บทที่ 6 ฟังก์ชัน

ในการเขียนโปรแกรมขนาดใหญ่ เราจำเป็นต้องมีการแบ่งเป็นโปรแกรมย่อย (Subprogram) เพื่อให้ง่ายต่อการเขียน และการแก้ไข อีกทั้งยังเป็นการป้องกันการผิดพลาด และเพิ่มประสิทธิภาพของโปรแกรม ซึ่งโปรแกรมย่อยที่ว่านี้ ในภาษา C เรียกว่า ฟังก์ชัน (*function*) ในบทนี้ เราจะศึกษาถึงวิธีการนำหลักการเขียนโปรแกรมแบบ Top-down Programming และ Modular Programming มาใช้ในการเขียนโปรแกรมภาษา C กล่าวคือ ศึกษาวิธีการเขียนฟังก์ชันในภาษา C

6.1 ฟังก์ชัน (Function)

โปรแกรมย่อย (subprogram) ในแต่ละภาษาคอมพิวเตอร์ จะมีชื่อเรียกที่ต่างกันไป เช่น ในภาษา C, FORTRAN, LISP และภาษาอื่น ๆ อีกหลายภาษา เรียกว่า *function* ในขณะที่ภาษา PASCAL เรียกว่า *Procedure* เป็นต้น function ในภาษา C มีทั้ง function ที่อยู่ใน Library มาตรฐาน ซึ่งเราสามารถเรียกใช้ได้ และ function ที่กำหนดนิยามขึ้นมาเอง ในการสั่งให้ function คืนค่าใดค่าหนึ่ง ทำได้โดยใช้ *return* statement ซึ่งมีวิธีการใช้ดังนี้

```
return ( expression )
```

ยกตัวอย่าง เช่น

```
return (x),  return (1),  return ( a + b )
```

เนื่องจากภาษาคอมพิวเตอร์มีรากฐานมาจากคณิตศาสตร์ ดังนั้น function ในการเขียนโปรแกรมจึงไม่ต่างไปจาก function คณิตศาสตร์ กล่าวคือ

1. ถ้า y = f(x) function f(x) จะหมายถึง ความสัมพันธ์ระหว่างค่า x กับค่า y โดยค่า y จะขึ้นอยู่กับค่า x ดังนั้น x เป็น independent variable และ y เป็น dependent variable ค่า f(x) ของค่า x ค่าใดค่าหนึ่ง จะต้องมีเพียง 1 ค่า ในการเขียนโปรแกรม จะเรียก x ว่าเป็น *argument* ของ Function ซึ่งอาจจะมีมากกว่า 1 ก็ได้ และ y เป็นค่าที่ได้จาก function f(x) เรียกว่า *return value* ซึ่งจะต้องมีชนิดของข้อมูลสอดคล้องกับค่าของ function นั้น ยกตัวอย่าง เช่น Function *sin*(*x*) จะมี return value เป็นค่าจำนวนจริง เป็นต้น
2. ถ้าให้ X, Y เป็นเซ็ต $\{(x,f(x)) \mid x \in X, f(x) \in Y\}$ จะเป็น SUBSET ของ $X \times Y$ และแสดงถึงกราฟ y = f(x) ในที่นี้ เรียก X ว่า Domain และเรียก Y ว่า Range ซึ่งจะต้องสอดคล้องกับตัว function อย่างเช่น sin(x) x จะต้องเป็นเลขจำนวนจริง และหน่วยเป็น Radian และ y จะเป็นจำนวนจริงเช่นกัน Domain จะเท่ากับ SET ของเลขจำนวนจริง และ Range คือ $\{y \mid y \in \mathbf{R}, -1 \leq y \leq 1\}$ เป็นต้น

ซึ่งในการใช้ Function ในภาษา C จำเป็นต้องคำนึงถึงคุณสมบัติ 2 ประการข้างต้น โดยสรุป จะต้องใช้ Data Type ให้เหมาะสมทั้งในส่วนของ Argument และ Return Value

Function Prototype Declaration

การประกาศ function prototype เป็นการแจ้งให้ Compiler รู้จัก function ที่กำหนดนิยามขึ้นเอง โดยมีรูปแบบดังนี้

```
ReturnValueType FunctionName(DataType parameter1,DataType parameter2,...);
```

ดังตัวอย่างเช่น

```
int power(int x);
```

Function Definition

การกำหนดนิยามของ function ตามมาตรฐานที่ ANSI กำหนด จะทำในลักษณะต่อไปนี้

```
ReturnValueType FunctionName(DataType parameter1,DataType parameter2,...)
{
      local variable declarations
      executable statements
}
```

โดยบรรทัดแรกของนิยามมีชื่อเรียกว่า function header และส่วนที่เหลือมีชื่อเรียกว่า function body

อนึ่ง ในกรณีที่เขียนในรูปแบบของ K&R C (Traditional C) จะเขียนในลักษณะดังนี้

```
return value type   function_name(par1, par2,..)
data_type    par1 ;
data_type    par2 ;
...
{
       local variable declarations
       executable statements
}
```

ยกตัวอย่าง เช่น Function สำหรับคำนวณเลขยกกำลังสองของจำนวนเต็ม สามารถเขียนได้ดังนี้

```
int power (int x)
/*     or     int power(x)
              int    x ;        in old C  */
{
       return (x*x) ;      /* return the value x*x to main function */
}
```

หมายเหตุ ในปัจจุบัน ANSI C เป็นมาตรฐานที่โปรแกรม C Compiler ทั่วไปยึดถืออยู่ ในขณะที่ C Compiler บางโปรแกรมไม่ได้สนับสนุนการใช้ K&R C ดังนั้น ควรจะเขียนในสไตล์ ANSI C

6.2 การเรียกใช้ function

การเรียกใช้ function ในภาษา C ขึ้นอยู่กับประเภทของ function ถ้า function นั้น เป็น function ที่มีการคืนค่ามาที่ main จะต้องมีตัวแปรสำหรับรับค่าที่คืนกลับมา แต่ถ้าไม่มีการคืนค่า ก็ไม่ต้องมีตัวแปรที่ว่านี้ ดังนั้น จึงแบ่งได้เป็น 2 วิธี ใหญ่ ๆ คือ

1. function ที่มีการคืนค่า

```
Usage :     variable_name = function_name(arguments) ;
```

ยกตัวอย่าง เช่น

```
       x = power(10) ;
```

2. function ที่ถูกกำหนดให้ไม่ต้องคืนค่า หรือ function ที่เราไม่ต้องการรับค่าที่คืนมา เช่น การเรียกใช้ printf, scanf เป็นต้น

```
Usage :     function_name(arguments) ;
```

ยกตัวอย่าง เช่น

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

ข้อควรระวังเกี่ยวกับ function

- function แต่ละ function จะคืนค่าได้เพียง 1 ค่าเท่านั้น หรือไม่มีการคืนค่าในกรณีที่เรากำหนด return value type ให้เป็นแบบ void ซึ่งนิยมใช้ในกรณีที่ต้องการให้คืนค่าได้มากกว่า 1 ค่า โดยจะทำการคืนในรูปของ argument หรือ Call by reference
- data type ของ arguments ตอน declare กับตอนเรียกใช้จะต้องสอดคล้องกัน
- เราสามารถกำหนดให้ไม่ต้องรับค่าที่ function คืนมาได้ โดยใส่ (void) นำหน้า function_name(...) เช่น (void) swap(x, y)
- การ declare function เราสามารถเขียนไว้ใน *Header File* แล้ว include ไฟล์นั้นได้ หรือไม่ก็เขียน Function Body ไว้ก่อนหน้า main function ก็จะเป็นการ declare function ไปในตัว

ตัวอย่างที่ 6.1 : Usage of function (Function add)

```
#include <stdio.h>
int add(int a, int b); /* Function add declaration : notice semi-colon */
int main()
{
       int    c ;
       c = add(10, 20) ;   /* Call Function add, notice arguments */
```

```
        printf("result = %d\n", c) ;
        return 0;
}
int add(int a, int b)       /* Function Body : notice semi-colon-less */
{
        int    sum ;
        sum = a + b ;
        return(sum) ; /* Return the value sum to main */
}
```

ผลการรันโปรแกรม

```
result = 30
```

ตัวอย่างที่ 6.2 : Usage of function (Converting integer to Roman number)

```
#include <stdio.h>
char   *int2roman(int number) ;
              /* declare function int2roman , notice semi-colon */
int main()
{
        int    a ;
        char   *b ; /* Prepare string for receiving the return value */
        printf("Enter number (1-5) :") ;
        scanf("%d", &a) ;
        b = int2roman(a) ;          /* Function call */
        printf("%s\n", b) ;
        return 0;
}
char   *int2roman(int number)
              /* function int2roman , notice there is no semi-colon */
{
        switch ( number ) {
              case 1 : return("I");
              case 2 : return("II") ;
              case 3 : return ("III") ;
              case 4 : return ("IV") ;
              case 5 : return ("V") ;
              default :return ("UNKNOWN") ;
        }
}
```

ผลการรันโปรแกรม

```
Enter number (1-5) :4
IV
```

สังเกตว่า ไม่มี *break* statement ใน switch-statement-block จงพิจารณาว่า เพราะเหตุใด

คำถาม : ค่าที่ function *int2roman* คืน เป็นค่าประเภทไหน

ตัวอย่างที่ 6.3 : Usage of void type function (Converting integer number to Roman number)

```
#include <stdio.h>
void int2roman2(int number) ;     /* void type function declaration */
int main()
{
        int    a ;
        printf("Enter number (1-5) :") ;
        scanf("%d", &a) ;
        int2roman2(a) ;
              /* function call : notice assignment-operator-less */
        return 0;
}
void   int2roman2(int number)
{
        switch ( number ) {
              case 1 : printf("%d = I\n",number) ; break ;
              case 2 : printf("II\n") ; break ;
              case 3 : printf("III\n") ; break ;
              case 4 : printf("IV\n") ; break ;
              case 5 : printf("V\n") ; break ;
              default : printf("UNKNOWN\n") ;
```

```
        }
}
```

ผลการรันโปรแกรม

```
Enter number (1-5) :3
III
```

ผลการรันโปรแกรมของโปรแกรมนี้ จะเหมือนกับตัวอย่าง 6.2 และให้สังเกตว่า ต้องมี *break* statement ใน switch-statement-block จงพิจารณาว่า เพราะเหตุใด

6.3 วิธีการรับส่งค่าของตัวแปร (The way to call the value of variables)

ในภาษา C มีวิธีการรับส่งค่าของตัวแปรระหว่าง main function กับ function อื่น ๆ ได้ 2 วิธีคือ Call by value และ Call by reference

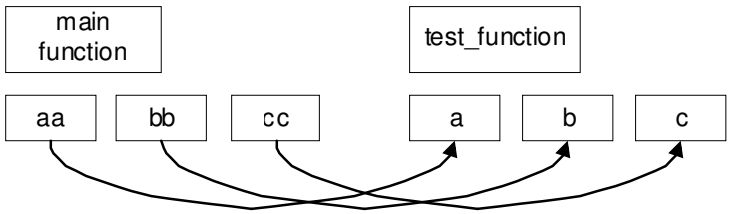
Call by value

Call by value เป็นการรับค่าที่ใช้กันทั่วไป มีรูปแบบง่าย ๆ คือ เมื่อต้องการส่งค่าของตัวแปรตัวไหนหรือค่าอะไรให้กับ function ก็ใส่เข้าไปเป็น argument ของ function นั้น ๆ กรณีที่ใช้วิธีนี้ในการรับส่งค่า โปรแกรมจะทำการ Copy ค่าที่จะส่งไปยังตัวแปรที่มีไว้สำหรับรับค่านั้น (รูปที่ 6.2) ยกตัวอย่าง เช่น ถ้า function declaration เป็น

```
int    test_function(int a, float b, double c) ;
```

แล้วใน main function จะต้องทำให้อยู่ในลักษณะดังนี้

```
int    aa, d ;                    /* main function */
float  bb ;
double        cc ;
d = test_function(aa, bb, cc) ;   /* function call */
```



รูปที่ 6.2 Call by value

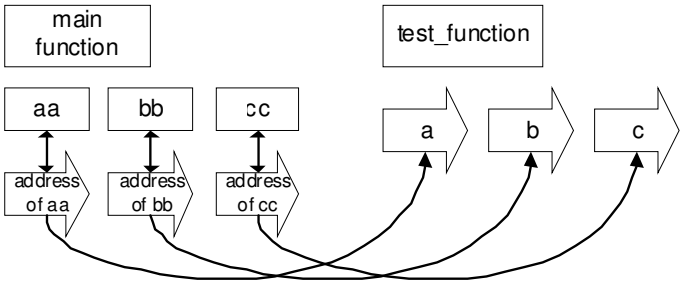
จะสังเกตเห็นว่า เราไม่มีความจำเป็นที่จะต้องใช้ชื่อตัวแปรให้เหมือนกัน เพราะเป็นตัวแปรคนละตัว หรืออาจจะใช้ชื่อเดียวกันก็ไม่ก่อให้เกิดปัญหาอะไร เพราะไม่ได้อยู่ใน function เดียวกัน แต่ทั้งนี้ ชนิดของข้อมูลของ Argument จะต้องสอดคล้องกัน และจะต้องระวังเรื่องการใช้ GLOBAL Variable (ตอนที่ 6.6)

วิธีการรับส่งข้อมูลวิธีนี้ มีข้อเสียคือ โปรแกรมจะต้องทำการ Copy ค่าทั้งหมดจาก function หนึ่งไปยังอีก function หนึ่ง ถ้าหากมีจำนวน argument มาก จะทำให้เสียเวลาในการส่งค่ามาก ในกรณีนี้ ควรใช้วิธี Call by reference แทน

Call by reference

Call by reference เป็นการรับส่งค่าของตัวแปร โดยอ้างอิงถึง pointer กล่าวคือ address ของตัวแปรนั้น ในวิธีนี้โปรแกรมจะทำการส่งค่า address ของตัวแปรนั้นไปให้ แทนการส่งค่าไป (รูปที่ 6.3) ยกตัวอย่าง เช่น

```
int    test_function(int *a, float *b, double *c) ;
int    aa, d ;                    /* main function */
float  bb ;
double        cc ;
d = test_function(&aa, &bb, &cc) ;      /* function call */
```



รูปที่ 6.3 Call by reference

Call by reference จะใช้ในกรณีที่มีการอ้างอิงถึงข้อมูลที่มีขนาดใหญ่ หรือข้อมูลที่สะดวกในการอ้างอิงโดยใช้ pointer อันได้แก่ การส่งข้อมูล array หรือ string เป็นต้น เพื่อเพิ่มประสิทธิภาพของโปรแกรม หรือในกรณีที่ต้องการใช้ตัวแปรร่วมกันระหว่าง main function กับ function เพื่อที่จะได้เป็นการรับค่าของตัวแปรนั้น (หรือที่เรียกว่า Multiple Return Value Function) สิ่งที่พึงระวังอย่างยิ่งเกี่ยวกับการใช้วิธี Call by reference ก็คือ อาจมีการเปลี่ยนค่าของตัวแปรในฟังก์ชันที่ถูกเรียกใช้ได้

ตัวอย่างที่ 6.4 Call array (Notice the difference of data type in arguments of 2 functions)

```
#include <stdio.h>
void cal_sum1(int *ptr_array, int number) ;
                  /* use pointer , number represents dimension */
void cal_sum2(int a[], int number) ;           /* use array */
int main()
{
       int    a[5] = {1, 2, 3, 4, 5} ;          /* initialization */
       cal_sum1(a, 5) ;
       /* call cal_sum1 function by using pointer
                  (the address of the first element, i.e. &a[0]). */
       cal_sum2(a, 5) ;           /* call cal_sum2 function by using array */
       return 0;
}
void cal_sum1(int *dt, int number)
{
       int    i, sum = 0 ;
       for (i = 0 ; i < number; i++) {
              sum += *dt ;  /* calculate sum of the value that dt points at */
              ++dt ;                      /* increment dt */
       }
       printf("sum1 = %d\n", sum) ;
}
void cal_sum2(int a[], int number)
{
       int    i, sum = 0 ;
       for (i = 0; i < number; i++)
              sum += a[i] ;  /* calculate sum of the value of the array */
       printf("sum2 = %d\n", sum) ;
}
```

ผลการรันโปรแกรม

```
sum1 = 15
sum2 = 15
```

ตัวอย่างที่ 6.5 Call string (สังเกตว่า ไม่จำเป็นต้องมี Function Declaration ถ้านำ Function มาไว้หน้า Main Function)

```
#include <stdio.h>
void strdisp1(char *p)
{
       printf("string1 = %s\n", p) ;
}
void strdisp2(char str[])
{
       printf("string2 = %s\n", str) ;
}
```

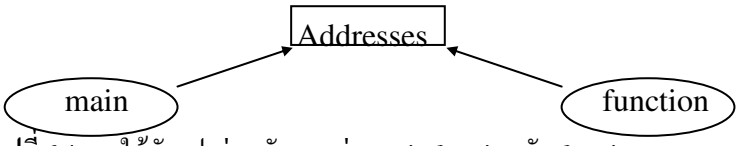
```
int main()
{
       char s[] = "abcdefgh" ;
       strdisp1(s) ;
       /* call by using pointer (address of the first element) */
       strdisp1("XYZ") ;   /* call by using pointer */
       strdisp2(s) ; /* call by using array */
       strdisp2("XYZ") ;   /* call by using array */
       return 0;
}
```

ผลการรันโปรแกรม

```
string1 = abcdefgh
string1 = XYZ
string2 = abcdefgh
string2 = XYZ
```

6.4 ฟังก์ชันที่คืนได้หลายค่า (Multiple return value function)

โดยทั่วไป Function ในภาษา C จะคืนค่าได้เพียง 1 ค่าเท่านั้น แต่เราสามารถทำให้ function สามารถคืนค่าได้ทีละหลาย ๆ ค่า โดยใช้ Call by reference หรือการใช้ pointer เป็น argument กล่าวคือ เป็นการใช้ตัวแปรร่วมกันระหว่าง main function กับ function โดยจะมีลักษณะดังรูปที่ 6.4



รูปที่ 6.4 การใช้ตัวแปรร่วมกันระหว่าง main function กับ function

ตัวอย่างที่ 6.6 Multiple-value return function : simple arithmetic program

```
#include <stdio.h>
void calculate(int x, int y, int *add, int *substract, int *multiply, int
*divide) ;
int main()
{
       int    a, b, c, d ;
       calculate(100, 20, &a, &b, &c, &d) ;
       printf("add : %d substract : %d multiply : %d divide : %d\n",a,b,c,d);
       return 0;
}
void calculate(int x, int y, int *add, int *substract, int *multiply, int
*divide)
{
       *add = x + y ;
       *substract = x - y ;
       *multiply = x * y ;
       *divide = x / y ;
}
```

ผลการรันโปรแกรม

```
add : 120 substract : 80 multiply : 2000 divide : 5
```

ตัวอย่างที่ 6.7 Swap program (2 Versions)

```
void swap(int *x, int *y)
{
       int    tmp ;
       tmp = *x ;
       *x = *y ;
       *y = tmp ;
}
```

```
void swap(int x, int y)
                /* Incorrect version */
{
       int    tmp ;
       tmp = x ;
       x = y ;
       y = tmp ;
}
```

คำถาม : จงเปรียบเทียบ swap function 2 version ในตัวอย่าง 6.7 และพิจารณาสาเหตุของความแตกต่าง

6.5 Recursion

ความหมาย

1 the act or an instance of returning

2 *Math.* the repeated application of a procedure or definition to a previous result to obtain a series of values

Recursion จะเกิดขึ้นเมื่อเกิดการเรียกตัว Function นั้นเองใน function function ที่มีการเรียกตัวเอง จะเรียกว่า Recursive Function และในการเขียนโปรแกรมสำหรับงานบางอย่าง จะมี Algorithm ที่มีลักษณะนี้ (เรียกว่า Recursive Algoritm) เช่น การเขียนโปรแกรมสำหรับคำนวณ factorial เป็นต้น

Recursive Fuction จะต้องยึดหลัก 2 ประการต่อไปนี้

1. ต้องมีจุดสิ้นสุด
2. ต้องทำให้การแก้ปัญหาซับซ้อนน้อยลง

Recursive Algorithms ที่เราจะเขียน โดยทั่วไปจะประกอบด้วย if statement ที่มีรูปแบบดังต่อไปนี้

if this is a simple case
    solve it
else
    redefine the problem using recursion

Definition of factorial :

```
fact(0) = 1
fact(n) = n * fact(n-1)
```

ตัวอย่างที่ 6.8 Factorial program (only fact function)

```
int fact(int number)
{
       if ( number == 0 )
              return (1) ;
       else
              return ( number * fact(number - 1) ) ;
}
```

ตัวอย่างที่ 6.9 Count character number in name

```
#include <stdio.h>
int    count(char ch, char str[], int first) ;
int main()
{
       char str[100] ;
       char first[10], last[30] ;
       printf("Enter your first name\n") ;
       scanf("%s", first) ;              /* read your first name */
       printf("Enter your last name\n") ;
       scanf("%s", last) ;               /* read your last name */
       printf("Enter a character\n") ;
       scanf("%c", &ch) ;                /* read a character */
       strcpy(str, first) ;              /* copy first to str */
       strcat(str, " ") ;         /* concatenate space to str */
       strcat(str, last) ;        /* concatenate last to str */
       printf(" number of %c in %s is %d \n",
              ch, str, count(ch, str, 0)) ;    /* print result */
       return 0;
}
int    count(char ch, char str[], int first)
{
       if ( str[first] == '\0' ) return (0) ;
       else
       {
              if ( ch == str[first] )
                     return (1 + count(ch, str, ++first)) ;
              else
                     return (count(ch, str, ++first)) ;
       }
}
```

ผลการรันโปรแกรม

```
Enter your first name
Kittirat
Enter your last name
Intarat
Enter a character
a
 number of a in Kittirat Intarat is 3
```

ตัวอย่างที่ 6.10 การหาค่า ห.ร.ม. โดย Recursive Function

การหาค่า ห.ร.ม. ของจำนวนเต็ม x, y สามารถหาได้จากสมการคณิตศาสตร์ต่อไปนี้

$$\gcd(x,y) = \begin{cases} x & (when\ y = 0) \\ \gcd(y, x \bmod y) & (when\ y \neq 0) \end{cases} \qquad (6.1)$$

ซึ่งเป็น Recursive Function คล้าย ๆ กับ Function Factorial ซึ่งสามารถเขียนเป็น Recursive Function ในภาษา C ได้ดังนี้

```
int gcd(int x, int y)
{
       if ( y == 0 ) return x ;
       else return gcd(y, x % y) ;
}
```

คำถาม ทดลองเขียน Recursive Function สำหรับคำนวณหา Fibonacci Number (ตัวอย่างที่ 3.4)

6.6 Variable Type and Storage Class

Variable type & Scope

เราสามารถแบ่งประเภทตัวแปรที่ใช้ในโปรแกรมในภาษา C ตาม Scope (ขอบเขตในโปรแกรมที่สามารถใช้ตัวแปร) ของตัวแปรนั้น ได้เป็น

1. **Global variable** เป็นตัวแปรที่ถูก declare ภายนอกทุก function และมีค่าเริ่มต้น default เท่ากับ 0 สามารถเรียกใช้ได้จากทุก function และจะถูกเก็บอยู่ในหน่วยความจำตลอดระยะเวลาที่รันโปรแกรมนั้นอยู่
2. **Local variable** เป็นตัวแปรที่ถูก declare ภายใน function ใด function หนึ่ง ใช้ได้แต่เฉพาะใน function นั้น ๆ เมื่อจบ function นั้น ๆ โปรแกรมก็จะทำการเลิกใช้ตัวแปรเหล่านี้ กล่าวคือ โปรแกรมจะยกเลิกการจองเนื้อที่ในหน่วยความจำสำหรับเก็บค่าตัวแปรเหล่านี้ เพื่อเอาไว้ใช้สำหรับเก็บค่าตัวแปรตัวอื่น ๆ หรือโปรแกรมส่วนอื่น ๆ

```
void func(int a, int b) ;
int    g ;                  /* g is global variable */
void main()
{
       int    a ;           /* a is local variable */
       ...
}
void func(int a, int b)
{
       int    c, d ;        /* c, d are local variables */
       ...
}
```

ข้อควรระวัง Global variable จะเป็นตัวแปรที่ตัวโปรแกรมทำการจองที่ไว้ตั้งแต่เริ่มทำงาน จนกระทั่งจบโปรแกรมนั้น ส่วน Local variable โปรแกรมจะจองที่เฉพาะที่มีการเรียกใช้ function เท่านั้น ดังนั้น หากใช้ตัวแปรประเภท Global เป็นจำนวนมาก จะทำให้ต้องใช้เนื้อที่ในหน่วยความจำมาก อาจมีผลต่อการทำงานของโปรแกรมได้ นอกจากนี้ ค่าของตัวแปรประเภท Global จะมีการเปลี่ยนแปลงตลอดเวลา แล้วแต่มีการกำหนดค่าที่ตรงตำแหน่งไหนของโปรแกรม ดังนั้น อาจจะเกิดความผิดพลาดได้ง่าย เพราะฉะนั้น ควรจะใช้ Global variable เฉพาะในกรณีที่ต้องการใช้ตัวแปรร่วมกันในหลาย ๆ function เท่านั้น

Storage Classes

ในภาษา C เราสามารถกำหนดประเภทของหน่วยความจำที่ตัวโปรแกรมจะใช้ในการเก็บค่าของตัวแปรได้ ทั้งนี้เพื่อประสิทธิภาพและความยืดหยุ่นในการเขียนโปรแกรม โดยมีดังนี้

1. auto เป็นแบบที่ใช้กันมากที่สุด C จะทำการจองที่ (Allocate) ในหน่วยความจำสำหรับตัวแปรแบบนี้ และจะยกเลิกการจองที่ (Unallocate) เมื่อใช้เสร็จ เช่น ตัวแปรประเภท local ใน function เป็นต้น หากไม่ได้ระบุ storage class เป็นอะไรชัดเจน Compiler ก็จะจัดให้เป็นแบบนี้
2. static ตัวแปรแบบนี้ จะจองเมื่อมีความจำเป็นต้องใช้ และมีค่าเริ่มต้น default เท่ากับ 0 โดยจะจองอยู่ตลอดเวลาที่โปรแกรมนั้นทำงานอยู่
3. extern ตัวแปรแบบ extern เป็นตัวแปรที่ถูก declare ไว้ที่อื่น เช่น ที่ไฟล์อื่น เป็นต้น
4. register ตัวแปรแบบ register จะเหมือนกับแบบ auto type เพียงแต่ว่าแทนที่จะเก็บค่าไว้ใน Memory จะเก็บค่าไว้ใน Register ซึ่งทำให้ความเร็วในการประมวลผลเร็วกว่าแบบ auto

```
Usage of Storage Classes : [storage class] data type variable_name ;
```

ตัวอย่าง เช่น

```
static int count ;
```

ตัวอย่างที่ 6.11 Usage of static class

```
#include <stdio.h>
void sum(int d)
{
       static int    dt ;   /* dt is static type, initial value is 0 */
       dt = dt + d ; /* Since dt is the static type, the value is stored. */
       printf("Sum = %d\n", dt) ;
}
int main()
{
       int    a ;
       for ( a = 1; a <= 4; a++ ) sum(a) ;
       return 0;
}
```

ผลการรันโปรแกรม

```
Sum = 1
Sum = 3
Sum = 6
Sum = 10
```

แบบฝึกหัด

1. ถ้าใส่ข้อมูลเป็น abcd จงหา Output ที่ได้จากโปรแกรมต่อไปนี้

```
#include <stdio.h>
void mystery( void ) ;
void main()
{
  mystery() ;
}
void mystery( void )
{
  int c ;
  if ( ( c = getchar() ) != '\n' ) {
    mystery() ;
    putchar(c) ;
  }
}
```

2. เขียนฟังก์ชันสำหรับคำนวณหาค่าความต้านทาน โดยรับค่าแถบสีเข้าไป ซึ่งแถบสีมีค่าตามตารางที่ 6.3

ตารางที่ 6.3 แถบสีสำหรับใช้แสดงค่าความต้านทาน

Color	*Code*	*Color*	*Code*	*Color*	*Code*
Black	0	Yellow	4	Gray	8
Brown	1	Green	5	White	9
Red	2	Blue	6	Silver	.01
Orange	3	Violet	7	Gold	.1

และมีวิธีการคำนวณดังต่อไปนี้

$$(\text{First Band Code} \times 10 + \text{Second Band}) \times 10^{\text{Third Band Code}} \qquad (6.5)$$

ยกตัวอย่าง เช่น RED BLACK ORANGE หมายถึง

$$(2 \times 10 + 0) \times 10^3 = 20\ \text{k}\Omega$$

เป็นต้น

3. เขียนฟังก์ชันสำหรับกำหนดแถบสีของตัวต้านทาน โดยรับค่าเป็นค่าความต้านทานที่มีหน่วยเป็น Ohm (Ω)
4. เขียนฟังก์ชันสำหรับคำนวณจำนวน Subset ที่มีจำนวนสมาชิก r ตัวของ Set ที่มีจำนวนสมาชิก n ตัว (Combination) ซึ่งแทนด้วยสัญลักษณ์ ${}_nC_r$ ที่สามารถคำนวณได้ตามสมการต่อไปนี้

$$\binom{n}{r} = {}_nC_r = \frac{n!}{r!(n-r)!} \qquad (6.6)$$

ทั้งนี้ ในการรับค่า จะต้องกำหนดเงื่อนไขให้ n มากกว่า r

5. เขียนฟังก์ชันสำหรับทำการแปลงตัวอักษรภาษาอังกฤษให้เป็น Morse Code ที่แสดงในตารางที่ 6.4

ตารางที่ 6.4 Morse Code Table

Letter	*Code*	*Letter*	*Code*	*Letter*	*Code*	*Letter*	*Code*
A	. _	H	. . . .	O	_ _ _	V	. . . _
B	_ . . .	I	. .	P	. _ _ .	W	. _ _
C	_ . _ .	J	. _ _ _	Q	_ _ . _	X	_ . . _
D	_ . .	K	_ . _	R	. _ .	Y	_ . _ _
E	.	L	. _ . .	S	. . .	Z	_ _ . .
F	. . _ .	M	_ _	T	_		
G	_ _ .	N	_ .	U	. . _		

6. เขียนฟังก์ชันสำหรับแปลง Morse Code ที่ได้จากโปรแกรมในข้อ 4 ให้เป็นตัวอักษรภาษาอังกฤษ
7. เขียนฟังก์ชัน power สำหรับคำนวณเลขยกกำลัง โดยรับค่า Argument 2 ตัว คือ x และ n และคืนค่า x^n กลับมา ทั้งนี้ ให้ชนิดตัวแปรของ x และ n เป็น double และ int ตามลำดับ
8. เขียนฟังก์ชันสำหรับหาค่า Square Root ด้วยการประมาณค่าดังต่อไปนี้

$$\frac{x/r + r}{2}$$

โดยเริ่มต้นจากค่า r เท่ากับ 1 และคำนวณจนกว่า $|r^2 - x| < \varepsilon$ ให้ Function รับค่า Argument 2 ค่าคือ ค่า x (ตัวเลขที่ต้องการหา Square Root) และค่า ε (ค่าความผิดพลาด เช่น 0.0001 เป็นต้น)

9. เขียนฟังก์ชันสำหรับคำนวณหาค่า Square Root เช่นเดียวกับในข้อ 7 เพียงแต่เปลี่ยนเงื่อนไขการสิ้นสุดการคำนวณเป็น ค่าความแตกต่างระหว่างค่าประมาณ 2 ค่าที่ต่อเนื่องกัน น้อยกว่า ε หรือ $|r_{I+1} - r_I| < \varepsilon$
10. เขียน Recursive Function สำหรับคำนวณหาค่าผลรวมต่อไปนี้

$$S(n) = 2 + 4 + 6 + \ldots + 2n \qquad (6.7)$$

11. เขียนฟังก์ชันสำหรับคำนวณหาเลขจำนวนมากที่สุด และเลขจำนวนรองลงมา โดยทำให้อยู่ใน Function เดียวกัน
12. เขียนฟังก์ชันสำหรับการหาค่า Integral $\int_0^1 \frac{4}{1+x^2}$ ด้วย Trapezoidal Rule ซึ่งค่าของ $\int_a^b f(x)\,dx$ สามารถประมาณค่าได้ตามสมการต่อไปนี้

$$T = \frac{h}{2}\sum_{i=1}^{n}[f(x_{i-1}) + f(x_i)] \qquad (6.8)$$

โดยให้ $h = (b - a)/n$, $i = 1, \ldots, n$, $x_0 = a$, $x_1 = a + h$, $x_2 = a + 2h$, ... , $x_{n-1} = a + (n-1)h$, $x_n = b$

13. เขียนฟังก์ชันสำหรับแก้สมการกำลังสอง $ax^2 + bx + c = 0$
14. เขียนฟังก์ชันสำหรับเรียงลำดับข้อมูลด้วยวิธี Insertion Sort, Bubble Sort และ Quick Sort
15. เขียนฟังก์ชันสำหรับคำนวณการบวก ลบ คูณ หารของเลขจำนวนเชิงซ้อน
16. เขียนฟังก์ชันสำหรับการหา Inverse Matrix
17. เขียนฟังก์ชันสำหรับแปลงตัวอักษรตัวเล็กเป็นตัวอักษรตัวใหญ่ (จำลอง Function toupper)
18. เขียนฟังก์ชันสำหรับแปลงเลขฐานสิบเป็นเลขฐาน 2, 4, 5, 6, 7, และ 8
19. เขียนฟังก์ชันสำหรับแปลงเลขฐานสิบเป็นเลขโรมัน
20. เขียนฟังก์ชันสำหรับคำนวณ Intersection, Union, Difference และ Complement ของเซ็ต 2 อัน