บทที่ 3 คำสั่งเงื่อนไขและคำสั่งควบคุม

ในการเขียนโปรแกรม นอกจากการเขียนคำสั่งในการคำนวณ ซึ่งหมายถึง การกำหนดตัวแปร การเลือกชนิดของข้อมูล และการใช้ตัวดำเนินการแล้ว เรายังต้องรู้วิธีการเขียนคำสั่งควบคุมให้โปรแกรมทำงานตามที่เราต้องการ อันได้แก่ การกำหนดเงื่อนไขต่างๆ การกำหนดเงื่อนไขในการทำซ้ำ(การวนลูป) ซึ่งเป็นโครงสร้างหลักของการเขียนโปรแกรมในลักษณะโครงสร้าง (Structured Programming)

3.1 คำสั่งเงื่อนไข if

คำสั่งเงื่อนไข (Conditional Statement หรือ Branching Statement) เป็นคำสั่งสำหรับกำหนดเงื่อนไขในการเขียนโปรแกรม ซึ่งเป็นโครงสร้างการเลือกของวิธีการเขียนโปรแกรมในลักษณะโครงสร้าง (Structured Programming)

คำสั่งเงื่อนไขที่ใช้ในภาษา C ได้แก่ คำสั่ง if และ switch โดยในการเขียน Expression สำหรับแสดงเงื่อนไข จะใช้ตัวดำเนินการความสัมพันธ์และตัวดำเนินการทางตรรกศาสตร์ ตามที่กล่าวไว้ในบทที่แล้ว ในตอนนี้จะกล่าวถึงคำสั่ง if

if Statement

คำสั่ง if ใช้ในการกำหนดเงื่อนไขในการทำงานส่วนหนึ่งของโปรแกรม เช่น ถ้า i มากกว่า 10 แล้วให้หยุดทำงาน เป็นต้น โดยรูปแบบทั่วไปของ if จะเป็นดังนี้

```
if (condition_expression)
      action_statement ;
```

โดย action_statement เป็นคำสั่งที่จะทำในกรณีที่เงื่อนไข condition_expression เป็นจริง (มีค่าเป็น 1) ดังแสดงในรูปที่ 3.1 ถ้าเงื่อนไขที่กำหนดไม่เป็นจริง จะข้ามไปทำคำสั่งต่อไป ยกตัวอย่าง เช่น

```
if (value < 0)
      printf("Less than zero.\n") ;
```

จะเป็นคำสั่งให้แสดงข้อความว่า Less than zero. ในกรณีที่ค่าของตัวแปร value น้อยกว่า 0

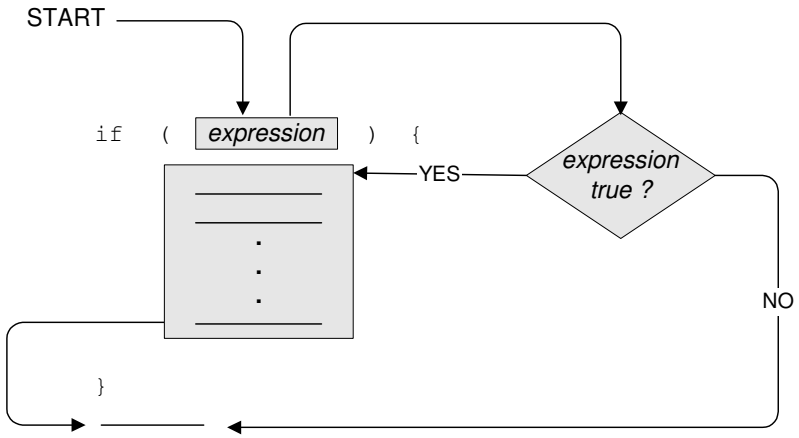
ในกรณีที่ต้องการให้โปรแกรมทำงานหลายอย่าง เมื่อเงื่อนไขที่กำหนดเป็นจริง จะต้องเขียนคำสั่งที่จะทำในวงเล็บปีกกา ดังนี้

```
if (condition_expression) {
      action_statements ;
      ...
}
```

เพื่อทำให้เป็นชุดคำสั่งที่จะทำก็ต่อเมื่อเงื่อนไขที่กำหนดเป็นจริง ยกตัวอย่าง เช่น

```
if (code == "le208") {
      printf("Introduction to Computer and Programming\n") ;
      printf("Credits : 3\n") ;
}
```

สิ่งที่พึงระวังเกี่ยวกับการใช้ if คือ โอเปอเรเตอร์ที่ใช้ในการกำหนดเงื่อนไขเท่ากับจะเป็น == ไม่ใช่ =



<u>รูปที่ 3.1</u> คำสั่ง if

if-else Statement

คำสั่ง if-else ใช้ในกรณีที่ต้องการกำหนดสิ่งที่จะทำ ในกรณีที่เงื่อนไขที่กำหนดขึ้นไม่เป็นจริงด้วย เช่น ถ้า x มากกว่าหรือเท่ากับ 0 ให้ค่าของตัวแปร abs_x ซึ่งใช้แทนค่า | x | (Absolute x) เท่ากับ x แต่ถ้า x น้อยกว่า 0 ให้ค่า abs_x เท่ากับ -x เป็นต้น ในกรณีนี้ เราจะใช้คำสั่ง *if-else* ในรูปแบบต่อไปนี้

```
if (condition_expression)
      true_statement ;
else
      false_statement ;
```

จะหมายถึง ถ้า condition_expression เป็นจริง ให้ทำคำสั่ง true_statement แต่ถ้าไม่เป็นจริง ให้ทำคำสั่ง false_statement ดังแสดงในรูปที่ 3.2 ยกตัวอย่าง เช่น

```
if (x >= 0)
      abs_x = x ;
else
      abs_x = -x ;
```

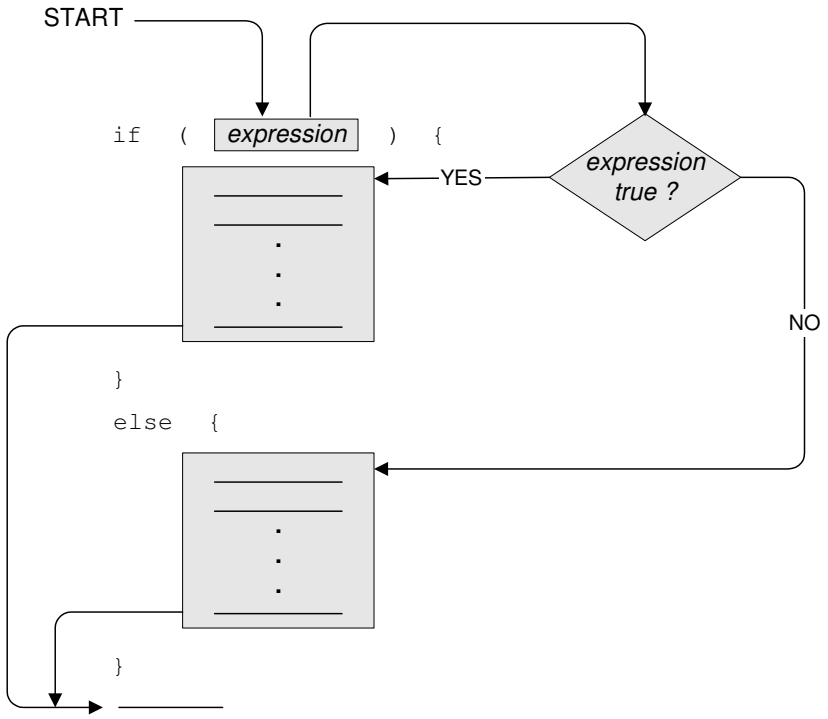
และเช่นเดียวกับในการใช้ประโยค *if* กรณีที่ต้องการให้ทำหลายคำสั่ง ภายใต้เงื่อนไขที่กำหนด จะต้องใช้เขียนคำสั่งหลาย ๆ คำสั่งนั้น ในวงเล็บปีกกา { } เพื่อทำให้เป็นชุดคำสั่ง

<u>ตัวอย่างที่ 3.1</u> ตัวอย่างการใช้ if-else Statement

```
#include <stdio.h>
int main()
{
      int   x ;
      printf("Enter an integer \n") ;
      scanf("%d", &x) ;
      if ((x % 2) == 0)
            printf(" %d is even \n", x) ;
      else
            printf(" %d is odd \n", x) ;
      return 0;
}
```

<u>ผลการรันโปรแกรม</u>

```
Enter an integer
5
 5 is odd
```



<u>รูปที่ 3.2</u> if-else Statement

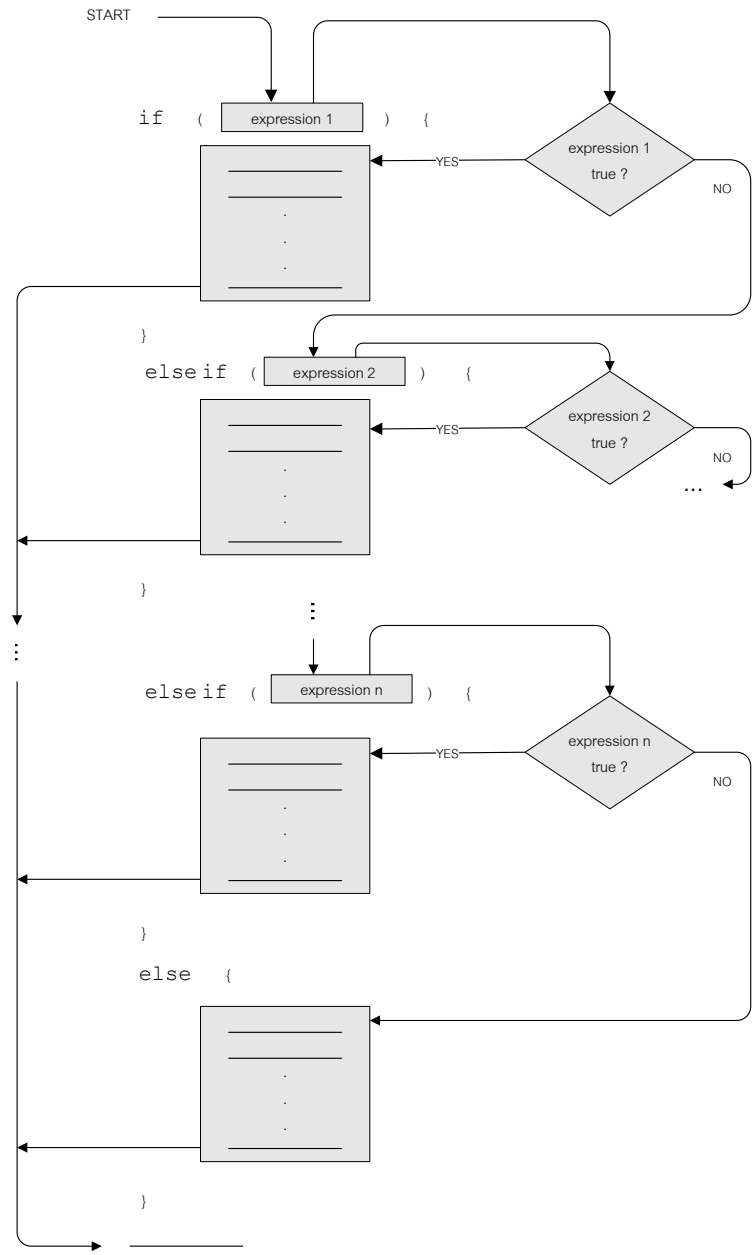
Nested* if-else *Statement

เราสามารถกำหนดเงื่อนไขซับซ้อนมากขึ้น โดยใช้คำสั่ง *if-else* หลาย ๆอันวางซ้อนกันในลักษณะลูกโซ่ ดังนี้

```
if (expression1)
{
      if (expression2)
            statement1 ;
      else
      {
            if (expression3)
                  statement2 ;
            else
                  statement3 ;
      }
}
else
      statement4 ;
```

ซึ่งมีความหมายดังนี้ คำสั่ง statement1 จะถูกทำก็ต่อเมื่อเงื่อนไข expression1 กับเงื่อนไข expression2 เป็นจริง คำสั่ง statement2 จะถูกทำเมื่อเงื่อนไข expression1 กับเงื่อนไข expression3 เป็นจริง และเงื่อนไข expression2 ไม่เป็นจริง ถ้าเงื่อนไข expression1 เป็นจริงและเงื่อนไข expression2 กับเงื่อนไข expression3 ไม่เป็นจริง จะทำคำสั่ง statement3 และท้ายที่สุด คำสั่ง statement4 จะถูกทำก็เมื่อเงื่อนไขทั้งหมดไม่เป็นจริง รูปที่ 3.3 แสดงลักษณะการทำงานของ if-else หลาย ๆ อันที่ต่อเนื่องกัน

สังเกตว่า คอมไพเลอร์จะตีความหมายคำสั่ง if ตามด้วยคำสั่ง 1 คำสั่ง เหมือนกับว่า เป็นคำสั่งประโยคเดียว ดังนั้น กรณีนี้ ไม่จำเป็นต้องมี { } แต่ทั้งนี้ เพื่อให้ง่ายต่อการอ่านโปรแกรม และเป็นการป้องกันข้อผิดพลาดที่อาจเกิดขึ้นได้ ควรจะใช้ { } แสดงชุดคำสั่งที่ขึ้นกับเงื่อนไขเสมอ



<u>รูปที่ 3.3</u> ลักษณะการทำงานของ if-else หลาย ๆ อันที่ต่อเนื่องกัน

<u>ตัวอย่างที่ 3.2</u> ตัวอย่างการใช้คำสั่ง if-else หลายๆอันที่ต่อเนื่องกัน

```
#include <stdio.h>
int main()
{
       int   code ;
       printf("Enter a faculty code \n") ;
       scanf("%d", &code) ;
       if (code == 1)
              printf("Laws\n") ;
       else if (code == 2)
              printf("Commerce and Accounting\n") ;
       else if (code == 10) {
              printf("Engineering ") ;
              printf("Consists of 5 departments: ") ;
              printf("ECE, IE, CE, ME & ChE\n") ;
       }
       else
              printf("No information\n") ;
```

```
        return 0;
}
```

ผลการรันโปรแกรม

ถ้าใส่ข้อมูล code เป็น 2 จะได้ผลดังนี้

```
Enter a faculty code
2
Commerce and Accounting
```

ถ้าใส่ข้อมูลเป็น 10 จะได้ผลดังนี้

```
Enter a faculty code
10
Engineering Consists of 5 departments: EE, IE, CE, ME & ChE
```

ถ้าใส่ข้อมูลเป็น 5 จะได้เป็น

```
Enter a faculty code
5
No information
```

3.2 switch Statement

คำสั่ง **switch** จัดเป็นหนึ่งในคำสั่งเงื่อนไข โดยมีลักษณะคล้ายกับการนำคำสั่ง **if/else** มาเชื่อมต่อกันเป็นลูกโซ่ โดยสามารถกำหนดเงื่อนไขตามค่าของตัวแปรหรือค่าของนิพจน์ เช่น ในกรณีที่มีให้เลือกได้หลายคำสั่ง เราก็สามารถใช้คำสั่ง switch ตามเลขที่ของคำสั่งนั้นได้ เป็นต้น

รูปแบบโดยทั่วไปของการใช้คำสั่ง switch จะเป็นดังนี้

```
switch (integer-expression) {
        case constant 1 :
                statement 1
                ......
                break ;
        case constant 2 :
                statement 2
                ......
                break ;

        ............
        case constant n :
                statement n
                ......
                break ;
        default :
                default statements ;
        }
```

หมายถึง ถ้าเงื่อนไขเท่ากับ constant 1 จะทำคำสั่งตาม statement 1 ถ้าเท่ากับ constant 2 ก็จะทำตาม statement 2 เช่นนี้ไป ถ้าไม่เท่ากับกรณีใดเลย ก็จะทำตามคำสั่ง default ลักษณะการทำงานของคำสั่ง Switch สามารถแสดงได้ตามรูปที่ 3.4

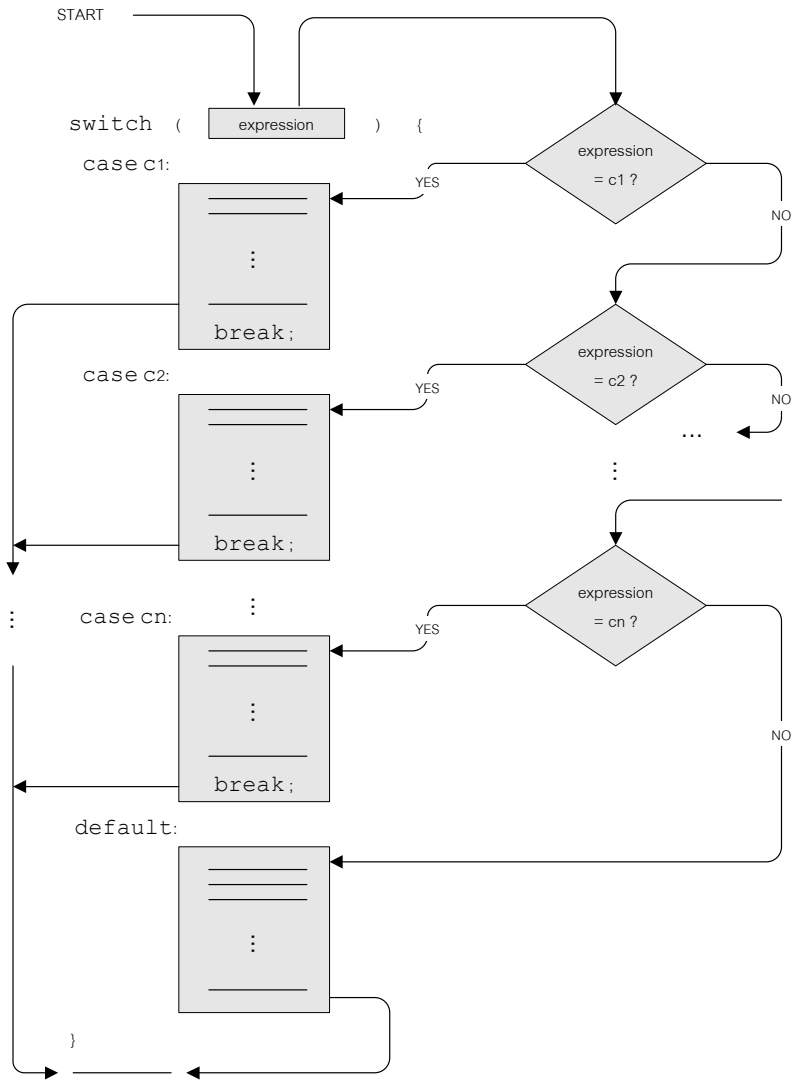
การกำหนดเงื่อนไขของ switch ทำโดยใช้ integer-expression ซึ่งอาจจะเป็นตัวแปร หรือเป็นนิพจน์ก็ได้ (แต่โดยมาก มักจะเป็นตัวแปร) ทั้งนี้ จะต้องเป็นค่าจำนวนเต็ม หลังคำว่า case จะต้องเป็นค่าคงที่ที่เป็นจำนวนเต็ม (constant 1...n) ที่สอดคล้องกับค่าของ integer-expression แล้วตามด้วยเครื่องหมาย : (Colon) ค่าคงที่ดังกล่าวนี้

สามารถกำหนดเป็นตัวอักษรได้ (เนื่องจากในหน่วยความจำ ตัวอักษรเก็บอยู่ในรูปแบบเดียวกับจำนวนเต็ม) แต่ไม่สามารถใช้ตัวแปรได้ ถ้าหากว่า integer-expression มีค่าเท่ากับ constant 1 โปรแกรมจะทำคำสั่งตาม statement 1 ถ้าเท่ากับ constant 2 จะทำ statement 2 กล่าวคือ ถ้า integer-expression เท่ากับ constant i จะทำตาม statement i และท้ายที่สุด ถ้าหาก integer-expression ไม่เท่ากับค่า constant ค่าใดค่าหนึ่ง โปรแกรมจะทำตาม default statement ซึ่งระบุโดยใช้คำว่า default แล้วตามด้วย :

รูปแบบทั่วไปของคำสั่ง switch ข้างต้น สามารถเขียนโดยใช้ประโยค if-else ได้ดังต่อไปนี้

```
if (integer-expression == constant 1)
       statement 1 ;
else if (integer-expression == constant 2)
       statement 2 ;
.....
else if (integer-expression == constant n)
       statement n ;
else
       default statement ;
```

ซึ่งเราจะสังเกตได้ว่า ถ้าหากเขียนโดยใช้ประโยค if-else จะทำให้ยากต่อการอ่านมาก และทำให้เกิดข้อผิดพลาดในการเขียนได้ง่าย ประโยค switch มีจุดเด่นตรงที่ทำให้เราสามารถเขียนโปรแกรมได้ซับซ้อนน้อยลงและมีความชัดเจนยิ่งขึ้น แต่ทั้งนี้ จะต้องอยู่ในรูปแบบของ Expression ที่มีค่าเป็นจำนวนเต็มหรือตัวอักษรเท่านั้น



<u>รูปที่ 3.4</u> คำสั่ง Switch

***ข้อควรระวังในการใช้* switch**

- ในแต่ละ case จะต้องมี break เป็นคำสั่งให้ออกจาก switch มิฉะนั้น โปรแกรมจะทำการตรวจสอบเงื่อนไขของ case อื่นด้วย ซึ่งอาจทำให้ผลการรันโปรแกรมผิดพลาดได้ ยกเว้นแต่กรณีที่ใช้ในฟังก์ชัน เมื่อมีคำสั่ง *return* แล้วก็จะถือว่าสิ้นสุดฟังก์ชัน ซึ่งก็เป็นการสิ้นสุดของ switch เช่นกัน ดังนั้น ไม่จำเป็นจะต้องมีคำสั่ง break ก็ได้ (ดูรายละเอียดในบทที่ 6)
- ควรจะมี default เป็นตัวกำหนดให้ทำอย่างไร ในกรณีที่ค่าของตัวแปรที่ใช้ควบคุม switch ไม่เท่ากับค่าของ case ที่กำหนดไว้เลย ทั้งนี้ เพื่อป้องกันข้อผิดพลาดที่อาจจะเกิดขึ้นได้
- ในส่วนของ default case จะมีคำสั่ง break หรือไม่มี ก็มีค่าเท่ากัน เนื่องจากเมื่อโปรแกรมจบจากการทำงานตามคำสั่ง default statement ก็จะออกจากคำสั่ง switch โดยอัตโนมัติ จึงไม่จำเป็นต้องมีคำสั่ง break (ในกรณีที่มีคำสั่ง break ใน default case โปรแกรม คอมไพเลอร์ บางโปรแกรม อาจจะแสดงข้อความเตือนว่า unreachable code ซึ่งหมายถึงว่า เป็นรหัสคำสั่งส่วนที่โปรแกรมไม่มีทางจะไปถึงได้ กล่าวคือ เป็นคำสั่งที่ไม่มีการทำ)

ตัวอย่างที่ 3.3 ตัวอย่างการใช้คำสั่ง switch

```
#include <stdio.h>
int main()
{
      int   input_number ;
      printf("Enter number :") ;
      scanf("%d", &input_number) ;
      switch(input_number) {
            case 0 :
                printf("zero\n") ; break ;
            case 1 :
                printf("one\n") ; break ;
            case 2 :
                printf("two\n") ; break ;
            default :
                printf("Otherwise\n") ;
            }
      return 0;
}
```

ผลการรันโปรแกรม

ถ้าใส่ข้อมูลเป็น 1 จะได้ผลดังนี้

```
Enter number :1
one
```

ถ้าใส่ข้อมูลเป็น 5 จะได้เป็น

```
Enter number :5
Otherwise
```

คำถาม หากไม่มีคำสั่ง break ในโปรแกรมตัวอย่างข้างต้น ผลการรันโปรแกรมจะเปลี่ยนเป็นอย่างไร

3.3 ทำไมถึงต้องมีคำสั่งการทำซ้ำ

โครงสร้างการทำซ้ำ (Repetition, Iteration) เป็น 1 ใน 3 โครงสร้างพื้นฐานตามหลักการเขียนโปรแกรมในลักษณะโครงสร้าง ซึ่งเป็นวิธีการพื้นฐานในการเขียนโปรแกรม และเป็นสิ่งที่คอมพิวเตอร์มีความสามารถสูงกว่ามนุษย์ กล่าวคือ ถ้าหากให้มนุษย์ทำเรื่องเดียวกันติดต่อกันเป็นพันหรือหมื่นครั้ง อาจจะทำผิดพลาดได้ และประสิทธิภาพจะลดลง แต่คอมพิวเตอร์จะไม่มีปัญหาเช่นนี้

ในภาษาการเขียนโปรแกรม เราเรียก *ชุดคำสั่งที่ทำต่อเนื่องหลาย ๆ ครั้ง* ว่า ลูป (*loop*) ซึ่งลูปเป็นพื้นฐานและเป็นสิ่งสำคัญในการออกแบบอัลกอริธึม สาเหตุหลัก 2 ประการที่ต้องมีลูป ได้แก่

1. ทำให้ประหยัดคำสั่ง (Economy of Expression) กล่าวคือ ทำให้ไม่ต้องเขียนคำสั่งเดียวกันหลายๆ ครั้ง
2. สามารถใช้กับจำนวนข้อมูลที่ไม่รู้ล่วงหน้า หรือสามารถเปลี่ยนแปลงขนาดของข้อมูลได้ เช่น ในการหาค่าสูงสุดของจำนวนข้อมูลที่ไม่แน่นอน ที่อาจจะเป็น 10, 100 หรือ 1000 จำนวน เป็นต้น

ยกตัวอย่าง เช่น ในการคำนวณหาผลรวมของเลขตั้งแต่ 1 - 5 ในขั้นแรก เราอาจจะเขียนในลักษณะ

```
int    sum ;
...
sum = 0 ;
sum += 1 ;
sum += 2 ;
sum += 3 ;
sum += 4 ;
sum += 5 ;
```

ซึ่งนอกจากจะดูยืดยาวแล้ว ยังขาดความยืดหยุ่นด้วย กล่าวคือ ถ้าหากเราต้องการหาผลรวมของตัวเลขที่มากขึ้น เราก็จำเป็นที่จะต้องทำการแก้ไขโปรแกรมใหม่ เราอาจเขียนใหม่โดยใช้ตัวแปรในลักษณะดังต่อไปนี้

```
int    sum, i ;
sum = 0 ;
i = 1 ;
sum += i ;
i++ ;
sum += i ;
i++ ;
sum += i ;
i++ ;
sum += i ;
i++ ;
sum += i ;
```

ในกรณีที่เราใช้ตัวแปร เราสามารถเปลี่ยนจากหาผลรวมของเลข 1-5 เป็น 11-15 ได้ โดยทำการแก้ไขเพียงบรรทัดเดียว (บรรทัดที่ 3) จึงอาจกล่าวได้ว่า Code ข้างต้นมีความยืดหยุ่นมากกว่า Code อันแรก แต่เราจะสังเกตได้ว่า มีการเขียน sum += i กับ i++ จำนวนหลายครั้งมาก ซึ่งแสดงให้เห็นถึงลูป หรือ คำสั่งที่ทำหลาย ๆ ครั้ง ดังนั้น เราจึงควรที่จะเขียนเป็นลูปในลักษณะดังต่อไปนี้

```
int    sum, i ;
sum = 0 ;
i = 1 ;
while (i < 6) {
       sum += i ;
       i++ ;
}
```

ในตัวอย่างข้างต้น จะใช้คำสั่ง while ซึ่งทำให้เราไม่ต้องเขียนคำสั่งซ้ำซ้อนกันหลายครั้ง ทำให้ความยาวของ Code สั้นลง นอกจากนั้น ยังเกิดความยืดหยุ่นยิ่งขึ้น ยกตัวอย่าง เช่น เราอาจจะแก้ไขบรรทัดที่ 4 ให้กำหนดเงื่อนไข เป็น i < n เพื่อทำให้เราสามารถเปลี่ยนจำนวนตัวเลขที่ต้องการหาผลรวมได้ หรือเราอาจทำการแก้ไขบรรทัดที่ 3 เพื่อทำ

ให้เราสามารถหาผลรวมของเลขจำนวนต่าง ๆ กันได้ อันจะส่งผลให้โปรแกรมที่เขียนขึ้นมี Generality ที่ดีขึ้น กล่าวคือ สามารถใช้งานได้หลากหลายขึ้น

ในการออกแบบการทำซ้ำ ส่วนประกอบสำคัญที่เราจะต้องคำนึงถึง ได้แก่

- **คำสั่งเริ่มต้น (initial-statement)** เพื่อกำหนดค่าของตัวแปรที่จะใช้ในลูป โดยทั่วไป จะเป็นการกำหนดค่าเริ่มต้นของตัวแปรที่ใช้ควบคุมการทำซ้ำ และค่าเริ่มต้นของตัวแปรที่จะเปลี่ยนไปในแต่ละครั้งของการวนลูป เช่น ค่าของผลรวม หรือ ผลคูณ เป็นต้น
- **เงื่อนไขการทำซ้ำ (Condition)** เพื่อใช้ในการพิจารณาว่า จะทำต่อหรือจะหยุดทำซ้ำ
- **คำสั่งในลูป (iteration-statement)** เป็นชุดคำสั่งที่ต้องการให้โปรแกรมทำซ้ำ โดยในกรณีที่ใช้ค่าของตัวแปรในการกำหนดเงื่อนไขการทำซ้ำ จะต้องมีการทำการเปลี่ยนค่าของตัวแปรตัวนั้น หรือถ้าไม่ใช้ค่าของตัวแปรในการกำหนดเงื่อนไข ก็จะต้องมีคำสั่งสำหรับกำหนดเงื่อนไขสิ้นสุดในลูป เพราะมิฉะนั้น โปรแกรมอาจจะทำซ้ำโดยไม่มีจุดสิ้นสุดได้ ซึ่งเรียกกันว่า *Endless Loop* ซึ่งเป็นสิ่งที่ต้องระวังในการใช้ลูป

กล่าวโดยสรุปแล้วก็คือ เราจำเป็นต้องมีกลไกสำหรับควบคุมการทำซ้ำ ซึ่งโดยทั่วไป ก็จะใช้ค่าของตัวแปรหรือฟังก์ชันเป็นตัวควบคุม ดังตัวอย่างข้างต้น ก็จะใช้ตัวแปร i เป็นตัวควบคุม ซึ่งถ้าเกิด i มากกว่าหรือเท่ากับ 6 ก็จะหยุดทำซ้ำ แต่ทั้งนี้ จะต้องทำการกำหนดค่าเริ่มต้นให้ถูกต้องด้วย มิฉะนั้น ก็อาจเกิดข้อผิดพลาดได้ เช่น ในตอนแรก ค่า i อาจไม่เท่ากับ 1 หรือค่า sum อาจไม่เท่ากับ 0 ทำให้ได้ผลไม่ถูกต้อง เป็นต้น

3.4 คำสั่งการทำซ้ำ

คำสั่งการทำซ้ำ (Looping Statement หรือ Repetition Statement) เป็นคำสั่งสำหรับสั่งให้โปรแกรมทำซ้ำคำสั่งที่เราต้องการ หรือโครงสร้าง Repetition ในการเขียนโปรแกรมในลักษณะโครงสร้าง ตัวอย่าง เช่น การทำซ้ำเพื่อคำนวณหาผลรวมของเลข 100 จำนวน เป็นต้น

ในภาษา C คำสั่งที่ใช้ในการกำหนดการทำซ้ำ ได้แก่ คำสั่ง while, do-while และ for ในตอนนี้ จะกล่าวถึงวิธีการใช้คำสั่งเหล่านี้

while Statement

คำสั่ง while เป็นการกำหนดให้ทำซ้ำงานอย่างใดอย่างหนึ่ง ระหว่างที่เงื่อนไขที่กำหนดขึ้นยังเป็นจริงอยู่ โดยมีรูปแบบทั่วไป ดังนี้

```
while (expression)
      action ;
```

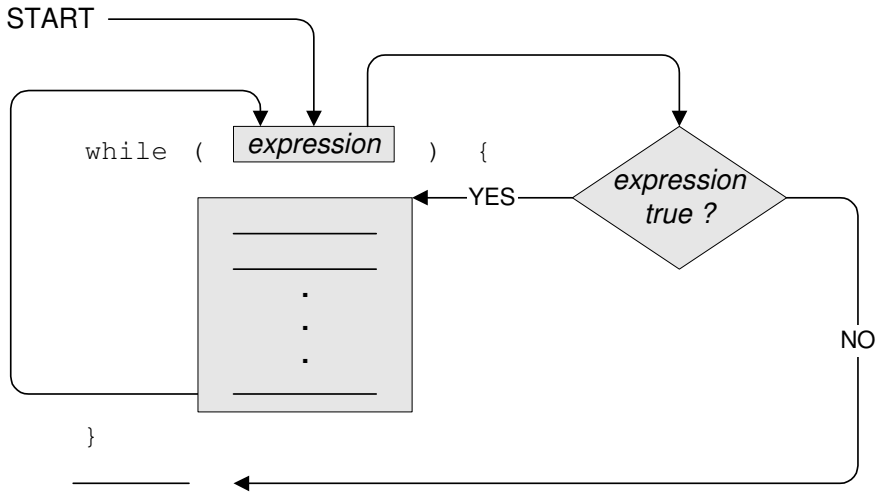
หมายถึง ให้โปรแกรมทำซ้ำ action ระหว่างที่เงื่อนไข expression เป็นจริง ดังแสดงในรูปที่ 3.5 ในกรณีที่ action มีหลายคำสั่ง จะต้องใช้วงเล็บปีกกา ดังนี้

```
while (expression)
{
      actions ;
      ....
}
```

ซึ่งการทำซ้ำด้วยคำสั่ง while นี้ มีชื่อเรียกว่า ลูป while (while loop) ตัวอย่าง เช่น

```
i = 1 ;
sum = 0 ;
while (i < 11)
{
      sum += i ;
      i++ ;
}
```

เป็นการสั่งให้โปรแกรมทำซ้ำการกำหนดค่าของตัวแปร sum และเพิ่มค่าของตัวแปร i ทีละ 1 จนกว่าค่า i จะเท่ากับ 10 (ระหว่างที่ i น้อยกว่า 11) พูดง่าย ๆ คือ เป็นการหาค่าผลรวมตั้งแต่ 1, 2 จนถึง 10



<u>รูปที่ 3.5</u> คำสั่ง while

ข้อควรระวังเกี่ยวกับการทำซ้ำ คือ จะต้องกำหนดเงื่อนไขที่มีจุดสิ้นสุด กล่าวคือ จะต้องมีตัวแปรสำหรับการทำซ้ำ หรือ จะต้องมีเงื่อนไขกำหนดจุดสิ้นสุดไว้ภายในลูปการทำซ้ำ โดยในตัวอย่างข้างต้น จะมีตัวแปร i เป็นตัวแปรสำหรับควบคุมการทำซ้ำ เริ่มต้นด้วย

```
i = 1 ;
```

เป็นการกำหนดค่าเริ่มต้นให้กับ i ในส่วนของคำสั่งในลูป while จะมีคำสั่ง

```
i++ ;
```

กล่าวคือ ทำการเพิ่มค่า i ทีละ 1 และเงื่อนไขในการทำซ้ำคือ

```
i < 11
```

หมายถึง ระหว่างที่ i น้อยกว่า 11 จะเป็นผลให้โปรแกรมข้างต้นทำซ้ำจำนวน 10 รอบ

นอกจากนี้ ในการทำซ้ำที่มีการเปลี่ยนค่าของตัวแปร ดังตัวอย่างข้างต้น จะต้องมีการกำหนดค่าเริ่มต้นของตัวแปรก่อน มิฉะนั้น อาจจะได้ผลลัพธ์ไม่ถูกต้อง

<u>ตัวอย่างที่ 3.4</u> Fibonacci number

Fibonacci number สามารถแสดงได้ตามสมการต่อไปนี้

$$f_n = f_{n-1} + f_{n-2} \tag{3.1}$$

โดยให้ $f_0 = f_1 = 1$ ยกตัวอย่าง เช่น $f_3 = f_2 + f_1 = f_1 + f_0 + f_1 = 3$ เป็นต้น

```
/* Program for calculating fibonacci number */
#include <stdio.h>
int main()
{
      int   old_number ;            /* previous Fibonacci number */
```

```
        int   current_number ;         /* current Fibonacci number */
        int   next_number ;            /* next number in the series */
        int   i = 1 ;                  /* index of f */
        old_number = 1 ;         /* initialize old_number = 1 (f0) */
        current_number = 1 ;   /* initialize current_number = 1 (f1) */
        while (current_number < 100) {
              /* Print current fibonacci number */
              printf("%d %d\n", i,current_number) ;
              /* Calculate next fibonacci number */
              next_number = current_number + old_number ;
              /* Change previous fibonacci number */
              old_number = current_number ;
              /* Change current fibonacci number */
              current_number = next_number ;
              i++ ;                    /* Increment the index */
        }
        return 0;
}
```

ผลการรันโปรแกรม

```
1 1
2 2
3 3
4 5
5 8
6 13
7 21
8 34
9 55
10 89
```

ตัวอย่างที่ 3.5 Endless Loop

```
#include <stdio.h>
int main()
{
        while(1)                  /* while (1) means ALWAYS TRUE */
              printf("ENDLESS LOOP  ") ;
        return 0;
}
```

ผลการรันโปรแกรม

โปรแกรมนี้ เป็นโปรแกรมที่ทำการแสดงข้อความว่า ENDLESS LOOP อย่างไม่มีที่สิ้นสุด เนื่องจากเงื่อนไขในการทำซ้ำเป็นจริงเสมอ ดังนั้น ถ้าต้องการหยุดการทำงานของโปรแกรมนี้ จะต้องทำการหยุดจากภายนอกโปรแกรม เช่น ถ้าในระบบ UNIX สามารถใช้คำสั่ง kill ในการหยุดการทำงานของโปรแกรมนี้ได้ เป็นต้น

do-while Statement

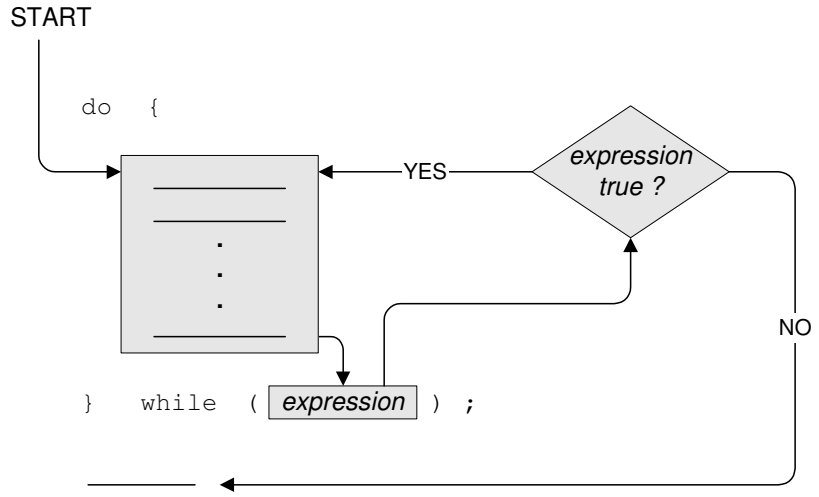
คำสั่ง do-while จะคล้ายกับคำสั่ง while ต่างกันก็แต่เฉพาะลำดับของการตรวจสอบเงื่อนไข โดยคำสั่ง while จะตรวจสอบเงื่อนไขก่อนที่จะเริ่มต้นทำซ้ำ (ซึ่งเรียกว่า *Precondition*) ในขณะที่ do-while จะทำรอบแรกก่อนแล้วจึงตรวจสอบเงื่อนไข (เรียกว่า *Postcondition*) รูปแบบทั่วไปของ do-while จะเป็นดังนี้

```
        do
              action
      while (condition) ;
```

หมายถึง จะทำ action ก่อน แล้วทำการตรวจสอบเงื่อนไข condition ถ้าเงื่อนไขเป็นจริง จะทำซ้ำไปเรื่อย ๆ จนกว่าเงื่อนไข condition จะไม่เป็นจริง ดังแสดงในรูปที่ 3.6 ยกตัวอย่าง เช่น

```
do {
      printf("Enter a positive integer: ") ;
      scanf("%d", &response) ;
} while (response <= 0) ;
```

โปรแกรมจะทำการพิมพ์ข้อความว่า Enter a positive integer แล้วทำการรับค่าจำนวนเต็ม 1 ค่า ถ้าค่าที่รับเข้ามาน้อยกว่าหรือเท่ากับ 0 ให้ทำซ้ำไปเรื่อย ๆ จนกว่าค่าที่รับเข้ามาจะเป็นจำนวนเต็มบวก กล่าวคือ เป็นการสั่งให้โปรแกรมทำการรับค่าจำนวนเต็มบวกเข้ามาจากอินพุตของผู้ใช้



<u>รูปที่ 3.6</u> คำสั่ง do-while

for Statement

คำสั่ง **for** ใช้กับการทำซ้ำที่รู้จำนวนรอบของการทำซ้ำที่แน่นอน ยกตัวอย่าง เช่น การหาผลรวมของเลข 10 จำนวน การแสดงตัวเลขตั้งแต่ 1 ถึง 100 เป็นต้น รูปแบบโดยทั่วไปของคำสั่ง for จะเป็นดังนี้

```
for (initial-statement ; condition ; iteration-statement)
      action-statement ;
```

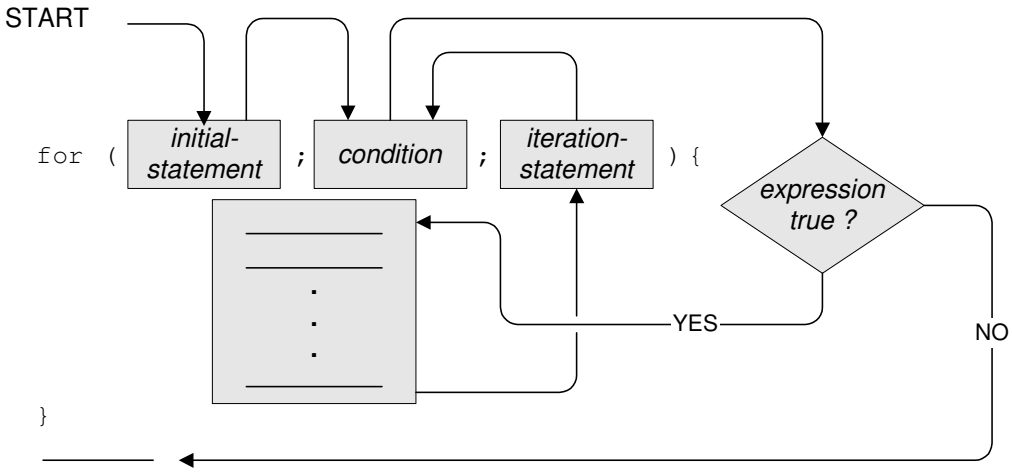
จะมีค่าเท่ากับลูป while ต่อไปนี้

```
initial-statement ;
while (condition) {
      action-statement ;
      iteration-statement ;
}
```

กล่าวคือ ในลูป for จะเริ่มต้นด้วยการกำหนดค่าเริ่มต้น แล้วทำตาม action-statement และ iteration-statement จากนั้น จึงค่อยตรวจสอบเงื่อนไขการทำซ้ำ ถ้าเป็นจริง ก็ทำซ้ำรอบต่อไป ถ้าไม่ ก็ยุติการทำซ้ำ ดังแสดงในรูปที่ 3.7 ยกตัวอย่าง เช่น

```
sum = 0 ;
for (i = 1 ; i < 11 ; i++)
      sum += i ;
```

จะมีความหมายอย่างเดียวกับตัวอย่างของลูป while



รูปที่ 3.7 คำสั่ง for

ตัวอย่างที่ 3.6 โปรแกรมสำหรับการแปลงองศาเซลเซียสให้เป็นองศาฟาเรนไฮต์โดยใช้คำสั่ง for

```
/* program for making a Celsius to Fahrenheit conversion chart */
#include <stdio.h>
int main()
{
      int   celsius ;
      for (celsius = 0 ; celsius <= 75 ; celsius = celsius+5)
            printf("%d Celsius = %d Fahrenheit \n",
                  celsius, (celsius * 9) / 5 + 32) ;
      return 0;
}
```

ผลการรันโปรแกรม

```
0 Celsius = 32 Fahrenheit
5 Celsius = 41 Fahrenheit
10 Celsius = 50 Fahrenheit
15 Celsius = 59 Fahrenheit
20 Celsius = 68 Fahrenheit
25 Celsius = 77 Fahrenheit
30 Celsius = 86 Fahrenheit
35 Celsius = 95 Fahrenheit
40 Celsius = 104 Fahrenheit
45 Celsius = 113 Fahrenheit
50 Celsius = 122 Fahrenheit
55 Celsius = 131 Fahrenheit
60 Celsius = 140 Fahrenheit
65 Celsius = 149 Fahrenheit
70 Celsius = 158 Fahrenheit
75 Celsius = 167 Fahrenheit
```

ตัวอย่างที่ 3.7 Endless loop (for version)

```
#include <stdio.h>
int main()
{
      for (; ;) printf("ENDLESS LOOP  ") ;
      return 0;
}
```

3.5 break and continue Statement

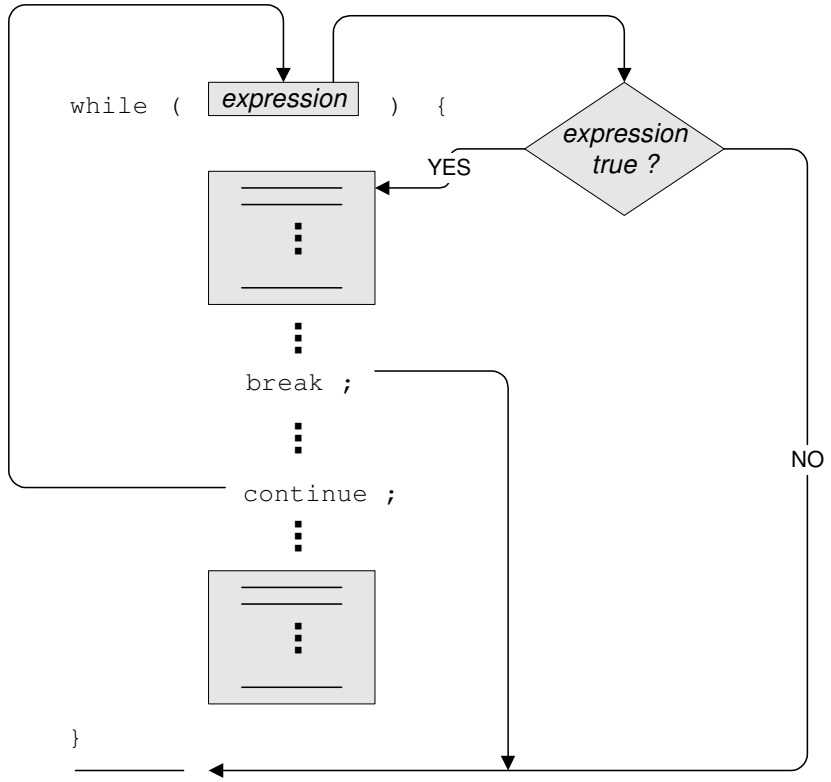
คำสั่ง **break** มีไว้สำหรับใช้ในกรณีที่เราต้องการให้โปรแกรมเลิกการทำซ้ำกลางครัน ในระหว่างการวนลูป ยกตัวอย่าง เช่น เราต้องการให้ยกเลิกการทำซ้ำ กรณีที่เกิดข้อผิดพลาดขึ้น เป็นต้น โดยคำสั่ง break จะมีผลเฉพาะลูปที่อยู่ในสุดของตำแหน่งที่คำสั่งนี้อยู่ ยกตัวอย่าง เช่น

```
while (expression1) {
      while (expression2) {
            …
            if (condition)
                  break ;
      }
      action
}
```

break จะมีผลยกเลิกการทำงานของลูปใน (ลูปที่มี expression2 เป็นเงื่อนไขการทำซ้ำ) เท่านั้น

ส่วนคำสั่ง **continue** จะคล้ายกับ **break** เพียงแต่ว่าแทนที่จะออกจากลูป จะกลับไปที่จุดเริ่มต้นของลูปแล้วทำซ้ำใหม่อีกครั้ง

การใช้คำสั่ง break และ continue สามารถแสดงได้ดังรูปที่ 3.8



<u>รูปที่ 3.8</u> การใช้คำสั่ง break และ continue

<u>ตัวอย่างที่ 3.8</u> ตัวอย่างการใช้คำสั่ง break และ continue

```
/* program for calculating the sum from 1 to the entered number */
#include <stdio.h>
int main()
{
      int   total=0,  number, i ;
      while (1) {       /* Iteration condition is always true. */
            printf("Enter number :") ;
            scanf("%d", &number) ;
            if (number == 0)   break ;
                        /* 0 input means to terminate program */
            if (number < 0)   continue ;
```

```
                        /* minus input means to re-input the value */
            i = 1;
            while (i <= number) {
                total += i ;
                i++ ;
            }
            printf("The sum from 1 to %d is %d.\n", number, total) ;
        }
        print("finished\n") ;
        return 0;
}
```

ผลการรันโปรแกรม

```
Enter number :10
The sum from 1 to 10 is 55.
Enter number :20
The sum from 1 to 20 is 23.
Enter number :-5
Enter number :0
finished
```

3.6 ข้อผิดพลาดในการเขียนโปรแกรม

1. ในการใช้คำสั่ง if ในภาษา C จะไม่มีการใช้คำว่า then เหมือนอย่างในภาษาอื่น เช่น FORTRAN BASIC เป็นต้น ดังนั้น จะเกิด Error ขึ้น ถ้าเขียนเป็น

```
if (condition) then     /* ------ ERROR ------ */
      action
```

2. เงื่อนไขในคำสั่ง if, while, do-while และ for จะต้องอยู่ในวงเล็บ ดังนั้น จะเกิด Error ขึ้น ถ้าเขียนเป็น

```
if condition            /* ------ ERROR ------ */
      action
```

หรือ

```
while condition
      action
```

3. หลังคำว่า else ในคำสั่ง if-else ไม่ต้องมี ; (semicolon) ดังนั้น จะเกิด Error ขึ้น ถ้าเขียนว่า

```
if (condition)
      action1
else ;
      action2
```

4. ระวังเรื่องการใช้ == กับ = เพราะเงื่อนไข 2 อันต่อไปนี้ จะต่างกันโดยสิ้นเชิง

```
if (x == 1) ...
if (x = 1) ...
```

โดยประโยคหลังจะเป็นจริงตลอดไป

5. ในกรณีที่ใช้ลูป while จะไม่มีการใช้คำสั่ง do ยกตัวอย่าง เช่น จะเกิด Error ในการคำนวณ ถ้าเขียนในลักษณะ

```
while (condition) do    /* ------ ERROR ------ */
```

```
        action
```

6. ในการใช้คำสั่ง do-while หลังคำสั่ง while จะต้องมีเครื่องหมาย ; เสมอ
7. คำสั่งที่กำหนดควบคุมการทำซ้ำในลูป for จะต้องถูกขั้นด้วย ; ไม่ใช่ ; ดังนั้น จะเกิด Error ขึ้น ถ้าเขียนเป็น

```
for  (initial-statement , condition , iteration-statement)
      action
```

8. หลังคำสั่ง for ไม่ต้องมีเครื่องหมาย ; ยกตัวอย่าง เช่น

```
for (i = 0 ; i < 10 ; i++) ;
      sum += i ;
```

เป็นข้อความที่ถูกต้องตามหลักไวยากรณ์ แต่ไม่ได้ความหมายตามที่ต้องการ เพราะจะทำ Null Statement จำนวน 10 ครั้ง และทำการกำหนดค่า sum เพียงครั้งเดียว

9. คำสั่ง break มีผลทำให้โปรแกรมออกจากการทำซ้ำของลูปในสุดที่มีคำสั่งนี้อยู่เท่านั้น
10. ในการใช้ประโยค switch integer expression ที่ตามหลัง จะต้องมีเครื่องหมาย () ดังนั้น จะเกิด Error ขึ้น ถ้าเขียนเป็น

```
switch i {  /* ------- ERROR ------ */
      …
}
```

11. ในการใช้ประโยค switch ค่าที่ตามหลังคำว่า case จะต้องเป็นค่าคงที่ที่เป็นจำนวนเต็ม จะเป็นตัวแปรไม่ได้ ยกตัวอย่าง เช่น

```
case i :          /* ------ ERROR ------ */
```

ไม่ถูกต้องตามหลักไวยากรณ์

12. โดยปกติ ทุก case ของประโยค switch จะต้องมีคำสั่ง break ไว้ท้ายสุด มิฉะนั้น โปรแกรมจะไปตรวจสอบ case อื่น และ default ต่อไป

3.7 แบบฝึกหัด

1. จงบอกผลที่ได้ของโปรแกรมต่อไปนี้

```
int x, y ;
x = 3 ;
y = 5 ;
if (x > 2)
      printf("%d\n", x) ;
else
      printf("%d\n", y) ;
```

2. จงบอกข้อผิดพลาดของส่วนของโปรแกรมต่อไปนี้

```
if (x >= 2) then
      printf("%d\n", x) ;
```

3. จงบอกข้อผิดพลาดของส่วนของโปรแกรมต่อไปนี้

```
if x >= 2
      printf("%d\n", x) ;
```

4. จงบอกที่ได้จากโปรแกรมต่อไปนี้

```
i = 1 ;
if (i == 0) {
      printf("zero\n") ;
      if (i == 1)
      printf("one\n") ;
      else
      printf("not one\n") ;
}
else
      printf("not zero\n") ;
```

5. จงบอกผลที่ได้จากโปรแกรมต่อไปนี้

```
int x = 5 ;
if (x < 2)
      printf("%d\n", x) ;
else if (x < 4)
      printf("%d\n", 2 * x) ;
else if (x < 6)
      printf("%d\n", 3 * x) ;
else
      printf("%d\n", 4 * x) ;
```

6. จงบอกผลที่ได้จากโปรแกรมต่อไปนี้

```
int x = 7 ;
while (x >= 0) {
      printf("%d\n", x) ;
      x -= 2 ;
}
```

7. จงบอกผลที่ได้จากโปรแกรมต่อไปนี้

```
int x = 7 ;
while (x >= 0) {
      x -= 2 ;
      printf("%d\n", x) ;
}
```

8. จงบอกผลที่ได้จากโปรแกรมต่อไปนี้

```
int x = 10 ;
do {
      printf("%d\n", x) ;
      x -= 3 ;
} while (x >= 2) ;
```

9. จงบอกผลที่ได้จากโปรแกรมต่อไปนี้

```
int x = 5 ;
while (x == 5) {
      x = x - 1 ;
      printf("%d\n", x) ;
}
```

10. จงบอกผลที่ได้จากโปรแกรมต่อไปนี้

```
for (i = 0 ; i <= 10 ; i ++) {
      printf("%d\n", i) ;
      i += 2 ;
}
```

11. จงบอกผลที่ได้จากโปรแกรมต่อไปนี้

```
int i, x = 10 ;
for (i = 0 ; i < x ; i++) {
      if (i % 2 == 0)
            printf("Even\n") ;
      else
            printf("Odd\n") ;
}
```

12. เมื่อรันโปรแกรมต่อไปนี้ในบางระบบ

```
#include <stdio.h>
void main()
{
      float x ;
      for (x = 0.0 ; x <= 1.0 ; x += 0.1)
            printf("%.1f ", x) ;
}
```

จะได้ผลลัพธ์เป็น

```
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
```

เพราะเหตุใด โปรแกรมจึงไม่พิมพ์ค่า 1.0 ออกมา

13. จงเปลี่ยนลูป while ในโจทย์ข้อ 6 และ 7 ให้เป็นลูป for

14. จงบอกผลที่ได้จากโปรแกรมต่อไปนี้

```
int i = 0 ;
while (i < 10) {
      if (i < 3) {
            i += 2 ;
            printf("%d\n", i) ;
            continue ;
      }
      else {
            printf("%d\n", ++i) ;
            break ;
      }
      printf("Bottom of Loop\n") ;
}
```

15. จงบอกผลที่ได้จากโปรแกรมต่อไปนี้

```
int i = 1 ;
switch (i) {
case 0 :
  printf("zero\n") ;
case 1 :
  printf("one\n") ;
case 2 :
  printf("two\n") ;
default :
  printf("otherwise\n") ;
}
```

16. จงเปลี่ยนคำสั่งเงื่อนไขต่อไปนี้ โดยใช้คำสั่ง switch

```
if (i == 0)
      printf("zero\n") ;
else if (i == 1)
      printf("one\n") ;
else if (i == 2)
      printf("two\n") ;
else if (i == 3)
      printf("three\n") ;
else
      printf("more than three\n") ;
```

17. จงเขียนโปรแกรมสำหรับคำนวณหาค่าเฉลี่ย ค่าสูงสุด ค่าต่ำสุด ของเลขต่อไปนี้

33 46 36 27 46 42 48 42 27 58 25 35 43 47 48 26 30 31 44 21 22 46 51

18. จงเขียนโปรแกรมสำหรับสร้างตารางเปรียบเทียบค่า sin(x) ที่ได้จาก function sin(x) กับค่าที่ได้จากการคำนวณจากสมการ

$$\sin(x) = \frac{1}{1!}x - \frac{1}{3!}x^3 + \cdots + \frac{(-1)^n}{(2n+1)!}x^{2n+1} + \cdots$$

19. จงเขียนโปรแกรมสำหรับหาจำนวนเฉพาะที่น้อยกว่า 1000

20. จงเขียนโปรแกรมสำหรับหา Combination ของเลขตั้งแต่ 1 ถึง n

ตัวอย่าง : {1,2,3} -> {1,2,3}, {1,3,2}, {2,1,3}, {2,3,1}, {3, 1,2}, {3,2,1}

21. จงเขียนโปรแกรมสำหรับคำนวณหาวันที่ของปี เช่น วันที่ 1 กุมภาพันธ์ เป็นวันที่ 32 ของปี เป็นต้น

22. จงเขียนโปรแกรมสำหรับแก้สมการ

$$ax^2 + bx + c = 0$$

โดยป้อนค่า *a, b, c* เข้าไป