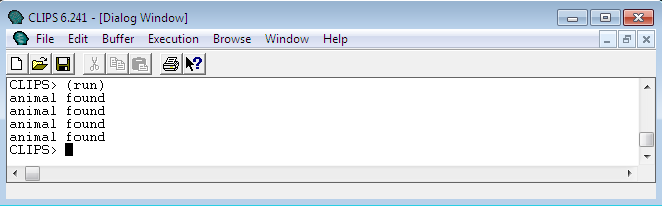
* Load
* Tampilkan fakta
* Reset
* Tampilkan fakta
* Jalankan (run)

Sampai sebelum run hasilnya adalah:



Review: harus di reset dulu supaya deffacts menjadi aktif.

Dijalankan dan tampilkan fakta hasilnya:



Ini memperlihatkan:

* Ia mentrigger 4 kali dimana terdapat empat fact yang sesuai dengan pola (animal ?)
* Pada pola (animal ?), simbol ? adalah suatu wildcard. Ia akan match semua simbol. Simbol pertama harus ada setelah itu boleh wildcards.

Dengan demikian dari deffacts yang didefinisikan, pola (child-of ? ?) legal dan akan menghasilkan tiga facts. Namun (? ? hatchling) ilegal

**Variabel pada Pola**

Sering diperlukan tidak cukup dengan wildcards saja, tetapi menggunakan variabel berbentuk ?var alih2 ? saja. Berikut ini adalah me-list nama animal.

(defrule list-animals

(animal ?name)

=>

(printout t ?name “ found” crlf)

Dengan deffacts yang sama akan menghasilkan:

CLIPS> (run)

turtle found

duck found

cat found

dog found

CLIPS>

Penggunaan variabel ini lebih jauh lagi dapat digunakan untuk dua atau lebih pola.

(defrule mammal

(animal ?nama)

(warm-blooded ?nama)

(not (lays-eggs ?nama))

=>

(assert (mammal ?nama))

(printout t ?nama “ is a mammal” crlf))

Hasilnya:

CLIPS> (run)

cat is a mammal

dog is a mammal

CLIPS>