

บทที่ 6

โครงสร้างควบคุมแบบทำซ้ำ (Repetition control structure)

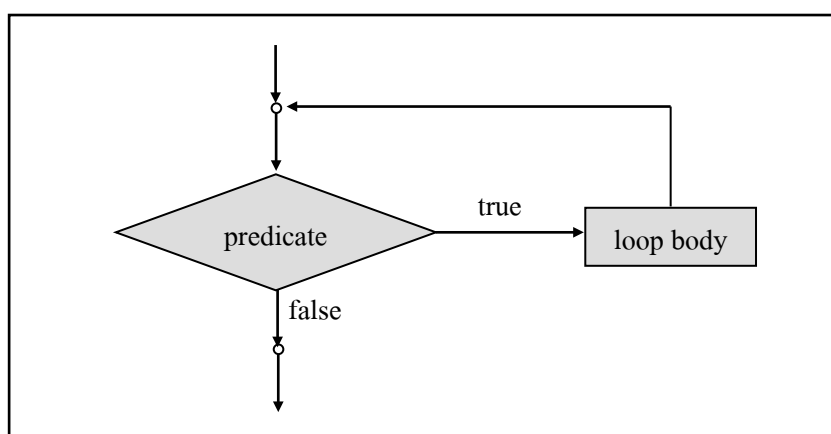
ในทุก ๆ ภาษาโปรแกรม (programming language) จะมีโครงสร้างบางอย่างที่ใช้นิยามควบคุมทำซ้ำหรือวนซ้ำ บนโปรแกรมบล็อก โปรแกรมบล็อกจะถูกกระทำการแบบซ้ำโดยข้อความสั่งที่ถูกทำซ้ำ จะถูกเรียกว่า ลำตัววนซ้ำ (loop body)

การทำซ้ำจะถูกควบคุมโดยเงื่อนไขที่เกี่ยวข้องกับข้อความสั่งทำซ้ำ
ข้อความสั่งทำซ้ำจะถูกแบ่งออกเป็น 2 แบบ คือ

1. การทดสอบก่อนทำข้อความสั่งทำซ้ำ (Pretest repetition statement)
2. การทดสอบหลังทำข้อความสั่งทำซ้ำ (Posttest repetition statement)

1. การทดสอบก่อนทำข้อความสั่งทำซ้ำ (Pretest repetition statement)

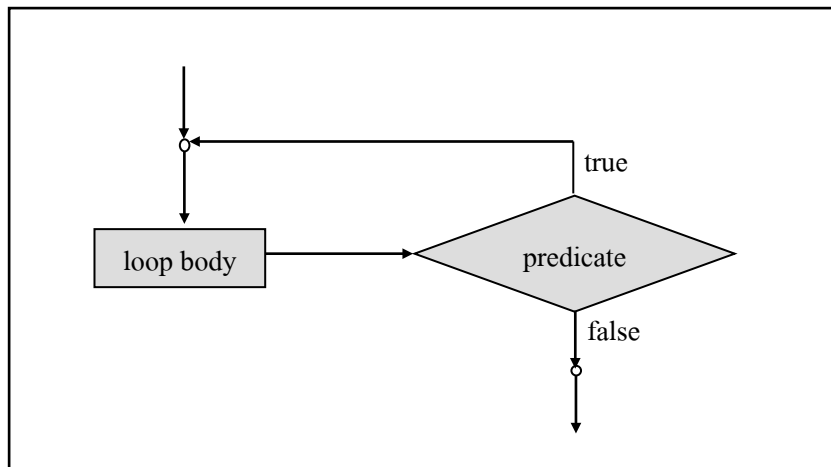
การทดสอบก่อนทำข้อความสั่งทำซ้ำ จะมีการคำนวณค่าที่เกี่ยวข้องกับเพรดิเคต (predicate) ก่อนที่จะเข้าไปในลำตัววนซ้ำ และลำตัววนซ้ำจะถูกกระทำการเมื่อเพรดิเคตมีค่าเป็นจริง มิฉะนั้นแล้วการทำซ้ำจะยุติ และการควบคุมโปรแกรมจะถูกส่งไปยังข้อความสั่งถัดไปจากข้อความสั่งทำซ้ำ ซึ่งสามารถแสดงเป็นผังงาน (flowchart) ได้ดังรูปที่ 6.1



รูปที่ 6.1

2. การทดสอบหลังทำข้อความสั่งทำซ้ำ (Posttest repetition statement)

การทดสอบหลังทำข้อความสั่งทำซ้ำ ส่วนของลำตัววนซ้ำจะถูกกระทำการก่อนอันดับแรก แล้วค่อยไปคำนวณเพรดิเคต ถ้าเพรดิเคตมีค่าเป็นจริง ส่วนของลำตัววนซ้ำจะถูกกระทำการอีกครั้ง มิฉะนั้นแล้ว การทำซ้ำจะสิ้นสุด ซึ่งสามารถแสดงเป็นผังงานได้ดังรูปที่ 6.2



รูปที่ 6.2

ในภาษา C จะมีข้อความสั่งทำซ้ำ 3 แบบ คือ

- 1) ข้อความสั่ง while ซึ่งจะมีการทดสอบก่อนทำข้อความสั่งทำซ้ำ
- 2) ข้อความสั่ง for ซึ่งจะมีการทดสอบก่อนทำข้อความสั่งทำซ้ำ
- 3) ข้อความสั่ง do-while ซึ่งจะมีการทดสอบหลังทำข้อความสั่งทำซ้ำ

6.1 รูปแบบทั่วไปของข้อความสั่ง while ดังรูป 6.3



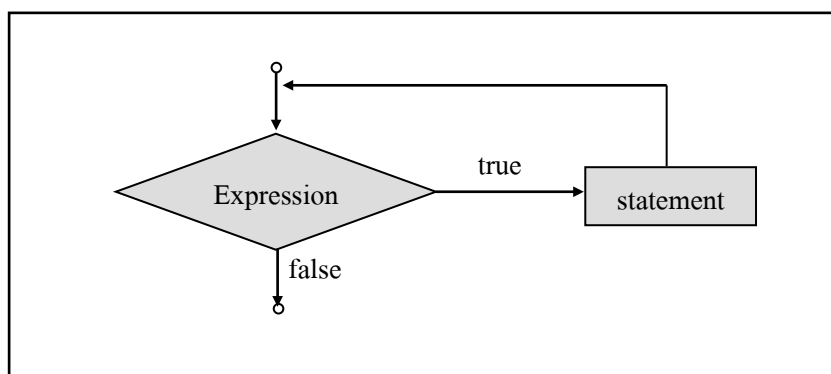
รูปที่ 6.3

ลักษณะโครงสร้างของ while จะประกอบด้วย

1) Expression จะต้องถูกปิดล้อมด้วยเครื่องหมาย (และ) โดยจะต้องอยู่ตามหลังคำสงวน while ซึ่งจะถูกเรียกว่า นิพจน์ควบคุมทำซ้ำ (loop control expression) ซึ่ง Expression อาจจะเป็นนิพจน์เลขคณิต หรือนิพจน์เงื่อนไข ค่าที่คำนวณได้นั้น ถ้าเป็นตัวเลขไม่ใช่ศูนย์ จะสมนัยกับค่าจริงหรือถ้าเป็นตัวเลขศูนย์จะสมนัยกับค่าเท็จ

2) นิพจน์ควบคุมทำซ้ำที่ถูกปิดล้อมด้วยเครื่องหมาย (และ) จะต้องถูกคำนวณเป็นอันดับแรกก่อน ถ้าผลที่ได้มีตัวเลขไม่ใช่ศูนย์ (เป็นจริง) แล้วค่าตัวทำซ้ำจะถูกกระทำการ แล้วการควบคุมโปรแกรมจะย้อนกลับไปยังนิพจน์ควบคุมทำซ้ำ และค่าของนิพจน์จะถูกคำนวณอีกครั้ง วัฏจักร (cycle) นี้จะถูกทำต่อเนื่องจนกระทั่งค่าของนิพจน์ที่คำนวณได้นั้น จะมีค่าเป็นศูนย์

3) ถ้านิพจน์ควบคุมทำซ้ำมีค่าเป็นศูนย์ (เป็นเท็จ) ลำตัววนซ้ำจะถูกเบี่ยง (bypass) และการควบคุมโปรแกรมจะหยุดลงไปทำข้อความสั่งถัดจากข้อความสั่ง while สามารถแสดงผังงานได้ดังรูปที่ 6.4



รูปที่ 6.4

ตัวอย่างโปรแกรมที่ 6.1

```

#include <stdio.h>

main ()
{
    int time = 1 ;      /* Counter variable. */

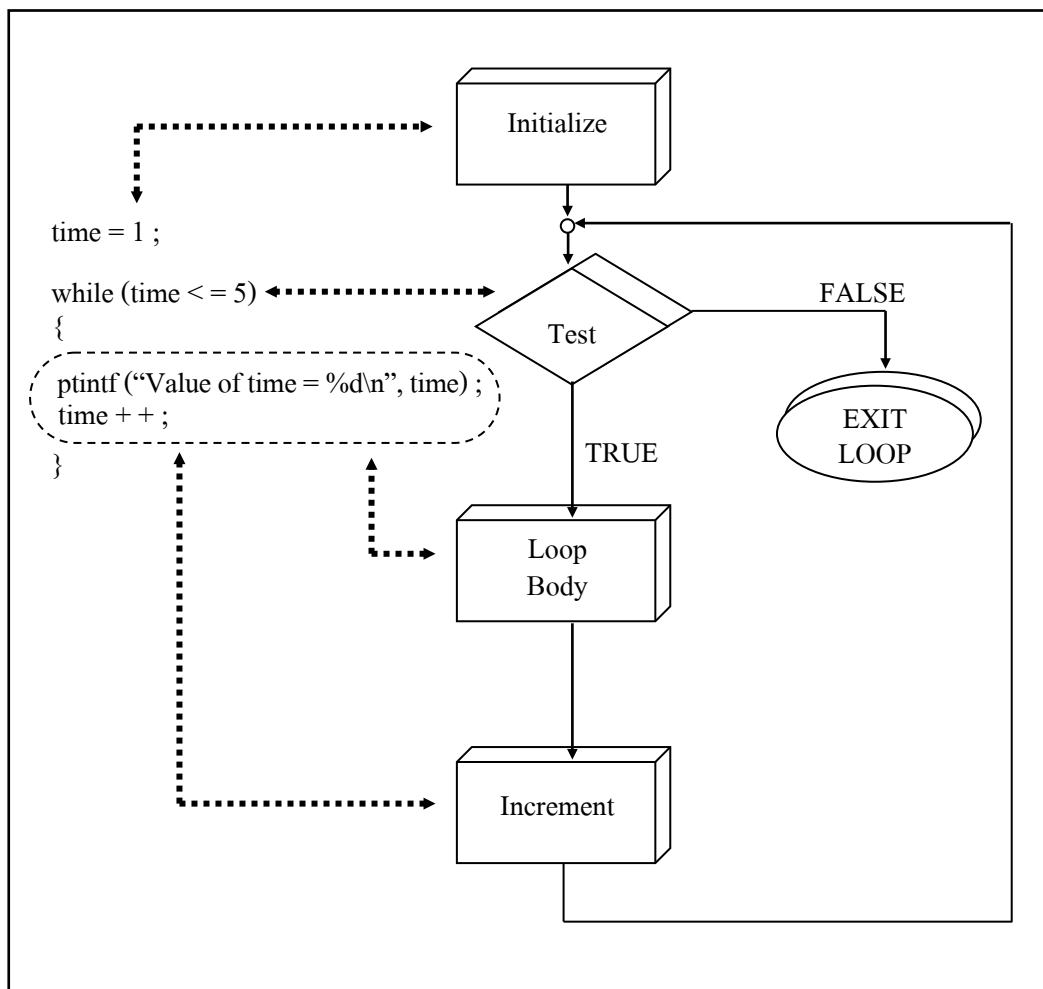
    while (time <= 5)
    {
        printf ( "Value of time = %d\n", time ) ;
        time++ ;
    } /* End of while. */

    printf ( "End of the loop." ) ;
}
  
```

```

Value of time = 1
Value of time = 2
Value of time = 3
Value of time = 4
Value of time = 5
End of the loop.
  
```

ซึ่งโปรแกรมนี้สามารถแสดงเป็นผังงานได้ดังรูปที่ 6.5



รูปที่ 6.5

ตัวอย่างโปรแกรมที่ 6.2 จะแสดงการพิมพ์ตัวเลข จาก 1 ถึง 10


```
/* Counter-control repetition */
#include <stdio.h>

/* function main begins program execution*/
int main ()
{
    int counter = 1 ; /* initialization */

    while (counter <= 10) { /* repetition condition */
        printf ( "%d\n", counter ) ; /* display counter */
        ++ counter ; /* increment */
    } /* end while */

    return 0 ; /* indicate program ended successfully */

} /* end function main */
```



```
1
2
3
4
5
6
7
8
9
10
```

จากโปรแกรมนี้ ส่วนข้อความสั่ง while ในภาษา C ขอมให้เขียนได้ดังนี้

```
while (++ counter <= 10)

    printf ( "%d\n", counter ) ;
```

แต่ต้องกำหนดค่าเริ่มต้นให้ counter = 0 ; ก่อน ข้อความสั่ง while ;

เนื่องจาก ++ counter หมายถึง เพิ่มค่า counter ขึ้น 1 ก่อน

ตัวอย่างโปรแกรมที่ 6.3 จะเป็นโปรแกรมการคำนวณหาค่าเฉลี่ยของเกรด

```

/* Class average program counter-controlled repetition */
#include <stdio.h>

/* function main begins program execution */
int main ()
{
    int counter ; /* number of grade to be entered next */
    int grade ; /* grade value */
    int total ; /* sum of grades input by user */
    int average ; /* average of grades */

    /* initialization phase */
    total = 0 ; /* initialize total */
    counter = 1 ; /* initialize loop counter */

    /* processing phase */
    while (counter <= 10) { /* loop 10 times */
        printf ( "Enter grade : " ) ; /* prompt for input */
        scanf ( "%d", &grade ) ; /* read grade from user */
        total = total + grade ; /* add grade to total */
        counter = counter + 1 ; /* increment counter */
    } /* end while */

    /* termination phase */
    average = total / 10 ; /* integer division */

    printf ( "Class average is %d\n", average ) ; /* display result */

    return 0 ; /* indicate program ended successfully */

} /* end function main */

```

```

Enter grade : 98
Enter grade : 76
Enter grade : 71
Enter grade : 87
Enter grade : 83
Enter grade : 90
Enter grade : 57
Enter grade : 79
Enter grade : 82
Enter grade : 94
Class average is 81

```

ตัวอย่างโปรแกรมที่ 6.4 จะเป็นโปรแกรมที่คำนวณรายจ่ายของบริษัทแห่งหนึ่ง โดยให้ผู้ใช้ป้อนจำนวนคนเข้ามา

```

/* Compute the payroll for a company */

#include <stdio.h>

int main (void)
{
    double  total_pay ;      /* company payroll      */
    int     count_emp ;      /* current employee   */
    int     number_emp ;     /* number of employee  */
    double  hours ;          /* hours worked        */
    double  rate ;           /* hourly rate         */
    double  pay ;            /* pay for this period */

    /* Get number of employees. */
    printf ( "Enter number of employees >" );
    scanf ( "%d", &number_emp );

    /* Compute each employee's pay and add it to the payroll. */
    total_pay = 0.0 ;
    count_emp = 0 ;
    while (count_emp < number_emp) {
        printf ( "Hours >" );
        scanf ( "%lf", &hours );
        printf ( "Rate > $" );
        scanf ( "%lf", &rate );
        pay = hours * rate ;
        printf ( "Pay is $%6.2f\n", pay );
        total_pay = total_pay + pay ;          /* Add next pay. */
        count_emp = count_emp + 1 ;
    }
    printf ( "All employees processed\n" );
    printf ( "Total payroll is $%8.2f\n", total_pay );

    return (0) ;
}

```

```

Enter number of employee> 3
Hours>50
Rate  >$5.25
Pay is  $262.50

```

```

Hours>6
Rate  >$5.00
Pay is  $30.00

```

```

Hours>15
Rate  >$7.00
Pay is  $105.00

```

```

All employees processed
Total payroll is $ 397.50

```

จากโปรแกรม สามารถแสดงการติดตาม (trace) ของการวนซ้ำ จำนวน 3 ครั้ง ได้ดังนี้

Statement	hours	rate	pay	total_pay	count_emp	Effect
	?	?	?	0.0	0	
count_emp < number_emp						true
scanf (“%lf”, &hours);	50.0					get hours
scanf (“%lf”, &rate);		5.25				get rate
pay = hours * rate ;			262.5			find pay
total_pay = total_pay				262.5		add to
+ pay ;						total_pay
count_emp = count_emp					1	increment
+ 1 ;						count_emp
count_emp < number_emp						true
scanf (“%lf”, &hours);	6.0					get hours
scanf (“%lf”, &rate);		5.0				get rate
pay = hours * rate ;			30.0			find pay
total_pay = total_pay				292.5		add to
+ pay ;						total_pay
count_emp = count_emp					2	increment
+ 1 ;						count_emp
count_emp < number_emp						true
scanf (“%lf”, &hours);	15.0					get hours
scanf (“%lf”, &rate);		7.0				get rate
pay = hours * rate ;			105.0			find pay
total_pay = total_pay				397.5		add pay to
+ pay ;						total_pay
count_emp = count_emp					3	increment
+ 1 ;						count_emp

ตัวอย่างโปรแกรมที่ 6.5 เป็นโปรแกรมคำนวณหาค่าของ ห.ร.ม. ของเลขจำนวนเต็มบวก 2 จำนวน

```
/* This program finds the Greatest Common Divisor
   of two nonnegative integer values */
#include <stdio.h>

main ()
{
    int u, v, temp ;

    printf ( "Please type in two nonnegative integers. \n" ) ;
    scanf ( "%d%d", &u, &v ) ;

    while ( v != 0 )
    {
        temp = u % v ;
        u = v ;
        v = temp ;
    }
    printf ( "Their Greatest Common Division is %d\n", u ) ;
}
```

Please type in two nonnegative integers.

150 35

Their Greatest Common Divisor is 5

Please type in two nonnegative integers.

1026 405

Their Greatest Common Divisor is 27

ตัวอย่างโปรแกรมที่ 6.6 เป็นโปรแกรมที่สลับหลักตัวเลขที่ถูกป้อนเข้ามา

```
/* Program to reverse the digits of a number */
#include <stdio.h>

main ()
{
    int number, right_digit ;

    printf ( "Enter your number. \n" ) ;
    scanf ( "%d", &number ) ;

    while (number != 0) ;
    {
        right_digit = number % 10 ;
        printf ( "%d", right_digit ) ;
        number = number / 10 ;
    }
    printf ( "\n" ) ;
}
```

```
Enter your number.
13579
97531
```

ตัวอย่างโปรแกรมที่ 6.7 เป็นโปรแกรมที่เกิดการวนซ้ำไม่รู้จบ

```
#include <stdio.h>
int main (void)
{
    int n = 0 ;

    while (n < 3)
        printf ( "n is %d\n", n ) ;
        n ++ ;
    printf ( "That's all this program does\n" ) ;
    return 0 ;
}
```

```
n is 0
n is 0
n is 0
n is 0
!
```


ตัวอย่างโปรแกรมที่ 6.9 เป็นโปรแกรมที่อ่านคะแนนเข้ามาโดยมีการใช้คำสั่ง `continue` และตัวดำเนินการเงื่อนไข (?:)

```
#include <stdio.h>
#define MIN 0.0
#define MAX 100.0
int main (void)
{
    float score ;
    float total = 0.0 ;
    int n = 0 ;
    float min = MAX ;
    float max = MIN ;

    printf ( "Enter the scores : \n" ) ;
    while (scanf ( "%f", &score ) == 1)
    {
        if (score < MIN || score > MAX)
        {
            printf ( "%0.1f is an invalid value. \n", score ) ;
            continue ;
        }
        printf ( "Accepting %0.1f : \n", score ) ;
        min = (score < min) ? score : min ;
        max = (score > max) ? score : max ;
        total += score ;
        n ++ ;
    }
    if (n > 0)
    {
        printf ( "Average of %d score is %0.1f. \n", n, total / n ) ;
        printf ( "Low = %0.1f, high = %0.1f\n", min, max ) ;
    }
    else
        printf ( "No valid scores were entered. \n" ) ;
    return 0 ;
}
```

โปรแกรมนี้อ่านข้อมูลเข้ามา ถ้ามีค่าน้อยกว่า 0 หรือมากกว่า 100 ก็จะมีข้อความแจ้งเตือน และให้กลับขึ้นป้อนคะแนนเข้ามาใหม่ โดยอาศัยข้อความสั่ง `continue` ถ้าคะแนนอยู่ในช่วงตั้งแต่ 0 ถึง 100 ก็จะมีการหาผลรวมคะแนนทุกจำนวนรวมทั้งหมด หาคะแนนสูงสุดและต่ำสุด เมื่อสิ้นสุดโปรแกรมก็จะแสดงค่าเฉลี่ย, คะแนนสูงสุดและคะแนนต่ำสุด

6.1.1 การใช้เซ็นทินัล (sentinel) ในการควบคุมการวนซ้ำ

ในหลาย ๆ โปรแกรมที่เกี่ยวข้องกับการวนซ้ำ บางครั้งอาจจะมีการรับข้อมูลมาทางแป้นพิมพ์ หรืออ่านจากแฟ้มข้อมูล และเราไม่ทราบจำนวนข้อมูลที่แน่นอน ดังนั้น เราจะหาทางที่จะส่งสัญญาณบอกให้โปรแกรมหยุดการอ่านข้อมูล วิธีการหนึ่งที่ได้ก็คือ จะต้องมียุคที่เรียกว่า เซ็นทินัล (sentinel) นั่นคือ เงื่อนไขการทำซ้ำแบบวนซ้ำ จะตรวจสอบค่าแต่ละจำนวนที่ถูกป้อนเข้ามาว่าตรงกับค่าเซ็นทินัลหรือไม่ ถ้าตรงกับค่าเซ็นทินัลก็จะออกจากวนซ้ำ (loop) ถ้าไม่ตรงก็จะอ่านข้อมูลเข้ามา

ตัวอย่างโปรแกรมที่ 6.10 เป็นโปรแกรมคำนวณหาผลรวมคะแนนของนักศึกษาที่อ่านเข้ามาทางแป้นพิมพ์ โดยการใช้ค่า sentinel โดยกำหนดให้ตัวแปร sum เป็นผลรวมของคะแนนนักศึกษาทุกคน โดยกำหนดค่าเริ่มต้นเป็นศูนย์ และ score เป็นคะแนนของนักศึกษาแต่ละคนที่ถูกอ่านเข้ามาทางแป้นพิมพ์

```
/* Compute the sum of a list of exam scores. */

#include <stdio.h>

#define SENTINEL -99

int main (void)
{
    int sum = 0,          /* output – sum of score input so far */
        score ;          /* input – current score */

    /* Accumulate sum of all score. */
    printf ( "Enter first score (or %d to quit) >", SENTINEL ) ;
    scanf ( "%d", &score ) ; /* Get first score. */
    while (score != SENTINEL)
    {
        sum += score ;
        printf ( "Enter next score (%d to quit) >", SENTINEL ) ;
        scanf ( "%d", &score ) ; /* Get next score. */
    }
    printf ( "\nSum of exam scores is %d\n", sum ) ;

    return (0) ;
}
```

```
Enter first score (or -99 to quit) > 55
Enter next score (-99 to quit) > 33
Enter next score (-99 to quit) > 77
Enter next score (-99 to quit) > -99

Sum of exam scores is 165
```

การทำงานของโปรแกรมนี้ ก่อนอื่นจะต้องมีการกำหนดค่า sentinel ให้มีค่าเท่ากับ -99 (อาจจะเป็นตัวเลขอื่นก็ได้เช่น -9 หรือ -1) และต้องมีการอ่านข้อมูลของนักศึกษาคนแรกก่อนเข้าวนซ้ำ (loop) ถ้าค่าที่อ่านเข้ามาไม่เท่ากับค่า sentinel ก็จะนำคะแนนของนักศึกษาแต่ละคนรวมกันไว้ในตัวแปร sum โดยใช้ข้อความสั่ง `sum += score` ซึ่งมีความหมายเหมือนกับ `sum = sum + score`

หลังจากนั้นก็จะอ่านคะแนนของนักศึกษาคนถัดไป ถ้าคะแนนของนักศึกษาไม่เท่ากับค่า sentinel ก็จะยังไม่ออกจากวนซ้ำ (loop) ถ้าเท่ากับค่า sentinel ก็จะออกจากวนซ้ำ จากตัวอย่างนี้จะเป็นคะแนนนักศึกษาเข้ามาจำนวน 3 คน ถ้าต้องการสิ้นสุดก็ป้อนค่า sentinel

จากตัวอย่างนี้ เราสามารถนำค่า sentinel ไปใช้ในข้อความสั่ง for ได้ดังนี้

```
/* Accumulate sum of all scores.
printf ( "Enter first core (or %d to quit) >", SENTINEL ) ;
for ( scanf ( "%d", &score ) ;
    score != SENTINEL ;
    scanf ( "%d", &score ) ) {
    sum += score ;
    printf ( "Enter next score (%d to quit) >", SENTINEL ) ;
}
```

(ข้อความสั่ง for จะได้เรียนในหัวข้อถัดไป)

ตัวอย่างโปรแกรมที่ 6.11 เป็นโปรแกรมที่ให้ผู้ใช้ป้อนข้อมูลแบบอักขระเข้ามา

```
#include <stdio.h>

main ()
{
    char answer = 'Y';          /* Response of program user. */

    while (answer != 'N')
    {
        printf ( "This is the body of the program. \n" );
        printf ( "Do you want to repeat this program (Y/N) = >" );
        answer = getche ();
        printf ( "\n" );
    } /* End of while. */

    printf ( "Thanks for using the program." );
}
```

```
This is the body of the program.
Do you want to repeat this program (Y/N) = > Y
This is the body of the program.
Do you want to repeat this program (Y/N) = > n
This is the body of the program.
Do you want to repeat this program (Y/N) = > N
Thanks for using the program.
```

โปรแกรมนี้ จะกำหนดค่าเริ่มต้นให้กับตัวแปร `char answer = 'Y'` ; เพื่อให้แน่ใจว่าไม่เท่ากับค่า sentinel คือ 'N' จึงจะเข้ามวนซ้ำ (loop) ได้ โดยจะตรวจสอบกับเงื่อนไข `while (answer != 'N')` หลังจากเงื่อนไขนี้เป็นจริง ก็จะกระทำการ block นี้ทั้งหมด คือ

```
{
    printf ( "This is the body of the program. \n" );
    printf ( "Do you want to repeat this program (Y/N) = >" );
    answer = getche ();
    printf ( "\n" );
} /* End of while. */
```

ซึ่งภายใน block นี้ มีการเรียกใช้ฟังก์ชัน `getche()` ซึ่งเป็นฟังก์ชันที่รับค่าตัวอักขระเข้ามา 1 ตัว และแสดงตัวอักขระบนจอภาพ โดยผู้ใช้ไม่ต้องกด Enter โดยเคอร์เซอร์ยังคงอยู่ที่บรรทัดเดิม

6.2 รูปแบบทั่วไปของข้อความสั่ง for แสดงได้ดังรูปที่ 6.6

```
for (InitializationExpression ; LoopControlExpression ; UpdateExpression)
    LoopBody
/* end for */
```

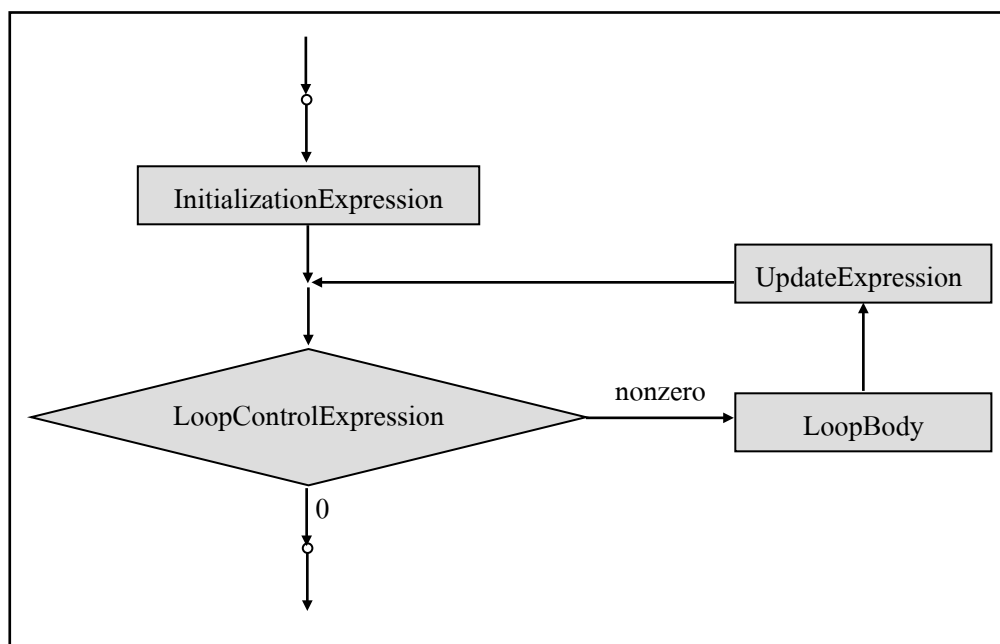
รูปที่ 6.6

โดย InitializationExpression จะเป็นนิพจน์มีการกำหนดค่าเริ่มต้นให้ตัวแปรควบคุมการวนซ้ำ (loop control variable) และ UpdateExpression เป็นค่าที่ถูกเปลี่ยนแปลงค่า

ส่วน LoopControlExpression จะเป็นนิพจน์ที่ถูกคำนวณจนกระทั่งเป็นศูนย์ (เป็นเท็จ) หรืออาจจะไม่ใช่เลขศูนย์ (เป็นจริง)

และ LoopBody อาจจะเป็น 1 ข้อความสั่ง หรือหลายข้อความสั่งก็ได้ ถ้ามีมากกว่า 1 ข้อความสั่ง ให้ปิดล้อมด้วยเครื่องหมาย { และ }

สามารถแสดงผังงานได้ดังรูปที่ 6.7



รูปที่ 6.7

จากผังงานนี้ จะมีขั้นตอนการกระทำดังนี้

- 1) กระทำที่ InitializationExpression ก่อน
- 2) คำนวณค่าของ LoopControlExpression ถ้ามีค่าเป็นศูนย์ก็จะออกจากวนซ้ำ (loop)
- 3) ถ้า LoopControlExpression มีค่าไม่เป็นศูนย์ LoopBody จะถูกกระทำ และ UpdateExpression ก็จะถูกคำนวณ
- 4) ทดสอบ LoopControlExpression อีกครั้ง โดย LoopBody จะถูกทำซ้ำจนกระทั่ง LoopControlExpression ถูกคำนวณจนมีค่าเป็นศูนย์

รูปแบบทั่วไปของข้อความสั่ง for กับ while มีความสมมูลกัน ซึ่งสามารถเขียนใช้แทนกันได้ ดังนี้

```
for (InitializationExpression ; LoopControlExpression ; UpdateExpression)
    LoopBody

/* end for */
```

จะมีความหมายเหมือนกับ

```
IntializationExpression

while (LoopControlExpression) {
    LoopBody
    UpdateExpression ;
} /* end while */
```

ตัวอย่างโปรแกรมที่ 6.12 จะเป็นโปรแกรมที่เขียนโดยใช้ข้อความสั่ง while จะมีความหมายเหมือนกับข้อความสั่ง for

กำหนด int number, factorial, counter ;

```
counter = 1 ;
factorial = 1 ;

while (counter <= number) {
    factorial = factorial * counter ;
    counter = counter + 1 ;
} /* end while */
```

จะมีความหมายเหมือนกับ

```
factorial = 1 ;

for (counter = 1 ; counter <= number ; counter = counter + 1)
    factorial = factorial * counter ;
/* end for */
```

ตัวอย่างโปรแกรมที่ 6.13 จะเป็นโปรแกรมแสดงการใช้ข้อความสั่ง for

```
int number, sum, counter ;
.....
.....
for (counter = 1 ; counter <= number ; counter = counter + 1)
    sum = sum + counter ;
/* end for */
```

ในที่นี้ตัวแปรจำนวนเต็ม counter จะเป็นตัวแปรควบคุมวนซ้ำ โดยกำหนดค่าเริ่มต้นเป็น 1 โดยนิพจน์ counter <= number นี้จะถูกทดสอบก่อน ถ้าเงื่อนไขเป็นจริงแล้วข้อความสั่ง sum = sum + counter ; นี้จะถูกกระทำการ มิฉะนั้นแล้วก็จะออกจากวนซ้ำ (loop) หลังจากนั้นก็จะมีการปรับเปลี่ยนค่าของ counter โดยใช้นิพจน์ counter = counter + 1 นี้ เพิ่มค่า counter ขึ้นอีก 1 เพื่อให้เข้าใจการทำงานของข้อความสั่ง for จะขออธิบายในตารางของการติดตาม โดยจะกำหนดให้ค่าปัจจุบันของ number มีค่าเท่ากับ 3 และจะแสดงค่าตัวแปร sum และ counter ในแต่ละขั้นตอนดังนี้

ขั้นตอนที่	นิพจน์ที่ถูกคำนวณ	sum	counter
1	sum = 0	0	-
2	counter = 1	0	1
3	counter <= number (1 ≤ 3 เป็นจริง)	0	1
4	sum = sum + counter	1	1
5	counter = counter + 1	1	2
6	counter <= number (2 ≤ 3 เป็นจริง)	1	2
7	sum = sum + counter	3	2
8	counter = counter + 1	3	3
9	counter <= number (3 ≤ 3 เป็นจริง)	3	3
10	sum = sum + counter	6	3
11	counter = counter + 1	6	4
12	counter <= number (4 ≤ 3 เป็นเท็จ) และออกจากวนซ้ำ (loop)	6	4

ตัวอย่างโปรแกรมที่ 6.14 เป็นโปรแกรมที่แสดงการพิมพ์ข้อความ จำนวน 5 ครั้ง

```
#include <stdio.h>

main ()
{
    int time;          /* Counter variable. */

    /* The loop starts here. */

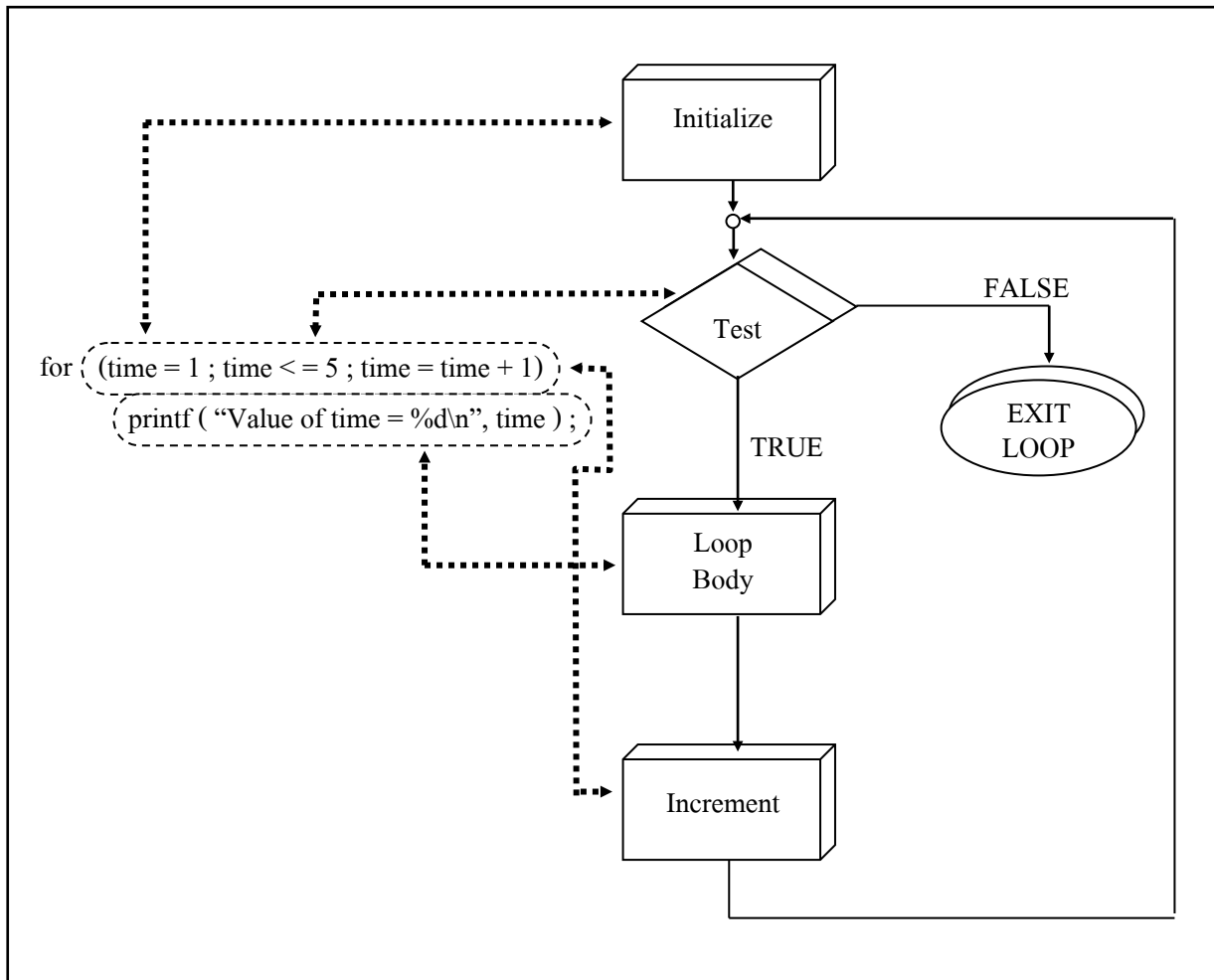
    for (time = 1 ; time <= 5 ; time = time + 1)
        printf ( "Value of time = %d\n", time ) ;

    /* The loop stops here. */

    printf ( "This is the end of the loop. " ) ;
}
```

```
Value of time = 1
Value of time = 2
Value of time = 3
Value of time = 4
Value of time = 5
This is the end of the loop.
```

และสามารถนำมาแสดงผังงานได้ดังรูปที่ 6.8



รูปที่ 6.8

ลักษณะเพิ่มเติมของข้อความสั่ง for

- 1) ส่วนของ UpdateExpression เพิ่มขึ้นครั้งละ 1 โดยมีค่าเริ่มต้นเท่ากับ 1

for (i = 1 ; i <= 100 ; i++)

- 2) ส่วนของ UpdateExpression เพิ่มขึ้นครั้งละ -1 (ลดค่าครั้งละ 1) โดยมีค่าเริ่มต้นเท่ากับ 100

for (i = 100 ; i >= 1 ; i--)

- 3) ส่วนของ UpdateExpression เพิ่มขึ้นครั้งละ 7 โดยมีค่าเริ่มต้นเท่ากับ 7

for (i = 7 ; i <= 77 ; i += 7)

ตัวอย่างโปรแกรมที่ 6.15 แสดงโปรแกรมที่มีส่วนของ UpdateExpression มีค่าลดลงครั้งละ 1

```
#include <stdio.h>
int main (void)
{
    int secs ;

    for (secs = 5 ; secs > 0 ; secs --)
        printf ( "%d seconds ! \n", secs ) ;
    printf ( "We have ignition ! \n" ) ;
    return 0 ;
}
```

```
5 seconds !
4 seconds !
3 seconds !
2 seconds !
1 seconds !
We have ignition !
```

ตัวอย่างโปรแกรมที่ 6.16 แสดงโปรแกรมที่มีส่วนของ UpdateExpression เป็นนิพจน์เลขคณิต เพิ่มขึ้นครั้งละ 13

```
#include <stdio.h>
int main (void)
{
    int n ;          /* count by 13s */

    for (n = 2 ; n < 60 ; n = n + 13)
        printf ( "%d \n", n ) ;
    return 0 ;
}
```

```
2
15
28
41
54
```

- 4) ส่วนของค่าเริ่มต้นเป็นตัวอักษรก็ได้

ตัวอย่างโปรแกรมที่ 6.17 เป็นโปรแกรมแสดงค่าของรหัส ASCII โดยมีค่าเริ่มต้น = 'a'

```
#include <stdio.h>
int main (void)
{
    char ch ;

    for (ch = 'a' ; ch <= 'z' ; ch ++ )
        printf ( "The ASCII value for %c is %d. \n", ch, ch ) ;
    return 0 ;
}
```

```
The ASCII value for a is 97.
The ASCII value for b is 98.
...
The ASCII value for x is 120.
The ASCII value for y is 121.
The ASCII value for z is 122.
```

- 5) ส่วนของ LoopControlExpression อาจจะเป็นเงื่อนไขในการทดสอบแทนที่จะเป็นจำนวนทำซ้ำ เช่น

```
เราจะแทน   for (num 1 ; num <= 6 ; num ++ )
ด้วย       for (num = 1 ; num * num * num <= 216 ; num ++ )
```

- 6) ส่วนของ UpdateExpression อาจจะเป็นนิพจน์ที่มีตัวดำเนินการ *

ตัวอย่างโปรแกรมที่ 6.18

```
#include <stdio.h>
int main (void)
{
    float debt ;

    for (debt = 100.0 ; debt < 150.0 ; debt = debt * 1.1)
        printf ( "Your debt is now $%.2f . \n", debt ) ;
    return 0 ;
}
```

```
Your debt is now $100.00.
Your debt is now $110.00.
Your debt is now $121.00.
Your debt is now $133.10.
Your debt is now $146.41.
```

7) ส่วนของ UpdateExpression อาจจะเป็นนิพจน์ที่ซับซ้อน

ตัวอย่างโปรแกรมที่ 6.19

```
#include <stdio.h>
int main (void)
{
    int x ;
    int y = 55 ;

    for (x = 1 ; y <= 75 ; y = (++ x * 5) + 50)
        printf ( "%10d %10d\n", x, y ) ;

    return 0 ;
}
```

1	55
2	60
3	65
4	70
5	75

8) ส่วนของ InitializationExpression, LoopControlExpression หรือ UpdateExpression อาจจะสามารถใส่ไว้ก็ได้ แต่จะต้องมีเครื่องหมาย semicolon (;) คั่นไว้ได้ ถ้าไม่มีทั้ง 3 ส่วน จะเป็นการวนซ้ำที่ไม่รู้จบ เช่น for (; ;)

ตัวอย่างโปรแกรมที่ 6.20 เป็นโปรแกรมที่ส่วนของ UpdateExpression ไม่ได้ใส่ไว้

```
#include <stdio.h>
int main (void)
{
    int ans, n ;

    ans = 2 ;
    for (n = 3 ; ans <= 25 ; )
        ans = ans * n ;
    printf ( "n = %d ; ans = %d.\n", n, ans ) ;
    return 0 ;
}
```

n = 3 ; ans = 54.

- 9) ส่วนของ InitializationExpression ไม่จำเป็นต้องเป็นตัวแปร แต่อาจจะเป็นข้อความสั่ง printf ก็ได้ แต่ต้องจำไว้ว่า ส่วนนี้จะทำเพียงครั้งแรกเพียงครั้งเดียว เท่านั้น

ตัวอย่างโปรแกรมที่ 6.21

```
#include <stdio.h>
int main (void)
{
    int num ;

    for (printf ( "Keep entering number ! \n" ) ; num != 6 ; )
        scanf ( "%d", &num ) ;
    printf ( "That's the one I want ! \n" ) ;
    return 0 ;
}
```

```
Keep entering numbers !
3
5
8
6
That's the on I want !
```

- 10) ส่วนของ LoopControlExpression อาจจะมีการเปลี่ยนแปลงค่าได้ภายในวนซ้ำ (loop) เช่น

```
for (n = 1 ; n <= 10000 ; n = n + delta)
```

ตัวแปร n มีการเปลี่ยนแปลงค่าโดยตัวแปร delta

6.2.1 ตัวดำเนินการเครื่องหมายจุลภาค (Comma opertor)

ในข้อความสั่ง for ในส่วนของ InitializationExpression, LoopControlExpression และ UpdateExpression อาจจะมีนิพจน์เลขคณิตมากกว่า 1 นิพจน์ อยู่ในแต่ละส่วนในข้อความสั่ง for ก็ได้ ถ้ามีก็ให้เขียนแยกกันโดยใช้เครื่องหมายจุลภาค (,) และลำดับในการทำงานจะเรียงจากซ้ายมือไปยังขวามือ

ตัวอย่างโปรแกรมที่ 6.22 จะเป็นโปรแกรมที่มีการใช้ตัวดำเนินการเครื่องหมายจุดภาคในส่วนของการ InitializationExpression และ UpdateExpression

```
#include <stdio.h>
#define FIRST 29
#define NEXT 23
int main (void)
{
    int ounces, cost ;

    printf ( "ounces cost\n" );
    for (ounces = 1, cost = FIRST ; ounces <= 16 ; ounces ++ , cost += NEXT)
        printf ( "%5d %7d\n", ounces, cost );
    return 0 ;
}
```

ounces	cost
1	29
2	52
3	75
!	!

ตัวอย่างโปรแกรมที่ 6.23 จะเป็นโปรแกรมหาผลบวก $2 + 4 + \dots + 100$

```
/* Summation with for */
#include <stdio.h>

/* function main begins program execution */
int main ()
{
    int sum = 0 ; /* initialize sum */
    int number ; /* number to be added to sum */

    for (number = 2 ; number <= 100 ; number += 2) {
        sum += number ; /* add number to sum */
    } /* end for */

    printf ( "Sum is %d\n", sum ) ; /* output sum */

    return 0 ; /* indicate program ended successfully */

} /* end function main */
```

Sum is 2550

ข้อสังเกต ส่วนของลำตัวของข้อความสั่ง for สามารถเขียนเป็นข้อความสั่ง ได้ดังนี้

```
for (number = 2 ; number <= 100 ; sum += number, number += 2) ;
```

```
/* empty statement */
```

ซึ่งเราจะเรียกว่า ตัวดำเนินการคอมม่า (comma operator)

ตัวอย่างโปรแกรมที่ 6.24 เป็นโปรแกรมที่แสดงการใช้ตัวดำเนินการเพิ่มก่อนและหลังครั้งละ 1

```
#include <stdio.h>
```

```
main ()
```

```
{
```

```
    int count ;                /* Program counter.          */
```

```
    int number1 = 0 ;          /* Number to post-increment. */
```

```
    int number2 = 0 ;          /* Number to pre-increment.  */
```

```
    for (count = 1 ; count <= 5 ; count ++)
```

```
    {
```

```
        printf ( "Post-incrementing number = %d", number1 ++ ) ;
```

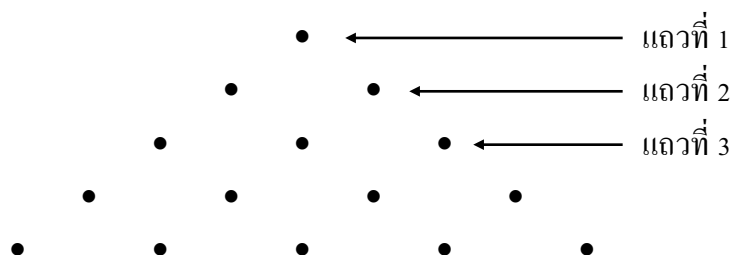
```
        printf ( "Pre-incrementing number = %d\n", ++ number2 ) ;
```

```
    }
```

```
}
```

Post-incrementing number = 0	Pre-incrementing number = 1
Post-incrementing number = 1	Pre-incrementing number = 2
Post-incrementing number = 2	Pre-incrementing number = 3
Post-incrementing number = 3	Pre-incrementing number = 4
Post-incrementing number = 4	Pre-incrementing number = 5

ตัวอย่างโปรแกรมที่ 6.25 เป็นโปรแกรมที่แสดงจำนวนสามเหลี่ยม เช่น



แถวที่ 1 จะมีจำนวน 1 จุด

แถวที่ 2 จะมีจำนวน 2 จุด รวม 2 แถว มีทั้งหมด $1 + 2 = 3$ จุด

แถวที่ 3 จะมีจำนวน 3 จุด รวม 3 แถว มีทั้งหมด $1 + 2 + 3 = 6$ จุด

ตัวเลข $1 + 2 + 3$ เราจะเรียกว่า จำนวนสามเหลี่ยม

```
/* Program to generate a table of triangular number */
#include <stdio.h>
```

```
main ()
```

```
{
```

```
    int n, triangular_number ;
```

```
    printf( "TABLE OF TRAINGULAR NUMBERS\n\n" );
```

```
    printf( " n   Sum from 1 to n\n" );
```

```
    printf( " ---   ----- \n" );
```

```
    triangular_number = 0 ;
```

```
    for ( n = 1 ; n <= 10 ; ++n )
```

```
    {
```

```
        triangular_number = triangular_number + n ;
```

```
        printf ( "%d           %d\n", n, triangular_number ) ;
```

```
    }
```

```
}
```

n	Sum from 1 to n
---	-----
1	1
2	3
3	6
4	10
5	15
6	21
7	28
8	36
9	45
10	55

ตัวอย่างโปรแกรมที่ 6.26 เป็นโปรแกรมที่แสดงลำดับ Fibonacci เช่น 1, 1, 2, 3, 5 ... โดยจะให้ผู้ใช้ป้อนจำนวนพจน์ที่ต้องการแสดง

```
#include <stdio.h>

main ()
{
    int n ;           /* User input number.          */
    int old, new ;    /* Fibonacci sequence variables. */
    int temp ;        /* Temporary storage variable.   */
    int j ;           /* For-loop counter.             */

    printf ( "Enter number of terms to generate = >" );
    scanf ( "%i", &n );
    old = 0 ;
    new = 1 ;
    printf ( "%4i %4i", old, new );
    for ( j = 1 ; j <= n - 2 ; j ++ )
    {
        temp = old + new ;
        old = new ;
        new = temp ;
        printf ( "%4i", new );
    }
}
```

Enter number of terms to generate = > 12 0 1 1 2 3 5 8 13 21 34 55 89
--

ข้อความสั่ง break จะทำให้การกระทำหยุดออกจากวงรอบวนซ้ำ

ข้อความสั่ง continue จะทำให้การกระทำกลับไปเริ่มต้นที่วงรอบวนซ้ำอีก

ตัวอย่างโปรแกรมที่ 6.27 เป็นโปรแกรมที่มีการนำข้อความสั่ง break มาใช้

```

/* Using the break statement in a for statement */
#include <stdio.h>

/* function main begins program execution */
int main ()
{
    int x ; /* counter */

    /* loop 10 times */
    for (x = 1 ; <= 10 ; x ++ ) {

        /* if x is 5, terminate loop */
        if (x == 5) {
            break ; /* break loop only if x is 5 */
        } /* end if */

        printf ( "%d", x) ; /* display value of x */
    } /* end for */

    printf ( "\nBroke out of loop at x == %d\n", x ) ;

    return 0 ; /* indicate program ended successfully */

} /* end function main */

```

```

1 2 3 4
Broke out of loop at x == 5

```

ตัวอย่างโปรแกรมที่ 6.28 เป็นโปรแกรมที่มีการนำข้อความสั่ง continue มาใช้

```

/* Using the continue statement in a for statement */
#include <stdio.h>

/* function main begins program execution */
int main ()
{
    int x ; /* counter */

    /* loop 10 times */
    for (x = 1 ; x <= 10 ; x ++ ) {

        /* if x is 5, continue with next iteration of loop */
        if (x == 5) {
            continue ; /* skip remaining code in loop body */
        } /* end if */

        printf ( "%d", x) ; /* display value of x */
    } /* end for */

    printf ( "\nUsed continue to skip printing the value 5\n" ) ;

    return 0 ; /* indicate program ended successfully */

} /* end function main */

```

```

1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5

```

6.3 รูปแบบทั่วไปของข้อความสั่งแบบ do-while แสดงได้ดังรูปที่ 6.9

```

do
    LoopBody
while (LoopControlExpression) ;
/* end do-while */

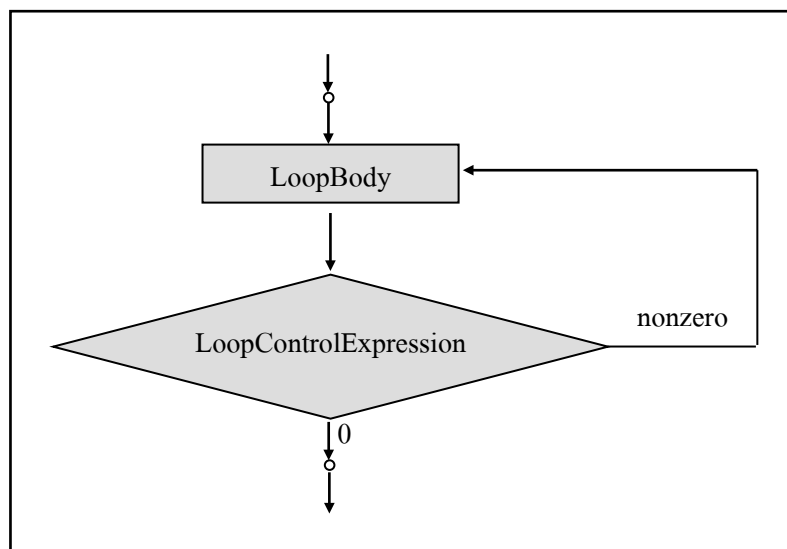
```

รูปที่ 6.9

โดย LoopControlExpression เป็นนิพจน์เลขคณิตที่มีค่าเป็นศูนย์ (เป็นเท็จ) หรือมีค่าไม่ใช่ศูนย์ (เป็นจริง)

ส่วน LoopBody อาจจะเป็น 1 ข้อความสั่งหรือมากกว่า 1 ข้อความสั่งก็ได้ ถ้ามีมากกว่า 1 ข้อความสั่ง ต้องปิดล้อมด้วย เครื่องหมาย { และ }

สามารถได้เป็นผังงานได้ ดังรูป 6.10



รูปที่ 6.10

จากผังงานนี้ จะมีขั้นตอนการกระทำดังนี้

- 1) กระทำการที่ LoopBody ก่อน
- 2) คำนวณค่าของ LoopControlExpression ถ้ามีค่าเป็นศูนย์ (เป็นเท็จ) ก็จะออกจากรอบวนซ้ำ
ถ้าค่าไม่ใช่ศูนย์ (เป็นเท็จ) แล้ว LoopBody จะทำอีกครั้ง

หมายเหตุ ข้อความสั่ง do-while จะต้องมีการกระทำอย่างน้อย 1 ครั้ง

ตัวอย่างโปรแกรมที่ 6.29 จากตัวอย่างโปรแกรมที่ 6.13 จะแสดงขั้นตอนการทำงานของข้อความสั่งแบบ do-while โดยจะกำหนดค่าตัวแปร number มีค่าเท่ากับ 3

ขั้นตอนที่	นิพจน์ที่ถูกคำนวณ	sum	counter
1	sum = 0	0	-
2	sum = 0	0	0
3	sum = sum + counter ;	0	0
4	counter = counter + 1 ;	0	1
5	counter <= number 1 <= 3 เป็นจริง	0	1
6	sum = sum + counter ;	1	1
7	counter = counter + 1 ;	1	2
8	counter <= number 2 <= 3 เป็นจริง	1	2
9	sum = sum + counter ;	3	2
10	counter = counter + 1 ;	3	3
11	counter <= number 3 <= 3 เป็นจริง	3	3
12	sum = sum + counter ;	6	3
13	counter = counter + 1 ;	6	4
14	counter <= number 4 <= 3 เป็นเท็จ และออกจากวนซ้ำ (loop)	6	4

ตัวอย่างโปรแกรมที่ 6.30 เป็นโปรแกรมแสดงตัวเลข 1 ถึง 5

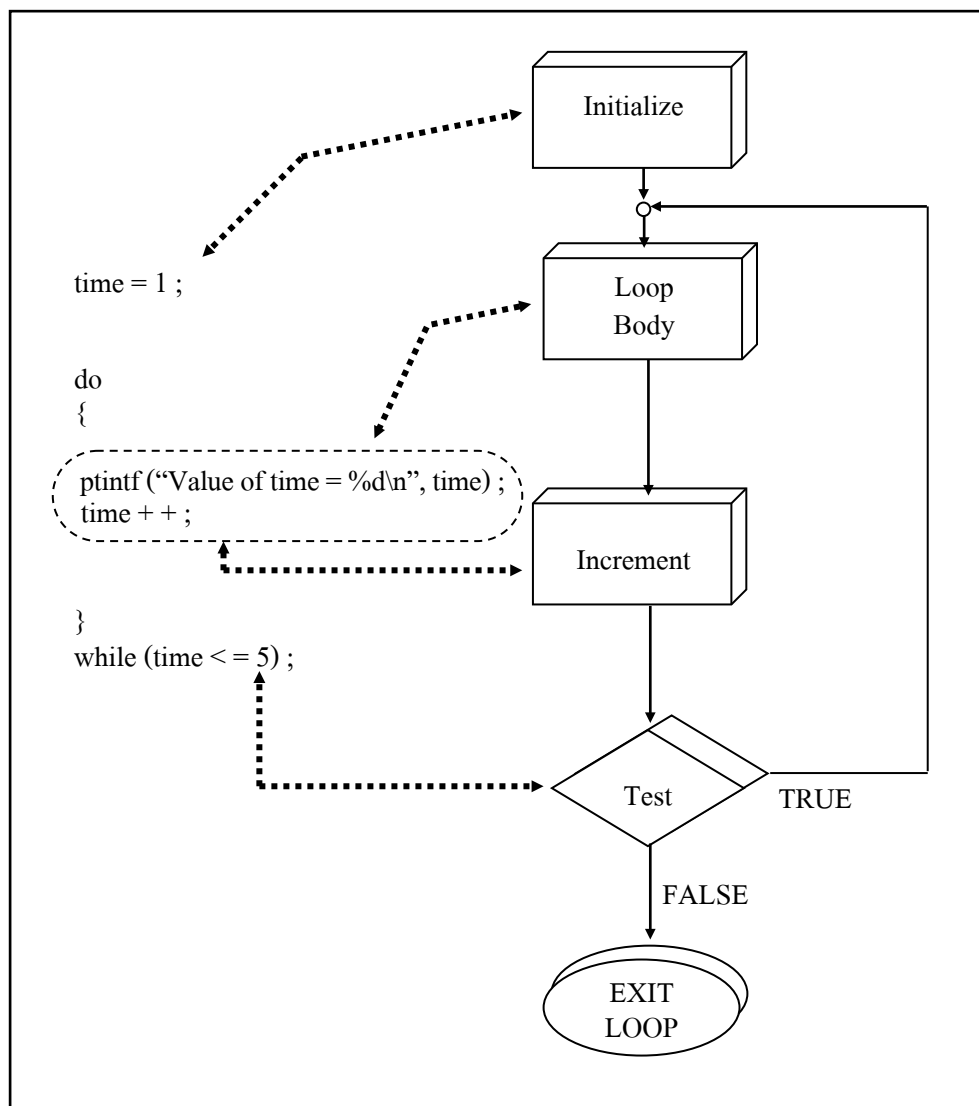
```
#include <stdio.h>

main ()
{
    int time ;          /* Counter variable. */
    time = 1 ;
    do {
        printf ( "Value of time = %d\n", time) ;
        time ++ ;
    } while (time <= 5) ;

    printf ( "End of the loop." ) ;
}
```

```
Value of time = 1
Value of time = 2
Value of time = 3
Value of time = 4
Value of time = 5
End of the loop.
```


สามารถแสดงผังงานได้ดังรูปที่ 6.11



รูปที่ 6.11

ตัวอย่างโปรแกรมที่ 6.31 เป็นโปรแกรมที่สลับหลักตัวเลขที่ถูกป้อนเข้ามา

```
/* Program to reverse the digits of a number */
#include <stdio.h>

main ()
{
    int number, right_digit ;

    printf ( "Enter your number. \n: ) ;
    scanf ( "%d", &number. \n" ) ;

    do
    {
        right_digit = number % 10 ;
        printf ( "%d", right_digit ) ;
        number = number ! / 10 ;
    }
    while (number != 0) ;

    printf ( "\n" ) ;
}
```

Enter your number.

13579

97531

Enter your number

0

0

ตัวอย่างโปรแกรมที่ 6.32 เป็นโปรแกรมพิมพ์ตัว 1 ถึง 10

```

/* Using the do/while repetition statement */
#include <stdio.h>

/* function main begins program execution */
int main ()
{
    int counter = 1 ; /* initialize counter */

    do {
        printf ( "%d", counter ) ; /* display counter */
    } while ( ++ counter <= 10 ) ; /* end do ... while */

    return 0 ; /* indicate program ended successfully */

} /* end function main */

```

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

ตัวอย่างโปรแกรมที่ 6.33 จะเป็นโปรแกรมการคำนวณโดยให้ผู้ใช้ป้อนนิพจน์เลขคณิตที่อยู่ในรูปค่าคงตัว ตัวดำเนินการ ค่าคงตัว โดยค่าคงตัวจะเป็นจำนวนทศนิยม และตัวดำเนินการ ได้แก่ เครื่องหมาย +, -, * และ / ถ้าผู้ใช้ป้อนตัวหารเป็นเลขศูนย์ ก็จะออกจาก loop และถ้าป้อนตัวดำเนินการผิด ก็จะมีการแจ้งข่าวสารเตือน และให้ป้อนนิพจน์เลขคณิตใหม่

```

#include <stdio.h>

int main (void) {
    double operand1, operand2, result ;
    char oper, zero_division = 'n', another_expression = 'y' ;
    char newline_cahr ;

    do {
        printf ( "Type an arithmetic expression as" ) ;
        printf ( "Constant Operator Constant" ) ;
        printf ( "\nand press Enter." ) ;
        printf ( "Operator can be +, -, * or / . \n" ) ;
        scanf ( "%lf %c %lf", &operand1, &oper, &operand2 ) ;
        scanf ( "%c", &newline_char ) ;

        if (oper != '+' && oper != '-' && oper != '*' && oper != '/') {
            printf ( "Incorrect operator ! \n" ) ;
            continue ;
        } /* end if */

        if (oper == '/' && operand2 == 0.0) {
            zero_division = 'y' ;
            break ;
        } /* end if */

        switch (oper) {
            case '+' :
                result = operand1 + operand2 ;
                break ;
            case '-' :
                result = operand1 - operand2 ;
                break ;
            case '*' :
                result = operand1 * operand2 ;
                break ;
            case '/' :
                result = operand1 / operand2 ;
        } /* end switch */

        printf ( "%f %c %f = %f\n", operand1, oper, operand2, result ) ;

        printf ( "Another expression? (y/n) : " ) ;
        scanf ( "%c", &another_expression ) ;
    } while (another_expression == 'y' || another_expression == 'Y') ;
    /* end do-while */

    if (zero_division == 'y')
        printf ( "Division by zero. Program terminating ..." ) ;
    else
        printf ( "Normal program termination ..." ) ;
    /* end if */

    return 0;
/* end function main */
}

```

6.3.1 การวนซ้ำแบบซ้อน (Nested loops)

จะมีวนซ้ำอยู่ในวนซ้ำอีกชั้นหนึ่ง ซึ่งการวนซ้ำแบบซ้อนอาจจะมี 2 วนซ้ำ หรือมากกว่าก็ได้ ลักษณะขั้นตอนการทำงานของการวนซ้ำแบบซ้อนให้พิจารณาจากตัวอย่างต่อไปนี้

ตัวอย่างโปรแกรมที่ 6.34 จะแสดงการพิมพ์ตัวเลขในรูปแบบของเมทริกซ์ ขนาด 3 แถว และ 4 หลัก

```
#include <stdio.h>

#define row 4      /* Number of rows.      */
#define cols 5     /* Number of columns. */

main ()
{
    int item = 0 ;    /* Item counter.      */
    int r, c ;       /* For-loop variables. */
    for (r = 1 ; r <= rows ; r ++ )
    {
        for (c = 1 ; c <= cols ; c ++ )
        {
            printf ( "%2i  ", item ) ;
            item ++ ;
        }
        printf ( "\n" ) ;
    }
}
```

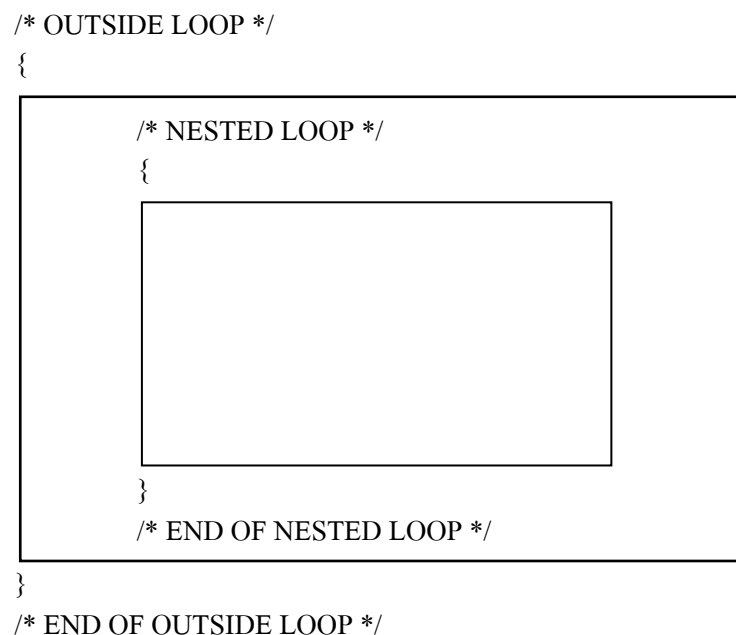
0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19

การทำงานของโปรแกรมนี้นี้ มีขั้นตอนดังนี้

- 1) เริ่ม $r = 1$
 ทดสอบเงื่อนไข $r \leq \text{rows}$
 ถ้าไม่จริง ออกจากวนซ้ำ (loop) จบโปรแกรม
 ถ้าจริง ทำขั้นตอน 2
- 2) เริ่ม $c = 1$
 ทดสอบเงื่อนไข $c \leq \text{cols}$
 ถ้าไม่จริง ออกจากวนซ้ำ (loop) ทำขั้นตอน 5
 ถ้าจริง ทำขั้นตอน 3

- 3) ทำข้อความสั่งทั้งหมดใน block คือ
 - 3.1) พิมพ์ค่า item
 - 3.2) เพิ่มค่า item ขึ้น 1
- 4) เพิ่มค่า c ขึ้น 1
 ทดสอบเงื่อนไข $c \leq \text{cols}$
 ถ้าไม่จริง ออกจากวนซ้ำ (loop) ทำขั้นตอน 5
 ถ้าจริง ก็ย้อนกลับไปทำขั้นตอน 3
- 5) ทำข้อความสั่ง `printf(“\n”);`
- 6) เพิ่มค่า r ขึ้น 1
 ทดสอบเงื่อนไข $r \leq \text{rows}$
 ถ้าไม่จริง ออกจากวนซ้ำ (loop) จบโปรแกรม
 ถ้าจริง ก็ย้อนกลับไปทำขั้นตอน 2

ซึ่งการทำงานของกรวนซ้ำแบบซ้อน จะมีลักษณะดังรูป 6.12



รูปที่ 6.12

ตัวอย่างโปรแกรมที่ 6.35 เป็นโปรแกรมแสดงการวนซ้ำแบบซ้อน 3 ชั้น

```
#include <stdio.h>

#define N 10

main ()
{
    char i, j, k      /* For-loop counters.    */
    int num = 0 ;     /* Overall loop counter. */

    for (i = 'a' ; i <= 'c' ; i++)
        for (j = 'a' ; j <= 'c' ; j++)
            for (k = 'a' ; k <= 'c' ; k++)
            {
                num++;
                printf( "%c%c%c ", i, j, k );
            }
    printf( "\nThere are %i different strings of letters. \n", num );
}
```

```
aaa aab aac aba abb abc aca acb acc baa bab bac
bba bbb bbc bca bcb bcc caa cab cac cba cbb cbc
cca ccb ccc
There are 27 different strings of letters.
```

การทำงานของโปรแกรมนี้ มีขั้นตอนดังนี้

- 1) เริ่ม $i = 'a'$
 ทดสอบเงื่อนไข $i \leq 'c'$
 ถ้าไม่จริง ทำขั้นตอน 8
 ถ้าจริง ทำขั้นตอน 2
- 2) เริ่ม $j = 'a'$
 ทดสอบเงื่อนไข $j \leq 'c'$
 ถ้าไม่จริง ออกจากวนซ้ำ (loop) ไปทำขั้นตอน 7
 ถ้าจริง ทำขั้นตอน 3

- 3) เริ่ม $k = 'a'$
 ทดสอบเงื่อนไข $k \leq 'c'$
 ถ้าไม่จริง ออกจากวนซ้ำ (loop) ไปทำขั้นตอน 6
 ถ้าจริง ทำขั้นตอน 4
- 4) ทำข้อความสั่งทั้งหมดใน block คือ
 - 4.1) เพิ่มค่า num ขึ้น 1
 - 4.2) พิมพ์ค่า i, j, k
- 5) เพิ่มค่า k ขึ้น 1
 ทดสอบเงื่อนไข $k \leq 'c'$
 ถ้าไม่จริง ก็ออกจากวนซ้ำ (loop) ไปทำขั้นตอน 6
 ถ้าจริง ก็ย้อนกลับมาทำขั้นตอน 4
- 6) เพิ่มค่า j ขึ้น 1
 ทดสอบเงื่อนไข $j \leq 'c'$
 ถ้าไม่จริง ออกจากวนซ้ำ (loop) ไปทำขั้นตอน 7
 ถ้าจริง ก็ย้อนกลับมาทำขั้นตอน 3
- 7) เพิ่มค่า i ขึ้น 1
 ทดสอบเงื่อนไข $i \leq 'c'$
 ถ้าไม่จริง ทำขั้นตอน 8
 ถ้าจริง ก็ย้อนกลับมาทำขั้นตอน 2
- 8) ทำข้อความสั่ง `printf("\nThere are %i different strings of letters. \n", num);`
- 9) จบโปรแกรม

ตัวอย่างโปรแกรมที่ 6.36 เป็นโปรแกรมแสดงจำนวนครั้งในแต่ละวนซ้ำแบบซ้อน

```
#include <stdio.h>

#define N 10

main ()
{
    int i, j, k ;          /* For-loop counters.    */
    int num = 0 ;          /* Overall loop counter. */

    for (i = 1 ; i <= N ; i ++ )
        for (j = 1 ; j <= N ; j ++ )
            num ++ ;
    printf ( "Group #1 : The innermost loop executed %i times. \n", num ) ;
    num = 0 ;
    for (i = 1 ; i <= N ; i ++ )
        for (j = 1 ; j <= i ; j ++ )
            num ++ ;
    printf ( "Group #2 : The innermost loop executed %i times. \n", num ) ;
    num = 0 ;
    for (i = 1 ; i <= N ; i ++ )
        for (j = i ; j <= N ; j ++ )
            for (k = 1 ; k <= N ; k ++ )
                num ++ ;
    printf ( "Group #3 : The innermost loop executed %i times. \n", num ) ;
    num = 0 ;
    for (i = 1 ; i <= N ; i ++ )
        for (j = 1 ; j <= N ; j ++ )
            for (k = 1 ; k <= N ; k ++ )
                num ++ ;
    printf ( "Group #4 : The innermost loop executed %i times. \n", num ) ;
}
```

```
Group #1 : The innermost loop executed 100 times.
Group #2 : The innermost loop executed 55 times.
Group #3 : The innermost loop executed 55 times.
Group #4 : The innermost loop executed 1000 times.
```

ตัวอย่างโปรแกรมที่ 6.37 เป็นโปรแกรมการวนซ้ำแบบซ้อน ที่มีการใช้ข้อความสั่ง for, while และ do

```
#include <stdio.h>

main ()
{
    int counter_1 = 0 ;    /* Counter for the first loop.    */
    int counter_2 = 0 ;    /* Counter for the second loop. */
    int counter_3 = 0 ;    /* Counter for the third loop.  */

    for (counter_1 = 0 ; counter_1 <= 2 ; counter_1++)
    {
        while (counter_2++ <= 3)
        {
            do
            {
                printf ( "counter_1 = %d\n", counter_1 ) ;
                printf ( "counter_2 = %d\n", counter_2 ) ;
                printf ( "counter_3 = %d\n", counter_3 ) ;
            }
            while (counter_3++ <= 3) ; /* End of third loop. */
        } /* End of second loop. */
    } /* End of first loop. */
}
```

```
counter_1 = 0
counter_2 = 1
counter_3 = 0
counter_1 = 0
counter_2 = 1
counter_3 = 1
counter_1 = 0
counter_2 = 1
counter_3 = 2
counter_1 = 0
counter_2 = 1
counter_3 = 3
counter_1 = 0
counter_2 = 1
counter_3 = 4
counter_1 = 0
counter_2 = 2
counter_3 = 5
counter_1 = 0
counter_2 = 3
counter_3 = 6
counter_1 = 0
counter_2 = 4
counter_3 = 7
```

ตัวอย่างโปรแกรมที่ 6.38 เป็นโปรแกรมที่ตรวจสอบว่า จำนวนที่ถูกป้อนเข้ามานั้นเป็นจำนวนเฉพาะหรือไม่ ถ้าไม่เป็น มีจำนวนใดหารได้ลงตัวบ้าง

```
#include <stdio.h>
#define NO 0
#define YES 1
int main (void)
{
    long num ;           /* number to checked */
    long div ;           /* potential divisors */
    int prime ;

    printf ( "Please enter an integer for analysis ;" ) ;
    printf ( "Enter q to quite. \n" ) ;
    while ( scanf ( "%ld", &num ) == 1 )
    {
        for ( div = 2, prime = YES ; (div * div) <= num ; div++ )
        {
            if ( num % div == 0 )
            {
                if ( (div * div) != num )
                    printf ( "%ld is divisible by %ld and %ld. \n", num, div, num / div ) ;
                else
                    printf ( "%ld is divisible by %ld. \n", num, div ) ;
                prime = NO ;
            }
        }
        if ( prime == YES )
            printf ( "%ld is prime. \n", num ) ;
        printf ( "Please enter another integer for analysis ;" ) ;
        printf ( "Enter q to quit. \n" ) ;
    }
    return 0 ;
}
```

```
Please enter an integer for analysis ; Enter q to quit.
36
36 is divisible by 2 and 18.
36 is divisible by 3 and 12.
36 is divisible by 4 and 9.
36 is divisible by 6.
Please enter another integer for analysis ; Enter q to quit.
149
149 is prime.
Please enter another integer for analysis ; Enter q to quit.
30077
30077 is divisible by 19 and 1583.
Please enter another integer for analysis ; Enter q to quit.
q
```

ตัวอย่างโปรแกรมที่ 6.39 เป็นโปรแกรมแสดงจำนวนเฉพาะตั้งแต่ 2 ถึง 50

```
/* Program to generate a table of prime numbers */
#include <stdio.h>

main ()
{
    int p, is_prime, d ;

    for (p = 2 ; p <= 50 ; ++p)
    {
        is_prime = 1 ;

        for (d = 2 ; d < p ; ++d)
            if (p % d == 0)
                is_prime = 0 ;

        if (is_prime != 0)
            printf ( "%d", p) ;
    }

    printf ("\n") ;
}
```

2	3	5	7	11	13	17	19	23	29	31	37	41	43	47
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----