บทที่ 6 การวนซ้ำ (Iteration)

วัตถุประสงค์

1) เข้าใจการวนซ้ำโดยใช้คำสั่งแบบต่างๆ

2) เลือกวิธีการวนซ้ำที่เหมาะสมในการแก้ปัญหาตัวอย่าง

3) เขียนโปรแกรมที่มีการวนซ้ำได้

ในการเขียนโปรแกรมภาษา C++ เราสามารถทำให้เกิดการวนทำซ้ำสำหรับคำสั่งและบล็อคของคำสั่งได้ อย่างเช่น ถ้าเราต้องการจะแสดงผลอักขระ a ออกทางหน้าจอ 50 ครั้ง เราไม่จำเป็นจะต้องสั่ง cout 50 ครั้ง แต่เราจะใช้การควบคุมโปรแกรมด้วยการวนซ้ำ (iteration หรือ loop) แทน การวนซ้ำนั้นสามารถใช้เพื่อแสดงผลออกทางหน้าจอซ้ำๆ ใช้อ่านข้อมูลเข้ามาซ้ำๆ ใช้จัดการกับแอเรย์ ใช้คำนวณแบบที่ต้องทำซ้ำๆ หรืออื่นๆ ในทำนองเดียวกันได้ ในภาษา C++ นั้น เราใช้คำสั่ง while, do while, และคำสั่ง for ในการวนซ้ำ

เรื่องการวนซ้ำเป็นเรื่องที่อาจจะเข้าใจยากสักหน่อยสำหรับผู้เริ่มต้น ที่สำคัญคือนักศึกษาต้องเข้าใจการเลือกทำให้ดีเสียก่อน ไม่เช่นนั้นจะไม่เข้าใจการวนซ้ำ เพราะการวนซ้ำนั้นคือการสั่งโปรแกรมให้ทำคำสั่งชุดเดิมถ้ามีเงื่อนไขที่เหมาะสม

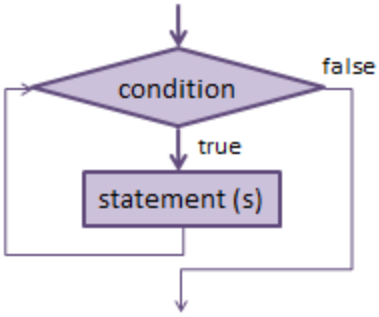
6.1 คำสั่ง while

คำสั่ง while จะทำคำสั่งหรือบล็อคของคำสั่งซ้ำไปเรื่อยๆ ในขณะที่เงื่อนไข (condition) นั้นเป็นจริง และจะหยุดทำซ้ำเมื่อเงื่อนไขนั้นเป็นเท็จ โดยรูปแบบการเขียนคำสั่งแสดงในรูปที่ 1

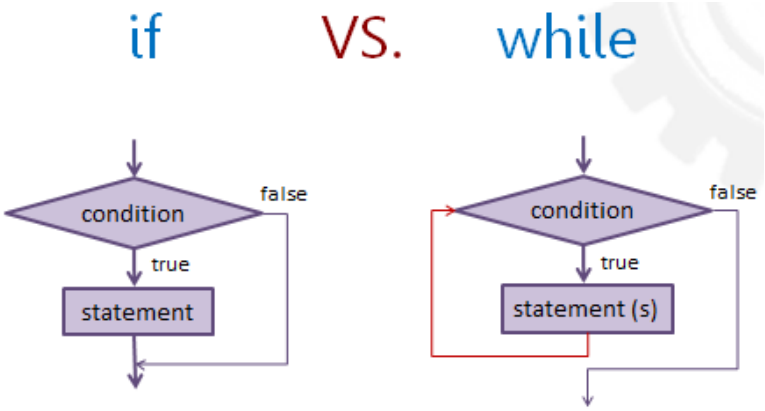
```
while(condition) statement;
```

รูปที่ 1 รูปแบบการใช้คำสั่ง while

เราจะเห็นได้ว่ารูปแบบการเขียนนั้นเหมือนคำสั่ง if มาก แต่ต่างกันตรงที่เมื่อคำสั่ง if ตรวจสอบเงื่อนไขแล้วถ้าพบว่าเป็นจริงจะทำคำสั่งครั้งเดียวแล้วออกไปทำคำสั่งอื่นต่อไป แต่คำสั่ง while จะตรวจสอบเงื่อนไขเหมือนกัน แต่ถ้าเป็นจริงก็จะทำคำสั่งนั้นแล้วกลับมาตรวจสอบเงื่อนไขใหม่ ถ้ายังเป็นจริงอยู่ก็จะทำคำสั่งนั้นอีกครั้ง และวนตรวจสอบและทำคำสั่งซ้ำๆ จนกว่าเงื่อนไขของ while จะเป็นเท็จ ดังแสดงการทำงานในผังงานในรูปที่ 2 ส่วนข้อแตกต่างการทำงานของ if และ while แสดงในรูปที่ 3



รูปที่ 2 ผังงานของคำสั่ง while



รูปที่ 3 เปรียบเทียบผังงานของคำสั่ง if และ while

ในการวนซ้ำนั้น เนื่องจากในที่สุดแล้วเงื่อนไขจะต้องเป็นเท็จโปรแกรมถึงจะหยุดการวนซ้ำได้ เพราะฉะนั้นเราจะต้องมีการปรับค่าของตัวแปร (update) ที่เป็นเงื่อนไขในการวนซ้ำในแต่ละรอบเสมอ ไม่ทางใดก็ทางหนึ่ง เช่น การเพิ่มหรือลดค่าในแต่ละรอบ หรือการเปลี่ยนค่าโดยการคำนวณหรือการรับเข้าผ่านทางแป้นพิมพ์

การวนซ้ำตามจำนวนครั้งที่กำหนดให้

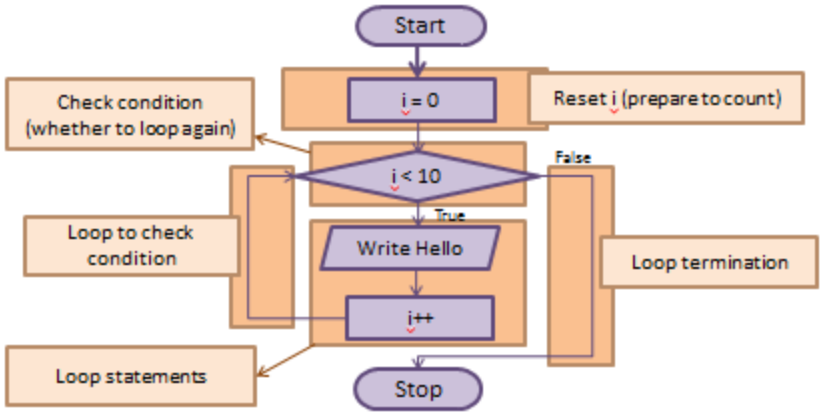
เราสามารถเขียนโปรแกรมเพื่อวนซ้ำเป็นจำนวนครั้งที่แน่นอนได้ โดยอาศัยตัวนับรอบ (counter) เพื่อเปรียบเทียบกับค่าจำนวนรอบที่เราต้องการว่าเราทำซ้ำครบตามจำนวนรอบนั้นหรือยัง

ตัวอย่างที่ 1

โจทย์ : จงเขียนโปรแกรมภาษา C++ เพื่อแสดงข้อความ hello ออกทางหน้าจอ 10 ครั้ง

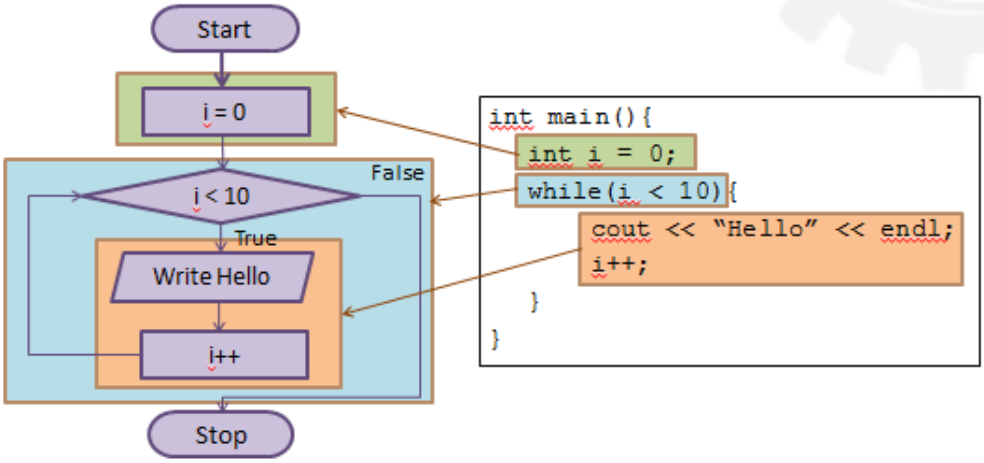
1) เอาต์พุต คือข้อความ hello 10 ครั้ง
2) อินพุต ไม่มี
3) โจทย์ไม่ได้กำหนดอะไรเพิ่มเติม
4) ไม่มีข้อมูลอะไรเพิ่มเติม

เนื่องจากตัวอย่างนี้เป็นตัวอย่างแรกของการวนซ้ำ เราจะขออธิบายผังงานในรูปที่ 4 ก่อน โจทย์ข้อนี้เป็นการวนซ้ำตามจำนวนครั้งที่กำหนดให้ โปรแกรมจะเริ่มต้นด้วยการตั้งค่าตัวแปร i ให้เท่ากับ 0 เราเรียกตัวแปร i ว่าตัวนับ (counter) เป็นตัวนับจำนวนรอบว่าเราวนทำไปกี่รอบแล้ว เพราะฉะนั้นในแต่ละรอบ i จะเพิ่มค่าขึ้นไปเรื่อยๆ ทีละหนึ่ง โจทย์กำหนดให้ทำตามจำนวนครั้ง ในที่นี้คือ 10 ครั้ง เพราะฉะนั้นเราจึงต้องใช้ตัวแปร i เป็นเงื่อนไขในการวนซ้ำ คือเราจะตรวจสอบทุกๆ รอบว่าเราได้แสดงข้อความ hello ไปทั้งหมด 10 ครั้งหรือยัง ถ้าเรายังแสดงข้อความได้น้อยกว่า 10 รอบ เราก็ทำต่อไป แต่ถ้าเราทำครบ 10 ครั้งแล้ว เราก็จะหยุดทำและออกจากลูป (loop) ไปทำคำสั่งต่อไป



รูปที่ 4 ผังงานการวนซ้ำ 10 รอบ

จากผังงาน เราสามารถเขียนโค้ดได้ดังรูปที่ 5 เราจะเห็นได้ว่าเราเขียนโค้ดโดยมีการประกาศและให้ค่า i ก่อนจะเข้าตรวจสอบเงื่อนไข จากนั้นเราเขียน while(i < 10) เพื่อแทนลูปของการตรวจสอบเงื่อนไขว่าจะทำคำสั่ง cout (เขียนคำว่า Hello ออกทางหน้าจอ) และการเพิ่มค่า i หรือไม่



รูปที่ 5 ความสัมพันธ์ของผังงานและโค้ดของการวนซ้ำ 10 รอบ

ทีนี้ถ้าเราต้องการแสดงข้อความ hello ออกทางหน้าจอ 100 ครั้ง เราต้องเปลี่ยนโค้ดอย่างไรบ้าง จากตัวอย่างที่ 1 เราทราบว่าถ้าต้องการวนซ้ำ 10 รอบ เราเอาค่า 10 ไปเปรียบเทียบกับ i (counter) เพื่อที่จะนับว่าเราทำครบ 10 รอบหรือยัง เพราะฉะนั้นถ้าเราต้องการแสดง hello ออกทางหน้าจอ 100 รอบ เราจึงเปลี่ยนตัวเปรียบเทียบจากค่า 10 เป็นค่า 100 ดังโค้ดต่อไปนี้

โค้ด:

```
int main(){
	int i = 0;
	while(i < 100){
		cout << "Hello " << endl;
		i++;
	}
}
```

แล้วถ้าโจทย์ต้องการให้รับจำนวนรอบที่ต้องวนซ้ำผ่านทางแป้นพิมพ์ เราต้องเปลี่ยนโค้ดอย่างไรบ้าง ในทำนองเดียวกัน แทนที่จะเทียบ i กับ 100 เราจะให้เปรียบเทียบ i กับ n ซึ่งเป็นจำนวนครั้งที่เรารับมาจากแป้นพิมพ์ เราเขียนโค้ดได้ดังนี้

โค้ด:

```
int main(){
```

```
      int i = 0, n;
      cout << "Enter n: ";
      cin >> n;

      while(i < n){
            cout << "Hello " << endl;
            i++;
      }
}
```

ตัวอย่างที่ 2

โจทย์ : จงเขียนโปรแกรมภาษา C++ เพื่อแสดงเลขจำนวนเต็มจาก 1-10 ออกทางหน้าจอโดยใช้การวนซ้ำ

1) เอาต์พุต คือเลข 1-10 เรียงกัน
2) อินพุต ไม่มี
3) โจทย์ไม่ได้กำหนดอะไรเพิ่มเติม
4) ไม่มีข้อมูลอะไรเพิ่มเติม

โจทย์ข้อนี้ เราใช้การวนซ้ำในการแสดงเลข 1-10 เราสามารถแสดง i (counter) ออกไปทางหน้าจอได้เลย นั่นคือในรอบที่ 1 เราแสดง 1 รอบที่ 2 เราแสดง 2 ไปเรื่อยๆ จนถึงรอบที่ 10 แต่เราจะต้องเปลี่ยนตำแหน่งการอัปเดตค่า i ไม่เช่นนั้นเราจะได้ผลลัพธ์เป็น 0-9 แทน เราเขียนโค้ดได้ดังนี้

โค้ด:

```
int main(){
      int i = 0;

      while(i < 10){
            i++; // เปลี่ยนมาอัปเดตก่อนแสดงผล
            cout << i << endl;
      }
}
```

จริงๆ แล้วในโจทย์ข้อนี้และในโจทย์ทั่วไป เราสามารถเขียนโค้ดได้หลายแบบ เนื่องจากการเปรียบเทียบค่า i เป็นตัวควบคุมการแสดงผลและการวนซ้ำ เราสามารถคงลำดับการเปลี่ยนแปลงค่า i ไว้ที่เดิมได้ แต่เราต้องปรับเปลี่ยนการเปรียบเทียบค่า i และ/หรือ เปลี่ยนค่าตั้งต้นของ i และ/หรือ เปลี่ยนการแสดงค่า i เช่น เราสามารถเขียนโปรแกรมได้ดังนี้

โค้ด:

```
int main(){
	int i = 1;

	while(i <= 10){
		cout << i << endl;
		i++;
	}
}
```

หรือ

โค้ด:

```
int main(){
	int i = 0;

	while(i < 10){
		cout << i+1 << endl;
		i++;
	}
}
```

ในทำนองเดียวกับการวนซ้ำเพื่อแสดงข้อความ hello ถ้าเราต้องการแสดงเลข 1-100 หรือ 1-n เราก็สามารถเปลี่ยนโค้ดตรงการเปรียบเทียบ i เพื่อแสดงค่าดังกล่าวได้ เราจะแสดงตัวอย่างของโค้ดที่แสดงจำนวนเต็มจาก 1-n ข้างล่างนี้

โค้ด:

```
int main(){
      int i = 1, n;
      cout << "Enter n: ";
      cin >> n;

      while(i <= n){
            cout << i << endl;
            i++;
      }
}
```

ตัวอย่างที่ 3

โจทย์ : จงเขียนโปรแกรมภาษา C++ เพื่อแสดงผลรวมของเลขจำนวนเต็มจาก 1-n ออกทางหน้าจอโดยใช้การวนซ้ำ

1) เอาต์พุต คือผลรวมของเลขจำนวนเต็มจาก 1-n
2) อินพุต คือจำนวนเต็ม n
3) โจทย์ไม่ได้กำหนดอะไรเพิ่มเติม
4) ไม่มีข้อมูลอะไรเพิ่มเติม

โจทย์ข้อนี้จะคล้ายๆ กับตัวอย่างที่ 2 แต่ต่างกันตรงที่แทนที่จะแสดงจำนวนออกไปเลย แต่เราเอาจำนวนนั้นมารวมกับจำนวนก่อนหน้านี้ ดังนั้น นอกจากที่เราต้องมีการนับรอบด้วยตัวนับ (counter) แล้ว เราต้องมีตัวแปรอีกตัวหนึ่งเพื่อที่จะมาเก็บค่าผลรวมของแต่ละรอบไว้ นั่นคือผลรวมของรอบปัจจุบันจะเท่ากันผลรวมของรอบที่แล้ว (ที่ถูกเก็บไว้ในตัวแปรหนึ่ง) บวกกับตัวนับ (counter) ของรอบนี้ ดังแสดงตัวอย่างในการบวกจำนวนเต็มตั้งแต่ 1-5 ในตารางที่ 1

ตารางที่ 1 ค่าของตัวแปรในแต่ละรอบของการวนซ้ำเพื่อหาผลรวม

รอบที่	ผลรวมรอบที่แล้ว (sum ก่อนบรรทัดที่ 7)	ตัวนับที่จะบวกเข้าไป (i)	ผลรวมใหม่ของรอบนี้ (sum หลังบรรทัดที่ 7)
เริ่มต้น	0	1	
1	0	1	1
2	1	2	3
3	3	3	6
4	6	4	10
5	10	5	15

เราสามารถเขียนโค้ดได้ดังนี้ โดยเรากำหนดตัวแปรเพิ่มคือ sum เพื่อเก็บค่าผลรวมในแต่ละรอบ แล้วให้ค่าเริ่มต้นเป็น 0 เพราะ 0 บวกค่าใดก็จะได้ค่านั้น ทำให้เราได้ค่าที่ถูกต้องในการบวกในรอบแรกของลูป

โค้ด:

```
1   int main(){
2         int i = 1, n, sum = 0;
3         cout << "Enter n: ";
4         cin >> n;
5
6         while(i <= n){
7               sum = sum + i; // or sum += i;
8               i++;
9         }
10  }
```

เราจะเห็นได้ว่า เราเก็บค่าผลรวมไว้ในตัวแปร sum ในรอบปัจจุบันตัวแปร sum จะเก็บค่าผลรวมของรอบที่แล้วไว้ก่อน (ก่อนบรรทัดที่ 7) เมื่อเราบวก i เข้าไปแล้ว เราก็เก็บผลรวมใหม่ของรอบนี้ไว้ในตัวแปร sum เหมือนเดิม เพื่อที่จะเอาไปใช้ในรอบถัดไปได้

การวนซ้ำตามค่าปัจจุบันของตัวแปร

ในหัวข้อที่แล้วเราวนซ้ำเป็นจำนวนรอบที่แน่นอน แต่ที่จริงแล้วเรายังสามารถทำการวนซ้ำจนกว่าเราจะได้ค่าที่ต้องการได้โดยไม่จำกัดจำนวนรอบ เราต้องทำการเปรียบเทียบค่าที่เราสนใจกับค่าที่เราต้องการ ดังตัวอย่างต่อไปนี้

ตัวอย่างที่ 4 การหาผลรวมของส่วนกลับ (reciprocal)

โจทย์ : จงเขียนโปรแกรมภาษา C++ เพื่อรับเลขจำนวนเต็ม bound จากแป้นพิมพ์ จากนั้นให้หาผลรวมของส่วนกลับ i ตัวที่มีค่าไม่เกิน bound

1) เอาต์พุต คือผลรวมเลขส่วนกลับ i ตัวที่มีค่าไม่เกิน bound
2) อินพุต คือจำนวนเต็ม bound
3) โจทย์ไม่ได้กำหนดอะไรเพิ่มเติม
4) ส่วนกลับ (reciprocal) คือเลขที่คูณกับส่วนของมันแล้วได้ 1 อย่างเช่น 1/3 คูณ 3 ได้ 1 นั่นคือ 1/3 เป็นส่วนกลับของ 3

ในโจทย์ข้อนี้ เราจะต้องบวก 1/i ไปเรื่อยๆ ตั้งแต่ i เท่ากับ 1 คือ 1 + ½ + 1/3 + ¼ + ... ไปเรื่อยๆ จนกว่าค่าผลรวมจะเกิน bound เพราะฉะนั้น เรารู้ได้ว่าเงื่อนไขในการวนซ้ำคือ ผลรวมปัจจุบันน้อยกว่า bound เราถึงทำการวนซ้ำต่ออีก

โค้ด:

```
int main(){
      float i = 1;
      float bound, sum = 0;
      cout << "Enter bound: ";
      cin >> bound;

      while(sum < bound){
            sum += 1/i;
            cout << "The sum of the first " << i << " reciprocals
is "
            << sum << endl;
            i++;
```

```
        }
}
```

การวนซ้ำรับค่าจากแป้นพิมพ์

นอกจากการวนซ้ำสองแบบข้างต้น เราสามารถวนซ้ำเพื่อรับค่าผ่านทางแป้นพิมพ์ได้ นั่นคือการวางคำสั่ง cin ไว้ในลูป ซึ่งเงื่อนไขในการวนซ้ำอาจจะเกี่ยวกับค่าที่รับเข้ามาผ่านทางแป้นพิมพ์หรือไม่เกี่ยวก็ได้

ตัวอย่างที่ 5

โจทย์ : จงเขียนโปรแกรมภาษา C++ เพื่อรับเลขจำนวนจริง x จากแป้นพิมพ์ จากนั้นแสดงค่ารากที่สองของ x ออกทางหน้าจอ แล้ววนรับค่า x และแสดงค่ารากที่สองของ x ซ้ำจนกว่าผู้ใช้จะใส่ 0 ผ่านทางแป้นพิมพ์

1) เอาต์พุต คือค่ารากที่สองของ x
2) อินพุต คือจำนวนจริง x โดยวนซ้ำเพื่อรับค่าเข้ามา จนกว่า x จะเป็น 0
3) โจทย์ไม่ได้กำหนดอะไรเพิ่มเติม
4) เรารู้ว่าการหารากที่สองของ x เราสามารถใช้ฟังก์ชัน sqrt() ในไลบรารี cmath ได้

ความท้าทายของโจทย์ข้อนี้อยู่ที่การวนรับค่าเข้ามา เรารู้ว่าคำสั่งในการรับค่าคือ cin เพราะฉะนั้น cin จึงต้องอยู่ในลูปและจะต้องอยู่ก่อนการตรวจสอบว่าค่า x มากกว่า 0 หรือไม่

โค้ด:

```
#include <cmath>

int main(){
      double x;
      cout << "Enter a positive number: ";
      cin >> x;
      while(x > 0){
            cout << "Square root (" << x <<
            ") = " << sqrt(x) << endl;
            cout << "Enter another positive "
```

```
            << "number (or 0 to quit): ";
            cin >> x;
      }
      return 0;
}
```

จริงๆ แล้วในโค้ดเราจะต้องมี #include <iostream> ด้วย แต่เราอนุมานเอาว่ามันมีในทุกไฟล์อยู่แล้ว จึงเพิ่มแค่ #include <cmath> เข้าไปเพื่อที่จะใช้ฟังก์ชัน sqrt()

การวนซ้ำโดยใช้ค่าที่มาจากในอดีตมากกว่าหนึ่งรอบ

ในบางกรณี เราต้องการใช้ค่าที่มาจากการวนซ้ำในอดีตมากกว่าหนึ่งรอบขึ้นไป ในกรณีนี้เราต้องเพิ่มตัวแปรขึ้นมาเพื่อเก็บค่าในอดีตในแต่ละรอบ ตัวอย่างเช่น การหาเลขไฟโบนาชชี (Fibonacci number) ซึ่งถูกนิยามคือ

$$f_n = f_{n-2} + f_{n-1}$$

โดยกำหนดให้ $f_0 = 0$ และ $f_1 = 1$ ซึ่งหมายความว่า ค่าของเลขไฟโบนาชชีตัวที่ n คือผลรวมของค่าตัวที่ n-1 และค่าตัวที่ n-2 ดังแสดงในรูปที่ 6

$$f_0 = 0$$
$$f_1 = 1$$
$$f_2 = f_0 + f_1 = 0 + 1 = 1$$
$$f_3 = f_1 + f_2 = 1 + 1 = 2$$
$$f_4 = f_2 + f_3 = 1 + 2 = 3$$
$$f_5 = f_3 + f_4 = 2 + 3 = 5$$
$$\ldots$$
$$f_n = f_{n-2} + f_{n-1}$$

รูปที่ 6 เลขไฟโบนาชชี

ในการหาเลขไฟโบนาชชีนี้เราสามารถใช้การวนซ้ำในการหาได้ โดยการหา f_n ในการวนรอบที่ n ซึ่งจะต้องใช้ค่าในรอบที่ n-1 (รอบที่แล้ว) และค่าในรอบที่ n-2 (สองรอบที่แล้ว) เราจึงต้องมีการใช้ตัวแปรสองตัวในการเก็บค่าของรอบที่แล้วและค่าในสองรอบที่แล้วด้วย โดยมีโค้ดในการคำนวณดังนี้

โค้ด:

```
1   #include <cmath>
2
3   int main(){
4         int n;
5         cout << "Enter a positive integer: ";
6         cin >> n;
7
8         cout << "First " << n << " Fib. numbers: ";
9         if(n >= 1) cout << "0";
10        if(n >= 2) cout << ", 1";
11
12        int a = 0, b = 1;
13        int i = 3;
14        while(i <= n){
15              int f = a + b;
16              cout << ", " << f;
17              a = b;
18              b = f;
19              i++;
20        }
21        return 0;
22  }
```

ในโค้ดนี้ โปรแกรมรับค่าจำนวนเต็มบวก n ผ่านทางแป้นพิมพ์ จากนั้นจะคำนวณเลขไฟโบนาชชี n ตัวแรก โดยเริ่มจาก f_0 ดังนั้นถ้า n มีค่าเป็น 1 (นั่นคือแสดงแค่ f_0) โปรแกรมก็จะพิมพ์ 0 ออกทางหน้าจอ (จากคำสั่งในบรรทัดที่ 9) แล้วจบโปรแกรม ถ้า n เป็น 2 (นั่นคือแสดงแค่ f_0 และ f_1) โปรแกรมก็จะพิมพ์ 0 (จากคำสั่งใน

บรรทัดที่ 9), 1 (จากคำสั่งในบรรทัดที่ 10) แล้วออกจากโปรแกรม เราไม่สามารถเริ่มวนซ้ำเมื่อ n เป็น 1 หรือ 2 ได้ เพราะเราไม่สามารถมีค่าในอดีตอย่างน้อย 2 รอบได้ จึงต้องมีเงื่อนไขมากำหนดค่าให้แทน

แต่ถ้า n มีค่ามากกว่า 2 เราสามารถเริ่มวนซ้ำได้ โดยกำหนดให้ i เริ่มที่ 3 (คำนวณ f_2) และกำหนดให้ตัวแปร b เก็บค่าของ f ในรอบที่แล้ว (f_{n-1}) โดยเริ่มจากมีค่าเท่ากับ 1 (ค่า f_1) และให้ตัวแปร a เก็บค่าของ f ในสองรอบที่แล้ว (f_{n-2}) โดยเริ่มจากมีค่าเท่ากับ 0 (ค่า f_0) เมื่อคำนวณค่า fn ในบรรทัดที่ 15 แล้ว เราก็จะต้องมีการปรับค่า (update) เพื่อการคำนวณในรอบถัดไป นั้นคือ ค่า a (fn-2) ในรอบถัดไปจะเป็นค่า fn-1 ในรอบนี้ (a = b ในบรรทัดที่ 17) ส่วนค่า b (fn-1) ในรอบถัดไปจะเป็นค่า fn ในรอบนี้ (b = f ในบรรทัดที่ 18) สุดท้ายให้เพิ่มค่า i ขึ้น เพื่อเป็นการนับรอบ ถ้าทำครบรอบแล้วก็จบโปรแกม แต่ถ้ายังทำไม่ครบ n ตัวก็ให้วนซ้ำหาค่า f โดยเริ่มจากบรรทัดที่ 15 อีกครั้งหนึ่ง เราสามารถไล่ค่าของ i, a, b และ f ในแต่ละรอบได้ดังนี้

ตารางที่ 2 ค่าของตัวแปรในแต่ละรอบของการหาเลขไฟโบนาชชี ที่บรรทัดที่ 15 ของโปรแกรม

i	f (f_n)	a (f_{n-2})	b (f_{n-1})
3	1	0	1
4	2	1	1
5	3	1	2
6	5	2	3
7	8	3	5
8	13	5	8
9	21	8	13
10	34	13	21

การหยุดการทำงานของลูป (Terminating a loop)

เราสามารถใส่เงื่อนไขอีกแบบเข้าไปในคำสั่ง while นั่นคือ true เมื่อเราใส่ true เป็นเงื่อนไขของ while ลูปจะทำการวนซ้ำไปเรื่อยๆ ไม่มีที่สิ้นสุด (infinite loop) เพราะเงื่อนไขเป็นจริงตลอดเวลา ดังนั้นหากเราต้องการหยุดการวนซ้ำ เราจึงต้องใส่คำสั่งต่อไปนี้เพื่อจะหยุดการวนซ้ำและให้ไปทำคำสั่งอื่นต่อไป

1) break; คำสั่งนี้จะหยุดเฉพาะลูปที่คำสั่งนี้อยู่เท่านั้น จากนั้นโปรแกรมจะทำคำสั่งถัดไปที่ต่อจากลูป
2) exit(0); คำสั่งนี้จะทำให้ลูปหยุดการทำงาน และทั้งฟังก์ชันจะจบการทำงานเลย จะไม่ทำคำสั่งที่เหลือในฟังก์ชันอีก

3) return; คำสั่งนี้จะทำให้ลูปหยุดการทำงาน และทั้งฟังก์ชันจะจบการทำงานเลย จะไม่ทำคำสั่งที่เหลือในฟังก์ชันอีก

การใช้คำสั่งเหล่านี้ โดยทั่วไปแล้วเราต้องมีเงื่อนไขเพิ่มเติมในลูป โดยเมื่อเงื่อนไขที่ว่าเป็นจริง โปรแกรมก็จะทำการหยุดลูป อย่างเช่น การกำหนดคำสั่ง

```
while(true)
      if(i > n) break;
```

หมายความว่าให้วนลูปไปเรื่อยๆ ไม่มีที่สิ้นสุด แต่เมื่อ i มากกว่า n แล้วให้ทำการหยุดลูป

ตัวอย่างที่ 6

โจทย์ : จงเขียนโปรแกรมภาษา C++ เพื่อแสดงผลรวมของเลขจำนวนเต็มจาก 1-n ออกทางหน้าจอโดยใช้การวนซ้ำแบบไม่มีที่สิ้นสุด

1) เอาต์พุต คือผลรวมของเลขจำนวนเต็มจาก 1-n
2) อินพุต คือจำนวนเต็ม n
3) โจทย์กำหนดให้ใช้การวนซ้ำแบบ infinite loop
4) ไม่มีข้อมูลอะไรเพิ่มเติม

เราเคยทำโจทย์ข้อนี้ไปแล้วในตัวอย่างที่ 3 โดยมีเงื่อนไขการวนซ้ำคือเมื่อ i <= n ให้ทำต่อ แต่โจทย์นี้กำหนดให้วนซ้ำแบบไม่มีที่สิ้นสุด ทำให้เราต้องกำหนดเงื่อนไขให้ลูปหยุด นั่นคือเมื่อ i > n เราต้องทำการหยุดลูป

โค้ด:

```
1   int main(){
2         int i = 1, n, sum = 0;
3         cout << "Enter n: ";
4         cin >> n;
5
6         while(true){
7               sum += i;
8               i++;
9               if(i > n) break;
10        }
```

```
11        cout << "The sum of the first " << n << " integers is "
12        << sum << endl;
13
14        return 0;
15  }
```

ถ้าเราใส่ข้อมูลผ่านทางแป้นพิมพ์เป็น 5 เราก็จะได้เอาต์พุตเป็นดังนี้

ผลการทำงาน:

```
Enter n: 5
The sum of the first 5 integers is 15
```

ถ้าหากเราเปลี่ยนคำสั่งจาก break เป็น exit(0) หรือ return จะเกิดอะไรขึ้น เหมือนที่กล่าวไปข้างต้นว่าคำสั่ง exit(0) และ return จะทำให้ทั้งฟังก์ชันจบการทำงานหลังจากบรรทัดที่ 9 ดังนั้นโปรแกรมจะไม่มีการ cout ผลการบวกออก ผลรันจึงจะเหลือแค่ดังนี้

ผลการทำงาน:

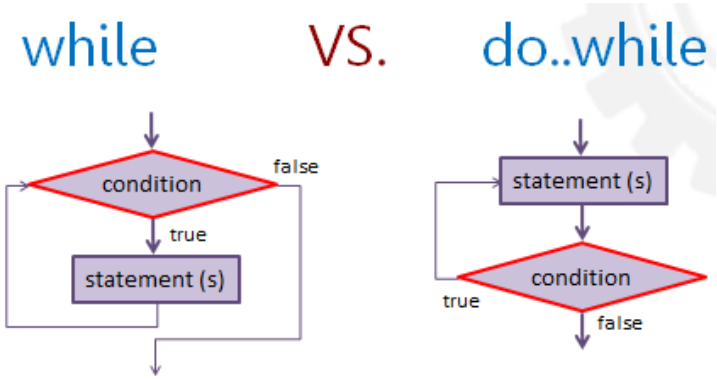
```
Enter n: 5
```

หมายเหตุ ที่จริงแล้วโปรแกรมจะมีการหาผลรวมได้ถูกต้อง แต่มันจะไม่แสดงผลออกทางหน้าจอ เนื่องจากต้องจบการทำงานของฟังก์ชันในบรรทัดที่ 9 ก่อนที่จะมีการทำคำสั่ง cout ในบรรทัดที่ 11

6.2 คำสั่ง do while

คำสั่ง do while เป็นอีกคำสั่งสำหรับการวนซ้ำ แต่ต่างจากคำสั่ง while ตรงที่คำสั่ง do while จะทำคำสั่งในลูปก่อนรอบหนึ่ง แล้วจึงค่อยตรวจสอบเงื่อนไขว่าจะทำการวนลูปอีกครั้งหรือไม่ แต่คำสั่ง while จะ

ตรวจสอบเงื่อนไขก่อนที่จะทำคำสั่งในรอบแรก ถ้าเงื่อนไขเป็นจริงจึงจะทำคำสั่งในลูป ผังงานแสดงความแตกต่างนี้แสดงในรูปที่ 7



รูปที่ 7 เปรียบเทียบผังงานการทำงานของ while และ do while

รูปแบบการเขียนคำสั่ง do while แสดงในรูปที่ 8 คำสั่งเขียนว่า do คือทำคำสั่งที่อยู่ในลูปก่อน แล้วจึงตรวจสอบ while ถ้าเงื่อนไขนั้นเป็นจริงก็ให้วนลูปอีก เราสังเกตว่าหลังส่วนของคำสั่ง do while นั้นมีเครื่องหมายเซมิโคลอน (;) ปิดอยู่ ต่างจากคำสั่ง while ธรรมดาที่จะไม่มีเครื่องหมายเซมิโคลอนปิดอยู่

```
do {
   statements;
}while(condition);
```

รูปที่ 8 รูปแบบคำสั่ง do while

โดยทั่วไปแล้วเราสามารถดัดแปลงโค้ดที่ใช้คำสั่ง while มาเป็นโค้ดที่ใช้คำสั่ง do while ได้ แต่ในบางกรณีก็จะต้องมีการปรับเปลี่ยนตำแหน่งของคำสั่งต่างๆ เพื่อความถูกต้อง ทั้งนี้เพราะในคำสั่ง while โปรแกรมจะทำการตรวจสอบเงื่อนไขก่อน แต่ในคำสั่ง do while โปรแกรมจะทำคำสั่งในลูปก่อนแล้วจึงจะตรวจสอบเงื่อนไขสำหรับการทำในรอบต่อไป ดังนั้นการเรียงคำสั่งในโปรแกรมอาจจะมีความแตกต่างกัน

ตัวอย่างที่ 7

โจทย์ : จงเขียนโปรแกรมภาษา C++ เพื่อแสดงผลรวมของเลขจำนวนเต็มจาก 1-n ออกทางหน้าจอโดยใช้การวนซ้ำด้วยคำสั่ง do while

1) เอาต์พุต คือผลรวมของเลขจำนวนเต็มจาก 1-n
2) อินพุต คือจำนวนเต็ม n
3) โจทย์กำหนดให้ใช้การวนซ้ำด้วยคำสั่ง do while

4) ไม่มีข้อมูลอะไรเพิ่มเติม

เราเคยทำโจทย์ข้อนี้ไปแล้วในตัวอย่างที่ 3 และ ตัวอย่างที่ 6 โดยใช้คำสั่ง while เราจะเห็นได้ว่า ในตัวอย่างนี้โค้ดนั้นเหมือนกับโค้ดในตัวอย่างที่ 3 ที่ใช้คำสั่ง while

โค้ด:

```
int main(){
      int i = 0, n, sum = 0;
      cout << "Enter n: ";
      cin >> n;

      do{
            i++;
            sum += i;
      }while(i < n)
      cout << "The sum of the first " << n << " integers is "
      << sum << endl;

      return 0;
}
```

ตัวอย่างที่ 8 การหาค่าแฟกทอเรียล n ตัวแรก

โจทย์ : จงเขียนโปรแกรมภาษา C++ เพื่อแสดงเลขแฟกทอเรียลจาก 1 ถึง n โดยโปรแกรมจะทำการรับค่า n เป็นจำนวนเต็มผ่านทางแป้นพิมพ์ ให้ทำสองแบบโดยใช้คำสั่ง do while และคำสั่ง while

1) เอาต์พุต คือเลขแฟกทอเรียลจาก 1! ถึง n!
2) อินพุต คือจำนวนเต็ม n
3) โจทย์กำหนดให้ใช้การวนซ้ำด้วยคำสั่ง do while และคำสั่ง while
4) ค่าแฟกทอเรียล คือผลคูณของจำนวนนับตั้งแต่ n ลงไปถึง 1 (หรือ 1 ถึง n) เช่น 5! คือ 5*4*3*2*1 ซึ่งเราได้อธิบายไปในตัวอย่างในบทอัลกอริทึมและผังงานแล้ว

การเขียนโปรแกรมวนซ้ำเพื่อหาค่าแฟกทอเรียลก็จะคล้ายกับการวนซ้ำเพื่อหาค่าผลรวมในตัวอย่างที่ผ่านมา แต่จะต่างกันตรงที่แทนที่จะเก็บค่าผลบวก เราก็เก็บค่าผลคูณแทน และค่าเริ่มต้นของตัวเก็บผลคูณ (fac) จะเป็น 1 เพราะ 1 คูณค่าใดจะได้ค่านั้นเสมอ ซึ่งจะทำให้ผลคูณในการวนรอบแรกนั้นถูกต้อง ค่าของ fac และ i ในแต่ละรอบแสดงในตารางที่ 3

ตารางที่ 3 ค่าของตัวแปรในแต่ละรอบของการวนซ้ำเพื่อหาผลคูณ

รอบที่	**ผลคูณรอบที่แล้ว (fac ก่อนบรรทัดที่ 10)**	**ตัวนับที่จะคูณเข้าไป (i)**	**ผลคูณใหม่ของรอบนี้ (fac หลังบรรทัดที่ 10)**
เริ่มต้น	1	1	
1	1	2	2
2	2	3	6
3	6	4	24
4	24	5	120

เราสามารถเขียนโค้ดโดยใช้คำสั่ง do while และ while ได้ดังนี้

โค้ด: ใช้คำสั่ง do while

```
1  int main(){
2        int n;
3        cout << "Enter a positive integer: ";
4        cin >> n;
5        cout << "Factorial numbers: 1 ";
6
7        long fac = 1, i = 1;
8        do{
9              i++;
10             fac *= i;
11             cout << fac << " ";
12       } while(i < n){
13       cout << endl;
14       return 0;
15 }
```

โค้ด: ใช้คำสั่ง while

```
int main(){
      int n;
      cout << "Enter a positive integer: ";
      cin >> n;

      cout << "Factorial numbers: 1 ";
      long fac = 1, i = 1;
      while(i < n){
            i++;
            fac *= i;
            cout << fac << " ";
      }
      cout << endl;
      return 0;
}
```

ในตัวอย่างนี้โค้ดโดยใช้คำสั่ง while และ do while มีการใช้คำสั่งต่างๆ คล้ายกันมาก แต่ให้ระลึกไว้ว่าไม่ใช่ทุกกรณีที่จะเป็นเช่นนี้ เราจึงต้องตรวจสอบผลรันของโค้ดเสมอ

ตัวอย่างที่ 9 การหาค่าแฟกทอเรียล n ตัวแรกที่ไม่เกินค่าที่กำหนดให้

โจทย์ :จงเขียนโปรแกรมภาษา C++ เพื่อแสดงเลขแฟกทอเรียลจาก 1 ถึงค่าแฟกทอเรียลที่มีค่าไม่เกิน bound โดยโปรแกรมจะทำการรับค่า bound เป็นจำนวนเต็มผ่านทางแป้นพิมพ์ ให้ทำสองแบบโดยใช้คำสั่ง do while และคำสั่ง while

1) เอาต์พุต คือเลขแฟกทอเรียลจาก 1 ถึงค่าแฟกทอเรียลที่มีค่าไม่เกิน bound
2) อินพุต คือจำนวนเต็ม bound
3) โจทย์กำหนดให้ใช้การวนซ้ำด้วยคำสั่ง do while และคำสั่ง while
4) ไม่มีข้อมูลอะไรเพิ่มเติม

ตัวอย่างนี้ เหมือนตัวอย่างที่แล้ว แต่ต่างกันตรงที่เราไม่สามารถวนซ้ำเป็นจำนวนรอบที่แน่นอนได้ แต่เราต้องตรวจสอบว่าค่าแฟกทอเรียลในปัจจุบันนั้นเกินค่า bound ที่รับเข้ามาจากแป้นพิมพ์หรือไม่

โค้ด: ใช้คำสั่ง do while

```
1  int main(){
2        int bound;
3        cout << "Enter a positive integer: ";
4        cin >> bound;
5
6        cout << "Factorial numbers: ";
7        long fac = 1, i = 1;
8        do{
9              cout << fac << " ";
10             i++;
11             fac *= i;
12       } while(fac <= bound);
13       cout << endl;
14       return 0;
15 }
```

โค้ด: ใช้คำสั่ง while

```
1  int main(){
2        int bound;
3        cout << "Enter a positive integer: ";
4        cin >> bound;
5
6        cout << "Factorial numbers: ";
7        long fac = 1, i = 1;
8        while(fac <= bound){
9              cout << fac << " ";
10             i++;
11             fac *= i;
12       }
```

```
13      cout << endl;
14      return 0;
15  }
```

ในตัวอย่างนี้โค้ดโดยใช้คำสั่ง while และ do while มีการใช้คำสั่งต่างๆ คล้ายกันมาก แต่ให้ระลึกไว้ว่าไม่ใช่ทุกกรณีที่จะเป็นเช่นนี้ เราจึงต้องตรวจสอบผลรันของโค้ดเสมอ

6.3 คำสั่ง for

นอกจากคำสั่ง while และ do while แล้ว เราสามารถควบคุมการวนซ้ำได้ด้วยคำสั่ง for เช่นกัน ในคำสั่ง while และ do while เราจะเห็นได้ว่าในเกือบทุกการวนซ้ำเราจะต้องมี

1) Initialization: การให้ค่าเริ่มต้นกับตัวแปรหนึ่งตัวหรือมากกว่า เช่น i = 0, sum = 0
2) Condition: **เงื่อนไขในการวนซ้ำ เช่น** i < n, i > 0
3) Update: การปรับค่า (การอัปเดตค่า) เช่น i++ หรือการอ่านค่าจากแป้นพิมพ์

คำสั่ง for จะนำคำสั่งสำหรับการกระทำทั้ง 3 อย่างมาไว้ในบรรทัดเดียวกันตอนเริ่มต้นเพื่อให้โปรแกรมอ่านและเขียนง่ายขึ้น ทำให้การควบคุมการวนซ้ำทำได้ง่ายขึ้น ในรูปแบบของคำสั่ง for ซึ่งแสดงในรูปที่ 9 เราจะเห็นได้ว่ามีการใช้คำว่า for ซึ่งแปลว่าสำหรับ ตามด้วยวงเล็บ และในวงเล็บมีการให้ค่าเริ่มต้น เงื่อนไข และการปรับค่า ตามลำดับ และแต่ละอย่างถูกคั่นด้วยเครื่องหมายเซมิโคลอน (;) (สังเกตว่าไม่ใช่เครื่องหมายลูกน้ำ (,)) เราสามารถอ่านได้ว่า สำหรับการวนซ้ำนี้ เรากำหนดให้มีการตั้งค่าเริ่มต้นคือ ... มีเงื่อนไขในการวนซ้ำคือ ... และมีการปรับค่าในแต่ละรอบคือ ...

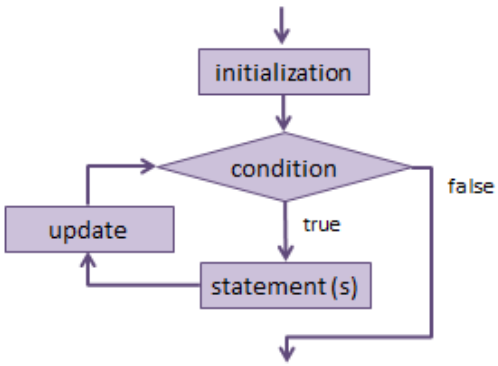
```
for(initialization; condition; update)
   statement;
```

รูปที่ 9 รูปแบบการใช้คำสั่ง for

จากรูปที่ 10 เราเห็นการทำงานของ for นั่นคือ

1) โปรแกรมจะต้องตั้งค่าเริ่มต้นตามที่เขียนไว้ในส่วน initialization และจะทำแค่ครั้งเดียวเท่านั้นก่อนที่จะเข้าลูป

2) โปรแกรมจะตรวจสอบเงื่อนไขตามที่เขียนไว้ในส่วน condition ก่อนที่จะเข้าลูปครั้งหนึ่งก่อน ถ้าเงื่อนไขเป็นจริง ให้ทำข้อ 3 ถ้าเงื่อนไขเป็นเท็จ ให้จบการทำงานของลูปนี้
3) โปรแกรมจะทำคำสั่งที่อยู่ในลูปตามลำดับ
4) จากนั้นโปรแกรมจะปรับค่าที่เขียนอยู่ในส่วน update หลังจากที่ทำคำสั่งต่างๆ และก่อนที่จะตรวจสอบเงื่อนไขอีกครั้งหนึ่ง
5) ทำซ้ำข้อ 2)-4)



รูปที่ 10 ผังงานการทำงานของคำสั่ง for

จากการทำงานที่อธิบายไปข้างต้น เราจะเห็นได้ว่าการทำงานของคำสั่ง for นั้นใกล้เคียงกับการทำงานของ while มาก ต่างกันแค่วิธีเขียนเท่านั้นที่แยกเอาการตั้งค่าเริ่มต้นและการปรับค่าในแต่ละรอบออกมาเขียนในบรรทัดแรก

ตัวอย่างที่ 10 การวนซ้ำเพื่อหาผลรวมโดยใช้คำสั่ง for

โจทย์ : จงเขียนโปรแกรมภาษา C++ เพื่อแสดงผลรวมของเลขจำนวนเต็มจาก 1-n ออกทางหน้าจอโดยใช้การวนซ้ำด้วยคำสั่ง for

1) เอาต์พุต คือผลรวมของเลขจำนวนเต็มจาก 1-n
2) อินพุต คือจำนวนเต็ม n
3) โจทย์กำหนดให้ใช้การวนซ้ำด้วยคำสั่ง for
4) ไม่มีข้อมูลอะไรเพิ่มเติม

เราเคยทำโจทย์ข้อนี้ไปแล้วในตัวอย่างที่ 3 ตัวอย่างที่ 6 และตัวอย่างที่ 7 แต่ข้อนี้เราต้องใช้คำสั่ง for เราจึงเขียนโค้ดเหมือนกัน แต่เอา int i = 0 (initialization) และ i++ (update) มาไว้ในวงเล็บหลัง for ตามรูปแบบ

โค้ด:

```
int main(){
      int n, sum = 0;
      cout << "Enter n: ";
      cin >> n;

      for(int i = 0; i < n; i++){
            sum += i;
      }
      cout << "The sum of the first " << n << " integers is "
      << sum << endl;

      return 0;
}
```

ขอบเขตของตัวแปรที่ถูกตั้งค่าเริ่มต้นในคำสั่ง for

ในคำสั่ง for เราสามารถประกาศตัวแปรเพื่อให้ค่าตั้งต้นในส่วนของ initialization ได้เลย หรือเราจะประกาศตัวแปรที่จะใช้ก่อนเขียนคำสั่ง for แล้วให้ค่าตั้งต้นในส่วนของ initialization ในคำสั่ง for ก็ได้ ความแตกต่างของการเขียน 2 แบบนี้อยู่ที่ขอบเขตของตัวแปร ถ้าเราประกาศตัวแปรในส่วนของ initialization ขอบเขตของตัวแปรจะอยู่ในแค่ for เท่านั้น เราไม่สามารถเอาตัวแปรไปใช้นอกลูปได้ นั่นคือถ้าเราเอาตัวแปรไปใช้นอกลูปโปรแกรมจะไม่รู้จักตัวแปรนั้นและจะทำให้คอมไพล์ไม่ผ่าน แต่ถ้าเราประกาศตัวแปรก่อนที่จะเข้า for เราจะสามารถใช้ตัวแปรนี้นอกลูปของคำสั่ง for ได้

ตัวอย่างที่ 11

เราสามารถเขียนโปรแกรมในตัวอย่างที่ 10 ได้อีกแบบ ดังนี้

โค้ด:

```
1  int main(){
2        int n, sum = 0;
```

```
3       cout << "Enter a positive integer: ";
4       cin >> n;
5       for(int i=0; i < n/2; i++)
6             sum += i;
7       for(int i=n/2; i <= n; i++)
8             sum += i;
9       cout << "The sum of the first " << n
10      << " integers is " << sum << endl;
11      return 0;
12  }
```

เราจะเห็นได้ว่าตัวแปร i ในคำสั่ง for อันแรกที่บรรทัดที่ 5 กับตัวแปร i ในคำสั่ง for อันที่สองที่บรรทัดที่ 7 ไม่ใช่ตัวแปรตัวเดียวกัน เนื่องจากตัวแปร i แรกถูกประกาศใน initialization ของ for ตัวแรก ทำให้ขอบเขตของมันอยู่ใน for ตัวแรกเท่านั้น (จากบรรทัดที่ 5 ถึงบรรทัดที่ 6) ส่วนตัวแปร i ตัวที่ 2 ก็เช่นเดียวกัน มันมีขอบเขตแค่จากบรรทัดที่ 7 ถึงบรรทัดที่ 8 เท่านั้น แต่ถ้าเรามีโปรแกรมดังนี้

โค้ด:

```
1   int main(){
2       int n, sum, i=0;
3       cout << "Enter a positive integer: ";
4       cin >> n;
5       for(sum = 0; i < n/2; i++)
6             sum += i;
7       for(i=n/2; i <= n; i++)
8             sum += i;
9       cout << "The sum of the first " << n
10      << " integers is " << sum << endl;
11      return 0;
12  }
```

ตัวแปร i ในโปรแกรมจะเป็น i ตัวเดียวกันหมด เพราะมันถูกประกาศครั้งเดียวที่บรรทัดที่ 2 นอกจากนี้ ตัวแปร sum ที่ใช้ใน for อันแรกที่บรรทัดที่ 6 และอันที่สองที่บรรทัดที่ 8 เป็นตัวเดียวกัน คือตัวที่ประกาศไว้ในบรรทัดที่ 2 ของโปรแกรมนั่นเอง แม้ว่ามันจะถูกตั้งค่าเริ่มต้นไว้ในส่วน initialization ของคำสั่ง for แต่เพราะมันประกาศก่อนที่จะเข้า for เราจึงสามารถใช้ sum ได้ในส่วนอื่นๆ ของฟังก์ชันเมน ในโค้ดตัวอย่างนี้คือใช้ในลูปของคำสั่ง for ตัวที่สอง

ตัวอย่างที่ 12

โจทย์: จงหาความแตกต่างของผลรันของโปรแกรมต่อไปนี้

โค้ด 1:

```
1  int main() {
2     int i = 10;
3     cout << "begin " << i << endl;
4
5     for(int i=0; i <= 20; i++)
6        cout << i;
7
8     cout << endl;
9     cout << "end loop " << i;
10
11    return 0;
12 }
```

ผลการทำงาน 1:

```
Begin 10
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
end loop 10
```

โค้ด 2:

```
1  int main() {
2     int i = 10;
3     cout << "begin " << i << endl;
```

```
4       for (i=0; i <= 20; i++)
5          cout << i;
6
        cout << endl;
7       cout << "end loop " << i;
8       return 0;
9    }
10
```

ผลการทำงาน 2:

```
Begin 10
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
end loop 21
```

ผลการทำงานของโค้ดทั้งสองแบบนี้ต่างกันแค่ที่ค่า i ที่แสดงออกที่บรรทัดที่ 9 หลังจากคำว่า end loop ในโค้ดที่ 1 ค่า i ที่บรรทัดที่ 9 เป็น 10 เหมือนตัวที่ประกาศและแสดงออกที่บรรทัดที่ 3 เพราะมันเป็นตัวเดียวกัน และไม่ได้ถูกเปลี่ยนค่าในลูป for ส่วนตัว i ที่อยู่ในลูป for นั้น เป็น i คนละตัว เนื่องจากมันถูกประกาศใหม่ ใช้ได้และเปลี่ยนแปลงได้เฉพาะในขอบเขตของคำสั่ง for เท่านั้น เมื่อพ้นจากขอบเขตของ for แล้ว i ตัวนั้นก็จะหายไป

ส่วนในโค้ดที่ 2 i ในคำสั่ง for นั้นเป็น i ตัวเดียวกับที่ประกาศในบรรทัดที่ 3 ซึ่งจะถูกใช้ได้และเปลี่ยนแปลงได้ทุกที่ในฟังก์ชันเมนนี้ ในคำสั่ง for โปรแกรมได้มีการให้ค่าเป็น 0 ณ ตอนเริ่มต้นการวนซ้ำที่บรรทัดที่ 5 จากนั้นi ถูกเพิ่มค่าขึ้นไปเรื่อยๆ และจบที่ค่า 21 ซึ่งทำให้ลูปจบเพราะค่ามันมากกว่า 20 จากนั้นมันถูกแสดงที่บรรทัดที่ 9 ค่าที่ได้จึงเป็น 21

ตัวอย่างที่ 13 การหาค่าแฟกทอเรียล n ตัวแรกโดยใช้คำสั่ง for

โจทย์ : จงเขียนโปรแกรมภาษา C++ เพื่อแสดงเลขแฟกทอเรียลจาก 1 ถึง n โดยโปรแกรมจะทำการรับค่า n เป็นจำนวนเต็มผ่านทางแป้นพิมพ์ ให้ทำสองแบบโดยใช้คำสั่ง for

1) เอาต์พุต คือเลขแฟกทอเรียลจาก 1! ถึง n!
2) อินพุต คือจำนวนเต็ม n
3) โจทย์กำหนดให้ใช้การวนซ้ำด้วยคำสั่ง for

4) ไม่มีข้อมูลอะไรเพิ่มเติม

โค้ด:

```
int main(){
      int n;
      cout << "Enter a positive integer: ";
      cin >> n;

      cout << "Factorial numbers: ";
      long fac = 1;
      for(int i=1; i<=n; i++)
            fac *= i;
            cout << fac << " ";
      }
      cout << endl;
      return 0;
}
```

เช่นเดียวกับคำสั่ง while และ do while เราสามารถเขียนโปรแกรมเพื่อควบคุมการวนซ้ำตามค่าปัจจุบันได้เช่นกัน ดังตัวอย่างต่อไปนี้

ตัวอย่างที่ 14 การหาค่าแฟกทอเรียลภายในค่าที่กำหนดให้ โดยใช้คำสั่ง for

โจทย์: จงเขียนโปรแกรมภาษา C++ เพื่อแสดงเลขแฟกทอเรียลจาก 1 ถึงค่าแฟกทอเรียลที่มีค่าไม่เกิน bound โดยโปรแกรมจะทำการรับค่า bound เป็นจำนวนเต็มผ่านทางแป้นพิมพ์ โดยใช้คำสั่ง for

1) เอาต์พุต คือเลขแฟกทอเรียลจาก 1 ถึงค่าแฟกทอเรียลที่มีค่าไม่เกิน bound
2) อินพุต คือจำนวนเต็ม bound
3) โจทย์กำหนดให้ใช้การวนซ้ำด้วยคำสั่ง for
4) ไม่มีข้อมูลอะไรเพิ่มเติม

โค้ด:

```
int main(){
      int bound;
      cout << "Enter a positive integer: ";
      cin >> bound;

      cout << "Factorial numbers: ";
      long fac = 1;
      for(int i=2; fac < bound; i++)
            cout << fac << " ";
            fac *= i;
      }
      cout << endl;
      return 0;
}
```

การปรับค่า (update) แบบต่างๆ ในคำสั่ง for

จากตัวอย่างต่างๆ ที่เราเรียนมา การปรับค่า (update) ในคำสั่ง for นั้นเป็นการปรับค่าแบบเพิ่มค่าทีละหนึ่ง (คือใช้ i++) แต่ในความเป็นจริง เราสามารถปรับค่าได้หลายแบบ เช่นลดค่าที่ละหนึ่ง (ใช้ i--) หรือเพิ่มหรือลดค่าทีละมากกว่า 1 เช่นเพิ่มค่าทีละสอง หรือลดค่าทีละสาม

ตัวอย่างที่ 15

โจทย์ : จงเขียนโปรแกรมภาษา C++ เพื่อรับค่าจำนวนเต็ม n ผ่านทางแป้นพิมพ์ แล้วแสดงเลข n ถึง 1 โดยค่อยๆ ลดค่าไปทีละ 1 ผ่านทางหน้าจอ โดยใช้คำสั่ง for

1) เอาต์พุต คือเลข n ถึง 1 โดยค่อยๆ ลดไปทีละ 1
2) อินพุต คือจำนวนเต็ม n
3) โจทย์กำหนดให้ใช้การวนซ้ำด้วยคำสั่ง for
4) ไม่มีข้อมูลอะไรเพิ่มเติม

เราเคยเขียนโปรแกรมให้วนซ้ำจาก 1 ถึง n โดยให้ i เริ่มจาก n แล้วพิมพ์ i โดยเพิ่ม i ไปทีละหนึ่งในแต่ละรอบ แต่โจทย์ข้อนี้เราต้องการพิมพ์จาก n ลดลงไปยัง 1 เราก็ทำแบบเดียวกันแต่แทนที่เราจะเริ่มต้นที่ 1 เราก็เริ่มต้นที่ n แล้วลดค่า i ไปทีละหนึ่ง

โค้ด:

```
int main(){
      int n;
      cout << "Enter a positive integer: ";
      cin >> n;

      for(int i=n; i>0; i--){
            cout << i << " ";
      }
      cout << endl;
      return 0;
}
```

ตัวอย่างที่ 16

โจทย์ : จงหาผลรันของโปรแกรมต่อไปนี้ เมื่อโปรแกรมรับค่า 10 ผ่านทางแป้นพิมพ์

โค้ด:

```
1  int main(){
2        int n;
3        cout << "Enter a positive integer: ";
4        cin >> n;
5        if(n < 2)
6              cout << n << " is not prime.\n";
7        else if(n < 4)
8              cout << n << " is prime.\n";
9        else if(n%2 == 0)
```

```
10              cout << n << " = 2 * " << n/2 << endl;
11        else{
12              for(int d = 3; d <= n/2; d += 2)
13                    if(n%d == 0){
14                          cout << n << " = " << d <<" * " << n/d <<
endl;
15
16                          exit(0);
17                    }
18              cout << n << " is prime.\n";
19        }
20        return 0;
21  }
```

โปรแกรมนี้เป็นโปรแกรมที่แสดงว่าจำนวนเต็ม n ที่รับเข้ามาเป็นจำนวนเฉพาะ (prime number) หรือไม่ โดยจำนวนเฉพาะ (prime number) เป็นจำนวนที่มีเพียงแค่ 1 และตัวมันเองเท่านั้นที่หารมันลงตัว โปรแกรมนี้ได้ตรวจสอบ n ตามลำดับขั้นตอนดังนี้

1) n น้อยกว่า 2 หรือไม่ ถ้าน้อยกว่า (คือเลข 1) ก็จะไม่เป็นจำนวนเฉพาะ
2) n น้อยกว่า 4 หรือไม่ ถ้าน้อยกว่า (คือเลข 2 และ 3) ก็จะเป็นจำนวนเฉพาะ
3) n หารด้วย 2 ลงตัวหรือไม่ ถ้าหารลงตัวก็ไม่เป็นจำนวนเฉพาะ
4) ถ้าไม่เข้าเงื่อนไขใดๆ แล้ว เราก็จะทำการวนซ้ำจาก d=3 ถึง d=n/2 ดูว่า n สามารถหารด้วย d ตัวใดได้ลงตัวหรือไม่ โดยจะเพิ่มค่า d ทีละสอง เนื่องจากเราจะไม่ตรวจสอบ d ที่เป็นเลขคู่เพราะถ้าหารด้วยเลขคู่นั้นลงตัว n ก็จะต้องหารด้วย 2 ลงตัว ซึ่งเราทำการตรวจสอบไปแล้วในข้อ 3) ส่วนการที่เราหยุด d ไว้ที่ครึ่งหนึ่งของจำนวนที่รับเข้ามา (n/2) เพราะว่าไม่มี d ใดที่มากกว่าจำนวนนี้แล้วที่จะสามารถหาร n ได้ลงตัว เมื่อสิ้นสุดลูปนี้โดยที่ไม่มี d ใดหารลงตัวแล้ว ก็แสดงว่า n นั้นเป็นจำนวนเฉพาะ เพราะไม่มีจำนวนเต็มบวกใดหารลงตัว นอกจาก 1 และตัวมันเอง

ดังนั้น จากโจทย์เมื่อโปรแกรมรับค่า 10 ผ่านทางแป้นพิมพ์ ก็จะเข้าเงื่อนไขในข้อ 3) คือหารด้วย 2 ลงตัว เพราะฉะนั้นโปรแกรมจะให้ผลรันว่า 10 = 2*5

การหยุดการวนซ้ำแบบอื่น

ในการเขียนการวนซ้ำด้วยคำสั่ง for เราไม่จำเป็นจะต้องใส่ข้อมูลในส่วนของ initialization, condition, update ให้ครบก็ได้ เราสามารถควบคุมการวนซ้ำด้วยคำสั่ง for ในที่อื่นๆ ถ้าเราไม่ต้องการใส่ค่าใด ก็ให้เราเว้นค่านั้นว่างไว้ แต่เราต้องมีเครื่องหมายเซมิโคลอน (;) ให้ครบเสมอ เช่นถ้าเราไม่ต้องการใส่ค่าเริ่มต้นหรือค่าอัปเดต เราก็สามารถเว้นว่างไว้ได้ โดยเขียน for(; condition ;) ถ้าเราไม่ต้องการใส่ค่าอะไรเลย เราก็เขียนว่า for(; ;) อย่างไรก็ตาม ถ้าเราไม่ใส่เงื่อนไขในการวนซ้ำ ลูปจะทำการวนซ้ำไปเรื่อยๆ ไม่มีที่สิ้นสุด เราต้องทำการหยุดลูปโดยใช้คำสั่ง break, exit(0), หรือ return ตามที่เคยได้เรียนไปแล้ว

ตัวอย่างที่ 17

โจทย์ : จงเขียนโปรแกรมเพื่อรับเลขจำนวนเต็มผ่านทางแป้นพิมพ์ โดยให้วนซ้ำรับค่าเข้ามาเรื่อยๆ จนว่าอินพุตจะมีค่าเท่ากับ 0 จากนั้นให้หาว่าอินพุตใดที่มีค่ามากที่สุด โดยให้ใช้คำสั่ง for

1) เอาต์พุต คือค่าอินพุตที่มากที่สุด
2) อินพุต คือจำนวนเต็ม n โดยวนรับค่าเข้ามาจนกว่าอินพุตจะมีค่าเป็น 0
3) โจทย์กำหนดให้ใช้คำสั่ง for
4) ไม่มีข้อมูลอะไรเพิ่มเติม

ในข้อนี้เราต้องใช้คำสั่ง for แต่ว่าเราไม่สามารถอัปเดตค่าตามตัวอย่างที่ผ่านๆ มาได้เพราะเราไม่ทราบว่าจะต้องรับค่าเข้ามากี่ค่า รู้แต่เงื่อนไขที่ว่าให้รับค่าจนกว่าอินพุตจะมีค่าเป็น 0 เท่านั้น ดังนั้นเราสามารถที่จะทิ้งตำแหน่ง update ของคำสั่ง for ให้ว่างได้

โจทย์ต้องการให้เราหาค่าที่มากที่สุดในค่าที่ผู้ใช้ใส่เข้ามาผ่านทางแป้นพิมพ์ ในการเขียนโค้ดนี้ เราจึงต้องมีตัวแปรตัวหนึ่งเพื่อเก็บค่าที่มากที่สุดในปัจจุบัน กำหนดให้เป็นตัวแปร max เมื่อโปรแกรมอ่านค่าจากผู้ใช้เข้ามา เราต้องเอามาเปรียบเทียบกับค่า max ว่าค่าที่เข้ามาใหม่นั้นมากกว่า max หรือไม่ ถ้าไม่มากกว่า แสดงว่าค่าที่มากที่สุดที่โปรแกรมเจอก็ยังเป็นค่า max อยู่ แต่ถ้ามากกว่า แสดงว่าค่าที่มากที่สุดที่โปรแกรมเจอคือค่าที่อ่านเข้ามาใหม่ ดังนั้นโปรแกรมจึงต้องเปลี่ยนค่า max ให้เป็นค่าที่เพิ่งใส่เข้ามา ดังแสดงในโค้ดต่อไปนี้

โค้ด:

```
int main(){
      int n, max;
      cout << "Enter positive integers (0 to quit): ";
```

```
	cin >> n;

	for(max = n; n != 0;)
	{
		if(n > max) max = n;
		cin >> n;
	}
	cout << "max = " << max << endl;
	return 0;
}
```

ตัวอย่างที่ 18

โจทย์ : จงเขียนโปรแกรมเพื่อรับเลขจำนวนเต็มบวกผ่านทางแป้นพิมพ์ โดยให้วนซ้ำรับค่าเข้ามาเรื่อยๆ จนว่าอินพุตจะมีค่าเท่ากับ 0 จากนั้นให้หาผลรวมของอินพุตทั้งหมด โดยให้ใช้คำสั่ง for แบบไม่กำหนดค่าใดๆ

1) เอาต์พุต คือผลรวมของอินพุต
2) อินพุต คือจำนวนเต็ม n โดยวนรับค่าเข้ามาจนกว่าอินพุตจะมีค่าเป็น 0
3) โจทย์กำหนดให้ใช้คำสั่ง for แบบไม่กำหนดค่าใดๆ คือใช้รูปแบบ for(; ;) เราจึงต้องมีการหยุดลูปด้วยคำสั่ง break, exit(0) หรือ return
4) ไม่มีข้อมูลอะไรเพิ่มเติม

โจทย์ข้อนี้ให้หาผลรวมเช่นเดียวกับที่เราเคยทำไปแล้ว แต่ต่างกันตรงที่ว่า 1) ไม่มีการกำหนดรอบที่แน่นอน และ 2) ค่าที่เอามารวมเป็นค่าต่างๆ ที่รับเข้ามาจากผู้ใช้ อย่างไรก็ตามโค้ดที่เขียนก็จะคล้ายเดิม คือมีการเก็บผลรวมใน sum และบวกค่าที่รับเข้ามาจากผู้ใช้ในแต่ละรอบ จนกว่าผู้ใช้จะใส่ค่า 0 เข้ามา เนื่องจากเราใช้คำสั่ง for แบบไม่กำหนดเงื่อนไข ทำให้เราต้องกำหนดเงื่อนไขในลูปแทนเพื่อที่จะให้ลูปนั้นหยุดทำงานเมื่อผู้ใช้ใส่อินพุตเข้ามาเป็น 0 ดังโค้ดข้างล่างนี้

โค้ด:

```
int main(){
	int n, sum = 0;
	cout << "Enter positive integers (0 or negative to quit):"
```

```
      << endl;

      for(;;){
            cin >> n;
            if(n <= 0) break;
            sum += n;
      }
      cout << "Sum = " << sum << endl;
      return 0;
}
```

ถ้าเราเปรียบเทียบการใช้ for(; ;) กับ while(true) ซึ่งเป็นการไม่กำหนดเงื่อนไขการหยุดลูปหลังคำสั่งเช่นกัน เราจะได้โค้ดที่คล้ายกันมากดังนี้

โค้ด:

```
int main(){
      int n, sum = 0;
      cout << "Enter positive integers (0 or negative to quit):"
      << endl;

      while(true){
            cin >> n;
            if(n <=0) break;
            sum += n;
      }
      cout << "Sum = " << sum << endl;
      return 0;
}
```

ตัวอย่างที่ 19

โจทย์ : จงเขียนโปรแกรมเพื่อรับเลขจำนวนเต็มบวกผ่านทางแป้นพิมพ์ โดยให้วนซ้ำรับค่าเข้ามาเรื่อยๆ จนว่าอินพุตจะมีค่าเท่ากับ 0 จากนั้นให้หาค่าเฉลี่ยของอินพุตทั้งหมด โดยให้ใช้คำสั่ง for แบบไม่กำหนดค่าใดๆ

1) เอาต์พุต คือผลรวมของอินพุต
2) อินพุต คือจำนวนเต็ม n โดยวนรับค่าเข้ามาจนกว่าอินพุตจะมีค่าเป็น 0
3) โจทย์กำหนดให้ใช้คำสั่ง for แบบไม่กำหนดค่าใดๆ คือใช้รูปแบบ for(; ;) เราจึงต้องมีการหยุดลูปด้วยคำสั่ง break, exit(0) หรือ return
4) ค่าเฉลี่ยคือผลรวมหารด้วยจำนวนอินพุตที่เรารับมาทั้งหมด เรามีโค้ดของการหาผลรวมแล้วในตัวอย่างที่ 18 เราเพียงแค่จะต้องเพิ่มตัวแปรเข้าไปเพื่อนับว่าเรามีอินพุตทั้งหมดกี่ตัว

โค้ด:

```
int main(){
      int n, sum = 0;
      cout << "Enter positive integers (0 or negative to quit):"
      << endl;

      while(true){
            cout << "\t" << count + 1 << " : ";
            cin >> n;
            if(n <= 0) break;
            ++count;
            sum += n;
      }
      cout << "The average of those " << count <<
      " positive numbers is " << float(sum)/count << endl;
      return 0;
}
```

6.4 การวนซ้ำแบบซ้อน หรือการซ้อนลูป (Nested loop)

ในเรื่องการเลือกทำ (selection) เรามีการเลือกทำแบบซ้อน (nested selection) คือการเอาคำสั่ง if มาซ้อนกัน ในเรื่องการวนซ้ำเราก็มีเช่นกัน เรียกว่า การวนซ้ำแบบซ้อน (nested loop) หรือการซ้อนลูป สำหรับบทเรียนนี้เราจะเรียนการซ้อนลูปแค่ 3 ชั้น แต่จะเน้นที่ลูป 2 ชั้น คือมีลูปชั้นนอก และลูปชั้นใน ดังตัวอย่างในรูปที่ 11 โดยกรอบใหญ่คือลูปชั้นนอก และกรอบเล็กคือลูปชั้นใน

```
int main(){
    for(int x = 1; x <= 12; x++){
        for(int y = 1; y <= 12; y++){
            cout << setw(4) << x * y;
        }
        cout << endl;
    }
    return 0;
}
```

รูปที่ 11 ตัวอย่างการวนแบบซ้อนหรือการซ้อนลูป

โดยหลักการทำงานของลูปซ้อนลูปคือมันจะทำซ้ำเป็นจำนวนเท่ากับ m x n รอบ เมื่อ m เป็นจำนวนรอบที่ต้องทำของลูปชั้นนอก และ n เป็นจำนวนรอบที่ต้องทำสำหรับลูปชั้นใน โดยการทำงานจะเป็นไปตามตารางที่ 4 นั่นคือโปรแกรมจะเข้าไปในรอบแรกของลูปชั้นนอกก่อน ซึ่งมีลูปชั้นในอยู่ มันจะทำการวนซ้ำลูปชั้นในให้ครบ n รอบก่อน จากนั้นจึงเริ่มรอบที่สองของลูปชั้นนอก นั่นคือการวนซ้ำลูปชั้นในให้ครบ n รอบ ทำแบบนี้ไปเรื่อยๆ จนถึงรอบที่ m (รอบสุดท้าย) ของลูปชั้นนอก

ตารางที่ 4 การวนซ้ำของลูปชั้นนอกและชั้นในในการซ้อนลูป

ลูปชั้นนอก รอบที่ 1	ลูปชั้นในรอบที่ 1
	ลูปชั้นในรอบที่ 2
	...
	ลูปชั้นในรอบที่ n
ลูปชั้นนอก รอบที่ 2	ลูปชั้นในรอบที่ 1
	ลูปชั้นในรอบที่ 2
	...
	ลูปชั้นในรอบที่ n

...	...
ลูปชั้นนอก รอบที่ m	ลูปชั้นในรอบที่ 1
	ลูปชั้นในรอบที่ 2
	...
	ลูปชั้นในรอบที่ n

ตัวอย่างที่ 20 ตารางสูตรคูณ

โจทย์ : จงหาผลรันของโปรแกรมต่อไปนี้

โค้ด:

```
1   #include <iomanip>
2
3   int main(){
4         for(int x = 1; x <= 12; x++){
5               for(int y = 1; y <= 12; y++){
6                     cout << setw(4) << x * y;
7               }
8               cout << endl;
9         }
10        return 0;
11  }
```

โจทย์ข้อนี้เป็นคำสั่ง for ซ้อนกัน โดยชั้นนอกมีการวนซ้ำ 12 รอบและมีคำสั่ง endl ปิดท้าย (endl เป็นคำสั่งที่อยู่ในลูปชั้นนอก) ส่วนลูปชั้นในก็วนซ้ำ 12 รอบเช่นกัน และมีคำสั่งแสดงค่าผลคูณของ x และ y โดยไม่มีการเคาะบรรทัด ดังนั้นในการวนรอบแรกของลูปชั้นนอกและวนซ้ำลูปชั้นในจนครบ 12 ครั้ง เราจะได้ค่าดังนี้

ตารางที่ 5 ค่า x, y และ x*y ในการวนรอบแรกของลูปชั้นนอก

ค่า x	ค่า y	ค่า x * y
1	1	1
1	2	2
1	3	3
1	4	4
1	5	5
1	6	6
1	7	7
1	8	8
1	9	9
1	10	10
1	11	11
1	12	12

นั่นคือเราจะได้ผลรันในรอบแรกเป็น

```
1    2    3    4    5    6    7    8    9    10   11   12
```

จากนั้นในรอบที่สองของลูปชั้นนอกและวนซ้ำลูปชั้นในจนครบ 12 ครั้ง เราจะได้ค่าดังนี้

ตารางที่ 6 ค่า x, y และ x*y ในการวนรอบที่สองของลูปชั้นนอก

ค่า x	ค่า y	ค่า x * y
2	1	2
2	2	4
2	3	6
2	4	8
2	5	10
2	6	12
2	7	14

2	8	16
2	9	18
2	10	20
2	11	22
2	12	24

นั่นคือเราจะได้ผลรันถึงรอบสองเป็น

```
   1    2    3    4    5    6    7    8    9   10   11   12
   2    4    6    8   10   12   14   16   18   20   22   24
```

เมื่อโปรแกรมรันจนจบ เราจะได้เอาต์พุตคล้ายตารางสูตรคูณจากแม่ 1 ถึงแม่ 12 ดังนี้

ผลการทำงาน:

```
   1    2    3    4    5    6    7    8    9   10   11   12
   2    4    6    8   10   12   14   16   18   20   22   24
   3    6    9   12   15   18   21   24   27   30   33   36
   4    8   12   16   20   24   28   32   36   40   44   48
   5   10   15   20   25   30   35   40   45   50   55   60
   6   12   18   24   30   36   42   48   54   60   66   72
   7   14   21   28   35   42   49   56   63   70   77   84
   8   16   24   32   40   48   56   64   72   80   88   96
   9   18   27   36   45   54   63   72   81   90   99  108
  10   20   30   40   50   60   70   80   90  100  110  120
  11   22   33   44   55   66   77   88   99  110  121  132
  12   24   36   48   60   72   84   96  108  120  132  144
```

หมายเหตุ คำสั่ง setw(4) เป็นคำสั่งจัดรูปแบบการแสดงผลในไลบรารี iomanip เป็นคำสั่งที่บอกว่าให้จัดการแสดงผลเป็นบล็อค บล็อคละ 4 ตัวอักขระ

ตัวอย่างที่ 21

โจทย์: จงเขียนโปรแกรมเพื่อสร้างผลรันออกทางหน้าจอตามที่กำหนดให้

ตัวอย่างผลรัน เมื่อใส่ 5 ผ่านทางแป้นพิมพ์

```
5
*****
*****
*****
*****
*****
```

ตัวอย่างผลรัน เมื่อใส่ 7 ผ่านทางแป้นพิมพ์

```
7
*******
*******
*******
*******
*******
*******
*******
```

1) เอาต์พุต คือ”กล่อง” ดอกจันแบบทึบ จำนวน n ตัวในหนึ่งบรรทัดและจำนวน n บรรทัด
2) อินพุต คือจำนวนเต็ม n
3) โจทย์ไม่ได้กำหนดอะไรให้เพิ่มเติม
4) ไม่มีข้อมูลเพิ่มเติม

จากตัวอย่างที่กำหนดให้ เราเห็นได้ว่าหากผู้ใช้ใส่ 5 ผ่านทางแป้นพิมพ์ เราจะได้ดอกจัน 5 ตัวในหนึ่งบรรทัด และมีจำนวน 5 บรรทัด แต่หากผู้ใช้ใส่ 7 ผ่านทางแป้นพิมพ์ เราจะได้ดอกจัน 7 ตัวในหนึ่งบรรทัด และมีจำนวน 7 บรรทัด แสดงว่าค่าที่ผู้ใช้ใส่เข้ามาเป็นตัวควบคุมจำนวนบรรทัด และจำนวนดอกจันในหนึ่งบรรทัด

โจทย์ข้อนี้เราสามารถใช้การวนซ้ำแบบซ้อนได้ โดยเราจะกำหนดให้ลูปชั้นนอกเป็นตัวควบคุมจำนวนบรรทัดที่ต้องวนซ้ำ และกำหนดในลูปชั้นในเป็นตัวควบคุมจำนวนดอกจันที่ต้องวนซ้ำในแต่ละบรรทัด ดังโค้ดต่อไปนี้

โค้ด:

```
int main(){
      cin >> n;
      for(int x = 1; x <= n; x++){
            for(int y = 1; y <= n; y++){
                cout << '*';
            }
            cout << endl;
      }
      return 0;
}
```

ตัวอย่างที่ 22

โจทย์: จงเขียนโปรแกรมเพื่อสร้างผลรันออกทางหน้าจอตามที่กำหนดให้

ตัวอย่างผลรัน เมื่อใส่ 5 ผ่านทางแป้นพิมพ์

```
5
*****
*   *
*   *
*   *
*****
```

ตัวอย่างผลรัน เมื่อใส่ 7 ผ่านทางแป้นพิมพ์

```
7
*******
*     *
*     *
*     *
*     *
*     *
*******
```

1) เอาต์พุต คือ”กล่อง” ดอกจันแบบโปร่ง จำนวน n ตัวในหนึ่งบรรทัดและจำนวน n บรรทัด
2) อินพุต คือจำนวนเต็ม n
3) โจทย์ไม่ได้กำหนดอะไรให้เพิ่มเติม
4) ไม่มีข้อมูลเพิ่มเติม

จากตัวอย่างที่กำหนดให้ เราเห็นได้ว่าหากผู้ใช้ใส่ 5 ผ่านทางแป้นพิมพ์ เราจะได้ดอกจัน 5 ตัวในบรรทัดแรกและบรรทัดสุดท้าย และ 2 ตัวที่ขอบกล่องในบรรทัดอื่นๆ (ตัวที่ไม่ใช่ดอกจันคือ space หรือช่องว่าง) และมีจำนวน 5 บรรทัด แต่หากผู้ใช้ใส่ 7 ผ่านทางแป้นพิมพ์ เ ราจะได้ดอกจัน 7 ตัวในบรรทัดแรกและบรรทัดสุดท้าย และ 2 ตัวที่ขอบกล่องในบรรทัดอื่นๆ (ตัวที่ไม่ใช่ดอกจันคือ space หรือช่องว่าง) และมีจำนวน 7 บรรทัด แสดงว่าค่าที่ผู้ใช้ใส่เข้ามาเป็นตัวควบคุมจำนวนบรรทัด และจำนวนดอกจันและจำนวนช่องว่างในหนึ่งบรรทัด

โจทย์ข้อนี้เราสามารถใช้การวนซ้ำแบบซ้อนได้แบบข้อที่แล้ว โดยเราจะกำหนดให้ลูปชั้นนอกเป็นตัวควบคุมจำนวนบรรทัดที่ต้องวนซ้ำ และกำหนดในลูปชั้นในเป็นตัวควบคุมดอกจันและช่องว่างที่ต้องวนซ้ำในแต่ละบรรทัด (ควบคุมการพิมพ์ออกในแต่ละคอลัมน์นั่นเอง) เราจะเห็นได้ว่าในข้อนี้ต้องมีการกำหนดเงื่อนไข นั่นคือถ้าเป็นบรรทัดแรกหรือบรรทัดสุดท้าย เราจะพิมพ์ดอกจันออกไปตามปกติ แต่ถ้าเป็นบรรทัดอื่นนั้น เราต้องดูว่าคอลัมน์ใดพิมพ์อะไร ซึ่งก็คือว่าถ้าเป็นคอลัมน์แรกและคอลัมน์สุดท้าย เราจะพิมพ์ดอกจันออก แต่ถ้าเป็นคอลัมน์อื่นๆ เราจะพิมพ์ช่องว่างออกไป (ไม่ใช่ไม่พิมพ์เลย) ดังโค้ดต่อไปนี้

โค้ด:

```
int main(){
```

```
      cin >> n;
      for(int x = 1; x <= n; x++){
            for(int y = 1; y <= n; y++){
               if(x==1 || x==n || y==1 || y==n)
                  cout << '*';
               else
                  cout >> ' ';
            }
            cout << endl;
      }
      return 0;
}
```

การควบคุมการวนซ้ำด้วยคำสั่ง break, continue และ goto

นอกจากจะรอให้ลูปหยุดโดยรอให้เงื่อนไขเป็นเท็จแล้ว เราสามารถจะควบคุมลูปให้หยุดหรือกระโดดข้ามคำสั่งต่างๆ ได้ โดยใช้คำสั่ง break, continue และ goto โดยคำสั่งต่างๆ ทำหน้าที่ดังนี้

1) คำสั่ง break จะหยุดการทำงานของลูป ข้ามรอบที่เหลือในลูปชั้นนั้นทั้งหมดและทำคำสั่งที่อยู่ต่อจากลูปชั้นนั้น
2) คำสั่ง continue จะข้ามคำสั่งที่เหลือในลูปชั้นในรอบนั้น เพื่อเริ่มลูปรอบใหม่ทันที
3) คำสั่ง goto จะกระโดดไปทำคำสั่งที่ปลายทางทันที

เพื่อให้เห็นความแตกต่างของคำสั่ง break และ continue ให้พิจารณาโค้ดต่อไปนี้

โค้ด:

```
1  int main() {
2        for (int i = 0; i<10;i++) {
3              cout << "Top half :" << i << endl;
4              if (i > 5 ) break;
5              cout << "Bottom half:" << i << endl;
6        }
7        cout << "Outside of loop.";
         return 0;
   }
```

โค้ดนี้ทำการวนซ้ำจาก i=0 ถึง i=9 โดยจะแสดงข้อความว่า Top half: ต่อด้วยค่า i และ Bottom half: ต่อด้วยค่า i แต่ถ้า i มากกว่า 5 (ในที่นี้คือ i=6) โปรแกรมก็จะแสดงแค่ Top half: ต่อด้วยค่า 6 แล้วทำการจบลูปที่บรรทัดที่ 4 และออกไปทำคำสั่งต่อจากลูปในบรรทัดที่ 7 และจบโปรแกรม เราก็จะได้ผลรันดังนี้

ผลการทำงาน:

```
Top half :0
Bottom half:0
Top half :1
Bottom half:1
Top half :2
Bottom half:2
Top half :3
Bottom half:3
Top half :4
Bottom half:4
Top half :5
Bottom half:5
Top half :6
Outside of loop.
```

ถ้าหากเราแทนที่คำสั่ง break ด้วยคำสั่ง continue โปรแกรมแค่ไม่ทำคำสั่งที่เหลือในรอบที่ i > 5 (คือ i=6, 7, 8, 9) โดยจะแสดงแค่ Top half: ต่อด้วยค่า i ในรอบเหล่านั้น แล้วเริ่มรอบใหม่เลย ดังนั้นโปรแกรมจะได้ผลรันดังนี้

ผลการทำงาน:

```
Top half :0
Bottom half:0
Top half :1
Bottom half:1
Top half :2
```

```
Bottom half:2
Top half :3
Bottom half:3
Top half :4
Bottom half:4
Top half :5
Bottom half:5
Top half :6
Top half :7
Top half :8
Top half :9
Outside of loop.
```

ดังนั้น เราจึงต้องระวังในการใช้คำสั่ง continue เพราะมันจะไม่ได้ทำให้ลูปนั้นจบโดยสิ้นเชิง แต่ให้เริ่มการวนซ้ำรอบใหม่ อย่างเช่นถ้าหากเราเปลี่ยนโค้ดเป็นดังนี้

โค้ด:

```
1  int main() {
2        int i = 0;
3        while( i < 10) {
               cout << "Top half :" << i << endl;
4              if (i > 5 ) continue;
5              cout << "Bottom half:" << i << endl;
6              i++;      // be careful of this
         }
7        cout << "Outside of loop.";
8        return 0;
9  }
10
```

เมื่อ i มีค่ามากกว่า 5 คือ i = 6 โปรแกรมจะไม่ทำคำสั่งที่เหลือ ซึ่งรวมถึงการเพิ่มค่า i ในบรรทัดที่ 7 ทำให้ค่า i ค้างอยู่ที่ 6 และทำให้ลูปนี้วนซ้ำไปเรื่อยๆ ไม่มีที่สิ้นสุด เพราะ 6 มีค่าน้อยกว่า 10 เสมอ

ตัวอย่างที่ 23 การหยุดลูปชั้นเดียวด้วยคำสั่ง break และ continue

โจทย์ : จงหาผลรันของโปรแกรมต่อไปนี้ เมื่อป้อน 4 5 6 และ 9 ผ่านทางแป้นพิมพ์ตามลำดับ

โค้ด:

```
1   int main(){
2         int n;
3         for(;;){
4               cout << "Enter int: ";
5               cin >> n;
6               if(n % 2 == 0) continue;
7               if(n % 3 == 0) break;
8               cout << "\tBottom of loop.\n";
9         }
10        cout << "\tOutside of loop.\n";
11        return 0;
12  }
```

ในตัวอย่างนี้มีทั้งคำสั่ง break และ continue ที่อยู่ในลูปไม่รู้จบ (infinite loop) โดยใช้คำสั่ง for แบบไม่มีเงื่อนไข เรามาไล่ดูการทำงานของโปรแกรมกันไปทีละอินพุต อินพุตแรกคือ 4 เมื่อ n เป็น 4 จะทำให้เงื่อนไข n%2 == 0 เป็นจริง โปรแกรมจะทำคำสั่ง continue ซึ่งแปลว่าให้ข้ามทุกอย่างที่เหลือในลูปรอบนั้น แล้วเริ่มทำลูปรอบใหม่ต่อไป โปรแกรมก็จะกลับไปทำบรรทัดที่ 4 อีกครั้งและรับอินพุตตัวใหม่ อินพุตตัวที่สองคือ 5 เมื่อ n เท่ากับ 5 ก็จะไม่เข้าเงื่อนไขใดเลย เพราะฉะนั้นโปรแกรมก็จะพิมพ์ Bottom of loop. ออกทางหน้าจอ เป็นอันจบลูปในรอบนี้ จากนั้นเมื่อขึ้นรอบใหม่ อินพุตเป็น 6 ก็จะเข้าเงื่อนไขในบรรทัดที่ 6 อีกครั้ง โปรแกรมก็จะข้ามทุกอย่างที่เหลือในลูปรอบนั้น แล้วเริ่มลูปรอบใหม่ แล้วรับอินพุตคือ 9 อินพุตเท่ากับ 9 จะทำให้เงื่อนไข n%3 == 0 เป็นจริง โปรแกรมจะทำคำสั่ง break นั่นคือการหยุดการวนซ้ำของลูปนั้น ข้ามรอบที่เหลือทั้งหมด แล้วไปทำคำสั่งหลังจากการจบลูปนั้นคือพิมพ์ Outside of loop. ออกทางหน้าจอ

ผลการทำงาน:

```
Enter int: 4
```

```
Enter int: 5
      Bottom of loop.
Enter int: 6
Enter int: 9
      Outside of loop.
```

ตัวอย่างที่ 24 การหยุดลูปสองชั้นด้วยคำสั่ง break

โจทย์ : จงหาผลรันของโปรแกรมต่อไปนี้

โค้ด:

```
1   #include <iomanip>
2
3   int main(){
4         for(int x = 1; x <= 12; x++){
5               for(int y = 1; y <= 12; y++){
6                   if(y > x) break;
7                   else
8                     cout << setw(4) << x * y;
9               }
10              cout << endl;
11        }
12        return 0;
13  }
```

โค้ดในข้อนี้คล้ายกับโค้ดใน

ตัวอย่างที่ **20** ยกเว้นมีคำสั่ง break ในลูปชั้นในเมื่อ y มากกว่า x นั่นคือเราจะหยุดทำลูปชั้นในเมื่อ y มากกว่า x แล้วขึ้นบรรทัดใหม่ ตามคำสั่ง cout << endl; ในบรรทัดที่ 10 ซึ่งเป็นคำสั่งต่อจากการวนลูปชั้นใน จากนั้นเราจะทำลูปชั้นนอกในรอบต่อไป ผลจากรันจึงได้ดังนี้

ผลการทำงาน:

```
 1
 2   4
 3   6   9
 4   8  12  16
 5  10  15  20  25
 6  12  18  24  30  36
 7  14  21  28  35  42  49
 8  16  24  32  40  48  56  64
 9  18  27  36  45  54  63  72  81
10  20  30  40  50  60  70  80  90 100
11  22  33  44  55  66  77  88  99 110 121
12  24  36  48  60  72  84  96 108 120 132 144
```

เราลองมาไล่โปรแกรมกันดูสัก 2-3 รอบ ในรอบแรกของลูปชั้นนอก x เท่ากับ 1 จากนั้นก็เข้าลูปชั้นในเริ่มที่ y เท่ากับ 1 ซึ่งค่า y <= x จึงแสดงผลคูณของ x และ y ออกทางหน้าจอ ต่อมา y เท่ากับ 2 ซึ่งทำให้ y > x ลูปชั้นในจึงต้องหยุดทำที่บรรทัดที่ 6 และไปเคาะบรรทัดที่บรรทัดที่ 10 แล้วเริ่มลูปชั้นนอกรอบที่ 2 ซึ่ง x เท่ากับ 2 และเริ่มลูปชั้นในใหม่อีกครั้ง โดยเริ่มจาก y เท่ากับ 1 ไปเรื่อยๆ จน y เท่ากับ 3 ซึ่งทำให้ y > x ลูปชั้นในจึงต้องหยุดทำที่บรรทัดที่ 6 และไปเคาะบรรทัดที่บรรทัดที่ 10 แล้วเริ่มลูปชั้นนอกรอบที่ 3 ซึ่ง x เท่ากับ 3 และเริ่มลูปชั้นในใหม่อีกครั้ง โดยเริ่มจาก y เท่ากับ 1 ไปเรื่อยๆ จน y เท่ากับ 4 ซึ่งทำให้ y > x ลูปชั้นในจึงต้องหยุดทำที่บรรทัดที่ 6 และไปเคาะบรรทัดที่บรรทัดที่ 10 แล้วเริ่มลูปชั้นนอกรอบที่ 4 ต่อไป

ตัวอย่างที่ 25 การหยุดลูปสามชั้นด้วยคำสั่ง goto

โจทย์ : จงหาผลรันของโปรแกรมต่อไปนี้

โค้ด:

```
1  int main(){
2        const int N = 5;
```

```
3       for(int i = 0; i < N; i++){
4             for(int j = 0; j < N; j++){
5                   for(int k = 0; k < N; k++){
6                         if(i + j + k > N) goto esc;
7                         else cout << i + j + k << " ";
8                   } // for k
9                   cout << "* ";
10            } // for j
11            esc: cout << "." << endl;
12      } // for i
13      return 0;
14  }
```

โจทย์ข้อนี้มีลูป 3 ชั้น คือชั้นนอก (ควบคุมโดย i) ชั้นกลาง (ควบคุมโดย j) และชั้นใน (ควบคุมโดย k) และมีการใช้คำสั่ง goto ที่ลูปชั้นในในบรรทัดที่ 6 ซึ่งเป็นคำสั่งให้กระโดดไปทำคำสั่งที่ปลายทาง ในที่นี้คือส่วนของโปรแกรมที่ชื่อว่า esc ซึ่งอยู่ในลูปชั้นนอกที่บรรทัดที่ 11 เงื่อนไขในการกระโดดคือเมื่อผลรวมของ i, j และ k นั้นมากกว่า N ก็ให้ออกไปที่ esc ซึ่งหมายความว่าให้หยุดทำลูปชั้นในและกลางไปเลย แล้วพิมพ์จุด (.) ในบรรทัดที่ 11 ก่อนเริ่มลูปชั้นนอกรอบใหม่ สำหรับลูปชั้นในถ้าผลรวมของ i, j และ k ไม่มากกว่า N เราจะพิมพ์ผลรวมออกและพอหมดลูปชั้นในแล้ว โปรแกรมจะพิมพ์เครื่องหมายดอกจัน (*) ในบรรทัดที่ 8 ซึ่งเป็นคำสั่งของลูปชั้นกลางก่อนจะเริ่มลูปชั้นกลางรอบต่อไป เราจะไล่ค่าของตัวแปรในโปรแกรมให้ดูดังนี้

ตารางที่ 7 ค่าของ i, j, k, i+j+k และผลที่แสดงออกทางหน้าจอในแต่ละรอบของลูป 3 ชั้น

ค่า i	ค่า j	ค่า k	i + j + k	cout
0	0	0	0	0
0	0	1	1	1
0	0	2	2	2
0	0	3	3	3
0	0	4	4	4
		จบลูปชั้นในเพราะ k >= N		*
0	1	0	1	1

ค่า i	ค่า j	ค่า k	i + j + k	cout
0	1	1	2	2
0	1	2	3	3
0	1	3	4	4
0	1	4	5	5
		จบลูปชั้นในเพราะ k >= N		*
0	2	0	2	2
0	2	1	3	3
0	2	2	4	4
0	2	3	5	5
0	2	4	6 (i+j+k > N)	. (เคาะบรรทัด)
	จบลูปชั้นกลางเพราะโดนข้ามไป	จบลูปชั้นในเพราะโดนข้ามไป		
1	0	0	1	1
1	0	1	2	2
1	0	2	3	3
1	0	3	4	4
1	0	4	5	5
		จบลูปชั้นในเพราะ k >= N		*
1	1	0	2	2
1	1	1	3	3
1	1	2	4	4
1	1	3	5	5
1	1	4	6 (i+j+k > N)	. (เคาะบรรทัด)

ค่า i	ค่า j	ค่า k	i + j + k	cout
	จบลูปชั้นกลาง เพราะโดนข้ามไป	จบลูปชั้นใน เพราะโดนข้ามไป		
2	0	0	2	2
2	0	1	3	3
2	0	2	4	4
2	0	3	5	5
2	0	4	6 (i+j+k > N)	. (เคาะบรรทัด)
	จบลูปชั้นกลาง เพราะโดนข้ามไป	จบลูปชั้นใน เพราะโดนข้ามไป		
3	0	0	3	3
3	0	1	4	4
3	0	2	5	5
3	0	3	6 (i+j+k > N)	. (เคาะบรรทัด)
	จบลูปชั้นกลาง เพราะโดนข้ามไป	จบลูปชั้นใน เพราะโดนข้ามไป		
4	0	0	4	4
4	0	1	5	5
4	0	2	6 (i+j+k > N)	. (เคาะบรรทัด)
	จบลูปชั้นกลาง เพราะโดนข้ามไป	จบลูปชั้นใน เพราะโดนข้ามไป		
จบลูปชั้นนอกเพราะ i >= N				

จากค่าที่ไล่มาในตาราง เราได้ผลรันดังนี้

ผลการทำงาน:

```
0 1 2 3 4 * 1 2 3 4 5 * 2 3 4 5 .
1 2 3 4 5 * 2 3 4 5 .
2 3 4 5 .
3 4 5 .
4 5 .
```

การใช้ธง (flag) ในการหยุดลูป

นอกจากจะหยุดการวนซ้ำด้วยเงื่อนไข คำสั่ง break, continue และ goto แล้ว เรายังสามารถหยุดลูปโดยการใช้ธง (flag) ซึ่งโดยปกติเราจะใช้ตัวแปรชนิดบูลีนที่มีค่าจริงหรือเท็จเป็นธง เช่น ถ้าเราเจอสิ่งที่เราต้องการแล้ว เราก็ตั้งค่าธงของเราให้เป็นจริง ตามตัวอย่างต่อไปนี้

ตัวอย่างที่ 26

โจทย์ : จงหาผลรันของโปรแกรมต่อไปนี้

โค้ด:

```
1  int main(){
2        const int N = 5;
3        bool done = false; // done is a flag
4        for(int i = 0; i < N; i++){
5              for(int j = 0; j < N && !done; j++){
6                    for(int k = 0; k < N && !done; k++){
7                          if(i + j + k > N) done = true;
8                          else cout << i + j + k << " ";
9                    } // for k
10                   cout << "* ";
11             } // for j
```

```
12              cout << "." << endl;
13              done = false; // reset flag
14        } // for i
15        return 0;
16  }
```

อันที่จริง ผลรันของตัวอย่างนี้เหมือนกับผลรันของตัวอย่างที่แล้ว เพียงแต่มีวิธีหยุดลูปที่ต่างกัน ในตัวอย่างที่แล้ว ลูปหยุดด้วยคำสั่ง goto แต่ในโค้ดนี้เราหยุดลูปด้วยการใช้ธง (flag) นั่นคือเรากำหนดตัวแปรชนิดบูลีน (boolean) ขึ้นมาหนึ่งตัวแล้วตั้งค่าเป็นเท็จ (false) ก่อน และในการวนซ้ำเราจะตั้งเงื่อนไขว่ายังให้วนทำซ้ำอยู่ตราบใดที่ตัวแปรธงตัวนี้ยังเป็นเท็จ เมื่อเราเจอเงื่อนไขที่จะต้องหยุดลูป เราจะเปลี่ยนค่าของตัวแปรธงตัวนี้เป็นจริง (true) ก็จะทำให้ในรอบต่อไปนั้นจะไม่มีการวนซ้ำอีก

อย่างไรก็ตาม ข้อควรระวังในการใช้ตัวแปรธงนี้คือ การรีเซ็ตค่าตัวแปรให้กลับเป็นเท็จเหมือนเดิมเมื่อจำเป็น เราจะต้องวางให้ถูกที่ด้วย ไม่เช่นนั้นลูปก็จะไม่หยุด หรือหยุดผิดจังหวะ ไม่เป็นไปตามที่เราต้องการ

สรุปคำสั่งในการหยุดลูป

คำสั่งที่หยุดการทำงานของลูป มีดังนี้

- break; หยุดลูปปัจจุบันไปเลย ข้ามรอบที่เหลือทั้งหมดและออกจากลูป
- exit(0); ออกจากฟังก์ชันปัจจุบันไปเลย
- return 0; ออกจากฟังก์ชันปัจจุบันไปเลย
- goto somewhere; กระโดดข้ามไปยังตำแหน่งที่กำหนดให้ทันที
- ใช้ตัวแปรธง done = true/false; วนซ้ำเป็นปกติ จึงต้องกำหนดให้เงื่อนไขการวนซ้ำขึ้นอยู่กับธงด้วย

คำสั่งที่เริ่มลูปรอบใหม่ทันที มีดังนี้

- continue; จบการทำลูปรอบนั้น ข้ามคำสั่งที่เหลือในรอบนั้น แล้วเริ่มลูปรอบใหม่
- goto somewhere; กระโดดข้ามไปยังตำแหน่งที่กำหนดให้ทันที

สรุปสิ่งที่ควรจะเข้าใจในบทเรียน

1) การวนซ้ำหรือการลูป คือการทำคำสั่งหรือบล็อคของคำสั่งซ้ำไปซ้ำมาจนกว่าเงื่อนไขในการวนซ้ำจะเป็นเท็จ
    a. วนซ้ำเป็นจำนวนรอบที่แน่นอน จะมีการใช้ตัวนับ (counter) และการปรับเปลี่ยนค่าของตัวนับในแต่ละรอบ
    b. วนซ้ำโดยไม่กำหนดรอบที่แน่นอน มีการวนซ้ำไปเรื่อยๆ จนกว่าจะได้ค่าของตัวแปรที่ต้องการ จะต้องมีการปรับค่าของตัวแปรที่เป็นเงื่อนไข อาจจะโดยการคำนวณต่างๆ
    c. วนซ้ำโดยไม่กำหนดรอบที่แน่นอน มีการวนซ้ำไปเรื่อยๆ โดยการกำหนดเงื่อนไขจากค่าที่ผู้ใช้รับผ่านทางแป้นพิมพ์
2) คำสั่งในการวนซ้ำในภาษา C++ คือ
    a. while
    b. do while
    c. for
3) ถ้าเงื่อนไขในการวนซ้ำเป็นจริงเสมอ หรือไม่ได้กำหนดเงื่อนไข เช่น while(true) และ for(; ;) เราสามารถทำให้ลูปหยุดได้โดยคำสั่ง break, exit(0), return หรือ goto