บทที่ 11

โครงสร้างควบคุม

11.1 โครงสร้างควบคุม

11.2 โปรแกรมย่อย

11.3 เอกสารอ้างอิงและเว็บไซต์ที่ควรรู้

วัตถุประสงค์

- โครงสร้างควบคุมระดับคำสั่งรูปแบบต่างๆ
- โครงสร้างควบคุมระดับกลุ่มคำสั่ง
- แนวคิดพื้นฐานของโปรแกรมย่อย
- กลไกการส่งผ่านพารามิเตอร์แบบต่างๆ

โปรแกรมในแต่ละภาษาจะต้องระบุลำดับในการทำงาน หรือทิศทางการควบคุม (Flow of control) ที่ชัดเจน ซึ่งทิศทางในการควบคุมนี้สามารถแบ่งออกได้หลายระดับ โปรแกรมในภาษาเชิงคำสั่ง โดยมากจะเป็นการหาค่านิพจน์ กำหนด ค่าหรือผลลัพธ์ให้กับตัวแปร นอกจากนี้ยังมีอีกสองกลไกที่จำเป็นที่ทำให้โปรแกรมยืดหยุ่นและมีประสิทธิภาพ นั่นคือ การที่โปรแกรมสามารถเลือกทำคำสั่งหรือเลือกทิศทางในการควบคุมได้ และการที่โปรแกรมสามารถทำกลุ่มคำสั่งหนึ่งซ้ำๆ กันได้ คำสั่งที่ทำให้เกิดกลไกนี้เราเรียกว่า คำสั่งควบคุม (Control statement) ในบทนี้จะกล่าวถึงโครงสร้างควบคุม (Control structure) ซึ่งเป็นสิ่งสำคัญสำหรับภาษาเชิงคำสั่ง

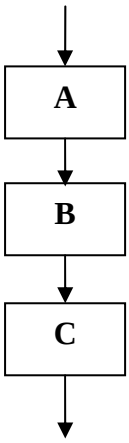
โครงสร้างควบคุม คือ คำสั่งควบคุม และกลุ่มคำสั่งที่ควบคุมให้ทำงานตามลำดับที่ต้องการ โดยทั่วไปภาษาโปรแกรมชั้นสูงแบ่งการควบคุมเป็น 2 ระดับใหญ่ๆ คือ การควบคุมระดับคำสั่ง และการควบคุมระดับกลุ่มคำสั่ง (หรือโปรแกรมย่อย)

11.1 โครงสร้างควบคุมระดับคำสั่ง (Statement level control structure)

โครงสร้างควบคุมระดับคำสั่งนี้ จะใช้จัดลำดับการทำงานของคำสั่งในโปรแกรม โดยทั่วไปจะมี 3 รูปแบบหลัก ดังต่อไปนี้

โครงสร้างแบบเรียงลำดับ (Sequencing structure)

เป็นโครงสร้างควบคุมรูปแบบที่ง่ายที่สุด โดยเป็นการกำหนดลำดับก่อนหลังของการทำงาน โดยถ้าคำสั่ง A อยู่ก่อนคำสั่ง B คำสั่ง A จะต้องถูกทำงานก่อน แต่ละภาษาโปรแกรมจะมีสัญลักษณ์ที่ใช้กำหนดหรือคั่นประโยคคำสั่ง เช่น semicolon (;) ใน pascal , จุด (.) ใน COBOL ซึ่งเขียนเป็นผังงาน (flowchart) ได้ดังนี้



ในบางภาษายอมให้มีการรวมหลาย ๆ คำสั่งเข้าเป็นกลุ่มของคำสั่ง ทำให้กลุ่มของคำสั่งนั้นเป็นเสมือน 1 คำสั่ง เรียกว่า Compound statement โดยกำหนดสัญลักษณ์เพื่อระบุจุดเริ่มต้นและจุดสิ้นสุดของกลุ่มคำสั่งที่ต้องการ ตัวอย่างเช่น เครื่องหมาย { } ในภาษา C ดังโปรแกรมต่อไปนี้

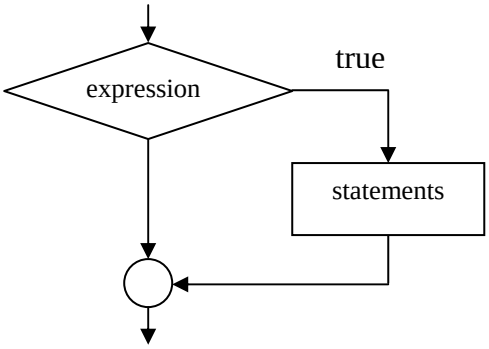
```
if (a>b)
{
      a++;
      printf("%d",a);
}
```

โครงสร้างแบบมีเงื่อนไข (Selection structure)

เป็นโครงสร้างควบคุมที่มีทางเลือกในการทำงานสองทางขึ้นไป โดยจะต้องตรวจสอบเงื่อนไขก่อน แล้วใช้ผลที่ได้จากการตรวจสอบ เป็นตัวกำหนดว่าต้องทำทางเลือกหรือคำสั่งใดต่อไป โครงสร้างแบบมีเงื่อนไขแบ่งย่อยออกเป็น 3 รูปแบบ คือ

- แบบง่าย (Simple condition) มีเงื่อนไขที่ต้องตรวจสอบเพียงเงื่อนไขเดียว โดยตรวจสอบว่าต้องทำงานตามคำสั่งที่กำหนดหรือไม่ มีผังงานและรูปแบบดังต่อไปนี้

```
if <boolean expression>
then <statements>
```

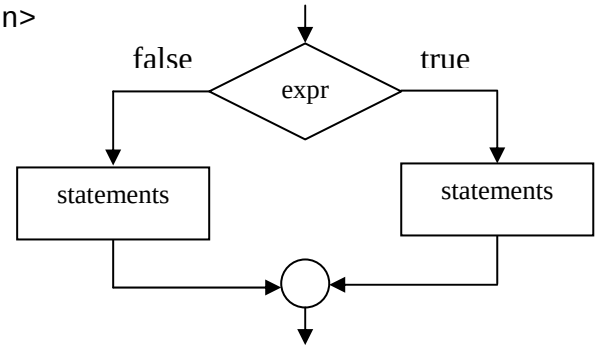


ตัวอย่างในภาษา C

```
if (i>0)
{
   sum = sum + i;
   i++;
}
```

- แบบสองทางเลือก (Two-alternative condition) มีเงื่อนไขที่ต้องตรวจสอบเพียงเงื่อนไขเดียว แต่มีทางเลือกในการทำงาน 2 ทาง โดยตรวจสอบเงื่อนไขว่าเป็นจริงหรือเท็จ มีผังงานและรูปแบบดังต่อไปนี้

```
if <boolean expression>
then <statements>
else <statements>
```



ตัวอย่างในภาษา C

```
if (i > 0)
   i++;
else
   i--;
```

การเขียนคำสั่งเงื่อนไขแบบสองเลือกเมื่อนำมาใช้ซ้อนกัน (Nested if) ก็อาจทำให้เกิดปัญหาเรื่องความกำกวมได้ เพราะอาจเกิดความสับสนว่าเป็นทางเลือกของตัวใด ดังตัวอย่างต่อไปนี้

```
if x>0 then if x<10 then x:=0 else x:= 1000
```

จากตัวอย่างโปรแกรมข้างต้นอาจตีความได้ 2 แบบ คือ

```
if x>0 then
   if x<10 then x:=0
   else x:= 1000
```

```
if x>0 then
   if x<10 then x:=0
else x:= 1000
```

ซึ่งการแก้ปัญหาอาจทำโดยการใช้คำสำคัญหรือคำเฉพาะเป็นคู่ เพื่อระบุจุดเริ่มต้นและจุดสิ้นสุดของกลุ่มของคำสั่งที่ต้องการ เช่น การใช้ begin ... end ในภาษา Algol60

```
if x>0 then
begin
   if x<10 then x:=0
   else x:= 1000
end
```

หรือใช้การกำหนดให้ else เป็นของ if ตัวใกล้สุด เช่น ในภาษา Pascal

```
if x>0 then
   if x<10 then x:=0
   else x:= 1000
```

หรืออาจใช้คำเฉพาะ เพื่อกำหนดขอบเขตของ if ที่ต้องการ เช่น fi ในภาษา Algol68

```
if x>0 then
   if x<10 then x:=0
   else x:= 1000
   fi
fi
```

- แบบหลายทางเลือก (Multi-alternative condition) โครงสร้างแบบ Nested if ซึ่งในบางครั้งทำให้โปรแกรมยาวและอ่านยาก ในบางภาษาจึงพัฒนาให้มีคำสั่งแบบมีทางเลือกหลายทางขึ้นมา เช่น select ในภาษา PL/I หรือ case ในภาษา Pascal ดังตัวอย่าง nested if ต่อไปนี้

```
if <expression1> then
   <statement1>
else if <expression2> then
         <statement2>
     else if <expression3> then
              <statement3>
```

สามารถเขียนให้อยู่ในรูปของคำสั่ง case ในภาษา Pascal ได้ดังนี้

```
case expression of
   value1 : statement1;
   value2 : statement2;
   value3 : statement3;
end;
```

โครงสร้างแบบวนซ้ำ (Iteration structure)

เป็นโครงสร้างควบคุมให้ทำงานที่ต้องการซ้ำกันหลาย ๆ ครั้ง ทำให้เขียนโปรแกรมได้ง่ายขึ้น และโปรแกรมมีขนาดสั้นลง รูปแบบโครงสร้างการทำซ้ำ แบ่งได้เป็น 4 รูปแบบดังนี้

- แบบไม่หยุด (Nonterminating iteration) เป็นรูปแบบที่ง่ายที่สุดของโครงสร้างการทำซ้ำ คือ การทำซ้ำแบบไม่มีจุดสิ้นสุด (indefinite repetition) ตัวอย่างการนำไปใช้งาน เช่น โปรแกรมด้านการสื่อสารที่จำเป็นต้องมีการตรวจสอบตลอดเวลาว่ามีการส่งข้อมูลเข้ามาหรือไม่ ผู้ฝึกเขียนโปรแกรมควรหลีกเลี่ยงโครงสร้างแบบนี้ เพราะจะมีผลให้โปรแกรมทำงานตลอดเวลาไม่หยุด
- แบบตรวจสอบเงื่อนไขก่อนวนซ้ำ (Pretest iteration) โครงสร้างนี้จะมีการกำหนดให้มีการตรวจสอบเงื่อนไขก่อนการทำซ้ำ (loop) โดยมีรูปแบบดังนี้

```
while (control expression)
   loop body
```

ซึ่งภาษาโดยส่วนใหญ่ เช่น ภาษา Pascal หรือ C# ถ้าผลการตรวจสอบเงื่อนไขเป็นจริงจะทำงานตามคำสั่งที่กำหนด ถ้าเป็นเท็จจะหยุดทำงานนั้น จำนวนครั้งของการทำซ้ำขึ้นกับเงื่อนที่กำหนด ตัวอย่างเช่น คำสั่ง while ใน Pascal ซึ่งมีรูปแบบดังนี้

```
while <condition> do
begin
   statement;
   statement;
end;
```

- แบบตรวจสอบเงื่อนไขหลังวนซ้ำ (Posttest iteration) โครงสร้างนี้จะมีการทำซ้ำก่อนการกำหนดให้มีการตรวจสอบเงื่อนไข ถ้าผลการตรวจสอบเงื่อนไขเป็นตามที่กำหนดจะหยุดทำงาน โครงสร้างนี้จะต่างจากแบบตรวจสอบเงื่อนไขก่อนวนซ้ำตรงที่ จะทำงานตามคำสั่งใน loop อย่างน้อย 1 ครั้งเสมอ โดยมีรูปแบบดังนี้

```
do
   loop body
while (control expression)
```

ส่วนการตรวจสอบเงื่อนไขเพื่อให้หยุดทำงานอาจจะใช้การตรวจสอบว่าเป็นจริงหรือเท็จก็ขึ้นอยู่กับภาษาโปรแกรมแต่ละภาษา ตัวอย่างเช่น คำสั่ง do ในภาษา C, C++ และ Java นั้น จะทำคำสั่งที่อยู่ใน loop จนกระทั่งเงื่อนไขเป็นเท็จ ส่วนคำสั่ง repeat ในภาษา Pascal ซึ่งมีรูปแบบดังแสดงด้านล่างนี้ ถ้าเงื่อนไขเป็นเท็จ จะทำงานตามคำสั่งที่อยู่ใน loop จนกระทั่งเงื่อนไขเป็นจริงจึงจะหยุดทำงาน

```
repeat
   statement;
   statement;
until  <condition>;
```

- แบบระบุจำนวนรอบการวนซ้ำ (Fixed count iteration) โครงสร้างนี้จะมีการทำซ้ำ (loop) โดยกำหนดจำนวนครั้งที่แน่นอนของการทำซ้ำ จะใช้ Iterator Control Variable (ICV) เป็นตัวแปรสำหรับกำหนดหรือนับจำนวนครั้งของการทำซ้ำ ตัวอย่างคำสั่ง for ในภาษา Pascal มีรูปแบบดังนี้

```
for <icv> := <initial> to<final>step<increment>do
begin
   statement;
   statement;
end;
```

โครงสร้างแบบไม่มีเงื่อนไข (Unconstrained Control Statement)

โครงสร้างควบคุมแบบนี้ ยอมให้ผู้เขียนโปรแกรมเขียนคำสั่งเพื่อให้คอมพิวเตอร์ข้ามไปทำงานในส่วนของโปรแกรมที่ต้องการได้ โดยไม่ต้องกำหนดเงื่อนไขใดๆ ข้อเสีย คือ ทำให้โปรแกรมอ่านยาก และต้องระวังเรื่องขอบเขตของการกระโดดข้ามไปทำงานของคำสั่ง จึงไม่นิยมใช้ ตัวอย่างเช่น คำสั่ง goto ใน Pascal

11.2 โปรแกรมย่อย (Subprogram)

ภาษาโปรแกรมชั้นสูงโดยมาก จะกำหนดให้มีโครงสร้างควบคุมระดับกลุ่มคำสั่ง (Unit level control structure) ซึ่งเป็นการรวบรวมกลุ่มคำสั่งที่ต้องการใช้ซ้ำกันหลายครั้งเป็นโปรแกรมย่อย ทำให้ช่วยในการนำกลับมาใช้ใหม่ได้สะดวก และประหยัดทั้งพื้นที่ในหน่วยความจำและเวลาที่ใช้ในการเขียนโปรแกรม โปรแกรมย่อยนี้อาจมีชื่อเรียกแตกต่างกันไปในแต่ละภาษาเช่น subprogram, procedure, function, sub เป็นต้น

ลักษณะพื้นฐานของโปรแกรมย่อย (Fundamental of subprogram)

ลักษณะที่สำคัญโดยทั่วไปของโปรแกรมย่อย มีดังต่อไปนี้

- แต่ละโปรแกรมย่อยจะต้องมีจุดเริ่มต้นเพียงจุดเดียว (single entry point)
- โปรแกรมที่เรียกใช้โปรแกรมย่อยจะหยุดการทำงาน ในระหว่างที่โปรแกรมย่อยที่ถูกเรียกใช้กำลังทำงาน นั่นคือ จะมีเพียงโปรแกรมย่อยเดียวที่ทำงาน ณ ขณะเวลาหนึ่ง
- การควบคุมจะส่งกลับไปยังโปรแกรมที่เรียกมาทุกครั้ง เมื่อโปรแกรมย่อยสิ้นสุดการทำงาน

โครงสร้างของโปรแกรมย่อยประกอบด้วยส่วนหัว ส่วนประกาศ และส่วนคำสั่ง เช่นเดียวกับโปรแกรม แต่ละโปรแกรมย่อยเป็นอิสระจากกัน ประเภทของตัวแปรแบ่งเป็น

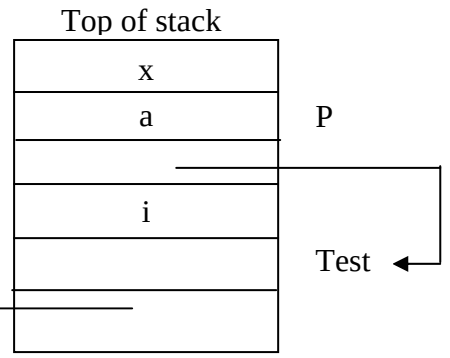
- Global variable เป็นตัวแปรที่กำหนดไว้ในส่วนประกาศของโปรแกรมหลัก และนำไปใช้ได้ทั้งในโปรแกรมหลัก และโปรแกรมย่อย
- Local variable เป็นตัวแปรที่กำหนดไว้ในส่วนประกาศของโปรแกรมย่อย จะใช้ตัวแปรนี้ได้เฉพาะในโปรแกรมย่อยหรือโปรแกรมย่อยภายใน

เมื่อโปรแกรมย่อยถูกเรียกใช้ จะมีผลทำให้คอมพิวเตอร์ทำงานตามคำสั่งที่อยู่ภายในโปรแกรมย่อยนั้น จึงต้องมีการเก็บรวบรวมข้อมูลหรือรายละเอียดตัวแปร ค่าคงที่ พารามิเตอร์ และค่าอื่นๆ ของโปรแกรมย่อยอย่างเป็นระบบข้อมูลเหล่านี้จะเก็บไว้ใน Activation Record (AR) ซึ่งประกอบด้วย

- Local Environment ข้อมูลแบบ local ทั้งหมดของ procedure
- Parameter Environment ข้อมูลของพารามิเตอร์ทั้งหมด
- Global Environment Pointer ชี้ไปยัง AR ของ procedure ที่เรียกใช้

เมื่อถูกเรียกใช้ โปรแกรมย่อยจะสร้าง AR ของโปรแกรมย่อยนั้นขึ้น AR ใหม่นี้จะถูก push ลงใน Runtime stack เมื่อโปรแกรมย่อยทำงานเสร็จ AR นี้จึงจะถูก pop ออกจาก stack ดังแสดงในตัวอย่างโปรแกรมในภาษา Pascal ต่อไปนี้

```
Program Test;
var i : integer;
Procedure P(a : integer);
var x:integer;
begin
     …
end;
Begin
     P(i);
End.
```



กลไกในการส่งผ่านค่าพารามิเตอร์ (Parameter passing mechanism)

การติดต่อเพื่อส่งผ่านค่าระหว่างโปรแกรมหลักและโปรแกรมย่อย หรือระหว่างโปรแกรมย่อยเอง ทำได้โดยอาศัยพารามิเตอร์ (Parameter) การส่งผ่านค่าแบ่งออกเป็น 3 กลุ่ม

- IN Parameter เป็นการส่งผ่านค่าเข้าไปยัง procedure ที่ถูกเรียก
- OUT Parameter เป็นการส่งผ่านค่ากลับมายัง procedure ที่ทำการเรียก
- INOUT Parameter เป็นการส่งผ่านค่าไปและกลับ ระหว่างโปรแกรมย่อยที่เรียกใช้ และโปรแกรมย่อยที่ถูกเรียก

การระบุตัวแปรหรือพารามิเตอร์ที่จะใช้ส่งผ่านค่า ต้องระบุ 2 ที่ในโปรแกรม คือ

- ในโปรแกรมย่อยที่เรียกใช้ โปรแกรมย่อยอื่น เรียกว่า Actual parameter
- ในโปรแกรมย่อยที่ถูกเรียกใช้ เรียกว่า Formal parameter

```
Program Test;
var x,y : integer;
Procedure Square(a,b:integer);
begin
      …
end;
Begin
      …
      Square(x,y);
      Square(2,3);
End.
```

จากตัวอย่าง โปรแกรมย่อย square มีการประกาศ formal parameter ไว้ในส่วนหัวของโปรแกรมย่อยชื่อ a และ b ตามลำดับ เมื่อเรียกใช้โปรแกรมย่อย ครั้งแรกโปรแกรมหลักทำการส่งตัวแปร x และ y เป็น actual parameter ให้กับ formal parameter ในโปรแกรมย่อยคือ a และ b ตามลำดับ ส่วนครั้งที่สองโปรแกรมหลักจะส่งค่า 2 และ 3 เป็น actual parameter ไปให้กับโปรแกรมย่อยตามลำดับอีกเช่นกัน

การ binding ระหว่าง actual parameter กับ formal parameter เพื่อส่งผ่านข้อมูล ชนิดข้อมูล ฯลฯ ระหว่างโปรแกรมย่อยที่ทำการติดต่อกันทำได้ 2 วิธี คือ

- Position method เป็นการ bind ตามตำแหน่งของพารามิเตอร์
- Name method เป็นการ bind ตามชื่อของพารามิเตอร์

เทคนิคที่ใช้การส่งผ่านค่าระหว่าง formal และ actual parameter แบ่งได้ดังต่อไปนี้

- Call by copy ในส่วน AR ต้องมีเนื้อที่สำรองสำหรับ Local variable และ formal parameter ด้วย การส่งผ่านค่าจะทำการคัดลอก (copy) ค่าจากที่หนึ่งไปยังอีกที่หนึ่ง แบ่งเป็น 3 วิธี
    - Call by value จะ copy ค่าจาก actual มายัง formal ในตอนเริ่มต้นเท่านั้น ทำให้เหมือนกำหนดค่าเริ่มต้นให้ formal

- Call by result เมื่อเรียกใช้ procedure จะ bind formal กับ actual แต่ยังไม่มีการคัดลอกค่า เมื่อทำงานเสร็จจึงจะคัดลอกค่าจาก formal ไปยัง actual
- Call by value-result เมื่อ bind แล้วจะส่งค่าจาก actual ไปยัง formal แล้วเริ่มทำงาน จบเสร็จ ก็ส่งค่ากลับจาก formal ไปยัง actual ด้วย

ตัวอย่างโปรแกรมที่ใช้ Call by copy ในภาษา Pascal

```
Procedure Swap(a,b:integer);
var     temp:integer;
begin
        temp := a;
        a := b;
        b := temp;
end;
Call Swap(2,3);
```

- Call by reference จะเป็นส่ง address ของ actual parameter ไปยัง formal parameter นั่นคือ เมื่อคำสั่งกระทำกับ formal จะมีผลกระทบต่อ actual ทันที (ไม่ใช่เป็นการส่งค่า) ข้อสังเกต : ต่างจาก call by value-result ซึ่งจะ formal จะยังไม่มีผลกระทบต่อ actual จนกว่าจะจบโปรแกรม ย่อยจึงจะมีการ copy ค่ากลับไปยัง actual

ตัวอย่างโปรแกรมที่ใช้ Call by reference ในภาษา Pascal

```
Procedure Swap(var a,b:integer);
var     temp:integer;
begin
        temp := a;
        a := b;
        b := temp;
end;
x:=2; y:=3;
Call Swap(x,y);
```

- Call by name เมื่อมีการ bind ระหว่าง formal กับ actual แล้ว จะมีการส่งชื่อไปให้ แต่จะยังไม่มีการส่งค่าใด ๆ จนกว่าจะพบคำสั่งที่เรียกใช้ formal parameter นั้น เช่น a bind x เมื่อพบคำสั่ง temp :=a จะไปค้นหา a จาก x

11.3 เอกสารอ้างอิงและเว็บไซต์ที่ควรรู้

Allen B. Tucker and Robert E. Noonan. Programming Languages – Principles and Paradigms.

Robert W. Sebesta. Concepts of Programming Languages.

สุจิตรา อดุลย์เกษม. เอกสารประกอบการสอนองค์ประกอบคอมพิวเตอร์และภาษา.