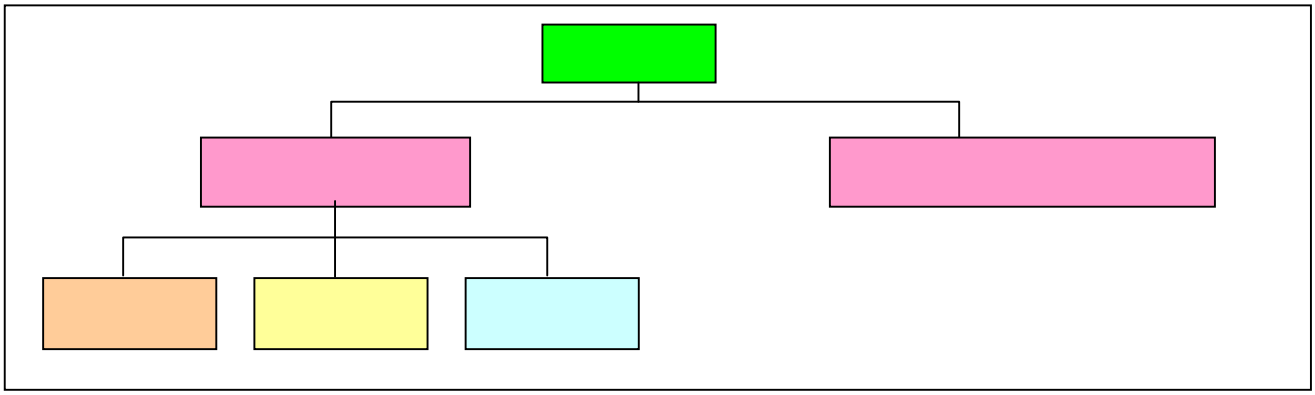
บทที่ 6 ฟังก์ชัน

(Function)

การเขียนโปรแกรมที่มีขนาดใหญ่ประกอบด้วยการทำงานหลายส่วน และมีความซับซ้อน นักเขียนโปรแกรมจึงแบ่งการทำงานออกเป็นส่วนย่อย หรือโมดูล (modules) โดยแต่ละโมดูลมีการทำงานที่เป็นอิสระ ทำให้สามารถควบคุมการดำเนินงานได้ง่าย สำหรับในภาษา C นั้น เรียกโมดูลว่าฟังก์ชัน(function) ฟังก์ชันเหล่านี้อาจเป็นฟังก์ชันสำเร็จรูปที่เก็บไว้ในคลังมาตรฐานของภาษา C (C Standard library) คลังมาตรฐานนี้เป็นที่รวมของฟังก์ชันต่าง ๆ เช่น ข้อมูลนำเข้าและข้อมูลส่งออก การดำเนินงานของสายอักขระ การคำนวณทางคณิตศาสตร์ และการดำเนินงานอื่น ๆ สิ่งเหล่านี้เป็นการอำนวยความสะดวกสำหรับนักเขียนโปรแกรม ฟังก์ชันอีกประเภทหนึ่งได้แก่ฟังก์ชันที่กำหนดขึ้นโดยนักเขียนโปรแกรม ซึ่งสามารถทำงานได้ตามคำสั่งที่กำหนดขึ้นโดยนักเขียนโปรแกรม และจะนำมากล่าวต่อไปในบทนี้



รูปที่ 6.1 แสดงการแบ่งประเภทของฟังก์ชัน

6.1 ฟังก์ชันทางคณิตศาสตร์

ฟังก์ชันในคลังมาตราฐาน แบ่งเป็น 3 ประเภท คือ

1. ฟังก์ชันทางคณิตศาสตร์
2. ฟังก์ชันสำหรับข้อมูลนำเข้าและข้อมูลส่งออก
3. ฟังก์ชันอื่นๆ เช่น ฟังก์ชันบูลีน และ ฟังก์ชันสายอักขระ เป็นต้น

ฟังก์ชันสำหรับข้อมูลนำเข้าและส่งออก และฟังก์ชันสายอักขระ ได้กล่าวถึงไปแล้วในบทก่อน ในบทนี้จะกล่าวถึงเฉพาะฟังก์ชันทางคณิตศาสตร์เท่ากัน

ผู้ใช้สามารถดำเนินการคำนวณได้โดยเรียกใช้ฟังก์ชันทางคณิตศาสตร์ที่กำหนดไว้แล้วในภาษา C การเรียกใช้ฟังก์ชันเหล่านี้สามารถทำได้โดยเรียกชื่อฟังก์ชัน แล้วใส่ตัวอาร์กิวเมนต์ไว้ในวงเล็บที่เขียนตามหลังชื่อฟังก์ชัน เช่น นักเขียนโปรแกรมต้องการคำนวณและพิมพ์ค่าของรากที่สองของ 900.00 สามารถเขียนได้ดังนี้

```
printf("%6.2f",sqrt(900.00));
```

เมื่อโปรแกรมเรียกใช้ฟังก์ชัน sqrt จะคำนวณหารากที่สองของมูลค่าที่กำหนดไว้ในวงเล็บ ในที่นี้คือ 900.00 ซึ่งเป็นอาร์กิวเมนต์ของฟังก์ชัน เมื่อโปรแกรมคำนวณเสร็จแล้วจะได้ค่า 30.00

ฟังก์ชัน	หน้าที่	ตัวอย่าง
sqrt(x)	หารากที่ 2 ของ x	sqrt(900.0) is 30.0 sqrt(9) is 3
exp(x)	ฟังก์ชันเอ็กโปเนนเชียล ของ x	exp(1.0) is 2.718282 exp(2.0) is 7.389056
log(x)	ล็อกฐาน e ของ x	log(2.718282) is 1.0 log(7.389056) is 2.0
log10(x)	ล็อกฐาน 10 ของ x	log10 (1.0) is 0.0 log10 (10.0) is 1.0 log10 (100.0) is 2.0
fabs(x)	ค่าสัมบูรณ์ของ x	if x > 0 then fabs(x) is x if x = 0 then fabs(x) is 0.0 if x < 0 then fabs(x) is -x
ceil(x)	ปรับค่า x ให้เป็นจำนวนเต็มที่มากกว่าหรือเท่ากับ x	ceil(9.2) is 10.0 ceil(-9.8) is -9.0
floor(x)	ปรับค่าของ x ให้เป็นจำนวนเต็มที่ใหญ่สุดที่น้อยกว่าหรือเท่ากับ	floor(9.2) is 9.0 floor(-9.8) is –10.0
pow(x,y)	x ยกกำลัง y	pow(2, 7) is 128.0 pow(9, 0.5) is 3.0
fmod(x,y)	เศษที่เหลือของการหาร x ด้วย y	fmod(13.657, 2.333) is 1.992
sin(x)	ค่าซายน์ของ x (x หน่วย เรเดียน)	sin(0.0) is 0.0
cos(x)	ค่าโคซายน์ ของ x (x หน่วย เรเดียน)	cos(0.0) is 1.0
tan(x)	ค่าเทนเจนท์ ของ x (x หน่วย เรเดียน)	tan(0.0) is 0.0

รูปที่ 6.2 แสดงฟังก์ชันทางคณิตศาสตร์ในภาษา C

6.2 การกำหนดฟังก์ชัน

การกำหนดฟังก์ชันในโปรแกรมประกอบด้วย 3 ส่วนหลักคือ

1. ต้นแบบของฟังก์ชัน(function prototype)
2. การกำหนดฟังก์ชัน(function prototype)
3. การเรียนใช้ฟังก์ชัน(function call)

โปรแกรมในภาษาซีทุกโปรแกรมประกอบด้วยฟังก์ชันหลักเรียกว่า main ซึ่งทำหน้าที่เรียกใช้ฟังก์ชันคลังมาตรฐานเพื่อทำงานต่าง ๆ รูปแบบของโปรแกรมที่ประกอบด้วยฟังก์ชัน main และฟังก์ชันอื่น ๆ มีดังนี้คือ

```
#include<stdio.h>
#define … /* constant definitions*/
#define … /* constant definitions*/
<global variable declarations>
main()
        /* main program body*/
<other function definitions>
```

ก. โครงร่างของฟังก์ชันในภาษา C

```
<header code>
int a;
float b;
int c;
char d;
<function definitions>;
```

ข. การกำหนดชนิดของตัวแปรครอบคลุมในภาษา C

```
<type><functionname>(parm1, parm2,…)
        <parm1 type>parm1; /*declare parameter*/
        <prm1 type>parm2;
        {/*begin*/
                <local function vars typed>;
                <function body>
                return(<expression>);
        } /*end*/
```

ค. การกำหนดฟังก์ชันในภาษา C

รูปที่ 6.3 โครงสร้างของภาษา C

6.2.1 ต้นแบบของฟังก์ชัน

ต้นแบบของฟังก์ชันเป็นส่วนที่กำหนดให้คอมไพเลอร์รู้ถึงคุณสมบัติต่าง ๆ ของฟังก์ชันได้แก่

1) ชนิดของข้อมูลที่ได้จากการทำงานของฟังก์ชัน
2) จำนวนของพารามิเตอร์ที่โปรแกรมส่งให้ฟังก์ชัน
3) ชนิดของพารามิเตอร์
4) ลำดับของพารามิเตอร์

ดังนั้นคอมไพเลอร์จะตรวจการเรียกใช้ฟังก์ชันจากต้นแบบของฟังก์ชัน

รูปแบบของต้นแบบของฟังก์ชัน

```
r_type f_name(arg_type arg_name, arg_type arg_name,…); หรือ
r_type f_name(arg_type,arg_type,arg_type,…);
```

r_type คือชนิดของข้อมูลผลลัพธ์ที่ได้จากการทำงานของฟังก์ชัน

f_name คือชื่อฟังก์ชัน

arg_type คือชนิดของอาร์กิวเมนต์

arg_name คือชื่อของอาร์กิวเมนต์ อาจใส่หรือไม่ใส่ก็ได้

ตัวอย่างของต้นแบบของฟังก์ชัน

```
int maximum (int a, int b, int c);
```

ต้นแบบของฟังก์ชันนี้กำหนดว่า ฟังก์ชัน maximum รับอาร์กิวเมนต์ 3 ตัว คือ a b และ c ที่มีชนิดข้อมูลเป็น int และคืนผลลัพธ์เป็นข้อมูลชนิด int

6.2.2 การกำหนดฟังก์ชัน

รูปแบบของการกำหนดฟังก์ชัน

```
r_type f_name (arg_type arg_name, arg_type arg_name,..)
{
…
function body
…
}
```

r_type คือชนิดของข้อมูลผลลัพธ์ที่ได้จากการทำงานของฟังก์ชัน ได้แก่ int , long int, float, double, long double, char และ void ถ้ากำหนดชนิดของฟังก์ชันเป็น void หมายถึงฟังก์ชันนั้นจะไม่มีผลลัพธ์คืนไปให้โปรแกรมหรือฟังก์ชันที่เรียกใช้

f_name คือชื่อของฟังก์ชัน การกำหนดชื่อของฟังก์ชันนี้มีหลักเกณฑ์เช่นเดียวกับการกำหนดตัวแปร void เป็นชนิดของฟังก์ชันที่ทำงานโดยไม่มีการส่งผลลัพธ์คืนไปให้โปรแกรมหรือฟังก์ชันที่เรียกใช้

arg_type คือชนิดของข้อมูลอากิวเมนท์

ในกรณีที่ฟังก์ชันไม่ได้รับพารามิเตอร์ซึ่งเป็นอากิวเมนท์ที่กำหนดไว้จากฟังก์ชันที่เรียกใช้ ให้กำหนดชนิดของอากิวเมนต์เป็น void

การสั่งให้ฟังก์ชันคืนค่าหลังการทำงานไปที่โปรแกรมใช้คำสั่ง return; ในกรณีที่ฟังก์ชันไม่คืนค่าให้โปรแกรม หรือใช้คำสั่ง return <expression>; อีกกรณีหนึ่งที่ฟังก์ชันคืนค่าให้โปรแกรมที่เรียกใช้เป็นนิพจน์

ตัวอย่างที่ 6.1 โปรแกรมคำนวณค่าสูงสุดของจำนวนเต็ม 3 จำนวน

```
/* Finding the maximum of three integers */
#include <stdio.h>
int maximum(int, int, int); /* function prototype */
main()
{
        int a, b, c;
        printf("Enter three integers: ");
        scanf("%d%d%d", &a, &b, &c);
        printf("Maximum is: %d\n", maximum(a, b, c));
        return 0;
} /* End main */
/* Function maximum definition */
int maximum(int x, int y, int z)
{
        int max = x;
        if (y > max)
                max = y;
        if (z > max)
                max = z;
        return max;
} /*int maximum */
```

ผลลัพธ์ที่ได้ จากรันครั้งที่หนึ่งโดยใส่ค่า 17 22 และ 85

```
Enter three integers: 17 22 85
Maximum is: 85
```

ผลลัพธ์ที่ได้ จากรันครั้งที่หนึ่งโดยใส่ค่า 85 22 และ 17

```
Enter three integers: 85 22 17
Maximum is: 85
```

ผลลัพธ์ที่ได้ จากรันครั้งที่หนึ่งโดยใส่ค่า 22 85 และ 17

```
Enter three integers: 22 85 17
Maximum is: 85
```

โปรแกรมนี้ประกอบด้วยฟังก์ชัน maximum ที่คำนวณค่าสูงสุดจากข้อมูลนำเข้า 3 จำนวน และคืนค่าสูงสุดให้ฟังก์ชัน main โดยใช้คำสั่ง return max

6.2.3 การเรียกใช้ฟังก์ชัน

การเรียกใช้ฟังก์ชัน(function call) ในภาษา C ทำโดยกำหนดมูลค่าให้กับฟังก์ชัน (call by value) ตัวอาร์-กิวเมนต์จะได้รับมูลค่าที่กำหนดให้ และส่งไปให้ฟังก์ชันทำงาน วิธีการนี้มูลค่าเดิมของตัวแปรในฟังก์ชัน main จะไม่มีการเปลี่ยนแปลง

ตัวอย่างที่ 6.2 การเรียกใช้ฟังก์ชันในโปรแกรมภาษา C

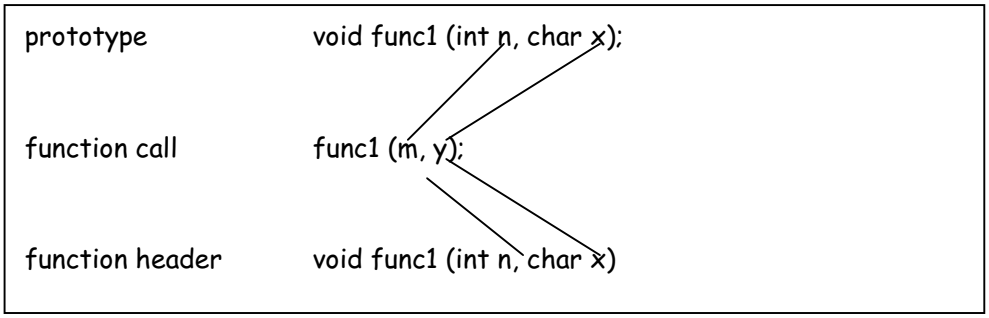
```
#include<stdio.h>
void functon2(int i, int j);
main()
{       int a=12, b=-2;
        function2 (a, b);
        printf ("a=%d b=%d\n", a, b);
}/* End main */

void function2 (int i, int j)
{
        printf("i=%d j=%d\n", i, j+1);
} /* End void function2 */
```

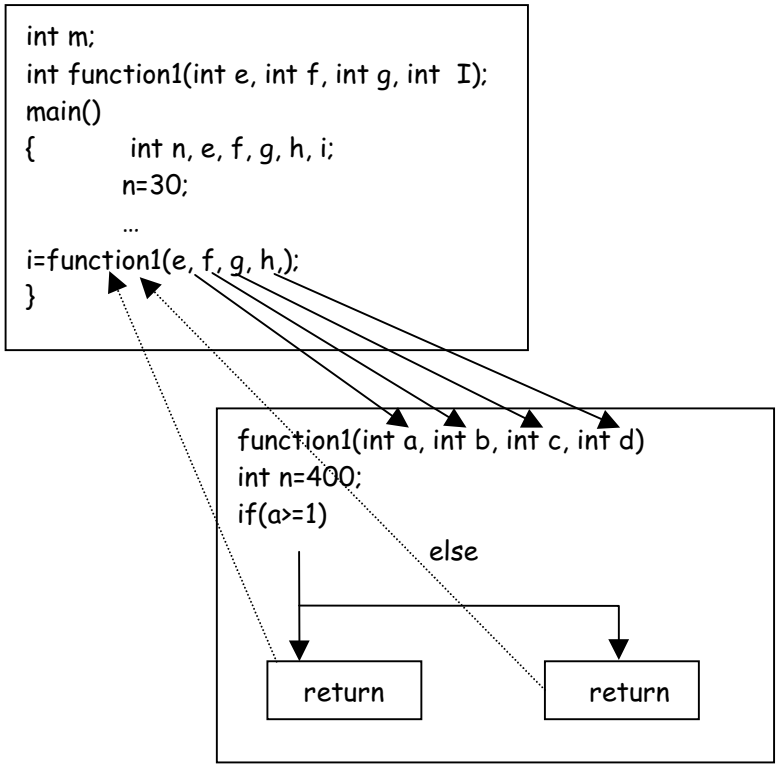
ผลลัพธ์ที่ได้

```
i=12 j=-1
a=12 b=-2
```

รูปแบบการกำหนดและการเรียกใช้ฟังก์ชัน

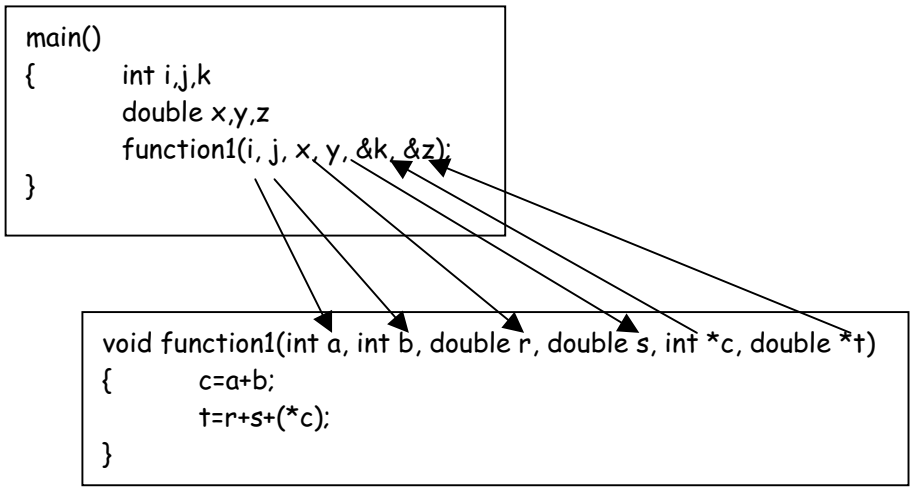


ตัวอย่างที่ 6.3 ฟังก์ชันที่มีการคืนค่า 1 ค่า



function1 รับอากิวเมนท์ 4 ตัวคือ e, f, g, h แล้วส่งค่าไปให้ function1 ทำงาน เพื่อคำนวณค่า a เมื่อคำนวณค่า a เรียบร้อยแล้ว จะคืนค่าให้ฟังก์ชัน main เพียง 1 ค่าคือ a ในกรณีที่ a >=1 อีกกรณีหนึ่ง คืนค่า c ถ้า a<1

ตัวอย่างที่ 6.4 ฟังก์ชันที่มีการคืนค่ามากกว่า 1 ค่า



โปรแกรมนี้ประกอบด้วย function1 ซึ่งมี อาร์กิวเมนต์ 6 ตัว คือ i, j, x, y, &k,&z โดยฟังก์ชัน main ส่งค่า 4 ค่าไปทาง i, j, x, y เพื่อไปยังตัวพารามิเตอร์ a, b, r, และ s และส่งเลขที่อยู่ทาง k กับ z ไปยังตัวชี้ c และ t เมื่อ function1 ทำงานเสร็จเรียบร้อยแล้วจึงคืนค่า 2 ค่ากลับไปที่ main โดยผ่านทางตัวชี้ *c และ *t

6.3 ฟังก์ชันเรียกซ้ำ

ฟังก์ชันเรียกซ้ำ (recursive function) คือฟังก์ชันที่เรียกใช้ตัวเองโดยทางตรงและทางอ้อม(โดยผ่านฟังก์ชันอื่น) ฟังก์ชันแบบเรียกซ้ำนี้มีการนำไปใช้แก้ปัญหาทางคอมพิวเตอร์อย่างแพร่หลาย คุณสมบัติของฟังก์ชันนี้คือ สามารถแก้ปัญหาในกรณีที่ง่ายที่สุด (simple case) เท่านั้น หรือเรียกกรณีนี้ว่า กรณีฐาน (base case) ถ้ามีการเรียกใช้ฟังก์ชันโดยกำหนดเงื่อนไขให้เป็นกรณีฐานฟังก์ชันนี้จะให้ผลลัพธ์และหยุดการทำงาน

สำหรับการแก้ปัญหาที่ซับซ้อน เราจะแบ่งปัญหาออกเป็น 2 ส่วน คือ ส่วนแรกเป็นปัญหาที่แก้ด้วยฟังก์ชัน ส่วนที่สองเป็นส่วนที่ฟังก์ชันไม่สามารถแก้ปัญหาได้ คือนำมาแบ่งในลักษณะเดิม จะทำให้ปัญหามีขนาดเล็กลงกว่าปัญหาเดิม การเรียกใช้ตัวเองเพื่อแก้ปัญหาที่มีขนาดเล็กลงนี้เรียกว่า การเรียกซ้ำ (recursion call) และอาจเรียกว่า ขั้นตอนเรียกซ้ำ (recursion step) ขั้นตอนการเรียกซ้ำนี้จะรวมถึงคำสำคัญ return ซึ่งเป็นคำสั่งที่ทำให้ฟังก์ชันส่งผลลัพธ์ที่ได้จากการดำเนินงานกลับไปที่โปรแกรมซึ่งเรียกใช้ฟังก์ชันนั้น

ขั้นตอนการเรียกซ้ำทำให้เกิดการเรียกซ้ำหลายครั้ง เนื่องจากฟังก์ชันมีการแบ่งปัญหาออกเป็น 2 ส่วนย่อย ในแต่ละครั้งที่ฟังก์ชันมีการเรียกใช้ตัวเองนั้นจะมีการลดความซับซ้อนหรือขนาดของปัญหาลงเรื่อย ๆ จนกระทั่งปัญหานี้มีขนาดลดลงจนถึงกรณีฐาน ณ จุดนี้ ฟังก์ชันจะมองเห็นกรณีฐานแล้วคืนค่าที่ได้จากการทำงานไปยังฟังก์ชันล่าสุดที่เรียกใช้ และจะมีการคืนค่าให้ฟังก์ชันที่เรียกใช้ถัดไปเป็นลำดับ จนกระทั่งถึงฟังก์ชันแรกสุดและส่งคืนไปยังฟังก์ชัน main

ตัวอย่างของฟังก์ชันเรียกซ้ำได้แก่ การคำนวณค่าแฟกทอเรียลของจำนวนเต็ม n

n! คือ n*(n-1)*(n-2)*…*1

ซึ่ง 1! มีค่าเป็น 1 และ 0! คือ 1

5! คือผลคูณของ 5*4*3*2*1 มีค่าเป็น 120

การคำนวณแฟกทอเรียลของจำนวนเต็ม number ซึ่งมีค่ามากกว่าหรือเท่ากับ 0 สามารถหาได้จาก การวนซ้ำ โดยใช้คำสั่ง for ดังนี้

```
factorial = 1;
for (counter = number; counter >= 1; counter --)
        factorial * = counter;
```

สามารถเขียนคำจำกัดความของแฟกทอเรียล ฟังก์ชันได้ตามความสัมพันธ์ดังนี้

n! = n*(n-1)!

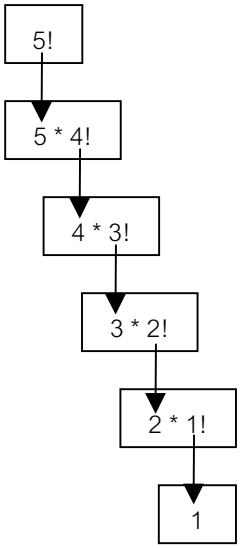
5! = 5*(4!)

5! = 5*4*3*2*1

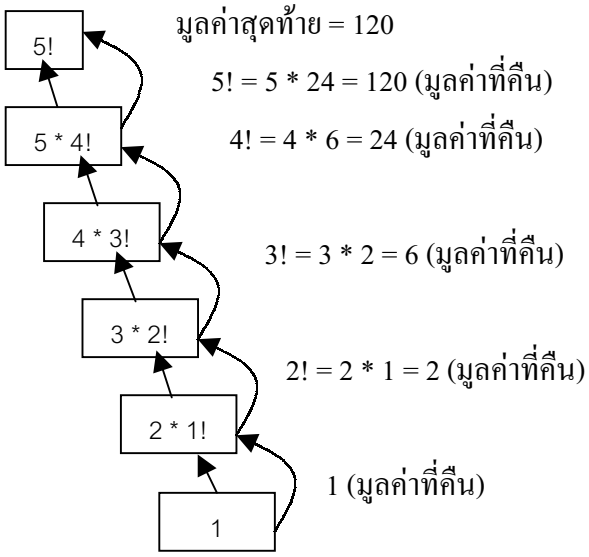
5! = 5*(4*3*2*1)

การประมวลผลของ 5! สามารถแสดงได้ดังรูปที่ 6.6 โดยรูปที่ 6.6 ก. แสดงการเรียกซ้ำกรณีที่ปัญหามีขนาดเล็กลงจนกระทั่งถึง 1! และมีการประมวลผล 1! ได้ค่า 1 ซึ่งเป็นการจบการเรียกซ้ำ รูป 6 ข. แสดง

มูลค่าที่ได้รับคืนจากการเรียกซ้ำแต่ละครั้งจนกระทั่งมูลค่าสุดท้ายได้รับการคำนวณและส่งคืนไปยังโปรแกรมที่เรียกใช้



รูป 6.6 ก. กระบวนการเรียกซ้ำ



รูป 6.6 ข. มูลค่าที่คืนไปยังฟังก์ชันที่เรียกใช้

ตัวอย่างที่ 6.5 การคำนวณหาแฟกทอเรียลโดยใช้ฟังก์ชันเรียกซ้ำ

```
/* Recursive factorial function */
# include <stdio.h>
long factorial(long);
main()
{
        int i;
        for (i =1; i <= 10; i++)
                printf("%2d! = %ld\n", i, factorial(i));
        return 0;
} /* End main */

/* Recursive definition of function factorial */
long factorial (long number)
{
        if (number <= 1)
                return 1;
        else
                return ( number * factorial(number -1));
} /* End long factorial */
```

ผลลัพธ์ที่ได้

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
```

ในตัวอย่างที่ 6.5 โปรแกรมนี้แสดงการคำนวณหาค่าแฟกทอเรียลโดยใช้การเรียกซ้ำ โดย n มีค่าตั้งแต่ 1 ถึง 100 ฟังก์ชันเรียกซ้ำนี้จะมีการทดสอบว่า ค่าเงื่อนไขจบการทำงานนั้นว่าจริงหรือไม่ กล่าวคือ ถ้า number มีค่าน้อยกว่าหรือเท่ากับ 1 ฟังก์ชัน factorial ให้ค่าเป็น 1 และไม่มีการเรียกซ้ำ แล้วโปรแกรมจะจบการทำงาน ถ้า number มีค่ามากกว่า 1 จะเรียกซ้ำโดยคำสั่ง

```
return (number * factorial (number -1);
```

คำสั่งนี้เป็นการคำนวณผลคูณของ number และฟังก์ชันเรียกซ้ำ factorial(number -1) จะเห็นได้ว่าฟังก์ชัน factorial(number -1) มีความซับซ้อนของการคำนวณน้อยกว่าฟังก์ชัน factorial(number) เพราะขนาดของอากิวเมนต์ การคำนวณ number ลดลง 1 ซึ่งความซับซ้อนของการคำนวณจะลดลงเรื่อยๆ จนกระทั่งถึง 1

แบบฝึกหัด

1. จากโปรแกรมที่กำหนดให้จงบอกว่าส่วนไหนของโปรแกรมคือ
   ก. ต้นแบบของฟังก์ชัน (function prototype)
   ข. การกำหนดฟังก์ชัน (function definition)
   ค. การเรียกฟังก์ชัน (function call)
   ง. การคืนค่าให้โปรแกรม

```
#include<stdio.h>
int cube(int y);
void main()
{
        int x;
        for (x=1; x<=10; x++)
        printf("%d\n", cube(x));
}/* End main */
int cube(int y);
{
        return y*y*y;
} /* End int cube */
```

2. จงเขียนฟังก์ชันที่สามารถทำงานต่อไปนี้
   ก. ฟังก์ชัน celcius ที่เปลี่ยนค่าอุณหภูมิหน่วยฟาเรนไฮต์เป็นองศาเซลเซียส
   ข. ฟังก์ชัน fahrenheit ที่เปลี่ยนค่าองศาฟาเรนไฮต์เป็นองศาเซลเซียส
   ค. เขียนโปรแกรมที่ประกอบด้วยฟังก์ชันในข้อ ก และ ข ซึ่งโปรแกรมนี้สามารถพิมพ์ตารางแสดงค่าขององศาฟาเรนไฮต์ที่มีค่าเท่ากับองศาเซลเซียส โดยมีอุณหภูมิจาก 0 ถึง 100 องศาเซลเซียส และค่าฟาเรนไฮต์มีค่าจาก 32 ถึง 212 องศา
3. จงเขียนฟังก์ชันที่รับค่าจำนวนเต็ม และคืนค่าเป็นจำนวนเต็มที่มีตัวเลขเรียงลำดับกลับกับข้อมูลนำเข้า เช่น กำหนดจำนวนเต็มเป็น 7631 ฟังก์ชันจะให้ค่า 1367
4. จงเขียนฟังก์ชัน QualityPoints ซึ่งรับค่าคะแนนเฉลี่ยของนักเรียนในการสอบ และมีการคืนค่าเป็นเกรดที่ได้รับดังต่อไปนี้

คะแนน	เกรดที่ได้รับ
90-100	4
80-89	3
70-79	2
60-69	1
ต่ำกว่า 60	0

5. ตัวหารร่วมมาก (ห.ร.ม) ของจำนวนเต็ม 2 จำนวน คือ จำนวนเต็มที่ใหญ่ที่สุดซึ่งสามารถหารจำนวนเต็ม 2 ตัวนั้นลงตัว จงเขียนฟังก์ชัน gcd เพื่อหา ห.ร.ม ค่าตัวหารของจำนวนเต็ม 2 จำนวนที่รับเข้า
6. จงพิจารณาว่าโปรแกรมต่อไปนี้หน้าที่อะไร

```
main()
{
int c;
if ((c=getchar ()) !=EOF)
        {
        main();
        printf("%c", c);
        }
        return 0;
}
```

7. จงเขียนโปรแกรมซึ่งสามารถนำไปใช้สอนการคูณสำหรับนักเรียนประถมศึกษา โดยโปรแกรมนี้จะเลือกสุ่มจำนวนขึ้นมา 2 จำนวน และแสดงจำนวนทั้ง 2 บนจอภาพแล้วให้นักเรียนตอบผลคูณของ 2 นั้น โดยการพิมพ์ผลลัพธ์บนแผงแป้นอักขระ ถ้านักเรียนตอบถูก โปรแกรมพิมพ์ “Very good!” ถ้านักเรียนตอบผิดโปรแกรมพิมพ์ “No Corret” โปรแกรมจะให้นักเรียนตอบได้ไม่เกิน 3 ครั้ง ถ้ายังตอบไม่ถูกก็จะเฉลยคำตอบ และให้หาผลคูณของจำนวนใหม่ต่อไป

สำหรับการสุ่มจำนวนนั้น กระทำได้โดยใช้ฟังก์ชัน rand ที่อยู่ในคลังฟังก์ชัน การเรียกใช้ฟังก์ชัน rand มีดังนี้

```
#include<stdio.h>
#include<stdlib.h>
main()
{
int i;
i=rand();
}
```