Chapter 8 Function

- ในโปรแกรมทุกโปรแกรมจะต้องมีส่วนประกอบหนึ่งคือฟังก์ชัน
- จากโปรแกรมที่ผ่านมา printf() และ scanf() ก็เป็นตัวอย่างของฟังก์ชัน หรือแม้แต่ main() ก็เช่นกัน
- แต่ฟังก์ชัน main() เป็นฟังก์ชันพิเศษที่บอกจุดเริ่มต้นการทำงานของโปรแกรม ดังนั้นฟังก์ชันนี้จะต้องมีในทุกโปรแกรม
- ฟังก์ชันทำให้โปรแกรมง่ายต่อการเขียน อ่าน และแก้ไข นอกจากนี้ฟังก์ชันที่พัฒนาไว้ในโปรแกรมหนึ่ง ยังสามารถนำไปใช้กับโปรแกรมอื่นได้โดยง่าย

8.1 การประกาศฟังก์ชัน

- เราจะทำความเข้าใจความหมายฟังก์ชันก่อนที่จะนำมันไปใช้กับโปรแกรมของเราให้เกิดประสิทธิภาพสูงสุด
- ย้อนกลับไปที่โปรแกรมสำหรับแสดงข้อความออกทางหน้าจอ

```
#include <stdio.h>
int main (void)
{
    printf("Programming is fun.\n");
    return 0;
}
```

- แยกส่วนการแสดงผลออกทางหน้าจอมาใส่ในฟังก์ชันเพื่อทำหน้าที่อย่างเดียวกัน

```
#include <stdio.h>

void printMessage(void)
{
    printf("Programming is fun.\n");
}

int main(void)
{
    printMessage();
    return 0;
}
```

- โปรแกรมนี้ประกอบด้วยสองฟังก์ชันคือ printMessage() และ main() โดยการทำงานโปรแกรมจะเริ่มทำที่ main() และเรียกใช้ฟังก์ชัน printMessage()
- ฟังก์ชัน printMessage() ถูกเรียกใช้จากฟังก์ชั่น main() โดยไม่มีการส่งค่าหรือ argument ใดๆเข้าไปในฟังก์ชัน ซึ่งสอดคล้องกับการประกาศฟังก์ชันนั้น
- เมื่อ printMessage() ถูกเรียก การทำงานของโปรแกรมจะย้ายไปยังฟังก์ชันนั้น ซึ่งในฟังก์ชันจะมีการทำคำสั่ง printf() เพื่อแสดงคำว่า “Programming is fun” ออกทางหน้าจอ หลังจากนั้นการทำงานในฟังก์ชั่น printMessage() จะเสร็จสิ้นและโปรแกรมจะย้อนกลับมาทำงานต่อที่ฟังก์ชัน main() ณ ตำแหน่งที่กระโดดออกไป
- เราสามารถใส่ return; ท้ายฟังก์ชั่น printMessage() ได้โดยไม่ต้องใส่ค่าใดๆ
- จริงๆแล้วทั้ง printf() และ scanf() ต่างก็เป็นฟังก์ชั่น และการใช้งานคล้ายกับ printMessage() ที่เราได้สร้างขึ้น ต่างกันเพียงที่สองฟังก์ชั่นแรกอยู่ในไลบรารีมาตรฐานของภาษาซี และเมื่อเราสั่ง printf() การทำงานของโปรแกรมก็จะย้ายไปยัง printf() และกลับมาเมื่อเสร็จสิ้น

Program 8.2 Calling Functions

```
#include <stdio.h>

void printMessage (void)
{
    printf("Programming is fun.\n");
}

int main (void)
{
    printMessage();
    printMessage();
    return 0;
}
```

```
Programming is fun.
Programming is fun.
```

- การทำงานของโปรแกรม 8.2 นี้เริ่มต้นที่ main() เช่นกัน และมีการเรียกใช้งานฟังก์ชัน printMessage() สองครั้ง

Program 8.3 More on Calling Functions

```
#include <stdio.h>

void printMessage(void)
{
    printf("Programming is fun.\n");
}

int main(void)
{
    int i;
    for(i = 1; i <= 5; ++i )
        printMessage ();
    return 0;
}
```

8.2 การส่งค่าเข้าไปในฟังก์ชัน

- ในการเรียกใช้งานฟังก์ชั่น printf() จะต้องมีการผ่านค่าอย่างน้อยหนึ่งค่าเข้าไปในฟังก์ชันเสมอ โดยค่าแรกคือตัวอักษรที่มีการจัดรูปแบบและค่าที่เหลือเป็นผลลัพธ์จากโปรแกรมที่ต้องการแสดง โดยค่าเหล่านี้มีชื่อเรียกว่า argument
- เราสามารถสร้างฟังก์ชันที่รับ argument ได้เช่นเดียวกัน
- ในตัวอย่างเป็นฟังก์ชันที่ใช้คำนวณ triangular number โดยฟังก์ชันรับค่า triangular number ที่ต้องการคำนวณ

Program 8.4 Calculating the nth Triangular Number

```
// Function to calculate the nth triangular number
#include <stdio.h>

void calculateTriangularNumber(int n)
{
    int i, triangularNumber = 0;
    for (i = 1; i <= n; ++i )
        triangularNumber += i;
    printf("Triangular number %i is %i\n", n, triangularNumber);
}

int main(void)
{
    calculateTriangularNumber(10);
    calculateTriangularNumber(20);
    calculateTriangularNumber(50);
    return 0;
}
```

```
Triangular number 10 is 55
Triangular number 20 is 210
Triangular number 50 is 1275
```

- ในกรณนี้ void calculateTriangularNumber(int n) จะไม่ส่งค่าใดๆ ออกมาจากฟังก์ชัน และรับ argument หนึ่งตัวนั่นคือตัวแปร n ซึ่งเป็นชนิด integer
- โดยตัวแปรชื่อ n นี้จะถูกเรียกใช้งานได้ภายในฟังก์ชัน
- ในฟังก์ชัน calculateTriangularNumber() มีการประกาศตัวแปรขึ้นสองตัวเพื่อใช้งานภายในฟังก์ชัน คือ i และ triangularNumber
- ตัวแปรที่ประกาศขึ้นในฟังก์ชันนี้ถือเป็น local variable โดยจะถูกสร้างขึ้นทุกครั้งที่ฟังก์ชันถูกเรียกใช้งาน และ local variable จะถูกเรียกใช้งานได้จากภายในฟังก์ชันที่ประกาศมันเท่านั้น
- หลังจากประกาศ local variable ในฟังก์ชันแล้ว ฟังก์ชันทำการคำนวณ triangular number และแสดงผลออกทางหน้าจอ
- จากฟังก์ชัน main() มีการเรียกใช้งาน calculateTriangularNumber() สามครั้ง โดยในแต่ละครั้งมี argument แตกต่างกันไป ในการเรียกใช้ครั้งแรกส่งค่า 10 ซึ่งค่านี้จะถูกนำไปเก็บในตัวแปร n ภายในฟังก์ชันเพื่อคำนวณต่อไป
- เมื่อฟังก์ชันทำงานเสร็จสิ้นในแต่ละครั้ง local variable และค่าที่เก็บอยู่ในตัวมันจะถูกทำลาย เราสามารถเก็บค่าไว้ใน local variable สำหรับการเรียกใช้งานฟังก์ชันครั้งต่อไปได้โดยประกาศตัวแปรเป็น static local variable เช่น static int i;
- ตัวอย่างถัดไปเป็นการเขียนฟังก์ชันที่รับ argument มากกว่าหนึ่งตัวเพื่อใช้หาค่าหารร่วมมาก (gdc)

Program 8.5 Revising the Program to Find the Greatest Common Divisor

```
/* Function to find the greatest common divisor
of two nonnegative integer values */
#include <stdio.h>

void gcd(int u, int v)
{
    int temp;
    printf ("The gcd of %i and %i is ", u, v);
    while( v != 0 ) {
        temp = u % v;
        u = v;
        v = temp;
    }
    printf ("%i\n", u);
}

int main (void)
{
    gcd(150, 35);
    gcd(1026, 405);
    gcd(83, 240);
    return 0;
}
```

```
The gcd of 150 and 35 is 5
The gcd of 1026 and 405 is 27
The gcd of 83 and 240 is 1
```

- ฟังก์ชัน gcd() รับ argument เป็นเลขจำนวนเต็มสองค่า โดยใช้ตัวแปร u และ v เพื่อใช้คำนวณหาค่าหารร่วมมาก

8.3 การส่งค่าออกจากฟังก์ชัน

- ในโปรแกรมตัวอย่างที่ 8.4 และ 8.5 มีการแสดงผลลัพธ์จากการคำนวณออกทางหน้าจอโดยใช้คำสั่ง printf ในฟังก์ชันเลย อย่างไรก็ดีเราอาจจะไม่ต้องการแสดงผลลัพธ์นั้นทันที
- เราสามารถส่งค่าที่คำนวณได้กลับออกมาจากฟังก์ชัน โดยใช้คำสั่ง return ในรูปแบบ

  return expression;
- การใส่คำสั่งนี้ไว้ท้ายฟังก์ชันทำให้มีการคืนค่า expression กลับมาที่ตำแหน่งที่เรียกใช้งานฟังก์ชันนั้น
- นอกจากการใส่คำสั่ง return ไว้ท้ายฟังก์ชันแล้ว ยังจำเป็นจะต้องประกาศชนิดของข้อมูลที่ฟังก์ชันจะส่งกลับออกมา ซึ่งชนิดของข้อมูลนี้จะใส่ไว้ที่ตำแหน่งก่อนหน้าชื่อฟังก์ชัน ตัวอย่างเช่น

  float kmh_to_mph(float km_speed)
  int gcd(int u, int v)
- เราสามารถแก้โปรแกรม 8.5 เพื่อให้ฟังก์ชัน gcd คืนค่า หรม กลับออกมาที่ฟังก์ชัน main() ดังแสดง

Program 8.6 Finding the Greatest Common Divisor and Returning the Results

```
/* Function to find the greatest common divisor of two
nonnegative integer values and to return the result */
#include <stdio.h>

int gcd (int u, int v)
{
    int temp;
    while( v != 0 ){
        temp = u % v;
        u = v;
        v = temp;
    }
    return u;
}

int main (void)
{
    int result;

    result = gcd(150, 35);
    printf("The gcd of 150 and 35 is %i\n", result);

    result = gcd(1026, 405);
    printf("The gcd of 1026 and 405 is %i\n", result);

    printf("The gcd of 83 and 240 is %i\n", gcd(83, 240));

    return 0;
}
```

```
The gcd of 150 and 35 is 5
The gcd of 1026 and 405 is 27
The gcd of 83 and 240 is 1
```

- ค่าที่ส่งกลับออกมาจากฟังก์ชันสามารถเก็บลงในตัวแปร หรือนำไปใช้แสดงผลได้เช่นกัน

- หากฟังก์ชันไม่มีการส่งค่าออกมา (คือชนิดของข้อมูลที่นำหน้าชื่อฟังก์ชันเป็น void) แต่มีการเขียนโปรแกรมในลักษณะที่จะนำค่าที่ return จากฟังก์ชันไปใช้งาน จะเกิด error ขณะคอมไพล์
- ตัวอย่างต่อไปเป็นโปรแกรมที่มีฟังก์ชันสำหรับหาค่าสัมบูรณ์ของจำนวน

Program 8.7 Calculating the Absolute Value

```
// Function to calculate the absolute value

#include <stdio.h>

float absoluteValue (float x)
{
    if( x < 0 )
        x = -x;
    return x;
}

int main (void)
{
    float f1 = -15.5, f2 = 20.0, f3 = -5.0;
    int i1 = -716;
    float result;

    result = absoluteValue(f1);
    printf("result = %.2f\n", result);
    printf("f1 = %.2f\n", f1);

    result = absoluteValue(f2) + absoluteValue(f3);
    printf("result = %.2f\n", result);

    result = absoluteValue( (float) i1 );
    printf("result = %.2f\n", result);

    result = absoluteValue(i1);
    printf("result = %.2f\n", result);

    printf("%.2f\n", absoluteValue(-6.0) / 4 );

    return 0;
}
```

```
result = 15.50
f1 = -15.50
result = 25.00
result = 716.00
result = 716.00
1.50
```

- ฟังก์ชัน absoluteValue รับค่าเลขทศนิยมเข้าไปและทดสอบว่าน้อยกว่าศูนย์หรือไม่ ถ้าน้อยกว่าศูนย์ให้คูณด้วยลบหนึ่ง จากนั้นคืนค่ากลับออกมาด้วยคำสั่ง return
- ในการผ่านค่า f1 เข้าไปในฟังก์ชัน ค่าใน f1 จะถูก copy ใส่ใน x ซึ่งเป็น local variable และการเปลี่ยนแปลงใดๆกับ x ไม่มีผลกับ f1
- ค่าที่ส่งเข้าไปในฟังก์ชันจะต้องเป็นชนิดเดียวกับชนิดของตัวแปรที่ใช้รับค่านั้น หากชนิดของข้อมูลต่างกัน จะมีการแปลงชนิดข้อมูลให้โดยอัตโนมัติ

8.4 ฟังก์ชันเรียกใช้งานฟังก์ชัน

- ในการคำนวณค่า square root ในคอมพิวเตอร์จะใช้การประมาณค่าด้วยวิธี Newton-Raphson Iteration
- เราจะใช้วิธีนี้ในการเขียนโปรแกรมเพื่อหา square root ในโปรแกรมที่ 8.8
- วิธีของ Newton-Raphson จะเริ่มต้นด้วยการกำหนดค่าเริ่มต้นใดๆ ซึ่งอาจได้จากการเดา จากนั้นจะใช้ค่าเริ่มต้นนั้นไปหารตัวตั้ง แล้วนำผลลัพธ์ไปบวกค่าเริ่มต้นนั้น แล้วหารด้วยสอง ผลลัพธ์ที่ได้จะนำไปทำซ้ำกระบวนการเดิมหลายๆ รอบ

- เงื่อนไขของการหยุดลูปการคำนวณคือความแตกต่างของตัวตั้งกับค่ายกกำลังสองของผลลัพธ์ จะต้องน้อยกว่าค่า ε ที่กำหนด
- กระบวนการคำนวณทั้งหมดแปลงเป็นอัลกอรึทึมได้ดังนี้

อัลกอรึทึมการหาค่ารากที่สองของ x โดยวิธี Newton-Raphson
ขั้นที่ 1: ให้ค่าเริ่มต้นเป็น guess = 1
ขั้นที่ 2: หาก $\lvert guess^2 - x \rvert < \varepsilon$ ให้ข้ามไปยังขั้นที่ 4
ขั้นที่ 3: ให้ guess = (x/guess + guess)/2 และกลับไปยังขั้นที่ 2
ขั้นที่ 4: ค่า guess ที่ได้เป็นค่าประมาณของรากที่สองของ x

- สังเกตว่ามีการใช้ absolute ในขั้นตอนที่ 2 ด้วย
- โปรแกรมต่อไปจะเขียนขึ้นจากอัลกอรึทึมนี้ โดยให้ $\varepsilon = 0.00001$

Program 8.8 Calculating the Square Root of a Number

```
// Function to calculate the absolute value of a number
#include <stdio.h>

float absoluteValue(float x)
{
    if( x < 0 )
        x = -x;
    return(x);
}

// Function to compute the square root of a number
float squareRoot(float x)
{
    const float epsilon = .00001;
    float guess = 1.0;
    while( absoluteValue (guess * guess - x) >= epsilon )
        guess = ( x / guess + guess )/2.0;
    return guess;
}

int main (void)
{
    printf("squareRoot(2.0) = %f\n", squareRoot (2.0));
    printf("squareRoot(144.0) = %f\n", squareRoot (144.0));
    printf("squareRoot(17.5) = %f\n", squareRoot (17.5));
    return 0;
}
```

```
squareRoot(2.0) = 1.414216
squareRoot(144.0) = 12.000000
squareRoot(17.5) = 4.183300
```

- ผลลัพธ์จากการรันโปรแกรมนี้อาจแตกต่างในทศนิยมหลักท้ายๆ เนื่องจากความแม่นยำในการคำนวณที่ต่างกันของแต่ละระบบ
- ฟังก์ชัน absoluteValue ถูกประกาศก่อน ตามด้วยฟังก์ชัน squareRoot
- ฟังก์ชัน squareRoot รับค่าตัวตั้ง x และคืนค่าผลลัพธ์ออกมาเป็นชนิด float โดยที่ภายในฟังก์ชันนี้มีการประกาศตัวแปร local ชื่อ epsilon และ guess

- ลูป while ใช้ในการคำนวณซ้ำเพื่อหาค่ารากที่สองของตัวตั้ง โดยลูปจะวนไปเรื่อยๆจนกระทั่งเงื่อนไข $|guess^2-x| >= \varepsilon$ เป็นเท็จ นั่นคือผลการคำนวณลู่เข้าสู่ผลลัพธ์
- สังเกตว่าทั้งสองฟังก์ชันใช้ตัวแปรชื่อเดียวกัน คือ x เพื่อรับค่าเข้าไปในฟังก์ชัน อย่างไรก็ตามเนื่องจาก x แต่ละตัวเป็น local variable ของฟังก์ชันนั้น จึงถือเป็นคนละตัวแปร
- สังเกตลำดับในการประกาศฟังก์ชัน ฟังก์ชันจะต้องถูกประกาศก่อนเรียกใช้งานเสมอ ดังนั้นฟังก์ชัน absoluteValue ที่ถูกเรียกใช้งานในฟังก์ชั่น squareRoot จะถูกประกาศไว้หน้าสุด
- หากต้องการวางฟังก์ชัน absoluteValue ไว้หลัง squareRoot จำเป็นจะต้องประกาศให้คอมไพล์เลอร์รู้จักฟังก์ชัน absoluteValue เสียก่อน ซึ่งทำได้โดยเพิ่มบรรทัด

  float absoluteValue(float);

  หรือ

  float absoluteValue(float x);

  ไว้หลัง Header หรือหลัง #include <> นั่นเอง สังเกตว่าในวงเล็บไม่จำเป็นต้องมีชื่อตัวแปร มีเพียงแต่ชนิดของตัวแปรก็ได้
- ฟังก์ชัน squareRoot ที่เขียนขึ้นจะมีปัญหาหากส่งค่าเป็นจำนวนลบเข้าไปในฟังก์ชัน ซึ่งจะทำให้อัลกอริทึมของ Newton-Raphson ไม่ลู่เข้า และลูปจะวนเป็นอนันต์ครั้ง
- วิธีป้องกันคือให้ฟังก์ชันตรวจสอบค่าที่รับเข้ามาก่อนทำการคำนวณ

```
/* Function to compute the square root of a number.
If a negative argument is passed, then a message
is displayed and -1.0 is returned. */

float squareRoot (float x)
{
   const float epsilon = .00001;
   float guess = 1.0;

   if( x < 0 )
   {
      printf("Negative argument to squareRoot.\n");
      return -1.0;
   }

   while( absoluteValue (guess * guess - x) >= epsilon )
      guess = ( x / guess + guess ) / 2.0;

   return guess;
}
```

8.5 ฟังก์ชันกับอาร์เรย์

- เราสามารถส่งค่าของสมาชิกแต่ละตัวในอาเรย์หรือส่งทั้งอาเรย์ให้กับฟังก์ชันได้
- การส่งสมาชิกของอาร์เรย์ให้กับฟังก์ชันทำได้โดยอ้างอิงถึงสมาชิกตัวที่ต้องการ เช่น sq_root_result = squareRoot(averages[i]);
- แต่หากต้องการส่งทั้งอาร์เรย์ให้กับฟังก์ชัน ให้ใส่เฉพาะชื่ออาร์เรย์โดยไม่ต้องมี index ตามท้าย ยกตัวอย่างเช่นถ้า gradeScore เป็นอาร์เรย์ที่มีสมาชิก 100 ตัว และมีฟังก์ชัน minimum() การเรียกใช้ minimum(gradeScore); จะผ่านค่าสมาชิกทั้ง 100 ตัวเข้าไปในฟังก์ชัน โดยที่ฟังก์ชัน minimum นี้จะต้องเขียนให้รองรับสมาชิกทั้งหมดด้วย ดังนั้นฟังก์ชัน minimum อาจเขียนได้ดังนี้

```
int minimum(int values[100])
{
     ...
     return minValue;
}
```

- การเรียกใช้ตัวแปร values ในฟังก์ชัน จะอ้างอิงไปที่ตัวแปร gradeScore เช่น values[4] จะหมายถึง gradeScore[4] และการเปลี่ยนแปลงค่าตัวแปร values จะเปลี่ยนแปลงค่าในตัวแปร gradeScore ด้วย

Program 8.9 Finding the Minimum Value in an Array

```
// Function to find the minimum value in an array
#include <stdio.h>

int minimum(int values[10])
{
    int minValue, i;
    minValue = values[0];
    for( i = 1; i < 10; ++i )
        if( values[i] < minValue )
            minValue = values[i];
    return minValue;
}

int main (void)
{
    int scores[10], i, minScore;

    printf("Enter 10 scores\n");

    for(i = 0; i < 10; ++i )
        scanf("%i", &scores[i]);

    minScore = minimum(scores);
    printf("\nMinimum score is %i\n", minScore);
    return 0;
}
```

```
Enter 10 scores
69 97 65 87 69 86 78 67 92 90
Minimum score is 65
```

- ฟังก์ชันในโปรแกรม 8.9 ยังรับอาร์เรย์ที่มีขนาดตายตัว เราสามารถแก้ไขให้ฟังก์ชันรับอาร์เรย์ขนาดใดๆ ได้ ตามตัวอย่างต่อไปนี้

Program 8.10 Revising the Function to Find the Minimum Value in an Array

```
// Function to find the minimum value in an array
#include <stdio.h>

int minimum(int values[], int numberOfElements)
{
    int minValue, i;

    minValue = values[0];
    for( i = 1; i < numberOfElements; ++i )
        if( values[i] < minValue )
            minValue = values[i];

    return minValue;
}

int main (void)
{
    int array1[5] = { 157, -28, -37, 26, 10 };
    int array2[7] = { 12, 45, 1, 10, 5, 3, 22 };

    printf("array1 minimum: %i\n", minimum(array1, 5));
    printf("array2 minimum: %i\n", minimum(array2, 7));
    return 0;
}
```

```
array1 minimum: -37
array2 minimum: 1
```

- ในกรณีนี้ฟังก์ชัน minimum รับสอง argument คืออาร์เรย์ที่ต้องการหาสมาชิกที่มีค่าน้อยที่สุด และ จำนวนสมาชิกในอาร์เรย์นั้น สังเกตว่า values[] มีวงเล็บเปิดปิดบ่งบอกว่าตัวแปร values เป็นชนิดอาร์เรย์

Program 8.11 Changing Array Elements in Functions

```
#include <stdio.h>

void multiplyBy2(float array[], int n)
{
    int i;
    for( i = 0; i < n; ++i )
        array[i] *= 2;
}

int main (void)
{
    float floatVals[4] = { 1.2f, -3.7f, 6.2f, 8.55f };
    int i;

    multiplyBy2(floatVals, 4);
    for ( i = 0; i < 4; ++i )
        printf ("%.2f ", floatVals[i]);
    printf ("\n");
    return 0;
}
```

```
2.40 -7.40 12.40 17.10
```

- จากโปรแกรม ค่าใน floatVals จะถูกเปลี่ยนด้วย
- ลักษณะการจัดการของฟังก์ชันกับตัวแปรทั่วไป และฟังก์ชันกับอาร์เรย์จะแตกต่างกันนั่นคือหากฟังก์ชันเปลี่ยนแปลงค่าของอาร์เรย์ การเปลี่ยนแปลงนั้นจะเกิดขึ้นกับอาร์เรย์ตัวที่ส่งเข้ามายังฟังก์ชัน
- เหตุผลที่เป็นเช่นนี้เพราะเมื่อส่งอาร์เรย์เข้าไปยังฟังก์ชัน ค่าของสมาชิกแต่ละตัวไม่ได้ถูกคัดลอกเข้าไปในฟังก์ชัน แต่เป็นตำแหน่งในหน่วยความจำที่เก็บอาร์เรย์นั้นๆ ที่ถูกส่งเข้าไปในฟังก์ชัน

8.6 การเรียงค่าในอาร์เรย์

- การเรียงค่า หรือ sorting เป็นปัญหาทางคอมพิวเตอร์ที่ได้รับความสนใจมาก โดยมีอัลกอรึทึมมากมายถูกคิดค้น เพื่อให้สามารถเรียงค่าได้รวดเร็วที่สุดโดยใช้หน่วยความจำของคอมพิวเตอร์น้อยที่สุด
- เราจะเขียนฟังก์ชัน sort โดยใช้อัลกอรึทึมอย่างง่ายเพื่อเรียงค่าในอาร์เรย์จากน้อยไปมาก

อัลกอรึทึมการเรียงค่าในอาร์เรย์ที่มีสมาชิก n ตัว

ขั้นที่ 1: ให้ i เป็น 0

ขั้นที่ 2: ให้ j เป็น i+1

ขั้นที่ 3: ถ้า a[i] > a[j] ให้สลับที่ของสมาชิกสองตัวนี้

ขั้นที่ 4: ให้ j เป็น j+1 ถ้า j<n ให้ไปยังขั้นที่ 3

ขั้นที่ 5: ให้ i เป็น i+1 ถ้า i<n-1 ให้ไปยังขั้นที่ 2

ขั้นที่ 6: อาร์เรย์ a ถูกเรียงจากน้อยไปมากแล้ว

จากอัลกอรึทึมเขียนได้เป็นโปรแกรมที่ 8.12

Program 8.12 Sorting an Array of Integers into Ascending Order

```
// Program to sort an array of integers into ascending order
#include <stdio.h>
void sort(int a[], int n)
{
    int i, j, temp;

    for( i = 0; i < n - 1; ++i )
        for( j = i + 1; j < n; ++j )
            if( a[i] > a[j] )
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
}

int main (void)
{
    int i;
    int array[16] = { 34, -5, 6, 0, 12, 100, 56, 22, 44, -3, -9,
12, 17, 22, 6, 11 };

    printf("The array before the sort:\n");
    for( i = 0; i < 16; ++i )
        printf ("%i ", array[i]);

    sort(array, 16);

    printf("\n\nThe array after the sort:\n");
    for( i = 0; i < 16; ++i )
        printf("%i ", array[i]);
    printf ("\n");

    return 0;
}
```

```
The array before the sort:
34 -5 6 0 12 100 56 22 44 -3 -9 12 17 22 6 11

The array after the sort:
-9 -5 -3 0 6 6 11 12 12 17 22 22 34 44 56 100
```

8.7 Global Variable

- ตัวอย่างต่อไปนี้เป็นโปรแกรมแปลงเลขฐานซึ่งดัดแปลงมาจากตัวอย่างในบทที่ผ่านมา โดยแยกการทำงานออกเป็นฟังก์ชันย่อยๆสามส่วน คือส่วนรับค่าเลขตัวตั้งและเลขฐานมาจากผู้ใช้ ส่วนแปลงเลขฐาน และส่วนแสดงผล
- ฟังก์ชันทั้งสามส่วนนั้นติดต่อกันโดยผ่านทาง global variable
- ฟังก์ชันใดๆในโปรแกรมสามารถเข้าถึงและแก้ไข global variable ได้ (ต่างจาก local variable ที่ประกาศ และใช้ในฟังก์ชันหนึ่งๆเท่านั้น)
- การประกาศ global variable จะประกาศไว้นอกฟังก์ชัน

Program 8.14 Converting a Positive Integer to Another Base

```
// Program to convert a positive integer to another base
#include <stdio.h>

int convertedNumber[64];
long int numberToConvert;
int base;
int digit = 0;

void getNumberAndBase(void)
{
    printf(“Number to be converted? “);
    scanf(“%li”, &numberToConvert);
    printf(“Base? “);
    scanf(“%i”, &base);
    if( base < 2 || base > 16 ) {
        printf (“Bad base – must be between 2 and 16\n”);
        base = 10;
    }
}

void convertNumber(void)
{
    do{
        convertedNumber[digit] = numberToConvert % base;
```

```
        ++digit;
        numberToConvert /= base;
    }
    while( numberToConvert != 0 );
}

void displayConvertedNumber(void)
{
    const char baseDigits[16] =
        { ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’,
            ‘8’, ‘9’, ‘A’, ‘B’, ‘C’, ‘D’, ‘E’, ‘F’ };
    int nextDigit;
    printf (“Converted number = “);
    for(--digit; digit >= 0; --digit ) {
        nextDigit = convertedNumber[digit];
        printf (“%c”, baseDigits[nextDigit]);
    }
    printf (“\n”);
}

int main (void)
{
    getNumberAndBase();
    convertNumber();
    displayConvertedNumber();
    return 0;
}
```

```
Number to be converted? 100
Base? 8
Converted number = 144
```

```
Number to be converted? 1983
Base? 0
Bad base – must be between 2 and 16
Converted number = 1983
```

- จากโปรแกรมจะเห็นว่า global variable ถูกใช้สำหรับตัวแปรที่ถูกใช้งานในหลายฟังก์ชัน
- ค่าเริ่มต้นของ global variable จะถูกเซ็ทเป็น 0 เสมอเมื่อเริ่มโปรแกรม

8.8 ฟังก์ชันเรียกใช้งานตัวเอง (Recursive function)

- ภาษาซีอนุญาตให้ฟังก์ชันมีการเรียกใช้งานตัวเองได้ เรียกว่า recursive function
- ตัวอย่างคือนำมาคำนวณหาค่า factorial ของตัวเลขจำนวนเต็ม ซึ่งสามารถเขียนได้เป็น n! = n x (n-1)! เช่น 6! = 6 x 5!

Program 8.16 Calculating Factorials Recursively

```
#include <stdio.h>

int main (void)
{
    unsigned int j;
    unsigned long int factorial(unsigned int n);

    for( j = 0; j < 11; ++j )
        printf("%2u! = %lu\n", j, factorial (j));
    return 0;
}

unsigned long int factorial(unsigned int n)
{
    unsigned long int result;
    if( n == 0 )
        result = 1;
    else
        result = n * factorial(n - 1);

    return result;
}
```

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
```

```
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
```

- สังเกตว่าฟังก์ชัน factorial() มีการเรียกใช้ตัวเองจึงเป็น recursive
- ถึงแม้จะเป็นฟังก์ชันเดียวกันเอง แต่การเรียกใช้ซ้ำแต่ละครั้งจะมีการสร้างชุดของตัวแปรขึ้นมาต่างหากแยกจากกัน ในที่นี้คือตัวแปร result และ n