บทที่ 8

การเรียกใช้ฟังก์ชัน

วัตถุประสงค์การเรียนรู้

อธิบายการทำงานของโปรแกรมในรูปแบบการเรียกใช้ฟังก์ชัน

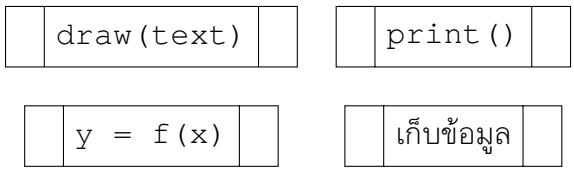
1. กำหนดพารามิเตอร์และรับค่าคืนจากฟังก์ชันได้เมื่อกำหนดต้นแบบของฟังก์ชันมาให้
2. แยกแยะขอบเขตการใช้งานของตัวแปรเมื่อมีฟังก์ชันมาให้ได้
3. เขียนฟังก์ชันเพื่อให้ทำงานตามที่ต้องการได้

8.1 นิยามและสัญลักษณ์

โปรแกรมย่อย (Pre-defined process หรือ subroutine) เป็นชุดคำสั่งซึ่งมีการกำหนดการทำงานไว้ล่วงหน้าแล้ว โดยมากเรามักแบ่งโปรแกรมขนาดใหญ่ออกเป็นโปรแกรมย่อยหลายๆ ส่วน เพื่อให้เข้าใจได้ง่าย บำรุงรักษาสะดวก นอกจากนี้ ส่วนของโปรแกรมใดที่ถูกเรียกใช้บ่อย ก็มักจะถูกเขียนแยกออกมาเป็นโปรแกรมย่อยด้วย การใช้สัญลักษณ์โปรแกรมย่อยแทนขั้นตอนการทำงานแบบละเอียดทำให้ผังงานกระชับ เข้าใจง่าย ในรหัสต้นฉบับเองก็เช่นกัน

โปรแกรมย่อยกับฟังก์ชันมีลักษณะเป็นชุดคำสั่งซึ่งทำงานจบในตัวเหมือนกัน แต่แตกต่างกันที่ฟังก์ชันนั้นจะสามารถคืนค่าได้เช่นเดียวกับฟังก์ชันทางคณิตศาสตร์ แต่โปรแกรมย่อยจะไม่มีการคืนค่า ภาษาโปรแกรมบางภาษา เช่น ภาษาซี ใช้โครงสร้างแบบฟังก์ชันทั้งหมด และเพิ่มลักษณะการคืนค่าชนิด `void` แทนการไม่คืนค่าเข้ามา ดังนั้นบางครั้งอาจใช้คำว่าฟังก์ชันและโปรแกรมย่อยแทนกันได้โดยไม่ผิดความหมาย

ผังงานใช้สัญลักษณ์สี่เหลี่ยมมุมฉากซึ่งมีขอบข้างเป็นเส้นขนานแทนการเรียกใช้โปรแกรมย่อย และมีชื่อโปรแกรมย่อยกำกับในสัญลักษณ์ ซึ่งจะอ้างอิงไปถึงการทำงานอย่างใดอย่างหนึ่งซึ่งกำหนดไว้ในผังงานอื่นหรือผังงานหน้าอื่น ดังตัวอย่างในรูปที่ 8.1



รูปที่ 8.1: สัญลักษณ์การเรียกใช้โปรแกรมย่อยหรือฟังก์ชัน

เราอาจระบุค่าต่างๆ ที่ต้องการส่งให้โปรแกรมย่อยหรือฟังก์ชัน และตัวแปรสำหรับรับค่าคืนลงในผังงานด้วยก็ได้ หรือไม่ระบุก็ได้ เพื่อให้ผังงานแสดงขั้นตอนการทำงานที่ชัดเจน ควรระบุค่าที่รับส่ง

สัญลักษณ์ของโปรแกรมย่อยในรหัสต้นฉบับขึ้นกับภาษาที่ใช้ โดยมากจะเป็นการประกาศหรือนิยามชื่อก่อนใช้งาน การอ้างอิงถึงโปรแกรมย่อยหรือฟังก์ชันทำได้ผ่านชื่อของโปรแกรมย่อยหรือฟังก์ชันนั้น ในกรณีที่มีการคืนค่า การเรียกใช้ฟังก์ชันนั้นสามารถมองเสมือนเป็นนิพจน์ซึ่งให้ผลลัพธ์เป็นค่าในชนิดที่กำหนดได้

8.2 โครงสร้างของฟังก์ชันและการเรียกใช้

ฟังก์ชันประกอบด้วยองค์ประกอบหลัก 4 ส่วน ได้แก่

1. ชื่อของฟังก์ชัน มีข้อกำหนดในการตั้งเช่นเดียวกับตัวแปร ในบางภาษาจะยอมให้ตั้งชื่อฟังก์ชันซ้ำกันได้หากมีพารามิเตอร์ที่ต่างกัน เครื่องจะเลือกฟังก์ชันที่ถูกต้องจากลักษณะของพารามิเตอร์ที่ป้อนเข้ามาให้เอง บางภาษาเช่นไพธอนยอมให้ตั้งชื่อซ้ำได้ โดยจะเป็นการนิยามชื่อใหม่ตามการประกาศล่าสุดที่มี อย่างไรก็ตาม ควรหลีกเลี่ยงการใช้ชื่อซ้ำเพื่อป้องกันความสับสนเมื่อเรียกใช้
2. พารามิเตอร์ คือตัวแปรสำหรับรับค่าที่ส่งเข้ามาให้ใช้ในฟังก์ชัน จะมีหรือไม่มีก็ได้
3. ชนิดของค่าที่ส่งคืนให้ตัวเรียก หรือไม่มีหากไม่ต้องการคืนค่าใดๆ
4. ชุดคำสั่งสำหรับขั้นตอนการทำงานของฟังก์ชัน และคำสั่งสำหรับคืนค่า ตัวชุดคำสั่งนี้คือสิ่งที่นำไปเขียนเป็นผังงานของฟังก์ชัน

หากเป็นโปรแกรมย่อย ก็จะตัดส่วนการคืนค่าออกไปได้ การประกาศฟังก์ชันโดยทั่วไปมีโครงสร้างดังรหัสคำสั่งที่ 8.1

```
ชนิดของตัวแปรคืนค่า ชื่อฟังก์ชัน ( ตัวแปรรับค่า,  ตัวแปรรับค่า, ...)
    ชุดคำสั่ง
    return ค่าส่งคืน
```

รหัสคำสั่งที่ 8.1: โครงสร้างของฟังก์ชัน

การเรียกใช้โปรแกรมย่อยหรือฟังก์ชันทำโดยการระบุชื่อโปรแกรมย่อยนั้นในโปรแกรมหลัก พร้อมทั้งส่งค่าต่างๆ ที่จำเป็นไปให้ ในกรณีที่เป็นฟังก์ชันและมีค่าคืนกลับมา หากต้องการเก็บค่าไว้ใช้งาน ก็สามารถตั้งตัวแปรมาแล้วกำหนดค่าให้เป็นผลลัพธ์ของฟังก์ชันได้

ต้นแบบ (signature, prototype) ของฟังก์ชัน ประกอบด้วย ชื่อฟังก์ชัน ชนิดของตัวแปรรับค่าต่างๆ ซึ่งเป็นพารามิเตอร์ของฟังก์ชัน และชนิดของตัวแปรคืนค่า การเรียกใช้ฟังก์ชันนั้นไม่จำเป็นต้องรู้ขั้นตอนการทำงานภายในฟังก์ชัน เพียงรู้ต้นแบบก็จะสามารถเรียกใช้ฟังก์ชันเหล่านั้นได้ถูกต้อง

ตัวอย่าง 8.1 (การเรียกใช้ฟังก์ชันการหารากที่สอง). ฟังก์ชันการหารากที่สอง มีต้นแบบดังนี้

```
double sqrt(double x);
```

แสดงว่าฟังก์ชันหารากที่สองชื่อ `sqrt` รับค่าเป็นจำนวนจริงนัยสำคัญสูงหนึ่งค่า และคืนค่าผลลัพธ์เป็นจำนวนจริงนัยสำคัญสูงเช่นกัน เมื่อเขียนคำสั่ง

```
double x;
x = sqrt(10.5);
```

จะมีการทำงานดังนี้

1. ประกาศตัวแปร `x` ให้เป็นจำนวนจริงนัยสำคัญสูง
2. เรียกใช้ฟังก์ชัน `sqrt` โดยส่งจำนวน 10.5 ให้
3. ได้ผลลัพธ์กลับมา แล้วกำหนดค่าให้ `x` เท่ากับผลลัพธ์นั้น

หากสั่ง

```
printf("%f",sqrt(2.5));
```

จะเป็นการเรียกใช้ฟังก์ชัน `sqrt` และป้อนจำนวน 2.5 ให้ฟังก์ชัน และเมื่อได้ผลลัพธ์กลับมาแล้ว ใช้ผลลัพธ์นั้นเป็นข้อมูลเข้าส่งต่อให้ฟังก์ชัน `printf` แต่ไม่เก็บผลลัพธ์ไว้ใช้งานอีก □

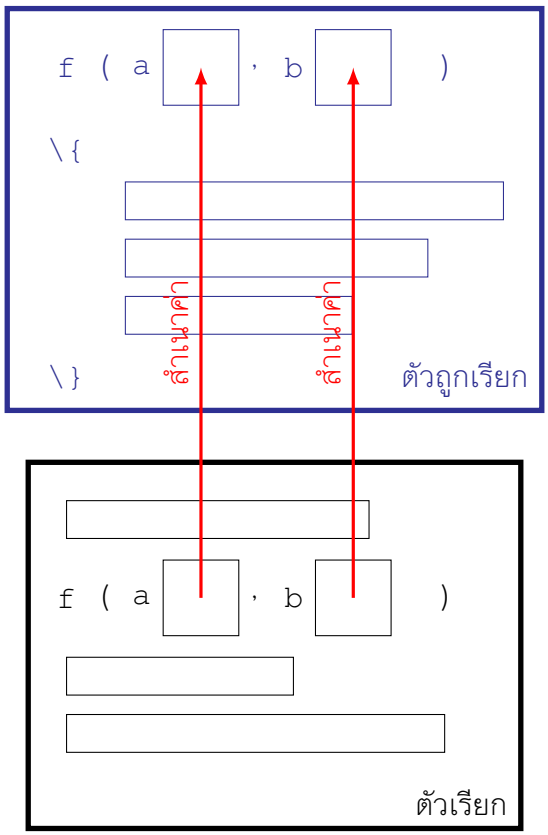
การเขียนโปรแกรมย่อยหรือฟังก์ชันนั้นเหมือนกับการออกแบบโปรแกรมปกติ คือ

1. ระบุข้อมูลเข้า ซึ่งอาจรับมาจากโปรแกรมหลักผ่านวิธีการส่งค่าแบบต่างๆ หรือรับจากผู้ใช้
2. ระบุข้อมูลออกถ้ามี อยู่ในรูปของการคืนค่าแบบต่างๆ
3. ออกแบบขั้นตอนการทำงานของโปรแกรม

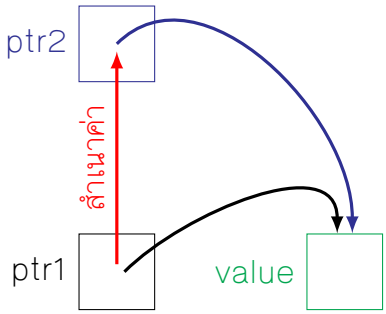
ดังนั้น เราจึงสามารถเขียนผังงานย่อยสำหรับโปรแกรมย่อยหรือฟังก์ชันได้เช่นเดียวกับโปรแกรมทั่วไป

8.3 การรับส่งค่า

การรับส่งค่าให้ฟังก์ชันหรือโปรแกรมย่อยมีสองรูปแบบ คือ ผ่านพารามิเตอร์ของฟังก์ชัน ซึ่งเป็นการสำเนาค่าจากตัวเรียก ส่งไปให้ตัวถูกเรียก และผ่านตัวแปรร่วมที่ประกาศไว้สำหรับทั้งโปรแกรม



(ก) การส่งค่าผ่านพารามิเตอร์



(ข) การส่งค่าผ่านตัวแปรชนิดตัวชี้

8.3.1 พารามิเตอร์

พารามิเตอร์ (formal parameter) เป็นตัวแปรรับค่าของฟังก์ชันหรือโปรแกรมย่อย ซึ่งตัวเรียกส่งให้ในการเรียกใช้แต่ละครั้ง การส่งค่าผ่านพารามิเตอร์เป็นการ**สำเนาค่า**จริงที่อยู่ในตัวเรียก (actual parameter) ไปให้ตัวแปรรับค่าในตัวถูกเรียก ดังรูปที่ 8.2(ก) การส่งค่าผ่านพารามิเตอร์ยังแบ่งออกเป็นสองรูปแบบ คือการสำเนาค่าทั่วๆไป เช่น จำนวนเต็ม จำนวนจริง และอักขระ กับการสำเนาค่าตัวชี้ซึ่งเป็นตำแหน่งที่อยู่ในหน่วยความจำส่งไปยังตัวถูกเรียก การสำเนาค่าทั่วๆ ไปไม่ทำให้ค่าที่มีอยู่ในตัวแปรของตัวเรียกเปลี่ยนแปลงไป แต่ในกรณีที่เป็นตัวแปรชนิดตัวชี้ เช่นตัวแปร ptr1 ในรูปที่ 8.2(ข) ซึ่งเก็บตำแหน่งที่อยู่ของค่าในตัวแปร value (ชี้ไปยัง value) เมื่อสำนาเค่าไปยังตัวแปร ptr2 ถึงแม้ค่าในตัวชี้จะถูกสำเนาไปเช่นเดียวกับการสำเนาค่าปกติ แต่ค่านั้นคือตำแหน่งที่อยู่ของข้อมูลจริง การเปลี่ยนแปลงค่าต่างๆ โดยอาศัยที่อยู่ที่กำหนดในตัวชี้ เช่น

```
*ptr2 = 10
```

จะทำให้ตัวชี้อื่นๆ (ในที่นี้คือ ptr1) เห็นค่าที่เปลี่ยนแปลงตามไปด้วย จึงสามารถใช้วิธีนี้ในการคืนค่าโดยไม่ต้องผ่านตัวแปรคืนค่า แต่อาศัยการกำหนดค่าใหม่ให้ตำแหน่งข้อมูลระบุโดยตัวแปรชนิดตัวชี้ในพารามิเตอร์แทน เช่นในกรณีคำสั่ง `scanf` ซึ่งไม่ได้คืนค่าข้อมูลที่อ่านผ่านตัวแปรคืนค่า แต่ใช้การนำข้อมูลที่อ่านได้ไปใส่ในตำแหน่งที่ระบุด้วยตัวแปรตัวชี้แต่ละตัวแทน

จำนวนของตัวแปรรับค่าขึ้นกับผู้เขียนโปรแกรมจะกำหนด ชนิดของตัวแปรแต่ละตัวไม่จำเป็นต้องเหมือนกัน การส่งค่าในตัวเรียกจะส่งเรียงลำดับตามที่กำหนดในฟังก์ชันหรือโปรแกรมย่อย คือ ตัวที่ 1 2 3 ... ตามลำดับ

8.3.2 การคืนค่า

ตัวแปรคืนค่าใช้สำหรับส่งค่าบางอย่างคืนให้ตัวเรียกนำกลับไปใช้ต่อ โดยมีชนิดของตัวแปรตามที่ประกาศไว้ในฟังก์ชัน นอกจากชนิดตัวแปรพื้นฐานแล้ว ส่วนของโปรแกรมย่อยอาจไม่คืนค่าใดๆเลยก็ได้ ในภาษาซีกำหนดชนิดไม่คืนค่านี้ว่า ชนิด `void`

คำสั่ง `return` ใช้สำหรับกำหนดค่าให้ตัวแปรส่งคืนค่า และออกจากการทำงานของฟังก์ชัน หากไม่มีการคืนค่า ในกรณีที่ไม่ต้องการคืนค่า อาจใช้คำสั่ง `return` โดยไม่ระบุค่าก็ได้เช่นกัน

8.4 ขอบเขตของตัวแปร

ตัวแปรที่ใช้มีสองประเภท แบบแรกคือตัวแปรซึ่งใช้ร่วมกันทั้งโปรแกรม ตัวแปรชนิดนี้สามารถเรียกใช้จากส่วนใดของโปรแกรมก็ได้ ไม่ว่าจะเป็นโปรแกรมหลักหรือโปรแกรมย่อย แบบที่สองคือตัวแปรเฉพาะที่ ตัวแปรชนิดนี้จะใช้ได้เฉพาะในส่วนที่ประกาศเท่านั้น ตัวแปรทั้งหมดที่ประกาศอยู่ในส่วนของโปรแกรมย่อย หากประกาศให้เป็นตัวแปรเฉพาะที่ แม้จะเป็นชื่อเดียวกับตัวแปรที่ประกาศที่อื่น ค่าต่างๆจะใช้ได้ในส่วนของโปรแกรมย่อยนี้เท่านั้น

หากต้องการส่งค่าจากส่วนอื่นๆของโปรแกรมเข้ามาให้ส่วนของโปรแกรมย่อย การส่งค่าทำได้ผ่านตัวแปรรับค่า ตัวแปรรับค่านี้จะใช้เฉพาะในส่วนของโปรแกรมย่อย และไม่มีการเปลี่ยนแปลงค่าของตัวแปรนอกส่วนของโปรแกรมย่อย หากต้องการนำค่าที่คำนวณได้ในส่วนของโปรแกรมย่อยออกไปใช้ที่ส่วนอื่นๆ ต้องประกาศตัวแปรนั้นให้เป็นตัวแปรร่วม หรือคืนค่าที่คำนวณได้ผ่านตัวแปรคืนค่าไป

8.5 ลำดับการทำงานของฟังก์ชัน

การทำงานของโปรแกรมจะเริ่มที่โปรแกรมหลัก และมีการเรียกใช้โปรแกรมย่อยต่างๆ โปรแกรมย่อยเองก็สามารถเรียกโปรแกรมย่อยอื่นๆ รวมถึงเรียกใช้ตัวโปรแกรมย่อยเองได้เช่นเดียวกัน ในการวิเคราะห์การเรียกใช้โปรแกรมย่อย เราแบ่งโปรแกรมย่อยตามลักษณะการทำงานเป็น 2 ส่วน คือ ตัวเรียก (caller) และตัวถูกเรียก (callee) ทั้งตัวเรียกและตัวถูกเรียกสามารถเป็นโปรแกรมย่อยใดๆ ก็ได้ เป็นรหัสต้นฉบับชุดเดียวกันก็ได้ แต่การทำงานจะเป็นไปตามลำดับ คือ เริ่มจากตัวเรียกก่อน เมื่อตัวเรียกนั้นเรียกใช้ตัวถูกเรียก การทำงานจะข้ามไปเริ่มต้นที่จุดเริ่มต้นของตัวถูกเรียก จนกว่าจะเสร็จสิ้นการทำงานในส่วนของตัวถูกเรียกแล้ว จึงจะกลับมาทำงานส่วนที่เหลือของตัวเรียกต่อ

ตัวแปรต่างๆ ที่ประกาศภายในฟังก์ชันนั้น จะใช้ได้ในขอบเขตของการทำงานในการเรียกใช้ครั้งหนึ่งๆ เท่านั้น การเรียกใช้โปรแกรมย่อยจะเสมือนกับมีการสร้างชุดของตัวแปรเฉพาะที่สำหรับโปรแกรมย่อยนั้น

ขึ้นใหม่ โดยไม่เกี่ยวข้องกับตัวแปรเฉพาะที่อื่นๆ ที่เคยประกาศมาก่อนล่วงหน้า และเมื่อการทำงานของตัวถูกเรียกนั้นสิ้นสุด ตัวแปรเฉพาะที่ต่างๆ ของตัวถูกเรียกก็จะหมดอายุตามไปด้วย หากต้องการอ้างถึงค่าต่างๆ ที่มีอยู่ในตัวถูกเรียก ต้องส่งค่านั้นคืนให้ตัวเรียกผ่านการคืนค่าในรูปแบบต่างๆ

ตัวอย่าง 8.2 (ฟังก์ชันแฟกทอเรียลและขั้นตอนการทำงาน). โปรแกรมในรหัสต้นฉบับที่ 8.2 ประกอบด้วยฟังก์ชัน `main` และ `factorial`

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 int nRound;
 5
 6 int factorial(int n)
 7 {
 8     printf("round %d, n = %d\n", nRound, n);
 9     nRound++;
10
11     if (n < 0)
12         return -1;
13     else if (n <= 2)
14         return n;
15     else
16         return n * factorial(n-1);
17 }
18
19 int main()
20 {
21     int n;
22     printf("Enter n: ");
23     scanf("%d", &n);
24
25     nRound = 1;
26
27     printf("%d! = %d\n", n, factorial(n));
28
29     return 0;
30 }
```

รหัสคำสั่งที่ 8.2: โปรแกรมแฟกทอเรียล

ข้อสังเกตที่น่าสนใจในโปรแกรมนี้มี 4 จุด ได้แก่

1. ตัวแปร `nRound` เป็นตัวแปรร่วม ประกาศที่บรรทัดที่ 4 อยู่นอกทั้งสองฟังก์ชัน ทำให้ทั้งสองฟังก์ชันสามารถใช้งานตัวแปรนี้ได้ และการเปลี่ยนแปลงค่าใดๆ ในตัวแปรนี้ด้วยฟังก์ชันหนึ่ง จะส่งผลให้ฟังก์ชันอื่นๆ เห็นค่าที่เปลี่ยนไปตามไปด้วย
   เช่น ใน `main` มีการกำหนดค่า `nRound` ให้เป็น 1 ในบรรทัดที่ 25 มื่อเรียกฟังก์ชัน `factorial` ครั้งแรกในบรรทัดที่ 27 ในฟังก์ชัน `factorial` ก็จะเห็นว่า `nRound` มีค่า 1 ตามไปด้วย
   ส่วนในฟังก์ชัน `factorial` นั้นมีการเพิ่มค่า `nRound` ไปทีละหนึ่งในแต่ละครั้งของการเรียกฟังก์ชันด้วยคำสั่งในบรรทัดที่ 9 การเรียกฟังก์ชันในรอบหลังๆ ก็จะเห็นค่า `nRound` ที่เปลี่ยนไปแล้วเช่นกัน
2. ตัวแปร `n` ประกาศอยู่ในทั้ง `main` และ `factorial` แต่ทั้งสองตัวแปรเก็บแยกจากกันโดยเด็ดขาด ความเชื่อมโยงของทั้งสองตัวแปรมีเฉพาะตอนที่ `main` สำเนาค่าใน `n` ของ `main` ไปให้ `n` ของ `factorial` ผ่านการเรียกใช้ในครั้งแรกเท่านั้น
3. ฟังก์ชันใดๆ สามารถเรียกใช้ตัวมันเองได้ เช่นในฟังก์ชัน `factorial` ก็มีการเรียกใช้ฟังก์ชัน `factorial` ด้วยเช่นกัน แต่การเก็บตัวแปรเฉพาะที่ต่างๆ ของตัวเรียกและตัวถูกเรียกนั้นแยกจากกัน ไม่ว่าชื่อตัวแปรหรือฟังก์ชันจะซ้ำกันหรือไม่ก็ตาม
4. ในรหัสคำสั่งที่ 8.2 เป็นโปรแกรมภาษาซี ประกาศฟังก์ชัน `factorial` และการทำงานไว้ก่อนการเรียกใช้ในฟังก์ชัน `main` จึงไม่จำเป็นต้องเขียนต้นแบบของฟังก์ชันก่อนการเรียกใช้อีก แต่หากสลับตำแหน่งของรหัสต้นฉบับสำหรับฟังก์ชัน `factorial` และฟังก์ชัน `main` ก่อนประกาศฟังก์ชัน `main` ต้องระบุต้นแบบของฟังก์ชัน `factorial` เอาไว้ก่อนฟังก์ชัน `main` ด้วย

□

8.6 คลังโปรแกรม

โปรแกรมย่อยและฟังก์ชันต่างๆ ที่ใช้บ่อยมักจะถูกรวบรวมเก็บไว้ในคลังโปรแกรม (library) เพื่อจัดให้เป็นหมวดหมู่และให้ผู้ที่ต้องการใช้สามารถดึงมาใช้ได้โดยไม่ต้องเขียนใหม่อีก การเรียกใช้ฟังก์ชันในคลังโปรแกรมทำได้โดยระบุชื่อคลังโปรแกรมที่ใช้ เพื่อให้คอมไพเลอร์หรืออินเทอร์พรีเตอร์รู้จักฟังก์ชันที่ต้องการเรียกใช้ และให้ตัวเชื่อมโยงสามารถไปดึงออบเจกต์โค้ดของฟังก์ชันเหล่านั้นมารวมเป็นโปรแกรมที่ทำงานได้

คลังโปรแกรมจึงขึ้นกับภาษาโปรแกรมที่ใช้ รวมถึงขึ้นกับระบบปฏิบัติการที่ใช้ด้วย คลังโปรแกรมที่เป็นที่นิยมนั้นจะสนับสนุนหลายภาษาและระบบปฏิบัติการ เช่น OpenGL (**Open** **G**raphic **L**ibrary) เป็นคลังโปรแกรมสำหรับงานกราฟิก สนับสนุนทั้ง Windows, Linux, OS X และมีส่วนต่อประสานโปรแกรม (API) ในหลากหลายภาษา เช่น ซี จาวา

แบบฝึกหัด

1. กำหนดฟังก์ชัน `longdiv` เป็นฟังก์ชันการหารยาวทศนิยม 1 ตำแหน่ง กำหนดให้ `dividend` เป็นตัวตั้ง และ `divisor` เป็นตัวหาร มีต้นแบบดังนี้

   ```
   float longdiv(float dividend, float divisor);
   ```

   เช่น 10/4 จะได้ผลลัพธ์เป็น 2.5
   หากต้องการหาผลลัพธ์ของ 20/3 จะต้อง**เรียกใช้**ฟังก์ชัน `longdiv` นี้อย่างไร
2. ต้องการเขียนฟังก์ชัน `contains` เพื่อตรวจสอบว่า อักขระอินพุตอยู่ในสายอักขระที่ป้อนเข้ามาหรือไม่ โดยให้ผลลัพธ์เป็นจำนวนเต็ม 1 หากพบ หรือ 0 หากไม่พบ เช่น ตรวจสอบว่าอักขระ 'a' อยู่ในสายอักขระ "contains" หรือไม่ ฟังก์ชันนี้ควรมีต้นแบบเป็นอย่างไร
3. จากแบบฝึกหัด Fibonacci ในบทที่ 7 จงเขียนฟังก์ชันการคำนวณเลขฟีโบนักชีลำดับที่ n เมื่อฟังก์ชันการหาค่าในลำดับฟีโบนักชีคือ

   $$F_n = F_{n-1} + F_{n-2}$$

   กำหนด $F_0 = 0$ และ $F_1 = 1$