**LAPORAN AKHIR**

**MATAKULIAH PRAKTIKUM KECERDASAN BUATAN**

**ANALISA PROBLEM PADA KOMPUTER UNTUK PEMULA VERSI BETA**

****

**Disusun Oleh :**

Sajid M Wildan 1515015226

Rizwan Fahrony 1515015228

M. farid Effendy 1515015094

Farhan Armani 1515015140

**Asisten Praktikum :**

|  |  |
| --- | --- |
| Alvian Nur Wahyudhi  1415015066 | Asdar Zulkiawan  1415015052 |

**PROGRAM STUDI TEKNIK INFORMATIKA**

**JURUSAN TEKNOLOGI INFORMASI DAN KOMUNIKASI**

**FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI**

**UNIVERSITAS MULAWARMAN**

**2017**

**KATA PENGANTAR**

Puji syukur penulis panjatkan kehadirat Allah SWT, karena berkat rahmat dan hidayah-Nya, penulis dapat menyelesaikan Laporan Praktikum ini dengan baik dan lancar, dimana laporan ini dibuat untuk memenuhi tugas akhir praktikum Kecerdasan Buatan pada semester 4 ini.

Dalam proses penyelesaian laporan ini, penulis memperoleh benyak

bantuan dari berbagai pihak. Baik berupa bimbingan, dorongan, petunjuk, saran,

keterangan-keterangan kritik serta data-data baik secara tertulis maupun lisan. Oleh karena itu, penulis mengucapkan terima kasih kepada :

1**. Joan Angelina, Si** selaku dosen pengajar matakuliah Kecerdasan Buatan Universitas Mulawarman

2. **Alvian Nur Wahyudhi dan Asdar Zulkiawan** selaku asisten laboratorium kelompok praktikum Kecerdasan Buatan

3. Rekan-rekan sesamamahasiswa yang telah memberikan masukan dan bantuan dalam pengerjaan program dan laporan Semoga apa yang telah diberikan mereka kepada penulis, akan mendapat imbalan dari Allah SWT. Akhir kata semua kritik dan saran atas laporan praktikum ini akan penulis terima dengan senang hati, dan akan menjadi bahan pertimbangan bagi penulis untuk menyempurnakan laporan ini.

Kelompok 1

**DAFTAR ISI**

**Kata Pengantar** i

**Daftar Isi** ii

**Daftar Tabel** iv

**Daftar Gambar**  v

**BAB I Pendahuluan** 1

1.1. Latar belakang 1

1.2. Rumusan masalah 2

1.3. Batasan masalah 2

1.4. Tujuan dan Manfaat 3

**BAB II Landasan Teori**  4

2.1 Fungsi-Fungsi Visual Prolog yang Digunakan 4

2.1.1Dasar- dasar Prolog 4

2.1.2 Unifikasi dan Lacak balik 12

2.1.3Data objek sederhana dan jamak........................................... 14

2.1.4 Perulangan & Rekursi........................................................... 16  
 2.1.5 List........................................................................................ 18

2.1.6 Section Facts......................................................................... 19

2.2 Teori Keilmuan yang di implementasikan 22

2.2.1 Trojan 22

2.2.2 Komputer Tidak Menyala 23

2.2.3 System error 23

2.2.4 Layar Tidak Muncul Apapun 23

2.2.5 Lag 24

**BAB III Metodologi** 25

3.1 Alur Pembuatan Sistem................................................................ 25

**BAB IV Hasil dan Pembahasan** 26

4.1 Tabel Kebenaran 26

4.2 Aturan 26 4.3 Analisis Aplikasi 28 4.3.1 Tampilan penyimpanan database Program 28

4.3.2 Tampilan Pengisian Nama 28

4.3.3 Tampilan Untuk Memulai Program 29

4.3.4 Tampilan jika memasukkan input selain yang seharusnya 29

4.3.5 Tampilan alasan dari inputan 30

4.3.6 Tampilan jika solusi tidak ditemukan pada database 30

4.3.7 Tampilan jika ditemui masalah 31

**BAB V Penutup** 32

5.1 Kesimpulan 32

5.2 Saran 32

**Daftar Pustaka**  33

**Lampiran**  34

1 Source Code 34

**DAFTAR TABEL**

**Tabel 4.1** Tabel Kebenaran 26

**DAFTAR GAMBAR**

**Gambar 4.1.1** Tampilan awal . 9

**Gambar 4.1.2** Login Untuk Masuk ke Program................................................. 9

**Gambar 4.1.3**Jika Mengetik ‘N’........................................................................ 10

**Gambar 4.1.4**Setelah Mengetik ‘Y’ atau ‘y’..................................................... 10

**Gambar 4.1.5**Program Mengajukan Pertanyaan............................................... 11

**BAB I**

**PENDAHULUAN**

* 1. **Latar Belakang**

Dilihat dari sejarah perkembangan teknologi informatika yang demikian pesatnya, ini akan membutuhkan sumber daya manusia yang handal untuk menanganinya. Oleh karena itu, biasanya yang bergerak dalam bidang computer berusaha menguasai dalam bidang computer terutama yang berhubungan dengan hardware computer.

                  Didalam penggunaannya, hardware computer tidak dapat luput dari kerusakan atau masalah, meskipun kerusakan itu adalah kerusakan kecil. Dan peranan seorang teknisi pun sangat dibutuhkan terutama bagi para pengguna atau pemilik computer yang tiudak mengetahui penyebab-penyebab kerusakan dan cara memperbaiki disaat computer mengalami kerusakan. Sangat disayangkan jika kerusakan yang terjadi hanyalah kerusakan kecil yang semestinya dapat diperbaiki sendiri. Sementara waktu untuk menunggu perbaikan sudah cukup lama dan membutuhkan biaya yang cukup besar.

                  Dalam tugas system pakar ini, akan dirancang suatu perangkat lunak yang dapat membantu orang-orang awam (user) yang tidak memiliki pengetahuan tentang computer atau tempat untuk mengidentifikasi kerusakan yang terjadi pada umumnya. Perangkat lunak ini akan menuntun user untuk mengidentifikasi kerusakan dengan cara memilih jenis kerusakan, mengikuti langkah-lagkah pemeriksaan dan akhirnya menemukan penyebab kerusakan.

Untuk mengetahui jenis kerusakan yang dialami berdasarkan gejala ataupun ciri-ciri dari pengguna narkoba, maka diperlukan suatu sistem pakar (*expert system*) yang berfungsi untuk menggantikan peranan teknisi dalam menangani kerusakan yang dilatarbelakangi oleh kekurangtahuan user awam. Sistem pakar tersebut dapat mengetahui ciri-ciri dan jenis kerusakan yang dialami oleh user. Sistem pakar (*expert system*) adalah sistem yang berusaha mengadopsi pengetahuan manusia ke komputer, agar komputer dapat menyelesaikan masalah seperti yang biasa dilakukan oleh para ahli. Ide dasarnya adalah kepakaran ditransfer dari seorang pakar ke komputer, pengetahuan yang ada disimpan dalam komputer, dan pengguna dapat berkonsultasi pada komputer itu untuk suatu nasehat, lalu komputer dapat mengambil inferensi atau menyimpulkannya, seperti layaknya seorang pakar, kemudian menjelaskannya ke pengguna tersebut, sistem pakar terkadang lebih baik cara kerjanya dari pada seorang pakar manusia.

Sistem pakar yang baik dirancang agar dapat menyelesaikan suatu permasalahan tertentu dengan meniru kerja dari para ahli khususnya peranan pakar teknisi dalam masalah ini. Dengan sistem pakar ini, orang awam pun dapat menyimpulkan dan mengetahui masalah yang cukup rumit yang sebenarnya hanya dapat diselesaikan dengan bantuan para ahli seperti halnya pihak teknisi.

* 1. **Rumusan Masalah**

Untuk dapat mendiagnosa pengguna narkoba melalui sistem pakar maka rumusan masalahnya adalah “Bagaimana merancang dan membangun sebuah Sistem Pakar pendeteksi kerusakan umum yang dialami pecandu. Dengan Menggunakan Program Visual Prolog”*.*

* 1. **Batasan Masalah**

Agar pembahasan penelitian ini tidak menyimpang dari apa yang telah dirumuskan, maka diperlukan batasan-batasan. Batasan-batasan dalam penelitian ini adalah :

1. Program digunakan hanya untuk mendeteksi bagian yang rusak dan human error
2. Tidak memberikan solusi spesifik untuk memperbaiki
3. tanda-tanda yang dibahas adalah tanda-tanda umum yang terjadi di komputer.
4. Perancangan program  aplikasi Analisa kerusakan computer  menggunakan  bahasa pemrograman *Visual Prolog 5.2*
   1. **Tujuan dan Manfaat**

Tujuan yang ingin dicapai adalah :

1. Membangun suatu sistem analisa deteksi jenis kerusakan yang dialami komputer, yang dapat membantu dan mempermudah proses mendeteksi kerusakan tanpa harus pergi ke technical support langsung jika tidak terlalu parah.
2. Memberikan informasi deteksi kerusakan komputer.
3. Menerapkan metode system pakar visual prolog

**BAB II**

**LANDASAN TEORI**

**2.1 Fungsi-Fungsi Visual Prolog yang Digunakan**

Terdapat beberapa fungsi yang digunakan dalam program sistem analisa kerusakan komputer untuk pemula

**2.1.1 Dasar-dasar Prolog**

**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.

**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 (.).

**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.

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.

**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

**d. Komentar**

Programmer yang baik selalu memberikan catatan atau komentar untuk menjelaskan sesuatu yang mungkin tidak jelas bagi orang lain (atau bahkan bagiprogrammer 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 \*/

2.1.3 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**).

* **Section Domains**

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

* **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.

**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.

* **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.

**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.

**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.

**2.1.2 Unifikasi Dan Lacakbalik (Unification And Backtracking)**

**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.

**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 diantara 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 alternative subgoal yang lain.

**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 kasuskasus 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 alternative 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.

**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*).

**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).

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 variable bebas dipanggil melalui ***not***, maka Visual Prolog akan mengeluarakan pesan kesalahan Free variables not allowed in ‘not’ or ‘retractall’.

**2.1.3 Data Object Sederhana Dan Jamak (Simple Dan Compound Data Object)**

**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).

1. **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.

1. **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**, yaitukarakter-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.

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.

**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.

**2.1.4 Perulangan Dan Rekursi (Repetition And Recursion)**

**A. 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 ayang sudah terpenuhi. Lacakbalik merupakan salah satu cara untuk melakukan proses perulangan.

* **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 duaklausa.

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).

* **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.

* **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 procedural 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.

**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*).

**2.1.5 List**

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 \*/

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.

***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.

* + 1. **Section Facts**

*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.

* **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

**2.2 Teori Keilmuan yang Diimplementasikan**

Terdapat beberapa keilmuan yang di implementasikan dalam mendeteksi kerusakan komputer. Teori keilmuwan yang di implementasikan pada program kali ini yaitu :

**2.2.1 Trojan**

 bersifat "stealth" (siluman dan tidak terlihat) dalam operasinya dan seringkali berbentuk seolah-olah program tersebut merupakan program baik-baik

- terjadi setelah menginstall program baru baru saja,

- Antivirus mendeteksi sesuatu yang mencurigakan.

Saat mengunjungi suatu situs, terdapat beberapa pop-up yang muncul. Dan ketika akan mengakhiri kunjungan, tiba-tiba browser membuka secara otomatis beberapa halaman tidak dikenal.

•      Tampilan Kotak Pesan tak dikenal yang tampak di layar perangkat. Pesan biasanya berisi beberapa pertanyaan yang bersifat pribadi.

•      Tampilan mengalami perubahan dengan sendirinya, misalnya teks screensaver yang baru, tanggal/waktu, perubahan volume bunyi dengan sendirinya, pointer mouse bergerak sendirinya, CD-ROM drive membuka dan menutup sendiri.

•      Outlook Express memakan waktu yang cukup lama atau terlihat hang saat Anda menutupnya.

•      Adanya file yang rusak atau hilang

•      Program yang tidak diketahui terlihat aktif di task manager  
  
•      Notifikasi firewall muncul tentang outbound komunikasi dari sumber yang tidak diketahui.

**2.2.2 Komputer tidak menyala**

ada banyak kemungkinan komputer tidak mau menyala umumnya selain hal teknis bisa juga disebabkan human error, penyebab yang mungkin terjadi

- stop kontak rusak

- PSU belum tercolok

- ada drive yang tertinggal di dalam input komputer, baik cdrom, floppy drive dan sebagainya

**2.2.3 System error**

umumnya sistem error bisa terjadi karena user melakukan pengaturan yang tidak seharusnya, baik dari user resmi maupun user usil

komputer yang terdeteksi mengalami sistem error antara lain yaitu :

- komputer mengeluarkan syntax aneh,

- ada peringatan peringatan aneh,

**2.2.4 Layar tidak muncul apa-apa**

umumnya user sering kebingungan karena layar monitor tidak muncul sesuatu, komputer yang terdeteksi mengalami kerusakan, penyebabnya antara lain yaitu :

- kabel monitor lupa dicolok,

- video card tidak layak pakai,

- Monitor memang sudah,

**2.2.5 Lag**

lag adalah keadaan stagnan pada proses komputer, hal ini sangat merepotkan bagi user yang ingin menggunakan komputer karena dituntut untuk bekerja secara cepat. beberapa penyebab lag dapat terdeteksi, antara lain yaitu :

-ram dan video card sudah melemah,

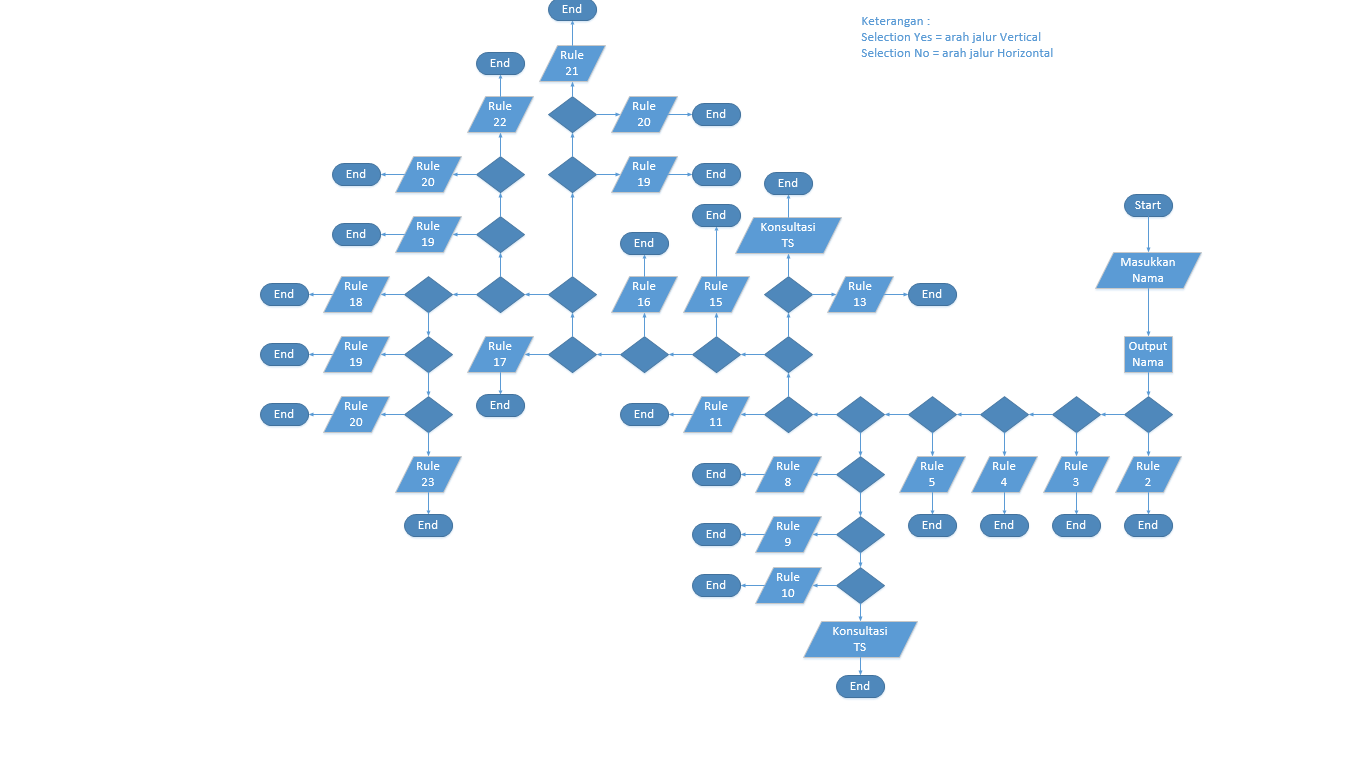
- microprocessor sudah tidak bagus

selain hal itu ada beberapa hal sulit untuk dilakukan dan dikerjakan bagi seorang pemula sehingga disarankan untuk berkonsultasi pada technical support untuk dukungan dan pemeliharaan computer anda.

**BAB III**

**METODOLOGI**

**3.1 Alur Pembuatan Sistem**



Gambar 3.1 Alur Pembuatan Sistem.

Pertama kita jalan dulu programnya setelah itu akan ada tampilan untuk memasukkan nama, setelah mengisi nama, maka user akan disambut dan disuguhkan pada pertanyaan pertanyaan? jika ada menjawab ya maka tekan tombol ‘Y’. Jika tidak maka anda menekan tombol ‘t’ untuk memilih jawaban tidak, ada bisa tekan tombol m untuk memunculkan alasan dan akibat yang terjadi berdasarkan aturan.

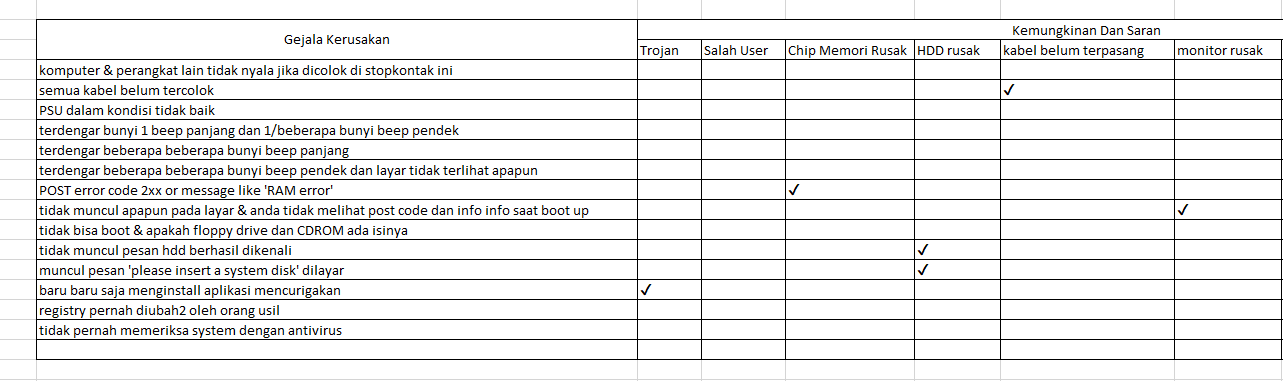
Setelah memilih program untuk dijalankan maka gejala – gejala yang ada pada program akan ditampilkan satu per satu, untuk memilih gejala kerusakan yang anda hadapi maka tekan tombol ‘y’ apabila gejala tersebut tidak terjadi pada anda maka tekan tombol ‘t’ dan begitu seterusnya. Tetapi apa bila semua gejala yang ada tidak terjadi pada anda berarti anda tidak mengalami kerusakan apapun.

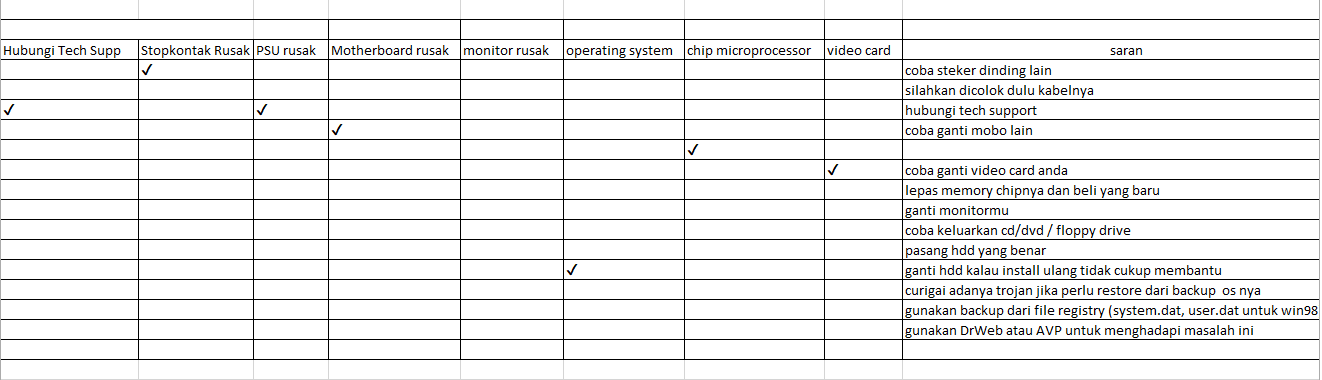
**BAB IV**

**HASIL DAN PEMBAHASAN**

* 1. **Tabel Kebenaran dan Keputusan**

Tabel 4.1 Tabel Kebenaran





Tabel Kebenaran adalah tabel yang digunakan melihat nilai kebenaran dari suatu premis/pernyataan. Terdapat penyebab kerusakan yang dapat dilihat diatas

Tabel 4.2 Aturan

rule(1, "ada aliran listrik di area sekitar anda").

rule(2, "coba steker dinding lain",).

rule(3, "lebih teliti, kalimat anda tidak selaras",).

rule(4, "mencolok kabelnya, silahkan dicolok dulu kabelnya",).

rule(5"harus berhati-hati,kondisi terlihat bahaya silahkan buka jendela dan beritahu technical support masalah anda").

rule(6, "computer sedang menyala").

rule(7, "mobo processor dan video card baik baik saja").

rule(8 , "coba mobo lain").

rule(9, " tahu bahwa ada pada microprocessor masalahnya",).

rule(10, "ganti video card anda").

rule(11, "coba lepas memory chipnya dan beli yang baru",).

rule(12, "monitor dan memory baik baik saja").

rule(13 , "ganti monitormu",).

rule(14, "system mencoba boot up dari hdd").

rule(15, " keluarkan cdrom atau floppy disk lebih dahulu").

rule(16, "pasang dengan benar").

rule(17"ganti hdd kalau install ulang tidak cukup membantu").

rule(18, "coba ganti user",).

rule(19, "curigai adanya trojan jika perlu restore dari backup os nya").

rule(20, "system mencoba boot up dari hdd", "meminta bantuan spesialis unix atau technical support anda").

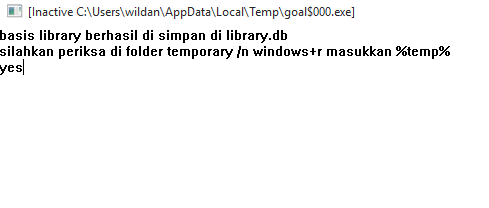
rule(21, "meminta bantuan OS ini terlalu sulit untuk anda, silahkan hubungi technical support anda").

rule(22,"gunakan backup dari file registry (system.dat, user.dat untuk win98)".

rule(23"gunakan DrWeb atau AVP untuk menghadapi masalah ini").

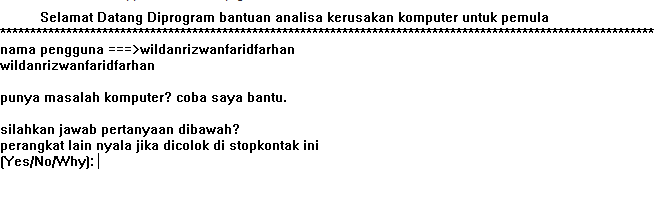
rule(24"hubungi technical support anda yang memahami error pada os anda").

4.3 Analisis Aplikasi



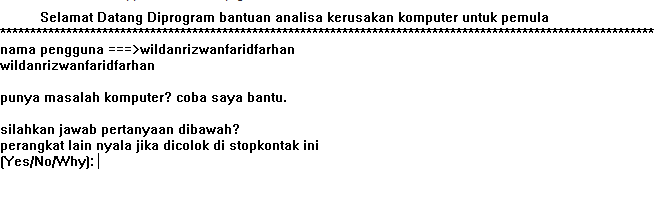
Gambar 4.3.1 : Tampilan penyimpanan database Program.

Dimana pada saat program pertamakali mau dijalankan maka kita harus menciptakan pusat library sebagai sumber database, yang berisi aturan dan kondisi yang akan dijalankan dan dialami user.



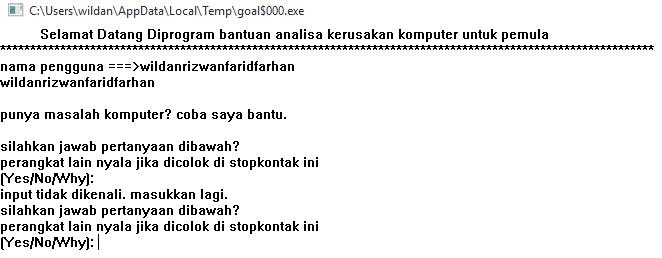
Gambar 4.3.2 : Tampilan Pengisian Nama.

Setelah menu tampilan awal, disini juga ada pengisian data diri dari seseorang yang menggunakan program ini, jadi sebelum masuk untuk memulai program pengguna wajib mengisi nama dan umur terlebih dahulu.



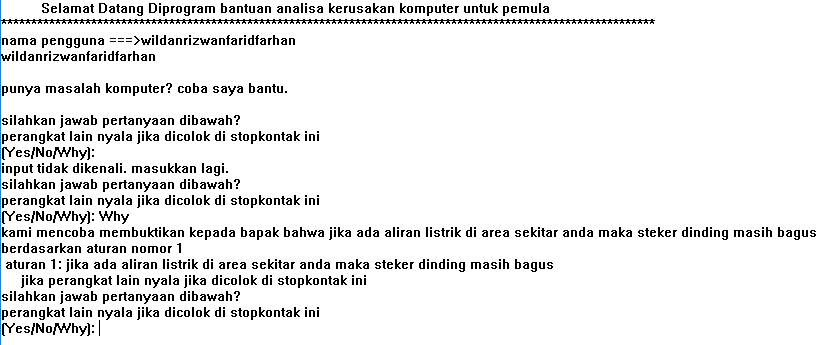
Gambar 4.3.3 : Tampilan Untuk Memulai Program.

Jadi, setelah mengisi nama anda, setelah itu tekan enter maka anda akan masuk kedalam program, setelah itu untuk memulai proses menjalankan program. Anda akan dihadapkan pada pertanyaan awal.



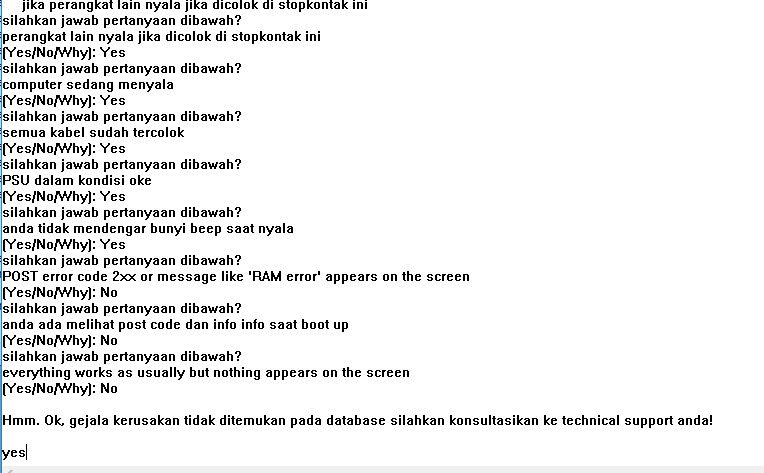
Gambar 4.3.4 : Tampilan jika memasukkan input selain yang seharusnya.

Jadi apabila anda menekan tombol selain ‘Y/y’ atau N/n maka program akan mengarahkan anda ke tampilan sebelumnya yaitu meminta anda memasukkan inputan yang seharusnya.



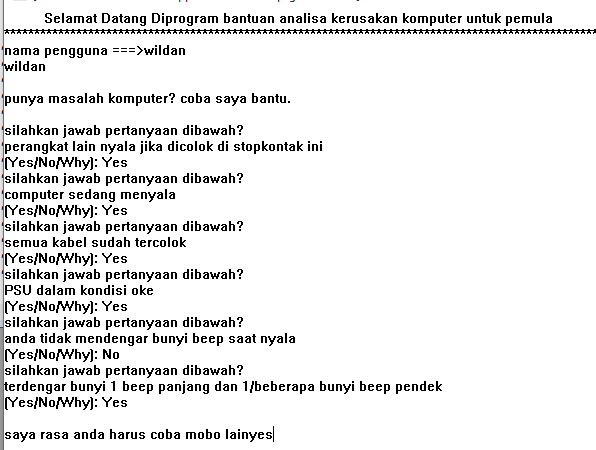
Gambar 4.3.5 : Tampilan alasan dari inputan.

jika kita masukkan inputan w yang artinya why maka kita akan diberikan alasan pertanyaan dan pembuktian dari pernyataan.



Gambar 4.3.6 : Tampilan jika solusi tidak ditemukan pada database.

Untuk memulai program tekan tombol ‘Y/y’. Setelah itu gejala – gejala yang ada akan tampil satu per satu, seperti contoh diatas gejala yang pertama muncul yaitu Apakah pupil mata menyempit ? disini kami memilih ‘t’ maka program akan mengarahkan ke gejala yang selenjutnya yaitu merasa gatal pada hidung? jika ‘y’ maka akan diarahkan ke gejala merasa mual dan muntah – muntah? jika ‘y’ maka akan diarahkan ke gejala suka menyendiri? jika ‘y’ maka akan diarahkan ke gejala selanjutnya yaitu apakah pupil mata mengecil? jika ‘y’ maka akan diarahkan ke gejala tekanan darah menurun? jika ‘y’ maka dari semua gejala yang ada bisa disimpulkan bahwa anda mengkonsumsi narkotika jenis heroin. Dan setelah mengetahui jenis narkoba yang anda konsumsi program ini juga memberitahukan dampak yang akan terjadi apa bila ada masih mengkonsumsinya dalam jangka waktu yang panjang. Setelah itu ada menu pilihan jika anda ingin kembali ke gejala – gejala narkotika silahkan tekan tombol ‘M/m’ dan jika ingin keluar dari program tekan tombol ‘X/x’.



Gambar 4.3.7 : Tampilan jika ditemui masalah.

saat program dijalankan dan user menjawab pertanyaan dengan seharusnya maka program akan mengeluarkan output tertentu sesuai jawaban yes atau no yang dihadapi.

**BAB V**

**PENUTUP**

**5.1 Kesimpulan**

Setelah melalui tahap pengujian pada sistem Analisa kerusakan komputer untuk pemula, maka dapat diambil beberapa kesimpulan sebagai berikut :

System pakar adalah system yang berusaha mengadopsi pengetahuan manusia kekomputer, agar komputer dapat menyelesaikan masalah yang seperti biasa dilakukan oleh para ahli.

Ada beberapa hal penting yang perlu diperhatian bahwa secangih apapun suatu system atau sebesar apapun basis pengetahuan yang dimiliki, tentu saja ada kelemahannya sebagai konsekuensi logis kelemahan manusia sebagai penyusun element-elementnya. Bahwa system tidak memilik inisiatif untuk melakukan suatu tindakan diluar dari apa yang telah diprogamkan untuknya.

Berdasarkan hasil pengujian, program system pakar ini berguna untuk membantu dan mempermudah pengguna dalam memperoleh informasi mengenai jenis kerusakan yang dialami serta mendapatkan saran atau informasi bagi pengguna.

**5.2 Saran**

Beberapa hal yang dapat diungkap sebagai saran untuk pengembangan sistem analisa pendeteksi jenis kerusakan komputer adalah sebagai berikut :

Dilakukan pengembangan program sejenis dengan permasalahan domain yang lebih luas.

Data mengenai penyebab dan semua mengenai komputer kiranya dapat lebih dimaksimalkan sampai mendapatkan informasi atau perhitungan yang lebih akurat serta dicari alternative lain yang memungkinkan penyelesaian yang jauh lebih baik.

**DAFTAR PUSTAKA**

<http://kuliah-oti.blogspot.co.id/2015/11/pengolahan-string-prolog-pertemuan-5.html>

http://shimaduasembilan.blogspot.co.id/2014/02/sistem-pakar-diagnosis-kerusakan.html

http://tanti1308100067.blogspot.co.id/2009\_09\_01\_archive.html

Lampiran

1. Source Code Knowledgebase

domains

CondNum=Integer.

RuleNum=Integer.

FactNum=Integer.

Conditions=CondNum\*.

HistoryTree=RuleNum\*.

Kind=String.

facts

rule(RuleNum, Kind, Kind, Conditions).

condition(CondNum, String).

nama(String).

clauses

condition(1, "perangkat lain nyala jika dicolok di stopkontak ini").

condition(2, "computer sedang menyala").

condition(3, "semua kabel sudah tercolok").

condition(4, "PSU dalam kondisi oke").

condition(5, "anda tidak mendengar bunyi beep saat nyala").

condition(6, "terdengar bunyi 1 beep panjang dan 1/beberapa bunyi beep pendek").

condition(7, "terdengar beberapa beberapa bunyi beep panjang").

condition(8, "terdengar beberapa beberapa bunyi beep pendek dan layar tidak terlihat apapun").

condition(9, "anda ada melihat post code dan info info saat boot up").

condition(10, "everything works as usually but nothing appears on the screen").

condition(11, "floppy drive dan CDROM dalam kondisi kosong").

condition(12, "muncul tulisan 'harddisk berhasil dikenali'").

condition(13, "muncul pesan 'please insert a system disk' dilayar").

condition(14, "OS merupakan produk windows").

condition(15, "registry tidak pernah dirubah oleh orang orang usil").

condition(16, "anda sudah memeriksa sistem dengan antivirus").

condition(17, "OS merupakan produk unix").

condition(18, "anda beberapa lama ini ada menginstall program baru").

condition(19, "POST error code 2xx or message like 'RAM error' appears on the screen").

% Complimentaries of above conditions

condition(20, "not perangkat lain nyala jika dicolok di stopkontak ini").

condition(21, "not computer sedang menyala").

condition(22, "not semua kabel sudah tercolok").

condition(23, "not PSU dalam kondisi oke").

condition(24, "not anda ada melihat post code dan info info saat boot up").

condition(25, "not anda tidak mendengar bunyi beep saat nyala").

condition(26, "not floppy drive dan CDROM dalam kondisi kosong").

condition(27, "not muncul tulisan 'harddisk berhasil dikenali'").

condition(28, "not OS merupakan produk windows").

condition(29, "not registry tidak pernah dirubah oleh orang orang usil").

condition(30, "not anda sudah memeriksa sistem dengan antivirus").

condition(31, "not OS merupakan produk windows").

condition(32, "not anda beberapa lama ini ada menginstall program baru").

rule(1, "ada aliran listrik di area sekitar anda", "steker dinding masih bagus", [1]).

rule(2, "ada aliran listrik di area sekitar anda", "coba steker dinding lain", [20]).

rule(3, "ada aliran listrik di area sekitar anda", "lebih teliti, kalimat anda tidak selaras", [21]).

rule(4, "ada aliran listrik di area sekitar anda", "mencolok kabelnya, silahkan dicolok dulu kabelnya", [22]).

rule(5, "ada aliran listrik di area sekitar anda", "harus berhati-hati,kondisi terlihat bahaya silahkan buka jendela dan beritahu technical support masalah anda", [23]).

rule(6, "steker dinding masih bagus", "computer sedang menyala" , [2, 3, 4]).

rule(7, "computer sedang menyala", "mobo processor dan video card baik baik saja", [5]).

rule(8, "computer sedang menyala", "coba mobo lain", [25, 6]).

rule(9, "computer sedang menyala", " tahu bahwa ada pada microprocessor masalahnya", [25, 7]).

rule(10, "computer sedang menyala", "ganti video card anda", [25, 8]).

rule(11, "mobo processor dan video card baik baik saja", "coba lepas memory chipnya dan beli yang baru", [19, 24]).

rule(12, "mobo processor dan video card baik baik saja", "monitor dan memory baik baik saja", [9]).

rule(13, "mobo processor dan video card baik baik saja", "ganti monitormu", [10, 24]).

rule(14, "monitor dan memory baik baik saja", "system mencoba boot up dari hdd", [11, 12]).

rule(15, "monitor dan memory baik baik saja", " keluarkan cdrom atau floppy disk lebih dahulu", [26]).

rule(16, "monitor dan memory baik baik saja", "pasang dengan benar", [27]).

rule(17, "system mencoba boot up dari hdd", "ganti hdd kalau install ulang tidak cukup membantu", [13]).

rule(18, "system mencoba boot up dari hdd", "coba ganti user", [14, 15, 16]).

rule(19, "system mencoba boot up dari hdd", "curigai adanya trojan jika perlu restore dari backup os nya", [18]).

rule(20, "system mencoba boot up dari hdd", "meminta bantuan spesialis unix atau technical support anda", [17]).

rule(21, "system mencoba boot up dari hdd", "meminta bantuan OS ini terlalu sulit untuk anda, silahkan hubungi technical support anda", [28, 31]).

rule(22, "system mencoba boot up dari hdd", "gunakan backup dari file registry (system.dat, user.dat untuk win98)", [29]).

rule(23, "system mencoba boot up dari hdd", "gunakan DrWeb atau AVP untuk menghadapi masalah ini", [30]).

rule(24, "system mencoba boot up dari hdd", "hubungi technical support anda yang memahami error pada os anda", [28, 31]).

goal

save("library.db"),

write("basis library berhasil di simpan di library.db"),

nl,

write("silahkan periksa di folder temporary /n windows+r masukkan %temp%"),

nl.

Source Code Running Program

domains

CondNum=Integer. /\*kita deklarasikan integer\*/

RuleNum=Integer.

Conditions=CondNum\*. /\*kita deklarasikan bahwa conditions menggunakan pointer condnum\*/

HistoryTree=RuleNum\*.

Kind=String. /\*kind adalah variabel biasa menggunakan string\*/

Pasien,nama=string.

facts

rule(RuleNum, Kind, Kind, Conditions). /\*disini fakta yang disediakan\*/

condition(CondNum, String). /\*condnum dipakai \*/

yes(CondNum).

no(CondNum).

nama(string).

predicates

% mesin Inferencenya

tambahnama.

nondeterm mulai(HistoryTree, Kind).

priksa(RuleNum, HistoryTree, Conditions).

noTest(CondNum).

% mengambil data dari user, tanyauser adalah cari kata benda, prosesjawab adalah bagian utama

% Getting data from user. tanyauser is a hull, prosesjawab is a core part.

tanyauser(HistoryTree, RuleNum, CondNum, String).

prosesjawab(HistoryTree, RuleNum, String, CondNum, Integer).

% Providing explanatory data for user.

subCat(Kind, Kind, Kind).

showConditions(Conditions, String).

munculaturan(RuleNum, String).

report(HistoryTree, String).

reverse(Conditions, Conditions).

reverse1(Conditions, Conditions, Conditions).

clauses

% Main predicate. Invokes routine-action perfoming predicates.

mulai(\_, MyGoal) :-

not(rule(\_, MyGoal, \_, \_)),

!,nl,

write("saya rasa anda harus ", MyGoal),!.

mulai(HistoryTree, MyGoal) :-

rule(RuleNum, MyGoal, SubGoal, Conditions),

priksa(RuleNum, HistoryTree, Conditions),

mulai([RuleNum|HistoryTree], SubGoal).

% checking conditions before implementing a rule

priksa(RuleNum, HistoryTree, [CondListHead|CondListTail]) :-

yes(CondListHead), % User answered positively

!,

priksa(RuleNum, HistoryTree, CondListTail). % Continue

priksa(\_, \_, [CondListHead|\_]) :-

no(CondListHead), % User answered negatively. Stop further checking.

!,

fail,write("failkepake").

% Two predicates dealing with conditions starting with 'not'

priksa(RuleNum, HistoryTree, [CondListHead|CondListTail]) :-

condition(CondListHead, ConditionString),

fronttoken(ConditionString, "not", \_NewCondString), % Delete not from ConditionString if suitable

frontchar(\_NewCondString ,\_, NextCondString), % Delete one character from \_NewCondString

condition(NewCondListHead, NextCondString),

noTest(NewCondListHead),

!,

priksa(RuleNum, HistoryTree, CondListTail).

% See above

priksa(\_, \_, [CondListHead|\_]) :-

condition(CondListHead, ConditionString),

fronttoken(ConditionString, "not", \_NewCondString),

frontchar(\_NewCondString, \_, NextCondString),

condition(NewCondListHead, NextCondString),

yes(NewCondListHead),

!,

fail.

% Call tanyauser predicate if above mentioned don't work

priksa(RuleNum, History, [CondListHead|CondListTail]) :-

condition(CondListHead, Text),

!,

tanyauser(History, RuleNum, CondListHead, Text),

priksa(RuleNum, History, CondListTail).

priksa(\_,\_,[]).

% Helping predicates for negatively formulated conditions

noTest(Condition) :-

no(Condition),

!.

noTest(Condition) :-

not(yes(Condition)),

!.

tanyauser(HistoryTree, RuleNum, CondNum, Text) :-

write("silahkan jawab pertanyaan dibawah?\n", Text, "\n(Yes/No/Why): "),

readchar(Key), % sperti scanf pada bahasa c, berguna terima inputan

upper\_lower(Key, Answer), % Convert ke lower

prosesjawab(HistoryTree, RuleNum, Text, CondNum, Answer).

prosesjawab(\_, \_, \_, CondNum,'y') :-

!,

assert(yes(CondNum)), % Add to knowledge base

write("Yes\n").

prosesjawab(\_, \_, \_, CondNum, 'n') :-

!,

assert(no(CondNum)),

write("No\n"),

fail.

% Answering user 'why' questions and displaying them good formated

prosesjawab(HistoryTree, RuleNum, Text, CondNum, 'w') :-

!,

write("Why\n"),

rule(RuleNum, StartGoal, SubGoal, \_),

!,

subCat(StartGoal, Subgoal, String1),

concat("kami mencoba membuktikan kepada bapak bahwa ", String1, String2),

concat(String2, "\nberdasarkan aturan nomor ", String3),

str\_int(RuleNumAsString, RuleNum), % Convert integer to string.

concat(String3, RuleNumAsString, Answer1),

munculaturan(RuleNum, String4),

concat(Answer1, String4, Answer2),

report(HistoryTree, Output),

concat(Answer2, Output, Answer3),

write(Answer3),

nl,

tanyauser(HistoryTree, RuleNum, CondNum, Text).

% Otherwise

prosesjawab(HistoryTree, RuleNum, Text, CondNum, \_) :-

write("\ninput tidak dikenali. masukkan lagi.\n"),

tanyauser(HistoryTree, RuleNum, CondNum, Text).

munculaturan(RuleNum, String) :-

rule(RuleNum, MyGoal1, MyGoal2, CondList),

!,

str\_int(RuleNumAsString, RuleNum), % Convert RuleNum from integer to string

concat("\n aturan ", RuleNumAsString, Answer), % Following four lines just to make output nicer

concat(Answer, ": ", Answer1),

subCat(MyGoal1, MyGoal2, String1),

concat(Answer1, String1, Answer2),

concat(Answer2, "\n jika ", Answer3),

reverse(CondList, ReversedCondList), % Output to user in better way

showConditions(ReversedCondList, Conditions),

concat(Answer3, Conditions, String).

showConditions([], ""). % Nothing to show

showConditions([Conditions], Answer) :-

condition(Conditions, Answer),

!.

showConditions([Head|Tail], Answer) :-

%

condition(Head, Text),

!,

concat("\n dan ", Text, NextString),

showConditions(Tail, NextAnswer),

concat(NextAnswer, NextString, Answer).

% Provides good indenting

subCat(MyGoal1, MyGoal2, String) :-

format(String, "jika % maka %", MyGoal1, MyGoal2).

report([], ""). % Nothing to report

report([RuleListHead|RuleListTail], String) :-

rule(RuleListHead, MyGoal1, MyGoal2,\_),  
 !,

subCat(MyGoal1, MyGoal2, String1),

concat("\nsaya telah menunjukan bahwa: ", String1, String2),

concat(String2,"\nBerdasarkan aturan nomor ", String3),

str\_int(RuleListHeadAsString, RuleListHead),

concat(String3, RuleListHeadAsString, String4),

concat(String4, ":\n ", String5),

munculaturan(RuleListHead, Str),

concat(String5, Str, String6),

report(RuleListTail, NextString),

concat(String6, NextString, String).

reverse(Conditions, NewConditions) :-

reverse1([], Conditions, NewConditions).

reverse1(Conditions, [], Conditions).

reverse1(Conditions, [CondListHead|CondListTail], NewConditions) :-

reverse1([CondListHead|Conditions], CondListTail, NewConditions).

tambahnama:- write("nama pengguna ===>"),

readln(Pasien),

assert(nama(Pasien)),write(Pasien).

goal

consult("sage.db"),

write(" Selamat Datang Diprogram bantuan analisa kerusakan komputer untuk pemula "),nl,

write("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"),nl,

tambahnama,

nl,nl,

write("punya masalah komputer? coba saya bantu."),

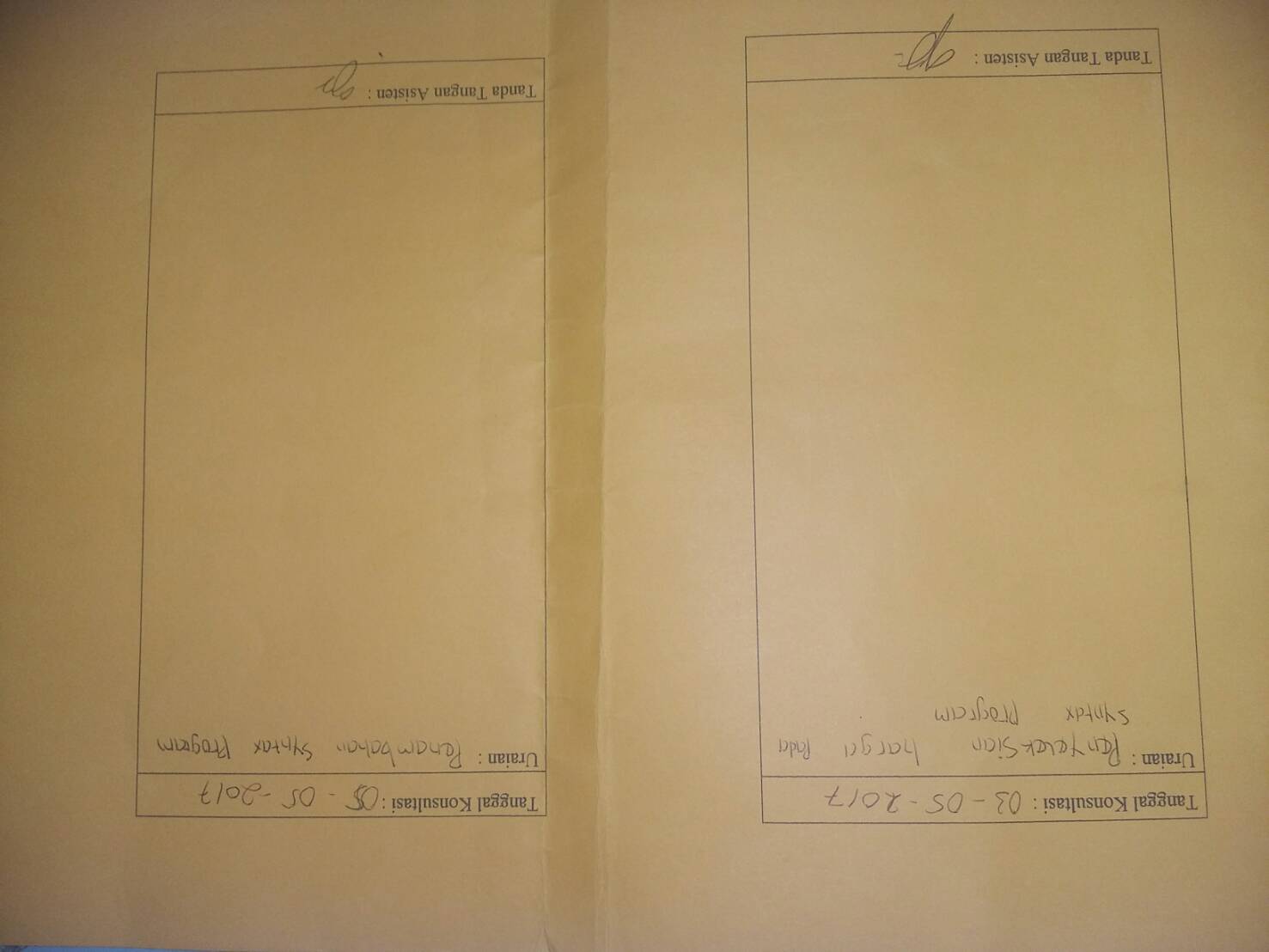
nl, nl,

mulai([], "ada aliran listrik di area sekitar anda");

write("\nHmm. Ok, gejala kerusakan tidak ditemukan pada database silahkan konsultasikan ke technical support anda!"),

nl, nl.

Lampiran

****