# บทที่ 2
# ความรู้ที่เกี่ยวข้องกับการพัฒนาระบบ

## 2.1 แนวความคิดเชิงวัตถุ (Object Oriented Concept) ของภาษาจาวา

แนวความคิดในการรวมกันระหว่างข้อมูล และ การทำงาน ได้ถูกนำกลับมาพัฒนาขึ้นในทิศทางใหม่ที่เรียกว่าการโปรแกรมเชิงวัตถุ(Object Oriented Programming) การโปรแกรมด้วยภาษาต่าง ๆ ในอดีตจะอาศัยลำดับการทำงานของชุด คำสั่งต่างๆ มาประกอบเป็นโปรแกรม ภาษาเชิงวัตถุได้รวบรวมคุณสมบัติ ที่ดีของโปรแกรมแบบเชิงกระบวนการไว้ และเพิ่มวิธีการจัดการกับอัลกอลิทึมและโครงสร้างข้อมูลขึ้นใหม่ โดยอาศัยแนวความคิดที่สำคัญสามประการ ได้แก่ การป้องกันข้อมูลภายในวัตถุ (Encapsulation) ความหลากหลายรูปแบบ(Polymorphism) และการสืบทอดคุณสมบัติ(Inheritance)

แนวความคิดในการโปรแกรมเชิงวัตถุมีความเหมาะสมกับผู้ใช้หลายระดับ ไม่ว่าจะเป็นผู้จัดการระบบ การโปรแกรมในลักษณะนี้จะเป็นเสมือนสัญญาที่ให้ไว้กับการพัฒนาและการบำรุงรักษาระบบจะมีความรวดเร็วยิ่งขึ้นในขณะเดียวกันก็จะมีราคาต่ำลง ในส่วนของนักวิเคราะห์และออกแบบระบบการวิเคราะห์การทำงานของระบบโดยการสร้างแบบจำลองหรือโมเดลจะสามารถทำได้ง่ายขึ้น รวมไปถึงการออกแบบระบบจะมีความถูกต้องมาขึ้นตามไปด้วย ส่วนความเหมาะสมประการสุดท้ายจะได้แก่ในส่วนของนักพัฒนาโปรแกรม ซึ่งการออกแบบตลอดจนความชัดเจนของแบบจำลองของวัตถุ(Object Model) รวมไปถึงความพร้อมของเครื่องมือและไลบรารี(Libraries) ต่าง ๆ ที่ใช้สนับสนุนการโปรแกรมเชิงวัตถุจะยิ่งทำให้การโปรแกรมลักษณะนี้มีความน่าสนใจมากขึ้น ความสำเร็จของการโปรแกรมเชิงวัตถุไม่ได้จำกัดอยู่ที่รูปแบบและวิธีการที่ภาษากำหนดและตัวคอมไพเลอร์เท่านั้น แต่รวมไปถึงสิ่งแวดล้อมอื่น ๆ เช่น ไลบรารีที่ได้รับการออกแบบมาเป็นอย่างดี ง่ายต่อการเรียกใช้ ดังนั้นงานหลักของนักพัฒนาโปรแกรมส่วนใหญ่จะเป็นการนำวัตถุที่มีอยู่แล้วมาใช้สำหรับการแก้ไขปัญหาที่ต้องการ หากจะมีข้อเสียอยู่บ้างก็จะได้แก่ วิธีการในการเรียนรู้การคิดในเชิงวัตถุจะแตกต่างไปจากแนวความคิดของการโปรแกรมแบบเชิงกระบวนการอย่างสิ้นเชิง และการออกแบบเชิงวัตถุจะมีความท้าทายสูงกว่าการออกแบบด้วยวิธีการแบบเดิม โดยเฉพาะ อย่างยิ่งในกรณีที่ต้องการสร้างวัตถุที่สามารถนำกลับมาใช้ใหม่ได้ (Reusable Objects)

แม้ว่าการโปรแกรมเชิงกระบวนการจะมีประสิทธิภาพในการทำงานสูง แต่ก็ยังมีข้อจำกัดประการหนึ่งได้แก่ เมื่อโปรแกรมมีขนาดใหญ่ขึ้น แนวโน้มของโค้ดภายในโปรแกรมจะกลายเป็นลักษณะที่เรียกว่า สปาเกตตีโค้ด(Spaghetti Code) ซึ่งจะเกิดขึ้นเนื่องจากสาเหตุดังต่อไปนี้

ในความเป็นจริงแล้วงานหลักของนักพัฒนาโปรแกรมจะเริ่มต้นอีกครั้งหนึ่งเมื่อสิ้นสุดการเขียนโปรแกรมและนำไปทดลองใช้จากผู้ใช้จริง ซึ่งหลังจากนั้นการแก้ไขโปรแกรมอาจเกิดขึ้นอย่างมากมาย ทั้งนี้เนื่องมาจากความต้องการเพิ่มเติมจากผู้ใช้ ตลอดจนถึงรายงานข้อผิดพลาดที่เกิดจากการใช้งานจริง เป็นต้น จากเหตุผลดังกล่าวโปรแกรมจะถูกบังคับให้มีการแก้ไขโค้ดใหม่ นั่นก็หมายถึงความจำเป็นในการเพิ่มจำนวนของโครงสร้างควบคุม(Flow Control) มากขึ้น ตลอดจนการเพิ่มฟังก์ชันรวมไปถึงตัวแปรต่างๆ ซึ่งการเพิ่มสิ่งเหล่านี้สำหรับการโปรแกรมเชิงกระบวนการแล้ว จะทำได้ค่อนข้างยาก ทั้งนี้เนื่องจากการโปรแกรมในลักษณะนี้ไม่เพียงแต่มีแนวโน้มที่จะยากต่อการทำความเข้าใจแล้วยังยากที่จะทำการแก้ไขโปรแกรมเหล่านั้นด้วย เนื่องจากทุกๆ อย่างภายในโปรแกรมจะมีลักษณะเป็นเสมือนมีการยึดติดระหว่างกันโดยไม่มีส่วนใดส่วนหนึ่งที่เป็นอิสระต่อกันนั่นเอง ดังนั้นหากมีการแก้ไขโปรแกรมไม่ว่าจะเล็กน้อยเพียงใด มักจะมีแนวโน้มของผลกระทบที่จะเกิดขึ้นต่อส่วนอื่นๆ ค่อนข้างสูงเสมอ ดังนั้นแทนที่จะทำการออกแบบการทำงานของโปรแกรมใหม่ซึ่งจะเสียเวลาค่อนข้างมาก นักพัฒนาโปรแกรมส่วนใหญ่จึงเลือกใช้วิธีการ แก้ไขที่ตัวโค้ดโดยตรง ทั้งนี้เนื่องจากเงื่อนไขที่ถูกจำกัดด้วยเวลาเป็นหลัก

สาเหตุสุดท้ายที่ทำให้เกิดสปาเกตตีโค้ดจะได้แก่ โค้ดที่ถูกเขียนขึ้นในโปรแกรมก่อนหน้านี้จะไม่สามารถนำมาใช้กับโปรแกรมใหม่ได้ทันที ทั้งนี้เนื่องจากการโปรแกรมแบบนี้แม้จะสามารถคัดลอกโค้ดเดิมที่มีอยู่ไปเพิ่มลงในโปรแกรมใหม่ได้ แต่ก็มีแนวโน้มที่จะต้องทำการแก้ไขบางส่วนอยู่เสมอในทุก ๆ โปรแกรมที่ถูกเขียนขึ้นใหม่ อย่างไรก็ตามแม้ว่าการโปรแกรมเชิงกระบวนการที่อยู่ในรูปของไลบรารีที่มีการออกแบบที่ดีและสามารถนำกลับมาใช้ใหม่ได้ง่ายจะมีอยู่บ้าง แต่จะค่อนข้างหาได้ยาก โดยเฉพาะอย่างยิ่งในกรณีของการทำงานเป็นทีมที่มีการพัฒนาหลาย ๆ เวอร์ชันร่วมกัน ในปัจจุบันค่าใช้จ่ายทางด้านโปรแกรมจะถือเป็นค่าใช้จ่ายส่วนใหญ่ในระบบ ความสำคัญในด้านนี้จึงกลายเป็นสิ่งจำเป็นที่จะต้องพิจารณามากเป็นพิเศษ ด้วยความก้าวหน้าทางด้านเทคโนโลยีในปัจจุบัน ได้ส่งผลให้ทรัพยากรของระบบมีจำนวนมากขึ้น ในขณะเดียวกันความซับซ้อนของการเขียนโปรแกรมก็มากขึ้นตามไปด้วย แต่อย่างไรก็ตามการโปรแกรมแบบโครงสร้างก็ได้มาถึงจุดๆ หนึ่งที่ไม่สามารถควบคุมได้ โดยเฉพาะอย่างยิ่งจะก่อให้เกิดปัญหาในกรณีที่มีการทำงานเป็นทีมระหว่างนักพัฒนาโปรแกรมซึ่งต้องทำงานหรือใช้โค้ดบางส่วนร่วมกัน และในกรณีที่ขนาดและความซับซ้อนของโปรแกรมสูงขึ้นเรื่อย ๆ ตัวอย่างเช่น ในภาษาซีหากจำนวนบรรทัดในการเขียนโปรแกรมมากกว่า 100,000 บรรทัด การควบคุมการทำงานของโปรแกรมจะกระทำได้ยากยิ่งขึ้น (รังสิต ศิริรังษี, 2545)

ภาษาเชิงวัตถุ (Object Oriented Languages) ถือเป็นภาษาในระดับที่สาม (3GL) ที่เริ่มเข้ามามีบทบาทในช่วงปลายทศวรรษที่ 80 แม้ว่าจะถูกคิดค้นมาตั้งแต่ปลายทศวรรษ 1960 แต่ก็ไม่ได้รับความนิยมจนกระทั่งปลายทศวรรษที่ 80 เมื่อทรัพยากรของระบบมีเพียงพอที่จะสามารถสนับสนุนการทำงาน

ของภาษาเชิงวัตถุ จุดดังกล่าวนี้เองที่ทำให้การโปรแกรมเชิงวัตถุ ถูกนำมาใช้ในการพัฒนาโปรแกรมแทนการโปรแกรมแบบเดิมอย่างรวดเร็ว และเป็นที่ยอมรับของนักพัฒนาโปรแกรมในปัจจุบัน

การเขียนโปรแกรมเชิงวัตถุได้นำเอาส่วนที่ดีที่สุดของการโปรแกรมแบบโปรซีเยอร์มาพัฒนาร่วมกับแนวความคิดใหม่ ๆ ที่ประยุกต์มาจากสิ่งที่เกิดขึ้นในชีวิตประจำวันของมนุษย์ ซึ่งผลที่ได้จะนำไปสู่แนวทางการเขียนโปรแกรมรูปแบบใหม่ที่มีประสิทธิภาพดีกว่าแบบเดิม โดยปกติแล้วการเขียนโปรแกรมเชิงวัตถุจะทำการแบ่งขอบเขตของงานที่ต้องการออกเป็นส่วนย่อย ๆ ที่มีความสัมพันธ์กันรวมไว้ด้วยกันเป็นกลุ่ม จากนั้นกลุ่มดังกล่าวจะถูกเรียงลำดับในรูปของโครงสร้างที่มีลักษณะเป็นลำดับชั้น (Hierarchical Structure) เพื่อเพิ่มประสิทธิภาพในการเรียกใช้งาน

ภาษาโปรแกรมที่สนับสนุนการโปรแกรมเชิงวัตถุจะต้องประกอบไปด้วยคุณสมบัติ 4 ประการดังต่อไปนี้

1. การซ่อนข้อมูล(Data Hiding) หรือการป้องกันข้อมูล ซึ่งจะเป็นลักษณะของการปิดบังหรือซ่อนเร้นข้อมูลบางส่วนออกจากผู้ใช้ โดยผ่านกลไกการเข้าถึงข้อมูลที่ออกแบบไว้เป็นพิเศษเพื่อป้องกันไม่ให้ผู้ใช้สามารถเข้าถึงข้อมูลที่กำหนดไว้โดยตรง โดยปกติแล้วในการโปรแกรมเชิงวัตถุจะอาศัยการดำเนินการผ่านหน่วยที่เรียกว่า วัตถุ ซึ่งจะทำหน้าที่ในการเก็บทั้งข้อมูลและโค้ดซึ่งใช้เป็นวิธีการในการเข้าถึงข้อมูลนั้น ๆ รวมไว้ในหน่วยเดียวกัน

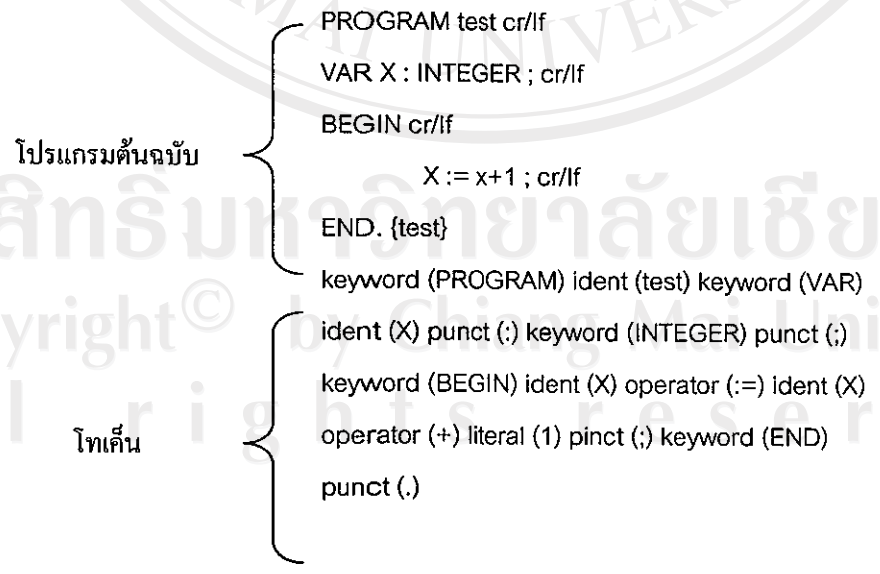
2. ความหลากหลายรูปแบบในการใช้งาน เป็นวิธีการเข้าถึงข้อมูลผ่านการเรียกใช้ฟังก์ชันที่สามารถใช้งานได้กับ วัตถุ หลาย ๆ ชนิด เช่น ฟังก์ชันที่ออกแบบไว้สำหรับการรับค่าพารามิเตอร์จำนวน 3 ค่า แต่สามารถเรียกใช้งานได้แม้ว่าจำนวนพารามิเตอร์จะไม่ครบก็ตาม นอกจากนั้นแล้วโปรแกรมเชิงวัตถุยังยอมให้ผู้ใช้สามารถกำหนดใช้ฟังก์ชันที่มีชื่อเดียวกัน แต่มีจำนวนพารามิเตอร์ที่แตกต่างกันได้

3. การสืบทอดคุณสมบัติ(Hierarchy of Object Definition) เป็นความสามารถของวัตถุหนึ่งที่สามารถสืบทอดคุณสมบัติในการใช้งานไปยังวัตถุอื่นได้ โดยที่คุณสมบัติของวัตถุเดิมไม่มีการเปลี่ยนแปลง การสืบทอดจะมีลักษณะเป็นลำดับชั้น (Hierarchical) ระหว่าง คลาสหลักไปยังคลาสที่ถูกสืบทอด (Derived) ต่อ ๆ ไปได้

4. ตัวแปรชนิดเดียว(Single type) เป็นคุณสมบัติของตัวแปรที่ถูกกำหนดขึ้นในรูปของวัตถุดังที่ได้กล่าวมาแล้วว่าการโปรแกรมเชิงวัตถุจะดำเนินการผ่านวัตถุเป็นหลัก โดยมิได้คำนึงว่าชนิดของวัตถุจะเป็นแบบใด ดังนั้นการเปลี่ยนแปลงใด ๆ ที่เกิดขึ้นกับชนิดของข้อมูลภายในวัตถุ จะไม่ส่งผลกระทบต่อการทำงานภายในโปรแกรม

## 2.2 การวิเคราะห์ศัพท์

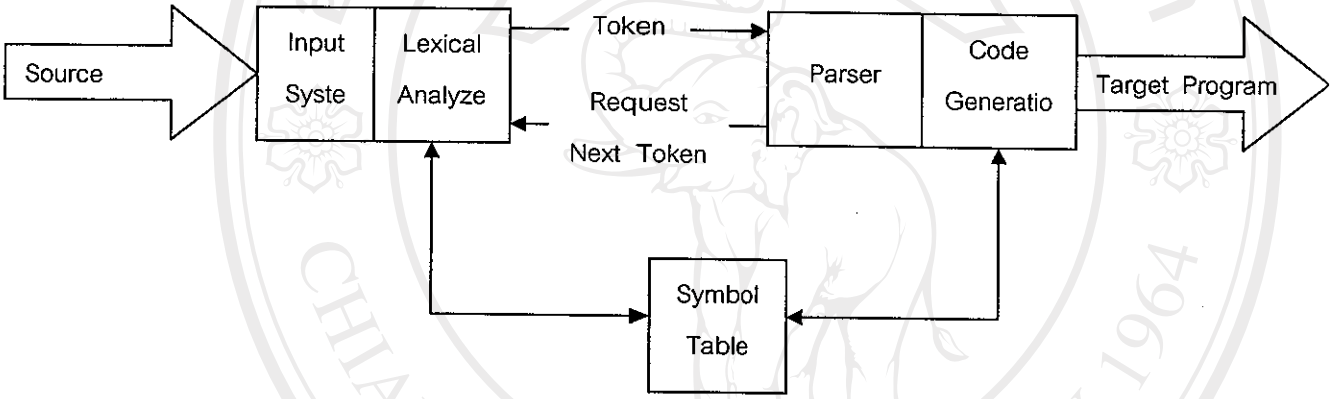
การวิเคราะห์ศัพท์ หรือในตำราบางเล่มเรียกว่าการกราดตรวจ เป็นขั้นตอนแรกในการทำงานของคอมไพเลอร์ ซึ่งมีหน้าที่อ่านข้อมูลในรูปแบบของสายอักขระจากแฟ้มที่เก็บโปรแกรมต้นฉบับแล้วนำอักขระที่อ่านได้มาจัดเป็นกลุ่ม โดยอักขระแต่ละกลุ่มนั้นเรียกว่าหน่วยศัพท์ หรือโทเค็น (Aho et al, 1986, p.84; Louden, 1997,p.31; Wilhelm & Maurer, 1996,p.236) สำหรับในตำราเล่มนี้จะใช้คำว่าโทเค็นแทนกลุ่มของอักขระดังกล่าว โทเค็นมีลักษณะและคุณสมบัติคล้ายกับคำในภาษาธรรมชาติ (natural language) เช่น "กิน" "นอน" "นั่ง" เป็นต้น (แต่ละคำจะมีความหมายในตัวเอง) โดยแต่ละโทเค็นจะประกอบด้วยลำดับของอักขระที่จัดเป็นกลุ่ม (เช่นเดียวกับคำว่า "นอน" ประกอบด้วยลำดับของอักขระ "น" "อ" "น" เป็นต้น) โปรแกรมต้นฉบับจะประกอบด้วยโทเค็นหลาย ๆ โทเค็น ซึ่งโทเค็นภายในโปรแกรมอาจจะมีคุณลักษณะที่แตกต่างกัน เช่น โทเค็นที่เป็นคำสงวน หรือคำเฉพาะ เช่น "If" "While" หรือ "Begin" เป็นต้น โดยคำสงวนหรือคำเฉพาะเหล่านี้จะมีขนาดและรูปแบบที่ตายตัว โทเค็นที่เป็นชื่อที่ผู้เขียนโปรแกรมกำหนด (user define) เพื่อใช้งานในโปรแกรม นอกจากนี้ โทเค็นยังสามารถเป็นสัญลักษณ์ต่าง ๆ ที่สามารถใช้ในการเขียนโปรแกรมได้ เช่น "+" "*" โดยในการที่โทเค็นจะสามารถแทนคำสงวน ชื่อ หรือสัญลักษณ์ต่าง ๆ ที่กล่าวมานั้นได้ แต่ละโทเค็นต้องได้รับการตรวจสอบจากตัววิเคราะห์ศัพท์แล้วว่าเป็นโทเค็นที่ประกอบขึ้นจากอักขระ หรือสัญลักษณ์ที่ภาษานั้นอนุญาตให้ใช้ในการเขียนโปรแกรม และมีรูปแบบที่ถูกต้องตาม ข้อกำหนดของภาษาที่ใช้ในการเขียนโปรแกรมต้นฉบับ (Appel, 1998, p. 16) โดยการตรวจจับและจำแนก โทเค็นของโปรแกรมต้นฉบับแสดงในรูป 2.1




**รูป 2.1** การกราดตรวจ และ จำแนกชนิดของโทเค็น

### 2.2.1 หน้าที่การทำงานของตัววิเคราะห์ศัพท์

ขั้นตอนการวิเคราะห์ศัพท์เป็นขั้นตอนที่มีการทำงานเป็นอิสระจากขั้นตอนการทำงานอื่นๆ ของตัวแปลภาษา แต่จะมีการเชื่อมต่อกับตัววิเคราะห์กระจาย หรือ ตัววิเคราะห์วากยสัมพันธ์ (Tremblay & Sorenson, 1985, p. 139) ซึ่งเป็นส่วนหนึ่งของตัวแปลภาษา โดยเมื่อตัววิเคราะห์กระจายต้องการใช้งาน โทเค็นเพื่อใช้ในการทำงาน ตัววิเคราะห์กระจายจะทำการร้องขอโทเค็นจากโปรแกรมการวิเคราะห์ศัพท์ เพื่อให้โปรแกรมวิเคราะห์ศัพท์อ่านสายอักขระ และสร้างโทเค็นส่งกลับไปยังตัววิเคราะห์กระจายใช้ในการทำงานต่อไป (Holub, 1990, p. 33) โดยโครงสร้างการประสานงานระหว่างการวิเคราะห์ศัพท์ กับตัววิเคราะห์กระจายแสดงในรูป 2.2




**รูป 2.2** การเชื่อมต่อระหว่างการวิเคราห์ศัพท์กับตัววิเคราะห์กระจาย

เนื่องจากการทำงานของตัวกราดตรวจเป็นรูปแบบหนึ่งของการตรวจสอบรูปเหมือน (pattern matching) และมีลักษณะของการทำงานเฉพาะตัวซึ่งมีความแตกต่างจากกระบวนการทำงานอื่นของตัวแปลคำสั่งโดยใช้แนวคิดของนิพจน์ปกติ และเครื่องจักรสถานะจำกัด (finite state auttomata) มาประยุกต์ใช้ในการทำงาน (Martin, 1997, p. 69) และการที่มีการออกแบบให้ส่วนของการวิเคราะห์ศัพท์แยกอิสระจากส่วนอื่น ๆ ของคอมไพเลอร์ ซึ่งจะสามารถลดความซับซ้อนในการสร้างคอมไพเลอร์ลงได้ และการแก้ไขปรับปรุงในส่วนของการวิเคราะห์ศัพท์จะไม่ก่อให้เกิดผลกระทบใด ๆ กับการทำงานของคอมไพเลอร์โดยรวม นอกจากนี้คอมไพเลอร์ส่วนใหญ่สามารถที่จะใช้โปรแกรมการวิเคราะห์ศัพท์ที่มีใช้งานทั่วไป เช่น โปรแกรมเล็กซ์ (lex) ให้สอดคล้องกับข้อกำหนดของภาษาที่ตนออกแบบ ก็จะได้โปรแกรมการวิเคราะห์ศัพท์ของคอมไพเลอร์ที่ตนออกแบบได้ โดยไม่จำเป็นต้องเขียนโปรแกรมในส่วนของการวิเคราะห์ศัพท์เอง ทำให้สามารถประหยัดเวลาในการสร้างคอมไพเลอร์ลงได้เป็นอย่างมาก

นอกจากจะทำหน้าที่แยกโทเค็นเพื่อส่งให้ตัววิเคราะห์กระจายนำไปใช้งานแล้ว ตัววิเคราะห์ศัพท์ยังทำหน้าที่พิเศษอื่น ๆ อีก เช่น การขจัดหรือข้ามที่ว่าง และหมายเหตุต่าง ๆ ในโปรแกรม รวมถึงทำหน้าที่นับจำนวนบรรทัดของโปรแกรม เพื่อประโยชน์ในการแจ้งหมายเลขบรรทัดในกรณีที่มีข้อผิดพลาดเกิดขึ้นในโปรแกรมต้นฉบับ (Aho et al, 1986, p. 84 ) โปรแกรมการวิเคราะห์ศัพท์บางตัวอาจจะมีคุณสมบัติพิเศษ อื่น ๆ อีก เช่น การพยายามแก้ไขข้อผิดพลาดที่มีในโปรแกรมต้นฉบับ (ในกรณีที่สามารถทำได้) ซึ่งคุณสมบัติพิเศษต่าง ๆ เหล่านี้อาจจะมีหรือไม่มีก็ได้ แล้วแต่การออกแบบของผู้ออกแบบโปรแกรมการวิเคราะห์ศัพท์

ในคอมไพเลอร์บางตัวอาจจะมีการแยกส่วนของวิเคราะห์ศัพท์ออกเป็นสองส่วน คือ ส่วนของการกราดตรวจและส่วนของการวิเคราะห์ศัพท์ โดยการกราดตรวจจะทำหน้าที่พื้นฐานและการวิเคราะห์ศัพท์จะทำงานที่มีความซับซ้อนมาก ๆ ตัวอย่างเช่น ในคอมไพเลอร์ภาษาใดภาษาหนึ่งอาจจะออกแบบให้มีส่วนของการกราดตรวจเพื่อทำหน้าที่ในการข้ามหมายเหตุ หรือที่ว่าง โดยเฉพาะและงานส่วนที่เหลือซึ่งก็คือ การตรวจจับโทเค็นก็จะเป็นงานของตัววิเคราะห์ศัพท์ ซึ่งจะเป็นการลดภาระงานของตัววิเคราะห์ศัพท์ลงได้

**2.2.1.1 โทเค็นและเลกซีม**

ภาษาคอมพิวเตอร์แต่ละภาษาจะมีการแบ่งชนิดของโทเค็นออกเป็นกลุ่มตามคุณสมบัติของโทเค็น โดยกลุ่มของโทเค็นแต่ละภาษาจะมีจำนวนจำกัด เมื่อมีการกล่าวถึงการทำงานของ ตัววิเคราะห์ศัพท์ จะมองรูปแบบของโทเค็น และเลกซีมในความหมายเฉพาะของแต่ละตัว โทเค็นทางตรรกะต้องมีคุณสมบัติเฉพาะตัว หรือต้องสามารถจำแนกความแตกต่างออกจากโทเค็นอื่น ๆ ได้ เช่น คำสงวน IF ต้องมีคุณสมบัติที่แตกต่างจากสายอักขระ if โดยการที่จะสามารถจำแนกความแตกต่างได้นั้น สายอักขระที่แทนด้วยโทเค็น หรือที่เรียกว่าค่าของสายอักขระ (string value) หรือเลกซีม ซึ่งโทเค็นบางโทเค็นจะมีค่าสายอักขระเพียง รูปแบบเดียว เช่น คำสงวนต่างๆ เป็นต้น แต่สำหรับโทเค็นที่แทนชื่อนั้นอาจจะมีเลกซีมได้มากกว่าหนึ่งรูปแบบที่แทนโดยโทเค็นที่เป็นชื่อ กล่าวคือ สายอักขระที่เป็นชื่อนั้นจะถูกแทนด้วยโทเค็นของชื่อ แต่ข้อมูลภายในสายอักขระที่ถูกแทนดังกล่าวอาจจะแตกต่างกัน โดยในตารางที่ 2.1 จะแสดงให้เห็นความแตกต่างของความหมายของแต่ละคำเมื่อมีการใช้ในการทำงานของตัววิเคราะห์ศัพท์ โดยโทเค็นจะหมายถึงชื่อที่ใช้แทนความหมายของสตริงต่าง ๆ ที่อยู่ในโปรแกรมต้นฉบับ โดยรูปแบบจะเป็นการอธิบายถึงโครงสร้างและรายละเอียดของโทเค็นแต่ละโทเค็น สำหรับเลกซีมนั้นจะแสดงให้เป็นถึงค่าต่าง ๆ ที่เป็นไปได้ของแต่ละโทเค็นดังตัวอย่างคำสั่งในภาษาซี ดังนี้

index = 2*count+17;

ตัววิเคราะห์ศัพท์จะทำการแยกกระทงความนี้ออกเป็นโทเค็นรูปแบบต่างๆ ตามข้อกำหนดของภาษาซี โดยมีรายละเอียดดังตารางที่ 2.1

**ตารางที่ 2.1** ตัวอย่างของโทเค็น รวมถึงรายละเอียดและค่าที่เป็นไปได้

| โทเค็น | เลกซีม | รูปแบบ |
|---|---|---|
| identifier | index | ขึ้นต้นด้วยตัวอักษรแล้วตามด้วยตัวอักษรหรือตัวเลขก็ได้ |
| equal_sign | = | "=" |
| int_constant | 2 | ค่าคงที่ที่เป็นเลขจำนวนเต็ม |
| mul_op | * | "*" |
| identifier | count | ขึ้นต้นด้วยตัวอักษรแล้วตามด้วยตัวอักษรหรือตัวเลขก็ได้ |
| add_op | + | "+" |
| int_constant | 7 | ค่าคงที่ที่เป็นเลขจำนวนเต็ม |
| semicolon | ; | ";" |

โดยปกติแล้วภาษาคอมพิวเตอร์ส่วนใหญ่จะมองโทเค็นในรูปของคำสงวน ตัวกระทำ (operators) ชื่อ ค่าคงที่ ข้อความ (literal) สายอักขระ และ เครื่องหมายวรรคตอน รวมกันเป็นคำว่า "index" ปรากฏขึ้นในโปรแกรมต้นฉบบ ตัววิเคราะห์ศัพท์จะถือว่าเป็นโทเค็นที่เป็นชื่อซึ่งจะส่งต่อชื่อนั้นไปให้กับตัวกระจายคำต่อไป

ดังที่ได้กล่าวมาแล้วข้างต้นว่าตัววิเคราะห์ศัพท์จะทำการแยกโปรแกรมต้นฉบับออกเป็นโทเค็น โดยมีการจัดกลุ่มชนิดของโทเค็นนั้น เป็นกลุ่ม ๆ และจะแทนกลุ่มของโทเค็นด้วยตัวเลข (ทั้งนี้เพื่อความสะดวก และความรวดเร็วในการอ้างถึง) และจะใช้ค่าตัวเลขดังกล่าวในการติดต่อสื่อสารกับส่วนอื่น ๆ ของคอมไพเลอร์ (Tremblay & Sorenson, 1985, p. 422) ตัวอย่าง เช่น กำหนดให้โทเค็นที่เป็นชื่อมีค่าเป็น 1 ค่าคงที่มีค่าเป็น 2 คำสงวนมีค่าเป็น 3 เครื่องหมายเท่ากับมีค่าเป็น 4 เครื่องหมายคูณมีค่าเป็น 5 เครื่องหมายบวกมีค่าเป็น 6 และเครื่องหมายเซมิโคลอนมีค่าเป็น 7 ดังนั้น ข้อมูลที่ตัววิเคราะห์ศัพท์จะส่งไปยังตัววิเคราะห์กระจายก็จะประกอบด้วยหมายเลขของโทเค็น และตำแหน่งในตารางสัญลักษณ์ที่จัดเก็บโทเค็นนั้น ซึ่งหากพิจารณาตัวอย่างคำสั่ง index = 2*count+17; จะ

พบว่าผลลัพธ์ที่เกิดจากการทำงานของตัววิเคราะห์ศัพท์ที่จะส่งไปยังตัววิเคราะห์กระจายจะเป็นหมายเลขของโทเค็น และตำแหน่งในตารางสัญลักษณ์ ดังแสดงในตารางที่ 2.2

**ตารางที่ 2.2** ข้อมูลที่ได้จากการทำงานของตัววิเคราะห์ศัพท์

| โทเค็น | หมายเลข | ตำแหน่งในตารางสัญลักษณ์ |
|---|---|---|
| index | 1 | 1 |
| = | 4 | 0 |
| 2 | 2 | 0 |
| * | 5 | 0 |
| count | 1 | 2 |
| + | 6 | 0 |
| 17 | 22 | 0 |
| ; | 7 | 0 |

จะเห็นว่าข้อมูลที่จัดเก็บในตารางสัญลักษณ์นั้นจะไม่จัดเก็บเครื่องหมายหรือค่าคงที่ ทั้งนี้เนื่องจากเป็นโทเค็นที่จะไม่มีการอ้างอิงเพื่อใช้งานอีก

**2.2.1.2 ลักษณะเฉพาะของโทเค็น**

เมื่อมีรูปแบบที่สามารถใช้แทนเลกซีมได้มากกว่า 1 เลกซีม ตัววิเคราะห์ศัพท์ต้องมีการเตรียมข้อมูลและรายละเอียดอื่น ๆ ที่สามารถทำให้เลกซีมสามารถจับคู่กับรูปแบบที่ถูกต้องได้ โดยตัววิเคราะห์ศัพท์จะทำการรวบรวมข้อมูลและรายละเอียดของแต่ละโทเค็นไว้ในรูปของลักษณะเฉพาะ ซึ่งแต่ละโทเค็นจะมีลักษณะเฉพาะเพียงรูปแบบเดียวเท่านั้น โดยลักษณะเฉพาะของแต่ละโทเค็นจะเป็นข้อมูลในการพิจารณาตัดสินใจในการทำงานของตัวกระจายคำ และทำให้การจับคู่ระหว่างเลกซีมกับรูปแบบเป็นไปได้อย่างถูกต้อง (Wilhelm & Maurer, 1995, p. 236) ในการจัดเก็บรายละเอียดต่าง ๆ ของโทเค็น แต่ละตัวจะทำการจัดเก็บไว้ในตารางสัญลักษณ์ ซึ่งจะมีตัวชี้ที่ชี้ไปยังตารางสัญลักษณ์นั้นเมื่อต้องการนำรายละเอียดดังกล่าวมาใช้งาน ตัวอย่างต่อไปนี้จะแสดงความสัมพันธ์ระหว่างโทเค็นกับลักษณะเฉพาะของแต่ละโทเค็น โดยมีตัวอย่างการทำงานดังนี้

```
index=2*count+17;
```

ในที่นี้จะสามารถแยกโทเค็น และเลกซีม ได้ดังตารางที่ 2.3

**ตารางที่ 2.3** โทเค็น และลักษณะเฉพาะของโทเค็น

| โทเค็น | ลักษณะเฉพาะ | เลกซีม |
|---|---|---|
| identifier | ตัวชี้ที่ชี้ไปยังตารางสัญลักษณ์ตำแหน่งที่เก็บรายละเอียดของชื่อ index | index |
| equal_sign | <equal_sign,> | = |
| int_constant | ค่าจำนวนเต็ม | 2 |
| mul_op | <mul_op,> | * |
| identifier | ตัวชี้ที่ชี้ไปยังตารางสัญลักษณ์ตำแหน่งที่เก็บรายละเอียดของชื่อ count | count |
| add_op | <add_op,> | + |
| int_constant | ค่าจำนวนเต็ม | 7 |
| semicolon | <semicolon,> | ; |

จากตัวอย่างข้างต้นจะเห็นว่าบางโทเค็นนั้นไม่จำเป็นต้องมีลักษณะเฉพาะ ทั้งนี้เนื่องจากโทเค็นเหล่านั้นมีค่าที่แน่นอน (มีรูปแบบเดียวที่ใช้อธิบายและจับคู่กับเลกซีม) วัตถุประสงค์อีกประการหนึ่งของการวิเคราะห์ศัพท์ คือ การที่จะสามารถจำแนกเลกซีมของโทเค็นและการจับคู่ระหว่างโทเค็นแต่ละโทเค็นกับคุณสมบัติเฉพาะของโทเค็นนั้น ๆ ได้อย่างถูกต้อง โดยในตารางที่ 2.4 จะแสดงให้เห็นถึงตัวอย่างค่าคุณสมบัติเฉพาะของโทเค็นรูปแบบต่าง ๆ

**ตารางที่ 2.4** ตัวอย่างค่าคุณสมบัติเฉพาะของโทเค็น

| นิพจน์ปกติ | โทเค็น | คุณสมบัติเฉพาะของโทเค็น |
|---|---|---|
| WS | - | - |
| repeat | repeat | - |
| until | until | - |
| if | if | - |
| then | then | |
| else | else | |
| id | id | ตัวชี้ไปยังตารางสัญลักษณ์ตำแหน่งที่เก็บชื่อ |

**ตารางที่ 2.4** ตัวอย่างค่าคุณสมบัติเฉพาะของโทเค็น (ต่อ)

| นิพจน์ปกติ | โทเค็น | คุณสมบัติเฉพาะของโทเค็น |
|---|---|---|
| num | num | ตัวชี้ไปยังตารางสัญลักษณ์ตำแหน่งที่เก็บตัวแปร |
| < | relational operator | LT |
| <= | relational operator | LE |
| = | relational operator | EQ |
| <> | relational operator | NE |
| > | relational operator | GT |
| >= | relational operator | GE |

### 2.2.2 นิพจน์ปกติสำหรับโทเค็นของภาษาคอมพิวเตอร์

#### 2.2.2.1 ชนิดของโทเค็น

โทเค็นของภาษาคอมพิวเตอร์แบ่งออกเป็นหลายกลุ่ม ซึ่งแต่ละภาษาอาจจะมีการจำแนกกลุ่มที่แตกต่างกันออกไปบ้าง แต่โดยทั่วไปแล้วจะแบ่งออกเป็นกลุ่มดังนี้

##### 2.2.2.1.1 คำสงวน และ ชื่อ

คำสงวน เป็น โทเค็นของสายอักขระที่มีขนาดและรูปแบบที่ตายตัว และมีความหมายเฉพาะ เช่น "if" "while" หรือ "do" เป็นต้น คำสงวนในแต่ละภาษาอาจจะมีการกำหนดที่แตกต่างกัน ทั้งนี้ขึ้นอยู่กับการออกแบบของผู้ออกแบบภาษาคอมพิวเตอร์ภาษานั้น

การเขียนนิพจน์ปกติสำหรับอธิบายคำสงวนนั้นสามารถทำได้ง่ายและสะดวกกว่าโทเค็นแบบอื่น ๆ เนื่องจากเป็นโทเค็นที่มีขนาด และ รูปแบบตายตัว ซึ่งสามารถรวบรวมคำสงวนทั้งหมดในภาษาคอมพิวเตอร์ไว้ด้วยกัน โดยการกำหนดนิพจน์ปกติ ดังนี้

$$\mathrm{Re}\,servedWord = if|while|do|.....$$

สำหรับชื่อที่ผู้เขียนโปรแกรมกำหนดนั้น เป็นกลุ่มของโทเค็นที่ใช้ในการแทนชื่อต่าง ๆ ในการเขียนโปรแกรม เช่น ชื่อตัวแปร หรือชื่อของกระบวนงาน เป็นต้น โดยทั่วไปแล้วภาษาคอมพิวเตอร์มักจะมีข้อกำหนดในการตั้งชื่อให้ขึ้นต้นด้วยตัวอักษรเสมอ

การกำหนดนิพจน์ปกติของชื่อนั้นมีความยุ่งยากซับซ้อนกว่า เนื่องจาก ชื่อมีขนาดของสายอักษรที่ไม่ตายตัว (ขึ้นอยู่กับการกำหนดของผู้เขียนโปรแกรม) แต่อย่างไรก็ตามดังที่ได้กล่าวมาแล้วว่า ชื่อในภาษาคอมพิวเตอร์มักจะกำหนดให้เริ่มต้นด้วยตัวอักษร แล้วตามตัวอักษรหรือตัวเลขจำนวนกี่ตัวก็ได้ (ในภาษาคอมพิวเตอร์บางภาษาอาจจะอนุญาตให้ใช้สัญลักษณ์พิเศษอื่น ๆ ในการกำหนดชื่อได้ เช่น ภาษาซี สามารถกำหนดให้ชื่อเริ่มต้นด้วยขีดล่างได้) ซึ่งจากข้อกำหนดดังกล่าวจะสามารถเขียนเป็นนิพจน์ปกติสำหรับชื่อได้ดังนี้

$$Letter = [a - zA - Z]$$
$$Digit = [0 - 9]$$
$$Identifier = Letter(Letter|Digit)*$$

**2.2.2.1.2 อักขระพิเศษ**

เป็นกลุ่มของโทเค็นที่เป็นสัญลักษณ์ต่าง ๆ ที่อนุญาตให้ใช้ในการเขียนโปรแกรมของแต่ละภาษา เช่น เครื่องหมายทางคณิตศาสตร์ต่าง ๆ อักขระพิเศษเหล่านี้อาจจะอยู่ในรูปของอักขระเดี่ยว เช่น "+" "-" "*" หรือ "/" หรืออาจจะอยู่ในรูปของกลุ่มของสัญลักษณ์ เช่น ":=" หรือ "==" ซึ่งใช้แทนเครื่องหมายเท่ากับในภาษาปาสคาล และภาษาซี ตามลำดับ

**2.2.2.1.3 ค่าคงที่**

เป็นกลุ่มของโทเค็นที่ใช้แทนค่าคงที่ของจำนวน เช่น 24 2.134 หรือ เป็นค่าคงที่ของสายอักขระ เช่น "Computer" หรือแม้แต่เป็นค่าคงที่ของอักขระ เช่น "a" "c" ก็จัดเป็นกลุ่มของโทเค็นที่เป็นค่าคงที่เช่นเดียวกัน

ในกรณีที่โทเค็นนั้นเป็นค่าตัวเลข ซึ่งเป็นลำดับของตัวเลขที่ใช้ในการแสดงค่าจำนวนเต็ม จำนวนจริง ซึ่งอาจจะอยู่ในรูปของเลขฐานสิบ ฐานสอง หรือฐานอื่น ๆ รวมถึงการแสดงค่าของจำนวนในรูปแบบของเอ็กโปเนนท์ เช่น 2.17E-2 ซึ่งมีค่าเท่ากับ 0.0217 โดยในการกำหนดนิพจน์ปกติสำหรับค่าตัวเลขสามารถทำได้ดังนี้

$$Digit = [0 - 9]$$

$$NaturalNumber = Digit +$$

$$SignNaturalNumber = (+|-)?\, NaturalNumber$$

$$Number = SignNaturalNumber("." NaturalNumber)?(E\ SignNaturalNumber)?$$

**2.2.2.1.4 หมายเหตุ**

หมายเหตุทั้งหมดที่มีอยู่ในรูปโปรแกรมต้นฉบับจะถูกขจัดออกไปในขั้นตอนของการวิเคราะห์ศัพท์ จึงจำเป็นต้องมีนิพจน์ปกติสำหรับตรวจสอบหมายเหตุเหล่านั้น เพื่อให้สามารถขจัด ออกไปจากโปรแกรมต้นฉบับได้อย่างถูกต้อง และเช่นเดียวกันกับข้อกำหนดของการตั้งชื่อ และ คำสงวน การกำหนดหมายเหตุในแต่ละภาษาก็มีความแตกต่างกันด้วย แต่โดยส่วนใหญ่มักจะเป็น การเขียนหมายเหตุแบบไร้รูปแบบ (free format) โดยมีสัญลักษณ์ในการกำหนดขอบเขตของข้อความที่เป็นหมายเหตุ ดังตัวอย่างต่อไปนี้

การเขียนหมายเหตุในภาษาปาสคาล {this is Pascal comments}
การเขียนหมายเหตุในภาษาซี /* this is C comments */

การเขียนหมายเหตุด้วยข้อกำหนดลักษณะนี้ สามารถรองรับข้อความที่เป็นหมายเหตุได้มากกว่าหนึ่งบรรทัด โดยจะเขียนหมายเหตุกี่บรรทัดก็ได้ แต่ต้องสิ้นสุดด้วยสัญลักษณ์ที่ บ่งบอกจุดสิ้นสุดของหมายเหตุ

ในบางภาษาอาจจะไม่มีสัญลักษณ์ในการกำหนดขอบเขตของข้อความที่เป็น หมายเหตุ แต่จะใช้สัญลักษณ์ในการเริ่มต้น ที่จะบ่งชี้ว่าข้อความที่ตามมานั้นเป็นหมายเหตุ

การเขียนหมายเหตุในภาษาเอดา -- this is Ada comments

การกำหนดหมายเหตุด้วยวิธีการนี้จะสามารถเขียนหมายเหตุได้คราวละ 1 บรรทัดเท่านั้น หากต้องการเขียนหมายเหตุมากกว่า 1 บรรทัดจะต้องกำหนดสัญลักษณ์ของหมายเหตุใหม่ในบรรทัดนั้น ๆ

การเขียนนิพจน์ปกติสำหรับหมายเหตุที่กำหนดด้วยอักขระเดียว เช่น หมายเหตุในภาษาปาสคาล ทำได้ค่อนข้างง่าย โดยสามารถกำหนดเป็นนิพจน์ปกติได้ดังนี้

{(~})* }

ซึ่งหมายความว่าให้เริ่มต้นด้วยสัญลักษณ์ "{" แล้วตามด้วยอักขระหรือสัญลักษณ์ใด ๆ จำนวนกี่ตัวก็ได้ แต่ต้องไม่ใช่สัญลักษณ์ "}" แล้วสิ้นสุดด้วยสัญลักษณ์ "}" สำหรับหมายเหตุที่กำหนดคราวละบรรทัด สามารถกำหนดด้วยนิพจน์ปกติดังนี้

-- (~newline)*

โดย newline หมายถึงการขึ้นบรรทัดใหม่

หมายเหตุที่กำหนดด้วยอักขระมากกว่าหนึ่งตัวจะมีความยุ่งยากในการจัดการ มากขึ้น โดยเพื่อให้มองเห็นปัญหาที่เกิดขึ้นได้ชัดเจน ให้พิจารณาตัวอย่างของสายอักขระ ba...ab (การใช้ สายอักขระ ba...ab ประกอบในตัวอย่างนี้ ก็เพื่อให้สอดคล้องกับการเขียนหมายเหตุในภาษาซี ซึ่งมีรูปแบบเป็น /*...*/) จะสามารถกำหนดด้วยนิพจน์ปกติดังนี้

ba((~ab))*ab

แต่จากที่ได้ศึกษามาแล้วว่าการกระทำแบบยกเว้นนั้นจะมีผลกับอักขระเพียงตัวเดียวเท่านั้น ดังนั้นจึงจำเป็นต้องปรับปรุงนิพจน์นี้ ซึ่งอาจจะเขียนใหม่เป็น

ba(~a~b)*ab

หรือ ba(~(a|b))*ab

หรือ b*(a*~(a|b)b*)*a*

นอกจากนี้แล้ว ภาษาคอมพิวเตอร์บางภาษายังมีรูปแบบการกำหนดหมายเหตุที่ซับซ้อนขึ้น เช่น การกำหนดหมายเหตุในภาษาโมดูลล่า-2 ซึ่งอนุญาตให้กำหนดหมายเหตุซ้อนหมายเหตุ (nested comments) ได้ ดังนี้

(* this is (* a Modula-2 *)comments *)

ซึ่งในการตรวจสอบหมายเหตุในลักษณ์จะต้องมีการจำคู่ของสัญลักษณ์ที่กำหนดขอบเขตของหมายเหตุด้วย

**2.2.2.2 พื้นที่ว่าง**

วิธีการในการพิจารณาสายอักขระที่มีขนาดยาวที่สุดนี้ ทำให้โดยการค้นหาอักขระที่บ่งบอกจุดสิ้นสุดของสายอักขระที่สามารถแยกเป็นโทเค็นได้ โดยพิจารณาจากอักขระที่ไม่สามารถประกอบหรือเป็นสมาชิกของโทเค็นนั้นได้ เช่น สายอักขระ xtemp = ytemp ซึ่งจะเห็นว่าเครื่องหมาย "=" เป็นเครื่องหมายที่ไม่สามารถประกอบ หรือรวมเป็นส่วนหนึ่งของโทเค็นที่ชื่อ xtemp ได้ (ตามข้อกำหนดของการตั้งชื่อ) ดังนั้นจะถือว่าโทเค็นจะสิ้นสุดที่อักขระก่อนหน้าเครื่องหมาย "=" นอกจากนี้แล้ว ที่ว่าง (การเว้นวรรค) การขึ้นบรรทัดใหม่ และย่อหน้า ก็เป็นโทเค็นในกลุ่มของสัญลักษณ์บ่งบอกจุดสิ้นสุดของโทเค็นเช่นเดียวกัน เช่น คำสั่ง while x ..... จะประกอบด้วยสองโทเค็นคือ คำสงวน "while" และชื่อ "x" โดยที่ว่างระหว่าง "while" และ "x" เป็นสัญลักษณ์บ่งบอกจุดสิ้นสุดของโทเค็น "while" นอกจากนี้แล้วส่วนของการเขียนหมายเหตุ ก็ถือว่าเป็นส่วนหนึ่งของการกำหนดจุดสิ้นสุดของโทเค็นได้เช่นเดียวกัน เช่น ตัวอย่างในภาษาซีต่อไปนี้

do /*...*/ if

การวิเคราะห์ศัพท์จะถือว่าคำสั่งนี้ประกอบด้วยโทเค็นสองโทเค็น คือ "do" และ "if" (โดยส่วนของหมายเหตุเป็นตัวบ่งบอกจุดสิ้นสุดของโทเค็น) และเพื่อความสะดวกในการอ้างและการใช้งานภาษาคอมพิวเตอร์โดยทั่วไป มักจะจัดกลุ่มของอักขระเหล่านี้ให้อยู่ในกลุ่มเดียวกัน เรียกว่า โทเค็นที่เป็นพื้นที่ว่าง (white space) โดยสามารถกำหนดในรูปของนิพจน์ปกติได้ดังนี้

$$whitespace = (newline|blank|tab|comment)+$$

ในระหว่างที่มีการวิเคราะห์ศัพท์ พื้นที่ว่างในโปรแกรมทั้งหมดจะถูกขจัดออกไป (เนื่องจากเป็นโทเค็นที่ไม่มีความหมาย หรือไม่มีประโยชน์ในการทำงานใด ๆ ในขั้นตอนอื่น ๆ ในการทำงานของคอมไพเลอร์)

**2.2.2.3 การตรวจสอบรูปแบบของสายอักขระ**

ดังที่ได้กล่าวมาแล้วข้างต้นว่านิพจน์ปกติเป็นข้อกำหนดของรูปแบบของสายอักขระที่ต้องการตรวจสอบ โดยในการตรวจสอบรูปแบบของสายอักขระดังกล่าวนั้น จะทำการแยกสายอักขระออกเป็นกลุ่ม ๆ โดยมีรูปแบบตามที่กำหนดในนิพจน์ปกติ เช่น กำหนดให้ภาษาคอมพิวเตอร์ภาษาหนึ่ง มีข้อกำหนดของไวยากรณ์เพียงข้อกำหนดเดียว ดังนี้

(number + number)

จากข้อกำหนดของไวยากรณ์ดังกล่าวจะต้องมีการกำหนดนิพจน์ปกติเพื่อใช้ในการตรวจสอบสายอักขระที่เป็นองค์ประกอบของไวยากรณ์ ดังนี้

| โทเค็น | นิพจน์ปกติ |
|---|---|
| ( | ( |
| ) | ) |
| + | + |
| number | [0-9]+ |

โดยในการทำงานของตัววิเคราะห์ศัพท์เพื่อทำการแยกโทเค็นนี้จะอาศัยการกำหนดสถานะเพื่อตรวจสอบรูปแบบที่คาดหวังของสายอักขระที่กำลังตรวจสอบ โดยจากตัวอย่างข้างต้นจะพบว่าสำหรับโทเค็น "(" ")" และ "+" นั้นจะสามารถตรวจสอบได้ง่ายทั้งนี้เนื่องจากเป็นโทเค็นที่มีสัญลักษณ์เพียงตัวเดียว และเมื่ออ่านพบข้อมูลนำเข้าเป็นสัญลักษณ์ดังกล่าวก็จะสามารถบ่งชี้ได้ว่าเป็นโทเค็นใด แต่สำหรับโทเค็นที่เป็น number นั้นจำเป็นต้องมีการตรวจสอบองค์ประกอบของโทเค็นทั้งหมด (เนื่องจากอาจจะประกอบด้วยสัญลักษณ์มากกว่า 1 ตัว) ซึ่งในการตรวจสอบลักษณะดังกล่าวว่าเป็นต้องอาศัยการกำหนดสถานะช่วยในการตรวจสอบ โดยเมื่อมีการอ่านข้อมูลนำเข้าตัวแรก (ซึ่งเป็นตัวเลข) ตัววิเคราะห์ศัพท์จะย้ายสถานะของการตรวจสอบไปยังตัวเลขดังกล่าวจะถูกนำไปรวมกับข้อมูลตัวก่อนหน้า แล้วทำการอ่านข้อมูลตัวถัดไป ในกรณีที่ข้อมูลตัวถัดไปไม่ใช่ตัวเลขตัววิเคราะห์ศัพท์ก็จะออกจากสถานะของการตรวจสอบตัวเลข และจะรู้ได้ทันทีว่าข้อมูลที่มีอยู่เดิมในขณะนั้นเป็นโทเค็นที่เป็นค่าตัวเลข และส่งคืนค่าโทเค็นและชนิดของโทเค็น กลับไปยังผู้เรียกใช้ (ตัววิเคราะห์วากยสัมพันธ์) ต่อจากนั้นก็จะทำการตรวจสอบข้อมูลนำเข้าตัวต่อไป ตัวอย่างของการตรวจสอบโทเค็นสำหรับไวยากรณ์ข้างต้น ในกรณีที่มีข้อมูลนำเข้าเป็น (50+12) จะมีสถานะของการตรวจสอบ ดังแสดงในตารางที่ 2.5

**ตารางที่ 2.5** การตรวจสอบโทเค็นของ (50+12)

| ข้อมูลนำเข้า | สถานะการตรวจสอบ | การทำงาน |
|---|---|---|
| ( | สถานะเริ่มต้น | ส่งค่าโทเค็น "(" กลับไปยังผู้เรียกใช้ |
| 5 | สถานะเริ่มต้น | จัดเก็บค่า 5 และเข้าไปทำงานในสถานะตรวจสอบตัวเลข |
| 0 | ตรวจสอบตัวเลข | จัดเก็บค่า 0 |
| + | ตรวจสอบตัวเลข | พบเครื่องหมายบวก ออกจากสถานะตรวจสอบตัวเลข<br>ส่งค่าโทเค็น 50 กลับไปยังผู้เรียกใช้ |
| + | สถานะเริ่มต้น | ส่งค่าโทเค็น "+" กลับไปยังผู้เรียกใช้ |
| 1 | สถานะเริ่มต้น | จัดเก็บค่า 1 และเข้าไปทำงานในสถานะตรวจสอบตัวเลข |
| 2 | ตรวจสอบตัวเลข | จับเก็บค่า 2 |
| ) | ตรวจสอบตัวเลข | พบเครื่องหมาย ")" ออกจากสถานะตรวจสอบตัวเลข<br>ส่งค่าโทเค็น 12 กลับไปยังผู้เรียกใช้ |
| ) | สถานะเริ่มต้น | ส่งค่าโทเค็น ")" กลับไปยังผู้เรียกใช้ |

### 2.2.3 ข้อกำหนดในการตั้งกฎของไวยากรณ์ไร้บริบท

เช่นเดียวกันกับนิพจน์ปกติ การกำหนดไวยากรณ์ไร้บริบทประกอบด้วยลำดับของอักขระ หรือ เซตของสัญลักษณ์ ซึ่งในนิพจน์ปกติแล้วสัญลักษณ์เหล่านี้ก็คืออักขระนั่นเอง แต่สำหรับกฎไวยากรณ์แล้วสัญลักษณ์ที่ใช้กำหนดจะเป็นสายอักขระที่ใช้แทนชื่อขององค์ประกอบในไวยากรณ์นั้น กฎของไวยากรณ์ของ BNF ประกอบด้วย สัญลักษณ์ (สายอักขระ) โดยสัญลักษณ์แรกหมายถึงชื่อของกฎไวยากรณ์ แล้วตามด้วยเครื่องหมายลูกศร (บ่งบอกการอธิบายไวยากรณ์) และส่วนของการอธิบายโครงสร้างของไวยากรณ์ ดังตัวอย่างต่อไปนี้

$$< \exp r > \rightarrow < \exp r >< \exp r > \mid (< \exp r >) \mid < number >$$
$$< op > \rightarrow + \mid - \mid * \mid /$$

นอกจากจะใช้เครื่องหมายลูกศรในการกำหนดโครงสร้างของกฎแล้วยังสามารถใช้เครื่องหมายอื่น ๆ ในการกกำหนดได้เช่นเดียวกัน เช่น

$$< \exp r >::=< \exp r >< op >> \exp r > \mid (< \exp r >) \mid < number >$$
$$< op >::= + \mid - \mid * \mid /$$

นอกจากนี้อาจจะมีการยุบรวมกฎหลาย ๆ กฎให้เป็นกฎเดียวกันเพื่อความกระชับในการกำหนด แต่อาจจะยุ่งยากในการอ่าน หรือทำความเข้าใจโครงสร้างหรือกฎของไวยากรณ์นั้น เช่น

$$< \exp r > \rightarrow < \exp r > ("+" \mid "-" \mid "*" \mid "/") < \exp r > \mid "(" < \exp r > ")" \mid < number >$$

### 2.2.4 ลำดับการแปลงและการกำหนดภาษาโดยการใช้ไวยากรณ์

บีเอ็นเอฟเป็นเครื่องมือในการอธิบายข้อกำหนดของภาษา โดยประโยคของภาษานั้นถูกสร้างขึ้นโดยอาศัยกฎของภาษา ซึ่งจะมีสัญลักษณ์พิเศษที่ใช้ในการกำหนดจุดเริ่มต้นของกฎเรียกว่าสัญลักษณ์เริ่มต้น (start symbol) และเมื่อมีการสร้างประโยคโดยอาศัยกฎของภาษาเราเรียกขั้นตอนของการสร้างประโยคนั้นว่าลำดับการแปลง (derivation)(Aho, Sethi & Ullman, 1986, p. 167) ไวยากรณ์ที่สมบูรณ์ของภาษาคอมพิวเตอร์ จะมีสัญลักษณ์เริ่มต้นที่กำหนดโครงสร้างโดยสมบูรณ์ของโปรแกรมที่เขียนโดยภาษานั้น ซึ่งสัญลักษณ์เริ่มต้นนี้ส่วนใหญ่มักจะกำหนดให้เป็นคำว่า “program” ดังตัวอย่างต่อไปนี้

$$< program > \rightarrow \textbf{begin} < stmt\_list > \textbf{end}$$
$$< \text{stmt\_list} > \rightarrow < \text{stmt} > \mid < \text{stmt} >; < \text{stmt\_list} >$$
$$< stmt > \rightarrow < \text{var} <:= < \exp r >$$
$$< \text{var} > \rightarrow A \mid B \mid C$$
$$< \exp r > \rightarrow < \text{var} > + < \text{var} > \mid < \text{var} > - < \text{var} > \mid < \text{var} >$$

จากตัวอย่างกฎของภาษาข้างต้น ในการที่จะตรวจสอบความถูกต้องของภาษา หรือความถูกต้องของโทเค็นที่ใช้ในการเขียนโปรแกรมด้วยภาษานั้น สามารถทำการตรวจสอบได้โดยอาศัยหลักของลำดับการแปลง กล่าวคือ เป็นการตรวจสอบองค์ประกอบของกฎว่าถูกต้องตามข้อกำหนดหรือไม่ เช่น จากตัวอย่างกฎของภาษาข้างต้น หากต้องการตรวจสอบว่าโปรแกรมที่เขียนขึ้นด้วยข้อกำหนดของภาษาข้างต้น ดังรูป 2.3

```
Begin
    A:=A+B;
    B:=C
End
```

**รูป 2.3** ตัวอย่างโปรแกรมต้นฉบับของภาษาคอมพิวเตอร์

## 2.3 สมการทางคณิตศาสตร์ของเรขาคณิตวิเคราะห์

### 2.3.1 สมการเส้นตรง

เช่น

$$y = mx + b, Ax + By + C = 0$$

### 2.3.2 สมการวงกลม

เช่น

$$(x-h)^2 + (y-k)^2 = r^2$$

### 2.3.3 สมการวงรี

เช่น

$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$$

### 2.3.4 สมการพาราโบลา

เช่น

$$x = y^2 + ay + bx + c = 0; b \neq 0, y = x^2 + bx + cy = 0; b \neq 0$$

### 2.3.5 สมการไฮเพอร์โบลา

เช่น

$$\frac{(x-h)^2}{a^2} - \frac{(y-k)^2}{b^2} = 1$$

**2.3.6 สมการตรีโกณมิติของ sin(x)**

เช่น

$$y = \sin x^2$$

**2.3.7 สมการตรีโกณมิติของ cos(x)**

เช่น

$$y = \cos(2x)$$

**2.3.8 สมการตรีโกณมิติของ tan(x)**

เช่น

$$y = \tan(x)$$

**2.3.9 สมการตรีโกณมิติของ arcsin(x)**

เช่น

$$y = \arcsin(3x)$$

**2.3.10 สมการตรีโกณมิติของ arccos(x)**

เช่น

$$y = \arccos(x+2)$$

**2.3.11 สมการตรีโกณมิติของ arctan(x)**

เช่น

$$y = \arctan(3x-1)$$

**2.3.12 สมการตรีโกณมิติของ ln(x)**

เช่น

$$y = \ln(x^2+1)$$

### 2.3.13 สมการกำลังสอง $(x^2)$

เช่น

$$y = 3x^2 + 2$$

### 2.3.14 สมการกำลังมากกว่าสอง $(x^3)$

เช่น

$$y = \left(\frac{x}{2}\right)^3 - 3$$

### 2.3.15 การประยุกต์ของสมการ

เช่น

$$y = \sin^2 x + 4\sin x + 3$$

## 2.4 VRML

### 2.4.1 VRML คือ อะไร?

VRML ย่อมาจาก Virtual Reality Modeling Language คือ ภาษาคอมพิวเตอร์ที่ใช้สร้างแบบจำลองสามมิติเป็นภาพเคลื่อนไหว ที่สามารถตอบโต้ผู้ใช้ได้ในโลกเสมือน ( Virtual World ) ที่ถูกสร้างขึ้น ผ่านระบบเครือข่ายอินเตอร์เน็ต ซึ่งแต่ละโลกเสมือนสามารถเชื่อมโยง ( Hyperlink ) ถึงกันและกันได้โดย WWW ( World Wide Web ) สามารถพูดได้ว่า VRML คือโฮมเพจสามมิติ ( 3D-Homepage ) ส่วน HTML ก็คือโฮมเพจสองมิติ ( 2D-Homepage )

VRML กับการสร้างโลกเสมือนนี้ ภายในโลกเสมือนมีจุดเชื่อมต่อ ( Hyperlink ) ไปยังโลกเสมือนอื่นหรืออาจจะถูกเชื่อมไปยังโฮมเพจที่เป็นเอกสาร HTML ธรรมดาหรืออาจจะเป็นเอกสารชนิดอื่นๆ ที่มีชนิด MIME ( Multipurpose Internet Mail Extension ) ที่ browser ( ผู้ขอรับบริการ ) และ server ( ผู้ให้บริการ ) รู้จักทั้งคู่ โดย MIME คือวิธีการส่งข้อมูลในรูปแแบของแฟ้มข้อมูลชนิดต่างๆ ซึ่งทราบได้จากนามสกุลของแฟ้มข้อมูล ( Filename Extension ) ชนิดนั้น ๆ เช่น .html, .txt, .jpg, .au, .mov, .zip เป็นต้น ผ่านทางระบบจดหมายอิเล็กทรอนิกส์ ( E-mail ) จาก server ไปยัง browser โดยที่ทั้งสองฝ่ายต้องทราบว่าข้อมูลเป็นชนิดใด และจะทำอะไรกับข้อมูลชนิดนี้ดี สำหรับ VRML ใช้นามสกุล .wrl ซึ่งย่อมาจาก world และเป็น MIME ชนิด x-world/x-vrml เมื่อ server ได้รับการเรียกขอบริการแฟ้มข้อมูล .wrl จาก browser ที่ตัว server ก็จะตรวจดูว่าเป็น MIME ชนิดใดจากนามสกุลของแฟ้มข้อมูล เมื่อ server ทราบว่าเป็นชนิด x-world/x-vrml จากนั้นก็ส่งข้อมูลของแฟ้มข้อมูลนี้ไปยัง browser

พร้อมทั้งบอกชนิด MIME ให้แก่ browser ให้ทราบด้วย เมื่อ browser ได้รับแฟ้มข้อมูลแล้ว ก็ทราบว่าเป็นแฟ้มข้อมูล VRML และทำการแสดงภาพ ( Rendering ) สามมิติ ของแฟ้มข้อมูลนี้ ซึ่งภาพสามมิตินี้เราสามารถใช้เมาส์และคีย์บอร์ดควบคุมทิศทางของการมองของเราเสมือนกับว่าเรากำลังเดินเข้าไปในโลกหนึ่ง เราสามารถ เดินตรง ถอยหลัง เลี้ยวซ้าย เลี้ยวขวา หมุนภาพ เลื่อนขึ้น เลื่อนลงได้

ภาษาที่ใช้เขียนจะอยู่ในรูปของไฟล์ ASCII ไม่ขึ้นกับแพลตฟอร์มหรือ เครื่องคอมพิวเตอร์ที่ใช้ เพราะ ASCII CODE เป็นรหัสพื้นฐานที่ไม่ว่าคอมพิวเตอร์รุ่นใด หรือยี่ห้อไหน ก็ตามจะใช้แบบเดียวกันนี้ทั้งหมด และเนื่องจากไฟล์ VRML นี้ใช้สำหรับแสดงรายละเอียดของข้อมูลแบบสามมิติ ดังนั้นขนาดของไฟล์จึงอาจมีขนาดใหญ่ ในปัจจุบันมีมาตรฐานการบีบอัดข้อมูลของไฟล์ VRML ก่อนทำการส่ง อยู่แล้ว (GZIP) โดยนามสกุลที่ได้หลังจากการย่อขนาดก็คือ .wrl.gz

### 2.4.2 เครื่องคอมพิวเตอร์ที่เหมาะสม

เครื่องคอมพิวเตอร์คือสิ่งแรกที่เราต้องมี เพื่อจะสร้างงานสามมิติโดยภาษา VRML จากคุณสมบัติข้อหนึ่งของ VRML คือ VRML จะไม่ขึ้นกับชนิดของเครื่องคอมพิวเตอร์ หรือแพลตฟอร์ม เพราะใช้รหัสพื้นฐาน ( Source Code ) ในรูปแบบเดียวกันหมด สามารถทำงานได้ทุกเครื่องที่ VRML - Browser สามารถทำงานได้ อย่างไรก็ตามที่เครื่องคอมพิวเตอร์จะสามารถสร้างภาพจำลองสามมิติเคลื่อนไหวได้อย่างมีประสิทธิภาพดีเยี่ยม ย่อมต้องการเครื่องคอมพิวเตอร์ที่มีประสิทธิภาพสูง

— เครื่องพีซี ( IBM Compatible ) ความเร็วของซีพียู อย่างต่ำควรจะเป็นเพนเทียม หน่วยความจำอย่างน้อย 8 เมกกะไบต์ขึ้นไป แต่ควรจะมี 16 เมกะไบต์จะดีกว่า และระบบปฏิบัติการใช้ Microsoft Windows 3.11 หรือ Microsoft Windows 95 / NT ก็ได้

— เครื่องแมคอินทอช ควรจะเป็นระดับ System 7 ขึ้นไป แรมอย่างต่ำ 8 เมกะไบต์

### 2.4.3 ข้อตกลงของภาษา VRML

ก่อนที่จะเขียนโปรแกรมได้จำเป็นจะต้องรู้ข้อตกลงของภาษา VRML เสียก่อน

#### 2.4.3.1 หน่วยที่ใช้

VRML จะใช้หน่วยวัดดังนี้

— ระยะห่าง และ ขนาด : เมตร

— มุม : เรเดียน

— ค่าอื่น ๆ : เปอร์เซ็นต์ ( เปรียบเทียบจาก 1 )

**2.4.3.2 ส่วนหัวของโปรแกรม**

VRML บราวเซอร์ มีความสามารถที่จะรู้ว่าเป็น VRML โค้ดได้โดยตรวจสอบที่ ไฟล์เฮดเดอร์ ( file header ) ที่อยู่ตอนต้นของไฟล์

#VRML V2.0 utf8

ไฟล์เฮดเดอร์จำเป็นที่จะต้องอยู่ที่บรรทัดแรกเพื่อที่จะแปลได้ถูกต้อง แต่เมื่ออยู่อื่น pound symbol (#) จะหมายถึง คอมเมนต์ ( comment ) คำว่า utf8 จะใช้สำหรับอ้างอิงรูปแบบของตัวอักษรตามมาตรฐาน ISO หรือที่รู้จักกันดีในเรื่องของการเข้ารหัสตัวอักษรแบบ UTF-8 สำหรับผู้ที่เคยเขียนโปรแกรมด้วยภาษา VRML เวอร์ชัน 1.0 มาก่อนแล้ว สามารถนำไฟล์เก่ามาแปลงเป็นเวอร์ชัน 2.0 ได้ด้วย โดยการนำเอาไฟล์ที่เขียนในรูปแบบของเวอร์ชัน 1.0 มาแปลงกลับด้วยโปรแกรม " vrml1to2.exe " ไฟล์ตัวนี้ได้ถูกบรรจุรวมอยู่กับโปรแกรม Cosmo Player 2.0

นอกจากนี้ เวลาเขียนโปรแกรม จะต้องระมัดระวังการเขียนตัวอักษรตัวเล็ก กับตัวอักษรด้วยว่ามีความแตกต่างกัน

**2.4.3.3 ฟิลด์**

แบ่งออกได้ 2 ประเภท คือ ฟิลด์ที่มีได้เพียงหนึ่งค่า และฟิลด์ที่มีได้หลายค่า ฟิลด์ที่มีค่าเดียว จะมีชื่อขึ้นต้น “SF” ฟิลด์ที่มีหลายค่า จะมีชื่อขึ้นต้นด้วย “MF”ฟิลด์หลายค่าจะเขียนเป็นลำดับของค่าโดยแบ่งด้วยเครื่องหมายคอมม่า “,” ล้อมรอบด้วยเหลี่ยมปีกกา “[ ]” ถ้าฟิลด์ใดไม่มีค่าจะเขียนเพียงเหลี่ยมปีกกาเท่านั้น ( “[ ]“ ) ส่วนที่เหลืออาจจะตามด้วยคอมม่าก็ได้ ตัวอย่างเช่น ทั้งหมดนี้จะหมายถึง ฟิลด์หลายค่าที่มีค่าจำนวนเต็มเท่ากับ 1 เหมือนกัน

1 [1,] [ 1 ]

**2.4.3.4 โหนด**

โหนดมีลักษณะต่าง ๆ ดังนี้

— ชนิดของวัตถุ โหนดอาจเป็นทรงกลม, สี่เหลี่ยม, พื้นผิว หรือ อื่น ๆ

— พารามิเตอร์ ใช้จำแนกความแตกต่างของแต่ละโหนด โหนดที่มีชนิดเดียวกัน อาจจะมีคุณสมบัติแตกต่างกันไป เช่น โหนดทรงกลมอาจจะมีรัศมีต่างกัน ตำแหน่งต่างกัน พารามิเตอร์เหล่านี้เรียกว่า ฟิลด์ ( field ) โหนดนึงอาจจะมีศูนย์หรือหลายฟิลด์ก็ได้

– ชื่อที่บ่งถึงโหนด เราสามารถตั้งชื่อให้กับโหนดใด ๆ ก็ได้ และอ้างถึงโหนดนั้น ๆ ได้จากที่ใด ๆ มีประโยชน์อย่างมากโดยเฉพาะการใส่สคริปต์ โหนดไม่จำเป็นต้องมีชื่อก็ได้ แต่ถ้ามีชื่อ จะมีได้เพียงชื่อเดียวเท่านั้น อย่างไรก็ตาม เราสามารถตั้งชื่อซ้ำกันได้

– โหนดลูก การวางลำดับชั้นของวัตถุ อนุญาตให้บางโหนดเป็นส่วนประกอบของโหนดอื่น โหนดที่จะมีโหนดลูก จะเรียกโหนดนั้นว่า โหนดกลุ่ม ( group nodes ) โหนดกลุ่มสามารถมีโหนดลูกศูนย์หรือหลายโหนดก็ได้

กฎเกณฑ์การเขียนโค้ด จะมีลำดับดังนี้

DEF objectname objecttype { fields children }

เฉพาะออบเจ็กต์ไทป์ และ ปีกกาเท่านั้นที่จำเป็นต้องมี โหนดอาจจะมีหรือไม่มีชื่อ ฟิลด์ หรือ โหนดลูกก็ได้ ชื่อโหนดต้องไม่เริ่มต้นด้วยตัวเลข และต้องไม่มีเว้นวรรค เครื่องหมายคำพูด อักขระควบคุมต่าง ๆ ปีกกา อยู่ในชื่อ

**2.4.3.5 Events**

อีเวนต์ เป็นคุณสมบัติของฟิลด์ ที่สามารถรับค่าจากภายนอก หรือเปลี่ยนค่าใหม่ตามที่เราต้องการ มีประโยชน์ในการเขียนสคริปต์ หรือทำแอนิเมชัน

***รู้จักกับอินเตอร์เฟสของฟิลด์***

คอลัมน์แรกที่อยู่ในข้อกำหนดของโหนดต่าง ๆ จะเรียกว่า Interface type ใช้ควบคุมการเข้าถึงจากโหนดอื่น แบ่งเป็นประเภทต่าง ๆ ดังนี้

– Field ไม่สามารถเขียน หรืออ่านได้

– EventIn สามารถเขียนได้ โดยส่งค่าเข้าไป แต่ไม่สามารถอ่านออกมาได้

– EventOut สามารถอ่านได้ แต่ไม่สามารถแก้ไขได้

– ExposedField สามารถเขียน และอ่านได้

***การส่งค่า***

เราสามารถส่งค่าจากโหนดหนึ่งไปยังโหนดอื่นที่ต้องการได้โดยใช้คำสั่ง ROUTE มีหลักไวยากรณ์ดังนี้

ROUTE node1.eventOutName_changed TO node2.set_eventInName

ชื่อโหนด จะได้มาจากการตั้งชื่อโดยคำว่า " DEF " เราสามารถเชื่อมได้เพียงอ่านค่าจาก eventOut ไปยัง eventIn และฟิลด์นั้นต้องมีชนิดเดียวกันด้วย สำหรับ exposedField เป็นได้ทั้ง evnetIn และ EventOut

**ตัวอย่าง**

```
ROUTE time.fraction_changed TO position.set_fraction
ROUTE position.keyValue_changed TO transform.translation
ROUTE touch.isActive TO time.enabled
```

**2.4.3.6 การเขียน Script**

การใส่สคริปต์เข้าไปจะช่วยเพิ่มความสามารถในการควบคุม เพิ่มลูกเล่น สร้างความน่าสนใจให้กับแบบจำลองของเราได้มาก การเขียนสคริปต์ก็คล้ายกับการเขียนโปรแกรมเข้าไป มีการกำหนดตัวแปร สร้างฟังก์ชัน ส่งพารามิเตอร์ เราสามารถใช้ภาษาสคริปต์ได้เหมือนที่เราใช้ประกอบกับเอกสารโฮมเพจ เช่น CGI Script, JavaScript รวมไปถึงสามารถใช้ภาษา Java ก็ได้

การนำ CGI Script เข้าไปใช้กับ VRML โดยใส่ไว้ใน Anchor Node ดังตัวอย่างนี้

```
Anchor {
url "/cgi-bin/myworld.cgi LENGTH=50 WIDTH=35"
children [ # child list here ]
}
```

ปัญหาที่สำคัญของการใช้ CGI ก็คือ จะเป็นการแทนที่โลกที่ถูกสร้างขึ้นมาใหม่ ไม่ได้เป็นการเปลี่ยนแปลงบางโหนด เช่น ต้องการให้ผู้ใช้กดปุ่ม แล้วกล่องสี่เหลี่ยมเปลี่ยนสี กรณีนี้ จะเป็นการสร้างโลกขึ้นมาใหม่แทนที่โลกเก่าเดิม

ด้วยเหตุนี้ นักออกแบบจึงนิยมใช้ภาษาสคริปต์อื่น ๆ คือ JavaScript และ Java การนำ สคริปต์ไปใช้ เราจะเก็บมันไว้ในโหนด Script ดังนี้

```
Script {
exposedField MFString url []
field SFBool directOutputs FALSE
field SFBool mustEvaluate FALSE
# And any number of:
eventIn eventTypeName eventName
field fieldTypeName fieldname initialValue
evnetOut eventTypeName eventName
}
```

สำหรับรายละเอียดของข้อกำหนดที่ใช้กับ JavaScript และ Java จะมีความแตกต่างกันบ้างพอสมควร

**2.4.3.7 JavaScript**

ภาษา JavaScript สร้างโดย Netscape Communication Corporation รายละเอียดของภาษา สามารถพบได้ที่

http://home.netscape.com/comprod/products/navigator/version_2.0/script/script_info/

การนำ JavaScript ไปใช้กับ VRML โดยการใส่ชื่อไฟล์ไปไว้ในฟิลด์ url ของโหนด Script โดยอาจเขียนเป็นโปรแกรมแยกออกมาต่างหาก มีนามสกุลเป็น " .js " หรือ เขียนเป็นฟังก์ชันอยู่ภายในโหนดก็ได้ ตัวอย่างเช่น

```
Script { url "http://foo.com/myScript.js" }
Script { url "JavaScript: function foo() { ... }" }
Script {
url [ "http://foo.com/myScript.js",
"javascript: function foo() { ... }" ]
}
```

อีเวนต์จะถูกส่งไปในสคริปต์ โดยที่ชื่อฟังก์ชันจะต้องเป็นชื่อเดียวกับชื่อของ EventIn และต้องมีสองอากิวเมนต์คือ value และ timestamp ตัวอย่างเช่น เรามีฟิลด์ EventIn หนึ่งตัวชื่อ " start "

```
Script {
eventIn SFBool start
url "javascript: function start(value, timestamp) { ... }
}
```

จากตัวอย่าง เมื่อ start EventIn ถูกส่งมา ฟังก์ชัน start ก็จะทำงานเราสามารถใส่คำสั่ง ROUTE เข้าไปเพื่อให้เกิดการส่งอีเวนต์ต่อกันไปได้ ดังตัวอย่าง

```
DEF toggle Script {
Field SFBool active FALSE
EventIn SFBool isClicked
EventIn SFBool click_changed
url { ... }
}
ROUTE touch.isActive TO toggle.isClicked
ROUTE toggle.click_changed TO time.enabled
```

**2.4.3.8 Java**

ภาษา Java เป็นภาษาแบบ Object-Oriented คล้ายภาษา C++ สามารถทำงานข้ามแพลตฟอล์มได้ ถูกพัฒนาโดย Sun Microsystems, Inc รายละเอียดของภาษาค้นหาได้ที่ http://java.sun.com/

การนำ Java ไปใช้ โดยการใส่ชื่อคลาสของไฟล์ไปไว้ในฟิลด์ url ในโหนด Script มีนามสกุลเป็น " .class " และ ชื่อไฟล์จะต้องมีชื่อเดียวกับชื่อคลาสในโปรแกรมนั้น การรับค่าไว้ใน EventIn ทำเหมือนกับ JavaScript คือ เอาชื่ออีเวนต์เป็นชื่อเดียวกับฟังก์ชันในโปรแกรมตัวอย่างเช่น โหนดสคริปต์ มีหนึ่งอีเวนต์ ชื่อ start

```
Script {
url "http://foo.co.jp/Example.class"
eventIn SFBool start
}
```

โปรแกรมของ Example.java อาจจะเป็น

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;
class Example extends Script {
…
// This method is callled when any event is received
public void processEvent ( Event e ) {
// … perform some operation …
private void start( ConstantSFBool value ) {
…
}
}
```

เช่นเดียวกับ JavaScript เราสามารถใส่คำสั่ง ROUTE เข้าไปเพื่อให้เกิดการส่งอีเวนต์ต่อกันไปได้ ดังตัวอย่าง เมื่อ TouchSensor ถูกกระตุ้น มันก็จะส่งอีเวนต์ไปสองตัว ดังนั้นสคริปต์จึงได้รับไปสองอีเวนต์เช่นเดียวกัน

```
Transform {
children [
DEF TS TouchSensor { }
Shape { geometry Cone { } }
]
}
DEF SC Script {
url "Example.class"
eventIn SFBool isActive
eventIn SfTime touchTime
}
```

```
ROUTE TS.isActive TO SC.isActive
ROUTE TS.touchTime TO SC.touchTime
```

ความสัมพันธ์ระหว่าง VRML types กับ Java types มีดังนี้

| VRML Type | Java Type |
|---|---|
| SFString | String |
| SFFloat | float |
| SFInt32 | int |
| MFString | String[ ] |
| MFNode | Node[ ] |

## 2.5 ความสามารถและข้อจำกัด

ถึงแม้ว่าภาษา VRML จะถูกพัฒนามาเพื่อให้สอดคล้องกับเทคโนโลยีของอินเตอร์เน็ต แต่ภาษา VRML ก็ยังถือว่าเป็นภาษาที่ใหม่มาก ความสามารถต่าง ๆ ที่ได้ออกแบบก็ยังไม่สอดคล้องกับสภาพความเป็นจริงได้ดีนัก ในปัจจุบันยังมีข้อจำกัดหลายประการที่ทำให้การแพล่หลายเป็นไปได้ช้า

**ความสามารถ**

1. ผู้ใช้มีอิสระในการเคลื่อนไหว สามารถเดินหน้า ถอยหลัง ลอยตัว เหาะเหิน ก็ได้ สามารถมองภาพเคลื่อนไหวได้หลากหลายไม่จำกัดวิธีการ
2. ผู้ใช้สามารถมองภาพได้หลายมุมมอง ไม่ถูกจำกัดไว้กับทิศทางที่กำหนด
3. สามารถใช้ประโยชน์จากภาพสามมิติ ในการซ่อนวัตถุบางอย่างได้ เราสามารถที่จะบังวัตถุหนึ่งด้วยวัตถุอีกอย่างหนึ่งได้
4. การสร้างภาพสามมิติ ทำให้ผู้ใช้เข้าใจโครงร้างของภาพได้ชัดเจนกว่า

**ข้อจำกัด**

1. อุปกรณ์ที่ใช้ยากต่อการควบคุม เช่น เมาส์ คีย์บอร์ด การควบคุมจำเป็นต้องใช้การเรียนรู้พอสมควร ยากกว่าการใช้วินโดส์แบบปกติ
2. ความยากลำบากในการแสดงผล เช่น มอนิเตอร์ การเคลื่อนไหวจะช่วยให้ผู้ใช้เข้าใจตำแหน่งของวัตถุต่าง ๆ ได้ แต่ยากที่จะพิจาณาหาความลึกของจอภาพสองมิติ

3. เวลาที่ใช้ในการคำนวณค่อนข้างมาก ต้องการหน่วยความประมวลผลที่มีความเร็วที่สูงพอสมควร นอกจากนี้ยังมีอุปสรรคกับการส่งข้อมูลผ่านเครือข่ายอีกด้วย

## 2.6 การใช้ Browser

การสั่ง Run เพื่อดูผลการทำงาน คุณจะต้องติดตั้งโปรแกรมจำพวก Live3D เพื่อไว้ใช้สำหรับอ่านไฟล์ประเภท VRML จะแยกการติดตั้งโปรแกรม หรือจะติดตั้งรวมกับเว็บบราวเซอร์เป็นแบบ Plug-in ก็ได้ ถ้าคุณติดต้อง VRML Browser แบบแยกติดตั้งจะมีประโยชน์ตรงที่คุณสามารถดูเอกสารทั้ง VRML และ HTML ไปพร้อม ๆ กันได้

VRML Browser ส่วนใหญ่ที่มีอยู่ในขณะนี้ จะรู้จักเพียง VRML เวอร์ชันหนึ่งเท่านั้น ยังมีไม่กี่โปรแกรมที่สามารถอ่าน VRMl เวอร์ชันสองได้เช่น Cosmo Player 2.0 ซึ่งเป็นปลั๊กอินสำหรับโปรแกรม Netscape 4.0 ขึ้นไป หรือ Netscape 3.0 เวอร์ชันมาตรฐาน