บทที่ 7 ฟังก์ชัน (Function)

วัตถุประสงค์

1) เข้าใจการเขียนโปรแกรมในรูปแบบฟังก์ชัน

2) สามารถเขียนโปรแกรมแบบฟังก์ชันได้

ฟังก์ชัน คือโปรแกรมย่อยที่ทำงานในโปรแกรมใหญ่อีกที โดยปกติแล้วในโปรแกรมหนึ่งจะมีการทำงานหลายๆ ส่วน การแยกการทำงานแต่ละส่วนออกเป็นฟังก์ชันนั้น ทำให้เราสามารถจัดการโปรแกรมได้โดยง่าย เราสามารถคอมไพล์และทดสอบฟังก์ชันโดยแยกจากกันได้ และเราสามารถใช้ฟังก์ชันที่เราเขียนขึ้นมานั้นซ้ำในโปรแกรมอื่นได้

เราสามารถแบ่งฟังก์ชันออกเป็นหลายชนิด โดยถ้าเราแบ่งโดยชนิดของการนิยามฟังก์ชันจะแบ่งได้ 2 แบบ คือ

1) ฟังก์ชันที่ถูกกำหนดไว้ก่อนในคลังโปรแกรม (Pre-defined function)
2) ฟังก์ชันที่ผู้เขียนกำหนดขึ้นมาเอง (User-defined function)

7.1 ฟังก์ชันที่ถูกกำหนดไว้ก่อนในคลังโปรแกรม (Pre-defined function)

เราสามารถใช้ฟังก์ชันที่ถูกกำหนดไว้ก่อนนี้จะอยู่ในคลังโปรแกรมหรือไลบรารีมาตรฐาน (standard C++ library) โดยการใส่ (include) ชื่อไลบรารีไว้บนหัวไฟล์ ตัวอย่างเช่น ถ้าเราจะใช้ฟังก์ชัน sqrt() เพื่อหารากที่สองของเลขจำนวนจริง ซึ่งถูกกำหนดไว้ในไลบรารี cmath เราต้องใส่ #include <cmath> ไว้ที่หัวไฟล์ด้วย และเช่นกัน ถ้าเราต้องการใช้ฟังก์ชัน rand() เพื่อสร้างเลขสุ่ม ซึ่งถูกกำหนดไว้ในไลบรารี cstdlib เราต้องใส่ #include <cstdlib> ไว้ที่หัวไฟล์ด้วย เพื่อให้โปรแกรมรู้ว่าฟังก์ชันดังกล่าวทำงานอย่างไร เราสามารถหาข้อมูลสำหรับฟังก์ชันอื่นๆ ในคลังข้อมูลได้ที่เว็บไซต์ http://www.cplusplus.com/reference/

ในวิชานี้ เราจะใช้ไลบรารีในตารางที่ โดยเราต้องพิมพ์คำสั่ง #include <library> ไว้ที่หัวไฟล์ โดยกำหนดให้ library คือชื่อไลบรารีที่มีฟังก์ชันที่เราต้องการอยู่

ตารางที่ 1 รายชื่อไลบรารีที่เราใช้ในวิชานี้

ชื่อไลบรารี	ฟังก์ชันที่อยู่ในไลบรารี
<cmath>	ฟังก์ชันทางคณิตศาสตร์
<cstring>	ฟังก์ชันที่เกี่ยวกับสายอักขระ (string)
<cstdlib>	ยูทิลิตี้ฟังก์ชัน
<ctime>	ฟังก์ชันที่เกี่ยวกับเวลา เช่นเรียกดูเวลาปัจจุบัน
<climits>	ฟังก์ชันเกี่ยวกับขอบเขตของจำนวนเต็ม เช่นจำนวนต่ำสุด สูงสุดของจำนวนเต็ม

การทำงานของฟังก์ชันที่ถูกกำหนดไว้ก่อนในคลังโปรแกรม

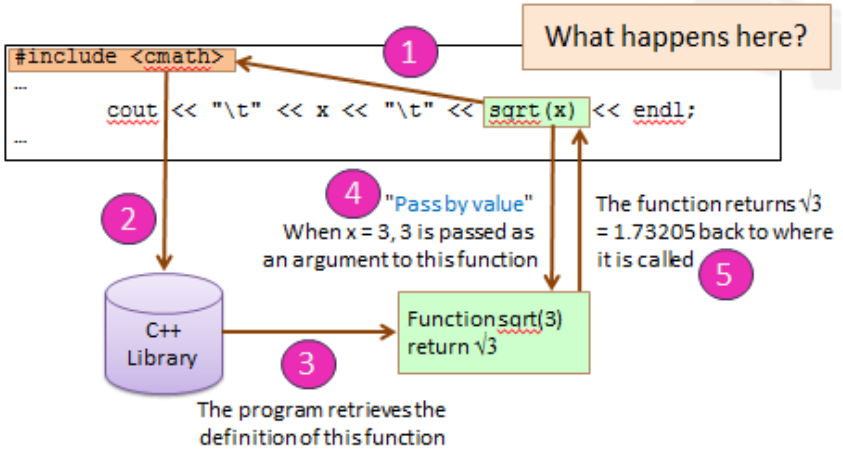
เราจะยกตัวอย่างการทำงานและการใช้ฟังก์ชันที่ถูกกำหนดไว้ก่อนในคลังโปรแกรม (Pre-defined function) โดยใช้ตัวอย่างฟังก์ชันทางคณิตศาสตร์ ได้แก่ sqrt(), sin() และ cos() และฟังก์ชันที่เกี่ยวกับการสุ่มเลข ได้แก่ rand(), srand() และ time()

โค้ดข้างล่างนี้แสดงตัวอย่างการเรียกใช้ฟังก์ชัน sqrt() เวลาเรียกใช้ฟังก์ชันเราต้องทำดังนี้

1) เขียน #include <cmath> ที่หัวไฟล์ เป็นการอ้างอิงชื่อไลบรารีที่ฟังก์ชันนั้นถูกเก็บอยู่ เพื่อโปรแกรมจะได้ทราบว่าฟังก์ชันนั้นทำงานอย่างไร ในกรณีนี้ ไลบรารีมีชื่อว่า cmath
2) เราเรียกใช้ฟังก์ชันโดยการเรียกชื่อฟังก์ชันนั้นเลย ในกรณีนี้ ชื่อฟังก์ชันคือ sqrt
3) ถ้าฟังก์ชันนั้นมีการใส่พารามิเตอร์หรือค่าที่จะส่งเข้าไปทำงานในฟังก์ชัน เราต้องใส่ค่าหรือตัวแปรให้ถูกชนิดด้วย ในกรณีนี้เราต้องใส่ค่าพารามิเตอร์ที่ถูกส่งไปให้ฟังก์ชัน sqrt() ด้วยการใส่ x เข้าไป (ค่าของ x จะถูกส่งไปทำงานในฟังก์ชัน sqrt())

```
1  #include <cmath>
2  int main(){
3        for(int x = 0; x < 6; x++)
4              cout << "\t" << x << "\t" << sqrt(x) << endl;
5        return 0;
6  }
```

โดยการทำงานของฟังก์ชันเมื่อฟังก์ชันถูกเรียกที่บรรทัดที่ 4 นั้น เป็นไปตามขั้นตอนต่อไปนี้ ตามรูปที่



รูปที่ 1 ขั้นตอนการทำงานของฟังก์ชัน sqrt()

1) เมื่อฟังก์ชันถูกเรียกและใส่พารามิเตอร์เข้ามา โปรแกรมจะไปดูรายชื่อไลบรารีที่เราใส่เข้ามาที่หัวไฟล์
2) จากนั้นโปรแกรมจะดูว่าในไลบรารีนั้นมีฟังก์ชันนั้นอยู่จริงๆ หรือไม่
3) ถ้ามีโปรแกรมจะไปดูนิยามของฟังก์ชันในไลบรารีดังกล่าว นั่นคือไปดูว่าฟังก์ชันนั้นทำงานอย่างไร
4) จากนั้นโปรแกรมจะส่งค่าพารามิเตอร์ x เข้าไปในฟังก์ชัน
5) เมื่อฟังก์ชันรับค่าเข้าไปแล้ว ก็จะทำงานตามนิยามที่ได้กำหนดไว้ และส่งคืนค่ากลับไปยังจุดที่เรียกฟังก์ชันนั้นถ้ามีการส่งค่าคืน ในกรณีของฟังก์ชัน sqrt() นี้จะมีการส่งค่าคืน ซึ่งคือค่ารากที่สองของเลขจำนวนจริง

จากโค้ดข้างต้น เราจะเห็นได้ว่ามีการเรียกฟังก์ชัน sqrt() จากฟังก์ชันเมนเป็นจำนวน 6 ครั้ง โดยแต่ละครั้งนั้นโปรแกรมได้ส่งค่า x เท่ากับ 0, 1, 2, 3, 4 และ 5 เข้าไปตามลำดับ โดยแต่ละครั้งที่ทำการเรียกฟังก์ชันโปรแกรมก็จะทำตามขั้นตอนที่ 1) - 5) และได้ผลการทำงานดังนี้

ผลการทำงาน:

```
0       0
1       1
2       1.41421
3       1.73205
4       2
5       2.23607
```

ตัวอย่างที่ 1 ตรีโกณมิติ

โจทย์ : จงหาผลรันของโปรแกรมต่อไปนี้

โค้ด:

```
1   #include <cmath>
2   int main(){
3         for(float x = 0; x < 5; x += 2){
4            cout << x << "\t\t"
5                  << sin(2*x) << "\t"
6                  << 2 * sin(x) * cos(x)
7                  << endl;
8         }
9         return 0;
10  }
```

โปรแกรมนี้เป็นโปรแกรมที่แสดงค่าเปรียบเทียบระหว่าง sin(2x) และ 2*sin(x)*cos(x) โดยลูปจะทำงาน 3 รอบเมื่อ x เป็น 0, 2, และ 4 ตามลำดับ โดยในแต่ละรอบโปรแกรมจะส่งค่า 2*x (คือค่า 0, 4, และ 8 ในแต่ละรอบตามลำดับ) เข้าไปในฟังก์ชัน sin() ในบรรทัดที่ 5 และค่า x เข้าไปในฟังก์ชัน sin() และฟังก์ชัน cos() ในบรรทัดที่ 6 จากนั้นจะแสดงค่าออกมาที่หน้าจอ โดยเราจะได้ผลรันเป็น

ผลการทำงาน:

```
0           0     0
2           -0.756802   -0.756802
4           0.989358    0.989358
```

ตัวอย่างที่ 2 การใช้ฟังก์ชันสุ่มเลข (Pseudo-random number functions)

โดยปกติแล้ว การสุ่มเลขในคอมพิวเตอร์จะเป็นการสุ่มเลขเทียม ไม่ใช่การสุ่มอย่างแท้จริง โดยจะมีการกำหนดชุดของเลขสุ่มไว้ก่อนหลายชุด โดยแต่ละชุดที่กำหนดไว้จะมีคุณสมบัติคล้ายเลขสุ่มจริง จากนั้นเมื่อมีการเรียกฟังก์ชัน rand() ในไลบรารี cstdlib ฟังก์ชัน rand() ก็จะส่งค่าเลขในชุดหนึ่งออกมาเรื่อยๆ ตามลำดับที่กำหนดไว้ก่อน เราสามารถใช้ฟังก์ชัน rand() ในการสุ่มเลขได้ตามตัวอย่างโค้ดข้างล่างนี้ โดยโปรแกรมนี้จะสุ่มเลขออกมา 8 ตัว ก่อนที่จะแสดงค่า RAND_MAX ออกมา ซึ่งเป็นค่าของเลขสุ่มที่มากที่สุดที่สามารถส่งออกมาจากฟังก์ชัน rand() ได้

โค้ด:

```
#include <cstdlib>
int main(){
      for(int i = 0; i<8; i++)
            cout << rand() << endl;
      cout << "RAND_MAX = " << RAND_MAX << endl;
      return 0;
}
```

ผลการทำงาน (บนเครื่องของอาจารย์):

```
41
18467
6334
26500
19169
15724
11478
29358
RAND_MAX = 32767
```

ถ้าเราลองรันโปรแกรมนี้ดู เราจะพบว่าในการรันแต่ละครั้ง เราจะได้เลขชุดเดิมออกมาเสมอ (แต่ผลการรันของแต่ละเครื่องอาจจะไม่เหมือนกัน) ในคอมพิวเตอร์จะมีการกำหนดชุดของเลขสุ่มแบบนี้ไว้หลายชุด ถ้าเราต้องการเรียกเลขสุ่มให้ออกมาเป็นชุดที่ต่างกันในแต่ละครั้งที่เรารันโปรแกรมเราต้องใช้ seed ที่ต่างกัน ซึ่ง seed เป็นเหมือนเบอร์ของชุดเลขสุ่มนั้น เช่นถ้าเราใช้ seed เท่ากับ 5 และ 100 ก็จะได้ชุดของเลขสุ่มที่ต่างกัน แต่การเรียกที่ใช้ seed เดิมจะได้เลขสุ่มชุดเดิมเสมอ ในการกำหนด seed นั้นเราใช้คำสั่ง srand(seed) เพียงครั้งเดียวในโปรแกรมนั้นก่อนที่จะใช้คำสั่ง rand() โดย seed เป็นเลขชนิด unsigned

โค้ด:

```
#include <cstdlib>
int main(){
      unsigned seed;
      cout << "Enter seed: ";
      cin >> seed;
```

```
      srand(seed);
      for(int i = 0; i<8; i++)
            cout << rand() << endl;
      return 0;
}
```

ผลการทำงาน โดยใช้ seed เป็น 5 (บนเครื่องของอาจารย์):

```
Enter seed: 5
54
28693
12255
24449
27660
31430
23927
17649
```

ผลการทำงาน โดยใช้ seed เป็น 100 (บนเครื่องของอาจารย์):

```
Enter seed: 100
365
1216
5415
16704
24504
11254
24698
1702
```

ในตัวอย่างข้างต้น เรารับ seed มาจากผู้ใช้ คือให้ผู้ใช้กำหนดค่า seed แต่ถ้าเราต้องการให้โปรแกรมกำหนด seed โดยอัตโนมัติและไม่ซ้ำกันในแต่ละครั้งที่รัน เราจะใช้ค่าเวลาปัจจุบันเป็น seed แทน เนื่องจากค่าเวลาปัจจุบันจะเปลี่ยนไปเรื่อยๆ และจะไม่ซ้ำกัน เราจึงจะมีความเป็นไปได้สูงที่จะได้ชุดเลขสุ่มที่จะไม่ซ้ำกันนั่นเอง เราสามารถหาค่าเวลาปัจจุบันของระบบได้จากฟังก์ชัน time() ซึ่งอยู่ในไลบรารี ctime โค้ดข้างล่างนี้แสดงตัวอย่างการใช้เวลาปัจจุบันของระบบเป็น seed ในการสุ่มเลข ซึ่งมีการให้ค่า seed โดยใช้เวลาปัจจุบันในบรรทัดที่ 4

โค้ด:

```
1   #include <cstdlib>
2   #include <ctime>
3   int main(){
4         unsigned seed = time(NULL);
5         cout << "seed = " << seed << endl;
6         srand(seed);
7         for(int i = 0; i<8; i++)
8               cout << rand() << endl;
9         return 0;
10  }
```

ผลการทำงาน (บนเครื่องของอาจารย์):

```
seed = 1444378265
7467
27844
11111
21352
11036
650
14229
8611
```

จากตัวอย่างที่ผ่านมา เราจะเห็นได้ว่าในการสุ่มเลข เราจะได้เลขสุ่มที่เป็นจำนวนเต็มบวกที่อยู่ในช่วงระหว่าง 0 และ RAND_MAX (http://www.cplusplus.com/reference/cstdlib/rand/) แต่ถ้าเราต้องการจำกัดช่วงของเลขสุ่ม (range) เป็นช่วงอื่น วิธีหนึ่งที่เราสามารถทำได้คือการสุ่มเลขโดยใช้วิธีที่เราใช้มา จากนั้นใช้การหาเศษ (%) มาจำกัดช่วงที่เราต้องการแทน อย่างเช่นในตัวอย่างต่อไปนี้ สูตรที่เราใช้คือ

range = max – min + 1

r = rand() / 100 % range + min

โดย max คือเลขสูงสุดในช่วงที่เราต้องการ min คือเลขต่ำสุดในช่วงที่เราต้องการ range คือจำนวนตัวเลขในช่วงที่เราต้องการ เราสามารถสุ่มเลขในช่วงที่เราต้องการได้โดยเอาเลขที่เราสุ่มได้มาหาร 100 (ตัดเลขสองหลักสุดท้ายของค่าที่สุ่มได้ออกไป) แล้วเอาค่าที่ได้มา mod (%) กับ range (นั่นคือการหาเศษถ้าหารด้วย range

เราก็จะได้เลขที่อยู่ระหว่าง 0 ถึง range -1) จากนั้นจึงบวก min ซึ่งทำให้ค่าที่เราได้นั้นอยู่ระหว่าง min และ max ตามที่เราต้องการ

โค้ด:

```
#include <cstdlib>
#include <ctime>
int main(){
	unsigned seed = time(NULL);
	cout << "seed = " << seed << endl;
	srand(seed);
	int min, max;
	cout << "Enter min and max: ";
	cin >> min >> max;
	int range = max - min + 1;
	for(int i = 0; i < 20; i++){
		int r = rand() / 100 % range + min;
		cout << r << endl;
	}
	cout << endl;
	return 0;
}
```

ตัวอย่างเช่น ถ้าเราต้องการสุ่มเลขให้อยู่ในช่วง 1-100 ค่า max จะเป็น 100 ค่า min จะเป็น 1 และ range จะเป็น 100 -1 + 1 = 100 ตัว สมมติว่าเราสุ่มเลขได้ 123456 พอเราหาร 100 เราก็จะได้ 1234 จากนั้นเราก็เอา 1234 % 100 เราก็จะได้ 34 เป็นเลขสุ่มระหว่าง 1-100 ที่เราต้องการ

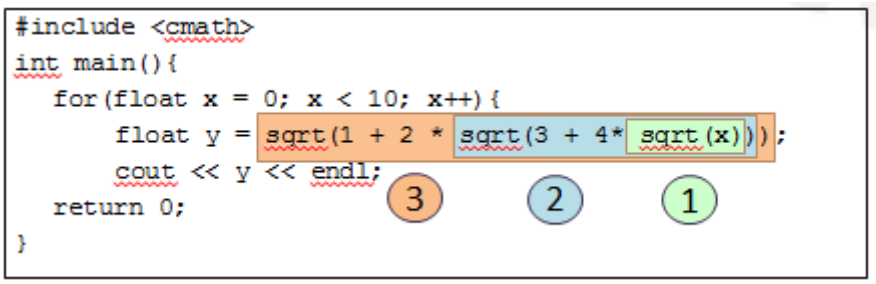
ผลการทำงาน (บนเครื่องของอาจารย์):

```
seed = 1444378965
Enter min and max: 1 100
98
51
15
30
23
97
3
40
32
```

```
88
14
22
45
21
100
74
92
75
28
91
```

การเรียกฟังก์ชันซ้อนกัน (nested function call)

เราสามารถเขียนโปรแกรมเพื่อเรียกฟังก์ชันซ้อนกันได้ นั่นคือการให้ผลของฟังก์ชันหนึ่งเป็นพารามิเตอร์หรือส่วนหนึ่งของพารามิเตอร์ให้กับอีกฟังก์ชันหนึ่งได้ในกรณีที่มีฟังก์ชันซ้อนกันอยู่ โปรแกรมจะเรียกทำฟังก์ชันที่ซ้อนอยู่ข้างในสุดก่อน เมื่อได้ผลลัพธ์แล้วก็จะใช้ผลลัพธ์นั้นเพื่อเรียกฟังก์ชันชั้นถัดมา ดังแสดงในรูปที่

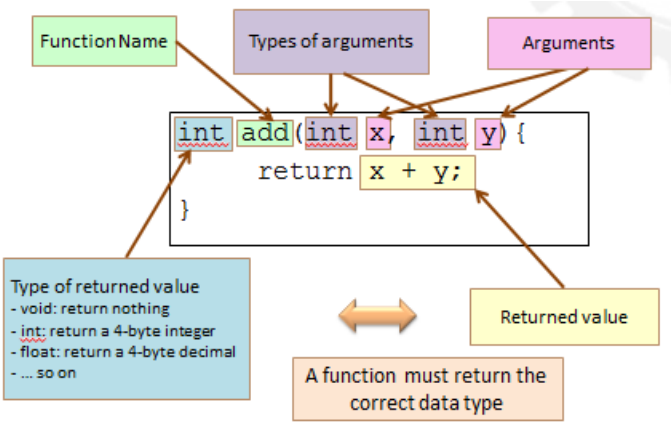


รูปที่ 2 ตัวอย่างการเรียกฟังก์ชันแบบซ้อนกัน

ในรูปที่ โปรแกรมมีการเรียกฟังก์ชัน sqrt() แบบซ้อนกัน 3 ชั้น นั่นคือผลลัพธ์ของ sqrt(x) เป็นส่วนหนึ่งของพารามิเตอร์ของการเรียก sqrt(3 + 4*sqrt(x)) ซึ่งมีผลลัพธ์เป็นส่วนหนึ่งของพารามิเตอร์ในการเรียก sqrt(1 + 2* sqrt(3 + 4*sqrt(x))) ตัวที่จะถูกเรียกทำงานอันดับแรกจะเป็น sqrt(x) เพราะอยู่ในสุด อีกนัยหนึ่งคือเป็นตัวที่ไม่ต้องรอผลการทำงานของตัวอื่น สามารถถูกคำนวณได้เลย เมื่อโปรแกรมคืนค่า sqrt(x) กลับมาแล้ว สมมติว่าผลลัพธ์คือ a โปรแกรมก็จะเรียกทำฟังก์ชันในชั้นต่อไปคือ sqrt(3 + 4*a) แล้วเมื่อโปรแกรมคืนค่ากลับมาแล้วสมมติว่าคือ b โปรแกรมก็จะเรียกทำฟังก์ชันในชั้นสุดท้าย คือ sqrt(1 + 2*b) พอได้ผลลัพธ์มาก็จะนำค่าไปไว้ในตัวแปร y และแสดงผลออกทางหน้าจอในบรรทัดถัดมา

7.2 ฟังก์ชันที่ผู้ใช้นิยามขึ้นมาเอง (User-Defined Function)

แม้ว่าเรามีฟังก์ชันในไลบรารีต่างๆ เป็นจำนวนมาก แต่มันก็ไม่ได้ตอบโจทย์สิ่งที่เราต้องการจะทำทั้งหมด ดังนั้นจึงมีการกำหนดให้เราสามารถเขียนฟังก์ชันขึ้นมาเองได้ เรียกว่า ฟังก์ชันที่ผู้ใช้นิยามขึ้นมาเอง (user-defined function) เช่น ถ้าเราต้องการเขียนฟังก์ชันที่บวกเลขจำนวนเต็ม 2 ตัว เราก็สามารถจะเขียนขึ้นมาได้เอง โดยไม่ต้องพึ่งไลบรารีใดๆ ก่อนที่เราจะเขียนฟังก์ชันได้ เราต้องรู้จักส่วนประกอบต่างๆ ของฟังก์ชันก่อน



รูปที่ 3 ส่วนประกอบของการนิยามฟังก์ชัน

ส่วนประกอบของการนิยามฟังก์ชัน

ฟังก์ชันที่เราจะต้องนิยามประกอบด้วยส่วนต่างๆ ดังรูปที่ ดังนี้

1) ชื่อฟังก์ชัน (function name) เอาไว้ใช้เรียกฟังก์ชัน
2) ชนิดของฟังก์ชัน (function type) คือชนิดของข้อมูลที่ฟังก์ชันนั้นคืนค่าหรือส่งค่ากลับไป (return) ให้ตัวที่เรียกมันมา เช่นถ้าฟังก์ชันมีชนิดเป็น int ฟังก์ชันนี้ก็จะต้องส่งค่าที่เป็น int กลับไปให้ตัวที่เรียกมันมา ถ้าฟังก์ชันไม่มีการส่งค่ากลับไป เราเรียกชนิดของฟังก์ชันนั้นว่า void
3) พารามิเตอร์ (parameter หรือ argument) เป็นตัวแปรหรือค่าที่ส่งผ่านเข้ามายังฟังก์ชัน ในหนึ่งฟังก์ชันสามารถมีพารามิเตอร์กี่ตัวก็ได้ หรือไม่มีก็ได้
4) ชนิดของพารามิเตอร์ (types of parameter/argument) คือชนิดของข้อมูลที่ถูกส่งผ่านเข้ามาใช้ในฟังก์ชัน
5) ค่าส่งกลับ (returned value) คือข้อมูลที่ถูกส่งกลับไปยังตำแหน่งที่เรียกฟังก์ชันมา ชนิดข้อมูลของค่าส่งกลับจะต้องเป็นชนิดเดียวกันกับชนิดของฟังก์ชัน

การนิยามฟังก์ชัน

การนิยามฟังก์ชันคือ การเขียนชุดคำสั่งว่าฟังก์ชันนั้นทำงานอย่างไร ซึ่งการนิยามฟังก์ชันจะต้องมีส่วนประกอบของฟังก์ชันให้ครบตามรูปที่ เราต้องนิยามฟังก์ชันก่อนฟังก์ชันนั้นถูกเรียกใช้ ไม่เช่นนั้นโปรแกรมจะคอมไพล์ไม่ผ่าน เพราะ ณ จุดที่ฟังก์ชันหนึ่งถูกเรียกโปรแกรมจะไม่รู้จักฟังก์ชันที่เราเรียกใช้ เปรียบได้กับว่า ถ้าเราไม่ลงทะเบียนเรียนไว้ก่อนแล้วเราไปเข้าเรียนและแนะนำตัวกับอาจารย์ อาจารย์ก็จะไม่รู้จักเราเพราะอาจารย์ไม่มีข้อมูลของเรามาก่อน ดังนั้นถ้าฟังก์ชันที่เราจะใช้จะถูกเรียกในฟังก์ชันเมน เราจะต้องใส่นิยามของฟังก์ชันนั้นก่อนฟังก์ชันเมน ตามตัวอย่างในโค้ดข้างล่างนี้

โค้ด:

```
int add(int x, int y){
            return x + y;
}

int main(){
      int a = 1, b = 1;
      while(a != 0 && b != 0){
            cin >> a >> b;
            cout << "\t" << a << " + " << b << " = "
                  << add(a,b) << endl;
      }
      return 0;
}
```

โปรแกรมนี้ได้นิยามฟังก์ชัน add() ซึ่งเป็นชนิด int และมีพารามิเตอร์ที่เป็นชนิด int สองตัว เราจะเห็นได้ว่าเราต้องเขียนนิยามของฟังก์ชัน add() ก่อนที่จะเขียนฟังก์ชันเมน ในฟังก์ชันเมน โปรแกรมจะวนรับค่า a และ b แล้วหาผลรวมโดยใช้ฟังก์ชัน add() จนกว่า a หรือ b จะเป็น 0 ดังตัวอย่างผลรันต่อไปนี้ เมื่อผู้ใช้ใส่ค่า 1 และ 2, 4 และ 5, 345 และ 666, และ 0 และ 5 เข้ามาตามลำดับ

ผลการทำงาน:

```
1 2
        1 + 2 = 3
4 5
        4 + 5 = 9
```

```
345 666
        345 + 666 = 1011
0 5
        0 + 5 = 5
```

การเรียกใช้ฟังก์ชัน

ดังที่ได้อธิบายไปแล้ว ในการเรียกใช้ฟังก์ชัน เราจะเขียนเฉพาะชื่อฟังก์ชันเท่านั้น และถ้าฟังก์ชันมีการรับพารามิเตอร์ เราต้องใส่พารามิเตอร์เข้าไปด้วย โดยพารามิเตอร์ที่เราใส่เข้าไปสามารถเป็นค่าคงที่ ตัวแปร หรือนิพจน์ก็ได้แล้วแต่กรณี (อ่านเพิ่มเติมในเรื่องการส่งผ่านพารามิเตอร์ไปยังฟังก์ชันโดยการใช้ค่าและโดยการอ้างอิง) ถ้าฟังก์ชันมีพารามิเตอร์มากกว่า 1 ตัว เราจะต้องใส่พารามิเตอร์ให้ถูกต้องตามลำดับและตามชนิดในลำดับที่กำหนดไว้ในนิยามของฟังก์ชัน เช่น ถ้าเรานิยามฟังก์ชันและเรียกใช้ฟังก์ชันตามโค้ดต่อไปนี้

โค้ด:

```
1   int func(int a, float b, char c)
2   {
3         cout << "my int = " << a << endl;
4         cout << "my float = " << b << endl;
5         cout << "my char = " << c << endl;
6         return a+2;
7   }
8
9   int main()
10  {
11        int x = 3;
12        float y = 9.2;
13        char z = '$';
14        cout << "func = " << func(x, y, z) << endl;
15
16        int m = func(2*x, y-1, z);
17        cout << "m = " << m << endl;
18        return 0;
19  }
```

ฟังก์ชัน func() มี 3 พารามิเตอร์ เป็นชนิด int, float และ char ตามลำดับ เวลาเราเรียกฟังก์ชัน func เราจะต้องใส่ค่า 3 ค่าตามลำดับดังนี้ ค่าแรกเป็นค่า นิพจน์ หรือตัวแปรที่เป็นชนิด int ส่วนค่าที่สอง เป็นค่า นิพจน์ หรือตัวแปรที่เป็นชนิด float และค่าที่สามเป็นค่า นิพจน์ หรือตัวแปรที่เป็นชนิด char ตามตัวอย่างการเรียกในบรรทัดที่ 14 และ 16 ทั้งนี้เพราะเมื่อเราเรียกฟังก์ชันและใส่ค่าที่จะผ่านมายังฟังก์ชันแล้ว ฟังก์ชันจะรับค่าเหล่านั้นมาจับคู่กับพารามิเตอร์ที่ประกาศไว้ในนิยามตามลำดับ เช่น ในบรรทัดที่ 14 เราเรียกฟังก์ชัน func() และใส่ค่า x, y และ z เข้าไปตามลำดับ โปรแกรมจะนำสำเนาค่าของ x ไปใส่ในตัวแปร a ในฟังก์ชัน func() ค่าของ y ไปใส่ในตัวแปร b และ ค่าของ z ไปใส่ในตัวแปร c ในกรณีที่เราใส่ค่าไปผิดชนิด เช่น แทนที่จะใส่ค่าที่เป็น int เราไปใส่ค่าที่เป็น float โปรแกรมจะพยายามแปลงค่าที่ใส่เข้ามาให้เป็น int ถ้าเป็นไปได้ ไม่เช่นนั้น โปรแกรมจะไม่สามารถคอมไพล์ให้ผ่านได้

เราสังเกตว่าเราสามารถเรียกใช้ฟังก์ชันได้ในหลายลักษณะ ถ้าเป็นฟังก์ชันที่มีการคืนค่า เราสามารถเรียกฟังก์ชันนั้นในคำสั่ง cout ได้เลย สิ่งที่เกิดขึ้นคือ โปรแกรมจะทำการแสดงผลค่าที่คืนมาจากฟังก์ชันนั้นทางหน้าจอ ดังตัวอย่างการเรียกในบรรทัดที่ 14 นอกจากนี้ เราสามารถเรียกฟังก์ชันเพื่อให้ค่ากับตัวแปรได้ ดังตัวอย่างการเรียกในบรรทัดที่ 16 แต่ตัวแปรที่เราจะเอาไปใส่ค่าที่คืนมาจากฟังก์ชันจะต้องเป็นชนิดเดียวกับฟังก์ชัน เช่น ถ้าฟังก์ชันคืนค่าเป็น int (ชนิดของฟังก์ชันเป็น int) เราจะต้องเอาตัวแปรชนิด int ไปรับค่าของฟังก์ชันนั้นด้วย ส่วนการเรียกฟังก์ชันที่ไม่มีการคืนค่า เราจะต้องเรียกแบบลอย นั่นคือ ไม่ต้องมีคำสั่ง cout หรือตัวแปรใดมารองรับ ให้เรียกลอยๆ เสมือนเป็นหนึ่งคำสั่งได้เลย เช่น

```
printDate();
```

ตัวอย่างที่ 3

โจทย์ : จงเขียนโปรแกรมเพื่อวนรับค่าจำนวนเต็ม 2 จำนวน จากนั้นให้เขียนฟังก์ชันเพื่อหาค่าสูงสุดในสองค่าที่รับเข้ามาในแต่ละรอบ ถ้าผู้ใช้ใส่จำนวนแรกเข้ามาเป็น 0 ให้หาค่าสูงสุดแล้วหยุดรับอินพุตในรอบถัดไปแล้วจบโปรแกรม

1) เอาต์พุต คือค่าสูงสุดของจำนวนเต็มสองจำนวนที่รับเข้ามา
2) อินพุต คือจำนวนเต็ม 2 จำนวน ให้วนรับค่าจนกว่าจำนวนแรกจะเป็น 0
3) โจทย์กำหนดให้ใช้ฟังก์ชันในการหาค่าสูงสุด
4) เราสามารถหาค่าสูงสุดได้โดยเทียบว่าค่าใดมีค่ามากกว่ากัน

ข้อนี้โจทย์กำหนดให้ใช้ฟังก์ชันในการหาค่าสูงสุด เพราะฉะนั้นเราจึงรู้ว่าฟังก์ชันนี้จะต้องรับพารามิเตอร์เป็นเลขจำนวนเต็ม 2 ตัว ทำการหาค่าสูงสุดในฟังก์ชันนั้น และส่งค่าจำนวนเต็มที่มากที่สุดกลับไป (returned value) เพราะฉะนั้นจึงมีชนิดของฟังก์ชันเป็นจำนวนเต็ม (int)

โค้ด:

```
int max(int x, int y ){
            if(x > y)  return x;
      return y;
}

int main(){
      int a, b;
      do{
            cin >> a >> b;
            cout << "\tmax(" << a << "," << b << ") = " << max(a,b)
            << endl;
      }while(a != 0);
      return 0;
}
```

ถึงตอนนี้เราเขียนนิยามฟังก์ชันไว้ก่อนฟังก์ชันเมน ซึ่งเป็นฟังก์ชันหลักของโปรแกรม เราสังเกตว่าถ้านิยามของฟังก์ชันเรายาวมาก หรือเรามีนิยามของฟังก์ชันเป็นจำนวนมาก แล้วเราต้องการเข้ามาอ่านหรือแก้โค้ดเราก็จะหาฟังก์ชันเมน ซึ่งเป็นจุดเริ่มต้นของโปรแกรมได้ยาก เราจึงมีวิธีนิยามฟังก์ชันอีกแบบหนึ่งคือ ก่อนฟังก์ชันเมน เราจะแค่ทำการประกาศฟังก์ชันที่เราจะนิยามขึ้นมา (declaring a function) ส่วนนิยามนั้นเราจะยกไปไว้หลังฟังก์ชันเมน เพื่อให้โค้ดของเรานั้นดูเรียบร้อยและอ่านง่าย

โดยสรุปแล้วหากเราต้องการใช้ฟังก์ชันที่เรานิยามขึ้นมาเอง เราสามารถเขียนโค้ดได้ 2 รูปแบบดังนี้

1. การเขียนนิยามไว้ก่อนฟังก์ชันเมน เราสามารถเขียนได้ 2 ขั้นตอน ดังนี้

    1.1 การนิยามฟังก์ชัน (defining a function) เราเขียนนิยามของฟังก์ชันไว้ก่อนฟังก์ชันเมน (หลัง using namespace std;)

    1.2 การเรียกฟังก์ชัน (calling a function) เมื่อเราต้องการใช้ฟังก์ชัน เราจะเรียกชื่อฟังก์ชัน และใส่พารามิเตอร์ซึ่งเป็นค่าหรือตัวแปรเข้าไปด้วย ตามตัวอย่างนี้ เมื่อ a และ b เป็นตัวแปรที่ถูกให้ค่าแล้ว

```
max(a, b)
```

2. การเขียนนิยามไว้หลังฟังก์ชันเมน จะมีขั้นตอนการเขียนและการใช้ 3 ขั้นตอนดังนี้

2.1 การประกาศฟังก์ชัน (function declaration) เราจะเขียนชนิดของฟังก์ชัน ชื่อฟังก์ชัน และชนิดของพารามิเตอร์ทุกตัวในวงเล็บ แล้วปิดด้วยเครื่องหมายเซมิโคลอน (;) โดยเราต้องเขียนไว้ก่อนฟังก์ชันเมน ตามตัวอย่างนี้

```
int max(int, int);
```

ทั้งนี้ เราสามารถประกาศฟังก์ชันโดยมีชื่อตัวแปรที่เป็นพารามิเตอร์ด้วยก็ได้ เช่น

```
int max(int a, int b);
```

2.2 การนิยามฟังก์ชัน (defining a function) เราเขียนนิยามของฟังก์ชันไว้หลังจากฟังก์ชันเมน

2.3 การเรียกฟังก์ชัน (calling a function) เมื่อเราต้องการใช้ฟังก์ชัน เราจะเรียกชื่อฟังก์ชัน และใส่พารามิเตอร์ซึ่งเป็นค่าหรือตัวแปรเข้าไปด้วย ตามตัวอย่างนี้ เมื่อ a และ b เป็นตัวแปรที่ถูกให้ค่าแล้ว

```
max(a, b)
```

ดังนั้น จากตัวอย่างที่ 3 เราสามารถเขียนโค้ดได้อีกแบบได้ดังนี้

โค้ด:

```
int max(int x, int y);

int main(){
      int a, b;
      do{
            cin >> a >> b;
            cout << "\tmax(" << a << "," << b << ") = " << max(a,b)
                  << endl;
      }while(a != 0);
      return 0;
}
```

```
int max(int x, int y){
      if(x < y) return y;
      return x;
}
```

ตัวอย่างที่ 4 การหาพื้นที่วงกลม

โจทย์ : จงเขียนโปรแกรมเพื่อรับค่ารัศมีของวงกลมในฟังก์ชันเมน จากนั้นให้เขียนฟังก์ชันเพื่อหาพื้นที่ของวงกลม แล้วส่งค่าคืนกลับมาแสดงผลที่ฟังก์ชันเมน

1) เอาต์พุต คือพื้นที่ของวงกลม
2) อินพุต คือรัศมีของวงกลม ซึ่งมีชนิดข้อมูลเป็นจำนวนจริง
3) โจทย์กำหนดให้ใช้ฟังก์ชันในการหาพื้นที่ โดยให้แสดงผลในฟังก์ชันเมน
4) พื้นที่วงกลมคือ πr^2

ข้อนี้โจทย์กำหนดให้ใช้ฟังก์ชันในการหาพื้นที่ และให้แสดงผลในฟังก์ชันเมน เราจึงต้องส่งค่ารัศมีที่อ่านได้จากฟังก์ชันเมนเข้าไปในอีกฟังก์ชันหนึ่ง แล้วให้ฟังก์ชันนั้นคืนค่าพื้นที่วงกลมที่คำนวณได้กลับมาแสดงผลที่ฟังก์ชันเมน

Code แบบที่ 1:

```
float fiArea(float r) {
    const float PI = 3.14159;
    float area;
    area = PI * r * r;
    return area;
}

int main() {
    float r;
    cout << "Enter radius: ";
    cin >> r;
    cout << "Area of a circle is " << fiArea(r) << endl;
    return 0;
}
```

Code แบบที่ 2:

```
float fiArea(float);

int main() {
      float r;
      cout << "Enter radius: ";
      cin >> r;
      cout << "Area of a circle is " << fiArea(r) << endl;
      return 0;
}

float fiArea(float r) {
      const float PI = 3.14159;
      float area;
      area = PI * r * r;
      return area;
}
```

โค้ดแบบที่ 1 เป็นการเขียนโค้ดแบบนิยามฟังก์ชันก่อนฟังก์ชันเมน ส่วนโค้ดแบบที่ 2 เป็นการเขียนโค้ดแบบประกาศฟังก์ชันก่อนฟังก์ชันเมน แล้วจึงนิยามฟังก์ชันหลังฟังก์ชันเมน ซึ่งเราสามารถเขียนได้ทั้งสองแบบและได้ผลลัพธ์เหมือนกัน

ฟังก์ชันลักษณะต่างๆ

ฟังก์ชันเมน (main() function)

โปรแกรมภาษา C++ จะเริ่มทำงานที่ฟังก์ชันเมนเป็นฟังก์ชันแรกเสมอ ฟังก์ชันเมนมีชนิดฟังก์ชันเป็น int ไม่มีพารามิเตอร์ และส่งคืนค่า 0 ให้ระบบเพื่อระบุว่าฟังก์ชันจบการทำงานแบบไม่มีปัญหา คอมไพเลอร์บางตัวอนุญาตให้เราเขียนโค้ดโดยไม่มีคำสั่ง return 0; ได้เพราะมันจะทำให้โดยปริยาย (by default) แต่คอมไพเลอร์บางตัวก็จะเตือนถ้าเราไม่ใส่ แต่ตามแนวปฏิบัติในการเขียนโปรแกรมที่ดีแล้ว เราควรจะใส่ return 0; ในฟังก์ชันเมนด้วย อย่างไรก็ตาม คำสั่ง return 0; ไม่จำเป็นจะต้องอยู่ท้ายของฟังก์ชันเมนเสมอไป เราสามารถจบฟังก์ชันเมนตามเงื่อนไขที่กำหนดให้ได้ เช่น ถ้าอินพุตที่เรารับเข้ามาไม่เป็นไปตามต้องการ เราสามารถเลือกจบฟังก์ชันทันที

ได้ ดังตัวอย่างในโค้ดต่อไปนี้ เราสามารถจบฟังก์ชันเมนได้เมื่อตรวจสอบว่า d เท่ากับ 0 (ตัวหารเป็นศูนย์ จึงไม่สามารถคำนวณผลการหารได้ จึงเลือกที่จะจบโปรแกรมไป ไม่คำนวณผลหาร)

โค้ด:

```
int main() {
      int n, d;
      cout << "Enter two integers: ";
      cin >> n >> d;
      if (d == 0) return 0;
      cout << n << "/" << d << " = " << n/d << endl;
  return 0;
}
```

นอกจากนี้เรายังสามารถจบโปรแกรม (จบฟังก์ชันเมน) ด้วยคำสั่งอื่นๆ เช่น exit(0) หรือ abort() ซึ่งเราจะไม่ค่อยใช้ในวิชานี้

ฟังก์ชันที่ไม่คืนค่า (void function)

เราเรียกฟังก์ชันที่ไม่มีการคืนค่าใดๆ (ไม่มี return) ว่าฟังก์ชันวอยด์ (void function) ส่วนมากเราจะใช้ฟังก์ชันที่ไม่คืนค่าในการทำส่วนงานประจำย่อยๆ (subroutine) ที่ถูกเรียกใช้บ่อยๆ เช่นการพิมพ์ค่าแอเรย์ และการพิมพ์ตัวเลือกให้ผู้ใช้ใส่ข้อมูล ฟังก์ชันที่ไม่คืนค่านี้อาจจะมีพารามิเตอร์หรือไม่มีก็ได้ เหมือนกับฟังก์ชันทั่วไป

ตัวอย่างที่ 5 ฟังก์ชันที่ไม่คืนค่า

โค้ด:

```
void printHeader(){
      cout << "Table of Content‟ << endl;
}
int main(){
      printHeader();
      return 0;
}
```

ฟังก์ชันบูลีน (boolean function)

ฟังก์ชันบูลีนเป็นฟังก์ชันที่คืนค่าเป็นจริง (true) หรือเท็จ (false) นิยมใช้ในการให้ค่าในเงื่อนไขของคำสั่ง if, while, และ do while เช่น if(func(x)) หรือ while(func(x)) ถ้า func(x) คืนค่ามาเป็นจริง ก็ให้ทำคำสั่งหลังเงื่อนไขหรือวนซ้ำอีกรอบ ถ้า func(X) คืนค่ามาเป็นเท็จ ก็ไม่ต้องทำคำสั่งหลังเงื่อนไขหรือไม่ต้องวนซ้ำอีกรอบ

ตัวอย่างที่ 6 ฟังก์ชันตรวจสอบเลขคู่

โจทย์ : จงหาผลรันของโปรแกรมต่อไปนี้ เมื่ออินพุตเป็น 5 และเมื่ออินพุตเป็น 10

โค้ด:

```
1   bool isEven(int x){
2         if(x%2 == 0) return true;
3         return false;
4   }
5
6   int main() {
7         int x;
8         cout << "Enter integer x: ";
9         cin >> x;
10        if (isEven(x))
11              cout << x << " is even.” << endl;
12        else
13              cout << x << " is odd.” << endl;
14        return 0;
15  }
```

โปรแกรมจะเริ่มทำงานที่ฟังก์ชันเมน ในกรณีแรกเมื่อโปรแกรมรับอินพุตเข้ามาเป็น 5 แล้ว โปรแกรมจะเรียกฟังก์ชัน isEven() โดยผ่านค่า 5 เข้าไป เพื่อตรวจสอบเงื่อนไขสำหรับคำสั่ง if ในฟังก์ชัน isEven() นั้น 5 จะถูกใส่ไว้ในตัวแปร x และจะส่งคืนค่าเท็จ (false) กลับมายังฟังก์ชันเมนเพราะ 5 ไม่ใช่เลขคู่จึงหารด้วยสองไม่ลงตัว ดังนั้นเงื่อนไขของคำสั่ง if ในบรรทัดที่ 10 จะมีค่าเป็นเท็จ ทำให้โปรแกรมแสดงผลว่า 5 is odd. ออกมา

ในกรณีที่อินพุตเท่ากับ 10 โปรแกรมก็ทำงานเช่นเดียวกัน แต่รับค่ามาเป็น 10 ฟังก์ชัน isEven() จะคืนค่าจริง (true) กลับมายังเมน ทำให้โปรแกรมแสดงผลว่า 10 is even. ออกทางหน้าจอ

ตัวอย่างที่ 7 ฟังก์ชันตรวจสอบปีอธิกสุรทิน (leap year)

โจทย์ : จงเขียนโปรแกรมเพื่อวนรับค่าปี ค.ศ. จากแป้นพิมพ์ตราบใดที่ปีที่รับเข้ามานั้นมากกว่าหรือเท่ากับ 1 โดยให้วนรับในฟังก์ชันเมน จากนั้นในแต่ละรอบให้เรียกฟังก์ชันเพื่อระบุว่าปีที่รับเข้ามานั้นเป็นปีอธิกสุรทินหรือไม่ และให้กลับมาแสดงผลในฟังก์ชันเมน

1) เอาต์พุต คือการระบุว่าปีที่ใส่เข้ามาเป็นปีอธิกสุรทินหรือไม่
2) อินพุต คือปี ค.ศ. โดยวนรับค่าจนกว่าผู้ใช้จะใส่ค่าที่น้อยกว่า 1 เข้ามา
3) โจทย์กำหนดให้ใช้ฟังก์ชันในการหาว่าปีที่ใส่เข้ามาเป็นปีอธิกสุรทินหรือไม่ โดยให้แสดงผลในฟังก์ชันเมน
4) ปีอธิกสุรทินคือปีที่มี 366 วัน เราทราบว่าปีอธิกสุรทินคือปีที่
    a. หารด้วย 4 ลงตัวและหารด้วย 100 ไม่ลงตัว <u>หรือ</u>
    b. หารด้วย 400 ลงตัว

ตัวอย่างนี้เป็นอีกตัวอย่างที่ใช้ฟังก์ชันบูลีน โดยฟังก์ชันบูลีนของเราจะตรวจสอบปี ค.ศ. ที่รับเข้ามาตามเงื่อนไขในข้อ 4) และเราสามารถเขียนโค้ดได้ดังนี้

โค้ด:

```
bool isLeapYear(int y){
      return y%4 == 0 && y%100 != 0 || y%400 == 0;
}

int main(){
      int n = 1;
      while(true){
            cin >> n;
            if(n < 1) break;
            if(isLeapYear(n))
                  cout << n << " is a leap year.\n";
            else
                  cout << n << " is not a leap year.\n";
      }
      return 0;
}
```

ฟังก์ชันรับอินพุตหรือแสดงผลเอาต์พุต (I/O function)

ดังที่ได้กล่าวไปแล้วว่า จุดประสงค์หนึ่งของการนำโค้ดมาเขียนแยกเป็นฟังก์ชันคือทำให้โปรแกรมสะอาด เรียบร้อยและอ่านง่าย ดังนั้นส่วนของโปรแกรมแบบหนึ่งที่นิยมถูกแยกออกไปเป็นฟังก์ชันคือการทำงานส่วนการรับเข้าข้อมูลและการแสดงผล เพราะหลายครั้งนอกจากจะรับข้อมูลเข้ามาตามปกติแล้ว เราต้องมีการตรวจสอบข้อมูลที่รับเข้ามาก่อนที่จะนำมาใช้ได้ด้วย หรือโปรแกรมมีการแสดงผลที่มีขั้นตอนยาว ทำให้โค้ดส่วนนี้จะยาว ดังนั้นการแยกส่วนของโปรแกรมเหล่านี้ออกไปเป็นฟังก์ชันจึงจะดีกว่า เราเรียกฟังก์ชันเหล่านี้ว่า ฟังก์ชันรับอินพุตหรือแสดงผลเอาต์พุต (Input/Output function หรือ I/O function) ส่วนของการแสดงผลนั้นอาจจะใช้ void function เหมือนที่กล่าวไปแล้ว แต่ส่วนของการรับเข้านั้นเราอาจจะต้องใช้ฟังก์ชันที่คืนค่าตามชนิดของอินพุตที่เราต้องการจะรับ หรือจะใช้ฟังก์ชันวอยด์ (void function) ก็ได้ถ้าเราจะคืนค่าโดยการผ่านพารามิเตอร์โดยการอ้างอิง

ตัวอย่างที่ 8 การอ่านและตรวจสอบค่าอายุจากแป้นพิมพ์

โจทย์ : จงเขียนโปรแกรมเพื่อรับค่าอายุจากแป้นพิมพ์และแสดงค่าออกทางหน้าจอในฟังก์ชันเมน โดยในการรับค่านั้นให้เขียนในฟังก์ชันและตรวจสอบว่าค่าอยู่ระหว่าง 0-120 หรือไม่ ถ้าไม่ใช่ให้วนรับจนกว่าจะได้ค่าที่ถูกต้อง

1) เอาต์พุต คืออายุระหว่าง 0-120 ปี
2) อินพุต คืออายุ ให้ตรวจสอบและวนรับค่าจนกว่าจะได้ค่าที่อยู่ระหว่าง 0-120
3) โจทย์กำหนดให้ใช้ฟังก์ชันในการรับค่าและตรวจสอบค่าอายุให้อยู่ระหว่าง 0-120 ปี
4) ไม่มีข้อมูลเพิ่มเติม

ในโจทย์ข้อนี้เราต้องเขียนฟังก์ชันเพื่อวนรับค่าอายุเข้ามาจากแป้นพิมพ์และตรวจสอบจนกว่าจะได้ค่าที่ถูกต้อง เราจะเห็นได้ว่าการรับอินพุตในข้อนี้ ไม่ได้มีแค่ 3 ขั้นตอนที่เราเคยใช้ประจำ คือประกาศตัวแปร แสดงข้อความรับค่า และรับค่าอีกต่อไป แต่ต้องมีการวนตรวจสอบค่าเพิ่มเข้ามาด้วย (บรรทัดที่ 14-16) ทำให้โค้ดในส่วนของการรับค่านั้นยาว (บรรทัดที่ 10-18) จึงสมควรจะแยกออกไปเป็นฟังก์ชันดีกว่า จากนั้นเมื่อได้ค่าที่เราต้องการเราค่อยส่งค่านั้นกลับมาแสดงผลในฟังก์ชันเมน เราสามารถเขียนโค้ดได้ดังนี้

โค้ด:

```
1  int getAge();
2
3  int main(){
```

```
4       int age = getAge();
5       cout << "Your age is " << age << endl;
6       return 0;
7   }
8
9   int getAge(){
10      int n;
11      while(true){
12          cout << "How old are you: ";
13          cin >> n;
14          if(n < 0) cout << "\a\tnegative.";
15          else if(n > 120) cout << "\a\tover 120.";
16          else return n;
17          cout << "\n\tTry again.\n";
18      }
19  }
```

เราสังเกตว่าในฟังก์ชันเมนมีแค่ 3 บรรทัดเท่านั้น เมื่อเราตั้งชื่อฟังก์ชันให้มีความหมายก็จะทำให้การอ่านว่าโปรแกรมนี้ทำงานอะไรบ้างนั้นง่ายขึ้นมาก สังเกตในฟังก์ชัน getAge() เราจะวนรับค่าโดยใช้การวนไม่รู้จบ โดยไม่มีเงื่อนไขสำหรับคำสั่ง break แต่จะเป็นเงื่อนไขให้ return n; ในบรรทัดที่ 16 แทน นั่นคือ เมื่อเราได้ค่าที่เราต้องการ (ไม่น้อยกว่า 0 และไม่มากกว่า 120) แล้ว เราก็จะจบฟังก์ชันและคืนค่า n ที่รับเข้ามาไปให้ฟังก์ชันเมนแสดงผล ตัวอย่างผลการทำงานเมื่อผู้ใช้ใส่ -8, 200 และ 36 ผ่านทางแป้นพิมพ์ตามลำดับเป็นดังนี้

ผลการทำงาน:

```
How old are you: -8
        negative.
        Try again.
How old are you: 200
        over 120.
        Try again.
How old are you: 36
Your age is 36
```

หมายเหตุ ตัวอักขระ ‘\a’ เป็นอักขระพิเศษทำให้เวลาโปรแกรมแสดงผลจะมีเสียงเตือน (alert) จากระบบ (นักศึกษาควรลองเอาโค้ดไปรันดู มันจะมีเสียงติ๊ดตึ่ง – แล้วแต่เสียงเตือนในแต่ละเครื่อง – น่ารักดี ^^)

7.3 ขอบเขตของตัวแปรและตัวแปรเฉพาะที่ (Scope of a variable and a local variable)

ตัวแปรหนึ่งๆ มีขอบเขตที่ถูกกำหนดไว้ คือเราสามารถเอาตัวแปรหนึ่งๆ ไปใช้ได้โดยเริ่มนับจากการประกาศตัวแปรจนจบบล็อคในสุดที่มันถูกประกาศอยู่ ดังที่ได้กล่าวไปแล้วบล็อคของโปรแกรมจะถูกจำกัดเขตโดยปีกกา ({ }) คู่หนึ่ง เราเรียกตัวแปรที่ประกาศใช้ในขอบเขตของบล็อคๆ หนึ่งว่า ตัวแปรเฉพาะที่ (local variable) ตัวอย่างเช่น

โค้ด:

```
1   int main(){
2         int a, b;
3         cin >> a >> b;
4         cout << a << b << endl;
5         if(a > b)
6         {
7               int m = 2;
8         }
9         return 0;
10  }
```

ในโค้ดนี้ ตัวแปร a และ b ถูกประกาศในบล็อคของฟังก์ชันเมน ดังนั้นเราเรียก a และ b ว่าเป็นตัวแปรเฉพาะที่ของฟังก์ชันเมน เราสามารถเรียกใช้ตัวแปรทั้งสองได้ในเฉพาะฟังก์ชันเมนเท่านั้น ฟังก์ชันอื่นๆจะไม่สามารถเรียกใช้ตัวแปรทั้งสองตัวนี้ได้ ส่วนตัวแปร m นั้นถูกประกาศอยู่ในบล็อคจากบรรทัดที่ 6 ถึงบรรทัดที่ 8 เพราะฉะนั้นเราเรียก m ว่าตัวแปรเฉพาะที่ของบล็อคดังกล่าว เราจะใช้ m ได้เฉพาะในระหว่างปีกกาคู่นี้เท่านั้น ถ้าเราเรียกใช้มันจากเขตนอกปีกกา (นอกเหนือจากบรรทัดที่ 6-8) ถึงแม้จะเป็นในฟังก์ชันเมนก็ตาม เราจะไม่สามารถเรียกใช้ได้ (โปรแกรมจะคอมไพล์ไม่ผ่าน เพราะมันไม่รู้จัก m นอกขอบเขตของมัน)

เนื่องจากเราสามารถแบ่งส่วนของโปรแกรมเป็นบล็อคได้โดยตัวแปรจะใช้ได้เฉพาะในบล็อคนั้นๆ เราจึงสามารถประกาศตัวแปรชื่อเดียวกันในบล็อคต่างกันได้ถ้าบล็อคเหล่านั้นซ้อนอยู่ข้างใน (nested) หรือแยกขาดกันโดยสิ้นเชิง (disjointed) ตัวอย่างเช่น

โค้ด:

```
1   void change(int x) {
2         x = x + 100;
```

```
3       cout  <<  "Value of x in change() is: " << x << '\n';
4   }
5
6   int main( ){
7       int x;
8       x = 100;
9       cout  <<  "Value of x in  before change() is: "
10      << x << '\n';
11      change(x);
12      cout  <<  "Value of x in  after change() is: "
13      << x << '\n';
14      return 0;
15  }
```

เราจะเห็นได้ว่ามี x ในโปรแกรมนี้อยู่ 2 ตัว ตัวหนึ่งอยู่ในฟังก์ชันเมนซึ่งถูกประกาศและให้ค่าในบรรทัดที่ 7-8 อีกตัวหนึ่งอยู่ในฟังก์ชัน change() ตัวแปรสองตัวนี้ถึงจะชื่อเหมือนกันแต่จริงๆ แล้วเป็นคนละตัวกัน พื้นที่เก็บค่าในหน่วยความจำ (memory) ก็เป็นคนละที่กัน ให้ลองเปรียบเทียบว่ามีคนชื่อสมชาย 2 คน คนหนึ่งอยู่กรุงเทพ แต่อีกคนอยู่ขอนแก่น สมมติว่าไม่มีการถ่ายเทประชากรระหว่างสองจังหวัดนี้และคนในจังหวัดจะไม่รู้จักคนในจังหวัดอื่น ในกรุงเทพนั้นเวลาเราเรียกสมชาย เราก็จะหมายถึงสมชายที่อยู่ในกรุงเทพ ในกรณีเดียวกันในจังหวัดขอนแก่นเวลาเราเรียกสมชาย เราก็จะหมายถึงสมชายที่อยู่ในจังหวัดขอนแก่นเป็นต้น นั่นคือถึงชื่อจะเหมือนกันแต่เขาเป็นคนละคนกัน

ส่วนการส่งค่าตัวแปรผ่านฟังก์ชันในบรรทัดที่ 11 นั้น ณ ขณะที่เรียกฟังก์ชัน change() ตัวแปร x มีค่าเท่ากับ 100 โปรแกรมจึงส่งผ่านค่า 100 ไปยังฟังก์ชัน change() (ไม่ได้ส่งผ่านตัวแปร x แต่ส่งผ่านค่าของตัวแปร x) เมื่อฟังก์ชัน change() รับตัวแปรมา มันก็จะสำเนา (copy) ค่า 100 เอามาเก็บไว้ในตัวแปร x ที่อยู่ในฟังก์ชัน change() และในบรรทัดที่ 2 ตัวแปร x ตัวนี้จะถูกเพิ่มค่า 100 เป็น 200 และถูกแสดงผลออกทางหน้าจอโดยที่ค่า x ในฟังก์ชันเมนไม่ได้เปลี่ยนตามไปด้วย ทำให้เมื่อฟังก์ชัน change() ทำงานเสร็จแล้ว และโปรแกรมกลับมาทำงานในฟังก์ชันเมนโดยทำคำสั่งแสดงผลค่า x ออกในบรรทัดที่ 12-13 ค่าของ x จึงจะยังเป็น 100 อยู่ ดังนั้นผลรันของโปรแกรมนี้จะเป็นดังนี้

ผลการทำงาน:

```
Value of x in  before change() is: 100
Value of x in  in change() is: 200
```

```
Value of x in  after change() is: 100
```

นอกจากตัวแปรเฉพาะที่ (local variable) แล้ว เรายังมีตัวแปรอีกชนิดหนึ่งเรียกว่าตัวแปรโกลบอล (global variable) ตัวแปรโกลบอลเป็นตัวแปรที่ประกาศนอกฟังก์ชันทุกฟังก์ชันในโปรแกรมนั้น ปกติแล้วมักประกาศไว้ที่หัวไฟล์หลังจากการประกาศฟังก์ชันแต่อยู่ก่อนฟังก์ชันเมน (ถ้าต้องใช้ตัวแปรโกลบอลนี้ในฟังก์ชันเมนด้วย ต้องประกาศก่อนฟังก์ชันเมน) ตัวแปรชนิดนี้มีลักษณะพิเศษคือเราสามารถเข้าถึง (อ่านค่าหรือเปลี่ยนค่า) ในทุกๆ ฟังก์ชันได้เลย ถ้ามีการเปลี่ยนค่าตัวแปรโกลบอล ณ ฟังก์ชันใด ค่าของตัวแปรโกลบอลก็จะเปลี่ยนไปเมื่อเรียกใช้ในฟังก์ชันอื่นๆ ทีหลังด้วย ในกรณีที่ในฟังก์ชันหนึ่งมีการประกาศชื่อตัวแปรเหมือนตัวแปรโกลบอล เราแยกการเข้าถึงตัวแปรโกลบอลด้วยการเขียนเครื่องหมายโคลอนสองตัว (::) นำหน้าตัวแปร เช่น เราสามารถเข้าถึงตัวแปรตัวแปรโกลบอลที่ชื่อว่า x ได้โดยเขียนว่า ::x ส่วนตัวแปรเฉพาะที่ในฟังก์ชันนั้นที่ชื่อเหมือนกัน เราก็เขียนชื่อตัวแปรนั้นเฉยๆ

ตัวอย่างที่ 9

โจทย์ : จงหาผลรันของโปรแกรมต่อไปนี้

โค้ด:

```
1   void f();
2   void g();
3
4   int x = 11;
5
6   int main() {
7         int x = 22;
8         {     int x = 33;
9               cout << "In block inside main(): x = " << x << endl;
10        }
11        cout << "In main(): x = " << x << endl;
12        cout << "In main(): ::x = " << ::x << endl;
13        f();
14        g();
15  }
16
17  void f() {
18        int x = 44;
```

```
19        cout << "In f(): x = " << x << endl;
20  }
21
22  void g() {
23        cout << "In g(): x = " << x << endl;
24  }
25
```

ในโปรแกรมนี้มีฟังก์ชัน 3 ฟังก์ชันได้แก่ฟังก์ชันเมน ฟังก์ชัน f() และฟังก์ชัน g() และมีตัวแปร x ที่เป็นตัวแปรโกลบอลซึ่งมีค่าเท่ากับ 11 ในการหาผลรันของโปรแกรม เราต้องเริ่มอ่านจากฟังก์ชันเมนเสมอ เพราะเป็นจุดเริ่มของโปรแกรม ในฟังก์ชันเมนมีการประกาศตัวแปรเฉพาะที่ชื่อ x อีกซึ่งมีค่าเท่ากับ 22 จากนั้นเป็นบล็อคจากบรรทัดที่ 8 ถึงบรรทัดที่ 10 เรียกว่าบล็อค A ซึ่งก็มีการประกาศตัวแปรเฉพาะที่ชื่อ x อีกตัวเช่นกันซึ่งมีค่าเท่ากับ 33 ในกรณีนี้โปรแกรมจะสามารถคอมไพล์และรันได้เพราะ x อยู่ในขอบเขตที่ซ้อนกันอยู่ข้างใน (nested) อย่างชัดเจน ในบรรทัดที่ 9 โปรแกรมให้แสดงค่า x ออกทางหน้าจอ เราจะได้ค่า x เป็น 33 เพราะเป็นค่า x ของบล็อคในสุดที่ x ถูกประกาศอยู่ จากนั้นในบรรทัดที่ 11 ซึ่งอยู่นอกบล็อค A แล้วก็มีการแสดงค่า x ออกทางหน้าจอ เราจะได้ค่า x เป็น 22 เพราะตอนนี้บล็อคในสุดของ x ตัวนี้จะเป็นฟังก์ชันเมน ต่อมาในบรรทัดที่ 12 โปรแกรมแสดงค่า ::x ออกทางหน้าจอ เรารู้ว่าสัญลักษณ์นี้หมายความถึงตัวแปร x ที่เป็นตัวแปรโกลบอล เราจึงจะได้ค่า x เป็น 11

จากนั้น โปรแกรมเรียกฟังก์ชัน f() และ g() ตามลำดับ เราจึงต้องไปทำฟังก์ชัน f() ก่อน ในฟังก์ชัน f() โปรแกรมได้ประกาศตัวแปรเฉพาะที่มีค่าเท่ากับ 44 และแสดงตัวแปรนั้นออกทางหน้าจอ เราจึงได้ค่า x เป็น 44 เป็นการจบฟังก์ชันและกลับมาเริ่มทำบรรทัดที่ 14 ในฟังก์ชันเมนซึ่งเรียกฟังก์ชัน g() ในฟังก์ชัน g() โปรแกรมได้แสดงค่า x ออกโดยที่ไม่ได้ประกาศหรือรับตัวแปร x ผ่านมายังฟังก์ชัน ในกรณีนี้เราจะแสดงค่า x ที่เป็นตัวแปรโกลบอลออกทางหน้าจอ ซึ่งมีค่าเท่ากับ 11 โดยสรุปแล้วผลรันของโปรแกรมนี้คือ

ผลการทำงาน:

```
In block inside main(): x = 33
In main(): x = 22
In main(): ::x = 11
In f(): x = 44
In g(): x = 11
```

7.4 การส่งผ่านพารามิเตอร์ไปยังฟังก์ชัน (Passing parameters to a function)

การส่งผ่านพารามิเตอร์ไปยังฟังก์ชันนั้นมี 2 แบบคือการส่งผ่านโดยใช้ค่า (passing by value) และการส่งผ่านโดยการอ้างอิง (passing by reference) มีรายละเอียดการทำงานและการใช้งานดังนี้

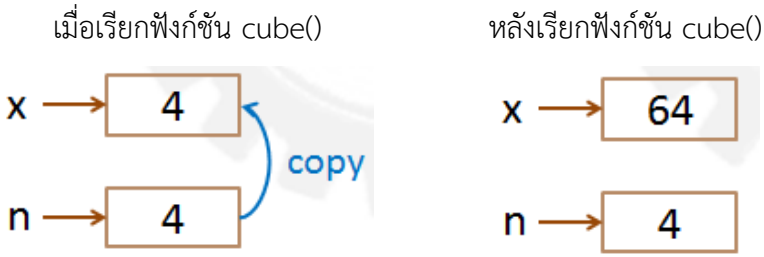
1) การส่งผ่านโดยใช้ค่า (passing by value)

การส่งผ่านโดยใช้ค่า (passing by value) เป็นการส่งสำเนาค่าของตัวแปรนั้นไปยังฟังก์ชันที่โปรแกรมเรียก (ปลายทาง) ดังนั้นถ้ามีการเปลี่ยนแปลงค่าที่เราส่งผ่านในฟังก์ชันที่ถูกเรียก (เปลี่ยนค่าในฟังก์ชันปลายทาง) ค่าของตัวแปรนั้นในฟังก์ชันที่ทำการเรียก (ต้นทาง) จะไม่มีการเปลี่ยนแปลง เพราะการเปลี่ยนค่าตัวแปรในฟังก์ชันปลายทางนั้นเป็นแค่การเปลี่ยนค่าของสำเนา ไม่ได้เปลี่ยนค่าจริงๆของตัวแปรที่อยู่ต้นทาง

โค้ด 1:

```
1   int cube(int x){
2         x = x * x * x;
3         return x;
4   }
5
6   int main(){
7         int n = 4;
8         cout << "\tcube(" << n << ") = " << cube(n) << endl;
9         cout << "\t current n = " << n << endl;
10        return 0;
11  }
```

ตัวอย่างในโค้ด 1 เป็นตัวอย่างการส่งผ่านโดยใช้ค่า เมื่อโปรแกรมเรียกฟังก์ชัน cube() ในฟังก์ชันเมนในบรรทัดที่ 8 โปรแกรมจะทำการสำเนา (copy) ค่า n (เท่ากับ 4) และส่งไปเก็บไว้ที่ตัวแปร x ในฟังก์ชัน cube() จากนั้นโปรแกรมจะเขียนค่า x^3 ลงใน x (เท่ากับ 64) และทำการส่งคืนค่า x มาแสดงผลออกทางหน้าจอที่ฟังก์ชันเมนในบรรทัดเดียวกันนั้น ค่า n ในฟังก์ชันเมนก็จะไม่เปลี่ยน ดังแสดงในรูปที่ 4 เมื่อทำการแสดงผลออกทางหน้าจอในบรรทัดที่ 9 เราก็จะเห็นว่าค่า n มีค่าเท่ากับ 4 เหมือนเดิม



รูปที่ 4 ค่าของตัวแปรเมื่อถูกส่งผ่านโดยการใช้ค่า

2) การส่งผ่านโดยการอ้างอิง (passing by reference)

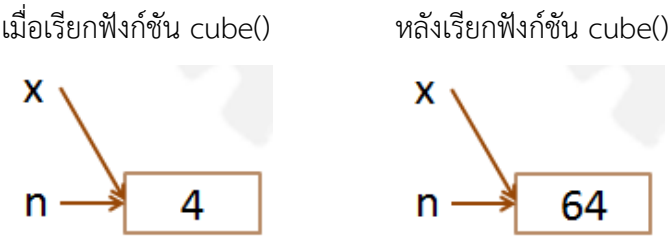
การส่งผ่านโดยการอ้างอิง (passing by reference) เป็นการส่งที่อยู่ของหน่วยความจำ (memory) ที่เก็บค่าของตัวแปรนั้นไปยังฟังก์ชันที่โปรแกรมเรียก (ปลายทาง) ถ้ามีการเปลี่ยนแปลงค่าที่เราส่งผ่านในฟังก์ชันที่ถูกเรียก (ปลายทาง) ค่าของตัวแปรนั้นในฟังก์ชันที่ทำการเรียก (ต้นทาง) จะถูกเปลี่ยนไปด้วย เพราะค่าของตัวแปรนั้นถูกเปลี่ยนในหน่วยความจำไปแล้ว

โค้ด 2:

```
1   int cube(int &x){
2         x = x * x * x;
3         return x;
4   }
5
6   int main(){
7         int n = 4;
8         cout << "\tcube(" << n << ") = " << cube(n) << endl;
9         cout << "\t current n = " << n << endl;
10        return 0;
11  }
```

ในโค้ด 2 เป็นตัวอย่างการส่งผ่านโดยการอ้างอิง นั่นคือการส่งที่อยู่ของหน่วยความจำที่เก็บค่าของตัวแปรนั้นไปยังฟังก์ชันที่ถูกเรียก เราจะสังเกตว่าถ้าเป็นการส่งผ่านโดยการอ้างอิง เราจะเขียนเครื่องหมาย & กำกับไว้หลังชนิดและหน้าชื่อของพารามิเตอร์ในการนิยามฟังก์ชันดังแสดงในบรรทัดที่ 1 และกำกับไว้หลังชนิดของพารามิเตอร์ในการประกาศฟังก์ชัน เมื่อโปรแกรมเรียกฟังก์ชัน cube() ในฟังก์ชันเมนในบรรทัดที่ 8 โปรแกรมจะทำการส่งที่อยู่ของ n ไปให้ฟังก์ชัน cube() และให้ตัวแปร x ในฟังก์ชัน cube() “ชี้” ไปที่หน่วยความจำที่เก็บค่า n หรือเรียกได้

ว่ามีที่อยู่เดียวกับตัวแปร n ในฟังก์ชันเมน ดังแสดงในรูปที่ จากนั้นโปรแกรมจะเขียนค่า x^3 ลงใน x (ที่ชี้ไปที่เดียวกับ n แล้ว) และทำการส่งคืนค่า x เท่ากับ 64 มาแสดงผลออกทางหน้าจอที่ฟังก์ชันเมน ค่า n ในฟังก์ชันเมนก็จะเปลี่ยนไปด้วยเพราะมันคือกล่องหน่วยความจำที่เดียวกับ x ที่ถูกเปลี่ยนค่าไปแล้ว เมื่อทำการแสดงผลออกทางหน้าจอในบรรทัดที่ 9 เราก็จะเห็นว่าค่า n มีค่าเท่ากับ 64



รูปที่ 5 ค่าของตัวแปรเมื่อถูกส่งผ่านโดยการอ้างอิง

หมายเหตุ: เนื่องจากการส่งผ่านค่าโดยการอ้างอิงจะส่งที่อยู่ของตัวแปรไป ดังนั้นตอนที่เรียกฟังก์ชันเราต้องใส่ตัวแปรที่ตำแหน่งของพารามิเตอร์ที่ส่งผ่านแบบอ้างอิงเท่านั้น เราไม่สามารถใส่เป็นค่าคงที่หรือนิพจน์ได้ เช่น เราต้องใส่เป็น cube(n) แต่เราไม่สามารถใส่ cube(4) ได้

ตัวอย่างที่ 10 ฟังก์ชัน swap()

โจทย์ : จงหาผลรันของโปรแกรมต่อไปนี้

โค้ด:

```
1   void swap(float&, float&);
2
3   int main(){
4         float a = 22.2, b = 44.4;
5         cout << "a = " << a << " b = " << b << endl;
6         swap(a,b);
7         cout << "a = " << a << " b = " << b << endl;
8         return 0;
9   }
10
11  void swap(float &x, float &y){
12        float temp = x;
13        x = y;
14        y = temp;
15  }
```

โจทย์ข้อนี้แสดงค่า a และ b ก่อนและหลังการเรียกฟังก์ชัน swap() (swap แปลว่าการสลับที่) ฟังก์ชัน swap() เป็นฟังก์ชันที่ไม่คืนค่าใดๆ (void function) มีพารามิเตอร์ทั้งสองตัวเป็นแบบอ้างอิง ซึ่งหมายความว่าถ้ามีการเปลี่ยนค่าของพารามิเตอร์ทั้งสองในฟังก์ชัน swap() แล้ว ค่าของตัวแปรในฟังก์ชันเมนก็จะถูกเปลี่ยนไปด้วย การทำงานของฟังก์ชันนี้คือการสลับที่ตัวแปร x และ y ดังนั้นเมื่อฟังก์ชันทำงานเสร็จแล้ว ตัวแปรที่ถูกส่งเข้าไปทั้งตัวจะสลับค่ากัน ในกรณีนี้ ค่า a และ b จะสลับกัน เป็นค่า 44.4 และ 22.2 ตามลำดับ

ผลการทำงาน:

```
a = 22.2 b = 44.4
a = 44.4 b = 22.2
```

สรุปการผ่านพารามิเตอร์โดยใช้ค่าและโดยการอ้างอิง

ตารางที่ 2 เปรียบเทียบการผ่านพารามิเตอร์โดยใช้ค่าและโดยการอ้างอิง

	การผ่านพารามิเตอร์โดยใช้ค่า	**การผ่านพารามิเตอร์โดยการอ้างอิง**
การประกาศพารามิเตอร์	int x	int &x
ชนิดของตัวแปร	x เป็นตัวแปรเฉพาะที่	x เป็นตัวแปรเฉพาะที่
ค่าที่ถูกส่งผ่าน	สำเนาของค่าตัวแปร	ที่อยู่ที่ตัวแปร
ถ้ามีการเปลี่ยนค่าในฟังก์ชัน	ค่าตัวแปรไม่เปลี่ยนหลังจบฟังก์ชัน	ค่าตัวแปรเปลี่ยนหลังจบฟังก์ชัน
การส่งค่า	ใช้ค่าคงที่ ตัวแปร หรือนิพจน์	ใช้ตัวแปรเท่านั้น

การคืนค่ามากกว่าหนึ่งค่าจากฟังก์ชัน

โดยปกติแล้วฟังก์ชันจะสามารถคืนค่าได้ค่าเดียวจากคำสั่ง return ถ้าเราต้องการคืนค่ามากกว่าหนึ่งค่า เช่นเราต้องการเขียนฟังก์ชันเพื่อรับค่าจากผู้ใช้มากกว่าหนึ่งค่า เราจะต้องใช้การผ่านตัวแปรแบบอ้างอิงเข้าไปที่ฟังก์ชันนั้น แล้วให้ฟังก์ชันให้ค่าตัวแปรที่เราต้องการ “คืนค่า” เราก็จะได้การ “คืนค่า” มากกว่าหนึ่งค่า ดังแสดงในตัวอย่างที่ 11

ตัวอย่างที่ 11 พื้นที่วงกลมและเส้นรอบวง

โจทย์ : จงเขียนโปรแกรมภาษา C++ เพื่อรับค่ารัศมีเข้ามาจากแป้นพิมพ์ จากนั้นให้ใช้ฟังก์ชันในการคำนวณหาพื้นที่วงกลมและเส้นรอบวงของวงกลมนั้น แล้วคืนค่ากลับมาแสดงผลในฟังก์ชันเมน

1) เอาต์พุต คือพื้นที่และเส้นรอบวงของวงกลม
2) อินพุต คือรัศมีของวงกลม
3) โจทย์กำหนดให้รับค่าและแสดงค่าในฟังก์ชันเมน แต่ให้คำนวณพื้นที่และเส้นรอบวงโดยใช้ฟังก์ชันอื่น
4) พื้นที่ของวงกลมคือ πr^2 เส้นรอบวงคือ $2\pi r$

ข้อนี้โจทย์กำหนดให้รับค่าและแสดงค่าในฟังก์ชันเมน แต่ให้คำนวณพื้นที่และเส้นรอบวงโดยใช้ฟังก์ชัน เราจึงต้องรับค่ารัศมีที่ฟังก์ชันเมนและส่งรัศมีเข้าฟังก์ชันไปเพื่อคำนวณพื้นที่และเส้นรอบวง แต่ในการส่งค่าคืนมานั้น เราต้องการสองค่าคือค่าพื้นที่และค่าเส้นรอบวง เราจึงต้องใช้การส่งค่าโดยการอ้างอิงที่อยู่ของตัวแปร 2 ตัว คือ a สำหรับพื้นที่ (area) และ c สำหรับเส้นรอบวง (cirbum) ฟังก์ชันเมนจะได้สามารถรับค่าทั้งสองค่าที่ถูกคำนวณในฟังก์ชัน แล้วมาแสดงผลอย่างถูกต้อง

โค้ด:

```
void computeCircle(double &area, double &circum, double r){
      const double PI = 3.141592653589793;
      area = PI * r * r;
      circum = 2 * PI * r;
}

int main(){
      double r,a,c;
      cout << "Enter a radius: ";
      cin >> r;

      computeCircle(a,c,r);
      cout << "area = " << a << " circumference = " << c << endl;
      return 0;
}
```

การผ่านค่าคงที่โดยการอ้างอิง (passing by constant reference)

การผ่านค่าโดยการอ้างอิงมีข้อดีคือ สามารถให้ฟังก์ชันเปลี่ยนค่าของตัวแปรเพื่อที่จะคืนค่าได้หลายค่า และทำให้ประหยัดพื้นที่ในการเก็บตัวแปรเพราะไม่ต้องทำสำเนาตัวแปรเวลาส่งไปที่ฟังก์ชัน ถ้าเราต้องการประหยัดพื้นที่ในการเก็บตัวแปรแต่ไม่ต้องการให้ฟังก์ชันเปลี่ยนค่าของตัวแปรได้ เราต้องใช้การผ่านค่าคงที่โดยการอ้างอิง โดยเราจะประกาศพารามิเตอร์โดยใช้คำว่า const และเครื่องหมาย & กำกับไว้ เช่น const int &pi

โค้ด:

```
1   void f(int x, int &y, const int &z){
2         x += z;
3         y += z;
4         cout << "x = " << x << " y = " << y << " z = " << z
5         << endl;
6   }
7
8   int main(){
9         int a = 22, b = 33, c = 44;
10        cout << a << " " << b << " " << c << endl;
11        f(a, b, c);
12        cout << a << " " << b << " " << c << endl;
13        f(2*a-3, b, c);
14        cout << a << " " << b << " " << c << endl;
15        return 0;
16  }
```

จากตัวอย่างโค้ดข้างบนเราผ่านค่า c เข้าฟังก์ชัน f() แบบค่าคงที่โดยการอ้างอิง เพราะฉะนั้นฟังก์ชัน f() จะไม่สามารถเปลี่ยนค่า z (ค่า c ในฟังก์ชันเมน) ได้ แม้ว่าจะผ่านโดยการอ้างอิงก็ตาม ถ้าเราเขียนให้ฟังก์ชันสามารถเปลี่ยนค่า z ได้ โปรแกรมจะคอมไพล์ไม่ผ่าน

ผลการทำงาน:

```
22 33 44
x = 66 y = 77 z = 44
22 77 44
x = 85 y = 121 z = 44
22 121 44
```

สรุปสิ่งที่ควรได้จากบทเรียน

1) การใช้ฟังก์ชันที่ถูกกำหนดไว้ก่อนในคลังโปรแกรมจะต้องมีการ #include (library) ที่หัวไฟล์เสมอ
2) ชนิดของฟังก์ชันจะต้องเป็นชนิดเดียวกับค่าที่ถูกส่งคืน (returned value)
3) เราสามารถเขียนนิยามฟังก์ชันก่อนหรือหลังฟังก์ชันเมนก็ได้ ถ้าเราเขียนไว้หลังฟังก์ชันเมน เราต้องทำการประกาศฟังก์ชันนั้นก่อนฟังก์ชันเมน
4) การส่งผ่านพารามิเตอร์ต้องส่งผ่านให้ถูกตำแหน่งและชนิด เพราะฟังก์ชันจะจับคู่ค่าที่ถูกส่งผ่านเข้ามาและตัวแปรที่ใช้ในฟังก์ชันตามลำดับที่ถูกส่งเข้ามา
5) ตัวแปรเฉพาะที่ (local variable) จะสามารถถูกใช้ได้ภายในขอบเขตของบล็อคในสุดที่มันถูกประกาศไว้เท่านั้น ส่วนตัวแปรโกลบอล (global variable) จะสามารถถูกใช้ได้ในทุกที่ของโปรแกรม
6) เราสามารถผ่านค่าเข้าไปยังฟังก์ชันได้ 2 แบบ คือการส่งผ่านโดยใช้ค่า (ส่งสำเนาค่าของตัวแปร) และการส่งผ่านโดยการอ้างอิง (ส่งที่อยู่ของตัวแปร)