บทที่ 1

==========

แนวคิดของภาษาการทำโปรแกรม
(Programming language concept)

วัตถุประสงค์

1) เพื่อแนะนำผู้เรียนให้เข้าใจแนวคิดของภาษาการทำโปรแกรม
2) เพื่อสร้างความเข้าใจเกี่ยวกับการวิเคราะห์ข้อเด่น-ข้อด้อยของแต่ละภาษา
3) เพื่อให้ผู้เรียนรู้จักการจัดกลุ่มภาษาตามลักษณะการทำงาน
4) เพื่ออธิบายวิวัฒนาการของภาษาคอมพิวเตอร์

ภาษาคอมพิวเตอร์เป็นเครื่องมือที่โปรแกรมเมอร์ใช้ในการทำโปรแกรม การศึกษาเกี่ยวกับแนวคิดและหลักการของภาษาแบบต่างๆ จึงเปรียบเสมือนการทำความเข้าใจรูปแบบและวิธีการใช้งาน เครื่องมือซึ่งจะช่วยให้การทำโปรแกรมมีประสิทธิภาพมากขึ้น

1.1 ความสัมพันธ์ของภาษาและวัฏจักรการพัฒนาซอฟต์แวร์

(Relationship of language and software development cycle)

การศึกษาเกี่ยวกับแนวคิดหรือหลักการของภาษาการทำโปรแกรม มิใช่เพียงการศึกษารูปแบบคำสั่งและรายละเอียดของภาษาใดภาษาหนึ่งเป็นการเฉพาะ แต่เป็นการศึกษาแง่มุมต่างๆ ของภาษาที่จะมีผลกระทบต่อวัฏจักรการพัฒนาซอฟต์แวร์

วัฏจักรการพัฒนาซอฟต์แวร์ ประกอบด้วยขั้นตอนต่างๆ คือ

1) การวิเคราะห์ความต้องการและการกำหนดคุณลักษณะ (requirement analysis and specification)

เป็นขั้นตอนของการสัมภาษณ์ความต้องการของผู้ใช้ ว่าต้องการซอฟต์แวร์ที่มีลักษณะอย่างไร มีความสามารถใดบ้าง ขั้นตอนนี้จะนำไปสู่การกำหนดคุณลักษณะซอฟต์แวร์ว่าจะต้องมีลักษณะหรือหน้าที่การทำงานอะไรบ้าง แต่ยังไม่มีการระบุว่าจะสร้างลักษณะต่างๆเหล่านั้นได้อย่างไร ผลผลิตที่ได้จากขั้นตอนนี้คือ เอกสารระบุความต้องการ (requirement document)

2) การออกแบบซอฟต์แวร์ (software design)

ขั้นตอนนี้จะต่อเนื่องจากขั้นตอนแรก ลักษณะต่างๆของซอฟต์แวร์ที่ผู้ใช้ต้องการจะถูกนำมาออกแบบในรายละเอียดว่าซอฟต์แวร์นี้จะต้องมีโมดูลอะไรบ้าง รวมถึงการออกแบบส่วนเชื่อมต่อต่างๆ ผลผลิตของขั้นตอนนี้คือ เอกสารการออกแบบคุณลักษณะซอฟต์แวร์ (design specification document)

3) การเขียนโปรแกรม หรือ การลงรหัส (implemention or coding)

ผลการออกแบบซอฟต์แวร์ในขั้นตอนก่อนหน้า จะถูกนำมาลงรหัสหรือสร้างเป็นโปรแกรม ผลผลิตที่ได้คือ ซอฟต์แวร์และเอกสารประกอบโปรแกรม

4) การทวนสอบและการตรวจสอบความถูกต้อง (verification and validation)

ขั้นตอนการตรวจสอบนี้ เป็นการตรวจสอบความถูกต้องของซอฟต์แวร์ว่าทำงานได้ถูกต้อง และทำงานได้ตรงกับความต้องการของผู้ใช้ ในระหว่างการพัฒนาซอฟต์แวร์จะมีการตรวจความถูกต้องทั้งในระดับโมดูล (module testing) และระดับการเชื่อมต่อระหว่างโมดูล (integration testing)

5) การบำรุงรักษา (maintenance)

เมื่อส่งมอบซอฟต์แวร์ให้ลูกค้าหรือผู้ใช้แล้ว งานที่มักจะเกิดตามมาคือการปรับปรุงหรือการแก้ไขซอฟต์แวร์ ที่อาจจะเกิดจากความต้องการที่เปลี่ยนไปของผู้ใช้ หรือเกิดจากข้อผิดพลาดของซอฟต์แวร์

ภาษาการทำโปรแกรมหรือภาษาคอมพิวเตอร์ จะเข้ามามีบทบาทในบางขั้นตอนของวัฏจักรการพัฒนาซอฟต์แวร์ ได้แก่ ขั้นตอนการลงรหัส และขั้นตอนการออกแบบ แนวทางการออกแบบซอฟต์แวร์มีได้หลายลักษณะ เช่น ออกแบบในลักษณะโครงสร้าง (structured design), ออกแบบในลักษณะบนลงล่าง (top-down design), ออกแบบในเชิงวัตถุ (object-oriented design)

การเลือกลงรหัสในภาษาที่สอดคล้องกับแนวทางการออกแบบจะช่วยให้งานพัฒนาซอฟต์แวร์ทำได้ง่ายขึ้น เช่น เลือกใช้ภาษา Pascal กับการออกแบบในลักษณะโครงสร้าง, เลือกใช้ภาษา C++ หรือภาษา Java กับการออกแบบในเชิงวัตถุ

การศึกษาเกี่ยวกับแนวคิดและหลักการของภาษาการทำโปรแกรมจึงมีความจำเป็น เนื่องจากไม่มีภาษาคอมพิวเตอร์ใดที่เหมาะสมใช้งานได้ดีกับงานทุกประเภท โปรแกรมเมอร์ที่ดีจึงควรจะเรียนรู้มากกว่าหนึ่งภาษาและรู้ข้อเด่น-ข้อด้อยของภาษานั้นๆเพื่อการเลือกใช้อย่างถูกต้อง นอกจากนี้การรู้ถึงแนวคิดและหลักการของภาษาจะช่วยเพิ่มความเข้าใจเกี่ยวกับภาษานั้นๆได้ลึกซึ้งมากขึ้นซึ่งจะส่งผลโดยตรงต่อแนวทางการเขียนโปรแกรมที่จะมีประสิทธิภาพมากขึ้น และช่วยให้การศึกษาภาษาใหม่ๆในอนาคตทำได้ง่ายขึ้น เพราะการศึกษาอย่างรู้หลักการย่อมจะทำได้เร็วกว่าการศึกษาแบบท่องจำ ในบางโอกาสที่นักศึกษาจำเป็นต้องออกแบบภาษาใหม่เพื่อใช้ในงานเฉพาะกิจบางอย่าง การรู้หลักการของภาษาจะทำให้การออกแบบภาษาใหม่ทำได้อย่างถูกต้อง ซึ่งจะช่วยให้เกิดพัฒนาการใหม่ๆได้ในวงการคอมพิวเตอร์

1.2 การพิจารณาคุณลักษณะของภาษา

(Language property criteria)

ภาษาคอมพิวเตอร์ถูกสร้างขึ้นมาเพื่อใช้เป็นเครื่องมือในการพัฒนาซอฟต์แวร์ ดังนั้นภาษาที่ดีจึงเป็นภาษาที่เอื้อให้เราสามารถพัฒนาซอฟต์แวร์ที่ดีได้ ซอฟต์แวร์ที่ดีจะต้องมีคุณสมบัติพื้นฐาน 3 ประการ คือ

- **ความเชื่อถือได้ (reliability)**
  ซอฟต์แวร์ที่ดีจะต้องทนทานต่อข้อบกพร่องและเหตุผิดปกติ ทั้งในด้านฮาร์ดแวร์และซอฟต์แวร์
- **ความสามารถบำรุงรักษาได้ (maintainability)**
  ซอฟต์แวร์ที่ดีจะต้องสามารถปรับปรุงแก้ไขได้ง่าย
- **ความมีประสิทธิภาพ (efficiency)**
  ความมีประสิทธิภาพของซอฟต์แวร์พิจารณาจากความเร็วในการทำงาน(speed), ความสามารถในการนำบางส่วนกลับมาใช้ได้ใหม่ (reusable) และความสามารถในการดำเนินงานได้บนเครื่องหลากหลายประเภท (portable)

ดังนั้นลักษณะของภาษาที่ดีจึงสามารถพิจารณาได้จากเกณฑ์ต่อไปนี้

1) ความชัดเจนและความง่าย (clarity and simplicity)

ภาษาที่ดีควรจะมีรูปแบบคำสั่งที่ชัดเจน เข้าใจได้ง่าย และไม่ใช้รูปแบบย่อจนเกินไป ตัวอย่างต่อไปนี้แสดงโปรแกรมในรูปแบบของภาษา APL ที่รับค่าจำนวนเต็ม (N) แล้วพิจารณาว่าตัวเลขนั้นเป็นจำนวนเฉพาะหรือไม่

คำสั่งในภาษา APL	คำอธิบายค่า N ที่เปลี่ยนแปลงไป (สมมุติ N มีค่าเริ่มต้น = 11)
`N-2`	9
`iN-2`	1 2 3 4 5 6 7 8 9
`1+iN-2`	2 3 4 5 6 7 8 9 10
`(1+iN-2) | N`	1 2 3 1 5 4 3 2 1
`0 ≠ (1+iN-2) | N`	1 1 1 1 1 1 1 1 1
`^ / 0 ≠ (1+iN-2) | N`	1 So, N= 11 is prime

หรือตัวอย่างของข้อความในภาษา C ดังต่อไปนี้

```
int i;
i = (1&&2)+3;
```

จากทั้งสองตัวอย่างจะเห็นได้ว่า ภาษา APL และภาษา C มีการใช้รูปแบบคำสั่งที่สั้น กระทัดรัด แต่พิจารณาความหมายของคำสั่งได้ยาก ทำให้ไม่เหมาะสมสำหรับผู้เริ่มต้นศึกษาการเขียนโปรแกรม

นอกจากนี้ความง่ายยังหมายถึงการเขียนข้อความสั่งเพื่อการทำงานบางอย่าง ควรจะมีรูปแบบเดียว เช่น ในภาษา C การสั่งบวก 1 ให้ค่าของตัวแปร x เขียนได้ถึงสี่รูปแบบดังต่อไปนี้

```
/* รูปแบบที่ 1 */      x = x+1;
/* รูปแบบที่ 2 */      x+ = 1;
/* รูปแบบที่ 3 */      ++x;
/* รูปแบบที่ 4 */      x++;
```

จึงถือได้ว่าภาษา C เป็นภาษาที่ขาดคุณสมบัติความชัดเจนและความง่าย

2) ความเป็นอิสระในการผสมคำสั่ง (orthogonality)

ภาษาที่มีความเป็นอิสระสูง จะอนุญาตให้ผู้เขียนโปรแกรมใช้ลักษณะต่างๆ ที่มีในภาษานั้นผสมให้เป็นข้อความสั่งได้โดยไม่มีข้อจำกัด ตราบเท่าที่ข้อความสั่งยังเป็นไปตามกรอบไวยากรณ์ของภาษา ตัวอย่างเช่น ภาษา C กำหนดไวยากรณ์ของคำสั่ง if ไว้ดังต่อไปนี้

if <expression> <statement>;

ดังนั้นโปรแกรมเมอร์สามารถเขียนคำสั่งต่อไปนี้ได้

if (x = 0) printerr( );

หรือคำสั่งต่อไปนี้

if (x = = 0) printerr( );

ทั้งสองคำสั่งถูกต้องตามไวยากรณ์ แต่มีความหมายแตกต่างกัน

ภาษา Algol เป็นภาษาที่ให้อิสระแก่โปรแกรมเมอร์สูงมากในการผสมลักษณะต่างๆ เข้าเป็นชุดคำสั่ง ตัวอย่างต่อไปนี้แสดงการเขียนคำสั่งวนรอบ 10 รอบ ในสามรูปแบบที่เป็นไปได้

/* *รูปแบบที่ 1* */	`for count := 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 do` `list[count] :=0;`
/* *รูปแบบที่ 2* */	`for count := 1 step 1 until 10 do` `list[count] :=0;`
/* *รูปแบบที่ 3* */	`for count := 1, count +1 while (count <= 10) do` `list[count] :=0;`

ภาษาที่ให้อิสระสูง จะอำนวยความสะดวกให้กับโปรแกรมเมอร์ในการเขียนโปรแกรมได้หลายลักษณะ โดยไม่มีข้อจำกัดหรือไม่ต้องพะวงกับกรณียกเว้นต่างๆ (ตัวอย่างการมีกรณียกเว้นเช่น ในภาษา C ตัวแปร int จะส่งผ่านค่าระหว่างฟังก์ชันโดยการสำเนาค่า แต่ตัวแปรแถวลำดับ (array) จะส่งผ่านค่าโดยการอ้างอิง) แต่ข้อเสียของการให้อิสระสูงเกินไปคือ ถ้าโปรแกรมเมอร์เขียนคำสั่งอย่างไม่ระมัดระวังจะมีโอกาสเกิดข้อผิดพลาดในเชิงตรรกะ (logical error) ได้ง่าย และถ้าเขียนคำสั่งซับซ้อนมากเกินไปจะทำให้โปรแกรมอ่านยาก ซึ่งจะเป็นผลเสียในขั้นตอนการบำรุงรักษาโปรแกรม

3) ความเชื่อถือได้ (reliability)

ภาษาที่ดีจะต้องมีคุณลักษณะต่างๆ ที่ช่วยสนับสนุนให้โปรแกรมเมอร์สามารถพัฒนาซอฟต์แวร์ที่ทนต่อข้อบกพร่อง และสามารถเชื่อถือได้ว่าจะให้ผลลัพธ์ที่ถูกต้องทุกคราวที่รันโปรแกรม คุณลักษณะที่ช่วยสนับสนุนความเชื่อถือได้นี้ประกอบด้วย

- *มีการควบคุมกรณีนามแฝง (alias) และการอ้างอิงหน่วยความจำผิดพลาด (memory leak)*
  กรณีนามแฝงจะเกิดขึ้นเมื่อชื่อมากกว่าหนึ่งชื่ออ้างอิงตำแหน่งเดียวกันในหน่วยความจำ ตัวอย่างเช่น ในภาษา C

  int x;
  *int *y;*
  y = &x;

  ชื่อ x และ *y อ้างถึงตำแหน่งเดียวกันในหน่วยความจำทำให้อาจเกิดกรณีการอ้างอิงหน่วยความจำผิดพลาด ความผิดพลาดจะเกิดขึ้นได้เมื่อตัวแปรพ็อยเตอร์ถูกเปลี่ยนค่าให้ชี้ไปยังตำแหน่งต่างๆ ใน

หน่วยความจำที่อาจจะเป็นส่วนบรรจุชุดคำสั่ง (code area) แทนที่จะเป็นส่วนบรรจุข้อมูล (data area) ดังตัวอย่างคำสั่งในภาษา C ต่อไปนี้

```
int x;
int *y = & x;
y += 4000 ;
```

ภาษา Java มีการป้องกันเหตุผิดพลาดเหล่านี้ด้วยการไม่อนุญาตให้ใช้ตัวแปรพ็อยเตอร์ ภาษา Java จึงถือว่ามีความเชื่อถือได้มากกว่าภาษา C

- *สนับสนุนการตรวจสอบชนิดข้อมูล (type checking)*

  การตรวจสอบชนิดข้อมูล หมายถึงการตรวจประเภทของข้อมูลว่า สามารถถูกปฏิบัติการได้ด้วยคำสั่งดำเนินการที่ระบุ ตัวอย่างเช่น

  $$X = A + B * C$$

  ก่อนที่จะดำเนินการคูณและบวก เพื่อบรรจุผลลัพธ์ในตัวแปร X จะต้องมีการตรวจสอบชนิดข้อมูล A, B, C ก่อนว่า สามารถนำมาคูณและบวกได้ นั่นคือจะต้องเป็นข้อมูลประเภทจำนวนเลข ภาษาที่ดีมักจะกำหนดให้มีการตรวจสอบชนิดข้อมูลก่อนที่จะดำเนินการตามคำสั่งที่ระบุในโปรแกรม

- *มีรูปแบบคำสั่งและความหมายที่ชัดเจน*

  ภาษาบางภาษา (เช่น Java) อนุญาตให้เครื่องหมายดำเนินการ สามารถทำงานได้หลายหน้าที่ (operator overloading) เช่น เครื่องหมาย + หมายถึงได้ทั้งการบวกทางคณิตศาสตร์และการเชื่อมต่อข้อความ (string concatenation) ทำให้ความหมายของเครื่องหมาย + ไม่ชัดเจนโดยตัวเอง ต้องพิจารณาความหมายจากรูปแบบประโยคคำสั่งว่าเป็นคำสั่งทำงานกับนิพจน์คณิตศาสตร์ หรือคำสั่งทำงานกับข้อความ ลักษณะเช่นนี้ทำให้การอ่านโปรแกรมทำได้ยากขึ้น (แต่การเขียนโปรแกรมทำได้สะดวก เพราะไม่ต้องสร้างสัญลักษณ์ใหม่เพื่อแทนการเชื่อมต่อข้อความ)

4) การประยุกต์ใช้ได้ (applicabiliy)

ภาษาที่ถูกสร้างขึ้นมาเพื่อใช้กับงานเฉพาะด้าน เช่น งานปัญญาประดิษฐ์, งานจำลอง จะต้องมีโครงสร้างที่ครบถ้วนและเหมาะสมกับงานนั้นๆ

5) ความเป็นนามธรรม (abstraction)

ความเป็นนามธรรมหมายถึง การพิจารณาแนวคิดรวบยอดโดยไม่ต้องแจกแจงในรายละเอียด ตัวอย่างต่อไปนี้แสดงความเป็นนามธรรมในงานคณิตศาสตร์ที่ใช้แพทเทิร์นแทนการแจกแจงค่า

Abstraction	
Sequence	Pattern
1, 2, 3, 4, 5, 6, …	n
2, 4, 6, 8, 10, 12, …	2n
1, 4, 9, 16, 25, 36, …	n^2
1, 3, 7, 15, 31, 63, …	2^n-1

ตัวอย่างความเป็นนามธรรมในงานพัฒนาซอฟต์แวร์ ได้แก่ การสร้างฟังก์ชันเก็บไว้ในไลบารีของภาษา C หรือการสร้างคลาสเก็บไว้ในไลบารีของภาษา Java เพื่อให้สามารถเรียกใช้ (ด้วยการ include หรือ import) โดยการระบุชื่อฟังก์ชันหรือชื่อคลาสที่ต้องการโดยไม่ต้องพะวงในรายละเอียดของฟังก์ชัน หรือคลาสนั้นๆ ภาษาที่ดีจึงควรมีโครงสร้างที่อำนวยความสะดวกให้กับผู้ใช้ในการสร้างหรือเรียกใช้ลักษณะนามธรรมเหล่านี้

6) ความมีประสิทธิภาพ (efficiency)

ความมีประสิทธิภาพของภาษา หมายถึง การมีตัวแปลภาษาที่ทำงานได้รวดเร็ว การมีสภาพแวดล้อมที่ช่วยให้การพัฒนาโปรแกรมทำได้สะดวก เช่นมีส่วนเอดิเตอร์ (editor) เพื่อการพิมพ์คำสั่ง มีดีบักเกอร์ (debugger) ที่ช่วยในการตรวจสอบหาข้อผิดพลาดในโปรแกรม เป็นต้น

1.3 แนวทางการจัดกลุ่มภาษา

(Programming language paradigms)

ภาษาคอมพิวเตอร์ในปัจจุบันมีมากกว่า 3,000 ภาษา และมีผู้สร้างภาษาใหม่ๆ เพิ่มขึ้นทุกวัน ในจำนวนภาษาที่มีมากมายเหล่านี้ สามารถนำมาจัดกลุ่มได้เป็น 4 แนวทางหลัก ตามแนวทางการทำโปรแกรม คือ

1) กลุ่มภาษาเชิงคำสั่ง (imperative languages)

ภาษาในกลุ่มนี้ถูกสร้างขึ้นเพื่อใช้ในการทำโปรแกรมที่มีแนวคิดว่า โปรแกรมประกอบขึ้นจากชุดคำสั่งที่มีลำดับการทำงานที่แน่นอน โครงสร้างพื้นฐานของโปรแกรมคือ การรวมกลุ่มคำสั่งเป็นกลุ่มที่มีลำดับก่อนหลัง (procedure), การใช้คำสั่งกำหนดค่าให้กับตัวแปร, การใช้คำสั่งแบบมีเงื่อนไขและแบบวนรอบ ตัวอย่างภาษาในกลุ่มนี้ ได้แก่ ภาษา FORTRAN, COBOL, C

2) กลุ่มภาษาเชิงวัตถุ (object-oriented languages)

ภาษาในกลุ่มนี้สะท้อนแนวคิดการทำโปรแกรมเชิงวัตถุที่มองว่า โปรแกรมคือแหล่งรวมของวัตถุที่มีปฏิสัมพันธ์ระหว่างกันด้วยการส่งผ่านข้อความที่จะทำให้วัตถุมีการเปลี่ยนสถานะภายใน ตัวอย่างภาษาในกลุ่มนี้ได้แก่ Smalltalk, Eiffel, C++, Java

3) กลุ่มภาษาเชิงหน้าที่ (functional languages)

ภาษาในกลุ่มนี้ใช้สำหรับแนวทางการทำโปรแกรมที่มีแนวความคิดว่า โปรแกรมประกอบด้วยฟังก์ชันในเชิงคณิตศาสตร์จำนวนมาก ฟังก์ชันเหล่านี้ทำงานด้วยการรับข้อมูลเข้า (domain) แล้วเชื่อมโยงข้อมูลเหล่านี้ไปสู่ผลลัพธ์ (range) ที่สอดคล้องกัน ฟังก์ชันสามารถซ้อนอยู่ภายในฟังก์ชันอื่นได้ และการควบคุมการทำงานของโปรแกรมจะใช้การแปรค่าตามเงื่อนไข และการเรียกตัวเองซ้ำ (recursion) ตัวอย่างภาษาในกลุ่มนี้ได้แก่ ภาษา LISP, Scheme, ML, Haskell

4) กลุ่มภาษาเชิงตรรกะ (logic languages)

ภาษาหลักในกลุ่มนี้คือ ภาษา Prolog ใช้สนับสนุนแนวทางการทำโปรแกรมเชิงตรรกะ (logic programming) โปรแกรมจะประกอบด้วย ข้อความที่เป็นจริง (facts) และกฎ (rules) ที่ใช้แสดงความสัมพันธ์ ในลักษณะ IF…THEN เพื่อเชื่อมโยงความจริงที่มีอยู่ไปสู่ความจริงใหม่อื่นๆ ในลักษณะการทำนิรนัย (deduction)

เพื่อเป็นการเปรียบเทียบแนวทางการทำโปรแกรมในแต่ละกลุ่ม พิจารณาปัญหาการคำนวณค่าแฟคตอเรียล ที่มีแนวทางการคำนวณทางคณิตศาสตร์ ดังนี้

```
N! = N*(N-1)*(N-2)*(N-3)*…*2*1
```

การทำโปรแกรมเชิงคำสั่ง และเชิงวัตถุ จะมีแนวทางคล้ายกันดังนี้

```
Initialize a result variable to 1
For index values 2 through N do
     Multiply the result value by the current index value
```

การทำโปรแกรมเชิงหน้าที่ จะใช้แนวทางการเรียกตัวเองซ้ำ โดยลดค่า N ลงในแต่ละรอบจนกระทั่ง N มีค่าเป็น 1 ดังนี้

```
if (N=1)    (1)
      (Factorial (N-1)*N)
```

การทำโปรแกรมเชิงตรรกะจะใช้การกำหนดกฎ จากนั้นกระตุ้นให้โปรแกรมดำเนินงานโดยการป้อนคำถาม (query) ดังนี้

```
Rule 1: Factorial (N, N)  is true if N is 1 or 2
Rule 2: Factorial (N, R)  is true if there exist A and B
                          such that all of the followings
                          are true:
                          Factorial (A, B) is true,
                          A = N-1,
                          B = R/N.

Query: What values of R make Factorial (3, R) true?
```

การจัดกลุ่มภาษาคอมพิวเตอร์ นอกจากจัดตามแนวทางการทำโปรแกรมแล้วยังสามารถจัดตามวัตถุประสงค์การใช้งานโดยจำแนกเป็น การใช้ในงานคำนวณทางวิทยาศาสตร์ (ภาษา FORTRAN), ใช้ในงานธุรกิจ (ภาษา COBOL, RPG), ใช้ในงานปัญญาประดิษฐ์ (LISP, Prolog), ใช้ในงานการโปรแกรมระบบ (ภาษา C), ใช้ในงานที่มีวัตถุประสงค์เฉพาะ (ภาษา Javascript, Postscript, tcl/tk)

ถ้าจัดกลุ่มภาษาโดยพิจารณาระดับความใกล้เคียงกับภาษาเครื่อง จะแบ่งได้เป็นภาษาระดับต่ำ (ภาษา Assembly), ภาษาระดับสูง (ภาษา C, C++, Java, และอื่นๆ), และภาษาระดับสูงมาก (ภาษา Prolog)

1.4 ประวัติของภาษาคอมพิวเตอร์

(History of programming languages)

ภาษาคอมพิวเตอร์ที่เกิดขึ้นเป็นภาษาแรกคือ ภาษาเครื่อง (ตัวอย่างดังรูปที่ 1.1) เกิดขึ้นตั้งแต่ช่วงทศวรรษที่ 1940 จากความไม่สะดวกของการลงรหัส และการอ่านรวมถึงการแก้ไขโปรแกรม ทำให้เริ่มมีการพัฒนาเป็นภาษาแอสเซมบลี (ตัวอย่างดังรูปที่ 1.2) ที่คนสามารถอ่านและทำความเข้าใจได้ง่ายขึ้น แต่ในขณะเดียวกัน ก็ต้องมีการสร้างโปรแกรมแปลภาษาแอสเซมบลีให้เป็นภาษาเครื่อง เรียกว่า แอสเซมเบลอร์ (assembler) เพื่อให้เครื่องคอมพิวเตอร์เข้าใจและดำเนินการได้

```
27bdffd0   afbf0015   0c1002a8   00000000   0c1002a8   afa2001c   8fa4001c
00401825   10820008   0064082a   10200003   00000000   1000002    00832023
00641823   1483fffa   0064082a   0c1002b2   00000000   8fbf0014   27bd0020
03e00008   00001025
```

รูปที่ 1.1 โปรแกรมภาษาเครื่องคำนวณตัวหารร่วมมาก

```
     addiu   sp,sp,-32
     sw      ra,20(sp)              b       C
     jal     getint                 subu    a0,a0,v1
     nop                         B: subu    v1,v1,a0
     jal     getint              C: bne     a0,v1,A
     sw      v0,28(sp)              slt     at,v1,a0
     lw      a0,28(sp)           D: jal     putint
     move    v1,v0                  nop
     beq     a0,v0,D                lw      ra,20(sp)
     slt     at,v1,a0               addiu   sp,sp,32
A:   beq     at,zero,B              jr      ra
     nop                            move    v0,zero
```

รูปที่ 1.2 โปรแกรมภาษาแอสเซมบลีคำนวณตัวหารร่วมมาก

ภาษาแอสเซมบลีจัดว่าเป็นภาษาระดับต่ำ เนื่องจากคำสั่งที่ใช้ยังสัมพันธ์โดยตรงกับภาษาเครื่อง เพียงแต่เปลี่ยนจากรหัสตัวเลขเป็นข้อความภาษาอังกฤษเท่านั้น นั่นคือยังเป็นการแปลงหนึ่งคำสั่งในภาษาเครื่องให้เป็นหนึ่งคำสั่งในภาษาแอสเซมบลี ภาษาฟอร์แทรน(FORTRAN--FORmula TRANslator) เป็นภาษาระดับสูงภาษาแรกที่ถูกสร้างขึ้นในช่วงปี 1954 ภาษาฟอร์แทรนจัดเป็นภาษาระดับสูงเนื่องจากเริ่มมีการใช้คำสั่งที่สอดคล้องกับตรรกะการคิดแก้ไขปัญหาของมนุษย์มากขึ้น เช่นมีคำสั่งที่ทำงานแบบมีเงื่อนไข มีการใช้ชื่อเรียกแทนตำแหน่งต่างๆในหน่วยความจำในลักษณะของตัวแปร มีการสั่งวนรอบ เป็นต้น คำสั่งเหล่านี้หนึ่งคำสั่งต้องแปลเป็นภาษาเครื่องหลายคำสั่ง ไม่ใช่การแปลหนึ่งต่อหนึ่งเหมือนภาษาแอสเซมบลี

จากจุดกำเนิดของภาษาฟอร์แทรน ทำให้เริ่มมีการพัฒนาภาษาที่มีโครงสร้างที่สูงขึ้น เพื่อให้ใช้ในงานเขียนโปรแกรมได้มีประสิทธิภาพมากขึ้น ภาษารุ่นต่อมาที่อาจจัดได้ว่าเป็นภาษาระดับสูงรุ่นที่ 2 ที่สำคัญได้แก่ ภาษา ALGOL (ALGOrithmic Language), ภาษา LISP (LISt Processor), ภาษา COBOL (COmmon Business Oriented Language), ภาษา BASIC (Beginner's All-purpose Symbolic Instruction Code), ภาษา APL (A Programming Language), ภาษา SNOBOL (StriNg Oriented symBOlic Language)

ภาษาระดับสูงในรุ่นที่สอง เป็นภาษาที่พัฒนาขึ้นสำหรับงานเฉพาะด้าน เช่น ภาษา LISP ใช้ในงานปัญญาประดิษฐ์ที่เน้นการทำงานกับโครงสร้างข้อมูลลิสต์เป็นหลัก ภาษา COBOL ใช้ในงานทางธุรกิจที่ทำงานกับแฟ้มข้อมูลและรายการเปลี่ยนแปลงของข้อมูลเป็นหลัก ภาษา SNOBOL ที่ใช้กับงานประมวลผลข้อความ

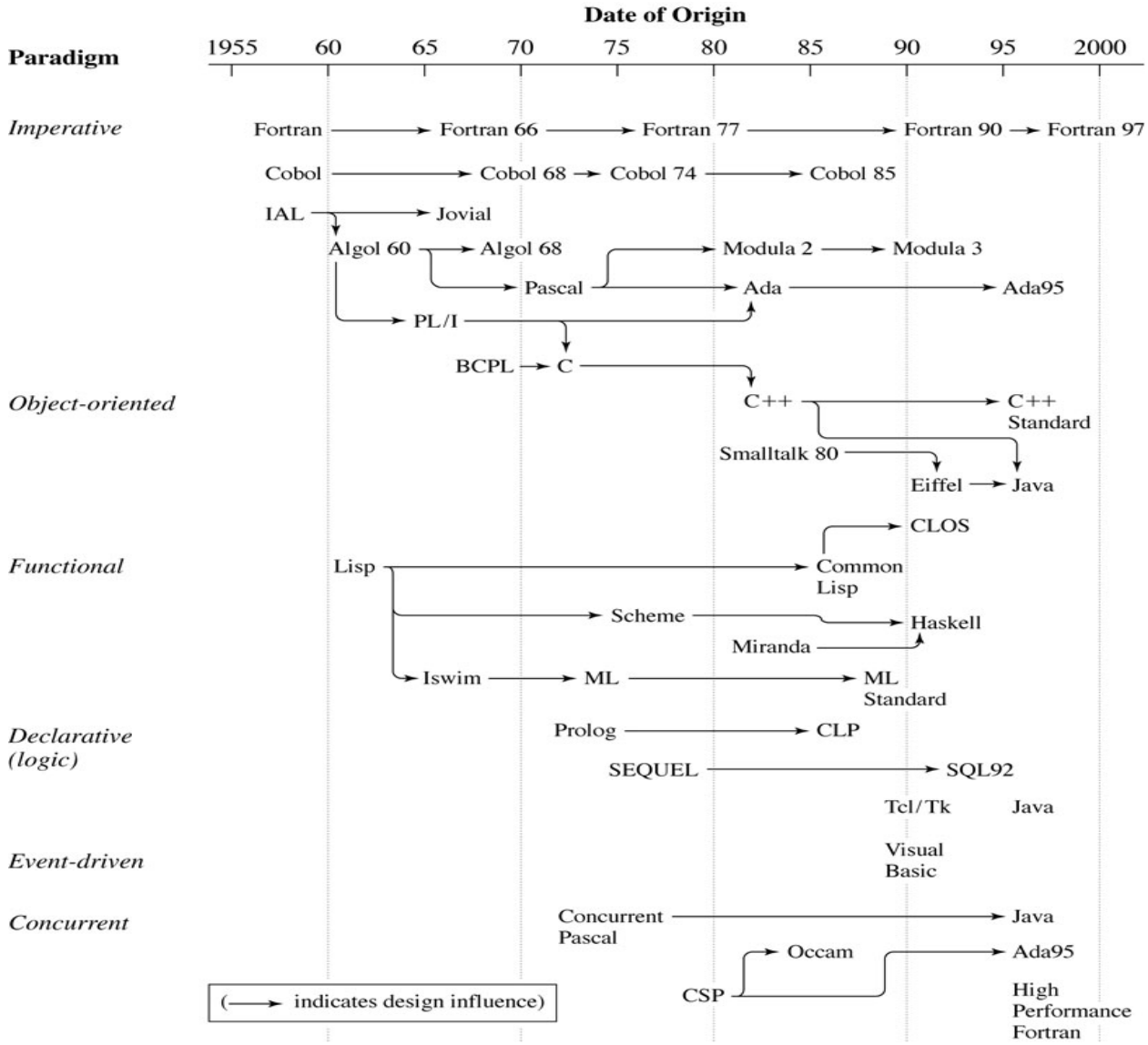
ภาษาระดับสูงในรุ่นที่สาม เป็นภาษาที่ถูกพัฒนาขึ้นเพื่อให้ทำงานได้ทุกประเภท หรือเรียกว่าเป็นภาษาเอนก ประสงค์ ได้แก่ภาษา PL/I, ภาษา SIMULA, ภาษา BCPL, ภาษา C, ภาษา Pascal ภาษาในกลุ่มนี้เป็นต้นกำเนิดของภาษาเชิงวัตถุในยุคต่อมาได้แก่ ภาษา C++, ภาษา Java

ภาษาระดับสูงในรุ่นที่สี่ เป็นภาษาที่มีระดับโครงสร้างของโปรแกรมสูงขึ้น การเขียนคำสั่งไม่ต้องระบุขั้นตอนการประมวลผลที่ละเอียด เพียงแต่ระบุรูปแบบผลลัพธ์ที่จะเกิดขึ้น เรียกภาษาในลักษณะนี้ว่าภาษาเชิงประกาศ (declarative language) ตัวอย่างภาษาในกลุ่มนี้ได้แก่ ภาษา PROLOG (PROgramming in LOGic), ภาษา SQL (Structure Query Language)

รูปที่ 1.3 แสดงจุดกำเนิดของภาษาคอมพิวเตอร์ระดับสูง ตั้งแต่ยุคเริ่มต้นจนถึงปัจจุบันและวัตถุประสงค์ของการใช้งานภาษา ในรูปที่ 1.4 แสดงเส้นเวลาที่เป็นจุดกำเนิดของภาษาคอมพิวเตอร์แยกตามกลุ่มการทำโปรแกรม

ภาษา	ปีที่สร้าง	ผู้ริเริ่ม	ภาษาที่เป็นต้นแบบ	วัตถุประสงค์การใช้งาน
FORTRAN	1954-57	J.Backus		Numeric computing
ALGOL 60	1958-60	Commitee	FORTRAN	Numeric computing
COBOL	1959-60	Commitee		Business data processing
APL	1956-60	K.Iverson		Vector processing
LISP	1956-62	J.McCarthy		Symbolic computing
SNOBOL 4	1962-66	R.Griswold		String processing
PL/I	1963-64	Commitee	FORTRAN ALGOL 60, COBOL	General purpose
BASIC	1964	J.Kemeny and T.Kurtz	FORTRAN	Educational Interactive
SIMULA 67	1967	O.-J.Dahl	ALGOL 60	Simulation
Algol 68	1963-68	Commitee	ALGOL 60	General purpose
Pascal	1971	N.Wirth	ALGOL 60	Educational (gen. purpose)
PROLOG	1972	A.Colmerauer		Artificial intelligence
C	1972	D.Ritchie	Algol 68	System programming
Mesa	1974	Commitee	SIMULA 67	System programming
SETL	1974	J.Schwartz		Very high level lang.
Concurrent Pascal	1975	P.Brinch Hansen	Pascal	Concurrent programming
Scheme	1975	G.Steele and G.Sussman	LISP	Educational (functional programming)
CLU	1974-77	B.Liskov	SIMULA 67	ADT programming
Euclid	1977	Commitee	Pascal	Verifiable programming
Modula-2	1977	N.Wirth	Pascal	System programming
Ada	1979	J.Ichbiah	Pascal, SIMULA 67	General purpose Embedded systems
Smalltalk	1971-80	A.Kay	SIMULA 67, LISP	Personal computing
OPS5	1981	Carnegie- Mellon Univ.	LISP	Expert systems
ICON	1983	R.Griswold and M.Griswold	SNOBOL 4	String processing
C++	1984	B.Stroutrup	C , SIMULA 67	General purpose
CLIPS	1984	NASA	C	C-based expert systems
ML	1984	R.Milner	LISP	Symbolic computing
Oberon-2	1987	N.Wirth etal.	Modula-2	General purpose
Eiffel	1988	B.Meyer	SIMULA 67	General purpose
CLOS	1988	Commitee	LISP, Smalltalk	OO extension of LISP
Modula-3	1989	Committee (Olivetti and DEC)	Mesa, Modula-2	System programming
Tcl/Tk	1988	J.K. Ousterhout	OS shell languages	Scripting language
Perl	1990	L.Wall	OS shell languages	Scripting language
Python	1991	G.vanRossum	OS shell languages	Scripting language
Java	1995	SUN Microsystems	C++	Network computing

รูปที่ 1.3 จุดกำเนิดของภาษาคอมพิวเตอร์ระดับสูงตั้งแต่ยุคเริ่มต้นจนถึงปัจจุบัน



รูปที่ 1.4 เส้นเวลาแสดงจุดกำเนิดของภาษาคอมพิวเตอร์ จำแนกตามกลุ่มการทำโปรแกรม

ตัวอย่างต่อไปนี้แสดงข้อความสั่งเพื่อให้พิมพ์ข้อความ “Hello World” ขึ้นทางจอภาพ ในรูปแบบของภาษาคอมพิวเตอร์แต่ละภาษา (คัดมาจาก http://www2.latech.edu/~acm/HelloWorld.shtml)

Ada	with Text_Io; use Text_Io; procdure hello is begin put ("Hello world!"); end hello;

Algol
(BEALGOL)
	BEGIN
	FILE F (KIND=REMOTE);
	EBCDIC ARRAY E [0:11];
	REPLACE E BY "HELLO WORLD!";
	WHILE TRUE DO
	BEGIN
	WRITE (F, *, E);
	END;
	END.

APL	'HELLO WORLD'

Assembly
(Z80)
	ORG 32768
	ENT
	
	LD IY, #5C3A
	RES 0, (IY+02)
	RES 1, (IY+01)
	
	LD HL, HELLO
	LD A,22
	RST #10
	LD A,0
	RST #10
	LD A,0
	RST #10
	LOOP LD A,(HL)
	PUSH AF
	PUSH HL
	AND #7F
	RST #10
	POP HL
	INC HL
	POP AF
	BIT 7,A
	JR Z, LOOP
	LD A,13
	RST #10
	LD HL, HELLO
	JR LOOP
	RET
	HELLO DEFM /Hello World/
	DEFB 161

AWK	BEGIN {
	for (;;) {
	printf("Hello World ")
	}
	}

BASIC	10 print"Hello World!" 20 goto 10

C	#include <stdio.h>
	
	main()
	{
	for(;;)
	{
	printf ("Hello World!\n");
	}
	}

C++	#include <iostream.h>
	
	main()
	{
	for(;;)
	{
	cout << "Hello World! ";
	}}

COBOL	000100 IDENTIFICATION DIVISION. 000200 PROGRAM-ID. HELLOWORLD. 000300 DATE-WRITTEN. 02/05/96 21:04. 000400* AUTHOR BRIAN COLLINS 000500 ENVIRONMENT DIVISION. 000600 CONFIGURATION SECTION. 000700 SOURCE-COMPUTER. RM-COBOL. 000800 OBJECT-COMPUTER. RM-COBOL. 000900 001000 DATA DIVISION. 001100 FILE SECTION. 001200 100000 PROCEDURE DIVISION. 100100 100200 MAIN-LOGIC SECTION. 100300 BEGIN. 100400 DISPLAY " " LINE 1 POSITION 1 ERASE EOS. 100500 DISPLAY "HELLO, WORLD." LINE 15 POSITION 10. 100600 STOP RUN. 100700 MAIN-LOGIC-EXIT. 100800 EXIT.

Euphoria	puts(1, "Hello World!\n")
FORTRAN	PROGRAM HELLO DO 10, I=1,10 PRINT *,'Hello World' 10 CONTINUE STOP END
Haskell	main = print("Hello World")
HTML	<HTML> <HEAD> <TITLE>Hello, World Page!</TITLE> </HEAD> <BODY> Hello, World! </BODY> </HTML>

Java	class HelloWorld {
	public static void main (String args[]) {
	for (;;) {
	System.out.print("Hello World ");
	}
	}
	}

JavaScript	<TITLE> Hello World in JavaScript </TITLE> <SCRIPT> document.write ("Hello, world!") </SCRIPT>
LISP	; LISP (DEFUN HELLO-WORLD () (PRINT (LIST 'HELLO 'WORLD)))
Pascal	Program Hello (Input, Output); Begin Writeln ('Hello World!'); End.
Perl	print "Hello, World!\n" while (1);

PL/1	HELLO: PROCEDURE OPTIONS (MAIN); /* A PROGRAM TO OUTPUT HELLO WORLD */ FLAG = 0; LOOP: DO WHILE (FLAG = 0); PUT SKIP DATA('HELLO WORLD!'); END LOOP; END HELLO;
Python	while (1) : print "Hello World";
RPG	H FSCREEN O F 80 80 CRT C EXCPT OSCREEN E 1 O 12 'HELLO WORLD!'
Scheme	(define hello-world (lambda () (begin (write 'Hello-World) (newline) (hello-world))))

Simula	Begin
	while 1 = 1 do begin
	outtext ("Hello World!");
	outimage;
	end;
	End;

Smalltalk	Transcript show:'Hello World';cr
SNOBOL	OUTPUT = 'Hello World!' END

tcl	while {1} {
	puts "Hello World "
	}

tk	#!/usr/bin/wish button .bob -text " Hello World !" -command { puts " Hello World !" } button .bye -text "bye" -command { destroy .} pack .bob pack .bye

CC+ (concurrent language)	`#include`
	
	`int main()`
	`{`
	`  char *s1, *s2;`
	`  par  {`
	`  s1 = "hello, ";`
	`  s2 = "world\n";`
	`  }`
	`  cout << s1 << s2 << endl;`
	`  return(0);`
	`}`

Visual BASIC	`VERSION 2.00`
	`Begin Form Form1`
	`  Caption  =  "Hello World!"`
	`  ClientHeight  =  540`
	`  ClientLeft  =  1095`
	`  ClientTop  =  1515`
	`  ClientWidth  =  2445`
	`  Height  =  945`
	`  Left  =  1035`
	`  LinkTopic  =  "Form1"`
	`  ScaleHeight  =  540`
	`  ScaleWidth  =  2445`
	`  Top  =  1170`
	`  Width  =  2565`
	`  Begin Label Label1`
	`  Alignment  =  2  'Center`
	`  Caption  =  "Hello World!"`
	`  FontBold  =  -1  'True`
	`  FontItalic  =  0  'False`
	`  FontName  =  "Arial"`
	`  FontSize  =  12`
	`  FontStrikethru  =  0  'False`
	`  FontUnderline  =  0  'False`
	`  Height  =  255`
	`  Left  =  120`
	`  TabIndex  =  0`
	`  Top  =  120`
	`  Width  =  2175`
	`  End`
	`End`

SQL	`CREATE TABLE HELLO (HELLO CHAR(12))`
	`UPDATE HELLO`
	`  SET HELLO = 'HELLO WORLD!'`
	`SELECT * FROM HELLO`

PHP	<?php echo "<html> <head> <title>Hello World Page</title> </head> <body> Hello World! </body> </html>" ?>

1.5 สรุป

การศึกษาแนวคิดและหลักการของภาษาการทำโปรแกรม เป็นการศึกษาในเชิงวิเคราะห์เปรียบเทียบโครงสร้างของภาษาในแต่ละกลุ่มการทำโปรแกรม โดยจำแนกเป็น 4 กลุ่มหลัก คือ ภาษาที่ใช้ในการทำโปรแกรมเชิงคำสั่ง (imperative programming languages), ภาษาที่ใช้ในการทำโปรแกรมเชิงวัตถุ (object-oriented programming languages), ภาษาที่ใช้ในการทำโปรแกรมเชิงหน้าที่ (functional programming languages) และ ภาษาที่ใช้ในการโปรแกรมเชิงตรรกะ (logic programming languages)

การวิเคราะห์ภาษาในกลุ่มต่างๆ ดังกล่าว เป็นการวิเคราะห์รูปแบบไวยากรณ์ของภาษา ความหมายของคำสั่งในภาษา โครงสร้างคำสั่งพื้นฐานที่ภาษานั้นๆมีให้ใช้ ตลอดจนการควบคุมการทำงานของโปรแกรมด้วยคำสั่งในภาษานั้นๆ การศึกษานี้เพื่อประโยชน์ในการทำความเข้าใจจุดเด่น-จุดด้อยของแต่ละภาษาเพื่อให้สามารถเลือกใช้ภาษาได้เหมาะสมกับงาน สามารถเขียนโปรแกรมภาษานั้นๆได้อย่างมีประสิทธิภาพมากขึ้น ตลอดจนถึงสามารถเข้าใจกระบวนการออกแบบและสร้างภาษาคอมพิวเตอร์

แบบฝึกหัดท้ายบทที่ 1

คำถามอัตนัย

1. ให้เขียนวัฏจักรการพัฒนาซอฟต์แวร์ (software development life cycle) ในลักษณะแผนภาพหรือไดอะแกรม
2. ภาษาคอมพิวเตอร์มีผลกระทบอย่างไรกับวัฏจักรการพัฒนาซอฟต์แวร์?
3. ทำไมเราจึงต้องศึกษาเกี่ยวกับหลักการของภาษาการทำโปรแกรม?
   ให้อธิบายเหตุผลมาเป็นข้อๆ อย่างน้อย 3 ข้อ
4. ภาษาคอมพิวเตอร์ในปัจจุบันมีมากกว่า 3,000 ภาษา ในจำนวนภาษาที่มากมายเหล่านี้ สามารถจัดกลุ่มได้เป็นกี่กลุ่ม? กลุ่มใดบ้าง? ให้ระบุชื่อภาษาคอมพิวเตอร์ในแต่ละกลุ่มมาอย่างน้อยหนึ่งภาษา
5. ภาษาคอมพิวเตอร์ที่ดีควรมีลักษณะอย่างไรบ้าง? ให้ชี้แจงมาเป็นข้อๆ อย่างน้อย 3 ข้อ

คำถามปรนัย: ให้เลือกคำตอบที่ถูกต้องที่สุด

1. ในการออกแบบภาษาคอมพิวเตอร์เราต้องคำนึงถึงอะไรบ้าง เพื่อให้เกิดความสะดวกในการใช้งานมากที่สุด ?
   ก. รูปแบบไวยากรณ์จะต้องสั้นกระทัดรัด
   ข. ไวยากรณ์ของภาษาจะต้องง่ายต่อการจดจำ
   ค. รูปแบบไวยากรณ์จะต้องเขียนอย่างมีโครงสร้าง
   ง. ไวยากรณ์จะต้องนิยามชัดเจน ไม่กำกวม

2. ข้อใดไม่ใช่ลักษณะของภาษาที่ดี ?
   ก. ง่ายต่อการใช้งาน
   ข. เกิด exceptions ได้ง่าย
   ค. มีความยืดหยุ่นของภาษาพอดี
   ง. ไวยากรณ์ของภาษาจะต้องเอื้อให้เราเข้าใจง่าย

3. ภาษาที่มี orthogonality สูง มีลักษณะเป็นอย่างไร ?
   ก. การใช้ภาษาจะมีข้อจำกัดและข้อยกเว้นมาก
   ข. มีความยืดหยุ่นสูง เขียนโปรแกรมได้ง่าย
   ค. อำนวยความสะดวกในการเขียน exception
   ง. มีไวยากรณ์สั้นและรูปแบบชัดเจน

4. ข้อใดกล่าวถูกต้อง ?
   ก. ภาษา Java ไม่อนุญาตให้มี alias
   ข. ภาษาที่มี alias ทำให้เขียนโปรแกรมได้สะดวก
   ค. ภาษาที่ดีจะต้องห้ามมี alias
   ง. ภาษาที่มี alias ทำให้ใช้งานยาก

5. ข้อใดต่อไปนี้เป็น Programming Language Paradigms ?

ก. Imperative,	Object-oriented,	Logic,	Functional
ข. Object-oriented,	Abstract,	Logic,	Functional
ค. Procedural,	Object-oriented,	Logic,	von Neumann
ง. Nonprocedural,	Object-oriented,	Functional,	von Neumann

6. คุณสมบัติของภาษาคอมพิวเตอร์ในข้อใดที่ช่วยเพิ่มความถูกต้องให้กับการทำงานของโปรแกรม ?

ก. exception handling และ orthogonality

ข. exception handling และ type checking

ค. type checking และ overloading

ง. type coercion และ orthogonality

7. ลักษณะใดถือเป็นลักษณะของภาษาคอมพิวเตอร์ที่ไม่ดี ?

ก. การมีตัวแปร boolean และการมีคำสั่ง goto

ข. การมี exception และการทำ aliasing

ค. การมีคำสั่ง goto และการทำ aliasing

ง. การมี exception และการทำ overloading

8. ภาษาคอมพิวเตอร์ที่ดีจะต้องถูกออกแบบให้สามารถตรวจสอบความถูกต้องของชนิดข้อมูลได้ ความสามารถเช่นนี้เรียกว่าอะไร ?

ก. parameter checking ข. type checking

ค. orthogonality ง. functionality

9. ภาษาคอมพิวเตอร์ต่อไปนี้ภาษาใดเกิดขึ้นก่อน ?

ก. FORTRAN I ข. ASSEMBLY

ค. COBOL ง. ALGOL

10. ภาษาในระดับต่ำ (low level) มีลักษณะอย่างไร ?

ก. การใช้งานมีความซับซ้อนต่ำ เหมาะสำหรับผู้เริ่มต้นเขียนโปรแกรม

ข. เป็นภาษาที่มีระดับความซับซ้อนต่ำ ใกล้เคียงกับภาษาเครื่อง

ค. จำนวนผู้ใช้งานมีต่ำ ไม่นิยมนำมาเขียนโปรแกรม

ง. การลงทุนเพื่อนำภาษามาใช้งานต่ำ ราคาถูก