

บทที่ 4-2

การควบคุมทิศทางการทำงานของโปรแกรม
การทำซ้ำแบบตรวจสอบเงื่อนไข (Repetition)

ทำไมต้องมีคำสั่งทำซ้ำ (Repetition)

- หากเราต้องการเขียนโปรแกรม เพื่อ แสดงข้อความ จำนวน **100** ครั้ง

Welcome to C

OH GOD



WHY

```
printf("Welcome to C\n");
```

```
printf("Welcome to C\n");
```

...

```
printf("Welcome to C\n");
```

```
printf("Welcome to C\n");
```

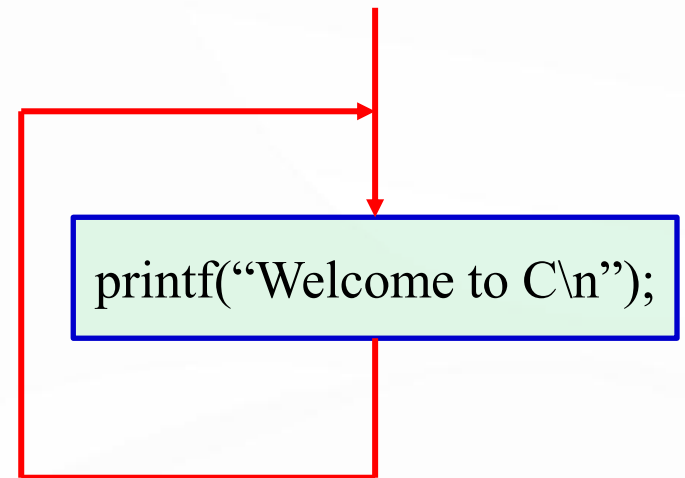
100 ครั้ง

Repetition in C Programming

คอมพิวเตอร์สามารถทำงานซ้ำๆ ด้วย statement ชุดหนึ่งได้ โดยใช้คำสั่งควบคุมให้ทำงานซ้ำบางครั้งเรียกว่า ลูป (Loop)

```
printf("Welcome to C\n");  
printf("Welcome to C\n");  
...  
printf("Welcome to C\n");  
printf("Welcome to C\n");
```

ไม่มีการทำงานซ้ำ



การทำงานซ้ำ

Repetition in C Programming

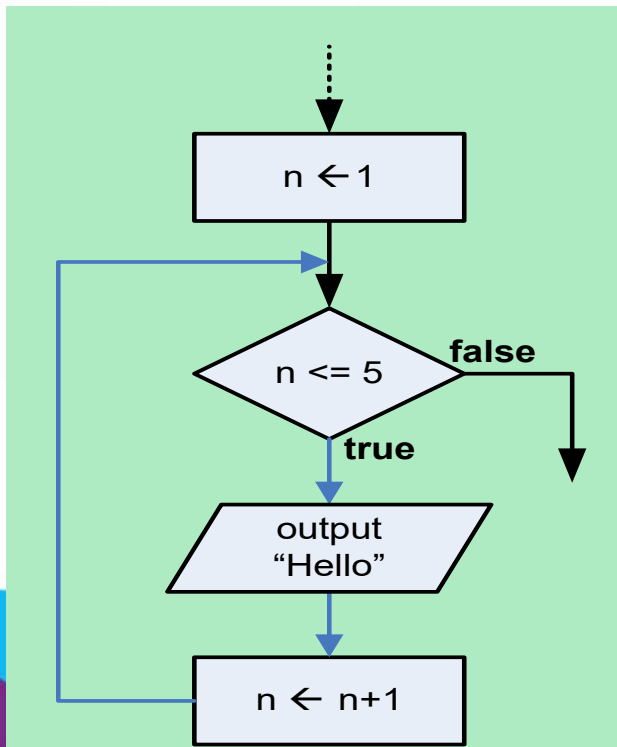
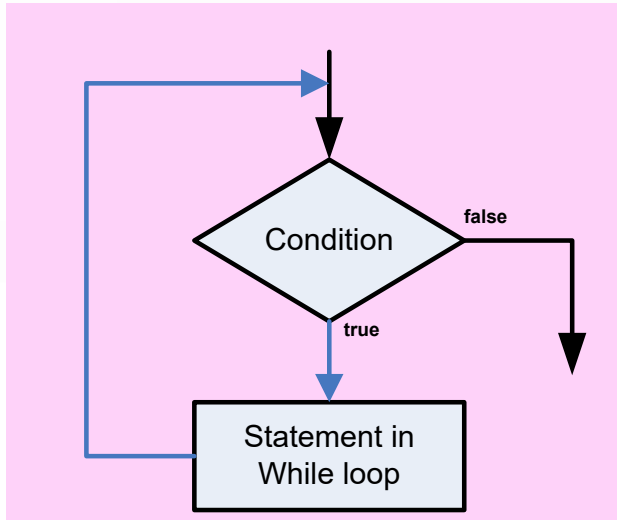
วงรอบ (Loop) ในภาษาการโปรแกรมคอมพิวเตอร์ หมายถึงลำดับของคำสั่ง
ในภาษาโปรแกรมที่ทำงานซ้ำต่อเนื่อง จนกว่าเงื่อนไขในการนั้นจะสิ้นสุด

- ภาษา C มีคำสั่งเพื่อใช้สำหรับควบคุมการทำซ้ำ ดังนี้
 - คำสั่ง **while**
 - คำสั่ง **do-while**
 - คำสั่ง **for**

1. คำสั่ง while

■ โครงสร้างของคำสั่ง while

- **Condition** คือ เงื่อนไขที่ใช้ในการเปรียบเทียบ ถ้าผลลัพธ์ของการเปรียบเทียบเป็น**จริง** จะมีการเข้าไป**ทำงาน**คำสั่งในวงรอบ
- statement คือ คำสั่งที่จะถูกกระทำเมื่อเงื่อนไขการเปรียบเทียบเป็น**จริง**
- คำสั่ง while จะต้องมีการกำหนดค่าเริ่มต้นให้กับ **index** (ในที่นี้คือตัวแปร n) **ก่อนที่จะเข้า loop** ดังตัวอย่าง คือ **$n \leftarrow 1$**
- ในวงรอบจะต้องมีคำสั่งสำหรับ **update** ค่า index เพื่อให้เงื่อนไขมีแนวโน้ม**จบวงรอบได้** ในที่นี้คือ **$n \leftarrow n+1$**



คำสั่ง while

■ ไวยากรณ์ของคำสั่ง while

`while (condition)`

`Statement ;`

ตัวอย่างเช่น

```
n = 1;
while (n <= 5)
{
    printf("%d. RMUTT\n", n);
    n++;
}
```

ผลลัพธ์การทำงาน

1. RMUTT
2. RMUTT
3. RMUTT
4. RMUTT
5. RMUTT

- คำสั่ง while สามารถทำซ้ำได้เพียงหนึ่งคำสั่งเท่านั้น ถ้าต้องการให้ทำซ้ำมากกว่าหนึ่งคำสั่ง ต้องใส่คำสั่งเหล่านั้นไว้ภายใต้เครื่องหมาย { }

ตัวอย่างการใช้งาน while

ตัวอย่างโปรแกรม สำหรับการคำนวณผลรวมของ 1+...+10

```
1  #include <stdio.h>
2  int main(void)
3  {
4      int  sum, n;
5      sum = 0;
6      n = 1;
7      while (n <= 10)
8      {
9          sum = sum + n;
10         n++;
11     }
12     printf("Summation of 1-10 is : %d", sum);
13     return(0);
14 }
```

ผลลัพธ์การทำงาน

Summation of 1-10 is 55

ตัวอย่างการใช้งาน while



ข้อควรระวัง ห้ามใช้ค่าจำนวนจริง (floating-point) สำหรับการตรวจสอบเงื่อนไข while-loop และ do-while ในกรณีที่ $=$, $!=$ 0

ตัวอย่างโปรแกรม สำหรับการคำนวณผลรวมของ $1+0.9+0.8+\dots+0.1$

```
1  #include <stdio.h>
2  int main(void)
3  {
4      float  sum, n;
5      sum = 0;
6      n = 1;
7      while (n != 0) // ไม่การันตีว่า 0.9, 0.8, ..., 0.1 จะถูกมองเป็นค่า 0
8      {
9          sum = sum + n;
10         n = n - 0.1;
11     }
12     printf("Summation of 1-0.1 is: %f", sum);
13     return(0);
14 }
```

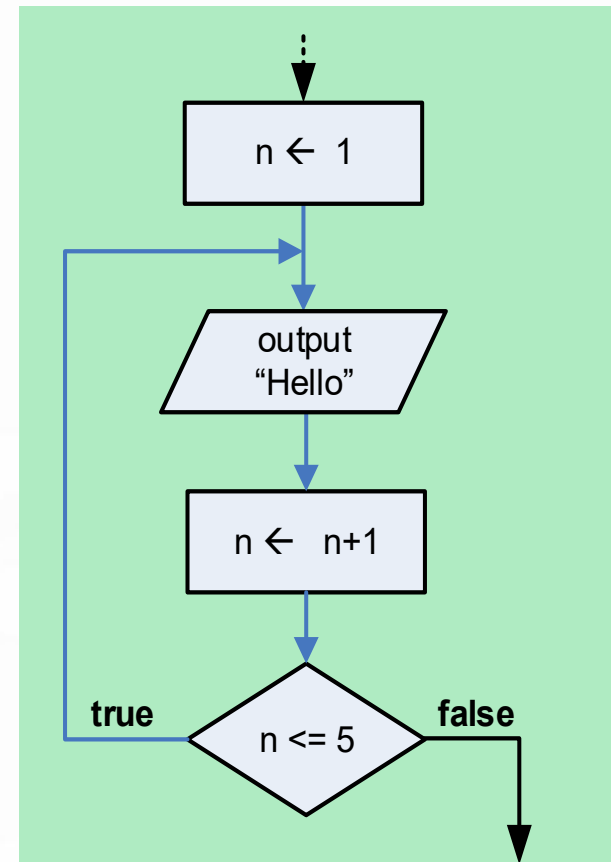
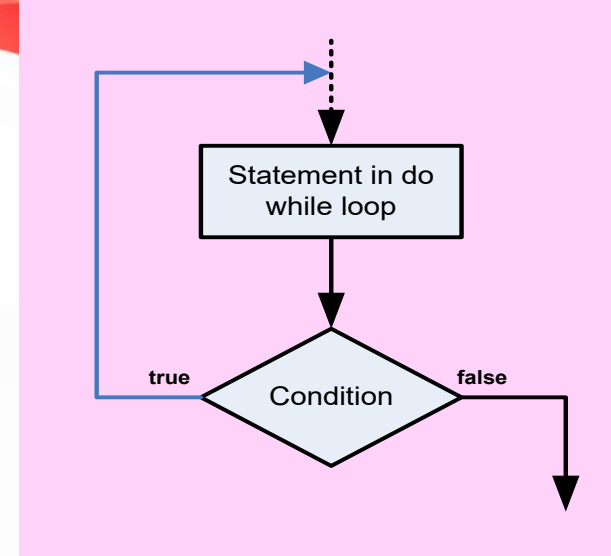
ผลลัพธ์การทำงาน

Summation of 1-0.1 is: 1.000000

2. คำสั่ง do-while

■ โครงสร้างของคำสั่ง do-while

- **Condition** คือ เงื่อนไขที่ใช้เปรียบเทียบซึ่งถ้าผลลัพธ์ของการเปรียบเทียบเป็นจริง จะมีการเข้าไปทำงานคำสั่งในวงรอบ ถ้าเป็นเท็จจะออกนอกวงรอบ
- statement คือ คำสั่งที่จะถูกกระทำเมื่อเงื่อนไขการเปรียบเทียบเป็นจริง (true)
- 😊 จุดเด่นของ do คำสั่งทำซ้ำที่จะถูกกระทำก่อนอย่างน้อยหนึ่งครั้ง while คือ ก่อนจะออกจากรอบ
- ในการ update และ initial ค่าเริ่มต้นยังคงเป็นหลักการเดียวกันกับคำสั่ง while



ตัวอย่างการใช้งาน do-while

■ ไวยากรณ์ของคำสั่ง do-while

```
do {  
    statement in do-while  
} while (condition);
```

ตัวอย่างเช่น

```
n = 1;  
do {  
    printf("%d. RMUTT\n", n);  
    n++;  
} while (n <= 5);
```

ผลลัพธ์การทำงาน

1. RMUTT
2. RMUTT
3. RMUTT
4. RMUTT
5. RMUTT

ตัวอย่างการใช้งาน do-while

■ ไวยากรณ์ของคำสั่ง do-while

ตัวอย่างเช่น

```
n = 1;   เมื่อ n มีค่าน้อยกว่าเท่ากับ 5 จะทำงานในลูป
do { ← -----
    printf("%d. RMUTT\n", n);
    n++;
} while (n <= 5); -----
```

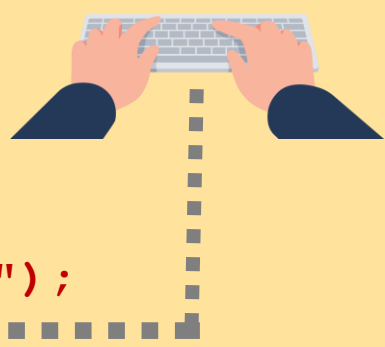
เมื่อ n มีค่ามากกว่า 5 จะไม่มีการทำงานในลูปอีก

ผลลัพธ์การทำงาน

1. RMUTT
2. RMUTT
3. RMUTT
4. RMUTT
5. RMUTT

ตัวอย่างการใช้งาน do-while

```
do {  
    printf("****Menu****\n");  
    printf(" 1. Show odd\n");  
    printf(" 2. Show even\n");  
    printf(" 3. Exit\n");  
    printf(" Please enter number [1-3] : ");  
    scanf("%d", &choice);  
} while (choice != 3);  
printf("End");
```



ผลลัพธ์การทำงาน

```
****Menu****  
1. Show odd  
2. Show even  
3. Exit  
Please enter number [1-3] : 1
```

ผลลัพธ์การทำงาน

```
****Menu****  
1. Show odd  
2. Show even  
3. Exit  
Please enter number [1-3] : _
```

- ❖ หมายเหตุ โปรแกรมจะวนไปรับข้อมูลไปเรื่อยๆ จนกว่า ตัวแปร choice จะมีค่าเท่ากับ 3 จึงจะออกจากการวนรอบ

สรุปการควบคุมทิศทางการทำงาน ของโปรแกรมการซ้ำ

- หลักการในการทำซ้ำนั้นจะประกอบไปด้วยสิ่งสำคัญ 3 สิ่ง คือ
 - การกำหนดค่าเริ่มต้นให้กับค่าดัชนี (Index)
 - เงื่อนไข (Condition) ที่จะตัดสินว่า จะให้ทำซ้ำในวงรอบหรือไม่
 - การปรับค่าดัชนี (Update index) ภายในวงรอบ

```
i = 1; ①
while (i <= 5) ②
{
    printf("RMUTT");
    i++; ③
}
```

```
i = 1; ①
do {
    printf("RMUTT");
    i++; ③
} while (i <= 5); ②
```

3. คำสั่ง for

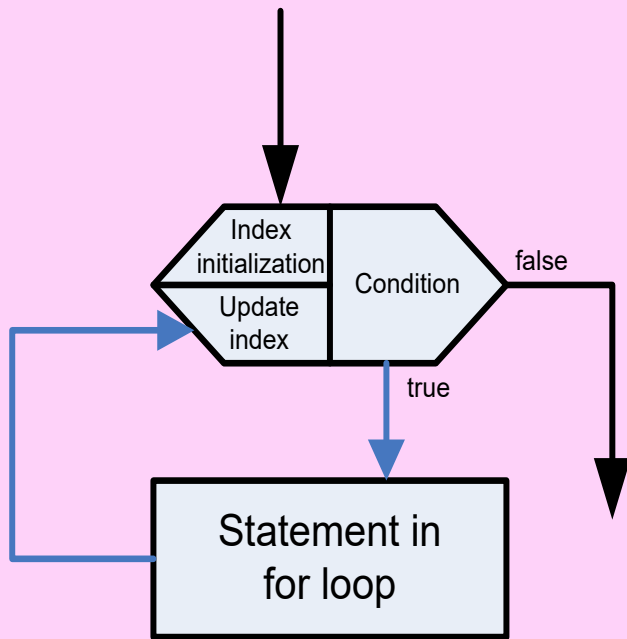
โครงสร้างของคำสั่ง for

- **Index initialization** คือ การกำหนดค่าเริ่มต้นให้กับตัวแปรดัชนี

ตัวแปรดัชนี (index) คือ ตัวที่ใช้ควบคุมการทำงานในรอบ

- **Update index** คือ การเพิ่มค่า หรือลดค่า index เพื่อให้ผลของการเปรียบเทียบเงื่อนไข (condition) มีแนวโน้มที่จะเป็นเท็จได้ เช่น $n++$ หรือ $n--$

- **Condition** คือ เงื่อนไขในการเปรียบเทียบ เช่น $n \leq 9$



คำสั่ง for

■ ไวยากรณ์ของคำสั่ง for

for (index initialization ; Condition ; Update Index)
Statement ;

ตัวอย่างเช่น

```
for (n = 1; n <= 10; n++)  
    printf("RMUTT\n");
```

หรือ

```
for (n = 10; n >= 1; n--)  
    printf("RMUTT\n");
```

ผลลัพธ์ของ Loop นี้จะแสดงคำว่า
RMUTT ทั้งหมด 10 ครั้ง
(รอบที่ 1-10)

ผลลัพธ์ของ Loop นี้จะแสดงคำว่า
RMUTT ทั้งหมด 10 ครั้ง
(รอบที่ 10-1)

คำสั่ง for

- คำสั่ง for สามารถทำซ้ำได้เพียงหนึ่งคำสั่งเท่านั้น ถ้าต้องการให้ทำซ้ำมากกว่าหนึ่งคำสั่งต้องใส่คำสั่งเหล่านั้นไว้ภายใต้ Compound Statement โดยใช้เครื่องหมาย { }

```
for (n = 0; n <= 9; n++)  
{  
    printf("Hello");  
    printf("World");  
}
```

- ในทางเดียวกัน ถ้าใส่เครื่องหมาย ; หลังเครื่องหมายวงเล็บ) ของ for ดังตัวอย่าง

```
for (n=0; n<=9; n++) ;
```

จะทำให้ไม่เกิดการซ้ำคำสั่งต่อจากนี้ เนื่องจาก ; หมายถึง จบหนึ่งคำสั่ง ดังนั้น คำสั่งที่ถูกทำซ้ำคือ คำสั่งที่อยู่ก่อนหน้า ; ซึ่งก็คือ ไม่มีคำสั่งใด ๆ นั่นเอง

ตัวอย่างการใช้งาน for

```
1  #include <stdio.h>
2  int main(void)
3  {
4      int i;
5      for (i=1; i<100; i++)
6      {
7          if ((i%2)== 0)
8              printf("This is even number (%d)\n", i);
9      }
10     return(0);
11 }
12
```

ผลลัพธ์การทำงาน

This is even number (2)

This is even number (4)

This is even number (8)

This is even number (10)

..

This is even number (100)

ตัวอย่างการใช้งาน for กับ continue

```
1  #include <stdio.h>
2  int main(void)
3  {
4  int i;
5  for (i=1; i<100; i++)
6  { if ((i%2)== 0){
7      continue;
8  }
9      printf("This is odd number (%d) \n", i);
10 }
11 return (0);
12 }
```

ผลลัพธ์การทำงาน

This is odd number (1)

.

.

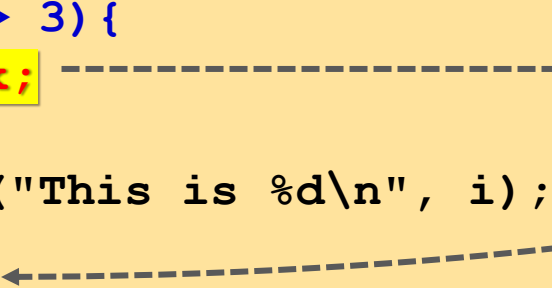
This is odd number (99)

หมายเหตุ

- คำสั่ง **continue** ทำหน้าที่กระโดดไปยังจุด update (i++) ทันทีโดยไม่สนใจคำสั่งต่อไป

ตัวอย่างการใช้งาน for กับ break

```
1  #include <stdio.h>
2  int main(void)
3  {
4  int i;
5  for (i=1; i<100; i++)
6      { if (i > 3){
7          break;
8      }
9      printf("This is %d\n", i);
10     }
11     return(0);
12 }
```



ผลลัพธ์การทำงาน

This is 1

This is 2

This is 3

หมายเหตุ

- คำสั่ง **break** ทำหน้าที่กระโดด **ออกจาก loop**

ตัวอย่างการใช้งาน for

```
1  #include <stdio.h>
2  int main(void)
3  {
4  int i;
5  for (i=1; i<100; i++)
6      { if ((i%2)== 0)
7          continue;
8          printf("This is odd number (%d) \n", i);
9          if (i>5) break;
10     }
11 return (0);
12 }
```

ผลลัพธ์การทำงาน

This is odd number (1)

This is odd number (3)

This is odd number (5)

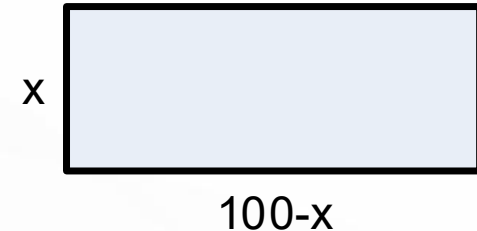
This is odd number (7)

หมายเหตุ

- คำสั่ง **continue** ทำหน้าที่กระโดดไปยังจุด update (i++) ทันทีโดยไม่สนใจคำสั่งต่อไป
- คำสั่ง **break** ทำหน้าที่กระโดดออกจาก loop

ตัวอย่างการใช้งาน for

มีลวดหนามยาว 200 เมตร จะล้อมลวดหนามแต่ละด้าน
อย่างไร ให้มีพื้นที่ที่มากที่สุด และพื้นที่นั้นมีขนาดเท่าไร



```
1  #include <stdio.h>
2  int main(void)
3  {
4      int x, area, max;
5      printf("Value of x      Area\n");
6      max = 0;
7      for (x=1; x<100; x++)
8      {
9          area = x*(100-x);
10         printf("%d          %d\n", x, area);
11         if (area > max) max = area;
12     }
13     printf("Maximum area is %d\n", max);
14     return(0);
15 }
```

ผลลัพธ์การทำงาน

Value of x	Area
1	99
...	...
10	900
11	979
...	...
50	2500
...	...
58	2436
...	...
99	99
Maximum area is 2500	
21	

การประยุกต์ใช้งาน Nested for

- ตัวอย่างการแสดงสูตรคูณแม่ 1 ถึงสูตรคูณแม่ 9

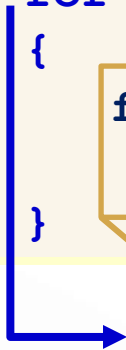
```
for (int m = 1; m <= 9; m++)
```

```
{
```

```
    for (int n = 1; n <= 9; n++)
```

```
        printf("%dx%d = %d\n", m, n, m*n);
```

```
}
```



①

n



$$1 \times 1 = 1$$

$$1 \times 2 = 2$$

.

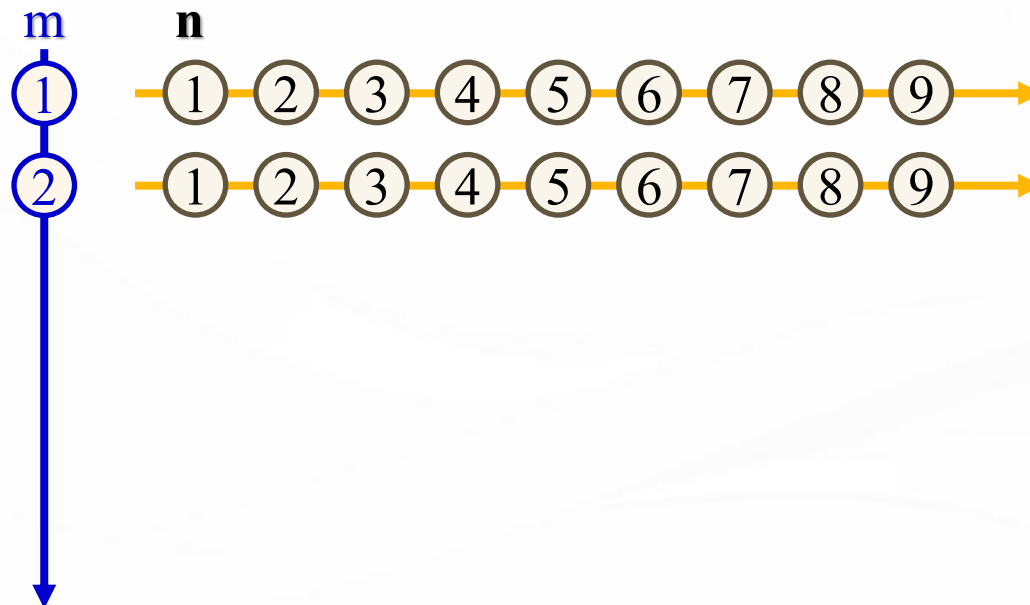
.

$$1 \times 9 = 9$$

การประยุกต์ใช้งาน Nested for

- ตัวอย่างการแสดงสูตรคูณแม่ 1 ถึงสูตรคูณแม่ 9

```
for (int m = 1; m <= 9; m++)  
{  
    for (int n = 1; n <= 9; n++)  
        printf("%dx%d = %d\n",m,n,m*n);  
}
```



$$1 \times 9 = 9$$

$$2 \times 1 = 2$$

.

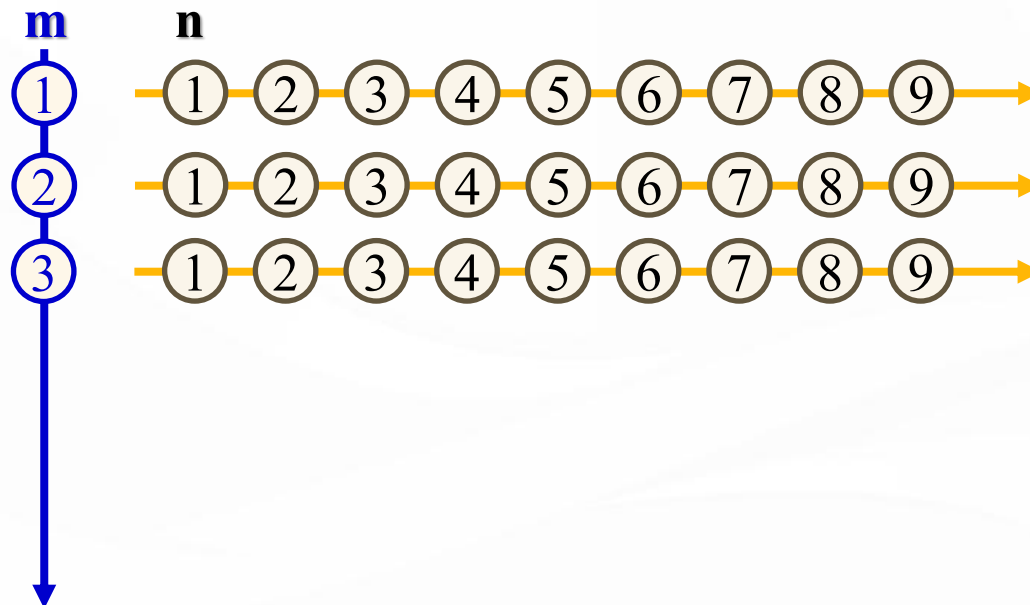
.

$$2 \times 9 = 18$$

การประยุกต์ใช้งาน Nested for

- ตัวอย่างการแสดงสูตรคูณแม่ 1 ถึงสูตรคูณแม่ 9

```
for (int m = 1; m <= 9; m++)  
{  
    for (int n = 1; n <= 9; n++)  
        printf("%dx%d = %d\n",m,n,m*n) ;  
}
```



$$2 \times 9 = 18$$

$$3 \times 1 = 2$$

.

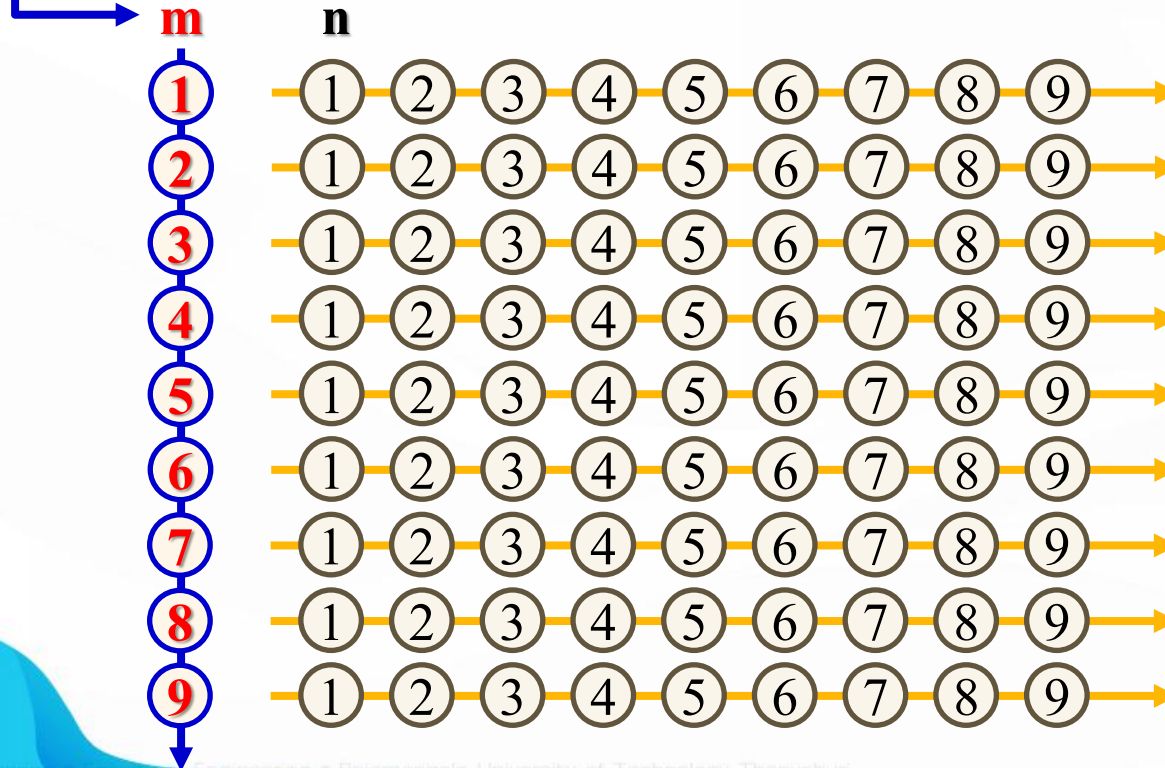
.

$$3 \times 9 = 27$$

การประยุกต์ใช้งาน Nested for

- ตัวอย่างการแสดงสูตรคูณแม่ 1 ถึงสูตรคูณแม่ 9

```
for (int m = 1; m <= 9; m++)  
{  
    for (int n = 1; n <= 9; n++)  
        printf("%dx%d = %d\n", m, n, m*n);  
}
```



$$8 \times 9 = 18$$

$$9 \times 1 = 9$$

.

.

$$9 \times 9 = 81$$

คำสั่ง for

- สามารถประกาศตัวแปรในส่วนของ index initialization ได้หลายตัว เช่น

for (index initialization; Condition; Update Index)
Statement ;

```
for (int x=0,y=0; x+y<30; ++x, y+=5)  
    printf("%d+%d = %d\n", x, y, x+y);
```

****ในที่นี้ Index มี 2 ตัวคือ ตัวแปร x และตัวแปร y ****

$$0+0 = 0$$

$$1+5 = 6$$

$$2+10 = 12$$

$$3+15 = 18$$

$$4+20 = 24$$

จากตัวอย่าง แสดงให้เห็นว่า ในส่วนของ index initialization สามารถมีตัวแปร
ดัชนี (index) ได้มากกว่า 1 ตัว และในส่วนของ condition ก็สามารถใส่นิพจน์
การคำนวณได้ด้วย

สรุปการควบคุมทิศทางการทำงานของโปรแกรม

การทำซ้ำ

- หลักการในการทำซ้ำนั้นจะประกอบไปด้วยสิ่งสำคัญ 3 สิ่ง คือ
 - การกำหนดค่าเริ่มต้นให้กับค่าดัชนี (Index)
 - เงื่อนไข (Condition) ที่จะตัดสินว่า จะให้ทำซ้ำในวงรอบหรือไม่
 - การปรับค่าดัชนี (Update index) ภายในวงรอบ

```
i = 1;
while (i <= 5)
{
    printf("RMUTT");
    i++;
}
```

```
i = 1;
do {
    printf("RMUTT");
    i++;
} while (i<=5);
```

```
for(i=1; i<=5; i++)
{
    printf("RMUTT");
}
```

จบบทที่ 4-2

การควบคุมทิศทางการทำงานของโปรแกรม
การทำซ้ำ (Repetition)