บทนำ

1.1 แนวคิดของภาษาโปรแกรม

1.2 การออกแบบภาษาโปรแกรม

1.3 เกณฑ์ในการพิจารณาภาษา

1.4 การจัดกลุ่มของภาษา

1.5 การแปลภาษา

1.6 สภาพแวดล้อมในการพัฒนาโปรแกรม

วัตถุประสงค์

- เพื่อแนะนำแนวคิดและที่มาของภาษาโปรแกรม
- เพื่อให้เข้าใจแนวคิดในการออกแบบภาษา และเกณฑ์ที่ใช้ในการพิจารณาภาษาโปรแกรม
- เพื่อให้รู้จักภาษาโปรแกรมหลายกลุ่ม และเข้าใจแนวคิดในการจัดกลุ่มภาษาโปรแกรม
- เพื่อให้เข้าใจขั้นตอนการแปลภาษาแบบอินเตอร์พรีตเตอร์ คอมไพเลอร์ และไฮบริด

1.1 แนวคิดของภาษาโปรแกรม

คอมพิวเตอร์เป็นอุปกรณ์อิเล็กทรอนิกส์ประเภทหนึ่งที่ทำงานตามชุดคำสั่งที่รับเข้าไป เราเรียกชุดคำสั่งเหล่านั้นว่าโปรแกรม การทำงานภายในคอมพิวเตอร์จะควบคุมด้วยสัญญาณไฟฟ้าซึ่งมีสองลักษณะคือเปิดและปิด ซึ่งสามารถแทนด้วยเลข 0 และ 1 ในระบบเลขฐานสอง ดังนั้นภาษาที่ใช้ในการสื่อสารภายในเครื่องจึงใช้ระบบเลขฐานสอง หรือเรียกว่าภาษาเครื่อง (machine Language) ผู้เขียนโปรแกรมหรือโปรแกรมเมอร์ในยุคแรกจึงเขียนโปรแกรมด้วยภาษาเครื่อง เพื่อใช้ควบคุมหน่วยประมวลผลให้ทำงานตามที่ต้องการ หน่วยประมวลผลสามารถเข้าใจคำสั่งเหล่านี้ได้ทันทีและสั่งให้เครื่องทำงานตามคำสั่ง เช่น การบวก การเปรียบเทียบ การเคลื่อนย้ายข้อมูลจากตำแหน่งหนึ่งไปยังอีกตำแหน่งหนึ่ง โดยจะทำเช่นนี้ไปเรื่อยๆ ทีละคำสั่งตามลำดับที่ระบุ จากตัวอย่างโปรแกรมในรูป 1.1 เป็นโปรแกรมในการคำนวณหาค่าหารร่วมมากของเลขจำนวนเต็ม 2 จำนวน ตามอัลกอริทึมของ Euclid เขียนด้วยภาษาเครื่องสำหรับเครื่องตระกูล x86 (Pentium)แสดงอยู่ในรูปเลขฐาน 16 จะเห็นได้ว่าภาษาเครื่องเป็นภาษาที่มนุษย์เข้าใจได้ยาก เขียนก็ยาก แต่โปรแกรมเมอร์ในยุคแรกนั้นก็ยังเลือกที่จะเขียนโปรแกรมด้วยภาษาเครื่อง เนื่องจากพวกเขาเชื่อว่า เวลาที่คอมพิวเตอร์ใช้ในการทำงานมีค่ามากกว่าเวลาที่โปรแกรมเมอร์ใช้ในการเขียนโปรแกรม

```
55 89 e5 53 83 ec 04 83 e4 f0 e8 31 00 00 00 89 c3 e8 2a 00
00 00 39 c3 74 10 8d b6 00 00 00 00 39 c3 7e 13 29 c3 39 c3
75 f6 89 1c 24 e8 6e 00 00 00 8b 5d fc c9 c3 29 d8 eb eb 90
```

รูปที่ 1.1: โปรแกรมหารร่วมมากเขียนด้วยภาษาเครื่องสำหรับเครื่องในตระกูล x86

เมื่อโปรแกรมเมอร์เขียนโปรแกรมที่มีขนาดใหญ่ขึ้น โอกาสที่จะเขียนโปรแกรมผิดพลาดก็มีมากขึ้นตามไปด้วย ดังนั้นจึงได้พัฒนาภาษาโปรแกรม เพื่อให้อ่านและเขียนได้ง่ายขึ้น ช่วยลดข้อผิดพลาดให้น้อยลง โดยออกแบบให้ใช้ภาษาสัญลักษณ์ที่เขียนด้วยตัวอักษรภาษาอังกฤษหรือรหัสนีโมนิก (mnemonic) แทนคำสั่งต่างๆ ที่เขียนด้วยเลขฐานสอง และเรียกภาษาโปรแกรมนี้ว่าภาษาแอสเซมบลี (Assembly language) ตัวอย่างโปรแกรมในรูป 1.1 เป็นโปรแกรมหารร่วมมากที่เขียนด้วยภาษาแอสเซมบลีสำหรับเครื่อง x86

```
        movl    esp, ebp                    subl  eax, ebx
        pushl ebx                   B:      cmpl  eax, ebx
        subl    4, esp                      jne   A
        andl    -16, esp            C:      movl  ebx, (esp)
        call    getint                      call  put int
        movl    eax, ebx                    movl  -4(ebp), ebx
        call    getint                      leave
        cmpl    eax, ebx                    ret
        je C                    D:  subl  ebx, eax
A:      cmpl    eax, ebx                    jmp   B
```

รูปที่ 1.2: โปรแกรมหารร่วมมากเขียนด้วยภาษาแอสเซมบลีสำหรับเครื่อง x86

ภาษาแอสเซมบลีออกแบบให้หนึ่งคำสั่งในภาษาแอสเซมบลี สามารถแทนด้วยหนึ่งคำสั่งในภาษาเครื่อง โดยมีโปรแกรมระบบที่ทำหน้าที่ในการแปลภาษาสัญลักษณ์ไปเป็นภาษาเครื่องเรียกว่า แอสเซมเบลอร์ (Assembler) ข้อเสียของภาษาแอสเซมบลี คือ เป็นภาษาที่ขึ้นอยู่กับเครื่องถ้าคอมพิวเตอร์ที่มีสถาปัตยกรรม

หรือชุดคำสั่ง (instruction set) ต่างกันก็จะมีภาษาแอสเซลบลีเฉพาะสำหรับเครื่องนั้น ทำให้โปรแกรมเมอร์ต้องเรียนรู้คำสั่งที่ใช้กับเครื่องที่ตนทำงาน ทั้งภาษาเครื่องและภาษาแอสเซมบลีจัดเป็นภาษาระดับต่ำ (low-level language) เพราะเป็นภาษาที่ขึ้นอยู่กับเครื่อง อีกทั้งยังอ่านและเข้าใจได้ยากเมื่อคอมพิวเตอร์มีการพัฒนาไปอย่างรวดเร็ว จึงเป็นการยากที่จะเขียนโปรแกรมขึ้นเพื่อใช้กับเครื่องที่พัฒนาขึ้นใหม่ทุกๆ ครั้ง และรายละเอียดของภาษาแอสเซมบลีก็มีมากขึ้นเรื่อยๆ จึงเริ่มมีความต้องการภาษาที่ไม่ขึ้นกับเครื่อง จึงเป็นที่มาของการพัฒนาภาษาโปรแกรมในยุคที่สามเรียกว่า ภาษาระดับสูง (high-level language) ซึ่งออกแบบให้เขียนคำสั่งต่างๆ ในรูปคำในภาษาอังกฤษภาษา FORTRAN เป็นภาษาระดับสูงภาษาแรกที่พัฒนาขึ้นเพื่อใช้งานด้านการคำนวณต่างๆสำหรับงานด้านวิทยาศาสตร์ ต่อมาได้มีการพัฒนาภาษาระดับสูงอื่นๆ ตามมา เช่น LISP, COBOL, Algol โปรแกรมระบบที่ทำหน้าที่แปลภาษาระดับสูงให้เป็นภาษาแอสเซมบลี หรือภาษาเครื่องเรียกว่าคอมไพเลอร์ซึ่งมีการทำงานที่ซับซ้อนกว่าแอสเซมเบลอร์ เพราะหนึ่งคำสั่งในภาษาระดับสูงอาจแปลงเป็นภาษาระดับต่ำได้มากกว่าหนึ่งคำสั่ง ในช่วงแรกภาษาฟอร์แทรนไม่ได้รับความนิยมมากนัก เพราะโปรแกรมเมอร์ส่วนใหญ่ยังสามารถเขียนโปรแกรมด้วยภาษาแอสเซมบลีที่ทำงานได้ไวกว่าภาษาที่แปลด้วยคอมไพเลอร์ ต่อมาช่องว่างในเรื่องประสิทธิภาพการแปลนี้ได้ลดลงจนกลายเป็นตรงกันข้ามในที่สุด เนื่องจากฮาร์ดแวร์มีความซับซ้อนมากขึ้น และมีการพัฒนาคอมไพเลอร์ให้ดีขึ้น ทำให้สามารถสร้างโปรแกรมได้ดีกว่าที่มนุษย์เขียนขึ้นเอง แม้จะมีบางกรณีที่มนุษย์ทำได้ดีกว่าแต่ความเร็วของคอมพิวเตอร์ และขนาดของโปรแกรมมีผลต่อการลดต้นทุนของมนุษย์ ไม่ว่าจะเป็นต้นทุนในแง่การสร้างโปรแกรม การปรับปรุงและแก้ไขโปรแกรม ดังนั้น ค่าแรงของโปรแกรมเมอร์ในปัจจุบันจึงมีค่ามากกว่ามูลค่าของเครื่องคอมพิวเตอร์

ภาษาโปรแกรม (programming language) มีข้อแตกต่างจากภาษาธรรมชาติที่สำคัญอยู่สองประการประการแรก ภาษาโปรแกรมสามารถใช้ในการสื่อสารระหว่างมนุษย์กับเครื่องคอมพิวเตอร์ได้ และยังสามารถใช้เป็นสื่อกลางในการสื่อสารกันระหว่างโปรแกรมเมอร์อีกด้วย ประการที่สอง ภาษาโปรแกรมมีขอบเขตที่ใช้ในการแสดงความหมายแคบกว่าภาษาธรรมชาติ เพราะเพียงต้องการช่วยให้การสื่อสารแนวความคิดทางคอมพิวเตอร์ทำได้ง่าย ภาษาโปรแกรมจึงมีแนวคิดที่แตกต่างจากภาษาธรรมชาตินักศึกษาในสาขาวิชาวิทยาการคอมพิวเตอร์จึงจำเป็นต้องศึกษาวิชาเกี่ยวกับแนวคิดของภาษาโปรแกรมโดยจะเน้นที่หลักการของภาษาระดับสูงในฐานะเป็นผู้ใช้ภาษาของวันนี้และภาษาโปรแกรมใหม่ๆ ที่จะเกิดขึ้นในอนาคต นักคอมพิวเตอร์จำเป็นต้องศึกษาเกี่ยวกับหลักการของภาษาโปรแกรมด้วยเหตุผลหลายประการ ดังจะกล่าวเป็นตัวอย่างพอสังเขปต่อไปนี้

- **เพื่อเพิ่มความสามารถในการเขียนโปรแกรม** ความรู้ความเข้าใจภาษาโปรแกรมอย่างกระจ่าง รู้ถึงลักษณะต่างๆ ของภาษา จะช่วยเพิ่มความสามารถในการถ่ายทอดอัลกอริทึมให้อยู่ในรูปของโปรแกรมได้ดีขึ้น
- **เพื่อเลือกภาษาโปรแกรมให้เหมาะสมกับงาน** การศึกษาถึงข้อดีและข้อเสียของภาษาโปรแกรมหลากหลายรูปแบบ จะช่วยในการนำไปประยุกต์ใช้ให้เหมาะสมกับงาน
- **เพื่อเพิ่มความสามารถในการเรียนรู้ภาษาใหม่** ภาษาโปรแกรมโดยทั่วไปมีหลักการที่คล้ายคลึงกัน จึงเป็นพื้นฐานที่ดีในการเรียนรู้ภาษาโปรแกรมใหม่ๆ ได้ง่ายขึ้น
- **เพื่อเพิ่มประสิทธิภาพในการเขียนโปรแกรม** เมื่อนักศึกษาเข้าใจเหตุผลของการออกแบบภาษา เข้าใจวิธีการต่างๆ ที่ใช้ในการพัฒนาภาษาโปรแกรม ทำให้เลือกวิธีการในการเขียนโปรแกรมที่มีประสิทธิภาพมากขึ้น

- **เพื่อพัฒนาการในวงการคอมพิวเตอร์** ในอนาคตนักศึกษาอาจจำเป็นจะต้องออกแบบภาษาใหม่เพื่อใช้งานเฉพาะบางอย่าง จึงควรจะต้องรู้ลึกซึ้งในแง่ของการออกแบบภาษา เพื่อเป็นพื้นฐานที่สำคัญในการสร้างภาษาโปรแกรมภาษาใหม่ได้อย่างถูกต้องเหมาะสม ทั้งยังอาจจะก่อให้เกิดพัฒนาการใหม่ๆ ในวงการคอมพิวเตอร์

ดังนั้น การศึกษาภาษาโปรแกรมและการนำไปใช้งานเพียงภาษาเดียวจึงไม่สามารถก่อให้เกิดความรู้ความเข้าใจได้อย่างถ่องแท้ ในรายวิชานี้จะกล่าวถึงหลักการทั่วไปของภาษาโปรแกรม แนวคิดในการออกแบบภาษา และกล่าวถึงภาษาโปรแกรมโดยยกตัวอย่างจากหลายภาษาและหลายรูปแบบ

1.2 การออกแบบภาษาโปรแกรม

ภาษาโปรแกรมเป็นเครื่องมือที่สั่งให้คอมพิวเตอร์ทำงานได้ตามที่ต้องการ และเป็นเครื่องมือหรือวิธีการที่โปรแกรมเมอร์ใช้ในการสื่อสารกันด้วย นอกจากนี้ยังอาจมองได้ว่าภาษาโปรแกรมเป็นสัญลักษณ์ที่ใช้สำหรับขั้นตอนการแก้ปัญหาซึ่งเป็นวิธีการแสดงความสัมพันธ์ระหว่างแนวคิด เพื่อใช้ในการควบคุมอุปกรณ์คอมพิวเตอร์ปัจจุบันมีภาษาโปรแกรมระดับสูงที่ใช้งานอยู่จำนวนมากมายหลายพันภาษา และยังมีการพัฒนาภาษาโปรแกรมใหม่อยู่เรื่อยๆ นักศึกษาในระดับปริญญาตรีหรือโปรแกรมเมอร์ทั่วไปสามารถใช้งานภาษาได้ไม่กี่ภาษา แล้วเพราะเหตุใดจึงมีภาษาให้ใช้มากมาย เหตุผลก็มีหลากหลายเช่นกัน ตัวอย่างเช่น

- **วิวัฒนาการ** วิทยาการคอมพิวเตอร์ถือเป็นศาสตร์ที่เกิดขึ้นไม่นานนัก และยังมีความพยายามที่จะค้นหาวิธีการที่ดีขึ้นอยู่ตลอดเวลา ในช่วงปลายยุค 1960 และต้นยุค 1970 ภาษาที่ใช้การควบคุมแบบ goto ในยุคแรกเช่น FORTRAN, COBOL และ BASIC ถูกแทนที่ด้วยภาษาแบบมีโครงสร้าง (structure programming) ซึ่งใช้คำสั่งควบคุมแบบคำสั่งวนซ้ำคำสั่งเลือกทำ หรือคำสั่งอื่นๆ ในลักษณะเดียวกันมาแทนคำสั่ง goto ในยุคปลาย1980 ภาษาแบบมีโครงสร้างก็ถูกแทนที่ด้วยภาษาเชิงวัตถุ เช่น Smalltalk และ C++ เป็นต้น
- **วัตถุประสงค์การใช้งานเฉพาะ** มีหลายภาษาที่ออกแบบมาเพื่อแก้ปัญหาเฉพาะด้าน เช่น COBOL ออกแบบมาเพื่องานด้านธุรกิจที่เกี่ยวข้องตัวเลขฐานสิบ, LISP ออกแบบมาเพื่อจัดการกับข้อมูลเชิงสัญลักษณ์และโครงสร้างข้อมูลที่ซับซ้อน, PROLOG เหมาะสำหรับการประมวลผลในการหาเหตุผลจากข้อเท็จจริงที่มีโดยใช้หลักการทางตรรกศาสตร์, C เหมาะสำหรับงานเขียนโปรแกรมในระดับต่ำ, ICON และ AWK ออกแบบมาเพื่องานประมวลผลตัวอักษรและสตริง ภาษาเหล่านี้สามารถนำไปใช้งานด้านอื่นๆ ได้เช่นกัน แต่จะเหมาะสมสำหรับงานเฉพาะด้านดังวัตถุประสงค์หลักที่ออกแบบภาษามากกว่างานด้านอื่นๆ
- **ความชอบส่วนบุคคล** ประการสุดท้าย ผู้ใช้ภาษาโปรแกรมแต่ละคนก็มีความชอบที่แตกต่างกันไป บางคนชอบความกระชับของภาษาซี แต่บางคนอาจมองว่าอ่านและเขียนยาก บางคนชอบแนวคิดแบบเรียกซ้ำตัวเอง แต่บางคนอาจจะชอบแบบทำซ้ำ บางคนชอบทำงานกับพอยต์เตอร์ แต่บางคนอาจชอบแบบ implicit dereference ความชอบและจุดแข็งของแต่ละคนจึงทำให้มีภาษาที่หลากหลาย จึงทำให้ไม่มีภาษาใดที่จะได้รับการยอมรับจากทุกคนทั่วโลกเพียงภาษาเดียว

จากภาษาโปรแกรมที่หลากหลาย มีภาษาโปรแกรมไม่มากนักที่จะได้รับความนิยมอย่างยาวนานซึ่งอาจถือได้ว่านักออกแบบภาษาเหล่านั้นประสบความสำเร็จในการออกแบบ ปัจจัยที่จะทำให้ภาษาโปรแกรมประสบความสำเร็จมีหลายประการดังต่อไปนี้

- **ง่ายต่อการเขียน** ภาษาที่มีลักษณะต่างๆ ที่เอื้อต่อการเขียนโปรแกรมตามอัลกอริทึมได้ง่ายสะดวกและกระชับ เช่น C, APL, Algol68 และ Perl เป็นต้น
- **ง่ายต่อการเรียนรู้** ภาษาที่ง่ายต่อการเขียนสำหรับผู้เริ่มต้นเขียนโปรแกรม เช่น BASIC, Pascal และ Logo
- **ง่ายต่อการพัฒนาตัวแปลภาษา** ภาษาที่ไม่ซับซ้อนมากนักทำให้ง่ายต่อการพัฒนาตัวแปลภาษาเพื่อให้โปรแกรมเมอร์ใช้งาน เช่น BASIC และ Forth
- **มีมาตรฐาน** ภาษาที่มีมาตรฐานที่ดี เช่น C และ Java
- **แจกจ่ายฟรี** ภาษาที่มีการพัฒนาตัวแปลภาษาแล้วนำไปแจกจ่ายให้ใช้ได้โดยไม่เสียค่าใช้จ่ายเช่น PASCAL, Java
- **มีตัวแปลภาษาที่ดี** ภาษาที่เมื่อโปรแกรมผ่านการคอมไพล์แล้วทำให้โค้ดมีขนาดเล็กและทำงานได้ไว เช่น FORTRAN
- **มีผู้สนับสนุน** การมีบริษัทต่างๆ เป็นผู้สร้างหรือสนับสนุนภาษาก็ทำให้ภาษานั้นมีโอกาสที่จะประสบความสำเร็จได้มาก เช่น COBOLที่ได้รับการสนับสนุนจากกระทรวงกลาโหมของสหรัฐอเมริกา, PL/I ได้รับการสนับสนุนจาก IBM หรือVisual Basic และ C# ที่ได้รับการสนับสนุนจาก Microsoft

1.3 เกณฑ์ในการพิจารณาภาษา

ภาษาโปรแกรมที่ดีจะช่วยให้เราพัฒนาซอฟต์แวร์ที่ดีได้ จึงต้องมีเกณฑ์ที่ใช้ในการประเมินภาษาว่ามีลักษณะของภาษาที่ดีหรือไม่ โดยทั่วไปจะพิจารณาจากเกณฑ์พื้นฐานหลัก3 ประการ คือ

1) **อ่านง่าย (readability)** เป็นเกณฑ์ที่สำคัญที่สุดเกณฑ์หนึ่ง โดยพิจารณาว่าโปรแกรมสามารถอ่านและเข้าใจได้ง่าย ในวัฏจักรการพัฒนาโปรแกรมหลังยุค 1970 เริ่มให้ความสำคัญและคำนึงถึงขั้นตอนการบำรุงรักษาโปรแกรมค่อนข้างมาก เนื่องจากเป็นขั้นตอนที่มีค่าใช้จ่ายค่อนข้างสูง ถ้าโปรแกรมอ่านและเข้าใจได้ง่ายก็จะทำให้บำรุงรักษาได้ง่ายเช่นกัน ลักษณะที่จะช่วยทำให้ภาษาโปรแกรมอ่านง่ายมีดังต่อไปนี้

2) **เขียนง่าย (writability)** ภาษาโปรแกรมที่ดีจะต้องง่ายต่อพัฒนาโปรแกรมเพื่อแก้ปัญหา ลักษณะต่างๆ ที่มีทำให้อ่านง่ายก็มีผลทำให้เขียนง่ายด้วยเช่นกัน ในการจะเปรียบเทียบภาษาในแง่อ่านง่ายหรือเขียนง่าย ควรเปรียบเทียบภาษาที่ออกแบบมาเพื่อการประยุกต์ใช้กับงานในลักษณะเดียวกันเท่านั้น เช่น ไม่ควรเปรียบเทียบเกณฑ์การเขียนง่ายกับปัญหาที่ต้องใช้อาร์เรย์สองมิติ หรือการสร้างรายงานของภาษา FORTRAN กับ COBOL เนื่องจากทั้งสองภาษาออกแบบมาเพื่อแก้ปัญหาคนละประเภทกัน

3) **เชื่อถือได้ (reliability)** ในที่นี้หมายถึงภาษาที่ดีจะต้องออกแบบมาเพื่อช่วยสนับสนุนในโปรแกรมเมอร์พัฒนาซอฟต์แวร์ที่ทนต่อข้อผิดพลาด และเชื่อถือได้ว่าโปรแกรมจะทำงานได้ถูกต้องทุกครั้ง

นอกจากนี้ยังมีเกณฑ์อื่นๆ ที่ใช้พิจารณาประกอบอีกด้วย ตัวอย่างเช่นค่าใช้จ่าย (cost)ซึ่งเราจะต้องคำนึงถึงตั้งแต่ค่าใช้จ่ายในการฝึกอบรมโปรแกรมเมอร์หรือจ้างคนที่มีประสบการณ์สำหรับภาษานั้น เวลาที่ต้องใช้ในการพัฒนาซอฟต์แวร์ทั้งการเขียนโปรแกรม การคอมไพล์ การรันและการบำรุงรักษาโปรแกรม

ความสามารถของภาษาในการทำงานบนเครื่องหลายประเภท (portability) ก็เป็นส่วนหนึ่งที่ต้องพิจารณาด้วยเช่นกัน ถ้าภาษามีมาตรฐานที่ดีก็ทำให้มีผลต่อความสามารถนี้ด้วยเช่นกัน และสุดท้ายการประยุกต์ใช้ (applicability) และการนิยามภาษาที่ดี (well-defineness) ก็เป็นอีกประเด็นที่นิยมนำมาพิจารณากัน

คุณลักษณะที่ช่วยสนับสนุนเกณฑ์พื้นฐานหลักทั้ง3 ประการที่กล่าวมาแล้วข้างต้น มีหลากหลายคุณลักษณะดังแสดงในตารางที่ 1.1 ประกอบด้วย

- **ความง่ายของภาษา (overall simplicity)** ประการแรก ภาษาโปรแกรมที่ดีควรมีขนาดกะทัดรัด เช่น มีโครงสร้างพื้นฐานไม่มากนัก เพราะหากภาษาโปรแกรมมีขนาดใหญ่ โปรแกรมเมอร์ส่วนใหญ่มักจะเรียนรู้เพียงแค่ส่วนหนึ่งของภาษาที่ตนเองต้องการใช้งาน และไม่รู้จักคุณสมบัติอื่นๆ ของภาษานั้น ปัญหาจึงมักจะเกิดขึ้นเมื่อผู้เขียนและผู้อ่านโปรแกรมรู้จักคุณสมบัติของภาษาไม่เหมือนกัน ผู้อ่านจึงไม่เข้าใจโปรแกรมที่ผู้เขียนพัฒนาขึ้น ประการที่สอง ภาษาโปรแกรมควรมีรูปแบบคำสั่งที่ชัดเจน เข้าใจง่าย ไม่มีหลายคำสั่งที่มีการทำงานที่เหมือนกัน (few feature multiplicity) ตัวอย่างเช่น ในภาษา C คำสั่งที่ใช้ในการบวกค่า1 ให้กับตัวแปรสามารถเขียนได้ถึง 4 รูปแบบดังต่อไปนี้

```
count = count + 1
count += 1
count++
++count
```

ประการที่สาม คือ operator overloading ซึ่งเป็นการเปลี่ยนหน้าที่ของoperator ให้ทำหน้าที่หรือมีความหมายใหม่ตามที่กำหนดตัวอย่างเช่น ผู้ใช้สร้างชนิดข้อมูลเมทริกซ์แล้วเปลี่ยน operator + และ * เพื่อใช้กับเมทริกซ์ ถึงแม้ว่าจะมีประโยชน์ แต่ก็ทำให้ลดความสามารถในการอ่านง่ายลงได้ ถ้าผู้ใช้สามารถทำการเปลี่ยนหน้าที่ operator ใดๆ แล้วใช้ในการกระทำที่ไม่สื่อความหมายกับ operator นั้น

ตารางที่ 1.1 คุณลักษณะต่างๆ ที่ช่วยสนับสนุนเกณฑ์เกณฑ์ที่ใช้ในการพิจารณาภาษาโปรแกรม

คุณลักษณะ	**เกณฑ์ในการพิจารณา**		
	อ่านง่าย	**เขียนง่าย**	**เชื่อถือได้**
ความเรียบง่าย	✓	✓	✓
ความเป็นอิสระในการผสมคำสั่ง	✓	✓	✓
โครงสร้างควบคุม	✓	✓	✓
ชนิดข้อมูลและโครงสร้าง	✓	✓	✓
การออกแบบไวยากรณ์	✓	✓	✓
การสนับสนุนลักษณะนามธรรม		✓	✓
การแสดงคำสั่งได้ง่าย		✓	✓
การตรวจสอบชนิดข้อมูล			✓
การจัดการกับข้อผิดพลาด			✓
การควบคุมการใช้นามแฝง			✓

- **ความเป็นอิสระในการผสมคำสั่ง (orthogonality)** ภาษาที่มีความเป็นอิสระสูง คือ ภาษาที่สามารถประยุกต์ใช้โครงสร้างพื้นฐานเพื่อสร้างคำสั่งที่ใช้ควบคุมและโครงสร้างข้อมูลได้โดยปราศจากข้อกำหนดหรือกฎเกณฑ์ใดๆ ตัวอย่างเช่น การประกาศ (declaration) ของภาษามีอยู่ 3 รูปแบบคือ การประกาศชนิดข้อมูล การประกาศค่าเริ่มต้น การประกาศค่าคงที่ ซึ่งการประกาศนี้สามารถทำได้กับชนิดข้อมูลต่างๆ ที่กำหนดไว้ในภาษานั้น สมมติว่ามีชนิดข้อมูลพื้นฐานอยู่ 4 ประเภท คือ integer, float, double, character นั่นแปลว่า เราจะต้องสามารถทำการประกาศทั้ง 3 รูปแบบกับชนิดข้อมูลทั้ง 4 ประเภทนี้ได้โดยไม่มีข้อยกเว้น จึงจะถือว่ามีความเป็นอิสระในการผสมคำสั่งสูง ซึ่งจะทำให้ภาษาง่ายต่อการเรียนรู้ อ่านง่าย เข้าใจง่ายและโปรแกรมเมอร์ก็เขียนโปรแกรมได้ง่ายด้วย เพราะไม่ต้องพะวงกับข้อยกเว้นต่างๆ แต่ถ้าหากภาษามีข้อยกเว้นจำนวนมาก จะทำให้ขาดความเป็นอิสระในการผสมคำสั่ง ตัวอย่างเช่น ในภาษา C ที่เราสามารถประกาศค่าเริ่มต้นให้กับข้อมูลชนิดต่างๆ ได้ยกเว้น union หรือ การส่งผ่านค่าพารามิเตอร์ระหว่างฟังชันจะเป็นการส่งผ่านแบบ passed by value ยกเว้น array ที่มีการส่งผ่านแบบ passed by reference

  ความเป็นอิสระยังหมายถึง การที่ความหมายของข้อความหรือคำสั่งไม่ขึ้นกับบริบทอีกด้วย (orthogonal มาจากภาษากรีกแปลว่าเส้นตั้งฉาก นำมาใช้กับการเรียกเวกเตอร์ในคณิตศาสตร์ที่มีลักษณะเป็นอิสระต่อกัน) ตัวอย่างของความไม่เป็นอิสระเนื่องจากความหมายขึ้นกับบริบท เช่น

```
a + b
```

  นิพจน์ในภาษา C ข้างต้นมีความหมายคือ นำค่าของ a มาบวกค่าของ b แต่ถ้าหาก a เป็น pointer จะมีผลต่อค่าของ b ทันที เช่น ถ้า a เป็น pointer ที่ชี้ไปยัง float ซึ่งใช้พื้นที่ในการเก็บ 4 ไบต์ จะต้องทำการคูณค่าของ b ด้วย 4 ก่อนจึงจะนำไปบวกกับ a

  ภาษามีความเป็นอิสระสูงเกินไป ก็ทำให้เกิดปัญหาได้เช่นกัน เพราะทำให้ภาษามีความซับซ้อน เนื่องจากมีการผสมผสานคำสั่งได้เป็นจำนวนมาก ทำให้ภาษาขาดคุณลักษณะความง่าย และหากโปรแกรมเมอร์ไม่ระมัดระวังก็อาจทำให้เกิดข้อผิดพลาดได้ง่าย หรือถ้าเขียนคำสั่งที่ผสมกันจนเกิดความซับซ้อนก็ทำให้โปรแกรมอ่านยาก ภาษาที่ถือว่ามีความเป็นอิสระสูงมาก ได้แก่ Algol68

- คำสั่งควบคุม (control statements) วิวัฒนาการของภาษาโปรแกรมแบบมีโครงสร้างในยุคปี 1970 ทำให้เกิดกลุ่มคำสั่งควบคุม ทำให้ง่ายต่อการอ่านและเขียนมากขึ้น เมื่อเทียบกับภาษาในยุคก่อนหน้าที่ใช้คำสั่ง goto เช่น FORTRAN และ BASIC ตัวอย่างเช่น เราลองพิจารณาคำสั่งทำซ้ำของโปรแกรมในภาษา C ที่เขียนโดยใช้คำสั่ง while เมื่อเทียบกับการเขียนโดยใช้คำสั่ง goto ดังต่อไปนี้

```
while (incr < 20) {
     while (sum <= 100) {
          sum += incr;
     }
     incr++;
}
```

```
loop1:
if (incr >=20) go to out;
loop2:
if (sum >100) go to next;
      sum += incr;
      go to loop2;
next:
incr++;
go to loop1;
out:
```

จะสังเกตว่าเมื่ออ่านโปรแกรมจากบนลงล่างจะเข้าใจลำดับการทำงานของโปรแกรมที่ใช้คำสั่ง while ได้ง่าย ส่วนการใช้คำสั่ง goto ต้องอ่านคำสั่งแบบกระโดดไปกระโดดมาจึงเข้าลำดับการทำงานได้ยากกว่า

- **ชนิดข้อมูลและโครงสร้าง (data types and structures)** ภาษาที่มีโครงสร้างพื้นฐานที่ทำให้สามารถนิยามชนิดข้อมูลและโครงสร้างข้อมูลได้ตามที่ต้องการก็ทำให้ภาษานั้นอ่านและเขียนได้ง่ายตัวอย่างเช่น ในภาษา C ที่ไม่มีชนิดข้อมูลแบบ boolean โดยมากจึงนิยมใช้ตัวแปร integer แทนเมื่อต้องการใช้ในการระบุ flag

```
timeout = 1
```

ซึ่งอ่านแล้วไม่ชัดเจนเมื่อเทียบกับการใช้ชนิดข้อมูลแบบ Boolean เช่น

```
timeout = true
```

- **การออกแบบไวยากรณ์ (syntax design)** รูปแบบหรือไวยากรณ์ของภาษาก็มีส่วนสำคัญต่อการอ่านและเขียนง่ายของโปรแกรม ตัวอย่างเช่นคำสั่ง foreach (คำสั่งวนลูปที่นำข้อมูลออกมาจากตัวแปร array) ในภาษายุคใหม่ เป็นการใช้คำที่สื่อความหมายได้ชัดเจนทำอ่านแล้วเข้าใจได้ง่าย

  ข้อจำกัดในการตั้งชื่อของภาษา FORTRAN ในยุคแรก อนุญาตให้ตั้งชื่อได้ไม่เกิน 6 ตัวอักษร อาจทำให้ไม่สามารถตั้งชื่อที่สื่อความหมายได้ หรือ การกำหนดคำพิเศษ (special words) สำหรับใช้ในภาษาก็มีส่วนทำให้อ่านได้ง่าย หากใช้รูปแบบและคำที่สื่อความหมาย เช่น การใช้ { และ } ของภาษา C ในการระบุกลุ่มคำสั่ง ในบางภาษาใช้ begin และ end แต่เนื่องจากโครงสร้างคำสั่งอาจมีการใช้กลุ่มคำสั่งซ้อนกัน จึงทำให้อ่านแล้วเข้าใจยากว่า end หรือ } นั้นเป็นของ begin ตัวใด บางภาษาเช่น Ada จึงใช้ end if และ end loop เพื่อมาช่วยให้อ่านง่ายขึ้น

  บางภาษาอนุญาตให้สามารถนำคำพิเศษมาใช้ในการตั้งชื่อได้ ก็จะทำให้เกิดความสับสนได้ง่ายว่าคำนั้นเป็นคำพิเศษหรือชื่อที่มีความหมายอื่น เช่น ในภาษา FORTRAN อนุญาตให้ใช้คำพิเศษ อย่าง Do หรือ End มาใช้ในการตั้งชื่อตัวแปรได้

  รูปแบบและความหมายที่ชัดเจนของคำที่ใช้เป็นอีกลักษณะที่มีผลต่อการอ่านง่าย หากคำที่ใช้นั้นมีความหมายเหมือนเดิมไม่ว่าจะปรากฏที่ใดในโปรแกรม ตัวอย่างเช่น ในภาษา C ความหมายของ static เมื่อใช้กับการประกาศตัวแปรแบบ global จะแตกต่างจากการใช้กับตัวแปรแบบ localทำให้ผู้โปรแกรมเมอร์ต้องระมัดระวังในการเขียน และผู้อ่านก็ต้องระมัดระวังในการแปลความหมายด้วยเช่นกัน

- **การสนับสนุนลักษณะนามธรรม (support for abstraction)** คำว่า นามธรรมหรือ abstraction ในทางภาษาโปรแกรมหมายถึง ความสามารถในการนิยามโครงสร้างหรือการกระทำ แล้วนำไปใช้ได้โดยไม่จำเป็นต้องสนใจรายละเอียดของมัน abstraction ถือเป็นหัวใจสำคัญของการพัฒนาซอฟต์แวร์ในยุคปัจจุบัน และมีผลต่อการเขียนง่าย ตัวอย่างเช่นในภาษาที่สนับสนุน process abstractionเราสามารถสร้างโปรแกรมย่อย sort เพื่อใช้ในการจัดเรียงข้อมูลใน array ได้ และโปรแกรมเมอร์คนอื่นก็สามารถเรียกใช้ sort ได้โดยไม่จำเป็นต้องรู้ว่าโปรแกรมย่อยนั้นใช้อัลกอริทึมแบบใดในการจัดเรียงข้อมูล หรือสำหรับ data abstraction เราสามารถสร้างชนิดข้อมูลแบบ queue ให้โปรแกรมเมอร์คนอื่นนำไปใช้ได้โดยไม่จำเป็นต้องรู้ว่าโครงสร้างของ queue ใช้ array หรือ linked list

- **การสื่อสารได้สะดวก (expressivity)** หมายถึง ลักษณะหรือรูปแบบของภาษาโปรแกรมที่ช่วยทำให้สะดวกในการเขียนโปรแกรม หรือทำให้โปรแกรมเมอร์สามารถแปลงอัลกอริทึมหรือวิธีคิดเป็นโปรแกรมได้ง่าย ตัวอย่างเช่น ในภาษา C ที่สามารถใช้คำสั่ง count++ ซึ่งมีลักษณะสั้นกะทัดรัด แทนการใช้ count = count+ 1 ก็ทำให้สะดวกต่อการเขียนหรือ รูปแบบคำสั่ง foreach ในภาษา C# ซึ่งใช้ในการวนลูปที่นำข้อมูลออกมาจากตัวแปรarray ช่วยให้เราเรียกใช้ข้อมูลได้ง่ายขึ้น

- **การสนับสนุนการตรวจสอบชนิดข้อมูล (type checking)** เป็นลักษณะของภาษาที่สามารถตรวจสอบข้อผิดพลาดที่เกิดจากการกระทำของ operator กับชนิดข้อมูลที่ไม่ถูกต้อง (type error) ได้ ทำให้เกิดความคงทนหรือความน่าเชื่อถือของโปรแกรม โดยทั่วไปนิยมตรวจสอบในช่วงที่ทำการคอมไพล์มากกว่าช่วงที่รันโปรแกรม ตัวอย่างเช่น ในภาษา Java จะมีการตรวจสอบชนิดข้อมูลของตัวแปรและนิพจน์ในการคอมไพล์ทุกครั้ง จึงสามารถกำจัดข้อผิดพลาดที่เกิดจาก type error ขณะรันโปรแกรมได้ดี ตัวอย่างเช่น

```
c = a + b
```

ก่อนทำการบวกสำหรับนิพจน์นี้ จะต้องทำการตรวจสอบชนิดข้อมูลของ a และ b ว่าเป็นชนิดข้อมูลที่สามารถนำมาบวกกันได้หรือไม่ และต้องตรวจสอบชนิดข้อมูลของ c ว่าสามารถใช้ในการบรรจุผลลัพธ์ของการบวกได้หรือไม่เช่นกัน หาก a, b และ cเป็นตัวแปร integer จึงจะดำเนินการตามคำสั่งที่ระบุ แต่หาก b เป็นตัวแปร string ก็จะแจ้งว่าเกิด type error ทำให้ไม่สามารถดำเนินการตามคำสั่งนี้ได้

- **การจัดการกับข้อผิดพลาด (exception handling)** เป็นความสามารถในการดักจับข้อผิดพลาดที่อาจเกิดขึ้นขณะรันโปรแกรม (run-time error) และทำให้โปรแกรมยังสามารถทำงานต่อได้ ไม่ยุติการทำงานไป แม้ว่าจะมีสภาวะไม่ปกติเกิดขึ้น ทำให้โปรแกรมมีความคงทนและเชื่อถือได้ สาเหตุของข้อผิดพลาดอาจเกิดจากความผิดพลาดของผู้ใช้ หรือเกิดจากสภาวะอื่นๆ เช่น ปัญหาในการอ่านข้อมูลจากดิสก์ ปัญหาที่เกิดการติดต่อสื่อผ่านเครือข่าย

- **การควบคุมการใช้นามแฝง (aliasing)** นามแฝง หมายถึง การที่มีชื่อหรือวิธีการมากกว่าหนึ่งที่ใช้ในการอ้างถึงหน่วยความจำในตำแหน่งเดียวกัน ตัวอย่างเช่น ในภาษา C มีการใช้ pointer เพื่อชี้ไปยังตัวแปรใดๆ ทำให้สามารถเข้าถึงพื้นที่ในหน่วยความจำตำแหน่งเดียวกันกับตัวแปรนั้นได้ ดังคำสั่งต่อไปนี้

```
int x;
int *ptr;
ptr = &x;
```

ในกรณีนี้ชื่อ x และ *ptr สามารถเข้าถึงหน่วยความจำตำแหน่งเดียวกันได้ จึงถือว่าชื่อ x และ *ptr เป็นนามแฝงซึ่งกันและกัน คุณลักษณะของนามแฝงนี้เป็นสิ่งที่โปรแกรมเมอร์ควรระมัดระวังเป็นอย่างยิ่งในการใช้งาน เนื่องจากการใช้นามแฝงอาจก่อให้เกิดข้อผิดพลาดในการเขียนโปรแกรมได้ง่ายหากหลงลืม เช่น เมื่อมีการเปลี่ยนให้ตัวแปร pointer ชี้ไปยังตำแหน่งที่เข้าถึงไม่ได้ หรือ การคืนพื้นที่ของตัวแปรในขณะที่ยังมีอีกตัวแปรหนึ่งชี้มายังตำแหน่งเดียวกัน ข้อผิดพลาดอาจเกิดขึ้นเหล่านี้ทำให้ความเชื่อถือได้ของภาษาลดลง

1.4 การจัดกลุ่มของภาษา

ภาษาโปรแกรมสามารถจัดแบ่งออกได้เป็นหลายกลุ่มตามแนวคิดในการเขียนโปรแกรม นิยมเรียกกลุ่มของภาษาว่า Paradigm (โดยทั่วไปหมายถึงรูปแบบของความคิดที่ลักษณะเป็นไปในแนวทางเดียวกัน) คำว่า Paradigm ในทางการโปรแกรมนั้นจึงหมายถึง รูปแบบวิธีการคิดในการแก้ปัญหาภายใต้ลักษณะเฉพาะของภาษาโปรแกรม ภาษาโปรแกรมโดยทั่วไปในปัจจุบันนิยมแบ่งออกได้เป็น 4 กลุ่มหลัก คือ

- ภาษาเชิงคำสั่ง (Imperative programming)
- ภาษาเชิงวัตถุ (Object-oriented programming- OOP)
- ภาษาเชิงฟังก์ชัน (Functional programming)
- ภาษาเชิงตรรกะ (Logic programming)

ภาษาโปรแกรมบางภาษาออกแบบมาเพื่อรองรับมากกว่าหนึ่งรูปแบบ ตัวอย่างเช่น ภาษา C++ ซึ่งเป็นทั้งภาษาเชิงคำสั่งและภาษาเชิงวัตถุ หรือภาษาเชิงทดลองอย่างเช่น ภาษา Leda ก็ออกแบบมาให้รองรับทุกรูปแบบ ในขณะที่ภาษาโปรแกรมในยุคแรกๆ เช่น PL/I หรือ Algol68 หรือ Ada มักออกแบบโดยรองรับเพียงรูปแบบเดียวเพื่อการใช้งานทั่วไป

ภาษาเชิงคำสั่ง

ภาษาเชิงคำสั่งเป็นรูปแบบที่เก่าแก่ที่สุด เนื่องจากมีรากฐานมาจากคอมพิวเตอร์ที่มีสถาปัตยกรรมแบบ Von Neumannที่ออกแบบโดยมีลักษณะที่ต้องเก็บโปรแกรมและตัวแปรไว้ในหน่วยความจำ และโปรแกรมจะประกอบด้วยชุดคำสั่งที่ทำการคำนวณ กำหนดค่าให้กับตัวแปล รับข้อมูลเข้า แสดงผล รวมทั้งการควบคุมลำดับการทำงานของชุดคำสั่งการสร้างกลุ่มคำสั่ง (block) โดยใช้ Procedural abstraction เป็นสิ่งจำเป็นสำหรับภาษาเชิงคำสั่ง ลักษณะที่สำคัญของภาษาเชิงคำสั่งประกอบด้วย

- คำสั่งกำหนดค่า (Assignment)
- คำสั่งวนซ้ำ (Loop)
- คำสั่งทำงานตามลำดับ (Sequence)
- คำสั่งเงื่อนไข (Conditional statement) และ
- คำสั่งในการจัดการความผิดปกติของโปรแกรม (Exception handling)

ตัวอย่างภาษาเชิงคำสั่งที่เด่นๆ ได้แก่ COBOL, FORTRAN, C, Ada และPerl

ภาษาเชิงวัตถุ

ภาษาเชิงวัตถุจะกำหนดรูปแบบของโปรแกรมในลักษณะของกลุ่มของวัตถุที่มีการติดต่อสื่อสารกันโดยการส่ง message เพื่อเปลี่ยนสถานะ การส่ง message นี้ทำให้ข้อมูลของวัตถุมีลักษณะเป็นแบบ active แทนที่จะเป็นแบบ passive ลักษณะเช่นนี้เป็นสิ่งที่ทำให้ภาษาเชิงวัตถุแตกต่างจากภาษาเชิงคำสั่งอย่างชัดเจน การสร้างกลุ่มคำสั่งจะอาศัยหลักการพื้นฐานดังต่อไปนี้

- การแบ่งแยกวัตถุ (Object classification)
- การสืบทอดคุณสมบัติ (Inheritance)
- การส่งผ่านข้อความ (Message passing)

ตัวอย่างภาษาเชิงวัตถุ ได้แก่ Smalltalk,C++,Javaและ C#

ภาษาเชิงฟังก์ชัน

เป็นภาษาที่จำลองปัญหาทางคอมพิวเตอร์ในรูปของฟังก์ชันทางคณิตศาสตร์ ซึ่งประกอบด้วยข้อมูลเข้า (หรือ domain) และผลลัพธ์ (หรือ range) ภาษาโปรแกรมในกลุ่มนี้แตกต่างจากภาษาที่ใช้คำสั่งกำหนดค่า

ตัวอย่างเช่น คำสั่งกำหนดค่า x=x+1 เป็นคำสั่งที่ไม่สมเหตุสมผลทางคณิตศาสตร์และในภาษาเชิงฟังก์ชันด้วยฟังก์ชันจะมีการสัมพันธ์และรวมกับฟังก์ชันอื่นโดยการใช้

- การประกอบฟังก์ชัน (functional composition)
- การใช้เงื่อนไข (conditional)
- การวนซ้ำ (recursion)

ตัวอย่างภาษาเชิงฟังก์ชันที่สำคัญ ได้แก่ LISP,Scheme, Haskell และML

ภาษาเชิงตรรกะ

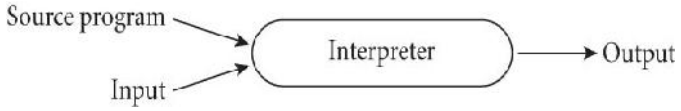
เป็นภาษาเชิงประกาศ (declarativelanguage) ที่แก้ปัญหาโดยประกาศผลของโปรแกรมที่น่าจะได้รับแทนที่จะระบุว่าทำอย่างไรจึงจะได้ผลลัพธ์ บางครั้งภาษากลุ่มนี้อาจเรียกว่าเป็นภาษา rule-based language เนื่องจากโปรแกรมมีลักษณะคล้ายกับเซตของกฎหรือข้อบังคับของปัญหาแทนที่จะเป็นลำดับการทำงานของคำสั่งการแปลความการประกาศของโปรแกรมเชิงตรรกะจะสร้างเซตของการแก้ปัญหาที่เป็นไปได้ทั้งหมดตามที่โปรแกรมระบุไว้ ภาษาเชิงตรรกะเหมาะสำหรับการแก้ปัญหาที่มีรายละเอียดไม่สมบูรณ์
ตัวอย่างภาษาเชิงตรรกะที่สำคัญ ได้แก่ Prolog

1.5 การแปลภาษา

ภาษาระดับสูงเป็นภาษาที่มนุษย์อ่านและเขียนได้ง่ายกว่าภาษาเครื่อง แต่เครื่องคอมพิวเตอร์ไม่สามารถเข้าใจคำสั่งได้ทันที จึงจำเป็นต้องมีโปรแกรมที่ทำหน้าที่ในการแปลจากภาษาระดับสูงเป็นภาษาเครื่อง วิธีการแปลภาษามีอยู่ 3 รูปแบบ คือ อินเตอร์พรีตเตอร์ (interpreter), คอมไพเลอร์ (compiler) และไฮบริด (hybrid)

อินเตอร์พรีตเตอร์

การทำงานของอินเตอร์พรีตเตอร์จะทำการแปลคำสั่งในภาษาระดับสูงให้เป็นภาษาเครื่องและทำงานตามคำสั่งไปพร้อมๆ กัน ดังแสดงในรูปที่ 1.3 ขั้นตอนการทำงานจะคล้ายคลึงกับวัฏจักรเครื่อง (machine cycle) นั่นคือจะทำการอ่านคำสั่งจากโปรแกรมต้นฉบับ (source program) มาครั้งละ 1 คำสั่งหรือมากกว่า แล้วแปลความหมาย จากนั้นจะประมวลผลตามคำสั่ง วนรอบแบบนี้ไปเรื่อยๆ จนกว่าจะครบทุกคำสั่ง

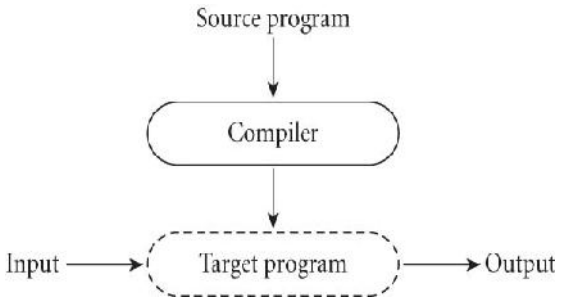


รูปที่ 1.3 การทำงานของอินเทอร์พรีตเตอร์

ข้อดีของอินเตอร์พรีตเตอร์ คือ มีความยืดหยุ่นสูง และสามารถค้นหาข้อผิดพลาดได้ง่าย เนื่องจากการประมวลผลตามโปรแกรมต้นฉบับทีละคำสั่ง เมื่อเกิดข้อผิดพลาดที่คำสั่งใด อินเตอร์พรีตเตอร์ก็จะแจ้งเตือนในทันทีว่าผิดพลาดเพราะอะไร และหยุดการแปล เมื่อโปรแกรมเมอร์แก้ไขข้อผิดพลาดนั้นให้ถูกต้อง จะต้องสั่งให้อินเตอร์พรีตเตอร์ทำการแปลใหม่ตั้งแต่ต้น ข้อเสียคือทำงานช้ากว่าคอมไพเลอร์ ตัวอย่างของภาษาที่ใช้อินเตอร์พรีตเตอร์ คือ BASIC, Prolog, Smalltalk และภาษาในกลุ่ม Scripting language เช่น PHP

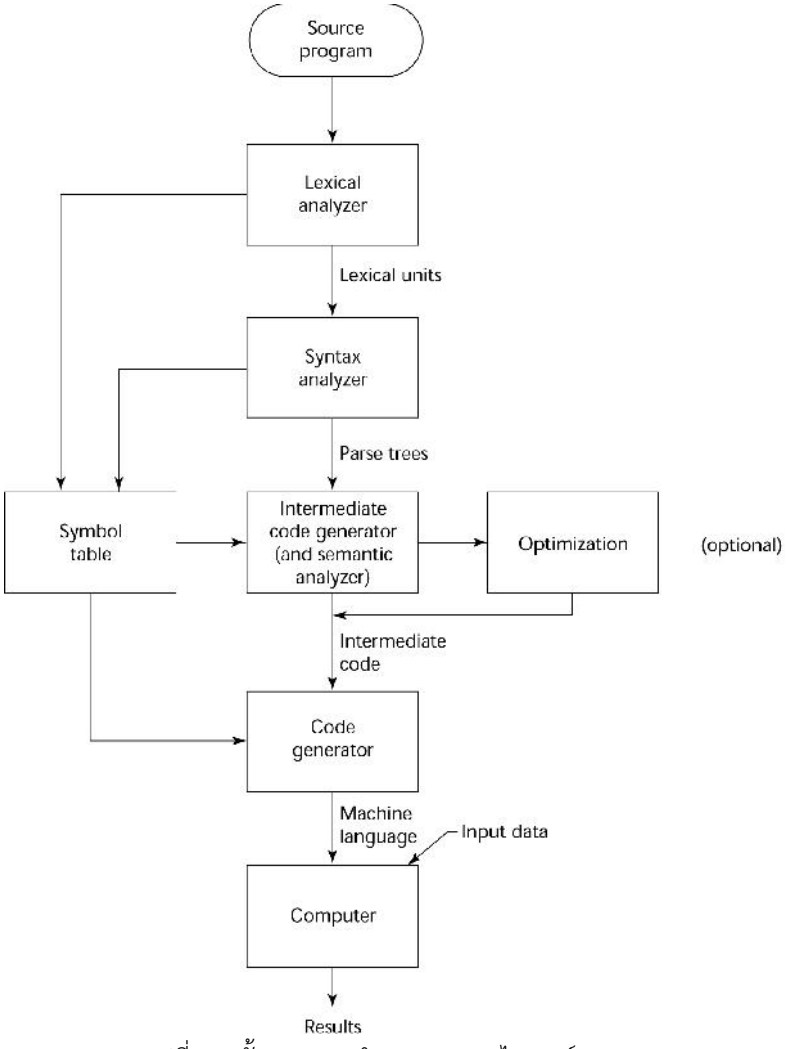
คอมไพเลอร์

คอมไพเลอร์ทำงานโดยการแปลคำสั่งภาษาระดับสูงในโปรแกรมต้นทางทั้งหมดไปเป็นภาษาเครื่องในครั้งเดียว เมื่อแปลเป็นโปรแกรมภาษาเครื่องแล้ว สามารถนำโปรแกรมนั้นไปใช้ประมวลผลได้เลย โดยไม่จำเป็นต้องใช้คอมไพเลอร์อีก เพราะคอมไพเลอร์ทำการแยกการแปลออกจากสั่งการประมวลผล ดังแสดงในรูปที่ 1.4 ภาษาโปรแกรมส่วนใหญ่จะใช้ตัวแปรภาษาแบบคอมไพเลอร์ เช่น C, COBOL และ Pascal



รูปที่ 1.4 การทำงานของคอมไพเลอร์

ข้อดีของคอมไพเลอร์ คือ โปรแกรมที่ผ่านการคอมไพล์แล้วจะทำงานได้เร็ว แต่ขั้นตอนในการทำงานของคอมไพเลอร์ค่อนข้างซับซ้อน ดังแสดงในรูป 1.5 จึงทำให้การพัฒนาคอมไพเลอร์ยากและใช้เวลานาน

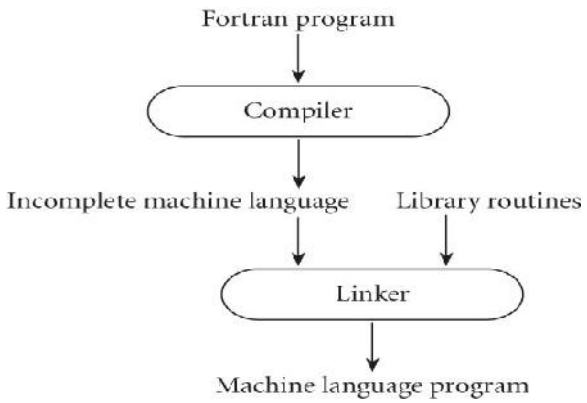


รูปที่ 1.5 ขั้นตอนการทำงานของคอมไพเลอร์

ขั้นตอนการทำงานของคอมไพเลอร์ เริ่มต้นจากกระบวนการตรวจสอบความถูกต้องของโปรแกรมต้นฉบับ โดยเริ่มจากขั้นตอนการวิเคราะห์หน่วยคำ (lexical analyzer) เป็นการนำโปรแกรมต้นฉบับมาวิเคราะห์ในระดับหน่วยคำที่เรียกว่า lexeme หน่วยคำเหล่านี้มีการจัดกลุ่มตามหน้าที่ของคำที่เรียกว่า token ประกอบด้วย identifier, special word, operator และ punctuation และจัดทำการจัดเก็บคำและหน้าที่ของคำไว้ในตารางสัญลักษณ์ (symbol table) เพื่อใช้ในขั้นตอนต่อไป ในขั้นตอนนี้จะตัดส่วน comment ออกไปด้วย ขั้นตอนถัดไปเป็นการวิเคราะห์โครงสร้างไวยากรณ์ (syntax analyzer) โดยทำการวิเคราะห์ว่าคำที่ได้ในขั้นตอนก่อนหน้านั้นประกอบกันขึ้นเป็นโครงสร้างของโปรแกรมที่ถูกต้องตามหลักไวยากรณ์ที่ภาษากำหนดไว้หรือไม่ ในการตรวจสอบไวยากรณ์นิยมสร้างในรูปของต้นไม้ที่เรียกว่า parse tree

ต่อมาเป็นขั้นตอนของการวิเคราะห์ความหมาย (semantic analyzer) โดยจะพิจารณาในแง่ความหมายของคำสั่งว่าสามารถทำได้หรือไม่ ผิดข้อกำหนดของภาษาหรือไม่ ตัวอย่างเช่น ตรวจสอบ type error, ในภาษาที่ต้องมีการประกาศตัวแปร จะไม่สามารถเรียกใช้ตัวแปรในคำสั่งได้หากยังไม่ได้ประกาศไว้ก่อน, ตรวจสอบชนิดข้อมูลของ formal parameter และ actual parameter เมื่อตรวจสอบโครงสร้างของโปรแกรมว่าถูกต้องแล้ว ก็จะเริ่มกระบวนการแปลงเป็นภาษาเครื่อง โดยจะทำการแปลเป็นภาษากลาง (intermediate code) ซึ่งเป็นภาษาที่อยู่ในระดับกลาง ระหว่างภาษาระดับสูงกับภาษาเครื่อง โดยทั่วไปจะมีลักษณะคล้ายกับภาษาแอสแซมบลี หรือในระดับที่สูงกว่า หรือบางครั้งก็จะแปลงเป็นภาษาแอสแซมบลีเลย การแปลเป็นภาษากลางก่อนมีข้อดีคือ ทำให้สามารถปรับปรุงโปรแกรมให้มีขนาดเล็ก และทำงานเร็วได้ ในขั้นตอนของการทำ optimization (ถ้าสามารถทำได้) หลังจากนั้นจึงเป็นขั้นตอนของการแปลเป็นภาษาเครื่อง ซึ่งในขั้นตอนวิเคราะห์ความหมายและแปลเป็นภาษาเครื่องนี้ก็จะนำข้อมูลจากตารางสัญลักษณ์มาใช้

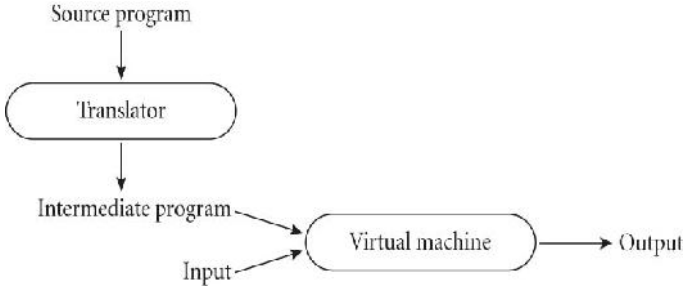
ในการเขียนโปรแกรมนั้น โดยส่วนใหญ่โปรแกรมเมอร์ไม่จำเป็นต้องขึ้นเพื่อใช้งานเองทั้งหมด ภาษาโปรแกรมมักจะมีฟังก์ชั่นมาตรฐานให้เรียกใช้ เช่น คำสั่งในการรับและแสดงผลข้อมูล (เช่น printf, scanf ในภาษา C), ฟังก์ชันทางคณิตศาสตร์ (เช่น sin, cos, log) ซึ่งฟังก์ชันเหล่านี้จะเก็บอยู่ใน library ดังนั้น ภาษาเครื่องที่ได้ในขั้นตอนก่อนหน้าจึงยังไม่สามารถใช้งานได้ ต้องมีภาษาเครื่องนั้นไปทำการเชื่อมโยงเข้ากับ libray ก่อน จึงจะได้เป็นภาษาเครื่องที่สมบูรณ์ในรูปของ executable program (เช่น ไฟล์ที่มีนามสกุล .exe) สามารถนำไปใช้งานได้ ขั้นตอนในการเชื่อมโยงกับ library นี้จะทำโดยโปรแกรมที่เรียกว่า linker ดังตัวอย่างของการแปลภาษา FORTRAN ในรูปที่ 1.6



รูปที่ 1.6 การแปลภาษา FORTRAN

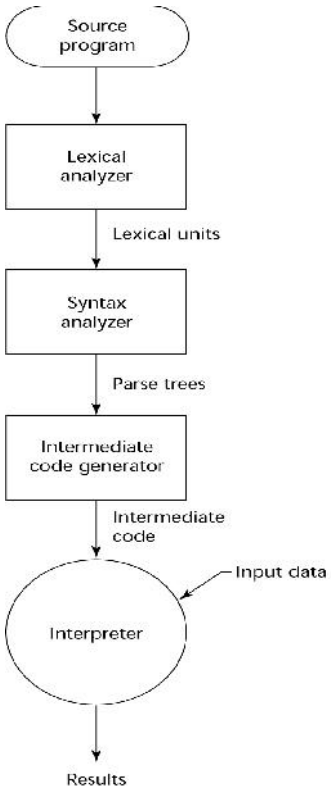
ไฮบริด

เนื่องจากการแปลภาษาแบบอินเตอร์พรีตเตอร์และคอมไพเลอร์ต่างก็มีทั้งข้อดีและข้อด้อย จึงทำให้เกิดการแปลภาษารูปแบบใหม่ที่ดึงเอาข้อดีของทั้งสองรูปแบบมาใช้ผสมกันเรียกว่าแบบไฮบริดดังแสดงในรูป 1.7



รูปที่ 1.7 การทำงานของตัวแปลภาษาแบบไฮบริด

โดยจะทำการแปลเป็นภาษาระดับกลาง (intermediate program) โดยใช้วิธีการแบบคอมไพเลอร์ก่อน แล้วจึงค่อยใช้วิธีการอินเตอร์พรีตเตอร์เพื่อแปลงเป็นภาษาเครื่อง วิธีการนี้จะทำงานเร็วกว่าแบบอินเตอร์พรีตเตอร์ เนื่องจากมีการแปลภาษามาแล้วในระดับหนึ่ง ขั้นตอนการแปลภาษาแบบไฮบริดแสดงดังรูปที่ 1.8



รูปที่ 1.8 ขั้นตอนการทำงานของตัวแปลภาษาแบบไฮบริด

ตัวอย่างของภาษาที่มีการแปลแบบไฮบริด เช่น Perl ที่ทำการแปลโปรแกรมต้นฉบับโดยคอมไพเลอร์เพื่อตรวจหาข้อผิดพลาดก่อน แล้วจึงค่อยใช้อินเตอร์พรีตเตอร์แปลเป็นภาษาเครื่อง และ Java ที่ทำการแปลโปรแกรมต้นฉบับ (ไฟล์ .java) ให้เป็นโปรแกรมในภาษาระดับกลางเรียกว่า byte code (ไฟล์ .class) เพื่อทำให้เกิดคุณสมบัติ portability สำหรับเครื่องใดๆ ที่มี Java Virtual Machine หรือ JVM ซึ่งเป็นอินเตอร์พรีตเตอร์ทำหน้าที่แปลง byte code ให้เป็นภาษาเครื่องที่ขึ้นอยู่กับแพลตฟอร์มนั้น แล้วทำการรันโปรแกรม

1.6 สภาพแวดล้อมของการพัฒนาโปรแกรม

ในปัจจุบันโปรแกรมเมอร์พัฒนาโปรแกรมได้สะดวกและรวดเร็วขึ้นกว่าในสมัยก่อน เนื่องจากมีโปรแกรมที่ช่วยอำนวยความสะดวกในเขียนโปรแกรมจำนวนมาก เช่น editor ใช้ในการเขียนโปรแกรม, compiler หรือ interpreter และ linker ใช้ในการแปลโปรแกรมต้นฉบับเป็นภาษาเครื่อง, debugger ที่ใช้ช่วยตรวจสอบหาข้อผิดพลาดในโปรแกรม เรานิยมเรียกโปรแกรมเหล่านี้ว่าเครื่องมือ (tools) หากนำเครื่องมือเหล่านี้มารวมกันเป็นโปรแกรมเดียวแล้วสร้าง GUI และ file system ที่ช่วยในอำนวยความสะดวกสำหรับการเขียนโปรแกรม เราเรียกเครื่องมือนี้ว่า Integrated development environment หรือ IDE

โดยทั่วไปโปรแกรม IDE มักจะมีความสามารถเฉพาะบางภาษา เช่น Eclipse หรือ NetBeans ใช้สำหรับภาษา Java, CodeBlocks หรือ Dev-Cpp ใช้สำหรับ C หรือ C++, Microsoft Visual Studio.NET สำหรับ 5 ภาษาในกลุ่ม .NET ได้แก่ C#, Visual BASIC .NET, Jscript, J#, C++ เป็นต้น กล่าวได้ว่า IDE เป็นโปรแกรมที่สร้างสภาพแวดล้อมในการพัฒนาโปรแกรม (programming environment) ให้เหมาะแก่การเขียนโปรแกรม

เอกสารอ้างอิงและภาพประกอบ

Michael L. Scott. *Programming Language Pragmatics.* Morgan Kaufmann Publishers, 2009. (Chapter 1)

Robert W. Sebesta. *Concepts of Programming Languages*, Addison Wesley, 2006. (Chapter 1)

ทัศนวรรณ ศูนย์กลาง. *เอกสารประกอบการสอนรายวิชา องค์ประกอบคอมพิวเตอร์และภาษาโปรแกรม.* มหาวิทยาลัยศิลปากร, 2552. (บทที่ 8)

นิตยา เกิดประสพ. *เอกสารคำสอนรายวิชาหลักการของภาษาการทำโปรแกรม.* มหาวิทยาลัยเทคโนโลยีสุรนารี, 2547. (บทที่ 1 และบทที่ 4)