บทที่ 3
ผังควบคุม
(Flow chart)

โปรแกรมในภาษา C ทุกโปรแกรมจะต้องเริ่มต้นตัวด้วยฟังก์ชัน main ( ) ซึ่งแสดงว่าเป็นโปรแกรมหลัก จากนั้นก็เป็นการสั่งให้ทำงานตามลำดับก่อนหลังเรื่อยไป เช่นเดียวกันในภาษาอื่น ๆ โครงสร้างที่ใช้ควบคุมและนิยมใช้กันมากในภาษา C มี 3 แบบคือ แบบ if-then-else (แปลว่าถ้าเงื่อนไขเป็นจริงให้ทำนี่มิเช่นนั้นให้ทำโน่น) แบบ while (แปลว่าทำคำสั่งชุดนั้นซ้ำ ๆ อยู่เช่นนั้นขณะที่เงื่อนไขต่าง ๆ เป็นจริง) และแบบ case (แปลว่าทำเฉพาะชุดคำสั่งที่กำหนดเฉพาะกรณี) ดังรายละเอียดต่อไปนี้

3.1 คำสั่ง if

คำสั่ง if ใช้ทดสอบตรรกของนิพจน์ว่าเป็นจริงหรือว่าเป็นเท็จกล่าวคือถ้านิพจน์เป็นจริงก็ให้ปฏิบัติตามหรือทำคำสั่ง (execute) ที่ต่อจาก if หากไม่จริงให้ย้ายไปทำคำสั่งอื่น ไวยากรณ์ของคำสั่ง if ปรากฏดังนี้ ส่วนโครงสร้างปรากฏดังภาพ

```
if (exp) stmt
```

เช่น if (x= =5) y=3; หมายความว่า ถ้า x มีค่าเท่ากับ 5 จริง (ถ้าจริงให้กำหนดค่าของนิพจน์ x= =5 ให้เท่ากับ 1) แล้วละก็ให้ใส่มูลค่าเท่ากับ 3 ลงใน y(ขอให้ดูผังควบคุม)

Flowchart IF

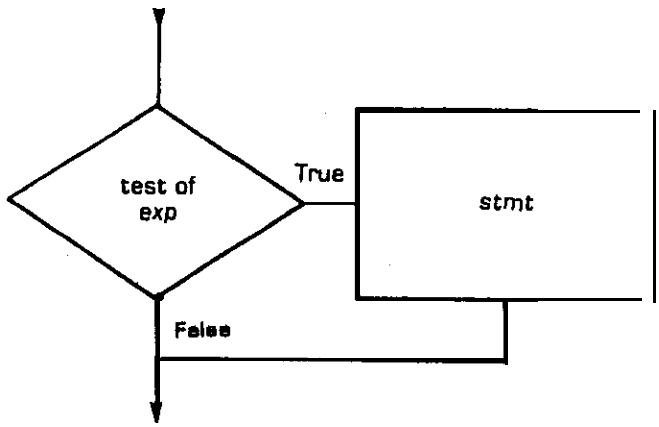


Exhibit 3.1 IF IN OTHER LANGUAGES

BASIC

```
10 IF (X=5) THEN Y=3
```

FORTRAN

```
IF (X.EQ.5) Y=3
```

PASCAL

```
IF (X=5) THEN Y:=3;
```

PL/I

IF (X=5) THEN Y=3;

COBOL 1/

IF X EQUALS 5 THEN COMPUTE Y=3.

3.2 **คำสั่ง** if-else

คำสั่ง if-else เป็นคำสั่งที่แสดงให้เห็นการทำงานของตัวดำเนินการเปรียบเทียบ (relational operator) โดยเราจะใช้คำสั่งนี้สำหรับสั่งให้แยกทำคำสั่งใดคำสั่งหนึ่งในระหว่าง 2 คำสั่งตามค่าของนิพจน์ รูปไวยากรณ์ของ if - else ปรากฏดังนี้

```
if (exp หรือ test criterion)
    stmt 1;
else
    stmt 2;
```

คำสั่ง if-else ในภาษา C ก็คือคำสั่ง IF-THEN-ELSE ในภาษาเบสิค exp หรือ test criterion คือเงื่อนไขที่ใช้ทดสอบโดยที่ถ้าเงื่อนไขเป็นจริงให้ทำคำสั่งใน stmt 1 ถ้าเงื่อนไขไม่จริงให้ทำคำสั่งใน stmt 2 ขอให้สังเกตว่าหลังคำว่า if จะมีวงเล็บเปิดปิด หลังวงเล็บปิดห้ามมีเครื่องหมายอัฒภาค (;) และหลังคำว่า else ก็ห้ามมีเครื่องหมายอัฒภาคเช่นกัน

ตัวอย่างเช่นถ้าเราประสงค์จะพิมพ์คำว่า male ถ้า x=0 และพิมพ์คำว่า female ถ้า $x \neq 0$ จะพบว่าเงื่อนไขคือ x= =0 ซึ่งถ้าเงื่อนไขนี้เป็นจริงเครื่องก็จะ

1/ ตัวอย่างอื่น ๆ เช่น if(i= =3)y=5; if(i=3)y=5; if(i!=0)y=3; if(i) y = 3;

ปฏิบัติตามคำสั่งที่ตามหลังคำว่า if หากไม่จริงเครื่องจะปฏิบัติตามคำสั่งที่ตามหลังคำว่า else คำสั่งที่ตามหลังคำว่า if และ else อาจเป็นคำสั่งเดี่ยว (single statement) หรือกลุ่มคำสั่ง (compound statement หรือ block) ซึ่งจัดกลุ่มไว้ด้วยวงเล็บปีกกาเปิด-ปิดก็ได้ กล่าวคือถ้าเงื่อนไขเป็นจริงเครื่องจะทำตามบล็อคที่ตามหลังคำว่า if ถ้าเงื่อนไขไม่เป็นจริงเครื่องจะทำตามบล็อคที่ตามหลังคำว่า else

ตามตัวอย่างข้างบนเราสามารถเขียนโปรแกรมสำหรับคำสั่ง if-elseได้ดังนี้

```
/* 0=male otherwise female */
if(x= =0)
   'printf ("male");
else
    printf ( "female " );
```

หรือ

```
/* 0 = male'otherwise female and count each */
 if(x==0)
 {
 printf("male")
 is-male  = is_male  + 1
 }
 else
 {
 printf ("female");
 is_famale = is_female+1;
 }
```

<u>หมายเหตุ</u> เราสามารถใช้สไตล์การเขียนบล็อคในคำสั่ง if-else ได้อีกแบบหนึ่งดังนี้ ซึ่งได้ผลตรงกันแต่ใช้เนื้อที่เขียนโปรแกรมน้อยกว่ากระชับกว่า ขอให้สังเกตเครื่องหมาย

วงเล็บปีกกาเปิด-ปิด

```
if (x = =0){
    printf ("male ");
    is_male = is_male+1;
}else{
    printf ( "female" );
    is-female = is_female + 1 ;
}
```

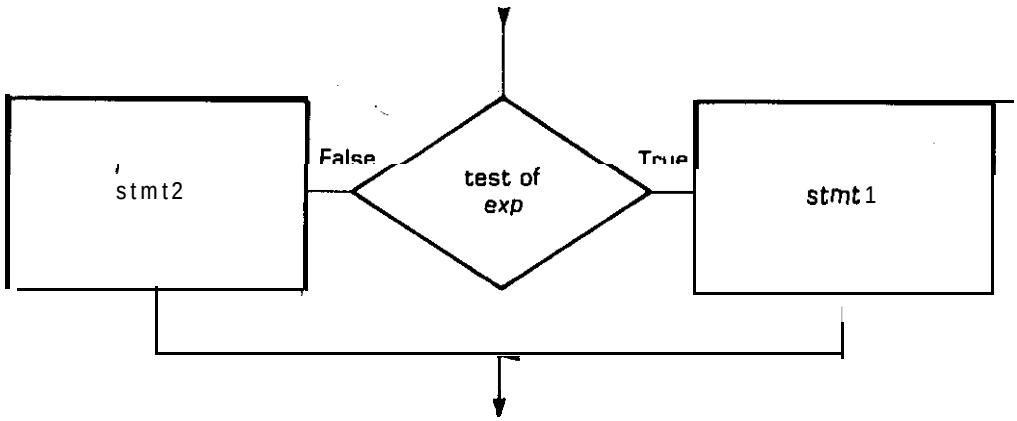
อนึ่งในภาษา C นั้นเรามีตัวดำเนินการตรรก(&&, ||, <<, >>, !, ~, &, |, ^,-) ได้ใช้ในงานทางตรรกโดยถือว่า 0 หมายถึงเท็จ (false) และไม่ใช่ 0 (nonzero) หมายถึงจริง (true) ซึ่งหากเรานำเอาความรู้นี้มาใช้เขียนคำสั่ง if-else จะทำให้ได้โปรแกรมที่รัดกุมและเป็นรูปแบบที่นิยมใช้กันในภาษา C กล่าวคือ ถ้าให้ x คือเงื่อนไขจะพบว่าถ้า x เป็นจริง ค่าของ x จะไม่เป็น 0 ผลก็คือพิมพ์ male โปรแกรมคำสั่ง if-else ข้างต้นจะปรากฏเป็นรูปใหม่ดังนี้

```
if (x){
    printf ("female");
    is_female = is_female+1 ;
} else {
    printf ("male") ;
    is_male = is_male + 1 ;
}
```

ถ้าส่งค่า x=0 เข้ามาจากภายนอก ผลคือ x= =0 ซึ่งก็คือ false ผลลัพธ์คือการพิมพ์คำว่า male ออกมา สำหรับตัวอย่างอื่นปรากฏดังนี้

สมมุติถ้าเรากำหนดให้ y = 1 ถ้า x < 6 ถ้า x มิได้น้อยกว่า 6 ให้กำหนดให้ y = 2 เราสามารถเขียนโปรแกรมและผังโปรแกรมได้ดังนี้ ตัวอย่างที่อยู่ใต้ผังคือคำสั่ง if-elseในภาษาอื่น

```
if (x < 6)
   y = 1 ;
else
   y = 2 ;
```



Flowchart IF-ELSE

BASIC

```
10 Y = 1
20 IF (X >= 6) THEN Y = 2
```

or

```
10 IF (X<6) THEN Y = 1
20 IF (X>=6) THEN Y = 2
```

or

```
10 IF (X<6) THEN Y = 1 : GOTO 30
20 Y = 2
30
```

MBASIC

```
10 IF (X<6) THEN Y = 1 ELSE Y = 2
```

FORTRAN

```
Y = 1
IF (X.GE.6) Y = 2
```

PASCAL

```
IF X < 6 THEN
    Y := 1
ELSE
    Y := 2;
```

PL/I

```
IF X < 6 THEN
    Y = 1;
ELSE
    Y = 2;
```

COBOL

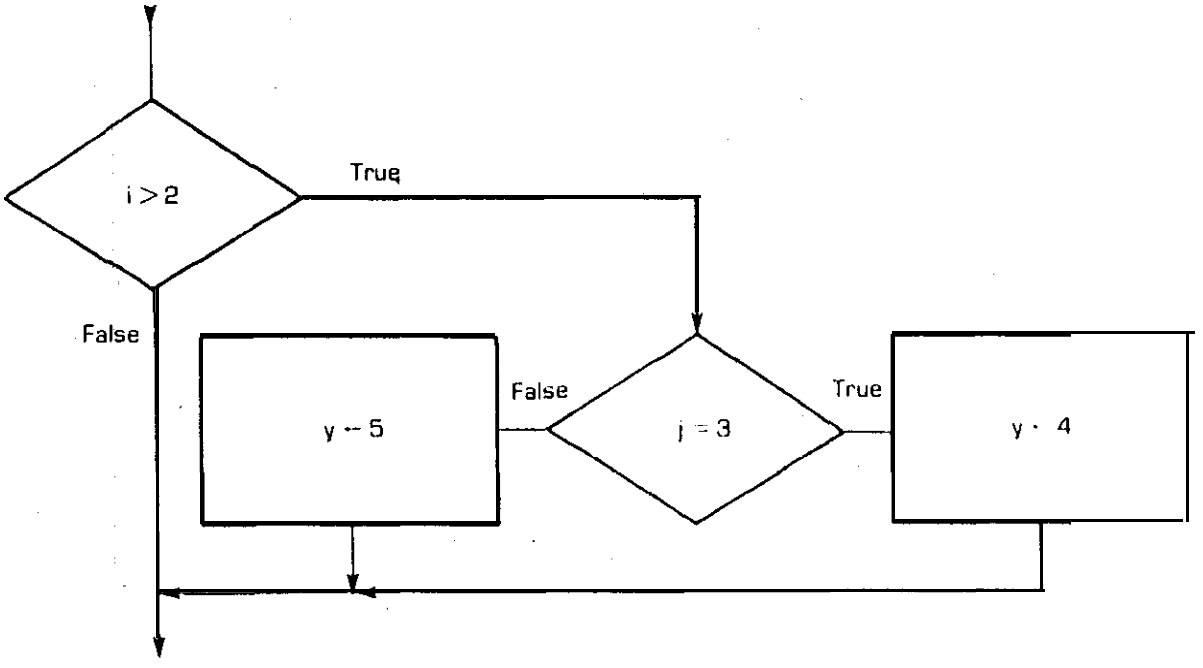
```
IF X LESS THAN 6 THEN
    COMPUTE Y = 1
ELSE
    COMPUTE Y = 2
```

อนึ่งคำสั่ง if-else สามารถซ้อนคำสั่ง if (เรียกว่า nested if) ได้ในกรณี nested if คำสั่ง else จะเป็น else ของ if สุดท้าย เช่น

```
if (i>2)
   if(j= =3)
      y = 4;
   else
      y = 5;
```

ขอให้สังเกตว่าการเขียนโปรแกรมเราจะย่อหน้าให้ else ตรงกับ if สุดท้าย ภาพแสดง nested if ข้างต้นปรากฏดังนี้ จากโปรแกรมแสดงว่า ถ้า i>2 และ j เท่ากับ 3 ให้ใส่ค่า 4 ใน y มิเช่นนั้นใส่ 5 ใน y

Flowchart EXAMPLE OF IF-ELSE

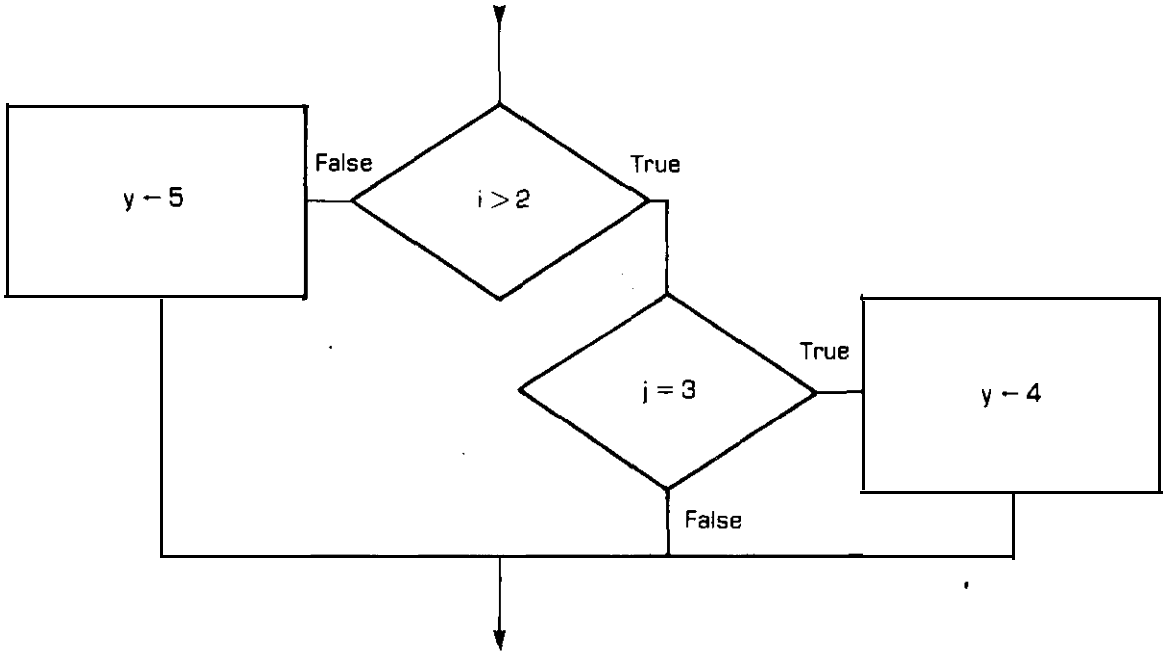


เราอาจเขียนคำสั่ง if-else ซึ่งเป็น nested if ในลักษณะของบล็อคหรือคำสั่งเชิงซ้อนได้ดังนี้

```
if (i>2){
    if(j==3);
    y=4;
}else
    y=5;
```

ผังควบคุมกรณีนี้ปรากฏดังภาพ

Flowchart EXAMPLE OF IF-ELSE



นอกจากเราใช้ตัวดำเนินการตรรกช่วยเขียนคำสั่ง if-else แล้วเรายังสามารถใช้ตัวดำเนินการเงื่อนไข (conditional operator) ช่วยเขียนคำสั่งให้รัดกุมได้เช่นกัน เช่น

```
if (a > b) c = a ;
else c = b ;
```

สามารถเขียนได้ใหม่เป็น `c = (a>b? a:b)`

หรือคำสั่ง

```
if (x<6) y=1;
else y=2;
```

สามารถเขียนได้ใหม่เป็น `y=(x<6?1 :2)`

3.3 while loop

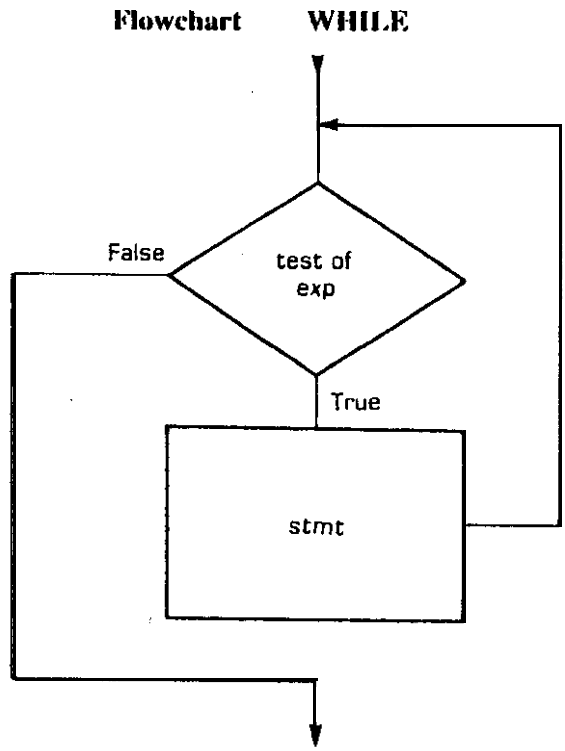
while loop ใช้สำหรับสั่งให้ทำคำสั่งเดิมหรือกลุ่มคำสั่งเดิมซ้ำ ๆ อยู่เช่นนั้นตราบเท่าที่เงื่อนไข (test criterion) ยังเป็นจริง และหยุดทำคำสั่ง (คือออกจาก loop) เมื่อเงื่อนไขไม่เป็นจริง บางครั้งเราอาจใช้คำสั่ง break ช่วยโดยผนวกคำสั่ง break เข้าไปในกลุ่มคำสั่ง (ใช้ได้ทั้งใน for loop และwhile loop) เพื่อเป็นการบังคับให้เกิดสภาวะที่เงื่อนไขไม่เป็นจริง เพื่อเป็นการป้องกันมิให้เกิด infinite loop ซึ่งจะได้กล่าวถึงต่อไป

รูปไวยากรณ์ของ while loop ปรากฏดังนี้

```
while (exp หรือ test criterion){
    stmts ;
    }
```

ถ้ามีคำสั่งเพียงคำสั่งเดียวไม่จำเป็นต้องใช้บล็อค ขณะที่ test criterion นั้นเราถือเอาจริง-เท็จเป็นเกณฑ์ จริงก็คือ nonzero เท็จก็คือ o ถ้าเงื่อนไขเป็นจริงเครื่องจะทำคำสั่งที่ตามหลังวงเล็บเล็ก ถ้าเงื่อนไขไม่เป็นจริงจะไม่ทำคำสั่งใด ๆ จึงขอให้สังเกตว่า while loopอาจไม่ทำคำสั่งเลยก็ได้ ต่างกับ do-while loop ที่จะกล่าว

ต่อไปซึ่งจะต้องทำคำสั่งอย่างน้อย 1 ครั้งก่อนเสมอ เพราะ while อยู่ต่อจาก do ผังควบคุมของ while loop ปรากฏดังนี้



ตัวอย่าง while loop ปรากฏดังนี้

```
i = 0 ;                    i = 5 ;
while (i<5)     หรือ       while (i)
  i++;                       i--;
```

หมายความว่าตราบใด i มีค่าน้อยกว่า 5 คำสั่ง i = i + 1 ; ก็จะทำงานอยู่ เมื่อ i=5 ซึ่งเงื่อนไข (i < 5) เป็นเท็จคำสั่ง i = i + 1 ; ก็จะไม่ถูกสั่งทำงาน และย้ายการควบคุมไปยังจุดอื่น (ดูผังควบคุม) สำหรับตัวอย่างที่อยู่คู่กันซึ่งเป็น loop เดียวกันแต่เขียนคนละแบบจะพบว่าเราระบุ test criterion เป็น (i) เฉย ๆ กรณีนี้การตรวจว่าจริง-เท็จ (true-false) คอมไพเลอร์จะดูว่า i เท่ากับ 0 หรือ nonzero ถ้า i เท่ากับ 0 แสดงว่าเงื่อนไขไม่จริง ถ้า i เป็น nonzero แสดงว่าเงื่อนไขเป็นจริง i-- ; หมายถึง i = i-1 ; เมื่อกำหนดค่าเริ่มต้นให้เป็น 5 ค่าของ i จะลดลงคราวละหนึ่งเป็น 4, 3, 2,1 และออกจาก while loop เมื่อ i เท่ากับ 0

ตัวอย่างข้างบนนี้คำสั่งหลัง while เป็นคำสั่งเดียว ถ้าเป็นกลุ่มคำสั่งคือบล็อคจะพบว่าเราต้องมีวงเล็บปีกกาเปิด-ปิด ตัวอย่างเช่น

```
j = 0 ;
i = 0 ;
while (i < 5){
   j = j+2 ;
   i++;
}
```

คำสั่งในบล็อค { } จะทำงานเรื่อยไป (5 รอบ) ตราบเท่าที่ i มีค่าน้อยกว่า 5 และเมื่อ i = 5 แสดงว่าเงื่อนไขเป็นเท็จ เครื่องจะไม่ทำคำสั่งในบล็อค แต่จะย้ายไปทำคำสั่งที่อยู่ ณ จุดอื่นของโปรแกรม

หรือ ในตัวอย่างต่อไปนี้ ซึ่งแสดงการสั่งให้เครื่องส่งเสียงออด (Bell) เมื่อ loop ทำงานครบ 30,000 ครั้ง

```
# define BELL 7
main ( )
   {
   int x ;
   putchar (BELL) ;
   x = 0;
   while (x! = 30000)
     + + x;
   putchar (BELL) ;
   printf ("loop finished\n at the bell") ;
   }
```

จากตัวอย่างจะพบว่าเงื่อนไขหรือ test criterion คือ (x ! = 30000) โดยเรากำหนดค่าเริ่มต้นให้ x เท่ากับ 0 โดย x เป็น int จะเห็นว่าตราบใดก็ตามที่ x ไม่เท่ากับ 30000 คำสั่ง x = x+1 ; จะถูกสั่งให้ทำงาน จนกระทั่งเมื่อ x = 30000 ซึ่งทำให้เงื่อนไข (x ! = 30000)เป็นเท็จ การควบคุมก็ถูกส่งออกนอก while loop ไปทำคำสั่ง putchar (BELL) ; สำหรับตัวอย่างนี้ while loop ใช้เวลาทำงานเพียง 2 วินาที ขณะที่ถ้าเขียนคำสั่งเป็นภาษาเบสิคจะใช้เวลาทำงานนานถึง 96 วินาที !!

อนึ่ง จากตัวอย่างที่ผ่านมาเรามีทั้งตัวอย่างของ pre-increment เช่น + + x ; และ post increment เช่น i + + ; จึงขอถือโอกาสนี้อธิบายเพิ่มเติมเรื่องของ increment และ decrement อีกครั้งหนึ่ง ขอให้ดูจากตัวอย่างต่อไปนี้

```
main ( )
{
    int x, is-female,  is-male  ;
    is-female = 0 ;
    is-male  = 0 ;
    x = 0;
    if (x){
         printf  ( "female")  ;
         ++is_female ;
   }else{
        printf ( "male" ) ;
        ++  is-male  ;
   }
}
```

จากตัวอย่าง ++is_male ; และ ++is_female ; เป็น pre-incrcment จากคำสั่ง ifแสดงว่า x = ++is_male ; และ x=++is_female ; จะต้องได้รับการเพิ่มค่าก่อน คือเพิ่มจำนวนให้แก่ counter คือ x หนึ่งค่าก่อนจึงทำงาน ตามโปรแกรม ขณะที่ is_male++ ; และ is_female ++ ; แสดงว่าจะต้องมีการทำงานตามโปรแกรมก่อนหนึ่งรอบจึงค่อยเพิ่มค่าให้แก่ x เราเรียกกรณีเช่นนี้ว่า - post increament

สำหรับกรณี decrement เราจะมีทั้ง Pre-decrement และ post-decrement ซึ่งก็มีหลักการตรงกันกับทำ incrementเพียงแต่เปลี่ยนจากเพิ่มค่าเป็นลดค่า

ตัวอย่างคำสั่งโปรแกรมภาษาอื่นเรื่อง while loop ปรากฏดังนี้

BASIC

```
10 J=0
20 I=0
30 IF (I>=5) GOTO 70
40 J=J+2
50 I=I+1
60 GOTO 30
70 . . .
```

MBASIC

```
10 J=0
20 I=0
30 WHILE (I<5)
40 J=J+2
50 I=I+2
60 WEND
```

FORTRAN

```
    J=0
    I=0
30 IF (I.GE.5) GOTO 70
    J=J+2
    I=I+1
    GOTO 30
70 . . .
```

PASCAL

```
J := 0;
I := 0;
WHILE (I<5) DO
      BEGIN
            J := J+2;
            I := I+1
        END;
```

PL/I

```
J=0;
I=0;
DO WHILE (I<5) ;
        J=J+2;
        I=I+1;
        END;
```

COBOL

```
COMPUTE J=0.
COMPUTE I=0.
PERFORM INC
        UNTIL I IS GREATER THAN 5 OR I IS EQUAL TO 5

INC.
        COMPUTE J = J + 2
        COMPUTE I=I+ 1.
```

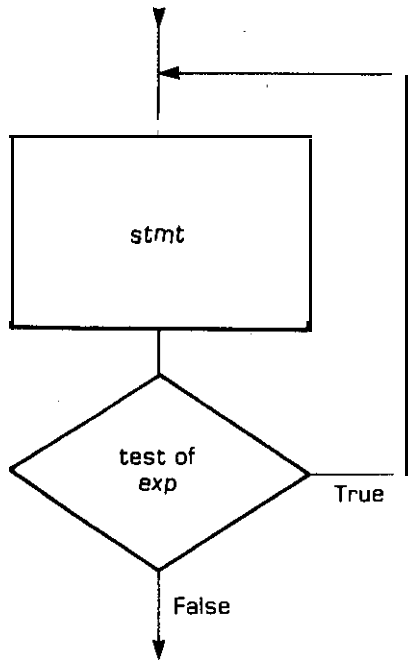
3.4 do-while loop

do while loop มีรูปไวยากรณ์ดังนี้

```
do { stmts ;
  } while (exp หรือ test criterion) ;
```

จากรูปไวยากรณ์จะเห็นว่า do ดำเนินก่อนถึง while การทำงานของ do-while loop จึงมีขึ้น 1 ครั้งก่อนจึงจะตรวจเงื่อนไขคือ test criterion หากพบว่าเงื่อนไขเป็นจริงก็จะทำคำสั่งที่อยู่ต่อจาก do ใหม่อีก วนเวียนอยู่เช่นนี้จนกว่าจะพบว่าเงื่อนไขไม่เป็นจริงจึงออกจาก loop คือย้ายการควบคุมไปสู่จุดอื่น

ผังควบคุมของ do-while loop ตามรูปไวยากรณ์ข้างต้นปรากฏดังนี้



Flowchart 3.6 DO-WHILE

ตัวอย่างเช่น

```
j = 4;
do {
   i = j -- ;
  } while (i > 5) ;
```

หรือ

```
# define BELL 7
main ( )
  {
    int x;
    putchar (BELL) ;
    x = 0 ;
    do
       {
         ++ x;
       while (x ! = 30000) ;
    putchar (BELL) ;
    printf ("loop finished \n at' the bell ") ;
   }
  }
```

ขอให้สังเกตว่าเราจะต้องมีเครื่องหมายอัฒภาค(;) หลัง while ( ) เสมอ สำหรับ do-while ในภาษาอื่นปรากฏดังนี้

BASIC

```
10 J=4
20 I=J
30 J=J-1
40 IF (I>5) GOTO 20
```

FORTRAN

```
   J=4
10 I=J
   J=J-1
   IF (I>5) GOTO 10
```

PASCAL

```
J=4;
REPEAT
      I=J;
      J=J-1
UNTIL I<=5;
```

PL/I

```
J=4;
DO UNTIL (I<=5);
      I=J;
      J=J-1;
      END;
```

COBOL

```
      COMPUTE J=4.
      COMPUTE I=6.
      PERFORM INC. UNTIL I<=5.
          . . .
INC.
      COMPUTE I=J.
      COMPUTE J=J-1.
```

3.5 for loop

for loop มีลักษณะคล้าย while loop โดยสามารถเขียนแทนกันได้ for ในภาษา C ต่างจาก for ในภาษาเบสิคตรงที่ในภาษา C นั้น for loop จะนำเอาข้อสนเทศทั้งปวงที่เกี่ยวข้อง เช่นค่าเริ่มต้นค่าสุดท้ายและเงื่อนไขมารวมไว้ในที่เดียวกัน รูปไวยากรณ์ของ for loop ปรากฏดังนี้คือ

```
                (1)              (2)            (3)
for (initialized value(s); test criteria; loop increment){
      stmts controlled by loop;
    }
```

เช่น โปรแกรมในภาษาเบสิคที่สั่งให้เปลี่ยนค่าของ J ไปเรื่อยแล้วนับจำนวนครั้งเก็บไว้ในที่ของตัวแปรชื่อ X ปรากฏดังนี้ ขอให้สังเกตว่า for-loop เป็น loop ของตัวแปร J ที่มีค่าเริ่มต้นที่ 0 สั่งให้ทำงานคือ X = X + 1 เงื่อนไขที่บังคับให้ทำคำสั่งนี้คือ J มีค่าไม่เกิน 30000 และ loop increment คือ NEXT J

```
100 X = 0
110 FOR J = 0 TO 30000
120 X=X+1
130 NEXT J
140 END
```

หากเขียนเป็นภาษา C จะปรากฏดังนี้คือ

```
int x, j;
x = 0;
for (j = 0; j < = 30000; + + j)
     + + x;
```

จะเห็นได้ว่า j=0 ; คือค่าเริ่มต้นของ loop (j=0 หมายถึงการกำหนดค่าเริ่มต้นให้ j เป็น 0 เรียกตัวแปรนี้ว่าตัวแปรควบคุม loop) j <= 30000 คือเงื่อนไขทดสอบ เงื่อนไขทดสอบโดยปกติจะใช้ตัวดำเนินการเปรียบเทียบเช่น != , == , < , <= , > , >= หรือตัวดำเนินการตรรกเช่น && , || หรืออื่น ๆ แล้วแต่กรณี ++j คือ loop increment ในภาษาเบสิคคือ NEXT J ขณะที่ตัว loop ก็คือ stmt ในที่นี้คือ++x ในภาษาเบสิคคือ X = X+1 ขอให้สังเกตว่าตัว loop จะต้องตามด้วยเครื่องหมายอัฒภาค (;) เรียกว่า statement terminator เสมอ หากเราไม่เขียนจะผิดไวยากรณ์ เพราะตัว loop จะต้องมีเพื่อให้ for loop ควบคุมการทำงาน หากเรายกเอาตัว-loop ไปรวมไว้ในวงเล็บเล็ก ตัว loop จะหายไป การทำงานตามคำสั่งfor loopจะไม่ปรากฏ เพราะตัว loop ไม่ได้สั่งอะไร ขอให้สังเกตว่าเรายังต้องเขียน ; ไว้ในที่ของตัว loop เช่น

```
for (j=0; j <= 30000; ++j, ++x)
      ;
```

และ

```
for (j=0, x=0; j<= 30000; ++j, ++x)
      ;
```

กรณีนี้ ++x จะทำหน้าที่เป็น loop increment ในภาษาเบสิคจะเขียนเป็น NEXT X ขอให้สังเกตเครื่องหมายจุลภาค (,) ที่แยกค่าเริ่มต้น x=0 และ j=0 กับ ++j และ ++x ออกจากกัน ซึ่งเป็นไวยากรณ์ของกรณี for loop ตัว loop คือ ; เรียกว่า null statement หรือ do nothing statement แปลว่า loopจะวิ่งไปโดยไม่มีการทำคำสั่งใดเพราะตัว loop ไม่ได้สั่งให้ทำอะไร การเขียนคำสั่งในfor loop จึงต้องระวังอย่าเผลอเขียนเครื่องหมายอัฒภาคหลังวงเล็บเล็กปิดเพราะจะทำให้เกิด null statement โดยอัตโนมัติ เช่น

```
int sum, x, y;
sum = 0,
y = 5;
for (x=1 ; x<= y ; x++) ;
    sum = sum + x*y ;
```

อนึ่งคำว่า loop increment นั้นมิได้หมายถึง increment อย่างเดียว แต่หมายรวมถึง decrement ด้วยเช่น

```
for (i=5; i>0; i-- )
    j=j*2 ;
```

ตัวอย่าง for loop ในภาษาอื่นสำหรับ loop เดียวกันกับ loop ต่อไปนี้ปรากฏดังภาพ

```
for (i=0 ; i<5 ; i++)
    j++ ;
```

BASIC

```
10 FOR I=1 TO 5 STEP 1
20 J=J+1
30 NEXT I
```

FORTRAN

```
   DO 10 I=1,5,1
   J=J+1
10 CONTINUE
```

PASCAL

```
   FOR I:= 1 TO 5 DO
       J:= J+1;
```

PL/I

```
   DO I=1 TO 5 BY 1;
       J=J+1;
       END;
```

COBOL

```
   PERFORM INC VARYING I FROM1 TO 5 BY 1.
            . . .
INC.
   COMPUTE J=J+1.
```

สิ่งที่พิเศษสำหรับภาษา C ในเรื่องของ loop ก็คือตัวแปรควบคุมloop (loop control variable) จะเป็นตัวแปรแบบใดก็ได้ไม่จำเป็นต้องที่จะต้องเป็นได้เฉพาะ int เช่นในภาษาอื่น นอกนี้เรายังสามารถละเลยไม่เขียนตัวแปรควบคุม (ไม่เขียนค่าเริ่มต้น) หรือไม่เขียนเงื่อนไขหรือไม่เขียน loop increment รวมถึง loop body เช่น

```
for ( ; ; )
   ;
```

แสดงว่าเราไม่ระบุทั้งจุดเริ่ม จุดปลายหรือเงื่อนไขทดสอบ และ increment ของ loop กรณีนี้เงื่อนไขจะเป็นจริงเสมอ คือจุดปลายไม่สิ้นสุดทำให้มีการทำคำสั่งตาม null statement อย่างไม่สิ้นสุด (infinite loop)

loop นี้สามารถเขียนเป็น while loop ได้เป็น

```
while (1)
  ;
```

หรือใน loop ต่อไปนี้คือ

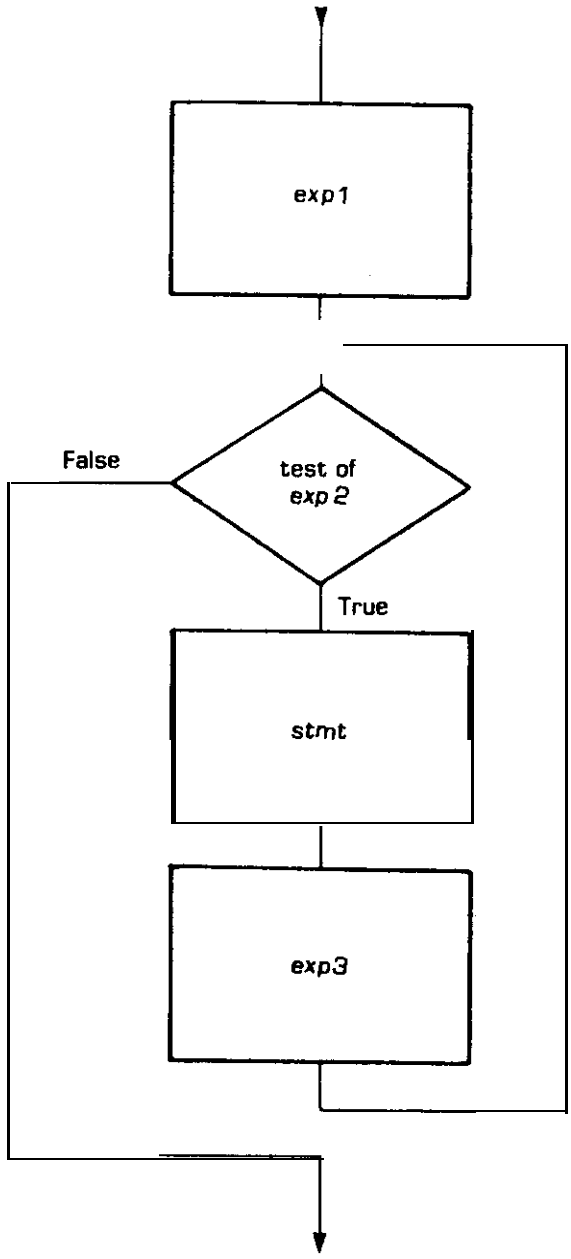
```
for ( ; i<5 ; )      ก็คือ      while (i<5)
    i++;                            i++;
```

ผังควบคุมสำหรับ for loop ปรากฏดังนี้

3.6 คำสั่ง break

คำสั่ง break เป็นคำสั่งที่เราใช้สำหรับสั่งให้ตัดออกจาก for loop while loop หรือ do while loop ก่อนที่ loop incerement จะวิ่งถึงค่าสุดท้ายของตัวควบคุม loop ตัวอย่างเช่น

Flowchart FOR



```
main ( )
{
    int x ;
    x = 0 ;
    for ( ;; ++x ){
        if (x >= 30000 )
            break ;
    }
    printf ("%d", x );
}
```

ตามตัวอย่างนี้เรากำหนดให้ for loop เป็น infinite loop โดยให้ x มีค่าเพิ่มขึ้นคราวละ 1 ขอให้สังเกตบล็อคที่ตามหลังวงเล็บเล็กปิดคือ { if (x > = 30000) break ;} จะทำหน้าที่เป็น loop body ซึ่งสั่งให้ทดสอบว่า x มากกว่าหรือเท่ากับ 30000 หรือยัง ถ้ามากกว่าหรือเท่ากับ 30000 แล้วให้ตัดออกจาก for loop ตามคำสั่ง break ไปทำคำสั่ง printf ("%d" , x) ;

หรือในตัวอย่างต่อไปนี้

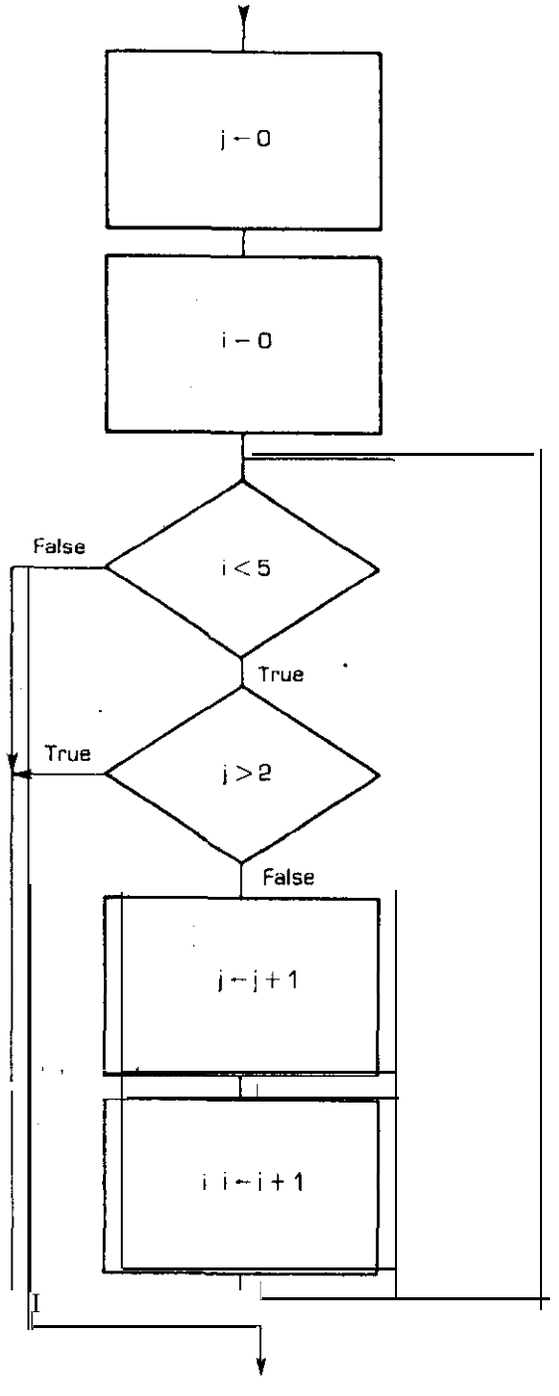
```
j = 0 ;
for (i = 0 ;i< 5 ; i++){
    if (j > 2) break ;
    j++ ;
}
```

ตัว loop คือ { if (j > 2) break ; j++ ;} จะพบว่าการทดสอบตามบล็อคนี้จะทำงาน 4 ครั้งคือ j = 0, 1, 2, 3 โดยที่เมื่อ j=3, เครื่องจะทำตามคำสั่ง break คือตัดออกจาก for loop ไปทำงานที่อยู่ภายนอกขอให้พิจารณาผังควบคุมข้างล่างควบคู่กันไป

กรณีที่เป็น loop ซ้อน (nested loop) คำสั่ง break จะใช้ตัดออกจาก loop ที่อยู่ด้านในที่สุด

ภาพแสดงการควบคุมสำหรับคำสั่ง break ข้างต้นปรากฏดังนี้

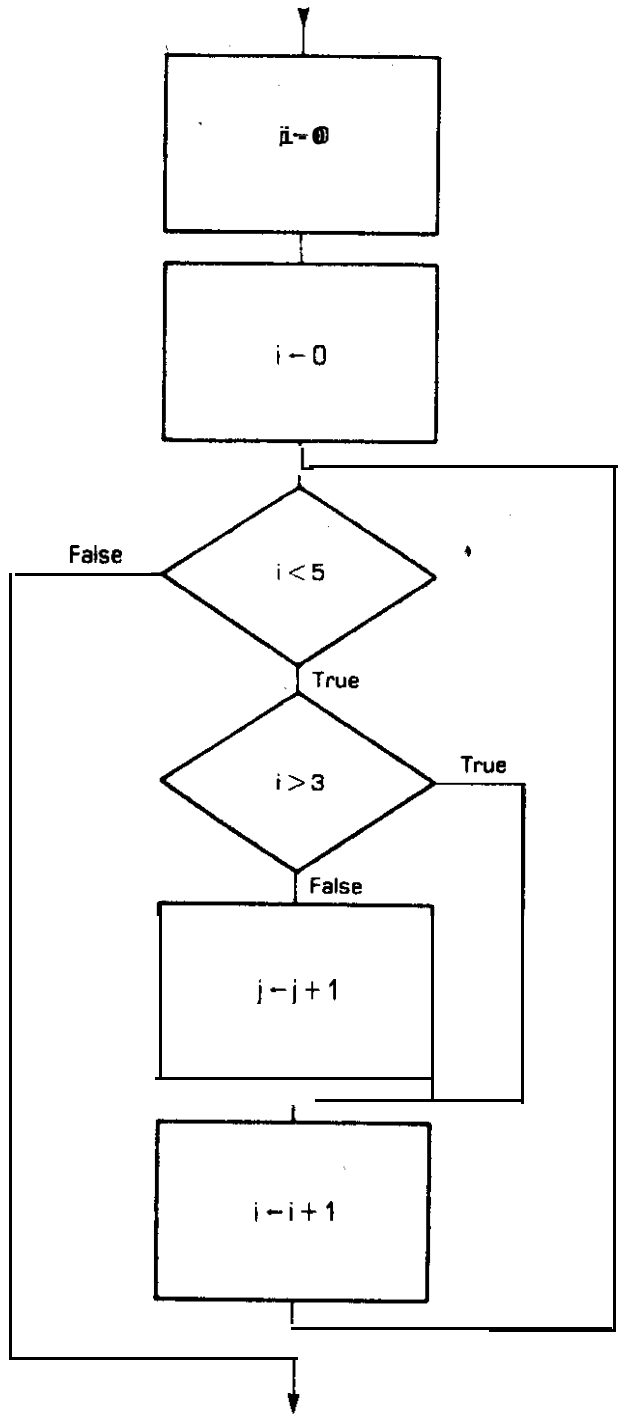
Flowchart EXAMPLE OF **BREAK**



3.7 คำสั่ง continue

คำสั่ง continue ใช้สั่งให้ loop increment ทำงานต่อไปทันทีที่เงื่อนไขเป็นจริง โดยโคดข้ามคำสั่งที่ต่อจากคำว่า continueไปก่อนได้ ผังควบคุมปรากฏดังนี้

Flowchart EXAMPLE OF CONTINUE



จากผังควบคุมจะเห็นว่าเป็น for loop ของ i ที่ j วิ่งตาม i ตราบเท่าที่ i มีค่าไม่เกิน 3 เมื่อ i มากกว่า 3 คำสั่ง continue จะสั่งให้โดดไปทำงานตาม loop increment โดยข้าม j ไปก่อน loop นี้จะทำงาน 5 รอบตาม i (แต่ j ทำงาน 4 รอบ) ถ้า i เท่ากับ 5 j เท่ากับ 4 การควบคุมจะหลุดออกจาก for loop ไปทำงานที่จุดอื่น

จากผังควบคุมเราสามารถเขียนโปรแกรมได้ดังนี้

```
j = 0;
for (i=0 ; j<5 ; i++){
    if (i<3) continue ;
    j++ ;
}
```

จะเห็นได้ว่า loop body คือ {if (i>3) continue ; j++;} ขอให้ลองเช็คตัวเลขดู ข้อแตกต่างระหว่าง break กับ continue ก็คือกรณี break นั้นหากเงื่อนไขเป็นจริงจะตัดออกนอก loop ทันทีขณะที่ continue จะสั่งให้โดดไปทำ loop increment แล้วค่อยตรวจสอบเงื่อนไข เมื่อเงื่อนไขไม่จริงจึงตัดออกนอก loop (ดูผังควบคุมเปรียบเทียบ)

3.8 คำสั่ง switch

คำสั่ง switch ใช้สำหรับช่วยให้การทำงานประเภท nested loop รวดเร็วขึ้น ชัดเจนและเข้าใจง่าย คำสั่ง switch ใช้สั่งให้แยกไปทำงานตาม case ที่ต้องการ เช่น

```
switch (i){
    case 1 :
    j = j+5
    case 2 :
    j = j+2
    case 3 :
       j = j+4
    }
```

หมายความว่า ถ้าหาก i=1 คำสั่ง switch จะสั่งให้แยกไปทำคำสั่งใน case 1 : คือ j=j+5 ; ถ้าหาก i=2 คำสั่ง switch จะสั่งให้แยกไปทำคำสั่งในcase 2 : คือ j=j+2 ; ถ้าหากว่า i=3 ก็จะสั่งให้แยกไปทำคำสั่งใน case 3 : คือ j=j+4 ; เป็นต้น

ขอให้สังเกตว่า เมื่อมีการแยกไปทำคำสั่งใดแล้วก็ควรที่จะตัดออกจาก loop ได้และเราน่าจะเผื่อทางเลือกไว้บ้าง ตามตัวอย่างข้างบนเราน่าจะเผื่อทางเลือกกรณีที่ i มิใช่ 1, 2 และ 3 ไว้ด้วย ด้วยเหตุนี้ในคำสั่ง switch จึงมักมีคำสั่ง break และ default ไว้เสมอ

รูปไวยากรณ์ของคำสั่ง switch ที่มีทั้ง break และ default ปรากฏดังนี้

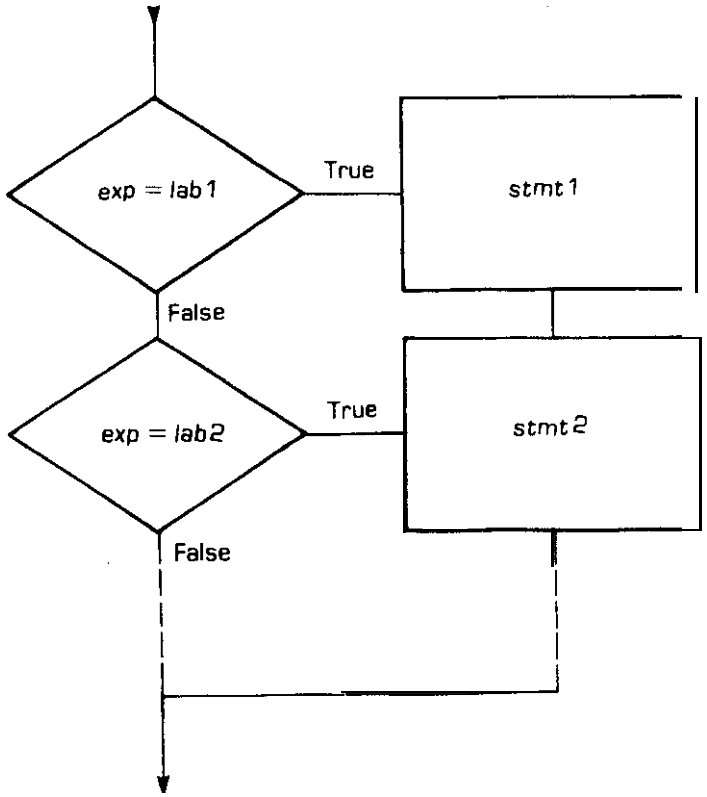
```
switch(integer exp){
    case lab 1:
        stmt 1(optional)
        break;
    case lab 2:
        stmt 2(optional)
        break;
        .
        .
        .
    default
        stmt(optional)
        break;
}
```

คำว่า **lab** หมายถึง **label** จะต้องเป็นจำนวนเต็มบวกเสมอ ลองดูตัวอย่างต่อไปนี้

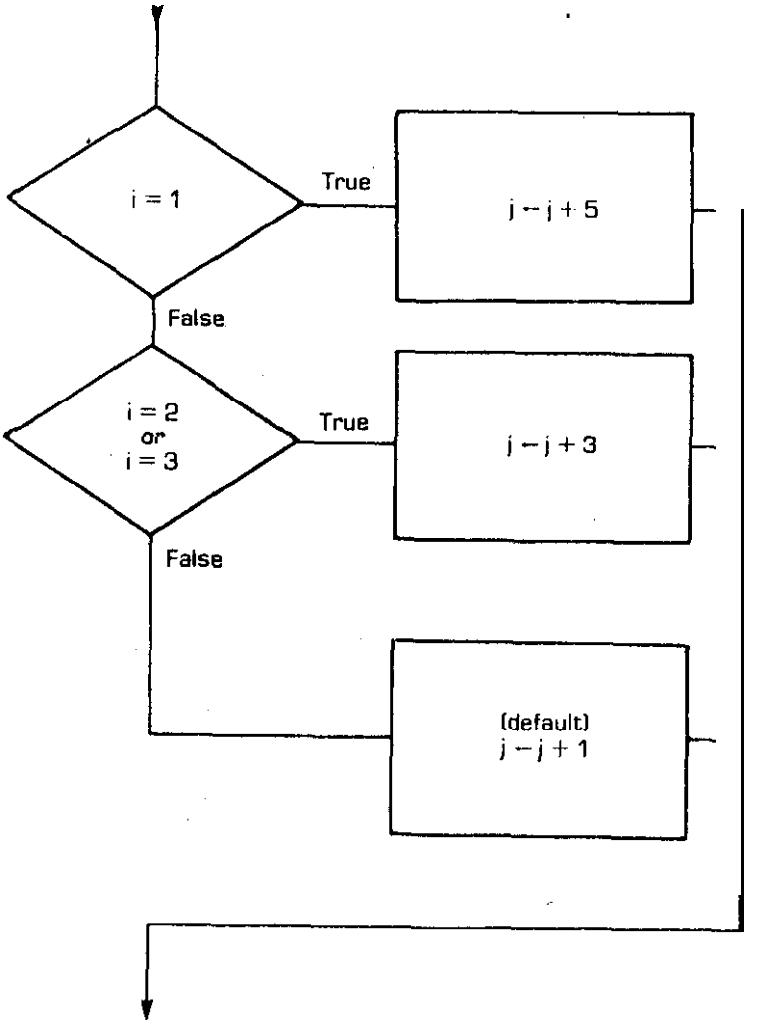
```
switch(i){
    case 1 :
        j=j+5;
        break;
    case 2:
    case 3:
        j=j+3;
        break;
    default:
        j=j+1;
}
```

หมายความว่า ถ้า i = 1 จะทำ case 1 คือคำสั่ง j = j + 5 ; แล้วออกจาก loop ถ้า i = 2 หรือ 3 จะทำคำสั่ง j = j + 3 ; แล้วตัดออกจาก loop ถ้า i มีค่านอกเหนือไปจากนี้จะทำคำสั่ง j = j + 1 ; ผังควบคุมของคำสั่งข้างต้นปรากฏดังนี้

Flowchart SWITCH (NO BREAKS)



Flowchart **SWITCH (WITH BREAKS)**



คำสั่งที่เขียนในภาษาอื่นจากผังควบคุมเดียวกันนี้ปรากฏดังนี้

BASIC

```
5   ON (I) GOTO 10,20,20
7   GOTO 30
10 J=J+5
15 GOTO 40
20 J=J+3
25 GOTO 40
30 J=J+1
40 CONTINUE
```

FORTRAN

```
    GO TO (10,20,20), I
    GOTO 30
10 J=J+5
    GO TO 40
20 J=J+3
    GO TO 40
30 J=J+1
40 CONTINUE
```

PASCAL

```
CASE I OF
        1:    J := J+5;
        2,3: J := J+3;
        END
```

There is no default case for standard PASCAL.

PL/I

```
SELECT (I);
        WHEN (1) J = J+5;
        WHEN (2,3) J= J+3;
        OTHERWISE J= J+1;
        END;
```

COBOL

```
    GO TO A
          B
          B
          DEPENDING ON I.
    COMPUTE J = J+1.
    GO TO C.
A. COMPUTE J=J+5.
    GO TO C.
B. COMPUTE J=J+3.
    GO TO C.
C.
```

เพื่อซักซ้อมความเข้าใจขอให้ผู้อ่านลองเขียนผังควบคุมและใช้คำสั่ง switch กับคำสั่ง nested if ต่อไปนี้

```
if (i==1) j = j + 5 ;
else if ((i==2) || (i==3)) j = j + 3 ;
else j = j + 1 ;
```

คำตอบควรจะปรากฏดังนี้

```
switch (i){
   case 1 :
    j = j + 5 ;
  case 2 :
  case 3 :
    j = j + 3 ;
  default :
     j=j+1 ;
  }
```