# Modul 1
# LINGKUNGAN VISUAL PROLOG
# (*VISUAL PROLOG ENVIRONMENT, VPE*)

## A. TUJUAN

1. Mengetahui cara memulai program VPE.
2. Dapat membuat program menggunakan jendela editor VPE.
3. Dapat menjalankan dan menguji program pada VPE.
4. Dapat membuka file dari disk.
5. Mengetahui (jika) adanya kesalahan dan di baris mana pada program kesalahan itu terjadi.

## B. DASAR TEORI

### 1. Visual Prolog Environment (VPE)

VPE didesain agar seorang programmer dapat dengan mudah, nyaman dan cepat dalam membangun, menguji dan memodifikasi suatu aplikasi atau program yang ditulis dalam Visual Prolog. VPE memiliki beberapa varian sehingga mendukung untuk digunakan di beberapa platform sistem operasi seperti MS-DOS, MS Windows 3.1, MS Windows 95, MS Windows NT, Win-OS/2, atau OS/2 PM, yang digunakan pada platform prosesor 16 bit ataupun prosesor 32 bit dari keluarga prosesor Intel® 80x86 ataupun kompatibelnya seperti prosesor AMD®.

Pengguna VPE diasumsikan mempunyai pengalaman dan pengetahuan dalam menggunakan sistem GUI (*Graphical User Interface*), seperti menggunakan menu, menutup, meminimize, memaximize ataupun meresize suatu jendela (*window*), meloading file dari jendela File Open dialog, mengklik toolbar dan lain-lain. Jika praktikan belum punya pengetahuan ini, praktikan dapat mempelajari dari literatur yang berkaitan dengan sistem operasi terkait, seperti buku cara menggunakan MS Windows 95 atau yang lainnya.

### 2. Menjalankan VPE

Sebelum menjalankan VPE, tentu saja, diasumsikan program VPE sudah terinstall di komputer yang digunakan. Program instalasi akan membuat sebuah program group yang di dalamnya terdapat icon yang digunakan untuk menjalankan VPE, yaitu dengan cara mengklik icon tersebut. Namun ada banyak cara untuk menjalankan VPE, seperti, dengan menggunakan Windows Explorer, men-double klik file VIP.EXE di direktori BIN\WIN\16 untuk platform 16 bit atau direktori BIN\WIN\32 untuk platform 32 bit yang direktori tersebut terletak di bawah direktori utama VIP.

Jika Visual Prolog telah pernah membuka suatu project (dengan ekstensi .VPR) terakhir kali dan VPE ditutup, maka secara otomatis akan membuka project tersebut ketika VPE dijalankan kembali.

### 3. Membuka jendela editor (*editor window*)

Untuk menciptakan jendela editor yang baru, praktikan dapat menggunakan menu perintah **File | New**. Setelah itu akan muncul jendela editor baru dengan

judul "NONAME". Editor ini layaknya seperti editor teks standar lainnya, seperti NOTEPAD yang dimiliki oleh MS Windows. Praktikan dapat menggunakan tombol kursor (tanda panah atas, bawah, kiri dan kanan) dan mouse untuk menggerakkan kursor seperti layaknya editor lain. Juga editor ini mendukung operasi *cut, copy and paste, undo/redo* yang diaktifkan dari menu **Edit** atau melalui penekanan tombol akselerator yang dapat dilihat pada menu **Edit**.

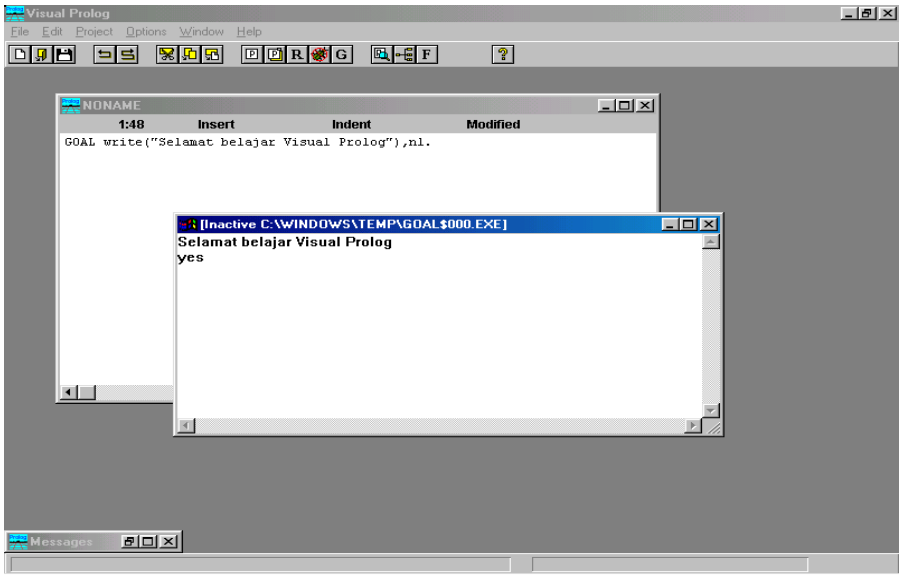
Untuk lebih mendalami editor ini lebih jauh praktikan dapat membaca dari manual Visual Development Environment (VDE) pada file VDE.DOC di direktori DOC yang terletak di bawah direktori utama VIP.

**4. Menjalankan dan menguji suatu program**

Untuk mengecek bahwa sistem diset dengan baik, praktikan dapat mencoba mengetikkan teks berikut pada jendela editor:

GOAL write("Selamat belajar Visual Prolog"),nl.

Baris kode di atas pada Prolog dinamakan GOAL dan baris tersebut telah cukup syarat untuk menjadi program yang bisa dieksekusi. Untuk mengeksekusi GOAL, aktifkan item menu **Project | Test Goal**, atau cukup dengan menekan tombol akselerator **Ctrl+G**. Jika sistem terinstall dengan baik, maka di layar akan tampak seperti gambar berikut:




Hasil eksekusi akan ditampilkan pada jendela yang berbeda yang harus ditutup sebelum menguji GOAL lainnya. Visual Prolog memperlakukan GOAL sebagai sebuah program yang telah ter-*compile*, kemudian di-*links* dan dibangkitkan menjadi suatu bentuk jendela yang dapat dieksekusi (*executable*).

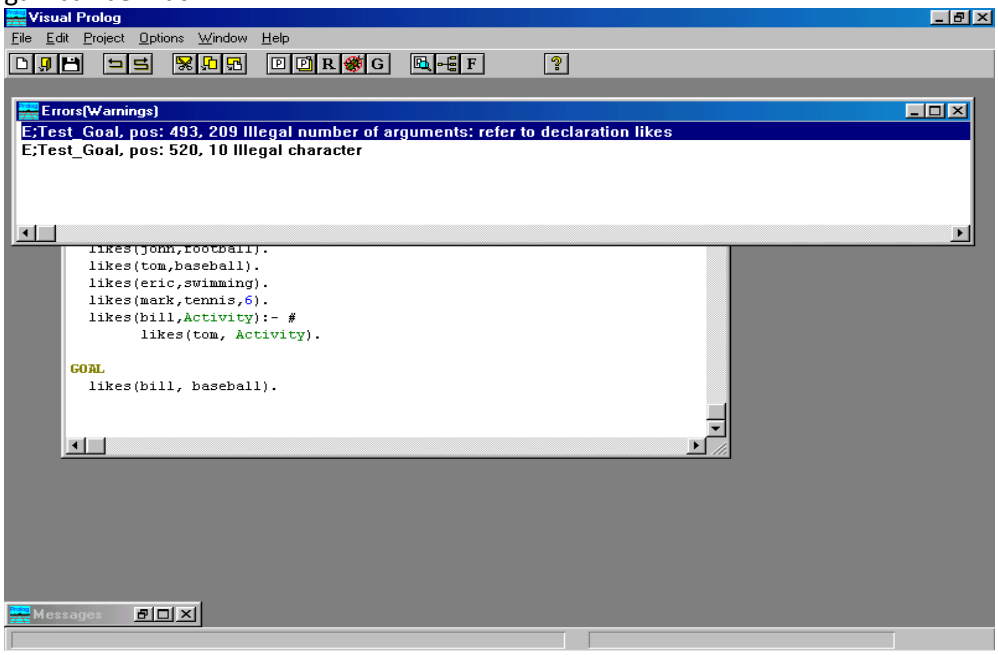
**5. Membuka file dari disk**

Contoh-contoh program untuk mendukung manual Language Tutorial Visual Prolog disediakan di direktori /DOC/EXAMPLES pada direktori utama VIP (dengan

syarat manual beserta contohnya terinstall pada waktu menginstall Visual Prolog pertama kali). Salah satu file contoh tersebut dapat dibuka dengan menggunakan item menu **File | Open** atau menekan tombol akselerator **F8**. Pilih salah satu file (berekstensi .PRO) dan uji GOAL program tersebut dengan menekan tombol **Ctrl+G**.

**6. Melihat dan memperbaiki kesalahan**

Jika programmer membuat kesalahan dalam menuliskan kode Visual Prolog, maka VPE akan menampilkan jendela kesalahan (*error window*) di mana pada jendela tersebut terdapat daftar kesalahan yang dibuat. Programmer dapat men-double klik kesalahan tersebut agar kursor beralih ke posisi di mana kesalahan tersebut dibuat pada baris kode di jendela editor, sehingga dengan lebih cepat kesalahan tersebut dapat diperbaiki. Tampilan jendela kesalahan dapat dilihat pada gambar berikut:




## C. PRAKTIK

1. Buka jendela editor yang baru.
2. Ketikkan program di bawah ini.

```
PREDICATES
  putra(STRING,STRING)
  saudara_perempuan(STRING,STRING)
  saudara_laki(STRING,STRING)
  menikah(STRING,STRING)
  ayah(STRING ayah,STRING putra)
  kakek(STRING kakek,STRING cucu)
  nondeterm ipar_perempuan(STRING,STRING)
CLAUSES
  putra("Ikhsan","Bentang").
```

```
saudara_perempuan("Dini","Dina").
saudara_laki("Adi","Lintang").
menikah("Ikhsan","Dini").
menikah("Lintang","Surga").

ayah(A,B):-putra(B,A).

kakek(A,B):-ayah(A,C), ayah(C,B).

ipar_perempuan(A,B):-menikah(A,C), saudara_perempuan(C,B).
ipar_perempuan(A,B):-saudara_laki(A,C), menikah(C,B).
```

GOAL

```
ipar_perempuan("Ikhsan",X).
```

Catatan:

- Keyword nondeterm pada *section* PREDICATES yang mendahului predikat ipar_perempuan berfungsi untuk memberitahu ke compiler Visual Prolog bahwa predikat tersebut mempunyai lebih dari satu kemungkinan jawaban (non-deterministik). Secara *default* Visual Prolog akan menganggap predikat yang ada di *section predicates* sebagai deterministik (kecuali pada *section facts*, merupakan kebalikannya).
- Kesalahan karena tidak memberikan keyword nondeterm pada predikat yang nondeterministik, pada Visual Prolog 5.1 hanya akan diberi peringatan (*warning*) tetapi eksekusi program tetap diteruskan dan memberikan solusi yang diinginkan. Namun pada Visual Prolog 5.2, hal ini dianggap *error* (kesalahan) dan kompiler tidak meneruskan eksekusi dan memberikan pesan kesalahan.

3. Uji goal tersebut. Amati hasilnya.
4. Ganti goal tersebut dengan:

```
ipar_perempuan("Adi",Y).
```

dan lakukan uji goal (catatan: sebelum menjalankan uji goal berikutnya, jendela hasil uji goal sebelumnya harus ditutup terlebih dahulu). Amati hasilnya.

## D. PERTANYAAN/TUGAS

1. Sisipkan baris kode berikut dibawah kata CLAUSES:

```
putra("Bentang",”Genta”).
```

Berikan GOAL sebagai berikut.

```
kakek(”Genta”,Cucu).
```

Kemudian Uji goal dan apa hasilnya?

2. Apa tombol-tombol akselerator untuk perintah-perintah berikut : copy, cut, paste, delete, redo, undo, new, open, save, exit, dan test goal.

# Modul 2
# DASAR-DASAR PROLOG

## A. TUJUAN

1. Mengetahui apa yang dimaksud dengan pemrograman logika (*PROgramming in LOGic*).
2. Dapat mengkonversikan dari bahasa natural (natural language) ke bahasa PROLOG dan begitu juga sebaliknya.
3. Mengetahui konsep dasar fakta-fakta (*facts*), aturan-aturan (*rules*), predikat-predikat (*predicates*) dan *variable*.

## B. DASAR TEORI

### Pemrograman Logika

Prolog dikenal sebagai bahasa deskriptif (*descriptive language*), yang berarti dengan diberikan serangkaian fakta-fakta dan aturan-aturan, Prolog, dengan menggunakan cara berpikir deduktif (*deductive reasoning*), akan dapat menyelesaikan permasalahan suatu program. Ini dikontraskan dengan bahasa komputer tradisional seperti C, BASIC, Pascal yang dikenal sebagai bahasa prosedural (*procedural language*). Dalam bahasa prosedural, programmer harus memberikan instruksi tahap demi tahap agar komputer dapat dengan pasti bagaimana menyelesaikan permasalahan yang diberikan. Dengan kata lain, programmer harus tahu lebih dahulu bagaimana cara menyelesaikan permasalahan sebelum diinstuksikan ke komputer. Lain jika dibandingkan dengan programmer Prolog. Programmer Prolog hanya membutuhkan deskripsi/gambaran permasalahan, lalu menerjemahkannya ke bahasa Prolog. Selanjutnya tinggal sistem Prolog yang menentukan bagaimana mencari solusinya.

Prolog didasarkan pada klausa-klausa Horn (Horn *clauses*), yang merupakan himpunan bagian dari sistem formal yang dinamakan logika predikat (*predicate logic*). Logika predikat menyederhanakan cara agar jelas bagaimana berpikir akan dilakukan. Prolog menggunakan variasi sintak logika predikat yang telah disederhanakan dengan demikian sintaknya mudah dimengerti dan sangat mirip dengan bahasa natural.

Prolog mempunyai mesin inferensi (*infrence engine*) yang merupakan suatu proses berpikir logis mengenai informasi. Mesin inferensi mempunyai pencocok pola (*pattern matcher*) yang akan mengambil informasi yang telah disimpan (diketahui) dan kemudian mencocokkan jawaban atas pertanyaan. Satu *feature* penting dari Prolog adalah bahwa, sehubungan mencari jawaban logis atas pertanyaan yang diajukan, ia dapat berhubungan dengan banyak alternatif dan mencari semua kemungkinan dari pada hanya satu solusi.

Logika predikat dibangun agar mudah menerjemahkan ide-ide berbasis logika menjadi bentuk tertulis. Prolog mengambil keunggulan dari sintak ini untuk membangun suatu bahasa pemrograman yang berbasis logika. Dalam logika predikat, pertama kali harus membuang semua kata-kata yang tidak dibutuhkan

dari suatu kalimat. Kemudian mentransformasi kalimat tersebut dengan mencari relasi terlebih dahulu, kemudian setelah itu melakukan pengelompokkan *object*. *Object* kemudian menjadi argumen dari relasi atas *object* tersebut. Berikut ini merupakan contoh kalimat bahasa natural ditransformasikan menjadi sintak logika predikat:

| **Bahasa Natural** | **Logika Predikat** |
|---|---|
| Eskrim rasanya enak | enak(eskrim) |
| Mawar berwarna merah | merah(mawar) |
| Miftah menyukai mobil | suka(miftah,mobil) |
| Wening suka bakso jika bakso rasanya enak | suka(wening, bakso) if enak(bakso) |

**a. Kalimat : Fakta dan Aturan**

Programmer Prolog mendefinisikan *object-object* dan relasi-relasi, kemudian mendefinisikan aturan mengenai kapan relasi-relasi ini dikatakan benar (*true*). Contoh kalimat :

Wening suka bakso.

memperlihatkan relasi antara *object* Wening dan bakso, relasi ini adalah suka. Berikutnya membuat aturan kapan Wening suka bakso adalah kalimat yang benar:

Wening suka bakso jika bakso rasanya enak.

Fakta : Apa yang diketahui

Dalam Prolog, relasi antara *object-object* dinamakan **predikat**. Dalam bahasa natural relasi disimbolisasikan oleh suatu kalimat. Dalam logika predikat yang Prolog gunakan, suatu relasi adalah kesimpulan dalam suatu frase sederhana. Suatu fakta memiliki nama relasi diikuti *object* atau *object-object* di dalam tanda kurung. Sebagaimana kalimat, fakta tersebut diakhiri dengan tanda titik (.).

Berikut ini beberapa fakta yang mengekspresikan relasi *suka* dalam bahasa natural:

Toni suka Inung.
Inung suka Toni.
Toni suka kucing.

Dan dalam sintak Prolog, fakta di atas ditulis:

```
suka(toni,inung).
suka(inung,toni).
suka(toni,kucing).
```

Fakta juga bisa mengekspresikan sifat (*property*) dari suatu *object* sebagaimana suatu relasi. Dalam bahasa natural kalimat "Kodok berwarna hijau" dan "Komeng adalah laki-laki" diubah ke dalam sintak Prolog menjadi:

```
hijau(kodok).
lakilaki(komeng).
```

Aturan: Apa yang dapat disimpulkan dari fakta yang ada

Aturan membuat kita dapat mengambil suatu fakta dari fakta yang lain. Dengan bahasa lain, aturan adalah suatu konklusi diketahui benar jika satu atau lebih konklusi atau fakta lain ditemukan benar. Berikut ini beberapa aturan yang berhubungan dengan relasi suka:

```
Inung suka apapun yang Toni suka.
Komeng suka apapun yang berwarna hijau.
```

Dengan aturan di atas, dapat disimpulkan dari beberapa fakta sebelumnya mengenai sesuatu yang Inung dan Komeng suka:

```
Inung suka Inung.
Komeng suka Kodok.
```

Untuk menuliskan aturan tersebut dalam sintak Prolog hanya dibutuhkan sedikit perubahan seperti ini:

```
suka(inung, Sesuatu):- suka(toni, Sesuatu).
suka(komeng, Sesuatu):- hijau(Sesuatu).
```

Simbol :- diucapkan dengan "jika" dan memisahkan dua bagian dari aturan yaitu *head* dan *body*. Aturan di atas dalam makna prosedur, bisa berarti "untuk membuktikan Inung suka sesuatu, buktikanlah bahwa Toni suka hal yang sama" dan "untuk membuktikan Komeng suka sesuatu, buktikanlah bahwa sesuatu itu berwarna hijau".

**b. Query**

Kalau kita sudah memberikan Prolog sekumpulan fakta, selanjutkan dapat diajukan pertanyaan sehubungan dengan fakta-fakta. Ini dikenal dengan nama memberikan query ke sistem Prolog (*querying the Prolog system*). Pertanyaan yang diajukan ke Prolog sama tipenya seperti dalam bahasa natural. Berdasarkan pada fakta dan aturan yang diberikan sebelumnya, kita dapat menjawab pertanyaan tentang relasi-relasi ini sebagaimana Prolog juga bisa:

Apakah Toni menyukai Inung?

dalam sintak Prolog:

suka(toni, inung).

dengan memberikan query ini, Prolog akan menjawab:

Yes

karena Prolog mempunyai fakta yang mengatakan hal tersebut. Berikut merupakan pertanyaan yang lebih rumit dalam bahasa natural:

Apa yang Toni suka?

dalam sintak Prolog:

suka(toni, Apa).

Sintak Prolog tidak berubah ketika mengajukan query dan tampak bahwa sintak query sangat mirip dengan sintak fakta. Yang penting untuk diingat adalah bahwa *object* kedua Apa diawali dengan huruf besar sedang *object* pertama toni tidak. Ini karena toni sudah *fix* sebagai konstanta *object* sedangkan Apa adalah sebuah variabel. Variabel selalu diawali dengan sebuah huruf besar atau sebuah garis bawah (*underscore,* _ ).

Prolog selalu mencari jawaban dari suatu query dengan memulai dari bagian paling atas (*top*) dari fakta-fakta dan melihat setiap fakta sampai ke bagian paling bawah (*bottom*) dimana tidak ada satupun fakta lagi. Untuk menjawab query di atas, Prolog akan menampilkan:

What=inung
What=kucing
2 Solutions

karena Prolog tahu bahwa:

suka(toni, inung).

dan

suka(toni, kucing).

jika kita bertanya pada Prolog:

suka(inung, What).

maka Prolog akan menjawab:

What=toni
What=inung
What=kucing

3 Solutions

karena Prolog tahu Inung suka Toni dan juga Inung suka apapun yang Toni suka, sedangkan Toni suka Inung dan juga suka kucing.

Prolog mendukung query jamak (*compound query*). Ada 2 jenis query jamak yaitu:

1. Konjungsi (logika and)
2. Disjungsi (logika or)

Konjungsi dalam sintak Prolog, antara satu query dengan query yang lain dipisahkan dengan tanda koma (,) sedangkan disjungsi dipisahkan dengan tanda semicolon (;). Contohnya untuk query "Siapakah orang yang suka berenang **dan** sekaligus suka membaca?" ditulis dalam sintak Prolog:

```
suka(Orang, renang), suka(Orang, baca).
```

sedangkan jika query berbunyi "Siapakah orang yang suka berenang **atau** suka membaca?" dalam Prolog ditulis:

```
suka(Orang, renang); suka(Orang, baca).
```

**c. Variabel**

Seperti yang telah disebutkan sebelumnya, untuk memberikan nama variabel dalam Visual Prolog harus diawali dengan huruf besar (*capital letter*) atau garis bawah (*underscore*), berikutnya dapat berupa huruf (besar atau kecil), angka ("0-9") dan garis bawah ("_"). Berikut ini merupakan contoh penamaan variabel yang valid:

```
Ini_adalah_variabel_yang_valid
Tanggal_Lahir_Anak_Saya_1_12_1999
_Ini_juga_valid
```

sedangkan berikut ini penamaan variabel yang tidak valid:

```
1desember_1999        %diawali dengan angka
ini_tidak_valid  %diawali dengan huruf kecil
$Pengganggu           %terdapat karakter yang dilarang ('$')
```

Penamaan variabel selain valid sebaiknya yang mudah dimengerti. Sebagai contoh daripada memberikan nama variabel X untuk:

```
suka(X, tennis).
```

akan lebih dimengerti jika menggunakan nama variabel Orang sehingga menjadi:

```
suka(Orang, tennis).
```

*Anonnymous Variables* (Variabel Anonim)

Variabel anonim digunakan jika hanya dibutuhkan informasi tertentu dari suatu query. Variabel anonim akan mengabaikan nilai-nilai yang tidak dibutuhkan. Dalam Prolog, variabel anonim direpresentasikan dengan tanda garis bawah tunggal ("_"). Contoh jika ada fakta-fakta berikut:

```
orangtua(daud,irfan).
orangtua(roger,ova).
orangtua(suroto,toni).
```

jika ingin tahu siapa saja yang berstatus orangtua tanpa melihat siapa anaknya, maka bisa dilakukan query sebagai berikut

```
orangtua(Ortu, _).
```

Maka Prolog akan memberikan jawaban:

```
Ortu=daud
Ortu=roger
Ortu=suroto
3 Solutions
```

Variabel anonim juga bias berlaku sebagai fakta. Jika ditulis:

```
punya(_, sepatu).
makan(_).
```

maka dalam bahasa natural dapat dituliskan:

```
Setiap orang mempunyai sepatu.
Setiap orang makan.
```

**d. Komentar**

Programmer yang baik selalu memberikan catatan atau komentar untuk menjelaskan sesuatu yang mungkin tidak jelas bagi orang lain (atau bahkan bagi programmer sendiri dalam setahun ke depan misalnya). Komentar akan membuat program menjadi lebih mudah untuk dimengerti.

Komentar dengan baris jamak harus dimulai dengan karakter /* (*slash, asterik*) dan diakhiri dengan karakter */ (*asterik, slash*). Untuk memberikan komentar dengan baris tunggal dapat menggunakan karakter yang sama atau dapat dimulai dengan tanda persen (%). Contoh:

```
/* Ini contoh sebuah komentar */

%  Ini juga contoh komentar
```

```
/* Ini komentar
   dengan
   tiga baris */

/* Ini contoh komentar /* dalam komentar */ di Visual Prolog */
```

**C. PRAKTIK**

1. Buat jendela baru.
2. Ketikkan program berikut:

```
PREDICATES
  nondeterm dapat_membeli(symbol, symbol)
  nondeterm orang(symbol)
  nondeterm mobil(symbol)
  suka(symbol, symbol)
  dijual(symbol)

CLAUSES
  dapat_membeli(X,Y):-
        orang(X),
        mobil(Y),
        suka(X,Y),
        dijual(Y).

  orang(nur).
  orang(yudi).
  orang(dian).
  orang(heni).

  mobil(atoz).
  mobil(kijang).

  suka(dian, atoz).
  suka(yudi, pecel).
  suka(heri, buku).
  suka(nur, komputer).

  dijual(kijang).
  dijual(atoz).
  dijual(buku).

GOAL
  dapat_membeli(Siapa,Apa).
```

3. Uji goal dan amati hasilnya.
4. Tambahkan fakta berikut pada bagian CLAUSES dan kelompokkan pada predikat yang sama:

   suka(heni, kijang).

5. Uji goal dan amati hasilnya
6. Ganti-ganti goal pada program di atas dengan:
   a. suka(_,Apa).
   b. suka(Siapa,_).
   c. dapat_membeli(_,Apa).
   d. dapat membeli(Siapa,_).
   dan uji masing-masing goal serta amati hasilnya.

**D. PERTANYAAN/TUGAS**

1. Ubah bahasa Prolog berikut menjadi bahasa natural:
   a. orang(dian).
   b. mobil(atoz)
   c. suka(dian, atoz)).
   d. dijual(atoz).
   e. dapat_membeli(X,Y):-
      orang(X),
      mobil(Y),
      suka(X,Y),
      dijual(Y).
   f. suka(_,Apa).

2. Ubah bahasa natural berikut menjadi bahasa Prolog:
   a. Ricky mempunyai hobi bermain catur.
   b. Embang orangnya pemalas.
   c. Yusida seorang vegetarian.
   d. Kusdiar pandai bermain gitar.
   e. Mobil yang berwarna merah itu milik Sadek, bermerk BMW.
   f. Awan seorang pemain piano, berasal dari Pontianak bersuku Jawa.
   g. Seseorang dikatakan baik jika mempunyai sifat penyayang dan dermawan.
   h. Seseorang dikatakan pintar jika salah satu dari Matematika atau IPA atau IPS mendapat nilai tinggi.
   i. Setiap orang pasti mati.

3. Diberikan fakta sebagai berikut :
   anak (Jhon, James).
   anak (James, Peter).
   istri (Mary, Peter).
   anak (Sue, Ann).
   istri (Ann, James).
   pria(Jhon).

pria (James).
pria (Peter).
wanita (Mary).
wanita(Sue).
wanita(Ann).
usia(Jhon, 10).
usia(Sue, 13).

Dari fakta di atas buatlah program dalam bahasa PROLOG dan Ujilah dengan rule-rule sebagai berikut : a. cucu
b. ibu
c. adik
d. Kakek

# Modul 3
# PROGRAM VISUAL PROLOG

## A. TUJUAN

1. Mengerti dan dapat menerapkan section-section dasar yang ada di Visual Prolog seperti PREDICATES, DOMAINS, CLAUSES, dan GOAL.
2. Mengetahui section-section lainnya yang ada di Visual Prolog seperti FACTS, CONSTANT, GLOBAL.
3. Mengerti tentang *Compiler Directives.*
4. Mengerti dan dapat menerapkan aritas jamak (*multiple arity*) pada Visual Prolog.
5. Mengerti konsep aturan (*rule*) pada Prolog dan perbedaannya dengan bahasa lain.
6. Mengerti mengenai konversi tipe otomatis (*automatic type conversion*).

## B. DASAR TEORI

### Program Visual Prolog

#### 1. Section dasar Visual Prolog

Secara umum, program Visual Prolog terdiri dari empat section dasar, yaitu section **clauses**, section **predicates**, section **domains**, dan terakhir section **goal**. Berikut akan dijelaskan secara singkat masing-masing section tersebut.

**Section Clauses**

Section clauses merupakan section yang paling penting pada program Visual Prolog. Pada section inilah kita meletakkan fakta dan aturan. Ketika mencari jawaban, Visual Prolog akan mencari dari bagian paling atas dari section clauses, melihat setiap fakta dan aturan untuk mendapat jawaban benar, hingga ke bagian paling bawah dari section ini.

**Section Predicates**

Sebelum mendefinsikan predikat di section clauses, maka predikat tersebut harus dideklarasikan terlebih dahulu di section predicates. Kalau tidak, Visual Prolog tidak akan mengenal predikat yang kita tuliskan tersebut. Ketika mendeklarasikan suatu predikat, kita memberitahu Visual Prolog domain dari argumen yang dimiliki predikat tersebut.

Visual Prolog mempunyai perpustakaan predikat yang kalau dipakai tidak perlu dideklarasikan, karena sudah *built-in*. Untuk melihat predikat apa saja serta manfaatnya yang ada di perpustakaan Visual Prolog dapat melihat *help* dari Visual Prolog (**Help | Contents**).

Deklarasi Predikat

Deklarasi predikat dimulai dengan nama predikat diikuti tanda kurung buka, kemudian diikuti nol atau lebih argumen dari predikat (setiap argumen dipisah dengan tanda koma) kemudian ditutup dengan tanda kurung tutup, seperti:

```
PredicatName(tipe_argumen1, tipe_argumen2,...,tipe_argumenN)
```

dan tidak seperti section clauses, deklarasi predikati tidak perlu diakhiri tanda titik.

*Nama Predikat*

Nama predikat harus dimulai dengan huruf diikuti dengan serangkaian huruf, angka dan atau garis bawah (*underscore*). Walaupun bisa dimulai huruf besar, namun sangat direkomendasikan untuk memakai huruf kecil diawal nama predikat (beberapa versi Prolog yang lain tidak memperbolehkan nama predikat diawali huruf besar). Panjang nama predikat bisa sampai 250 karakter. Contoh penamaan predikat :

| **Nama predikat yang legal** | **Nama predikat yang illegal** |
|---|---|
| Fakta | [fakta] |
| pemain_piano | *pemain_piano* |
| milik_umum_atau_pribadi | milik_umum/pribadi |
| PolaTindakKriminal | Pola-Tindak-Kriminal |
| pilih_Item_Menu | pilih Item Menu |
| nama_predikat | nama<predikat> |
| rangking_10_besar | #rangking_10_besar |

*Argumen predikat*

Argumen predikat harus yang sudah dikenal oleh domain Visual Prolog. Suatu domain bisa merupakan domain standar atau bisa juga domain yang sudah dideklarasikan pada section **domains**. Contohnya, jika dideklarasikan suatu predikat predikat_ku(symbol, integer) pada section **predicates** seperti ini:

```
PREDICATES
        predikat_ku(symbol, integer)
```

maka tidak perlu lagi mendeklarasikan domain dari argumen pada section **domains**, karena symbol dan integer adalah standar domain. Namun jika ingin mendeklarasikan predikat_ku(nama, nomor) seperti ini:

```
PREDICATES
```

```
predikat_ku(nama, nomor)
```

maka dibutuhkan deklarasi dari domain nama dan nomor, seperti ini:

```
DOMAINS
        nama = symbol
        nomor = integer

PREDICATES
        predikat_ku(nama, nomor)
```

**Section Domains**

Section domains mempunyai 2 manfaat utama, yaitu pertama, kita dapat memberikan nama yang berarti untuk domain, walaupun secara internal domain tersebut sama tipenya dengan domain yang telah ada; yang kedua, kita dapat mendeklarasi domain khusus yang digunakan untuk mendeklarasikan struktur data yang tidak didefinisikan oleh standar domain. Dengan mendeklarasikan domain juga dapat mencegah kesalahan logika pada program. Contoh:

```
DOMAINS
        nama, jender = symbol
        umur = integer

PREDICATES
        orang(nama, jender, umur)
```

Keuntungan utama dari deklarasi di atas adalah Visual Prolog dapat mendeteksi adanya kesalahan, jika menuliskan aturan seperti ini:

```
jender_sama(X,Y):-
        orang(X, Sex, _),
        orang(Sex, Y, _).
```

Walaupun nama dan jender sama-sama didefinsikan sebagai symbol, namun keduanya tidak ekivalen. Visual Prolog akan mendeteksi kesalahan tersebut, karena kita mencoba menukar keduanya.

Standar Domain Visual Prolog

Visual Prolog mempunyai standar domain *built-in*. Kita dapat menggunakan standar domain ketika mendeklarasikan tipe dari argumen suatu predikat. Standar domain sudah dikenal Visual Prolog dan tidak perlu didefinisikan lagi pada section **domains**. Table 3.1. memperlihatkan variasi standar domain bertipe integer (bilangan bulat). Tabel 3.2. memperlihatkan standar domain lainnya.

Table 3.1: Standar Domain Integer

| ***Domain*** | ***Description and implementation*** | | |
|---|---|---|---|
| ***short*** | A small, signed, quantity. | | |
| | All platforms | 16 bits,2s comp | 32768 .. 32767 |
| ***ushort*** | A small, unsigned, quantity. | | |
| | All platforms | 16 bits | 0 .. 65535 |
| ***long*** | A large signed quantity | | |
| | All platforms | 32 bits,2s comp | -2147483648 .. 2147483647 |
| ***ulong*** | A large, unsigned quantity | | |
| | All platforms | 32 bits | 0 .. 4294967295 |
| ***integer*** | A signed quantity, having the natural size for the machine/platform architecture in question. | | |
| | 16bit platforms | 16 bits,2s comp | -32768 .. 32767 |
| | 32bit platforms | 32 bits,2s comp | -2147483648 .. 2147483647 |
| ***unsigned*** | An unsigned quantity, having the natural size for the machine/platform architecture in question. | | |
| | 16bit platforms | 16 bits | 0 .. 65535 |
| | 32bit platforms | 32 bits | 0 .. 4294967295 |
| ***byte*** | | | |
| | All platforms | [3] 8 bits | 0 .. 255 |
| ***word*** | | | |
| | All platforms | 16 bits | 0 .. 65535 |
| ***dword*** | | | |
| | All platforms | 32 bits | 0 .. 4294967295 |

Table 3.2: Standar Domain Dasar

| Domain | Description and implementation |
|---|---|
| ***char*** | A character, implemented as an unsigned byte. Syntactically, it is written as a character surrounded by single quotation marks: 'a'. |
| ***real*** | A floating-point number, implemented as 8 bytes in accordance with IEEE conventions; equivalent to C's ***double***. Syntactically, a real is written with an optional sign (+ or -) followed by some digits *DDDDDDD*, then an optional decimal point (.) followed by more digits *DDDDDDD*, and an optional exponential part (e(+ or -)DDD):<br><+\|-> DDDDD <.> DDDDDDD <e <+\|-> DDD><br>Examples of real numbers:<br>42705 9999 86.72<br>9111.929437521e238 79.83e+21<br><br>Here 79.83e+21 means 79.83 x 10^21, just as in other languages. The permitted number range is 1 ) $10^{-307}$ to 1 ) $10^{308}$ (1*e*-307 to 1*e*+308). Values from the integral domains are automatically converted to real numbers when necessary. |
| ***string*** | A sequnce of characters, implemented as a pointer to a zero-terminated byte array, as in C. Two formats are permitted for strings:<br>1. a sequence of letters, numbers and underscores, provided the first character is lower-case; or<br>2. a character sequence surrounded by a pair of double quotation marks.<br><br>Examples of strings:<br>telephone_number "railway ticket" "Dorid Inc"<br><br>Strings that you write in the program can be up to 255 characters long. Strings that the Visual Prolog system reads from a file or builds up internally can be up to 64K characters long on 16-bit platforms, and (theoretically) up to 4G long on 32-bit platforms. |
| ***symbol*** | A sequence of characters, implemented as a pointer to an entry in a hashed symbol-table, containing strings. The syntax is the same as for strings. |

**Section Goal**

Secara esensial, section **goal** sama dengan *body* dari sebuah aturan (*rule*), yaitu sederetan sub-sub goal. Perbedaan antara section **goal** dengan suatu aturan adalah setelah kata kunci **goal** tidak diikuti tanda :- dan Visual Prolog secara otomatis mengeksekusi goal ketika program dijalankan.

Jadi program Visual Prolog mempunyai struktur dasar sebagai berikut:

```
DOMAINS
  /*------------------------
          deklarasi domain
  -------------------------*/

PREDICATES
  /*------------------------
          deklarasi predikat
  -------------------------*/

CLAUSES
  /*-------------------------------
          clauses (fakta dan aturan)
  --------------------------------*/

GOAL
  /*------------------------
          subgoal_1,
          subgoal_2,
          ...,
          subgoal_N.
  -------------------------*/
```

**2. Section Program Lainnya.**

Ada beberapa section lainnya yang digunakan di Visual Prolog yaitu section **facts**, section **constants**, dan section **global**. Kali ini akan dibahas secara singkat ketiga section tersebut sebagai perkenalan. Section **facts** akan dibahas lebih mendalam pada modul yang lain.

**Section Facts**

Program Visual Prolog merupakan suatu koleksi dari sekumpulan fakta dan aturan. Kadang ketika program sedang berjalan, kita ingin meng-*update* (merubah, menambah, atau menghapus) beberapa fakta dari program. Pada kasus ini fakta menjadi suatu database yang dinamis atau database internal, dan fakta tersebut dapat berubah ketika program sedang berjalan. Visual Prolog menyediakan section khusus untuk mendeklarasikan fakta di program yang menjadi bagian dari database dinamis, yaitu section **facts**.

Kata kunci **facts** untuk memulai section **facts**. Visual Prolog menyediakan sejumlah predikat *built-in* yang mempermudah penggunaan section fakta dinamis ini. Kata kunci **facts** dapat digantikan dengan kata kunci **database**, untuk maksud yang sama.

**Section Constants**

Konstanta simbolis dapat digunakan di program Visual Prolog. Untuk itu sebelumnya harus dideklarasikan terlebih dahulu. Deklarasi konstanta diakukan pada section **constants**, diikuti dengan deklarasi menggunakan sintak:

<id> = <Macro_definition>

<id> adalah nama dari konstanta simbolis dan <Macro_definition> adalah apa yang akan diisikan (assign) kedalam konstanta. Setiap <Macro_definition> diakhiri dengan baris baru. Dengan demikian hanya ada satu deklarasi konstanta pada tiap barisnya. Contoh:

```
CONSTANT
        nol = 0
        satu = 1
        dua = 2
        ratus = (10*(10-1)+10)
        pi = 3.121592653
```

Sistem tidak membedakan antara huruf besar dan huruf kecil pada deklarasi suatu konstanta. Konsekuensinya, ketika suatu pengenal konstanta digunakan pada section **clauses** pada program, huruf pertama harus huruf kecil untuk mencegah kerancuan antara konstanta dan variabel. Berikut ini merupakan contoh yang valid:

```
CONSTANTS
        Dua = 2

GOAL
        A=dua, write(A).
```

Deklarasi section **constants** bisa lebih dari satu kali dalam program, namun deklarasi suatu konstanta harus dilakukan sebelum konstanta itu digunakan pada section yang lain. Suatu konstanta hanya boleh dideklarasikan satu kali, dan jika ada deklarasi lain untuk konstanta yang sama maka akan terjadi kesalahan.

**Section Global**

Visual Prolog memperbolehkan untuk mendeklarasikan beberapa domain, predikat dan klausa menjadi global (daripada hanya lokal). Caranya dengan menset secara terpisah section **global domains**, **global predicates**, dan **global facts** pada bagian paling atas dari program. Modul ini bukan tempatnya untuk membahas secara detail mengenai section **global**.

***3. Compiler Drectives***

Visual Prolog mendukung *compiler directives* yang dapat ditambahkan ke badan program untuk memberitahukan ke komputer bagaimana secara spesifik memperlakukan kode-kode waktu di-*compile.* Untuk menset *compiler directives*

sebagian besar dilakukan melalui menu **Options | Project | Compiler Options**. Modul ini tidak akan membahas secara detil mengenai *compiler directives*, namun akan memperkenalkan salah satu diantaranya yaitu *include directive*.

Kalau sudah terbiasa membuat program menggunakan Visual Prolog, seringkali kita memakai suatu prosedur tertentu berulang kali, sehingga setiap kali membuat program baru dan menggunakan prosedur tersebut, prosedur tersebut harus diketikkan kembali. Untuk menghemat waktu, maka dapat digunakan *include directive*. Contohnya:

- Buatlah file (misalnya MYPROC.PRO) yang berisikan deklarasi predikat yang sering digunakan (menggunakan section **domains** dan section **prtedicates** ) dan menuliskan prosedur yang didefinisikan pada predikat di section **clauses**.
- Buat program baru dan tuliskan kode:

  include “myproc.pro”

  di tempat di mana biasa dituliskan section **domains, facts, predicates, clauses** atau **goal**.

Ketika program di-*compile*, Visual Prolog juga akan meng-*compile* isi dari file MYPROC.PRO.

**4. Aritas jamak (*multiple arity*)**

Aritas (*arity*) suatu predikat adalah jumlah argumen yang ada pada predikat tersebut. Visual Prolog memperbolehkan kita mempunyai 2 atau lebih predikat dengan nama yang sama namun dengan aritas yang berbeda. Aritas yang berbeda dari nama predikat yang sama harus dikelompokkan bersama baik pada section **predicates** maupun pada section **clauses**. Perbedaan aritas oleh Visual Prolog akan diperlakukan secara berbeda pula. Contoh:

```
DOMAINS
        orang = symbol

PREDICATES
        ayah(orang)             % orang ini adalah seorang ayah
        ayah(orang, orang) % orang 1 adalah ayah bagi yg ke-2

CLAUSES
        ayah(Seseorang):-
                ayah(Seseorang, _).
        ayah(erwin, diena).
        ayah(erwin, latifah).
```

**5. Sintak Aturan (*Rule Syntax*)**

*Rule* pada Prolog adalah ketika kebenaran sebuah fakta tergantung pada kesukesan (kebenaran) dari satu atau lebih fakta yang lain. Seperti yang telah

dijelaskan pada modul sebelumnya aturan terdiri dari 2 bagian yaitu *head* dan *body*. Berikut ini merupakan aturan generik penulisan sintak *rule* pada Visual Prolog:

HEAD:- <subgoal>, <subgoal>, ..., <subgoal>.

Bagian *body* dari *rule* terdiri dari satu atau lebih subgoal. Setiap subgoal dipisahkan oleh koma, menspesifikasikan konjungsi, dan subgoal terakhir diakhiri dengan tanda titik.

Untuk membuat suatu *Rule* dikatakan sukses (benar), Prolog harus memeriksa kebenaran dari setiap subgoal yang ada pada aturan tersebut. Jika ada subgoal yang gagal (salah), Prolog akan kembali ke atas dan mencari alternatif bagi subgoal yang paling awal, kemudian kembali memproses dengan nilai variabel yang berbeda. Ini dinamakan lacakbalik (*backtracking*). Penjelasan yang lebih rinci mengenai lacakbalik akan diberikan pada modul yang lain.

**6. Konversi Tipe Otomatis (*Automatic Type Conversions*)**

Ketika Visual Prolog mencocokkan 2 variabel, keduanya tidak selalu berasal dari domain yang sama. Juga kadang variabel diikat (*bound*) menjadi konstan dari domain lain. Percampuran domain ini diperbolehkan karena Visual Prolog melakukan konversi tipe otomatis dengan syarat konversi bisa terjadi bila:

- Antara **strings** dan **symbols.**
- Antara semua domain **integer** dan juga **real**. Ketika suatu karakter (**char**) dikonversikan ke nilai numeris, angka nilai ASCII dari karakter tersebut yang digunakan.

## C. PRAKTIK

1. Buka jendela baru dan ketik program berikut:

```
DOMAINS
 kali,jumlah = integer

PREDICATES
 tambahkan(jumlah,jumlah,jumlah)
 kalikan(kali,kali,kali)

CLAUSES
 tambahkan(X,Y,Jumlah):-
        Jumlah=X+Y.
 kalikan(X,Y,Kali):-
        Kali=X*Y.

GOAL
 tambahkan(32,54,Jumlah).
```

2. Uji goal dan amati.
3. Ganti goal untuk mencari hasil perkalian 13 dengan 25.

4. Uji goal dan amati.
5. Buka jendela baru dan ketik program berikut:

```
DOMAINS
  merek,warna = symbol
  usia = byte
  harga, kilometer = ulong

PREDICATES
  nondeterm mobil(merek,kilometer,usia,warna,harga)

CLAUSES
  mobil(atoz,130000,3,merah,120000000).
  mobil(karimun,90000,4,silver,100000000).
  mobil(ceria,8000,1,hitam,75000000).

GOAL
  mobil(karimun, 90000,4, silver, 100000000).
```

6. Uji goal dan amati.
7. Ganti goal untuk mencari mobil dengan harga dibawah 100 juta rupiah.
8. Uji goal dan amati.
9. Buka jendela baru dan ketik program berikut:

```
PREDICATES
  nondeterm suka(symbol,symbol)

CLAUSES
  suka(asep,membaca).
  suka(asari,computers).
  suka(nunung,bulutangkis).
  suka(vida,bulutangkis).
  suka(astana,renang).
  suka(astana,membaca).

GOAL
  suka(Orang, membaca),
  suka(Orang, renang).
```

10. Uji goal dan amati. Apa arti goal tersebut dalam bahasa natural?
11. Ganti goal untuk mencari “siapakah orang yang suka olahraga bulutangkis?”.
12. Uji goal dan amati.

## D. PERTANYAAN/TUGAS

**Buatlah program Visual Prolog dari narasi berikut:**

Pada suatu semester di STMIK AKAKOM. Ada 5 mahasiswa yang mengambil mata kuliah Intelejensi Buatan (Irfan, Komeng, Dati, Fatima, dan Maspion); 5 mahasiswa mengambil mata kuliah PDE (Ricky, Embang, Salmin, Vina, dan Sondang) dan 5 mahasiswa lagi mengambil mata kuliah Sistem Operasi (Pamuji, Luki, Sadek, Yusida dan Eka). Setelah ujian selesai masing-masing mendapat nilai (sesuai urutan nama dari pertama) : A, D, C, B, C, E, A, D, B, C, D, E, B, A, dan A. Mahasiswa yang tidak lulus adalah mahasiswa yang nilainya di bawah C (D dan E tidak lulus).

Kemudian buatlah goal untuk mencari:

a. Nama mahasiswa yang mengikut mata kuliah Intelejensi Buatan.
b. Nama mahasiwa yang tidak lulus.
c. Nama mahasiswa yang lulus.
d. Seluruh nama matakuliah yang diajarkan.
e. Seluruh nama mahasiswa yang ada.

# Modul 4
# UNIFIKASI DAN LACAKBALIK
# (*UNIFICATION AND BACKTRACKING*)

## A. TUJUAN

1. Mengerti apa yang dimaksud unifikasi dan proses terjadinya.
2. Mengerti cara kerja lacak balik dan prinsip-prinsipnya.
3. Dapat mengendalikan proses lacak balik menggunakan predikat *fail*, *cut (!)* dan *not.*

## B. DASAR TEORI

### Unifikasi dan Lacakbalik

#### 1. Unifikasi (*Unification*)

Pada waktu Visual Prolog mencoba untuk mencocokkan suatu panggilan (dari sebuah subgoal) ke klausa (pada section **clauses**), maka proses tersebut melibatkan suatu prosedur yang dikenal dengan unifikasi (*unification*), yang mana berusaha untuk mencocokkan antara struktur data yang ada di panggilan (subgoal) dengan klausa yang diberikan. Unifikasi pada Prolog mengimplementasikan beberapa prosedur yang juga dilakukan oleh beberapa bahasa tradisional seperti melewatkan parameter, menyeleksi tipe data, membangun struktur, mengakses struktur dan pemberian nilai (*assignment*). Pada intinya unifikasi adalah proses untuk mencocokkan dua predikat dan memberikan nilai pada variabel yang bebas untuk membuat kedua predikat tersebut identik. Mekanisme ini diperlukan agar Prolog dapat mengidentifikasi klausa-klausa mana yang dipanggil dan mengikat (*bind*) nilai klausa tersebut ke variabel.

Untuk lebih jelasnya diberikan contoh berikut:

```
/* Contoh 1
------------*/

DOMAINS
 judul,pengarang = symbol
 halaman       = unsigned

PREDICATES
 buku(judul, halaman)
 nondeterm ditulis_oleh(pengarang, judul)
 nondeterm buku_tebal(judul)

CLAUSES
 ditulis_oleh(emha, "Markesot Bertutur").
 ditulis_oleh(kahlil, "Sang Nabi").

 buku("Sang Nabi", 132).
 buku("Markesot Bertutur", 379).
```

```
buku_tebal(Judul):-
        ditulis_oleh(_, Judul),
        buku(Judul, Tebal),
        Tebal > 300.
```

misal diberikan goal ditulis_oleh(X, Y). Untuk memenuhi goal tersebut Visual Prolog harus menguji setiap klausa ditulis_oleh yang ada di program untuk mencari kecocokan. Dalam usaha untuk mencocokkan argumen X dan Y dengan argumen yang ditemukan pada masing-masing klausa ditulis_oleh, Visual Prolog akan mencari dari yang paling atas (*top*) hingga ke paling bawah (*bottom*) dari program. Jika ditemukan klausa yang cocok dengan goal, maka akan mengikat nilai yang ada pada klausa ke variabel bebas sehingga goal dan klausa identik, dan goal bisa dikatakan bersatu (*unify*) dengan klausa. Operasi pencocokan ini dinamakan unifikasi.

Mari kita lihat langkah satu persatu bagaimana Visual Prolog memenuhi goal yang diberikan.

1. Karena X dan Y variabel bebas dari goal dan variabel bebas dapat diunifikasikan ke variabel manapun dari argumen (bahkan ke variabel bebas yang lain), maka pemanggil (goal) dapat diunifikasikan dengan klausa ditulis_oleh yang pertama pada program, seperti berikut:

   ```
   ditulis_oleh( X  ,   Y    ).
                 |      |
   ditulis_oleh(emha, "Markesot Bertutur").
   ```

   maka Visual Prolog akan membuat pencocokan, X diikat ke emha dan Y diikat ke "Markesot Bertutur". Pada titik ini, Visual Prolog akan menampilkan:

   ```
   X=emha, Y=Markesot Bertutur
   ```

2. Karena Prolog akan mencari semua kemungkinan solusi, maka goal juga akan diunifikasikan ke klausa ditulis_oleh yang kedua pada program, yaitu ditulis_oleh(kahlil, “Sang Nabi”) dan Visual Prolog akan menampilkan:

   ```
   X=kahlil, Y=Sang Nabi
   2 Solutions
   ```

Untuk lebih dapat memahami unifikasi, diberikan goal yang lebih kompleks yaitu buku_tebal(X)., dan kita lihat langkahnya satu persatu:

1. Pertama kali Visual Prolog akan mencari fakta atau *head* dari *rule* yang cocok dengan goal, maka ditemukan buku_tebal(Judul). Kemudian mencari pada klausa tersebut untuk mengunifikasi argumennya. Karena X variabel bebas, maka X dapat diunifikasikan ke variabel manapun. Judul juga merupakan variabel bebas dari *head rule* buku_tebal, maka terjadi unifikasi antara X dan Judul, sehingga X=Judul.
2. Goal yang cocok dengan *head* dari *rule* dan unifikasi membuat Visual Prolog melanjutkan pengujian pada bagian *body* dari *rule*. Untuk menguji *body* dari *rule*,

maka akan dipanggil subgoal yang pertama dari *body* tersebut, yaitu ditulis_oleh(_, Judul). Perhatikan bahwa untuk mencari buku yang tebal kita tidak perlu mengetahui siapa pengarang buku tersebut. Untuk itu variable anonim ("_") muncul di posisi argumen pengarang. Prolog kemudian akan mencari kecocokan dari subgoal ini mulai dari bagian atas program hingga ke bagian bawah, maka pertama kali ditemukan:

```
ditulis_oleh( _  ,  Judul   ).
              |      |
ditulis_oleh(emha, "Markesot Bertutur").
```

Sehingga variabel Judul akan diikat ke nilai "Markesot Bertutur" dan subgoal berikutnya akan dipanggil, yaitu buku(Judul, Tebal).

3. Visual Prolog sekarang akan memulai pencarian untuk melakukan pencocokan pemanggil buku. Karena Judul telah diikat dengan nilai "Markesot bertutur", maka pemanggilan yang sebenarnya adalah buku("Markesot Bertutur", Tebal). Sekali lagi Visual Prolog akan mencari dari atas sampai ke bawah dari program. Pada kesempatan pertama akan melakukan pencocokan pada klausa buku("Sang Nabi", 132) dan gagal.
4. Karena gagal, maka Visual Prolog akan mencoba ke klausa yang kedua yaitu buku("Markesot Bertutur", 310) dan karena nilai argumen judul dari klausa cocok dengan dengan subgoal yang memanggil, kemudian Visual Prolog akan mengikat variabel Tebal ke nilai 310. Selanjutnya Visual Prolog akan memanggil subgoal berikutnya.
5. Subgoal terakhir dari *body rule* buku_tebal, yaitu Tebal > 300. Visual Prolog melakukan perbandingan apakah Tebal yang telah diikat nilainya ke 310 lebih besar nilainya dari 300 dan sukses (true). Pada titik ini semua subgoal yang ada pada *rule* telah sukses (true) sehingga berarti pemanggilan goal buku_tebal(X) juga sukses. Karena variabel X telah diunifikasi dengan variabel Judul, maka nilai yang telah diikat ke variabel Judul pada waktu pemanggilan *rule* juga akan diikat ke variabel X. Judul mempunyai nilai "Markesot bertutur" ketika *rule* sukses dijalankan, maka Visual Prolog akan menampilkan:

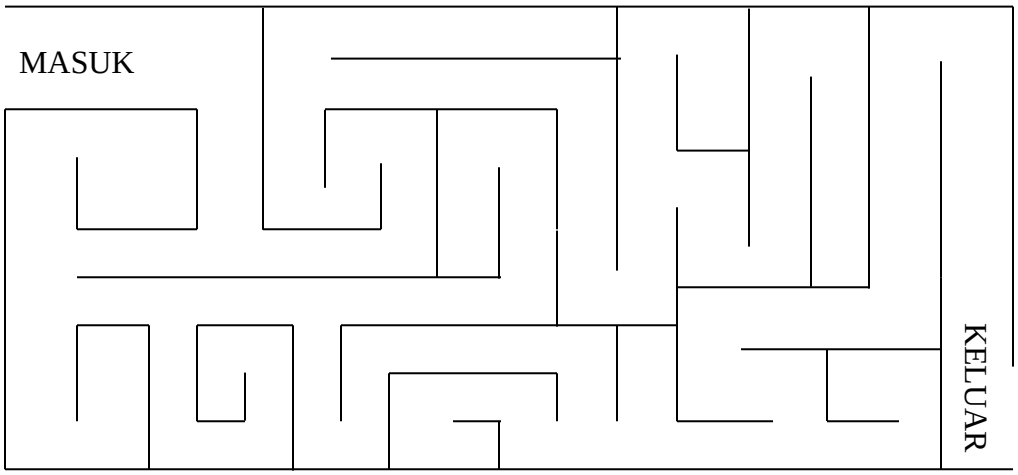
```
X=Markesot Bertutur
1 Solution
```

Ada beberapa hal penting dalam proses pencocokan atau unifikasi, yaitu:

- Pada waktu Prolog berusaha untuk memenuhi sebuah goal, Prolog memulainya dari bagian paling atas (*top*) dari program dalam rangka mencari pencocokan.
- Ketika sebuah panggilan baru terjadi, pencarian pencocokan juga dimulai dari bagian paling atas dari program.
- Ketika sebuah panggilan mengalami pencocokan yang sukses, pemanggil kembali (*is said to return*), dan giliran subgoal berikutnya diuji.
- Ketika suatu variabel telah diikat (*bound*) pada sebuah klausa, cara-cara satu-satunya untuk membebaskan ikatan tersebut adalah melalui lacakbalik (*backtracking*).

**2. Lacakbalik (*Backtracking*)**

Pada waktu menyelesaikan masalah, seringkali, seseorang harus menelusuri suatu jalur untuk mendapatkan konklusi yang logis. Jika konklusi ini tidak memberikan jawaban yang dicari, orang tersebut harus memilih jalur yang lain. Perhatikan permainan *maze* berikut. Untuk mencari jalan keluar dari *maze*, seseorang harus selalu mencoba jalur sebelah kiri terlebih dahulu pada setiap percabangan hingga menemukan jalan buntu. Ketika menemukan jalan buntu maka orang tersebut harus kembali ke percabangan terakhir (*back*-up) untuk mencoba lagi (*try again*) ke jalur kanan dan jika menemukan percabangan lagi maka tetap harus mencoba jalur kiri terlebih dahulu. Jalur kanan hanya akan sekali-sekali dipilih. Dengan begitu orang tersebut akan bisa keluar dari *maze*, dan memenangkan permainan.




Metode balik-ke-atas-dan-coba-lagi (*backing-up-and-trying-again*) ini pada Visual Prolog disebut lacakbalik (*backtracking*). Visual Prolog menggunakan metode ini untuk menemukan suatu solusi dari permasalahan yang diberikan.Visual Prolog dalam memulai mencari solusi suatu permasalahan (atau goal) harus membuat keputusan di antara kemungkinan-kemungkinan yang ada. Ia menandai di setiap percabangan (dikenal dengan titik lacak balik) dan memilih subgoal pertama untuk telusuri. Jika subgoal tersebut gagal (ekivalen dengan menemukan jalan buntu), Visual Prolog akan lacakbalik ke titik lacakbalik (*back-tracking point*) terakhir dan mencoba alternatif subgoal yang lain.

Agar jelasnya diberikan contoh program berikut:

```
/* Contoh 2
------------*/
PREDICATES
  nondeterm jenis(symbol, symbol)
  nondeterm adalah(symbol, symbol)
  hidup(symbol, symbol)
  nondeterm dapat_berenang(symbol)

CLAUSES
```

```
jenis(vertebrata,hewan).
jenis(ikan,hewan).

adalah(zebra,vertebrata).
adalah(lele,ikan).
adalah(tuna,ikan).

hidup(zebra,di_darat).
hidup(kodok,di_darat).
hidup(kodok,di_air).
hidup(tuna,di_air).

dapat_berenang(Y):-
        jenis(X,hewan),
        adalah(Y,X),
        hidup(Y,di_air).

GOAL
        dapat_berenang(Apa).
```

Untuk mengetahui terjadinya lacakbalik, akan diamati langkah demi langkah bagaimana Visual Prolog mencari solusi dari goal yang diberikan.

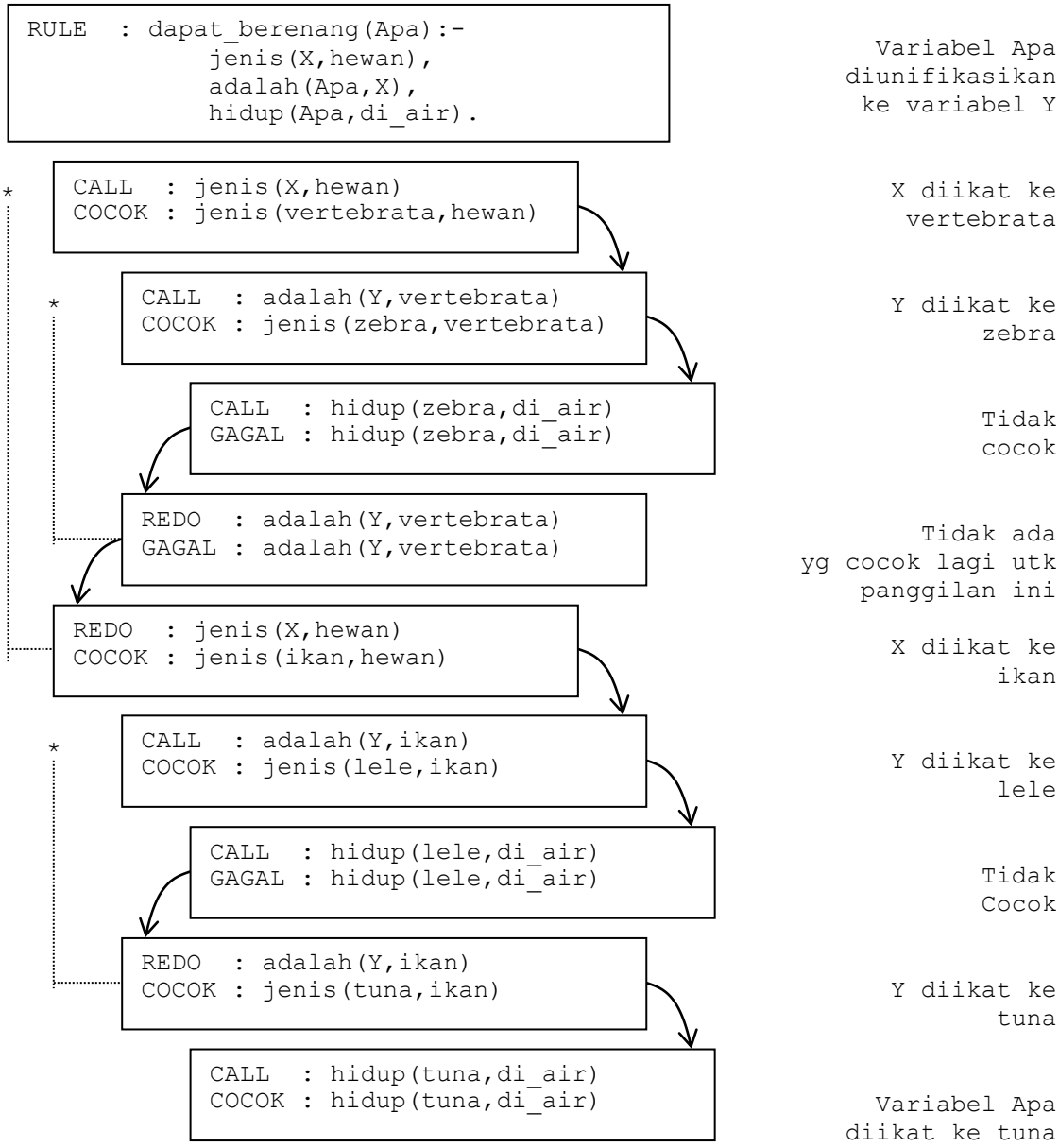
1. Visual Prolog memanggil predikat dapat_berenang dengan variabel bebas Apa. Untuk mencoba menjawab panggilan ini, Visual Prolog mencari di program untuk pencocokan. Ditemukan kecocokan dengan klausa dapat_berenang, dan variabel Apa diunifikasikan dengan variabel Y.
2. Kemudian, Visual Prolog berusaha untuk memenuhi bagian *body* dari *rule.* Untuk melakukannya, Visual Prolog memanggil subgoal yang pertama pada *body* dari *rule* tersebut yaitu jenis(X, hewan), dan mencari pencocokan untuk panggilan ini. Ditemukan pencocokan dengan fakta pertama dari klausa relasi jenis. Pada titik ini X diikat dengan nilai vertebrata. Kemudian Visual Prolog menset titik lacakbalik pada fakta jenis(vertebrata,hewan).
3. Dengan X diikat pada nilai vertebrata, Visual Prolog membuat panggilan untuk subgoal yang kedua yaitu adalah(Y, vertebrata), dan mencari pencocokan. Dan menemukan dengan fakta yang pertama adalah(zebra, vertebrata). Y diikat dengan nilai zebra dan Prolog menset titik lacakbalik pada adalah(zebra, vertebrata).
4. Kemudian Prolog mencoba untuk memenuhi subgoal yang terakhir yaitu, hidup(zebra, di_air). Karena tidak ditemukan fakta tersebut maka panggilan gagal dan Prolog memulai lacakbalik untuk menemukan solusi lain.
5. Ketika Visual Prolog lacakbalik, proses kembali ke titik lacakbalik terakhir. Pada kasus ini, titik lacakbalik terakhir berada pada subgoal yang kedua dari *rule* yaitu fakta adalah(zebra, vertebrata).
6. Ketika Visual Prolog berada pada titik lacakbalik, ia akan membebaskan variabel yang diberi nilai pada titik lacakbalik tersebut dan berusaha mencari jawaban lain untuk panggilan tersebut dalam kasus ini adalah(Y, vertebrata).

7. Visual Prolog melanjutkan pencarian ke bawah dari titik lacakbalik tersebut untuk pencocokan klausa yang lain. Karena tidak ada klausa yang ditemukan, maka terjadi lacak balik dan proses kembali ke titik lacakbalik terakhir yaitu di jenis(vertebrata, hewan).
8. Visual Prolog membebaskan variabel dan mencoba untuk menemukan solusi lain dari panggilan jenis(X, hewan). Pencarian dimulai setelah titik lacakbalik dan menemukan kecocokan dengan fakta jenis di program pada klausa jenis(ikan, hewan). X kemudian diikat pada nilai ikan, dan titik lacakbalik baru di set pada fakta tersebut.
9. Kemudian Visual Prolog bergerak ke subgoal berikutnya pada *rule*. Karena ini panggilan baru, maka pencarian dimulai dari bagian atas program dengan panggilan adalah(Y, ikan). Visual Prolog menemukan kecocokan dengan panggilan ini. Y diikat dengan nilai lele, dan sebuah titik lacakbalik baru diset pada fakta adalah(lele, ikan).
10. Karena Y diikat dengan nilai lele, panggilan subgoal berikutnya adalah hidup(lele, di_air). Karena panggilan baru, maka pencarian dimulai dari bagian atas program. Prolog mencoba setiap fakta namun gagal menemukan kecocokan dan subgol dinyatakan gagal.
11. Visual Prolog, kemudian kembali ke titik lacakbalik terakhir yaitu pada adalah(lele, ikan). Variabel yang diikat pada pencocokan ini kemudian dibebaskan. Beranjak dari titik ini, Visual Prolog mencari solusi baru untuk panggilan adalah(Y, ikan).
12. Visual Prolog menemukan kecocokan dengan klausa adalah berikutnya. Y kemudian diikat dengan simbol tuna. Kemudian Visual Prolog mencoba lagi subgoal yang terakhir dengan variabel Y diikat pada nilai tuna. Panggilannya menjadi hidup(tuna, di_air). Pencarian kembali dimulai dari bagian atas program karena merupakan panggilan baru dan menemukan kecocokan dengan fakta yang ada dan subgoal terakhir sukses (true).
13. Pada titik ini, *body* dari *rule* dapat_berenang dapat dipenuhi. Visual Prolog mengembalikan nilai Y pada pemanggil (goal) dapat_berenang(Apa). Karena Apa diikat nilainya ke Y dan Y diikat nilainya tuna, maka sekarang Apa juga diikat nilainya ke tuna pada goal tersebut. Visual Prolog kan menampilkan jawaban:

    Apa=tuna
    1 Solution.

    dan program berhenti dengan sukses.

Untuk melihat lebih jelasnya proses lacakbalik pada contoh di atas dapat dilihat pada gambar berikut:




Berikut ini beberapa prinsip dasar lacakbalik dari contoh di atas:

1. Subgoal harus dipenuhi/diuji dari urutan paling atas (*top*) sampai ke bagian paling bawah (*bottom*).
2. Klausa predikat diuji dalam urutan menurut kemunculannya di program yaitu dari atas ke bawah.
3. Ketika suatu subgoal cocok dengan bagian *head* dari *rule*, maka *body* dari *rule* tersebut berikutnya juga harus dipenuhi/diuji. Bagian *body* dari *rule* merupakan sekumpulan subgoal baru yang kemudian juga harus dipenuhi/diuji.
4. Suatu goal dikatakan terpenuhi (true) ketika suatu fakta yang cocok ditemukan untuk setiap titik ekstrim dari pohon goal tersebut.

Suatu panggilan yang menghasilkan solusi jamak disebut non-deterministik (kata kuncinya **nondeterm**) sedangkan suatu panggilan yang hanya menghasilkan satu dan hanya satu solusi dinamakan deterministik.

**3. Pengendalian proses lacakbalik**

Mekanisme lacak balik dapat menghasilkan pencarian yang tidak perlu, akibatnya program menjadi tidak efisien. Misalnya adanya beberapa jawaban yang muncul ketika kita hanya membutuhkan solusi tunggal dari masalah yang diberikan. Pada kasus lain, suatu kebutuhan untuk memaksa Visual Prolog untuk melanjutkan mencari jawaban tambahan walaupun goal tersebut sudah terpenuhi. Untuk kasus-kasus tersebut, kita harus mengontrol proses lacakbalik.

Visual Prolog menyediakan 2 alat yang memperbolehkan kita untuk mengendalikan mekanisme lacak balik yaitu predikat ***fail*** yang digunakan untuk memaksa lacakbalik dan predikat ***cut*** (ditandai dengan **!**) yang digunakan untuk mencegah lacakbalik.

**Predikat *fail***

Visual Prolog akan memulai lacakbalik jika ada panggilan yang gagal. Pada situasi tertentu, ada kebutuhan untuk memaksa lacakbalik dalam rangka mencari alternatif solusi. Visual Prolog menyediakan predikat khusus ***fail*** untuk memaksa kegagalan sehingga memicu terjadinya lacakbalik. Efek dari ***fail*** sama dengan efek memberikan perbandingan 2=3 atau subgoal yang tidak mungkin (*impossible*) lainnya. Contoh berikut ini mengilustrasikan penggunaan predikat tersebut.

```
/* Contoh 3
------------*/
DOMAINS
  nama = symbol

PREDICATES
  nondeterm ayah(nama, nama)
  setiap_orang

CLAUSES
  ayah(sunarto,cici).
  ayah(andi,udin).
  ayah(roland,yansen).
  setiap_orang:-
        ayah(X,Y),
        write(X," adalah ayah ",Y,"\n"),
        fail.
  setiap_orang.

GOAL
  setiap_orang.
```

Ketika goal internal sudah sukses, tidak ada yang memberitahu Visual Prolog untuk melakukan lacakbalik. Akibatnya, pemanggilan internal ***ayah*** hanya akan menghasilkan solusi tunggal. Sehingga digunakan ***fail*** pada predikat ***setiap_orang*** seperti pada contoh program di atas untuk memaksa lacakbalik dan dengan demikian menemukan semua jawaban yang mungkin.

**Predikat *Cut* (!)**

Visual Prolog memiliki *cut* yang digunakan untuk mencegah lacakbalik, ditulis berupa sebuah tanda seru (!). Efek dari *cut* adalah sederhana, yaitu tidak akan memungkinkan terjadinya lacakbalik melewati sebuah *cut*.

Kita menempatkan *cut* dalam program sama persis seperti menempatkan sebuah subgoal pada *body* dari suatu *rule*. Ketika proses melewati *cut*, pemanggil ke *cut* dinyatakan sukses dan subgoal berikutnya (jika ada) dipanggil. Sekali sebuah *cut* dilewati, adalah menjadi tidak mungkin untuk melakukan lacakbalik pada subgoal yang berada pada sebelum *cut* pada klausa yang sedang diproses dan adalah menjadi tidak mungkin untuk melakukan lacakbalik ke predikat lain yang mendefinisikan predikat yang sekarang diproses (predikat yang mengandung *cut*). Berikut ini contoh penggunaan *cut*.

```
/* Contoh 4
------------*/
PREDICATES
  beli_mobil(symbol,symbol)
  nondeterm mobil(symbol,symbol,integer)
  warna(symbol,symbol)

CLAUSES
  beli_mobil(Model,Warna):-
        mobil(Model,Warna,Harga),
        warna(Warna,seksi),!,
        Harga < 25000.

  mobil(ferrari,hijau,25000).
  mobil(jaguar,hitam,24000).
  mobil(jaguar,merah,26000).
  mobil(porsche,merah,24000).

  warna(merah,seksi).
  warna(hitam,wibawa).
  warna(hijau,sejuk).

GOAL
  beli_mobil(jaguar, Y).
```

Pada contoh di atas goalnya adalah mencari mobil Jaguar yang mempunyai warna seksi dengan harga yang tidak terlalu mahal. *Cut* yang ada pada *rule* beli_mobil mempunyai asumsi bahwa hanya satu Jaguar yang mempunyai warna seksi yang ada di

database dan jika harganya terlalu tinggi maka tidak perlu mencari mobil yang lain. Untuk melihat bagaimana *cut* mempengaruhi proses, kita program di atas langkah demi langkah.

1. Visual Prolog memanggil mobil, subgoal pertama dari predikat beli_mobil.
2. Tes pada mobil pertama, ferrari menghasilkan gagal.
3. Kemudian tes klausa mobil selanjutnya menemui kecocokan dan mengikat variabel Warna ke nilai hitam.
4. Karena berhasil, maka akan memanggil subgoal berikutnya apakah warna mobil yang dipilih adalah warna seksi. Hitam bukan warna seksi pada program sehingga tes gagal.
5. Visual Prolog lacakbalik ke pemanggil mobil dan sekali lagi mencari sebuah Jaguar yang memenuhi kriteria.
6. Kemudian ditemukan dan sekali lagi menguji warna mobil tersebut. Kali ini warnanya adalah seksi dan Prolog melanjutkan ke subgoal berikutnya pada *rule* yaitu *cut*. *Cut* kemudian sukses dan secara efektif "membekukan" variabel yang diikat pada klausa sebelumnya.
7. Visual Prolog sekarang melanjutkan ke subgoal selanjutnya (terakhir) pada *rule* yaitu perbandingan:

   Harga < 25000.

   Uji ini gagal dan Visual Prolog akan melakukan lacakbalik dalam rangka untuk mencari mobil yang lain untuk diuji. Namun karena adanya *cut* yang mencegah terjadinya lacakbalik, maka tidak ada jalan lain lagi untuk subgoal yang terakhir dan goal berhenti pada kondisi gagal (*failure*).

**4. Predikat *Not***

Program berikut memperlihatkan bagaimana penggunaan predikat ***not*** untuk mengidentifikasikan seorang mahasiswa teladan yaitu mahasiswa yang mempunyai Indeks Prestasi Kumulatif (IPK) minimal 3,5 dan tidak sedang dalam masa percobaan (sedang menjalani masa hukuman karena melakukan tindak kejahatan).

```
/* Contoh 5
------------*/
DOMAINS
  nama = symbol
  ipk  = real

PREDICATES
  nondeterm mahasiswa_teladan(nama)
  nondeterm mahasiswa(nama, ipk)
  masa_percobaan(nama)

CLAUSES
  mahasiswa_teladan(Nama):-
        mahasiswa(Nama, IPK),
```

```
        IPK>=3.5,
        not(masa_percobaan(Nama)).

mahasiswa("Vina Panduwinata", 3.5).
mahasiswa("Helmi Yahya", 2.0).
mahasiswa("Syahrul Gunawan", 3.7).

masa_percobaan("Vina Panduwinata").
masa_percobaan("Helmi Yahya").

GOAL
mahasiswa_teladan(X).
```

Ada satu catatan ketika menggunakan ***not*** yaitu predikat ***not*** akan sukses ketika subgoal tidak bisa dibuktikan kebenarannya. Hal ini untuk mencegah suatu situasi variabel yang belum diikat akan diikat menggunakan ***not***. Jika subgoal dengan variabel bebas dipanggil melalui ***not***, maka Visual Prolog akan mengeluarakan pesan kesalahan Free variables not allowed in 'not' or 'retractall'.

**C. PRAKTIK**

1. Ketik program pada dasar teori (Contoh 1 s.d. Contoh 5)
2. Uji masing-masing goal tersebut dan lihat hasilnya.

**D. PERTANYAAN/TUGAS**

Diberikan program sebagai berikut:

```
DOMAINS
 nama,jender,pekerjaan,benda,alasan,zat = symbol
 umur=integer

PREDICATES
 nondeterm orang(nama, umur, jender, pekerjaan)
 nondeterm selingkuh(nama, nama)
 terbunuh_dengan(nama, benda)
 terbunuh(nama)
 nondeterm pembunuh(nama)
 motif(alasan)
 ternodai(nama, zat)
 milik(nama, benda)
 nondeterm cara_kerja_mirip(benda, benda)
 nondeterm kemungkinan_milik(nama, benda)
 nondeterm dicurigai(nama)

/* * * Fakta-fakta tentang pembunuhan * * */
CLAUSES
```

```
orang(budi,55,m,tukang_kayu).
orang(aldi,25,m,pemain_sepak_bola).
orang(aldi,25,m,tukang_jagal).
orang(joni,25,m,pencopet).

selingkuh(ina,joni).
selingkuh(ina,budi).
selingkuh(siti,joni).

terbunuh_dengan(siti,pentungan).
terbunuh(siti).

motif(uang).
motif(cemburu).
motif(dendam).

ternodai(budi, darah).
ternodai(siti, darah).
ternodai(aldi, lumpur).
ternodai(joni, coklat).
ternodai(ina, coklat).

milik(budi,kaki_palsu).
milik(joni,pistol).

/* * * Basis Pengetahuan * * */

cara_kerja_mirip(kaki_palsu, pentungan).
cara_kerja_mirip(balok, pentungan).
cara_kerja_mirip(gunting, pisau).
cara_kerja_mirip(sepatu_bola, pentungan).

kemungkinan_milik(X,sepatu_bola):-
    orang(X,_,_,pemain_sepak_bola).
kemungkinan_milik(X,gunting):-
    orang(X,_,_,pekerja_salon).
kemungkinan_milik(X,Benda):-
    milik(X,Benda).

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * dicurigai semua orang yang memiliki senjata yang      *
 * kerjanya mirip dengan senjata penyebab siti terbunuh. *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

dicurigai(X):-
    terbunuh_dengan(siti,Senjata) ,
```

```
    cara_kerja_mirip(Benda,Senjata) ,
    kemungkinan_milik(X,Benda).

/* * * * * * * * * * * * * * * * * * * * * * * * * * *
 * dicurigai laki-laki yang selingkuh dengan siti. *
 * * * * * * * * * * * * * * * * * * * * * * * * * * */

 dicurigai(X):-
    motif(cemburu),
    orang(X,_,m,_),
    selingkuh(siti,X).

/* * * * * * * * * * * * * * * * * * * * * * *
 * dicurigai perempuan yang selingkuh dengan *
 * laki-laki yang juga selingkuh dengan siti *
 * * * * * * * * * * * * * * * * * * * * * * * */

 dicurigai(X):-
    motif(cemburu),
    orang(X,_,f,_),
    selingkuh(X,Lakilaki),
    selingkuh(siti,Lakilaki).

/* * * * * * * * * * * * * * * * * * * * * * * * * * *
 * dicurigai pencopet yang mempunyai motif uang.    *
 * * * * * * * * * * * * * * * * * * * * * * * * * * */

 dicurigai(X):-
    motif(uang),
    orang(X,_,_,pencopet).

 pembunuh(Pembunuh):-
    orang(Pembunuh,_,_,_),
    terbunuh(Terbunuh),
    Terbunuh <> Pembunuh, /* Bukan bunuh diri */
    dicurigai(Pembunuh),
    ternodai(Pembunuh,Zat),
    ternodai(Terbunuh,Zat).

GOAL
 pembunuh(X).
```

Terangkan langkah demi langkah bagaimana Visual Prolog memberikan jawaban terhadap goal tersebut (melalui proses unifikasi dan lacak balik).

# Modul 5
# DATA *OBJECT* SEDERHANA DAN JAMAK
# (*SIMPLE DAN COMPOUND DATA OBJECT*)

## A. TUJUAN

1. Mengerti dan dapat mengimplementasikan data *object* sederhana dan jamak.
2. Mengerti apa yang dimaksud dengan *functor* pada data *object* jamak.
3. Mengerti proses unifikasi pada data *object* jamak.

## B. DASAR TEORI

### Data *Object* Sederhana dan Jamak

#### 1. Data *Object* sederhana

Data *object* sederhana terdiri dari 2 yaitu variabel atau konstanta. Konstanta yang dimaksud tidak sama dengan konstanta simbolis yang ditulis di *section* **constants** pada bagian program. Yang dimaksud dengan konstanta di sini adalah apapun yang diidentifikasikan sebagai sebuah *object* bukan *subject* yang nilainya bisa bervariasi, seperti sebuah karakter (**char**), angka (integer atau real) atau sebuah **atom** (symbol atau string).

**Variabel**

Variable harus dimulai dengan sebuah huruf kapital (A ..Z) atau sebuah *underscore* ( _ ). Sebuah *underscore* tunggal merepresentasikan sebuah variable anonim. Variabel dalam prolog bersifat lokal bukan global, oleh karena itu jika ada dua klausa yang mengandung sebuah variabel X maka X pada kedua klausa tersebut adalah variabel yang berbeda.

**Konstanta**

Konstanta meliputi karakter, angka, dan atom. Suatu nilai konstanta juga merupakan nama dari konstanta tersebut. Konstanta 2 merepresentasikan angka 2 dan konstanta halo merepresentasikan simbol halo.

- **Karakter**
  Karakter bertipe **char**, yaitu karakter-karakter yang bisa tercetak (ASCII 32 – 127), karakter angka (0 – 9), huruf kecil (a – z), huruf kapital (A – Z) dan tanda baca. Konstanta karakter ditulis dengan diapit oleh tanda petik tunggal (‘). Contoh : ‘a’, ‘*’, ‘{‘, ‘W’.
  Jika kita menginginkan sebuah backslash atau petik tunggal menjadi karakter menuliskannya harus didahului dengan sebuah tanda backslash. Contoh : ‘\\’ (backslash), ‘\’’ (tanda petik tunggal). Beberapa karakter mempunyai fungsi khusus ketika didahului oleh karakter escape ( \ ). Contoh : ‘\n’ (ganti baris), ‘\t’ (tabulasi).
  Konstanta karakter dapat juga ditulis berdasarkan kode ASCII-nya, dengan didahului backslash. Contoh : ‘\64’ (@), ‘\90’ (Z)
- **Angka**
  Angka merupakan salah satu dari integer atau real.

- **Atom**
  Sebuah atom terdiri dari sebuah simbol atau sebuah string. Perbedaan antara simbol dan string berkaitan dengan representasi dan implementasi mesin. Visual Prolog melakukan konversi tipe otomatis diantara domain string dan domain simbol. Jadi simbol dapat digunakan untuk domain string dan string dapat digunakan untuk domain simbol. Namun string dinyatakan sebagai sesuatu yang berada diantara tanda petik ganda sedang simbol tidak membutuhkan tanda petik ganda.
  Simbol dimulai dengan sebuah huruf kecil dan hanya boleh berisikan huruf, angka, dan *underscore.* String adalah sesuatu yang diapit tanda petik ganda dan berisikan kombinasi dari karakter, kecuali ASCII NULL (0), yang dipakai untuk menandai akhir dari string.

**2. Data object jamak**

Data object jamak memperlakukan beberapa informasi sebagai sebuah item tunggal. Contohnya : tanggal 1 desember 1999. Tanggal tersebut terdiri dari 3 jenis informasi yaitu hari, bulan dan tahun. Deklarasi suatu domain yang mengandung data object jamak tanggal :

```
DOMAINS
        tanggal_jamak = tanggal(unsigned, string, unsigned)
```

dan kemudian pada section CLAUSES dapat dituliskan :

```
T = tanggal(1,"desember",1999).
```

Penulisan ini mirip dengan penulisan suatu fakta tetapi ini bukan fakta. Ini adalah sebuah data object. Data object dimulai dengan sebuah nama yang biasa disebut functor (dalam contoh adalah tanggal) yang diikuti oleh 3 argumen. Sebuah functor dalam Visual Prolog tidak seperti sebuah fungsi pada bahasa pemrograman lain. Functor tidak melakukan apa-apa, hanya sebuah nama yang mengidentifikasi sebuah jenis data object jamak yang didalamnya terdapat argumen. Argumen dari sebuah data object jamak bisa dalam bentuk data object jamak pula. Contoh:

```
tanggal = tanggal(unsigned, string, unsigned)
orang = orang(namapertama, namakedua)
ulangtahun = ulangtahun(orang,tanggal)
```

Contoh Program yang menggunakan data object jamak adalah:

```
DOMAINS
 orang                          = orang(nama,alamat)
 nama                           = nama(pertama,kedua)
 alamat                         = alamat(jalan,kota_kab,propinsi)
 jalan                          = jalan(nama_jalan,nomor)
 kota_kab,propinsi,nama_jalan = string
 pertama,kedua                  = symbol
 nomor                          = integer
```

```
GOAL
 P1 = orang(nama(diena,fatihah),alamat(jalan("Wijaya Kusuma", 12), "Berbah", "DIY")),
 P1 = orang(nama(_,fatihah),Alamat),
 P2 = orang(nama(nur,fatihah),Alamat),
 write("P1=",P1),nl,
 write("P2=",P2),nl.
```

**3. Deklarasi Domain-Campuran Jamak (*Compound Mix-Domain*)**

Deklarasi domain-campuran jamak bermaksud:

- memiliki sebuah argumen dengan kemungkinan lebih dari satu tipe argumen;
- memiliki beberapa macam argumen, masing-masing dengan tipe yang berbeda;
- memiliki beberapa macam argumen, beberapa diantaranya dengan kemungkinan lebih dari satu tipe argumen.

Agar suatu predikat Visual Prolog dapat menerima suatu argumen yang memberikan informasi dengan tipe yang berbeda maka functor tersebut harus dideklarasikan. Contoh berikut memperlihatkan klausa umur yang dapat menerima suatu argumen usia dengan tipe yang berbeda yaitu **string**, **real** atau **integer**.

```
DOMAINS
        usia = i(integer); r(real); s(string)

PREDICATES
        umur(usia)

CLAUSES
        umur(i(USIA)):- write(USIA).
        umur(r(USIA)):- write(USIA).
        umur(s(USIA)):- write(USIA).
```

Visual Prolog tidak memperbolehkan deklarasi domain sebagai berikut:

```
DOMAINS
        usia = integer; real; string
```

Berikut ini merupakan contoh program yang memperlihatkan deklarasi domain-campuran jamak.

```
DOMAINS
 benda          = buku(judul, pengarang) ;
                          kuda(nama) ; kapal ;
                          bukubank(saldo)
 judul, pengarang, nama = symbol
 saldo           = real

PREDICATES
 nondeterm milik(nama,benda)
```

```
CLAUSES
  milik(erwin, buku("Markesot Bertutur", "Emha Ainun Najib")).
  milik(erwin, kuda(buraq)).
  milik(erwin, kapal).
  milik(erwin, bukubank(1000)).


GOAL
  milik(erwin, Benda).
```

**C. PRAKTIK**

1. Ketik program berikut:

```
DOMAINS
  nama = orang(symbol,symbol) /* (Pertama , Kedua) */
  hari_lahir = tanggal_lahir(integer,symbol,integer) /* (Hari, Bulan, Tahun) */
  telepon = symbol  /* Nomor telepon */

PREDICATES
  nondeterm daftar_telepon(nama,symbol,hari_lahir)
  yang_ulang_tahun_bulan_ini
  konversi_bulan(symbol,integer)
  cek_bulan_ulang_tahun(integer,hari_lahir)
  cetak_orang(nama)

CLAUSES
  yang_ulang_tahun_bulan_ini:-
          write("**** Daftar Orang Yang Ulang Tahun Bulan Ini ****"),nl,
          write(" Nama Pertama\t\t Nama Kedua\n"),
          write("**************************************************"),nl,
          date(_, Bulan_ini, _),  /* Ambil bulan pada sistem komputer */
          daftar_telepon(Orang, _, Tanggal),
          cek_bulan_ulang_tahun(Bulan_ini, Tanggal),
          cetak_orang(Orang),
          fail.

  yang_ulangtahun_bulan_ini:-
          write("\n\n Tekan sembarang tombol..."),nl,
          readchar(_).

  cetak_orang(orang(Pertama,Kedua)):-
          write("  ",Pertama,"\t\t\t   ",Kedua),nl.

  cek_bulan_ulang_tahun(Bul,tanggal_lahir(_,Bulan,_)):-
          konversi_bulan(Bulan,Bulan1),
          Bul = Bulan1.
```

```
daftar_telepon(orang(erwin,effendy),"767-8463",tanggal_lahir(3,jan,1955)).
daftar_telepon(orang(pramudya,kurniawan),"438-8400",tanggal_lahir(5,feb,1985)).
daftar_telepon(orang(kusdiar,prihatin),"555-5653",tanggal_lahir(3,mar, 1935)).
daftar_telepon(orang(ui,yansen),"767-2223",tanggal_lahir(29,apr,1951)).
daftar_telepon(orang(roland,hutagalung),"555-1212",tanggal_lahir(12,may,1962)).
daftar_telepon(orang(andi,nuruddin),"438-8400",tanggal_lahir(17,jun,1980)).
daftar_telepon(orang(syarif,musadek),"767-8463",tanggal_lahir(20,jun,1986)).
daftar_telepon(orang(lidya,widyawati),"555-5653",tanggal_lahir(16,jul,1981)).
daftar_telepon(orang(yusida,andriani),"767-2223",tanggal_lahir(10,aug,1981)).
daftar_telepon(orang(slamet,riyadi),"438-8400",tanggal_lahir(25,sep,1981)).
daftar_telepon(orang(nur,harjanto),"438-8400",tanggal_lahir(20,oct,1952)).
daftar_telepon(orang(dian,marlini),"555-1212",tanggal_lahir(9,nov,1984)).
daftar_telepon(orang(teguh,heni),"767-2223",tanggal_lahir(15,nov,1987)).
daftar_telepon(orang(eka,ardiyanti),"438-8400",tanggal_lahir(31,dec,1981)).

konversi_bulan(jan, 1).
konversi_bulan(feb, 2).
konversi_bulan(mar, 3).
konversi_bulan(apr, 4).
konversi_bulan(may, 5).
konversi_bulan(jun, 6).
konversi_bulan(jul, 7).
konversi_bulan(aug, 8).
konversi_bulan(sep, 9).
konversi_bulan(oct, 10).
konversi_bulan(nov, 11).
konversi_bulan(dec, 12).

GOAL
  yang_ulang_tahun_bulan_ini.
```

2. Uji goal tersebut dan perhatikan hasilnya
3. Tambahkan beberapa klausa daftar_telepon untuk orang yang bernama “sri sugiarti”, “aldi badwin”, “gigi gilang” dan “titi coklat”. Berikan nilai untuk bulan pada predikat tanggal_lahir sesuai dengan bulan sekarang (pada waktu praktikum ini dilakukan) dan nilai yang lainnya terserah (temasuk nomor telepon).
4. Kemudian uji kembali goal tersebut dan perhatikan hasilnya.

## D. PERTANYAAN/TUGAS

1. Apa perbedaan data *object* sederhana dan jamak?
2. Buatlah program dengan narasi berikut ini (gunakan data *object* jamak):

   Suatu semester di AKAKOM. Ada 3 mata kuliah yang diajarkan :
   a. Nama Mata Kuliah : Intelejensi Buatan
      Dosen: Abdul Kadir

Program Studi : Manajemen Informatika
Ruang : 1

b. Nama Mata Kuliah : PDE
Dosen: Indra Yatini
Program Studi : Teknik Informatika
Ruang : 2

c. Nama Mata Kuliah : Teknik Antar Muka
Dosen : Sigit Anggoro
Program Studi : Teknik Komputer
Ruang : 3

Masing-masing matakuliah diikuti 3 orang mahasiswa, yaitu untuk mata kuliah "Intelejensi Buatan" diikuti oleh:

- Nama : Sugeng Riyadi
  Jenis Kelamin: Laki-Laki
  NIM:2002001
  Alamat asal: Jl. Sudirman No. 2, Pontianak, Kalimantan Barat
- Nama : Yulia Sugondo
  Jenis Kelamin: Perempuan
  NIM:2002002
  Alamat asal: Jl. A. Yani No. 10, Klaten, Jawa Tengah
- Nama : Budiman Sejati
  Jenis Kelamin: Laki-Laki
  NIM:2002003
  Alamat asal: Jl. Slamet Riyadi No. 45, Solo, Jawa Tengah

Untuk mata kuliah "PDE" diikuti oleh:

- Nama : Laksamana Sukardi
  Jenis Kelamin: Laki-Laki
  NIM:2002004
  Alamat asal: Jl. MT Haryono No. 10, Palembang, Sumatera Selatan
- Nama : Rini Suwandi
  Jenis Kelamin: Perempuan
  NIM:2002005
  Alamat asal: Jl. Letjen Suprapto No. 12, Surabaya, Jawa Timur
- Nama : Kwik Kian Gie
  Jenis Kelamin: Laki-Laki
  NIM:2002006
  Alamat asal: Jl. WR Supratman No. 100, Makasar, Sulawesi Selatan

Untuk mata kuliah "Teknik Antar Muka" diikuti oleh:

- Nama : Riri Reza
  Jenis Kelamin: Laki-Laki
  NIM:2002007
  Alamat asal: Jl. RW Monginsidi No. 30, Purwokerto, Jawa Tengah
- Nama : Rachel Maryam
  Jenis Kelamin: Perempuan
  NIM:2002008
  Alamat asal: Jl. Otista No. 112, Bandung, Jawa Barat

- Nama : Garin Nugroho
  Jenis Kelamin: Laki-Laki
  NIM:2002009
  Alamat asal: Jl. Tanjung Pura No. 101, Jaya Pura, Papua

Dari fakta-fakta di atas berikan pertanyaan berikut:

1. Cari Mahasiswa yang mengikuti mata kuliah “Intelejensi Buatan” dan tampilkan dengan menyertakan nama dosen, ruang kuliah, alamat asal mahasiswa.
2. Cari mata kuliah apa saja yang diajarkan pada semester tersebut dan tampilkan dengan menyertakan nama mahasiswa yang mengambil mata kuliah tersebut, ruang dan nama dosen yang mengajar.

# Modul 6
# PERULANGAN DAN REKURSI
# (*REPETITION AND RECURSION*)

## A. TUJUAN

1. Mengerti dan dapat membuat proses perulangan pada Visual Prolog.
2. Mengerti dan dapat membuat proses rekursif.
3. Mengerti apa yang dimaksud dengan rekursi ekor dan bagaimana mengimplentasikannya.
4. Mengetahui dan dapat membuat struktur data rekursif seperti tipe data pohon (*tree*).

## B. DASAR TEORI

### Perulangan dan Rekursi

Komputer memiliki bermacam kemampuan yang berguna salah satunya adalah kemampuan melakukan sesuatu berulang-ulang. Prolog dapat melakukan perulangan dalam dua hal yaitu berupa prosedur dan struktur data. Ide dari struktur data repetitif (rekursif) adalah bagaimana menciptakan struktur data yang ukuran (*size*) akhirnya belum diketahui ketika struktur tersebut pertama kali dibuat (*create*).

### 1. Proses Perulangan

Prolog menyediakan dua jenis perulangan yaitu lacakbalik (mencari jawaban jamak dari satu pertanyaan) dan rekursi (prosedur pemanggilan dirinya sendiri).

- **Lacakbalik**

Ketika suatu prosedur melakukan lacakbalik, prosedur akan mencari alternatif jawaban dari sebuah goal yang sudah terpenuhi. Lacakbalik merupakan salah satu cara untuk melakukan proses perulangan. Berikut contoh programnya:

```
/* Contoh 1
------------*/
PREDICATES
  nondeterm negara(symbol)
  cetak_negara

CLAUSES
  negara("Inggris").
  negara("Perancis").
  negara("Jerman").
  negara("Denmark").

  cetak_negara:-
        negara(X),
        write(X),       /* tulis nilai  X */
        nl,           /* ganti baris baru */
```

```
        fail.
 cetak_negara.

GOAL
 cetak_negara.
```

**Implementasi Lacakbalik dengan Loop**

Lacakbalik merupakan cara yang baik untuk mencari alternatif jawaban dari sebuah goal. Namun jika suatu goal tidak memiliki alternatif jawaban, lacakbalik masih dapat digunakan untuk melakukan perualangan. Berikut ini didefiniskan predikat dua-klausa.

```
ulang.
ulang:-ulang.
```

Ini untuk mengakali struktur kendali Prolog agar berpikir bahwa terdapat sejumlah jawaban berbeda yang tak terbatas (cara kerjanya akan dibahas pada bagian mengenai rekursi ekor / *tail recursion*). Kegunaan ulang adalah agar lacakbalik terjadinya tak terhingga). Berikut ini diberikan contoh implementasinya.

```
/* Contoh 2
------------*/
PREDICATES
 nondeterm ulang
 nondeterm mesinketik

CLAUSES
 ulang.
 ulang:-ulang.

 mesinketik:-
        ulang,
        readchar(C), /* baca sebuah karakter, ikat ke variabel C */
        write(C),
        C = '\r'.  /* Apakah ditekan Enter? Gagal jika tidak */

GOAL
 mesinketik,nl.
```

- **Rekursi**

Cara lain untuk melakukan perulangan adalah melalui rekursi. Prosedur rekursi adalah prosedur yang di dalamnya ada pemanggilan terhadap dirinya sendiri. Prosedur rekursi dapat merekam perkembangannya karena ia melewatkan (*passing*) pencacah, total, dan hasil sementara sebagai argumen dari satu iterasi ke iterasi berikutnya. Berikut ini merupakan contoh program untuk mencari faktorial dari suatu angka.

```
/* Contoh 3
-----------*/
PREDICATES
  faktorial(unsigned,real)

CLAUSES
  faktorial(1,1):-!.

  faktorial(X,FaktoX):-
          Y=X-1,
          faktorial(Y,FaktoY),
          FaktoX = X*FaktoY.

GOAL
  X=3,
  faktorial(X,Y).
```

Beberapa keunggulan dari rekursi adalah:

- Dapat mengekspresikan suatu algoritma yang secara konvensional tidak bisa dilakukan.
- Secara logika lebih sederhana dari perulangan lain.
- Digunakan terutama dalam memproses struktur data *list*.

**Rekursi Ekor (*Tail Recursion*)**

Rekursi mempunyai kelemahan yaitu memakan memori. Ketika suatu prosedur memanggil dirinya, keadaan pemanggil prosedur dari eksekusi harus disimpan sehingga prosedur pemanggil dapat meresume keadaan tersebut setelah prosedur pemanggil selesai. Ini berarti jika ada suatu prosedur memanggil dirinya 100 kali, maka ada 100 keadaan dari eksekusi yang harus disimpan. Keadaan (*state*) yang disimpan tersebut dikenal dengan nama *stack frame*. Ukuran *stack* maksimum pada platform 16 bit, seperti IBM-PC dengan sistem operasi DOS, adalah 64KByte yang bisa mengandung sekitar 3000 atau 4000 *stack frame*. Pada paltform 32 bit, secara teoritis bisa sampai ukuran Giga Byte.

Untuk mengatasi kelemahan tersebut, maka digunakan optimasi rekursi ekor (*tail recursion optimization*). Diumpamakan, selain memanggil prosedur C, prosedur B memanggil dirinya sendiri pada langkah terakhir. Ketika prosedur B memanggil B, *stack frame* dari pemanggilan B akan ditimpa nilainya oleh sebuah *stack frame* dari pemanggil B, jadi tidak menambah *stack frame* baru. Hanya argumen yang perlu di-*update* nilainya dan kemudian proses akan melompat ke awal prosedur. Dalam perspektif prosedural adalah sama seperti memperbaharui variabel pengendali perulangan.

Syarat dari rekursi ekor adalah:

- Pemanggil merupakan subgoal terakhir dari klausa tersebut.
- Tidak ada titik lacak balik sebelumnya pada klausa.

Berikut ini merupakan perbaikan dari program mencari faktorial suatu nilai dengan menggunakan rekursi ekor.

```
/* Contoh 4
------------*/
PREDICATES
  faktorial(unsigned,real)
  faktorial(unsigned,real,unsigned,real)

CLAUSES
  faktorial(N,FaktoN):-
        faktorial(N,FaktoN,1,1).

  faktorial(N,FaktoN,N,FaktoN):-!.
  faktorial(N,FaktoN,I,P):-
        IBaru = I+1,
        PBaru = P*IBaru,
        faktorial(N, FaktoN, IBaru, PBaru).

GOAL
  faktorial(3,X).
```

**2. Struktur Data Rekursif**

Tidak hanya *rule* yang bisa rekursif tapi juga struktur data. Prolog merupakan satu-satunya bahasa pemrograman yang digunakan secara luas yang memperbolehkan mendefinisikan tipe struktur data rekursif. Salah satu tipe struktu data rekursif yaitu struktur data pohon (*tree*). Pohon dalam Prolog dapat didefinsikan sebagai berikut:

```
DOMAINS
        tipepohon = pohon(string, tipepohon, tipepohon)
```

Deklarasi tersebut menyatakan bahwa sebuah pohon ditulis sebagai sebuah functor pohon yang argumennya adalah sebuah string dan dua tambahan pohon lainnya. Berikut ini merupakan contoh program untuk membuat struktur data pohon.

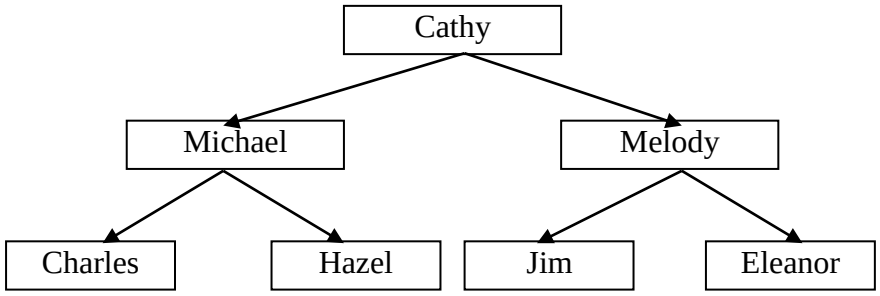
```
DOMAINS
  tipepohon = pohon(string,tipepohon,tipepohon) ; empty()

/* Contoh 5
------------*/
PREDICATES
  buat_pohon(string,tipepohon)
  sisip_kiri(tipepohon,tipepohon,tipepohon)
  sisip_kanan(tipepohon, tipepohon, tipepohon)
  bentuk_pohon(tipepohon)

CLAUSES
  buat_pohon(A,pohon(A,empty,empty)).
  sisip_kiri(X,pohon(A,_,B),pohon(A,X,B)).
```

```
sisip_kanan(X,pohon(A,B,_),pohon(A,B,X)).

bentuk_pohon(Ca3):-
        buat_pohon("Charles",Ch),
        buat_pohon("Hazel",H),
        buat_pohon("Michael",Mi),
        buat_pohon("Jim",J),
        buat_pohon("Eleanor",E),
        buat_pohon("Melody",Me),
        buat_pohon("Cathy",Ca),

        sisip_kiri(Ch, Mi, Mi2),
        sisip_kanan(H, Mi2, Mi3),
        sisip_kiri(J, Me, Me2),
        sisip_kanan(E, Me2, Me3),
        sisip_kiri(Mi3, Ca, Ca2),
        sisip_kanan(Me3, Ca2, Ca3).
```

GOAL

```
        bentuk_pohon(Pohon).
```

Pohon yang terbentuk dapat digambarkan sebagai berikut:




Untuk menjelajahi sebuah pohon, algoritma yang sering digunakan adalah:

- Jika pohon tersebut *empty* maka tidak melakukan apa-apa.
- Jika tidak, lakukan proses pada titik tersebut, kemudian jelajahi subpohon sebelah kiri, kemudian jelajahi subpohon sebelah kanan.

Dalam Prolog algoritma tersebut dapat ditulis dengan:

```
jelajahi(empty).

jelajahi(pohon(X,Y,Z):-
        lakukan sesuatu terhadap X,
        jelajahi(Y),
        jelajahi(Z).
```

Algoritma penjelajahan pohon di atas dikenal dengan *depth-first search* (pencarian sampai ke tingkat yang paling dalam terlebih dahulu) karena akan bergerak

ke bawah sejauh mungkin pada tiap percabangan yang ada kemudian baru kembali ke atas (*backing*-up) dan mencoba ke cabang yang lain (*trying another*).

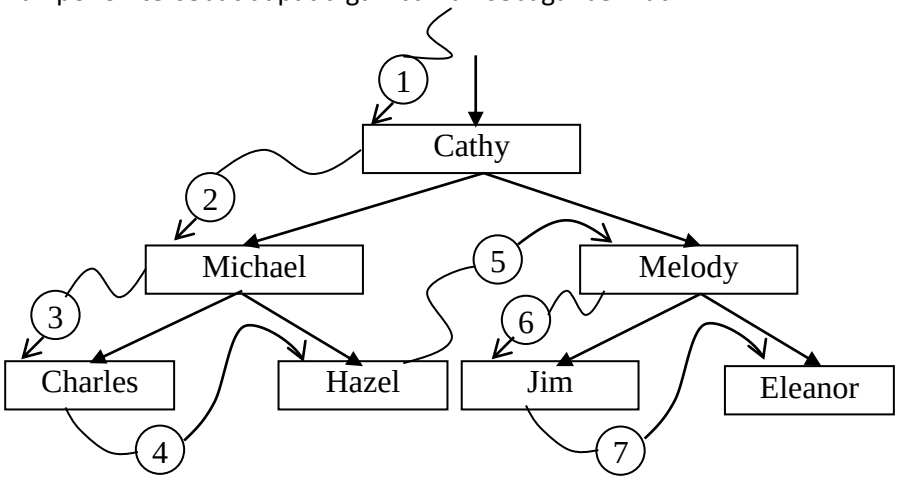
Untuk mencoba algoritma tersebut tambahkan predikat jelajah(tipepohon) di bagian section PREDICATES pada program sebelumnya dan juga tambahkan klausa berikut pada section CLAUSES:

```
jelajahi(empty).

jelajahi(pohon(Nama,Kiri,Kanan)):-
        write(Nama),nl,
        jelajahi(Kiri),
        jelajahi(Kanan).
```

Kemudian goalnya diganti menjadi:

```
GOAL
        bentuk_pohon(Pohon),
        jelajahi(Pohon).
```

Penjelajahan pohon tersebut dapat digambarkan sebagai berikut.




## C. PRAKTIK

1. Ketik program pada Contoh 1 s.d. Contoh 5 di atas.
2. Uji masing-masing goal tersebut dan perhatikan hasilnya.

## D. PERTANYAAN/TUGAS

**Ketik program di bawah ini:**

```
DOMAINS
  pohonchar = pohon(char, pohonchar, pohonchar); akhir
```

```
PREDICATES
  nondeterm lakukan(pohonchar)
  kerjakan(char, pohonchar, pohonchar)
  buat_pohon(pohonchar, pohonchar)
  sisip(char, pohonchar, pohonchar)
  cetak_pohon(pohonchar)
  nondeterm ulang

CLAUSES
  lakukan(Pohon):-
      ulang,nl,
      write("**********************************************************"),nl,
      write("Ketik 1 meng-update pohon\n"),
      write("Ketik 2 mencetak pohon\n"),
      write("Ketik 7 keluar\n"),
      write("**********************************************************"),nl,
      write("Masukkan angka - "),
      readchar(X),nl,
      kerjakan(X, Pohon, PohonBaru),
      lakukan(PohonBaru).

  kerjakan('1',Pohon,PohonBaru):-
      write("Ketik karakter # untuk mengakhiri: "),
      buat_pohon(Pohon, PohonBaru).
  kerjakan('2',Pohon,Pohon):-
      cetak_pohon(Pohon),
      write("\nTekan sembarang tombol"),
      readchar(_),nl.
  kerjakan('7', _, akhir):-
      exit.

  buat_pohon(Pohon, PohonBaru):-
      readchar(C),
      C<>'#',!,
      write(C, " "),
      sisip(C, Pohon, PohonTemp),
      buat_pohon(PohonTemp, PohonBaru).
      buat_pohon(Pohon, Pohon).

  sisip(Baru,akhir,pohon(Baru,akhir,akhir)):-!.
  sisip(Baru,pohon(Elemen,Kiri,Kanan),pohon(Elemen,KiriBaru,Kanan)):-
      Baru<Elemen,!,
      sisip(Baru,Kiri,KiriBaru).
  sisip(Baru,pohon(Elemen,Kiri,Kanan),pohon(Elemen,Kiri,KananBaru)):-
      sisip(Baru,Kanan,KananBaru).
```

```
 cetak_pohon(akhir).
 cetak_pohon(pohon(Item,Kiri,Kanan)):-
     cetak_pohon(Kiri),
     write(Item, " "),
     cetak_pohon(Kanan).

 ulang.
 ulang:-ulang.

GOAL
 write("********** Sortir Pohon Karakter ***********"),nl,
 lakukan(akhir).
```

Uji program tersebut dan pilihlah angka 1, kemudian ketikkan nama lengkapmu masing-masing (tanpa spasi) dan diakhiri tanda #. Kemudian pilih angka 2 untuk melihat hasilnya. Contoh:

```
*************** Sortir Pohon Karakter ********************

*****************************************************
Ketik 1 meng-update pohon
Ketik 2 mencetak pohon
Ketik 7 keluar
*****************************************************
Masukkan angka –

Ketikkan angka 1 dan muncul:
Ketik karakter # untuk mengakhiri:

Ketik namamu misalnya erwin effendy dan diakhiri tanda pagar

Ketik karakter # untuk mengakhiri: e r w i n e f f e n d y

Lalu muncul menu kembali:

*************** Sortir Pohon Karakter ********************

*****************************************************
Ketik 1 meng-update pohon
Ketik 2 mencetak pohon
Ketik 7 keluar
*****************************************************
Masukkan angka –

Ketikkan angka 2 maka akan muncul:
```

d e e e f f i n n r w y
Tekan sembarang tombol

Pertanyaannya adalah:

1. Gambarkan struktur pohon yang terbuat dari serangkaian karakter dari namamu!
2. Apa yang dimaksud dengan struktur data rekursif.
3. Mengapa rekursi ekor (*tail recursion*) lebih baik dari rekursi biasa?

# Modul 7
# LIST

## A. TUJUAN

1. Mengerti apa yang dimaksud dengan *list* dan dapat mengimplementasikannya pada bahasa Prolog.
2. Memahami konsep *head* (kepala) dan *tail* (ekor) pada *list.*
3. Dapat memproses *list* seperti menampilkan *list*, menghitung jumlah anggota *list,* keanggotaan *list* dan menambahkan suatu *list* ke *list* yang lain.

## B. DASAR TEORI

Pada Prolog, yang dimaksud dengan *list* adalah sebuah *object* yang didalamnya mengandung sejumlah *object* yang lain (jumlahnya dapat berubah-ubah). *List* dalam bahasa pemrograman lain bisa disamakan dengan tipe data *pointer* (C dan Pascal). Berikut ini cara penulisan *list* pada Prolog.

```
[ 1, 2, 3 ]      /* list yang mengandung integer 1, 2 dan 3 */
[ kucing, anjing, tikus ] /* list yang terdiri dari 3 buah symbol */
[ “Syarif Musadek”, “Yusida Andriani”, “Diana Putri” ]
    /* list yang terdiri dari 3 buah string */
```

Untuk mendeklarasikan *list* yang elemennya bertipe integer dapat dilakukan seperti berikut ini.

```
DOMAINS
    integerlist = integer*
```

Tanda asterik (*) berarti domain tersebut merupakan sebuah *list*. Tanda asterik juga dipakai di bahasa C untuk pendeklarasian tipe data *pointer*. Pada Pascal pendeklarasian *pointer* menggunakan tanda ^. Elemen *list* bisa berupa apapun, termasuk suatu *list* yang lain, namun semua elemen dari suatu *list* harus berasal dari domain yang sama. Contoh:

```
DOMAINS
    elementlist = element*
    element = i(integer); r(real); s(string)
```

Contoh di atas bermaksud untuk mendeklarasikan suatu *list* yang elemennya bisa mempunyai 3 tipe yang berbeda yaitu integer, real atau string.

***Head* dan *Tail* (Kepala dan Ekor)**

*List* adalah suatu data *object* jamak rekursif (*recursive compound object*). *List* terdiri dari 2 bagian yaitu *head*, yang merupakan elemen pertama dari *list* dan *tail*, elemen sisanya. *Tail* dari *list* adalah juga merupakan sebuah *list*, sedangkan *head* dari *list* merupakan sebuah elemen. Contoh:

head dari list [a, b, c] adalah a
tail dari list [a, b, c] adalah [b, c]

Bagaimana jika suatu *list* hanya mempunyai satu elemen?

head dari [c] adalah c
tail dari [c] adalah []

Jika kita selalu mengambil elemen pertama dari suatu *list* maka pada akhirnya kita akan mendapat sebuah *list* kosonng (*empty list*) yang dtulis dengan tanda [ ] (kurung siku buka dan kurung siku tutup). *List* kosong tidak bisa dipecah lagi menjadi *head* dan *tail.*
Prolog juga menyediakan cara untuk secara eksplisit memisahkan antara bagian *head* dan *tail* dari suatu *list*. Pemisah tersebut menggunakan tanda *vertical bar* ( | ). Contoh:

[a, b, c] ekivalen dengan [a|[b, c]]
ekivalen juga dengan [a|[b|[c]]]
dan ekivalen juga dengan [a|[b|[c|[]]]]

Pemisah tersebut bisa terletak pada bagian *list* manapun seperti:

[a, b, c, d] ekivalen denga [a, b|[c, d]]

**Mencetak elemen *list***
Berikut ini merupakan program untuk mencetak ke layar setiap elemen dari suatu *list*.

```
/* Contoh 1
-----------*/
DOMAINS
 list = integer*

PREDICATES
 cetak_list(list)

CLAUSES
 cetak_list([]).

 cetak_list([H|T]):-
    write(H),nl,
    cetak_list(T).

GOAL
 cetak_list([1,2,3]).
```

**Menghitung elemen *list***
Berikut merupakan program untuk menghitung jumlah elemen dari suatu *list*.

```
/* Contoh 2
------------*/
DOMAINS
  list = integer*

PREDICATES
  jumlah_elemen(list,integer,integer)

CLAUSES
  jumlah_elemen([], Hasil, Hasil).
  jumlah_elemen([_|T],Hasil,Pencacah):-
    PencacahBaru = Pencacah + 1,
       jumlah_elemen(T, Hasil, PencacahBaru).

GOAL
  jumlah_elemen([1,2,3],L,0).
```

**Keanggotaan *list***

Untuk mengetahui apakah suatu *object* merupakan anggota dari suatu *list* dapat digunakan program berikut.

```
/* Contoh 3
------------*/
DOMAINS
  listnama = nama*
  nama = symbol

PREDICATES
  nondeterm anggota(nama, listnama)

CLAUSES
  anggota(Nama, [Nama|_]).
  anggota(Nama, [_|Ekor]):-
      anggota(Nama,Ekor).

GOAL
  anggota(susan,[ian,susan,john]).
```

**Menambahkan suatu *list* ke *list* yang lain**

Berikut ini merupakan program untuk menambahkan suatu *list* ke *list* yang lain.

```
/* Contoh 4
------------*/
DOMAINS
  integerlist = integer*
```

```
PREDICATES
  tambah(integerlist,integerlist,integerlist)

CLAUSES
  tambah([],List,List).
  tambah([H|L1],List2,[H|L3]):-
    tambah(L1,List2,L3).

GOAL
  tambah([1,2,3],[5,6],L).
```

**C. PRAKTIK**

1. Ketik program pada Contoh 1 s.d. Contoh 4 di atas.
2. Uji goal tersebut dan perhatikan hasilnya.
3. Untuk program contoh 4, coba ganti goalnya dengan:

```
GOAL
        tambah([1, 2], [3], L), tambah(L, L, LL).
```

4. Uji goal tersebut dan perhatikan hasilnya.

**D. PERTANYAAN/TUGAS**

**Buat program dari narasi berikut dengan menggunakan *list.***

Pada suatu semester di STMIK AKAKOM. Ada 3 mata kuliah yang diajarkan yaitu “Intelejensi Buatan”, “PDE” dan “Sistem Operasi”. Yang mengikuti matakuliah “Intelejensi Buatan” adalah Supardi, Suradi, Suyatmi, Suparni dan Sujiman. Yang mengikuti mata kuliah “PDE” : Suharto, Sudirman, Supardi, Suyatmi, Sutini. Yang mengambil mata kuliah “Sistem Operasi” : Suharto, Sutini, Supardi, Suparni, Suripah. Tidak ada mahasiswa yang mempunyai nama yang sama. Jika ada nama yang sama mengikuti lebih dari 1 mata kuliah berarti mahasiswa tersebut memang mengikuti lebih dari 1 mata kuliah. Adapun nilai akhir dari mata kuliah tersebut (sesuai dengan urutan nama mahasiswa) adalah A, B, C, D, C, B, C, C, B, D, B, A, A, B, dan C.

Buat goal untuk menampilkan:

1. Nilai-nilai yang dimiliki oleh Supardi beserta mata kuliahnya.
2. Untuk mengecek apakah Suripah mengikuti perkuliahan pada semester tersebut.
3. Mahasiswa yang tidak lulus (nilai < C) pada semester tersebut beserta mata kuliahnya.

# Modul 8
# SECTION FACTS

## A. TUJUAN

1. Memahami apa kegunaan *section facts* pada Visual Prolog.
2. Dapat menyusun fakta-fakta pada *section facts*.
3. Dapat mengimplementasikan program yang menggunakan *section facts*.

## B. DASAR TEORI

*Section facts* terdiri dari fakta-fakta yang mana fakta-fakta tersebut dapat ditambah dan dihapus secara langsung dari sebuah program pada saat program sedang berjalan (*at run time*). Kita dapat mendeklarasikan sebuah predikat pada *section facts* dan predikat tersebut dapat digunakan sama halnya seperti kalau dideklarasikan pada *section predicates*.

Visual Prolog menyediakan beberapa predikat *built-in* untuk menangani hal yang berkaitan dengan penggunaan *section facts*, antara lain:

- assert, asserta dan assertz untuk menambah fakta baru pada *section facts*.
- retract dan retractall untuk menghapus fakta yang ada.
- consult untuk membaca fakta dari sebuah file dan menyertakan fakta tersebut ke dalam fakta internal.
- save menyimpan isi fakta internal ke dalam sebuah file.

**Deklarasi *Section Facts***

Kata kunci facts atau bisa juga database menandai permulaan sederetan deklarasi dari predikat yang ada pada *section facts*. Kita dapat menambahkan fakta-fakta (bukan *rule*) pada suatu *section facts* dari keyboard pada saat *run time* dengan menggunakan asserta dan assertz atau memanggil predikat consult untuk mengambil fakta tambahan dari sebuah file. Contoh deklarasinya seperti berikut ini.

```
DOMAINS
    nama, alamat = string
    umur = inteteger
    jender = lakilaki; perempuan

FACTS
    orang(nama, alamat, umur, jender)

PREDICATES
    lakilaki(nama, alamat, umur)
    perempuan(nama, alamat, umur)
    anak(nama, alamat, umur)

CLAUSES
    lakilaki(Nama, Alamat, Umur):-
            orang(Nama, Alamat, Umur, lakilaki).
```

Ada 2 syarat dalam menggunakan predikat yang dideklarasikan pada *section facts*:

1. Penambahan predikat pada *section facts* hanya berlaku sebagai fakta saja, tidak bisa sebagai *rule*.
2. Fakta-fakta yang ada di *section facts* tidak boleh mempunyai variabel bebas.

Visual Prolog memungkinkan suatu program untuk memiliki lebih dari satu *section facts*, tapi untuk membedakannya harus secara eksplisit diberi nama untuk setiap *section facts*. Contoh:

```
FACTS – db_orang
    orang(nama, alamat,umur, jender)
    lakilaki(nama, alamat, umur)
    perempuan(nama, alamat, umur)
    anak(nama, alamat, umur)
```

Nama *section facts* di atas adalah **db_orang**. Jika tidak ada maka Visual Prolog akan memberi nama standar (*default*) **dbasedom**.

**Menambah fakta pada saat *run time***

Pada saat *run time*, fakta-fakta dapat ditambah ke *section facts* dengan menggunakan predikat assert, asserta dan assertz atau me-*load* sebuah file yang berisikan fakta menggunakan predikat consult. Cara penulisannya adalah sebagai berikut:

```
asserta(<fakta>[, nama_section_facts])

assertz(<fakta>[, nama_section_facts])

assert(<fakta>[, nama_section_facts])

consult(namafile[, nama_section_facts])
```

Perbedaan assert, asserta dan assertz adalah asserta menyertakan sebuah fakta baru pada *section facts* sebelum fakta-fakta yang telah ada untuk predikat tersebut, sedangkan assertz menyertakan sebuah fakta baru setelahnya, sedangkan assert berfungsi sama seperti assertz. Sedangkan consult membaca dari sebuah file dan menyertakan fakta-fakta yang ada di file tersebut sesudah fakta-fakta yang telah ada.

Tidak seperti assertz, jika consult dipanggil hanya dengan satu argumen (tidak ada nama *section facts*) maka hanya akan menyertakan fakta-fakta yang predikatnya telah dideklarasikan di *section facts default* yaitu **dbasedom**. Jika memanggil consult dengan dua argumen (nama file dan nama *section facts*) maka hanya akan menyertakan fakta-fakta yang predikatnya dideklarasikan pada *section facts* dengan nama yang sesuai. Jika file tersebut mengandung fakta-fakta yang bukan milik dari *section facts* tersebut, maka akan terjadi *error* pada saat membaca bagian tersebut. Perlu diperhatikan bahwa consult membaca fakta satu demi satu, jika pada file ada 10 fakta dan pada fakta ke-7 terjadi *error*, maka consult akan menyertakan 6 fakta pertama pada *section facts* kemudian menampilkan pesan kesalahan.

Sebagai catatan, consult hanya bisa membaca sebuah file dengan syarat format file tersebut sama persis dengan format file yang disimpan menggunakan predikat save, yaitu:
- tidak ada karakter kapital kecuali dalam tanda petik dua (penulisan string).
- tidak spasi kecuali dalam tanda petik dua.
- tidak ada komentar.
- tidak ada baris kosong.
- tidak ada symbol tanpa di dalam tanda petik dua.

**Menghapus fakta pada saat *run time***

Predikat retract mengunifikasi suatu fakta dan menghapus fakta tersebut dari *sectio facts*. Cara penulisannya adalah sebagai berikut:

retract(<fakta>[, nama_section_facts])

retract akan menghapus fakta pertama yang cocok dengan argumen <fakta> yang diberikan. Karena retract merupakan predikat nondeterministik maka selama lacakbalik, retract akan menghapus fakta-fakta yang cocok dengan argumen <fakta>, kecuali jika fakta yang akan dihapus, predikatnya dideklarasikan deterministik. Ketika semua fakta yang cocok sudah terhapus, pemangilan retract berikutnya akan gagal.

Predikat retractall akan menghapus semua fakta yang cocok dengan argumen <fakta> dan penulisannya sebagai berikut:

retractall(<fakta>[, nama_section_facts])

retractall berperilaku sama seperti kalau didefinsiskan sebagai berikut:

```
retractall(X):- retract(X), fail. %fail untuk memaksa lacak balik
retractall(_).
```

**Menyimpan database fakta-fakta pada saat *run time***

Predikat save berfungsi untuk menyimpan fakta-fakta yang ada pada *section facts* ke dalam sebuah file. Cara penulisannya sebagai berikut:

save(nama_file[, nama_section_facts])

Jika memanggil save hanya dengan satu argumen (tidak ada nama *section facts*), maka akan menyimpan fakta-fakta dari *section facts default* **dbasedom** ke file dengan nama yang sesuai dengan argumen. Jika memanggil save dengan dua argumen (nama file dan nama *section facts*), maka akan menyimpan semua fakta yang ada pada *section facts* yang sesuai ke dalam file dengan nama yang sesuai pula.

**Kata kunci pada *section facts***

Fakta-fakta pada *section facts* dapat dideklarasikan dengan beberapa kata kunci opsional berikut:

nondeterm menentukan bahwa kemungkinan ada sejumlah fakta dari suatu

| | |
|---|---|
| | predikat sepanjang program berjalan (default) |
| determ | menentukan bahwa hanya boleh ada satu fakta dari suatu predikat sepanjang program berjalan. |
| global | menentukan bahwa *setion facts* adalah global dalam program. |
| single | menentukan hanya satu fakta dari predikat yang akan selalu ada. |
| nocopy | normalnya pemanggilan fakta akan mengikat variabel ke sebuah string atau ke sebuah *object* jamak. String atau *object* jamak tersebut akan disalin ke tumpukan dari Visual Prolog Global Stack (GStack). Dengan kata kunci ini maka tidak ada proses penyalinan terserbut. |

**C. PRAKTIK**

1. Ketik program berikut ini.

```
DOMAINS
        nama = symbol
        umur = integer
        jender = lakilaki; perempuan

FACTS
        orang(nama, umur, jender)

PREDICATES
        nondeterm cetak

CLAUSES
        orang(yulianto, 31, lakilaki).
        orang(yuliani, 31, perempuan).

        cetak:-
                orang(Nama, Umur, Jender),
                write("Nama   : " ,Nama),nl,
                write("Umur   : " ,Umur),nl,
                write("Jender : " ,Jender),nl,nl,
                fail.
        cetak.

GOAL
        save("data_org.txt"),
        cetak.
```

2. Uji goal tersebut dan perhatikan hasilnya.

3. Setelah goal tersebut diuji, maka akan terbentuk file “data_org.txt”. Cari file tersebut dan buka dengan program text editor seperti NOTEPAD.
4. Tambahkan data berikut:

```
orang(gilang, 20, lakilaki)
orang(ranti, 19, perempuan)
orang(bayu, 10, lakilaki)
orang(wening, 9, perempuan)
```

5. Kemudian ganti goalnya menjadi:

```
GOAL
        consult("data_org.txt"),
        cetak.
```

6. Uji goal tersebut dan perhatikan hasilnya.

**D. PERTANYAAN/TUGAS**

Buat program seperti pada bagian pertanyaan/tugas pada modul VII, namun semua fakta (yang ada pada *section clauses*) tidak berada pada badan program namun berada pada file “akakom.txt”. Dan uji goal-goal seperti yang juga diperintahkan pada modul VII.