บทที่ 1
ขอบข่ายของเนื้อหา
(A perspective)

1.1 ภาษาชุดคำสั่ง
1.2 ประวัติของภาษาชุดคำสั่ง
1.3 การประยุกต์ใช้ภาษาชุดคำสั่ง
1.4 การประเมินผลภาษาชุดคำสั่ง
1.5 การออกแบบภาษาชุดคำสั่ง

บทที่ 1
ขอบข่ายของเนื้อหา
(A perspective)

ในการศึกษา ภาษาชุดคำสั่ง รวมสิ่งที่น่าสนใจแตกต่างกันสามสิ่ง (conjoins three different interests) คือ นักเขียนโปรแกรมอาชีพ (professional programmer), นักออกแบบภาษา (language designer) และนักทำภาษาให้เกิดผล (language implementer) นอกจากนี้ บุคคลทั้ง 3 กลุ่มที่น่าสนใจเหล่านี้ ทำงานภายในข้อจำกัดและความสามารถซึ่งจัดให้โดย การจัดการของคอมพิวเตอร์ (computer organization) และขีดจำกัดพื้นฐาน ของ ความสามารถในการคำนวณของมัน

คำว่า นักเขียนโปรแกรม (programmer) เป็นคำที่ไม่มีรูปแบบชัดเจน ในแง่ที่ว่ามีความแตกต่างสำคัญระหว่าง ระดับที่ต่างกันและการประยุกต์ใช้ของการเขียนโปรแกรม จะเห็นชัดเจนเช่น นักเขียนโปรแกรมซึ่งเข้าชั้นเรียน COBOL 12 สัปดาห์ หลังจากนั้นเข้าทำงานสาขาการประมวลผลข้อมูล จะแตกต่างจาก นักเขียนโปรแกรมซึ่งเขียน ตัวแปลชุดคำสั่ง (compiler) ภาษา Pascal หรือนักเขียนโปรแกรมซึ่งออกแบบ การทดลองสาขาปัญญาประดิษฐ ในภาษา ลิสป์ (LISP = List Programming) หรือนักเขียนโปรแกรม ซึ่งรวม ซับรูทีนต่างๆ ของ FORTRAN เพื่อแก้ปัญหาวิศวกรรมที่ซับซ้อน (complex engineering problem) หรือนักเขียนโปรแกรมผู้พัฒนาระบบปฏิบัติการของตัวประมวลผลมากกว่าหนึ่งตัว (a multiprocessor operating system) โดยใช้ภาษา Ada ในการศึกษานี้ เราจะพยายามทำข้อแตกต่างเหล่านี้ให้ชัดเจนโดยการอภิปราย ภาษาชุดคำสั่งต่างๆ ในการประยุกต์ใช้งานที่แตกต่างกันในแต่ละสาขา

นักออกแบบภาษา (language designer) เป็นอีกเทอมหนึ่งที่ใกล้เคียงกัน บางภาษา เช่น เอพีแอล (APL = A Programming Language) และ ลิสป์ (LISP = List Language) ซึ่งออกแบบโดยคนหนึ่งคน ด้วยแนวคิดที่เป็นเพียงหนึ่งอย่างเท่านั้น (were designed by a single person with a unique concept) ในขณะที่ ภาษาอื่น ๆ เช่น FORTRAN และ COBOL เป็นผลผลิตของการพัฒนาหลายๆปี โดยกลุ่มนักออกแบบภาษา (language design committees)

นักทำภาษาให้เกิดผล (language implementer) หมายถึงคนหรือกลุ่มคน ซึ่งพัฒนาตัวแปลชุดคำสั่ง หรือ ตัวแปลคำสั่ง สำหรับภาษาหนึ่งบนคอมพิวเตอร์เครื่องหนึ่ง หรือคอมพิวเตอร์หลาย ๆ เครื่อง (is that person or group which develops a compiler or interpreter for a language on a particular machine or machine species)

ส่วนใหญ่ที่พบบ่อยคือ ตัวแปลชุดคำสั่งยุคแรก สำหรับ ภาษา Y บนเครื่อง X พัฒนาโดยบริษัทผู้ผลิตเครื่อง X ตัวอย่างเช่น ตัวแปลชุดคำสั่ง FORTRAN ที่ใช้อยู่มีหลายชุด ตัวแปลชุดหนึ่งพัฒนาโดย IBM สำหรับเครื่องคอมพิวเตอร์ IBM, อีกชุดหนึ่งพัฒนาโดย DEC สำหรับเครื่อง DEC, อีกชุดหนึ่งพัฒนาโดย CDC สำหรับเครื่อง CDC เช่นนี้เป็นต้น บริษัทซอฟต์แวร์เองก็เป็นผู้พัฒนาตัวแปลชุดคำสั่งด้วย เช่นเดียวกับกลุ่มนักวิจัยในมหาวิทยาลัย (university research groups) ตัวอย่างเช่น มหาวิทยาลัย Waterloo พัฒนาตัวแปลชุดคำสั่ง FORTRAN และ Pascal ซึ่งเป็นประโยชน์ในสิ่งแวดล้อม การเขียนโปรแกรมของนักศึกษา เพราะว่าความเร็วของมันในการแปลชุดคำสั่ง และการวินิจฉัยสูงมาก (because of their superior diagnostics and compile speed)

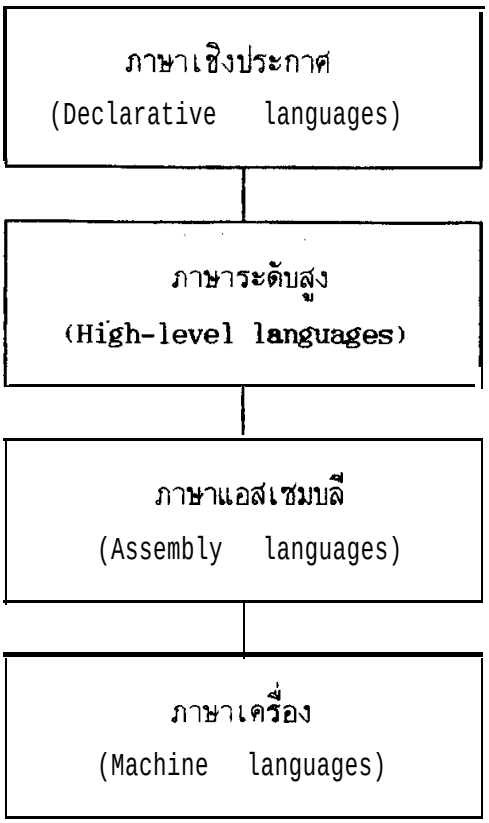
มีความร่วมมือกันอย่างมากระหว่าง นักเขียนโปรแกรม นักออกแบบภาษา และ นักทำภาษาให้เกิดผล แต่ละกลุ่มต้องเข้าใจความจำเป็นและขีดจำกัดซึ่งกำหนดโดย กิจกรรมอื่นอีกสองอย่าง โดยบทนิยาม นักออกแบบภาษาที่ดี ต้องเป็นนักเขียนโปรแกรมที่ดี (By definition, a good language designer must be a good programmer) ในหลายกรณี นักออกแบบภาษามักจะเป็นนักทำภาษาให้เกิดผลคนแรกด้วย

ไม่จำเป็นต้องกล่าวก็ได้ว่า ไม่มีการพัฒนาการทำภาษาให้เกิดผลได้อย่างมีประสิทธิผล เว้นไว้แต่นักทำภาษาให้เกิดผล ซึ่งไม่เคยเขียนโปรแกรม และไม่มีทักษะของวิศวกรรมซอฟต์แวร์ (Needless to say, no effective implementation of a language can be developed unless the implementer has unusual programming and software engineering skills.)

กล่าวอีกอย่างหนึ่งคือ, ถ้านักทำภาษาให้เกิดผล ไม่เคยเขียนโปรแกรม ไม่มีความชำนาญทางด้านวิศวกรรมซอฟต์แวร์ การทำให้เกิดผลของภาษาจะไม่สามารถพัฒนาให้สำเร็จได้อย่างมีประสิทธิผล

1.1 ภาษาชุดคำสั่ง (Programming languages - An overview)

มีอยู่อย่างน้อยที่สุดสองวิธีหลัก ในการมองภาษาชุดคำสั่ง หรือจำแนกภาษาชุดคำสั่ง คือ โดยระดับของมัน และโดยการประยุกต์ใช้งานหลักของมัน (by their level and by their principal applications) นอกจากนั้น ภาษาเหล่านี้ยังแสดงให้เห็นโดย วิวัฒนาการของอดีต ผ่านกระแสของภาษาต่างๆ นอกจากนั้น ภาษาชุดคำสั่ง แบ่งออกเป็น 4 ระดับ ตามรูป 1-1



รูป 1-1 ระดับของภาษาชุดคำสั่ง

ภาษาเชิงประกาศ เหมือนภาษาอังกฤษมากที่สุด ทั้งในด้านกำลังการแสดงออก และเชิงหน้าที่ (are the most like English in their expressive power and functionality) เป็นภาษาที่อยู่ในระดับสูงสุดเมื่อเปรียบเทียบกับภาษาอื่น ๆ โดยพื้นฐาน เป็นภาษาคำสั่งงาน (command languages) ควบคุมโดยข้อความสั่งที่ว่า "จะทำอะไร" ("what to do") มากกว่า "จะทำอย่างไร" ("how to do it") ตัวอย่างต่างๆ ของภาษาเหล่านี้ ได้แก่ภาษาเชิงสถิติ เช่น SAS และ SPSS และภาษาค้นคืนฐานข้อมูล (database retrieval languages) เช่น NATURAL และ IMS ภาษาเหล่านี้ถูกพัฒนาด้วย ความคิดซึ่งเป็นความชำนาญของมืออาชีพ เป็นภาษาซึ่งสามารถรับรู้ได้อย่างรวดเร็ว เช่นภาษา ซึ่งใช้ในงานของเขา โดยไม่จำเป็นต้องเป็นนักเขียนโปรแกรม หรือ มีความชำนาญในการเขียนโปรแกรม

ภาษาระดับสูง เป็นภาษาชุดคำสั่งซึ่งใช้แพร่หลายมากที่สุด (are the most widely

used programming languages) ถึงแม้ว่า จะไม่เป็นการประกาศโดยหลัก ภาษาเหล่านี้ทำให้อัลกอริทึมถูกแสดงให้เห็น ในระดับหนึ่งและสไตล์การเขียน ซึ่งอ่านง่าย เข้าใจได้ง่าย โดยนักเขียนโปรแกรมอื่น ๆ นอกจากนั้น ภาษาระดับสูง ปกติจะมีลักษณะของ การเคลื่อนย้ายง่าย (portability) สิ่งนี้ หมายความว่า มันถูกทำให้เกิดผล (implemented) บนคอมพิวเตอร์ได้หลายเครื่อง ดังนั้น การเคลื่อนย้ายโปรแกรม จากคอมพิวเตอร์เครื่องหนึ่งไปยังคอมพิวเตอร์อีกเครื่องหนึ่ง จึงทำได้ง่าย โดยไม่ต้องมีการแก้ไขใหม่อย่างมากมาย ในความรู้สึกนี้ ภาษาเหล่านั้น จึงเรียกว่า เป็นอิสระจากเครื่อง (machine independent) ตัวอย่างของภาษาระดับสูง ได้แก่ Pascal, APL และ FORTRAN (สำหรับงานวิทยาศาสตร์), COBOL (สำหรับงานประมวลผลข้อมูล), SNOBOL (สำหรับงานประมวลผลข้อความ), LISP และ PROLOG (สำหรับงานสาขาปัญญาประดิษฐ, C และ Ada (สำหรับงานเขียนโปรแกรมระบบ) และ PL/1 (สำหรับงานทั่วไป) ในหนังสือเล่มนี้ จะจำกัดการศึกษา ชนิดของภาษา การออกแบบ และการนำไปประยุกต์ใช้งาน เฉพาะภาษาเหล่านี้เท่านั้น

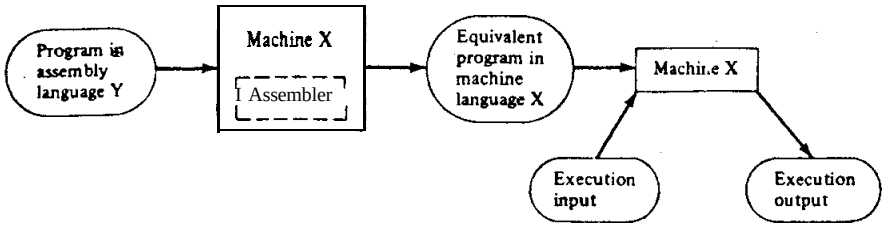
ภาษาแอสเซมบลี และ ภาษาเครื่อง เป็นภาษาขึ้นอยู่กับเครื่อง (machine dependent) คอมพิวเตอร์แต่ละเครื่อง เช่น Digital's VAX จะมีภาษาเครื่องของตนโดยเฉพาะ และเกี่ยวพันธ์กับภาษาแอสเซมบลี ภาษาแอสเซมบลีเป็นรูปแบบการแทนที่เชิงสัญลักษณ์ สำหรับภาษาเครื่องที่เกี่ยวข้องกัน (The assembly language is simply a symbolic representation form for its associated machine language.) ภาษานี้ทำให้การเขียนโปรแกรมน่าเบื่อหน่ายน้อยกว่าภาษาเครื่อง อย่างไรก็ตาม การเข้าใจอย่างดีเกี่ยวกับสถาปัตยกรรมของคอมพิวเตอร์ เป็นสิ่งจำเป็นสำหรับการเขียนโปรแกรม ที่มีประสิทธิผล ไม่ว่าจะเป็นภาษาระดับใดก็ตาม

ต่อไปนี้ เป็นส่วนหนึ่งของโปรแกรมในสามภาษาที่ทำงานเหมือนกัน แสดงให้เห็น ข้อแตกต่างพื้นฐานระหว่างภาษาระดับสูง ภาษาแอสเซมบลี และภาษาเครื่อง

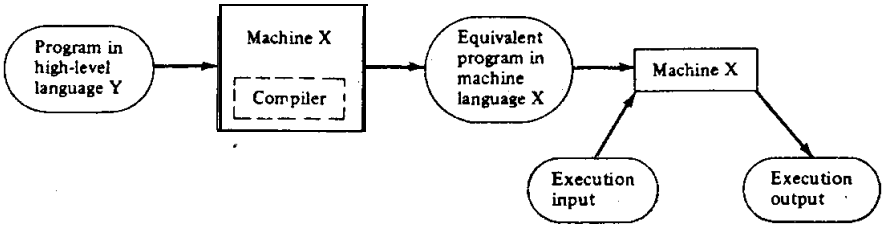
Pascal	Assembly language		Machine language
Z := W + X * Y	L	3,X	4130C1A4
	M	2,Y	3A20C1A8
	A	3,W	1A30C1A0
	ST	3,Z	5030C1A4

ตัวอย่างข้างต้นนี้ แสดงให้เห็นว่า ภาษาซึ่งมีระดับต่ำกว่า จะใกล้ชิดความเข้าใจอย่างสมบูรณ์กับตัวเครื่องคอมพิวเตอร์มากกว่า และห่างไกลกว่าจากความเข้าใจของมนุษย์ปกติ (The lower the level of language, the closer it is to complete comprehension by a particular machine species and the further it is from comprehension by an ordinary human.) มีการสมนัยแบบหนึ่งต่อหนึ่ง ระหว่าง ข้อความสั่งภาษาแอสเซมบลี กับรูปแบบการเข้ารหัสภาษาเครื่องของมัน ข้อแตกต่างสำคัญ ในที่นี้คือ สัญลักษณ์ (X, Y, Z, A สำหรับ "บวก", M สำหรับ "คูณ") ซึ่งใช้ในการเขียนโปรแกรมภาษาแอสเซมบลี ในขณะที่ รหัสตัวเลข (0C1A4, เป็นต้น) เข้าใจได้โดยคอมพิวเตอร์

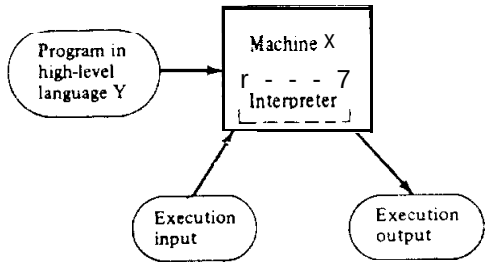
การเขียนโปรแกรมด้วยภาษาระดับสูง หรือภาษาแอสเซมบลี ต้องการตัวเชื่อมประสานบางอย่าง (some sort of interface) กับภาษาเครื่อง ณ เวลาที่มีการวิ่งโปรแกรม ในรูป 1-2a ถึง 1-2c แสดงให้เห็นตัวเชื่อมประสานร่วมที่สำคัญที่สุดสามตัว ได้แก่ แอสเซมเบลอ ตัวแปลชุดคำสั่ง และ ตัวแปลคำสั่ง



รูป 1-2a The assembler interface



รูป 1-2b The compiler interface



รูป 1-2c The interpreter interface

จากภาพ แสดงให้เห็นว่า ตัวเชื่อมประสานแอสเซมเบลอ และตัวเชื่อมประสานตัวแปลชุดคำสั่ง แต่ละชุดนี้ แปลโปรแกรม ให้เป็นโปรแกรมที่เหมือนกัน (equivalent program) ในภาษาเครื่อง X ของคอมพิวเตอร์แม่งาน (host) เป็น ขั้นตอนแยกต่างหาก ก่อนการกระทำการ (execution) ในทางตรงกันข้าม ตัวแปลคำสั่ง execute คำสั่งโดยตรง ใน ภาษาระดับสูง Y โดยไม่ต้องมีขั้นตอนการประมวลผลก่อนหน้า

โดยทั่วไป การแปลชุดคำสั่ง เป็นกระบวนการที่ห่างไกลจากประสิทธิภาพของเครื่องคอมพิวเตอร์ มากกว่า การแปลคำสั่งสำหรับเครื่องคอมพิวเตอร์ส่วนใหญ่ (Compilation is a far more machine-efficient process, in general, than interpretation for most common machine species.) สิ่งนี้เกิดขึ้นอย่างสำคัญ เพราะว่าข้อความสั่งต่างๆ ภายใน ลูป (loops) ต้องมีการตีความใหม่ทุกครั้งที่มันถูก execute โดย ตัวแปลคำสั่งส่วนตัวแปลชุดคำสั่ง แต่ละข้อความสั่งได้ถูกตีความ และจากนั้นแปลเป็นภาษาเครื่องเพียงครั้งเดียวเท่านั้น

บางภาษาโดยหลักแล้วเป็น ภาษาตีความ (interpreted languages) เช่น APL, PROLOG และ LISP ภาษาส่วนที่เหลือในหนังสือนี้ ได้แก่ Pascal, FORTRAN, COBOL, Pl/1, SNOBOL, C, Ada และ Modula-2 ปกติเป็นภาษาแปลความ (compiled languages) ในบางกรณี ตัวแปลชุดคำสั่งจะมีให้เลือกสำหรับภาษาตีความ เช่น LISP และในทางกลับกัน เช่น ตัวแปลคำสั่ง SNOBOL4 ของ Bell Laboratories ก็มีตัวแปลชุดคำสั่งให้เลือกเช่นกัน

การแปลคำสั่ง บ่อยครั้ง เป็นที่ชื่นชอบมากกว่า การแปลชุดคำสั่ง ในการทดลอง หรือ

ในสิ่งแวดล้อมของการศึกษาเขียนโปรแกรม โดยการวิ่งโปรแกรมใหม่ แต่ละครั้ง เกี่ยวข้องกับการเปลี่ยนแปลงในตัวโปรแกรมของมันเอง (where each new run of a program involves a change in the program text itself.)

คุณภาพของการวินิจฉัย และ สิ่งสนับสนุนการแก้จุดบกพร่อง สำหรับภาษาตีความ โดยทั่วไปแล้ว ดีกว่าภาษาแปลความ เพราะว่า ข้อความบอกความผิดพลาด ผูกติดโดยตรงกับข้อความสั่ง ใน ตัวโปรแกรมเดิม

(The quality of diagnostic and debugging support for interpreted languages is generally better than that of complied languages, since error messages are tied directly to statements in the original program text.)

นอกจากนี้แล้ว ข้อดีอย่างมีประสิทธิภาพคือ traditionally enjoyed ของภาษาแปลความ ซึ่งมีอยู่ เหนือ ภาษาตีความ ในไม่ช้านี้ อาจถูกขจัดออกไป อันเนื่องจาก วิวัฒนาการของเครื่องคอมพิวเตอร์ ซึ่งภาษาของมันเป็นภาษาระดับสูง โดยตัวมันเอง ตัวอย่างเช่น เครื่อง LISP ตัวใหม่ซึ่งออกแบบเมื่อเร็ว ๆ นี้ โดย Symbolics and Xerox Corporations

1.2 ประวัติของภาษาชุดคำสั่ง (Historical Prespective for Programming Languages)

นักเขียนโปรแกรม นักออกแบบภาษา และนักทำภาษาให้เกิดผล ของภาษาชุดคำสั่ง ต้องเข้าใจ วิวัฒนาการเชิงประวัติของภาษาต่างๆ เพื่อที่ว่า จะได้รู้คุณค่าว่าทำไมภาษาจึงมีคุณลักษณะแตกต่างกันเช่นที่เป็นอยู่ปัจจุบัน (Why different features are present.) ตัวอย่างเช่น ภาษาใหม่ๆ มีข้อห้าม ไม่ให้ใช้ข้อความสั่ง GO TO ซึ่งเป็นกลไกควบคุมระดับต่ำ และนี่คือสิ่งถูกต้องในเนื้อหาของปรัชญาปัจจุบัน ของวิศวกรรมซอฟต์แวร์ และ การเขียนโปรแกรมโครงสร้าง

ในอดีตนั้นข้อความสั่ง GO TO รวมกับ IF เป็นโครงสร้างควบคุมที่มีให้ใช้เพียงเท่านั้น นักเขียนโปรแกรม ไม่มี รูปแบบของโครงสร้าง WHILE หรือ IF-THEN-ELSE ให้เลือก ดังนั้น เมื่อมอง ภาษา FORTRAN ซึ่งมีรากหยั่งลึก ใน ประวัติศาสตร์ ของภาษาชุดคำสั่งต่างๆ เราอาจจะตกใจ ที่เห็นข้อความสั่ง GO TO เก่าๆ อยู่ในตัวโปรแกรม

สิ่งสำคัญมากกว่านี้คือ ประวัติศาสตร์ทำให้ (1) เราเห็นวิวัฒนาการของครอบครัว ของภาษาชุดคำสั่ง (2) เห็นอิทธิพลของการพัฒนา สถาปัตยกรรมของเครื่องคอมพิวเตอร์ และ

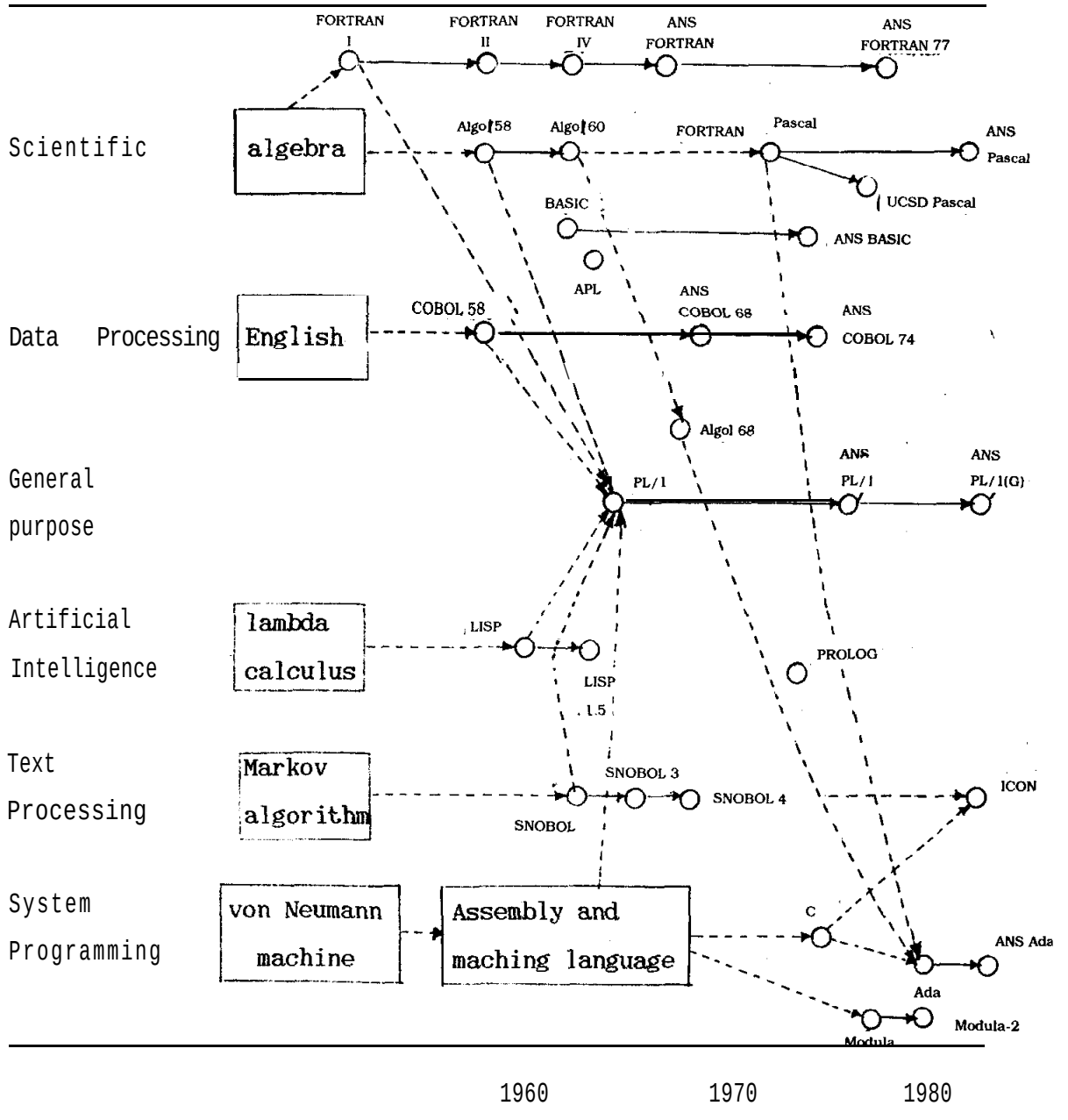
และ การประยุกต์ใช้บนการออกแบบภาษา และ (3) เพื่อหลีกเลี่ยงการออกแบบที่ผิดพลาดในอนาคต โดยเรียนรู้ บทเรียนของอดีต (to see the evolution of families of programming languages, to see the influence of evolving computer architectures and applications on language design, and to avoid future design mistakes by learning the lessons of the past.) รูป 1-3 เป็นแผนภาพโดยย่อสรุปการออกแบบของภาษาระดับสูงที่สำคัญบางภาษา แนวโน้มที่เกิดขึ้นในรอบ 30 ปีที่ผ่านมา

รูปภาพนี้ แทน ตัวอย่างเล็ก ๆ ของภาษาชุดคำสั่งทั้งหมดเท่านั้น ทุกวันนี้ มี ภาษาชุดคำสั่งที่แตกต่างกัน ที่นำมาใช้ มากกว่า 100 ภาษา มีชื่อต่าง ๆ กัน เช่น AMBIT, BASEBALL, LOGO และ MAD ภาษาย่อย ๆ เหล่านี้ หลายภาษา ได้แสดงไว้ในรูป 1-3 หลายภาษาในหนังสือนี้ถูกเลือกขึ้นมา เพราะว่า อิทธิพลที่แข็งแรงและหนักแน่น ซึ่งใช้กันมากระหว่างนักเขียนโปรแกรมต่างๆ เช่นเดียวกับการออกแบบ และคุณลักษณะการทำให้เกิดผลของภาษาที่ดีชัดเจน ครอบคลุมหัวข้อสำคัญส่วนใหญ่ ซึ่ง นักออกแบบภาษา เผชิญหน้า (confront) อยู่ และงานส่วนใหญ่ซึ่งนักเขียนโปรแกรมเผชิญหน้าอยู่

รูป 1-3 มีเส้นเชื่อมต่อภาษาต่างๆ เส้นทึบ หมายถึง บรรพบุรุษโดยตรง (direct ancestry) ส่วนเส้นประ หมายถึง ได้รับอิทธิพลมา (strong influence) ตัวอย่างเช่น FORTRAN 1 เป็นบรรพบุรุษโดยตรงของ FORTRAN II ในขณะที่ FORTRAN, COBOL, ALGOL 60, LISP, SNOBOL และ Assembly language ภาษาทั้งหมดนี้ มี อิทธิพลต่อ การออกแบบภาษา PL/1

หลายภาษาในรูปนี้ มีตัวอักษร ANS นำหน้า สิ่งนี้หมายความว่า สถาบันมาตรฐานแห่งชาติของสหรัฐอเมริกา (American National Standards Institute) ให้การรับรองว่า เวอร์ชัน (version) ของภาษาเป็นมาตรฐานแห่งชาติ ภาษาซึ่งเป็นมาตรฐาน หมายถึง เครื่องคอมพิวเตอร์ใดๆ ซึ่ง implement ภาษานั้น ต้องเข้ากันได้ (conform) กับข้อกำหนดของมาตรฐานทั้งหมด (all of the standards's specifications) ดังนั้น โปรแกรมจึงสามารถเคลื่อนย้ายได้สูงสุด จากคอมพิวเตอร์เครื่องหนึ่ง ไปยังคอมพิวเตอร์อีกเครื่องหนึ่งได้ นโยบายของรัฐบาลไม่ต้องการให้ซื้อคอมพิวเตอร์ใดๆ ซึ่งไม่เข้ากันกับ เวอร์ชันมาตรฐานของภาษาใด ๆ ซึ่งสนับสนุนมัน แต่ต้องการผลักดันให้เป็นกระบวนการที่เป็นมาตรฐาน เพราะว่า รัฐบาลคือผู้ซื้อคอมพิวเตอร์รายใหญ่ที่สุด ในชาติ

สุดท้ายในรูป 1-3 แสดงให้เห็นอิทธิพลบน หลายปี ก่อน ค.ศ.1960 การออกแบบ



รูป 1-3 Historical prespectives for several programming languages

ภาษาชุดคำสั่ง ตัวอย่างเช่น สัญลักษณ์ทางพีชคณิต มีอิทธิพลอย่างมากต่อการออกแบบภาษา FORTRAN และ ภาษา ALGOL ในทางตรงกันข้าม ภาษาอังกฤษ มีอิทธิพลต่อการพัฒนาภาษา COBOL ส่วนวิชาคำนวณขั้นสูง(Chursh's lambda calculus) เป็นพื้นฐานของสัญลักษณ์เชิงฟังก์ชันของภาษา LISP ขณะที่อัลกอริทึมของ Markov เป็นแรงจูงใจ ให้เกิดสไตล์รูปแบบการจับคู่ของภาษา SNOBOL สถาปัตยกรรมคอมพิวเตอร์ ของ Von Neumann ซึ่งวิวัฒนาการจาก Turing machine ก่อนหน้านั้น เป็นตัวแบบพื้นฐาน (basic model) สำหรับออกแบบเครื่องคอมพิวเตอร์ส่วนใหญ่ ในสามสิบปีหลัง เครื่องคอมพิวเตอร์เหล่านี้ ไม่เพียงแต่มีอิทธิพลต่อภาษายุคแรกๆ เท่านั้น แต่ยังเป็น กรอบงานเชิงปฏิบัติ (operational framework) ภายในซึ่งพัฒนาการเขียนโปรแกรมระบบ การอภิปรายในรายละเอียดมากขึ้นของ ตัวแบบยุคแรกเหล่านี้ ไม่ใช่วัตถุประสงค์ของหนังสือเล่มนี้ อย่างไรก็ตาม มันเป็นสิ่งสำคัญ ที่ชี้ให้เห็นตรงนี้ เพราะว่าอิทธิพลหลักของมัน ในวิวัฒนาการของภาษาชุดคำสั่งยุคแรก ๆ อย่างหนึ่ง และสถานะของมัน ซึ่งเป็นหัวใจของทฤษฎีการคำนวณอีกอย่างหนึ่งด้วย

อีกจุดหนึ่ง อัลกอริทึม ใดๆ ซึ่งสามารถอธิบายได้ด้วยภาษาอังกฤษ มีความหมายเหมือนกับ เขียนด้วย turing machine (von Neumann machine), อัลกอริทึมแบบ Markov หรือฟังก์ชันเวียนบังเกิด

(Any algorithm that can be described in English can be equivalently written as a Turing machine (Von Neumann machine), a Markov algorithm or a recursive function.)

ข้อยืนยันนี้ เป็นที่รู้จักกันกว้างขวาง คือ "Chursh's Thesis" ทำให้เราเขียนอัลกอริทึม ใน สไตล์การเขียนโปรแกรมต่างๆ ได้อย่างหลากหลาย โดยไม่ต้อง เสียสละ การวัดใด ๆ ของ การเป็นทั่วไป (generality) หรือกำลังการเขียนโปรแกรมในหัวเลี้ยวหัวต่อ (programming "power" in the transition)

1.3 การประยุกต์ใช้ภาษาชุดคำสั่ง

(Applications of Programming Languages)

จากรูป 1-3 แสดงให้เห็น ภาษาชุดคำสั่งต่างๆ กับการพัฒนา สำหรับงานสาขาต่างๆ โดยหลัก ได้แก่งานทางด้านวิทยาศาสตร์ การประมวลผลข้อมูล ปัญญาประดิษฐ การประมวลผลข้อความ, และ การเขียนโปรแกรมระบบ ในหัวข้อนี้ จะกล่าวถึงลักษณะโดยสรุป ของ การประยุกต์ใช้งานในแต่ละสาขา เพื่อว่าผู้อ่านจะได้มี ความคิดที่ชัดเจน ของ ความสนใจหลัก และความแตกต่างของภาษาเหล่านี้

งานทางด้านวิทยาศาสตร์ (Scientific applications)

จะมีลักษณะเฉพาะ ดังนี้ เป็นงานซึ่งเกี่ยวกับ การจัดการอย่างชัดเจนกับตัวเลข และแถวลำดับของตัวเลข โดยใช้หลักเชิงคณิตศาสตร์และสถิติ เป็นมูลฐานสำหรับอัลกอริทึม โดยที่อัลกอริทึมเหล่านี้ ล้อมรอบด้วยปัญหา เช่น การทดสอบนัยสำคัญทางสถิติ การเขียนโปรแกรมเชิงเส้น การวิเคราะห์การถดถอย และการประมาณค่าเชิงตัวเลข สำหรับผลเฉลยของ สมการเชิงอนุพันธ์ และอินทิกรัล

บ่อยครั้งที่จำนวนของข้อมูล ในปัญหาเช่นนี้ ค่อนข้างน้อย (เป็นอันดับของ จำนวนเลขไม่กี่ร้อย) มีโครงสร้างง่าย (แถวลำดับหนึ่งมิติหรือสองมิติ) แต่ปัญหาทางวิทยาศาสตร์ บางอย่างจะมีจำนวนข้อมูลมาก (เช่นข้อมูลที่ได้จาก อุปกรณ์ดาวเทียม หรือเครื่องส่งวิทยุ) และปัญหาซึ่งต้องเผชิญหน้าทันทีของ การลดลงของข้อมูล (data reduction) ก่อนการวิเคราะห์ความเป็นไปได้ และการตีความหมายใด ๆ จะเกิดขึ้น

ความซับซ้อนเชิงคณิตศาสตร์ เป็นลักษณะเฉพาะที่สำคัญของ ปัญหาทางด้านวิทยาศาสตร์ นักเขียนโปรแกรม ต้องมีประสบการณ์อย่างดี ใน หลักคณิตศาสตร์ ภายใน อัลกอริทึม เพื่อที่จะวินิจฉัยปัญหาได้อย่างถูกต้อง หรือ ทำให้ละเอียดยิ่งขึ้น นอกจากนั้น อัลกอริทึมซึ่งไม่เที่ยงตรง (unstable algorithms) หมายถึง อัลกอริทึมซึ่งให้ผลลัพธ์ ไม่ถูกต้องสำหรับ ข้อมูลเข้าบางอย่าง ต้องจัดการอย่างถูกต้องด้วย (Unstable algorithms - ones whose results may become inaccurate for certain input data values - must be properly handled.)

สุดท้าย ปัญหาทางด้านวิทยาศาสตร์ ปกติจะใช้ หน่วยประมวลผลกลางของคอมพิวเตอร์ มากกว่า อุปกรณ์รับข้อมูลเข้า หรือส่งข้อมูลออก นั่นคือ ส่วนใหญ่ของเวลาเครื่องที่ใช้ จะเป็นการคำนวณทางคณิตศาสตร์ ซึ่งตรงกันข้ามกับ การปฏิบัติการรับและส่งข้อมูล สิ่งนี้เรียกว่า "compute bound"

ในวิธีหลักเหล่านี้ สาขาการเขียนโปรแกรมทางวิทยาศาสตร์ จึงแตกต่างจาก สาขาอื่นซึ่งจะกล่าวต่อไป กรณีศึกษาที่ 1 อธิบายใน ภาคผนวก A เป็นตัวอย่างการประยุกต์ใช้งานทางวิทยาศาสตร์ ซึ่งจะใช้ในบทต่อไป เพื่อแสดงให้เห็นและเปรียบเทียบกับการเขียนโปรแกรมเชิงวิทยาศาสตร์ ของภาษา Pascal, FORTRAN และ APL

งานประมวลผลข้อมูล (Data processing applications)

ลักษณะเฉพาะของปัญหาการเขียนโปรแกรมเหล่านี้ คือ สิ่งที่น่าสนใจเห็นชัดเจน ได้แก่

การสร้าง การบำรุงรักษา การนำมาใช้ การทำสรุปข้อมูล ในระเบียนและแฟ้ม (creation, maintenance, extraction, and summarization of data) โดยปกติ การประมวลผลข้อมูลจะพบว่าเป็นหัวใจ ของ การสนับสนุน การจัดการ ขององค์กร และรวมไปถึง หน้าที่ทำบัญชีเงินเดือน ทำบัญชี ทำใบเสร็จรับเงิน สินค้าคงคลัง การผลิต และการประมวลผลข้อมูลยอดขาย ปริมาณของข้อมูล ในแฟ้มเหล่านี้ โดยทั่วไป จะมีขนาดใหญ่ คือปกติหลายพันระเบียนต่อหนึ่งแฟ้ม และหลายร้อยตัวอักขระ ของ สารสนเทศต่อหนึ่งระเบียน

ในทางตรงกันข้าม ปริมาณของการคำนวณทางคณิตศาสตร์ หรือ การจัดการอื่น ๆ ของข้อมูลในแต่ละระเบียน จะค่อนข้างน้อย โปรแกรมการประมวลผลข้อมูล ปกติจะใช้เวลาเครื่องส่วนใหญ่ทำ การปฏิบัติการ รับข้อมูลเข้าและส่งข้อมูลออก ได้แก่ ส่งระเบียนของแฟ้ม ไปยังตำแหน่งที่เก็บ และ/หรือ ปรับสารสนเทศ บางอย่างให้เป็นปัจจุบัน โดยหลักทำอย่างสม่ำเสมอ (ทุกวัน ทุกสัปดาห์ หรือทุกเดือน) ดังนั้น ทรัพยากร (resource) ในงานประมวลผลข้อมูลปกติจะเป็น เครื่องขับเทป หรือ เนื้อที่จานแม่เหล็ก มากกว่า ตัวประมวลผลกลาง

โดยธรรมชาติของมัน งานประมวลผลข้อมูลส่วนใหญ่ จะเป็นประเภท งานแบบกลุ่ม (batch) มากกว่า งานแบบเชิงโต้ตอบ (interactive) นั่นคือ เป็นงานจัดลำดับการทำงานสม่ำเสมอ บนหลักการคาดคะเนได้ และต้องการใช้คอมพิวเตอร์ โดยทั่วไป เป็นความรู้ในขั้นสูง ตัวอย่างเช่น บัญชีจ่ายเงินค่าจ้างต่อสัปดาห์ของบริษัท (a company's weekly payroll) จะมีวัฏจักรการประมวลผลหลักทุกสัปดาห์ และแต่ละขั้นตอนเป็นตารางการทำงาน อย่างง่าย

มีข้อยกเว้นหลายอย่าง สำหรับแนวทั่วไปนี้ องค์กรมีการเพิ่มขึ้นของงานประเภทเชิงโต้ตอบมากขึ้นเรื่อย ๆ มากกว่า งานแบบกลุ่ม การคำนวณเป็นสิ่งหลีกเลี่ยงไม่ได้ สำหรับงานประมวลผลข้อมูล ตัวอย่างเช่น งานสั่งสินค้า เมื่อลูกค้าสอบถามสินค้าชิ้นหนึ่งทางโทรศัพท์ ต้องให้คำตอบทันที หรือการสอบถามที่นั่งบนเครื่องบิน หรือการจัดหัวเทียน สำหรับเครื่องยนต์

งานประมวลผลข้อมูลปกติต้องเกี่ยวกับ บูรณภาพของข้อมูล (data integrity) มากกว่างานสาขาอื่นๆ โดยหลัก สิ่งนี้เกี่ยวข้องไม่เฉพาะแต่คำถามของความแม่นตรง และเชื่อถือได้ แต่ยังเป็นคำถาม ของ ความปลอดภัยของข้อมูลด้วย (Mainly, this involves not only the question of accuracy and reliability but also the question of data security.)

โปรแกรมประมวลผลข้อมูล อย่างน้อยที่สุด ต้องป้องกันแฟ้มไม่ให้เสียหาย จาก ข้อมูลที่ไม่ถูกต้อง นอกจากนั้น ต้องแน่ใจว่า ข้อมูลที่รับรู้ไว (sensitive data) จะสามารถเข้าถึงได้เฉพาะบุคคลที่จำเป็นต้องเข้าถึงเท่านั้น ไม่ใช่บุคคลทั่วไป

กรณีศึกษาที่ 2 เป็นตัวอย่างของงานประมวลผลข้อมูล ซึ่ง implemented ในบทของภาษา COBOL และ PL/1 ของหนังสือนี้

งานประมวลผลข้อความ (Text processing applications)

มีลักษณะเฉพาะดังนี้ เป็นกิจกรรมหลักซึ่งเกี่ยวข้องกับการจัดการข้อมูลซึ่งเป็น เนื้อหาในภาษาธรรมชาติ มากกว่าตัวเลข (those whose principal activity involves the manipulation of natural language text, rather than numbers, as their data.) วิวัฒนาการของเทคโนโลยีการประมวลผลคำสมัยใหม่ วางอยู่บนหลักการของ อัลกอริทึมประมวลผลข้อความ เพื่อกระทำการ จัดรูปแบบต่างๆ และหน้าที่อื่น ๆ ซึ่งพนักงานพิมพ์ดีด ใช้ระหว่างการเตรียมต้นฉบับ ตัวอย่างเช่น เนื้อหาของหนังสือเล่มนี้ ได้เตรียมจากการใช้ซอฟต์แวร์การประมวลผลคำ บนคอมพิวเตอร์ขนาดเล็กมาก

ปกติแล้ว ข้อความ (text) ในงานสาขาเช่นนี้ จะเป็นภาษาอังกฤษ ถึงแม้ว่า การประมวลผลคำขั้นสูง เมื่อไม่นานมานี้ จะแสดงให้เห็น มีความสามารถหลายภาษา ในรูปแบบของแป้นพิมพ์เลือก (alternative keyboards) และตัวอักษรต่าง ๆ เช่น ภาษาอารบิก และภาษาญี่ปุ่น อัลกอริทึมประมวลข้อความ แตกต่างจากอัลกอริทึมอื่น ๆ ในประเภทของ ความท้าทายการเขียนโปรแกรมที่นำเสนอ สิ่งนี้จะปรากฏในผลเฉลย สำหรับกรณีศึกษาที่ 3 ซึ่งเป็นปัญหาการประมวลผลข้อความ เราจะแสดงให้เห็นปัญหานี้ โดยใช้ภาษา SNOBOL และภาษา C เพราะว่ามันมีความสามารถประมวลผลคำที่แข็ง

งานทางด้านปัญญาประดิษฐ์ (Artificial intelligence applications)

มีลักษณะเฉพาะเป็นโปรแกรม ซึ่ง โดยหลักออกแบบมา เพื่อให้เข้ากันได้ (emulate) กับพฤติกรรมทางปัญญา ซึ่งรวมทั้ง อัลกอริทึม เล่นเกมส์ เช่น หมากรุก โปรแกรมเข้าใจภาษาธรรมชาติ (natural language understanding programs) ภาพคอมพิวเตอร์ (computer vision) หุ่นยนต์ (robotics) และ ระบบผู้เชี่ยวชาญ (expert systems) ต่อไป ในตัวอย่างหลังนั้น คอมพิวเตอร์ถูกโปรแกรมให้เล่นบทบาทของผู้เชี่ยวชาญ เช่น นักวินิจฉัยทางยา (a medical diagnostician) ในการทำงานของเขาให้สำเร็จ (เช่น วินิจฉัยโรคจากเซตของอาการที่กำหนดให้)

จนกระทั่งเมื่อเร็ว ๆ นี้ ปัญญาประดิษฐ์ (หรือ AI) ได้จำกัด งานของมัน ใน ห้องปฏิบัติการวิจัย ซึ่งการทดลองนำร่องหลายอย่างได้เป็นตัวแบบชนิดต่าง ๆ ของพฤติกรรมปัญญาประดิษฐ ที่น่าสนใจ อย่างไรก็ตาม ขณะนี้การทดลองเหล่านี้ หลายสิ่งมีการนำไปสู่โดเมนซึ่งใช้งานได้จริง และผลกระทบของมัน ได้แสดงให้เห็นใน สาขาต่าง ๆ กันเช่น สายการผลิต

รถยนต์ และ จอภาพของเครื่องมือที่ซับซ้อน

งานทางด้าน AI ได้กำหนดไว้แล้ว ในกรณีศึกษาที่ 4 และแสดงให้เห็น ในบทเรื่อง ภาษา LISP และ PROLOG, ภาษา LISP มีอิทธิพลมานาน กับภาษาเชียนโปรแกรมทางด้าน AI ในขณะที่ PROLOG เป็นภาษาใหม่กว่า พัฒนาบนหลักของ การเขียนโปรแกรมเชิงตรรกศาสตร์(LISP has long been the predominant AI programming language, while PROLOG is a newer language designed on the principle of "logic programming.")

งานทางด้านการเขียนโปรแกรมระบบ

(System programming applications)

เกี่ยวข้องกับการพัฒนา โปรแกรมต่าง ๆ ซึ่งเป็นตัวเชื่อมประสาน ระบบคอมพิวเตอร์ (ฮาร์ดแวร์) กับนักเขียนโปรแกรม และ พนักงานควบคุมเครื่อง โปรแกรมเหล่านี้ ได้แก่ ตัวแปลชุดคำสั่ง แอสเซมเบลอ ตัวแปลคำสั่ง อินพุท-เอ้าพุท รูทีน โปรแกรมจัดสิ่งอำนวยความสะดวก และชุดคำสั่งจัดลำดับงาน สำหรับอำนวยประโยชน์ และให้บริการ ทรัพยากรต่างๆ ซึ่งประกอบกันเป็นระบบคอมพิวเตอร์

(involve developing those programs that interface the computer systemcthe hardware) with the programmer and operator. These programs include compilers, assemblers, interpreters, input-output routines. program management facilities, and schedulers for utilizing and serving the various resources that comprise the corn puter system.)

ลักษณะเฉพาะ สองสิ่ง ซึ่งทำให้ การเขียนโปรแกรมระบบแตกต่างจาก การเขียน โปรแกรม ชนิดอื่น ๆ คือ

(1) สามารถทำงานได้อย่างมีประสิทธิผลกับเหตุการณ์ ซึ่งไม่อาจคาดการณ์ได้ หรือ "ข้อยกเว้นต่าง ๆ" (เช่น ข้อผิดพลาดทาง I/O)

(the requirement to deal effectively with unpredictatble events, or "exceptions" (such as an I/O error))

(2) จำเป็นต้องประสานกับกิจกรรมอื่น ๆ ของโปรแกรมปฏิบัติการต่างๆ หรืองาน ซึ่งไม่ประสานจังหวะกัน

(the need to coordinate the activities of various asynchonously executing programs, or tasks.)

โดยปกติสนับสนุนระบบปฏิบัติการ ตัวอย่างเช่น กิจกรรมที่ทำพร้อม ๆ กัน ของ โปรแกรมอิสระหลายชุด (หรือ on-line users) ซึ่งส่วนสำคัญที่สุด คือ แต่ละโปรแกรม ไม่มีความจำเป็นต้องมีการโต้ตอบ (interaction) ซึ่งกันและกัน ในเหตุการณ์ที่เกิดไม่บ่อยนัก คือ มันมีการโต้ตอบกัน (interact) (หรือขัดแย้งกัน ในการใช้แฟ้มเดียวกัน พร้อม ๆ กัน) ระบบต้อง ตอบรับอย่างสวยงามและจัดการสิ่งโต้ตอบนั้นได้ (หรือแก้ปัญหาขัดแย้งได้)

สมัยก่อน การเขียนโปรแกรมระบบส่วนใหญ่ กระทำด้วยภาษาแอสเซมบลี อย่างไรก็ตาม เมื่อเร็ว ๆ นี้ มีความพยายามอย่างเข็มแข็งที่จะทำลายแนวโน้มนี้ และ เราจะเห็นผลลัพธ์ของความพยายามนั้นเมื่อเราศึกษาภาษา Ada และ Modula-2 เราจะใช้สองภาษานี้ แก้ปัญหากรณีศึกษาที่ 5 ซึ่งเป็นงานทางด้าน การเขียนโปรแกรมระบบ

ปัญหาการเขียน โปรแกรมส่วนใหญ่ ไม่จำกัดว่าต้องอยู่ ในหนึ่งสาขาใดๆ เท่านั้นในห้าสาขาที่กล่าวนี้ ตัวอย่างเช่น ต้องการใช้ เทคนิค "ฐานข้อมูล" อย่างมีประสิทธิผล เริ่มต้นจากสาขาการประมวลผลข้อมูล แต่ขณะนี้ ความจำเป็นนี้ มีการร่วม (share) กัน โดยงานด้านวิทยาศาสตร์ และงานทางด้านปัญญาประดิษฐ ซึ่งเก็บข้อมูลปริมาณมาก (ตัวอย่างเช่น ข้อมูลสำมะโน (sensus data) หรือข้อมูลความรู้ในโลก (world knowledge data)) ในวิธีที่มีประสิทธิผล

เขตแดน (boundaries) ระหว่างงานทางปัญญาประดิษฐ และการประมวลผลข้อความยังเป็นเรื่องแบบบางมาก (fuzzy) สามารถแสดงให้เห็น เช่น งานด้านการแปลภาษาธรรมชาติ (natural language translation) สำหรับการแปลภาษาธรรมชาติ ให้ได้ประสิทธิผลนั้น โปรแกรม อย่างน้อยที่สุด ต้องสามารถเข้าใจ ข้อความ ที่จะทำการแปล ได้ ดังนั้น ความจำเป็นสำหรับ AI ใส่เพิ่มเติมเข้ากับ การประมวลผลข้อความสำหรับปัญหานี้ จึงปรากฏขึ้น

อย่างไรก็ตาม ในหนังสือเล่มนี้ ได้แยกงานออกเป็น 5 สาขา เพราะว่า มันแทนโดเมน ของ การเขียนโปรแกรมที่นิยามดี และพัฒนาได้ดีอย่างมีเหตุผล (because they tend to resonably represent well-defined and well-developed programming "domain".) นอกจากนี้ ภาษาชุดคำสั่งต่างๆ ที่ใช้ในงานแต่ละสาขาเหล่านี้ ยังแตกต่างกันมาก เราจะเห็นความจำเป็น ของ งานสาขาต่างๆ กัน กับ การเลือกใช้ภาษาที่แตกต่างกันที่ช่วยเหลือซึ่งกันและกัน (foster) ปัญหากรณีศึกษา 5 ชนิดนี้ และบทที่ของภาษาที่จะใช้แก้ปัญหา ได้สรุปไว้แล้วในรูป 1-4

Case study	Application area	Language (Chapters) where illustrated
1. Matrix inversion	Scientific	Pascal(2),Fortran(3),APL(8)
2. Employee file maintenance	Data processing	COBOL(4), PL/1(5)
3. Text formatter	Text processing	SNOBOL(7), C(12)
4. Missionaries and cannibals	Artificial Intelligence	LISP(9), PROLOG(11)
5. Job scheduler	Systems programming	Ada(3), Modula-2(14)

รูป 1-4 The case study problems and their illustrations

ตามโครงร่างนี้ กรณีศึกษาแต่ละชุดออกแบบมาเพื่อให้เป็นแบบฝึกหัดดูความแข็งแกร่ง (strengths) และความอ่อน (weakness) ของภาษาต่างๆ ในงานสาขานั้น ซึ่ง มันเป็นตัวแทน นอกจากนี้แล้ว การคู่กัน ของภาษาสำหรับกรณีศึกษาแต่ละชุด จะให้เห็นเป็นมูลฐาน สำหรับการเปรียบเทียบที่สำคัญและน่าสนใจหลายสิ่ง ตัวอย่างเช่น โปรแกรมภาษา FORTRAN และภาษา APL สำหรับกรณีศึกษาที่ 1 จะเห็น การเปรียบเทียบ สองภาษานี้ สำหรับ การเขียนโปรแกรมเชิงวิทยาศาสตร์

ถึงแม้ว่า ชื่อเรื่องของกรณีศึกษาต่าง ๆ จะแสดงให้เห็นเนื้อหาของมัน รายละเอียดอย่างสมบูรณ์ ของ กรณีศึกษาแต่ละชุด ได้ให้ไว้แล้ว ในภาคผนวก A ถึง E ตามลำดับ ผู้อ่านควรจะทำความคุ้นเคย กับรายละเอียดกรณีศึกษาเหล่านี้ ก่อนทบทวนโปรแกรมที่สมนัยกัน ใน บทภาษาต่างๆ โดยตัวมันเอง

1.4 การประเมินภาษาชุดคำสั่ง (Programming Language Evaluation)

ความมุ่งหมายหลัก ของหนังสือเล่มนี้คือ ต้องการอธิบายและ ให้เกณฑ์ (criterior)

ที่มีประสิทธิผล สำหรับการประเมินค่า และ การเปรียบเทียบภาษาชุดคำสั่งต่าง ๆ เนื้อหานี้ได้รับการสนใจเป็นครั้งคราวเท่านั้น จนกระทั่งเมื่อเร็ว ๆ นี้ เมื่อ การขยายเซตของเกณฑ์การประเมินผล ภาษา ได้ถูกพัฒนาขึ้นระหว่าง กระบวนการออกแบบภาษา Ada เกณฑ์เหล่านี้ ใช้เป็นมูลฐานร่วม ซึ่ง ภาษาที่มีอยู่แล้วหลายภาษา ได้ถูกนำมาประเมินว่า จะเป็นไปตามความต้องการ ภาษาระดับสูง ของ กระทรวงกลาโหม (the Defense Department) หรือไม่ ระหว่าง สามภาษาที่นำมาพิจารณา ได้แก่ Pascal, JOVIAL และ PL/1 ผลลัพธ์จากการประเมินนี้ คือ ไม่มีภาษาใด ที่มีคุณสมบัติตามที่ต้องการ ดังนั้น กระทรวงกลาโหม จึงตั้งต้น ออกแบบภาษาใหม่ทั้งหมด ผลลัพธ์คือ ได้ภาษา Ada และบทสรุปอย่างสมบูรณ์ ของภาษานี้ อยู่ในบทที่ 13

อย่างไรก็ตาม เซตที่สมบูรณ์ ของ ความต้องการซึ่ง กระทรวงกลาโหมใช้ ในการประเมินภาษา ยังไม่เหมาะสมที่จะเป็นเครื่องมือ สำหรับการใช้ของเรา เพราะว่ามันยาวมาก มีรายละเอียดมาก และเป็นงานที่น่าเบื่อ เป้าหมายของเราในที่นี้ คือเพื่อนิยาม รายการของคุณลักษณะขนาดเล็กมีเหตุผล ซึ่งคนโดยทั่วไปมองหา ใน ภาษาชุดคำสั่ง (Our goal here is rather to define a resonably small list of characteristics that one generally looks for in a programming language.) ลักษณะเฉพาะแต่ละอย่างนี้ควรจะเข้าใจได้ชัดเจน และลักษณะเฉพาะทุกข้อเมื่อนำมารวมกัน ควรจะนิยามว่า อะไรซึ่งทำให้เป็นภาษา ดี (good) ในความรู้สึกที่ใช่แน่นอน ลักษณะเฉพาะเหล่านี้ รวมทั้งจุดต่างๆ กัน ซึ่งมองโดย นักเขียนโปรแกรม นักออกแบบโปรแกรมและนักทำภาษาให้เกิดผล รายการที่สมบูรณ์ของ ลักษณะเฉพาะนี้ เรียกว่า เกณฑ์ สำหรับ การประเมินและเปรียบเทียบภาษา (criteria for language evaluation and comparison) ซึ่งกำหนดให้แล้วในรูป 1-5 และทั้งหมดนี้ จะอภิปรายทีละหัวข้อ และแสดงให้เห็น ในพาราการาฟต่างๆ ที่ตามมา

1. Expressivity (การแสดงออกชัดเจน)
2. Well-definedness (นิยามได้เป็นอย่างดี)
3. Data types and structures (ชนิดข้อมูลและโครงสร้าง)
4. Modularity (สภาพมอดุลาร์)
5. Input-output facilities (สิ่งอำนวยความสะดวกด้านอินพุท-เอ้าพุท)
6. Portability (การเคลื่อนย้ายได้)
7. Efficiency (ประสิทธิภาพ)
8. Pedagogy (การเรียนการสอน)
9. Generality (ใช้ได้ทั่วไป)

รูป l-5 Nine criteria for language evaluation and comparison

ถึงแม้ว่าเกณฑ์เหล่านี้ จะไม่ได้กล่าวทั้งหมดในรายละเอียด แต่ก็พอเพียงที่จะเห็นความหมายกว้าง ๆ ในการประเมินค่าและเปรียบเทียบภาษาซึ่งได้ กระทำขึ้น นั่นคือ ถ้าภาษาหนึ่ง จะถูกมองว่า ดีเยี่ยม (excellent) ในทั้งหมด (หรือส่วนใหญ่) ของเกณฑ์ 9 ข้อที่มีอยู่ ในรูป 1-5 แสดงว่า ภาษานั้นค่อนข้างจะเป็นภาษาชุดคำสั่งที่ดีที่สุด (That is, if a language is viewed as "excellent" in all (or most) of the nine criteria listed in Figure 1-5, it is likely to be a superior programming language.)

(1) การแสดงออกชัดเจน หมายถึงความสามารถของภาษาที่จะสะท้อน ความหมายโดยตั้งใจของ นักออกแบบอัลกอริทึม (นักเขียนโปรแกรม) ได้อย่างชัดเจน (By "expressivity", we mean the ability of a language to clearly reflect the meaning intended by the algorithm designer (the programmer).) ดังนั้น ภาษาซึ่งแสดงออกชัดเจน จึงต้องกล่าวอย่างรวบรัด (compactly stated) และสามารถใช้คำสั่งในรูปแบบที่เกี่ยวกับ การเขียนโปรแกรมโครงสร้างได้ (ปกติ ได้แก่ คำสั่ง "while" ลูป และ ข้อความสั่ง "if-then-else") นอกจากนี้แล้ว ภาษาซึ่งแสดงออกได้อย่างชัดเจน ได้รวมสัญลักษณ์ซึ่งพ้องกัน (consistency) กับสิ่งปกติที่ใช้ในสาขาของภาษาซึ่งออกแบบมานั้น ด้วยเหตุนี้ นักคณิตศาสตร์ ใช้ภาษา FORTRAN ควรจะคาดได้ว่าการหา นิพจน์เชิงพีชคณิตของมัน

พ้องกันกับ ที่ใช้ในวิชาพีชคณิตโดยทั่วไป สุดท้าย ภาษาซึ่งแสดงออก ได้ชัดเจน คือภาษาซึ่งสัญลักษณ์ของมันใช้เป็นรูปแบบอย่างเดียวกัน (uniformly used) ความหมายของสัญลักษณ์ จะต้องไม่แปรเปลี่ยนอย่างไม่มีเหตุผล จาก หน่วยหนึ่ง ใน โปรแกรม ไปยังอีก หน่วยหนึ่ง

ตัวอย่าง

จงพิจารณา ข้อความสั่งต่างๆ ข้างล่างนี้ ซึ่งมีความหมายเหมือนกันจาก สี่ภาษาที่แตกต่างกัน แสดงการคำนวณหาผลบวก และการกำหนดค่าของมัน.

C = A + B	ภาษา FORTRAN
C := A + B	ภาษา PASCAL
(SETQ C(+AB))	ภาษา LISP
ADD A, B GIVING C	ภาษา COBOL

ในสองคำสั่งแรก ตัวปฏิบัติการ (assingment และ +) เป็น สัญกรณ์เติมกลาง (infixed) เช่นที่ใช้ในวิชาพีชคณิตทั่วไป ขณะที่คำสั่งที่สาม ตัวปฏิบัติการ เป็นสัญกรณ์เติมหน้า (prefixed) และข้อความสั่งต้องใส่เครื่องหมายวงเล็บอย่างสมบูรณ์ ตัวอย่างที่ 4 คล้ายกับประโยคในภาษาอังกฤษ ทั้งสไตล์ของมัน และความรวบรัดน้อยกว่า ข้อความสั่งอื่นๆ สุดท้าย เฉพาะสัญลักษณ์ = และ := เท่านั้น ที่ทำให้ตัวอย่างที่หนึ่ง แตกต่างจาก ตัวอย่างที่สอง ในภาษาที่สอง สัญลักษณ์ = สงวนไว้เฉพาะใช้ เป็นตัวปฏิบัติการเปรียบเทียบอย่างเดียว (เช่น ใน "if A = B then ...") และด้วยเหตุนี้ สัญลักษณ์ที่เป็นเพียงหนึ่งเดียวเท่านั้น := จึงแยกการกำหนดค่าออกจากการเปรียบเทียบ ลักษณะนี้ เป็นคุณสมบัติที่แตกต่างจากภาษาที่หนึ่ง ดังนั้น ปัญหาง่ายๆ ของการเลือก สัญลักษณ์ที่จะใช้แทนการปฏิบัติการ จึงมี อิทธิพลโดยตรงกับ การแสดงออกอย่างชัดเจนของภาษา

(2) นิยามได้เป็นอย่างดี หมายถึงวากยสัมพันธ์ และความหมายของภาษา ต้องไม่กำกวม มีความพ้องกันภายใน และสมบูรณ์ (By "well-definedness", we mean that the language's syntax and semantics are free of ambiguity, are internally consistent, and complete.) ดังนั้น นักทำภาษาให้เกิดผล ของภาษาซึ่งนิยามได้อย่างดี ควรจะมีข้อกำหนดรายละเอียดที่สมบูรณ์ ของ รูปแบบ การแสดงออกของภาษาทั้งหมด และความหมายต่างๆ ภายในบทนิยามของมัน นักเขียนโปรแกรม ก็เช่นเดียวกัน ควรจะสามารถคาดคะเนพฤติกรรม ของ การแสดงออกแต่ละอย่าง ได้อย่างถูกต้อง ก่อน การกระทำจริง (actually executed) ด้วยเหตุนี้ ภาษาที่นิยามได้อย่างดี จึงสนับสนุนการเคลื่อนย้ายง่าย และการพิสูจน์ได้

ตัวอย่าง ภาษายุคแรก บางภาษา มีการนิยามได้ไม่ดี เช่น การใช้ DO loops ในภาษา FORTRAN

DO 10 I = 1

ได้ถูกวิเคราะห์กระจาย (to be parsed) เป็น คำสั่งกำหนดค่า เพราะว่า ที่ว่าง (spaces) ไม่สำคัญ และตัวแปร (เช่น DO10I) ไม่จำเป็นต้องมีการประกาศ ส่วนภาษา FORTRAN เวอร์ชันเมื่อเร็วๆ นี้ ปัญหาเหล่านี้ส่วนใหญ่ได้ถูกขจัดออกไป อันเนื่องมาจาก วิวัฒนาการ ของวิธีกำหนดรายละเอียดภาษาทางการ และกระบวนการมาตรฐานของภาษา (due to the evolution of formal language description methods and language standardization procedures.)

(3) ชนิดข้อมูลและโครงสร้าง หมายถึง ความสามารถของภาษา ที่จะสนับสนุนความหลากหลายของค่าข้อมูล (จำนวนเต็ม จำนวนจริง สายอักขระ ตัวชี้ เป็นต้น) และการรวมกลุ่มของ ข้อมูลเหล่านี้

(By "data types and structures", we mean the ability for a language to support a variety of data values (integer, real, strings, pointers, etc.) and nonelementary collections of these.) ข้อมูลกลุ่มหลังนี้ รวม แถวลำดับ และระเบียนเป็นเบื้องต้น ทั้งนี้ยังรวม โครงสร้างข้อมูลพลวัต เช่น รายการโยง แถวคอย กองซ้อน และต้นไม้

(4) สภาพมอดุลาร์ มอง 2 ด้านคือ

ความสนับสนุนของภาษา สำหรับ การเขียนโปรแกรมย่อย และความสามารถในการขยายของภาษา ในความรู้สึก ของ การอนุญาตให้มี ตัวปฏิบัติการ และชนิดข้อมูล ซึ่ง นักเขียนโปรแกรม นิยามขึ้นมาใช้เองได้ ("Modularity" has two aspects: the language's support for subprogramming and the language's extensibility in the sense of allowing programmer-defined operators and data types.) โดยการเขียนโปรแกรมย่อย หมายถึง ความสามารถที่จะนิยาม โปรซีเดอร์ และฟังก์ชัน (โปรแกรมย่อย) ที่เป็นอิสระ และการติดต่อผ่าน พารามิเตอร์ หรือ ตัวแปรส่วนกลาง ด้วย **โปรแกรมเรียก** (invoking program) ภาษาส่วนใหญ่ มีความเข้มแข็ง (strong) ในส่วนหลังนี้ ในขณะที่มีภาษาเพียงจำนวนน้อย ที่มีความสามารถในการขยาย ใน ความรู้สึกที่บรรยายข้างต้น ในหนังสือเล่มนี้ บท Pascal และ Ada จะแสดงให้เห็น ความสามารถในส่วนขยายของภาษา

(5) ในการประเมิน สิ่งอำนวยความสะดวก อินพุท-เอ้าพุท เรามองที่ การสนับสนุน

ของภาษา สำหรับแฟ้มข้อมูลแบบลำดับ ดรรชนี และ เข้าถึงโดยสุ่ม เช่นเดียวกับ การสนับสนุนของภาษา สำหรับฐานข้อมูล และฟังก์ชัน ในการค้นคืนสาระสนเทศ (In evaluating a language's input-output facilities, we are looking at its support for sequential, indexed, and random access files, as well as its support for database and information retrieval functions.) แฟ้มโดยทั่วไปอยู่ในหน่วยเก็บสำรอง (เข้าถึงโดยตรงหรือแถบบันทึกแม่เหล็ก) ตรงกันข้ามกับ โครงสร้างข้อมูล ซึ่ง โดยทั่วไปอยู่ใน หน่วยความจำหลัก แฟ้มข้อมูล ปกติจะมีขนาดใหญ่มากเกินไป สำหรับที่จะ เอาระเบียนทั้งหมดของมัน เก็บเข้าไปภายในหน่วยความจำเพียงครั้งเดียว ดังนั้น ยุทธวิธีการเรียนโปรแกรมต่างๆ จึงนำมาประยุกต์ใช้ กับแฟ้ม มากกว่าที่จะเป็น โครงสร้างข้อมูล

(6) ภาษาซึ่งมี <u>การเคลื่อนย้ายได้</u> เป็น อีกหนึ่งสิ่งซึ่งถูกทำให้เกิดผลบนคอมพิวเตอร์ต่างๆ นั่นคือ การออกแบบภาษาเพื่อให้มันสัมพันธ์กับ "การเป็นอิสระจากเครื่อง" (A language which has "portability" is one which is implemented on a variety of computers. That is, its design is relatively "machine independent") ภาษาซึ่งมีเวอร์ชั่นมาตรฐาน ANS ปกติเคลื่อนย้ายได้ง่ายกว่าภาษาที่ ไม่มี (แต่ก็มีข้อยกเว้น) ภาษาซึ่งนิยามอย่างดี มักจะเคลื่อนย้าย ได้ง่ายกว่า ภาษาอื่น ๆ ดังนั้น คงจะไม่ประหลาดใจที่พบว่า ANS FORTRAN (1977) หรือ ANS COBOL (1974) สนับสนุน บนคอมพิวเตอร์เครื่องใหญ่ได้เกือบทั้งหมด เพราะว่า มันเป็นภาษามาตรฐาน เช่นเดียวกัน ไม่น่าประหลาดใจที่พบว่าภาษา LISP หรือ C ทำให้เกิดผลบนคอมพิวเตอร์ได้หลากหลาย เพราะว่ามันถูกนิยามมาอย่างดี ถึงแม้ว่าจะ ไม่เป็นภาษามาตรฐาน ก็ตาม

ตรงกันข้ามกับ ภาษา BASIC ซึ่งมีความสามารถในการเคลื่อนย้ายได้ น้อยกว่าภาษาเหล่านี้ ถึงแม้ว่า ภาษา BASIC จะถูกทำให้เกิดผล บนคอมพิวเตอร์ได้เกือบทุกเครื่อง ก็ตาม แต่การทำให้เกิดผลแต่ละครั้งมีการกระทบบ้างกับ ส่วนย่อย (dialect) ที่แตกต่างกัน ตัวอย่างเช่น ส่วนย่อยบางอย่าง สนับสนุนกลุ่มหนึ่งของฟังก์ชันเชิงเรขภาพ บางอย่างสนับสนุนอีกสิ่งหนึ่ง และบางอย่างไม่สนับสนุนสิ่งใดทั้งสิ้น ดังนั้น เมื่อมีความพยายามที่จะ เคลื่อนย้าย ("port") โปรแกรมภาษา BASIC จากคอมพิวเตอร์เครื่องหนึ่ง ไปยังคอมพิวเตอร์อีกเครื่องหนึ่ง นักเขียนโปรแกรม ต้อง ระมัดระวังทั้งส่วนย่อยที่แตกต่าง และส่วนที่ตั้งใจเขียนใหม่ ของโปรแกรม ให้กลมกลืนกัน ภาษาเคลื่อนย้ายได้ แท้จริงแล้ว ไม่ควรจะมี สิ่งปลีกย่อยเช่นนี้ และโปรแกรมควรจะสามารถวิ่ง (run) บนคอมพิวเตอร์ที่แตกต่างกันได้โดยไม่ต้องมีการเปลี่ยนแปลงใดๆ เลย

(7) ภาษาที่มีประสิทธิภาพ เป็นอีกสิ่งหนึ่ง ซึ่งทำให้ การแปลชุดคำสั่งและการกระทำการ ทำได้เร็ว บนคอมพิวเตอร์ ซึ่งมันถูกทำให้เกิดผล (An "efficient" language is one which permits fast compilation and execution on the machines which it is implemented.) สมัยก่อน FORTRAN และ COBOL เป็นภาษาที่ค่อนข้างมีประสิทธิภาพในงานสาขาที่มันประยุกต์ใช้ ในขณะที่ ภาษา PL/1 เป็นภาษาที่ไม่มีประสิทธิภาพเมื่อนำมาเปรียบเทียบกัน โปรแกรมกรณีศึกษาของเรา จะแสดงให้เห็นความแตกต่างเหล่านี้

เช่นที่กล่าวข้างต้น ตัวแปลชุดคำสั่ง (compilers) โดยปกติ จะมีประสิทธิภาพ มากกว่า ตัวแปลคำสั่ง (interpreters) ตัวอย่างเช่น โปรแกรมภาษา LISP ซึ่ง compiled แล้ว จะ execute หลายครั้ง ทำงานได้เร็วกว่า โปรแกรมเดียวกัน ซึ่ง วิ่ง ภายใต้ ตัวแปลคำสั่ง อย่างไรก็ตาม การใช้ตัวแปลคำสั่ง มีข้อดีอื่น ๆ ระหว่างการพัฒนาโปรแกรม ในกรณีนี้ ประสิทธิภาพของเครื่อง คอมพิวเตอร์ เป็นสิ่งสำคัญอันดับที่สอง กับเวลาของนักเขียนโปรแกรมที่จะทำให้โปรแกรมสำเร็จ

(8) บางภาษา มี "pedagogy" ดีกว่าภาษาอื่น ๆ นั่นคือ โดยเนื้อแท้แล้ว สอนง่ายกว่า เรียนรู้ได้ง่ายกว่า มีตำราที่ดีกว่า ทำให้เกิดผลในสิ่งแวดล้อม ของ การพัฒนาโปรแกรมดีกว่า เป็นภาษาที่รู้จักกันแพร่หลาย และถูกใช้โดยนักเขียนโปรแกรมที่ดีที่สุดในงานด้านต่าง ๆ (Some languages have better 'pedagogy" than others. That is, they are intrinsically easier to teach and to learn; they have better text-books; they are implemented in a better program development environment; they are widely known and used by the best programmers in an application area.) บางภาษาเช่น BASIC และ Pascal พัฒนาขึ้นมาเพื่อจะให้เป็นเครื่องมือสำหรับการเรียนการสอน (pedagogical tools) โดยเฉพาะ ทุกวันนี้ ภาษา BASIC ซึ่งเป็นที่นิยมใน โรงเรียนประถม และโรงเรียนมัธยม ซึ่งสอน การเขียนโปรแกรม ในขณะที่ภาษา Pascal เป็นที่นิยมกัน เช่นเดียวกัน ในหลักสูตรคอมพิวเตอร์ ใน มหาวิทยาลัย

(9) ใช้ได้ทั่วไป เกณฑ์การประเมินภาษาข้อสุดท้าย หมายถึง ภาษานั้นเป็นประโยชน์ใน พิสัยที่กว้างของงานด้านเขียนโปรแกรม ("Generality", the last of our nine language evaluation criteria, means that a language is useful in a wide range of programming applications.) ตัวอย่างเช่น ภาษา APL ใช้ในงานด้านคณิตศาสตร์ เกี่ยวกับ พีชคณิตเชิงเมตริกซ์ (matrix algebra) และงานด้านธุรกิจ business application) ได้ดีเช่นเดียวกัน ในบรรดาภาษาปัจจุบันทั้งหมด ภาษาซึ่ง ใช้ได้ทั่วไป

มากที่สุด น่าจะเป็นภาษา PL/1 ภาษานี้ตั้งใจออกแบบ มาเพื่อให้เป็นภาษาเอนกประสงค์ (general-purpose language) และขนาดที่แท้จริงของมัน พิสูจน์ว่าเป็นเช่นนั้น ในบทที่ 5 เราจะเห็น ข้อดี และ ข้อไม่ดีบางอย่าง ของการใช้ ภาษาเอนกประสงค์ เช่น PL/1 ภาษาอื่น ๆ ส่วนใหญ่ออกแบบมาเพื่อวัตถุประสงค์และการสนทนาที่แคบกว่า (for a narrow purpose and audience.)

ดังนั้น เกณฑ์ทั้ง 9 ข้อนี้ จะให้เราใช้ประเมินแต่ละภาษา ในตำรานี้ โดยมาตราส่วนที่เป็นแบบเดียวกัน (by a uniform scale) สิ่งที่ยอมรับคือ เกณฑ์เหล่านี้ทั้งหมด ไม่ได้เป็นการวัดเชิงปริมาณ และสรุปขั้นสุดท้าย เช่น เกณฑ์ที่มีประสิทธิภาพ (Admittedly, they are not all as quantifiable and conclusive as the efficiency criterion) ซึ่งวัดในหน่วยของ นาโนวินาที (nanosecond = 10^{-9} วินาที) และยังมีส่วนที่สำคัญกว่าอีกมากมาย เช่น "ฉันใช้ภาษา X เพราะว่า ปกติฉันใช้ภาษานี้" แม้ว่า เราจะไม่ได้คาดคิดตรงนี้ว่า จะเปลี่ยนความในใจมากมาย เกี่ยวกับว่า ภาษาชุดคำสั่งอะไรดีที่สุด เราหวังว่าจะสามารถกระตุ้นผู้อ่าน ให้มีความรอบคอบและวิเคราะห์อย่างมีเหตุผลเกี่ยวกับ การเลือก ภาษาที่ดี

1.5 การออกแบบภาษาชุดคำสั่ง (Programming Language Design)

เกณฑ์การประเมินภาษาหลายข้อ ที่นำเสนอข้างต้น สะท้อนความสนใจ ของ นักเขียนโปรแกรม ขณะที่สิ่งอื่น ๆ สะท้อน ความสนใจของนักออกแบบภาษา การออกแบบภาษาชุดคำสั่งเป็นการรวมและประนีประนอม (comprises and compromise) ความรู้ จากหลายสาขา โดยเฉพาะสถาปัตยกรรมคอมพิวเตอร์ ภาษาทางการ ทฤษฎีออโตมาตา และภาษาศาสตร์ เนื่องจากการศึกษาภาษาชุดคำสั่ง คือการยอมรับโดยผู้คน ซึ่งจะเข้ามามีส่วนร่วมในการออกแบบภาษาในอนาคต เราให้ความสนใจพิเศษ กับ หลักพื้นฐานของการออกแบบภาษา ในตำราเล่มนี้ ซึ่งมีให้เลือกระหว่างแต่ละภาษา ใน บทต่อไปนี้

บทที่ 6 การออกแบบภาษา : วากยสัมพันธ์

บทที่ 10 การออกแบบภาษา : ความหมาย

บทที่ 15 การออกแบบภาษา : เกี่ยวกับการปฏิบัติ

วากยสัมพันธ์ ของภาษาชุดคำสั่งส่วนมาก จะให้ บทนิยามอย่างเคร่งครัด หรือเป็นทางการ (The "syntax" of most programming languages permits fairly rigorous or "formal" definition.) นั่นคือ นิพจน์เชิงคณิตศาสตร์ อย่างสรุป นำไปใช้บอกคุณลักษณะ ซึ่ง ข้อความสั่งอันไหน ถูกต้องเชิงวากยสัมพันธ์ หรือ ข้อความสั่งอันไหน ไม่ถูกต้อง (That is, a

brief mathematical expression can be used to characterize which statements are syntactically valid and which not.)
รูปแบบทางการนี้ ทำให้งานของ นักทำภาษาให้เกิดผล ง่ายขึ้น ซึ่งโปรแกรมแปลชุดคำสั่ง หรือ โปรแกรมแปลคำสั่ง ต้องวิเคราะห์เชิงวากยสัมพันธ์ ข้อความของโปรแกรมก่อนที่จะดำเนินการ หัวข้อ ใน วากยสัมพันธ์ จะรวมทั้ง อุปกรณ์รายละเอียดอย่างเป็นทางการ (its formal description device) เซตตัวอักขระของภาษา (the language's character) การขจัดความคลุมเคลือในภาษา (the elimination of ambiguity in the language) การกระจายคำ (parsing) การวิเคราะห์ศัพท์ (lexical analysis) และอิทธิพลของวากยสัมพันธ์ บน รูปแบบการเขียนโปรแกรม และในทางกลับกันด้วย หัวข้อเหล่านี้ จะพัฒนา และแสดงให้เห็นใน บทที่ 2

ความหมาย ในทางตรงกันข้าม เกี่ยวกับ ความหมาย ของรูปแบบเชิงวากยสัมพันธ์ ต่าง ๆ ในโปรแกรม และ การเลือก การแทนที่เครื่อง อย่างเหมาะสม ("Semantics," on the other hand, deals with the meaning of the various syntactic forms in a program and the selection of suitable machine representation for them.) หัวข้อที่สำคัญ ของ ความหมายของภาษา ได้แก่ การผูกโยง (binding), การบังคับ (coerion) การทำให้เกิดผลของชนิดและโครงสร้างข้อมูลต่าง ๆ (implementation of different data types and structures) การจัดสรรหน่วยเก็บพลวัต (dynamic storage allocation) การวินิจฉัยเวลาดำเนินงาน (run-time diagnostics) ความสามารถในการแก้ไขข้อยกเว้น (exception handling) ขอบเขต (scope) separately compiled procedures, input-ouput และ code optimization หัวข้อเหล่านี้ จะพัฒนาในบทที 10

เกี่ยวกับการปฏิบัติ ของ การออกแบบภาษาชุดคำสั่ง อภิปรายในบทที่ 15 ในที่นี้เรา พิจารณา หัวข้อเพิ่มเติม ใน การทำให้เกิดผล และรูปแบบของการเขียนโปรแกรม ซึ่งจะมีอิทธิพล ลักษณะของภาษาร่วมสมัย

("Pragmatics" of programming language design constitute the discussion in Chapter 15. Here, we consider additional issue in implementation and programming style which tend to influence contemporary language features.)
หัวข้อในการทำให้เกิดผล ได้แก่ the management of concurrency, optimization,

programming environments, and diagnosics. หัวข้อรูปแบบการเขียนโปรแกรม ได้แก่ ปรัชญา เช่น "logic programming", "object programming", "data abstraction", "functional programming", "software engineering", ความสัมพันธ์ระหว่างหัวข้อเหล่านี้ และความสัมพันธ์ ของสิ่งเหล่านี้ กับภาษาต่างๆ ที่นำเสนอในบทต่างๆ ของหนังสือเล่มนี้

บทที่ 15 ยังรวมถึง การประเมินราคา (assessment) ของภาษาชุดคำสั่ง จากการมอง หลายด้าน ของนักเขียนโปรแกรม นักออกแบบภาษา และนักทำภาษาให้เกิดผล ที่นี้ เราทำการประเมินอย่างเปรียบเทียบ หลายภาษา ซึ่งได้ศึกษามาแล้ว ในบทต้น ๆ การเปรียบเทียบนั้น ใช้หลัก 9 ข้อที่ได้แนะนำไว้ รวมกับ การทำให้เกิดผลกรณีศึกษา ในภาษาต่าง ๆ ผลลัพธ์เป็นเรื่องน่าสนใจ แลในกรณีส่วนใหญ่ ไม่น่าประหลาดใจ แต่เพราะว่า มันอยู่บนหลักค่อนข้างเป็นเชิงรูปธรรม และด้วยประสบการณ์

ผลลัพธ์เหล่านี้ เชื่อถือได้มากกว่า สัญชาตญาณของเราอย่างเดียว นอกจากนี้แล้ว ระเบียบวิธี ของ การประเมินภาษา ที่ใช้ในที่นี้ สามารถนำไปประยุกต์ใหม่ กับภาษาอื่น ๆ นอกหนือจากภาษาเหล่านี้ และ ในวิธีที่กระทำได้เท่าเทียมกัน (and in an equally even handed way.)

TABLE 1 Genealogy of Programming Languages

Language	Year	Originator	Predecessor Languages	Intened Purpose
FORTRAN	1954-57[a]	J.Backus(IBM)		Numeric computation
ALGOL 60	1958-60[b]	Committee	FORTRAN	Numeric Computation
COBOL	1959-60[b]	Committee		Business data processing
APL	1956-60[b]	K. Iverson (Harvard)		Array processing
LISP	1956-62[a]	J. McCarthy (MIT)		Symbolic computation

TABLE 1 (Cont.)

Language	Year	Originator	Predecessor Languages	Intened Purpose
SNOBOL4	1962-66[a]	R. Griswold (Bell Labs)	-	String processing
PL/I	1963-64[b]	IBM Committee	FORTRAN ALGOL 60 COBOL	General purpose
SIMULA 67	1967[c]	O.J. Dahl et al. (Norwegian Computing Center)	ALGOL 60	General purpose simulation
ALGOL 68	1963-68[b]	Committee	ALGOL 60	General purpose
Rliss	1971[c]	W.Wulf et al. (Carnegie Mellon U.)	ALGOL 68	Systems programming
Pascal	1971[c]	N.Wirth(ETH Zurich)	ALGOL 60	General and education purpose. Supporting structured programming
PROLOG	1972[a]	A.Colmerauer (Marseille, France)	-	Artificial intelligence
C	1974[c]	D.Ritchie(Bell Labs)	ALGOL 68 BCPL[d]	Systems programming
Mesa	1974[b]	Xerox PARC	Pascal SIMULA 67	Systems programming

TABLE 1 (Cont.)

Language	Year	Originator	Predecessor Languages	Intened Purpose
SETL	1974[a]	J.Schwartz (NYU)	-	Very high-level programming
Concurrent Pascal	1975[c]	P.Brinch Hansen	Pascal	Concurrent programming
CLU	1974-77[a]	B.Liskov et al. (MIT)	SIMULA 67	Supporting a methodology based on abstraction
Euclid	1977[c]	Committee	Pascal	Verifiable systems programs
Gypsy	1977[c]	D.Good et al.(U. of Texas-Austin)	Pascal	Verifiable systems programs
Modula-2	1977[c]	N.Wirth (ETH Zurich)	Pascal Mesa	Systems programming, real-time
Ada	1979[c]	J.Ichbiah et al. (CII Honeywell Bull)	Pascal SIMULA 67	General purpose, embedded applications, real-time
Smalltalk	1971-80[a]	A.Kay (XEROX PARC)	SIMULA 67	Personal computing environment

[a] Language design and initial implementation.

[b] Language design.

[c] First official language description.

[d] See (Richards 1969).