บทที่ 7

==========

การทำโปรแกรมเชิงคำสั่ง
(Imperative programming)

วัตถุประสงค์

1) เพื่อให้ผู้เรียนรู้จักแนวคิดของการทำโปรแกรมเชิงคำสั่ง
2) เพื่อให้ผู้เรียนเข้าใจถึงลักษณะการทำงานของเครื่องคอมพิวเตอร์ที่มีอิทธิพลต่อการออกแบบภาษาคอมพิวเตอร์และวิธีการทำโปรแกรม
3) เพื่อให้เข้าใจบทบาทของตัวแปรและความสำคัญของชนิดข้อมูล
4) เพื่อให้รู้จักวิธีการควบคุมลำดับการทำงานของโปรแกรมด้วยคำสั่งประเภทต่างๆ
5) เพื่อให้เข้าใจการทำโปรแกรมแบบโครงสร้าง และวิธีการพัฒนาโปรแกรมที่ทนต่อข้อบกพร่อง

การทำโปรแกรมเชิงคำสั่ง เป็นรูปแบบหรือวิธีการทำโปรแกรมที่ใช้มาตั้งแต่ยุคแรกของคอมพิวเตอร์ ที่การสั่งงานยังต้องใช้ภาษาเครื่องที่เป็นรหัส 0, 1 และพัฒนามาเป็นภาษาที่รหัสคำสั่งจดจำได้ง่ายขึ้น เช่น ภาษา FORTRAN, ADA, Pascal, C โครงสร้างคำสั่งที่ภาษาเหล่านี้เตรียมไว้ให้ผู้ใช้เรียกใช้งานมีมากขึ้นและช่วยให้การทำโปรแกรมสะดวกขึ้น แต่หลักคิดหรือวิธีการคิดเพื่อการสร้างโปรแกรมยังคงเป็นเช่นเดิมไม่เปลี่ยนแปลง นั่นคือ การทำโปรแกรมจะใช้วิธีการเขียนคำสั่งเพื่อควบคุมตัวแปรที่ทำหน้าที่เก็บข้อมูล จากนั้นทำการควบคุมการเปลี่ยนค่าของตัวแปรไปตามลำดับจนกระทั่งได้ค่าสุดท้ายที่เป็นผลลัพธ์ที่ต้องการ โดยการควบคุมการเปลี่ยนแปลงค่านี้ จะกระทำผ่านการใช้คำสั่งประเภทต่างๆ เช่นคำสั่งกำหนดค่า, คำสั่งเลือกทำ, คำสั่งทำซ้ำ และการสั่งงานนี้จะต้องกระทำเป็นขั้นตอนโดยละเอียด เมื่อโปรแกรมมีรายละเอียดที่ซับซ้อนขึ้นการระบุขั้นตอนการทำงานก็จะเพิ่มมากขึ้นตามไปด้วย จึงต้องมีการแบ่งคำสั่งออกเป็นกลุ่มย่อยเพื่อทำหน้าที่เฉพาะส่วน การทำโปรแกรมเชิงคำสั่งถึงแม้จะมีขั้นตอนการเขียนโปรแกรมค่อนข้างมากเมื่อเปรียบเทียบกับวิธีการทำโปรแกรมแบบอื่น เช่น การทำโปรแกรมเชิงตรรกะ, การทำโปรแกรมเชิงหน้าที่ แต่วิธีการคิดในการทำโปรแกรม จะตรงกับวิธีการคิดโดยปกติของมนุษย์เมื่อจะต้องมีการปฏิบัติงานตามขั้นตอน เช่น ขั้นตอนการปรุงอาหาร, ขั้นตอนการลงทะเบียนเรียน, ขั้นตอนการทำบัตรประจำตัวประชาชน ดังนั้นการทำโปรแกรมเชิงคำสั่งจึงยังคงเป็นวิธีการทำโปรแกรมที่นิยมใช้อยู่มากในปัจจุบัน

7.1 หลักการทำโปรแกรมเชิงคำสั่ง

(Principles of imperative programming)

เครื่องคอมพิวเตอร์ที่ถูกสร้างขึ้นในยุคแรก ประมาณปลายทศวรรษที่ 1940 แยกส่วนเก็บโปรแกรมไว้ต่างหากจากส่วนหน่วยความจำที่ใช้เก็บข้อมูล เมื่อจะรันโปรแกรมจะใช้วิธีนำบอร์ดมาต่อสายเชื่อมกับซีพียู เพื่อโปรแกรมผ่านสวิทชิงบอร์ดนั้น นักคณิตศาสตร์และนักคอมพิวเตอร์ในยุคนั้นประกอบด้วย John von Neumann, Alan Turing และนักคอมพิวเตอร์อื่นๆ ได้สังเกตเห็นถึงความด้อยประสิทธิภาพที่เกิดจากการใช้คอมพิวเตอร์ในรุ่นแรก จึงได้ออกแบบระบบคอมพิวเตอร์ขึ้นใหม่ให้มีหน่วยประมวลผลกลางบรรจุวงจรต่างๆ ที่ต้องใช้ในการคำนวณ และมีหน่วยความจำเพื่อใช้เก็บทั้งข้อมูลและโปรแกรม การนำโปรแกรมมาเก็บไว้ในหน่วยความจำ เป็นจุดริเริ่มที่สำคัญที่ทำให้การประมวลผลด้วยเครื่องคอมพิวเตอร์ที่มีประสิทธิภาพสูงขึ้นมาก โครงสร้างพื้นฐานในการออกแบบระบบคอมพิวเตอร์นี้ถูกใช้มาตลอดจนถึงคอมพิวเตอร์ในยุคปัจจุบัน และเรียกโครงสร้างนี้ว่าสถาปัตยกรรมฟอนนอยแมน (von Neumann architecture)

จุดเด่นของสถาปัตยกรรมฟอนนอยแมนก็คือ เครื่องคอมพิวเตอร์ (ซึ่งในยุคแรกก็คือเครื่องจักรเพื่อการคำนวณที่ทำงานได้โดยอัตโนมัติ) ต้องมีหน่วยความจำ และในหน่วยความจำจะใช้บันทึกทั้งข้อมูลและชุดคำสั่งที่ทำงานกับข้อมูลนั้น ลักษณะของสถาปัตยกรรมนี้จึงมีอิทธิพลต่อการทำโปรแกรม หรือวิธีการเขียนชุดคำสั่ง รวม

ไปถึงมีอิทธิพลต่อการออกแบบภาษาเพื่อใช้ในการเขียนโปรแกรม โดยภาษาจะต้องมีคำสั่งในการประกาศชื่อตัวแปร, คำสั่งที่ทำงานกับนิพจน์, คำสั่งที่กำหนดลำดับการประมวลผลในโปรแกรม

คำสั่งประกาศตัวแปรใช้เพื่อกำหนดชื่อเรียกแทนแต่ละตำแหน่งในหน่วยความจำที่จะใช้เก็บค่าของข้อมูล รวมถึงกำหนดชนิดของข้อมูลที่เก็บในหน่วยความจำตำแหน่งนั้นๆ คำสั่งเกี่ยวกับนิพจน์ ใช้ทำงานกับค่าปัจจุบันที่เก็บอยู่ในหน่วยความจำตำแหน่งต่างๆ คำสั่งกำหนดลำดับการประมวลผล ซึ่งประกอบด้วยคำสั่งกำหนดค่า, คำสั่งมีเงื่อนไข, คำสั่งแยก (branching statement) ซึ่งในภาษาคอมพิวเตอร์รุ่นแรกๆ ใช้ในการสั่งทำงานซ้ำ ดังตัวอย่างต่อไปนี้ที่เป็นการคำนวณ a = a + b คำสั่งแยกคือการสั่ง goto ร่วมกับการใช้ เลเบล L

```
L :   a = a + 1
      b = b – 1
      if b > 0 then goto L
      halt
```

ภาษาคอมพิวเตอร์จะเรียกว่า สมบูรณ์ตามข้อกำหนดทัวริง (Turing complete) ถ้าภาษานั้นมีตัวแปรชนิดเลขจำนวนเต็มพร้อมทั้งค่า และการคำนวณกับค่าเลขจำนวนเต็ม, มีคำสั่งกำหนดค่าให้กับตัวแปร, และมีคำสั่งควบคุมลำดับการทำงานภายในโปรแกรม ซึ่งประกอบด้วย การทำงานตามลำดับ (sequencing), การทำงานแบบมีเงื่อนไข (conditional), การทำงานแบบมีทางแยก (branching)

ในปัจจุบันภาษาคอมพิวเตอร์เพื่อการทำโปรแกรมเชิงคำสั่ง คือภาษาที่สมบูรณ์ตามข้อกำหนดทัวริง และมีลักษณะต่อไปนี้เพิ่มเติม

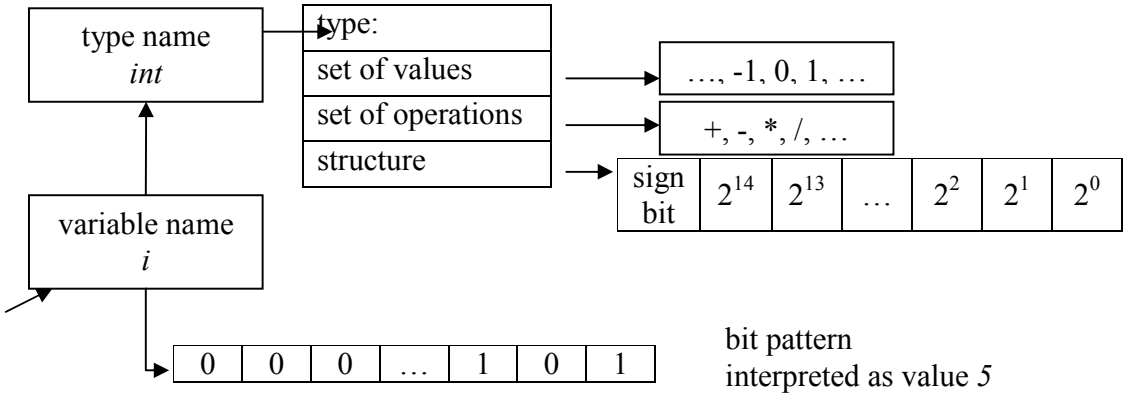
- ชนิดข้อมูลสำหรับเลขจำนวนจริง, อักขระ, สตริง, ค่าตรรกะ พร้อมทั้งโอเปอเรเตอร์ ที่ใช้ทำงานกับข้อมูลเหล่านั้น
- คำสั่งเพื่อการทำงานซ้ำในลักษณะวนรอบ (for และ while) และคำสั่งเลือกทำ (case หรือ switch)
- ข้อมูลอาร์เรย์และคำสั่งทำงานกับข้อมูลในอาร์เรย์
- ข้อมูลเรคคอร์ด (หรือ struct) และคำสั่งทำงานกับข้อมูลในแต่ละฟิลด์ของเรคคอร์ด
- คำสั่งรับข้อมูลและแสดงผลข้อมูล
- ข้อมูลพ็อยเตอร์ หรือข้อมูลเพื่อการอ้างอิงค่าโดยอ้อม (indirect addressing)
- โพรซีเจอร์และฟังก์ชัน

การทำโปรแกรมเชิงคำสั่งในบางครั้งเรียกว่า การทำโปรแกรมแบบกระบวนงาน (procedural programming) เพราะใช้วิธีการระบุโพรซีเจอร์ หรือระบุขั้นตอนการทำงานโดยละเอียด และภาษาเพื่อการทำโปรแกรมเชิงคำสั่ง อาจเรียกชื่ออีกอย่างได้ว่าภาษากระบวนงาน (procedural language)

7.2 ตัวแปรและชนิดข้อมูล

(Variables and types)

ในการทำโปรแกรมเชิงคำสั่ง จะมีการใช้ตัวแปรเพื่อเป็นชื่อที่ใช้อ้างถึงตำแหน่งในหน่วยความจำ ซึ่งเป็นเนื้อที่ที่ใช้เก็บข้อมูล การระบุชื่อตัวแปรจึงเป็นการเชื่อมโยงถึงค่าของข้อมูล ตัวแปรไม่เพียงเชื่อมโยงถึงข้อมูลแต่ยังเชื่อมโยงถึงชนิดข้อมูล ชนิดข้อมูลทำหน้าที่ระบุว่าค่าประเภทใดจึงจะเก็บในตัวแปรนั้นได้, เซตของค่าที่ถูกต้องเป็นค่าใดบ้าง, โอเปอเรเตอร์ที่ทำงานกับตัวแปรนั้นๆ ได้มีอะไรบ้าง, และโครงสร้างภายในที่ใช้เก็บค่าของตัวแปรมีรูปแบบอย่างไร รูปที่ 7.1 แสดงความสัมพันธ์ของชื่อตัวแปร, ค่าของตัวแปร และโครงสร้างภายในที่สัมพันธ์กับชนิดข้อมูลของตัวแปร

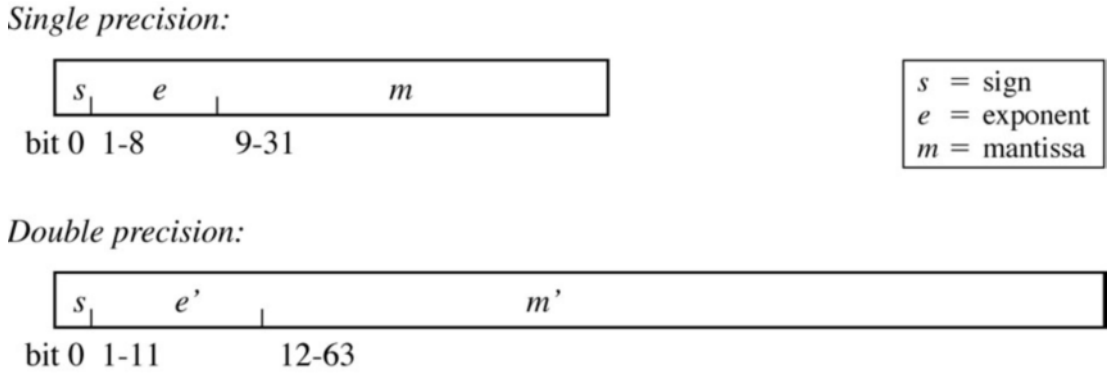


รูปที่ 7.1 ความสัมพันธ์ของชื่อตัวแปร, ค่าของข้อมูลและโครงสร้างที่เกี่ยวข้องเมื่อมีการประกาศตัวแปร *int i=5*

ค่าของตัวแปรในมุมมองด้านฮาร์ดแวร์เป็นเพียงรูปแบบบิต และชนิดข้อมูลเป็นวิธีการแปลงค่ารูปแบบบิตเหล่านั้น แต่ในมุมมองของภาษาของคอมพิวเตอร์ ชนิดข้อมูลจะเป็นเซตของค่าและเซตของการดำเนินงานที่สามารถกระทำกับค่าเหล่านั้น ในทางฮาร์ดแวร์จะมีการเตรียมค่าของข้อมูลไว้ให้ใช้เพียง 3 ประเภท คือ อักขระ, เลขจำนวนเต็ม, เลขจำนวนจริง

- อักขระ (characters) คือการใช้เนื้อที่หนึ่งไบต์เก็บตัวเลข {0..255} ซึ่งตัวเลขแต่ละจำนวน จะถูกใช้เป็นรหัสแทนอักขระแต่ละตัว เช่น 68 แทน h
- เลขจำนวนเต็ม (integer numbers) จะมีขนาดและช่วงของค่าที่แตกต่างกัน เช่น signed integer ขนาด 16 บิต จะมีช่วงของค่า {-32768..+32767}, unsigned integer ขนาด 16 บิต มีช่วงของค่า {0..65535}, signed integer ขนาด 32 บิต มีช่วงของค่า {-2,147,483,648 .. +2,147,483,647}
- เลขจำนวนจริง (real number or floating point number) จะมีขนาดและช่วงของค่าที่แตกต่างกัน รูปที่ 7.2 แสดงขนาดของเลขจำนวนจริงสองขนาดตามมาตรฐาน IEEE 754 การแทนเลขจำนวนจริงจะใช้

รูปแบบค่าทางวิทยาศาสตร์ เช่น 123.45 จะถูกเปลี่ยนเป็นรูปแบบ $1.2345*10^2$ ซึ่งเมื่อแปลงเป็นเลขฐานสองจะอยู่ในรูปแบบ $\pm 1.m *2^{127-e}$ โดย m เรียกว่า mantissa และ e เรียกว่า exponent



รูปที่ 7.2 การแทนเลขจำนวนจริงตามมาตฐาน IEEE 754

ภาษาคอมพิวเตอร์โดยทั่วไปจึงมักจะมีชนิดข้อมูลพื้นฐานประเภท อักขระ, เลขจำนวนเต็ม, เลขจำนวนจริง เตรียมไว้ให้ผู้ใช้เรียกใช้ได้ โดยไม่ต้องนิยามหรือสร้างขึ้นใหม่ รูปที่ 7.3 แสดงชนิดข้อมูลพื้นฐานในภาษา C, C++, Ada และ Java

Type	**C/C++**	**Ada**	**Java**
Byte			byte
Integer	short, int, long	integer	short, int, long
Real number	float, double, long	float, decimal	float, double
Character	char	character	Char
Boolean		boolean	Boolean
Pointer	*	access	

รูปที่ 7.3 ชนิดข้อมูลพื้นฐานในภาษาคอมพิวเตอร์บางภาษา

จากโครงสร้างของชนิดข้อมูลที่ประกอบด้วยเซตของค่า, เซตของการดำเนินงาน และอื่นๆ ในกรณีที่เซตของค่า เป็นเซตว่าง จะหมายถึงรูปแบบบิตเป็นบิต 0 ทั้งหมด ซึ่งในภาษา C, C++, Java จะเรียกข้อมูลนี้ว่า void

ภาษาคอมพิวเตอร์ในปัจจุบันนอกจากจะเตรียมชนิดข้อมูลพื้นฐานไว้ให้ผู้ใช้เรียกใช้ได้ทันทีแล้ว ยังมี type constructors หรือ ตัวสร้างชนิดข้อมูล ที่ทำหน้าที่อำนวยความสะดวกให้ผู้ใช้สามารถสร้างชนิดข้อมูลใหม่ขึ้นจากชนิดข้อมูลเดิมที่มีอยู่ เช่น

int a[10];

เป็นการสร้างชนิดข้อมูลอาร์เรย์ขึ้นจากชนิดข้อมูล int ที่มีอยู่เป็นพื้นฐานแล้ว โดย type constructor คือ สัญลักษณ์ […] หรือตัวอย่างในภาษา C ต่อไปนี้

```
typedef enum {
        red, amber, green
} traffic_light_color;
```

เป็นการประกาศสร้างชนิดข้อมูลขึ้นใหม่ ชื่อ traffic_light_color โดยมี type constructor คือ enum {…} และตัวอย่างต่อไปนี้แสดง type constructor เพื่อสร้างชนิดข้อมูลเรคคอร์ด struct {…} โดยตัวอย่างแรกเป็นเรคคอร์ดที่เกี่ยวกับรายละเอียดรถยนต์ และตัวอย่างที่สองเป็น เรคคอร์ดของโครงสร้างไบนารีทรี

```
typedef enum { BMW, TOYOTA, HONDA, FORD } brand_type;
typedef struct {
        brand_type brand;
        unsigned int number_of_doors;
        int price;
} car_type;
typedef struct {
        struct bin_tree_type *left, *right;
          int value;
} bin_tree_type;
```

การมี type constructor เพื่อใช้ในการสร้างชนิดข้อมูลใหม่ขึ้นจากชนิดข้อมูลเดิมที่มีอยู่ ทำให้เกิดหลักการในการออกแบบภาษาที่เรียกว่า ความเป็นอิสระในการผสมลักษณะ หรือ orthogonality นั่นคือ โดยหลักการแล้ว type constructors ที่ปรากฏในภาษาคอมพิวเตอร์ควรจะสามารถถูกกผสมรวมได้อย่างอิสระ เพื่อสร้างชนิดข้อมูลใหม่ เช่น ผู้ใช้ควรจะสามารถสร้างอาร์เรย์ที่มีเรคคอร์ดเป็นสมาชิกในอาร์เรย์, สร้างเรคคอร์ดที่มีอาร์เรย์เป็นฟิลด์ภายในเรคคอร์ด, สร้างเซตของเซตของพ็อยเตอร์ที่ชี้ไปที่ข้อมูลยูเนียน เป็นต้น

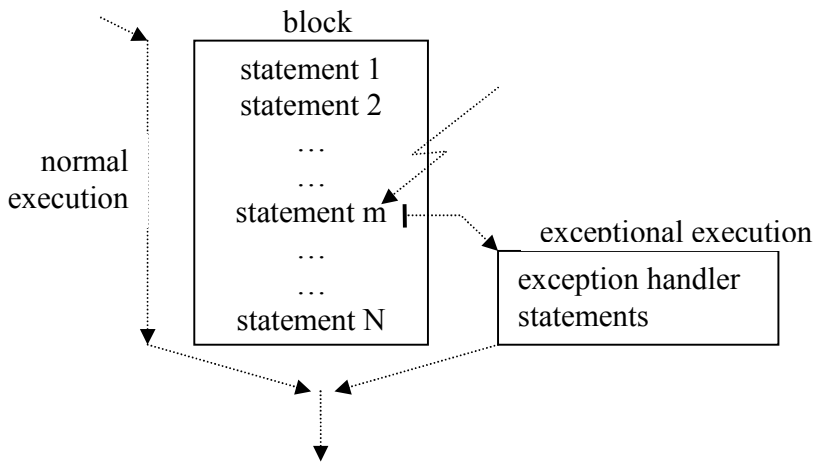
ความเป็นอิสระในการผสมลักษณะ เริ่มถูกใช้เป็นเกณฑ์ในการออกแบบภาษาคอมพิวเตอร์ตั้งแต่มีการออก แบบภาษา Algol โดยภาษา Algol 68 จัดว่าเป็นภาษาที่ให้อิสระกับโปรแกรมเมอร์สูงมากในการผสมลักษณะต่างๆเป็นข้อความสั่ง ในขณะที่ภาษา C จัดว่าให้ความเป็นอิสระต่ำกว่า เมื่อเปรียบเทียบกับ Algol 68 เนื่องจากยังมีข้อยกเว้นหรือข้อจำกัดในการผสมบางลักษณะ เช่น การประกาศตัวแปร int สามารถกำหนดค่าเริ่มต้นพร้อมกับการประกาศ (ตัวอย่าง int x=10;) แต่การประกาศตัวแปรที่เป็นชนิดยูเนียนไม่สามารถกำหนดค่าเริ่มต้นได้

7.3 การควบคุมลำดับการทำงาน

(Flow of control)

การทำงานตามปกติของโปรแกรมเชิงคำสั่งจะเป็นแบบ sequence หรือ การทำงานแบบเป็นลำดับโดยเริ่มจากคำสั่งที่ 1, 2, 3, .. เรียงลำดับเรื่อยไปจนจบโปรแกรม แต่ถ้าการทำงานของโปรแกรมเป็นแบบ sequence เพียงอย่างเดียว ความสามารถของโปรแกรมจะมีจำกัด ภาษาคอมพิวเตอร์จึงต้องมีคำสั่งประเภทอื่นๆ ให้เลือกใช้เพื่อเปลี่ยนลำดับการทำงานตามปกติของโปรแกรมได้ คำสั่งประเภทที่ต้องมีในภาษาคอมพิวเตอร์คือ คำสั่งแบบมีเงื่อนไขในการเลือกทำหรือไม่ทำคำสั่งบางช่วง เรียกว่า คำสั่งแบบมีการเลือก (selection) และคำสั่งอีกประเภทที่ต้องมีในภาษาคอมพิวเตอร์คือ คำสั่งทำงานซ้ำ (iteration) คำสั่งทั้งสามประเภทนี้ (sequence, selection, iteration) จะเป็นโครงสร้างหลักที่ใช้ควบคุมลำดับการทำคำสั่งภายในโปรแกรม

การควบคุมลำดับการทำงานของโปรแกรมด้วยคำสั่งเป็นลำดับ, มีการเลือกและการวนซ้ำ เป็นการควบคุมในกรณีที่ไม่มีเหตุการณ์ผิดปกติเกิดขึ้นในโปรแกรม แต่เมื่อมีเหตุการณ์พิเศษเกิดขึ้น เช่น การหารด้วยค่าศูนย์, การเปิดไฟล์เพื่ออ่านข้อมูลไม่สามารถทำได้, หรือ การอ้างตำแหน่งค่าสมาชิกในอาร์เรย์เกินขอบเขตของอาร์เรย์ ในกรณีผิดปกติเช่นนี้ ลำดับการทำงานของโปรแกรมจะเปลี่ยนไป ซึ่งถ้าไม่มีการควบคุมอื่นเตรียมรองรับไว้ ผลการทำงานของโปรแกรมจะผิดพลาด กลไกที่ใช้ควบคุมกรณีไม่ปกติต่างๆ คือตัวจัดการกรณียกเว้น (exception handler) ซึ่งเป็นกลไกที่สำคัญในการพัฒนาโปรแกรมที่ต้องการความน่าเชื่อถือสูง ลำดับการทำงานในกรณีที่มีเหตุการณ์ผิดปกติ แสดงเป็นแผนภาพได้ดังรูปที่ 7.4



รูปที่ 7.4 บล็อกของคำสั่งที่มีการควบคุมลำดับการทำงานทั้งในกรณีปกติและกรณีมีเหตุการณ์ยกเว้น

7.4 ส่วนประกอบของโปรแกรม

(Program composition)

ในการทำโปรแกรมคำสั่งส่วนประกาศตัวแปร และคำสั่งควบคุมลำดับการทำงานทั้งหลาย จะถูกรวมกลุ่มเพื่อประกอบกันเป็นโครงสร้างของหนึ่งโปรแกรม ภาษา Pascal เป็นภาษาที่มีการกำหนดโครงสร้างของโปรแกรมอย่างชัดเจน แสดงได้ดังรูปที่ 7.5

```
program program_name (files);
declarations of constants, types, variables, procedures and functions;
begin
        statements (no declarations)
end.
```

รูปที่ 7.5 โครงสร้างโปรแกรมในภาษา Pascal

ภาษาปาสคาลกำหนดโครงสร้างของโปรแกรมให้ประกอบด้วยชื่อโปรแกรม, ส่วนประกาศและส่วนคำสั่ง ส่วนประกาศจะเริ่มต้นด้วยการประกาศค่าคงที่, การประกาศชนิดข้อมูลที่ผู้ใช้สร้างขึ้นใหม่ (เช่น เรคคอร์ด) , การประกาศตัวแปรรวมถึงชนิดข้อมูลของตัวแปร, และส่วนสุดท้ายเป็นการประกาศโพรซีเจอร์และฟังก์ชัน โดยอาจจะประกาศเฉพาะโครงสร้างโดยยังไม่มีรายละเอียดคำสั่ง (เรียกว่า การประกาศล่วงหน้า หรือ forward declaration ซึ่งตรงกับฟังก์ชัน prototype ในภาษา C) หรือเป็นการเขียนโพรซีเจอร์/ฟังก์ชันโดยสมบูรณ์ก็ได้ ต่อจากส่วนประกาศจะเป็นส่วนคำสั่งที่มีคำหลัก begin ใช้บอกจุดเริ่มต้นของส่วนคำสั่ง และคำหลัก end ใช้บอกจุดจบ ซึ่งในภาษา C/ C++ จะใช้เครื่องหมาย {…} จุดที่ภาษา Pascal แตกต่างจาก ภาษา C/ C++ ก็คือ ในส่วนคำสั่งจะไม่มีการประกาศตัวแปรใดๆ ขึ้นใหม่ ทั้งนี้เพื่อให้โปรแกรมภาษา Pascal มีโครงสร้างที่เป็นระเบียบชัดเจน อ่านง่าย และประโยชน์ที่สำคัญคือ คอมไพเลอร์ภาษา Pascal จะเขียนได้ง่ายขึ้น

การรวมกลุ่มส่วนประกอบต่างๆ ของโปรแกรมให้เป็นโครงสร้างที่เป็นระเบียบ มีความชัดเจน เพื่อให้สามารถอ่านโปรแกรมได้ง่าย มีจุดเริ่มต้นมาตั้งแต่ ทศวรรษที่ 1960 โดยภาษา Algol 60 เป็นภาษาแรกที่กำหนดโครงสร้างที่เรียกว่า <u>บล็อก</u> (block) เพื่อใช้เรียกกลุ่มของคำสั่ง โดยมีคำหลัก begin .. end หรือเครื่องหมายพิเศษ {…} ใช้บอกขอบเขตของบล็อก ตัวอย่างของบล็อกในภาษา C/ C++ และภาษา Ada แสดงได้ดังต่อไปนี้

ภาษา C/ C++	ภาษา Ada

{
int t;
/ swap x and y */*
t=x ; x=y ; y=t ;
}
	`declare`
	`  T:Integer;`
	`begin`
	`  - - swap x and y`
	`  T:=X; X=Y; Y=T;`
	`end;`

ตัวอย่างข้างต้นเป็นบล็อกของคำสั่ง หรือกลุ่มคำสั่งที่ทำหน้าที่สลับค่าของ x และ y เช่น ถ้า x มีค่า 10 และ y มีค่า 20 เมื่อบล็อกนี้ทำงานเสร็จ (จุดที่บอกการเสร็จสิ้นคือ } หรือ end) x จะมีค่า 20 และ y จะมีค่า 10 สังเกตได้ว่าการสลับค่านี้ต้องมีการใช้ตัวแปร t ช่วยเก็บค่าชั่วคราวระหว่างการย้ายค่า ก่อนที่จะนำตัวแปร t ไปใช้ต้องมีการประกาศชื่อ และชนิดข้อมูลของตัวแปร t ไว้ก่อน และมักจะนิยมประกาศไว้ที่ต้นปล็อกเพื่อให้โปรแกรมอ่านง่าย หลักการประกาศตัวแปรก่อนใช้งานนี้เป็นหลักการพื้นฐานที่มักจะใช้ภาษาเชิงคำสั่งในปัจจุบันเพราะจะช่วยให้คอมไพเลอร์ทำงานได้เร็วขึ้น และสามารถตรวจสอบข้อบกพร่องในโปรแกรมได้มากขึ้น

ในกรณีของการเรียกใช้โพรซีเจอร์ หรือฟังก์ชันก็มีข้อกำหนดเช่นเดียวกันว่า โพรซีเจอร์/ฟังก์ชัน นั้นต้องมีการประกาศมาก่อน ตัวอย่างต่อไปนี้แสดงการทำงานของฟังก์ชัน f และ g โดยมีการเรียกใช้ซึ่งกันและกัน การประกาศฟังก์ชันจึงต้องใช้วิธี ประกาศล่วงหน้า หรือ ประกาศ โพรโทไทพ์

ภาษา C/ C++	ภาษา Ada

void f(void);
void g(void);
 	
void f(void)
{ ...
* g( ); /* use g( ) */*
* ...*
}
void g(void)
{ ...
* f( ); /* use f( ) */*
* ...*
}
	`procedure F;`
	`procedure G;`
	
	`procedure F is`
	`begin`
	`  …`
	`  G;  - - use G`
	`  …`
	`end F;`
	`procedure G is`
	`begin`
	`  …`
	`  F;  - - use F`
	`  …`
	`end G;`

การจัดกลุ่มคำสั่งด้วยโครงสร้างของบล็อก สามารถเขียนซ้อนกันได้เรียกว่า nested blocks ดังตัวอย่างต่อไปนี้ที่แสดงการใช้บล็อกซ้อนกันสามชั้น

ภาษา C/ C++	ภาษา Ada

{ int x;
...
{ int y;
...
{ int t;
t=x;
x=y;
y=t;
}
...
}
...
}
	declare
	X:Integer;
	begin
	…
	declare
	Y:Integer;
	begin
	…
	declare
	T:Integer;
	begin
	T:=X;
	X:=Y;
	Y:=T;
	end;
	…
	end;
	…
	end;

เมื่อมีการใช้บล็อกซ้อนกัน โดยบล็อกแต่ละชั้นมีการประกาศชื่อตัวแปร และชื่อตัวแปรที่อยู่ต่างบล็อกกันอาจมีชื่อซ้ำกันได้ (ในตัวอย่างข้างต้นไม่มีการใช้ชื่อซ้ำ) การออกแบบภาษาจึงต้องมีการกำหนดกฎเกณฑ์ที่เกี่ยวข้องกับชื่อตัวแปรและการเรียกใช้ตัวแปรดังนี้

- ขอบเขตของตัวแปร (scope) ตัวแปรจะต้องมีการประกาศก่อนการใช้งาน และขอบเขตของตัวแปรนั้นจะเริ่มตั้งแต่จุดที่ประกาศไปจนถึงจุดสิ้นสุดบล็อก เช่นในตัวอย่างข้างต้น ตัวแปร t มีขอบเขตตั้งแต่บรรทัดที่มีการประกาศ t ไปจนจบบล็อกชั้นในสุด
- การมองเห็น (visibility) หมายถึง การเกิดขึ้นของตัวแปร และการที่ตัวแปรนั้นสามารถถูกเรียกใช้งานได้ เช่น ในตัวอย่างข้างต้น ตัวแปร x เกิดขึ้นในบล็อกชั้นนอกสุด และมีขอบเขตยาวตลอดจนถึงบรรทัดสุดท้ายของโปรแกรมตัวอย่าง ตัวแปร x จึงสามารถถูกมองเห็นและถูกเรียกใช้ได้ในทุกคำสั่งตลอดโปรแกรม ในขณะที่ตัวแปร t ถูกประกาศไว้ในบล็อกชั้นในสุด ขอบเขตจะสิ้นสุดที่เครื่องหมายวงเล็บปิด (}) อันแรก ทำให้การถูกมองเห็นและการถูกเรียกใช้งาน จำกัดอยู่ภายในขอบเขตบล็อกชั้นในสุดเท่านั้น คำสั่งในบล็อกชั้นกลางและชั้นนอกสุดไม่สามารถเรียกใช้ t ได้
- ช่วงชีวิต (lifetime) หมายถึง การมีอยู่ของตัวแปร ระยะเวลาการคงอยู่ของตัวแปรจะสัมพันธ์กับขอบเขตของตัวแปร ถ้าขอบเขตของตัวแปรยังไม่สิ้นสุด ช่วงชีวิตของตัวแปรนั้นก็ยังไม่สิ้นสุด

เช่นเดียวกัน (ซึ่งหมายถึง คอมไพเลอร์ยังต้องจองเนื้อที่หน่วยความจำเพื่อเก็บค่าของตัวแปรนั้นต่อไป)

รูปที่ 7.6 แสดงขอบเขตการมองเห็นตัวแปรในกรณีที่แต่ละบล็อกใช้ชื่อตัวแปรซ้ำกัน โดยแสดงอาณาเขตของแต่ละบล็อกด้วยขอบเส้นประ

```
1   A:  {
2           int i, j;
3           typedef struct {
4                   int i, j;
5           } record_ij;
6           record_ij s;
7           …
8           j=i;
9           j=s.i;
10      B:  {
11                  int i, j;
12                  record_ij t;
13                  …
14                  j=i;
15                  j=s.i;
16                  j=t.i;
17                  …
18          }
19  …
20  }
```

รูปที่ 7.6 ขอบเขตและการมองเห็นของชื่อตัวแปร i

จากโปรแกรมตัวอย่างในรูปที่ 7.6 จะเห็นได้ว่ามีการประกาศตัวแปร i (และ j) อยู่ในทุกบล็อก เมื่อมีการรันโปรแกรมนี้ ตัวแปร i ในบล็อก A มีขอบเขตและช่วงชีวิตตั้งแต่มีการทำคำสั่งบรรทัดที่ 2 ถึงบรรทัดที่ 20 ส่วนตัวแปร i ใน struct record_ij จะมีช่วงชีวิตเกิดขึ้นเมื่อมีการประกาศตัวแปรที่เป็นชนิด record_ij เช่นในบรรทัดที่ 6 ที่มีการประกาศตัวแปร s ทำให้เกิดตัวแปร i ใน record_ij และช่วงชีวิตของ i จะเป็นไปตามช่วงชีวิตของ s

ในบล็อก B มีการประกาศตัวแปร i เช่นเดียวกัน และขอบเขตของ i นี้จะเริ่มจากบรรทัดที่ 12 จนถึงบรรทัดที่ 19 ดังนั้นเมื่อมีการใช้คำสั่ง j=i; ในบรรทัดที่ 14 ตัวแปร i นี้จะเป็น i ในขอบเขตปัจจุบันซึ่งเป็นขอบเขตของบล็อก B

การจัดส่วนประกอบของโปรแกรมให้รวมกลุ่มคำสั่งเป็นบล็อก รวมถึงรวมกลุ่มเป็นโพรซีเจอร์/ฟังก์ชัน มีส่วนช่วยจัดโครงสร้างของโปรแกรมขนาดเล็กให้เป็นระเบียบอ่านง่าย แต่เมื่อโปรแกรมมีขนาดใหญ่มากขึ้น แนวคิดของการจัดเป็นบล็อกและจัดเป็นโปรแกรมย่อยยังไม่เพียงพอ ผู้ออกแบบภาษาคอมพิวเตอร์จึงได้กำหนด

โครงสร้างใหม่เพื่อการจัดระเบียบโปรแกรมขึ้นมาเรียกว่าแพ็คเกจ (package) แนวคิดของแพ็คเกจใช้การแบ่งโปรแกรมขนาดใหญ่ออกเป็นโมดูล (module) ย่อยๆ แล้วสร้างเป็นแพ็คเกจสองแพ็คเกจ แพ็คเกจแรกเก็บเฉพาะส่วนอินเตอร์เฟซของโมดูล แพ็คเกจที่สองเก็บรายละเอียดคำสั่งของโมดูล ภาษา Ada เรียกแพ็คเกจที่เก็บเฉพาะส่วนอินเตอร์เฟซว่า package specification และเรียกแพ็คเกจที่เก็บรายละเอียดว่า package body ตัวอย่างแพ็คเกจในภาษา Ada ที่ใช้คำนวณวันที่ตามปฏิทิน แสดงได้ดังรูปที่ 7.7 และ 7.8

```
package Date is
  type Date_Type is private;
  function Create_Date(Y,M,D:Integer) return Date_Type;
  function Last_of_The_Month(Date:Date_Type) return Boolean;
  private
  type Date_Type is record
          Year,Month,Day : Integer ;
  end record;
end Date;
```

รูปที่ 7.7 แพ็คเกจส่วนอินเตอร์เฟซในภาษา Ada

```
package body Date is
  Normal_Month_Length : constant array(1..12) of Integer :=
                        (31,28,31,30,31,30,31,31,30,31,30,31);
  function Is_Leap_Year(Y:Integer) return Boolean is
  begin
     retrun ((Y mod 4 =0)and (y mod 100/=0)) or (y mod 400=0);
  end Is_Leap_Year;

  function Month_Lenght(Y,M:Integer) return Integer is
  begin
          if M=2 and Is_Leap_Year(Y) then return 29;
          else return Normal_Month_Lenght(M);
          end if;
  end Month_Length;

  function Create_Date(Y,M,D:Integer) return Date_Type is
  begin
          return (Year => Y,Month => M, Day => D);
  end Create_Date;

  function Last_of_The_Month(Date: Date_Type) return Boolean is
  begin
          return Date, Day = Month_Length(Date,Year,Dat.Month);
  end Last_Of_The_Month;
end Date;
```

รูปที่ 7.8 แพ็คเกจส่วนรายละเอียดคำสั่งในภาษา Ada

เมื่อโปรแกรมอื่นต้องการใช้งานแพ็คเกจ Date จะใช้คำสั่ง with Date; ซึ่งจะมีความหมายเช่นเดียวกับการใช้คำสั่ง #include “date.h” ในภาษา C รูปที่ 7.9 แสดงตัวอย่างการเรียกใช้แพ็คเกจ Date

```
with Text_IO;          -- import package Text_IO
with Date;             -- import package Date
use Date;              -- make the names in it visible
procedure Use_Date is
  Date : Date_Type;
begin
  Date ;= Create_Date(1992,2,29); -- Feb.29,1992
  if Last_of_the_Month(Date) then
          Text_IO.Put(“plan day!”);
  end if;
end Use_Date;
```

รูปที่ 7.9 ตัวอย่างการใช้แพ็คเกจ Date ในภาษา Ada

แนวคิดการแบ่งโปรแกรมใหญ่ออกเป็นโมดูลย่อยๆ และแยกส่วนอินเตอร์เฟซออกจากส่วนรายละเอียดถูกนำมาใช้ในภาษา C เช่นเดียวกันโดยส่วนอินเตอร์เฟซจะถูกแยกเป็น header file รูปที่ 7.10 แสดงคำสั่งใน header file ที่มีโครงสร้างอินเตอร์เฟซเดียวกับแพ็คเกจในภาษา Ada และรูปที่ 7.11 แสดงรายละเอียดในฟังก์ชันต่าง ๆ โดยการเรียกใช้แพ็คเกจ date ในรูปแบบภาษา c แสดงได้ดังรูปที่ 7.11

```
typedef struc {

        int dt_year, dt_month,dt_year;
}date_type;
extern date_type Creat_date(int,int,int);
extern int last_of_month(date_type);
```

รูปที่ 7.10 เฮดเดอร์ไฟล์ date.h เพื่อคำนวณวันตามปฏิทิน

```
#include “date.h”
static int normal_month_length [12] = {31,28,31,30,31,30,31,31,30,31,30,31};
static int is_leap_year(int y){
        return ((y%4 = = 0) && (y%100 != 0)) || (y%400 = = 0) ;
}
static int month_length(int y,int m){
        if (m = = 2 && is_leap_year(y)) return 29;
        else return nomal_month_length[m-1];
}
date_type create_date (int y, int m, int d){
        date_type dt;
        dt.dt_year = y, dt.dt_manth = m, dt.dt_day = d;
        return dt;
}
int last_of_the_month(date_type date){
        return date.dt_day == month_length(date.dt_year,date.dt_month);
}
```

รูปที่ 7.11 ไฟล์ date.c เพื่อคำนวณวันตามปฏิทิน

```
#include <stdio.h>
#include “date.h”
use_date(void){
        date_type date;
        date = Create_date(1992,2,29);  /*Feb.29,1992*/
        if(last_of_month(date)){
            printf(“Play day!”);
        }
}
```

รูปที่ 7.12 วิธีการใช้แพ็คเกจ date ในรูปแบบภาษา C

7.5 การทำโปรแกรมเชิงคำสั่งในภาษาซี, ปาสคาล, เอดา และฟอร์แทรน

(Imperative programming in C, Pascal, Ada and FORTRAN languages)

การเปรียบเทียบรูปแบบการทำโปรแกรมในภาษาเชิงคำสั่งต่างๆ เช่นภาษา C ,Pascal, Ada, FORTRAN จะใช้ตัวอย่างโปรแกรมการบวกค่าจำนวนเลขในอาร์เรย์ โดยผู้ใช้สามารถกำหนดสมาชิกในอาร์เรย์ให้มีจำนวนตามต้องการแต่สูงสุดไม่เกิน 99 เมื่อโปรแกรมทำงานเสร็จจะแสดงผลลัพธ์ที่เป็นผลบวกของจำนวนเลขทั้งหมดในอาร์เรย์ด้วยข้อความ *SUM = ……*

<u>ตัวอย่างโปรแกรมในภาษา C</u>

```
#include <stdio.h>
const int maxsize = 99;
main()
{       int a[maxsize];
        int j,k;
        while((k=convert(getchar())) != 0){
            for(j=0; j<k;j++) a[j] = convert(getchar());
            for(j=0; j<k;j++) printf("%d",a[j]);
            printf("; SUM= %d\n",addition(a,k));
                    while(getchar() != '\n');
        }       }
        /*Function convert subprogram*/
        int convert(char ch)
                {return ch-'0';}
        /*Function addition subprogram*/
        int addition(v,n)
                int v[] ,n;
                {int sum,j;
            sum =0;
                for(j=0; j<n; j++) sum = sum+v[j];
                return sum;
                }
```

<u>ตัวอย่างโปรแกรมในภาษา Pascal</u>

```
program main(intput,output,infile);
const size = 99 ;
type Vector = array [1..size] of integer;
var infile: text;
        a: Vector;
        j,k: integer;
function sum(v:Vector;n:integer): real;
        var temp: real;
        i: integer;
                {Body of function sum}
                begin
                temp := 0;
                for i:= 1 to n do temp := temp + v[i];
                sum := temp
                end; {sum}
begin {of main}
                reset (infile,'sample.data');
                while not (eof(infile)) do
                    begin
                      read(infile,k);
                      for j := 1 to k do
                          begin
                             read(infile, a[j]);
                             write(a[j]:10:2)
                             end ;
                        writeln;
                        writeln('sum = ',sum(a,k):6:4);
                        readln(infile)
                        end
end.
```

<u>ตัวอย่างโปรแกรมในภาษา Ada</u>

```
1 package ArrayCalc is
2         type Mydata is private;
3         function sum return integer;
4         procedure setval (arg:in integer);
5         private
6                 size: constant := 99;
7                 type myarray is array (1..size) of integer;
8                 type Mydata is record
9                         val: myarray;
10                                sz: integer := 0;
11                                end record;
12                        v: Mydata;
13        end;
14        package body ArrayCalc is
15                function sum return integer is
16                        temp: integer;
17                - - Body of function sum
18                        begin
19                        temp := 0;
20                        for i in 1.. v.sz loop
21                                temp:= temp+v.val(i);
22                                end loop;
23                        v.sz :=0;
24                        return temp;
25                        end sum;
26         procedure setval (arg:in integer) is
27                begin
28                        v.sz:= v.sz+1;
29                        v.val(v.sz):= arg;
30                        end setval ;  end ;
31        with Text_IO ; use Text_IO ;
 32       with ArrayCalc ; use ArrayCalc ;
33        procedure main is
34                k, m: integer;
35        begin - - of main
36                get(k);
37                while k>0 loop
38                        for j in 1..k loop
39                                get(m); put(m,3);
40                                setval(m);
41                                end loop;
42                        new_line; put("SUM =");
43                        put (ArrayCalc.sum,4);
44                        new_line; get(k);
45                        end loop;
46        end;
```

<u>ตัวอย่างโปรแกรมในภาษา FORTRAN</u>

```
1                PROGRAM MAIN
2                        PARAMETER (MAXSIZ = 99)
3                        REAL A(MAXSIZ)
4 10     READ (5, 100, END = 999)  K
5 100    FORMAT (I5)
6                        IF (K.LE.O.OR. K.GT.MAXSIZ) STOP
7                        READ *, (A(I), I=1,K)
8                        PRINT *, (A(I), I=1,K)
9                        PRINT *, 'SUM=', SUM(A,K)
10                               GO TO 10
11       999     PRINT *, "All Done"
12                       STOP
13                       END
14       C SUMMATION SUBPROGRAM
15                       FUNCTION SUM(V,N)
16                               REAL :: V(N) ! New style declaration
17                               SUM = 0.0
18                               DO 20 I=1,N
19                                       SUM=SUM+V(I)
20       20                      CONTINUE
21                               RETURN
22                               END
```

7.6 สรุป

วิธีการทำโปรแกรมเชิงคำสั่งเป็นการทำงานกับค่าข้อมูลที่เก็บอยู่ในหน่วยความจำคอมพิวเตอร์ คำสั่งในโปรแกรมจะเป็นคำสั่งที่ใช้ควบคุมการเปลี่ยนค่าของข้อมูล ดังนั้นจึงอาจกล่าวได้ว่าวิธีการทำโปรแกรมเชิงคำสั่งเป็นการทำโปรแกรมที่ยึดข้อมูลเป็นหลัก (data_driven) และลำดับการทำโปรแกรมจะเป็นลำดับที่ต่อเนื่องตั้งแต่ต้นโปรแกรมจนถึงจุดสุดท้ายของโปรแกรม โดยการรันโปรแกรมแต่ละครั้งอาจจะมีบางคำสั่งถูกข้ามการทำงาน แต่บางคำสั่งอาจจะถูกกระทำซ้ำหลายครั้งทั้งนี้ขึ้นอยู่กับค่าของข้อมูลที่ได้รับมาขณะรันโปรแกรม

การทำโปรแกรมเชิงคำสั่งที่ขนาดของโปรแกรมไม่ใหญ่มากนัก โปรแกรมเมอร์จะใช้วิธีจัดกลุ่มคำสั่งเป็นบล็อกและแยกชุดคำสั่งออกเป็นรูทีนด้วยการใช้โพรซีเจอร์หรือฟังก์ชัน ทั้งนี้เพื่อให้โปรแกรมมีโครงสร้างที่เป็นระเบียบอ่านง่าย (เรียกว่าการทำโปรแกรมอย่างเป็นโครงสร้าง หรือ structured programming) แต่ถ้าโปรแกรมมีขนาดใหญ่มากการแยกชุดคำสั่งออกเป็นรูทีนเพียงอย่างเดียวยังไม่เพียงพอ โปรแกรมเมอร์จะใช้วิธีแยกโมดูล โดยแยกส่วนระบุชนิดข้อมูลและส่วนฟังก์ชันโพรโทไทพ์ออกเป็นหนึ่งโมดูล (ภาษา Ada เรียกแพ็คเกจ) และแยกรายละเอียดคำสั่งในแต่ละฟังก์ชันออกเป็นอีกหนึ่งโมดูล (ภาษา Ada เรียกแพ็คเกจบอดี้) วิธีการทำโปรแกรมแบบนี้เรียกว่า modular programming เป็นการช่วยจัดโครงสร้างโปรแกรมขนาดใหญ่ให้อ่านง่ายขึ้น และช่วยให้การบำรุงรักษาโปรแกรมทำได้สะดวกขึ้น

แบบฝึกหัดท้ายบทที่ 7

คำถามปรนัย: ให้เลือกคำตอบที่ถูกต้องที่สุด

พิจารณาส่วนของโปรแกรมภาษา C ต่อไปนี้ แล้วตอบคำถามข้อ 1-10

```
/* 1 */   int a = 5, b = 1;
/* 2 */   int g( ) {
/* 3 */      int c = a + b;
/* 4 */      return c;
/* 5 */    }
/* 6 */   void f( ) {
/* 7 */      int d;
/* 8 */      if ( b == 1) {
/* 9 */                    b = 2;
/* 10 */                    f( );
/* 11 */        } else {
/* 12 */                d = g( );
/* 13 */              }
/* 14 */     }
/* 15 */   main( ) {
/* 16 */        f( );
/* 17 */   }
```

1. ขอบเขต (scope) ของตัวแปร c อยู่ในช่วงใด ?

   ก. ตลอดทั้งโปรแกรม
   ข. ระหว่างบรรทัดที่ 2 ถึงบรรทัดที่ 5
   ค. ระหว่างบรรทัดที่ 3 ถึงบรรทัดที่ 14
   ง. ระหว่างบรรทัดที่ 3 ถึงบรรทัดที่ 17

2. ขอบเขต (scope) ของตัวแปร b อยู่ในช่วงใด ?

   ก. ตลอดทั้งโปรแกรม
   ข. ระหว่างบรรทัดที่ 3 ถึงบรรทัดที่ 9
   ค. ระหว่างบรรทัดที่ 3 ถึงบรรทัดที่ 14
   ง. ระหว่างบรรทัดที่ 3 ถึงบรรทัดที่ 17

3. ช่วงอายุ (lifetime) ของตัวแปร d เริ่มต้นและสิ้นสุดเมื่อใด ?

   ก. เริ่มต้นบรรทัดที่ 6 สิ้นสุดบรรทัดที่ 12
   ข. เริ่มต้นบรรทัดที่ 7 สิ้นสุดบรรทัดที่ 12
   ค. เริ่มต้นเมื่อ f() ถูกเรียกใช้ สิ้นสุดเมื่อ f() จบการทำงาน
   ง. เริ่มต้นเมื่อ f() ถูกเรียกใช้ สิ้นสุดเมื่อ f() เรียกใช้ g()

4. ค่าของตัวแปร c จะเกิดขึ้นเมื่อใด ?

   ก. เมื่อคอมไพเลอร์แปลคำสั่งไปถึงบรรทัดที่ 3
   ข. เมื่อคอมไพเลอร์ run คำสั่งไปถึงบรรทัดที่ 3
   ค. เมื่อฟังก์ชัน g() ถูกเรียกใช้งาน
   ง. เมื่อฟังก์ชัน main() เริ่มทำงาน

5. ระหว่างบรรทัดที่ 13 และ 14 ถ้าแทรกคำสั่ง printf(...) เราสามารถพิมพ์ค่าของตัวแปรใดได้บ้าง ?

ก. a, b, c
ข. a, b, d
ค. b, d
ง. a, b, c, d

6. ตัวแปรใดเรียกว่า ตัวแปร local ?

ก. a, b
ข. c, d
ค. f, g
ง. a, b, c, d

7. ตัวแปรใดเรียกว่า ตัวแปร global ?

ก. a, b
ข. c, d
ค. f, g
ง. a, b, c, d

8. เมื่อโปรแกรมทำงานถึงบรรทัดที่ 13 r-value ของ d มีค่าเป็นอะไร ?

ก. 0
ข. 6
ค. 7
ง. ไม่ทราบค่า

9. ตัวแปร d ถูกจัดสรรเนื้อที่อยู่ในส่วนใดของหน่วยความจำ ?

ก. static memory
ข. stack frame ของฟังก์ชัน main()
ค. stack frame ของฟังก์ชัน f()
ง. stack frame ของฟังก์ชัน g()

10. ตัวแปร a ถูกจัดสรรเนื้อที่อยู่ในส่วนใดของหน่วยความจำ ?

ก. static memory
ข. stack frame ของฟังก์ชัน main()
ค. stack frame ของฟังก์ชัน f()
ง. stack frame ของฟังก์ชัน g()

11. พิจารณาส่วนของโปรแกรมภาษา C ต่อไปนี้

```
int * ptr;
int count, init = 1;
ptr = &init;
count = *ptr;
```

ส่วนของโปรแกรมนี้มีผลการทำงานเทียบเท่ากับชุดคำสั่งในข้อใด ?

ก.
```
int count;
int init = 1;
count = init;
```

ข.
```
int count;
int init = 1;
int * ptr = &init;
count = *ptr;
```

ค.
```
int count,init;
count = init = 1;
```

ง. ถูกทุกข้อ

12. การประกาศใช้ตัวแปร ptr ในข้อ 11 ทำให้เกิดเหตุการณ์ใด ?

ก. alias
ข. dangling pointer
ค. garbage
ง. semantic error

13. พิจารณาว่ามีสิ่งผิดปกติใดกับส่วนของโปรแกรมภาษา C ต่อไปนี้หรือไม่ ?

```
int x, y;
int array [10][3];
int main()
   {  for (x =0; x < 3; x++)
         for (y= 0; y < 10; y++)
               array[x][y] =0;
      return 0;
      }
```

ก. ไม่มีอะไรผิด โปรแกรมนี้กำหนดค่า 0 ให้กับ array ทุกช่อง

ข. ผิด ฟังก์ชัน main() ไม่ต้องมี int และไม่ต้องมีคำสั่ง return 0

ค. ผิด ต้องแก้ loop for ชั้นนอกให้ x มีค่า 0 ถึง 9

ง. ผิด ต้องแก้ loop for ชั้นในให้ y มีค่า 0 ถึง 2

14. การเขียนนิพจน์ (3+7) * (2+1) ให้อยู่ในรูปแบบ 3 7 + 2 1 + *
เรียกว่ารูปแบบใด ?

ก. postfix
ข. prefix
ค. infix
ง. polish

15. พิจารณาคำสั่งต่อไปนี้

```
d = a;
a = b + c;
```

คำกล่าวในข้อใดผิด ?

ก. a ในบรรทัดแรกเรียกว่า l-value
ข. a ในบรรทัดที่สองเรียกว่า l-value
ค. บรรทัดแรกระบุ value ของ a
ง. บรรทัดที่สองระบุ address ของ a

16. พิจารณาส่วนของโปรแกรมภาษา C ต่อไปนี้

```
int A[ ] = {1, 2, 3, 4, 5 } ;
int i = 5;
A[--i] = i++ ;
```

คำกล่าวในข้อใดผิด ?

ก. โปรแกรม error เนื่องจากระบุ index เกินขอบเขตของ array A

ข. เมื่อทำทั้งสามคำสั่งเสร็จ A มีค่า { 1, 2, 3, 4, 4 }

ค. เมื่อทำทั้งสามคำสั่งเสร็จ i มีค่า 5

ง. คำสั่งในบรรทัดที่สาม เป็นการกำหนดค่าให้ A[4]

17. พิจารณาส่วนของโปรแกรมภาษา C ต่อไปนี้

```
 int a = 5;
int fun1() {
        a = 17;
        return 3;
    }
void fun2() {
        a = a + fun1() ;
    }
void main() {
         fun2() ;
         printf(" a =  %d ", a );
    }
```

ผลลัพธ์ที่ได้จากการ run โปรแกรมนี้คืออะไร ?

ก. โปรแกรม error

ข. a = 3

ค. a = 8

ง. a = 20

18. ลักษณะที่ปรากฏในข้อ 17 เรียกว่าอะไร ?

ก. runtime error

ข. side effect

ค. associative

ง. precedence

19. พิจารณาส่วนของโปรแกรมภาษา C ต่อไปนี้

```
int fun(int * i) {
        *i += 5;
        return 4;
    }
void main() {
         int x = 3 ;
         x = x + fun( &x );
         printf(" x =  %d ", x );
    }
```

ผลลัพธ์ที่ได้จากการ run โปรแกรมนี้คืออะไร ?

ก. โปรแกรม error

ข. x = 3

ค. x = 7

ง. x = 12