การพัฒนาระบบผู้เชี่ยวชาญช่วยซ่อมบำรุงเครื่องคอมพิวเตอร์ด้วย CLIPS

A Development of Computer Maintenance Expert System by CLIPS

โดย

นายจุฑาสิทธิ์ โรหิตรัตนะ

รหัส 42067016

อาจารย์ที่ปรึกษา

รศ. นุชรี เปรมชัยสวัสดิ์

รายงานนี้เป็นส่วนหนึ่งของวิชาโครงการพัฒนาระบบงาน
หลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
ภาคเรียนที่ 2 ปีการศึกษา 2543
คณะเทคโนโลยีสารสนเทศ
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ชื่อหัวข้อ	การพัฒนาระบบผู้เชี่ยวชาญช่วยซ่อมบำรุงเครื่องคอมพิวเตอร์ด้วย CLIPS
นักศึกษา	นาย จุฑาสิทธ์ โรหิตรัตนะ
อาจารย์ที่ปรึกษา	รศ. นุชรี เปรมชัยสวัสดิ์
ระดับการศึกษา	วิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
แขนงวิชา	วิทยาการสารสนเทศ
ปีการศึกษา	2543

บทคัดย่อ

ระบบผู้เชี่ยวชาญ(Expert System) คือ ระบบคอมพิวเตอร์ซึ่งจำลองความสามารถในการตัดสินใจของผู้เชี่ยวชาญเพื่อแก้ปัญหาเฉพาะด้าน ภาษาที่ใช้ในการสร้างระบบผู้เชี่ยวชาญมีอยู่หลายตัว เช่น LISP หรือ PROLOG เครื่องมือตัวหนึ่งก็คือ CLIPS ซึ่งสร้างขึ้นมาเพื่อใช้สร้าง rule แบบ forward chaining โดยเฉพาะ โครงการนี้เป็นการศึกษาการวิธีใช้งาน CLIPS ในการสร้างระบบผู้เชี่ยวชาญช่วยแก้ปัญหาเกี่ยวกับอุปกรณ์คอมพิวเตอร์ทั้งภายนอก เช่น จอภาพ เมาส์ คีย์บอร์ด ลำโพง โมเด็ม และเครื่องพิมพ์ และอุปกรณ์ภายในเช่น การ์ดแสดงผล การ์ดเสียง ซีดีรอม ฟล็อปปี้ดิสก์ เมนบอร์ด แรม และฮาร์ดดิสก์ รวมทั้งแก้ปัญหาระบบปฏิบัติการ Windows 95/98 ในบางเรื่องด้วย

Title A Development of Computer Maintenance Expert System by CLIPS

Student Mr. Juthasit Rohitratana

Advisor Nucharee Praemchaisawas

Level of Study Master of Science in Information Technology

Major Information Science

Academic Year 2000

Abstract

Expert System is a computer system that emulates the decision-making ability of a human expert for solving the special problems that the expert has knowledge or skill. There are many languages for building expert system such as LISP or PROLOG. A Tool for this is CLIPS. CLIPS supports only forward chaining rules. This project is to study development Maintenance Computer Expert System by CLIPS which solve problems about monitor, mouse, keyboard, modem, printer, speaker and inner peripheral such as sound card, display card, RAM, hard disk, floppy disk, CD ROM, case and mainboard including some problems about Windows 95/98.

กิตติกรรมประกาศ

ขอขอบคุณเพื่อนๆและพี่ๆทุกคนในห้อง Project และห้อง Thesis ที่ให้ความร่วมมือ ให้กำลังใจ ให้ความสนุกสนาน ให้เสียงหัวเราะ และบรรยากาศกับความรู้สึกดีๆ

ขอขอบคุณผู้รับผิดชอบห้องทั้งสองที่เอื้อเฟื้อสถานที่ให้พวกเราได้ทำงานกัน

ขอขอบคุณพี่จี พี่ติ๊ก พี่บุญช่วย ที่ช่วยประสานงานและช่วยเหลือพวกเราเป็นอย่างดี และให้ความอบอุ่นเป็นกันเองกับพวกเราทุกคน

ขอขอบคุณรศ.นุชรี เปรมชัยสวัสดิ์ อาจารย์ที่ปรึกษาวิชานี้ที่ช่วยชี้แนะแนวทางและให้คำปรึกษาที่มีค่ายิ่ง จนกระทั่งโครงการนี้เป็นรูปเป็นร่างและสำเร็จลุล่วงด้วยดี

ขอขอบคุณดร.ภัทรชัย ลิลิตโรจน์วงศ์ อาจารย์ประจำวิชา Expert System ที่ช่วยให้ความรู้เกี่ยวกับ Expert System กระจ่างชัดเจนยิ่งขึ้น

และสุดท้ายขอขอบคุณคุณพ่อ คุณแม่ ที่ให้แต่สิ่งดีๆแก่ผมโดยไม่หวังสิ่งใดตอบแทน

สารบัญ

บทที่	หน้า

สารบัญตาราง

หน้า

ตารางที่

สารบัญภาพ

หน้า

ภาพที่

บทที่ 1
บทนำ

1.1 ความเป็นมาของโครงการ

ในปัจจุบัน(ปีพ.ศ. 2543) เครื่องคอมพิวเตอร์เข้ามามีบทบาทอย่างมากในชีวิตประจำวัน มีอยู่แทบทุกสถานที่ ทั้งสถานที่ทำงานไม่ว่าจะภาครัฐหรือเอกชน สถานที่สาธารณะ เช่นห้องสมุด หรือแม้กระทั่งภายในบ้านพักส่วนบุคคล มีทั้งเครื่องคอมพิวเตอร์ที่ต่อระบบเครือข่าย ไม่ว่าจะเป็น LAN, WAN, INTRANET หรือ INTERNET และเครื่องคอมพิวเตอร์ที่ไม่ได้ต่อเข้ากับเครือข่ายใดๆ หรือที่เรียกว่า Stand Alone หรือเครื่องคอมพิวเตอร์ส่วนบุคคลที่เรียกว่า Personal Computer (PC) ไม่ว่าจะเป็นเครื่องคอมพิวเตอร์ประเภทไหนก็ตาม ล้วนเป็นสิ่งอำนวยความสะดวกที่จำเป็น จนเกือบจะกลายเป็นปัจจัยที่ห้าของมนุษย์บางคนไปเสียแล้ว

เนื่องจากการใช้งานอย่างหลากหลายในทุกหน่วยงาน ผู้ใช้งานเครื่องคอมพิวเตอร์(User) ก็ย่อมมีหลายระดับ ทั้งที่เป็นระดับผู้เชี่ยวชาญการใช้งาน ระดับพอใช้งานได้ จนถึงระดับเริ่มต้น เมื่อใช้งานเครื่องคอมพิวเตอร์ก็ย่อมมีบ้างที่จะเกิดปัญหาขึ้น ผู้ใช้ที่มีความรู้ก็อาจแก้ปัญหาได้ด้วยตนเอง แต่ผู้ใช้ที่ไม่มีประสบการณ์หรือความรู้เกี่ยวกับปัญหาที่เกิดขึ้น มักจะแก้ปัญหาไม่ได้ และต้องเรียกผู้เชี่ยวชาญหรือผู้ที่มีความรู้มาช่วย ในหน่วยงานบางแห่ง ผู้ที่มีความรู้หรือผู้เชี่ยวชาญด้านคอมพิวเตอร์อาจมีน้อย ดังนั้นเมื่อเกิดปัญหาขึ้นบ่อยครั้ง ไม่ว่าจะเป็นปัญหาเล็กหรือใหญ่ก็ตาม มักจะเรียกผู้เชี่ยวชาญไปช่วยเสมอ นับเป็นภาระอย่างมากของผู้เชี่ยวชาญเหล่านี้ โครงการนี้จึงถูกจัดทำขึ้นเพื่อช่วยแบ่งเบาภาระของผู้เชี่ยวชาญเหล่านี้

1.2 วัตถุประสงค์ของโครงการ

- **เพื่อแบ่งเบาภาระหน้าที่ของผู้เชี่ยวชาญด้านคอมพิวเตอร์ในหน่วยงาน ในบางหน่วยงานที่มีผู้เชี่ยวชาญน้อย เมื่อเกิดปัญหาขึ้นมากๆ ผู้เชี่ยวชาญที่มีจำนวนน้อยจะรับภาระที่หนักมาก เมื่อมีภาระหน้าที่มาก ก็ส่งผลกระทบต่อภาวะจิตใจและอารมณ์ของผู้เชี่ยวชาญเหล่านั้น อาจเป็นเหตุให้ประสิทธิภาพการทำงานลดลงได้ ถ้านำระบบผู้เชี่ยวชาญเข้ามาช่วย ก็จะทำให้ผู้ใช้ไม่ต้องรบกวนผู้เชี่ยวชาญ ทั้งระบบผู้เชี่ยวชาญนี้ก็ยังทำงานได้อย่างมีประสิทธิภาพ เรียกใช้งานได้บ่อยเท่าที่ต้องการ และไม่ต้องกังวลเรื่องอารมณ์ด้วย**

- เพื่อให้ผู้ใช้งานได้มีประสบการณ์ในการแก้ปัญหาเครื่องคอมพิวเตอร์ด้วยตนเอง เป็นการพัฒนาบุคลากรในองค์กรด้วย

1.3 ขอบเขตของโครงการ

ระบบผู้เชี่ยวชาญซ่อมบำรุงเครื่องคอมพิวเตอร์ มีขอบเขตในการแก้ปัญหาเรื่องของ Hardware โดยเฉพาะ โดยจะครอบคลุมในเรื่องของการแก้ปัญหาอุปกรณ์ภายนอกต่างๆ เช่น จอภาพ, ลำโพง, เครื่องพิมพ์, Modem, Mouse และ Keyboard รวมทั้งอุปกรณ์ภายในบางตัวเช่น การ์ดแสดงผล, Main board, RAM, Hard disk, Case, Sound Card, Floppy disk และ CD ROM โดยจะครอบคลุมการแก้ปัญหาของระบบปฏิบัติการ Windows 95/98 เป็นบางเรื่องด้วย

1.4 ขั้นตอนการดำเนินงาน

1.4.1 ขั้นขอนการวางแผน

- กำหนดเป้าหมายของโครงการ
- กำหนดขอบเขตของโครงการ
- ศึกษาและเลือกเครื่องมือที่จะใช้ในการพัฒนาระบบ

1.4.2 ขั้นตอนการปฏิบัติงาน

- ศึกษาค้นคว้า รวบรวมข้อมูล เพื่อทำ Knowledge base
- เขียนโปรแกรมพัฒนาระบบ
- ทดสอบโปรแกรม โดยผู้ใช้และผู้เชี่ยวชาญ
- ปรับปรุงโปรแกรม ตาม Feedback ที่ได้รับ

บทที่ 2
ระบบผู้เชี่ยวชาญเบื้องต้น

2.1 ปัญญาประดิษฐ์ (Artificial Intelligence)

Artificial Intelligence (AI) หรือปัญญาประดิษฐ์ เป็นศาสตร์แขนงหนึ่งซึ่งทำให้เครื่องจักรมีความสามารถทำสิ่งต่างๆได้เหมือนมนุษย์ วิทยาการที่เป็น AI ได้แก่

- Speech understanding/recognition – ทำให้เครื่องจักรสามารถเข้าใจ รับรู้คำสั่งจากเสียงได้
- Natural language understanding – ทำให้เครื่องจักรมีความเข้าใจภาษาของมนุษย์ที่ใช้
- Case-base reasoning – อาศัยเหตุการณ์ที่เกิดขึ้นมาก่อนช่วยในการตัดสินใจ เช่น ในงานของทนายความและแพทย์
- Neural network – เลียนแบบการทำงานของสมองมนุษย์
- Computer vision – ทำให้เครื่องจักรมองเห็นภาพเหมือนตามนุษย์มองเห็น
- Machine translation – การแปลงภาษาเครื่อง
- Robotic - นำไปใช้สร้างหุ่นยนต์
- Automatic programming – เครื่องจักรโปรแกรมตัวเองโดยอัตโนมัติ
- Expert system – ระบบผู้เชี่ยวชาญที่สามารถตัดสินใจได้เหมือนมนุษย์ผู้เชี่ยวชาญ

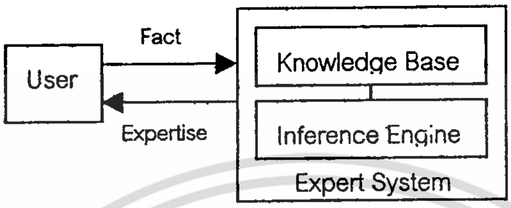
2.2 ระบบผู้เชี่ยวชาญ (Expert System)

ระบบผู้เชี่ยวชาญ (Expert System) คือ ระบบคอมพิวเตอร์ที่จำลองความสามารถในการตัดสินใจของมนุษย์ผู้เชี่ยวชาญ เพื่อแก้ปัญหาเฉพาะด้านที่ผู้เชี่ยวชาญมีความรู้หรือความชำนาญ

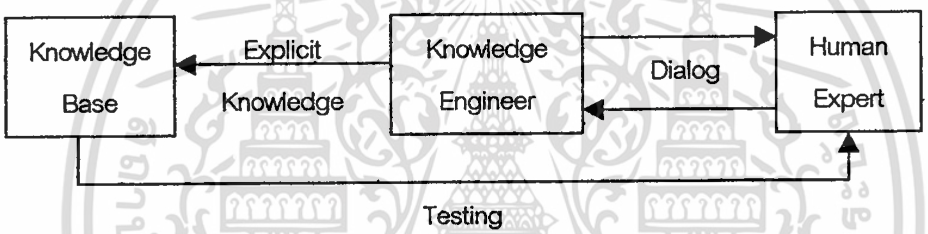
ความรู้(Knowledge)ในระบบผู้เชี่ยวชาญอาจได้มาจากประสบการณ์ หรือหาได้จากหนังสือ แมกกาซีน หรือผู้มีความรู้(ผู้เชี่ยวชาญ) โดยมี **Knowledge Engineer** เป็นผู้สัมภาษณ์หรือดึงเอาความรู้มาจากผู้เชี่ยวชาญ ซึ่งขั้นตอนการนำความรู้มาจากผู้เชี่ยวชาญมาสร้างเป็น knowledge base นั้นใช้เวลานานพอสมควร เมื่อ knowledge engineer สร้าง knowledge base เสร็จแล้วก็อาจมีการนำไปให้ผู้เชี่ยวชาญทดสอบก่อนการใช้งาน ขั้นตอนการพัฒนาดังรูปที่ 2.

หลักการพื้นฐานของระบบผู้เชี่ยวชาญแสดงให้เห็นในรูปที่ 1. ผู้ใช้จะให้ข้อเท็จจริง(Fact)หรือข้อมูล(Information) แก่ระบบผู้เชี่ยวชาญ และผู้ใช้จะได้รับคำแนะนำหรือประสบ

การณ์ตอบกลับมา ภายในระบบผู้เชี่ยวชาญประกอบด้วยสองส่วนหลัก ได้แก่ **Knowledge Base** ซึ่งเก็บความรู้ไว้ และ **Inference Engine** ซึ่งเป็นตัวใช้ความรู้ใน knowledge base และสร้างข้อสรุปเพื่อส่งเป็นคำตอบของระบบผู้เชี่ยวชาญให้แก่ผู้ใช้



รูปที่ 2.1 หลักการทำงานพื้นฐานของ Expert System



รูปที่ 2.2 การพัฒนาระบบผู้เชี่ยวชาญ (Knowledge Engineering)

2.3 คุณลักษณะของระบบผู้เชี่ยวชาญ

- สามารถจำลองการคิดหาเหตุผล(reasoning)ในการแก้ปัญหาของมนุษย์
- Knowledge Base ถูกแยกจากส่วนที่ทำหน้าที่หาเหตุผล(Inference Engine)
- แก้ปัญหาโดยใช้ **Heuristic(ประสบการณ์ที่มี) หรือใช้วิธี Approximate methods**
- ทำงานได้ดี มีประสิทธิภาพสูง ใช้เวลาตอบสนอง(respond time)ไม่นาน มีความน่าเชื่อถือดี
- มีความสามารถอธิบายการแก้ปัญหาได้ว่าสรุปมาได้ยังไง

2.4 ข้อดีของระบบผู้เชี่ยวชาญ

- มีความน่าเชื่อถือมากขึ้น สามารถเรียกใช้งานได้ทุกเวลา
- ลดค่าใช้จ่าย ลดความเสี่ยงที่จะเกิดขึ้น
- ได้คำตอบตรงความต้องการอย่างรวดเร็ว และยังสามารถอธิบายที่มาของข้อสรุปได้
- เพิ่มประสิทธิภาพการทำงานให้ดีขึ้น

- คงที่ ไม่อารมณ์เสีย และให้คำตอบได้ครบถ้วนทุกครั้ง ไม่เหมือนผู้เชี่ยวชาญที่เป็นมนุษย์

Expert Systems	Conventional Programs
ทำงานกับ knowledge	ทำงานกับ data
ใช้ inferential process และ heuristic	ใช้ sequential process และ algorithm
ไม่รับประกันคำตอบที่ได้	ยืนยันคำตอบที่ได้รับ

ตารางที่ 2.1 เปรียบเทียบการทำงานของระบบผู้เชี่ยวชาญกับการเขียนโปรแกรมแบบธรรมดา

2.5 ข้อจำกัดของระบบผู้เชี่ยวชาญ

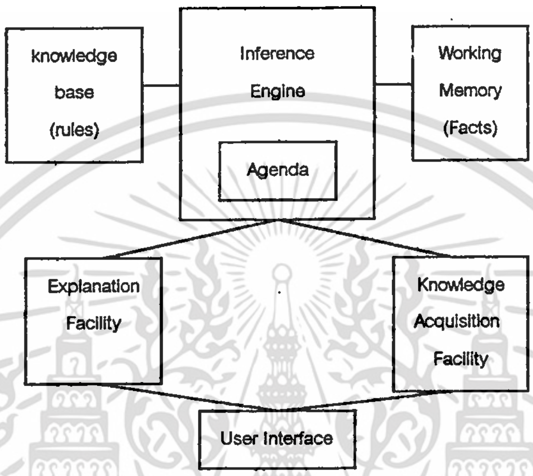
- ไม่สามารถเรียนรู้ได้จากประสบการณ์
- มีข้อจำกัดในการดึงความรู้(Knowledge acquisition)
- เหมาะสมกับการแก้ปัญหาเฉพาะอย่างเท่านั้น
- ข้อมูลที่นำเข้าสู่ระบบต้องเป็นรูปแบบเดียวกันกับ Knowledge base ในปัจจุบัน
- ผู้เชี่ยวชาญที่เป็นมนุษย์ย่อมทำงานได้ยืดหยุ่นกว่าโปรแกรมคอมพิวเตอร์อยู่ดี

2.6 โครงสร้างของระบบผู้เชี่ยวชาญ

- **User Interface** – เป็นกลไกที่ใช้ติดต่อระหว่างระบบผู้เชี่ยวชาญและผู้ใช้
- **Explanation Facility** – ส่งผลลัพธ์จากระบบให้ผู้ใช้
- **Working Memory** – เป็น global database ที่ใช้เก็บข้อเท็จจริงเพื่อให้ rule เรียกใช้
- **Inference Engine** – ตัดสินใจว่า rule ไหนตรงกับข้อเท็จจริงที่ให้มา ทำการ prioritize แก่ rule และทำการ execute หรือ fire ให้ rule ที่มี priority สูงสุด
- **Agenda** – เป็นรายการของ rule ที่ถูกจัด priority ให้แล้วที่สร้างจาก inference engine ซึ่ง pattern ของ rule ก็จะตรงกับข้อเท็จจริงหรือ object ที่อยู่ใน working memory
- **Knowledge Acquisition Facility** – เป็นวิธีที่จะให้ผู้ใช้ใส่ความรู้ให้แก่ระบบแทนที่จะให้ knowledge engineer ทำให้

ในการทำงาน ผู้ใช้จะส่งข้อเท็จจริง(Facts)ผ่าน User Interface เข้าสู่ระบบ โดยผู้ใช้อาจมีการใส่ knowledge ใหม่ๆให้กับระบบโดยผ่านส่วนของ Knowledge Acquisition Facility ซึ่งระบบจะนำไปสร้างเป็น rule ใหม่และเก็บไว้ใน Knowledge Base เมื่อเริ่มทำงาน กฎ(rule)ที่มีเงื่อนไขตรง

กับ fact ที่อยู่ใน Working Memory จะถูกนำมาเก็บไว้ใน Agenda โดยที่ Inference Engine จะทำการเลือก rule ที่มี priority สูงสุดไปใช้งาน(fire) ซึ่ง rule จะถูก fire ได้ทีละ rule เท่านั้น ผลลัพธ์หรือวิธีการแก้ปัญหาที่ได้จะนำไปแสดงผลโดยผ่าน Explanation Facility โดยที่ผู้ใช้สามารถขอคำอธิบายได้ว่าข้อสรุปที่ได้นั้น ได้มาอย่างไร



รูปที่ 2.3 โครงสร้างของระบบผู้เชี่ยวชาญ

บทที่ 3
การพัฒนาระบบผู้เชี่ยวชาญด้วย CLIPS

3.1 CLIPS คืออะไร

CLIPS เป็นเครื่องมือ (tool) ช่วยในการพัฒนาระบบผู้เชี่ยวชาญ (expert system) ถูกพัฒนาขึ้นโดย Software Technology Branch (STB), NASA/Lyndon B. Johnson Space Center คำว่า CLIPS เป็นตัวย่อของ C Language Integrated Production System โดย CLIPS สนับสนุนการเขียนโปรแกรมแบบ rule-based, object-oriented และ procedural ซึ่งตัว CLIPS จะสนับสนุนการทำงานแบบ forward chaining rule โดยเฉพาะ

CLIPS ถูกจัดว่าเป็น **tool** ในการสร้างระบบผู้เชี่ยวชาญเพราะว่ามันมีอุปกรณ์ที่ใช้ในการพัฒนาระบบผู้เชี่ยวชาญอย่างครบถ้วน รวมทั้งส่วนของ editor และ debugging tool คำว่า shell เป็นส่วนของ CLIPS ที่ใช้ในการปฏิบัติการ Inference หรือการเลือกว่าจะใช้ rule ใดปฏิบัติการ ส่วนประกอบเบื้องต้นของ CLIPS shell มี 3 ส่วน ดังนี้

1. **fact list และ instance list** ประกอบด้วยข้อมูลต่างๆ ที่ inference จะนำไปใช้
2. **knowledge base** ประกอบด้วย rule ทั้งหมด
3. **inference engine** ควบคุมการ execution ของ rule ทั้งหมด

โปรแกรมที่เขียนด้วย CLIPS อาจประกอบไปด้วย **rule, fact** และ **object** โดยจะมี inference engine เป็นตัวตัดสินใจว่า rule ใดและเมื่อไหร่ที่ควรถูก execute ระบบผู้เชี่ยวชาญที่เขียนขึ้นโดย CLIPS จะเป็นแบบ data-driven program ซึ่ง fact และ object จะเป็นตัวกระตุ้นให้เกิดการ execution ใน inference engine

ตัวอย่างของ CLIPS ที่ทำให้ต่างไปจากการเขียนโปรแกรมแบบ procedural เช่นพวก Pascal, Ada, BASIC, FORTRAN และ C นั่นคือ ในภาษาแบบ procedural การ execution สามารถทำได้โดยไม่ต้องมีข้อมูล (data) เช่น ในภาษา BASIC สามารถทำงานคำสั่ง PRINT 2+2 ได้เลยซึ่งไม่จำเป็นต้องมี data เข้าไปเกี่ยวข้องด้วย แต่ว่าใน CLIPS จำเป็นต้องมี data เพื่อใช้ในการ execution ของ rule

3.2 วิธีการใช้งานเบื้องต้นของ CLIPS

เมื่อเริ่มเข้าสู่ CLIPS จะปรากฏ prompt ขึ้นดังนี้

CLIPS>

จากจุดนี้ เราสามารถป้อนคำสั่งลงใน CLIPS ซึ่งโหมดที่เราสามารถใส่คำสั่งได้โดยตรงอย่างนี้เรียกว่า **top-level** หรือระดับบนสุด เมื่อเราต้องการออกจาก CLIPS ทำได้โดยการใส่คำสั่ง **exit** โดยพิมพ์ดังนี้

CLIPS> (exit)

แล้วกด enter จะสังเกตได้ว่า การพิมพ์คำสั่งลงใน CLIPS จะอยู่ในวงเล็บ ถ้าเราพิมพ์แต่แค่ exit แล้ว CLIPS จะส่งค่ากลับเป็นตัวอักษร exit ถ้าเราพิมพ์ (exit) จะเป็นการปิดโปรแกรม CLIPS

3.2.1 ข้อเท็จจริง (Fact)

ในการแก้ปัญหา โปรแกรม CLIPS จำเป็นต้องมีข้อมูลเพื่อทำการ reasoning ซึ่งข้อมูลเหล่านี้เราเรียกว่าข้อเท็จจริง (fact) ซึ่งประกอบไปด้วย **relation name** ตามด้วย **slot** (มีตั้งแต่ 0 ขึ้นไป) ตัวอย่างได้แก่

(node (name root) (type decision) (question "เปิดสวิตช์จอภาพและหน่วยประมวลผลกลางแล้วหรือไม่?") (yes-node node1) (no-node node0) (answer nil) (image nil))

ในตัวอย่าง ส่วนที่เป็น relation name คือ node หมายถึงชื่อของ fact ที่ใช้สำหรับอ้างอิง ส่วนของ slot ได้แก่ name, type, question, yes-node, no-node, answer และ image ซึ่งเปรียบเหมือนกับ attribute ใน database นั่นเอง

ก่อนจะสร้าง fact ขึ้นมา CLIPS จะต้องมีการกำหนดรูปแบบของ fact เสียก่อน เราใช้คำสั่ง **deftemplate** ซึ่งมีรูปแบบการใช้งานดังนี้

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

เครื่องหมาย * หมายถึงอาจมีเพิ่มมากกว่า 1 ก็ได้ จากตัวอย่าง person เราอธิบายการกำหนดรูปแบบโดยใช้ deftemplate ได้เป็น

```
(deftemplate node "An example of deftemplate"
        (slot name)      (slot type)
        (slot question)
        (slot yes-node)  (slot no-node)
        (slot answer)    (slot image))
```

ในกรณีข้างบนเรียกว่า *single-field slot* นั่นคือแต่ละ field จะมีค่าได้เพียงหนึ่งเดียว หากมีบางกรณีที่ต้องการมากกว่าหนึ่งค่าจะใช้ **multislot** เช่น มีหลายชื่อเป็นต้น การใช้ multislot เหมือนกันกับการใช้ slot ธรรมดา มีรูปแบบดังนี้

(multislot <slot-name>)

ข้อเท็จจริง (fact) ที่มี relation name และต้องใช้คำสั่ง deftemplate สร้างเหล่านี้เรียกว่า **deftemplate fact** แต่มีข้อเท็จจริงอีกประเภทที่ไม่ต้องสร้างด้วยคำสั่ง deftemplate เราเรียกว่า **ordered fact** ซึ่งประกอบด้วย multi slot เพียงตัวเดียวในการเก็บค่าทั้งหมด ตัวอย่างของ ordered fact นี้ได้แก่ (number-list 7 9 3 4 20) ซึ่งมีความหมายเดียวกับ (deftemplate number-list (multislot values))

การเพิ่ม-ลด Fact

คำสั่งที่ใช้ในการเพิ่ม fact เข้าไปคือ **assert** รูปแบบการใช้งานของคำสั่งนี้คือ

(assert <fact>+)

คำสั่งนี้สามารถใส่ fact ได้มากกว่าหนึ่งเข้าไปในคราวเดียวกัน ตัวอย่างการใช้งานดังนี้

CLIPS> (assert (node (name node1) (type decision) (question "หลอด LED ด้านล่างของจอภาพติดหรือไม่?") (yes-node node3) (no-node node2) (answer nil) (image nil))

<Fact-0>

CLIPS>

เราสามารถใช้คำสั่ง **facts** เพื่อแสดงรายการของ fact ขึ้นมาบนจอได้ดังนี้

CLIPS> (facts)

f-0 (node (name root) (type decision) (question "เปิดสวิตซ์จอภาพและหน่วยประมวลผลกลางแล้วหรือไม่?") (yes-node node1) (no-node node0) (answer nil) (image nil))

f-1 (node (name node1) (type decision) (question "หลอด LED ด้านล่างของจอภาพติดหรือไม่?") (yes-node node3) (no-node node2) (answer nil) (image nil))

For a total of 2 facts.

CLIPS>

คำว่า "f-0" เป็น **fact identifier** ที่ถูกกำหนดให้โดย CLIPS แต่ละ fact จะมีไม่ซ้ำกัน ขึ้นต้นด้วยตัวอักษร f แล้วตามด้วยตัวเลขซึ่งเรียกว่า **fact index** ซึ่งเริ่มจาก 0 อีกคำสั่งหนึ่งที่ใช้ในการลบ fact ออกจากรายการ เราใช้คำสั่ง **retract** ซึ่งมีรูปแบบการใช้งานดังนี้

(retract <fact-index> +)

ซึ่งสามารถใช้ลบ fact ออกได้มากกว่าหนึ่ง ตัวอย่างการใช้งานได้แก่

CLIPS> (retract 0)

การแก้ไข(Modifying)และการทำซ้ำ(Duplicating)ข้อเท็จจริง

ค่าใน slot ของข้อเท็จจริงสามารถนำมาแก้ไขได้โดยใช้คำสั่ง **modify** ซึ่งมีรูปแบบการใช้งานดังนี้

(modify <fact-index> <slot-modifier>+) ซึ่งตัว slot-modifier คือ (<slot-name> <slot-value>)

จากตัวอย่างข้อเท็จจริงข้างต้น เราจะเปลี่ยนคำถามใน Root node ได้ดังตัวอย่าง

CLIPS> (modify 0

(question "จอภาพแสดงภาพหรือตัวอักษรเมื่อเปิดสวิตซ์คอมพิวเตอร์หรือไม่?"))

<Fact-2>

การทำงานของคำสั่ง modify คือจะทำการลบ (retract) ข้อเท็จจริงเดิม แล้วจึงใส่ (assert) ข้อเท็จจริงใหม่ที่แก้ไขค่าเข้าไป ดังนั้น fact index ตัวใหม่จึงถูกสร้างขึ้นใหม่สำหรับข้อเท็จจริงใหม่ที่ใส่เข้าไป คำสั่ง **duplicate** ก็ทำงานเช่นเดียวกันกับ modify แต่จะไม่ทำการลบข้อเท็จจริงเดิม คือจะใส่ข้อเท็จจริงใหม่เข้าไปที่เหมือนข้อเท็จจริงเดิมทุกอย่าง ยกเว้นบางส่วนที่แก้ไข ตัวอย่างการใช้คำสั่งนี้ได้แก่

CLIPS> (duplicate 1 (name node2))

<Fact-2>

CLIPS> (facts)

f-0 (node (name root) (type decision) (question "เปิดสวิตซ์จอภาพและหน่วยประมวลผลกลางแล้วหรือไม่?") (yes-node node1) (no-node node0) (answer nil) (image nil))

f-1 (node (name node1) (type decision) (question "หลอด LED ด้านล่างของจอภาพติดหรือไม่?") (yes-node node3) (no-node node2) (answer nil) (image nil))

f-2 (node (name node2) (type decision) (question "หลอด LED ด้านล่างของจอภาพติดหรือไม่?") (yes-node node3) (no-node node2) (answer nil) (image nil))

For a total of 2 facts.

CLIPS>

คำสั่ง Watch

เป็นคำสั่งที่มีประโยชน์ในการ debugging โปรแกรม โดยมีรูปแบบการใช้งานดังนี้

(watch <watch-item>)

ซึ่ง watch-item จะเป็นได้ทั้ง *facts, rules, activations, statistics, complication, focus* หรือ *all* ตัวอย่างการใช้งานคำสั่งนี้ได้แก่

CLIPS> (facts 2 2)

f-2 (node (name node2) (type decision) (question "หลอด LED ด้านล่างของจอภาพติดหรือไม่?") (yes-node node3) (no-node node2) (answer nil) (image nil))

For a total of 1 fact.

CLIPS> (watch facts)

CLIPS> (modify 2 (question "จอภาพแสดงภาพนิ่ง ภาพไม่สั่นใช่หรือไม่?"))

<== f-2 (node (name node2) (type decision) (question "หลอด LED ด้านล่างของจอภาพติดหรือไม่?") (yes-node node3) (no-node node2) (answer nil) (image nil))

==> f-3 (node (name node2) (type decision) (question "จอภาพแสดงภาพนิ่ง ภาพไม่สั่นใช่หรือไม่?") (yes-node node3) (no-node node2) (answer nil) (image nil))

<Fact-3>

CLIPS>

เครื่องหมาย <== บ่งบอกว่าข้อเท็จจริงนี้กำลังถูกลบ (retract) เครื่องหมาย ==> บอกว่าข้อเท็จจริงนั้นกำลังถูกใส่เข้าไป (assert) เมื่อต้องการยกเลิกคำสั่ง watch จะใช้คำสั่ง **unwatch** โดยมีรูปแบบการใช้งานดังนี้

(unwatch <watch-item>)

ใน CLIPS การสร้าง **comment** สามารถทำได้โดยการใส่เครื่องหมาย ; นำหน้าข้อความนั้น อีกประการที่ควรระวังคือ CLIPS เป็น **case sensitive** หมายความว่าตัวอักษรใหญ่-เล็ก มีผลต่างกัน

คำสั่ง Deffacts

จะเป็นการสะดวกถ้าเราสามารถใส่กลุ่มของข้อเท็จจริงได้อัตโนมัติแทนที่จะพิมพ์คำสั่ง assert หลายๆครั้งใน top level ซึ่งกลุ่มของข้อเท็จจริงเหล่านี้สามารถถูกกำหนด (initial) ก่อนการรันโปรแกรม เรากำหนดได้โดยใช้คำสั่ง **deffacts** ต่อไปนี้เป็นตัวอย่างการใช้ deffacts ในการ initial ข้อมูลของกลุ่มคนบางคนที่เรารู้จักดีอยู่แล้ว

(deffacts node "Some nodes"

(node (name root) (type decision) (question "เปิดสวิตซ์จอภาพและหน่วยประมวลผลกลางแล้วหรือไม่?") (yes-node node1) (no-node node0) (answer nil) (image nil))

(node (name node1) (type decision) (question "หลอด LED ด้านล่างของจอภาพติดหรือไม่?") (yes-node node3) (no-node node2) (answer nil) (image nil)) (node (name node2) (type answer) (question nil) (yes-node nil) (no-node nil) (answer "1. ตรวจสอบว่าสายไฟสำหรับเสียบจอภาพเสียบดีพอหรือไม่ 2. ตรวจสอบว่ามีกระแสไฟฟ้า 220 โวลต์จ่ายให้กับจอภาพหรือไม่") (image "back_monitor.jpg"))

รูปแบบการใช้งานทั่วไปของ deffacts คือ

(deffacts <deffacts name> [<optional comment>]

<facts> *)

ข้อเท็จจริงใน deffacts จะถูก assert โดยใช้คำสั่ง **reset** ซึ่งคำสั่งนี้จะลบข้อเท็จจริงทั้งหมดออกจาก fact list แล้วจึงใส่ข้อเท็จจริงที่อยู่ใน deffacts โดยจะมีข้อเท็จจริงใหม่อันหนึ่งที่ใส่เข้ามาให้โดยอัตโนมัติโดยคำสั่ง reset เราเรียกว่า **initial-fact** ดังตัวอย่างข้างล่างนี้

CLIPS> (reset)

CLIPS> (facts)

f-0 (initial-fact)

f-1 (node (name root) (type decision) (question "เปิดสวิตซ์จอภาพและหน่วยประมวลผลกลางแล้วหรือไม่?") (yes-node node1) (no-node node0) (answer nil) (image nil))

f-2 (node (name node1) (type decision) (question "หลอด LED ด้านล่างของจอภาพติดหรือไม่?") (yes-node node3) (no-node node2) (answer nil) (image nil))

f-3 (node (name node2) (type answer) (question nil) (yes-node nil) (no-node nil) (answer "1. ตรวจสอบว่าสายไฟสำหรับเสียบจอภาพเสียบดีพอหรือไม่ 2. ตรวจสอบว่ามีกระแสไฟฟ้า 220 โวลต์จ่ายให้กับจอภาพหรือไม่") (image "back_monitor.jpg"))

For a total of 4 facts.

CLIPS>

3.2.2 กฎเกณฑ์ (Rule)

ในการทำงาน เราจำเป็นต้องมี rule และ fact ซึ่ง rule เปรียบเหมือนกับ IF THEN ในภาษา Ada, C หรือ Pascal ในการเขียน rule ที่ดีเราควรสร้าง **pseudocode** เสียก่อน pseudocode นี้ไม่สามารถนำไป execute ได้ในโปรแกรมจริง แต่จะช่วยให้เขียนโปรแกรมได้ดีขึ้น ตัวอย่างต่อไปนี้เป็น pseudocode ของ rule ที่เกี่ยวกับการวินิจฉัยฮาร์ดดิสก์ว่าเสียหรือไม่

IF ฮาร์ดดิสก์มีเสียงดังแปลกๆ

THEN คอมพิวเตอร์ใกล้เสีย

การสร้าง rule ทำได้โดยใช้คำสั่ง **defrule** ซึ่งมีรูปแบบการใช้งานดังนี้

```
(defrule <rule-name> [<comment>]      ; Rule Header
        <patterns>*                   ;Left-Hand Side(LHS)ของ rule
=>                                    ; THEN arrow
        <patterns>*                   ; Right-Hand Side (RHS) ของ rule (เป็น action)
```

ตัวอย่างการสร้าง rule ที่ได้จาก pseudocode ข้างต้นเป็นดังนี้

```
CLIPS> (clear)
CLIPS> (assert (harddisk (sound loud)))
<Fact-0>
CLIPS> (defrule harddisk
        (harddisk (sound loud))
=>
        (assert (computer (infected yes))))
CLIPS>
```

เราต้องทำการกำหนดโครงสร้างของข้อเท็จจริง harddisk และ computer มาก่อนแล้ว โดยใช้คำสั่ง deftemplate ที่เอ่ยถึงตอนต้น คำสั่ง **clear** เป็นการสั่งให้ CLIPS ลบคำสั่งเดิมและค่าเดิมออกให้หมดจาก memory จากนั้นเราใส่ข้อเท็จจริง harddisk มี sound เป็น loud เข้าไป

CLIPS จะทำการเปรียบเทียบ pattern ของ rule กับ fact ที่อยู่ใน fact list ถ้าทุก pattern ของ rule ตรงกันกับ fact แล้ว rule นั้นจะถูก **activated** และจะถูกส่งไปยัง **agenda** ซึ่งเป็นที่เก็บกลุ่มของ rule ที่ activated อาจจะมีหรือไม่มี rule ใน agenda ก็ได้

ในส่วนสุดท้ายของ rule จะเป็น pattern ที่จะบอกว่า rule นี้จะมี action อย่างไรเมื่อ rule ถูก **fire** ซึ่ง fire มีความหมายว่า CLIPS ได้ execute การกระทำ(action)ของ rule จาก agenda โดยปกติโปรแกรมจะหยุดการ execute เมื่อไม่มี rule เหลืออยู่ใน agenda เมื่อมีหลายๆ rule ใน agenda ตัว CLIPS จะเป็นผู้กำหนดว่า rule ไหนสมควรจะถูก fire ใน CLIPS จะเรียง rule ใน agenda โดยใช้ priority เมื่อ rule อยู่นานๆ จะมีการเพิ่มค่า priority ให้ และ rule ที่มีค่า priority สูงสุดจะถูก fire ค่า priority ของ rule เป็นตัวเลข integer ที่เรียกว่า **salience**

3.2.3 Agenda และการ Execution

ใน CLIPS สามารถรันโปรแกรมด้วยคำสั่ง **run** ซึ่งมีรูปแบบการใช้งานดังนี้

```
(run [<limit>] )
```

ส่วน limit เป็นส่วนที่จะมีหรือไม่ก็ได้ limit หมายถึงจำนวน rule สูงสุดที่กำหนดให้ fire ได้ ถ้าไม่ใส่เข้าไปจะถือว่าให้ fire rule ทั้งหมดใน agenda ส่วนคำสั่งที่จะแสดง rule ที่มีใน agenda เราจะพิมพ์ **agenda** มีรูปแบบการใช้งานคือ (agenda) ตัวอย่างในการใช้งานของคำสั่ง run และ agenda ได้แก่

CLIPS> (agenda)

0 hardisk: f-0

For a total of 1 activation.

CLIPS>

ตัวเลข 0 เป็น salience ของ "harddisk" activation และ f-0 ก็เป็นตัว fact identifier ของข้อเท็จจริง (harddisk (sound loud)) ซึ่ง match กับ activation ค่าของ salience ตามปกติจะตั้งไว้เป็น 0 แต่จะมีค่าที่เป็นไปได้ตั้งแต่ –10,000 ถึง 10,000

CLIPS> (run)

CLIPS> (facts)

f-0 (harddisk (sound loud))

f-1 (computer (infected yes))

For a total of 2 facts.

CLIPS>

จากตัวอย่างข้างบน เมื่อเรา ใช้คำสั่ง run แล้ว จาก rule ที่เราสร้างไว้ข้างต้นก่อนหน้านี้จะทำการเพิ่มข้อเท็จจริง (computer (infected yes)) ดังนี้เมื่อทำการรันแล้ว เราจึงได้ข้อเท็จจริงเพิ่มขึ้นใหม่อีกหนึ่งตัว

มีคำสั่งที่ใช้ในการแสดงรายการของ rule และ fact ที่เราได้สร้างเอาไว้แล้ว เราใช้คำสั่ง **list-defrules** เพื่อดูรายชื่อ rule ที่สร้างไว้แล้ว ในทำนองเดียวกัน **list-deftemplates** และ **list-deffacts** ใช้แสดงรายชื่อ deftemplate และรายชื่อ deffact โดยมีรูปแบบการใช้งานดังนี้ตามลำดับ (list-defrules), (list-deftemplates)และ (list-deffacts) ตัวอย่างการใช้คำสั่ง list-defrules มีดังนี้

CLIPS> (list-defrules)

harddisk

For a total of 1 rule.

CLIPS>

คำสั่งในการแสดงรายละเอียดของ defrule, deftemplate และ deffacts คือ **ppdefrule, ppdeftemplate** และ **ppdeffacts** ตามลำดับ มีรูปแบบการใช้งานดังนี้

(ppdefrule <defrule-name>)

(ppdeftemplate <deftemplate-name>)

(ppdeffacts <deffacts-name>)

ตัวอย่างการใช้งานได้แก่

```
CLIPS> (ppdefrule  harddisk )
(defrule  MAIN :: harddisk
        (harddisk (sound loud))
=>
        (assert  (computer (infected yes))))
CLIPS>
```

คำสั่งที่ใช้ในการลบ defrule, deftemplate และ deffacts คือ **undefrule, undeftemplate** และ **undeffacts** โดยมีรูปแบบการใช้งานดังนี้

(undefrule <defrule-name>)

(undeftemplate <deftemplate-name>)

(undeffacts <deffacts-name>)

คำสั่งพิมพ์ออกหน้าจอ (Printout)

แทนที่จะทำการใส่ข้อเท็จจริงเพิ่มเข้าไป เราสามารถให้พิมพ์ออกทางหน้าจอโดยใช้คำสั่ง **printout** โดยมีตัวอย่างการใช้งานดังนี้

```
CLIPS> (defrule  harddisk
        (harddisk (sound loud))
=>
        (printout  t  "Your computer is infected"  crlf) )
CLIPS> (run)
Your computer is infected
CLIPS>
```

ตัวอักษร **"t"** ที่ตามหลังคำสั่ง printout เป็นการบอก CLIPS ว่าจะแสดงผลออกทาง **standard output device** ของเครื่องคอมพิวเตอร์ ซึ่งโดยปกติคือหน้าจอ แต่เราก็สามารถสั่งให้ออกไปที่ device ตัวอื่นเช่น disk และ modem ก็ได้ ส่วนอักษร **crlf** มาจาก carriage return/line feed ซึ่งจะบอกให้ CLIPS ขึ้นบรรทัดใหม่ให้

คำสั่งบันทึกและเรียกดูข้อมูล (Loading and Saving)

คำสั่ง **load** ใช้ในการเรียกดูข้อมูลเดิม มีรูปแบบการใช้งานดังนี้

(load <file-name>)

ตัวอย่างการใช้งานได้แก่การพิมพ์ (load "A:harddisk.clp") แต่ในบางกรณีที่ต้องมีการใส่ path จึงต้องมีเครื่องหมาย \ เข้าเกี่ยวข้องด้วย แต่ใน CLIPS จะถือว่าเครื่องหมาย \ เป็นสัญลักษณ์อย่างหนึ่ง ดังนั้นเวลาเราใส่ path จึงต้องเขียนเครื่องหมาย \ สองครั้งดังตัวอย่าง (load "A:\\usr\\clips\\harddisk.clp")

คำสั่ง **save** ใช้สำหรับบันทึกข้อมูล มีรูปแบบการใช้งานดังนี้

(save <filename>)

3.2.4 ตัวแปร (Variable) และ Wildcard

ตัวแปรใน CLIPS ใช้ในการเก็บค่า เราเขียนเครื่องหมาย "?" หน้าตัวอักษรเพื่อบอกว่าสัญลักษณ์นี้แทนตัวแปร โดยมากมักเขียนชื่อตัวแปรให้สอดคล้องกับการใช้งาน ตัวอย่างตัวแปรได้แก่ ?speed, ?name, ?value หรือ ?color การให้ค่าตัวแปรเราเรียกว่าการ **bind** ตัวอย่างในการใช้ตัวแปรใน rule ได้แก่

```
CLIPS> (clear)
CLIPS>
(defrule ask-decision-node-question
        (node (name ?name)
        (type decision)
        (question ?question))
 =>
        (printout t ?question crlf))
CLIPS>
(deffact node
        (node (name node1) (type decision) (question "หลอด LED ด้านล่างของจอภาพติดหรือไม่?") (yes-node node3) (no-node node2) (answer nil) (image nil))
        (node (name node2) (type answer) (question nil) (yes-node nil) (no-node nil) (answer "1. ตรวจสอบว่าสายไฟสำหรับเสียบจอภาพเสียบดีพอหรือไม่ 2. ตรวจสอบว่ามีกระแสไฟฟ้า 220 โวลต์จ่ายให้กับจอภาพหรือไม่") (image "back_monitor.jpg"))
```

CLIPS> (reset)

CLIPS> (run)

หลอด LED ด้านล่างของจอภาพติดหรือไม่?

CLIPS>

Rule นี้จะทำงานเมื่อมี node ที่มี slot type เป็น decision และจะนำค่าใน slot question ใส่ตัวแปร ?question และนำมาแสดงผลบนหน้าจอ ดังนั้นจึงมี fact เดียวที่ match กับ rule นี้คือ node1 เพราะ node2 มี slot type เป็น answer ดังนั้น rule จึงแสดงค่าของ slot question ของ node1 บนหน้าจอ

Fact Address

เราใช้ fact address ในการให้ค่าตัวแปร ข้อแตกต่างในการใช้ตัวแปรเก็บค่ากับการใช้ fact address เราใช้ตัวอย่างในการอธิบาย (answer ~yes&~no) ตัวแปร ?answer จะเก็บค่าของ field นั้นของ fact ถ้าเราต้องการลบ fact answerที่มีค่าอื่นนอกจาก yes และ no เราต้องบอกค่า *address* ของ fact ที่ต้องการลบ เราใช้เครื่องหมาย left arrow “<-“ ในการประกาศ fact address เรามาดูตัวอย่างกัน

```
CLIPS> (clear)
CLIPS> (defrule bad-answer
  ?answer <- (answer ~yes&~no)
  =>
  (retract ?answer))
CLIPS> (deffacts answer
        (answer  yes)
        (answer  no)
        (answer  unknow))
CLIPS> (reset)
CLIPS> (run)
CLIPS> (facts)
f-0     (answer yes)
f-1     (answer no)
For a total of 2 facts.
CLIPS>
```

Wildcard

เราใช้ **wildcard** ในการละ field ที่เราไม่จำเป็นต้องใส่ค่าให้ได้ การใช้ wildcard ทำได้โดยใส่เครื่องหมาย "?" เฉยๆ ไว้ตรง field นั้นๆ สมมติว่าเราต้องการหาคนชื่อหนึ่งโดยไม่สนใจว่านามสกุลจะเป็นเช่นไร เราใช้ wildcard ช่วยได้ ดังตัวอย่าง

```
CLIPS> (clear)
CLIPS> (defrule ask-decision-node-question
        ?node <- (current-node ?name)
        (node (name ?name)
        (type decision)
        (question ?question))
        (not (answer ?))
  =>
        (bind ?*the_question* ?question))
```

ในตัวอย่างนี้เราถือว่า answer จะมีค่าเป็นอะไรไม่สำคัญ ดังนั้น rule จะ fire เฉพาะข้อเท็จจริงที่มีค่าของ type เป็น decision เท่านั้น การใช้ wildcard เช่นนี้เป็น **single-field wildcard** เพราะเครื่องหมาย ? หนึ่งตัวจะแทน filed เพียงแค่ 1 ตัวเท่านั้น ไม่ขาดไม่เกิน เราอาจใช้ **multifield wildcard** ซึ่งแทนด้วย "$?" แทนจำนวน field ที่อาจมีหรือไม่มีก็ได้ ในกรณีที่มีค่าหลายค่า

3.2.5 Field Constant

สมมติเราต้องการหา answer ไหนก็ได้ที่มีค่าอะไรก็ได้ที่ไม่ใช่ yes เราใช้ **not constraint** เข้าช่วย โดยใส่เครื่องหมาย "~" นำหน้า ดังตัวอย่าง

```
(defrule bad-answer
  ?answer <- (answer ~yes)
  =>
  (retract ?answer))
```

ถ้าเราต้องการหา answer ไหนก็ได้ที่มีค่า yes หรือ no เราใช้ **or constraint** แทนด้วย "|" ดังตัวอย่าง

```
(defrule bad-answer
  ?answer <- (answer yes | no)
  =>
```

(retract ?answer))

ในการใช้ **and constraint** เราแทนด้วยเครื่องหมาย "&" เราใช้ร่วมกับ constraint ตัวอื่นด้วยเท่านั้น สมมติว่าเราต้องการนำค่าชื่อของผู้มีผมสีน้ำตาลหรือสีดำใส่ลงตัวแปร เราใช้ and constraint ดังนี้

```
(defrule bad-answer
  ?answer <- (answer ~yes&~no)
  =>
  (retract ?answer))
```

3.2.6 เครื่องหมายทางคณิตศาสตร์

ใน CLIPS การเขียนสมการจะเขียนเครื่องหมายไว้ทางซ้ายหรือข้างหน้า เป็น **prefix form** เช่น (+ 2 3) หมายถึง 2+3 เราสามารถเขียนได้ที่ top level เลยดังตัวอย่าง

```
CLIPS> (+ 2 2)
4
CLIPS>
```

ถ้ามีทศนิยม ผลลัพธ์จะมีจำนวนทศนิยมเท่ากับจำนวนทศนิยมที่มากที่สุดของตัวเลขที่ใช้คำนวณ

Arithmetic Operators	Meaning
+	บวก
-	ลบ
*	คูณ
/	หาร

ตารางที่ 3.1 เครื่องหมายทางคณิตศาสตร์ที่ใช้ใน CLIPS

3.2.7 Function การทำงานบางตัวของ CLIPS

Read Function

Expert system มักต้องการ input จากผู้ใช้ CLIPS มีคำสั่ง **read** ไว้รับค่าจากคีย์บอร์ด คำสั่ง read จะรับค่าได้ 1 field ซึ่งรวมทั้งตัวเลข ทศนิยม ตัวอักษร หรือสัญลักษณ์ได้ ตัวอย่างการทำงานได้แก่

```
CLIPS> (clear)
CLIPS> (defrule  get-answer
=>
        (printout  t  "จอภาพแสดงภาพนิ่ง ภาพไม่สั่นใช่หรือไม่?" crlf)
        (bind  ?response  (read))
        (assert  (answer ?response) ) )
CLIPS> (reset)
CLIPS> (run)
จอภาพแสดงภาพนิ่ง ภาพไม่สั่นใช่หรือไม่?
yes
CLIPS> (facts)
f-0      (initial-fact)
f-1      (answer  yes)
For a total of 2 facts.
CLIPS>
```

คำสั่ง **bind** เป็นคำสั่งที่ใช้ในการใส่ค่าให้กับตัวแปร มีรูปแบบการใช้งานคือ

(bind <variable> <value>)

Open Function

ในการทำงาน เราอาจมีการอ่านหรือเขียนลงไฟล์ข้อมูล ก่อนจะมีการอ่านหรือเขียนข้อมูลบนไฟล์ จำเป็นต้องใช้คำสั่ง **open** เพื่อเปิดไฟล์นั้นเสียก่อน รูปแบบการใช้งานมีดังนี้

(open <file-name> <file-ID> [<file-access>])

ตัวอย่างเช่น

(open "input.dat" data "r")

หลังคำสั่ง open ตัวแรกจะบอกชื่อไฟล์ที่จะเปิด ตัวที่สองเป็น **logical name** คือจะแทนชื่อไฟล์ที่จะเปิด ตัวสุดท้ายเป็นการบอกว่าจะเข้าถึงไฟล์แบบไหน โดยตารางข้างล่างจะบอกรายละเอียดชนิดการเข้าถึงไฟล์

Mode	Action
“r”	อ่านเท่านั้น
“w”	เขียนเท่านั้น
“r+”	อ่านและเขียน
“a”	เขียนต่อท้าย

ตารางที่ 3.2 Mode การทำงานการเข้าถึงไฟล์

เมื่อเราเปิดไฟล์แล้ว เราสามารถใช้คำสั่ง read ในการอ่านไฟล์นั้นได้ด้วย แต่เป็นการอ่านทีละ field โดยมีรูปแบบการใช้งานคือ (read [<logical-name>])

Close Function

เมื่อมีการเปิดใช้ไฟล์ก็จำเป็นต้องมีการปิดไฟล์เมื่อเลิกใช้งาน เราใช้คำสั่ง **close** โดยมีรูปแบบการใช้งานดังนี้

(close [file-ID])

โดยที่ file-ID เป็น logical name ของไฟล์ที่ต้องการปิด จากตัวอย่างการเปิดไฟล์ input.dat ซึ่งมี file-ID ชื่อ data ถ้าเราต้องการปิดไฟล์นี้ก็ใช้คำสั่ง (close data)

Readline Function

ใช้อ่าน input ทั้งบรรทัด มีรูปแบบการใช้งานคือ (readline [<logical-name>]) เรามาดูตัวอย่าง

```
CLIPS> (clear)
CLIPS> (defrule get-answer
=>
        (printout t "ฮาร์ดดิสก์มีอาการผิดปกติอย่างไร?" crlf)
        (bind ?response (readline))
        (assert (answer ?response) ) )
CLIPS> (reset)
CLIPS> (run)
ฮาร์ดดิสก์มีอาการผิดปกติอย่างไร?
เสียงดัง อ่านช้า ไฟค้าง
```

CLIPS> (facts)

f-0 (initial-fact)

f-1 (answer "เสียงดัง อ่านช้า ไฟค้าง")

For a total of 2 facts.

ในตัวอย่างนี้ "เสียงดัง อ่านช้า ไฟค้าง" ถูกใส่ลงใน single field ซึ่งเวลาอ่านออกมาแล้วจะไม่แสดงทั้งชื่อและสกุลออกมาพร้อมกัน เราใช้คำสั่ง **explode$** ในการรับค่าสริงตัวหนึ่งแล้วกระจายเป็น multifield value

```
CLIPS> (clear)
CLIPS> (defrule get-answer
=>
        (printout t "ฮาร์ดดิสก์มีอาการผิดปกติอย่างไร?" crlf)
        (bind ?response (explode$ (readline)))
        (assert (answer ?response) ) )
CLIPS> (reset)
CLIPS> (run)
ฮาร์ดดิสก์มีอาการผิดปกติอย่างไร?
เสียงดัง อ่านช้า ไฟค้าง
CLIPS> (facts)
f-0     (initial-fact)
f-1     (user's-name เสียงดัง อ่านช้า ไฟค้าง)
For a total of 2 facts.
```

3.3 ระบบฐานความรู้(Knowledge Base System)

3.3.1 Knowledge Representation

ในโครงการนี้เราใช้ Rule-based System ในการแสดงความรู้(knowledge representation) วิธีการนี้เป็นวิธีที่ใช้กันมากในการสร้างระบบผู้เชี่ยวชาญ มักเขียนในรูปแบบของ IF-THEN rules ซึ่งคล้ายกับการเขียน condition statement ในการเขียนโปรแกรมภาษาชั้นสูง

คุณสมบัติทั่วไปของ Rule

- Name of the rule – มีชื่อของ rule
- Premise – เป็นส่วนเงื่อนไข หรืออาจเรียกว่าส่วน Left-Hand Side (LHS)

- Conclusion – เป็นส่วนของข้อสรุปของ rule หรือเรียกว่าเป็น Right-Hand Side (RHS) โดยที่ข้อสรุปของ rule หนึ่งอาจเป็นเงื่อนไขของอีก rule หนึ่งก็ได้
- Note – เป็นส่วนอธิบายเพิ่มเติม (comment)
- Reference – อ้างอิงว่า rule ข้อนี้มาจากไหน มี pattern ที่ตรงกับ fact ใดบ้าง
- Confidence Factor – หรือค่า CF เป็นค่าที่ใช้วัดความมั่นใจของ rule ว่ามีความถูกต้องแค่มากน้อยเท่าไร เช่น 0-10, 0-100 หรืออาจเป็น 0 กับ 100% ในกรณีที่คำตอบเป็นใช่ และไม่ใช่
- Priority – rule ที่มีค่า priority มากที่สุดและมีค่า cost น้อยที่สุด จะเป็น rule ที่ถูกนำไปทำงาน
- Cost – เป็นค่าใช้จ่ายของ rule
- Chaining preference – เป็น forward chaining หรือ backward chaining
- Status – ค่าสถานะของ rule

Forward Chaining และ Backward Chaining

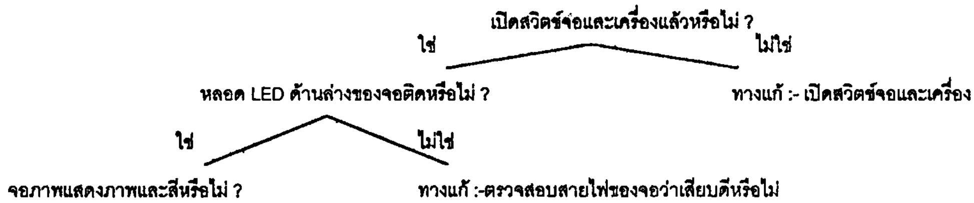
Forward chaining เป็นการทำงานหาเหตุผลโดยเริ่มจากข้อเท็จจริง (fact) ไปสู่ข้อสรุป (conclusion) ส่วน Backward chaining เป็นการ reasoning โดยการตั้งข้อสรุปหรือสมมติฐานแล้วหาว่ามีข้อเท็จจริงใดที่ตรงกับข้อสรุปนั้นบ้าง และมีเงื่อนไขอะไรบ้างที่ต้องเป็นจริง

Forward Chaining	Backward Chaining
การวางแผน การตรวจสอบ การควบคุม	การวินิจฉัย
จากเงื่อนไขไปสู่ข้อสรุป	จากข้อสรุปไปสู่เงื่อนไข
Data-driven, top-down reasoning	Goal-driven, bottom-up reasoning
ทำงานไปข้างหน้าเพื่อหา solution จากข้อเท็จจริงที่มีอยู่	ทำงานย้อนกลับเพื่อหาข้อเท็จจริงที่สนับสนุนสมมติฐาน

ตารางที่ 4.1 เปรียบเทียบ Forward chaining และ backward chaining

3.3.2 Inference หรือ Reasoning

Inference หรือ reasoning เป็นการหาข้อสรุป มีวิธีการหาข้อสรุปได้หลายวิธี ในโครงการนี้เราใช้วิธีการสร้าง Decision Tree เพื่อช่วยในการหาข้อสรุป decision tree ที่ใช้เป็นประเภท Binary Tree คือเป็น tree ที่มี child node เพียง 2 node



รูปที่ 4.1 ตัวอย่าง Decision Tree ประเภท Binary Tree

ปัญหาที่เหมาะกับ Decision tree มีสองชนิด ชนิดแรก ปัญหาต้องมีจำนวนคำตอบในขอบเขตจำกัด เช่น ปัญหาเกี่ยวกับภาษีหรือการวินิจฉัยโรค เพราะฉะนั้น decision tree จึงไม่เหมาะกับงานจัดตารางเวลา หรืองานวางแผน เนื่องจากคำตอบต้องมีเตรียมไว้ก่อน

Decision Tree ประกอบด้วย Node และ Branch แต่ละ node แทนตำแหน่งใน tree ซึ่งอาจเป็นได้ทั้ง decision node หรือ answer node ส่วน branch หรือกิ่งนั้นแทนการติดต่อกันระหว่าง node ส่วน node ที่อยู่บนสุดไม่มี parent เรียกว่า root ส่วน node ที่อยู่ล่างสุดไม่มี child เรียกว่า leave ซึ่งมักใช้แทน solution ของปัญหา

เมื่อขอบเขตของปัญหากว้างขึ้น Decision tree ก็ต้องมีการเรียนรู้ด้วย วิธีการง่ายๆคือให้ผู้ใช้กำหนดคำถามใหม่และคำตอบเข้าไป คำถามจะแทน decision node ส่วนคำตอบจะแทน answer node ของคำถามนั้น ๆ

ในการสร้าง Rule-base decision tree เนื่องจาก tree ต้องมีการเรียนรู้จึงเหมาะสมกว่าที่จะแทน decision tree ด้วย fact มากกว่า rule เพราะ fact สามารถเปลี่ยนแปลงได้ง่ายกว่า ดังนั้นแต่ละ node จะแทน fact หนึ่งตัว แต่ว่า node มีทั้งเป็น decision และ answer node ดังนั้น template จึงต้องต่างกัน การสร้าง rule ก็สร้างให้มีหน้าที่ initialize หรือ load fact เข้ามาในตอนแรก มีการกำหนด node ปัจจุบันอยู่ที่ root เมื่อทำงานต่อไป node ปัจจุบันจะเคลื่อนที่ลงมาตาม tree แล้วก็มี rule กำหนดให้ทำงานตาม node ลงมาเรื่อยๆ ถ้าปัจจุบันเป็น decision node ก็จะถามคำถามผู้ใช้ แล้วรับคำตอบ ไปเรื่อยๆจนถึง answer node จึงแสดงผลลัพธ์ให้ผู้ใช้ และมี rule ในการถามย้ำว่าถูกหรือไม่ ถ้าไม่คำตอบที่จริงควรเป็นเช่นไร มีคำถามเช่นไร รับคำตอบและคำถามจากผู้ใช้แล้วสร้าง fact(node) ขึ้นใหม่ใน decision tree

3.3.3 Knowledge Acquisition

Knowledge Acquisition หมายถึงกระบวนการเพื่อให้ได้มาซึ่งความรู้ของผู้เชี่ยวชาญ กระบวนการนี้มีอยู่ 2 ประเภท

1. **Manual** เป็นวิธีการนำความรู้มาจากผู้เชี่ยวชาญโดยตรง เช่น การสัมภาษณ์หรือการค้นคว้าตำราต่างๆ
2. **Automated** เป็นการใช้ข้อมูลเก่าๆใน history record แล้วทำการหาความรู้(knowledge)ที่มีอยู่ใน record เหล่านั้นเพื่อสร้าง decision tree ขึ้นมาหรือเรียกว่า rule induction

ในโครงการนี้ การทำ Knowledge Acquisition ทำโดยการค้นคว้าหาความรู้ตามเว็บไซต์ที่เกี่ยวข้องกับเครื่องคอมพิวเตอร์หรือการซ่อมคอมพิวเตอร์บน internet โดยส่วนใหญ่จะใช้วิธีการเข้าไปในส่วนที่เป็นการถามตอบปัญหาคอมพิวเตอร์ที่พบเป็นประจำ(Frequency Ask Question หรือ FAQ) รวบรวมคำถามให้มากที่สุดแล้วนำมาแยกประเภทของคำถามว่าเป็นคำถามที่เกี่ยวกับอุปกรณ์ใดบ้าง จากนั้นจึงเริ่มสร้าง decision tree แล้วนำมาทำ rule induction

ขั้นตอนในการทำ Knowledge Acquisition

- กำหนดขอบเขตของปัญหา (Problem Domain)
- ทำการสร้าง Knowledge Base
- ทดสอบและทำการ Debugging ตัว Knowledge Base ที่สร้าง

3.4 ขั้นตอนการพัฒนาระบบ

ในขั้นตอนการพัฒนาระบบ เราใช้ Expert System Shell คือ CLIPS โดยใช้ **Visual Basic 6.0** เป็น user interface การที่จะทำให้ VB (Visual Basic) ทำงานได้กับ CLIPS จำเป็นต้องใช้ OCX ตัวหนึ่งคือ **CLIPS OCX** โดย download จาก http://ourworld.compuserve.com/homepages/olsont/ มา install ลงเครื่องเสียก่อน โดยเครื่องที่จะใช้งานนั้นต้องมีโปรแกรม Visual Basic version 6.0 ด้วย

ไฟล์ส่วนที่เกี่ยวข้องกับการทำงานของ CLIPS มีไฟล์หลักหนึ่งตัวคือ Main.CLP ซึ่งเป็นไฟล์ที่เขียนโดย CLIPS จะเขียนคำสั่งและ rule ที่ใช้ในการทำงาน เป็น text file อย่างหนึ่ง โดยจะเขียน text file ซึ่งมีสกุล .DAT ขึ้นมาอีกต่างหากเพื่อใช้ในการเก็บ fact ทั้งหมด โดยจำแนกตามประเภทของปัญหา เช่น ไฟล์ที่เก็บ fact สำหรับแก้ปัญหาจอภาพก็ตั้งชื่อว่า Monitor.DAT โดย fact ที่ถูกเก็บในแต่ละไฟล์นี้ จะแทน decision tree ในการแก้ปัญหาของแต่ละอุปกรณ์นั่นเอง

3.4.1 Fact

ในแต่ละไฟล์สกุล .DAT จะเก็บ fact หรือแทนด้วยแต่ละ node ใน decision tree เอาไว้ ดังนั้นเรากำหนดชื่อของ fact เหล่านั้นว่า “node” โดยใน fact node มีคุณสมบัติดังนี้

- **Name** แทนชื่อของ node เช่น root, node1 หรือ node2
- **Type** แทนชนิดของ node โดยใน decision tree จะมี node 2 ชนิดคือ decision node และ answer node
- **Question** แทนคำถามของ decision node โดยใน answer node จะมีค่าเป็น nil หรือค่าว่าง
- **Yes-node** แทนชื่อของ child node ทางซ้าย(เมื่อคำตอบคือใช่) ใน answer node จะมีค่าเป็น nil
- **No-node** แทนชื่อของ child node ทางขวา(เมื่อคำตอบคือไม่ใช่) ใน answer node จะมีค่าเป็น nil หรือค่าว่าง
- **Answer** แทนคำตอบของ answer node โดยใน decision node จะมีค่าเป็น nil
- **Image** แทนชื่อไฟล์รูปภาพประกอบ

ตัวอย่างเช่น

```
(node (name root) (type decision) (question "สวิตช์จอและเครื่องเปิดแล้วหรือไม่?")
    (yes-node node0) (no-node node1) (answer nil) (image nil))
(node (name node0) (type decision) (question "หลอด LED ด้านล่างของจอติดหรือไม่?")
    (yes-node node2) (no-node node3) (answer nil) (image nil))
(node (name node1) (type answer) (question nil) (yes-node nil) (no-node nil)
    (answer "ทำการเปิดสวิตช์จอและเครื่องเสีย") (image "Monitor1.JPG"))
```

3.4.2 Rule

Rule ที่จำเป็นต่อการทำงานได้เก็บไว้ในไฟล์ Main.CLP ได้แก่ rule ต่อไปนี้

- **Initialize** เมื่อเริ่มทำงาน โปรแกรมจะเริ่มที่ rule นี้ก่อน การทำงานของ rule นี้คือ ถ้า node ปัจจุบัน ไม่ใช่ root ก็จะทำการ load fact จากไฟล์สกุล .DAT เข้าสู่ working memory และกำหนดให้ค่าใน fact current-node เป็น root เราใช้คำสั่งดังต่อไปนี้ในการสร้าง initial rule

```
(defrule initialize
            (not (node (name root)))
=>
            (load-facts ?*the_file*)
            (assert (current-node root)))
```

- **Ask-decision-node-question** การทำงานของ rule นี้คือ ถ้าใน current-node เป็น decision node และใน fact answer ไม่มีค่าอยู่แล้ว ให้ใส่คำถามลงในตัวแปร global ที่ชื่อ the_question เพื่อที่ VB จะนำค่านี้ไปแสดงคำถามบนหน้าจอ เราใช้คำสั่งสร้าง rule ดังนี้

```
(defrule ask-decision-node-question
    ?node <- (current-node ?name)
        (node (name ?name)
        (type decision)
        (question ?question))
        (not (answer ?))
=>
        (bind ?*the_question* ?question))
```

- **Bad-answer** เป็น rule ที่ใช้เช็คว่าคำตอบของ user ที่ส่งมาใช้ได้หรือไม่ ถ้านอกเหนือจาก yes และ no จะทำการลบค่าใน fact answer เพื่อให้ใส่ค่าลงไปใหม่ มีคำสั่งสร้าง rule ดังนี้

```
(defrule bad-answer
        ?answer <- (answer ~yes&~no)
=>
        (retract ?answer))
```

- **Proceed-to-yes-branch** เนื่องจากโครงสร้างของ decision tree ที่ใช้เป็น binary tree เมื่อตอบใช่จะเลื่อน current-node ไปยัง yes-node การทำงานของ rule นี้คือถ้า current-node เป็น decision node และคำตอบที่ส่งมาเป็น yes แล้วให้ใส่ค่าใน current-node เป็นชื่อ node ใน slot yes-node โดยมีคำสั่งในการสร้าง rule ดังนี้

```
(defrule proceed-to-yes-branch
        ?node <- (current-node ?name)
    (node (name ?name)
        (type decision)
        (yes-node ?yes-branch))
        ?answer <- (answer yes)
=>
        (retract ?node ?answer)
        (assert (current-node ?yes-branch)))
```

- **Proceed-to-no-branch** การทำงานเหมือน proceed-to-yes-branch แต่แทนที่จะใส่ค่าใน current-node เป็นชื่อ node ใน slot yes-node ก็ไปใส่ค่าใน no-node แทน

```
(defrule proceed-to-no-branch
        ?node <- (current-node ?name)
        (node (name ?name)
        (type decision)
        (no-node ?no-branch))
        ?answer <- (answer no)
=>
        (retract ?node ?answer)
        (assert (current-node ?no-branch)))
```

- **Display-answer-and-picture** หน้าที่การทำงานหลักคือส่งค่าคำตอบและรูปภาพเป็นผลลัพธ์ให้กับ user โดยค่าคำตอบจะส่งไปเก็บใน bufffer ที่เรียกว่า wdialog เพื่อให้ VB นำไปแสดงผลต่อไป และนำชื่อไฟล์รูปภาพไปเก็บไว้ในตัวแปร global ที่ชื่อ the_picture ให้ VB สามารถนำตัวแปรตัวนี้ไปหาไฟล์รูปภาพนำมาแสดงผลได้ โดยมีเงื่อนไขการทำงานคือ current-node ต้องมีชนิดเป็น answer node และไม่มีค่าใน fact answer

```
(defrule display-answer-and-picture
        ?node <- (current-node ?name)
        (node (name ?name) (type answer) (answer ?value) (image ?pic))
        (not (answer ?))
=>
        (printout wdialog ?value crlf)
        (bind ?*rep_flag* no)
        (bind ?*the_picture* ?pic))
```

- **Answer-node-guess-is-correct** การทำงาน ถ้า current-node มีชนิดเป็น answer node และ fact answer มีค่าเป็น yes และมีการใส่ fact ชื่อว่า try-again เข้ามาแล้ว จะทำการลบค่าใน current-node และ answer

```
(defrule answer-node-guess-is-correct
        ?node <- (current-node ?name)
        (node (name ?name) (type answer))
```

```
        (try-again)
        ?answer <- (answer yes)
  =>
        (retract ?node ?answer))
```

- **Answer-node-guess-is-incorrect** ในกรณีที่ผลลัพธ์ที่ได้ไม่ถูกต้อง จะมีการใส่ค่าใหม่ลงใน knowledge base การทำงานถ้า current-node เป็น answer node และ answer มีค่า no และ มีการใส่ fact ชื่อว่า incorrect-answer แล้ว จะทำการใส่ค่า node ปัจจุบันลงใน fact replace-answer-node เพื่อทำการสร้าง decision และ answer node ขึ้นใหม่ มีคำสั่งสร้าง rule ดังนี้

```
  (defrule answer-node-guess-is-incorrect
        ?node <- (current-node ?name)
        (node (name ?name) (type answer))
        ?phase <- (incorrect-answer)
        ?answer <- (answer no)
  =>
        (assert (replace-answer-node ?name))
        (bind ?*needInput* ?name)
        (retract ?phase ?node ?answer))
```

- **One-more-time** หน้าที่การทำงานคือจะกำหนดค่าใน current-node เป็น root เพื่อเริ่มต้นการทำงานใหม่อีกครั้ง ถ้ามีการใส่ fact try-again และไม่มีค่าใน fact answer

```
  (defrule one-more-time
        ?phase <- (try-again)
        (not (answer ?))
  =>
        (retract ?phase)
        (assert (current-node root)))
```

- **No-more** ในกรณีที่สิ้นสุดการทำงาน จะมีการใส่ fact not-try-again การทำงานจะบันทึกข้อมูล fact ใน working memory ในขณะนั้น(ก่อนสิ้นสุดการทำงาน)ลงในไฟล์ สกุล .DAT

```
  (defrule no-more
        ?phase <- (not-try-again)
  =>
```

```
        (retract ?phase)
        (save-facts ?*the_file* local node))
```

- **Replace-answer-node** ในกรณีที่ต้องการเปลี่ยนคำตอบหรือผลลัพธ์ โดย user การทำงานคือถ้ามีการใส่ fact replace-answer-node เข้ามาแล้ว จะเปลี่ยน node นั้น(เดิมเป็น answer node)เป็น decision node และใส่ค่า slot question เป็นคำถามที่ส่งมาจาก user (ผ่านทาง VB) และค่าใน slot answer และ image เป็นค่าว่าง โดยจะสร้าง node ใหม่ขึ้นอีก 2 node เป็น yes-node และ no-node โดยนำคำตอบใหม่และชื่อไฟล์รูปภาพใหม่ที่ได้มาจาก user ใส่ลงใน slot answer และ image ของ yes-node และนำคำตอบและรูปภาพเก่าไปใส่ลงใน no-node มีคำสั่งในการสร้าง rule ดังนี้

```
(defrule replace-answer-node
        ?phase <- (replace-answer-node ?name)
        ?data <- (node (name ?name)
    (type answer)
    (answer ?value)
        (image ?pic))
=>
        (retract ?phase)
        (bind ?new-remedy ?*new_remedy*)
        (bind ?question ?*new_question*)
        (bind ?new-pic ?*new_picture*)
        (bind ?newnode1 (gensym*))
        (bind ?newnode2 (gensym*))
        (modify ?data (type decision)
                    (question ?question)
                    (yes-nodc ?newnode1)
                    (no-node ?newnode2)
                        (answer nil)
                        (image nil))
        (assert (node (name ?newnode1)
                    (type answer)
```

```
                    (answer ?new-remedy)
                              (image ?new-pic)))
          (assert (node (name ?newnode2)
                    (type answer)
                    (answer ?value)
                              (image ?pic)))
     (bind ?*rep_flag* yes))
```

3.4.3 การพัฒนาโปรแกรมด้วย Microsoft Visual Basic 6.0 ให้ทำงานร่วมกับ CLIPS

ในการพัฒนาโปรแกรมโดยให้ Microsoft Visual Basic 6.0 ทำงานร่วมกันได้กับ CLIPS จำเป็นต้องมี ActiveX Control ตัวหนึ่งคือ **CLIPS OCX** ซึ่งสามารถหา download ได้จาก website ของ http://ourworld.compuserve.com/homepages/olsont/ โดย download ไฟล์ชื่อ CLIPSOCX.EXE มา install ลงเครื่องเสียก่อน เมื่อทำการ install สำเร็จแล้วจะปรากฏ control ตัวใหม่บน toolbar นั่นคือ CLIPS ActiveX Control การเรียกใช้งานก็เพียงแค่เลือก control ตัวนี้ไปแปะไว้บน form ของ VB (Visual Basic) เท่านั้น ก็สามารถเรียกใช้ Classes, Methods และ Properties ของ control ตัวนี้ได้ วิธีการใช้งาน control ตัวนี้มีรายละเอียดอยู่ใน CLIPS ActiveX Help ซึ่งจะถูก install มาพร้อมกัน โดยจะถูกสร้างเป็น shortcut อยู่ในเมนู Program ของ Windows ในโครงการนี้จะใช้ Methods ที่สำคัญบางตัว ได้แก่

Clear Method

- BOOL Clear()
- ใช้เตรียม CLIPS Environment โดยจะลบทุกคำสั่งใน Workspace ออก เหมือนคำสั่ง Clear ใน CLIPS นั่นเอง

Reset Method

- VOID Reset()
- ใช้ reset CLIPS Environment

Run Method

- LONG Run(LONG activations)
- ใช้ Execute CLIPS Environment ในปัจจุบัน โดยจะรับค่า parameter เป็น LONG ซึ่งหมายถึงจำนวน activation ที่จะทำงาน ถ้ามีค่าเป็น –1 การทำ activation จะทำงานทุก rule ใน agenda จนครบ โดย method นี้จะ return ค่า LONG เป็นจำนวนของ activation ที่ทำสำเร็จ

AssertString Method

- BOOL AssertString(String theFact)
- ใช้เพิ่ม Fact ลงใน CLIPS Fact-list โดยค่า string ที่ส่งไปจะแทน Fact ตัวอย่างเช่น ถ้าส่งค่า "(example (x 5) (y red) (z 1.5 b))" จะหมายถึงการสร้าง fact ชื่อ example โดยมี 3 slot ได้แก่ x, y และ z โดย method นี้จะ return ค่า TRUE ถ้าสำเร็จและค่า FALSE ถ้าไม่สำเร็จ

GetDefglobalString Method

- String GetDefglobalString(String defglobalName)
- รับค่า String ที่เป็น defglobal หรือตัวแปรแบบ global ใน CLIPS โดยจะ return ค่า string ของตัวแปร defglobal หรืออาจเป็น NULL ถ้ามีข้อผิดพลาด

Load Method

- BOOL Load(String FileName)
- ใช้ Load คำสั่งจากไฟล์ตามชื่อที่ส่งให้ เช่น Load("Main.CLP") จะคืนค่า TRUE ถ้าสำเร็จและค่า FALSE ถ้าผิดพลาด

DoCLIPSCommand Method

- BOOL DoCLIPSCommand(String theCommand)
- ใช้ Execute คำสั่งของ CLIPS โดยมีรูปแบบคำสั่งตามที่ส่งค่า string เข้าไป ตัวอย่างเช่น DoCLIPSCommand("(bind ?*x* 5)") หมายถึง ทำคำสั่ง bind ตั้งค่าตัวแปร defglobal x ให้มีค่าเป็น 5 โดย method นี้จะคืนค่า TRUE ถ้าสำเร็จ FALSE ถ้าผิดพลาด

การแสดงคำถามใน Form ของ VB

คำถามที่แสดงบนระบบผู้เชี่ยวชาญ มาจาก Slot Question ของ Decision Node หรือ node ที่มี Slot Type เป็น Decision นั่นเอง ไฟล์ที่เก็บ fact เหล่านี้เป็น text file ธรรมดา จัดเก็บในนามสกุล .DAT โดยในไฟล์ Main.CLP จะมีการประกาศตัวแปร defglobal ที่ชื่อ the_question ที่ใช้เก็บค่า string คำถามเพื่อนำมาแสดงใน form ของ VB

ในการทำงาน เมื่อเริ่มรันโปรแกรม หาก Current Node ใน Decision Tree เป็น Decision Node จากกฎ ask-decision-node-question จะนำค่าจาก Slot Question ของ Current Node ไปใส่ในตัวแปร the_question ซึ่งใน VB จะใช้คำสั่ง GetDefglobalString ดึงค่าคำถามนี้มาเก็บในตัวแปร string ตัวหนึ่งคือตัวแปร Question เพื่อรอการนำไปแสดงบน form โดยมี code ตัวอย่างดังนี้

```
Dim Question as String
Question = CLIPS.GetDefglobalString("the_question")
```

โดยที่ CLIPS แทน CLIPS ActiveX Control ที่ถูกแปะบนฟอร์ม ค่าที่ได้ในตัวแปร Question จะติดเครื่องหมาย Double Quote (") มาด้วย เพราะฉะนั้นการนำค่าตัวแปรนี้ไปใช้งานจึงควรเขียน function ตัดเครื่องหมายนี้ออกเสียก่อน ตัวอย่างของ function นี้มีดังนี้

```
Private Function CutDoubleQuote(ByVal Tmp As String) As String
Dim Fact As String
Dim start, pos As Long
  If Tmp <> "" Then
    start = 2
    pos = InStr(start, Tmp, """", vbTextCompare)
    CutDoubleQuote = Mid(Tmp, start, pos - start)
  End If
End Function
```

การแสดงคำตอบและรูปภาพ

เมื่อการทำงานมาถึงส่วนที่เป็น Leaf ของ Decision Tree จะเป็น Answer Node นั่นคือจะเป็นส่วนที่เก็บคำตอบหรือผลลัพธ์ของระบบเพื่อแสดงให้แก่ผู้ใช้ Answer Node จะมี Slot Type เป็น Answer และมีค่าใน Slot Answer เป็นคำตอบที่เราต้องการ และมีรูปประกอบที่ถูกเก็บไว้ใน Slot Image ด้วย จากกฎ display-answer-and-picture คำตอบจะถูกเก็บไว้ใน buffer ตัวหนึ่งคือ wdialog และชื่อไฟล์รูปภาพจะถูกเก็บไว้ในตัวแปร defglobal ที่ชื่อ the_picture

ในการเรียกใช้ค่าใน wdialog เราใช้ Object ตัวหนึ่งที่ชื่อ CLIPSRouter มี code ดังนี้

```
Dim Mydata as New CLIPSRouter
Dim Temp, Tmp as String
'Display Result
While Not Mydata.EOF
  Temp = Temp + Mydata.Value
  Mydata.MoveNext
Wend
Mydata.Empty
If Temp <> "" Then
  'Display Images
  Tmp = CLIPS.GetDefglobalString("the_picture")
```

```
On Error Resume Next
Image1.Picture = LoadPicture(App.Path + "\Images\" + CutDoubleQuote(Tmp))
```

บทที่ 4
การใช้งานโปรแกรม

4.1 โปรแกรมนี้เหมาะสำหรับใครบ้าง

- **ผู้เริ่มใช้งานคอมพิวเตอร์(Beginner User)** ได้แก่ผู้ที่มีความรู้พื้นฐานเกี่ยวกับอุปกรณ์คอมพิวเตอร์น้อย โปรแกรมนี้จะช่วยแก้ปัญหาและเพิ่มประสบการณ์แก่ผู้ใช้เหล่านี้ได้
- **ผู้ที่สามารถใช้คอมพิวเตอร์ได้ระดับปานกลาง(Medium User)** สำหรับผู้ใช้ประเภทนี้สามารถใช้โปรแกรมเพื่อช่วยแก้ปัญหาและให้คำแนะนำ ช่วยเพิ่มความรู้และประสบการณ์
- **หน่วยงานที่มีผู้เชี่ยวชาญเรื่องคอมพิวเตอร์น้อย** สามารถใช้ประโยชน์จากโปรแกรมนี้ได้เพื่อช่วยลดภาระของผู้เชี่ยวชาญ และยังสามารถนำไปใช้ประกอบการอบรมให้แก่ผู้ใช้ที่ยังไม่มีประสบการณ์เรื่องคอมพิวเตอร์ได้

4.2 คุณสมบัติของเครื่องที่สามารถใช้งานได้

- เป็นเครื่องคอมพิวเตอร์ตั้งแต่รุ่น Pentium 100 ขึ้นไป
- หน่วยความจำ(RAM) 32 MB พื้นที่ใน Hard disk เหลืออย่างต่ำ 30 MB เพื่อให้การทำงานไม่ช้าเกิน(ขนาดของโปรแกรม ประมาณ 1-2 MB)
- ระบบปฏิบัติการ Windows 9x, Windows 2000, Windows ME
- ติดตั้งโปรแกรม Microsoft Visual Basic เวอร์ชัน 6.0
- ไฟล์ CLIPSOCX.EXE

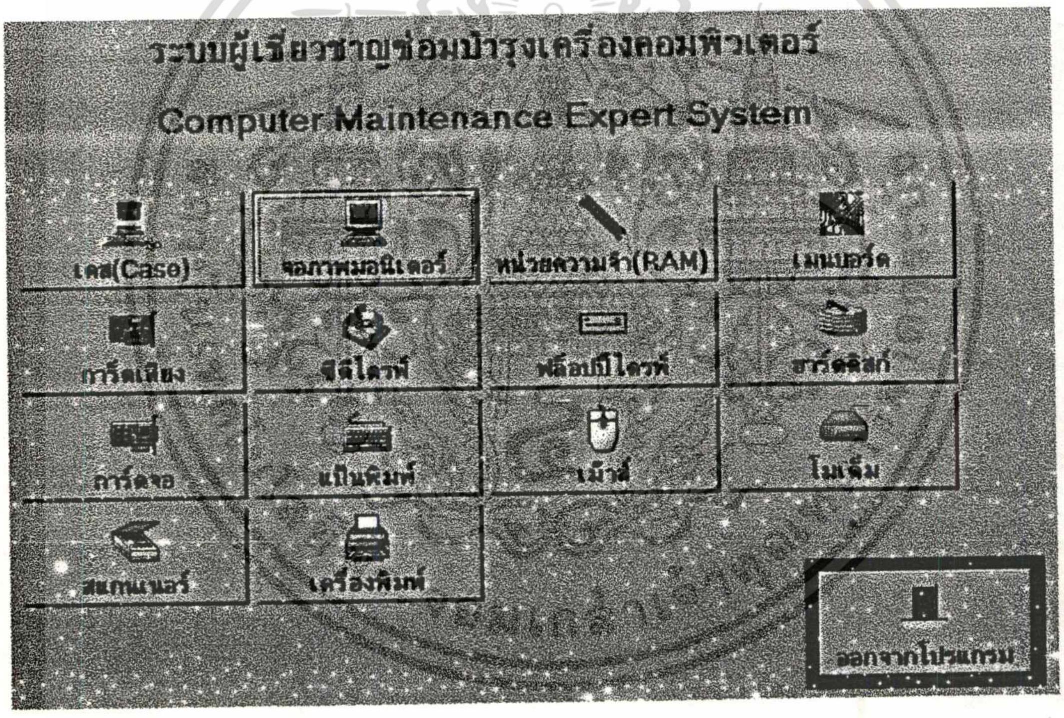
4.3 การติดตั้ง CLIPS OCX

CLIPS OCX เป็น ActiveX Control ตัวหนึ่งของ Microsoft Visual Basic หากไม่ติดตั้ง OCX ตัวนี้เข้าไป โปรแกรมจะไม่สามารถทำงานได้ วิธีการติดตั้งให้ double click ที่ CLIPSOCX.EXE ใน Windows Explorer จะเข้าสู่หน้าจอการติดตั้งโปรแกรม ให้กดปุ่ม next ไปเรื่อยๆ โปรแกรมจะทำการ copy ไฟล์เองโดยอัตโนมัติ เมื่อทำการติดตั้งเสร็จแล้วในส่วนของ

Program Menu จะมี shortcut ของ CLIPS ActiveX Control ขึ้น แสดงว่าติดตั้งเสร็จเรียบร้อย โดยใน shortcut ชื่อ CLIPSActiveX Help เป็นคู่มือการใช้ CLIPS OCX

4.4 หน้าจอการใช้งานโปรแกรม

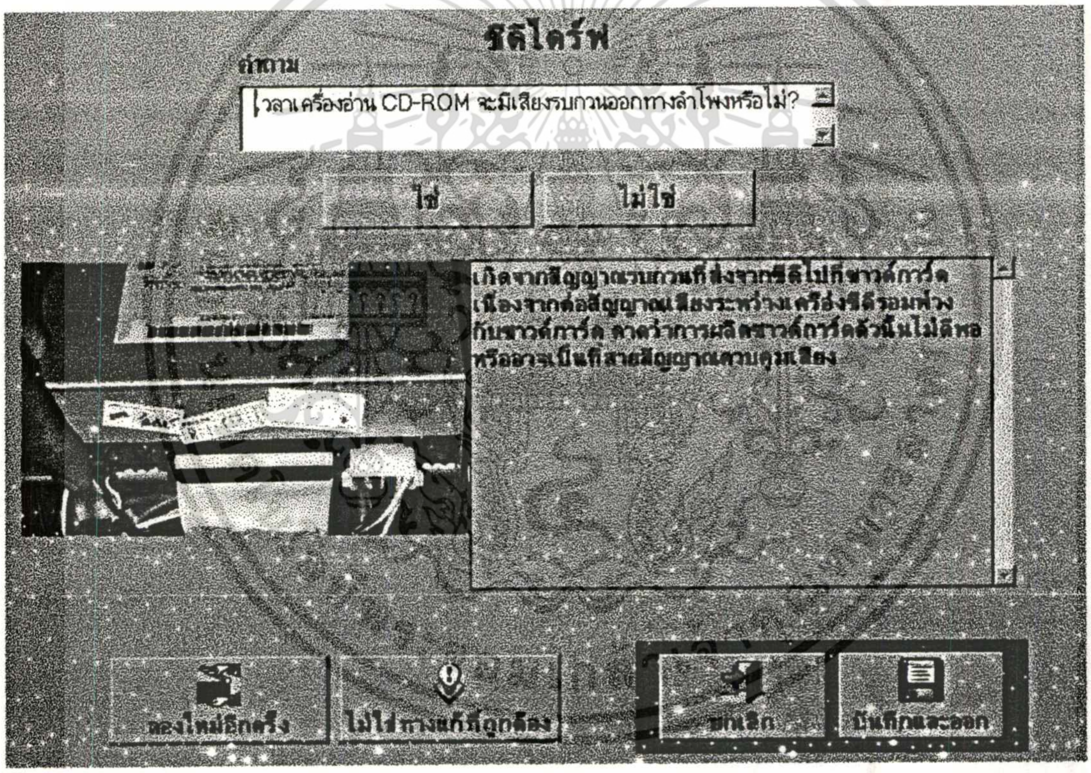
- **หน้าจอหลัก (Main Menu)** เป็นหน้าจอแรกเมื่อเริ่มเข้าสู่โปรแกรม มีปุ่มให้เลือกว่าจะเข้าสู่การตอบปัญหาเกี่ยวกับอุปกรณ์ประเภทใด เมื่อเลือกกดปุ่มของอุปกรณ์หนึ่งแล้วจะทำการเปิดหน้าจอเพื่อถามคำถามวินิจฉัยปัญหาของผู้ใช้เกี่ยวกับอุปกรณ์นั้นๆ เช่น เลือกกดปุ่ม"ซีดีไดร์ฟ" ก็จะไปสู่หน้าจอวินิจฉัยปัญหาซีดีไดร์ฟ เป็นต้น
    - **ปุ่มออกจากโปรแกรม** เมื่อกดปุ่มจะทำการออกจากโปรแกรมระบบผู้เชี่ยวชาญ



รูปที่ 5.1 หน้าจอหลัก(Main Menu)

- **หน้าจอคำถาม(Question Form)** เป็นหน้าจอการทำงานหลักของระบบ หน้าจอนี้จะถามคำถามแก่ผู้ใช้ โดยมีคำตอบให้เลือกคือ **ใช่** และ **ไม่ใช่** ผู้ใช้จะตอบคำถามไปเรื่อยๆ จนกระทั่งได้ผลลัพธ์ปรากฏขึ้นบนหน้าจอ
    - **ปุ่มลองใหม่อีกครั้ง** เมื่อผู้ใช้ได้คำตอบหรือผลลัพธ์จากระบบแล้ว สามารถเริ่มตอบคำถามใหม่ได้อีก โดยกดปุ่มนี้ ระบบจะเริ่มตั้งคำถามใหม่ตั้งแต่ต้น

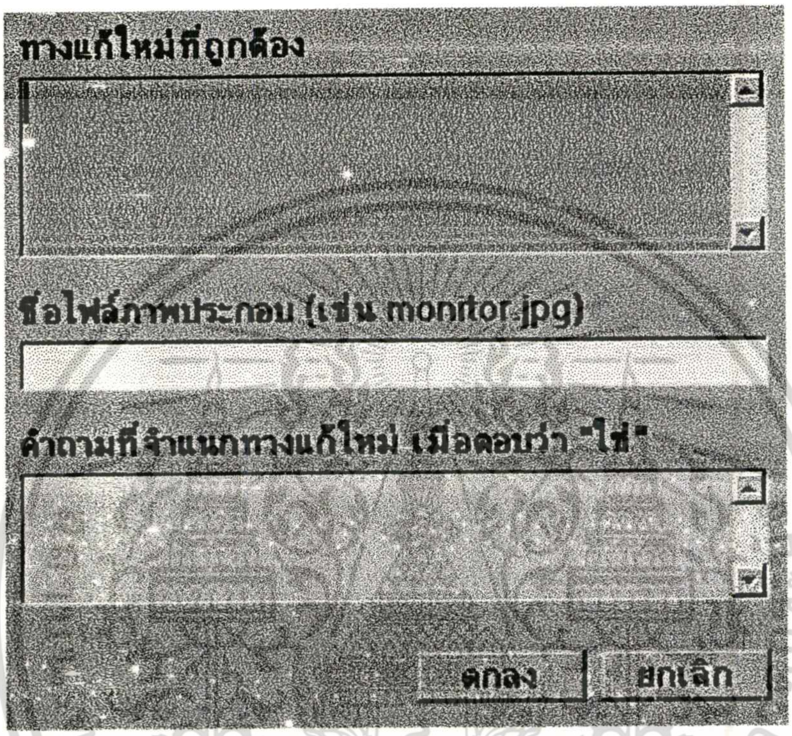
- ○ **ปุ่มไม่ใช่ทางแก้ที่ถูกต้อง** เมื่อผลลัพธ์ที่ระบบให้มาไม่ถูกต้องหรือไม่เหมาะสม ผู้ใช้สามารถเปลี่ยนคำตอบใหม่ได้โดยกดปุ่มนี้ และจะปรากฏหน้าจอขึ้นให้ตั้งคำถามและคำตอบใหม่ขึ้น
- ○ **ปุ่มยกเลิก** ผู้ใช้สามารถยกเลิกการทำงานของระบบในปัจจุบัน แล้วกลับสู่หน้าจอหลักอีกครั้งโดยกดปุ่มนี้
- ○ **ปุ่มบันทึกและออก** เมื่อผู้ใช้มีการเปลี่ยนแปลงแก้ไขคำถามและคำตอบขึ้นใหม่ ปุ่มนี้จะทำการบันทึกข้อมูลใหม่ที่ได้รับลงไฟล์และกลับไปสู่หน้าจอหลัก



รูปที่ 5.2 หน้าจอคำถาม(Question Form)

- **หน้าจอกำหนดทางแก้ใหม่(New Remedy Form)** เป็นหน้าจอสำหรับตั้งคำถามและคำตอบใหม่ ในกรณีที่ระบบแสดงผลลัพธ์ที่ไม่ถูกต้องหรือไม่เหมาะสม เมื่อเข้าสู่หน้าจอนี้ระบบจะให้ใส่คำตอบใหม่ ชื่อไฟล์รูปประกอบ(โดยที่ไฟล์รูปประกอบทุกรูปจะอยู่รวมกันใน directory ชื่อ Images) และคำถามเพื่อจำแนกทางแก้ใหม่ เมื่อตอบว่า"ใช่" เมื่อกดปุ่มตกลง โปรแกรมจะทำการ update ความรู้ภายใน knowledge base ใหม่ เมื่อผู้ใช้ตอบคำถามใหม่

อีกครั้ง คำถามใหม่ที่ใส่เข้ามานี้จะไปแทนที่คำตอบเก่า เมื่อผู้ใช้ตอบว่าใช่ในคำถามใหม่นี้จะได้คำตอบใหม่ตามที่ใส่เข้าไป หากผู้ใช้ตอบว่าไม่จะได้คำตอบเก่าเหมือนกับก่อนที่จะแก้ไข



รูปที่ 5.3 หน้าจอกำหนดทางแก้ใหม่(New Remedy Form)

บทที่ 5
สรุป

5.1 ผลการพัฒนาโปรแกรม

ในการพัฒนาโปรแกรมพบว่า การพัฒนาระบบผู้เชี่ยวชาญโดยใช้ CLIPS OCX ของ Visual Basic 6.0 สามารถเขียนโปรแกรมควบคุมการทำงานได้ดีกว่าการใช้โปรแกรมสำเร็จรูปสำหรับพัฒนาระบบผู้เชี่ยวชาญ เนื่องจากการทำงานโดยใช้ VB เป็น interface ทำให้สามารถกำหนดรูปแบบการใช้งานและทำงานได้ดียิ่งขึ้น ผู้พัฒนาสามารถพัฒนาได้ตามความต้องการไม่จำเป็นต้องติดอยู่กับ template ที่มีให้มากับโปรแกรมสำเร็จรูปอื่นๆ ทำให้การทำงานยืดหยุ่นยิ่งกว่า และได้ผลลัพธ์ที่น่าพอใจ มี interface ที่สวยงาม มีภาพประกอบ ซึ่งเมื่อเปรียบเทียบกับ CLIPS ธรรมดาที่ทำงานบน text mode แล้ว มีความน่าใช้งานมากกว่ามาก

5.2 ปัญหาที่เกิดขึ้น

เนื่องจาก CLIPS OCX ไม่ได้เป็น freeware เหมือน CLIPS ดังนั้นเมื่อครบกำหนด 30 วันจึงใช้งานไม่ได้ จำเป็นต้อง uninstall ออกโดยต้องเข้าไปแก้ใน register ด้วย แล้วจึงทำการ install ลงใหม่ จึงใช้งานได้ต่อไป

วิธีการ Uninstall CLIPS OCX

1. เปิด Control Panel เลือก Add/Remove Programs
2. ในแทป Install/Uninstall ให้เลือก CLIPS OCX ในช่องรายการแล้วกดปุ่ม Add/Remove จะปรากฏข้อความถามว่าจะ uninstall CLIPS OCX หรือไม่ ให้เลือก yes จากนั้นเครื่องจะทำการลบไฟล์ที่เกี่ยวข้องกับ CLIPS OCX
3. เมื่อ Uninstall เสร็จแล้วเราต้องไปลบบางส่วนที่ถูก registry เอาไว้ด้วย ให้กดปุ่ม start ใน Windows แล้วเลือก Run จากนั้นให้พิมพ์ regedit แล้วกด Enter
4. ในโปรแกรม Registry Editor ให้เข้าไปที่ห้อง HKEY_LOCAL_MACHINE แล้วเข้าไปที่ห้อง SOFTWARE หาห้องที่ชื่อ Athena และ Athena Information Services, Inc (ถ้ามี) แล้วลบทั้งสองห้องนี้เสีย
5. ออกจากโปรแกรม Registry Editor เป็นการเสร็จสิ้นการ uninstall CLIPS OCX

5.3 ข้อเสนอแนะและแนวทางการศึกษาในอนาคต

- การพัฒนาในอนาคตอาจมีการทำเป็น Web Application โดยติดตั้งโปรแกรมและ knowledge base เอาไว้ที่ server โดยฝั่ง client เมื่อจะใช้งานก็ทำการ request ไปยังฝั่ง server การทำงานจะทำบนฝั่ง server เป็น server process
- เนื่องจากเวลาในการพัฒนาที่มีน้อย เพราะผู้พัฒนาโครงการเปลี่ยนโครงการกะทันหัน ทำให้ข้อมูลที่รวบรวมเป็น Knowledge base อาจยังมีไม่มากพอ การออกแบบ decision tree ก็ยังทำได้ไม่ optimize นัก ระบบที่พัฒนาก็ยังเป็น stand alone อยู่
- เมื่อเป็น Web Application ไปแล้ว ก็อาจมีการขยายขอบเขตการทำงาน ให้สามารถตรวจสอบราคาอุปกรณ์คอมพิวเตอร์ในท้องตลาดหรือบน internet โดยอาจทำเป็น Intelligent Agent ต่อไปได้
- การนำ Binary Tree มาสร้าง Decision Tree มีข้อจำกัดอยู่มาก คำตอบจึงมีแค่ 2 ทางเลือกเท่านั้น หากต้องการมีทางเลือกมากกว่าสองทางนอกจากใช่ไม่ใช่เช่น ไม่ทราบ(unknow) ก็จะทำไม่ได้ ดังนั้นจึงต้องหาวิธีใหม่ในการสร้าง Decision Tree ที่มีความยืดหยุ่นมากยิ่งขึ้น
- คำตอบที่ได้จาก Expert system ควรมีค่าความมั่นใจ (Confident Factor หรือค่า CF) ซึ่งจะบอกระดับความถูกต้องของผลลัพธ์ที่ได้ อาจบอกเป็นเปอร์เซ็นต์(0-100%)หรือค่า 1-10 ได้
- ในการใช้งานระบบ เมื่อเข้าสู่หน้าจอตอบคำถามเกี่ยวกับอุปกรณ์หนึ่งๆ ควรมี Link หรือปุ่มที่นำไปสู่หัวข้อที่เกี่ยวข้องกับอุปกรณ์นั้นๆเพื่อเป็นประโยชน์กับผู้ใช้ เช่น เมื่อเข้าสู่หน้าจอของอุปกรณ์ Hard disk ก็ควรมีปุ่มไปยังหน้าจอการทำ Defragment หรือการทำ FDISK เป็นต้น

บรรณานุกรม

Athena Information Services, Inc. CLIPS ActiveX Control. [Online]. Available : http://ourworld.compuserve.com/homepages/olsont/

Awad, Elias M. Building Expert System: Principles, Procedures, and Applications.

Bonnet, A. Et. Al. Expert System: Principle and Practice.

Giarratano, Joseph C. 1997. CLIPS: User's Guide. Version 6.05.

Girratano and Riley. Expert System: Principle and Programming.

ประวัติผู้เขียน

ชื่อ-นามสกุล	นาย จุฑาสิทธ์ โรหิตรัตนะ
วัน เดือน ปี ที่เกิด	10 พฤศจิกายน พ.ศ. 2521
ภูมิลำเนาที่เกิด	กรุงเทพมหานคร
ประวัติการศึกษา	ปริญญาตรี วิทยาศาสตร์บัณฑิต(วทบ.) สาขาวิชาเอกวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศรีนครินทรวิโรฒ ประสานมิตร ปริญญาโท แขนงวิชาวิทยาการสารสนเทศ คณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ผลงานในอดีต	1. ระบบขายและจองตั๋วรถไฟ – หนึ่งใน Project ในระดับปริญญาตรี พัฒนาโดย Microsoft Access 2. ระบบคิดราคาและคัดพันธุ์ปลา – เป็นระบบที่ใช้ร่างสัญญาซื้อขาย ทำการ Sizing(คัดขนาดปลา) และคิดราคาตามสัญญา เป็น Project ของบริษัท TriMarine พัฒนาโดย Microsoft Visual Basic 6.0 และ Microsoft Access 3. ระบบการซ่อมบำรุงเครื่องคอมพิวเตอร์ในมหาวิทยาลัย – เป็นระบบที่ช่วยงาน PM หรือการซ่อมบำรุงเครื่องคอมพิวเตอร์ทั้งมหาวิทยาลัยศรีนครินทรวิโรฒ ประสานมิตร พัฒนาโดย Microsoft Visual Basic 6.0