

# การเขียนโปรแกรม ภาษา C ขั้นพื้นฐาน

ครอบคลุมเนื้อหาสำคัญ  
ทั้งหมด พร้อมตัวอย่างโค้ด  
จำนวนมาก



เขียนโค้ดภาษา C จากเริ่มต้น  
จนถึงระดับการใช้งานได้จริง



อธิบายเป็นขั้นตอน มีภาพประกอบ  
และสามารถเรียนรู้ได้ด้วยตนเอง



ดาวน์โหลดโค้ดหนังสือเล่มนี้ได้ที่  
<http://www.developerthai.com>

**บัญชา ปะสีลະเตสัง**

ผู้เขียนหนังสือขายดีระดับ Best Seller  
ด้าน Programming



# การเขียนโปรแกรมภาษา C ขั้นพื้นฐาน

โดย บัญชา ปะลีละเตสัง

สงวนลิขสิทธิ์ตามกฎหมาย โดย บัญชา ปะลีละเตสัง © พ.ศ. 2566

ห้ามคัดลอก ลอกเลียน ตัดแปลง ทำซ้ำ จัดพิมพ์ หรือกระทำการอื่นใด โดยวิธีการใดๆ ในรูปแบบใดๆ ไม่ว่าส่วนหนึ่งส่วนใดของหนังสือเล่มนี้ เพื่อเผยแพร่ในลักษณะใดๆ หรือเพื่อวัตถุประสงค์ใดๆ นอกจากจะได้รับอนุญาต

ข้อมูลทางบรรณานุกรมของหอสมุดแห่งชาติ

บัญชา ปะลีละเตสัง.

การเขียนโปรแกรมภาษา C ขั้นพื้นฐาน.-- กรุงเทพฯ : ชีเอ็ดดูเคชั่น, 2566.

392 หน้า.

1. ชี (ภาษาคอมพิวเตอร์).

I. ชื่อเรื่อง.

005.133

Barcode (e-book) : 9786160847518

ผลิตและจัดจำหน่ายโดย



บริษัท ชีเอ็ดดูเคชั่น จำกัด (มหาชน)  
SE-EDUCATION PUBLIC COMPANY LIMITED

เลขที่ 1858/87-90 ถนนเพชรบุรีตัน แขวงบางนาใต้ เขตบางนา กรุงเทพฯ 10260  
โทรศัพท์ 0-2826-8000

[หากมีคำแนะนำหรือติชม ติดต่อที่ [comment@se-ed.com](mailto:comment@se-ed.com)]

# คำนำ

ภาษา C เป็นหนึ่งในภาษาคอมพิวเตอร์ที่มีผู้ใช้งานเป็นจำนวนมากในปัจจุบัน เนื่องจาก ลักษณะโครงสร้างที่ไม่ซับซ้อนจนเกินไป จึงเรียนรู้ได้ไม่ยาก รวมทั้งความเร็วในการประมวลผลและความสามารถที่โดดเด่นในอีกหลาย ๆ ด้าน ดังนั้น สถาบันการศึกษาส่วนใหญ่จึงมักเลือกภาษา C ในการเรียนการสอนสำหรับวิชาการเขียนโปรแกรมระดับเริ่มต้น และแม้ว่าภาษา C จะถูกสร้างมานานมากแล้วก็ตาม แต่ซอฟต์แวร์หลัก ๆ ที่เราใช้งานกันอยู่ในทุกวันนี้ ล้วนถูกสร้างจากพื้นฐานของภาษา C แทนทั้งสิ้น เช่น ระบบปฏิบัติการต่าง ๆ โปรแกรมด้านฐานข้อมูลและธุรกิจอุตสาหกรรม การคำนวณทางวิทยาศาสตร์ Game Engine และอื่น ๆ อีกมากมาย ดังนั้น การเขียนโปรแกรมด้วยภาษา C จึงเป็นสิ่งสำคัญอีกอย่างหนึ่งที่ผู้ต้องการก้าวสู่สายงานด้านการพัฒนาซอฟต์แวร์ระดับมืออาชีพควรต้องเรียนรู้เอาไว้

หนังสือเล่มนี้ได้รวบรวมเนื้อหาที่จำเป็นต้องรู้ในขั้นพื้นฐานทั้งหมด พร้อมตัวอย่างประกอบ เพื่อเสริมความเข้าใจอีกเป็นจำนวนมาก โดยใช้เครื่องมือยอดนิยมอย่าง Visual Studio Code (VS Code) ร่วมกับตัวแปลงภาษา MinGW ซึ่งผู้เริ่มต้นศึกษาการเขียนโปรแกรม สามารถเรียนรู้ ด้วยตนเอง รวมถึงเป็นแนวทางสำหรับศึกษาเพิ่มเติมจากแหล่งข้อมูลอื่น ๆ ต่อไปได้

บัญชา ประสีลະเตสัง

banchar\_pa@yahoo.com  
facebook.com/DeveloperThai



# สารบัญ

## บทที่ 1 การจัดเตรียมเครื่องมือ.....13

การติดตั้ง VS Code.....	13
การติดตั้ง MinGW.....	15
การเซตทำแน่งของตัวแปลภาษา.....	17
การติดตั้ง C/C++ Package สำหรับ VS Code.....	18
การติดตั้ง Code Runner สำหรับ VS Code.....	19
การทดสอบโค้ดภาษา C .....	20
● สร้างโฟลเดอร์สำหรับเก็บโค้ด.....	20
● การสร้างไฟล์และทดสอบโค้ดภาษา C .....	22
การใช้เครื่องมือพื้นฐานของ VS Code.....	25
● หน้าจอเริ่มต้น (Get Started) .....	25
● การเลือกรูปแบบสี (Theme) .....	26
● การปรับเปลี่ยนฟอนต์.....	27
● แนวทางการติดตั้งส่วนเสริมเพิ่มเติม .....	28
การนำโค้ดของหนังสือมาใช้งาน.....	30

## บทที่ 2 การเขียนโค้ดภาษา C ในเบื้องต้น .....31

ประวัติโดยย่อของภาษา C.....	31
ข้อดีและข้อเสียของภาษา C.....	32
คำส่วนของภาษา C.....	34
ทบทวนการสร้างไฟล์ภาษา C .....	35

องค์ประกอบของการเขียนโค้ดภาษา C ในเบื้องต้น.....	35
● คำสั่ง #include.....	35
● ฟังก์ชัน main .....	37
● การกำหนดล็อกคำสั่งด้วยวงเล็บ {} .....	38
● การกำหนดจุดลิ้นสุดคำสั่งด้วยเครื่องหมาย ;.....	39
● การเขียนคำอธิบายโค้ด (Comment) .....	40
การเขียนคำอธิบายแบบบล็อก (Block Comment).....	40
การเขียนคำอธิบายแบบบรรทัด (Line Comment).....	40
การแสดงข้อความในเบื้องต้น .....	41
● การแสดงข้อความด้วยฟังก์ชัน puts().....	41
● การแสดงข้อความด้วยฟังก์ชัน printf() .....	42
● อักขระพิเศษสำหรับบรรทัดใหม่ (\n).....	44
<b>บทที่ 3 เชิดข้อมูลและตัวแปร.....</b>	<b>47</b>
ชนิดข้อมูลในภาษา C.....	47
● ข้อมูลประเภทเลขจำนวนเต็ม .....	48
● ข้อมูลประเภทเลขทศนิยม.....	49
● ข้อมูลประเภทอักขระ .....	49
ลักษณะของตัวแปรในภาษา C .....	51
การประกาศตัวแปร .....	52
การกำหนดค่าให้กับตัวแปร .....	53
พื้นฐานการแปลงชนิดข้อมูล.....	55
● Implicit Conversion .....	56
● Explicit Conversion .....	56
ค่าคงที่ .....	57
● การกำหนดค่าคงที่ด้วยคำสั่ง const.....	58
● การกำหนดค่าคงที่ด้วยคำสั่ง define.....	58
<b>บทที่ 4 สตริง การรับและแสดงผลข้อมูล .....</b>	<b>61</b>
อักขระและสตริง .....	61
การกำหนดและแก้ไขค่าของตัวแปรสตริง.....	65

● การใช้ฟังก์ชัน strcpy() .....	65
● การใช้ฟังก์ชัน strcat() .....	67
อักขระพิเศษ (Escape Sequence) .....	68
การจัดรูปแบบสตริงสำหรับแสดงผล .....	69
● การจัดรูปแบบในเบื้องต้นสำหรับ printf() .....	70
● การกำหนดความกว้างของสตริง .....	73
● การอ่านค่าสตริงที่จัดรูปแบบด้วย sprintf() .....	76
● การจัดรูปแบบตัวเลขโดยมี , คั่นหลักพัน .....	77
การรับข้อมูลทางคีย์บอร์ด .....	78
● การรับข้อมูลชนิดอักขระด้วย getchar() .....	79
● การรับข้อมูลแบบสตริงด้วย gets() .....	80
● การรับข้อมูลหลายชนิดด้วย scanf() .....	81
การแปลงชนิดข้อมูลระหว่างสตริงและตัวเลข .....	86
● การแปลงสตริงเป็นตัวเลขด้วยฟังก์ชัน atoi() .....	86
● การแปลงตัวเลขเป็นสตริงด้วยฟังก์ชัน sprintf() .....	88
<b>บทที่ 5 ตัวเลขและการคำนวณ .....</b>	<b>91</b>
เครื่องหมายสำหรับการทำหนดค่า .....	91
เครื่องหมายสำหรับการคำนวณทางคณิตศาสตร์ .....	93
เครื่องหมายสำหรับการทำหนนและกำหนดค่า .....	99
เครื่องหมายสำหรับเพิ่มและลดค่า .....	103
ลำดับการประมวลผลของเครื่องหมาย .....	105
ฟังก์ชันทางด้านคณิตศาสตร์ .....	107
การสร้างเลขสุ่ม .....	109
การแปลงสูตรคณิตศาสตร์เป็นโค้ดภาษา C .....	114
<b>บทที่ 6 การเปรียบเทียบและการกำหนดเงื่อนไข .....</b>	<b>119</b>
ข้อมูลชนิดบูลีน (_Bool และ bool) .....	119
เครื่องหมายสำหรับการเปรียบเทียบ .....	121
การเปรียบเทียบสตริง .....	124
ลักษณะพื้นฐานของคำสั่ง if .....	124

การกำหนดเงื่อนไขด้วยเครื่องหมายเปรียบเทียบ.....	127
การใช้คำสั่ง if-else .....	131
การใช้คำสั่ง if-<else if> .....	133
การใช้คำสั่ง if-<else if>-else .....	137
การเปรียบเทียบทางตระกูล .....	139
การกำหนดหลายเงื่อนไขด้วย Logical Operator.....	140
ตัวดำเนินการแบบ Ternary .....	145
การใช้คำสั่ง switch-case.....	146
การตรวจสอบอักขระด้วยฟังก์ชันที่คืนค่าแบบบูลีน .....	151
<b>บทที่ 7 การกำหนดแบบวนรอบ.....</b>	<b>153</b>
การใช้ลูปแบบ for ขั้นพื้นฐาน.....	153
การกำหนดลูป for ซ้อนกัน .....	157
การใช้คำสั่ง break.....	160
การใช้คำสั่ง continue.....	161
การกำหนดลูป for แบบไม่รู้จบ (Infinite Loop).....	161
ลักษณะที่ควรรู้เพิ่มเติมเกี่ยวกับลูป for .....	162
การใช้ลูปแบบ while.....	165
การใช้ลูปแบบ do-while .....	169
<b>บทที่ 8 รวมตัวอย่างโคดเพิ่มเติม ชุดที่ 1 .....</b>	<b>175</b>
<b>บทที่ 9 การสร้างและใช้งานฟังก์ชัน .....</b>	<b>201</b>
ลักษณะของฟังก์ชัน.....	201
พารามิเตอร์และอาร์กิวเมนต์ .....	203
ฟังก์ชันแบบไม่ส่งค่ากลับ.....	204
● การสร้างฟังก์ชันแบบไม่ส่งค่ากลับ .....	205
● ฟังก์ชัน main() และตำแหน่งการเรียกฟังก์ชันที่สร้างเอง .....	206
● การเรียกใช้ฟังก์ชันแบบไม่ส่งค่ากลับ .....	207
ฟังก์ชันแบบส่งค่ากลับ .....	210
● การสร้างฟังก์ชันแบบส่งค่ากลับ.....	210
● การเรียกฟังก์ชันแบบส่งค่ากลับ .....	212

ตัวแปรแบบ global และ local.....	216
ตัวแปรแบบ static.....	218
การเรียกฟังก์ชันแบบ Recursion.....	219
<b>บทที่ 10 อาร์เรย์และสตริง (เพิ่มเติม).....</b>	<b>225</b>
ลักษณะพื้นฐานของอาร์เรย์.....	225
การสร้างและกำหนดค่าสมาชิกของอาร์เรย์.....	227
การเข้าถึงสมาชิกของอาร์เรย์ .....	230
● การอ่านค่าสมาชิกของอาร์เรย์.....	230
● การหาขนาดของอาร์เรย์ .....	234
การใช้อาร์เรย์เป็นพารามิเตอร์ของฟังก์ชัน .....	238
อาร์เรย์แบบ 2 มิติ.....	240
● ลักษณะของอาร์เรย์ 2 มิติ.....	240
● การหาขนาดของอาร์เรย์ 2 มิติ .....	244
● การเข้าถึงสมาชิกในอาร์เรย์โดยใช้ลูป.....	244
อาร์เรย์ของสตริง .....	245
<b>บทที่ 11 การใช้ดำเนินการด้วยพอยน์เตอร์ .....</b>	<b>253</b>
หลักการเบื้องต้นของพอยน์เตอร์.....	253
การประกาศตัวแปรประเภทพอยน์เตอร์.....	255
การกำหนดดำเนินการให้พอยน์เตอร์ชี้ไป.....	256
การอ่านและกำหนดค่าของดำเนินการที่พอยน์เตอร์ชี้ไป .....	258
การใช้พอยน์เตอร์ร่วมกับฟังก์ชัน .....	260
● การรับพารามิเตอร์ในแบบพอยน์เตอร์.....	260
● การส่งค่าแบบพอยน์เตอร์กลับจากฟังก์ชัน.....	262
การใช้พอยน์เตอร์ร่วมกับอาร์เรย์.....	267
● การชี้ดำเนินการของสมาชิกด้วยพอยน์เตอร์.....	267
● การเลื่อนดำเนินการของพอยน์เตอร์.....	268
● การใช้พอยน์เตอร์ร่วมกับเครื่องหมาย ++ และ -- .....	269
การใช้พอยน์เตอร์ร่วมกับสตริง .....	272
● การใช้พอยน์เตอร์แทนตัวแปรแบบสตริง .....	273
● การใช้พอยน์เตอร์ร่วมกับอาร์เรย์ของสตริงในแบบ 2 มิติ.....	274

บทที่ 12 รวมตัวอย่างโค้ดเพิ่มเติม ชุดที่ 2 ..... 279

บทที่ 13 สตรัคเจอร์ ยูเนี่ยน และอีนนัม ..... 313

โครงสร้างข้อมูลแบบสตรัคเจอร์ (Structure).....	313
การกำหนดและอ่านข้อมูลจากสตรัคเจอร์ .....	315
การสร้างอาร์เรย์ของสตรัคเจอร์ .....	320
การใช้พอยน์เตอร์ร่วมกับสตรัคเจอร์ .....	324
การใช้สตรัคเจอร์ร่วมกับฟังก์ชัน.....	324
● การใช้สตรัคเจอร์เพื่อเป็นพารามิเตอร์ .....	325
● การส่งค่าแบบสตรัคเจอร์กลับจากฟังก์ชัน.....	325
การสร้างสตรัคเจอร์ซ้อนกัน.....	327
โครงสร้างข้อมูลแบบยูเนี่ยน (Union) .....	329
ชุดข้อมูลแบบอีนนัม (Enum) .....	332

บทที่ 14 การเขียนและอ่านไฟล์ ..... 335

การเปิดและปิดไฟล์ .....	335
● การระบุตำแหน่งไฟล์ .....	335
● โหมดในการเปิดไฟล์ .....	336
● การเปิดไฟล์.....	337
● การปิดไฟล์ .....	339
การเขียนไฟล์.....	339
● ฟังก์ชัน fprintf() .....	340
● ฟังก์ชัน fputs() .....	342
● ฟังก์ชัน fputc() .....	342
การอ่านไฟล์ .....	345
● ฟังก์ชัน fgets() .....	345
● ฟังก์ชัน fgetc() .....	348
● การเลื่อนตำแหน่งของพอยน์เตอร์.....	350
● ฟังก์ชัน fscanf() .....	351
เก็บข้อมูลสตรัคเจอร์ด้วยไฟล์ใบนารี .....	355
● โหมดในการเปิดไฟล์ใบนารี.....	355
● การจัดแบ่งข้อมูลแบบ Record.....	356

● การเขียนข้อมูลสตรัคเจอร์ลงในไฟล์ใบหน้า	356
● การอ่านข้อมูลสตรัคเจอร์จากไฟล์ใบหน้า	358
● การเข้าถึงข้อมูลแบบเจาะจง	361
<b>บทที่ 15 พรีโพรเซสเซอร์และมาโคร</b>	<b>365</b>
เกี่ยวกับพรีโพรเซสเซอร์และไดเรกทีฟ	365
กลุ่มไดเรกทีฟ Macro Definition	366
ค่าคงที่ในกลุ่ม Predefined Macro	370
กลุ่มไดเรกทีฟ Conditional Compilation	371
<b>บทที่ 16 รวมตัวอย่างโค้ดเพิ่มเติม ชุดที่ 3</b>	<b>373</b>





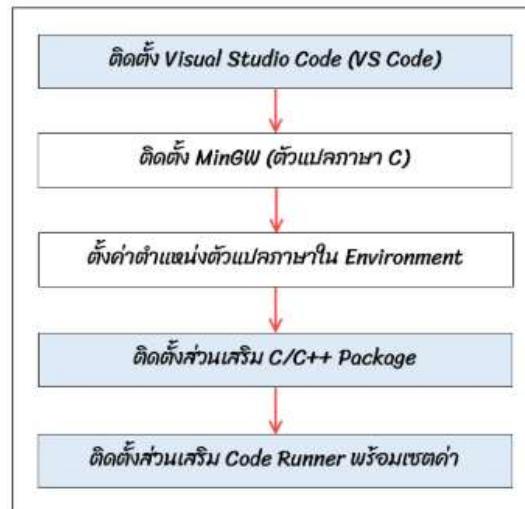


# การจัดเตรียมเครื่องมือ

ก อนที่เราจะเข้าสู่ขั้นตอนการเขียนโค้ดภาษา C ตามที่จะกล่าวถึงในบทต่อ ๆ ไป สิ่งที่เราจำเป็นต้องจัดเตรียมเอาไว้ล่วงหน้าก็คือ เครื่องมือในการเขียนโค้ดและทดสอบเพื่อคุณลักษณะ ซึ่งแม้จะมีตัวเลือกอยู่มากหลาย แต่ในหนังสือเล่มนี้ จะเลือกมานำเสนอเฉพาะเครื่องมือที่ได้รับความนิยมสูงสุดในปัจจุบัน โดยเครื่องมือที่เลือกนี้ ไม่ใช่เฉพาะกับการเขียนโค้ดภาษา C เท่านั้น แต่ยังสามารถนำไปใช้สำหรับการเขียนโค้ดภาษาคอมพิวเตอร์อื่น ๆ ได้เกือบทั้งหมด จึงถือเป็นข้อได้เปรียบอย่างหนึ่งเมื่อเรามีความต้องการเขียนโปรแกรมภาษาอื่น ๆ เพิ่มเติมในอนาคต

## การติดตั้ง VS Code

การเขียนโค้ดภาษา C นั้น แม้เราจะมีเครื่องมือให้เลือกใช้มากมาย แต่ในที่นี้ผู้เขียนจะแนะนำวิธีการที่ได้รับความนิยมสูงสุดในปัจจุบัน เพียงวิธีเดียว นั่นก็คือ การใช้งานผ่าน Visual Studio Code (เรานิยมเรียกว่า VS Code) อย่างไรก็ตาม เนื่องจาก VC Code ไม่ได้ถูกสร้างมาสำหรับการเขียนโค้ดภาษา C โดยตรง แต่สามารถใช้กับภาษาคอมพิวเตอร์ทั่ว ๆ ไป ได้เกือบทั้งหมด ดังนั้น เราจึงต้องติดตั้งเครื่องมืออื่น ๆ เพิ่มเติมอีก โดยขั้นตอนที่จำเป็นต้องทำสามารถสรุปได้ดังนี้

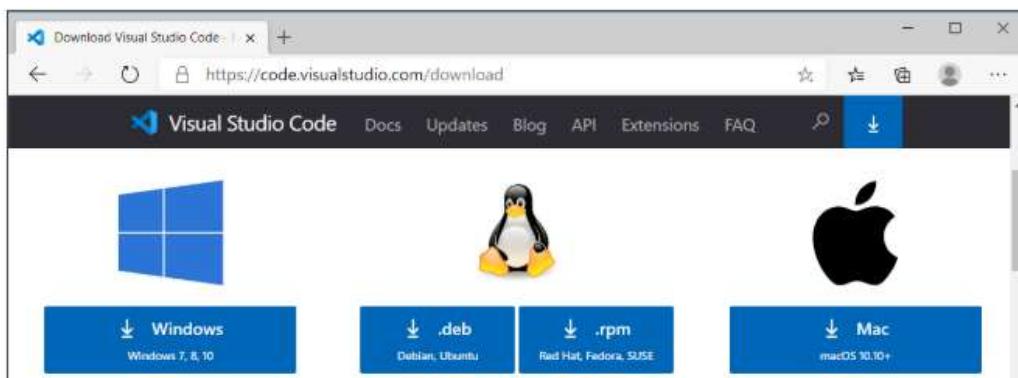


1. ติดตั้ง **Visual Studio Code (VS Code)** เพื่อใช้เป็นเครื่องมือหลักในการเขียนและแสดงผลลัพธ์รวมถึงจัดการไฟล์ของโค้ด
2. ติดตั้ง **MinGW** เพื่อเป็นตัวแปลงภาษา (Compiler) ของภาษา C
3. เช็คตัวแหน่งที่ติดตั้ง MinGW ลงใน **Environment Variables** ของระบบ เพื่อให้โปรแกรมอื่นๆ รู้ตัวแหน่งของตัวแปลงภาษา
4. ติดตั้งส่วนเสริม **C/C++ Package** ลงใน VS Code เพื่อเป็นตัวช่วยในการเขียนโค้ดภาษา C
5. ติดตั้งส่วนเสริม **Code Runner** ลงใน VS Code เพื่อเชื่อมโยงระหว่าง VS Code กับ MinGW ซึ่งจะช่วยให้เราสามารถรันทดสอบโค้ดได้ง่ายขึ้น รวมทั้งตั้งค่าการแสดงผลลัพธ์อีกด้วย

ทั้งหมดที่กล่าวมา จะแยกกล่าวรายละเอียดเป็นหัวข้อต่างๆ ในลำดับต่อไป แต่ในหัวข้อนี้ จะกล่าวถึงขั้นตอนแรกนั่นก็คือ การติดตั้ง VS Code ซึ่งเป็นโปรแกรมประเภท IDE (Integrated Development Environment) ที่ Microsoft สร้างขึ้นมาสำหรับใช้ในการเขียนโค้ดทั่วๆ ไป และได้รับความนิยมสูงสุดในปัจจุบัน เนื่องจากมีลักษณะที่โดดเด่นหลายประการ เช่น ให้ใช้งานพร้อมรับการเขียนโค้ดได้หลายภาษา และที่สำคัญคือ มีส่วนเสริมการทำงานให้เลือกใช้มากมาย

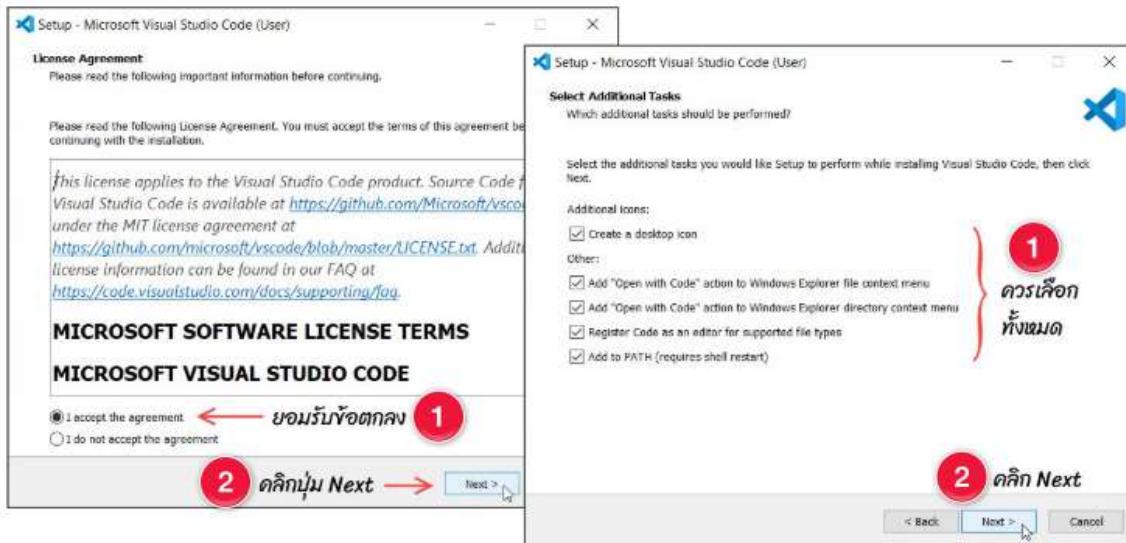
สำหรับการติดตั้ง VS Code ลงในเครื่องที่เราจะใช้งาน มีขั้นตอนโดยสังเขปคือ

1. สามารถดาวน์โหลดชุดติดตั้งได้ที่ <https://code.visualstudio.com/download>

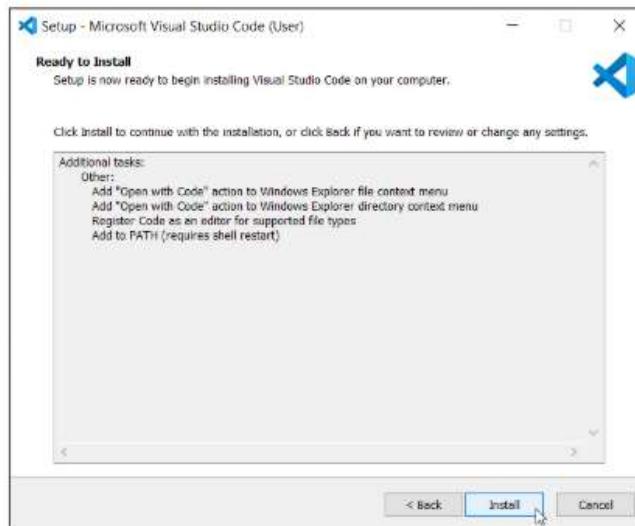


2. เลือกดาวน์โหลดให้ตรงกับระบบปฏิบัติการที่เรากำลังใช้งาน (ในที่นี้จะกล่าวถึงเฉพาะระบบ Windows เท่านั้น)
3. หลังจากดาวน์โหลดเสร็จ ให้ดับเบิลคลิกไฟล์ติดตั้งที่ได้มา และในหน้าจอตัดไป ก็ ยอมรับเงื่อนไขแล้วคลิกปุ่ม Next

#### 4. ขั้นตอนต่อไป จะมีอปชันให้เลือก ก็แนะนำให้เลือกทั้งหมด และคลิกปุ่ม Next



#### 5. ต่อจากนั้น คลิกปุ่ม Install เพื่อเริ่มการติดตั้งไฟล์ จากนั้นกรอจนกว่าจะแล้วเสร็จ



## การติดตั้ง MinGW

การที่ภาษาคอมพิวเตอร์จะทำงานได้นั้น ต้องอาศัยตัวแปลภาษา (Compiler) ซึ่งลิ๊งนี้ไม่ได้ถูกติดตั้งมาพร้อมกับ VS Code โดยเราต้องติดตั้งเพิ่มเติมลงไปต่างหาก ซึ่งตัวแปลภาษา C มีผู้สร้างให้เลือกใช้มากมาย สำหรับในที่นี้จะใช้ตัวแปลภาษาของ GCC (GNU Compiler Collection) ที่ได้รับความนิยมสูงสุด แต่จะใช้ในเวอร์ชัน MinGW (Minimalist GCC for Windows) ซึ่งเป็นการลดขนาดของตัวแปลภาษาง่ายๆ สำหรับใช้งานบนระบบ Windows โดยตรง ทั้งนี้ MinGW มีทั้ง

แบบติดตั้งด้วยไฟล์ Installer และแบบคัดลอกไฟล์มาไว้ในเครื่องก็ใช้ได้เลย โดยแบบ Installer นั้นจะมีขั้นตอนการติดตั้งปลิกย่ออย่างชัดเจน ไม่ต้องเสียเวลาอ่านรายละเอียด ก็สามารถติดตั้งได้โดยอัตโนมัติ แต่แบบคัดลอกไฟล์ อาจจะต้องเสียเวลาอ่านรายละเอียดและตั้งค่าต่างๆ ให้ถูกต้อง แต่ก็สามารถทำได้โดยไม่ต้องมีความรู้มาก่อน

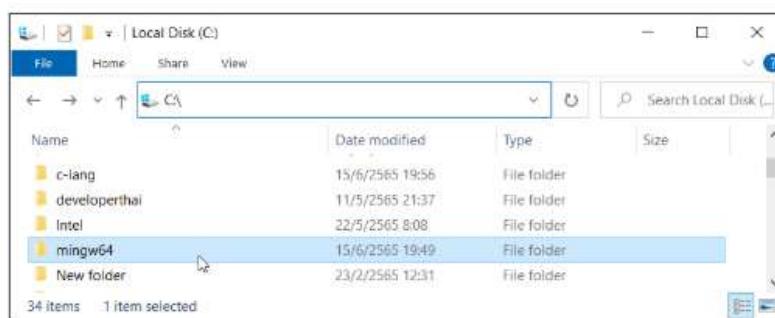
### 1. สามารถดาวน์โหลด MinGW ได้ที่

<https://sourceforge.net/projects/mingw-w64/files/>

แล้วเลือนลงมาคลิกที่ลิงก์ **x86\_64-win32-sjlj** ดังภาพ (ชื่อไฟล์อาจถูกเปลี่ยนแปลงไปเรื่อยๆ แต่ควรเลือกไฟล์ชื่อเดียวกันด้วย x86\_64-win32 อันแรกสุด ซึ่งตามปกติ จะเป็นเวอร์ชันล่าสุด)



### 2. หลังจากดาวน์โหลดเสร็จ ปักติจะได้ไฟล์ที่ถูกบีบอัดเอาไว้ ก็ให้ขยายไฟล์ดังกล่าว ซึ่งหลังการขยายจะได้โฟลเดอร์ชื่อ mingw64 พร้อมกับไฟล์จำนวนมากในโฟลเดอร์นั้น ก็ให้เราคัดลอกโฟลเดอร์ mingw64 ไปเก็บไว้ยังตำแหน่งที่เราอ้างอิงได้สะดวก ซึ่งโดยทั่วไป เรา尼ยมเก็บไว้ที่ไดรฟ์ C ดังนั้น ตำแหน่งของมันจะเป็น C:\mingw64 (หรือจะขยายไฟล์มาไว้ที่นี่โดยตรงเลยก็ได้)

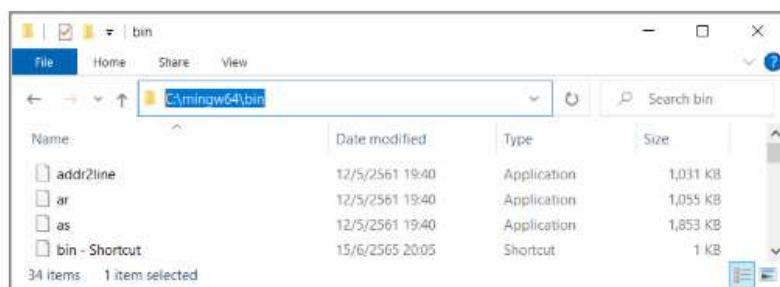


วิธีการตามที่ผู้เขียนเลือกใช้ดังที่กล่าวมานี้ เราเพียงแค่คัดลอกไฟล์มา ก็ใช้งานได้แล้ว แต่อย่าลืมตำแหน่งที่เราเก็บไฟล์เอาไว้คือ **C:\mingw64** ซึ่งต้องใช้อ้างอิงในหัวข้อต่อไป

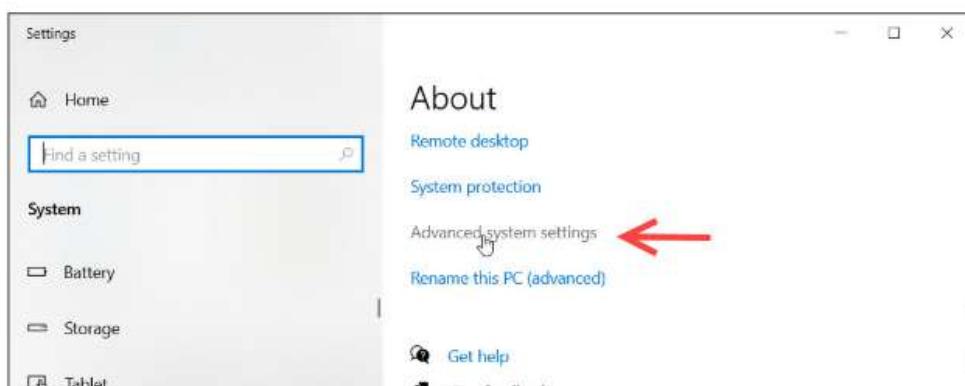
## การเซตต่าแห่งของตัวแปลภาษา

แม้เราจะนำชุดไฟล์ของ MinGW มาจัดเก็บในเครื่องแล้ว แต่ระบบหรือโปรแกรมอื่นๆ ก็ยังไม่รู้จักตำแหน่งของมัน ดังนั้น เราต้องเซตต่าแห่งของ MinGW ลงใน Environment ของระบบ เพื่อให้โปรแกรมอื่นๆ รู้จักตำแหน่งของมัน และเข้าถึงได้ โดยมีขั้นตอนดังนี้

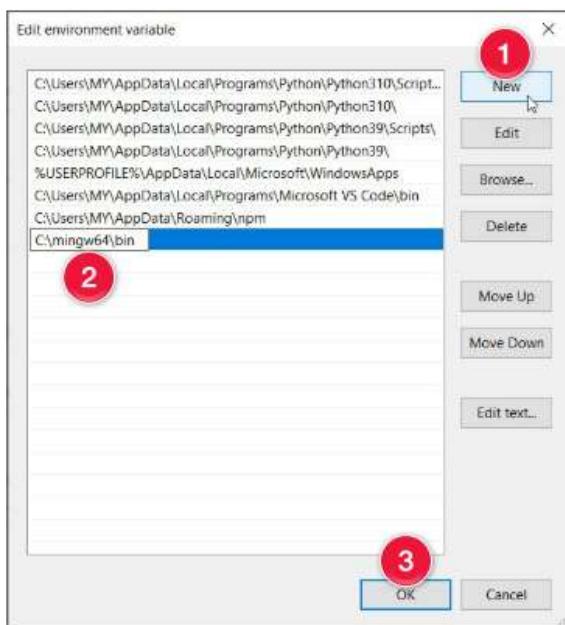
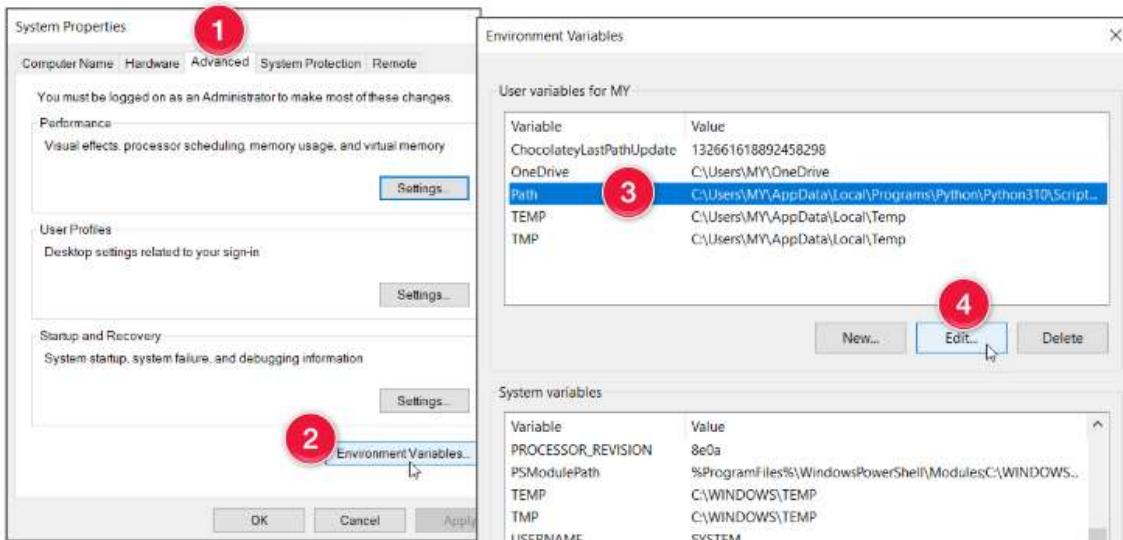
1. จากขั้นตอนการติดตั้งในหัวข้อที่แล้ว ผู้เขียนได้บอกไว้ว่า เราต้องจำไว้ด้วยว่าติดตั้ง MinGW เอาไว้ที่ใด ซึ่งตามปกติหากใช้ตามค่าที่ผู้เขียนแนะนำ ตำแหน่งของตัวแปลภาษาที่จะอยู่ที่ C:\mingw64\bin ถ้าไม่แน่ใจ ให้เปิดไปตรวจสอบอีกครั้งก็ได้



2. ที่ระบบ Windows ให้คลิกขวาที่ปุ่ม Start และเลือก **System** (หรือคลิกขวาที่ไอคอน My Computer หรือ This PC จากนั้นเลือกเมนู Properties หรือใช้วิธีอื่นๆ นอกเหนือจากนี้ก็ได้) และในขั้นตอนถัดไปให้คลิกที่ลิงก์หรือเมนู **Advanced system settings** (ระบบปฏิบัติการแต่ละเวอร์ชันอาจจัดวางตำแหน่งที่แตกต่างกัน)



3. ในขั้นตอนถัดไป เมื่อปรากฏหน้าจอ System Properties ให้เลือกแท็บ **Advanced** จากนั้นคลิกที่ปุ่ม **Environment Variables...**
4. เมื่อปรากฏหน้าจอ Environment Variables ให้คลิกที่ตัวแปร (Variable) **Path** จากนั้นคลิกปุ่ม **Edit...**



5. เมื่อปรากฏหน้าจอ Edit environment variable ให้คลิกปุ่ม **New** และใส่ตำแหน่งของ MinGW ลงไปตามที่ได้จัดเก็บไฟล์เอาไว้ ซึ่งในกรณีของผู้เขียนคือ C:\mingw64\bin ทั้งนี้หากผู้อ่านจัดเก็บที่ตำแหน่งอื่น ก็ให้ระบุไปตามนั้น เสร็จแล้วคลิกปุ่ม **OK**

6. เมื่อย้อนกลับมาอยังหน้าจอในขั้นตอนก่อนนี้ ก็ให้คลิกปุ่ม **OK** ทั้งหมด ก็ถือเป็นอันเสร็จลิ้นขั้นตอนการเซตค่า

## การติดตั้ง C/C++ Package สำหรับ VS Code

จากที่ได้กล่าวไปบ้างแล้วว่า VS Code เป็นเครื่องมือสำหรับเขียนโค้ดทั่วๆ ไป ที่ไม่เจาะจงว่าจะใช้กับภาษาคอมพิวเตอร์อันใดอันหนึ่ง ดังนั้น หากเราจะใช้ในการเขียนโค้ดภาษาใด ก็อาจต้องติดตั้งส่วนเสริม (Extension) เพื่อเป็นตัวช่วยสำหรับใช้งานร่วมกับภาษาหนึ่งๆ สำหรับในกรณีของภาษา C ก็ให้เราติดตั้งส่วนเสริมที่ชื่อ C/C++ Package ที่พัฒนาโดยทีมงานของ Microsoft โดยตรง ดังขั้นตอนต่อไปนี้

1. เปิดเข้าสู่ VS Code แล้วคลิกที่ไอคอน Extensions ที่ແນບด้านซ้ายมือ (การติดตั้งส่วนเสริมต้องเชื่อมต่ออินเทอร์เน็ต)
2. ในช่องค้นหา ให้พิมพ์คำว่า **C/C++** ลงไป
3. หลังจากปรากฏรายชื่oS่วนเสริม ซึ่งมีอยู่ค่อนข้างมาก แต่อันที่แนะนำให้ติดตั้งโดยคลิกปุ่ม Install ตรงส่วนเสริมอันนั้น (หมายเลข 3 ในภาพ) คือ
  - 1) **C/C++**
  - 2) **C/C++ Extension Pack**

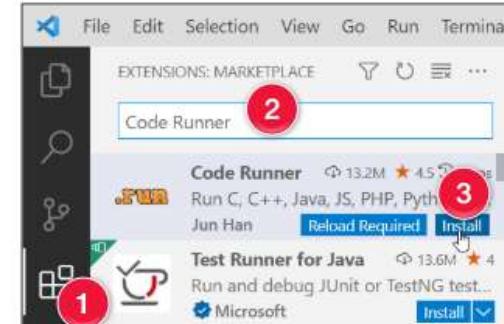


ส่วนเสริม C/C++ เราติดตั้งดังขั้นตอนที่ผ่านมา เป็นเพียงตัวช่วยในการเขียนโค้ดเท่านั้น แต่หากเราต้องการความสะดวกสบายในการทดสอบโค้ดที่เขียน ก็อาจต้องติดตั้งส่วนเสริมอื่นๆ เพิ่มลงไบอิก ตามที่จะกล่าวถึงในหัวข้อต่อไป

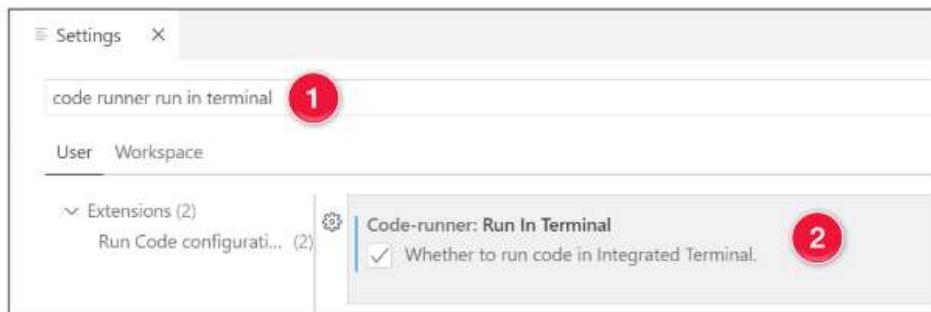
## การติดตั้ง Code Runner สำหรับ VS Code

การติดตั้ง VS Code และ MinGW รวมทั้งส่วนเสริมของ C/C++ ดังขั้นตอนที่ได้กล่าวมา แม้จะเพียงพอที่จะเขียนโค้ดภาษา C บน VS Code ได้แล้ว แต่ก็ยังมีความยุ่งยากในการรันทดสอบ เพราะหากต้องการดูผลลัพธ์ของโค้ดที่เขียน เราต้องพิมพ์คำสั่งลงเบราว์เซอร์ ดังนั้น เพื่อลดความยุ่งยากดังกล่าว เราอาจติดตั้งส่วนเสริมของ VS Code ที่ชื่อ Code Runner เป็นตัวช่วยในขั้นตอนการรันทดสอบเพื่อดูผลลัพธ์ ดังขั้นตอนต่อไปนี้

1. เปิดเข้าสู่ VS Code แล้วคลิกที่ไอคอน Extensions ที่ແນບด้านซ้ายมือ (ต้องเชื่อมต่ออินเทอร์เน็ต)
2. ในช่องค้นหา ให้พิมพ์คำว่า **Code Runner** ลงไป
3. หลังจากปรากฏรายชื่oS่วนเสริม Code Runner ก็ให้คลิกปุ่ม Install
4. หลังติดตั้งเสร็จ ให้ปิดโปรแกรม VS Code แล้วค่อยเปิดขึ้นมาใหม่



5. ต่อไปเราจะตั้งค่า เพื่อให้ Code Runner นำผลลัพธ์ไปแสดงที่ Terminal ของ VS Code โดยเลือกเมนู **File > Preferences > Settings**
6. การค้นหาตัวเลือกเป้าหมาย วิธีที่สะดวกรวดเร็วที่สุดคือ เมื่อเปิดเข้าสู่แท็บหรือหน้าจอ Settings ให้พิมพ์ค้นหาด้วยคำว่า "**code runner run in terminal**" ก็จะปรากฏตัวเลือกเป้าหมายคือ **Code-runner: Run In Terminal** (รายการนี้จะปรากฏเมื่อติดตั้ง Code Runner เข้าไว้แล้วเท่านั้น) ต่อไปเราจะเลือก (เช็ค) ที่รายการนั้น ดังภาพ



7. หลังจากทำตามขั้นตอนดังที่กล่าวมาจนครบแล้ว [ต้องบันทึกการเปลี่ยนแปลงเสมอ](#) เช่น เลือกเมนู **File > Save** หรือกด **<Ctrl + S>** จากนั้นก็สามารถปิดแท็บ Settings ได้เลย และถือเป็นอันเสร็จลิ้นขั้นตอนการตั้งค่า

การเลือกที่ **Run In Terminal** ตามที่เราเซตค่าในขั้นตอนดังกล่าวมานี้ ก็เพื่อให้ Code Runner แสดงผลลัพธ์และรับข้อมูลจากคีย์บอร์ดผ่านทาง Terminal ของ VS Code โดยตรงนั่นเอง

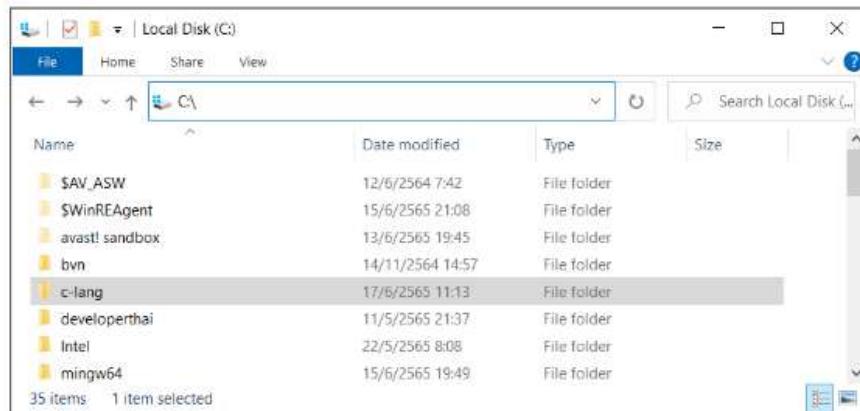
## การทดสอบโค้ดภาษา C

หลังจากที่เราได้จัดเตรียมเครื่องมือที่ต้องใช้สำหรับการทดสอบภาษา C จนครบทั้งหมดแล้ว ดังหัวข้อที่ผ่านๆ มา ขั้นตอนต่อไปเราจะทดสอบเครื่องมือเหล่านั้นว่าใช้งานร่วมกันได้จริงหรือไม่ โดยมีแนวทางดังต่อไปนี้

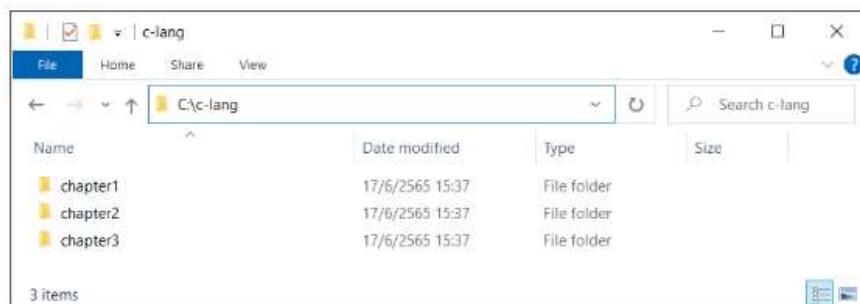
### สร้างโฟลเดอร์สำหรับเก็บโค้ด

ตามหลักการที่ถูกต้องของ VS Code นั้น เราควรสร้างโฟลเดอร์สำหรับจัดเก็บไฟล์ของโค้ดให้เป็นสัดส่วน เพราะการดำเนินการบางอย่างเราต้องทำกับโฟลเดอร์ที่บรรจุไฟล์เท่านั้น ซึ่งแนวทางที่ผู้เขียนจะเลือกใช้ประกอบในหนังสือเล่มนี้คือ

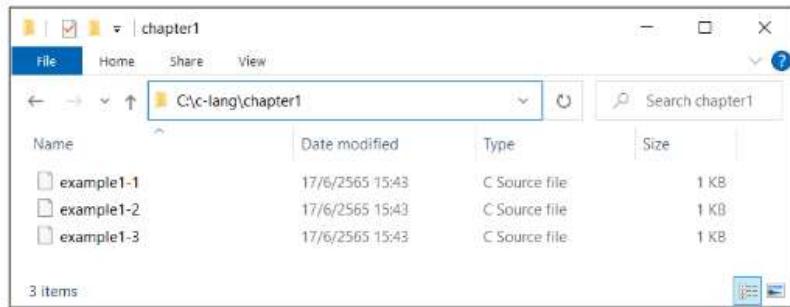
- กำหนดโฟลเดอร์ชื่อ c-lang สำหรับเก็บโค้ดรวมของทุกบท โดยโฟลเดอร์นี้จะสร้างไว้ที่ C:\c-lang



- เนื่องจากในแต่ละบท จะมีไฟล์อยู่ๆ ของแต่ละตัวอย่างเป็นจำนวนมาก ดังนั้น เพื่อให้ง่ายต่อการเข้าถึงไฟล์เป้าหมาย เราจะสร้างโฟลเดอร์ย่อยๆ ขึ้นไว้ใน C:\c-lang เพื่อจัดเก็บโค้ดของแต่ละบท เช่น
  - โฟลเดอร์ chapter1 สำหรับเก็บโค้ดของบทที่ 1 จะอยู่ที่ C:\c-lang\chapter1\
  - โฟลเดอร์ chapter2 สำหรับเก็บโค้ดของบทที่ 2 จะอยู่ที่ C:\c-lang\chapter2\
  - โฟลเดอร์ chapter3 สำหรับเก็บโค้ดของบทที่ 3 จะอยู่ที่ C:\c-lang\chapter3\
  - โฟลเดอร์อื่นๆ ก็ใช้หลักการเดียวกัน ...

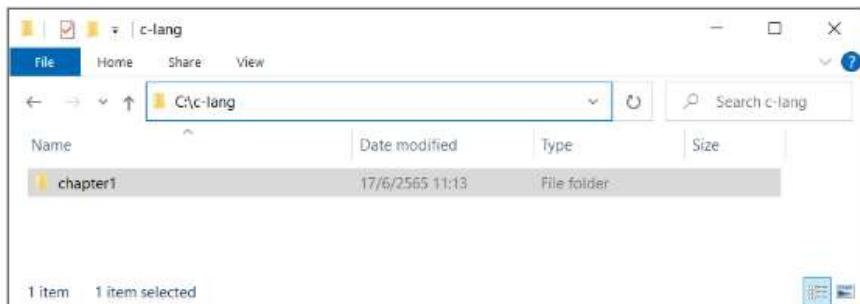


- ในแต่ละบท จะสร้างไฟล์ตามชื่อบท-ลำดับตัวอย่าง เช่น
  - โค้ดของตัวอย่าง 1-1 จะอยู่ที่ไฟล์ C:\c-lang\chapter1\example1-1.c
  - โค้ดของตัวอย่าง 1-2 จะอยู่ที่ไฟล์ C:\c-lang\chapter1\example1-2.c
  - โค้ดของตัวอย่าง 2-1 จะอยู่ที่ไฟล์ C:\c-lang\chapter2\example2-1.c
  - โค้ดของตัวอย่าง 3-5 จะอยู่ที่ไฟล์ C:\c-lang\chapter3\example3-5.c
  - โค้ดของตัวอย่างอื่นๆ ก็ใช้หลักการเดียวกัน ...



หลังจากที่เราทราบหลักการโดยลังเขปแล้ว ต่อไปเราจะลองทดสอบการใช้งาน โดยในขั้นตอนแรก ถ้าเรายังไม่มีโฟลเดอร์รวม ให้สร้างขึ้นมาก่อน ดังนี้

1. เปิด File Explorer ของระบบ Windows และไปที่ C:\
2. สร้างโฟลเดอร์ใหม่ชื่อ c-lang (เข้น คลิกขวาแล้วเลือก New > Folder)
3. เปิดเข้าไปในโฟลเดอร์ c-lang และสร้างโฟลเดอร์ชื่อ chapter1 สำหรับเก็บไฟล์ของบทที่ 1 (ขั้นตอนนี้ อาจไปสร้างใน VS Code ก็ได้)



## การสร้างไฟล์และทดสอบโค้ดภาษา C

หลังจากที่เราสร้างโฟลเดอร์สำหรับจัดเก็บโค้ดเอาไว้แล้ว ดังหัวข้อที่ผ่านมา ต่อไปเราจะสร้างไฟล์ภาษา C เพื่อทดสอบการทำงาน ดังนี้

1. เปิดเข้าสู่ VS Code
2. คลิกเมนู File > Open Folder หรือเลือกจากลิงก์ Open Folder บนหน้าจอ Get Started ก็ได้ จากนั้นเลือกโฟลเดอร์ของบทที่ 1 ตามที่เราได้สร้างเอาไว้ นั่นคือ C:\c-lang\chapter1

3. คลิกที่ไอคอน New File ดังภาพ แล้วกำหนดชื่อไฟล์ โดยให้มีส่วนขยายหรือนามสกุลของไฟล์เป็น .c เท่านั้น (หมายถึงไฟล์ภาษา C) เช่น ในที่นี่กำหนดชื่อเป็น example1-1.c

4. ให้ลองพิมพ์โค้ดง่ายๆ ลงไป ดังนี้ (รายละเอียดของโค้ดจะกล่าวถึงในบทต่อๆ ไป)



```

File Edit Selection View Go Run Terminal Help
EXPLORER ...
CHAPTER1
example1-1.c

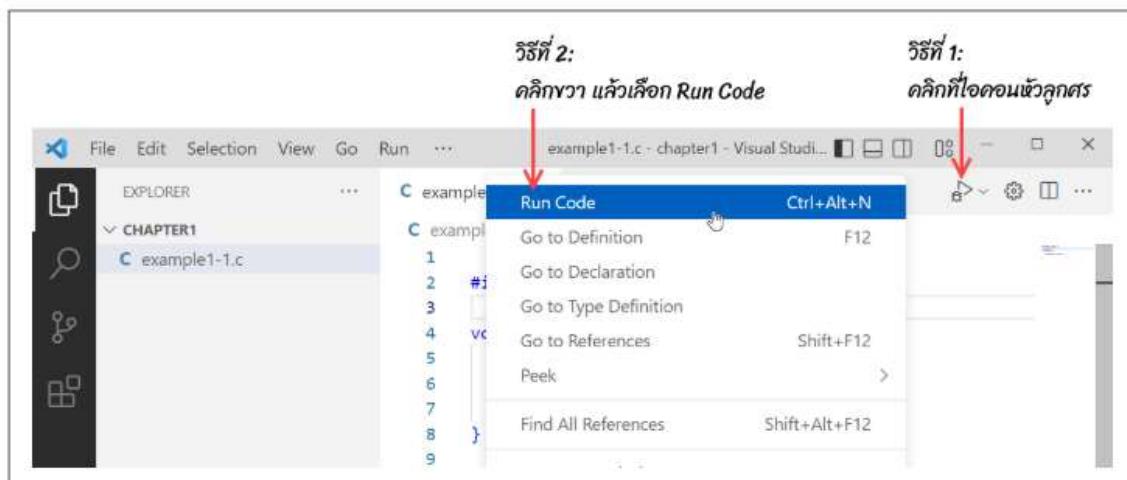
C example1-1.c •
C example1-1.c > ...
1
2 #include <stdio.h>
3
4 void main() {
5
6     puts("Hello World");
7
8 }
9

```

5. หลังการเขียนหรือแก้ไขโค้ด ต้องบันทึกการเปลี่ยนแปลงทุกครั้ง เช่น เลือกเมนู File > Save หรือกด <Ctrl + S>

6. รันทดสอบโค้ดที่เขียนด้วยวิธีใดวิธีหนึ่งคือ

- **วิธีที่ 1** คลิกที่ปุ่มไอคอนหัวลูกศรที่อยู่บริเวณมุมขวาบนของฟันที่เขียนโค้ด
- **วิธีที่ 2** คลิกขวาบริเวณพื้นที่เขียนโค้ด แล้วเลือกเมนู Run Code (เมนูรายการนี้จะปรากฏเฉพาะกรณีที่เราติดตั้งส่วนเสริม Code Runner เข้าไว้แล้วเท่านั้น) หรือ กด <Alt + Ctrl + N>



**7. เมื่อสั่งรันตามที่กล่าวมา ผลลัพธ์ที่ได้จะถูกนำมาแสดงบน Terminal (จะปรากฏโดยอัตโนมัติ) ดังภาพ**

The screenshot shows the Visual Studio Code interface. In the center, there is a code editor window titled 'example1-1.c' containing the following C code:

```

1 #include <stdio.h>
2
3 void main() {
4     puts("Hello World");
5 }

```

Below the code editor is a terminal window showing the command and its output:

```

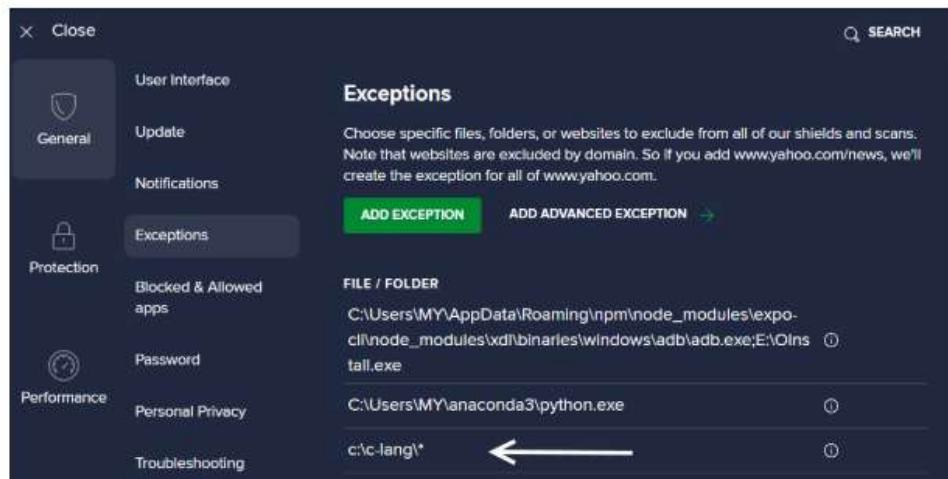
C:\c-lang\chapter1>cd "c:\c-lang\chapter1" && gcc example1-1.c -o example1-1 && "c:\c-lang\chapter1\example1-1"
Hello World
C:\c-lang\chapter1>

```

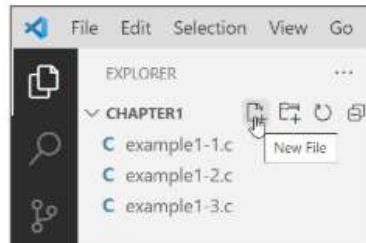
A red arrow points to the terminal output line 'Hello World'.

อย่างไรก็ตาม เนื่องจากการรันโค้ดภาษา C จะได้ไฟล์ .exe เพิ่มขึ้นมาอีกไฟล์ ซึ่งโปรแกรมประเภท Anti-Virus อาจมองว่าไม่ปลอดภัย และอาจบล็อกการรันก็เป็นได้ ซึ่งเราอาจแก้ปัญหาด้วยวิธีใดวิธีหนึ่งคือ

- วิธีที่ 1** อาจหยุดการทำงานของโปรแกรม Anti-Virus ชั่วคราวในระหว่างที่กำลังทดสอบการรันภาษา C แต่วิธีนี้อาจไม่ปลอดภัย หากในขณะนั้นเราใช้อินเทอร์เน็ตควบคู่กันไปด้วย
- วิธีที่ 2** อาจสร้าง Exception ภายในโปรแกรม Anti-Virus เพื่อให้มันยกเว้นการตรวจสอบตำแหน่งไฟล์เดอร์ที่เราจัดเก็บโค้ด เช่น ในที่นี่คือ C:\c-lang ดังในภาพถัดไป เป็นการสร้าง Exception บนโปรแกรม Avast (เลือกไอคอน Menu > Settings > Exceptions > Add Exception)



หากเราต้องการสร้างไฟล์สำหรับตัวอย่างต่อๆ ไป ก็ทำเหมือนเดิม นั่นคือ เปิดเข้าสู่ VS Code และเข้าสู่โฟลเดอร์ C:\c-lang\chapterX ที่ต้องการจัดเก็บไฟล์นั้น และก็คลิกไอคอน New File จากนั้นก็กำหนดชื่อที่มีส่วนขยายเป็น .c ตามเดิม ซึ่งเราจะสร้างเพิ่มกี่ไฟล์ก็ได้



หลังจากที่เราเสร็จสิ้นการใช้งาน อาจปิดโฟลเดอร์นั้น เช่น เลือกที่เมนู File > Close Folder แต่กรณีนี้ หากจะใช้งานเมื่อเข้าสู่ VS Code ครั้งต่อไป ต้องเปิดโฟลเดอร์ก่อน ตามหลักการเดิมทั้งหมด

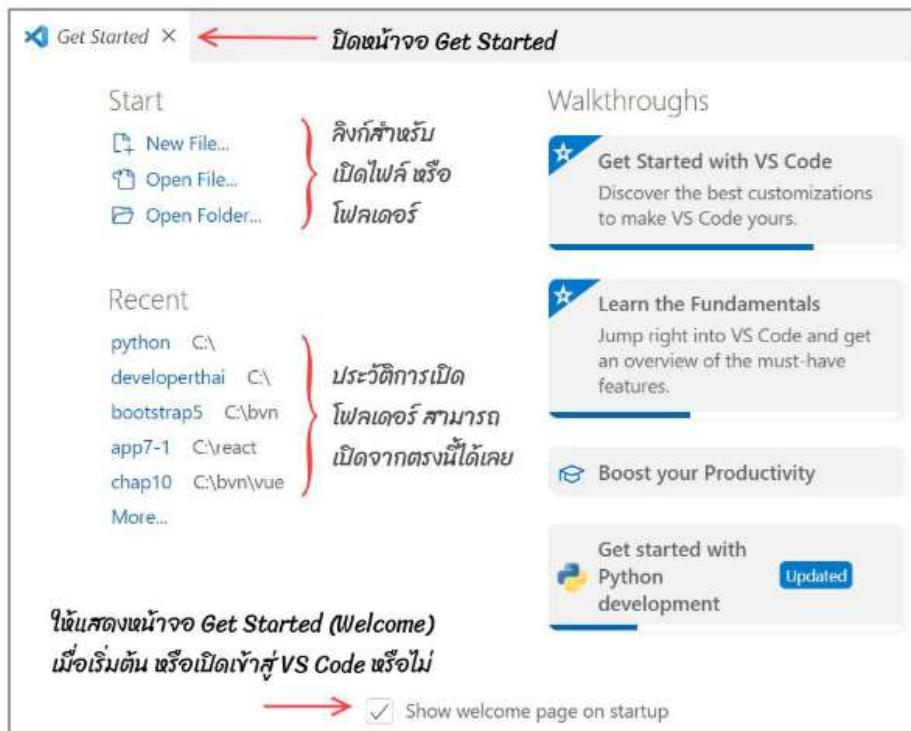
## การใช้เครื่องมือพื้นฐานของ VS Code

ก่อนที่เราจะเข้าสู่ขั้นตอนการเขียนโค้ดภาษา C อย่างเต็มรูปแบบในบทต่อไป ก็ควรรู้จักกับการใช้เครื่องมือพื้นฐานบางส่วนของ VS Code เพิ่มเติม เพื่อจำเป็นต้องใช้งานในบางโอกาส โดยลิสต์ที่เราควรรู้จักในเบื้องต้นมีดังนี้

### หน้าจอเริ่มต้น (Get Started)

ตามปกติ เมื่อเราเปิดเข้าสู่ VS Code จะปรากฏหน้าจอ Get Started เป็นอันดับแรก ซึ่งลักษณะที่สำคัญเกี่ยวกับหน้านี้คือ

- อาจมีการแจ้งข้อมูลช่าวสารที่น่าสนใจเกี่ยวกับ VS Code
- มีลิงก์สำหรับให้เราคลิกเพื่อเปิดไฟล์หรือโฟลเดอร์เป้าหมายที่ต้องการได้อีกหนึ่งวิธี นอกเหนือจากการเลือกที่เมนู
- มีลิงก์ที่แสดงประวัติการเปิดโฟลเดอร์ที่เราเข้าสู่ VS Code ในครั้งก่อนๆ ซึ่งหากมีโฟลเดอร์เป้าหมายที่เราต้องการรวมอยู่ด้วย ก็สามารถคลิกเปิดจากตรงนี้ได้ทันที

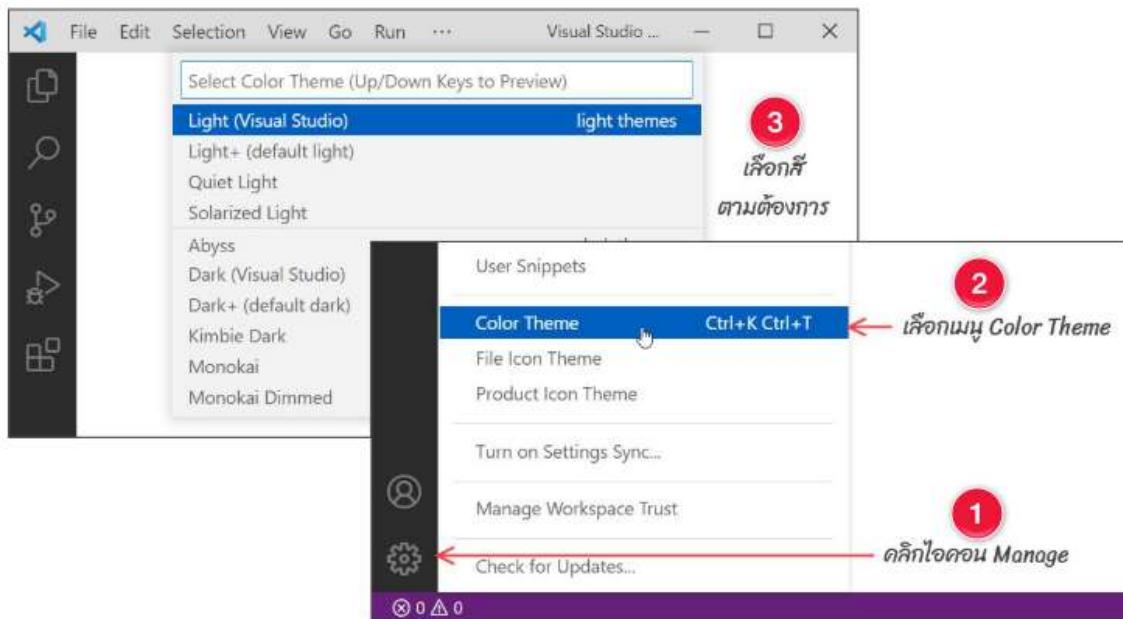


หากเราต้องการปิดหน้าจอนี้ ก็คลิกปุ่ม X ที่แท็บด้านบน หรือหากไม่ต้องการให้ปรากฏหน้าจอ Get Started ต่อไปอีก ก็ให้เราเอาเครื่องหมายถูกตรงตัวเลือก Show welcome page on startup ที่บริเวณขอบด้านล่างออก หรือหลังจากปิดไปแล้ว หากเราต้องการให้หน้าจอ Get Started กลับมาแสดงอีกครั้ง ก็ให้เลือกที่เมนู **Help > Get Started**

## การเลือกรูปแบบสี (Theme)

ตามปกติ VS Code จะเลือกธีม (Theme) เป็นสีดำ (Dark) เอาไว้ล่วงหน้า ซึ่งหากเราชอบแบบนี้ก็ไม่จำเป็นต้องปรับเปลี่ยน แต่ถ้าเราต้องการสีสันในรูปแบบอื่น ก็มีตัวเลือกอยู่จำนวนหนึ่ง โดยใช้ขั้นตอนดังต่อไปนี้

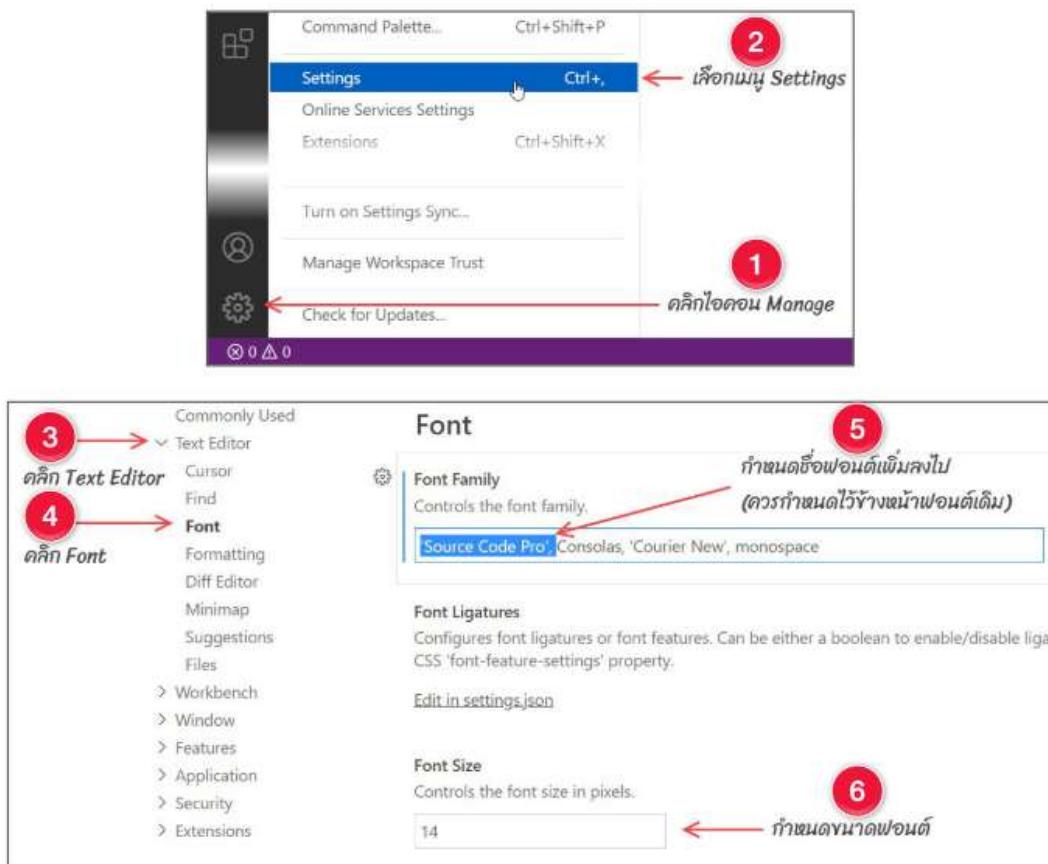
1. ที่มุมล่างขวาของโปรแกรม VS Code ให้คลิกที่ไอคอน Manage (รูปเฟือง)
2. เลือกที่เมนู Color Theme
3. จากนั้นจะปรากฏรายชื่อสีให้เลือก ถ้าเรายังไม่ทราบว่าแต่ละสีมีลักษณะเป็นยังไง ก็อาจลองเลือกดูก่อนก็ได้ ถ้าไม่ถูกใจก็ค่อยเลือกสีใหม่ สำหรับรูปแบบที่จะใช้ประกอบในหนังสือเล่มนี้ ผู้เขียนเลือกธีมสีแบบ Light (Visual Studio)



## การปรับเปลี่ยนฟอนต์

ฟอนต์ (Font: ในที่นี้คือตัวอักษรที่ใช้เขียนโค้ดและแสดงผลลัพธ์) ที่ VS Code กำหนด เอาไว้ล่วงหน้า แม้จะเป็นรูปแบบมาตรฐานที่นิยมใช้งานกันทั่วไป แต่หากต้องการใช้ฟอนต์ที่เรา ถูกใจ ก็สามารถเปลี่ยนแปลงได้ด้วยขั้นตอนดังต่อไปนี้

1. ที่มุมล่างขวาของโปรแกรม VS Code ให้คลิกที่ไอคอน Manage (รูปเฟือง)
2. เลือกที่เมนู Settings
3. คลิกแท็บ Text Editor
4. คลิกแท็บ Font
5. กำหนดชื่อฟอนต์ลงในช่อง Font Family ซึ่งตามปกติ จะมีฟอนต์เดิมที่ VS Code กำหนดไว้ให้ล่วงหน้า หากเราต้องการเปลี่ยนไปใช้ชนิดอื่น ควรเพิ่มชื่อฟอนต์ไว้ข้างหน้า ฟอนต์เดิม (ชื่อที่อยู่ก่อนจะถูกพิจารณา ก่อน และไม่แนะนำให้ลบชื่อฟอนต์เดิมออก) โดย ค้นด้วยเครื่องหมาย , และถ้าชื่อฟอนต์ชนิดนั้นมีช่องว่าง ให้เขียนไว้ในเครื่องหมาย '' ซึ่งชื่อฟอนต์ที่ระบุต้องถูกติดตั้งเอาไว้ในเครื่องนั้นอยู่ก่อนแล้ว (การติดตั้งฟอนต์ ให้ คัดลอกไฟล์ของฟอนต์ไปไว้ที่ C:\Windows\Fonts และเริ่มโปรแกรมที่จะใช้ฟอนต์ใหม่)
6. ถ้าต้องการกำหนดขนาด ก็ระบุตัวเลขลงในช่อง Font Size และเมื่อปิดหน้าจอ Setting ฟอนต์ที่ตั้งค่าเอาไว้ก็จะมีผลทันที



นอกจากนี้ ยังมีเทคนิคเพิ่มเติมอีกอย่างคือ การย่อหรือขยาย (Zoom) ขนาดของฟอนต์ ในโปรแกรม VS Code ให้ใหญ่ขึ้นหรือเล็กลง ซึ่งกรณีนี้ จะมีผลกับทั้งโค้ด เม뉴 และข้อความอื่นๆ ทั้งหมดบน VS Code โดยใช้วิธีการง่ายๆ คือ

- หากต้องการขยายขนาดเพิ่มขึ้น (Zoom In: คล้ายกับการเข้าไปมองใกล้ๆ ซึ่งจะเห็นขนาดที่ใหญ่ขึ้น) ให้กดคีย์บอร์ดปุ่ม **<Ctrl =>** หรือเลือกเมนู **View > Appearance > Zoom In** ซึ่งหากทำซ้ำเช่นนี้อีก ขนาดของฟอนต์จะเพิ่มขึ้น 1 ระดับ ไปเรื่อยๆ
- หากต้องการลดขนาดให้เล็กลง (Zoom Out: คล้ายกับการออกมากองไกลๆ ซึ่งเราจะเห็นขนาดที่เล็กลง) ให้กดคีย์บอร์ดปุ่ม **<Ctrl ->** หรือเลือกเมนู **View > Appearance > Zoom Out** ซึ่งหากทำซ้ำเช่นนี้อีก ขนาดของฟอนต์จะลดลง 1 ระดับ ไปเรื่อยๆ

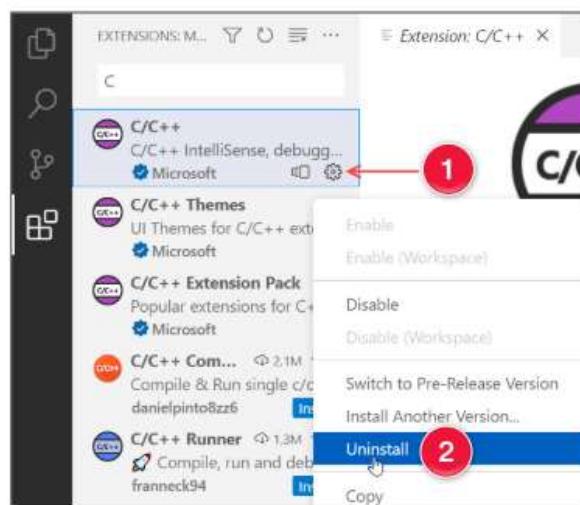
## แนวการการติดตั้งส่วนเสริมเพิ่มเติม

การเขียนโค้ดในบางกรณี เราอาจต้องใช้ส่วนเสริม (Extension) สำหรับการทำงานร่วมกับภาษาหนึ่งๆ ซึ่งในหัวข้อที่ผ่านมา เรายังได้ติดตั้งส่วนของภาษา C/C++ ไปแล้ว จึงขอกล่าวถึงอีกครั้งหนึ่ง เนื่องจากส่วนเสริมของ VS Code ยังมีอีกเป็นจำนวนมาก ซึ่งเราสามารถตรวจสอบและนำส่วนเสริมอื่นๆ มาใช้งานเพิ่มเติมได้ดังแนวทางต่อไปนี้

1. ในขณะนั้น ต้องเชื่อมต่อกับอินเทอร์เน็ต แล้วคลิกที่ไอคอน Extensions ที่ขอบด้านซ้าย (ดูภาพถัดไป) หรือเลือกที่เมนู View > Extensions
2. ใสคีย์เวิร์ดลงไป หลังจากนั้นจะปรากฏรายชื่อส่วนขยายที่อาจเกี่ยวข้องกับคีย์เวิร์ด ดังกล่าว
3. ถ้าเราต้องการติดตั้งส่วนเสริมใด ก็คลิกปุ่ม Install บนรายการนั้น หรือหากต้องการดูรายละเอียดเพิ่มเติมเกี่ยวกับส่วนเสริมใด ก็คลิกที่รายการดังกล่าว ดังภาพ



4. ถ้าค้นหาส่วนเสริมใดที่ถูกติดตั้งเอาไว้แล้ว จะมีรูปไอคอน Manage ที่มุมล่างขวาของรายการนั้น ทั้งนี้หากเราต้องการลบส่วนขยายนั้นทิ้งไป จากนั้นทำดังนี้
  - (1) คลิกที่ไอคอน Manage ที่ส่วนเสริมนั้น (รูปเฟือง)
  - (2) เมื่อปรากฏเมนู ให้คลิกที่ Uninstall



## การนำโค้ดของหนังสือมาใช้งาน

สำหรับโค้ดของหนังสือเล่มนี้ ผู้อ่านสามารถดาวน์โหลดได้ที่ <https://www.developerthai.com> หลังจากนั้นก็ให้ขยายไฟล์ดังกล่าว ซึ่งจะได้ไฟล์เดอร์ที่ชื่อ c-lang ก็ให้นำไฟล์เดอร์ดังกล่าวไปเก็บไว้ที่ C:\ ดังนั้น ตำแหน่งของมันก็จะเป็น C:\c-lang โดยภายในไฟล์เดอร์ c-lang จะประกอบด้วยไฟล์เดอร์ย่อยๆ ของบทต่างๆ เช่น chapter1, chapter2, ... ดังหลักการที่ได้กล่าวมาแล้ว

เมื่อต้องการเปิดดูโค้ด และ/หรือ รันทดสอบ ก็ให้เข้าสู่ VS Code และเปิดไฟล์เดอร์ C:\c-lang และไฟล์เดอร์ของทุกบทก็จะถูกเปิดพร้อมกันทั้งหมด หรือจะเลือกเปิดเฉพาะไฟล์เดอร์ของบทที่ต้องการ เช่น C:\c-lang\chapter2 ซึ่งจะเข้าถึงไฟล์ได้ง่ายขึ้น หลังจากนั้นก็คลิกชื่อไฟล์ของตัวอย่างที่ต้องการได้เลย

```

EXPLORER      ...
CHAPTER2 [+] ⌂ ⌂ ⌂ ⌂
> .vscode
C example2-1.c
C example2-2.c
C example2-3.c
C example2-4.c

C example2-3.c ✘
C example2-3.c > ...
1 #include <stdio.h>
2
3 void main() {
4     printf("123");
5     printf(" + ");
6     printf("456");
7     printf(" = ");
8     printf("579");
9 }

```

สิ่งที่เราได้เรียนรู้ในบทนี้ เป็นการจัดเตรียมเครื่องมือก่อนจะเข้าสู่ขั้นตอนการเขียนโค้ดภาษา C อย่างแท้จริงในบทต่อๆ ไป ซึ่งหากเรามีเครื่องมือที่ครบถ้วนและรู้จักวิธีการใช้งานที่ดี พอนั้นจะทำให้เราเรียนรู้ได้รวดเร็วยิ่งขึ้นด้วย



# 2

## การเขียนโค้ดภาษา C ในเบื้องต้น

ภาษา C มีองค์ประกอบและข้อกำหนดที่สำคัญหลายอย่างซึ่งเราควรรู้จักในเบื้องต้น โดยลิ้งที่จะกล่าวถึงในบทนี้ เช่น คำส่วน คำลั่งที่จำเป็นสำหรับการเขียนโค้ดในเบื้องต้น การแสดงผล การกำหนดบล็อกและคำอธิบายโค้ด เป็นต้น ซึ่งลิ้งเหล่านี้ล้วนเป็นพื้นฐานสำคัญของภาษา C ที่เราต้องใช้งานกันไปตลอด

### ประวัติโดยย่อของภาษา C

ภาษา C ถูกสร้างขึ้นเมื่อปี ค.ศ. 1972 โดย Dennis Ritchie ที่ห้องปฏิบัติการของ Bell Labs (หรือ AT & T ในปัจจุบัน) ซึ่งวัตถุประสงค์เริ่มแรกก็เพื่อใช้ในระบบ Unix อย่างไรก็ตาม Dennis Ritchie ไม่ใช่ผู้บุกเบิกในการสร้างภาษา C มาตั้งแต่เริ่มต้น แต่ภาษา C ได้จากการพัฒนาเพิ่มเติมต่อจากภาษาคอมพิวเตอร์อื่นๆ ที่มีอยู่ก่อนแล้วในขณะนั้น และหลังจากยุคของ Dennis Ritchie ก็มีหลายองค์กรที่ได้นำภาษา C มาพัฒนาเพิ่มเติมเรื่อยมา ซึ่งเราสามารถสรุปวิวัฒนาการที่เกี่ยวเนื่องกับภาษา C โดยลังเขปตั้งแต่อดีตจนถึงปัจจุบัน ได้ดังนี้

- 1960: **ภาษา ALGO**
  - สร้างโดยองค์กรระหว่างประเทศ (International Committee)
  - ไม่มีข้อกำหนดการใช้งานที่ชัดเจน เนื่องจากมีลักษณะที่เป็นนามธรรม (Abstract)มากเกินไป
- 1963: **ภาษา CPL (Combine Programming Language)**
  - สร้างขึ้นที่มหาวิทยาลัยเคมบริดจ์ (Cambridge University)
  - มีลักษณะที่เจาะจงมากเกินไป จึงไม่สะดวกต่อการใช้งานจริง

- 1967: **ภาษา BCPL** (Basic Combine Programming Language)
  - ◎ สร้างโดย Martin Richards ที่มหาวิทยาลัยคอมบริดจ์
  - ◎ ยังมีลักษณะที่เจาะจงเหมือนเดิม จึงเหมาะสมกับการใช้งานเฉพาะอย่าง เพราขาดความยืดหยุ่นต่อการใช้งานทั่วๆ ไป
- 1970: **ภาษา B**
  - ◎ สร้างโดย Ken Thompson ที่ห้องปฏิบัติการ Bell Labs (AT & T)
  - ◎ สร้างขึ้นจากพื้นฐานของภาษา BCPL (จึงเป็นที่มาของชื่อภาษา B) โดยแก้ไขข้อบกพร่องหลายอย่างและทำให้ภาษามีขนาดเล็กลง เช่น การลดจำนวนคีย์เวิร์ด แต่ก็ยังเหมาะสมกับการแก้ปัญหาที่มีลักษณะเฉพาะเจาะจงเหมือนเดิม
- 1972: **ภาษา C**
  - ◎ สร้างโดย Dennis Ritchie
  - ◎ สร้างขึ้นจากพื้นฐานของภาษา BCPL และภาษา B ร่วมกัน จึงกล้ายเป็นที่มาของชื่อภาษา C ว่าพัฒนาต่อจากภาษาในตระกูล B
  - ◎ ภาษา C มีวิวัฒนาการต่อเนื่องมาหลายเวอร์ชัน ซึ่งสามารถสรุปได้ดังนี้
    - ◎ 1972: กำเนิดภาษา C (หรือเรียกว่า C72)
    - ◎ 1978: **K&R C** (หรือเรียกว่า C78) โดย K มาจากชื่อของ Ken Thompson และ R มาจากชื่อของ Dennis Ritchie
    - ◎ 1989: **ANSI C** (หรือเรียกว่า C89)
    - ◎ 1990: **ISO C** (C90)
    - ◎ 1999: **C99**
    - ◎ 2011: **C11**
    - ◎ 2017: **C17**
    - ◎ 2023: **C23** (เป็นเพียงการคาดการณ์ในขณะที่เขียนหนังสือเล่มนี้ แต่ยังไม่มีการประกาศใช้งานจริง)

## ข้อดีและข้อเสียของภาษา C

ถึงแม้ภาษา C จะถูกสร้างมานานมากแล้ว รวมถึงเป็นต้นแบบให้กับภาษาอื่นๆ อีกหลายภาษา หากนับจากอดีตถึงปัจจุบัน ก็ถือว่าเป็นภาษาที่ประสบความสำเร็จและมีอิทธิพลต่อเทคโนโลยีด้านไอทีมากที่สุดอีกภาษาหนึ่ง และมีกลุ่มนักพัฒนาจำนวนมากจากทั่วโลกที่ยังคงเลือกใช้ภาษา C

กันต่อไป ถึงแม้ว่าภาษาอื่น ๆ ที่ถูกพัฒนาขึ้นมาภายหลังและพยายามจะมาทดแทนภาษานี้ก็ตาม แต่ภาษา C ยังเป็น 1 ใน 5 ของภาษาคอมพิวเตอร์ที่ผู้ใช้งานมากที่สุดในโลกไม่เคยเปลี่ยนแปลง ถึงอย่างไรก็ตาม ภาษา C ก็มีข้อดีและข้อเสียเช่นเดียวกับภาษาอื่น ๆ ซึ่งเราสามารถสรุปได้ดังนี้

### ข้อดีบางส่วนของภาษา C

- มีรูปแบบไวยากรณ์การเขียนโค้ดที่ไม่ยากและเข้าใจได้ง่าย
- สามารถเรียนรู้ได้ในระยะเวลาอันรวดเร็ว จึงเหมาะสมที่จะศึกษาเป็นภาษาเริ่มแรก ก่อน ก้าวไปสู่ภาษาอื่น ๆ ต่อไป เช่น C++, C# เป็นต้น
- ทำงานได้เร็วกว่าเมื่อเทียบกับภาษาอื่น ๆ ถือเป็นจุดเด่นที่สำคัญยิ่งของภาษานี้
- มีไลบรารีให้เลือกใช้จำนวนมากและครอบคลุมการทำงานที่สำคัญทั้งหมด
- รองรับการใช้งานในทุกระบบ เนื่องจากภาษา C เป็นพื้นฐานที่ถูกใช้ในการพัฒนาระบบปฏิบัติการทั้งหมดไม่ว่าจะเป็น Unix/Linux, Windows, Mac, iOS, Android, ... ดังนั้น จึงสามารถนำโค้ดที่เขียนไปใช้กับทุกระบบ ตามคำกล่าวที่ว่า write once, compile everywhere

### ข้อเสียบางส่วนของภาษา C

- ภาษา C เป็นภาษาแบบ Procedural (Function) ไม่ใช่ภาษาแบบ OOP (Object Oriented Programming) จึงอาจเกิดความยุ่งยากต่อการนำไปสร้างเป็นแอปพลิเคชันที่ซับซ้อนมากขึ้น เมื่อเทียบกับภาษาที่เป็น OOP เช่น C++, C#, Java, Objective-C เป็นต้น
- ไม่มีระบบจัดการปั๊มaha Garbage Collection หรือส่วนของหน่วยความจำที่ไม่จำเป็น ต้องใช้งานต่อไปอีก อาจทำให้เกิดปั๊มahaหน่วยความจำเต็ม หรือมีพื้นที่ว่างเหลือไม่พอ
- ไม่มีวิธีการป้องกันและจัดการข้อผิดพลาดในแบบ Exception Handling เหมือนที่มีในภาษาอื่น ๆ จึงอาจทำให้เกิดข้อผิดพลาดในระหว่างรันได้ง่าย
- ภาษา C ยังขาดเทคนิคที่จะช่วยให้การเขียนโค้ดนั้นง่ายขึ้น บางอย่างก็ใช้วิธีการที่ล้าสมัย ถ้าเทียบกับภาษาใหม่ ๆ ที่สร้างขึ้นในภายหลัง ซึ่งส่วนใหญ่จะมีแนวคิดและหลักการที่ทันสมัยมากกว่า ทั้งนี้ก็อาจเนื่องมาจากภาษา C นั้น ถูกสร้างมานานมากแล้ว และเป็นรากฐานของภาษาอื่น ๆ อีกหลายภาษา รวมถึงมีหลายมาตรฐาน ดังนั้น การเปลี่ยนแปลงกฎเกณฑ์ต่าง ๆ จึงทำได้ยาก

## คำส่วนของภาษา C

คำส่วน (Reserved Words) หรือคีย์เวิร์ด (Keywords) เป็นคำสำหรับใช้เป็นคำสั่งเพื่อควบคุมหรือกำหนดวิธีการทำงานในภาษา C ซึ่งเราจะนำคำเหล่านี้ไปตั้งเป็นชื่อตัวแปร หรือฟังก์ชันไม่ได้ โดยคำที่ถูกใช้เป็นคำส่วนในภาษา C คือ

auto	do	goto	signed	unsigned
break	double	if	sizeof	void
case	else	int	static	volatile
char	enum	long	struct	while
const	extern	register	switch	inline (C99+)
continue	float	return	typedef	restrict (C99+)
default	for	short	union	_Bool (C99+)

นอกจากนี้ ยังมีคำส่วนอีกบางส่วนที่ถูกเพิ่มเติมเข้ามาในเวอร์ชันหลังๆ แต่เรามักไม่ค่อยได้ใช้งานสำหรับการเขียนโปรแกรมทั่วๆ ไป กันบอยนัก จึงไม่ขอนำมากล่าวถึง โดยวัตถุประสงค์พอลังเขยปสำหรับคำส่วนบางส่วน มีดังนี้ (รายละเอียดที่ขัดเจนของคำเหล่านี้ เราจะได้เรียนรู้ในภายหลัง)

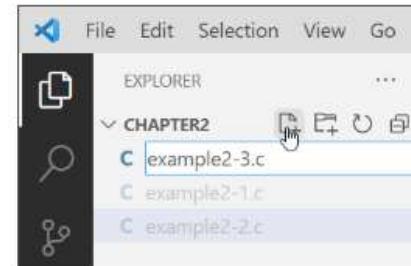
- **int, float, char, double, long, \_Bool**: ใช้ในการกำหนดชนิดข้อมูลของตัวแปร
- **if, else, switch, case, default**: ใช้ในการกำหนดเงื่อนไขเพื่อตัดสินใจ หรือควบคุมโครงสร้างการทำงานของโปรแกรมนั้นเอง
- **for, while, do**: รูปแบบการวนลูปเพื่อการทำซ้ำ
- **break**: ใช้เพื่อหยุดการวนรอบของลูป หรือออกจากเงื่อนไขแบบ switch case
- **void**: การส่งค่าเข้าและออกจากการฟังก์ชัน
- **goto**: กำหนดตำแหน่งที่จะข้ามไปทำงาน
- **auto, signed, const, extern, register, unsigned, volatile**: ใช้ร่วมกับการประกาศตัวแปรสำหรับกรณีเฉพาะ
- **return**: สำหรับการส่งค่ากลับออกจากฟังก์ชัน หรือหยุดการทำงานของฟังก์ชัน
- **continue**: ให้วนลูปถัดไปทันที โดยไม่ทำคำสั่งที่เหลือภายในลูป

- **enum**: สร้างชุดหรือกลุ่มของค่าคงที่ (Set of constants)
- **sizeof**: สำหรับการตรวจสอบขนาด
- **struct, typedef**: สร้างชนิดให้กับกลุ่มข้อมูลแบบโครงสร้าง (Structures)
- **union**: กำหนดกลุ่มตัวแปรที่จะใช้ตำแหน่งและพื้นที่จัดเก็บข้อมูลบนหน่วยความจำร่วมกัน

## บททวนการสร้างไฟล์ภาษา C

ในบทที่แล้ว เราได้ทราบแนวทางการสร้างและทดสอบไฟล์ภาษา C กันไปแล้ว โดยในบทนี้ ก็มีหลายกรณีที่เราต้องเขียนโค้ดตัวอย่าง เพื่อทดสอบการใช้งานควบคู่กันไป ทั้งนี้ เราจำเป็นต้องแยกแต่ละตัวอย่างไว้คนละไฟล์ จึงขอนำวิธีการสร้างไฟล์ภาษา C ที่เคยกล่าวไว้ในบทที่แล้ว มากล่าวทบทวนอีกครั้งหนึ่ง ดังนี้

- สำหรับตัวอย่างในบทที่ 2 เราจะเก็บไฟล์ทั้งหมดเอาไว้ที่ **C:\c-lang\chapter2** ดังนั้น เราจำเป็นต้องสร้างโฟลเดอร์ chapter2 ข้อนี้ไว้ในโฟลเดอร์ c-lang
- ใน VS Code ให้เปิดโฟลเดอร์ **C:\c-lang\chapter2** ที่ได้สร้างเอาไว้
- เมื่อต้องการสร้างไฟล์ตัวอย่างสำหรับบทนี้ ก็คลิกที่ไอคอน New File และกำหนดชื่อไฟล์โดยให้มีส่วนขยายเป็น .c เท่านั้น ชื่อในบทนี้จะตั้งชื่อตัวอย่างต่างๆ เป็น example2-1.c, example2-2.c ... เป็นเช่นนี้ไปเรื่อยๆ

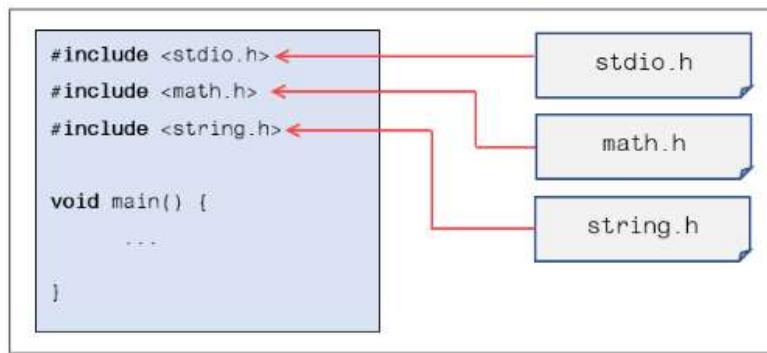


## องค์ประกอบของการเขียนโค้ดภาษา C ในเบื้องต้น

ไม่ว่าจะเขียนโค้ดเพื่อวัตถุประสงค์ใดก็ตาม เราจำเป็นต้องใช้ข้อกำหนดพื้นฐานของภาษาเหมือนๆ กัน ดังนั้น ก่อนที่เราจะเข้าสู่ขั้นตอนการเขียนโค้ดที่ซับซ้อนมากขึ้นในลำดับต่อๆ ไป ก็ควรรู้จักกับองค์ประกอบของการเขียนโค้ดในเบื้องต้นเอาไว้ล่วงหน้า เพราะมันคือสิ่งจำเป็นที่เราต้องนำไปใช้งานอยู่ตลอด โดยจะแยกแต่ละประเด็นที่นำเสนอเป็นหัวข้อต่างๆ ดังต่อไปนี้

### คำสั่ง #include

คำสั่ง include ในภาษา C จะใช้สำหรับการอ้างถึงไฟล์ภายนอกที่จัดเก็บชุดคำสั่งพิเศษ ที่ต้องนำเข้ามาใช้งานร่วมกับโค้ดในไฟล์นั้นดังแนวทางในภาพ



ทั้งนี้ สาเหตุที่เราต้องอ้างถึงชุดคำสั่งจากไฟล์ภายนอกก็เนื่องจาก ตามปกตินั้น ชุดคำสั่งดังกล่าว ไม่ใช่พื้นฐานที่จำเป็นสำหรับการทำงานทั่วๆ ไปของภาษา C แต่อาจใช้ในบางกรณี ดังนั้น เมื่อใดที่จำเป็นต้องใช้ ก็จะอ้างถึงด้วยคำสั่ง `include` ในรูปแบบดังนี้

```

#include <ชื่อไฟล์.h>
หรือ #include "ชื่อไฟล์"

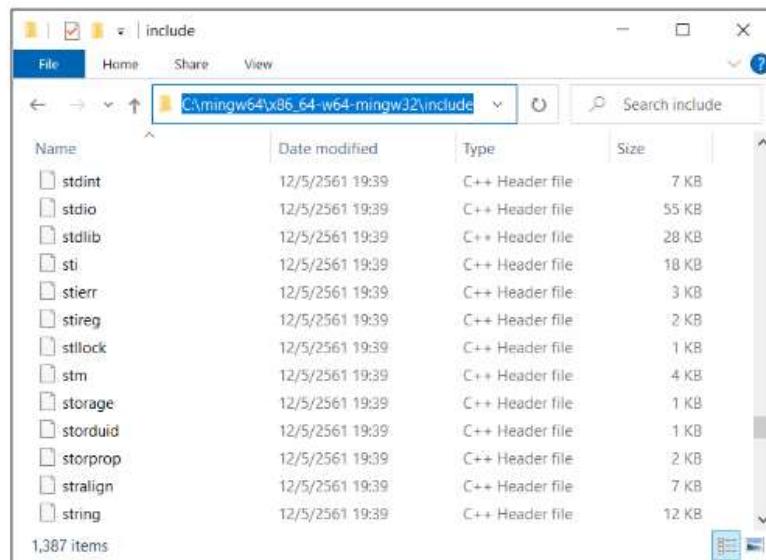
```

กลุ่มไฟล์ที่จัดเก็บชุดคำสั่งพิเศษดังกล่าว เราเรียกว่า เยอดเดอร์ (header) ด้วยเหตุนี้ จึงมีส่วนขยายเป็น `.h` ทั้งหมด โดยไฟล์เหล่านี้ เราจะได้เรียนรู้เพิ่มเติมต่อไปเรื่อยๆ แต่ยอดเดอร์ที่เราจะได้ใช้งานอยู่บ่อยๆ ก็คือ `stdio.h` ซึ่งจากที่ผ่านมาจะเห็นว่า เราเขียนโค้ดบรรทัดแรกเป็น

```
#include <stdio.h>
```

- **include** เป็นคำส่วนของภาษา C ที่ใช้อ้างถึงไฟล์ภายนอกดังที่กล่าวมาแล้ว โดยในการนี้ต้องเขียนเครื่องหมาย `#` กำกับไว้ข้างหน้า เพื่อให้แตกต่างจากคำสั่งทั่วไป
- **stdio.h** คำว่า `stdio` ย่อมาจาก standard input/output ซึ่งเป็นมาตรฐานการรับและแสดงผลข้อมูลทางจอภาพ ถ้าเราไม่อ้างถึงยอดเดอร์อันนี้ เมื่อเรียกใช้ฟังก์ชันที่เกี่ยวกับการรับและแสดงผลข้อมูล เช่น `puts`, `gets`, `printf`, `scanf` และอีกหลายฟังก์ชัน ก็จะเกิดข้อผิดพลาดเมื่อรันทดสอบ ซึ่งเราจะได้เรียนรู้เพิ่มเติมในภายหลัง

ถ้าเราติดตั้งตัวแปลงภาษา C ตามวิธีการของหนังสือเล่มนี้ (กล่าวไว้ในบทที่ 1) ตำแหน่งของไฟล์ยอดเดอร์ ก็จะอยู่ที่ `C:\mingw64\x86_64-w64-mingw32\include` ซึ่งถ้าเราลองเปิดเข้าไปดูจะพบกับไฟล์จำนวนมาก ในลักษณะดังนี้



การประกาศเอกสารให้เขียนไว้ส่วนหัวของไฟล์ก่อนจะเริ่มต้นโค้ดหลัก เพื่อให้มั่นถูก ประมวลผลก่อนส่วนโค้ดที่เราเขียนในลำดับถัดมา ซึ่งตามปกติ ในการเขียนโค้ดที่ขับช้อนขึ้น เรา อาจต้องอ้างถึงเอกสารมากกว่า 1 อัน โดยใช้หลักการเดิมทั้งหมด เช่น

```
#include <stdio.h>
#include <math.h>
#include <string.h>
```

## ฟังก์ชัน main

ในภาษา C (รวมถึงภาษาคอมพิวเตอร์โดยส่วนใหญ่) จะใช้ฟังก์ชันที่ชื่อ main เพื่อเป็น จุดเริ่มต้นในการทำงาน ซึ่งความจริงนั้น นอกจากคำว่า main แล้วอาจมีองค์ประกอบอื่นๆ ที่เรา สามารถเขียนเพิ่มเติมร่วมกับคำว่า main ได้ แต่ในเบื้องต้นนี้ จะแนะนำเพียงแบบเดียวไปก่อน นั่นคือ

```
void main() {
}
```

- **void** เป็นคำส่วนในภาษา C ซึ่งใช้เพื่อบ่งชี้ว่าฟังก์ชันนี้ จะไม่มีการส่งข้อมูลใด ๆ กลับ ออกไปจากฟังก์ชัน
- **main** เป็นชื่อฟังก์ชันหลักของโค้ดโปรแกรมนั้น ซึ่งเป็นส่วนแรกที่จะถูกเรียกขึ้นมา ทำงาน

## การกำหนดบล็อกคำสั่งด้วยวงเล็บ {}

การกระทำหรือดำเนินการบางอย่าง อาจประกอบด้วยคำสั่งย่อยๆ มากกว่า 1 คำสั่ง และเพื่อให้เราทราบขอบเขตว่าการกระทำนั้น ประกอบด้วยคำสั่งย่อยๆ ใดบ้าง ในภาษา C จะใช้วงเล็บ {} เป็นตัวกำหนดขอบเขต โดยนำคำสั่งย่อยๆ ที่จะดำเนินการต่อเนื่องกันไปตามลำดับมาเขียนไว้ในวงเล็บ {} ดังรูปแบบต่อไปนี้

```
{
    คำสั่งที่ 1
    คำสั่งที่ 2
    คำสั่งที่ 3
    ...
}
```

แต่ละคำสั่งภายในบล็อก ควรเยื่องเข้าไปเป็นระยะ 1 แท็บ โดยให้จุดเริ่มต้นของแต่ละบรรทัดอยู่ในแนวเดียวกัน แม้จะไม่ใช่ข้อบังคับ แต่ถ้าเป็นสิ่งที่นิยมทำกันทั่วไป เพราะช่วยให้ได้ดูนั้น อ่านง่ายขึ้น ซึ่งจากที่ผ่านมาจะเห็นว่า เราได้กำหนดวงเล็บ {} ต่อท้ายฟังก์ชัน main เพื่อกำหนด คำสั่งย่อยๆ ที่จะดำเนินการ เช่น

```
void main() {
    กำหนดคำสั่งย่อยๆ ไว้ในนี้
}
```

การเขียนวงเล็บเปิด { เพื่อเริ่มต้นบล็อกคำสั่งนั้น เราอาจวางไว้ที่บรรทัดถัดไป หรือจะเขียนต่อจากท้ายบล็อกในบรรทัดเดียวกันก็ได้ ดังวิธีการต่อไปนี้

- **วิธีที่ 1** New Line Block โดยเขียนวงเล็บเปิด { เพื่อเริ่มต้นบล็อกคำสั่ง ที่บรรทัดถัดไป เช่น

```
void main()
{
    คำสั่งย่อยๆ
    ...
}
```

● **วิธีที่ 2** In-Line Block โดยเขียนวงเล็บเปิด { ต่อท้ายบล็อกในบรรทัดเดียวกัน เช่น

```
void main() {
    คำสั่งย่อๆ
    ...
}
```

การเขียนโค้ดภาษา C ในยุคแรกๆ จะนิยมใช้แบบที่ 1 เป็นส่วนใหญ่ เพราะอ่านโค้ดได้ง่ายกว่า แต่ในยุคหลังๆ ที่การเขียนโค้ดจะซับซ้อนและยืดยาวมากขึ้นเรื่อยๆ เราจึงนิยมใช้แบบที่ 2 มากกว่า เพราะใช้พื้นที่ในการเขียนโค้ดน้อยกว่า จึงทำให้โค้ดดูลั้นลง แต่อย่างไรก็ตาม ไม่มีกฎเกณฑ์หรือข้อบังคับว่าเราต้องใช้แบบใด ทั้งนี้ผู้อ่านสามารถใช้แบบใดก็ได้ที่ตนถนัด และสำหรับในหนังสือเล่มนี้ จะเลือกใช้แบบที่ 2 เป็นหลัก ซึ่งเป็นรูปแบบที่พบเห็นในการใช้งานจริงได้มากกว่า

### การกำหนดจุดสิ้นสุดคำสั่งด้วยเครื่องหมาย ;

คำสั่งต่างๆ ในภาษา C ที่เราเขียน มักใช้เพื่อทำลิ๊งค์ให้ลิ๊งค์โดยเฉพาะ ดังนั้น เราอาจต้องนำคำสั่งย่อๆ หลายอันมาใช้งานร่วมกัน เพื่อดำเนินการอย่างใดอย่างหนึ่ง ด้วยเหตุนี้จึงต้องมีลิ๊งค์ที่ใช้ระบุขอบเขตหรือจุดสิ้นสุดในแต่ละคำสั่ง โดยในภาษา C จะใช้เครื่องหมาย Semicolon (;) เพื่อกำหนดจุดสิ้นสุดของแต่ละคำสั่ง โดยวางเครื่องหมายนี้ต่อท้ายคำสั่ง เช่น

```
void main() {
    คำสั่งที่ 1;      <= เขียนเครื่องหมาย ; ปิดท้าย เพื่อบ่งชี้ว่า ลิ๊งสุดคำสั่งที่ 1
    คำสั่งที่ 2;      <= เขียนเครื่องหมาย ; ปิดท้าย เพื่อบ่งชี้ว่า ลิ๊งสุดคำสั่งที่ 2
    คำสั่งที่ 3;      <= เขียนเครื่องหมาย ; ปิดท้าย เพื่อบ่งชี้ว่า ลิ๊งสุดคำสั่งที่ 3
    ...
}
```

และจากการที่เราจะบุเครื่องหมาย ; ต่อท้าย เพื่อรับ��จุดสิ้นสุดคำสั่ง ดังนั้น จึงสามารถนำคำสั่งถัดไป มาเขียนต่อเนื่องในบรรทัดเดียวกันได้ เพราะมีเครื่องหมาย ; เป็นตัวแบ่งหรือบ่งชี้จุดสิ้นสุดคำสั่งอยู่แล้ว เช่น

```
void main() {
    คำสั่งที่ 1; คำสั่งที่ 2; คำสั่งที่ 3;
    ...
}
```

## การเขียนคำอธิบายโค้ด (Comment)

เราสามารถเขียนข้อความแทรกลงในโค้ด เพื่ออธิบายเกี่ยวกับโค้ดส่วนนั้นพอลังเขย หรือจะเขียนข้อความในวัตถุประสงค์อื่นๆ ทั้งนี้ เมื่อเราย้อนกลับมาอ่านโค้ดในภายหลัง หรือเมื่อส่งโค้ดให้กับคนอื่นๆ ต่อไป ก็จะอ่านเข้าใจโค้ดส่วนนั้นได้เร็วขึ้น ซึ่งวิธีการเขียนคำอธิบายโค้ดในภาษา C นั้น อาจทำแบบใดแบบหนึ่ง ดังนี้

## การเขียนคำอธิบายแบบบล็อก (Block Comment)

วิธีนี้ จะกำหนดจุดเริ่มต้นและสิ้นสุดบล็อกด้วยสัญลักษณ์ /\* \*/ ตามลำดับ โดยนับตั้งแต่สัญลักษณ์ /\* เป็นต้นไป จะถือเป็นคำอธิบายทั้งหมด จนกว่าจะเจอลัญลักษณ์ \*/ จึงจะถือว่าสิ้นสุดคำอธิบาย ดังนั้น เราจึงสามารถเขียนคำอธิบายหลายบรรทัดไว้ในบล็อกเดียวกันได้ เช่น

```
/* example2-0.c */
/* C Programming Language
Written by Dennis Ritchie
*/

void main() {                                /* main program */
    puts("Hello World");                      /* display message */

    /* puts("Goodbye"); */                    /* โค้ดบรรทัดนี้ไม่ถูกประมวลผล
                                                แม้จะเป็นคำสั่งในภาษา C ก็ตาม
                                                เพราะเขียนไว้ใน Block Comment
*/
}
```

## การเขียนคำอธิบายแบบบรรทัด (Line Comment)

คำอธิบายแบบนี้ จะใช้สัญลักษณ์ // เป็นตัวกำหนดจุดเริ่มต้น และถือว่าจุดสิ้นสุดบรรทัด (ตำแหน่งที่เรากด <Enter> เพื่อขึ้นบรรทัดใหม่) เป็นจุดสิ้นสุดคำอธิบาย นั่นแสดงว่า เราจะเขียนคำอธิบายที่ยาวได้ไม่เกิน 1 บรรทัด และถ้าต้องการเขียนคำอธิบายที่ยาวหลายบรรทัด ต้องกำหนดด้วยสัญลักษณ์ // ไปทุกบรรทัด เช่น

```
//example2-0.c
//C Programming Language
//Written by Dennis Ritchie
```

```

void main() { //main program
    puts("Hello World"); //display message

    //puts( "Goodbye" ); //โค้ดบรรทัดนี้ไม่ถูกประมวลผล
    //แม้จะเป็นคำสั่งในภาษา C ก็ตาม
    //เพราะเขียนตามหลัง Line Comment
}

```

## การแสดงข้อความในเบื้องต้น

ในการเขียนโค้ดภาษา C นั้น ไม่ว่าจะเป็นการทดสอบหรือเพื่อใช้งานจริง ก็อาจมีกรณีที่ต้องแสดงข้อความบางอย่างแก่ผู้ใช้ ซึ่งอาจเป็นข้อความทั่วไป การแสดงผลลัพธ์ คำแจ้งเตือน รวมถึงการแสดงข้อผิดพลาดต่าง ๆ โดยการแสดงข้อความตามที่กล่าวมา อาจเป็นเพียงการแสดงข้อความแบบง่าย ๆ หรืออาจเป็นข้อความที่มีการจัดรูปแบบก็ได้ สำหรับข้อความที่มีการจัดรูปแบบ ที่ซับซ้อนขึ้นจะกล่าวถึงในบทที่ 4 แต่ในบทนี้ จะแนะนำเพียงการเขียนข้อความพื้นฐานหรรมดาไปก่อน ดังรายละเอียดต่อไปนี้

### การแสดงข้อความด้วยฟังก์ชัน puts()

วิธีการแสดงข้อความแบบที่ง่ายที่สุดในภาษา C น่าจะเป็นการใช้ฟังก์ชัน puts() ในรูปแบบดังนี้

```
puts( "ข้อความที่ต้องการแสดง" )
```

- ฟังก์ชัน puts() อยู่ในกลุ่มヘดเดอร์ stdio.h ดังนั้น เราต้องอ้างอิงヘดเดอร์นี้ด้วย #include <stdio.h> เอาไว้ล่วงหน้าเมื่อนำมาเคยทำมา จึงจะใช้งานฟังก์ชันนี้ได้
- ข้อความที่จะแสดงด้วย puts() ต้องเขียนไว้ในเครื่องหมายคำพูดแบบ " " เท่านั้น ถึงจะเป็นตัวเลขก็ต้องเขียนไว้ในเครื่องหมายนี้ เช่น กัน เช่น "123"
- ฟังก์ชัน puts() แสดงได้เพียงข้อความเดียว ถ้าเราจะแสดงหลายข้อความก็ต้องกำหนดแต่ละข้อความด้วยฟังก์ชัน puts() แยกกัน หรือใช้ฟังก์ชัน puts() หลายครั้งนั่นเอง
- ฟังก์ชัน puts() เป็นการแสดงข้อความ 1 บรรทัด ดังนั้น เมื่อเราใช้ฟังก์ชัน puts() เพื่อแสดงข้อความครั้งถัดไป มันจะถูกนำไปแสดงหรือขึ้นบรรทัดใหม่โดยอัตโนมัติ

- ข้อความที่อยู่ในเครื่องหมาย " " อันเดียวกัน ต้องเขียนให้จบในบรรทัดเดียวกัน จะแยก บางส่วนไปขึ้นบรรทัดใหม่ไม่ได้ (ความจริงมีวิธีที่ทำได้ แต่จะมีปัญหาการเว้นระยะห่าง) ถ้าข้อความนั้นยาวเกินไป ไม่สามารถเขียนจบในบรรทัดเดียวกัน วิธีแก้ปัญหานี้ในเบื้องต้น คือ ให้แยกเป็นข้อความย่อยๆ และกำหนดไว้ในเครื่องหมาย " " คนละอัน

สำหรับแนวทางการใช้งาน ให้เราดูจากตัวอย่างต่อไปนี้ ซึ่งวิธีการสร้างไฟล์ตัวอย่าง ในภาษา C ได้กล่าวมาไว้แล้ว

### ตัวอย่าง 2-1 การแสดงข้อความอย่างง่ายด้วยฟังก์ชัน puts()

```
#include <stdio.h>

void main() {
    puts("C is a real-world, widely available and ...");
    puts("C is a small, efficient, powerful, and ...");
    puts("C has been standardized, portable and ...");
}
```

```
C:\c-lang\chapter2>cd "c:\c-lang\chapter2\" && gcc example2-1.c -o example2-1
er2\example2-1
C is a real-world, widely available and popular language.
C is a small, efficient, powerful, and flexible language
C has been standardized, portable and close to hardware

C:\c-lang\chapter2>
```

### การแสดงข้อความด้วยฟังก์ชัน printf()

ตามปกตินั้น เรามักใช้ printf() กับข้อความที่มีการจัดรูปแบบบางอย่าง โดยตัว f ที่ เขียนต่อท้ายชื่อฟังก์ชัน มาจากคำว่า format ที่หมายถึงการจัดรูปแบบนั้นเอง แต่อย่างไรก็ตาม ในเบื้องต้นนี้ เราจะใช้ฟังก์ชัน printf() เพื่อแสดงข้อความธรรมดายังไม่มีการจัดรูปแบบใดๆ เช่นเดียวกับ puts() โดยข้อแตกต่างที่สำคัญในเบื้องต้นของ 2 ฟังก์ชันนี้คือ

- ฟังก์ชัน puts() จะแสดงข้อความโดยขึ้นบรรทัดใหม่โดยอัตโนมัติ
- ฟังก์ชัน printf() หากแสดงข้อความแบบธรรมดาก็จะแสดงที่บรรทัดเดิมต่อเนื่องไปเรื่อยๆ

สำหรับแนวทางการใช้ฟังก์ชัน `printf()` ในเบื้องต้น ให้ดูจากตัวอย่างต่อไปนี้

**ตัวอย่าง 2-2** การแสดงข้อความอย่างง่ายด้วยฟังก์ชัน `printf()`

```
#include <stdio.h>

void main() {
    printf("C is a real-world, widely available and ...");
    printf("C is a small, efficient, powerful, and ...");
    printf("C has been standardized, portable and ...");
}
```

```
C:\c-lang\chapter2>cd "c:\c-lang\chapter2\" && gcc example2-2.c -o example2-2 &&
c:\c-lang\chapter2\"example2-2
C is a real-world, widely available and popular language.C is a small, efficient,
powerful, and flexible language.C has been standardized, portable and close to har
dware.
C:\c-lang\chapter2>
```

จากผลลัพธ์จะเห็นได้ว่า `printf()` จะเขียนข้อความต่อท้ายข้อความเดิมในบรรทัดเดียวกัน ไปเรื่อยๆ ทั้งนี้ สำหรับวิธีการแก้ไขให้คำสั่ง `printf()` แสดงข้อความที่บรรทัดใหม่ จะกล่าวถึง ในหัวข้อถัดไป

**ตัวอย่าง 2-3** การแสดงข้อความด้วยฟังก์ชัน `printf()`

```
#include <stdio.h>

void main() {
    printf("123");
    printf(" + ");
    printf("456");
    printf(" = ");
    printf("579");
}
```

```
C:\c-lang\chapter2>cd "c:\c-lang\chapter2\" && gcc example2-3.c
c:\c-lang\chapter2\"example2-3
123 + 456 = 579
C:\c-lang\chapter2>
```

## อักษระพิเศษสำหรับขึ้นบรรทัดใหม่ (\n)

ถ้าเรามีข้อความย่อๆ หลายอันที่จะแสดงผล ก็ต้องใช้ฟังก์ชัน puts() หรือ printf() ช้าๆ ตามจำนวนข้อความจนครบ ดังตัวอย่างที่ผ่านๆ มาなん弄 ซึ่งการเรียกฟังก์ชันเดินช้าๆ แบบนี้ถือเป็นข้อบกพร่องที่เราควรแก้ไข โดยวิธีการที่จะแนะนำในเบื้องต้นคือ การแทรกอักษรพิเศษลงในข้อความเพื่อให้เกิดการขึ้นบรรทัดใหม่ โดยไม่จำเป็นต้องใช้ puts() หรือ printf() ช้าหลายครั้ง ดังหลักการต่อไปนี้

- ให้แทรกอักษรพิเศษคือ \n (หมายถึง New Line) ลงในข้อความ ณ ตำแหน่งที่ต้องการขึ้นบรรทัดใหม่
- ในข้อความเดียวกัน (ในเครื่องหมาย " " เดียวกัน) จะแทรก \n ลงใบกีครั้งก็ได้ โดยจะเกิดการขึ้นบรรทัดใหม่ทุกครั้งในตำแหน่งที่เป็นอักษร \n
- ความจริงแล้ว อักษรพิเศษที่ขึ้นต้นด้วย \ ยังมีอีกหลายอัน โดยเราเรียกอักษรพิเศษในกลุ่มนี้ว่า Escape Sequence ซึ่งจะกล่าวรายละเอียดเพิ่มเติมในบทที่ 3

เนื่องจากการแทรกอักษร \n ลงในข้อความ เป็นเพียงเรื่องง่ายๆ จึงขอให้ดูแนวทางการใช้งานจากตัวอย่างต่อไปนี้ได้เลย

### ตัวอย่าง 2-4 การแทรกอักษรพิเศษ \n ลงในข้อความเพื่อขึ้นบรรทัดใหม่

```
#include <stdio.h>

void main() {
    printf("one\n");
    printf("two\n");
    printf("three\n");
    printf("four\n");

    printf("five\nsix\nseven\n");

    puts("\nheight");
    puts("\nnine");
    puts("ten\n");
}
```

```
one
two
three
four
five
six
seven

eight

nine
ten
```

**ตัวอย่าง 2-5** การแทรกอักษรพิเศษ \t ลงในข้อความเดียวกัน (อยู่ในเครื่องหมาย " " คู่เดียวกัน) เพื่อขึ้นบรรทัดใหม่

```
#include <stdio.h>

void main() {
    puts("\n1989: ANSI C\n1990: ISO C\n1999: C99");
    printf("\n2011: C11\n2017: C17\n2023: C23\n");
}
```

```
C:\c-lang\chapter2>cd "c:\c-lang\chapter2" && gcc example4-1.c -o
example4-1 && "c:\c-lang\chapter2"\example4-1

1989: ANSI C
1990: ISO C
1999: C99

2011: C11
2017: C17
2023: C23

C:\c-lang\chapter2>
```

ลิงที่เราได้เรียนรู้ในบทนี้ ถึงจะเป็นเพียงพื้นฐานของภาษา C ในเบื้องต้น แต่มันคือลิงสำคัญที่เราต้องใช้งานกันอยู่ตลอด โดยการเขียนโค้ดในบทต่อๆ ไป เราจำเป็นต้องนำลิงที่ได้กล่าวถึงในบทนี้ ไปใช้งานร่วมกันแทนทุกกรณี





# 3

## เบนดข้อมูลและตัวแปร

ข้อ

อุปกรณ์แต่ละอย่างที่นำมาใช้งานในการประมวลผลต่าง ๆ นั้น ต้องตรงกับชนิดใดชนิดหนึ่งที่ภาษา C รองรับ เนื่องจากวิธีดำเนินการกับข้อมูลแต่ชนิดจะแตกต่างกันออกไป โดยข้อมูลแต่ละค่า เราต้องสร้างชื่ออ้างอิงให้กับมัน ซึ่งเรียกว่า ตัวแปร หรืออาจกล่าวได้ว่า สิ่งที่เกี่ยวข้องกับข้อมูลแต่ละค่า ก็คือตัวแปรและชนิดข้อมูลของตัวแปรนั้น เพื่อให้ตัวประมวลผลสามารถเลือกวิธีจัดการกับข้อมูลได้อย่างถูกต้อง ดังเนื้อหาที่เราจะได้เรียนรู้กันในบทนี้

### เบนดข้อมูลในภาษา C

ข้อมูลต่าง ๆ ที่เราจะนำมาประมวลผล จะต้องมีชนิดหรือประเภทที่แน่นอน ทั้งนี้ก็ เพราะข้อมูลแต่ละชนิดนั้นจะใช้วิธีจัดการและจัดเก็บที่แตกต่างกัน โดยในภาษา C จะแบ่งข้อมูลออกเป็นชนิดย่อย ๆ ค่อนข้างมาก ซึ่งความสามารถจำแนกเป็น 2 กลุ่มหลัก ๆ ดังนี้

- **Predefined Data Type** หรือ Primitive Data Type หรือ Base Type เป็นชนิดข้อมูลแบบพื้นฐานที่ถูกกำหนดมาล่วงหน้าหรือมีอยู่แล้วในภาษา C จึงสามารถใช้งานได้ทันที ซึ่งชนิดข้อมูลในกลุ่มนี้ ยังสามารถแบ่งเป็นกลุ่มย่อย ๆ ได้อีก โดยกลุ่มที่น่าสนใจคือ
  - ชนิดเลขจำนวนเต็ม (Integer) เช่น int, short, long เป็นต้น
  - ชนิดเลขทศนิยม (Float) เช่น float, double เป็นต้น
  - ชนิดอักษร (Character) เช่น char
  - ชนิดจริงเท็จ (Boolean) เช่น \_Bool
- **User-defined Data Type** เป็นชนิดข้อมูลที่ผู้ใช้ (ผู้เขียนโค้ด) กำหนดขึ้นมาเอง เช่น typedef, struct เป็นต้น

โดยส่วนใหญ่ เราจะใช้งานข้อมูลในกลุ่ม Predefined เป็นหลัก ส่วนกรณีของ User-defined อาจได้ใช้งานในบางกรณี สำหรับข้อมูลในกลุ่ม Predefined ซึ่งแบ่งย่อยออกเป็นชนิดต่างๆ ที่หลากหลาย โดยในบทนี้ เราจะศึกษารายละเอียดของบางชนิดที่น่าสนใจ และส่วนที่เหลือก็จะกล่าวเพิ่มเติมในบทต่อๆ ไป

### ข้อมูลประเภทเลขจำนวนเต็ม

ข้อมูลประเภทเลขจำนวนเต็ม (Integer) จะแบ่งย่อยออกเป็นชนิดอยู่ๆ ตามขนาดหรือช่วงตัวเลขที่สามารถจัดเก็บได้ ซึ่งสามารถสรุปได้ดังตารางต่อไปนี้

ชนิด	ขนาด	ค่าต่ำสุด	ค่าสูงสุด
<b>short</b>	2-byte	-32,768	32,767
<b>int</b>	4-byte	-2,147,483,648	2,147,483,647
<b>long</b>	4-byte	-2,147,483,648	2,147,483,647
<b>long long</b>	8-byte	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
<b>unsigned short</b>	4-byte	0	65,535
<b>unsigned int</b>	4-byte	0	4,294,967,295
<b>unsigned long long</b>	8-byte	0	18,446,744,073,09,551,615

- ค่าต่ำสุด (Minimum) และค่าสูงสุด (Maximum) เป็นตัวกำหนดขอบเขตของข้อมูลชนิดนั้น ว่าเราสามารถกำหนดค่าเป็นตัวเลขในช่วงใด ซึ่งหากกำหนดค่าที่ไม่อยู่ในช่วงของมันก็จะเกิดข้อผิดพลาด เช่น ข้อมูลชนิด short ต้องกำหนดค่าในระหว่าง -32,768 ถึง 32,767 เป็นต้น หรือเราจำเป็นง่ายๆ ดังนี้
  - ◎ **short** เก็บค่าได้สูงสุดประมาณ 3 หมื่น (ทั้งค่าบวกและลบ)
  - ◎ **int** และ **long** เก็บค่าได้สูงสุดประมาณ 2 พันล้าน (ทั้งค่าบวกและลบ)
- เราสามารถระบุคำว่า signed และ unsigned นำหน้าชนิดข้อมูล ซึ่งจะมีความหมายดังนี้
  - ◎ **signed** หมายถึง ข้อมูลนั้นสามารถมีเครื่องหมาย - (ลบ) นำหน้า หรือกล่าวได้ว่า อาจเป็นได้ทั้งค่าบวกและค่าลบนั้นเอง แต่อย่างไรก็ตาม เนื่องจากในการนี้ถือเป็นลักษณะปกติของเลขทั่วไปอยู่แล้ว ดังนั้น เราไม่จำเป็นต้องเขียน signed กำกับไว้ ข้างหน้าก็ได้ เช่น signed int อาจเขียนสั้นๆ แค่ int เป็นต้น

- ◎ **unsigned** หมายถึง ข้อมูลแบบไม่มีเครื่องหมายนำหน้า หรือเป็นได้แค่จำนวนเต็มบวกเท่านั้น และจะเห็นได้ว่า ค่าสูงสุดของข้อมูลในช่วงบวก จะเพิ่มเป็น 2 เท่าจากชนิดปกติ (signed) โดยในการนี้เราระบุคำว่า **unsigned** กำกับข้างหน้าชนิดข้อมูลเสมอ ทั้งนี้หากระบุแค่คำว่า **unsigned** จะหมายถึง **unsigned int**

## ข้อมูลประเภทเลขทศนิยม

ข้อมูลประเภทเลขทศนิยม (Float) หรือจำนวนที่สามารถมีทศนิยมได้ จะแบ่งย่อยออกเป็นชนิดต่าง ๆ ดังตารางต่อไปนี้

ชนิดข้อมูล	ขนาด	ค่าต่ำสุด	ค่าสูงสุด
<b>float</b>	4-byte	3.4e-38	3.4e+38
<b>double</b>	8-byte	1.7e-308	1.7e+308
<b>long double</b>	10-bytes	3.4e-4932	1.1e+4932

- ข้อมูลชนิด **float** สามารถมีทศนิยมได้ ไม่เกิน 6 ตำแหน่ง
- ข้อมูลชนิด **double** สามารถมีทศนิยมได้ ไม่เกิน 15 ตำแหน่ง
- ข้อมูลชนิด **long double** สามารถมีทศนิยมได้ ไม่เกิน 19 ตำแหน่ง
- ข้อมูลชนิด **double** เป็นการขยายขนาดให้เป็น 2 เท่าจากชนิด **float**
- ข้อมูลทั้ง 3 ชนิดในกลุ่มนี้ อาจเป็นได้ทั้งค่าวรากและลบ (ไม่สามารถระบุ **unsigned** กำกับได้)

## ข้อมูลประเภทอักขระ

คำว่าอักขระ (Character) อาจหมายถึง ตัวอักษร A-Z (พิมพ์เล็กหรือใหญ่ก็ได้), ตัวเลข 0-9 หรืออักขระอื่นๆ เช่น "@", "\$", "\*", "%", "#" เป็นต้น แต่ ต้องเป็นอักขระเพียงตัวเดียวเท่านั้น โดยข้อมูลชนิดอักขระจะกำหนดด้วยคำต่อไปนี้

ชนิดข้อมูล	ขนาด	ค่าต่ำสุด	ค่าสูงสุด
<b>char</b>	1-byte	-128	127
<b>unsigned char</b>	1-byte	0	255

ข้อมูลประเภทอักขระจะถูกจัดเก็บและจัดการในแบบรหัส ASCII (American Standard Code for Information Interchange) ไม่ใช่การเก็บอักขระตัวนั้นโดยตรง โดยอักขระแต่ละตัวจะมีรหัสของมันเอง เช่น A (65), a (97), C (67), \$ (36) เป็นต้น ตัวอย่างรหัส ASCII ของอักขระบางส่วนเป็นดังตารางต่อไปนี้

รหัส	อักขระ	รหัส	อักขระ	รหัส	อักขระ
33	!	48	0	91	[
34	"	49	1	92	\
35	#	...	...	93	]
36	\$	57	9	94	^
37	%	58	:	95	_
38	&	59	;	96	'
39	'	60	<	97	a
40	(	61	=	98	b
41	)	62	>	...	...
42	*	63	?	122	z
43	+	64	@	123	{
44	,	65	A	124	
45	-	66	B	125	}
46	.	...	...	126	~
47	/	90	Z	127	DEL

สำหรับรายละเอียดเพิ่มเติมเกี่ยวกับข้อมูลชนิดอักขระ เราจะได้เรียนรู้เพิ่มเติมในลำดับต่อไป



### หมายเหตุ

นอกจากที่กล่าวมาแล้ว ในกลุ่ม Predefined Type ก็ยังชนิดข้อมูลอื่นๆ อีก เช่น void, \_Bool ซึ่งเรายังไม่จำเป็นต้องใช้สำหรับการเรียนรู้ในช่วงบทแรกๆ จึงขอนำไปกล่าวร่วมกับบทที่เกี่ยวข้อง กับข้อมูลชนิดนี้โดยตรง ซึ่งในบทนี้จะกล่าวถึงเฉพาะชนิดข้อมูลที่จำเป็นต้องใช้ประกอบการเรียนรู้ ในเบื้องต้นไปก่อน

## ลักษณะของตัวแปรในภาษา C

ในการเขียนโปรแกรม เราจำเป็นต้องใช้ข้อมูลหลายอย่างร่วมกัน ซึ่งข้อมูลแต่ละอย่าง จะต้องมีชื่อในการอ้างอิง โดยเราจะเรียกชื่อที่ใช้สำหรับการอ้างอิงข้อมูลว่า ตัวแปร (Variable) หรืออาจกล่าวได้ว่า

ตัวแปร ก็คือคำแนะนำที่ใช้จัดเก็บและอ้างถึงข้อมูล

ทั้งนี้ การตั้งชื่อตัวแปร จะต้องเป็นไปตามหลักการหรือข้อกำหนดของภาษา C มีดังนี้ อาจเกิดข้อผิดพลาดขึ้นได้ ดังนี้

- ชื่อตัวแปร ต้องประกอบด้วยตัวอักษร a-z, A-Z เลข 0-9 หรือเครื่องหมาย \_ (underscore) เท่านั้น จะมีอักษรอื่น ๆ นอกเหนือจากนี้รวมอยู่ด้วยไม่ได้
- ชื่อตัวแปรจะขึ้นต้น (อักษรตัวแรก) ด้วยตัวเลขไม่ได้ (แต่อาจมีตัวเลขอยู่ในคำแนะนำ อื่น ๆ ที่ไม่ใช่ตัวแรกได้)
- ต้องไม่ซ้ำกับคำส่วน (Reserved Words) ของภาษา C
- การเขียนด้วยตัวพิมพ์ในรูปแบบที่ต่างกัน ถือว่าไม่เหมือนกันแม้จะเป็นคำเดียวกันก็ตาม เช่น การเขียนเป็น value, Value, VALUE ไม่ถือว่าซ้ำกัน (ไม่ใช่ตัวแพรเดียวกัน)
- ชื่อตัวแปรในภาษา C เรา尼ยมเขียนเป็นตัวพิมพ์เล็กทั้งหมด (แต่ไม่ใช่ข้อบังคับ)
- ถ้านำมาอ่านแล้วเป็นชื่อตัวแปร ตามแนวทางที่ใช้กันส่วนใหญ่ในภาษา C เรา尼ยมเชื่อมแต่ละคำด้วยเครื่องหมาย underscore (\_) ซึ่งจะอ่านได้ง่ายกว่าการเขียนติดกัน เช่น first\_value, user\_name, day\_of\_week (แต่ไม่ใช่ข้อบังคับ)
- ลักษณะการตั้งชื่อตัวแปรที่ ถูกต้อง

<input type="radio"/> x	<input type="radio"/> string2number
<input type="radio"/> value	<input type="radio"/> v3_beta2
<input type="radio"/> firstname	<input type="radio"/> core_i_7
<input type="radio"/> pricePerUnit	<input type="radio"/> _x_x_
<input type="radio"/> num1	<input type="radio"/> _x_
<input type="radio"/> grand_total	<input type="radio"/> Void (ตัวพิมพ์ต่างจาก Reserved Word)
<input type="radio"/> freeDiskSpace	

- ลักษณะการตั้งชื่อตัวแปรที่ ไม่ถูกต้อง
    - price\$ เพราะมีอักษร \$
    - 3x เพราะขึ้นต้นด้วยตัวเลข
    - auto เพราะซ้ำกับคีย์เวิร์ด
    - date-month-year เพราะมีเครื่องหมาย -
    - lucky number เพราะมีช่องว่าง

## การประกาศตัวไป

ก่อนที่เราจะนำตัวแปรไปใช้งานได้ จำเป็นต้องประกาศตัวแปรนั้นขึ้นมาก่อน ทั้งนี้ก็เพื่อ  
จะพื้นที่บนหน่วยความจำสำหรับจัดเก็บข้อมูลนั้นเอง ดังนั้น เราจำเป็นต้องระบุชนิดข้อมูลที่  
จัดเก็บด้วยตัวแปรนั้นลงมาด้วย ดังรูปแบบต่อไปนี้

## ชนิดข้อมูล ชื่อตัวแปร;

- ชนิดข้อมูล ก็เลือกใช้ชนิดใดชนิดหนึ่งตามลักษณะของข้อมูลที่เราจะจัดเก็บ เช่น int, float, char เป็นต้น
  - ชื่อตัวแปร ก็กำหนดตามหลักการที่ได้กล่าวมาแล้ว

## แนวทางการประการศตัวแปร เช่น

```
int x;  
int y;  
long distance_from_earth_to_mars;  
float f;  
char ch;  
double latitude;
```

เราจะสร้างหรือประกาศด้วยเครื่องเขียนกัน (ชื่อเดียวกัน) ไม่ได้ แม้จะกำหนดชนิดข้อมูลต่างกันก็ตาม เช่น กรณีต่อไปนี้ จะเกิดข้อผิดพลาดทั้งหมด

```
int x;  
int x; //Error เพราะชื่อตัวแปรซ้ำกัน  
float x; //Error เพราะชื่อตัวแปรซ้ำกัน
```

กรณีที่เราจะประกาศตัวแปรหลาย ๆ ตัว ให้มีชนิดข้อมูลเดียวกัน สามารถนำมาเขียนรวม ในคำสั่งเดียวกัน โดยระบุชนิดข้อมูลเพียงครั้งเดียว ในรูปแบบดังนี้

ชนิดข้อมูล ชื่อตัวแปร\_1, ชื่อตัวแปร\_2, ชื่อตัวแปร\_3, ... ;

- ชนิดข้อมูล ก็กำหนดตามหลักการเดิม แต่เขียนเพียงครั้งเดียว
  - เราจะกำหนดตัวแปรจำนวนกี่ตัวก็ได้ โดยคั่นแต่ละตัวด้วยเครื่องหมาย Comma ( , )
- แนวทางการประกาศตัวแปรแบบใช้ชนิดข้อมูลร่วมกัน เช่น

```
int num1, num2, num3;           //ตัวแปรทั้งหมดเป็นชนิด int
float sum, average;            //ตัวแปรทั้งหมดเป็นชนิด float
double latitude, longitude;    //ตัวแปรทั้งหมดเป็นชนิด double

char letter1, letter2,
     char1, char2, char3;    //เขียนแยกบรรทัดก็ได้
```

## การกำหนดค่าให้กับตัวแปร

ตัวแปรที่เราประกาศขึ้นมาดังหัวข้อที่แล้ว จะเป็นตัวแปรแบบว่างเปล่า หรือไม่ได้จัดเก็บข้อมูลใด ๆ นั่นเอง ทั้งนี้หลังการประกาศตัวแปร เราต้องกำหนดค่าให้กับมันก่อนจะใช้งานได้ โดยรูปแบบพื้นฐานทั่วไปของการกำหนดค่าให้กับตัวแปรคือ

ชื่อตัวแปร = ค่าที่กำหนด;

โดยการกำหนดค่าให้กับตัวแปรนี้ จะต้องทำหลังจากที่ได้ประกาศเอาไว้แล้วเท่านั้น ส่วนค่าที่กำหนดมีหลักการพื้นฐานดังนี้

- ต้องสอดคล้องกับชนิดข้อมูลของตัวแปร
- ถ้าเป็นชนิดตัวเลข เช่น int, short, long, float, double ให้เขียนตัวเลขนั้นลงมาโดยตรง
- ถ้าเป็นชนิด char ให้เขียนอักษรไว้ในเครื่องหมาย '' เท่านั้น
- กรณีที่เป็นชนิดตัวเลข ต้องอยู่ในช่วงที่เป็นไปได้ (อยู่ภายในช่วงค่าต่ำสุดและสูงสุดของข้อมูลชนิดนั้น)
- ค่าที่กำหนดให้แก่ตัวแปร สามารถเปลี่ยนแปลงแก้ไขในภายหลังได้

แนวทางการกำหนดค่าให้กับตัวแปร เช่น

```
int quantity;
quantity = 10;

float price;
price = 199.50;

char grade;
grade = 'B';
```

ลักษณะการกำหนดค่าให้กับตัวแปรที่ผิดพลาด

```
int a;
a = 1.50;           //Error เพราะตัวแปร a เป็นชนิด int
                    //ซึ่งต้องกำหนดค่าที่เป็นจำนวนเต็ม

short b;
b = 1000000;        //Error เพราะ b เป็นชนิด short
                    //ซึ่งต้องมีค่าระหว่าง -32768 ถึง 32767

char c;
c = Z;              //Error
c = "Z"             //Error
                    //ต้องเขียนอักขระไว้ในเครื่องหมาย ''
```

อย่างไรก็ตาม หากเราทราบค่าที่จะกำหนดให้กับตัวแปรอยู่แล้วในขณะนั้น อาจลดขั้นตอนให้สั้นลง โดยประกาศตัวแปรแล้วกำหนดค่าให้กับมันในคำสั่งเดียวกันดังรูปแบบต่อไปนี้

ชนิดข้อมูล ชื่อตัวแปร = ค่าที่กำหนด;
--------------------------------------

แนวทางการประกาศและกำหนดค่าตัวแปรในคำสั่งเดียวกัน เช่น

```
int quantity = 10;
float price = 199.50;
char grade = 'B';

short n1 = 100;
short n2 = n1;           //นำค่าตัวแปร n1 มากำหนดให้ n2
```

ถ้าเราประกาศตัวแปรหลายตัวในรูปแบบการใช้ชั้นนิดข้อมูลร่วมกัน ก็สามารถกำหนดค่าแบบต่อเนื่องในรูปแบบดังนี้

```
ชั้นนิดข้อมูล ชื่อตัวแปร_1 = ค่าที่กำหนด, ชื่อตัวแปร_2 = ค่าที่กำหนด, ... ;
```

ประกาศตัวแปรหลายตัวโดยใช้ชั้นนิดข้อมูลร่วมกัน พร้อมกำหนดค่าให้กับมันไปพร้อมๆ กัน เช่น

```
int a = 10, b = 20, c = 30;
float x = 10.25, y = 11.50, z = 12.75;
char first='F', second='B', third='I';
```

เราอาจกำหนดค่าให้กับตัวแปรเพียงบางตัว แต่บางตัวอาจแค่ประกาศชื่อไว้ก่อนแล้วค่อยกำหนดค่าในภายหลังก็ได้ เช่น

```
int a = 10, b, c = 30;
float x, y, z = 12.75;
```

ในกรณีที่เราจะกำหนดค่าเดียวกันให้กับตัวแปรหลาย ๆ ตัว หรือให้ตัวแปรเหล่านั้นมีค่าเดียวกัน อาจกำหนดค่าแบบต่อเนื่องโดยระบุค่าเพียงครั้งเดียวที่ด้านขวาสุด ดังแนวทางดังต่อไปนี้

```
int x, y, z;
x = y = z = 10; //ตัวแปรทั้งหมดมีค่าเป็น 10
```

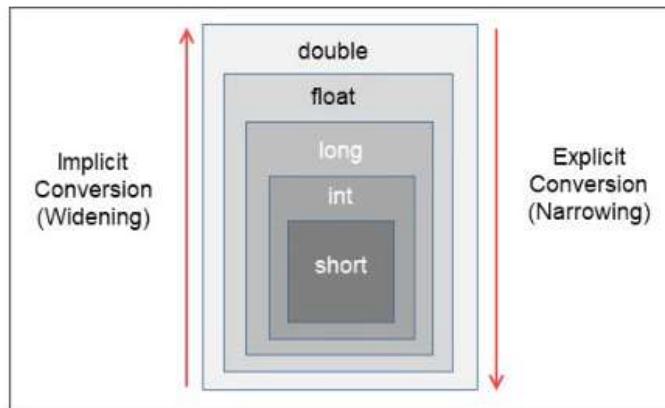
ที่กล่าวมาทั้งหมดเป็นการกำหนดค่าแบบพื้นฐานทั่วไป แต่อย่างไรก็ตาม เนื่องจากเราต้องกำหนดชนิดข้อมูลให้กับตัวแปร และข้อมูลแต่ละชนิดก็จะมีลักษณะปลีกย่อยเฉพาะตัวบางอย่างที่แตกต่างกัน ดังนั้น เราจะได้เรียนรู้เพิ่มเติมเกี่ยวกับข้อมูลแต่ละชนิดกันไปเรื่อยๆ

## พื้นฐานการแปลงชนิดข้อมูล

ในการประกาศตัวแปรนั้น เราต้องระบุชนิดข้อมูลที่จะจัดเก็บด้วยตัวแปรนั้นลงมาด้วย ดังที่เราได้เรียนรู้มาแล้ว อย่างไรก็ตาม เนื่องจากข้อมูลแต่ละชนิดมีขนาดหรือขอบเขตที่แตกต่างกัน ทั้งนี้ หากเราจัดเรียงตามขนาดจากเล็กไปใหญ่ก็จะได้เป็นดังนี้

**short > int > long > float > double**

โดยข้อกำหนดพื้นฐานที่เราได้เรียนรู้มา ก็คือ ต้องกำหนดค่าให้สอดคล้องกับชนิดข้อมูลของตัวแปร มิฉะนั้นอาจเกิดข้อผิดพลาด แต่อย่างไรก็ตาม ในภาษา C นั้น มีกลไกที่ช่วยในการแปลงชนิดข้อมูลจากอักษรนิดหนึ่งไปเป็นอักษรนิดหนึ่งได้ หรือที่เรียกว่า Type Conversion โดยแบ่งเป็น 2 รูปแบบคือ Implicit Conversion และ Explicit Conversion ดังรายละเอียดต่อไปนี้



## Implicit Conversion

Implicit Conversion หรือ Widening Conversion เป็นการแปลงจากชนิดข้อมูลที่มีขนาดเล็กกว่า ไปเป็นชนิดที่มีขนาดใหญ่กว่า เช่น แปลงจาก `short => int` หรือจาก `int => long` เป็นต้น ซึ่งสามารถทำได้โดยอัตโนมัติ และไม่จำเป็นต้องอาศัยการกระทำใดๆ เข้ามาช่วย ซึ่งก็เปรียบเสมือนการนำลิ้งของขนาดเล็ก ไปจัดเก็บในภาชนะที่มีขนาดใหญ่กว่า ย่อมทำได้โดยธรรมชาติอยู่แล้ว เช่น

```

short a = 100;
int i = a;           //Implicit Conversion

short x = 1000;
long y = x;          //Implicit Conversion

float f = 123.45F;
double d = f;        //Implicit Conversion
    
```

## Explicit Conversion

Explicit Conversion หรือ Narrowing Conversion เป็นการแปลงจากชนิดข้อมูลที่มีขนาดใหญ่กว่า ไปเป็นชนิดที่มีขนาดเล็กกว่า เช่น แปลงจาก `long => int` หรือจาก `double => int` เป็นต้น โดยเหตุผลที่ต้องทำเช่นนี้ มักเกิดจากการนิ่งที่เราต้องนำค่าที่มีชนิดข้อมูลใหญ่กว่า ไปกำหนด

ให้กับตัวแปรที่มีชนิดข้อมูลเล็กกว่า จึงเปรียบเสมือนการนำลิงของที่มีขนาดใหญ่ ไปใส่ลงในภาชนะที่มีขนาดเล็กกว่า ย่อมทำไม่ได้โดยวิธีการแบบปกติ แต่ในภาษา C มีวิธีการที่ช่วยแก้ปัญหานี้ได้โดยใช้การกระทำที่เรียกว่า การค่าสต์ (Cast) เช้ามาช่วย ในรูปแบบต่อไปนี้

(ชนิดข้อมูล) ค่าที่ต้องการแปลง

- ชนิดข้อมูลเป้าหมายของการแปลง ต้องเขียนไว้ในวงเล็บ ( )
- ต้องวางชนิดข้อมูล ไว้ข้างหน้าค่าของข้อมูลที่ต้องการแปลง

ถ้าเราแปลงจากจำนวนที่มีทศนิยมไปเป็นจำนวนเต็ม ส่วนทศนิยมจะถูกตัดทิ้งเหลือแค่จำนวนเต็ม แต่ถ้าแปลงจากชนิดจำนวนเต็มเป็นทศนิยม ก็จะมี .0 ต่อท้าย เช่น

```
double d = 123.45;
int i = (int)d;           // i = 123

short a = 123;
float b = (float)a;
// b = 123.0
```

อย่างไรก็ตาม สิ่งที่เรารออาจนำมาพิจารณาด้วยก็คือขอบเขตของข้อมูลแต่ละชนิด ทั้งนี้หากข้อมูลเดิมมีค่าเกินขอบเขตของชนิดข้อมูลปลายทางที่เราจะค่าสต์ ก็อาจได้ผลลัพธ์ที่ผิดพลาด เช่น กรณีต่อไปนี้

```
int value1 = 1234567;
short value2 = (short)value1;
//ไม่เกิดข้อผิดพลาด แต่ได้ผลลัพธ์ที่ไม่ตรงตามคาดหวัง
// เพราะข้อมูลชนิด short ต้องมีค่าระหว่าง -32768 ถึง 32767
```

การแปลงข้อมูลโดยวิธีการค่าสต์ จะมีประโยชน์เมื่อเราต้องนำไปใช้ร่วมกับการคำนวณที่มีข้อมูลหลายชนิดร่วมกัน ซึ่งเราจะได้เรียนรู้ในบทที่ 5 โดยในบทนี้ เพียงแค่แนะนำให้รู้จักกับหลักการในเบื้องต้นเอาไว้ล่วงหน้าเท่านั้น

## ค่าคงที่

ค่าคงที่ เป็นการจัดเก็บข้อมูลในอีกลักษณะหนึ่งที่เราต้องใช้ค่าได้ค่าหนึ่งไปตลอด จะเปลี่ยนแปลงแก้ไขไม่ได้ (ต่างจากตัวแปร ที่เราสามารถแก้ไขค่าของมันได้ตลอด) โดยการสร้างค่าคงที่ในภาษา C สามารถทำได้ 2 วิธีคือ การใช้คำสั่ง const และ define ดังรายละเอียดต่อไปนี้

### การกำหนดค่าคงที่ด้วยคำสั่ง const

`const` (มาจากคำว่า `constant`) จัดเป็นคำส่วนอิกอันหนึ่งของภาษา C ใช้สำหรับกำหนดค่าคงที่ในรูปแบบดังนี้

**const** ชนิดข้อมูล ชื่อค่าคงที่ = ค่าที่กำหนด;

- ต้องระบุคำว่า const นำหน้าเพื่อบ่งชี้ว่าเป็นค่าคงที่
  - ต้องกำหนดชนิดข้อมูล เช่นเดียวกับการประกาศตัวแปร
  - การตั้งชื่อค่าคงที่ใช้หลักเกณฑ์เดียวกับการตั้งชื่อตัวแปร และโดยทั่วไปเรานิยมตั้งชื่อค่าคงที่โดยเขียนเป็นตัวพิมพ์ใหญ่ทั้งหมด (แต่ไม่ใช่ข้อบังคับ) เช่น PI, VAT, MAX\_VALUE เป็นต้น ทั้งนี้ก็เพื่อให้ต่างจากตัวแปรนั้นเอง
  - ต้องกำหนดค่าให้กับมันในขั้นตอนการประกาศทันที จะกำหนดค่าในภายหลังเมื่อยกับตัวแปรไม่ได้ และค่าที่กำหนดต้องสอดคล้องกับชนิดข้อมูล เช่นเดียวกับตัวแปร
  - เราไม่สามารถแก้ไขค่าคงที่ในภายหลังได้ (ต้องใช้ค่านั้นไปตลอด)

แนวทางการกำหนดค่าคงที่ด้วย const เช่น

```
const float PI = 3.141;
const short MAXIMUM = 500;
const int FIRST_VALUE = 1, LAST_VALUE = 100;
const double LAT = -1.23456, LON = -100.001234;
const char LETTER = 'C';
```

ลักษณะการกำหนดค่าคงที่ที่ไม่ถูกต้อง เช่น

```
const float PI;  
PI = 3.141;           //Error ต้องกำหนดค่าในชั้นตอนการประกาศ  
                      //จะกำหนดค่าในภายหลังไม่ได้  
  
const short MAXIMUM = 500;  
MAXIMUM = 1000;      //Error จะแก้ไขค่าในภายหลังไม่ได้
```

#### การกำหนดค่าคงที่ตัวแยกรากชี้ define

**define** เป็นคำส่วนอีกอันหนึ่งของภาษา C ซึ่งในการสร้างนิยาม หรือข้อกำหนดบางอย่าง (เรียกว่า มาโคร) รวมถึงการสร้างค่าคงที่ก็ เช่นเดียวกัน โดยในที่นี้จะกล่าวถึงเฉพาะกรณี

การกำหนดค่าคงที่เท่านั้น ส่วนในการใช้ define สำหรับกรณีการกำหนดมาโดยจะกล่าวถึงในบทที่ 15 โดยการสร้างค่าคงที่ด้วย define จะใช้รูปแบบดังนี้

```
#define ชื่อค่าคงที่ ค่าที่กำหนด
```

- **#define** เป็นการบ่งชี้ว่าเราจะใช้กำหนดค่าคงที่ โดยต้องมีเครื่องหมาย # กำกับที่ ข้างหน้า define (ลักษณะเช่นนี้เรียกว่า Preprocessor ซึ่งจะได้เรียนรู้ในบทที่ 15)
- **ไม่** ต้องระบุชนิดข้อมูล
- การตั้งชื่อค่าคงที่ ก็ใช้หลักการเดิม (คล้ายกับ const)
- **ไม่** ต้องใช้เครื่องหมาย = เพื่อกำหนดค่า
- ก่อนกำหนดค่า ให้เว้นช่องว่างอย่างน้อย 1 ช่อง
- ค่าที่กำหนดจะเป็นข้อมูลชนิดใดก็ได้ แม้กระทั่งสตริง (ข้อความ) โดยหากเป็นข้อมูล ชนิดตัวเลขหรืออักษร จะใช้หลักการเดิม แต่หากเป็นสตริง (อักษรมากกว่า 1 ตัว) ให้กำหนดด้วยเครื่องหมาย " " เท่านั้น
- **ไม่** ต้องใช้เครื่องหมาย ; เพื่อปิดท้ายคำสั่ง
- โดยทั่วไปการกำหนดคงที่ด้วย define มักนำไปใช้ร่วมกันตลอดทั้งโปรแกรม ดังนั้น เรา นิยมกำหนดไว้ก่อนฟังก์ชัน main() หรือถัดจากส่วน include นั่นเอง แต่ไม่ใช้ข้อบังคับ
- เนื่องจากในกรณีนี้ เราไม่ได้กำหนดชนิดข้อมูลของค่าคงที่ แต่ชนิดข้อมูลจะขึ้นกับค่าที่ เรากำหนดให้กับมัน เช่น ถ้าเรากำหนดเป็นเลขจำนวนเต็ม ค่าคงที่นั้นก็อาจเป็นชนิด int ซึ่งมันจะส่งผลเมื่อเราคำนวณที่นั้นไปใช้งาน ทั้งนี้หากไม่สอดคล้องกับชนิดข้อมูล ที่มันจัดเก็บเอาไว้ ก็อาจเกิดข้อผิดพลาดขึ้นได้

แนวทางการกำหนดค่าคงที่ด้วยคำสั่ง define เช่น

```
#include <stdio.h>

#define PI 3.14159
#define LUCKY_NUMBER 13
#define LETTER 'C'
#define PROGRAM_NAME "Business Management"
#define VERSION "1.0.1"

void main() {
    //...
}
```

แนวทางการกำหนดค่าคงที่ด้วยคำสั่ง define ที่ไม่ถูกต้อง เช่น

```
#define float PI 3.14159 //Error ไม่ต้องระบุชนิดข้อมูล  
  
#define LUCKY_NUMBER = 13 //Error ไม่ต้องใช้เครื่องหมาย =  
  
#define LETTER 'C'; //Error ไม่ต้องปิดท้ายด้วย ;  
  
#define PROGRAM_NAME //Error ต้องกำหนดค่าให้มันทันที
```

ลิ่งสำคัญที่เราได้เรียนรู้ในบทนี้ก็คือ ชนิดข้อมูลพื้นฐานในภาษา C และขอบเขตของมัน พร้อมกับการนำชนิดข้อมูลเหล่านั้นมาใช้ในการประกาศตัวแปร รวมถึงการกำหนดค่าให้กับมัน ซึ่งลิ่งเหล่านี้ต้องสอดคล้องกัน เช่น หากเป็นตัวแปรชนิด short ก็ต้องกำหนดค่าที่เป็นเลขจำนวนเต็ม ที่อยู่ระหว่าง -32768 ถึง 32767 เป็นต้น ตลอดจนการสร้างค่าคงที่ด้วย const และ define ที่จะต้องใช้ค่านั้นไปตลอด จะเปลี่ยนแปลงในภายหลังไม่ได้ ซึ่งทั้งหมดนี้ถือเป็น หลักการที่เราต้องคำนึงถึงและนำไปใช้งานอยู่ตลอดในการเขียนโค้ดภาษา C





# สตริง การรับและแสดงผลข้อมูล

ในบทนี้ลิ่งที่เราจะได้เรียนรู้คือ ข้อมูลชนิดอักขระ สตริง รวมถึงการรับและแสดงผลข้อมูล ในลักษณะต่างๆ ซึ่งถือเป็นพื้นฐานที่สำคัญยิ่งของภาษา C เพราะสตริงเป็นข้อมูลหลัก ที่ถูกใช้งานบ่อยที่สุด และการแสดงผลก็ต้องทำความคู่กันไปตลอด ดังนั้น เรายังคงมาเรียนรู้เรื่องนี้ให้เกิดความเข้าใจที่ดีพอในระดับหนึ่ง มีอะไรบ้างที่จะเกิดปัญหาในการเรียนรู้เรื่องต่อๆ ไปได้

## อักขระและสตริง

เราได้ทราบไปแล้วว่า ข้อมูลชนิด char จะจัดเก็บอักขระเพียงตัวเดียว โดยกำหนดอักขระไว้ในเครื่องหมาย '' เพ่านั้น เช่น

```
char letter1 = 'a';
char letter2 = 'C';
char nl = '\n';           /* \n จัดเป็นอักขระ 1 ตัว */
```

หากเราต้องการแสดงข้อมูลชนิดอักขระ วิธีที่ง่ายที่สุดคือใช้ฟังก์ชัน putchar() ทั้งนี้ เราจะใช้ฟังก์ชัน puts() ไม่ได้ และหากจะใช้ printf() ต้องมีการจัดรูปแบบซึ่งจะกล่าวถึงในภายหลัง โดยแนวทางการแสดงอักขระด้วยฟังก์ชัน putchar() เช่น

```
char vowel1 = 'a';
char vowel2 = 'e';

putchar(vowel1);
putchar('\n');      //ขึ้นบรรทัดใหม่
putchar(vowel2);

puts(vowel1);      //Error
printf(vowel2);    //Error (ต้องจัดรูปแบบ)
```

อย่างไรก็ตาม ในการใช้งานจริงนั้น ข้อมูลส่วนใหญ่มักประกอบด้วยอักษรมากกว่า 1 ตัว ที่จัดเรียงต่อเนื่องกันในรูปแบบของข้อความหรือสตริง (String) แต่ในภาษา C ไม่มีชนิดข้อมูล สำหรับใช้กับสตริงโดยตรงเหมือนที่มีในหลาย ๆ ภาษา โดยเราต้องจัดการสตริงบนพื้นฐานของ ข้อมูลชนิด char ตามหลักการที่ว่า

สตริง เป็นการนำอักษรมาจัดเรียงต่อเนื่องกันไปตามลำดับ

หรือกล่าวอีกอย่างได้ว่า สตริง คือชุด ข้อมูลชนิดอักษร เช่น คำว่า THAILAND หรือ Hello World เกิดจากการนำอักษรมาจัดเรียงต่อ เนื่องกันดังภาพ



ความจริงแล้ว ชุดข้อมูลที่มีการจัดเรียงตามลำดับดังกล่าว เราเรียกว่า อาร์เรย์ (Array) ซึ่ง เป็นเนื้อหาของบทที่ 10 แต่หากกล่าวถึงในแบบอาร์เรย์ ก็คงว่าผู้อ่านจะสับสน ดังนั้น จะแทน ด้วยคำว่า "ชุดข้อมูล" ไปก่อน โดยใช้พื้นฐานเพียงบางส่วนของอาร์เรย์เท่านั้น ซึ่งในภาษา C จะ จัดการกับข้อมูลประเภทสตริงในแบบชุดอักษร ด้วยวิธีการดังต่อไปนี้

- สร้างตัวแปร โดยระบุชนิดข้อมูลเป็น char เพราะเป็นอาร์เรย์ของอักษร
- กำหนดตัวแปร พร้อมขนาด (จำนวนอักษร) ที่มีได้สูงสุดสำหรับสตริงที่จะจัดเก็บใน ตัวแปรนั้น โดยระบุตัวเลขไว้ในวงเล็บ [] ต่อท้ายชื่อตัวแปร เพื่อเป็นการจงพื้นที่ใน การจัดเก็บข้อมูล
- ต้องกำหนดค่าที่เป็นสตริง (ใช้เครื่องหมาย " ") ให้กับตัวแปรในขั้นตอนประกาศทันที จะกำหนดค่าในภายหลังไม่ได้

```
char name[30] = "John Smith";
char address[150] = "London UK ...";
char email[80] = "john@example.com",
career[30] = "Programmer";
```

- ถ้าเราไม่ต้องการระบุขนาดหรือจำนวนอักษรไว้แบบตายตัว ยังมีวิธีที่ยืดหยุ่นกว่าคือ ไม่ต้องกำหนดตัวเลขลงไป โดยให้มีเพียงวงเล็บ [] ว่าง ๆ ช่องกรณีนี้ เราจะกำหนด สตริงที่มีความยาวเท่าไหร่ก็ได้ เช่น

```
char name[] = "John Smith";
char address[] = "London UK ...";
char email[] = "john@example.com",
career[] = "Programmer";
```

- นอกจากจะต้องกำหนดค่าในขั้นตอนการประกาศแล้ว เรายังไม่สามารถแก้ไขค่าในภายหลังได้อีกด้วย (เนื่องจากสตริงเป็นอาร์เรย์ ถ้าจะแก้ไขต้องใช้วิธีการแบบอาร์เรย์) นอกจากนี้ยังไม่สามารถนำค่าจากตัวแปรหนึ่งไปกำหนดให้กับตัวแปรหนึ่งอีกด้วย (ให้ใช้ฟังก์ชัน strcpy() ที่จะกล่าวถึงในหัวข้อถัดไป) เช่น

```
char text[100];
text = "Hello World";           //Error ต้องกำหนดค่า
//ในขั้นตอนการประกาศ

char message[200] = "Sawasdee Thai people";
message = "Hello Thai people"; //Error แก้ไขค่าสตริงไม่ได้

char str1[] = "I can C";
char str2[] = str1;             //Error ใช้ strcpy() แทน
```

- ลักษณะการจัดเก็บข้อมูลที่แท้จริงของตัวแปรในแบบชุดอักขระคือ จะจัดเรียงอักขระต่อเนื่องกันไปตามลำดับ และปิดท้ายด้วยสัญลักษณ์ '\0' เพื่อบ่งชี้ว่าเป็นจุดสิ้นสุดสตริง แต่อักขระตัวปิดท้ายนี้ ตัวแปลภาษาจะใส่ให้เราเอง โดยที่เราจะมองไม่เห็นมัน
- ถ้าเราระบุตัวเลขในวงเล็บ [] แสดงว่า จำนวนอักขระสูงสุดที่ตัวนั้นจัดเก็บในตัวแปรนั้นได้เท่ากับ (ตัวเลขที่ระบุ - 1) เนื่องจากตำแหน่งสุดท้ายต้องใช้เก็บ \0 เช่น



```
char name[50] = "...";      //จะเก็บได้สูงสุด 49 อักขระ
char address[200] = "..."; //จะเก็บได้สูงสุด 199 อักขระ
char email[101] = "...";   //จะเก็บได้สูงสุด 100 อักขระ
```

- หากเรากำหนดขนาดหรือจำนวนอักขระไว้ในวงเล็บ [] ต่อท้ายชื่อตัวแปร จะมีผลดังนี้
  - ถ้ากำหนดสตริงที่มีความยาวหรือจำนวนอักขระ เกินค่าที่ระบุ จะเกิดข้อผิดพลาด หรือมีคำเตือน แล้วแสดงผลเท่ากับจำนวนความยาวที่ระบุวงเล็บ [] และอาจมีอักขระเปลกประหลาดมาต่อท้ายสตริง
  - ถ้ากำหนดสตริงที่มีความยาวหรือจำนวนอักขระ เท่ากับค่าที่ระบุ จะไม่เกิดข้อผิดพลาด แต่อาจมีอักขระเปลกประหลาดมาต่อท้ายสตริง
  - ถ้ากำหนดสตริงที่มีความยาวหรือจำนวนอักขระ น้อยกว่าค่าที่ระบุ จะไม่มีปัญหาใด ๆ

- ช่องว่าง 1 ช่องเรานับเป็น 1 อักขระ ดังนั้น หากมี 5 ช่องว่างก็นับเป็น 5 อักขระ
- เราสามารถนำค่าจากตัวแปรแบบชุดอักขระไปแสดงผลด้วยฟังก์ชัน puts() และ printf() ได้โดยตรง

สำหรับแนวทางการใช้งานให้ดูจากตัวอย่างต่อไปนี้ ดังต่อไปนี้ โดยตัวอย่างทั้งหมดในบทนี้ จะจัดเก็บไว้ที่ C:\c-lang\chapter4 ซึ่งเราจะสร้างไฟล์ของแต่ละตัวอย่างตามที่ได้กล่าวไว้แล้วในบทที่ 1

**ตัวอย่าง 4-1 การกำหนดตัวแปรแบบชุดอักขระ เพื่อจัดเก็บข้อมูลแบบสตริง พิรุณแนวทางการแสดงผล**

```
#include <stdio.h>

void main() {
    char firstname[] = "Elon", lastname[] = "Musk",
        address1[] = "Palo Alto California",
        address2[] = "United States of America",
        email[] = "elon_musk@example.com",
        career[] = "Tesla & SpaceX CEO";

    printf("\nName: ");
    printf(firstname);
    printf(" ");
    printf(lastname);
    putchar('\n');

    printf("Career: ");
    printf(career);
    putchar('\n');
}
```

Name: Elon Musk  
Career: Tesla & SpaceX CEO

**ตัวอย่าง 4-2 เป็นการทดสอบว่า หากเรากำหนดสตริงให้มีขนาดหรือจำนวนอักขระต่างจากค่าตัวเลขที่กำหนดไว้ในวงเล็บ [] จะส่งผลอย่างไร**

```
#include <stdio.h>

void main() {
    char str1[10] = "1234567890123"; // ยาว 13
    char str2[10] = "1234567890"; // ยาว 10
    char str3[10] = "1234578"; // ยาว 8
```

```

printf("str1 ==> ");
printf(str1);
putchar('\n');

printf("str2 ==> ");
printf(str2);
putchar('\n');

printf("str3 ==> ");
printf(str3);
putchar('\n');
}

```

```

C:\c-lang>cd "c:\c-lang\chapter4\" && gcc example4-2.c -o example4-2 && "c:\c-
example4-2.c: In function 'main':
example4-2.c:4:21: warning: initializer-string for array of chars is too long
    char str1[10] = "1234567890123";           ^
str1 ==> 1234567890†
str2 ==> 12345678901234567890†
str3 ==> 1234578

```

## การกำหนดและแก้ไขค่าของตัวแปรสตริง

จากที่ได้กล่าวไปในหัวข้อที่แล้ว จะพบปัญหางานอย่างคือ เราต้องกำหนดค่าให้กับตัวแปรแบบสตริง (ชุดอักษร) ตั้งแต่ขั้นตอนการประกาศ และนอกจากนี้ยังแก้ไขค่าในภายหลังโดยการกำหนดค่าใหม่แทนที่ค่าเดิมโดยตรงไม่ได้ ซึ่งเป็นปัญหาสำคัญที่เราจำเป็นต้องรู้วิธีการแก้ไข เพราะในบางครั้งเรามิ่งสามารถกำหนดค่าตัวแปรเอาไว้ล่วงหน้าแบบด้วยตัวได้ นอกเหนือไปในบางสถานการณ์ เราจำเป็นต้องเปลี่ยนแปลงค่าของตัวแปรแบบสตริงจากค่าหนึ่งไปเป็นอีกค่าหนึ่ง เพื่อให้สอดคล้องกับเงื่อนไขในแต่ละขณะ ซึ่งเป็นสิ่งที่เกิดขึ้นอยู่บ่อยครั้ง สำหรับแนวทางการแก้ไขปัญหาดังที่กล่าวมา สามารถใช้ฟังก์ชันที่จะกล่าวถึงดังต่อไปนี้

### การใช้ฟังก์ชัน strcpy()

ฟังก์ชัน strcpy() มาจากคำว่า string copy ตามปกติเราจะใช้ฟังก์ชันนี้ในการคัดลอกสตริงที่ระบุ หรือค่าจากตัวแปรหนึ่ง และนำมากำหนดค่าให้กับอีกตัวแปรหนึ่ง โดยใช้รูปแบบดังนี้

```
strcpy(ตัวแปรสตริง, "สตริงที่จะคัดลอก")
```

- พังก์ชัน `strcpy()` อยู่ในเขตเดอร์ **string.h** ดังนั้น เราจำเป็นต้องอ้างอิงด้วย `include` ก่อนการใช้งาน
- เราต้องประกาศตัวแปรสตริงที่จะใช้จัดเก็บสตริงที่คัดลอกมา เอาไว้ล่วงหน้า
- สตริงที่จะคัดลอกมา อาจกำหนดค่าโดยตรง หรือจะเป็นค่าที่จัดเก็บอยู่ในตัวแปรอื่น ๆ ได้ (ระบุชื่อตัวแปรนั้นลงไปแทนสตริงที่จะคัดลอก)
- ถ้ามีข้อมูลที่จัดเก็บตัวแปรนั้นเอาไว้อยู่ก่อนแล้ว ข้อมูลจะถูกแทนที่ด้วยค่าสตริงใหม่ ที่ระบุ

```
char str[100];
strcpy(str, "Hello World"); //str = "Hello World"

char greeting[20] = "Sawasdee";
char msg[] = "Hello";
strcpy(greeting, msg);      //greeting = "Hello"
```

- ข้อควรระวังคือ ขนาดของตัวแปรสตริง ต้องสามารถรองรับ (มีขนาดมากกว่า) สตริงใหม่ที่จะคัดลอกไป มิฉะนั้นสตริงอาจถูกคัดลอกไปเพียงบางส่วนเท่าที่เป็นไปได้ หรือไม่เกินขนาดของตัวแปร หรือสตริงอาจถูกคัดลอกไปทั้งหมดแต่มีคำแจ้งเตือนเมื่อรันโค้ด โดยลักษณะเช่นนี้อาจขึ้นกับตัวแปลงภาษา C ที่เราใช้ประมวลผล เช่น

```
char str[5];
strcpy(str, "Hello World");
//ตัวแปลง str ควรเก็บคำว่า Hell
//เท่ากับขนาด [5 - 1] แต่ตัวแปลงภาษา C บางอัน
//อาจคัดลอกไปได้ทั้งหมด และมีคำแจ้งเตือน
```

- เพื่อหลีกเลี่ยงปัญหาดังที่กล่าวมานี้ หากตัวแปรใด ที่เราจะใช้รองรับสตริงที่แก้ไขค่าได้ในภายหลัง ควรกำหนดขนาดเพื่อเอาไว้ให้ใหญ่พอสมควร ซึ่งสามารถรองรับข้อความที่ยาว ๆ ได้

จากที่กล่าวมา จะเห็นว่า พังก์ชัน `strcpy()` สามารถแก้ไขปัญหาได้ทั้งกรณีที่เรามีความสามารถกำหนดค่าตัวแปรแบบสตริงไว้ล่วงหน้าดังแต่ขั้นตอนการประกาศ และกรณีการเปลี่ยนแปลงค่าของตัวแปรในภายหลัง ซึ่งขอให้ดูเพิ่มเติมจากตัวอย่างต่อไปนี้

**ตัวอย่าง 4-3 การกำหนดและเปลี่ยนแปลงค่าของตัวแปรด้วยฟังก์ชัน strcpy()**

```
#include <stdio.h>
#include <string.h>

void main() {
    char str1[100];
    char str2[100] = "";

    putchar('\n');
    strcpy(str1, "Can you see?");
    printf("str1 = ");
    printf(str1);

    putchar('\n');
    strcpy(str2, "Yes, I can C");
    printf("str2 = ");
    printf(str2);
    putchar('\n');
}
```

str1 = Can you see?  
 str2 = Yes, I can C

### การใช้ฟังก์ชัน strcat()

strcat() มาจากคำว่า string concatenate ใช้ในการนำสตริงใหม่มาต่อท้ายกับสตริงเดิมที่จัดเก็บในตัวแปร (กรณีของฟังก์ชัน strcpy จะเป็นการแทนที่) โดยใช้รูปแบบดังนี้

```
strcat(ตัวแปรสตริง, "สตริงที่จะนำไปต่อท้าย")
```

- strcat() อยู่ในヘดเดอร์ **string.h** เช่นเดียวกับ strcpy() ดังนั้น จึงต้องอ้างอิงไฟล์นี้ด้วย include จึงจะใช้งานได้
- หลักการต่างๆ ของ strcat() ทั้งตัวแปรสตริง และสตริงที่จะนำไปต่อท้ายก็พิจารณาในแนวทางเดียวกับ strcpy() เช่น

```
char str[50] = "Hello ";
strcat(str, "World"); //str = "Hello World"

char msg[100];
strcat(msg, "Sawasdee"); //msg = "Sawasdee"
```

```

char a[100] = "Be better than";
char b[] = "that you can be";
strcat(a, " ");
strcat(a, b);
//a = "Be better than that you can be"

```

**ตัวอย่าง 4-4 การกำหนดและเชื่อมต่อค่าของตัวแปรด้วยฟังก์ชัน strcat()**

```

#include <stdio.h>
#include <string.h>

void main() {
    char str[100] = "Happy";

    strcat(str, " ");
    strcat(str, "New Year");
    puts(str); //Happy New Year

    strcpy(str, "Happy ");
    strcat(str, "Birthday");
    puts(str); //Happy Birthday
}

```

## อักขระพิเศษ (Escape Sequence)

อักขระบางตัว ไม่มีอยู่บนปุ่มคีย์บอร์ด หรืออักขระบางตัว เราจะเขียนลงในสตริงโดยตรงไม่ได เนื่องจากใช้เป็นข้อกำหนดบางอย่างในภาษา C ดังนั้น จึงมีการกำหนดลัญลักษณ์พิเศษเพื่อใช้แทนอักขระเหล่านี้ โดยลัญลักษณ์ดังกล่าวจะขึ้นต้นด้วยเครื่องหมาย \' และตามด้วยอักขระอื่น ๆ เพื่อให้มีความหมายหรือวัตถุประสงค์ที่แตกต่างกันออกไป ซึ่งเราเรียกลัญลักษณ์หรืออักขระพิเศษในกลุ่มนี้ว่า Escape Sequence โดยตัวที่นำสนใจซึ่งเรามีโอกาสได้นำไปใช้งานอยู่บ่อย ๆ มีดังนี้

อักขระ	ชื่อ	คำอธิบาย
\n	New Line	ขึ้นบรรทัดใหม่ ดังที่เราเคยใช้กันมาแล้ว
\a	Alert	ให้มีเสียง Beep 1 ครั้ง
\b	Backspace	เลื่อนเคอร์เซอร์ย้อนกลับไป 1 ตำแหน่ง
\r	Carriage Return	เลื่อนเคอร์เซอร์ไปที่ตำแหน่งเริ่มต้นบรรทัด

อักษร	ชื่อ	คำอธิบาย
\t	Horizontal Tab	เว้นระยะในแนวนอนเท่ากับ 1 แท็บ
\'	Single Quote	แสดงเครื่องหมาย '
\"	Double Quote	แสดงเครื่องหมาย "
\?	Question Mark	แสดงเครื่องหมาย ?
\\\	Backslash	แสดงเครื่องหมาย \

**ตัวอย่าง 4-5** แนวทางการใช้อักษรพิเศษที่น่าสนใจอันเพื่อแสดงผลลัพธ์คล้ายกับตารางโดยจะแทรกอักษรระเอียดี้ลงในสตริงที่แสดงผลด้วย printf()

```
#include <stdio.h>
```

Evolution of "C" language

Language	Year Created	Creator
BCPL	1967	Martin Richards
B	1970	Ken Thompson
C	1972	Dennis Ritchie

```
void main() {
    printf("\nEvolution of \"C\" language\n\n");
    printf("Language\tYear Created\tCreator\n");
    printf("-----\n");
    printf("BCPL\t\t1967\t\tMartin Richards\n");
    printf("B\t\t\t1970\t\tKen Thompson\n");
    printf("C\t\t\t1972\t\tDennis Ritchie\n");
}
```

## การจัดรูปแบบสตริงสำหรับการแสดงผล

ที่ผ่านมาแล้ว เราได้เรียนรู้ไปแล้วว่า พิมพ์ชั้นพื้นฐานที่ใช้แสดงผลข้อมูลแบบสตริงก็คือ puts() และ printf() โดยที่ puts() ใช้แสดงสตริงที่กำหนดเอาไว้อย่างแน่นอน ในขณะที่ printf() สามารถแสดงได้ทั้งสตริงที่มีกำหนดค่าไว้อย่างแน่นอนและสตริงที่มีการจัดรูปแบบ ด้วยเหตุดังกล่าว เราจึงอาจพบเห็นการใช้ printf() ได้มากกว่า puts() และในหัวข้อนี้ เราจะมาเรียนรู้วิธีการจัดรูปแบบสตริงที่จะแสดงผลด้วยพิมพ์ชั้น printf() ซึ่งถือเป็นลักษณะพื้นฐานที่สำคัญอีกอย่างหนึ่งของภาษา C โดยมีรายละเอียดดังต่อไปนี้

## การจัดรูปแบบในเบื้องต้นสำหรับ printf()

บางครั้ง ลดริบหรือข้อความที่เราต้องการแสดงผล ไม่สามารถกำหนดให้ครบสมบูรณ์ เอาไว้ล่วงหน้าได้ แต่อาจต้องนำข้อมูลจากที่อื่นๆ มาเติมลงไปด้วย เพื่อให้สอดคล้องกับเงื่อนไข ในแต่ละขณะ เช่น คำว่า

My name is \_\_\_\_\_ from \_\_\_\_\_ and I'm \_\_\_ years old.

จากข้อความ ล้วนที่เป็นขีดล่างซึ่งเว้นเอาไว้ันคือ ข้อมูล ที่เราจะนำมาเติมลงไปในประโยคนี้ แต่อย่างไรก็ตาม ถ้าลิ่งที่จะนำมาเติมอาจเป็นไปได้หลากหลาย ครั้นเราจะใช้วิธีการนำสตริง และข้อมูลมาเชื่อมต่อกัน ก็จะยุ่งยากจนเกินไป ดังนั้น ในภาษา C จึงมีวิธีการแทรกข้อมูลลงใน สตริง ก่อนจะแสดงผลด้วย printf() แต่อย่างไรก็ตาม เนื่องจากข้อมูลที่จะแทรกสามารถเป็นไป ได้หลายชนิด เช่น เลขจำนวนเต็ม ทศนิยม อักขระ หรือ สตริง (ชุดอักขระ) ซึ่งข้อมูลแต่ละชนิด จะใช้วิธีดำเนินการที่แตกต่างกัน ดังนั้น เพื่อให้ตัวประมวลผลสามารถจัดเตรียมวิธีการที่ถูกต้อง และสอดคล้องกับชนิดข้อมูลที่จะแทรก เราต้องบอกให้มันรู้ล่วงหน้าว่าเราจะแทรกข้อมูลชนิดใด ลงไป โดยแทนข้อมูลแต่ละชนิดด้วยสัญลักษณ์ดังต่อไปนี้

สัญลักษณ์	ความหมาย
%c	(char) สำหรับค่าที่เป็นอักขระตัวเดียว
%d	(decimal) สำหรับค่าที่เป็นเลขจำนวนเต็มทุกชนิด (ยกเว้น long long)
%e	(exponential) สำหรับแสดงตัวเลขแบบวิทยาศาสตร์ เช่น 1.23e4
%f	(float) สำหรับเลขทศนิยมทั้งชนิด float และ double ซึ่งโดยทั่วไป ผลลัพธ์ที่แสดง จะเป็นเลขที่มีทศนิยม 6 ตำแหน่ง ดังนั้น อาจมีการเติมเลข 0 ต่อท้ายให้ทศนิยมครบ 6 ตำแหน่ง แต่ถ้าเป็นเลขที่มีทศนิยมเกิน 6 ตำแหน่ง จะตัดให้เหลือแค่ 6 ตำแหน่ง
%0.?f	(float) สำหรับเลขทศนิยมทั้งชนิด float และ double โดยให้ ? แทนตัวเลขที่ใช้กำหนด จำนวนตำแหน่งทศนิยม เช่น %0.3f หมายถึง ให้มีทศนิยม 3 ตำแหน่ง (อาจมีการปัด ค่าของตัวเลข)
%g	สำหรับแสดงตัวเลขที่คล้ายกับ %e หรือ %f ในแบบที่ลั้นที่สุด กรณีที่เป็นเลขทศนิยม และต้องการให้แสดงผลลัพธ์ตรงกับค่าจริง (ไม่ตัด 0 ต่อท้ายทศนิยม ผู้เขียนแนะนำ ใช้ %g แทน %f)
%s	(string) สำหรับค่าที่เป็นสตริง
%%	แสดงเครื่องหมาย %

ลัญลักษณ์	ความหมาย
%lld	สำหรับค่าที่เป็นเลขจำนวนเต็มชนิด long long
%hu	สำหรับค่าที่เป็นเลขจำนวนเต็มชนิด short (ใช้ %d แทนได้)

รูปแบบโดยทั่วไปของฟังก์ชัน printf() ที่มีการจัดรูปแบบสติงคือ

```
printf("สติงที่จัดรูปแบบ", ค่าที่_1, ค่าที่_2, ค่าที่_3, ...)
```

- สติงที่จะจัดรูปแบบ ก็คือสติงที่เราจะแสดงผล ซึ่งภายในสติงนี้ จะมีลัญลักษณ์ ตามชนิดข้อมูลที่จะนำมาแทรกปะบันอยู่ด้วย โดยทางตำแหน่งลัญลักษณ์ ให้ตรงตาม ที่ต้องการแสดงผล
- ค่าที่\_1, ค่าที่\_2, ... คือค่าที่เราจะนำไปแทรกลงในสติง โดยเรียงลำดับให้ตรงกับลำดับ ลัญลักษณ์ที่วางไว้ในสติง และชนิดข้อมูลต้องสอดคล้องกับลัญลักษณ์ มิฉะนั้น อาจ ได้ผลลัพธ์ที่ไม่ถูกต้อง หรือบางกรณีอาจเกิดข้อผิดพลาด

```
printf("...%?....%?....%?...", value1, value2, value3)
```

แนวทางการใช้ลัญลักษณ์เพื่อแทรกข้อมูลในเมืองต้น เช่น

```
printf("%d", 12345);           //12345
printf("%f", 123.456);         //123.456000
printf("%g", 123.456);         //123.456

printf("I can %c", 'C');       //I can C
printf("%s", "I can C");

char color[] = "red";
printf("My car is %s", color); //My car is red

char happy[] = "Happy %s";
printf(happy, "Birthday");    //Happy Birthday
printf(happy, "New Year");   //Happy New Year
```

ในสตริงเดียวกัน สามารถแทรกสัญลักษณ์ได้มากกว่า 1 ตำแหน่ง ประปนอยู่ในสตริง รวมถึงอาจมีการคำนวณข้อมูลก่อนนำไปแทรก ซึ่งขอให้ดูเพิ่มเติมจากตัวอย่างดังต่อไปนี้

#### ตัวอย่าง 4-6 แนวทางการจัดรูปแบบ

สตริงสำหรับฟังก์ชัน printf() โดยแทรกสัญลักษณ์หลาย ๆ แบบไว้ในสตริงเดียวกัน

```
#include <stdio.h>
```

```
void main() {
    printf("\nThe lucky number is %d", 13);
    printf("\nPI (f) = %f", 3.141);
    printf("\nPI (0.2f) = %0.2f", 3.141);
    printf("\nPI (g) = %g", 3.141);

    printf("\nHi, I'm %s %s, creator of %c language",
           "Dennis", "Ritchie", 'C');

    printf("\nKhun %s please contact counter number %d",
           "Banchar", 5);

    char word[100] = "\nHappy %s";
    char ny[] = "New Year";
    char bd[] = "Birthday\n";
    printf(word, ny);
    printf(word, bd);
}
```

```
The lucky number is 13
PI (f) = 3.141000
PI (0.2f) = 3.14
PI (g) = 3.141
Hi, I'm Dennis Ritchie, creator of C language
Khun Banchar please contact counter number 5
Happy New Year
Happy Birthday
```

#### ตัวอย่าง 4-7 การจัดรูปแบบสตริง โดยค่าที่นำไปเติม จะได้จากการคำนวณแบบง่าย ๆ ซึ่งกำหนดไว้ที่ฟังก์ชัน printf() ก่อนนำไปแทรกลงในสตริง

```
#include <stdio.h>
#include <string.h>

void main() {
    printf("\n%d + %d = %d\n", 10, 20, (10 + 20));

    int year_born = 2000;
    int current_year = 2023;
    char str[200] = "\nI was born in year %d ";
```

```
10 + 20 = 30
I was born in year 2000
and current year is 2023
so I'm 23 years old
```

```

        strcat(str, "\nand current year is %d");
        strcat(str, "\nso I'm %d years old\n");
    }

}

```

## การกำหนดความกว้างของสตริง

ตามปกตินั้นขนาดความกว้างของสตริงที่แสดงผล จะเท่ากับจำนวนอักขระในสตริงนั้น แต่บางครั้งเราต้องการจัดรูปแบบโดยให้ความกว้างของพื้นที่แสดงผลมีขนาดแน่นอน เช่น การแสดงผลแบบหลายบรรทัดและหลายคอลัมน์คล้ายกับตาราง ก็สามารถจัดแนวข้อความให้ตรงหรือให้เยื้องตามต้องการได้ โดยการกำหนดความกว้างของสตริง จะใช้รูปแบบพื้นฐานดังนี้

%พ้องขระของชนิดข้อมูล

- ในที่นี้ จะให้ w แทนเลขจำนวนเต็ม ซึ่งใช้ระบุขนาดความกว้างที่ต้องการ
- อักขระของชนิดข้อมูล ก็ขึ้นกับชนิดข้อมูลที่เราจะนำมาแทรกลงในสตริง ดังที่กล่าวไปในหัวข้อที่แล้ว เช่น %5d, %7s หรือ %10f
- หากข้อมูลที่นำมาแทรกมีจำนวนอักขระ น้อยกว่า ขนาดความกว้างที่ระบุ สตริงจะถูก **จัดซีดขวา** (มีช่องว่างทางด้านซ้ายเกิดขึ้นตามจำนวนอักขระที่ต่างกัน)
- หากข้อมูลที่นำมาแทรกมีจำนวนอักขระ มากกว่า ขนาดความกว้างที่ระบุ ข้อมูลจะถูก นำมาย่อสั้นไปจนครบทุกอักขระโดยไม่เกิดข้อผิดพลาดใดๆ

สำหรับแนวทางการใช้งานในเบื้องต้น จะเปรียบเทียบให้เห็นลักษณะการจัดวางตำแหน่ง อักขระเมื่อเทียบกับความกว้างที่ระบุ

```

printf("%10s", "Thailand");
//ให้พื้นที่แสดงผลมีความกว้าง 10 อักขระ
//แต่คำว่า "Thailand" มีเพียง 8 อักขระ
//จึงจัดซีดขวา (มีช่องว่างด้านซ้าย)

printf("%7d", 1234);
printf("%5f", 12.345);

```

	T	h	a	i	l	a	n	d
	1	2	3	4				
1	2	.	3	4	5	0	0	0

จากการณ์ที่จำนวนอักขระของข้อมูลที่นำมาแทรกน้อยกว่าความกว้างที่ระบุ สริงจะถูกจัดชิดขวา แต่หากเราต้องการให้ชิดซ้าย ก็ใส่เครื่องหมายลบ (-) ข้างหน้าความกว้าง ในรูปแบบ

%-พ้อกขระของชนิดข้อมูล
------------------------

```
printf("%-10s", "Thailand");
printf("%-7d", 1234);
```

T	h	a	i		a	n	d		
1	2	3	4						

กรณีที่เป็นเลขศูนย์ (f) ถ้าเราไม่ระบุจำนวนตำแหน่งทศนิยม มันจะปรับให้มีทศนิยม 6 หลัก ดังໂດຍและภาพก่อนนี้ แต่เราสามารถกำหนดให้มีจำนวนทศนิยมตามต้องการได้โดยใช้รูปแบบ

%w.?f      หรือ      %-w.?f      (จัดชิดซ้าย)
---

เมื่อ ? คือตัวเลขซึ่งแสดงจำนวนตำแหน่งทศนิยมที่ต้องการ หรือถ้าต้องการจัดชิดซ้าย ก็ใส่เครื่องหมายลบข้างหน้าความกว้าง อย่างไรก็ตาม ค่าที่แสดงผลออกมา อาจถูกบัดเศษให้สอดคล้องกับจำนวนตำแหน่งทศนิยมที่ระบุเมื่ອอนเซ่นเคย ดังแนวทางต่อไปนี้

```
printf("%10.2f\n", 12345.678);
printf("%-10.2f\n", 12345.678);
printf("%10.2f", -12345.678);
```

		1	2	3	4	5	.	6	8
1	2	3	4	5	.	6	8		
-	1	2	3	4	5	.	6	8	

หรือในกรณีที่เป็นตัวเลขจำนวนเต็ม บางครั้งเราต้องการแสดงผลโดยให้มีตัวเลขครบตามจำนวนที่ต้องการ โดยส่วนใหญ่เราจะใช้วิธีเติม 0 ข้างหน้า เช่น ถ้าต้องการเลข 3 หลัก แต่ค่าที่ได้คือ 7 ก็อาจเติม 0 ข้างหน้าเป็น 007 เป็นต้น ซึ่งก็อาจกำหนดรูปแบบดังนี้ (ให้ w คือตัวเลขที่ใช้ระบุความกว้างหรือจำนวนหลักที่ต้องการ)

%0wd      หรือ      %0wi
--------------------------

```
int code = 7;
int d = 9, m = 7, y = 2023;

printf("%03d", code); //007
printf("%02d/%02d/%04d", d, m, y); /* 09/07/2023 */
```

**ตัวอย่าง 4-8 การจัดรูปแบบสตริง โดยกำหนดความกว้างในลักษณะที่แตกต่างกัน**

```
#include <stdio.h>

void main() {
    printf("\n%5d", 555);
    printf("\n%-5d", 555);
    printf("\n%-5d\n", -555);

    printf("\n%10.2f", 12345.6789);
    printf("\n%-10.2f", 12345.6789);
    printf("\n%10.2f\n", -12345.6789);

    char str[] = "C Language";
    printf("\n%-10s", str);
    printf("\n%11s", str);
    printf("\n%12s", str);
    printf("\n%13s\n\n", str);

    char strf[] = "%10s\n";
    printf(strf, "1");
    printf(strf, "1 2");
    printf(strf, "1 2 3");
    printf(strf, "1 2 3 4");
    printf(strf, "1 2 3 4 5");
}
```

555
555
-555
12345.68
12345.68
-12345.68
C Language
C Language
C Language
C Language
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

**ตัวอย่าง 4-9 แนวทางการกำหนดความกว้างของสตริง เพื่อแสดงผลคล้ายตาราง**

```
#include <stdio.h>
```

Products List:		
Item Name	Price (Bath)	Storage (GB)
iPhone	35,000	256
iPad	20,000	128
MacBook	80,000	512

```
void main() {
    //รูปแบบ (ความกว้าง) โดยแต่ละคอลัมน์เท่ากับ 15
    //คอลัมน์แรกเป็นข้อความให้ชิดซ้าย (ให้ความกว้างติดลบ)
    //ส่วนคอลัมน์ถัดไปให้ชิดขวา เพราะเป็นตัวเลข
    char fm[200] = "\n%-15s%15s%15s";

    //หัวตาราง
    puts("\nProducts List:");
    printf(fm, "Item Name", "Price (Bath)", "Storage (GB)");
    printf("\n-----");
```

```

// macro ที่ 1
printf(fm, "iPhone", "35,000", "256");

// macro ที่ 2
printf(fm, "iPad", "20,000", "128");

// macro ที่ 3
printf(fm, "MacBook", "80,000", "512");

putchar('\n');
}

```

## การอ่านค่าสติงที่จัดรูปแบบด้วย sprintf()

การใช้ฟังก์ชัน printf() ดังที่กล่าวมานั้น เป็นการแสดงสติงผลลัพธ์ออกไปโดยตรง ซึ่งเราไม่สามารถนำสติงดังกล่าวไปใช้งานอื่นๆ ต่อไปได้ แต่นอกจากนี้ ยังมีฟังก์ชันที่มีหลักการคล้ายคลึงกัน นั่นคือ sprintf() โดยมีรูปแบบดังนี้

```
sprintf(ตัวแปรเก็บข้อมูล, "สติงที่จัดรูปแบบ", ค่าที่_1, ค่าที่_2, ค่าที่_3, ...)
```

- ชื่อฟังก์ชันนี้ จะขึ้นต้นด้วยตัว "s" ดังนั้น อย่าสับสนกับอีกฟังก์ชันที่ชื่อ printf()
- สติงที่จัดรูปแบบ ก็เหมือนกับฟังก์ชัน printf() ซึ่งเราสามารถนำสัญลักษณ์การจัดรูปแบบมาใช้ได้ทั้งหมด
- ค่าที่จะนำไปแทรก ก็ใช้หลักการเดียวกับฟังก์ชัน printf() เช่นเดย
- ตัวแปรเก็บข้อมูล ฟังก์ชัน sprintf() จะไม่แสดงผลลัพธ์ไป แต่จะเก็บสติงผลลัพธ์ที่ได้ (หลังการแทรกข้อมูล) ไว้ในตัวแปรนี้แทน จากนั้น หากเราจะนำสติงไปใช้งาน ก็ทำผ่านตัวแปรนี้ นี่คือสิ่งที่แตกต่างจาก printf()

จากที่กล่าวมา อาจสรุปความแตกต่างระหว่าง 2 ฟังก์ชันนี้ได้ว่า printf() จะแสดงผลลัพธ์ออกไปทันที ส่วน sprintf() จะเก็บผลลัพธ์ไว้ในตัวแปร ซึ่งในหัวข้อต่อๆ ไป เราจะเห็นแนวทางการนำไป sprintf() ไปประยุกต์ใช้งานที่ชัดเจนขึ้น ส่วนแนวทางการใช้งานแบบพื้นฐาน ให้ดูจากตัวอย่างต่อไปนี้

**ตัวอย่าง 4-10** การจัดรูปแบบสตริงด้วยฟังก์ชัน `sprintf()` เพื่อเก็บผลลัพธ์ไว้ในตัวแปร ก่อนนำไปใช้งานอีกๆ ต่อไป โดยในตัวอย่างนี้ จะแสดงให้เห็นแนวทางการเชื่อมต่อสตริงใหม่กับค่าเดิม ในตัวแปรคล้ายกับ `strcat()` แต่กรณีนี้ เราสามารถแทรกค่าตัวแปรได้

```
#include <stdio.h>

void main() {
    char today[10];
    sprintf(today, "\nToday is: %d/%d/%d", 31, 10, 2023);
    puts(today);

    char str[200];
    sprintf(str, "\nI was born in year %d", 2000);
    sprintf(str, "%s\nand current year is %d", str, 2023);
    sprintf(str, "%s\nso I'm %d years old\n", str,
            (2023 - 2000));

    puts(str);
}
```

Today is: 31/10/2023  
 I was born in year 2000  
 and current year is 2023  
 so I'm 23 years old

## การจัดรูปแบบตัวเลขโดยมี , คั่นหลักพัน

ตัวเลขที่มีค่ามากๆ หากเราเขียนต่อเนื่องกันโดยไม่มีสัญลักษณ์ใดๆ มาคั่นระหว่างหลัก อาจเกิดความยุ่งยากในการอ่านค่าตัวเลข เช่น 387504244 หรือ 256520238014257 เป็นต้น ดังนั้น เพื่อแก้ปัญหาตามที่กล่าวมา เราควรใช้เครื่องหมาย , เพื่อคั่นหลักพัน เช่น 387,504,244 หรือ 256,520,238,014,257 อย่างไรก็ตาม ในภาษา C ไม่มีสัญลักษณ์ที่ใช้สำหรับกรณีนี้โดยตรง แต่ยังมีทางเลือกที่ใช้ทดแทนได้ ซึ่งวิธีการที่ผู้เขียนจะแนะนำต่อไปนี้ สามารถใช้งานได้บนตัวแปลงภาษา GCC/MinGW แต่ไม่แน่ใจว่าจะใช้ได้ผลกับตัวแปลงภาษา C ของผู้สร้างรายอื่นๆ หรือไม่ โดยมีแนวทางดังนี้

- เพิ่มค่าคงที่ และอ้างอิงไฟล์ヘดเดอร์ `locale.h` โดยกำหนดโค้ดตามนี้ (กำหนดไว้ก่อนเขตเดอร์อีกๆ)

```
#define _POSIX_C_SOURCE 200809L
#include <locale.h>
```

- ก่อนการแสดงผลตัวเลขที่จะคั่นหลักพัน ให้กำหนดคำสั่ง `setlocale()` ดังนี้

```
setlocale(LC_ALL, "");
```

- การแสดงผลตัวเลข ก็ใช้ฟังก์ชัน printf() ตามเดิม โดยกำหนดลักษณะดังนี้ (ต้องเลือกให้สอดคล้องกับชนิดข้อมูล)

%'d	// สำหรับเลขจำนวนเต็ม
%'f หรือ %'0.?f	// สำหรับเลขทศนิยม โดย ? คือจำนวนตำแหน่งทศนิยม
%'g	// สำหรับเลขทศนิยม

- ต้องใช้เครื่องหมาย ' ซึ่งจะถูกเปลี่ยนเป็น , เมื่อแสดงผล
- หลักการอื่นๆ ก็เหมือนกับที่เรากำหนดลักษณะสำหรับจัดรูปแบบให้กับฟังก์ชัน printf() ตามปกตินั้นเอง แค่เพิ่ม ' เข้ามา

สำหรับแนวทางการใช้งาน ให้ดูจากตัวอย่างต่อไปนี้ได้เลย เพราะใช้หลักการเดิมเป็นส่วนใหญ่

#### ตัวอย่าง 4-11 การจัดรูปแบบตัวเลขโดยมี คันหลักพ้น

```
#define _POSIX_C_SOURCE 200809L
#include <locale.h>
#include <stdio.h>

void main() {
    setlocale(LC_ALL, "");

    int a = 12345678;
    double b = 987654321.357;

    putchar('\n');
    printf("a = '%d", a);

    putchar('\n');

    printf("b = '%0.3f", b);
    putchar('\n');
}
```

a = 12,345,678  
b = 987,654,321.357

## การรับข้อมูลทางคีย์บอร์ด

ที่ผ่านมา/main เรากำหนดข้อมูล หรือค่าของตัวแปรลงโค้ดแบบตายตัว หรือเรียกว่า hard code ซึ่งเป็นวิธีการที่ขาดความยืดหยุ่น เพราะผู้ใช้งานไม่สามารถปรับเปลี่ยนค่าเองได้ ดังนั้น ในหัวข้อนี้ เราจะมาเรียนรู้วิธีการรับข้อมูลทางคีย์บอร์ด โดยผู้ใช้งานสามารถกำหนดข้อมูลได้เองตาม

ที่ต้องการ ซึ่งจะช่วยให้โปรแกรมที่เราระบบมีผลตอบสนองที่หลากหลายมากขึ้น ตามข้อมูลที่เรากำหนดให้กับมัน แต่อย่างไรก็ตาม การรับข้อมูลทางคีย์บอร์ดมีหลายลักษณะ ซึ่งในที่นี้จะขอแยกอธิบายตามฟังก์ชันที่ใช้งาน ดังรายละเอียดต่อไปนี้

### การรับข้อมูลชนิดอักขระ getchar()

ข้อมูลชนิดอักขระ (char) จะมีได้เพียงอักขระตัวเดียว ซึ่งหากเราจะรับข้อมูลชนิดนี้ทางคีย์บอร์ด ก็ต้องใช้ฟังก์ชัน getchar() ที่มีรูปแบบดังนี้

```
ตัวแปร = getchar()
```

- ตัวแปรที่จะใช้รับค่าจากฟังก์ชัน ต้องเป็นชนิด char เท่านั้น ซึ่งอาจประกาศไว้ล่วงหน้าหรือจะประกาศในขณะนั้นเลยก็ได้
- ก่อนรับข้อมูลด้วย getchar() เราควรแสดงข้อความเพื่อแจ้งให้ผู้ใช้ทราบพอลังเขบว่า ต้องใส่ข้อมูลเกี่ยวกับอะไร ซึ่งโดยส่วนใหญ่เรานิยมใช้ฟังก์ชัน printf() เพื่อให้จุดอรับข้อมูลอยู่ที่ห้ายข้อความ (ฟังก์ชัน printf() จะไม่ขึ้นบรรทัดใหม่)
- เมื่อการประมวลผลดำเนินมาถึงขั้นตอนที่ฟังก์ชัน getchar() ถูกเรียกขึ้นมาทำงาน ส่วนแสดงผล เช่น Terminal จะหยุดรอรับข้อมูลจนกว่าเราจะใส่ค่าลงไป และกด <Enter> จึงจะทำคำสั่งถัดไป
- ตามปกติ เราต้องใส่อักขระเพียงตัวเดียวแล้วกด <Enter> แต่ถ้าเราใส่ข้อมูลในแบบ สริง (อักขระมากกว่า 1 ตัว) มันจะอ่านเฉพาะอักขระตัวแรกที่เราใส่ลงไปเท่านั้น
- ค่าที่ได้ซึ่งเก็บในตัวแปร ก็สามารถนำไปใช้งานอื่นๆ ตามต้องการ

เนื่องจากฟังก์ชัน getchar() ไม่มีอะไรที่ยุ่งยาก จึงขอให้ใช้แนวทางการใช้งานจากตัวอย่าง ต่อไปนี้ได้เลย

**ตัวอย่าง 4-12** การรับข้อมูลชนิดอักขระทางคีย์บอร์ดด้วย getchar() และนำไปแสดงผล (หากเรากำหนดลัญลักษณ์เป็น %c จะแสดงอักขระตัวนั้น แต่หากกำหนดเป็น %d จะแสดงรหัส ASCII ของอักขระตัวนั้นแทน)

```
#include <stdio.h>
void main() {
    printf("\nEnter character >>");
```

```
Enter character >>C
ASCII code of C is 67
```

```

char ch = getchar();

printf("ASCII code of %c is %d\n", ch, ch);
}

```

## การรับข้อมูลแบบสตริงด้วย gets()

ฟังก์ชัน gets() มาจากคำว่า get string ใช้ในการรับข้อมูลแบบสตริงทางคีย์บอร์ด ซึ่งลักษณะของฟังก์ชันนี้คือ

gets(ตัวแปรที่ใช้รับข้อมูล)

- ตัวแปรที่จะใช้รับข้อมูลต้องประกาศเอาไว้ล่วงหน้า และเป็นชนิดชุดอักขระ (สตริง) เช่น char str[]
- ก่อนรับข้อมูลด้วย gets() เราควรแสดงข้อความเพื่อแจ้งให้ผู้ใช้ทราบพอลังเข็ปว่า ต้องใส่ข้อมูลเกี่ยวกับอะไร ซึ่งโดยส่วนใหญ่เรานิยมใช้ฟังก์ชัน printf() เพื่อให้จุดรอรับข้อมูลอยู่ที่ท้ายข้อความ (ฟังก์ชัน printf() จะไม่ขึ้นบรรทัดใหม่)
- ข้อมูลที่รับผ่านฟังก์ชัน จะถูกจัดเป็นชนิดสตริง แม้จะใส่เป็นอักขระตัวเดียว หรือตัวเลข ทั้งหมดก็ตาม และไม่สามารถนำไปคำนวณได้ (ยกเว้นจะผ่านการแปลงชนิดข้อมูลก่อน เช่นกรณีจะกล่าวถึงในหัวข้อต่อ ๆ ไป)

แนวทางการรับข้อมูลชนิดสตริงทางคีย์บอร์ดด้วยฟังก์ชัน gets() มีดังนี้

**ตัวอย่าง 4-13** การรับข้อมูลชนิดสตริงทางคีย์บอร์ดด้วย gets() และนำมายังผล

```

#include <stdio.h>

void main() {
    char name[30], age[3], from[200];

    printf("\nWhat's your name >> ");
    gets(name);

    printf("Where are you come from >> ");
    gets(from);

    printf("How old are you >> ");
    gets(age);
}

```

```

What's your name >>Tom Jerry
Where are you come from >>USA
How old are you >>30
-----
Your Information:
Name: Tom Jerry
Age: 30
From: USA

```

```

    puts("-----");
    puts("Your Information:");

    printf("\nName: %s \nAge: %s \nFrom: %s\n",
           name, age, from);
}

```

## การรับข้อมูลหลายชนิดด้วย scanf()

การรับข้อมูลด้วย getchar() หรือ gets() ดังที่กล่าวมานั้น เป็นการรับแบบเจาะจงชนิดข้อมูล ซึ่งอาจใช้งานได้เพียงบางกรณีเท่านั้น ถ้าเราต้องการรับข้อมูลได้หลายชนิด ก็สามารถเปลี่ยนไปใช้ฟังก์ชัน scanf() ซึ่งรองรับการรับข้อมูลทางคีย์บอร์ดได้หลายชนิด และมีความยืดหยุ่นมากกว่า โดยตัว f ที่ต่อท้ายชื่อฟังก์ชันมาจากคำว่า format หมายถึง เราสามารถระบุรูปแบบ (ชนิด) ข้อมูลที่จะรับผ่านฟังก์ชันนี้ได้ ด้วยรูปแบบดังนี้

```

scanf("สตริงควบคุมรูปแบบ", ตัวแปร)
หรือ scanf("สตริงควบคุมรูปแบบ", &ตัวแปร)

```

- สตริงควบคุมรูปแบบ (control-format-string) เป็นตัวบ่งชี้ หรือข้อกำหนดว่า เราจะใช้รับข้อมูลชนิดใด โดยใช้สัญลักษณ์เป็นอักขระในแบบเดียวกับที่เรากำหนดให้แก่ ฟังก์ชัน printf() แต่อย่างไรก็ตาม ในกรณีของเลขตัวเลข เราต้องระบุสัญลักษณ์ให้สอดคล้องกับชนิดข้อมูล เพราะบางอย่างใช้แทนกันไม่ได้ (แม้จะไม่เกิดข้อผิดพลาดแต่อาจทำให้ได้ผลลัพธ์ที่ไม่ถูกต้อง ซึ่งต่างจาก printf() ที่อาจใช้แทนกันได้ในบางกรณี) และเพื่อความชัดเจนยิ่งขึ้นผู้เขียนขอนำสัญลักษณ์ที่เรารู้จักใช้งานบ่อยๆ มาแสดงเปรียบเทียบให้ดูระหว่างการใช้ร่วมกับ printf() และ scanf() อีกครั้งหนึ่ง

data type	printf()	scanf()
int	%d	%d
unsigned int	%u หรือ %d	%u
short	%hd หรือ %d	%hd
unsigned short	%hd หรือ %d	%hu
long	%ld หรือ %d	%ld
unsigned long	%lu หรือ %d	%u

data type	printf()	scanf()
long long	%lld	%lld
float	%f	%f
double	%f	%lf
char	%c	%c
string	%s	%s

- ตัวแปร ที่จะใช้เก็บข้อมูล ต้องประกาศไว้ล่วงหน้า และให้ชนิดข้อมูลสอดคล้องกับสตริง ควบคุมรูปแบบ ทั้งนี้ การวางแผนเครื่องหมาย & ไวยากรณ์ตัวแปรหรือไม่ ให้พิจารณาดังนี้
  - กำหนดแค่ชื่อตัวแปร (ไม่มี & นำหน้า) จะได้ข้อมูลแบบสตริง แม้จะใส่ค่าเป็นตัวเลขทั้งหมดก็ตาม ซึ่งไม่สามารถนำไปคำนวณโดยตรงได้
  - กำหนดชื่อตัวแปรโดยมี & นำหน้า ถ้าใส่ข้อมูลเป็นตัวเลข ก็จะได้ข้อมูลชนิดตัวเลข ตามที่ระบุในสตริงสัญลักษณ์ และสามารถนำไปใช้ในการคำนวณได้ทันที
  - หรือกล่าวได้ว่า กรณีที่รับข้อมูลตัวเลข ให้เขียน & นำหน้าชื่อตัวแปรในแบบ &ชื่อตัวแปร
- ก่อนรับข้อมูลด้วย `scanf()` เราชาระแสดงข้อความเพื่อแจ้งให้ผู้ใช้ทราบพอลังเขปว่า ต้องใส่ข้อมูลเท่าใด ซึ่งโดยส่วนใหญ่เรานิยมใช้ฟังก์ชัน `printf()` เพื่อให้จุดรอรับข้อมูลอยู่ที่ท้ายข้อความ (ฟังก์ชัน `printf()` จะไม่ขึ้นบรรทัดใหม่) เช่น

```
char name[50];
int age;

printf("what's your name >> ");
scanf("%s", name); //รับข้อมูลชนิดสตริง

printf("how old are you >> ");
scanf("%d", &age); //รับข้อมูลชนิดเลขจำนวนเต็ม
```

- ฟังก์ชัน `scanf()` สามารถรับข้อมูลพร้อมกันได้มากกว่า 1 ค่า โดยใช้ ช่องว่าง เป็นตัวแบ่งข้อมูล ซึ่งเราต้องกำหนดสัญลักษณ์ควบคุมรูปแบบและตัวแปรให้ครบตามจำนวนข้อมูล และต้องเรียงลำดับให้สอดคล้องกัน เช่น

```
char firstname[30], lastname[50];
int age;
printf("enter your firstname lastname and age >>");
scanf("%s%s%d", firstname, lastname, &age);

/*แนวทางการใส่ข้อมูล เช่น
What's your...>>John Smith 40

ค่าที่เก็บในตัวแปรจะเป็น
firstname = "John"
lastname = "Smith"
age = 40
*/
```

- จากการนี้ที่ฟังก์ชัน `scanf()` ใช้ซ่องว่างเป็นตัวแบ่งค่าของข้อมูลที่รับจากคีย์บอร์ด จึงเกิดปัญหาเมื่อเราจะรับข้อมูลที่เป็นสตริงยาวๆ และมีหลายคำ ซึ่งโดยทั่วไปคำในภาษาอังกฤษจะใช้ซ่องว่างเป็นตัวแบ่งคำ ดังนั้น ถ้าเรารับค่าแบบสตริงโดยมีตัวแบ่งเดียว แล้วใส่ข้อมูลทางคีย์บอร์ดที่มีหลายๆ คำ เช่น `Hello how are you` คำที่เรารอได้จะเป็นคำว่า `Hello` เท่านั้น เช่น

```
char greeting[200];
printf("Enter string >>");
scanf("%s", greeting);
/*
Enter string >>Hello how are you

คำที่เก็บในตัวแบ่งจะเป็น
greeting = "Hello"
%/

```

- ในการรับข้อมูลสตริง ถ้าเรากำหนดลักษณ์เป็น `%s` อาจได้เฉพาะคำแรกที่ใส่เข้ามา คำที่อยู่ต่อจากนั้นอาจสูญหายไป ทั้งนี้ เราอาจเลี่ยงไปใช้ `gets()` แทนก็ได้ หรือถ้าเราอยากรู้ว่า `scanf()` เพียงแบบเดียว กับข้อมูลทุกชนิด ก็มีแนวทางแก้ไขได้โดยกำหนดลักษณ์ควบคุมรูปแบบเป็น `%[^\\n]` ซึ่งหมายความว่า ให้รับอักขระทั้งหมดยกเว้นอักขระการขึ้นบรรทัดใหม่ (การกด `<Enter>` นั่นเอง) ซึ่งกรณีนี้จะไม่ใช้ซ่องว่างในการแบ่งคำ ดังนั้น เราจะได้ค่าเป็นสตริงตามที่เรากำหนด เช่น

```
char str[100];
printf("Enter string >>");
scanf("%[^\\n]", str);
/*
Enter string >>I can C!
str ="I can C!"
*/
```

- `scanf()` จะไม่มีปัญหาในการรับข้อมูลชนิดตัวเลข เพราะเลขจำนวนเติมกันจะเขียนติดกันโดยไม่มีซ่องว่างคั่น
- ปัญหาที่สำคัญอีกอย่างหนึ่งคือ ถ้าเราใช้ `scanf()` เพื่อรับข้อมูลต่อเนื่องกันหลายครั้ง อาจมีบางกรณีที่มันไม่หยุดรอรับข้อมูล แต่จะข้ามไปทำคำสั่งถัดไปทันที เนื่องจากมันตีความหมายว่าปุ่ม `<Enter>` ที่เรากดหลังใส่ข้อมูลครบเป็น อักขระ 1 ตัว (`\n`) จากนั้นมันก็จะเก็บอักขระนี้ไว้ในบัฟเฟอร์ (Buffer) เมื่อเรารับข้อมูลครั้งถัดไปด้วย `scanf()`

มันจะนำอักขระจากบัฟเฟอร์ไปกำหนดค่าให้เองโดยอัตโนมัติ ด้วยเหตุนี้มันจึงไม่หยุดรอรับข้อมูล ซึ่งความจริงความสามารถแก้ไขปัญหานี้ได้หลายวิธี แต่ในเบื้องต้นนี้ผู้เขียนจะแนะนำวิธีง่ายๆ เพียงแบบเดียวไปก่อน โดยการเพิ่มคำสั่ง `getchar()` หลังจากเรียก `scanf()` เพื่อให้มันอ่านหรือล้าง (Clear) อักขระ `\n` บัฟเฟอร์ก่อนรับข้อมูลค่าต่อไป ซึ่งขั้นตอนนี้จะไม่ทำให้เกิดการหยุดรอแต่อย่างใด เช่น

```
char a[100], b[100];
...
scanf("%[^\\n]", a);
getchar();           //clear \n ในบัฟเฟอร์ (ไม่หยุดรอรับข้อมูล)
...
scanf("%[^\\n]", b);
getchar();           //clear \n ในบัฟเฟอร์ (ไม่หยุดรอรับข้อมูล)
...
```

ปัญหาที่กล่าวถึงในข้อสุดท้ายนี้ มักเกิดขึ้นในกรณีที่เรารับข้อมูลทางคีย์บอร์ดต่อเนื่องกันมากกว่า 1 ครั้ง โดยทั่วไปมักจะเกิดกับการรับข้อมูลแบบสตริงและอักขระ ส่วนข้อมูลชนิดตัวเลข มักไม่เจอปัญหาดังกล่าว แต่ถ้ามีกรณีเจอปัญหาแบบเดียวกัน ก็อาจนำวิธีการดังกล่าวนี้ไปใช้กับการรับข้อมูลชนิดตัวเลขก็ได้เช่นกัน สำหรับแนวทางการใช้งาน ให้ดูเพิ่มเติมจากตัวอย่างต่อไปดังต่อไปนี้

**ตัวอย่าง 4-14** เป็นการนำตัวอย่างที่ 4-13 มาปรับเปลี่ยนจากการรับข้อมูลด้วยฟังก์ชัน `get()` ก็เปลี่ยนมาใช้ `scanf()`

```
#include <stdio.h>

void main() {
    char name[30], from[100];
    int age;

    printf("\nWhat's your name >>");
    scanf("%[^\\n]", name);
    getchar();           //clear \n

    printf("Where are you come from >>");
    scanf("%[^\\n]", from);
    getchar();           //clear \n
```

```
What's your name >>Robert William Johnson
Where are you come from >>United Kingdom
How old are you >>40
-----
Your Information:
Name: Robert William Johnson
From: United Kingdom
Age: 40
```

```

printf("How old are you >>");
scanf("%d", &age);

puts("-----");
puts("Your Information:");

printf("\nName: %s \nFrom: %s \nAge: %d \n",
      name, from, age);
}

```

**ตัวอย่าง 4-15** การรับข้อมูลตัวเลขทางคีย์บอร์ดด้วย `scanf()` และเทียบว่าระหว่างการใช้และไม่ใช้เครื่องหมาย & นำหน้าชื่อตัวแปรที่เป็นชนิดตัวเลข จะให้ผลต่างกันอย่างไร

```

#include <stdio.h>

void main() {
    int num1, num2;

    puts("Using &");
    printf("First Number >>");
    scanf("%d", &num1);
    //getchar(); //ไม่มีก็ได้ เพราะรับข้อมูลตัวเลข

    printf("Second Number >>");
    scanf("%d", &num2);
    //getchar();

    printf("%d + %d = %d", num1, num2, (num1 + num2));

    puts("\n-----");

    puts("\nWithout &");
    printf("First Number >>");
    scanf("%d", num1); //ไม่ระบุ & นำหน้าตัวแปร กnum1
    //getchar();

    printf("Second Number >>");
    scanf("%d", num2); //ไม่ระบุ & นำหน้าตัวแปร กnum2
    //getchar();

    printf("%d + %d = %d", num1, num2, (num1 + num2));
}

```

```

Using &
First Number >>40
Second Number >>60
40 + 60 = 100
-----
Without &
First Number >>50
c:\c-lang\chapter4>-----  

← เกิดข้อผิดพลาด  

ไม่รับข้อมูลต่อ

```

**ตัวอย่าง 4-16** การรับข้อมูลทางคีย์บอร์ดครั้งละ 2 ค่าโดยเป็นผลการแข่งขันฟุตบอลในบางคู่ จากนั้นนำไปแสดงผล

```
#include <stdio.h>

void main() {
    int mu, lp, cs, as, mc, sp;

    puts("\nExample: \nTeam A - Team B >>3 2");

    printf("\nManU - Liverpool >>");
    scanf("%d%d", &mu, &lp);

    printf("Chelsea - Arsenal >>");
    scanf("%d%d", &cs, &as);

    printf("ManCity - Spur >>");
    scanf("%d%d", &mc, &sp);

    puts("\n-----\n");
    puts("Football Result:");
    printf("\nManU %d - %d Liverpool", mu, lp);
    printf("\nChelsea %d - %d Arsenal", cs, as);
    printf("\nManCity %d - %d Spur\n", mc, sp);
}
```

Example:  
Team A - Team B >>3 2

ManU - Liverpool >>2 4  
Chelsea - Arsenal >>3 1  
ManCity - Spur >>2 0

-----  
Football Result:

ManU 2 - 4 Liverpool  
Chelsea 3 - 1 Arsenal  
ManCity 2 - 0 Spur

## การแปลงชนิดข้อมูลระหว่างสตริงและตัวเลข

ภาษา C นั้น จะเข้มงวดเกี่ยวกับชนิดข้อมูลเป็นอย่างมาก ดังนั้น เราต้องเลือกใช้ชนิดข้อมูลให้ตรงตามที่กำหนด มีฉะนั้นอาจเกิดข้อผิดพลาด หรือได้ผลลัพธ์ที่ไม่ถูกต้อง ซึ่งปัญหาพื้นฐานอย่างหนึ่งที่เราต้องพบอยู่บ่อยๆ ก็คือ การใช้งานลับไปมาระหว่างข้อมูลแบบสตริงและตัวเลข เช่น หากนำข้อมูลแบบสตริงไปคำนวณก็จะเกิดข้อผิดพลาด หรือถ้านำข้อมูลชนิดตัวเลข ไปแสดงผลแบบสตริง ก็อาจเกิดข้อผิดพลาดได้ เช่นกัน ด้วยเหตุนี้ เราจึงควรเรียนรู้วิธีแก้ไข เพื่อช่วยลดข้อผิดพลาด และให้มีทางเลือกในการใช้งานที่หลากหลายมากยิ่งขึ้น ดังแนวทางต่อไปนี้

### การแปลงสตริงเป็นตัวเลขด้วยฟังก์ชัน atoi\_()

ตัวเลขที่เขียนหรือกำหนดค่าในแบบสตริง เราเรียกว่า string number เช่น "1234" โดยค่าเช่นนี้ถือเป็นข้อมูลประเภทสตริง ไม่สามารถนำไปคำนวณได้ ซึ่งเราอาจแก้ไขด้วยการเปลี่ยนสตริงให้เป็นข้อมูลชนิดตัวเลขอย่างแท้จริง และสามารถนำไปคำนวณต่างๆ ได้ตามต้องการ โดยใช้ฟังก์ชันอันดีอันหนึ่งดังต่อไปนี้

<code>atoi(string)</code>	แปลงจากสตริงเป็นเลขจำนวนเต็มชนิด int
<code>atof(string)</code>	แปลงจากสตริงเป็นเลขทศนิยมชนิด float
<code>atol(string)</code>	แปลงจากสตริงเป็นเลขจำนวนเต็มชนิด long
<code>atoll(string)</code>	แปลงจากสตริงเป็นเลขจำนวนเต็มชนิด long long

- ฟังก์ชันเหล่านี้ อยู่ในヘดเดอร์ `stdlib.h` ดังนั้น เราต้องอ้างอิงไฟล์ヘดเดอร์นี้ด้วยคำสั่ง `include` จึงจะใช้งานได้
- ค่าสตริงที่กำหนดให้แก่ฟังก์ชันเหล่านี้ ต้องอยู่ในหลักเกณฑ์ที่สามารถแปลงไปเป็นตัวเลขได้ เช่น
  - อาจเป็นเลขจำนวนเต็มหรือมีทศนิยม ที่มีค่าเป็นบวกหรือติดลบก็ได้
  - ถ้าสตริงนั้นมีอักษรที่ไม่ใช่ตัวเลขรวมอยู่ด้วย ต้องขึ้นต้นด้วยตัวเลข เช่นนั้น เช่น "120\$", "256GB", "0.95km", "-10 Celsius" ซึ่งสตริงเหล่านี้สามารถใช้ฟังก์ชันที่สอดคล้องกันเพื่ออ่านตัวเลขที่ขึ้นต้นมาใช้งานได้
  - ไม่ นับรวมกรณีที่ตัวเลขอยู่ระหว่างอักษรระหว่างท้ายตัวเลข เช่น "\$120", "one2thee", "G7" ซึ่งสตริงเหล่านี้ ไม่ สามารถใช้ฟังก์ชันเพื่ออ่านตัวเลขได้
- ฟังก์ชันดังที่แสดงในตาราง จะให้ผลลัพธ์เป็นข้อมูลในชนิดที่สอดคล้องกัน นั่นคือ
  - `atoi()` ให้ผลลัพธ์ข้อมูลชนิด int
  - `atof()` ให้ผลลัพธ์ข้อมูลชนิด float
  - `atol()` ให้ผลลัพธ์ข้อมูลชนิด long
  - `atoll()` ให้ผลลัพธ์ข้อมูลชนิด long long

สำหรับแนวทางการใช้งาน ให้ดูจากตัวอย่างต่อไปนี้

**ตัวอย่าง 4-17** การแปลงข้อมูลจากรูปแบบสตริง ให้เป็นตัวเลขชนิดต่างๆ และทดลองนำไปคำนวณแบบง่ายๆ เพื่อพิสูจน์ว่าค่าที่ถูกแปลงเป็นชนิดตัวเลขจริงหรือไม่

```
#include <stdio.h>
#include <stdlib.h>

void main() {
    char one23[] = "123";
    char ff_point_ss[] = "45.67";
    char storage[] = "256GB";
```

```
"123" + 1 = 124
"45.67" - 1 = 44.67
"256GB" + 64 = 320
"-110.75celsius" - 100 = -210.75

Enter float number >>3.14159
"3.14159" + 0.5 = 3.64
```

```

char tem[] = "-110.75celsius";
char float_num[20];

int a = atoi(one23);
float b = atof(ff_point_ss);
int c = atoi(storage);
float d = atof(tem);

printf("\n\"%s\" + 1 = %d \n", one23, a + 1);
printf("\"%s\" - 1 = %0.2f \n", ff_point_ss, b - 1);
printf("\"%s\" + 64 = %d \n", storage, c + 64);
printf("\"%s\" - 100 = %0.2f \n", tem, d - 100);

printf("\nEnter float number >>");
gets(float_num);
float e = atof(float_num);
printf("\n\"%s\" + 0.5 = %0.2f \n", float_num, e + 0.5);
}

```

## การแปลงตัวเลขเป็นสตริงด้วยฟังก์ชัน sprintf()

นอกจากการแปลงจากสตริงเป็นตัวเลขดังที่กล่าวมาในหัวข้อที่แล้ว ในบางครั้งเราต้องทำในลักษณะที่ตรงกันข้าม นั่นคือ การแปลงจากตัวเลขไปเป็นสตริง ซึ่งแม้จะทำได้หลายวิธี แต่อันที่น่าจะง่ายสุดก็คือ การนำฟังก์ชัน sprintf() มาดัดแปลงในรูปแบบดังนี้

sprintf(ตัวแปรสตริงที่จะเก็บข้อมูล, "ลัญลักษณ์รูปแบบตัวเลข", ตัวเลขที่จะแปลง)

- ฟังก์ชันนี้ก็คืออันเดิมที่เราเคยใช้ในการจัดเก็บสตริงผลลัพธ์หลังการจัดรูปแบบไว้ในตัวแปร เพียงแต่เราสามารถนำดัดแปลงเป็นการปรับเปลี่ยนชนิดข้อมูล
- ลัญลักษณ์รูปแบบตัวเลข ก็กำหนดเช่นเดียวกับฟังก์ชัน scanf() หรือ printf() แต่ในการนี้ เราต้องการสตริงผลลัพธ์ที่มีแค่ตัวเลข ดังนั้น ไม่ควรมีสตริงอื่น ๆ รวมอยู่ในลัญลักษณ์รูปแบบตัวเลข และต้องกำหนดให้สอดคล้องกับชนิดข้อมูลตัวเลขที่จะแปลง เช่น
  - ถ้าเป็นเลขชนิด int ก็กำหนดลัญลักษณ์เป็น "%d"
  - ถ้าเป็นเลขชนิด float ก็กำหนดลัญลักษณ์เป็น "%f"
  - ...

เนื่องจากเป็นฟังก์ชันเดิมที่เราผ่านการใช้งานมาแล้ว ซึ่งรายละเอียดปลีกย่อยต่าง ๆ ก็ยังเหมือนเดิม เพียงแต่มาปรับเปลี่ยนจากเดิมอีกเล็กน้อย สำหรับแนวทางการใช้งาน ให้ดูจากตัวอย่างต่อไปนี้

**ตัวอย่าง 4-18 แนวทางการแปลงจากชนิดข้อมูลตัวเลขให้เป็นแบบสตริง**

```
#include <stdio.h>

void main() {
    int i = 1234;
    float f = 56.789;
    char i_str[10], f_str[10];

    sprintf(i_str, "%d", i);
    sprintf(f_str, "%0.2f", f);

    // เมื่อนำไปแสดงผล เราสามารถใช้รูปแบบสตริงได้เลย
    printf("\ni = %s, f = %s \n", i_str, f_str);

    // ลักษณะผลลัพธ์คือ
    // i = 1234, f = 56.79
}
```

ลิ่งที่เราได้เรียนรู้ในบทนี้ คือข้อมูลชนิดอักขระ สตริง รวมถึงการรับและแสดงผลข้อมูลในลักษณะต่าง ๆ ซึ่งถือเป็นพื้นฐานที่สำคัญยิ่งของภาษา C เพราะสตริงเป็นข้อมูลหลักที่ถูกใช้งานบ่อยที่สุด และการแสดงผลก็ต้องทำความคุ้นเคยกับตัวอักษรที่ต้องการ ดังนั้น เรายังคงศึกษาหัวข้อ เรื่องนี้ให้เกิดความเข้าใจที่ดีพอในระดับหนึ่ง มีจะนั่นอาจเกิดปัญหาในการเรียนรู้เรื่องต่อๆ ไปได้



```
    = modifier_ob.modifiers.new("MIRROR")
    object_to_mirror_ob = mirror_ob
    mod.mirror_object = mirror_ob
    if len(mod.mirror_object.modifiers) > 0:
        for mod in mod.mirror_object.modifiers:
            if mod.type == "MIRROR_X":
                mod.use_x = True
                mod.use_y = False
                mod.use_z = False
            elif mod.type == "MIRROR_Y":
                mod.use_x = False
                mod.use_y = True
                mod.use_z = False
            elif mod.type == "MIRROR_Z":
                mod.use_x = False
                mod.use_y = False
                mod.use_z = True
    else:
        mod.type = "MIRROR_X"
        mod.use_x = True
        mod.use_y = False
        mod.use_z = False
    # add the end -add back the deselected
    # objects
    if len(context.selected_objects) < 2:
        print("please select exactly two objects, one to mirror to the selected object")
        return {'FINISHED'}
```

#### OPERATOR CLASSES

```
class MirrorOperator(bpy.types.Operator):
    """mirror to the selected object"""
    bl_idname = "object.mirror"
    bl_label = "Mirror"
    bl_options = {'REGISTER', 'UNDO'}
```

```
    def execute(self, context):
        if len(context.selected_objects) < 2:
            print("please select exactly two objects, one to mirror to the selected object")
            return {'FINISHED'}
```



# 5

## ตัวเลขและการคำนวณ

**ห 5** นิดข้อมูลที่สำคัญอย่างหนึ่งซึ่งเราต้องใช้งานกันอยู่ตลอดในการเขียนโปรแกรมนั่นก็คือ ข้อมูลชนิดตัวเลข และในบางครั้งก็จำเป็นต้องดำเนินการบางอย่างกับข้อมูลดังกล่าวด้วย เครื่องหมายทางคณิตศาสตร์ ก่อนจะนำไปใช้งานในขั้นตอนต่อไป เช่น การบวก ลบ คูณ หาร เป็นต้น แต่ในบางลักษณะอาจไม่มีเครื่องหมายสำหรับกรณีนั้นโดยตรง เราต้องใช้ฟังก์ชันที่ เกี่ยวข้อง หรือกำหนดวิธีการคำนวนเอาเอง ดังรายละเอียดที่เราจะได้เรียนรู้กันในบทนี้

### เครื่องหมายสำหรับการกำหนดค่า

เครื่องหมายสำหรับการกำหนดค่า (Assignment Operator) ก็คือเครื่องหมายเท่ากับ (=) นั้นเอง โดยการใช้เครื่องหมายนิดนี้ ต้องมีองค์ประกอบ 2 อย่างคือ ตัวแปรที่อยู่ทางฝั่งซ้ายของ เครื่องหมาย และค่าที่อยู่ทางฝั่งขวาของเครื่องหมาย ซึ่งสามารถสรุปได้ดังตาราง

เครื่องหมาย	ชื่อเรียก	ตัวอย่างการใช้งาน
=	เท่ากับ หรือกำหนดค่า (Assignment)	$x = 10$

ลักษณะการใช้เครื่องหมาย = โดยทั่วไปเราจะคุ้นเคยกันดีอยู่แล้ว เช่น

```
int x = 12345;  
float y = 6.789;  
char title[] = "C Programming Language";
```

นอกจากนี้ เราสามารถกำหนดค่าต่อเนื่องกันในแบบ Chaining Assignment โดยค่าจะ ถูกกำหนดให้แก่ตัวแปรจากขวาไปซ้าย เช่น

```
int x, y, z;
x = y = z = 10;
```

จากโค้ดข้างบน การกำหนดค่าจะเรียงตามลำดับคือ  $z = 10$ ,  $y = z$  และ  $x = y$  นั่นแสดงว่าทั้ง  $x$ ,  $y$  และ  $z$  จะมีค่าเป็น 10 เมื่ອอกันหมด

**ตัวอย่าง 5-1** หากเราต้องสลับค่าระหว่างตัวแปร 2 ตัว ในแบบที่ต้องใช้ตัวแปรที่สามเข้ามาช่วยสำหรับการพักข้อมูล มีหลักการดังนี้

$$\begin{array}{l} a = 66 \leftarrow c = a \text{ (66)} \quad a = b \text{ (99)} \\ b = 99 \leftarrow \qquad \qquad \qquad b = c \text{ (66)} \end{array}$$

คำอธิบาย	ได้	ค่าของตัวแปร
สมมติว่าเราจะสลับค่าระหว่าง 2 ตัวแปรคือ $a$ และ $b$	$a = 66$ $b = 99$	$a = 66$ $b = 99$
สร้างตัวแปรที่สาม ( $c$ ) เพื่อใช้พักข้อมูล โดยให้มีค่าเท่ากับตัวแปรที่หนึ่ง	$c = a$	$a = 66$ $b = 99$ $c = 66$
ให้ตัวแปรที่หนึ่ง มีค่าเท่ากับตัวแปรที่สอง	$a = b$	$a = 99$ $b = 99$ $c = 66$
ให้ตัวแปรที่สอง มีค่าเท่ากับตัวแปรที่สาม	$b = c$	$a = 99$ $b = 66$

ตัวแปรที่จะสลับค่าระหว่างกัน ต้องเป็นชนิดเดียวกัน หรือชนิดที่สามารถแปลงข้อมูลระหว่างกันได้โดยอัตโนมัติแบบ Implicit (Automatic) Conversion เช่น short, int, long เป็นต้น

```
#include <stdio.h>

void main() {
    int a = 108;
    int b = -1009;
    int c; // สร้างตัวแปรที่สามเพื่อใช้พักข้อมูล

    printf("\nbefore swapping: a = %d, b = %d", a, b);

    c = a; // ให้ตัวแปรที่สาม มีค่าเท่ากับตัวแปรที่หนึ่ง
```

```
before swapping: a = 108, b = -1009
after swapping: a = -1009, b = 108
```

```

a = b; //นำค่าที่สอง ไปเก็บในตัวแปรแรก
b = c; //นำค่าแรกที่เก็บในตัวแปรที่สาม ไปเก็บในตัวแปรที่สอง
printf("\nafter swapping: a = %d, b = %d \n", a, b);
}

```

## เครื่องหมายสำหรับการคำนวณทางคณิตศาสตร์

เครื่องหมายทางคณิตศาสตร์ (Arithmetic Operator) ก็คือเครื่องหมายพื้นฐานต่าง ๆ ที่ใช้ในการบวก ลบ คูณ และหาร นั่นเอง โดยต้องดำเนินการกับตัวเลข 2 จำนวน ซึ่งเครื่องหมายในกลุ่มนี้คือ

เครื่องหมาย	ชื่อเรียก	ตัวอย่างการใช้งาน
+	บวก (Addition)	10 + 20
-	ลบ (Subtraction)	100 - 50
*	คูณ (Multiplication)	5 * 10
/	หาร (Division)	50 / 10
%	ค่าที่เหลือของการหาร (Remainder)	13 % 5 = 3

- ถ้าเรากำหนดชนิดข้อมูลของตัวแปรที่ใช้เก็บผลลัพธ์เป็นชนิดจำนวนเต็ม เช่น short, int, long ผลลัพธ์จะถูกแปลงเป็นจำนวนเต็มเสมอ แม้ผลลัพธ์ที่ได้จะเป็นเลขทศนิยม ก็ตาม
- ถ้าเรากำหนดชนิดข้อมูลของตัวแปรที่ใช้เก็บผลลัพธ์เป็นชนิดทศนิยม เช่น float, double ผลลัพธ์จะถูกแปลงเป็นจำนวนทศนิยม แม้ผลลัพธ์ที่ได้จะเป็นเลขจำนวนเต็ม ก็จะมี .0 ต่อท้าย เช่น จาก  $5 = > 5.0$
- การคำนวณระหว่าง จำนวนเต็ม กับ จำนวนเต็ม ผลลัพธ์ที่ได้จะเป็นจำนวนเต็ม เสมอ ดังนั้น ในการนี้ เราควรกำหนดชนิดข้อมูลของตัวแปรที่ใช้เก็บผลลัพธ์เป็นชนิด จำนวนเต็ม เช่นกัน นี่คือประเด็นสำคัญที่เราต้องพึงระวังเป็นพิเศษ โดยเฉพาะอย่างยิ่ง กับการหาร เช่น  $15 / 10 = 1$
- การคำนวณระหว่าง จำนวนที่มีทศนิยม กับ จำนวนใดๆ ผลลัพธ์ที่ได้จะเป็นจำนวน ที่มีทศนิยมเสมอ ถึงจะได้ค่าเป็นจำนวนเต็มก็จะมี .0 ต่อท้าย ดังนั้น เราควรกำหนด ชนิดข้อมูลของตัวแปรผลลัพธ์ให้เป็นชนิดเลขทศนิยม เช่น float หรือ double มิฉะนั้น อาจได้ผลลัพธ์ที่ไม่ถูกต้อง เช่น  $10.5 + 0.5 = 11.0$  หรือ  $1 + 0.0 = 1.0$

- เครื่องหมาย + และ - ก็เป็นการบวก ลบ ตัวเลขธรรมด้า และลิ่งที่เรารู้จารณาอีกอย่างคือ ชนิดข้อมูลของผลลัพธ์อาจถูกแปลงให้สอดคล้องกับชนิดข้อมูลของตัวแปรที่ใช้เก็บผลลัพธ์ เช่น

```
int a = 15 + 5;           //20
float b = 15 + 5;         //20.0 (ชนิด float)
int c = 15 - 5;           //10
double d = 15 - 5;        //10.0
```

- เครื่องหมาย / จะเป็นการหารแบบปกติ ที่อาจได้ผลลัพธ์เป็นจำนวนเต็มหรือศูนย์ (ชนิดข้อมูลของผลลัพธ์จะขึ้นกับชนิดข้อมูลของตัวแปรที่ใช้เก็บผลลัพธ์) เช่น

```
int a = 10 / 2;           //5
int b = 100 / 5;          //20
double c = 10 / 2.5;       //4.0
float d = 10 / 4;          //2.5
int e = 10 / 4;           //ปกติเท่ากับ 2.5 (float)
                           //แต่ชนิดตัวแปรที่เก็บผลลัพธ์เป็น int
                           //จึงถูกแปลงจาก float => int
                           //นั่นคือ (int)2.5 => 2
```

- เครื่องหมาย % (หรือเรียกว่า Modulus) เป็นการหารแบบเอาเฉพาะค่าที่เหลือ (Remaining) โดย ทั้งตัวตั้งและตัวหารต้องเป็นจำนวนเต็ม มิฉะนั้นจะเกิดข้อผิดพลาด เช่น

```
int a = 8 % 3;
//a = 2 ค่าสูงสุดที่ 3 หารลงตัวคือ 6
//ดังนั้น ค่าที่เหลือคือ 2 (8 - 6)

int b = 10 % 5;
//b = 0 กรณีนี้เป็นการหารแบบลงตัว ดังนั้น ค่าที่เหลือคือ 0

int c = 13 % 4;
//c = 1 ค่าสูงสุดที่ 4 หารลงตัวคือ 12
//ดังนั้น ค่าที่เหลือคือ 1 (13 - 12)

float d = 50 % 2.5;
//Error ทั้งตัวตั้งและตัวหารต้องเป็นเลขชนิดจำนวนเต็ม
```

```

int e = 12.0 % 5;
//Error ทั้งตัวตั้งและตัวหารต้องเป็นเลขชนิดจำนวนเต็ม
//เนื่องจาก 12.0 มีทศนิยม จึงถือเป็นชนิด float

```

- การหาระหว่าง **จำนวนเต็ม กับ จำนวนเต็ม** ผลลัพธ์ที่ได้จะถูกแปลงเป็น จำนวนเต็ม โดยอัตโนมัติ แม้ว่าผลลัพธ์ที่เกิดขึ้นจริงจะมีทศนิยมด้วย (หารไม่ลงตัว) หรือกำหนดตัวแปรที่เก็บผลลัพธ์เป็นประเภท float หรือ double ก็ตาม จึงอาจได้ผลลัพธ์ที่ไม่ถูกต้อง ทั้งนี้ หากเราต้องการผลลัพธ์เป็นเลขทศนิยมตามผลหารจริงที่ได้ อาจแปลงชนิดข้อมูล (คลาสต์) ตัวตั้ง และ/หรือ ตัวหาร ให้เป็น float หรือ double พร้อมกำหนดตัวแปรที่เก็บเป็นข้อมูลประเภทนี้ด้วย เช่นกัน ดังแนวทางต่อไปนี้

```

//ปกตินั้น 10 / 4 = 2.5

int a = 10 / 4;
//a = 2 เพราะตัวแปรเก็บผลลัพธ์เป็นชนิด int
//ผลลัพธ์จึงถูกแปลงเป็น int โดยอัตโนมัติ
//ดังนั้น ผลลัพธ์ที่ได้จึงไม่ถูกต้อง

float b = 10 / 4;
//b = 2.0 เพราะทั้งตัวตั้งและตัวหารเป็นจำนวนเต็ม
//ผลหารก็จะเป็นจำนวนเต็มด้วย
//แม้ b จะเป็นชนิด float ก็ตาม
//ดังนั้น ผลลัพธ์ที่ได้จึงไม่ถูกต้อง

float c = (float)10 / 4;
//c = 2.5 เพราะแปลงตัวตั้งเป็นชนิด float
//ผลลัพธ์จึงเป็นชนิด float ด้วย

float d = 10 / (float)4;
//d = 2.5 เพราะแปลงตัวหารเป็นชนิด float
//ผลลัพธ์จึงเป็นชนิด float ด้วย

float e = (float)10 / (float)4;
//d = 2.5 เพราะแปลงทั้งตัวตั้งและตัวหารเป็นชนิด float
//ผลลัพธ์จึงเป็นชนิด float ด้วย

```

**ตัวอย่าง 5-2** รับค่าความกว้างและความสูงของรูปสี่เหลี่ยมมุมฉาก (Rectangle) ผ่านทางคีย์บอร์ดด้วยฟังก์ชัน `scanf()` และคำนวนหาพื้นที่และความยาวโดยรอบ ดังนี้

```
#include <stdio.h>

void main() {
    float width, height, area, perimeter;

    printf("\nrectangle width >>");
    scanf("%f", &width);

    printf("rectangle height >>");
    scanf("%f", &height);

    area = width * height;
    perimeter = 2 * (width + height);

    printf(
        "area = %g \nperimeter = %g \n",
        area, perimeter
    );
}
```

```
rectangle width >>20
rectangle height >>30
area = 600
perimeter = 100
```

**ตัวอย่าง 5-3** รับข้อมูลทางคีย์บอร์ดและดำเนินการดังนี้

1. รับข้อมูลจำนวนลินค้าที่จะซื้อ
2. รับข้อมูลราคาลินค้าต่อหน่วย
3. ให้ลินค้ามีส่วนลด 10% (หักจากราคา)
4. คำนวนหาผลรวมที่ต้องจ่ายจริง (จำนวน x ราคา)
5. รับข้อมูลจำนวนเงินสดที่ผู้ซื้อจ่ายมา
6. คำนวนหาจำนวนเงินที่ต้องทอน

```
#include <stdio.h>

void main() {
    int quantity;
    float price, total1, total2, pay, change;

    //จำนวนลินค้าต้องเป็นจำนวนเต็ม
    printf("\nquantity to buy >>");
```

```
quantity to buy >>5
price per unit >>150.50
total (before discount): 752.50
total (after discount 10%): 677.25

pay >>1000
change: 322.75
```

```

scanf("%d", &quantity);

//ราคาสินค้าอาจมีทศนิยมได้
printf("price per unit >>");
scanf("%f", &price);

//รวม (ก่อนหักส่วนลด)
total1 = quantity * price;
printf("total (before discount): %0.2f", total1);

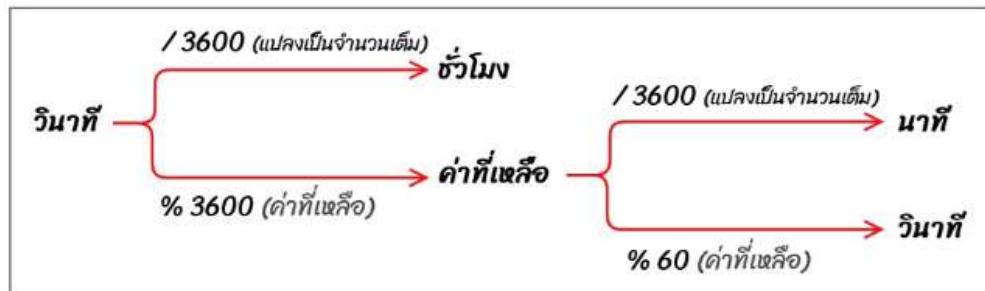
//รวม หลังหักส่วนลด 10%
total2 = (total1 * (100 - 10)) / 100;
printf(
    "\ntotal (after discount 10%): %0.2f", total2
);

//จำนวนเงินสดที่จ่าย
printf("\n\npay >>");
scanf("%f", &pay);

//เงินคงเหลือ
change = pay - total2;
printf("change: %0.2f \n", change);
}

```

**ตัวอย่าง 5-4** รับค่าทางคีย์บอร์ดเป็นจำนวนวินาที แล้วนำมาแปลงเป็นหน่วย ชั่วโมง นาที วินาที เช่น 3665 วินาที เท่ากับ 1 ชั่วโมง 1 นาที 5 วินาที เป็นต้น โดยให้เริ่มจากหน่วยที่มีค่ามากสุดที่เป็นไปได้เสมอ ซึ่งมีแนวทางดังนี้



ลำดับ	หลักการ	ตัวอย่างโค้ด
1	สมมติว่า เวลาที่กำหนดเป็นวินาทีเท่ากับ 4000 วินาที	$t = 4000$
2	แปลงจากชั่วโมง $\Rightarrow$ วินาที ซึ่ง 1 ชั่วโมง เท่ากับ 3600 วินาที ดังนั้น ต้องเอา 3600 ไปหารจากจำนวนวินาทีแล้วแปลงเป็นจำนวนเต็ม	$h = 4000 / 3600 = 1$
3	หาค่าที่เหลือจากการหารจำนวนชั่วโมง โดยใช้วิธีการหารแบบอากรค่าที่เหลือ	$r = 4000 \% 3600 = 400$
4	เอาค่าที่เหลือจากการหารจำนวนชั่วโมง (วินาที) มาแปลงเป็นนาที โดย 1 นาที เท่ากับ 60 วินาที ดังนั้น ต้องเอา 60 ไปหารแล้วแปลงเป็นจำนวนเต็ม	$m = r / 60 = 400 / 60$ $m = 6$
5	ค่าที่เหลือจากการหารจำนวนนาที จะกลายเป็นวินาที ซึ่งเรากำหนดไว้ 60 นาที แค่เอาค่าจากข้อ 3 มาหารด้วย 60 แบบอากรค่าที่เหลือ	$s = r \% 60 = 400 \% 60$ $s = 40$
6	สรุปได้ว่า 4000 วินาที เท่ากับ 1 ชั่วโมง 6 นาที 40 วินาที	

```
#include <stdio.h>

void main() {
    int total_time_sec, remain,
        hours, minutes, seconds;

    //กำหนดเวลาเป็นวินาที
    printf("\nTotal time in second >>");
    scanf("%d", &total_time_sec);

    hours = total_time_sec / 3600; //ชั่วโมง
    remain = total_time_sec % 3600; //หาค่าที่เหลือ

    minutes = remain / 60; //นาที
    seconds = remain % 60; //ค่าที่เหลือคือ วินาที

    printf("\nhours: %d", hours);
    printf("\nminutes: %d:", minutes);
    printf("\nseconds: %d \n", seconds);
}
```

```
total time in second >>8800
hours: 2
minutes: 26:
seconds: 40
-1000
```

**ตัวอย่าง 5-5** เปรียบเทียบผลลัพธ์ที่ได้ระหว่างการหาร จำนวนเต็ม กับ จำนวนเต็ม

```
#include <stdio.h>

void main() {
    int n1 = 55, n2 = 10;

    int a = n1 / n2;
    float b = n1 / n2;
    float c = (float)n1 / n2;
    float d = n1 / (float)n2;
    float e = (float)n1 / (float)n2;
    printf(
        "\na = %d, b = %g, c = %g, d = %g, e = %g \n",
        a, b, c, d, e
    );
}
```

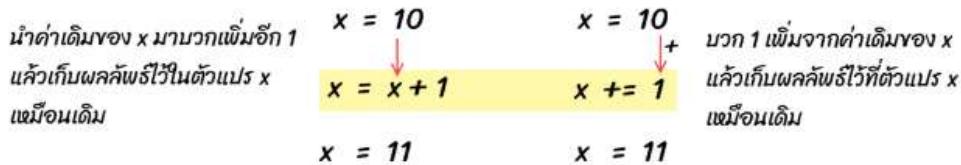
a = 5, b = 5, c = 5.5, d = 5.5, e = 5.5

## เครื่องหมายสำหรับการคำนวณและกำหนดค่า

เครื่องหมายสำหรับการคำนวณและกำหนดค่า (Arithmetic Assignment Operator หรือ Compound Operator) เป็นการนำเครื่องหมายสำหรับการคำนวณ (Arithmetic) มารวมกับ เครื่องหมายสำหรับกำหนดค่า (Assignment) เพื่อให้เป็นเครื่องหมายอันเดียวกัน ซึ่งมีดังนี้

เครื่องหมาย	ชื่อเรียก	ตัวอย่าง
$+=$	Addition Assignment (บวกเพิ่มด้วยค่าเท่ากัน)	$x += 10$
$-=$	Subtraction Assignment (ลบออกด้วยค่าเท่ากัน)	$x -= 50$
$*=$	Multiplication Assignment (คูณด้วยค่าเท่ากัน)	$x *= 20$
$/=$	Float Division Assignment (หารด้วยค่าเท่ากัน)	$x /= 10$
$\%=$	Remainder Assignment (หารแบบเอาค่าที่เหลือด้วยค่าเท่ากัน)	$x \%= 3$

- ข้อมูลผลลัพธ์ของเครื่องหมายทุกอัน จะถูกแปลงให้สอดคล้องกับชนิดข้อมูลของตัวแปรที่ใช้เก็บผลลัพธ์
- เครื่องหมาย  $+=$  เป็นการบวกค่าเดิมของตัวแปรด้วยค่าที่อยู่ทางด้านขวาของเครื่องหมาย แล้วผลลัพธ์หลังการบวก ก็จัดเก็บไว้ในตัวแปรนั้นตามเดิม (เหมือนกับการแทนที่ค่าเดิม) เช่น



```
int x = 10;
x += 1;           // x = x + 1 = 10 + 1 = 11
x += 1.5;        // x = (int)12.5 = 12

float y = 4.5;
y += 0.5;        // y = 4.5 + 0.5 = 5.0

short z = 10, a = 6;
z += -2;         // z = 10 + (-2) = 8
z += a;          // z = 8 + 6 = 14
```

- เครื่องหมาย  $-=$  เป็นการลบค่าเดิมของตัวแปรด้วยค่าทางด้านขวาของเครื่องหมาย เช่น

```
int x = 20;
x -= 15;        // x = x - 10 = 20 - 15 = 5

float y = -10.5;
y -= -0.5;      // y = -10.5 - (-0.5) = -10.0
```

- เครื่องหมาย  $*=$  เป็นการคูณค่าเดิมของตัวแปรด้วยค่าทางด้านขวาของเครื่องหมาย เช่น

```
int x = 5;
x *= 2;         // x = 5 * 2 = 10
x *= 1.25;      // x = (int)(12.5) = 12

float y = -2.5;
y *= -4;        // y = -2.5 * (-4) = 10.0
```

- เครื่องหมาย  $/=$  เป็นการหารค่าเดิมของตัวแปรด้วยค่าทางด้านขวาของเครื่องหมาย เช่น

```
int x = 100;
x /= 20;        // x = 100 / 20 = 5

float y = -40.8;
y /= -4;        // y = -40.8 / (-4) = 10.2
```

- เครื่องหมาย `%=` เป็นการหารค่าเดิมของตัวแปรด้วยค่าทางด้านขวาของเครื่องหมาย เพื่อหาค่าที่เหลือ (Remainder) ซึ่งเราก็ใช้หลักการเดียวกับเครื่องหมาย `%` ดังที่กล่าวมาแล้ว ถ้าหารไม่ลงตัวต้องเปลี่ยนเป็นค่ามากที่สุดที่ไม่เกินค่าของตัวตั้งและหารได้ลงไป แล้วคำนวนหาที่เหลือ ซึ่งก็คือผลต่างระหว่างค่านั้น ตามหลักการเดิม เช่น

```

int x = 31;
x %= 4;
//x = 31 % 4 => 28 % 4 => 31 - 28 = 3

int y = -26;
y %= 8;
//y = -26 % 8 => -24 % 8 => -26 - (-24) = -2

int z = -15;
z %= -7;
//z = -15 % -7 => -14 % -7 => -15 - (-14) = -1

```

**ตัวอย่าง 5-6** รับตัวเลข 5 จำนวนทางคีย์บอร์ด และหาผลรวมและค่าเฉลี่ย

```

#include <stdio.h>

void main() {
    int n1, n2, n3, n4;
    int sum = 0;           //เริ่มแรกให้ผลรวมเป็น 0
    float average;

    printf("\nvalue #1 >>");
    scanf("%d", &n1);
    sum += n1;

    printf("value #2 >>");
    scanf("%d", &n2);
    sum += n2;

    printf("value #3 >>");
    scanf("%d", &n3);
    sum += n3;

    printf("value #4 >>");
    scanf("%d", &n4);
    sum += n4;
}

```

```

value #1 >>30
value #2 >>20
value #3 >>50
value #4 >>10

sum = 110
average = 27.5

```

```

average = (float)sum / 4;
//สาเหตุที่ต้องคลาสต์ ก็ตามที่ได้กล่าวไปในหัวข้อก่อนนี้

printf("\nsum = %d", sum);
printf("\naverage = %g \n", average);
}

```

**ตัวอย่าง 5-7** สมมติว่ามียอดเงินคงเหลือในบัญชีจำนวนหนึ่ง (balance) ให้รับตัวเลขทางคีย์บอร์ด หากเป็นการฝาก (deposit) ให้เพิ่มจากค่าเดิมของยอดคงเหลือ แต่หากเป็นการถอน (withdraw) ให้ลบออกจากยอดคงเหลือ

```

#include <stdio.h>

void main() {
    int w, d, b = 1000;

    printf("\nbalance: %d", b);

    printf("\n\nwithdraw >>");
    scanf("%d", &w);
    b -= w;
    printf("balance: %d", b);

    printf("\n\ndeposit >>");
    scanf("%d", &d);
    b += d;
    printf("balance: %d", b);

    printf("\n\ndeposit >>");
    scanf("%d", &d);
    b += d;
    printf("balance: %d", b);

    printf("\n\nwithdraw >>");
    scanf("%d", &w);
    b -= w;
    printf("balance: %d", b);

    putchar('\n');
}

```

```

balance: 1000
withdraw >>400
balance: 600
deposit >>800
balance: 1400
deposit >>600
balance: 2000
withdraw >>900
balance: 1100

```

## เครื่องหมายสำคัญเพิ่มและลดค่า

เครื่องหมายล้ำหรับเพิ่มและลดค่า (Increment & Decrement Operator) จะใช้กับข้อมูลเพียงจำนวนเดียว ซึ่งเครื่องหมายในกลุ่มนี้ประกอบด้วย

เครื่องหมาย	ชื่อเรียก	ตัวอย่าง
<code>++</code>	Increment (เพิ่มจากค่าเดิมไปอีก 1)	<code>x++</code> หรือ <code>++x</code>
<code>--</code>	Decrement (ลดจากค่าเดิมลงไป 1)	<code>x--</code> หรือ <code>--x</code>

- เครื่องหมาย `++` เป็นการเพิ่มค่าของข้อมูลจำนวนนั้นจากเดิมไปอีก 1 จึงเทียบเท่ากับการใช้เครื่องหมาย `=+ 1` นั่นเอง เช่น

```
int a = 10;
a++;           //a = 11 หรือเทียบเท่ากับ a += 1

float b = 10.75;
++b;           //b = 11.75 หรือเทียบเท่ากับ b += 1

short c = -999;
short d = ++c; //d = -998
```

- เครื่องหมาย `--` เป็นการลดค่าข้อมูลจำนวนนั้นจากเดิมลงอีก 1 จึงเทียบเท่ากับการใช้เครื่องหมาย `=- 1` นั่นเอง เช่น

```
int a = 10;
a--;           //a = 9 หรือเทียบเท่ากับ a -= 1

float b = 10.75;
--b;           //b = 9.75 หรือเทียบเท่ากับ b -= 1

short c = -999;
short d = --c; //d = -1000 (-999-1)
```

- การวางเครื่องหมาย `++` หรือ `--` ไว้ด้านหน้า (Prefix) หรือด้านหลัง (Suffix หรือ Postfix) ของจำนวน หากตัวแปรนั้นอยู่เพียงลำพัง ค่าที่ได้จะไม่ต่างกัน เช่น

```
int x = 10;
x++;           //x = 11
++x;           //x = 12
```

```
x--;           //x = 11
--x;           //x = 10
```

- หากนำเครื่องหมาย `++` หรือ `--` ไปกระทำร่วมกับจำนวนอื่นๆ จะมีหลักการพิจารณาที่แตกต่างออกไป ดังนี้
  - ถ้าทางเครื่องหมาย `++` หรือ `--` ไว้ด้านหน้าจำนวนใด ให้เพิ่มหรือลดค่าของจำนวนนั้นอีก 1 ก่อน แล้วค่อยนำค่าที่ได้ไปกระทำกับจำนวนอื่น หรือสรุปสิ้นๆ ก็คือ ถ้าทางไว้ข้างหน้า ก็ให้คิดตัวที่มี `++` หรือ `--` ก่อนนั้นเอง เช่น

โค้ดที่เขียน	วิธีคิด
$a = 10;$ $b = ++a;$	(1) $++a = 10 + 1 = 11$ (2) $b = 11$
$c = 100;$ $d = --c;$	(1) $--c = c - 1 = 99$ (2) $d = 99$
$a = 10;$ $b = 100;$ $e = ++a + --b;$	(1) $++a = 10 + 1 = 11$ (2) $--b = 100 - 1 = 99$ (3) $e = 11 + 99 = 110$
$a = 100;$ $a = ++a;$	(1) $++a = 100 + 1 = 101$ (2) $a = ++a = 101$
$b = 100;$ $b = ++b + ++b;$	(1) $++b$ (ตัวแรก) $= 100 + 1 = 101$ (2) $++b$ (ตัวหลัง) $= 101 + 1 = 102$ (3) $++b + ++b = 101 + 102 = 203$ (4) $b = 203$
$c = 100;$ $c += ++c;$	(1) $++c = (100 + 1) = 101$ (2) $c += ++c \Rightarrow c = c + (++c) = 100 + 101$ (3) $c = 201$
$d = 9;$ $e = 99;$ $e -= --e - ++d;$	(1) $--e = 99 - 1 = 98$ (2) $++d = 9 + 1 = 10$ (3) $--e - ++d = 98 - 10 = 88$ (4) $e -= 88 \Rightarrow e = e - 88$ (5) $e = 99 - 88 = 11$ (ค่าเดิมของ $e$ คือ 99)

- ถ้าวางเครื่องหมาย `++` หรือ `--` ไว้ด้านหลังของจำนวน กรณีมีวิธีการคิดยุ่งยากเล็กน้อย โดยให้เพิ่มหรือลดค่าด้วย `++` และ `--` ทีหลังเครื่องหมายประเภท Assignment เช่น `=`, `+=`, `-=` เป็นต้น ให้ดูแนวทางจากโค้ดต่อไปนี้

โค้ดที่เขียน	วิธีคิด
<code>a = 10;</code> <code>b = a++;</code>	(1) $b = a = 10$ กำหนดค่าด้วย ก่อน ส่วน <code>++</code> ทำทีหลัง (2) $a = 10 + 1 = 11$
<code>c = 100;</code> <code>d = c--;</code>	(1) $d = c = 100$ (2) $c-- = 100 - 1 \Rightarrow c = 99$
<code>c = 100;</code> <code>c = c++;</code> <code>d = c;</code>	(1) $c = c = 100$ (2) <code>c++</code> ไม่มีผล (3) $d = c = 100$
<code>c = 100;</code> <code>c += c++;</code> <code>d = c;</code>	(1) $c += c = 100 + 100 = 200$ (2) $c++ = 200 + 1 \Rightarrow c = 201$ (3) $d = c = 201$
<code>c = 100;</code> <code>d = c++ + 5;</code>	(1) $d = c + 5 = 105$ (คำนวน <code>++</code> ทีหลัง) (2) $c++ = 100 + 1 = 101$
<code>a = 5;</code> <code>b = 10;</code> <code>c = a++ * b--;</code>	(1) $c = 5 * 10 = 50$ (2) $a++ = 5 + 1 \Rightarrow a = 6;$ (3) $b-- = 10 - 1 \Rightarrow b = 9$
<code>a = 5;</code> <code>b = 10;</code> <code>c = a++ * --b;</code>	(1) $--b = 10 - 1 = 9$ (2) $c = 5 * 9 = 45$ (3) $a++ = 5 + 1 = 6$

ความจริงแล้ว หลักการคำนวณดังที่กล่าวมานี้ พิจารณาตามลำดับความสำคัญของเครื่องหมาย ดังที่จะกล่าวถึงในหัวข้อถัดไป

## ลำดับการประมวลผลของเครื่องหมาย

หากเราต้องใช้เครื่องหมายร่วมกันมากกว่า 1 อย่างสำหรับการคำนวณแบบต่อเนื่องกัน ก็อาจเกิดปัญหาว่า จะนำเครื่องหมายใดมาพิจารณาก่อน เพราะการวางแผนหรือจัดกลุ่มที่ต่างกัน ผลลัพธ์ที่ได้ก็อาจแตกต่างกันไปด้วย เช่น ลองพิจารณาโค้ดต่อไปนี้

```
x = 10 + 20 / 5; //ถ้าคิดในแบบ  $(10 + 20) / 5 = 6$   
//ถ้าคิดในแบบ  $10 + (20 / 5) = 14$ 
```

จากโค้ด หากพิจารณาในมุมมองของเราว่า อาจตัดสินใจไม่ได้ว่าอันไหนถูกต้อง ดังนั้น ในภาษา C (รวมถึงภาษาคอมพิวเตอร์อื่นๆ) จึงมีการจัดลำดับความสำคัญของเครื่องหมาย ซึ่งจากเครื่องหมายการคำนวณทางคณิตศาสตร์ที่เราได้ศึกษามา สามารถจัดลำดับความสำคัญหรือ ลำดับการประมวลผลได้ดังนี้

ลำดับ	เครื่องหมาย
1	( ) พิจารณางานเป็นอันดับแรก
2	-x (ค่าติดลบ เช่น -5)
3	++x, --x (วงไว้ข้างหน้า)
4	*, /, //, %
5	+, -
6	=, +=, -=, /=, %=
7	x++, x-- (วงไว้ข้างหลัง)

เครื่องหมายที่มีลำดับเท่ากัน เครื่องหมายที่มาก่อนจะถูกนำมาใช้ในการคำนวณก่อน (หรือ ดำเนินการจากซ้ายไปขวา) โดยแนวทางการใช้งาน มีดังตัวอย่างด้านล่างนี้

**ตัวอย่าง 5-8** เป็นการทดสอบลำดับความสำคัญของเครื่องหมายทางคณิตศาสตร์

```
#include <stdio.h>

void main() {
    int a, b, c, d, e, f, g;

    a = 2 * 100 / 10;
    //a = (2 * 100) / 10 = 20

    b = 100 / 10 * 2;
    //b = (100 / 10) * 2 = 20

    c = 1 + 100 * 10;
    //c = 1 + (100 * 10)
```

```

d = 2 * 100 / 10 + 30;
//d = ((2 * 100) / 10) + 30

e = 5;
f = 10;
g = e++ * --f;
//g = 5 * (10 - 1) = 45
//เนื่องจาก e++ มีลำดับความสำคัญต่ำสุด
//ดังนั้น การประมวลผลด้วย * และกำหนดค่าด้วย =
//จึงเกิดขึ้นก่อน e++
printf(
    "\na = %d \nb = %d \nc = %d \nd = %d \ng = %d",
    a, b, c, d, g
);
}

```

```

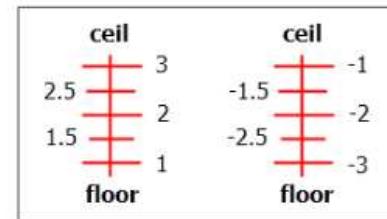
a = 20
b = 20
c = 1001
d = 50
g = 45

```

ในทางปฏิบัติเพื่อทำให้ได้อ่านง่ายขึ้น และหลีกเลี่ยงข้อผิดพลาดจากลำดับความสำคัญของเครื่องหมาย ควรใช้วงเล็บในการจัดแบ่งกลุ่มให้ชัดเจน เมื่ອนกับที่แสดงในคำอธิบาย

## ฟังก์ชันทางด้านคณิตศาสตร์

ในภาษา C จะมีฟังก์ชันทางด้านคณิตศาสตร์ให้เราเลือกใช้อยู่จำนวนหนึ่ง แต่ฟังก์ชันเหล่านี้อยู่ในเขตเดอร์ **math.h** ดังนั้น เราต้องอ้างอิงด้วยคำสั่ง **include** เลยก่อนจะใช้งานได้ โดยในที่นี้จะกล่าวถึงเฉพาะฟังก์ชันที่น่าสนใจซึ่งเรามีโอกาสได้ใช้งานสำหรับการเขียนโปรแกรมทั่วๆ ไปเท่านั้น (ภาพขวามือใช้ประกอบการพิจารณาฟังก์ชัน **ceil** และ **floor**)



<b>abs(เลขจำนวนเต็ม)</b>	หาค่า Absolute ของเลขที่กำหนด เช่น <code>abs(-10) =&gt; 10</code>
<b>pow(เลขฐาน, ยกกำลัง)</b>	หาค่าเลขยกกำลัง เช่น <code>pow(10, 3) =&gt; 1000</code>
<b>sqrt(จำนวนบวก)</b>	รากที่สองของจำนวนที่ระบุ (ต้องไม่ติดลบ) เช่น <code>sqrt(100) = 10</code>
<b>cbrt(ตัวเลข)</b>	(cube root) หารากที่สามของจำนวนที่ระบุ เช่น <code>cbrt(1000) = 10</code>
<b>round(ตัวเลข)</b>	ประมาณค่าเป็นเลขจำนวนเต็มที่ใกล้เคียง ถ้าเศษหรือทศนิยม ตั้งแต่ 0.5 จะปัดเป็นจำนวนเต็มถัดไป เช่น <ul style="list-style-type: none"> <li>• <code>round(1.1) = 1</code></li> <li>• <code>round(0.5) = 1</code></li> <li>• <code>round(-1.1) = -1</code></li> <li>• <code>round(1.5) = 2</code></li> <li>• <code>round(0.49) = 0</code></li> <li>• <code>round(-1.5) = -2</code></li> </ul>

<b>ceil(ตัวเลข)</b>	จำนวนเต็มที่อยู่ตัดไปจากค่าที่ระบุ (จากภาพที่แล้ว ให้ดูที่ด้านบนของตัวเลขที่ระบุ) เช่น
	<ul style="list-style-type: none"> <li>• <math>\text{ceil}(1.0) = 1</math></li> <li>• <math>\text{ceil}(1.49) = 2</math></li> <li>• <math>\text{ceil}(1.5) = 2</math></li> <li>• <math>\text{ceil}(1.1) = 2</math></li> <li>• <math>\text{ceil}(-1.1) = -1</math></li> <li>• <math>\text{ceil}(-1.5) = -1</math></li> </ul>
<b>floor(ตัวเลข)</b>	จำนวนเต็มที่อยู่ก่อนค่าที่ระบุ (จากภาพที่แล้ว ให้ดูจำนวนเต็มที่อยู่ด้านล่างของตัวเลขที่ระบุ) เช่น
	<ul style="list-style-type: none"> <li>• <math>\text{floor}(1.0) = 1</math></li> <li>• <math>\text{floor}(1.5) = 1</math></li> <li>• <math>\text{floor}(1.9) = 1</math></li> <li>• <math>\text{floor}(1.1) = 1</math></li> <li>• <math>\text{floor}(-1.1) = -2</math></li> <li>• <math>\text{floor}(-1.5) = -2</math></li> </ul>
<b>trunc(ตัวเลข)</b>	ตัดส่วนทศนิยมทั้งไป ไม่ว่าจะเป็นเท่าไหร่ก็ตาม เช่น
	<ul style="list-style-type: none"> <li>• <math>\text{math.trunc}(10.99) = 10</math></li> <li>• <math>\text{math.trunc}(-10.5) = -10</math></li> </ul>
<b>exp(x)</b>	หาค่า exponential ( $e^x$ ) เช่น $\text{exp}(2) = e^2 = 7.3890$
<b>log(x)</b>	หาค่า logarithm มีเลขฐานเป็น e เช่น $\text{log}(2.71828) = 1.0$
<b>sin(radian)</b>	หาค่า sine ของมุมในหน่วย เรเดียน เช่น $\text{sin}(3.14/2) = 1.0$
<b>cos(radian)</b>	หาค่า cosine ของมุมในหน่วย เรเดียน เช่น $\text{cos}(3.14) = -1.0$
<b>tan(radian)</b>	หาค่า tangent ของมุมในหน่วย เรเดียน เช่น $\text{tan}(3.14/4) = 1.0$
<b>degrees(radian)</b>	แปลงจากค่าเรเดียนเป็นองศา เช่น $\text{degrees}(3.14/2) = 90$
<b>radians(degree)</b>	แปลงจากค่าองศาเป็นเรเดียน เช่น $\text{radians}(90) = 1.57$

ตัวอย่าง 5-9 แนวทางการใช้ฟังก์ชันทางด้านคณิตศาสตร์

```
#include <stdio.h>
#include <math.h>

void main() {
    printf("\n%g", ceil(5.55));      //6
    printf("\n%g", floor(1.23));     //1
    printf("\n%g", sqrt(100));       //10

    int radius = 10;
    float area = 3.141 * pow(radius, 2);
    printf("\narea = %g", area);     //314.1
}
```

## การสร้างเลขสุ่ม

ในบางกรณี ค่าที่จะกำหนดให้แก่ตัวแปรอาจมีความไม่แน่นอน สามารถเป็นไปได้หลายอย่าง จึงระบุเอาไว้ล่วงหน้าไม่ได้ และไม่อาจรับค่าจากผู้ใช้ทางคีย์บอร์ดอีกด้วย เช่น

ค่าของตัวแปร  $a$  อาจเป็นค่าใดค่าหนึ่งระหว่าง 1 - 9

หรือค่าของตัวแปร  $b$  จะเป็นอะไรก็ได้ที่ไม่เกิน 100

ลักษณะการกำหนดค่าที่ไม่แน่นอนดังกล่าวนี้ เราสามารถนำวิธีการสร้างเลขสุ่มมาใช้ได้โดยในภาษา C มีวิธีการสร้างเลขสุ่มได้ อย่างไรก็ตาม เพื่อให้ได้เลขที่มีค่าไม่แน่นอนในแต่ละครั้ง ผู้เขียนจะแนะนำแนวทางในการสร้างเลขสุ่มที่เรานิยมใช้งานจริงไปเลย ดังขั้นตอนต่อไปนี้

- กรณีนี้ เอ็ดเดอร์ที่เราต้องอ้างอิงด้วย `#include` ก็คือ `stdlib.h` และ `time.h`
- ก่อนที่จะสร้างเลขสุ่มให้เรียกฟังก์ชัน `srand(ตัวเลข)` โดยตัวเลขที่กำหนดให้แก่ฟังก์ชันนี้ ควรเป็นเลขที่มีค่าไม่แน่นอน เพื่อให้การสุ่มแต่ละครั้งได้ผลลัพธ์ที่แตกต่างกันออกไป และล้วนใหญ่ เรานิยมใช้ค่าเวลาปัจจุบัน ซึ่งกำหนดด้วยฟังก์ชัน `time(0)` เพราะค่าเวลาในแต่ละขณะจะเปลี่ยนแปลงไปเรื่อยๆ
- หากเลขสุ่มด้วยฟังก์ชัน `rand()` ซึ่งจะได้เป็นเลขจำนวนเต็ม ทั้งนี้ เมื่อเรียกฟังก์ชัน `rand()` แต่ละครั้ง จะได้เลขที่แตกต่างกัน (โค้ดถัดไป เป็นแค่แนวทางเฉพาะส่วนที่เกี่ยวข้อง และผลลัพธ์ที่ได้จะไม่แน่นอน)

```
#include <stdlib.h>           // สำหรับฟังก์ชันการสุ่มตัวเลข
#include <time.h>             // สำหรับฟังก์ชันอ่านเวลาปัจจุบัน
...
srand(time(0));              // ให้ตัวแปลงภาษาจัดเตรียมการสร้างเลขสุ่ม
printf("\n%d", rand());      // 2743
printf("\n%d", rand());      // 257
printf("\n%d", rand());      // 16282
printf("\n%d", rand());      // 32784
```

- ถ้าเราต้องการเลขสูงสุดเท่าใด อาจนำเลขนั้นมาหารแบบเอาค่าที่เหลือ (เครื่องหมาย %) เช่น (มีคำอธิบายเพิ่มเติมถัดจากโค้ด)
  - ถ้าต้องการค่าน้อยกว่า 10 ให้หารด้วย 10 เช่น `rand() % 10`
  - ถ้าต้องการค่าน้อยกว่า 100 ให้หารด้วย 100 เช่น `rand() % 100`
  - ถ้าต้องการค่าน้อยกว่า 1000 ให้หารด้วย 1000 เช่น `rand() % 1000`
  - ถ้าต้องการค่าน้อยกว่า 20 ให้หารด้วย 20 เช่น `rand() % 20`
  - ...

- ◎ ถ้าต้องการค่า ไม่เกิน 10 ให้หารด้วย 11 เช่น `rand() % 11`
- ◎ ถ้าต้องการค่า ไม่เกิน 100 ให้หารด้วย 101 เช่น `rand() % 101`
- ◎ ...

```

srand(time(0));
int r = rand();
int a = r % 10;           // จะได้ค่า 0 - 9
int b = rand() % 100;    // จะได้ค่า 0 - 99
int c = rand() % 1000;   // จะได้ค่า 0 - 999
int d = rand() % 20;     // จะได้ค่า 0 - 19

int e = rand() % 11;    // จะได้ค่า 0 - 10
int f = rand() % 101;   // จะได้ค่า 0 - 100
int g = rand() % 21;    // จะได้ค่า 0 - 20

```

ผู้อ่านอาจสงสัยว่า การนำเลข 10, 100, ... ไปหารแบบเอาค่าที่เหลือ มันจะได้ผลลัพธ์เป็นอย่างไร ซึ่งขออธิบายเพิ่มเติม ดังแนวทางด่อไปนี้

`int r = rand();`  
 สมมติว่าสุ่มได้เลข 1234 ถ้าต้องการค่าน้อยกว่า 10  
 $1234 / 10 = 123.4 \Rightarrow 1234 \% 10 \Rightarrow 4$   
 หรือคิดง่ายๆ ว่าคือเศษตัวสุดท้ายของมันนั่นเอง  
 ถ้าเราหารด้วย 10 ถึงอย่างไรเศษที่เหลือก็ต้องน้อยกว่า 10

สมมติว่าสุ่มได้เลข 2029 ถ้าต้องการค่าน้อยกว่า 10  
 $2029 / 10 = 202.9 \Rightarrow 2029 \% 10 \Rightarrow 9$

สมมติว่าสุ่มได้เลข 35790 ถ้าต้องการค่าน้อยกว่า 10  
 $35790 / 10 = 3579.0 \Rightarrow 35790 \% 10 \Rightarrow 0$

สมมติว่าสุ่มได้เลข 1234 ถ้าต้องการค่าน้อยกว่า 100  
 $1234 / 100 = 12.34 \Rightarrow 1234 \% 100 \Rightarrow 34$   
 หรือคิดง่ายๆ ว่าคือเศษสองตัวสุดท้ายของมันนั่นเอง

สมมติว่าสุ่มได้เลข 35790 ถ้าต้องการค่าน้อยกว่า 100  
 $35790 / 100 = 357.90 \Rightarrow 35790 \% 100 \Rightarrow 90$

สมมติว่าสุ่มได้เลข 1009 ถ้าต้องการค่าน้อยกว่า 100  
 $1009 / 100 = 10.09 \Rightarrow 1009 \% 100 \Rightarrow 09 = 9$

ถ้าต้องการค่า ไม่เกิน 10  
 เราแก้ `a` 11 ไปหารแบบเอาค่าที่เหลือ  
 เพื่อให้ค่าที่เหลือ (เศษ) มีค่าสูงสุดได้แค่ 10  
 เพราะค่าที่เหลือไม่มีทางเท่ากับหรือเกินตัวหาร เช่น

```

35791 % 11 = 8
35792 % 11 = 9
35793 % 11 = 10
35794 % 11 = 0

```

หรือสามารถสรุปเป็นสูตรการสร้างเลขสุ่มระหว่าง 0 - max ได้ดังนี้

```
r_0_max = rand() % (max + 1)
```

แต่กรณีที่เราต้องการค่าในช่วงที่ไม่ต้องเริ่มจาก 0 เช่น 1-5, 10-20, 100-999 ให้ใช้สูตรดังนี้

```
r_min_max = min + rand() % (max - min + 1)
```

เมื่อ min และ max คือค่าต่ำสุดและสูงสุดของช่วงตัวเลขที่ต้องการ ตามลำดับ ซึ่งหลักการโดยสังเขปคือ สมมติว่าเราต้องการเลขสุ่มระหว่าง 5-20 แสดงว่าหาตัวเลขมาบวกเข้ากับ 5 เพื่อให้ได้ผลบวกตั้งแต่ 5 ขึ้นไปแต่ไม่เกิน 20 นั่นคือ ค่าที่นำมาบวกจะต้องอยู่ระหว่าง 0 ถึง 15 ( $20 - 5$ ) เนื่องจาก  $5 + 0 = 5$  และ  $5 + 15 = 20$  ซึ่งล้วนที่เราต้องพิจารณาต่อไปคือ ทำอย่างไร จึงจะได้เลขระหว่าง 0-15 ซึ่งช่วงเลขสุ่มที่เริ่มจาก 0 - max นี้ ก็ไปตรงกับหลักการหาราคาเลขสุ่มที่เราเรียนรู้มาแล้ว ดังนั้น

$$r_{5\_20} = 5 + \text{rand}() \% (20 - 5 + 1)$$

ลองพิจารณาที่  $\text{rand}() \% (20 - 5 + 1) \Rightarrow \text{rand}() \% (16) \Rightarrow$  จะได้ค่าระหว่าง 0-15

ถ้าสุ่มได้เลข 0  $\Rightarrow 5 + 0 = 5$

ถ้าสุ่มได้เลข 1  $\Rightarrow 5 + 1 = 6$

ถ้าสุ่มได้เลข 5  $\Rightarrow 5 + 5 = 10$

...

ถ้าสุ่มได้เลข 14  $\Rightarrow 5 + 14 = 19$

ถ้าสุ่มได้เลข 15  $\Rightarrow 5 + 15 = 20$

สำหรับแนวทางการหาราคาเลขสุ่มในลักษณะต่างๆ รวมถึงประยุกต์ใช้งานแบบง่ายๆ ให้ดูจากตัวอย่างต่อไปนี้

**ตัวอย่าง 5-10** การสร้างเลขสุ่มตามหลักการที่กล่าวมา โดยกำหนดขอบเขตที่แตกต่างกัน

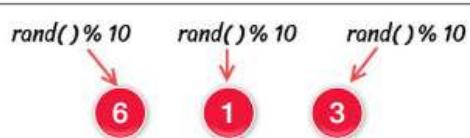
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void main() {
    srand(time(0));
    rand();
    printf("\n 0 - 10 => %d", rand() % 11);
    printf("\n 0 - 50 => %d", rand() % 51);
    printf("\n 0 - 100 => %d", rand() % 101);
    printf("\n 0 - 999 => %d \n", rand() % 1000);
    printf("\n-----");
    printf("\n 1 - 10 => %d", (1 + rand() % (10 - 1 + 1)));
    printf("\n 50 - 100 => %d", (50 + rand() % (100 - 50 + 1)));
    putchar('\n');
}
```

0 - 10 => 7
0 - 50 => 50
0 - 100 => 85
0 - 999 => 981
-----
1 - 10 => 3
50 - 100 => 97

**ตัวอย่าง 5-11** สร้างเลขสุ่มแบบจำนวนเต็ม โดยสมมติว่าเป็นการอกรางวัลเลขท้าย 2 ตัว จำนวน 1 รางวัล, เลขหน้า 3 ตัว และเลขท้าย 3 ตัวอย่างละ 2 รางวัล ซึ่งมีคำแนะนำดังนี้

- รางวัลเลขท้าย 2 ตัว ไม่ ควรใช้วิธีสร้างเลขสุ่มระหว่าง 0 - 99 เพราะอาจได้เลขหลักเดียว
- รางวัลเลขหน้าและท้าย 3 ตัว ไม่ ควรใช้วิธีสร้างเลขสุ่มระหว่าง 0 - 999 เพราะอาจได้เลข 1 - 2 หลัก
- สำหรับวิธีที่ควรใช้ในแต่ละรางวัลนั้น ให้สุ่มเลขโดยระหว่าง (0 - 9) ทีละหลัก แล้วนำมารวบกันจนครบตามจำนวนหลักของรางวัลนั้นๆ



```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void main() {
    srand(time(0));
    rand();

    // รางวัลเลขท้าย 2 ตัว
    printf(
        "\nlast two digits prize: %d %d",
        rand() % 10, rand() % 10);

    // รางวัลเลขหน้า 3 ตัว
    printf(
        "\nfirst three digits prize: %d %d %d",
        rand() % 10, rand() % 10, rand() % 10);

    // รางวัลเลขท้าย 3 ตัว
    printf(
        "\nlast three digits prize: %d %d %d",
        rand() % 10, rand() % 10, rand() % 10);
}
```

last two digits prize: 4 5
first three digits prize: 6 3 7
last three digits prize: 1 4 0

```

        rand()%10, rand()%10
    );

    //รางวัลเลขหน้า 3 ตัว
    printf(
        "\nfirst three digits prize: %d %d %d",
        rand()%10, rand()%10, rand()%10
    );

    //รางวัลเลขท้าย 3 ตัว
    printf(
        "\nlast three digits prize: %d %d %d \n",
        rand()%10, rand()%10, rand()%10
    );
}
}

```

**ตัวอย่าง 5-12** การสร้างรหัสผ่านแบบสุ่มที่ประกอบด้วย 0 - 9, a - z, A - Z รวมกัน 6 ตัว ซึ่งหลักการโดยลังเขปคือ

- นำอักษรที่จะใช้สุ่มเพื่อสร้างรหัสผ่านมาเรียงเป็นสตริงเดียวกัน เช่น '0123..89abcde...xyzABCD...XYZ' ซึ่งอักษรทั้งหมดรวมกันจะมีความยาวเท่ากับ  $10 + 26 + 26 = 62$
- วิธีการเข้าถึงอักษรตัวใดตัวหนึ่งภายในสตริง ให้เราระบุเลขลำดับของอักษรตัวนั้นไว้ในวงเล็บ [] ต่อท้ายชื่อตัวแปร โดยอักษรตัวแรกจะมีลำดับเป็น 0 เช่น
  - str[0] หมายถึง อักษรตัวแรกที่เก็บในตัวแปร str
  - str[1] หมายถึง อักษรตัวที่สอง
  - str[2] หมายถึง อักษรตัวที่สาม
  - ...
  - รายละเอียดทั้งหมดของเรื่องนี้ อยู่ในบทที่ 10 แต่เราจะนำสิ่งที่จำเป็นบางส่วนมาใช้งานล่วงหน้าไปก่อน
- สร้างเลขสุ่มให้ได้ค่าระหว่าง 0 - 61 (มีอักษร 62 ตัว โดยตัวแรกมีลำดับเป็น 0 แสดงว่าตัวสุดท้ายมีลำดับเป็น 61)
- นำตัวเลขที่สุ่มได้ไปใช้เป็นเลขลำดับในการอ่านนำอักษรจากสตริง จะได้อักษร 1 ตัว เช่น ถ้าสุ่มได้เลข 2 ก็อ่านอักษรในลำดับนั้นด้วย str[2] เป็นต้น
- เมื่อจากเราต้องการรหัสผ่านที่มีอักษร 6 ตัว ดังนั้น ต้องทำซ้ำกัน 6 ครั้ง

- นำอักษรที่ได้ในแต่ละครั้งมาเรียงต่อกัน ก็จะได้รหัสผ่านตามที่ต้องการ

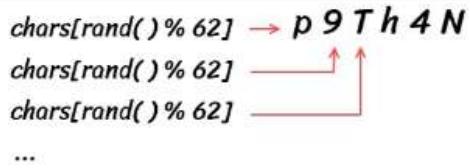
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

void main() {
    char chars[100] = "0123456789";
    strcat(chars, "abcdefghijklmnopqrstuvwxyz");
    strcat(chars, "ABCDEFGHIJKLMNOPQRSTUVWXYZ");

    srand(time(0));
    rand();

    int r = rand() % 62; // จะได้เลข 0-61
    char c1 = chars[r];
    char c2 = chars[rand() % 62];
    char c3 = chars[rand() % 62];
    char c4 = chars[rand() % 62];
    char c5 = chars[rand() % 62];
    char c6 = chars[rand() % 62];

    printf(
        "\npassword: %c%c%c%c%c%c \n",
        c1, c2, c3, c4, c5, c6
    );
    // ตัวอย่างผลลัพธ์ เช่น JT4H2m
}
```

*chars[rand() % 62]* → p 9 Th 4 N  
*chars[rand() % 62]*  
*chars[rand() % 62]*  
...  


## การแปลงสูตรคณิตศาสตร์เป็นโค้ดภาษา C

หากเราจำเป็นต้องคำนวณโดยใช้สูตรทางคณิตศาสตร์ ก็ต้องแปลงสูตรนั้นให้อยู่ในรูปแบบของโค้ดภาษา C ซึ่งสามารถสรุปแนวทางในเบื้องต้นได้ดังนี้

สูตร	โค้ดภาษา C
$ab$	$a * b$
$a^b$	$\text{pow}(a, b)$
$\frac{a}{b}$	$a / b$
$\sqrt{a}$	$\text{sqrt}(a)$ หรือ $\text{pow}(a, (1/2))$

สูตร	โค้ดภาษา C
$\sqrt[n]{a}$	$= a^{(1/n)} = \text{pow}(a, (1/n))$
$mc^2$	$m * \text{pow}(c, 2)$
$\sqrt{a^2 + b^2}$	$\text{sqrt}(\text{pow}(a, 2) + \text{pow}(b, 2))$
$\frac{4}{3}\pi r^3$	$(4 / 3) * 3.141 * \text{pow}(r, 3)$
$\frac{ab + c}{d + \frac{1}{e}}$	$((a * b) + c) / (d + (1 / e))$
$\frac{a \times b^2}{4 \times \tan\left(\frac{\pi}{a}\right)}$	$(a * \text{pow}(b, 2)) / (4 * (\tan(3.141 / a)))$

ตัวอย่าง 5-13 สูตรการหาปริมาตรทรงกลมคือ  $\frac{4}{3}\pi r^3$  ให้รับค่ารัศมีทางคีย์บอร์ด และคำนวณห้าปริมาตร

```
#include <stdio.h>
#include <math.h>

void main() {
    float radius, volume;

    printf("\nradius of sphere >>");
    scanf("%f", &radius);

    volume = (4/3) * (3.14) * pow(radius, 3);

    printf("volume = %g \n", volume);
}
```

radius of sphere >>15  
 volume = 10597.5

ตัวอย่าง 5-14 สูตรในการคำนวณมูลค่าในอนาคตของเงินลงทุน (Future Value) คือ

future\_value = amount × (1+rate)<sup>period</sup>

ซึ่งหากเราเขียนสูตรในแบบโค้ดภาษา C ก็จะเป็นดังนี้

future\_value = amount \* pow((1 + rate), period)

- amount คือจำนวนเงินที่ลงทุนเริ่มแรก
- rate คืออัตราดอกเบี้ย
- period คือช่วงระยะเวลาที่ลงทุน
- ทั้ง rate และ period ต้องเทียบกับช่วงเวลาที่เป็นหน่วยเดียวกัน เช่น ถ้า rate เป็น อัตราดอกเบี้ยแบบ % ต่อปี ดังนั้น period ก็ต้องมีระยะเวลาเป็นปีด้วย

ให้เขียนโค้ดเพื่อรับข้อมูลทางคีย์บอร์ดที่ต้องใช้ตามสูตรดังกล่าว แล้วคำนวณหาค่า Future Value

```
#include <stdio.h>
#include <math.h>

void main() {
    float amount, rate, fv;
    int year;

    // เงินลงทุนเริ่มแรก
    printf("\ninitial investment >>");
    scanf("%f", &amount);

    // อัตราดอกเบี้ยต่อปี (%)
    printf("yearly rate (%) >>");
    scanf("%f", &rate);

    // ระยะเวลาที่ลงทุน (ปี)
    printf("number of years >>");
    scanf("%d", &year);

    rate /= 100;    // เปลี่ยน % เป็นตัวเลข
    fv = amount * pow((1 + rate), year);

    printf("\nfuture value = %.2f \n", fv);
}
```

```
initial investment >>100000
yearly rate (%) >>1.5
number of years >>5
```

```
future value = 107728.39
```

**ตัวอย่าง 5-15** รูปหексายเหลี่ยม (Polygon) ที่ความยาวแต่ละด้านและมุมเท่ากันทั้งหมด มีสูตรในการคำนวณพื้นที่คือ

$$\text{polygon\_area} = \frac{n \times s^2}{4 \times \tan\left(\frac{\pi}{n}\right)}$$

ซึ่งหากเราเขียนสูตรในแบบโค้ดภาษา C ก็จะเป็นดังนี้

```
polygon_area = (n * pow(s, 2)) / (4 * tan(3.14/n))
```

เมื่อ n คือจำนวนด้าน และ s คือความยาวของแต่ละด้าน ให้เขียนโค้ดเพื่อรับข้อมูลเป็นจำนวนด้านและความยาวของแต่ละด้าน เล้าคำนวนหาพื้นที่ของรูปทรงดังกล่าว

```
#include <stdio.h>
#include <math.h>

void main() {
    int n;
    float s, a;

    //จำนวนด้านของรูปหลายเหลี่ยม
    printf("\nnumber of sides >>");
    scanf("%d", &n);

    //ความยาวของแต่ละด้าน
    printf("length of each side >>");
    scanf("%f", &s);

    a = (n * pow(s, 2)) / (4 * tan(3.14/n));

    printf("\npolygon area = %.2f \n", a);
}
```

```
number of sides >>6
length of each side >>10
polygon area = 259.97
```

ตัวเลขและการคำนวนทางคณิตศาสตร์ เป็นพื้นฐานสำคัญที่เราต้องนำไปใช้งานอยู่ตลอดในการเขียนโปรแกรม โดยเฉพาะอย่างยิ่ง การเลือกใช้เครื่องหมายหรือฟังก์ชันต่างๆ เราต้องเข้าใจหลักการและลักษณะของผลลัพธ์ที่เกิดขึ้น มีฉะนั้นเกิดข้อผิดพลาดจนส่งผลต่อการทำงานของโปรแกรมในขั้นตอนอื่นๆ ได้





# 6

## การเปรียบเทียบและกำหนดเงื่อนไข

ในบทนี้เราจะได้ศึกษาเกี่ยวกับการใช้คำสั่ง if, else if และ else เพื่อตรวจสอบเงื่อนไขก่อนที่จะดำเนินการในขั้นตอนต่อไป แต่อย่างไรก็ตาม การตรวจสอบเงื่อนไขดังกล่าวนั้นจะต้องใช้เครื่องหมายเปรียบเทียบและตรรกะ ดังนั้น ลิ่งที่เราจะต้องศึกษาเป็นลำดับแรกก็คือ การใช้เครื่องหมายในกลุ่ม Comparison และ Logic ซึ่งถือเป็นพื้นฐานที่สำคัญอย่างหนึ่งของการเขียนโปรแกรม เพื่อควบคุมการทำงานต่างๆ ให้อยู่ภายใต้เงื่อนไขที่เราต้องการ

### ข้อมูลชนิดบูลีน (\_Bool และ bool)

นอกจากข้อมูลชนิดตัวเลขและสตริงที่เราได้ศึกษาผ่านมาแล้ว ในภาษา C ยังมีชนิดข้อมูลพื้นฐานที่สำคัญอีกอย่างหนึ่ง นั่นคือ ข้อมูลแบบจริงเท็จ (บูลีน: Boolean) ซึ่งในภาษา C จะแทนด้วยคำว่า \_Bool และถือเป็นคำส่วนอีกคำหนึ่ง โดยลักษณะที่สำคัญของข้อมูลชนิดนี้คือ

- ต้องมีค่าอย่างใดอย่างหนึ่งระหว่าง
  - 0 สำหรับค่าที่เป็น เท็จ
  - 1 สำหรับค่าที่เป็น จริง

```
_Bool a, b;  
a = 1;           //จริง  
b = 0;           //เท็จ
```

- การใช้เลข 1 และ 0 แทนค่า จริงเท็จ ในภาษา C บางทีก็สร้างความสับสนเมื่อเราเปลี่ยนไปใช้ภาษาอื่นๆ หรือเคยใช้ภาษาอื่นแล้วมาเขียนภาษา C เนื่องจากภาษาคอมพิวเตอร์ส่วนใหญ่ในยุคปัจจุบันจะแทนค่าบูลีนด้วยคำว่า true และ false โดยตรงทั้งนี้ เราสามารถปรับเปลี่ยนให้เข้ากับรูปแบบของภาษาอื่นๆ ดังนี้

- ◎ อ้างอิงจากเดอร์ **stdbool.h** ด้วยคำสั่ง `include`
- ◎ กรณีนี้ เราอาจกำหนดชนิดข้อมูลเป็นคำว่า `_Bool` เหมือนเดิม หรือจะใช้คำว่า `bool` ก็ได้
- ◎ กำหนดค่าของตัวแปรเป็นคำว่า `true` หรือ `false` โดยตรงได้เลย ซึ่งก็จะมีค่าเทียบเท่ากับ 1 และ 0 ตามลำดับ

```
#include <stdbool.h>
...
_Bool a = true; //เทียบเท่ากับ 1
_Bool b = false; //เทียบเท่ากับ 0
bool c = true; //เทียบเท่ากับ 1
bool d = false; //เทียบเท่ากับ 0
bool e = 0; //เทียบเท่ากับ false
```

- ถ้าเรากำหนดค่าเป็นเลขอื่นๆ ที่ไม่ใช่ 1 กับ 0 หรือ `true` กับ `false` ให้กับตัวแปรชนิด `_Bool` หรือ `bool` ค่านั้นจะถูกเปลี่ยนเป็นเลข 1 เสมอ หรือให้มีค่าเป็น `true` นั้นเอง เช่น

```
bool a, b, c, d, e, f;
a = 0;
b = 1;
c = 2;
d = -1;
e = 3.141;
f = "xxx";

printf(
    "a = %d, b = %d, c = %d, d = %d, e = %d, f = %d",
    a, b, c, d, e, f
);
//ผลลัพธ์คือ a = 0, b = 1, c = 1, d = 1, e = 1, f = 1
```

สำหรับในหนังสือเล่มนี้ หากมีกรณีที่ต้องสร้างตัวแปรแบบจริงเท็จโดยตรง จะเลือกกำหนดค่าเป็นคำว่า `true/false` แม้ต้องอ้างอิงเดอร์ `stdbool.h` ก็ตาม เพื่อให้ผู้อ่านเกิดความเคยชิน เมื่อเปลี่ยนไปเขียนโปรแกรมภาษาอื่นๆ ในอนาคต

## เครื่องหมายสำหรับการเปรียบเทียบ

เครื่องหมายสำหรับการเปรียบเทียบ (Comparison Operator) ใช้สำหรับหาค่าความจริงระหว่างข้อมูล 2 อย่าง ซึ่งหากการเปรียบเทียบเป็นจริง ก็จะได้ผลลัพธ์เป็น true แต่หากเป็นเท็จ จะได้ผลลัพธ์เป็น false โดยเครื่องหมายในกลุ่มนี้คือ

เครื่องหมาย	ชื่อเรียก	การใช้งาน
<code>==</code>	เท่ากันกับ (Equality หรือ Equal To)	<code>a == b</code>
<code>!=</code>	ไม่เท่ากันกับ (Inequality หรือ Not Equal To)	<code>a != b</code>
<code>&gt;</code>	มากกว่า (Greater Than)	<code>a &gt; b</code>
<code>&lt;</code>	น้อยกว่า (Less Than)	<code>a &lt; b</code>
<code>&gt;=</code>	มากกว่าหรือเท่ากับ (Greater Than or Equal To)	<code>a &gt;= b</code>
<code>&lt;=</code>	น้อยกว่าหรือเท่ากับ (Less Than or Equal To)	<code>a &lt;= b</code>

- เครื่องหมายในกลุ่มนี้ จะให้ผลลัพธ์เป็น true (1) หรือ false (0) อย่างใดอย่างหนึ่งเท่านั้น
- เครื่องหมาย `==` เป็นการเปรียบเทียบว่าข้อมูลทั้ง 2 ค่าเท่ากันหรือไม่ เช่นในโค้ดถัดไป (ความจริง เราไม่จำเป็นต้องเขียนการเปรียบเทียบในวงเล็บก็ได้ แต่ถ้าเขียนจะอ่านโค้ดง่ายกว่า)

```

bool a = (1 == 2);      // 1 == 2 เป็นเท็จ ดังนั้น a = false
bool b = (5 == 5);      // 5 == 5 เป็นจริง ดังนั้น b = true
bool c = (a == b);      // false == true เป็นเท็จ
                        // ดังนั้น c = false

int x = 10;
int y = -20;
bool z = (x == y);      // 10 == -20 เป็นเท็จ ดังนั้น z = false
bool w = (z == false); // false == false เป็นจริง
                        // ดังนั้น w = true

bool v = (0 == -0.0);
// ไม่ว่า 0 จะเป็นบวกหรือลบก็มีค่าเท่ากันเสมอ ดังนั้น v = true

```

- เครื่องหมาย `!=` เป็นการเปรียบเทียบว่าทั้ง 2 ค่านั้น ไม่เท่ากัน ใช่หรือไม่ เช่น

```

bool a = (1 != 2);      //1 != 2 เป็นจริง ดังนั้น a = true
bool b = (5 != 5);      //5 != 5 เป็นเท็จ ดังนั้น b = false
bool c = (a != b);      //true != false เป็นจริง
                        //ดังนั้น c = true

int x = 10;
int y = -20;
bool z = (x != y);      //10 != -20 เป็นจริง ดังนั้น z = true
bool w = (z != 0);      //1 (true) != 0 เป็นจริง
                        //ดังนั้น w = true

```

- เครื่องหมาย `>` เป็นการเปรียบเทียบว่าค่าที่ระบุทางด้านซ้ายของเครื่องหมายมากกว่าค่าทางด้านขวาหรือไม่ เช่น

```

bool a = (1 > 2);      //1 > 2 เป็นเท็จ ดังนั้น a = false
bool b = (5 > 5);      //5 > 5 เป็นเท็จ ดังนั้น b = false
bool c = (a > b);      //0 (false) > 0 (false) เป็นเท็จ
                        //ดังนั้น c = false

int x = 10;
int y = -20;
bool z = (x > y);      //10 > -20 เป็นจริง ดังนั้น z = true
bool w = (z > 0);      //1 (true) > 0 เป็นจริง ดังนั้น w = true
bool v = (1 > 1.0);    //1 (true) > 1.0 เป็นเท็จ
                        //ดังนั้น v = false

```

- เครื่องหมาย `<` เป็นการเปรียบเทียบว่าค่าที่ระบุทางด้านซ้ายของเครื่องหมายน้อยกว่าค่าทางด้านขวาหรือไม่ เช่น

```

bool a = (1 < 2);      //1 < 2 เป็นจริง ดังนั้น a = true
bool b = (5 < 5);      //5 < 5 เป็นเท็จ ดังนั้น b = false
bool c = (a < b);      //1 (true) < 0 (false) เป็นเท็จ
                        //ดังนั้น c = false

int x = 10;
int y = -20;
bool z = (x < y);      //10 < -20 เป็นเท็จ ดังนั้น z = false
bool w = (0 < z);      //0 < 0 (false) เป็นเท็จ
                        //ดังนั้น w = false

```

- เครื่องหมาย  $\geq$  เป็นการเปรียบเทียบว่าค่าที่ระบุทางด้านซ้ายของเครื่องหมายมากกว่าหรือเท่ากับ ค่าทางด้านขวาหรือไม่ เช่น

```

bool a = (1 >= 2);      //1 >= 2 เป็นเท็จ ดังนั้น a = false
bool b = (5 >= 5);      //5 >= 5 เป็นจริง ดังนั้น b = true
bool c = (a >= b);      //0 (false) >= 1 (true) เป็นเท็จ
                        //ดังนั้น c = false

int x = 10;
int y = -20;
bool z = (x >= y);      //10 >= -20 เป็นจริง ดังนั้น z = true
bool w = (z >= 0);      //1 (true) >= 0 เป็นจริง ดังนั้น w = true

```

- เครื่องหมาย  $\leq$  เป็นการเปรียบเทียบว่าค่าที่ระบุทางด้านซ้ายของเครื่องหมายน้อยกว่าหรือเท่ากับ ค่าทางด้านขวาหรือไม่ เช่น

```

bool a = (1 <= 2);      //1 <= 2 เป็นจริง ดังนั้น a = true
bool b = (5 <= 5);      //5 <= 5 เป็นจริง ดังนั้น b = true
bool c = (a <= b);      //1 (true) <= 1 (true) เป็นจริง
                        //ดังนั้น c = true

int x = 10;
int y = -20;
bool z = (x <= y);      //10 <= -20 เป็นเท็จ ดังนั้น z = false
bool w = (z <= 0);      //0 (false) <= 0 เป็นจริง
                        //ดังนั้น w = true

```

- ความสามารถใช้เครื่องหมายในกลุ่มนี้ เพื่อเปรียบเทียบอักษรระหว่าง a-z (A-Z) ได้เช่นกัน โดยพิจารณาตามลำดับของอักษร เช่น 'a' < 'b' เป็นต้น ซึ่งเป็นหลักจริงๆ แล้วก็คือการเทียบจากรหัส ASCII ของอักษรนั้นเอง เช่น

```

bool a_b = ('a' < 'b');    //true
bool c_a = ('c' < 'a');    //false
bool x_z = ('x' < 'z');    //true
bool D_F = ('D' > 'F');   //false

```

## การเปรียบเทียบสตริง

การเปรียบเทียบด้วยเครื่องหมายหรือตัวดำเนินการเชิงเปรียบเทียบ ที่ได้กล่าวมานั้น อาจใช้ได้เฉพาะกับข้อมูลประเภทตัวเลขและอักษรระท่านั้น แต่ไม่สามารถมาใช้ในการเปรียบเทียบข้อมูลประเภทสตริงได้ ซึ่งหากนำมายัง แม้จะไม่เกิดข้อผิดพลาด แต่ก็ให้ผลลัพธ์ที่ไม่ถูกต้องสำหรับในภาษา C การเปรียบเทียบระหว่าง 2 สตริงว่าเท่ากันหรือไม่ จะต้องใช้ฟังก์ชันสำหรับกรณีนี้โดยตรง นั่นคือ `strcmp()` ดังรูปแบบและรายละเอียดต่อไปนี้

```
strcmp("สตริง1", "สตริง2")
```

- `strcmp` มาจากคำว่า string compare ซึ่งก็สื่อความหมายตรงตัวอยู่แล้วว่าใช้ในการเปรียบเทียบสตริง
- ฟังก์ชันนี้อยู่ในเขตเดอร์ `string.h` ซึ่งเราต้องอ้างอิงด้วย `include` ก่อนใช้งาน
- ผลลัพธ์ที่ได้จากฟังก์ชันนี้ จะไม่ใช่ค่าบูลีนแบบจริงเท็จ แต่จะเป็นตัวเลข ซึ่งผู้เขียนขอสรุปตามลักษณะตัวเลขที่เรานำไปใช้งานจริงเป็นส่วนใหญ่ ดังนี้คือ
  - ถ้าผลลัพธ์เป็นเลข 0 แสดงว่า สตริงทั้งสองอันนั้นเหมือนกัน (เป็นสตริงเดียวกัน) รวมถึงรูปแบบตัวพิมพ์ต้องตรงกันทุกอักษร
  - ถ้าผลลัพธ์เป็นเลขอื่นๆ ที่ไม่ใช่ 0 แสดงว่า สตริงทั้งสองอันนั้นไม่เหมือนกัน
- แนวทางการใช้งานในเมื่องต้น เช่น

```
#include <string.h>
...
int a = strcmp("Hello", "Hello"); //a = 0
int b = strcmp("Hello", "Hi"); //b != 0
int c = strcmp("Hello", "hello"); //c != 0 (ตัวพิมพ์ต่างกัน)
```

## ลักษณะพื้นฐานของคำสั่ง if

หากเรานำข้อมูลมาใช้ในการประมวลผลทันที โดยปราศจากการตรวจสอบความถูกต้องหรือข้อยกเว้นบางประการ ก็อาจก่อให้เกิดข้อผิดพลาด หรือไม่ได้ผลลัพธ์ตามต้องการ เช่น

```
int x = 10;
int y = 0;
float z = (float)x / y; //z มีค่าเป็น inf
```

จากโค้ด z มีค่าเป็น Infinity เพราะหารด้วย 0 ซึ่งลักษณะเช่นนี้เราจะวุ่นล่วงหน้า เพราะกำหนดค่าตัวแปรที่ແນ່ນອນເຂາໄວແລ້ວ แต่ถ้าสมมติว่าเราเปลี่ยนไปรับค่าทางคีย์บอร์ด ดังโค้ดด้านไป (เพื่อลดความยุ่งยาก จะแสดงโค้ดเพียงสั้นๆเท่านั้น)

```
scanf("%f", &x);
scanf("%f", &y);
z = x / y;
```

จากโค้ดข้างบน เราไม่อาจคาดเดาได้ว่าผู้ใช้จะใส่ค่าสำหรับตัวแปร y เป็นเลขอะไร ซึ่งถ้าใส่เป็นเลขที่ไม่ใช่ 0 ก็ไม่เกิดปัญหาอะไร แต่ถ้าใส่เลข 0 ก็จะเกิดปัญหาตามที่กล่าวมาแล้ว ด้วยเหตุนี้ในภาษาคอมพิวเตอร์แบบทั้งหมด จึงต้องมีกลไกการตรวจสอบข้อมูลบางอย่างเพื่อกำหนดทางเลือกก่อนกระทำการในขั้นตอนต่อไป เช่น จากโค้ดที่ผ่านมา ถ้าเรานำเงื่อนไขมาเขียนเป็นคำพูดในลักษณะของการตรวจสอบข้อมูลก่อนดำเนินการ ก็จะได้ดังนี้

```
scanf("%f", &x);
scanf("%f", &y);

ถ้า y เป็น 0 {
    ไม่ต้องดำเนินการต่อไป หรือแจ้งข้อผิดพลาด
}
ถ้า y ไม่เป็น 0 {
    z = x / y;
}
```

ในภาษา C จะใช้คำสั่ง if สำหรับการตรวจสอบข้อมูลว่าตรงกับเงื่อนไขที่ระบุหรือไม่ และกำหนดล็อกของกลุ่มคำสั่งที่จะดำเนินการต่อไป ถ้าหากเงื่อนไขเป็นจริง ซึ่งรูปแบบพื้นฐานคือ

```
if (เงื่อนไข) {
    คำสั่งต่างๆ ถ้าเงื่อนไขเป็นจริง (จะมีคำสั่งก็ได้)
}
```

- เงื่อนไขของคำสั่ง if ต้องเป็นข้อมูลที่มีค่าจริงเท็จ (1 หรือ 0) หรือการกระทำที่ให้ผลออกมากเป็นเลข 1 หรือ 0 เท่านั้น เช่น การเปรียบเทียบด้วย Comparison Operator หรือการใช้ Logical Operator ตามที่จะกล่าวถึงในหัวข้อต่อ ๆ ไป
- ต้องเขียนเงื่อนไขไว้ในวงเล็บ () เสมอ
- คำสั่งที่ต้องทำถ้าเงื่อนไขเป็นจริง (หรือ 1) ให้กำหนดไว้ในวงเล็บ {} แต่ถ้าเงื่อนไขเป็นเท็จ (หรือ 0) คำสั่งในบล็อกนี้จะถูกข้ามไป

สำหรับแนวทางการใช้งานเบื้องต้น จะกำหนดเงื่อนไขเป็นข้อมูลชนิดบูลีน (true/false) โดยตรงก่อน ส่วนเงื่อนไขที่เป็นการเปรียบเทียบด้วยเครื่องหมายหรือตัวดำเนินการ จะกล่าวถึง ในหัวข้อต่อๆ ไป

**ตัวอย่าง 6-1 แนวทางการเปรียบเทียบเงื่อนไขของ if ในเบื้องต้น**

```
#include <stdio.h>
#include <stdbool.h>

void main() {
    if (1) {
        puts("value 1 is true");
        // คำสั่งนี้ถูกดำเนินการ เพราะเงื่อนไขเป็นจริง (1 = true)
    }

    if (0) {
        puts("value 0 is true");
        // คำสั่งนี้ไม่ถูกดำเนินการ เพราะเงื่อนไขเป็นเท็จ (0 = false)
    }

    bool a = false;
    if (a) {
        puts("variable a is true");
        // คำสั่งในบล็อกนี้ไม่ถูกดำเนินการ
        // เพราะเงื่อนไขเป็นเท็จ
    }

    bool b = (1 == 1);      // b = true
    if (b) {
        puts("variable b is true");
        // คำสั่งในบล็อก if จะถูกดำเนินการ
        // เพราะเงื่อนไขเป็นจริง
    }

    bool c = true;
    if (c) {
        puts("variable c is true");
        // คำสั่งในบล็อก if จะถูกดำเนินการ
        // เพราะเงื่อนไขเป็นจริง
    }

    bool d = 0;
    if (d) {
        puts("variable d is true");
        // คำสั่งในบล็อก if ไม่ถูกดำเนินการ
    }
}
```

```
// เพราะเงื่อนไขเป็นเท็จ (0 = false)
}
}
```

อย่างไรก็ตาม หากภายในล็อกของ if มีเพียงคำสั่งเดียว เราไม่จำเป็นต้องเขียนคำสั่งนั้นไว้ในวงเล็บ {} ก็ได้ เช่น

```
if (true)
    puts("Hello");      // หากมีคำสั่งเดียว เขียนแบบนี้ได้

if (false)
    puts("Goodbye");   // หากมีคำสั่งเดียว เขียนแบบนี้ได้

if (...)
    puts("Hi");        // เฉพาะคำสั่งบรรทัดนี้ที่ขึ้นกับ if
    puts("Hey");       // คำสั่งนี้ ไม่ขึ้นกับ if จึงถูกดำเนินการเสมอ
```

อย่างไรก็ตาม ถึงแม่ในกรณีที่เงื่อนไข if มีเพียงคำสั่งเดียว เราไม่จำเป็นต้องกำหนดล็อกคำสั่งด้วยวงเล็บ {} ดังที่กล่าวมา ก็ตาม แต่ในทางปฏิบัติ เรายังหลีกเลี่ยงวิธีการดังกล่าว เพราะมีโอกาสเกิดข้อผิดพลาดได้ง่าย โดยเฉพาะอย่างยิ่งเมื่อเราเขียนโค้ดที่มีเงื่อนไขซ้อนกันหลายชั้น ดังนั้น แม้จะมีเพียงคำสั่งเดียว ก็ควรกำหนดล็อกด้วยวงเล็บ {} เสมอ

## การกำหนดเงื่อนไขด้วยเครื่องหมายเปรียบเทียบ

เครื่องหมายเปรียบเทียบ (Comparison Operator) ก็คือกลุ่มเครื่องหมายที่เราได้ศึกษาผ่านมาแล้วในหัวข้อก่อนๆ ของบทนี้ เช่น ==, >, <, >=, <= เป็นต้น เนื่องจากเครื่องหมายเหล่านี้ จะให้ผลลัพธ์ออกมาเป็นค่าจริงหรือเท็จ ดังนั้น จึงสามารถนำมาใช้สำหรับการกำหนดเงื่อนไขของคำสั่ง if ได้ ดังแนวทางต่อไปนี้

```
int a = 1;
if (a == 1) {      // ถ้าตัวแปร a มีค่าเท่ากับ 1
    puts("variable a stores 1");
}

int b = -5;
if (b == -5) {     // ถ้าตัวแปร b มีค่าเท่ากับ -5
    puts("variable a stores -5");
}
```

จากโค้ดข้างบน คำสั่งในบล็อก if จะถูกดำเนินการทั้งสองอัน เพราะเงื่อนไขตรงกับที่เรากำหนดหรือเป็นจริงทั้งคู่ อย่างไรก็ตาม สำหรับกรณีการใช้เครื่องหมาย == นั้น หากจะเปลี่ยนเที่ยบกับค่า true เราสามารถระบุเฉพาะชื่อตัวแปรก็ได้ ดังแนวทางในโค้ดด้านไป ซึ่งเราจะเขียนแบบคอลัมน์ด้านซ้ายหรือขวา ก็ให้ผลลัพธ์เหมือนกัน ซึ่งโดยทั่วไป เรานิยมเขียนแบบด้านขวามากกว่า

```
bool a = ...;
//true/false
```

```
if (a == true) {
    puts("...");
```

```
}
```

```
bool a = ...;
//false
```

```
if (a) {
    //ถ้า a เป็นจริง
    puts("...");
```

```
}
```

ส่วนในการเทียบกับค่า false เราต้องนำเครื่องหมาย ! (not) มาวางข้างหน้าเพื่อเทียบในทางตรงกันข้าม ( $!false = true \Rightarrow !0 = 1$ ) เช่นในโค้ดต่อไปนี้ ด้านบนและล่างคือรูปแบบที่เทียบทะกัน จะเขียนแบบไหนก็ได้ (รายละเอียดเพิ่มเติมเกี่ยวกับเครื่องหมาย ! จะกล่าวถึงในหัวข้อต่อๆ ไป)

```
//รูปแบบที่ 1
bool b = ...;      //true/false
if (b == false) { //ถ้า b เป็นเท็จ
    puts("...");
```

```
}
```

```
//รูปแบบที่ 2
bool b = ...;      //true/false
if (!b) {          //ถ้า b ไม่เป็นจริง
    puts("...");
```

```
}
```

สำหรับแนวทางการใช้ if เพิ่มเติมในแบบอื่นๆ มีดังนี้

```
int a = 10;
int b = 20;
if (a <= b) {
    puts("a is less than or equals to b");
    //กรณี 10 <= 20 เป็นจริง
    //ดังนั้น คำสั่ง puts() จึงถูกดำเนินการ
}
```

```

int x = 100;
int y = 0;
float z;
if (y == 0) {
    puts("dividend must be non-zero");
    //กรณีนี้ y == 0 เป็นจริง
    //ดังนั้น คำสั่ง puts() จึงถูกดำเนินการ
}

if (y != 0) {
    z = (float)x / y;
    printf("%d / %d = %.2f", x, y, z);
}

```

**ตัวอย่าง 6-2** รับตัวเลขทางคีย์บอร์ดเข้ามา และตรวจสอบว่าเป็นเลขคู่หรือคี่ โดยถ้าหารด้วย 2 ลงตัวหรือมีเศษ (ค่าที่เหลือ) เป็น 0 แสดงว่าเป็นเลขคู่ มิฉะนั้น ถือเป็นเลขคี่

```

#include <stdio.h>

void main() {
    int n;

    printf("\nEnter integer number >>");
    scanf("%d", &n);

    //ถ้าหาร 2 ลงตัว (ค่าที่เหลือเป็น 0) แสดงว่าเป็นเลขคู่
    if (n % 2 == 0) {
        printf("\n%d is even\n", n);
    }

    //ถ้าหาร 2 ไม่ลงตัว (ค่าที่เหลือไม่เป็น 0) แสดงว่าเป็นเลขคี่
    if (n % 2 != 0) {
        printf("\n%d is odd\n", n);
    }
}

```

enter integer number >>108  
 108 is even

**ตัวอย่าง 6-3** เป็นการหาค่ามากที่สุดจาก 3 จำนวนที่รับเข้ามา โดยมีแนวทางดังนี้

1. รับจำนวนแรกเข้ามา และกำหนดค่าสูงสุดให้เท่ากับจำนวนแรกไว้ก่อน
2. รับจำนวนที่สองเข้ามา และใช้ if เปรียบเทียบว่า จำนวนที่สองมากกว่าค่าสูงสุดเดิม (จำนวนแรก) หรือไม่ ถ้ามากกว่าก็ให้จำนวนที่สองเป็นค่าสูงสุดแทน

3. รับจำนวนที่สามเข้ามา และใช้ if เปรียบเทียบซึ่งเดิม ถ้าจำนวนที่สามมากกว่าค่าสูงสุดเดิม ก็ให้จำนวนที่สามเป็นค่าสูงสุดแทน
4. เมื่อเปรียบเทียบจนครบแล้ว ก็จะได้ค่าสูงสุดของจำนวนทั้งหมด

```
#include <stdio.h>

void main() {
    int n1, n2, n3, max;
    printf("\nnumber #1 >>");
    scanf("%d", &n1);

    max = n1;           //ให้ค่าสูงสุดเท่ากับจำนวนแรกไว้ก่อน

    printf("number #2 >>");
    scanf("%d", &n2);
    if (n2 > max) {    //ถ้าจำนวนที่ 2 มากกว่าค่าสูงสุดก่อนนี้
        max = n2;       //ให้จำนวนที่ 2 เป็นค่าสูงสุดแทน
    }

    printf("number #3 >>");
    scanf("%d", &n3);
    if (n3 > max) {    //ถ้าจำนวนที่ 3 มากกว่าค่าสูงสุดก่อนนี้
        max = n3;       //ให้จำนวนที่ 3 เป็นค่าสูงสุดแทน
    }

    printf("\nmax value: %d \n", max);
}
```

*n1 = 10* → *max = n1 = 10*

*n2 = 15* → *n2 > max* → *max = n2 = 15*

*n3 = 5* → *n3 < max* → *max = 15*

number #1 >>-10  
number #2 >>0  
number #3 >>-99  
max value: 0

**ตัวอย่าง 6-4** ถ้าเปลี่ยนจากการหาค่ามากที่สุดในตัวอย่างที่แล้ว มาเป็นการหาค่าที่น้อยที่สุด ก็ใช้หลักการเดิม เพียงแต่เปลี่ยนจากการใช้เครื่องหมาย > มาเป็น < เท่านั้นเอง ดังนี้

```
#include <stdio.h>

void main() {
    float n1, n2, n3, min;
    printf("\nnumber #1 >>");
    scanf("%f", &n1);

    min = n1;           //ให้ค่าต่ำสุดเท่ากับจำนวนแรกไว้ก่อน

    printf("number #2 >>");
}
```

number #1 >>3.141  
number #2 >>.357  
number #3 >>1.01  
min value: 0.357

```

scanf("%f", &n2);
if (n2 < min) { //ถ้าจำนวนที่ 2 น้อยกว่าค่าสูงสุดก่อนนี้
    min = n2; //ให้จำนวนที่ 2 เป็นค่าต่ำสุดแทน
}

printf("number #3 >>");
scanf("%f", &n3);
if (n3 < min) { //ถ้าจำนวนที่ 3 น้อยกว่าค่าสูงสุดก่อนนี้
    min = n3; //ให้จำนวนที่ 3 เป็นค่าต่ำสุดแทน
}

printf("\nmin value: %g \n", min);
}

```

## การใช้คำสั่ง if-else

คำสั่งในบล็อก if จะถูกดำเนินการ เมื่อเงื่อนไขตรงตามที่ระบุเท่านั้น แต่หากเงื่อนไขเป็นอย่างอื่นที่นอกเหนือไปจากนี้ และถ้าเรามีลิํงที่ต้องทำในกรณีดังกล่าว ก็สามารถกำหนดได้ด้วยคำสั่ง else แล้วตามด้วยบล็อกคำสั่งแบบเยื้องโดยเดียวกับ if ดังรูปแบบต่อไปนี้

```

if (เงื่อนไข) {
    คำสั่งต่างๆ ถ้าเงื่อนไขตรงกับที่กำหนด
} else {
    คำสั่งต่างๆ ถ้าเงื่อนไขไม่ตรงกับที่กำหนด
}

```

คำสั่ง else ไม่ต้องกำหนดเงื่อนไขใดๆ ให้กับมัน เพราะเป็นทางเลือกที่จะถูกดำเนินการเสมอหากเงื่อนไขของคำสั่ง if เป็นเท็จ (ไม่ตรงตามที่ระบุ) เช่น ในโค้ดต่อไปนี้

```

int n = ...;
//ถ้า n น้อยกว่าหรือเท่ากับ 0 คำสั่งในบล็อก if จะถูกดำเนินการ
//มิฉะนั้น (n ไม่น้อยกว่า 0) คำสั่งในบล็อก else จะถูกดำเนินการ
if (n <= 0) {
    puts("number must be greater than 0");
} else {
    puts("number is OK");
}

float x = 10;
int y = ...;
float z;

```

```
// หาก y ไม่เป็น 0 คำสั่งในบล็อก if ก็จะถูกดำเนินการ
// มิฉะนั้น (y เป็น 0) คำสั่งในบล็อก else ก็จะถูกดำเนินการ
if (y != 0) {
    z = x / y;
    printf("%g / %d = %g", x, y, z);
} else {
    puts("dividend can't be zero");
}
```

ภายในบล็อกของ else หากมีเพียง 1 คำสั่ง ไม่จำเป็นต้องเขียนคำสั่งไว้ในวงเล็บ {} ก็ได้ เหมือนกับ if ดังที่ได้กล่าวมาแล้ว เช่น

```
int x = ...;
if (x >= 0)
    puts(...);
else
    puts(...);
```

แต่อย่างไรก็ตาม การไม่เขียนคำสั่งไว้ในวงเล็บ {} ก่อให้เกิดข้อผิดพลาดได้ง่าย ดังนั้น แม้ จะมีเพียงคำสั่งเดียว ผู้เขียนแนะนำให้กำหนดบล็อกด้วยวงเล็บ {} เสมอ

**ตัวอย่าง 6-5** รับข้อมูลรหัสผ่านแล้วให้ใส่ครั้งที่ 2 เพื่อการยืนยันรหัส จากนั้นตรวจสอบว่าการใส่รหัสทั้ง 2 ครั้งตรงกันหรือไม่

```
#include <stdio.h>
#include <string.h>

void main() {
    char pw1[20], pw2[20];

    printf("\nEnter password >>");
    gets(pw1);

    printf("confirm password >>");
    gets(pw2);

    if (strcmp(pw1, pw2) == 0) {
        puts("\npassword OK");
    } else {
        puts("\npassword didn't match");
    }
}
```

```
enter password >>abcdef
confirm password >>AbCdef
password didn't match
```

**ตัวอย่าง 6-6** จากการตรวจสอบว่าเลขที่รับเข้ามาทางคีย์บอร์ดเป็นเลขคู่หรือเลขคี่ในตัวอย่างก่อนนี้ แทนที่เราจะใช้ if 2 ครั้ง ก็เปลี่ยนมาใช้ if-else แทนดังนี้ (ผลลัพธ์เหมือนเดิม)

```
#include <stdio.h>

void main() {
    int n;

    printf("\nEnter integer number >>");
    scanf("%d", &n);

    //ถ้าหาร 2 แล้วไม่มีค่าที่เหลือ (ค่าเป็น 0) แสดงว่าเป็นเลขคู่
    //มีฉะนั้น (ค่าที่เหลือไม่เป็น 0) แสดงว่าเป็นเลขคี่
    if (n % 2 == 0) {
        printf("\n%d is even\n", n);
    } else {
        printf("\n%d is odd\n", n);
    }
}
```

## การใช้คำสั่ง if-<else if>

ถ้าเงื่อนไขที่เราต้องการตรวจสอบสามารถแยกได้หลายกรณี แต่มีเพียงกรณีเดียวเท่านั้นที่จะถูกดำเนินการ เราอาจตรวจสอบแต่ละเงื่อนไขด้วยคำสั่ง else if ร่วมกับ if ดังรูปแบบต่อไปนี้

```
if (เงื่อนไขที่ 1) {
    คำสั่งต่างๆ ถ้าตรงกับเงื่อนไขที่ 1
} else if (เงื่อนไขที่ 2) {
    คำสั่งต่างๆ ถ้าตรงกับเงื่อนไขที่ 2
} else if (เงื่อนไขที่ 3) {
    คำสั่งต่างๆ ถ้าตรงกับเงื่อนไขที่ 3
}
...
} else if (เงื่อนไขที่ n) {
    คำสั่งต่างๆ ถ้าตรงกับเงื่อนไขที่ n
}
```

- เราต้องใช้ else if ร่วมกับ if เพื่อ และต้องวางไว้ตัดจากบล็อกของ if
- สิ่งที่จะเป็นเงื่อนไขของ else if ต้องมีค่าเป็น 1 (true) หรือ 0 (false) โดยใช้หลักการเดียวกับเงื่อนไขของ if ทุกประการ

- จะมีการตรวจสอบเงื่อนไขด้วย `else if` กี่ครั้งก็ได้ แต่จะมีเพียงกรณีเดียวเท่านั้นที่ถูกดำเนินการ

แนวทางการตรวจสอบเงื่อนไขในรูปแบบ `if-<else if>` มีลักษณะดังโคลัดต่างๆ ต่อไปนี้

```
int x = ...;
int y = ...;
if (x > y) {
    puts("x greater than y");
} else if (x < y) {
    puts("x less than y");
} else if (x == y) {
    puts("x equal to y");
}
```

```
char size = 'M';
char text[100];
if (size == 'S') {
    text = "Small";
} else if (size == 'M') {
    text = "Medium";
} else if (size == 'L') {
    text = "Large";
} else if (size == 'X') {
    text = "Extra Large";
}
```

ถ้าเงื่อนไขที่กำหนด `else if` อาจเป็นไปได้หลายกรณี เฉพาะเงื่อนไขแรกซึ่งตรงกับที่กำหนดเท่านั้น ที่จะถูกดำเนินการ ส่วนอันที่เหลือจะถูกข้ามไป ในลักษณะดังนี้

```
int age = 18;
char str[100];
if (age >= 60) {
    str = "senior";
} else if (age >= 20) {
    str = "adult";
} else if (age >= 12) {
    str = "teenager";
} else if (age >= 4) {
    str = "child";
} else if (age > 0) {
    str = "baby";
}

puts(str); //teenager
```

จากโค้ด เรากำหนด `age = 18` และตรวจสอบว่าตรงกับทั้งเงื่อนไข (`age >= 12`) รวมทั้ง (`age >= 2`) และ (`age > 0`) แต่เนื่องจากเราวางเงื่อนไข (`age >= 12`) เอาไว้ก่อนอันอื่น ดังนั้น เฉพาะบล็อกของเงื่อนไขอันนี้เท่านั้นที่ถูกดำเนินการ และหลังจากที่ทำการบล็อก `else if` อันนี้เสร็จ ก็จะข้ามการตรวจสอบในส่วน `if-<else if>` ทั้งหมดไปเลย ซึ่งตรงนี้เองที่เป็นข้อแตกต่างระหว่างการใช้คำสั่ง `if` หลายๆ ครั้ง กับการใช้ `if-<else if>` ดังการเปรียบเทียบในตัวอย่างดังไป

**ตัวอย่าง 6-7** เปรียบเทียบระหว่างการใช้ `if-<else if>` และการใช้ `if` หลายครั้ง โดยให้ดูโค้ดจากภาพต่อไปนี้

```
#include <stdio.h>
#include <string.h>

void main() {

    char login[] = "user1";
    char pswd[] = "";

    if (strcmp(login, "admin") != 0) {
        puts("login incorrect");
    } else if (strcmp(pswd, "qwert") != "secret") {
        puts("password incorrect");
    } else if (strcmp(pswd, "") == 0) {
        puts("re-enter password");
    }

    puts("goodbye");
}
```

```
//result
/*
login incorrect
goodbye
*/
}

char login[] = "user1";
char pswd[] = "";

if (strcmp(login, "admin") != 0) {
    puts("login incorrect");
}

if (strcmp(pswd, "secret") != 0) {
    puts("password incorrect");
}

if (strcmp(pswd, "") == 0) {
    puts("please enter password");
}

puts("goodbye");

//result
/*
login incorrect
password incorrect
please enter password
goodbye
*/
}
```

จากภาพจะเห็นว่า การใช้ if หลาย ๆ ครั้ง ทุกเงื่อนไขจะถูกตรวจสอบทั้งหมดและหากเป็นจริงดำเนินการตามคำสั่งในบล็อกนั้น แต่การใช้ if-<else if> เมื่อเจอเงื่อนไขแรกที่เป็นจริงแล้ว ก็จะหยุดตรวจสอบเงื่อนไขถัดไป หรือข้ามบล็อกของ if-<else if> ในส่วนที่เหลือทั้งหมดนั้นเอง

**ตัวอย่าง 6-8** ให้รับตัวเลขทางคีย์บอร์ดเป็นจำนวนประดุจที่ทีมเจ้าบ้านและทีมเยือนทำได้แล้วใช้ if-<else if> ตรวจสอบว่าทีมใดเป็นผู้แพ้ ผู้ชนะ หรือ เสมอ

```
#include <stdio.h>

void main() {
    int h_goals, g_goals; //home, guest

    printf("\nhome goals >>");
    scanf("%d", &h_goals);

    printf("guest goals >>");
    scanf("%d", &g_goals);

    if (h_goals > g_goals) {
        puts("\nhome team is the winner");
    } else if (h_goals < g_goals) {
        puts("\nguest team is the winner");
    } else {
        puts("\nboth teams play a draw");
    }
}
```

home goals >>3  
 guest goals >>5  
  
 guest team is the winner

**ตัวอย่าง 6-9** ให้รับข้อมูลทางคีย์บอร์ดเป็นคะแนนระหว่าง 0 - 100 จากนั้น พิจารณาว่า คะแนนดังกล่าวควรจะได้เกรดอะไร โดยเกณฑ์การให้เกรดเป็นดังโอดต่อไปนี้ (ถ้าคะแนนที่กำหนดตรงกับหลายเงื่อนไข ก็จะเลือกดำเนินการตามเงื่อนไขที่มาก่อนเพียงอันเดียว ดังหลักการที่ได้กล่าวมาแล้ว)

```
#include <stdio.h>

void main() {
    int score;
    char grade;

    printf("\nEnter score (0 - 100) >>");
    scanf("%d", &score);

    if (score >= 90) grade = 'A';
    else if (score >= 80) grade = 'B';
    else if (score >= 70) grade = 'C';
    else if (score >= 60) grade = 'D';
    else grade = 'F';
}


```

enter score (0 - 100) >>69  
  
 grade: C

```

if (score >= 80){
    grade = 'A';
} else if (score >= 70){
    grade = 'B';
} else if (score >= 60){
    grade = 'C';
} else if (score >= 50){
    grade = 'D';
} else {
    grade = 'F';
}

printf("\ngrade: %c \n", grade);

}

```

## การใช้คำสั่ง if-<else if>-else

ตามปกตินั้น เงื่อนไขที่กำหนดให้แก่ if-<else if> มักจะเป็นประเด็นหลัก ๆ ที่เราต้องการตรวจสอบ แต่หากมีกรณีอื่น ๆ ที่อยู่นอกเหนือไปจากนี้ และต้องถูกดำเนินการเมื่อเงื่อนไขต่าง ๆ ที่กำหนดให้แก่ if-<else if> ไม่มีอันได้ตรงตามที่ระบุ ก็ให้นำ else มาต่อท้าย if-<else if> ดังนี้

```

if (เงื่อนไขที่ 1) {
    คำสั่งต่างๆ ถ้าตรงกับเงื่อนไขที่ 1
} else if (เงื่อนไขที่ 2) {
    คำสั่งต่างๆ ถ้าตรงกับเงื่อนไขที่ 2
}
...
} else {
    คำสั่งต่างๆ ถ้าไม่ตรงกับเงื่อนไขใดๆ เลย
}

```

คำสั่ง else จะมีความจำเป็นก็ต่อเมื่อ เงื่อนไขต่าง ๆ ที่กำหนดแก่ if-<else if> ยังไม่ครอบคลุมกรณีที่เป็นไปได้ทั้งหมด เช่นในโค้ดต่อไปนี้

```

int a = ...;
if (a < 0) {
    puts("Negative");
} else if (a > 0) {
    puts("Positive");
} else { //a = 0
}

```

```

        puts("Neutral");
    }

char ch = '...';
if (ch == 'a') {
    puts("ant");
} else if (ch == 'b') {
    puts("boy");
} else if (ch == 'c') {
    puts("cat");
} else if (ch == 'd') {
    puts("dog");
} else {
    puts("sorry, character out of range");
}

```

**ตัวอย่าง 6-10** สมมติว่ามียอดเงินคงเหลือในบัญชี 30,000 บาท ให้รับค่าทางคีย์บอร์ดเป็นจำนวนเงินที่ต้องการถอนจากตู้ ATM ซึ่งมีเงื่อนไขที่สำคัญ เช่น ต้องไม่เกินยอดเงินคงเหลือ ต้องไม่เกิน 20,000 บาท และต้องเป็นจำนวนเต็มร้อย (หารด้วย 100 ลงตัว) โดยใช้คำสั่ง if-<else if>-else ตรวจสอบเงื่อนไขเหล่านี้ (สำหรับการจัดรูปแบบตัวเลขโดยใช้ , คั่นหลักพัน ได้กalgoไว้แล้วในบทที่ 3)

```

balance: 30,000
amount to withdraw >>1500

balance: 28,500

balance: 30,000
amount to withdraw >>35000
insufficient balance

balance: 30,000

```

```

#define _POSIX_C_SOURCE 200809L
#include <locale.h>
#include <stdio.h>

void main() {
    setlocale(LC_ALL, "");

    int withdraw, balance = 30000;

    printf("\nbalance: %'d", balance);

    //จำนวนเงินที่ต้องการถอน
    printf("\namount to withdraw >>");
    scanf("%d", &withdraw);

    //ถ้ายอดเงินคงเหลือไม่พอ
    if (balance < withdraw) {
        puts("insufficient balance");
    }
    //ถ้าจำนวนที่จะถอนเกิน 20,000

```

```

else if (withdraw > 20000) {
    puts("can't withdraw more than 20000");
}
//ถ้าไม่ใช่จำนวนเดิมร้อย (หาร 100 แล้วมีเศษ)
else if (withdraw % 100 != 0) {
    printf("amount to withdraw must be ");
    printf("the multiples of 100");
}
//ถ้าไม่ตรงตามเงื่อนไขทั้งหมด แสดงว่าถอนได้
else {
    balance -= withdraw;
}

printf("\nbalance: %'d \n", balance);
}

```

## การเปรียบเทียบทางตรรกะ

การเปรียบเทียบททางตรรกะ (Logical Operator) เป็นการเปรียบเทียบระหว่างข้อมูล 2 อาย่างที่จะต้องเป็นชนิดบูลีน นั่นคือ true (1) หรือ false (0) อาย่างโดยอ้างหนึ่ง และผลลัพธ์ที่ได้ ก็จะเป็น true หรือ false เช่นเดียวกัน โดยใช้เครื่องหมาย หรือตัวดำเนินการดังต่อไปนี้

ตัวดำเนินการ	ชื่อเรียก	การใช้งาน
&&	Logical AND (และ)	a && b
	Logical OR (หรือ)	a    b
!	Logical NOT (ไม่ใช่)	!a

- **&&** มีหลักการดังนี้

- ◎ ถ้าเป็นจริงทั้ง 2 คู่ (true && true) ผลลัพธ์จะเป็นจริง (true หรือ 1)
- ◎ กรณีนอกเหนือจากนี้ทั้งหมด ผลลัพธ์จะเป็นเท็จ (false หรือ 0)

```

bool a = true, b = true, c = false, d = false, r;

r = a && b;           //true && true = true
r = b && c;           //true && false = false
r = c && d;           //false && false = false
r = (1 == 1) && true; //true && true = true
r = (1 <= 1) && (1 > 2); //true && false = false
r = (1 != 1) && (1 == 2)); //false && false = false

```

- **|| มีหลักการดังนี้**

- ◎ ถ้าเป็นเท็จทั้ง 2 ค่า (false || false) ผลลัพธ์จะเป็นเท็จ (false)
- ◎ กรณีออกหนีออกจากทั้งหมด ผลลัพธ์จะเป็นจริง (true)

```
bool a = true; b = true; c = false; d = false, r;

r = a || b;           //true || true = true
r = b || c;           //true || false = true
r = c || e;           //false || false = false
r = (1 == 1) || true; //true || true = true
r = (1 <= 1) || (1 > 2); //true || false = true
r = (1 != 1) || (1 == 2); //false || false = false
```

- **!** จะเปลี่ยนค่าบูลินให้เป็นตรงกันข้าม นั่นคือ เปลี่ยนจาก true เป็น false และจาก false เป็น true หรือเปลี่ยนจาก 1 เป็น 0 และจาก 0 เป็น 1 เช่น

```
bool a = !true;          //false
bool b = !false;         //true
bool c = !0;              //c = !false = true

bool d = 1;               //true
d = !d;                  //d = false

bool e = !(true || false); //!(true) = false

bool e = !(1 == 1) && !(1 != 2);
//!(true) && !(true) = false && false = false
```

ให้ดูแนวทางการนำไปใช้งานจริงในหัวข้อดังไป

## การกำหนดหลายเงื่อนไขด้วย Logical Operator

ในบางกรณี การที่เราจะกระทำลิ่งใดหรือไม่นั้น อาจมีหลายเงื่อนไขที่ต้องพิจารณาร่วมกัน หรือเมื่อกรณีที่เงื่อนไขอย่างหนึ่งจะขึ้นอยู่กับเงื่อนไขอื่นๆ ด้วยนั้นเอง ซึ่งในลักษณะเช่นนี้เราสามารถเทียบเงื่อนไขที่เกี่ยวข้องทั้งหมดด้วยดำเนินการด้านตรรกะ (&& หรือ ||) หรือ Logical Operator ทั้งนี้หากเราพิจารณาการใช้ร่วมกับ if ก็สามารถสรุปได้ดังนี้

- ใช้ **&&** ถ้าต้องการให้เงื่อนไข เป็นจริงทั้งหมด จึงจะทำการคำสั่ง
- ใช้ **||** ถ้าต้องการให้เงื่อนไข เป็นจริงเพียงอันใดอันหนึ่ง ก็ทำการคำสั่ง

และตามหลักการของภาษา C ที่ต้องเขียนเงื่อนไขไว้ในวงเล็บ ( ) ดังนั้น เราต้องครอบเงื่อนไขย่อยๆ รวมทั้งหมดไว้ในวงเล็บ ( ) เดียวกัน หรือว่างเล็บชั้นนอกสุด ในลักษณะดังนี้ (ให้ \_\_\_\_\_ แทนตัวดำเนินการแบบตรรกะ && หรือ ||)

```
if ((เงื่อนไข 1) _____ (เงื่อนไข 2) _____ (เงื่อนไข 3) _____ (เงื่อนไข n)) {
    คำสั่ง เมื่อเปรียบเทียบเงื่อนไขรวมทั้งหมดแล้วเป็นจริง
}
```

เช่น ลองพิจารณาจากโค้ดต่อไปนี้ (ไม่ขอนำขั้นตอนการประกาศชนิดตัวแปรมาแสดง)

```
//ถ้าตัวแปร age ต้องมีค่าระหว่าง 12 - 19 จึงจะทำตามคำสั่ง
if ((age >= 12) && (age <= 19)) {
    puts("You are teenager");
}

//ถ้าไม่ใส่ login หรือ password เพียงอันใดอันหนึ่ง ก็แสดงคำเตือน
if ((strcmp(login, "") == 0) || (strcmp(pw, "") == 0)) {
    puts("ท่านต้องใส่ข้อมูลให้ครบถ้วนสองช่อง");
}
```

นอกจากนี้ ยังสามารถตรวจสอบได้มากกว่า 2 เงื่อนไข และควรใช้วงเล็บในการแยกแต่ละกลุ่มให้ชัดเจน ว่าเงื่อนไขใดจะเปรียบเทียบกับเงื่อนไขใด เพื่อป้องกันข้อผิดพลาดได้ เช่น

```
//ถ้าเงื่อนไขคือ x ต้องมีค่าระหว่าง 1-100 และหารด้วย 5 ลงตัว
if ((x > 0) && (x < 100) && (x % 5 == 0)) {
    ...
}
```

อย่างไรก็ตาม การเขียนรวมเงื่อนไขที่ซับซ้อนกว่านี้ อาจก่อให้เกิดข้อผิดพลาดได้ง่าย ถ้าเรายังไม่มีความชำนาญในการเขียนโค้ดมากพอ ซึ่งเราอาจแยกเปรียบเทียบเงื่อนไขทีละส่วนโดยเก็บผลลัพธ์ไว้ในตัวแปร แล้วค่อยนำตัวแปรทั้งหมดมาเทียบกัน เช่น

```
char gender[] = ...; // "male"/"female"
bool single = ...; // true/false

//หากเป็นผู้ชายต้องแต่งงานแล้ว (ไม่โสด)
bool m = ((strcmp(gender, "male") == 0) && (!single));
```

```

//ถ้าเป็นผู้หญิงต้องยังโสด
bool f = (strcmp(gender, "female") == 0) && (single);

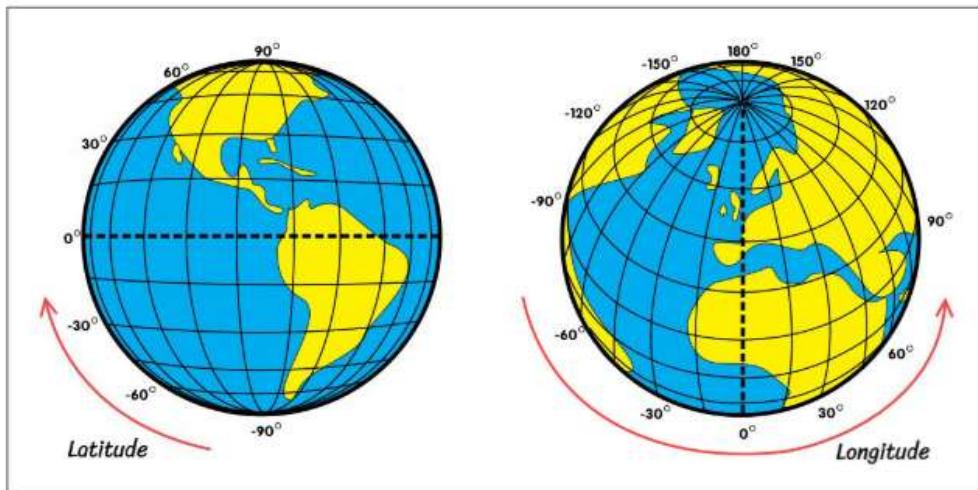
//เปรียบเทียบ 2 เงื่อนไข โดยต้องให้อันใดอันหนึ่งเป็นจริง
//ดังนั้น จึงเชื่อม 2 เงื่อนไขด้วย ||
if (m || f) {
    puts("You are welcome");
}

//ถ้าเรามีความชำนาญแล้ว อาจเขียนรวมทั้งหมดในขั้นตอนเดียว ดังนี้
/*
if (((strcmp(gender,"male") == 0) && (!single))
    ||
    ((strcmp(gender, "female") == 0) && (single)))
{
    puts("You are welcome");
}
*/

```

**ตัวอย่าง 6-11** ในการระบุตำแหน่งบนพื้นโลก เราต้องกำหนดค่า 2 อย่างคือ

- ละติจูด (Latitude) ต้องมีค่าระหว่าง -90 ถึง 90
- ลองจิจูด (Longitude) ต้องมีค่าระหว่าง -180 ถึง 180



ให้รับค่าทั้งสองอย่างทางคีย์บอร์ด และตรวจสอบโดยใช้ if ร่วมกับตัวดำเนินการทางตรรกะว่าอยู่ในช่วงที่เป็นไปได้หรือไม่

```
#include <stdio.h>

void main() {
    float lat, lon;

    printf("\nenter latitude >>");
    scanf("%f", &lat);
    if ((lat >= -90) && (lat <= 90)) {
        puts("latitude ok");
    } else {
        puts("latitude incorrect!");
    }

    printf("\nenter longitude >>");
    scanf("%f", &lon);
    if ((lon >= -180) && (lon <= 180)) {
        puts("longitude ok");
    } else {
        puts("longitude incorrect!");
    }
}
```

```
enter latitude >>100.1234
latitude incorrect!

enter longitude >>150.6789
longitude ok
```

**ตัวอย่าง 6-12** เราสามารถจำแนกบุคคลเป็น Generation ต่างๆ ได้ตามปี พ.ศ. ที่เกิดดังนี้ (แหล่งข้อมูลต่างๆ อาจกำหนดช่วงปี พ.ศ. ที่แตกต่างกันบ้างเล็กน้อย)

ปี พ.ศ. ที่เกิด	Generation
2443 และต่ำกว่า	Lost Generation
2444 - 2467	Greatest Generation
2468 - 2488	Silent Generation
2489 - 2507	Baby Boomer Generation
2508 - 2519	Generation X
2520 - 2538	Generation Y
2539 - 2555	Generation Z

ให้วันข้อมูลเป็นปี พ.ศ. ที่เกิด แล้วใช้ Logical Operator ร่วมกับ if-<else if>-else ในการตรวจสอบว่าปีดังกล่าวตรงกับ Generation ใด

```
#include <stdio.h>
#include <string.h>

void main() {
    int y;
    char gen[50];

    printf("\nyear born (B.E.) >>");
    scanf("%d", &y);

    if (y <= 2443) {
        strcpy(gen, "lost generation");
    }
    else if ((y >= 2444) && (y <= 2467)) {
        strcpy(gen, "greatest generation");
    }
    else if ((y >= 2468) && (y <= 2488)) {
        strcpy(gen, "silent generation");
    }
    else if ((y >= 2489) && (y <= 2507)) {
        strcpy(gen, "baby boomer generation");
    }
    else if ((y >= 2508) && (y <= 2519)) {
        strcpy(gen, "generation X");
    }
    else if ((y >= 2520) && (y <= 2538)){
        strcpy(gen, "generation Y");
    }
    else if ((y >= 2539) && (y <= 2555)) {
        strcpy(gen, "generation Z");
    }
    else {
        strcpy(gen, "unknown generation");
    }

    printf("you are %s \n", gen);
}
```

year born (B.E.) >>2525  
you are generation Y

## ตัวดำเนินการแบบ Ternary

ตัวดำเนินการแบบ Ternary เป็นการเปรียบเทียบเงื่อนไขในอีกลักษณะหนึ่ง ซึ่งมีหลักการคล้ายกับ if-else แต่เขียนได้กระชับกว่า โดยมีรูปแบบดังนี้

(เงื่อนไข) ? กรณีเงื่อนไขเป็นจริง : กรณีเงื่อนไขเป็นเท็จ

- เงื่อนไข ก็คล้ายกับที่เรากำหนดให้แก่ if หรือ else if โดยต้องให้ผลลัพธ์ออกมาเป็นค่าบูลีน (จริง/เท็จ) เท่านั้น
- กรณีเงื่อนไขเป็นจริง อาจเป็นการส่งค่าบางอย่างหรือค่าจากตัวแปรกลับออกไป โดยการระบุค่าหรือชื่อตัวแปรที่เก็บข้อมูลนั้น หรือเป็นคำสั่งให้ทำการบางอย่าง เช่น เรียกฟังก์ชันขึ้นมาทำงาน หรือการคำนวณ เป็นต้น ซึ่งส่วนนี้จะถูกดำเนินการเฉพาะกรณีที่เงื่อนไขเป็นจริงเท่านั้น
- กรณีเงื่อนไขเป็นเท็จ ก็เช่นเดียวกับกรณีเงื่อนไขเป็นจริง แต่ส่วนนี้จะถูกดำเนินการเฉพาะกรณีที่เงื่อนไขเป็นเท็จเท่านั้น

แนวทางการใช้ตัวดำเนินการแบบ Ternary เช่น

```
int a = 1, b = -10;
int max = (a > b) ? a : b;
/*
โดยนิ่มหมายความว่า ถ้า a มากกว่า b
ให้คืนค่าจากตัวแปร a กลับไป มิฉะนั้น ให้คืนค่าจากตัวแปร b
หรือเขียนในแบบ if-else คือ
if (a > b) {
    max = a;
} else {
    max = b;
}
*/
```

```
int total = ...;
int carriage = (total >= 1000) ? 0 : 50;
/*
โดยนิ่มหมายความว่า ถ้ายอดรวม ตั้งแต่ 1000 บาทขึ้นไป
ให้ยกเว้นค่าจัดส่ง (0 บาท) มิฉะนั้น ให้คิดค่าจัดส่ง 50 บาท
*/
*/
```

อย่างไรก็ตาม ในกรณีที่ตัวดำเนินการแบบ Ternary คืนค่าเป็นข้อมูลชนิดสตริง เราไม่สามารถนำค่ามากำหนดให้แก่ตัวแปรโดยตรงได้ เช่น โค้ดต่อไปนี้จะเกิดข้อผิดพลาด

```
int num = ...;
char s[] = (num % 2 == 0) ? "even" : "odd";           //Error
```

วัตถุประสงค์ของโค้ดข้างบนนี้คือ ถ้าจำนวนนับหารด้วย 2 ลงตัว (เศษหรือค่าที่เหลือเป็น 0) ให้คืนค่าแบบสตริงเป็นคำว่า "even" มิฉะนั้นให้คืนค่าเป็นคำว่า "odd" แต่ในภาษา C เราทำแบบนี้ไม่ได้ ทั้งนี้ หากเราต้องการคืนค่าเป็นสตริงแล้วนำไปกำหนดให้แก่ตัวแปร อาจเปลี่ยนไปใช้ฟังก์ชัน strcpy() หรืออื่นๆ แทน เช่น

```
int num = ...;
char s[20];

//ให้กำหนดค่าไว้ในส่วนจริง/เท็จ ของ Ternary (ไม่ต้องส่งค่ากลับออกไป)
(num % 2 == 0) ? strcpy(s, "even") : strcpy(s, "odd");
```

## การใช้คำสั่ง switch-case

คำสั่ง switch-case เป็นการแยกวิธีดำเนินการออกเป็นกรณีต่างๆ โดยนำข้อมูลมาตรวจสอบ หากตรงกับกรณีใด ลิستที่กำหนดเอาไว้สำหรับกรณีนั้นก็จะถูกดำเนินการ ซึ่งรูปแบบทั่วไปคือ

```
switch (ลิสท์ที่ต้องการตรวจสอบ) {
    case กรณีที่ 1:
        คำสั่ง
        break;
    case กรณีที่ 2:
        คำสั่ง
        break;
    ...
    case กรณีที่ n:
        คำสั่ง
        break;
    default:
        คำสั่ง ถ้าไม่ตรงกับกรณีใดเลยๆ (เทียบได้กับ else)
}
```

ลักษณะของ switch-case จะคล้ายกับ if-<else if>-else ซึ่งมีแนวทางการใช้งาน ดังนี้

```
int num = 1;
switch (num) {
    case 1:
        puts("positive");
        break;
    case 0:
        puts("neutral");
        break;
    case -1:
        puts("negative");
        break;
    default:
        puts("unknown");
}
```

จากโค้ดหมายความว่า กรณีที่ตัวแปร num เป็น 1 จะแสดงข้อความ "positive" กรณีที่เป็น 0 จะแสดงคำว่า "neutral" กรณีที่เป็นเลข -1 จะแสดงคำว่า "negative" แต่หากไม่ตรงกับกรณีใด (ไม่ใช่เลข 1, 0, -1) จะแสดงคำว่า "unknown" เป็นต้น โดยลักษณะเกี่ยวกับ switch-case ที่ควรรู้เพิ่มเติมคือ

- ตัวแปรหรือข้อมูลที่จะตรวจสอบด้วยคำสั่ง switch ต้องเป็นชนิดเลขจำนวนเต็ม หรือ อักขระ (char) หรือบูลิน (เที่ยบเท่ากับ 1 และ 0) เท่านั้น แต่ จะเป็นชนิดเลขทศนิยม หรือสตริงไม่ได้
- จะมีกรณีที่กำหนดด้วย **default** หรือไม่ก็ได้ ขึ้นกับว่าเราจะกำหนด case ให้ครอบคลุม กรณีที่จะเป็นไปได้ทั้งหมดหรือไม่ ซึ่งก็เช่นเดียวกับ else ที่ใช้ร่วมกับ if-<else if> ที่จะมี else ต่อท้ายหรือไม่ก็ได้
- การกำหนดคำสั่ง **break** ต่อท้ายในกรณีต่าง ๆ ก็เพื่อหยุดการทำงานของคำสั่ง switch ไว้แค่ครั้นนั้น เมื่อเจอกรณีที่ต้องการแล้ว และออกจากบล็อก switch ไป ทั้งนี้หากไม่มีคำสั่ง break กำหนดเอาไว้ คำสั่งทั้งหมดในส่วนที่เหลือก็ยังถูกตรวจสอบและดำเนินการทั้งหมด ซึ่งผิดกับหลักการของคำสั่ง switch เช่น ให้ลองดูโค้ดและผลลัพธ์ต่อไปนี้

```
char size = 'M';
switch (size) {
    case 'S': puts("Small");
    case 'M': puts("Medium");
    case 'L': puts("Large");
```

```

    default: puts("Unknown");
}
/* ลักษณะผลลัพธ์
Medium
Large
Unknown
*/

```

จากผลลัพธ์จะพบว่า คำสั่งตั้งแต่ case: 'M' เป็นต้นไป จะถูกดำเนินการทั้งหมด แม้ว่า เงื่อนไขของกรณีเหล่านี้ไม่ตรงกับค่า 'M' ก็ตาม ลักษณะเช่นนี้จึงขัดกับหลักการของ คำสั่ง switch-case ที่ควรจะเลือกทำเพียงกรณีใดกรณีหนึ่งเท่านั้น และจากเหตุผลที่ กล่าวมา จึงจำเป็นต้องมีคำสั่ง break ต่อท้ายในทุก case เสมอ

- ค่าที่เราจะกำหนดเป็นเงื่อนไขของแต่ละ case นั้น จะเป็นสิ่งต่อไปนี้ไม่ได้คือ
  - ◎ จะเป็นข้อมูลชนิด float หรือ double หรือค่า NULL ไม่ได้
  - ◎ จะนำค่าจากตัวแปรมาใช้โดยตรง หรือนำมาคำนวณแล้วกำหนดให้เป็นเงื่อนไขของ case ไม่ได้ ยกเว้นค่าคงที่ชนิดเลขจำนวนเต็มหรืออักขระ
  - ◎ ถ้าเป็นเลขหรืออักขระที่กำหนดค่าโดยตรงที่แต่ละ case สามารถนำมาคำนวณ ด้วยเครื่องหมายต่างๆ แล้วกำหนดให้เป็นเงื่อนไขของ case ได้ แต่ต้องเป็นเลข จำนวนเต็ม หรือคำนวณแล้วได้เลขจำนวนเต็ม

```

#include <stdio.h>

void main() {
    int a = 10;
    int b = 20;
    float c = 10.5;
    const int D = 30;
    char e = 'E';
    float f;

    switch (a) {
        case a + b: //Error เพราะใช้ค่าจากตัวแปร
            f = a + b; break;

        case 10 + 20: //OK แบบนี้ทำได้ เพราะกำหนดค่าโดยตรง
            f = 10 + 20; break;

        case 0.0 + 1: //Error เพราะเป็นเลขทศนิยม
            f = 0.0; break;
    }
}

```

```

        case c:          //Error เพราะใช้ค่าจากตัวแปร
            f = c * 2; break;

        case D * 2:    //OK แบบนี้ทำได้ เพราะ D เป็นค่าคงที่
            f = c * 2; break;

        case e:          //Error เพราะใช้ค่าจากตัวแปร
            break;

        case 'z':       //OK แบบนี้ทำได้ เพราะกำหนดค่าโดยตรง
            break;
    }
}

```

- หากมีหลาย case ที่ต้องทำคำสั่งในแบบเดียวกัน เราไม่จำเป็นต้องเขียนคำสั่งเหล่านั้นซ้ำซ้อนกันก็ได้ แต่สามารถนำไปเขียนไว้ที่ case ตัวสุดท้ายที่ตรงกับเงื่อนไขได้เลย เช่น

```

int month = ...; //1 - 12

switch (month) {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
        puts("this month has 31 days"); break;
    case 4:
    case 6:
    case 9:
    case 11:
        puts("this month has 30 days"); break;
    case 2:
        puts("this month has 28-29 days"); break;
    default:
        puts("number out of range")
}

```

โค้ดข้างบนหมายความว่า หาก month มีค่าอย่างใดอย่างหนึ่งระหว่าง 1, 3, 5, 7, 8, 10, 12 จะแสดงคำว่า "this month has 31 days" แต่หากเป็นลข 4, 6, 9, 11 จะแสดงคำว่า "this month has 30 days" หรือหากเขียนโดยใช้คำสั่ง if-<else if>-else ร่วมกับเครื่องหมาย || จะได้ดังนี้

```

int m = ...;
if (m==1 || m==3 || m==5 || m==7 || m==8 || m==10 || m==12) {
    puts("this month has 31 days");
} else if (m==4 || m==6 || m==9 || m==11) {
    puts("this month has 30 days");
} else if (m==2) {
    puts("this month has 28 - 29 days");
} else {
    puts("number out of range");
}

```

**ตัวอย่าง 6-13** ให้วรับตัวเลข 2 จำนวนเข้ามาทางคีย์บอร์ด และวรับเครื่องหมายอย่างใดอย่างหนึ่งระหว่าง +, -, \*, / และ % จากนั้นใช้ switch-case เพื่อแยกวิธีการคำนวณในแต่ละกรณี (ตัวอย่างนี้ มีการรับข้อมูลชนิด char ด้วยฟังก์ชัน scanf() ซึ่งจะมีปัญหาเช่นเดียวกับการรับข้อมูลชนิดสตริง ดังนั้น เราจะแก้ปัญหาด้วยวิธีการเดียวกันตามที่เคยกล่าวไว้แล้วในบทที่ 4)

```

#include <stdio.h>

void main() {
    float n1, n2, r;
    char op;

    printf("\nEnter num #1 >>");
    scanf("%f", &n1);
    getchar();

    printf("Enter operator (+, -, *, /, %) >>");
    scanf("%c", &op);

    printf("Enter num #2 >>");
    scanf("%f", &n2);

    switch (op) {
        case '+':
            r = n1 + n2; break;
        case '-':
            r = n1 - n2; break;
        case '*':
            r = n1 * n2; break;
        case '/':
            r = n1 / n2; break;
        case '%':
            r = (int)n1 % (int)n2; break;
    }
}

enter num #1 >>15
enter operator (+, -, *, /, %) >>/ 
enter num #2 >>3
15 / 3 = 5

```

```

    }
    printf("\n%g %c %g = %g \n", n1, op, n2, r);
}

```

## การตรวจสอบอักษรด้วยฟังก์ชันที่คืนค่าแบบบูลีน

ในภาษา C มีฟังก์ชันจำนวนมากที่คืนค่าหรือได้ผลลัพธ์เป็นค่าแบบบูลีน (true/false) โดยเฉพาะอย่างยิ่งฟังก์ชันประเภทการตรวจสอบอักษร ซึ่งฟังก์ชันบางส่วนเรายังมีโอกาสได้นำมาใช้บ่อยๆ ในบทต่อๆ ไป ดังนี้

<b>isdigit()</b>	ตรวจสอบว่าอักษรที่ระบุเป็นเลข 0 - 9 หรือไม่ ถ้าใช่จะได้ค่าเป็น 1 (true) มิฉะนั้นจะคืนค่า 0 (false)
<b>isalpha()</b>	ตรวจสอบว่าค่าที่ระบุเป็นตัวอักษร a - z หรือไม่ ถ้าเป็นตัวพิมพ์ใหญ่ (A - Z) จะคืนค่า 1 ถ้าเป็นตัวพิมพ์เล็ก (a - z) จะคืนค่า 2 กรณีอื่นๆ จะได้ค่าเป็น 0
<b>isalnum()</b>	ตรวจสอบว่าค่าที่ระบุเป็นตัวอักษร a - z หรือเลข 0 - 9 หรือไม่ ถ้าเป็นตัวพิมพ์ใหญ่ (A - Z) จะคืนค่า 1 ถ้าเป็นตัวพิมพ์เล็ก (a - z) จะคืนค่า 2 ถ้าเป็นตัวพิมพ์เลข (0 - 9) จะคืนค่า 4 กรณีอื่นๆ จะได้ค่าเป็น 0
<b>isupper()</b>	ตรวจสอบว่าค่าที่ระบุเป็นตัวอักษรพิมพ์ใหญ่หรือไม่ ถ้าเป็นตัวพิมพ์ใหญ่ (A - Z) จะคืนค่า 1 กรณีอื่นๆ จะได้ค่าเป็น 0
<b>islower()</b>	ตรวจสอบว่าค่าที่ระบุเป็นตัวอักษรพิมพ์ใหญ่หรือไม่ ถ้าเป็นตัวพิมพ์เล็ก (a - z) จะคืนค่า 1 กรณีอื่นๆ จะได้ค่าเป็น 0
<b>tolower(), toupper()</b>	แปลงอักษรเป็นตัวพิมพ์เล็กและพิมพ์ใหญ่ ตามลำดับ โดยฟังก์ชันเหล่านี้ จะคืนค่าเป็นอักษรหลังการแปลง เช่น char C = toupper('c');

ฟังก์ชันทั้งหมดที่แสดงในตารางนี้ อยู่ในไฟล์ **ctype.h** ดังนั้น เราต้องอ้างอิงด้วย **include** จึงจะใช้งานได้ และค่าที่จะกำหนดให้กับแต่ละฟังก์ชันต้องเป็นชนิด **char** หรือเป็นชนิด **int** ที่เป็นรหัสของอักษรที่ต้องการตรวจสอบ และเนื่องจากค่าที่คืนกลับมาเป็นชนิดตัวเลข ซึ่งเราสามารถเทียบเป็นค่าบูลีนได้นั่นคือ **false = 0, true > 0** ถ้าเราต้องการตรวจสอบแค่ค่าจริงเท็จก็สามารถใช้ร่วมกับคำสั่ง **if** ได้ ซึ่งขอให้ดูแนวทางการใช้งานจากตัวอย่างต่อไปนี้

**ตัวอย่าง 6-14** แนวทางการใช้ฟังก์ชัน isxxx() เพื่อตรวจสอบข้อมูลอักขระ

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    putchar('\n');
    char s[20];

    strcpy(s, (isdigit('1')) ? "" : "not ");
    printf("1 is %sdigit\n", s);

    strcpy(s, (isdigit('x')) ? "" : "not ");
    printf("x is %sdigit\n", s);

    strcpy(s, (isalpha('a')) ? "" : "not ");
    printf("a is %salpha\n", s);

    strcpy(s, (isalpha('A')) ? "" : "not ");
    printf("A is %salpha\n", s);

    strcpy(s, (isupper('a')) ? "" : "not ");
    printf("a is %supper\n", s);

    strcpy(s, (isupper('A')) ? "" : "not ");
    printf("A is %supper\n", s);

    char d;
    printf("\nEnter digit (0-9) >>");
    scanf("%c", &d);
    if (!isdigit(d)) {
        printf("error! please enter 0-9\n");
    } else {
        printf("thank! for entering digit\n");
    }
}
```

<pre>1 is digit x is not digit a is alpha A is alpha a is not upper A is upper</pre>	<pre>enter digit (0-9) &gt;&gt;x error! please enter 0-9</pre>
--	--

สิ่งสำคัญที่เราได้ศึกษาในหนึ่งก็คือ การใช้เครื่องหมายเปรียบเทียบรูปแบบต่างๆ ในกลุ่ม Comparison และ Logic ซึ่งให้ผลลัพธ์แบบบูลีน นอกจากนี้ยังได้กล่าวถึงการทำหนดเงื่อนไข ด้วยคำสั่ง if, else-if, else และ switch เพื่อตัดสินใจก่อนจะดำเนินการในขั้นตอนต่อไป ซึ่งถือเป็นพื้นฐานที่สำคัญอย่างหนึ่งของการเขียนโปรแกรม เพื่อควบคุมการทำงานต่างๆ ภายใต้เงื่อนไขที่เราต้องการ

# 7

## การทำซ้ำแบบวนรอบ

การทำซ้ำแบบวนรอบ หรือการวนลูป (Loop) เป็นการทำงานในลักษณะเดิมตั้งแต่เริ่มต้นไปจนถึงสุดแล้วกลับมาเริ่มทำใหม่ เช่นนี้ไปเรื่อยๆ จนกว่าจะครบจำนวนรอบหรือตรงกับเงื่อนไขที่กำหนดเอาไว้ โดยในภาษา C นั้นมีรูปแบบของการวนลูปให้เลือกใช้หลายอย่าง เช่น for, while หรือ do-while เป็นต้น ซึ่งเนื้อหาในบทนี้ก็จะกล่าวถึงหลักการของลูปแต่ละแบบ และแนวทางการเลือกลูปที่เหมาะสมกับงานแต่ละอย่างอีกด้วย

### การใช้ลูปแบบ for ขั้นพื้นฐาน

กรณีที่เราจะทำคำสั่งในลักษณะเดิมแบบซ้ำๆ ก็ต้องเขียนโค้ดที่คล้ายๆ กันหลายครั้ง เช่น การพิมพ์เลข 1 - 5 ออกໄປที่ส่วนแสดงผล

```
printf("\n%d", 1);
printf("\n%d", 2);
printf("\n%d", 3);
printf("\n%d", 4);
printf("\n%d", 5);
```

ลองคิดดูว่า หากจำนวนตัวเลขที่เราต้องการพิมพ์มากกว่านี้ จะเกิดความยุ่งยากเพียงใด หรือบางที่ก็ต้องนำเงื่อนไขอย่างอื่นมาพิจารณาด้วย เราอาจไม่ทราบล่วงหน้าว่าต้องทำซ้ำกี่ครั้ง ดังนั้น ในภาษาคอมพิวเตอร์แทนทั้งหมดรวมถึงภาษา C จึงมีคำสั่งสำหรับการทำงานแบบวงรอบ ตามจำนวนครั้งที่กำหนด หรือเรียกว่าการวนลูป (Loop) นั่นเอง โดยในหัวข้อนี้เราจะเริ่มศึกษาจากการใช้คำสั่ง for ขั้นพื้นฐานไปก่อน ซึ่งมีรูปแบบอย่างง่ายดังนี้

```
for (ตัวแปร = ค่าเริ่มต้น; เงื่อนไขการวนลูป; การเปลี่ยนค่าตัวแปร) {
    คำสั่งต่างๆ
}
```

- ข้อกำหนดทั้ง 3 ส่วนคือ ตัวแปร เงื่อนไข และการเปลี่ยนค่า ต้องอยู่ภายในวงเล็บ ( ) เสมอ โดยคั่นแต่ละส่วนด้วยเครื่องหมาย semicolon (:)
- ตัวแปร หรือตัวนับ ก็คือค่าที่จะถูกนำไปใช้มีการวนลูปในแต่ละรอบ โดยต้องระบุ ค่าเริ่มต้นให้กับตัวแปรเสมอ เพื่อกำหนดขอบเขตเริ่มแรกในการทำงานของลูป
- เงื่อนไขการวนลูป เป็นการกำหนดขอบเขตการทำงานหรือการวนรอบของลูป ซึ่งตาม ปกติแล้ว เงื่อนไขดังกล่าวเนี้ดองเป็นการเปรียบเทียบที่ให้ผลลัพธ์ออกมาเป็นบูลิน (true/false) โดยการวนลูปจะเกิดขึ้นเมื่อเงื่อนไขเป็นจริง แต่หากเงื่อนไขเป็นเท็จ ก็ จะลิ้นสุดการวนลูป
- การเปลี่ยนค่าของตัวแปร เราต้องเลือกเครื่องหมายอย่างใดอย่างหนึ่งมาเปลี่ยนค่า ของตัวแปรในการวนลูปแต่ละรอบ เช่น เพิ่มค่าทีละ 1 หรือเพิ่มค่าทีละ 5 หรือลดค่า ทีละ 2 เป็นต้น
- เราต้องค้นระหว่างการกำหนดค่าของตัวแปร เงื่อนไข และการเปลี่ยนค่า ด้วย เครื่องหมาย semicolon(:) เสมอ

เช่น จากการพิมพ์เลข 1 - 5 ออกໄປที่ส่วนแสดงผลดังโค้ดที่ผ่านมา หากเปลี่ยนมาเขียน เป็นลูปแบบ for จะได้ดังนี้

```
for (int i = 1; i <= 5; i++) {
    printf("\n%d", i);
}
```

จากโค้ดที่ผ่านมา ลักษณะการวนลูปคือ

- เราพิจารณาคร่าวๆ ได้ว่า ลูปเริ่มจาก  $i = 1$  จนกว่าเงื่อนไข  $i \leq 5$  จะเป็นเท็จ หรือ  $i > 5$  จึงจะลิ้นสุดการวนลูป และ  $i++$  แสดงว่าเพิ่มค่าตัวนับครั้งละ 1
- ( $i = 1$ ) เช้าสู่การวนลูปแรก และตรวจสอบเงื่อนไข ( $i \leq 5$ ) ซึ่งเป็นจริง ก็ทำคำสั่ง ภายในบล็อกของ for จนครบทุกคำสั่ง (นับเป็น 1 ลูป)
- ( $i++ \Rightarrow 2$ ) เพิ่มค่าของตัวแปรไปอีก 1 (คือส่วนที่กำหนดด้วย  $i++$ ) และตรวจสอบ เงื่อนไข ( $i \leq 5$ ) ซึ่งเป็นจริง ก็ทำตามคำสั่งทั้งหมดในบล็อก for
- ( $i++ \Rightarrow 3$ ) เพิ่มตัวแปรไปอีก 1 จากนั้นตรวจสอบเงื่อนไข ( $i \leq 5$ ) ซึ่งเป็นจริง ก็ทำ คำสั่งในบล็อก for จนครบ
- ( $i++ \Rightarrow \dots$ ) เพิ่มค่าตัวแปร และตรวจสอบเงื่อนไข เช่นนี้ไปเรื่อยๆ จนกว่าเงื่อนไข ( $i \leq 5$ ) จะเป็นเท็จ นั่นคือค่าของตัวแปรมีค่ามากกว่า 5 จึงจะลิ้นสุดการวนลูป

ลองมาดูโค้ดเพิ่มเติมในลักษณะอื่นๆ เช่น สมมติว่าเราต้องการพิมพ์เฉพาะเลขคู่ระหว่าง 1 - 10 นั้นแสดงว่าเราต้องเริ่มจาก 2 แล้วเพิ่มค่าตัวแปรทีละ 2 ชึ่งเขียนเป็นโค้ดได้ดังนี้

```
for (int i = 2; i <= 10; i += 2) {
    printf("%d ", i);
}
//ผลลัพธ์คือเลข 2 4 6 8 10
```

สำหรับตัวแปรหรือตัวนับที่ใช้กับลูป for ไม่จำเป็นต้องใช้ชนิด int เท่านั้น แต่อาจเป็นข้อมูลชนิดตัวเลขอื่นๆ ก็ได้ เช่น short, char, long, double หรือ float เช่น กรณีต่างๆ ต่อไปนี้

```
for (double d = 0.5; d <= 6.5; d += 1.5) {
    printf("%0.1f ", d);
}
//ผลลัพธ์คือ 0.5 2.0 3.5 5.0 6.5

//พิมพ์อักษร A-Z
for (char ch = 'A'; ch <= 'Z'; ch++) {
    printf("%c ", ch);
}
//ผลลัพธ์คือ A B C D ... X Y Z
```

การเปลี่ยนแปลงค่าของตัวแปรนั้น อาจเป็นการลดค่าลงจากเดิมก็ได้ แต่ในกรณีนี้ เราต้องพิจารณาค่าเริ่มต้นของตัวแปรและเงื่อนไขให้สัมพันธ์กันด้วย เช่น โค้ดต่อไปนี้

```
for (int i = 20; i >= 1; i -= 2) {
    printf("%d ", i);
}
//ผลลัพธ์คือ 20 18 16 ... 6 4 2
```

จากโค้ดข้างบนหมายความว่า ค่าของตัวแปรจะเริ่มจาก 20 แล้วเงื่อนไขคือ ถ้าตัวแปรมีค่ามากกว่าหรือเท่ากับ 1 ก็จะวนลูปต่อไปเรื่อยๆ พร้อมกับลดค่าลงครั้งละ 2 จนกว่าเงื่อนไข ( $i \geq 1$ ) เป็นเท็จ หรือจนกว่า  $i < 1$  นั่นเอง จึงจะออกจากลูป

ในอีกลักษณะหนึ่งคือ การพิมพ์อักษรจาก f => a เช่น

```
for (char ch = 'f'; ch >= 'a'; ch--) {
    printf("%c ", ch);
}
//ผลลัพธ์คือ f e d c b a
```

ภายในลูป for นอกจากจะกำหนดเป็นคำสั่งทั่วไปแล้ว ก็อาจมีการตรวจสอบเงื่อนไขด้วยคำสั่ง if หรือ switch ซ้อนอยู่ข้างใน ซึ่งไม่ได้มีข้อกำหนดอะไรเป็นพิเศษ ขอให้ดูแนวทางจากโค้ดต่อไปนี้

```
//ต้องการตรวจสอบว่า ตั้งแต่ 1-100
//มีจำนวนที่ทั้ง 6 และ 8 หารลงตัวกี่จำนวน
int count = 0;
for (int i = 1; i <= 100; i++) {
    if (i % 6 == 0 && i % 8 == 0) {
        count++;
    }
}
printf("%d", count); //4
```

**ตัวอย่าง 7-1** เป็นการใช้ลูป for เพื่อวนรับข้อมูลตัวเลข 5 จำนวนจากผู้ใช้ผ่านทางคีย์บอร์ด พร้อมกับน้ำกเพิ่มเข้าไปในผลรวม แล้วแสดงผลลัพธ์และหาค่าเฉลี่ย

```
#include <stdio.h>

void main() {
    float num, sum = 0.0;

    putchar('\n');
    for (int i = 1; i <= 5; i++) {
        printf("number #%d >>", i);
        scanf("%f", &num);
        sum += num;
    }
    printf("\nsum: %g", sum);
    printf("\naverage: %g \n", + sum / 5);
}
```

```
number #1 >>30
number #2 >>10
number #3 >>20
number #4 >>10
number #5 >>50

sum: 120
average: 24
```

**ตัวอย่าง 7-2** การหาค่าของเลขยกกำลัง ที่ทำการนำเลขฐานมาคูณกันเป็นจำนวนครั้งเท่ากับเลขชี้กำลัง เช่น  $10^3 = 10 * 10 * 10$  โดยให้รับข้อมูลทางคีย์บอร์ดเป็นเลขฐานและเลขยกกำลังจากนั้นใช้ลูป for เพื่อคำนวนหาค่าของเลขยกกำลัง และจะเทียบกับการใช้ฟังก์ชัน pow() ของภาษา C ให้ดูด้วยว่าได้ค่าเท่ากันกันที่เราคิดเองหรือไม่

```
#include <stdio.h>
#include <stdbool.h>
#include <math.h>
```

```
base >>-5
power (+int) >>3
-5 ^ 3 = -125
using pow(-5, 3) = -125
```

```

void main() {
    float base, result = 1;
    int power;
    bool error = false;

    printf("\nbase >>");
    scanf("%f", &base);

    printf("power (+int) >>");
    scanf("%d", &power);

    if (power == 0) {
        result = 1;
    } else if (power >= 1) {
        for (int i = 1; i <= power; i++) {
            result *= base;
        }
    } else {
        result = 0;
        error = true;
    }

    if (!error) {
        printf("\n%g ^ %d = %g\n",
               base, power, result);
        printf(
            "\nusing pow(%g, %d) = %g\n",
            base, power, pow(base, power)
        );
    } else {
        puts("\nerror! invalid number\n");
    }
}

```

## การกำหนดลูป for ซ้อนกัน

การกำหนดลูป for ซ้อนกันก็คล้ายกับกรณีที่เราใช้เงื่อนไข if ซ้อนกัน โดยเมื่อเกิดการวนลูปชั้นนอกในแต่ละรอบ จะทำให้ลูป for ที่อยู่ชั้นในวนไปจนครบรอบตามเงื่อนไข แล้วจึงจะกลับออกมารวนลูปชั้นนอกในรอบถัดไป ซึ่งเราจะวางแผนลูป for ซ้อนกันกี่ชั้นก็ได้ ดังรูปแบบต่อไปนี้

```

for (...) {
    ...
    for (...) {
        ...
        for (...) {
            ...
        }
    ...
}
...
}

```

แนวทางการใช้งาน เช่น สมมติว่าเราต้องการพิมพ์ตัวเลข 5 บรรทัด โดยในแต่ละบรรทัดนั้น ก็จะพิมพ์เลข 1- 10 นั้นแสดงว่าลูปซึ้นนอกจะต้องวน 5 รอบ และลูปซึ้นในจะต้องวน 10 รอบ ดังนี้

```

for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 10; j++) {
        printf("%d ", j);
    }
    putchar('\n'); //ขึ้นบรรทัดใหม่
}

```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10

จากโค้ด ลูปซึ้นใน (j) จะต้องวนไปจนครบรอบก่อน ดังจะเห็นได้จากการพิมพ์ตัวเลข 1-10 ในแต่ละบรรทัด และจึงจะกลับสู่ลูปซึ้นนอก (i) ซึ่งในการนี้ระหว่างลูป i และ j ไม่ได้เกี่ยวข้อง หรือมีผลต่อกัน เพียงแค่ว่าลูปให้ครบตามจำนวนรอบเท่านั้น

อย่างไรก็ตาม อาจมีบางกรณีที่ต้องนำตัวแปรของลูปซึ้นนอกไปใช้ร่วมกับตัวแปรของลูปซึ้นใน ดังในโค้ดต่อไปนี้ ซึ่งเราจะให้ลูปซึ้นในวนรอบเป็นจำนวนครั้งเท่ากับค่าในแต่ละรอบของลูปซึ้นนอก เช่น

วนลูปซึ้นนอกครั้งที่ 1	วนลูปซึ้นใน 1 รอบ
วนลูปซึ้นนอกครั้งที่ 2	วนลูปซึ้นใน 2 รอบ
วนลูปซึ้นนอกครั้งที่ 3	วนลูปซึ้นใน 3 รอบ
...	

ดังนั้น เราต้องนำตัวแปรของลูปซึ้นนอกมาเป็นขอบเขตของเงื่อนไขของลูปซึ้นใน ดังนี้

```

for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= i; j++) {
        printf("%d ", j);
    }
}

```

1				
1	2			
1	2	3		
1	2	3	4	
1	2	3	4	5

```

    }
    putchar( '\n' );
}

```

**ตัวอย่าง 7-3** เป็นการสร้างตารางสูตรคูณ โดยใช้ลูปแบบ for ซ้อนกัน 2 ชั้น

multiplication table									
1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

1 x 1	1 x 2	1 x 3	1 x 4
2 x 1	2 x 2	2 x 3	2 x 4
3 x 1	3 x 2	3 x 3	3 x 4

```

#include <stdio.h>
#include <string.h>

void main() {
    printf("\n----- ");
    printf("multiplication table");
    printf(" ----- \n");

    // ลูปชั้นนอก (i) สำหรับตัวเลขในแนวตั้ง
    for (int i = 1; i <= 10; i++) {

        // ลูปชั้นใน (j) สำหรับเลขในแนวนอน
        for (int j = 1; j <= 10; j++) {
            // คอลัมน์แรก ความกว้างเป็น 2 และจัดซีดซ้าย
            if (j == 1) {
                printf("%-2d", i * j);
            }
            // คอลัมน์ถัดไป ความกว้างเป็น 5 และจัดซีดขวา
            else {
                printf("%5d", i * j);
            }
        }

        putchar(' '); // ถ้าครบลูป j ให้ขึ้นแควใหม่
    }
}

```

## การใช้คำสั่ง break

ตามปกติแล้ว ลูปแบบ for จะเกิดการวนรอบไปจนกว่าเงื่อนไขจะเป็นเท็จ แต่อาจมีข้อกำหนดบางอย่างที่ต้องการให้ลูปหยุดทำงานในขณะใดขณะหนึ่ง ถึงแม้เงื่อนไขการวนลูปจะยังเป็นจริงอยู่ก็ตาม ซึ่งลักษณะเช่นนี้ เราสามารถนำคำสั่ง break มาวางแผน จุดที่ต้องการออกจากลูปได้เลย เช่น

```
int sum = 0;
for (int n = 1; n <= 10; n++) {
    sum += n;
    if (n >= 5) {
        break;
    }
}
printf("sum = %d", sum);
//1 + 2 + 3 + 4 + 5 = 15
```

จากโค้ดข้างบน เมื่อ  $n = 5$  การวนลูปจะลิ้นสุดลง เพราะถูกกำหนดด้วยคำสั่ง break ถึงแม้จะยังไม่ครบตามเงื่อนไข แต่อย่างไรก็ตาม กรณีที่ใช้คำสั่ง break ร่วมกับ switch และซ้อนอยู่ในลูป for จะไม่มีผลให้ลูปหยุดการทำงานแต่อย่างใด เช่น ตัวอย่างต่อไปนี้

**ตัวอย่าง 7-4** ตรวจสอบว่าเมื่อเราใช้คำสั่ง switch ที่ต้องมี break ในแต่ละ case หากซ้อนอยู่ในลูป for จะเกิดผลอย่างไร

```
#include <stdio.h>

void main() {
    int remainder = 0;
    for (int n = 1; n <= 10; n++) {
        remainder = n % 2;
        switch(remainder) {
            case 0:
                printf("%d is even\n", n);
                break;
            default:
                printf("%d is odd\n", n);
                break;
        }
    }
}
```

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
8 is even
9 is odd
10 is even
```

จากโค้ด จะเห็นว่าลูป for วนรอบตั้งแต่ค่าแรกจนถึงค่าสุดท้าย ( $n = 1$  ถึง  $n = 10$ ) แม้ภายในลูปจะมีคำสั่ง break ของ switch ก็ตาม นั่นแสดงว่า คำสั่ง break ของ switch ไม่มีผลต่อลูป for

## การใช้คำสั่ง continue

คำสั่ง continue เป็นการสั่งให้วนลูปถัดไปทันที โดยไม่ทำการคำนวณในลูป for ที่อยู่ถัดจาก continue ลงมา ดังนั้นจึงใช้ในทางตรงกันข้ามกับคำสั่ง break เช่น ในโค้ดต่อไปนี้

```
int sum = 0;
for (int n = 1; n <= 10; n++) {
    if (n == 5) {
        continue;
    }
    sum += n;
}

printf("sum = %d", sum);
//1 + 2 + 3 + 4 + 6 + 7 + 8 + 9 + 10 = 50
```

จากโค้ด เมื่อ  $n = 5$  จะวนลูปถัดไปทันที ดังนั้น คำสั่ง  $sum += 5$  จึงถูกข้าม และทำการคำนวณต่อไป (ถ้าเป็น break จะออกจากลูป)

## การกำหนดลูป for แบบไม่รู้จบ (Infinite Loop)

กรณีที่เราต้องการวนลูปไปเรื่อยๆ แบบไม่กำหนดเงื่อนไขไว้ล่วงหน้าว่าจะหยุดเมื่อใด ก็อาจสร้างเป็นลูปแบบไม่รู้จบ (Infinite Loop) โดยไม่ต้องมีส่วนที่กำหนดค่าเริ่มต้น ส่วนเงื่อนไข และส่วนการเปลี่ยนค่า แต่เขียนลงไปเฉพาะเครื่องหมาย semi-colon(); จำนวน 2 อันไว้ในวงเล็บ

```
for ( ; ; ) {
    ...
}
```

โค้ดข้างบนจะทำให้เกิดการวนลูปแบบไม่รู้จบ ดังนั้น เราจะต้องกำหนดเงื่อนไขบางอย่างไว้ในบล็อกร่วมกับการใช้คำสั่ง break เพื่อหยุดการวนรอบของลูป ดังตัวอย่างด้านไป

**ตัวอย่าง 7-5** เป็นการสร้างลูป for เพื่อรับตัวเลขระหว่าง 50 - 100 แต่หากผู้ใช้ใส่ค่าที่ไม่อยู่ในช่วงนี้ ก็จะรับค่าใหม่ไปเรื่อยๆ จนกว่าจะใส่ค่าได้ถูกต้อง ซึ่งในกรณีนี้ เราไม่ทราบว่า ล่วงหน้าจะต้องทำซ้ำกี่ครั้ง ดังนั้น จึงใช้ลูป for แบบไม่รู้จบ แล้วถ้าใส่ข้อมูลถูกต้องก็จะออกจากลูปด้วยคำสั่ง break

```
#include <stdio.h>

void main() {
    int n;
    for ( ; ; ) {
        printf("enter integer number 50 - 100 >>");
        scanf("%d", &n);
        if (n >= 50 && n <= 100) {
            puts("thanks!");
            break; //ออกจากลูป for
        }
    }
}
```

```
enter integer number 50 - 100 >>49
enter integer number 50 - 100 >>101
enter integer number 50 - 100 >>74
thanks!
```

จากโค้ดข้างบนเราจะวนลูปเพื่อรับค่าจากผู้ใช้ไปเรื่อยๆ จนกว่าจะใส่ค่าระหว่าง 50-100 ซึ่งในกรณีนี้เราไม่ทราบล่วงหน้าว่าต้องวนลูปกี่รอบ ดังนั้น จึงต้องมากำหนดเงื่อนไขภายในลูป หลังจากที่ได้รับข้อมูลเข้ามาแล้ว แต่โดยทั่วไป เราไม่นิยมใช้ลูป for ในแบบไม่รู้จบกันมากนัก เพราะยังมีลูปอื่นๆ ที่ถูกสร้างมาเพื่อวัตถุประสงค์นี้โดยตรงอยู่แล้ว เช่น while และ do-while ดังที่จะกล่าวถึงในหัวข้อต่อๆ ไป

## ลักษณะที่ควรรู้เพิ่มเติมเกี่ยวกับลูป for

นอกจากพื้นฐานทั่วไปของลูป for ที่ได้กล่าวในหัวข้อต่างๆ ที่ผ่านมา ก็ยังมีสิ่งที่เราควรรู้เพิ่มเติมอีกหลายอย่าง ซึ่งแยกพิจารณาได้ดังต่อไปนี้

- ตัวแปร หรือตัวบันทึกประการขั้นในลูป for จะใช้ได้เฉพาะภายในลูปนี้เท่านั้น โดยหลังจากลิ้นสุดลูปแล้ว ไม่สามารถอ้างถึงตัวแปรนี้ได้อีก ยกเว้นเราจะสร้างตัวแปรไว้ก่อนลูปแล้วนำมาเป็นตัวบันทึกในลูป for แม้ลูปจะลิ้นสุดแล้ว ก็ยังสามารถใช้ตัวแปรดังกล่าวต่อไปได้ เช่น

```
for (int x = 10; x <= 100; x += 10) {
    ...
    int y = ...;
```

```

}

printf("%d", x);           //Error
printf("%d", y);           //Error

int y;
for (y = 100; y >= 10; y -= 10) {
    ...
}
printf("%d", y);           //OK สามารถอ้างถึงตัวแปร y ตรงนี้ได้

```

- ถ้าตัวแปรนั้นถูกประกาศเอาไว้ก่อนนี้แล้ว เราไม่สามารถนำมาประกาศซ้ำในลูป for ได้อีก เช่น

```

int i;
int n;
for (int i = 1; i <= 10; i++) {      //Error
    int n = i;                      //Error
}

```

- ในการกำหนดเงื่อนไขของลูป ถ้าเป็นกรณีที่ต้องคำนวนค่าเพื่อนำมาเปรียบเทียบ ก็ควรทำการคำนวนให้ได้ผลลัพธ์ออกมาก่อน แต่ไม่ควรทำการคำนวน ณ ตำแหน่งที่กำหนดเงื่อนไข เพราะจะเกิดการคำนวนซ้ำๆ ในทุกรอบของการวนลูป ดังโค้ดตัดไปซึ่งเราไม่ควรใช้รูปแบบแรก แต่ควรใช้รูปแบบที่สองมากกว่า

```

//รูปแบบแรก คำนวนในขั้นตอนการระบุเงื่อนไขของลูป
//ไม่ควรใช้แบบนี้
int a = 10;
int b = 5;
for (int x = 1; x <= (a + b); x++) {
    ...
}

```

```

//รูปแบบที่สอง คำนวนก่อนแล้วค่อยนำมากำหนดเป็นเงื่อนไข
//ควรใช้แบบนี้มากกว่า
int a = 10;
int b = 5;
int c = a + b;
for(int x = 1; x <= c; x++) {
    ...
}

```

- ในบางครั้ง การเปลี่ยนค่าของตัวแปร อาจอยู่นอกขอบเขตของเงื่อนไขที่เรากำหนด เอาไว้ โดยเฉพาะอย่างยิ่งกรณีที่ใช้เครื่องหมาย != ในการเปรียบเทียบ ซึ่งลักษณะ เช่นนี้จะทำให้ลูปทำงานต่อไปเรื่อยๆ แบบไม่รู้จบ เช่นในโค้ดถัดไป เราเพิ่มค่า i ครั้งละ 2 ดังนั้น ในแต่ละลูปค่า i จะเป็น 1, 3, 5, 7, ... แต่เรากำหนดเงื่อนไขเป็น i != 6 จึงไม่มีลูปใดที่จะทำให้เงื่อนไขนี้เป็นเท็จและหยุดการทำงานลงได้ ซึ่งวิธีแก้ปัญหานี้ คือ หลีกเลี่ยงการเปรียบเทียบด้วยเครื่องหมาย != และเปลี่ยนไปใช้พวก >, >=, <=, <= แทน

```
for (int i = 1; i != 6; i += 2) {      //i = 1, 3, 5, 7, 9, ...
    printf("%d ", i);                  //เกิด infinite loop
}
```

- ในการนับที่เรากำหนดเงื่อนไขเป็นเท็จตั้งแต่เริ่มแรก คอมไพล์เวอร์จะไม่แจ้งข้อผิดพลาด และไม่มีการทำคำสั่งใดๆ ในบล็อกของ for เช่น โค้ดถัดไป ซึ่งเรากำหนดค่า n = 100 แต่เงื่อนไขคือ n <= 10 ทำให้เงื่อนไขเป็นเท็จตั้งแต่เริ่มแรก ดังนั้น คำสั่งทั้งหมดในบล็อก for จึงถูกข้ามไป

```
for (int n = 100; n <= 10; n += 5;) {
    printf("%d ", n);
}
```

- ในส่วนของค่าเริ่มต้น เราสามารถกำหนดตัวแปรและค่าเริ่มต้นได้มากกว่า 1 ตัว โดยคั่นแต่ละตัวด้วยเครื่องหมาย comma (,) เช่น

```
int i, sum;
for (i = 1, sum = 0; i < 10; i++) {
    sum += i;
}
printf("sum = %d", sum);    //45
```

- ในส่วนของการเปลี่ยนค่าตัวแปร เราสามารถอัปเดตตัวแปรพร้อมกันได้มากกว่า 1 ตัว โดยคั่นแต่ละตัวด้วยเครื่องหมาย comma (,) เช่น

```
for (int i = 1, j = 10; i <= 5; i++, j += 10) {
    printf("%d x %d = %d\n", i, j, i * j);
}
```

- เราสามารถใช้คำสั่ง for ในแบบ Expression โดยไม่ต้องมีบล็อก {} แต่ต้องมีเครื่องหมาย ; ปิดท้ายวงเล็บ แล้วนำคำสั่งที่ต้องการทำมาเขียนไว้ในส่วนของการเปลี่ยนค่าได้เลย โดยคั่นระหว่างคำสั่งด้วยเครื่องหมาย comma (,) เช่น

```
int i;
for (i = 0; i < 5; i++, printf("%d ", i));
//ต้องมี ; ปิดท้าย เพราะเป็น Expression
```

- ภาษา C ไม่มีข้อห้ามในการเปลี่ยนค่าของตัวแปรภายในลูป แต่เราต้องแน่ใจว่า การกระทำเข่นนั้นจะไม่ส่งผลให้ลูปทำงานผิดพลาด เช่นในโค้ดถัดไป ซึ่งจะเกิดการวนลูปแบบไม่รู้จบ เพราะค่าของ i จะเท่าเดิมตลอด

```
for (int i = 1; i <= 5; i++) { //เกิด infinite loop
    i--;
    printf("%d ", i);
}
```

## การใช้ลูปแบบ while

การทำงานเป็นวงรอบด้วยลูป for มักจะใช้เฉพาะกรณีที่เราทราบค่าเริ่มต้นและค่าสุดท้ายของการวนรอบ แต่หากไม่ทราบค่าเหล่านี้ล่วงหน้า แต่ทราบเพียงเงื่อนไขบางอย่างที่ทำให้เกิดการวนรอบ ก็อาจเปลี่ยนมาใช้ลูป while แทนได้ โดยลูปแบบ while จะมีการตรวจสอบเงื่อนไขก่อนวนรอบ หากตรงตามที่กำหนด (เป็นจริง) ก็จะเริ่มทำการคำสั่งในลูป แล้ววนรอบไปเรื่อยๆ จนกว่าเงื่อนไขจะเป็นเท็จ ซึ่งมีรูปแบบดังนี้

```
while (เงื่อนไข) {
    คำสั่งต่างๆ
}
```

สำหรับเงื่อนไข ก็กำหนดคล้ายกับคำสั่ง if คือต้องเป็นการเปรียบเทียบที่ให้ค่าอ กมาเป็นค่าบูลีน (true หรือ false) เท่านั้น เช่น การแสดงตัวเลข 1 - 5 โดยใช้ลูป while

```
int x = 1;
while (x <= 5) {
    printf("%d ", x);
    x++;
}
```

หลักการทำงานของโค้ดข้างบนคือ ลูป while จะตรวจสอบเงื่อนไขที่ต่อนตันลูป โดยเริ่มแรก  $1 \leq 5$  เป็นจริง จึงเริ่มทำการคำสั่งต่อๆ กันในบล็อกของมัน แล้วก่อนจะวนลูปถัดไป ก็ตรวจสอบเงื่อนไข  $x \leq 5$  ถ้าเป็นจริง ก็ทำการคำสั่งในลูปจนครบ จากนั้นก็ตรวจสอบเงื่อนไข เป็นเช่นนี้ไปเรื่อยๆ จนถึง  $x = 6$  ซึ่ง  $6 \leq 5$  เป็นเท็จ ดังนั้นลูป while จึงหยุดการวนรอบ และไม่ทำการคำสั่งภายในบล็อกของมันต่อ ซึ่งลักษณะที่เราควรรู้เพิ่มเติมเกี่ยวกับลูปแบบ while มีดังนี้

- ภายในบล็อกของลูป while จะต้องมีคำสั่งที่สามารถทำให้เงื่อนไขเปลี่ยนจากจริง เป็นเท็จได้ มิฉะนั้นจะเกิดการวนรอบไม่รู้จบ เช่นโค้ดต่อไปนี้

```
while (true) {           //เงื่อนไขเป็นจริงตลอด
    puts("Hello");       //จึงเกิดการวนลูปไม่รู้จบ (infinite)
}
```

```
int x = 1;
while (x <= 5) {         //เกิดการวนลูปไม่รู้จบเช่นกัน
    printf("%d ", x);
}
```

- ควรหลีกเลี่ยงการใช้เครื่องหมาย  $!=$  ในการเปรียบเทียบเงื่อนไขด้วยเหตุผลที่คล้ายกับลูปแบบ for เพราะหากเราพิจารณาไม่รอบคอบพอ เงื่อนไขนั้นอาจไม่มีทางที่จะเป็นเท็จได้ จนเกิดการวนลูปไม่รู้จบ ดังการเปรียบเทียบในโค้ดดังไป

```
//หากทำแบบนี้ จะเกิดการวนลูปไม่รู้จบ
// เพราะ x ไม่มีทางเท่ากับ 6 เงื่อนไขจึงเป็นจริงตลอด
int x = 1;
while (x != 6) {
    printf("%d ", x);
    x += 2;           //x = 1, 3, 5, 7, 9, ...
}
```

```
//ควรใช้ <, <=, >หรือ >= ในการเปรียบเทียบมากกว่า
int x = 1;
while (x < 6) {
    printf("%d ", x);
    x += 2;           //x = 1, 3, 5
}
```

- เราสามารถนำคำสั่ง break มาใช้กับ while เพื่อให้หยุดการวนรอบได้เช่นเดียวกับ for ดังโค้ดต่อไปนี้ ซึ่งจะพิมพ์ตัวเลข 1 - 5 แล้วหยุด

```

int x = 1;
while (true) {
    if (x > 5) {
        break;
    }
    printf("%d ", x);
    x++;
}

```

- หรือสามารถใช้คำสั่ง continue เพื่อวนลูปตัดไป โดยไม่ทำคำสั่งส่วนที่เหลือของบล็อก while

```

int x;
while (true) {
    printf("positive integer number >>");
    scanf("%d", &x);

    if (x < 0) {
        continue;
    }
}

```

- ในการนีการประกาศตัวแปร ก็สามารถนำหลักการที่ได้กล่าวไว้ในลูปแบบ for มาใช้งานได้ เช่น
  - ควรประกาศตัวแปรไว้ก่อนลูป แต่ไม่ควรประกาศไว้ภายในลูป เพราะจะเกิดการประกาศซ้ำ ๆ ในการวนลูปแต่ละรอบ
  - ไม่ควรทำการคำนวณ ณ จุดที่กำหนดเงื่อนไข เพราะเกิดการคำนวณแบบซ้ำ ๆ ในแต่ละลูป เช่นในโค้ดตัดไป ควรใช้รูปแบบทางด้านล่างมากกว่าที่จะใช้รูปแบบทางด้านบน

```

int x = 10;
while (x < 100) {
    ...
    int y = x; //ไม่ควรประกาศตัวแปรไว้ในลูป
}

```

```

int x = 10;
int y; //แต่ควรประกาศไว้ก่อนลูป
while (x < 100) {
    ...
}

```

```

y = x;
}

//ไม่ควรทำการคำนวนในขั้นตอนการตรวจสอบเงื่อนไข
// เพราะจะเกิดการทำซ้ำๆ ในทุก-loop
int a = 10, b = 5;
int x = 1;
while (x < (a + b)) {
    ...
}

//ควรคำนวนให้เสร็จก่อน แล้วค่อยไปกำหนดเงื่อนไข
int a = 10, b = 5;
int c = a + b;
int x = 1;
while (x < c) {
    ...
}

```

- เราสามารถใช้ลูป while ช้อนกัน หรือใช้จะใช้ลูป while ช้อนกับ for ก็ได้ ทั้งนี้ขึ้นกับลักษณะของงานที่จะทำเป็นหลัก ให้ดูตัวอย่างต่อๆ ไป

**ตัวอย่าง 7-6** การตรวจสอบว่า ถ้าเราสร้างสุ่มระหว่าง 0 - 100 จะต้องสุ่มกี่ครั้ง จึงจะได้เลขที่มีค่าตั้งแต่ 90 ขึ้นไป

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void main() {
    srand(time(0));
    int x = 0, times = 0;
    putchar('\n');

    while (x < 90) {          //ถ้าได้ค่าที่น้อยกว่า 90 ต้องวนลูปต่อไป
        x = rand() % 101; //ค่าเลขสุ่มที่จะได้ จะอยู่ระหว่าง 0 - 100
        printf("%d ", x);
        times += 1;
    }

    printf("\nrandomized %d time(s)\n", times);
}

```

37 48 15 13 78 19 18 38 46 14 17 93  
randomized 12 time(s)

**ตัวอย่าง 7-7** เป็นการรับรหัสผ่านจากผู้ใช้ ถ้ารหัสที่ถูกต้องคือ 567890 ซึ่งหากใส่ผิดก็ให้ใส่ใหม่ ดังนั้น จึงใช้ลูป while ในการรับข้อมูลไปเรื่อยๆ จนกว่าจะใส่ค่าที่ถูกต้อง

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

void main() {
    char password[10];
    putchar('\n');

    //ให้เงื่อนไขเป็น true เพื่อวนลูปไปเรื่อยๆ
    while (true) {
        printf("enter password >>");
        gets(password);

        //ถ้าใส่รหัสถูก ให้ออกจากลูป while
        if (strcmp(password, "567890") == 0) {
            break;
        }
    }
    puts("thanks!");
}
```

enter password >>555  
 enter password >>5678  
 enter password >>56789  
 enter password >>567890  
 thanks!

## การใช้ลูปแบบ do-while

ลูปแบบ do-while เป็นการวนลูปโดยใช้เงื่อนไขเป็นตัวกำหนดคล้ายกับลูปแบบ while แต่ในกรณีของ do-while การตรวจสอบเงื่อนไขจะเกิดขึ้นที่ท้ายลูป ดังนั้น จึงต้องทำการคำสั่งภายในบล็อกของมันอย่างน้อย 1 ครั้ง โดยทราบได้ที่เงื่อนไขยังเป็นจริง การวนลูปจะดำเนินไปเรื่อยๆ จนกว่าเงื่อนไขจะเป็นเท็จ ซึ่งรูปแบบของลูปนี้คือ

```
do {
    คำสั่งต่างๆ
} while (เงื่อนไข);
```

สำหรับการกำหนดเงื่อนไขก็มีหลักการเช่นเดียวกับลูปแบบ while หรือคำสั่ง if คือต้องเป็นการเปรียบเทียบที่ให้ค่าอ กมาเป็น true หรือ false เท่านั้น และต้องเขียนเงื่อนไขไว้ในวงเล็บ ดังตัวอย่างต่อไปนี้

**ตัวอย่าง 7-8** เป็นการใช้ลูป do-while เพื่อรับข้อมูลตัวเลขจากผู้ใช้จนกว่าจะได้ค่าที่มากกว่า 99

```
#include <stdio.h>

void main() {
    int x;

    putchar('\n');

    do {
        printf("enter number greater than 99 >>");
        scanf("%d", &x);
    } while (x <= 99);

    printf("thanks!\n");
}
```

```
enter number greater than 99 >>98
enter number greater than 99 >>99
enter number greater than 99 >>100
enter number greater than 99 >>100
thanks!
```

จากโค้ดข้างบน เมื่อเริ่มต้นลูป จะแสดงข้อความให้ผู้ใช้ใส่ข้อมูลทันที โดยยังไม่มีการตรวจสอบเงื่อนไขใด ๆ เมื่อทำการบุกคำสั่งในล็อก จึงจะตรวจสอบเงื่อนไขที่ท้ายลูป และถ้าเงื่อนไขเป็นจริง เช่น เราใส่ข้อมูลที่น้อยกว่าหรือเท่ากับ 99 ก็จะวนลูปตัดไปเรื่อย ๆ จนกว่าจะใส่ค่าที่มากกว่า 99 เพื่อให้เงื่อนไขเป็นเท็จ จึงจะออกจากลูป โดยลักษณะที่ควรรู้เพิ่มเติมเกี่ยวกับลูปแบบ do-while คือ

- กรณีของลูป do-while ต้องมี semi-colon (;) ปิดท้ายวงเล็บที่กำหนดเงื่อนไขของลูปเสมอ เช่น

```
do {
    ...
} while (...); //ต้องมี semi-colon ปิดท้ายเงื่อนไข
```

- ลูปแบบ do-while กับแบบ while มีข้อแตกต่างที่สำคัญดังนี้
  - ลูปแบบ while จะตรวจสอบเงื่อนไขก่อนเริ่มลูป และจะวนลูปก็ต่อเมื่อเงื่อนไขเป็นจริง
  - ลูปแบบ do-while จะตรวจสอบเงื่อนไขที่ท้ายลูป นั่นแสดงว่าลูปแบบนี้จะต้องทำงานคำสั่งที่กำหนดในล็อกของลูปอย่างน้อย 1 ครั้ง และจะวนลูปตัดไปเมื่อเงื่อนไขเป็นจริงเช่นเดียวกัน

- จากการที่ลูปแบบ do-while จะตรวจสอบเงื่อนไขที่ท้ายลูป ดังนั้น ลูปแบบนี้จึงเหมาะกับงานที่ต้องมีการกระทำการอย่างไปก่อนในครั้งแรก และค่อยตรวจสอบเงื่อนไขภายหลัง
- เนื่องจากลูปแบบนี้ จะใช้เงื่อนไขเป็นตัวกำหนดขอบเขตการวนลูป ดังนั้น คำลั่งภายในบล็อก do จะต้องมีกรณีที่จะทำให้เงื่อนไขมีค่าเป็นเท็จ หรือไม่ก็ใช้คำลั่ง break เพื่อหยุดการทำงานเช่นเดียวกับลูปแบบ while มิฉะนั้นจะเกิดการวนรอบแบบไม่รู้จบ เช่น

```
int result;
do {
    result = rand() %1000;
    if (result >= 500) {
        break;
    }
} while (true);

printf("random number = %d", result);
```

ลักษณะปลีกย่อยเพิ่มเติมอื่นๆ ก็สามารถนำหลักการที่ได้กล่าวไว้ในลูปแบบ while มาใช้ได้

**ตัวอย่าง 7-9** การนับจำนวนตัวอักษรที่เป็นสระ (vowel) ในภาษาอังกฤษ ด้วยการใช้ลูป do-while โดยในแต่ละลูปจะใช้ฟังก์ชัน getchar() เพื่อรับอักษรละ 1 ตัวจากผู้ใช้ด้วย และตรวจสอบว่าอักษรตัวนั้นเป็นสระหรือไม่ ถ้าเป็นก็นับเพิ่มไปอีก 1 ช่องเราจะวนลูปรับค่าไปเรื่อยๆ จนกว่าผู้ใช้จะใส่เลข 0 จึงหยุดหรือออกจากลูป

```
#include <stdio.h>
#include <stdbool.h>
#include <ctype.h>

void main() {
    int count_vowels = 0;
    char c;

    printf("\nenter 0 to stop\n");

    do {
        printf("enter char >>");
        c = getchar();

        getchar(); //clear \n

        if (c == '0') {
```

```
enter 0 to stop
enter char >>e
enter char >>d
enter char >>u
enter char >>c
enter char >>a
enter char >>t
enter char >>i
enter char >>o
enter char >>n
enter char >>0

vowel letters: 5
```

```

        break;
    }

    c = tolower(c); //แปลงเป็นตัวพิมพ์เล็ก

    if (c=='a'||c=='e'||c=='i'||c=='o'||c=='u') {
        count_vowels++;
    }

} while (true);

printf("\nvowel letters: %d\n", count_vowels);
}

```

**ตัวอย่าง 7-10** เป็นการสร้างเกมทายตัวเลข โดยมีหลักการดังต่อไปนี้

- เมื่อเริ่มเกมให้สร้างเลขสุ่ม ให้มีค่าระหว่าง 0-99
- ใช้ลูป do-while หรือ while หรือ for ก็ได้ แล้วแต่ถนัด เพื่อให้ผู้เล่นใส่ตัวเลข ที่ต้องการทาย
- ถ้าผู้เล่นใส่เลขที่มากกว่าค่าที่สุ่มได้ ให้แสดงข้อความว่า "น้อยกว่านี้"
- ถ้าผู้เล่นใส่เลขที่น้อยกว่าค่าที่สุ่มได้ ให้แสดงข้อความว่า "มากกว่านี้"
- ในที่นี้มีข้อกำหนดว่าให้ทายได้ไม่เกิน 7 ครั้ง ดังนั้นจึงให้วนลูปเพื่อทายไปจนกว่าจะถูก (ชนะ) แต่ถ้าทายครบ 7 ครั้งแล้วยังไม่ถูก ก็ให้ถือว่าแพ้

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>

void main() {
    srand(time(0));
    //สร้างเลขสุ่มสำหรับให้ทาย
    int rand_num = 1 + rand() % (100 - 1 + 1);
    int count = 1;           //ทายครั้งที่เท่าไหร่
    const int MAX = 7;       //ทายผิดไม่เกิน 7 ครั้ง
    int guess_num;          //ค่าที่ผู้เล่นทายแต่ละครั้ง
}

```

```

*** enter integer number 1-100 ***

guesstimate #1 >>50
less than 50

guesstimate #2 >>30
less than 30

guesstimate #3 >>20
greater than 20

guesstimate #4 >>25
greater than 25

guesstimate #5 >>28
greater than 28

guesstimate #6 >>29
yes! the number is 29

```

```

printf("\n*** enter integer number 1-100 ***\n");
do {
    //รับค่าตัวเลขที่พิมพ์ทางแป้นพิมพ์
    printf("\nguesstimate # %d >>", count);
    scanf("%d", &guess_num);

    //ถ้าเลขที่ผู้ใช้พิมพ์มากกว่า เลขที่สุ่มได้
    //ให้แสดงข้อความว่า น้อยกว่านี้
    //หมายถึง ให้ใส่เลขที่น้อยกว่านี้
    if (guess_num > rand_num) {
        printf("less than %d\n", guess_num);
    }
    //ถ้าเลขที่ผู้ใช้พิมพ์น้อยกว่า เลขที่สุ่มได้
    //ให้แสดงข้อความว่า มากกว่านี้
    //หมายถึง ให้ใส่เลขที่มากกว่านี้
    else if (guess_num < rand_num) {
        printf("greater than %d\n", guess_num);
    }
    //ถ้าเลขที่ผู้ใช้พิมพ์เท่ากับ เลขที่สุ่มได้
    //นั่นคือพิเศษมาก ให้ออกจากลูป
    else if (guess_num == rand_num) {
        printf("yes! the number is %d\n", guess_num);
        break;
    }

    //ผู้ใช้พิมพ์จำนวนครั้งที่กำหนดไว้
    //ให้จบเกม และออกจากลูป
    if (count == MAX) {
        printf("\ngame over!, you guessed ");
        printf("more than %d times\n", MAX);
        break;
    }

    count++;      //เพิ่มตัวนับ ว่าเป็นการพิมพ์ครั้งที่เท่าไร
} while (true);
}

```

เนื้อหาหลักที่เราได้เรียนรู้ในบทนี้คือ การวนลูป (Loop) แบบต่างๆ เช่น for, while หรือ do-while เป็นต้น ซึ่งลูปทั้งหมดเหล่านี้ล้วนเป็นพื้นฐานสำคัญของการเขียนโปรแกรมภาษา C เพื่อควบคุมการทำงานแบบช้าๆ จนกว่าจะครบถ้วนตามต้องการ และเนื่องจากลูปแต่ละแบบมีหลักการกำหนดเงื่อนไขที่แตกต่างกัน ดังนั้น เราต้องเลือกลูปให้เหมาะสมกับงานที่ต้องดำเนินการ จึงจะได้ผลลัพธ์ที่ถูกต้องและไม่เกิดข้อผิดพลาด



# 8

## รวมตัวอย่างโค้ดเพิ่มเติม ชุดที่ 1

ในบทนี้ เป็นการรวบรวมตัวอย่างโค้ดชุดแรกซึ่งจะนำพื้นฐานความรู้จากบทก่อนๆ มารวมเข้าด้วยกัน เพื่อเป็นการทบทวนเนื้อหาที่สำคัญให้เกิดความเข้าใจมากยิ่งขึ้น และมองเห็นแนวทางการนำไปใช้งานในรูปแบบที่แตกต่างออกไป นอกจากนี้ยังช่วยเสริมทักษะการเขียนโปรแกรมให้ก้าวหน้าได้อย่างรวดเร็วขึ้นอีกด้วย

### ตัวอย่าง 8-1 เครื่องคิดเลขแบบรับข้อมูลพร้อมกัน

จากตัวอย่างในบทที่ 6 เราสร้างเครื่องคิดเลขอย่างง่ายโดยรับข้อมูล 3 อย่างคือ จำนวนที่ 1 ตามด้วยเครื่องหมาย และจำนวนที่ 2 ซึ่งทั้งหมดรับผ่าน `scanf()` แยกกัน สำหรับในตัวอย่างนี้ เราจะนำตัวอย่างนั้นมาแก้ไขใหม่ โดยรับข้อมูลทั้ง 3 อย่างในคำสั่ง `scanf()` เดียวกัน แต่แยกจัดเก็บค่าละตัวแปร ส่วนขั้นตอนการนำไปคำนวณก็ใช้หลักการเดิม

```
#include <stdio.h>

void main() {
    float n1, n2, r;
    char op, buffer;

    printf("\nenter num1 operator num2");
    printf("\nex ample: 10 + 5 >>");
    scanf("%f%c%c%f", &n1, &buffer, &op, &n2);

    switch (op) {
        case '+': r = n1 + n2; break;
        case '-': r = n1 - n2; break;
        case '*': r = n1 * n2; break;
        case '/': r = n1 / n2; break;
        case '%': r = (int)n1 % (int)n2; break;
    }
    printf("\n%g %c %g = %g \n", n1, op, n2, r);
}
```

```
enter num1 operator num2
example: 10 + 5 >>-5 - -5.5
-5 - -5.5 = 0.5
```

**ตัวอย่าง 8-2 หาค่า Factorial (n!)**

การหาค่า Factorial ในทางคณิตศาสตร์นั้น มีหลักการที่ไม่ยุ่งยาก เช่น

$$\#5! = 5*4*3*2*1$$

$$\text{ดังนั้น } \#n! = (n)*(n-1)*(n-2)*(n-3)*...*1 ; n \geq 0$$

สำหรับการเขียนโค้ดในแบบภาษา C เราสามารถใช้ลูป for หรือ while เพื่อนำค่าของ n มาคูณกับผลคูณเดิมแล้วลดค่า n ลงเรื่อยๆ จนถึง 1 ดังโค้ดต่อไปนี้

```
number (>=0) to find factorial >>0
0! = 1
```

```
number (>=0) to find factorial >>7
7! = 5040
```

```
#include <stdio.h>

void main() {
    int n, fac;

    printf("\nnumber (>=0) to find factorial >>");
    scanf("%d", &n);

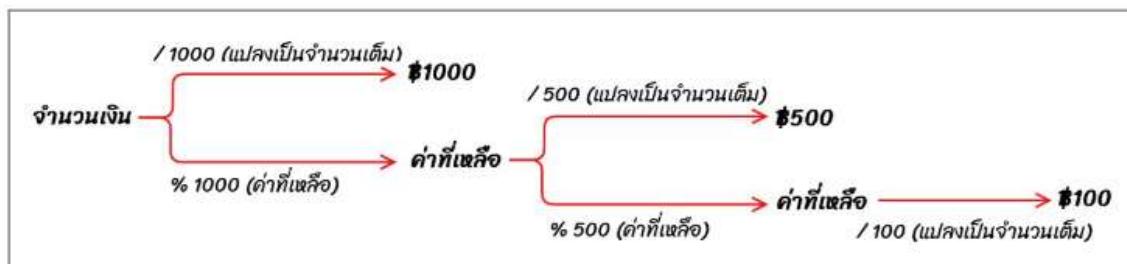
    if (n >= 0) {
        fac = 1;
        for (int i = n; i >= 1; i--) {
            fac *= i;
        }

        //กรณีใช้ลูป while
        /*
        int i = n;
        while (i >= 1) {
            fac *= i;
            i -= 1;
        }
        */

        printf("\n%d! = %d\n", n, fac);
    } else {
        printf("\nerror! number must be >= 0\n");
    }
}
```

### ตัวอย่าง 8-3 ตรวจสอบการถอนเงินจากตู้ ATM

ตู้ ATM สามารถจ่ายธนบัตรให้เราได้ 3 ชนิดคือ 1000, 500 และ 100 เมื่อเรากดเงินตามจำนวนที่ต้องการ ให้คำนวนหาจำนวนธนบัตรแต่ละชนิดที่เครื่องจะจ่ายให้เรา โดยให้เริ่มจากชนิดที่มีค่ามากสุดที่เป็นไปได้เสมอ ซึ่งมีแนวทางดังนี้



#	หลักการ	ตัวอย่างโค้ด
1	ให้เริ่มจากชนิด 1000 นาทก่อน โดยนำจำนวนที่จะถอนไปหารด้วย 1000 ด้วยเครื่องหมาย / ซึ่งจะได้ค่าเป็นจำนวนธนบัตรชนิด 1000	$w = 2800$ (จำนวนที่จะถอน) $b1000 = w / 1000$ $b1000 = 2800 / 1000 = 2$
2	นำค่าที่เหลือจากการหารด้วย 1000 โดยใช้เครื่องหมาย % เพื่อนำไปหารธนบัตรชนิด 500	$r = w \% 1000$ $r = 2800 \% 1000 = 800$
3	นำค่าที่ได้จากข้อ 2 ไปหารด้วย 500 โดยใช้เครื่องหมาย / เพื่อคำนวนหาจำนวนธนบัตรชนิด 500	$b500 = r / 500$ $b500 = 800 / 500 = 1$
4	คำนวนหาค่าที่เหลือจากการหารด้วย 500 โดยใช้เครื่องหมาย % เพื่อนำไปหารธนบัตรชนิด 100	$r = r \% 500$ $r = 800 \% 500 = 300$
5	นำค่าที่ได้จากข้อ 4 ไปหารด้วย 100 โดยใช้เครื่องหมาย / เพื่อคำนวนหาจำนวนธนบัตรชนิด 100	$b100 = r / 100$ $b100 = 300 / 100 = 3$
6	ดังนั้น ธนบัตรที่ได้คือ $b1000 = 2$ , $b500 = 1$ , $b100 = 3$	

```

#include <stdio.h>

void main() {
    int withdraw, remainder;
    int b100 = 0, b500 = 0, b1000 = 0;

    printf("\namount of money to withdraw >>10900");
    scanf("%d", &withdraw);

    if (withdraw > 20000) {
  
```

```

        b1000 = 10
        b500 = 1
        b100 = 4
  
```

```

        puts("can't withdraw more than 20000");
    } else if (withdraw % 100 != 0) {
        printf("amount to withdraw must be ");
        printf("the multiples of 100\n");
    } else {
        b1000 = withdraw / 1000;
        remainder = withdraw % 1000;

        b500 = remainder / 500;
        remainder %= 500;

        b100 = remainder / 100;
    }

    (b1000 > 0) ?
        printf("b1000 = %d\n", b1000) : printf("");

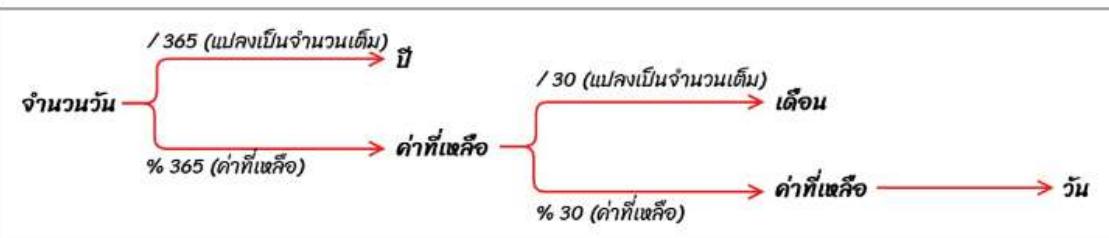
    (b500 > 0) ?
        printf("b500 = %d\n", b500) : printf("");

    (b100 > 0) ?
        printf("b100 = %d\n", b100) : printf("");
}

```

#### ตัวอย่าง 8-4 แปลงหน่วยจากจำนวนวันเป็น ปี เดือน วัน

ตัวอย่างนี้ ก็ใช้แนวทางเดียวกับตัวอย่างที่แล้ว แต่จะเป็นการแปลงหน่วยจากจำนวนวันไปเป็น ปี เดือน และวัน เช่น 500 วัน เท่ากับ 1 ปี 5 เดือน 15 วัน เป็นต้น โดยจำนวนวันจะใช้วิธีการสร้างเลขสุ่มให้ได้ค่าระหว่าง 100 - 1000 และเนื่องจากการแปลงหน่วยในลักษณะเช่นนี้ก็เป็นหลักการที่เราใช้งานมาหลายครั้ง ดังนั้น ขอให้ดูแนวทางจากภาพถัดไปได้เลย



```

#define _POSIX_C_SOURCE 200809L
#include <locale.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

```

```

total days: 996
year(s): 2
month(s): 8
days(s): 26

```

```

void main() {
    int total_days = 0, remainder;
    int y = 0, m = 0, d = 0;

    srand(time(0));
    rand();

    total_days = 100 + rand() % (1000 - 100 + 1);

    y = total_days / 365;
    remainder = total_days % 365;

    m = remainder / 30;
    remainder %= 30;

    d = remainder;

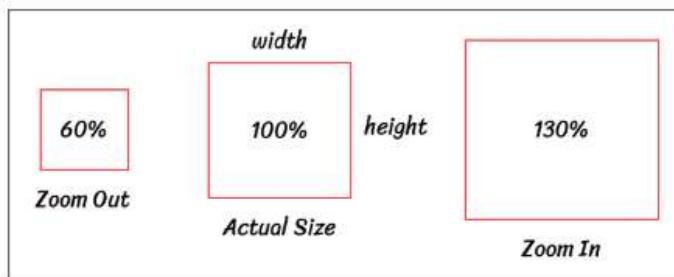
    setlocale(LC_ALL, "");
    printf("\ntotal days: %'d \n", total_days);

    (y > 0) ? printf("year(s): %d\n", y) : printf("");
    (m > 0) ? printf("month(s): %d\n", m) : printf("");
    (d > 0) ? printf("days(s): %d\n", d) : printf("");
}

```

#### ตัวอย่าง 8-5 Zoom In และ Zoom Out

การย่อและขยายขนาดของรูปภาพหรือที่เรียกว่าการซูม (Zoom) นั้น เราນักให้กำหนดค่าเป็น % ดังนี้



- Zoom = 100% คือขนาดจริงของภาพ (Actual Size)
- Zoom > 100% คือการขยายขนาดของภาพ (Zoom In) เช่น 130%
- Zoom < 100% คือการย่อขนาดของภาพ (Zoom Out) เช่น 60%

หากเราต้องการเปลี่ยนขนาดของภาพ ก็มีแนวทางโดยสังเขปดังนี้

1. นำค่า % ของการซูมมาคำนวนหาขนาดใหม่ เช่น ถ้าซูม 120% แสดงว่าขนาดเพิ่มขึ้น 20% ซึ่งมีค่าเป็น  $(0.2 \times \text{ขนาดเดิม})$  เช่น ถ้าความกว้างเดิมเป็น 200 แสดงว่าความกว้างจะเพิ่มขึ้น  $0.2 \times 200 = 40$  ส่วนความสูงก็ต้องทำเหมือนกัน
2. นำขนาดที่เพิ่มขึ้นไปบวกเพิ่มจากขนาดเดิม เช่น  $200 + 40$
3. กรณีการย่อขนาด ก็ใช้หลักการเดียวกัน เช่น ถ้าซูม 75% แสดงว่าขนาดลดลง 25% (หรือ -25%) เมื่อคำนวนขนาดที่ลดลงจะได้ค่าที่ติดลบ เช่น  $-0.25 \times 200 = -50$  หากนำไปบวกกับขนาดเดิม ก็จะได้ค่าลดลง เช่น  $200 + (-50) = 150$

ให้ตัวอย่างนี้ เราจะรับค่าความสูง ความกว้าง และ % การซูม ทางคีย์บอร์ด จากนั้นนำไปคำนวนหาขนาดใหม่ ดังหลักการที่กล่าวมาแล้ว

```
#include <stdio.h>

void main() {
    float w, h, ratio, dw, dh, z;

    printf("\nimage width >>");
    scanf("%f", &w);

    printf("image height >>");
    scanf("%f", &h);

    printf("zoom level (%) >>");
    scanf("%f", &z);

    //อัตราส่วนของการเปลี่ยนแปลง
    ratio = (100 - z) / 100;

    dw = w * ratio;      //ความกว้างที่เปลี่ยนแปลง
    dh = h * ratio;      //ความสูงที่เปลี่ยนแปลง
    w -= dw;              //ความกว้างหลังการเปลี่ยนขนาด
    h -= dh;              //ความสูงหลังการเปลี่ยนขนาด

    printf(
        "\nnew width: %g, new height: %g\n",
        w, h
    );
}
```

image width >>800  
 image height >>600  
 zoom level (%) >>75  
  
 new width: 600, new height: 450

### ตัวอย่าง 8-6 เทียบหน่วยระยะทาง

แสดงตารางเปรียบเทียบหน่วยระยะทางจากระหว่าง 1 - 10 กิโลเมตร เป็น เมตร หลา ฟุต และ ไมล์

<i>km</i>	<i>foot</i>	<i>yard</i>	<i>mile</i>
% 2 s	% 14 s	% 14 s	% 10 s
km	foot	yard	mile
-----			
1	3,280.84	1,093.61	0.62
2	6,561.68	2,187.22	1.24
	% ' 2 d % ' 14 . 2 f	% ' 14 . 2 f	% ' 10 . 2 f
	<i>km</i>	<i>foot</i>	<i>yard</i>
			<i>mile</i>

```
#define _POSIX_C_SOURCE 200809L
#include <locale.h>
#include <stdio.h>

void main() {
    // 1 km เท่ากับ
    float ft = 3280.84;           // ฟุต (foot)
    float yd = 1093.61;           // หลา (yard)
    float ml = 0.62;              // ไมล์ (mile)

    // ส่วนหัวตาราง
    printf(
        "\n%2s%14s%14s%10s",
        "km", "foot", "yard", "mile"
    );
    printf("\n-----");
    printf("-----");

    // ข้อมูลในตาราง
    setlocale(LC_ALL, "");
    for (int km = 1; km <= 10; km++) {
        printf(
            "\n%'2d%'14.2f%'14.2f%'10.2f",
            km, km*ft, km*yd, km*ml
        );
    }
    putchar('\n');
}
```

<i>km</i>	<i>foot</i>	<i>yard</i>	<i>mile</i>
1	3,280.84	1,093.61	0.62
2	6,561.68	2,187.22	1.24
3	9,842.52	3,280.83	1.86
4	13,123.36	4,374.44	2.48
5	16,404.20	5,468.05	3.10
6	19,685.04	6,561.66	3.72
7	22,965.88	7,655.27	4.34
8	26,246.72	8,748.88	4.96
9	29,527.56	9,842.49	5.58
10	32,808.40	10,936.10	6.20

### ตัวอย่าง 8-7 หน่วยที่เหมาะสมของขนาดไฟล์

ตัวอย่างนี้เป็นการเปลี่ยนหน่วยของขนาดไฟล์ให้เหมาะสม โดยรับข้อมูลเข้ามาในหน่วยไบต์ และเปรียบเทียบว่าสามารถเปลี่ยนเป็นหน่วยใหญ่ที่สุดหน่วยที่เป็นไปได้ โดยจำนวนไบต์ของแต่ละหน่วยเป็นดังตาราง

หน่วย	จำนวนไบต์
KB	1,024
MB	1,048,576
GB	1,073,741,824
TB	1,099,511,627,776

```
#define _POSIX_C_SOURCE 200809L
#include <locale.h>
#include <stdio.h>
#include <string.h>

void main() {
    long long size;
    float size2;
    char unit[10];

    setlocale(LC_ALL, "");

    printf("\nEnter file size (bytes) >>");
    scanf("%lld", &size);

    //ขนาด 1,099,511,627,776 ขึ้นไป แปลงเป็น TB
    if (size >= 91099511627776) {
        size2 = (float)size / 1099511627776;
        strcpy(unit, "TB");
    }

    //ถ้าขนาด 107,3741,824 ขึ้นไปแปลงเป็นหน่วย GB
    else if (size >= 1073741824) {
        size2 = (float)size / 1073741824;
        strcpy(unit, "GB");
    }

    //ถ้าขนาด 1,048,576 ขึ้นไป ให้แปลงเป็น MB
    else if (size >= 1048576) {
        size2 = (float)size / 1048576;
        strcpy(unit, "MB");
    }

    //ถ้าขนาด 1,024 ขึ้นไป ให้แปลงเป็น KB
    else if (size >= 1024) {
        size2 = (float)size / 1024;
    }
}
```

```
enter file size (bytes) >>3954972
3954972 bytes = 3.77 MB
```

```

        strcpy(unit, "KB");
    }

    //ถ้าขนาดน้อยกว่า 1,024 ให้หน่วยเป็น Byte เช่นเดิม
    else if (size < 1024) {
        strcpy(unit, "Byte(s)");
    }

    printf("\n%lld bytes = %'0.2f %s\n", size, size2, unit);
}

```

### ตัวอย่าง 8-8 อธิกสุรทิน (Leap Year)

เงื่อนไขของปีที่เป็นอธิกสุรทิน (Leap Year) ซึ่งเดือนกุมภาพันธ์มี 29 วัน ต้องตรงกับกรณีได้กรณีหนึ่ง ดังนี้ (ถ้าให้ตัวแปร year แทนปี ค.ศ. ที่ต้องการตรวจสอบ)

- กรณีที่ 1 ต้องหารด้วย 004 ลงตัว ซึ่งลักษณะเงื่อนไขนี้คือ

```
year % 400 == 0
```

- กรณีที่ 2 ต้องหารด้วย 001 ไม่ลงตัว แต่หารด้วย 4 ลงตัว จึงกำหนดเงื่อนไขเป็น

```
(year % 100 != 0) && (year % 4 == 0)
```

เงื่อนไขทั้งสองกรณีดังที่กล่าวมา หากเป็นจริงเพียงอันใดอันหนึ่ง ก็ถือว่าเป็นปีอธิกสุรทิน ดังนั้น จึงเชื่อมทั้งสองเงื่อนไขด้วย || เป็น

```
(year % 400 == 0) || ((year % 100 != 0) && (year % 4 == 0))
```

ในตัวอย่างนี้ เราจะตรวจสอบว่า ระหว่างปี 2020 - 2050 มีปีใดบ้างที่เป็นปีอธิกสุรทิน โดยใช้ลูป for

```

#include <stdio.h>
#include <stdbool.h>

void main() {
    short start = 2020;
    short end = 2050;
    bool is_first = true;

```

```
leap years between 2020 - 2050 are:
2020, 2024, 2028, 2032, 2036, 2040, 2044, 2048
```

```

printf("\nleap years between %d - %d are: \n", start, end);

for (int y = start; y <= end; y++) {
    if ((y % 400 == 0) || (y % 100 != 0 && y % 4 == 0)) {
        //ต้าไม่ใช่การพิมพ์อธิกสูตรทินเปแรก
        //ให้เขียนเครื่องหมาย , ไว้ข้างหน้า เพื่อคั่นกับปีก่อนนี้
        if (!is_first) {
            printf(", ");
        }
        printf("%d", y);
        is_first = false;
    }
}
putchar('\n');
}

```

### ตัวอย่าง 8-9 กราฟเบรี่ยนเทียบยอดขาย

ตัวอย่างนี้จะเป็นการแสดงกราฟในแนวอนเพื่อเบรี่ยนเทียบยอดขายลินค้าใน 7 วันที่ผ่านมา โดยสร้างเลขสุ่มเพื่อใช้แทนยอดขายลินค้าในแต่ละวัน ส่วนการวาดกราฟนั้น ให้ใช้อักษร \* แทนลินค้า 1 ชิ้นที่ขายได้ ส่วนหลักการเขียนโค้ดโดยลังเซปมีดังนี้

- วนลูปตามจำนวนวัน แล้วสร้างเลขสุ่มเพื่อใช้แทนยอดขายลินค้าของวันนั้นๆ
- ในแต่ละลูป ให้แสดงอักษร \* ตามยอดขายลินค้าของวันนั้นๆ
- การแสดงอักษร \* ขึ้นตามจำนวนที่กำหนด เรายังต้องวนลูปตามจำนวนอักษร \* นั้น แสดงว่า ในกรณีนี้ เราต้องใช้ลูป for ซ้อนกัน โดยลูปชั้นนอกจะวนรอบตามจำนวนวัน แล้วสร้างเลขสุ่มเพื่อใช้แทนยอดขายของวันนั้น และลูปชั้นในจะวนรอบเพื่อแสดงอักษร \* ตามจำนวนลินค้าที่ขายได้ในวันนั้นๆ

```

for (int d = 1; d <= 7; d++)
    for (int s = 1; s <= sales; s++)
        day 1 ***** (18)
        day 2 ***** (13)
        day 3 * (1)
        day 4 ***** (14)
        day 5 (0)
        day 6 ***** (6)
        day 7 ***** (11)

```

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void main() {
    srand(time(0));
    unsigned short sales;

    puts("\nproducts sold in past 7 days");

    for (int d = 1; d <= 7; d++) {
        printf("\nday %d", d);
        putchar('\t');

        sales = rand() % 21; //0 - 20

        //ถ้ายอดขายของวันนั้นเป็น 0
        //ให้แสดงแค่เลข 0 ในวงเล็บ แล้วข้ามไปวันถัดไป
        if (sales == 0) {
            printf("(0)");
            continue;
        }

        //วนลูปเพื่อแสดง * ตามจำนวนที่ขายได้
        for (int s = 1; s <= sales; s++) {
            printf("*");
        }

        //แสดงตัวเลขจำนวนที่ขายได้ในวงเล็บต่อท้ายกราฟ
        printf(" (%d)", sales);
    }

    putchar('\n');
}
```

products sold in past 7 days	
day 1	***** (6)
day 2	***** (9)
day 3	***** (16)
day 4	***** (12)
day 5	***** (20)
day 6	**** (4)
day 7	***** (9)

### ตัวอย่าง 8-10 ระดับ Rating

เป็นแนวทางการคำนวณระดับ Rating หรือจำนวนดาว โดยใช้หลักการดังนี้

- สมมติว่ามีผู้ให้ดาวจำนวนหนึ่ง ซึ่งเราต้องวนลูปเพื่อรับจำนวนดาวจากคนเหล่านั้น
- จำนวนดาวที่ถูกต้องจะอยู่ระหว่าง 1 - 5 ถ้าใส่เลขอื่นนอกเหนือจากนี้จะไม่อนุญาต
- นำจำนวนดาวทั้งหมดจากผู้ให้ทุกคนมารวมกันแล้วหารด้วย (จำนวนผู้ให้ดาว × 5) เพื่อให้เป็นค่าเฉลี่ยต่อคน (แต่ละคนจะให้ได้สูงสุด 5 ดาว)

- นำค่าเฉลี่ยที่ได้ไปเทียบด้วยหลักการดังนี้
  - ◎ ค่าเฉลี่ย ตั้งแต่ 0.8 ขึ้นไป ถือว่าได้ 5 ดาว
  - ◎ ค่าเฉลี่ย ตั้งแต่ 0.6 ขึ้นไป ถือว่าได้ 4 ดาว
  - ◎ ...
    - ◎ ค่าเฉลี่ย ตั้งแต่ 0.2 ขึ้นไป ถือว่าได้ 1 ดาว
    - ◎ ค่าเฉลี่ย ต่ำกว่า 0.2 จะไม่ได้ดาว
- แสดงผลโดยใช้อักษร \* ตามจำนวนดาวที่ได้

```
#include <stdio.h>

void main() {
    int score = 0;
    float person = 0;
    short rate;

    putchar('\n');

    //วนลูปตามจำนวนผู้ให้ดาว (ในที่นี้คือ 5 คน)
    for (int i = 1; i <= 5; i++) {
        //คนที่ i ให้กี่ดาว?
        printf("person #%d, num stars (1 - 5) >>", i);
        scanf("%d", &rate);

        //ตัดให้ดาวเป็นเลขระหว่าง 1 - 5 (ถูกต้อง)
        //ก็เพิ่มคะแนนสะสม และเพิ่มจำนวนผู้ให้ดาว (ได้ถูกต้อง)
        if (rate >= 1 && rate <= 5) {
            score += rate;
            person += 1;
        }
    }

    float avg = 0;
    short stars = 0;

    if (score > 0) {
        avg = score / (person * 5); //ค่าเฉลี่ย
    }

    if (avg >= 0.8) {
        stars = 5;
    } else if (avg >= 0.6) {
        stars = 4;
    }
}
```

```
person #1, num stars (1 - 5) >>3
person #2, num stars (1 - 5) >>2
person #3, num stars (1 - 5) >>4
person #4, num stars (1 - 5) >>5
person #5, num stars (1 - 5) >>4

rating: ****
```

```

} else if (avg >= 0.4) {
    stars = 3;
} else if (avg >= 0.2) {
    stars = 2;
} else if (avg > 0) {
    stars = 1;
}

if (stars > 0){
    printf("\nrating: ");

    //แสดง * ตามจำนวนดาวที่ได้
    for (int i = 1; i <= stars; i++) {
        printf("*");
    }
} else {
    printf("\nno rating");
}

putchar('\n');
}

```

### ตัวอย่าง 8-11 ดัชนีมวลกาย

การคำนวณหาค่าดัชนีมวลกาย (Body Mass Index: BMI) จะใช้สูตรดังนี้

$$\text{BMI} = \frac{\text{weight}}{(\text{height})^2}$$

โดยน้ำหนัก (weight) มีหน่วยเป็นกิโลกรัม และส่วนสูง (height) มีหน่วยเป็นเมตร ซึ่งช่วงการเปรียบเทียบของแหล่งข้อมูลแต่ละแห่งอาจมีความละเอียดแตกต่างกัน สำหรับในที่นี้จะเปรียบเทียบเป็นช่วงกว้าง ๆ ดังนี้

- BMI >= 40      Obese (อ้วนมาก)
- BMI >= 25      Over Weight (น้ำหนักเกิน)
- BMI >= 18.5    Normal Weight (น้ำหนักปกติ)
- BMI < 18.5     Under Weight (น้ำหนักต่ำกว่าปกติ)

ให้รับข้อมูลเป็นน้ำหนัก และส่วนสูง (รับค่าเป็น ช.ม.แล้วแปลงเป็นเมตร) จากนั้นนำไปคำนวณหาค่า BMI และเปรียบเทียบด้วย if ว่าค่า BMI ดังกล่าว มีลักษณะรูปร่างเป็นอย่างไร

```
#include <stdio.h>
#include <math.h>
#include <string.h>

void main() {
    float weight, height, bmi;
    char shape[50];

    printf("\nweight (kg) >>");
    scanf("%f", &weight);

    printf("height (cm) >>");
    scanf("%f", &height);

    height /= 100; //cm => m

    bmi = weight / pow(height, 2);

    printf("\nBMI: %g", bmi);

    if (bmi >= 30) {
        strcpy(shape, "obese");
    } else if (bmi >= 25) {
        strcpy(shape, "over weight");
    } else if (bmi >= 18.5) {
        strcpy(shape, "normal weight");
    } else if (bmi < 18.5) {
        strcpy(shape, "under weight");
    }

    printf("\nshape: %s \n", shape);
}
```

```
weight (kg) >>70
height (cm) >>170
BMI: 24.2215
shape: normal weight
```

### ตัวอย่าง 8-12 แสดงหมายเลขของสลากกินแบ่ง

หมายเลขของสลากกินแบ่งแต่ละใบจะประกอบด้วยเลข 6 หลัก และด้านล่างของเลขแต่ละตัวจะเป็นอักษรย่อ 3 ตัวของคำอ่านในภาษาอังกฤษของเลขตัวนั้น ดังภาพ



สำหรับในตัวอย่างนี้ เราจะเลียนแบบลักษณะดังกล่าว โดยแสดงหมายเลขทั้ง 6 หลักพร้อมคำอ่านภาษาอังกฤษแบบบอกรายอิเล็กทรอนิกส์ เอาไว้ที่ด้านล่างของเลขแต่ละตัว ซึ่งมีแนวทางพอลังเขปดังนี้

- เราจะเปลี่ยนการแสดงผลออกเป็น 2 บรรทัด โดยบรรทัดแรกจะใช้แสดงหมายเลขทั้ง 6 หลัก และบรรทัดที่สองจะใช้แสดงคำอ่านในภาษาอังกฤษ ซึ่งในแต่ละบรรทัดเราจะใช้ตัวแปรแบบสตริงเพื่อจัดเก็บข้อมูลแยกกัน
- เลขแต่ละหลักจะได้จากการสร้างเลขสุ่มที่มีค่าระหว่าง 0 - 9 จำนวน 6 หลัก แต่เนื่องจากในแต่ละหลักใช้วิธีการเดียวกัน ดังนั้น จึงใช้ลูป for เพื่อวนลูปจำนวน 6 ครั้ง ในการสร้างเลขสุ่มของแต่ละหลัก
- การวนลูปแต่ละรอบ เราจะสร้างเลขสุ่มระหว่าง 0 - 9 ขึ้นมา 1 ตัว และนำไปต่อท้ายตัวแปรสตริงที่ใช้เก็บข้อมูลบรรทัดแรก และเทียบตัวเลขนี้เป็นคำอ่านในภาษาอังกฤษ จากนั้นนำไปต่อท้ายตัวแปรสตริงที่ใช้เก็บข้อมูลบรรทัดที่สอง
- เมื่อวนลูปจนได้ตัวเลขครบทั้ง 6 หลักแล้ว ก็แสดงค่าของตัวแปรสตริงที่เก็บข้อมูลของบรรทัดแรกและบรรทัดที่สอง

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

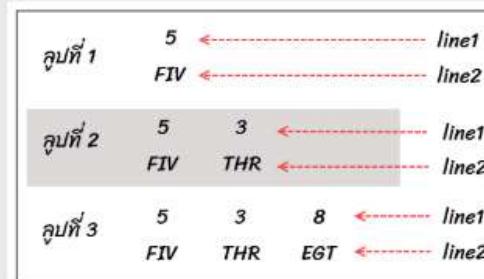
void main() {
    char line1[100] = "";
    char line2[100] = "";
    char str_digit[10];
    int x;

    srand(time(0));

    for (int i = 1; i <= 6; i++) {
        x = rand() % 10;
        sprintf(str_digit, " %d ", x);
        strcat(line1, str_digit);

        switch (x) {
            case 0: strcat(line2, " ZER "); break;
            case 1: strcat(line2, " ONE "); break;
            case 2: strcat(line2, " TWO "); break;
            case 3: strcat(line2, " THR "); break;
            case 4: strcat(line2, " FOR "); break;
            case 5: strcat(line2, " FIV "); break;
            case 6: strcat(line2, " SIX "); break;
            case 7: strcat(line2, " SEV "); break;
            case 8: strcat(line2, " EGT "); break;
        }
    }
}
```

4	8	0	9	3	5
FOR	EGT	ZER	NIN	THR	FIV



```

        case 9: strcat(line2, " NIN "); break;
    }
}

printf("\n%r\n%r\n", line1, line2);
}

```

### ตัวอย่าง 8-13 แยกจำนวนเป็นเลขโดดโดยใช้วิธีการหาร

ตัวอย่างนี้ เราจะรับเลขจำนวนเต็มจากทางคีย์บอร์ด และแยกตัวเลขออกมาเป็นเลขโดดที่ละตัว โดยมีข้อแม้ว่า ให้ใช้เพียงวิธีการหารตัวเลขทางคณิตศาสตร์เท่านั้น เช่น 6831 แยกเลขโดดได้เป็น 6, 8, 3, 1 ซึ่งหลักการแยกตัวเลขมีดังนี้

1	สมมติต้องการแยกเลข 123 ให้นำ 10 ไปหารแบบเอาเศษเศษ (Modulus) เราจะได้เลขตัวสุดท้ายมา	int x = 123; int a = x % 10; = 3
2	นำเลขที่ต้องการแยกมาหารด้วย 10 เพื่อเอาเศษจำนวนที่เหลือ ซึ่งก็คือการหารด้วย / เพื่อจะนำไปหารตัวถัดไป	x = x / 10; = 123 / 10; = 12
3	นำเลขที่ได้จากข้อ 2 ย้อนกลับไปทำขั้นตอนที่ 1 คือ %10 ได้จะ เลขท้ายมา	int b = x % 10; = 12 % 10; = 2
4	ทำเช่นนี้ไปเรื่อยๆ จนกว่าผลหารจะเป็น 0	

แนวคิดของตัวอย่างนี้ อาจค่อนข้างยากเล็กน้อย ทั้งนี้หากเราใช้วิธีการแบบสตริงหรือ อาร์เรย์ดังที่จะกล่าวถึงในบทต่อ ๆ ไปจะไม่ยุ่งยากแบบนี้ แต่อย่างให้ลองฝึกทักษะการคิดจากวิธีทางคณิตศาสตร์ไปก่อน

```

#include <stdio.h>
#include <stdbool.h>
#include <string.h>

void main() {
    int num, digit;
    char str_digit[100] = ""; //จัดเก็บเลขโดดแบบสตริง
    char str_separated_digits[100] = "";
    //ใช้จัดเก็บเลขโดดที่แยกออกจากแล้ว และเชื่อมต่อเป็นสตริง

    bool is_first = true;

    printf("\nEnter +integer number >>");

    fgets(str_digit, 100, stdin);
    str_separated_digits[0] = '\0';
    for (int i = 0; str_digit[i] != '\0'; i++) {
        if (str_digit[i] != ' ') {
            if (is_first) {
                str_separated_digits[0] = str_digit[i];
                is_first = false;
            } else {
                str_separated_digits[strlen(str_separated_digits)] = str_digit[i];
            }
        }
    }
    str_separated_digits[strlen(str_separated_digits)] = '\0';

    printf("separated digit(s): %s", str_separated_digits);
}

```

```

scanf("%d", &num);

int remainder = num;
do {
    digit = remainder % 10;

    //แปลงตัวเลขที่คัดแยกได้เป็นสตริง
    //ถ้าเป็นการแยกครั้งแรก (ขวาสุด) ไม่ต้องมี , ต่อท้าย
    //มีจะนั้น (ครั้งต่อๆ ไป) ให้มี , ต่อท้าย
    if (is_first) {
        sprintf(str_separated_digits, "%d", digit);
    } else {
        sprintf(str_separated_digits, "%d, ", digit);
    }

    //นำสตริงที่ได้จากการคัดแยกครั้งล่าสุด
    //มาวางต่อท้ายเลขที่คัดแยกครั้งก่อนนี้
    strcat(str_separated_digits, str_digit);

    //นำสตริงจัดเลขที่คัดแยกแล้วทั้งหมด
    //ไปเก็บแทนที่ค่าของตัวแปรใช้จัดเก็บการคัดแยกล่าสุด
    //เพื่อจะนำไปต่อท้ายเลขที่คัดแยกในลูปต่อไป
    strcpy(str_digit, str_separated_digits);

    //หลังการคัดแยกครั้งแรกผ่านไปแล้ว
    //เช็คค่าตัวแปรเป็น false เพื่อบ่งชี้ว่าครั้งต่อๆ ไปไม่ใช้ครั้งแรก
    is_first = false;

    remainder /= 10;
} while (remainder != 0);

printf("\nseparated digit(s): %s\n",
       str_separated_digits
);
}

```

#### ตัวอย่าง 8-14 แยกจำนวนเป็นเลขโดดพร้อมหาค่าสูงสุด-ต่ำสุด

จากตัวอย่างที่แล้ว เราจะรับเลขจำนวนเต็มบวกทางคีย์บอร์ด แล้วแยกตัวเลขออกมาเป็นเลขโดดทีละตัว ในตัวอย่างนี้ ให้เพิ่มเติมการตรวจสอบว่าเลขโดดตัวใดที่มีค่ามากที่สุด-น้อยที่สุด พร้อมกับผลรวมของเลขแต่ละตัวและค่าเฉลี่ย เช่น 6831 เลขโดดที่มีค่ามากสุดคือ 8 ผลรวมคือ 18 (เนื่องจากขั้นตอนการแยกเลขโดด ก็เหมือนกับตัวอย่างที่แล้ว ดังนั้น จึงไม่ขอจำรายละเอียดซ้ำอีก)

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

void main() {
    int num, digit;
    char str_digit[100] = "";
    char str_separated_ditgits[100] = "";
    bool is_first = true;
    int max, min, sum = 0;
    float average;

    printf("\nEnter +integer number >>");
    scanf("%d", &num);

    int remainder = num;
    int count = 0;

    while (remainder > 0) {
        digit = remainder % 10;

        if (is_first) {
            sprintf(str_digit, "%d", digit);
            min = digit;
            max = digit;
        } else {
            sprintf(str_digit, "%d, ", digit);
        }

        strcat(str_digit, str_separated_ditgits);
        strcpy(str_separated_ditgits, str_digit);

        min = (digit < min) ? digit : min;
        max = (digit > max) ? digit : max;
        sum += digit;

        is_first = false;
        count++;

        remainder /= 10;
    }

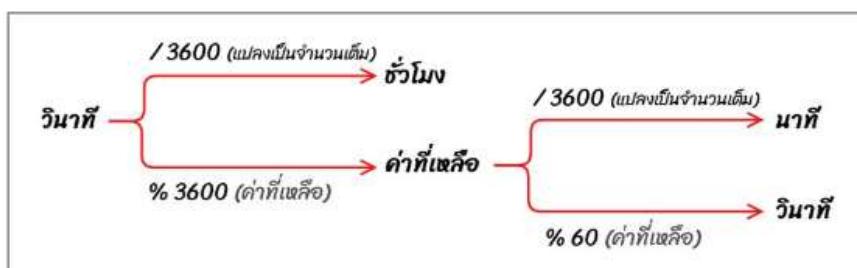
    printf("\ndigit(s): %s", str_separated_ditgits);
    printf("\nmin: %d", min);
    printf("\nmax: %d", max);
    printf("\nsum: %d", sum);
    printf("\naverage: %g\n", (float)sum/count);
}
```

```
enter +integer number >>190476
digit(s): 1, 9, 0, 4, 7, 6
min: 0
max: 9
sum: 27
average: 4.5
```

### ตัวอย่าง 8-15 ความแตกต่างระหว่างเวลา

ในตัวอย่างนี้เราจะคำนวณความแตกต่างระหว่าง 2 เวลา โดยรับค่าทางคีย์บอร์ดเป็น ชั่วโมง นาที วินาที และแปลงเวลาทั้ง 2 ค่าเป็นหน่วยวินาที จากนั้นเราก็สามารถหาค่าแตกต่างของ 2 เวลา ดังกล่าว ซึ่งจะได้ค่าในหน่วยวินาที แต่เรารู้แปลงจากวินาทีเป็นหน่วยที่เข้าใจง่ายกว่านั้นคือ ชั่วโมง นาที วินาที โดยใช้หลักการที่เคยทำมาแล้วในตัวอย่างที่ 5-4 (บทที่ 5) ดังแนวทางในภาพถัดไป

12:30:45 t1_sec = 12*3600 + 30*60 + 45	15:50:20 t2_sec = 15*3600 + 50*60 + 20
$\text{time\_dif\_sec} = \text{t2\_sec} - \text{t1\_sec}$	



```
#include <stdio.h>
#include <stdbool.h>
#include <math.h>

void main() {
    int h, m, s;

    printf("\nenter hour minute second for time#1");
    printf("\n(example: 20 45 39) >>");
    scanf("%d%d%d", &h, &m, &s);
    int t1_sec = (3600 * h) + (60 * m) + s;

    printf("\nenter hour minute second for time#2");
    printf("\n(example: 20 45 39) >>");
    scanf("%d%d%d", &h, &m, &s);
    int t2_sec = (3600 * h) + (60 * m) + s;

    int time_dif_sec = abs(t2_sec - t1_sec);

    h = time_dif_sec / 3600;
```

```
enter hour minute second for time#1
(example: 20 45 39) >>19 45 00

enter hour minute second for time#2
(example: 20 45 39) >>23 50 15

different time
4 hour(s)
5 minute(s)
15 second(s)
```

```

int remaining = time_dif_sec % 3600;
m = remaining / 60;

s = remaining % 60;

printf("\n不同时间");
(h > 0) ? printf("\n%d 小时(s)", h) : printf("");
(m > 0) ? printf("\n%d 分钟(s)", m) : printf("");
(s > 0) ? printf("\n%d 秒(s)", s) : printf("");

putchar('\n');
}

```

### ตัวอย่าง 8-16 ขอบเขตของข้อมูลเวลาที่เป็นไปได้

ตามปกตินั้น รูปแบบของเวลาคือ ชั่วโมง:นาที:วินาที โดยค่าชั่วโมงจะอยู่ระหว่าง 0 - 23 ส่วนนาทีและวินาทีจะอยู่ระหว่าง 0 - 59 ในตัวอย่างนี้ เราจะรับค่าชั่วโมง นาที และวินาที ทางคีย์บอร์ดพร้อมกันทั้ง 3 อย่าง และนำไปเก็บในตัวแปรแยกจากกัน เพื่อตรวจสอบว่า ค่าแต่ละอย่างอยู่ในขอบเขตที่เป็นไปได้หรือไม่

```

#include <stdio.h>
#include <stdbool.h>

void main() {
    int h, m, s;

    printf("\nEnter hour minute second");
    printf("\n(example: 20 45 39) >>");
    scanf("%d%d%d", &h, &m, &s);

    //ชั่วโมง ต้องอยู่ระหว่าง 0 - 23
    //นาทีและวินาที ต้องอยู่ระหว่าง 0 - 59
    bool valid_h = (h >= 0 && h <= 23);
    bool valid_m = (m >= 0 && m <= 59);
    bool valid_s = (s >= 0 && s <= 59);

    //ถ้าค่าทั้งหมดอยู่ในช่วงที่เป็นไปได้
    if (valid_h && valid_m && valid_s) {
        //จัดรูปแบบโดยหากเป็นเลขหลักเดียว ให้เติม 0 ข้างหน้า
        printf("\nTime is %02d:%02d:%02d \n", h, m, s);
    } else {
        puts("\ninvalid time");
    }
}

```

enter hour minute second

(example: 20 45 39) >>1 2 60

invalid time

enter hour minute second

(example: 20 45 39) >>9 10 7

time is 09:10:07

**ตัวอย่าง 8-17 ขอบเขตของข้อมูลวัน เดือน ปี ที่เป็นไปได้**

ขอบเขตข้อมูลวันเดือนปี คือ วันที่จะอยู่ระหว่าง 1 - 31 (แต่ต้องพิจารณาเดือนและปีประกอบด้วย) ลำดับของเดือนจะอยู่ระหว่าง 1 - 12 ส่วนปี ค.ศ. ถ้าพิจารณาตามข้อกำหนดในภาษาคอมพิวเตอร์อื่นๆ ส่วนใหญ่จะต้องอยู่ระหว่าง 1 - 9999 ในตัวอย่างนี้ เราจะรับค่าวันที่เดือน ปี ทางคีย์บอร์ดพร้อมกันทั้ง 3 อย่าง และนำไปเก็บในตัวแปรแยกจากกัน เพื่อตรวจสอบว่าค่าแต่ละอย่างอยู่ในขอบเขตที่เป็นไปได้หรือไม่ (หลักการตรวจสอบปีอธิกสุริยัน ให้ดูจากตัวอย่าง 8-8)

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

void main() {
    int d, m, y, days_in_month;
    char err[100];

    printf("\nEnter day month year (ce)\n");
    printf("(example: 19 7 2023) >>");
    scanf("%d%d%d", &d, &m, &y);

    //ถ้าเป็นเดือนกุมภาพันธ์
    if (m==2) {
        //ถ้าตรงกับปีอธิกสุริยัน (leap year) จะมีได้ 29 วัน
        //มิฉะนั้น จะมีได้เพียง 28 วัน
        if ((y%400==0) || (y%100!=0 && y%4==0)) {
            days_in_month = 29;
        } else {
            days_in_month = 28;
        }
    }

    //ถ้าเป็นเดือนที่ 4, 6, 9, 11
    //จะมีได้ไม่เกิน 30 วัน
    else if (m==4 || m==6 || m==9 || m==11) {
        days_in_month = 30;
    }

    //ถ้าเป็นเดือนอื่นๆ จะมีได้ไม่เกิน 31 วัน
    else {
        days_in_month = 31;
    }

    //ช่วงวันที่ เดือน ปี ที่เป็นไปได้
    bool valid_d = (d >= 1) && (d <= days_in_month);
```

enter day month year (ce)  
 (example: 19 7 2023) >>31 13 2023  
 invalid month

enter day month year (ce)  
 (example: 19 7 2023) >>29 2 2023  
 invalid day

```

bool valid_m = (m >= 1) && (m <= 12);
bool valid_y = (y >= 1) && (y <= 9999);

if (valid_d && valid_m && valid_y) {
    //จัดรูปแบบโดยหากเป็นเลขหลักเดียว ให้เติม 0 ข้างหน้า
    printf("date is %02d/%02d/%02d \n", d, m, y);
} else if (!valid_d) {
    puts("\ninvalid day");
} else if (!valid_m) {
    puts("\ninvalid month");
} else if (!valid_y) {
    puts("\ninvalid year");
}
}

```

### ตัวอย่าง 8-18 น้ำหนักและจำนวนผู้โดยสารในลิฟต์

สมมติว่าลิฟต์ตัวหนึ่งรับน้ำหนักร่วมได้ไม่เกิน 600 กิโลกรัม และรองรับจำนวนผู้โดยสารได้ไม่เกิน 8 คน ในตัวอย่างนี้ เราจะลองเขียนโค้ดเพื่อตรวจสอบน้ำหนักและจำนวนผู้โดยสารในลิฟต์ดังนี้

- ให้วนลูป เพื่อรับค่าน้ำหนักของผู้โดยสารที่ละคนทางคีย์บอร์ด
- เมื่อรับค่าน้ำหนักของผู้โดยสารคนใดเข้ามา ให้นำไปรวมกับน้ำหนักทั้งหมดของผู้โดยสารคนก่อนๆ ทั้งหมด หากเกินค่าน้ำหนักสูงสุดที่รองรับได้ (ในที่นี่คือ 600 กก.) จะไม่ยอมรับผู้โดยสารคนล่าสุด โดยวนลูปถัดไปเพื่อรับผู้โดยสารคนอื่นเข้ามาแทน
- นอกจากการตรวจสอบน้ำหนักแล้ว เรายังต้องมีตัวนับเพื่อตรวจสอบจำนวนผู้โดยสารที่เข้ามาในลิฟต์ เนื่องจากเรามีข้อกำหนดว่าให้รองรับได้ไม่เกิน 8 คน ดังนั้น ถ้าผู้โดยสารครบ 8 คนแล้วก็จะไม่ยอมรับคนถัดไปเข้ามาในลิฟต์อีก

```

#include <stdio.h>

void main() {
    //psg = passenger
    const float MAX_LOAD = 600; //น้ำหนักร่วมสูงสุด
    const int MAX_PSG = 8;       //จำนวนผู้โดยสารรวมสูงสุด

    float weight, total = 0;     //น.n.แต่ละคน และรวมทั้งหมด
    int count_psg = 0;           //นับจำนวนผู้โดยสาร
}

```

```

to cancel, enter weight >>0
passenger 1 weight (kg) >>75
passenger 2 weight (kg) >>80
passenger 3 weight (kg) >>65
passenger 4 weight (kg) >>70
passenger 5 weight (kg) >>65
passenger 6 weight (kg) >>55
passenger 7 weight (kg) >>50
passenger 8 weight (kg) >>60

passengers: 8
load: 520

```

```

puts("\nTo cancel, enter weight >>0\n");

//เงื่อนไข น้ำหนักรวมและจำนวนผู้โดยสารทั้งหมด
//ต้องไม่เกินค่าที่กำหนดไว้
while (total < MAX_LOAD &&
       count_psg < MAX_PSG)
{
    count_psg++;

    //รับค่าน้ำหนักของผู้โดยสารคนที่ ? (ตามลำดับของลูป)
    printf("passenger %d weight (kg) >>", count_psg);
    scanf("%f", &weight);

    if (weight < 0) {
        continue;
    }
    //ถ้ารวมน้ำหนักผู้โดยสารคนล่าสุดแล้วเกินค่าที่กำหนด
    //ให้คุณล่าสุดออกไป และรับคนอื่นเข้ามาแทน
    //โดยไม่ต้องรวมข้อมูลของคนล่าสุด
    else if ((total + weight) > MAX_LOAD) {
        printf("overload! (%g kgs.)", total + weight);
        printf("\npassenger %d out\n", count_psg);

        count_psg--;
        continue;
    } else if (weight == 0) {
        count_psg--;
        break;
    }
    total += weight;
}

printf("\n passengers: %d", count_psg);
printf("\n load: %g\n", total);
}

```

### ตัวอย่าง 8-19 ตาราง ASCII (1)

ในบทที่ 3 เราได้รู้จักกับตาราง ASCII กันมาแล้ว ซึ่งอักษรแต่ละตัวจะมีรหัสของมันเอง ทั้งนี้หากเรากำหนดเลขรหัสให้แก่ฟังก์ชัน printf() มันก็จะแสดงอักษรตัวนั้นออกมา เช่น

```
printf("%c", 65);    จะแสดงตัว A
```

โดยในตัวอย่างนี้เราจะลองสร้างตาราง ASCII ของอักษรบางส่วน ดังในตารางถัดไป

รหัส	อักษร	รหัส	อักษร	รหัส	อักษร
32	ช่องว่าง	65	A	97	a
33	!	66	B	98	b
34	"	67	C	99	c
...	...	...	...	...	...
56	8	89	Y	121	y
57	9	90	Z	122	z

เนื่องจากรหัสของอักษรจะมีค่าเป็นตัวเลขที่ต่อเนื่องกัน ดังนั้น เราสามารถใช้ลูปเพื่อวนซ้ำตามช่วงของตัวเลขได้ แต่ปัญหาคือ เราต้องแสดงผลทีละบรรทัด (ແຕວ) และไม่สามารถกำหนดให้แสดงทีละคอลัมน์ได้ ซึ่งแนวทางการแก้ปัญหาคือ ในแต่ละบรรทัด เราต้องแสดงให้ครบทั้ง 6 คอลัมน์ โดยแต่ละคอลัมน์เราจะให้มีค่าเริ่มต้น (รหัส) ที่แตกต่างกัน ดังนั้น เราต้องใช้ตัวแปรซึ่งเป็นตัวนับของแต่ละคอลัมน์แยกกัน สำหรับในที่นี้จะมีจำนวนແຕວเท่ากับ 26 แต่เพื่อให้ครบอักษร A - Z โดยลูปที่ใช้คือ for แบบกำหนด 3 ตัวนับ (ดูรายละเอียดเพิ่มเติมในบทที่ 7) ส่วนการแสดงผลจะทำเป็นแบบง่าย ๆ ไปก่อน ซึ่งการเขียนโค้ดเป็นแบบล้วน ๆ แต่อาจต้องใช้เวลาทำความเข้าใจบ้างเล็กน้อย

```
#include <stdio.h>

void main() {
    //ตัวนับเพื่อกำหนดขอบเขตให้แสดง 26 ແຕວ
    int count = 1;

    putchar('\n');

    //ตัวนับของลูป for จะให้ 3 ตัวสำหรับ 3 กลุ่มคอลัมน์
    //i เริ่มจากรหัส 32
    //j เริ่มจากรหัส 65 (A)
    //k เริ่มจากรหัส 97 (a)
    for (int i = 32, j = 65, k = 97;
        count <= 26;
        i++, j++, k++)
    {
        //รหัส อักษร | รหัส อักษร | รหัส อักษร (1 ແຕວ)
        printf(
            "%c%c%c | %c%c%c | %c%c%c\n",
            i, j, k, i, j, k, i, j, k);
        count++;
    }
}
```

```

        "%d %c | %d %c | %d %c\n",
        i, i, j, j, k, k
    );

    count++;
}
}

```

32		65	A	97	a
33	!	66	B	98	b
34	"	67	C	99	c
35	#	68	D	100	d
36	\$	69	E	101	e
37	%	70	F	102	f
38	&	71	G	103	g
39	'	72	H	104	h
40	(	73	I	105	i
41	)	74	J	106	j
42	*	75	K	107	k
43	+	76	L	108	l
44	,	77	M	109	m

45	-	78	N	110	n
46	.	79	O	111	o
47	/	80	P	112	p
48	0	81	Q	113	q
49	1	82	R	114	r
50	2	83	S	115	s
51	3	84	T	116	t
52	4	85	U	117	u
53	5	86	V	118	v
54	6	87	W	119	w
55	7	88	X	120	x
56	8	89	Y	121	y
57	9	90	Z	122	z

### ตัวอย่าง 8-20 ตาราง ASCII (2)

ในตัวอย่างที่แล้ว เราแสดงตาราง ASCII เป็นตารางในแบบง่ายๆ สำหรับในตัวอย่างนี้ เราจะแสดงเป็นตารางที่จัดรูปแบบให้สวยงามยิ่งขึ้น โดยใช้เทคนิคการกำหนดความกว้างของแต่ละคอลัมน์ดังที่เราได้เรียนรู้มาแล้วในบทที่ 4 ส่วนขั้นตอนเกี่ยวกับรหัสและอักษรจะเหมือนกับตัวอย่างที่แล้วทั้งหมด เพียงเปลี่ยนมาจัดรูปแบบการแสดงผลเป็นตารางเท่านั้นเอง

code	char	code	char	code	char	code	char
% 4 s \ t % 4 s \ t \ t % 4 s \ t % 4 s \ t \ t % 4 s \ t % 4 s							
code	char	code	char	code	char	code	char
-----							
32		65	A	97	a		
33	!	66	B	98	b		
% 4 d \ t % - 4 c \ t \ t % 4 d \ t % - 4 c \ t \ t % 4 d \ t % - 4 c							
code	char	code	char	code	char	code	char

} แม่ตาราง

} ข้อมูล

```

#include <stdio.h>

void main() {
    int count = 1;

    // ส่วนหัวตาราง
    putchar('\n');
    printf(

```

```

"%4s\t%4s\t\t%4s\t%4s\t\t%4s\t%4s\n",
"code", "char", "code", "char", "code", "char"
);
printf("-----");
printf("-----\n");

//ส่วนข้อมูลในตาราง
for (int i = 32, j = 65, k = 97;
     count <= 26;
     i++, j++, k++)
{
    printf(
        "%4d\t%-4c\t\t%4d\t%-4c\t\t%4d\t%-4c\n",
        i, i, j, j, k, k
    );
    count++;
}
}

```

code	char	code	char	code	char
32		65	A	97	a
33	!	66	B	98	b
34	"	67	C	99	c
35	#	68	D	100	d
36	\$	69	E	101	e
37	%	70	F	102	f
38	&	71	G	103	g
39	'	72	H	104	h
40	(	73	I	105	i
41	)	74	J	106	j
42	*	75	K	107	k
43	+	76	L	108	l
44	,	77	M	109	m
45	-	78	N	110	n
46	.	79	O	111	o
47	/	80	P	112	p
48	0	81	Q	113	q
49	1	82	R	114	r
50	2	83	S	115	s
51	3	84	T	116	t
52	4	85	U	117	u
53	5	86	V	118	v
54	6	87	W	119	w
55	7	88	X	120	x
56	8	89	Y	121	y
57	9	90	Z	122	z

ลิ๊งที่ได้กล่าวถึงในบทนี้ เป็นการรวมรวมตัวอย่างโดยใช้ความรู้พื้นฐานจากบทก่อนๆ มารวมเข้าด้วยกัน ซึ่งนอกจากจะเป็นการทบทวนเนื้อหาสำคัญให้เกิดความเข้าใจมากยิ่งขึ้นแล้ว ยังช่วยเสริมทักษะการเขียนโปรแกรมให้ก้าวหน้าไปอย่างรวดเร็วอีกด้วย

# 9

## การสร้างและใช้งานฟังก์ชัน

ถ้าเราพัฒนาโปรแกรมจนมีขนาดใหญ่และซับซ้อนมากขึ้น ก็อาจต้องแบ่งขั้นตอนการกระทำบางอย่างออกเป็นส่วนย่อยๆ ให้มีหน้าที่อย่างใดอย่างหนึ่ง และสามารถเรียกขึ้นมาทำงานได้ทันทีที่ต้องการโดยไม่ต้องเขียนโค้ดซ้ำซ้อนกันอีก ซึ่งส่วนของโปรแกรมย่อยๆ ดังกล่าวเราระบุว่าเป็น **ฟังก์ชัน (Function)** โดยเนื้อหาในบทนี้ เราจะได้เรียนรู้ทั้งการสร้างและเรียกใช้ฟังก์ชันรวมถึงข้อกำหนดสำคัญต่างๆ ที่เกี่ยวข้องทั้งหมด

### ลักษณะของฟังก์ชัน

ในภาษา C จะแบ่งฟังก์ชันออกเป็น 2 ประเภทหลักๆ คือ

- **Pre-defined function** (หรือ **Built-in function**) ซึ่งฟังก์ชันในกลุ่มนี้จะถูกสร้างไว้ล่วงหน้าแล้วโดยผู้พัฒนาภาษา C และเป็นส่วนหนึ่งของตัวภาษา เราจึงเรียกใช้งานได้ทันที ซึ่งก็คือฟังก์ชันต่างๆ ที่เราเคยใช้ในบทต่างๆ ที่ผ่านมานั่นเอง เช่น printf(), scanf(), puts(), strcpy() เป็นต้น โดยบางฟังก์ชันอาจจัดเก็บในเอกสารแยกต่างหาก ซึ่งเราต้องระบุตำแหน่งอ้างอิงด้วยคำสั่ง include จึงจะใช้ฟังก์ชันนั้นได้ ตามที่เราได้เรียนรู้มาแล้ว
- **User-defined function** คือฟังก์ชันที่ผู้เขียนโปรแกรมเป็นผู้สร้างขึ้นมาใช้งานตามวัตถุประสงค์ของตนเอง

เนื่องจาก Pre-defined function เป็นลิสต์ที่มีอยู่แล้วในภาษา C ซึ่งเราจะได้ศึกษาเพิ่มเติมในหัวข้อที่เกี่ยวข้องของแต่ละบทกันไปเรื่อยๆ โดยลิสต์ที่จะกล่าวถึงในบทนี้ก็คือ User-defined Function ซึ่งเราต้องสร้างขึ้นมาเองเพื่อกำหนดการกระทำการอย่างใดอย่างหนึ่ง เช่น การคำนวณหาผลลัพธ์ การแสดงผล หรืออ่านค่าบางอย่าง เป็นต้น โดยลักษณะที่เราจะนำมาสร้างฟังก์ชันมักเป็นลิสต์ที่ต้องซ้ำหลายๆ ครั้งในรูปแบบเดียวกัน เช่น สมมติว่าเราต้องการหาผลรวมของช่วงตัวเลขต่างๆ แล้วแสดงผลลัพธ์ ซึ่งตามปกติอาจจะต้องเขียนโค้ดในลักษณะดังนี้

```

int begin = 1, end = 10, sum = 0;
for (int x = begin; x <= end; x++) {
    sum += x;
}
printf("\nsum of %d to %d = %d", begin, end, sum);

-----
begin = 10;
end = 20;
sum = 0;
for (int x = begin; x <= end; x++) {
    sum += x;
}
printf("\nsum of %d to %d = %d", begin, end, sum);

-----
begin = 30;
end = 50;
sum = 0;
for (int x = begin; x <= end; x++) {
    sum += x;
}
printf("\nsum of %d to %d = %d", begin, end, sum);

```

sum of 1 to 10 = 55  
 sum of 10 to 20 = 165  
 sum of 30 to 50 = 840

จากโค้ดจะพบว่า เราต้องเขียนโค้ดในลักษณะเดิมๆ ซ้ำหลายครั้ง จึงเป็นข้อบกพร่องที่ควรจะแก้ไข โดยนำโค้ดส่วนนั้นมาสร้างเป็นฟังก์ชันที่สามารถเรียกใช้งานจากตำแหน่งต่างๆ ได้ตามต้องการ ดังแนวทางต่อไปนี้ (ให้ดูแค่หลักการก่อน ส่วนรายละเอียดจะกล่าวถึงในหัวข้อต่อๆ ไป)

```

void sum_range(int begin, int end) {
    int sum = 0;
    for (int x = begin; x <= end; x++) {
        sum += x;
    }
    printf("\nsum of %d to %d = %d", begin, end, sum);
}

void main() {
    show_range(1, 10);
    show_range(20, 30);
    show_range(30, 50);
}

```

จากโค้ด `sum_range()` คือฟังก์ชันที่เราสร้างขึ้นเพื่อลดขั้นตอนที่ต้องทำซ้ำกัน โดยเมื่อต้องการใช้งานก็เพียงแค่ลุ่งช่วงข้อมูลที่ต้องการคำนวนไปให้มัน และสามารถเรียกใช้งานซ้ำ

กีครั้งก็ได้ ดังนั้น ฟังก์ชันจึงเป็นองค์ประกอบที่สำคัญอย่างยิ่งของการเขียนโปรแกรมซึ่งต้องใช้งานกันอยู่ตลอด โดยเราจะแบ่งฟังก์ชันออกเป็น 2 ลักษณะ คือ

- ฟังก์ชันแบบไม่ส่งค่ากลับ ฟังก์ชันแบบนี้จะกำหนดเพียงการกระทำเท่านั้น แต่ไม่ส่งค่าใดๆ ออกไปจากฟังก์ชัน
- ฟังก์ชันแบบส่งค่ากลับ ฟังก์ชันแบบนี้นอกจากจะใช้กำหนดการกระทำแล้ว เมื่อได้ผลลัพธ์ออกมา เราจะส่งค่าที่ได้กลับไปให้ส่วนที่เรียกใช้ฟังก์ชันนั้นด้วย

ข้อกำหนดปลีกย่อยเกี่ยวกับฟังก์ชันยังมีอีกหลายกรณี ซึ่งจะกล่าวรายละเอียดในลำดับต่อไป

## พารามิเตอร์และอาร์กิวเม้นต์

พารามิเตอร์ (Parameter) คือข้อมูลที่รับจากภายนอก เพื่อนำเข้ามาใช้งานในฟังก์ชัน ซึ่งผลลัพธ์ที่ได้ก็จะเปลี่ยนไปตามค่าที่กำหนด ดังนั้น พารามิเตอร์จึงช่วยให้ฟังก์ชันมีความยืดหยุ่นในการทำงานตามข้อมูลที่ได้รับเข้ามา โดยหลักการเพิ่มเติมเกี่ยวกับพารามิเตอร์มีดังนี้

- เนื่องจากพารามิเตอร์คือตัวแปรอีกแบบหนึ่ง ดังนั้น การกำหนดชื่อพารามิเตอร์ก็ใช้หลักการเดียวกับการตั้งชื่อตัวแปร
- ถ้ามีพารามิเตอร์มากกว่า 1 ตัว ให้คั่นแต่ละตัวด้วยเครื่องหมาย ,
- ต้องระบุชนิดข้อมูลของพารามิเตอร์แยกแต่ละตัว จะใช้ชนิดข้อมูลร่วมกันเหมือนกับการประกาศตัวแปรไม่ได้

```
void func1(int a) {      //พารามิเตอร์ชนิด int ชื่อ a
    ...
}

//มีพารามิเตอร์ 3 ตัวคือ num, str และ is_first
void func2(float num, char str[100], bool is_first) {
    ...
}

//ต้องระบุชนิดข้อมูลของพารามิเตอร์แต่ละตัวแยกจากกัน
//จะกำหนดรวมแบบการประกาศตัวแปรไม่ได้
//แม้ชนิดข้อมูลจะเหมือนกันก็ตาม
//เช่น ฟังก์ชันต่อไปนี้ จะเกิดข้อผิดพลาด
void func3(int n1, n2, n3) {          //Error
    ...
}
//ที่ถูกต้องคือ func3(int n1, int n2, int n3)
```

- พารามิเตอร์ถือเป็นตัวแปรของฟังก์ชัน จึงสามารถใช้งานในฟังก์ชันได้เลย เช่น

```
void print_text(char text[100]){
    puts(text);
}

void show_info(char name[50], char country[50]) {
    printf(
        "My name is %s from %s",
        name, country
    );
}
```

- ตามหลักการแล้ว รูปแบบการรับข้อมูลซึ่งกำหนดไว้ที่ฟังก์ชัน จะเรียกว่า **Formal Parameter** หรือเรียกสั้นๆ ว่า พารามิเตอร์ (**Parameter**) ส่วนข้อมูลที่ส่งให้แก่ฟังก์ชัน เรียกว่า **Actual Parameter** หรือเรียกอีกอย่างว่า อาร์กิวเมนต์ (**Argument**) จึงอาจกล่าวได้ว่า อาร์กิวเมนต์ก็คือค่าของพารามิเตอร์นั้นเอง ดังนั้น ทั้งพารามิเตอร์และอาร์กิวเมนต์ก็หมายถึงสิ่งเดียวกัน เพียงแต้อ้างถึงคนละลักษณะเท่านั้น

```
float max(float num1, float num2) {
    ...
    Formal Parameter
}

Actual Parameter
void main() {
    (Argument)
    float m = max(10.5, 30);
}
```

อย่างไรก็ตาม ข้อแตกต่างระหว่างคำว่า พารามิเตอร์และอาร์กิวเมนต์ ไม่ใช่ลิ้งสำคัญอะไรมากนัก ซึ่งในแหล่งข้อมูลบางแห่งอาจใช้เพียงคำใดคำหนึ่งใน เช่น ใช้คำว่า พารามิเตอร์เพียงคำเดียว หรือใช้คำว่าอาร์กิวเมนต์เพียงคำเดียว เพื่ออ้างถึงทั้งพารามิเตอร์แบบ formal และ actual

## ฟังก์ชันแบบไม่ส่งค่ากลับ

สำหรับฟังก์ชันแบบไม่ส่งค่ากลับ (void function) จะไม่คืนค่าใด ๆ ไปยังส่วนที่เรียกใช้มัน ซึ่งรูปแบบโดยทั่วไปของฟังก์ชันประเภทนี้คือ

```
void ชื่อฟังก์ชัน(พารามิเตอร์) {
    คำสั่งต่างๆ
}
```

- **void** เป็นคีย์เวิร์ดที่ต้องเขียนกำกับเอาไว้หากไม่มีการส่งข้อมูลกลับออกจากฟังก์ชันนั้น
- ชื่อฟังก์ชัน มีหลักเกณฑ์คล้ายกับการตั้งชื่อตัวแปร โดยในภาษา C เรา尼ยมเขียนชื่อฟังก์ชันด้วยตัวพิมพ์เล็ก เช่น `display`, `calculate` และถ้าเป็นการนำหลายๆ คำมารวมกัน ก็ควรใช้ underscore () เพื่อคั่นระหว่างคำ จะดูชัดเจนกว่าการเขียนติดกัน เช่น `show_message`, `print_data`, `is_in_range`, `random_int` เป็นต้น นอกจากนี้ยังสามารถนำตัวเลขมาใช้ในการตั้งชื่อฟังก์ชันได้ เช่นเดียวกับชื่อตัวแปร
- **พารามิเตอร์ (Parameter)** เป็นข้อมูลที่จะนำเข้ามาใช้ในฟังก์ชัน แต่อาจไม่มีก็ได้ ทั้งนี้ก็ขึ้นกับความจำเป็นของงานที่จะทำในแต่ละฟังก์ชัน ซึ่งจะกล่าวรายละเอียดในภายหลัง
- คำสั่งต่างๆ ใช้สำหรับกำหนดการกระทำการในฟังก์ชันนั้นๆ ซึ่งจะมีกี่คำสั่งก็ได้ ทั้งนี้เราจะใช้วงล้อ {} ในการกำหนดขอบเขตหรือบล็อกของคำสั่ง เช่นเดียวกับการกำหนดเงื่อนไขหรือลูป
- ในไฟล์เดียวกัน (\*.c) เราจะสร้างฟังก์ชันขึ้นมาใช้งานเองจำนวนกี่ฟังก์ชันก็ได้

## การสร้างฟังก์ชันแบบไม่ส่งค่ากลับ

ลักษณะของฟังก์ชันแบบไม่ส่งค่ากลับคือ เราจะดำเนินการให้เสร็จลื้นตามวัตถุประสงค์ภายในฟังก์ชันนั้น โดยไม่ส่งผลลัพธ์หรือค่าใดๆ กลับออกไปนอกฟังก์ชัน เช่น

```
//ฟังก์ชันแบบไม่พารามิเตอร์
void say_hello(){
    puts("hello");
}

//ฟังก์ชันแบบมีพารามิเตอร์
void show_text(char text[]) {
    puts(text);
}

//ฟังก์ชันแบบมีพารามิเตอร์
void sum_range(int begin, int end) {
    int sum = 0;
    for (int x = begin; x < end; x++) {
        sum += x;
    }
    printf("sum of %d to %d = %d", begin, end, sum);
}
```

## ฟังก์ชัน main() และตำแหน่งการเขียนฟังก์ชันที่สร้างเอง

ในภาษา C นั้น จะกำหนดจุดเริ่มต้นในการทำงานด้วยฟังก์ชันที่ชื่อ main() ดังที่เราได้ทำกันมาตลอดนั้นเอง และ main() ก็จะเป็นฟังก์ชันที่เราต้องสร้างขึ้นมาเองเช่นเดียวกัน อย่างไรก็ตาม main() อาจเป็นได้ทั้งฟังก์ชันแบบไม่ส่งค่ากลับและแบบส่งค่ากลับ นอกจากนี้ก็อาจมีหรือไม่มีพารามิเตอร์ส่งเข้ามาในฟังก์ชันก็ได้ แต่สำหรับการใช้งานที่ผ่านมาก็มีหนึ่ง main() แบบไม่ส่งค่ากลับ (มี void นำหน้า) โดยไม่มีการรับพารามิเตอร์ใดๆ ซึ่งเรียบจะใช้รูปแบบนี้เป็นหลัก และลิ่งที่เราต้องพิจารณาในลำดับต่อไปคือ ถ้าเราสร้างฟังก์ชันอื่นๆ เพิ่มเข้ามา จะเขียนฟังก์ชันเหล่านั้นไว้ที่ตำแหน่งใด เนื่องจากในภาษา C นั้น ตำแหน่งการเขียนฟังก์ชันมีผลต่อการเรียกใช้ฟังก์ชันเหล่านั้นด้วย ดังที่จะกล่าวถึงในหัวข้อต่อๆ ไป ซึ่งเราสามารถเลือกใช้วิธีใดวิธีหนึ่งในการวางแผนตำแหน่งฟังก์ชันดังนี้

### วิธีที่ 1 : ให้ main() อยู่ในลำดับสุดท้าย

วิธีนี้ให้เราเขียนฟังก์ชันที่เราสร้างขึ้นมาเองเอาไว้ก่อนหรือด้านบน และค่อยเขียน main() ไว้ในลำดับสุดท้าย เช่น

```
#include <stdio.h>

void func1() {
    ...
}

void func2(int x) {
    ...
}

void main() { //วาง main ไว้ในลำดับสุดท้าย
    ...
}
```

วิธีนี้ ภายในฟังก์ชัน main() สามารถมองเห็นหรือเข้าถึงฟังก์ชันที่อยู่ก่อนมันได้โดยอัตโนมัติ เพราะการประมวลผลของโค้ด ตามปกติจะทำการบันลงล่าง ดังนั้น ก่อนจะมาถึง main() มันก็จะผ่านและรู้จักกับ func1() และ func2() มา ก่อนแล้ว อย่างไรก็ตาม ข้อเสียของวิธีนี้คือ หากฟังก์ชันที่เราสร้างเองมีเป็นจำนวนมาก การนำ main() ซึ่งเป็นจุดเริ่มต้นการทำงานไปวางไว้ในลำดับสุดท้ายอาจจะยุ่งยากต่อการค้นหา เพื่อเพิ่มเติมหรือแก้ไขในภายหลัง ดังนั้น วิธีการวาง main() ไว้ในลำดับสุดท้าย น่าจะเหมาะสมกับกรณีที่มีจำนวนฟังก์ชันที่สร้างเองไม่มากนัก

## วิธีที่ 2: ให้ main() อยู่ในลำดับแรก

ถ้าเราไม่สะดวกที่จะวาง main() ไว้ในลำดับสุดท้าย หรือตามหลังฟังก์ชันอื่นๆ ที่สร้างเอง ก็สามารถวาง main() เอาไว้ก่อนฟังก์ชันเหล่านั้นได้ อย่างไรก็ตาม กรณีนี้ ภายใน main() จะมองไม่เห็นหรือไม่สามารถเข้าถึงฟังก์ชันที่ตามหลังหรืออยู่ตัดจากมันได้ เพราะ main() จะถูกประมวลผลก่อน จึงยังไม่รู้จักกับฟังก์ชันที่อยู่ตัดจากมันลงมา แต่เราสามารถแก้ไขปัญหานี้ได้โดยนำชื่อฟังก์ชันเหล่านั้น มาระบุไว้ก่อน main() เพื่อให้ตัวประมวลผลรับทราบไว้ล่วงหน้า ว่าเราจะอ้างถึงหรือเรียกใช้งานใน main() หรือเรียกลักษณะดังกล่าวว่า prototype และส่วนที่เป็นตัวฟังก์ชันนั้นจริงๆ ก็นำไปเขียนตัดจาก main() ได้เลย เช่น

```
#include <stdio.h>

//กำหนด prototype
//โดยประกาศชื่อฟังก์ชันพร้อมชนิดข้อมูลส่งกลับและพารามิเตอร์ (ถ้ามี)
//ให้ตัวประมวลผลรับทราบไว้ล่วงหน้า (ต้องเขียนไว้ก่อน main)
void func1();
void func2(int x);

void main() { //วาง main ไว้ในลำดับแรกตามปกติ
    ...
}

//เขียนฟังก์ชันที่สร้างเองและระบุ prototype เอาไว้แล้ว
//ให้อยู่ตัดจาก main() ได้เลย
void func1() {
    ...
}

void func2(int x) {
    ...
}
```

อย่างไรก็ตาม การประกาศชื่อฟังก์ชันไว้ก่อน main() อาจมีลักษณะปลีกย่อยที่เราต้องพิจารณาเพิ่มเติมอีกบางส่วน ซึ่งจะกล่าวถึงในภายหลัง

## การเรียกใช้ฟังก์ชันแบบไม่ส่งค่ากลับ

ถึงแม้เราจะสร้างฟังก์ชันขึ้นมาแล้ว แต่คำสั่งในฟังก์ชันดังกล่าว จะไม่ถูกดำเนินการใดๆ หากเราไม่เรียกมันขึ้นมาทำงาน ซึ่งมีหลักการในเบื้องต้นคือ

- สร้างฟังก์ชันขึ้นมาก่อนที่จะเรียกมันขึ้นมาทำงาน โดยพิจารณาการวางแผนทำงานของฟังก์ชัน เทียบกับ main() ตามที่กล่าวไปในหัวข้อที่แล้ว
- การเรียกฟังก์ชันขึ้นมาทำงาน ให้ระบุชื่อฟังก์ชันเป้าหมายแล้วตามด้วยวงเล็บและค่าอาร์กิวเม้นต์ (ถ้ามี) ณ จุดที่ต้องการใช้งาน
- ฟังก์ชันที่ถูกเรียก จะทำงานไปจนลิ้นสุดคำสั่งที่กำหนดในล็อกของมัน แล้วก็จะกลับมาทำคำสั่งที่อยู่ถัดจากตำแหน่งที่เรียกฟังก์ชันขึ้นมาทำงานต่อไปเรื่อยๆ
- เราสามารถเรียกฟังก์ชันเดิมๆ ซ้ำๆ ได้

**ตัวอย่าง 9-1** การสร้างและเรียกฟังก์ชันที่สร้างขึ้นมาเองจาก main() โดยวางฟังก์ชัน main() ไว้ในลำดับสุดท้าย

```
#include <stdio.h>

void say_hello() {
    puts("hello");
}

void say_anything(char msg[100]) {
    puts(msg);
}

void main() {
    say_hello();
    say_anything("hi");
}
```

**ตัวอย่าง 9-2** การสร้างและเรียกฟังก์ชันที่สร้างขึ้นมาเองจาก main() โดยวางฟังก์ชัน main() ไว้ในลำดับแรก

```
#include <stdio.h>

//prototypes
void say_hello();
void say_anything(char msg[100]);

void main() {
```

```

        sayAnything("hi");
        sayHello();
    }

void sayHello() {
    puts("hello");
}

void sayAnything(char msg[100]) {
    puts(msg);
}

```

อย่างไรก็ตาม ไม่จำเป็นว่าฟังก์ชันที่เราสร้างขึ้นมาหนึ่ง จะต้องถูกเรียกจาก main() เสมอไป แต่อาจถูกเรียกจากฟังก์ชันอื่นๆ ที่ไม่ใช่ main() ก็ได้ เช่นด้วยอย่างต่อไปนี้

**ตัวอย่าง 9-3** การสร้างและเรียกใช้ฟังก์ชันต่อเนื่องกัน

```

#include <stdio.h>

//prototypes
void sayHello();
void saySawasdee();
void sayGoodbye();

void main() {
    sayHello();
}

void sayHello() {
    puts("hello");
    saySawasdee();
}

void saySawasdee() {
    puts("sawasdee");
    sayGoodbye();
}

void sayGoodbye() {
    puts("goodbye");
}

```

hello  
 sawasdee  
 goodbye

## ฟังก์ชันแบบส่งค่ากลับ

โดยทั่วไปแล้ว แต่ละฟังก์ชันที่สร้างขึ้น ก็เพื่อวัตถุประสงค์อย่างใดอย่างหนึ่ง ซึ่งอาจมีผลลัพธ์หรือข้อมูลบางอย่างที่เกิดจากการทำงานของฟังก์ชัน ทั้งนี้หากเราไม่จัดการกับผลลัพธ์ที่ได้ภายในฟังก์ชันนั้นโดยตรง ก็สามารถส่งค่าดังกล่าวกลับออกมายังส่วนที่เรียกฟังก์ชัน เพื่อนำไปใช้งานอื่นๆ ตามที่ต้องการ ซึ่งรูปแบบทั่วไปของฟังก์ชันที่ส่งค่ากลับมีดังนี้

```
ชนิดข้อมูลที่จะส่งกลับ ชื่อฟังก์ชัน(พารามิเตอร์) {
    คำสั่งต่างๆ ภายในฟังก์ชัน
    ...
    return ข้อมูลที่จะส่งกลับ
}
```

- ชนิดข้อมูลที่จะส่งกลับ ให้ระบุชนิดข้อมูลที่จะส่งออกจากฟังก์ชันนี้ เช่นเดียวกับที่เราใช้ในการประกาศตัวแปร เช่น int, float, char (สำหรับชนิดสตริง จะกล่าวถึงในบทที่ 11)
- ชื่อฟังก์ชัน และ พารามิเตอร์ จะเหมือนกับฟังก์ชันแบบไม่ส่งค่ากลับ จึงไม่ขอ拿来รายละเอียดมากกล่าวถึงอีก
- return** เป็นคีย์เวิร์ดสำหรับส่งข้อมูลจากฟังก์ชันออกไปยังส่วนที่เรียกมันขึ้นมาทำงาน

### การสร้างฟังก์ชันแบบส่งค่ากลับ

ลักษณะการสร้างฟังก์ชันแบบส่งค่ากลับชนิดต่างๆ มีแนวทางดังต่อไปนี้ (กรณีการส่งค่ากลับเป็นชนิดสตริง จะกล่าวถึงในบทที่ 11)

```
//แนวทางการสร้างฟังก์ชันที่ส่งผลลัพธ์กลับเป็นชนิดตัวเลข (จำนวนเต็ม)
int get_int() {
    int result;
    printf("enter integer num >>");
    scanf("%d", &result);
    return result;           //ส่งผลลัพธ์กลับออกไป
}
```

```
//แนวทางการสร้างฟังก์ชันที่ส่งผลลัพธ์กลับเป็นชนิดอักขระ
char select_menu() {
    char chr;
    printf("enter character for menu");
    print("\n(y for yes, n for no) >>");
    scanf("%c", &chr);
    return chr;
}
```

```
// แนวทางการสร้างฟังก์ชันที่ส่งผลลัพธ์กลับเป็นชนิดบูลีน (แบบที่ 1)
```

```
#include <stdbool.h>
...
bool is_positive(int num) {
    bool r;
    if (num > 0) {
        r = true;
    } else {
        r = false;
    }
    return r;
}
```

```
// แนวทางการสร้างฟังก์ชันที่ส่งผลลัพธ์กลับเป็นชนิดบูลีน (แบบที่ 2)
```

```
_Bool is_negative(int num) {
    _Bool r;
    if (num < 0) {
        r = 1;
    } else {
        r = 0;
    }
    return r;
}
```

เราต้องวาง `return` ไว้เป็นคำสั่งสุดท้ายของฟังก์ชัน เพราะหลังจากส่งค่ากลับออกไปแล้ว คำสั่งที่อยู่ต่อจากนั้นจะไม่มีผลใดๆ เลย เช่น โค้ดต่อไปนี้คำสั่ง `printf()` ที่อยู่ถัดจาก `return` จะไม่มีการถูกประมวลผล แต่ไม่ถือว่าเป็นข้อผิดพลาด ดังนั้น จึงรันโปรแกรมได้ตามปกติ เช่น (โค้ดเพียงบางส่วน)

```
int rand_1_100() {
    srand(time(0));
    rand();
    int r = 1 + rand() % (100 - 1 + 1);
    return r;

    printf("random num: %d", r);
    // คำสั่งบรรทัดนี้ ไม่ถูกประมวลผล เพราะอยู่หลัง return
}
```

แต่ในการส่งผลลัพธ์ที่ขึ้นอยู่กับเงื่อนไข ก็ให้กำหนดคำสั่ง `return` ไว้ในบล็อกของ `if` หรือ `else` หรือ `else if` หรือ `switch/case` ก็ได้ เช่น ฟังก์ชันต่อไปนี้ ซึ่งหากตัวเลขนั้นหารด้วย 2 ลงตัว และว่าเป็นเลขคู่ จะส่งตัว e (even) กลับไป มิฉะนั้นจะส่งตัว o (odd) กลับไป

```
char odd_even(int num) {
    if (num % 2 == 0) {
        return 'e';
    } else {
        return 'o';
    }
}
```



### หมายเหตุ

กรณีที่เราต้องการส่งผลลัพธ์กลับเป็นข้อมูลชนิดสตริง ต้องนำความรู้เรื่องอาร์เรย์และพอยน์เตอร์มาใช้ร่วมด้วย ซึ่งรายละเอียดของเรื่องนี้อยู่ในบทที่ 11 ดังนั้น จึงขอให้มีการส่งค่าแบบสตริงกลับจากฟังก์ชันไปกล่าวถึงในบทนั้น (หัวข้อ: การใช้พอยน์เตอร์ร่วมกับฟังก์ชัน) สำหรับในบทนี้ เราจะเรียนรู้แค่การส่งค่ากลับเป็นชนิดตัวเลขและอักษรไปก่อน

## การเรียกฟังก์ชันแบบส่งค่ากลับ

ถ้าฟังก์ชันไม่มีผลลัพธ์กลับคืนมา หากเราต้องการนำข้อมูลไปใช้งานอีก ๆ ต้องกำหนดตัวแปรเพื่อรับค่าดังกล่าว โดยชนิดข้อมูลของตัวแปรดังกล่าว ต้องสอดคล้องกับชนิดที่ส่งกลับจากฟังก์ชัน หรืออาจเรียกฟังก์ชัน ณ ตำแหน่งที่จะใช้ข้อมูลนั้นเลยก็ได้ ดังแนวทางต่าง ๆ ต่อไปนี้

```
//สมมติว่ามีฟังก์ชันชื่อ get_int() พร้อมพารามิเตอร์ msg
//สำหรับแสดงผลเพื่อบอกให้รู้ว่าต้องใส่ข้อมูลเกี่ยวกับอะไร
int get_int(char msg[100]) {
    int r;
    printf(msg);
    scanf("%d", &r);
    return r;
}

//แนวทางเรียกฟังก์ชัน get_int() ที่มีการส่งค่าชนิด int กลับมา
//โดยเราสามารถดักการกับผลลัพธ์ในลักษณะต่าง ๆ ได้ทั้งหลายแบบ
void main() {
    //เก็บค่าที่ส่งกลับมาไว้ในตัวแปร
    int quantity = get_int("product quantity >>");

    //นำไปแสดงผลทันที โดยไม่พักในตัวแปร
    printf("you entered: %d",
        get_int("enter number >>"));
}
```

```
//นำค่ามาบวกผล โดยไม่พักในตัวแปร
int sum = get_int("num 1 >>") + get_int("num 2 >>");

//นำค่ามาเปรียบเทียบทันที โดยไม่พักในตัวแปร
if (get_int("enter price >>") <= 0) {
    puts("price must be greater than 0");
}
}
```

**ตัวอย่าง 9-4** การสร้างฟังก์ชันเพื่อตรวจสอบตัวเลขชนิด float ว่าอยู่ในช่วงที่ต้องการ หรือไม่ โดยจะคืนค่าเป็นข้อมูลชนิดบูลลิน

```
#include <stdio.h>
#include <stdbool.h>

//prototype
bool is_in_range(float min, float max, float value);

void main() {
    putchar('\n');
    if (is_in_range(1, 10, 5.5)) {
        printf("5.5 is in range 1-10\n");
    }

    if (!is_in_range(1, 10, -1)) {
        printf("-1 isn't in range 1-10\n");
    }
}

bool is_in_range(float min, float max, float value) {
    if (min <= value && value <= max) {
        return true;
    } else {
        return false;
    }
/*
return (min <= value && value <= max);
*/
}
```

5.5 is in range 1-10  
 -1 isn't in range 1-10

**ตัวอย่าง 9-5** การสร้างเลขสุ่มที่สามารถกำหนดค่าในช่วงที่ต้องการได้นั้น เราต้องจัดทำสูตรให้ได้ แต่เพื่อลดความยุ่งยากดังกล่าว เราอาจนำมาสร้างเป็นฟังก์ชันที่สามารถระบุช่วงตัวเลขที่ต้องการได้ จากนั้นเมื่อต้องการใช้เลขสุ่มก็แค่เรียกฟังก์ชันนี้ขึ้นมาทำงานพร้อมระบุช่วงตัวเลข เป็นพารามิเตอร์

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int rand_range(int min, int max) {
    if (min < 0 || max < 0 || max < min) {
        return -1;
    }
    srand(time(0));
    rand();
    return min + rand() % (max - min + 1);
}

void main() {
    putchar('\n');

    int r1 = rand_range(1, 10);
    printf("#1 rand_range(1, 10): %d\n", r1);

    printf(
        "#2 rand_range(50, 100): %d\n",
        rand_range(50, 100)
    );

    //ถ้ากำหนดพารามิเตอร์ไม่ตรงตามเงื่อนไข
    int r3 = rand_range(100, 0);
    printf("#3 rand_range(100, 10): %d\n", r3);

    printf("#4 rand_range(1, 20) ");
    if (rand_range(1, 20) % 2 == 0) {
        printf("result is even\n");
    } else {
        printf("result is odd\n");
    }

    //การเรียกฟังก์ชันในแบบ ternary operator
    printf("#5 rand_range(10, 1) ");
    (rand_range(10, 1) >= 0) ?

```

```
#1 rand_range(1, 10): 7
#2 rand_range(50, 100): 52
#3 rand_range(100, 10): -1
#4 rand_range(1, 20) result is odd
#5 rand_range(10, 1) parameters incorrect
```

```

    printf("parameters ok") :
    printf("parameters incorrect");

    putchar('\n');
}

```

**ตัวอย่าง 9-6** เป็นการสร้างฟังก์ชันสำหรับการคำนวณย่อๆ ที่เกี่ยวข้องกับรูปทรงกรวยของหั้งพื้นที่วงกลมและเส้นรอบวง ซึ่งรายละเอียดเกี่ยวกับการคำนวณ ขอให้ดูจากคำอธิบายในโค้ด

```

#include <stdio.h>

float circle_area(float radius);
float circle_circumference(float radius);
float cyliner_surface(float radius, float height);
float cylinder_volume(float radius, float height);
float pi = 3.141;

void main() {
    float radius, height;
    printf("\nradius of cylinder >>");
    scanf("%f", &radius);

    printf("height of cylinder >>");
    scanf("%f", &height);

    float sf = cyliner_surface(radius, height);
    float vol = cylinder_volume(radius, height);

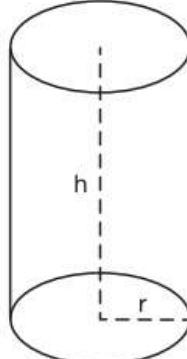
    printf("\ncylinder surface: %g", sf);
    printf("\ncylinder volume: %g\n", vol);
}

float circle_area(float radius) {
    return pi * (radius * radius);
}

float circle_circumference(float radius) {
    return 2 * pi * radius;
}

float cyliner_surface(float radius, float height) {
    // = side surface + top surface + bottom surface
    // = (circie_circumference * height) + (2 * circle_area)
}

```



radius of cylinder >>5  
height of cylinder >>15  
  
cylinder surface: 1334.93  
cylinder volume: 1177.88

```

    float side_sf = circle_area(radius) * height;
    float top_btm_sf = 2 * circle_area(radius);
    return side_sf + top_btm_sf;
}

float cylinder_volume(float radius, float height) {
    //= bottom area * height
    //= circle_area() * height
    return circle_area(radius) * height;
}

```

## ตัวแปรแบบ global และ local

ตัวแปรที่ประกาศอยู่นอกฟังก์ชันได้ จะเรียกว่าตัวแปรแบบ global ซึ่งส่วนใหญ่จะเป็นตัวแปรสำหรับใช้งานร่วมกันระหว่างฟังก์ชัน ส่วนตัวแปรที่ประกาศไว้ในฟังก์ชัน ถือเป็นตัวแปรแบบ local ซึ่งลิ่งที่เราต้องคำนึงถึงก็คือ ขอบเขตการอ้างถึงตัวแปร (scope of variable) ทั้งจากในและนอกฟังก์ชัน โดยมีหลักการพิจารณาดังนี้

- ภายในฟังก์ชัน เราสามารถอ้างถึงตัวแปรแบบ global ที่ประกาศอยู่นอกฟังก์ชันได้ แต่ ต้องประกาศตัวแปรนั้นไว้ก่อนฟังก์ชันที่จะใช้งาน ถ้าเราเปลี่ยนค่าตัวแปรแบบ global ที่ฟังก์ชันได ก็จะส่งผลไปถึงการใช้งานในฟังก์ชันอื่นๆ ทั้งหมด ( เพราะอ้างอิงค่าเดียวกัน ) เช่น

```

float pi = 3.141;           //ต้องประกาศไว้ก่อนฟังก์ชัน
                           //และอยู่ก่อนฟังก์ชันทั้งหมดที่จะใช้ตัวแปรนี้

```

```

float circle_area(float radius) {
    float area = pi * (10 * 10);
    return area;
}

float circle_circumference(float radius) {
    return 2 * pi * radius;
}

```

```

int n = 99;

void func1() {
    n++;
}

```

```

void func2() {
    printf("%d", n); //100 เพราะที่ main()
                    //เราเรียก func1() ก่อน func2()
                    //ค่าจึงถูกเพิ่มจากเดิมมาก่อนแล้ว
}

void main(){
    func1();
    func2();
}

```

- ตัวแปรที่ประกาศขึ้นภายในฟังก์ชันใดๆ จะเป็นตัวแปรแบบ local ที่ใช้งานได้เฉพาะในฟังก์ชันนั้นๆ แต่ไม่สามารถอ้างถึงจากฟังก์ชันอื่นๆ ได้ เช่น

```

void func1() {
    int x = 99;           //ตัวแปรแบบ local
}

void func2() {
    printf("%d", x);   //Error อ้างถึงตัวแปรแบบ local
                    //ในฟังก์ชันอื่นไม่ได้
}

```

- ภายในฟังก์ชัน ถ้าเราตั้งชื่อตัวแปรข้ามกับตัวแปรแบบ global ก็สมேกของการสร้างตัวแปรขึ้นมาใหม่ ซึ่งผลที่ได้คือ ตัวแปรนั้นจะถูกจัดเป็นแบบ local แต่ไม่ส่งผลต่อตัวแปร global เดิม และฟังก์ชันอื่นๆ ที่เรียกใช้ตัวแปรนั้น

```

int x = 10;           //ตัวแปรแบบ global

void func1() {
    int x = 99;           //การสร้างตัวแปรที่มีชื่อซ้ำกับ global
                        //ถือเป็นการสร้างตัวแปร local

    printf("%d", x);   //99 จะได้ค่าของตัวแปรแบบ local
}

void func2() {
    printf("%d", x);   //10 การเปลี่ยนค่าตัวแปร x ใน func1()
                        //ไม่มีผลต่อ func2()
}

```

## ตัวแปรแบบ static

ตามปกติแล้ว ค่าของตัวแปรแบบ local ที่ประกาศขึ้นในฟังก์ชันใด ๆ ก็จะมีผลเฉพาะช่วงเวลาที่เราเรียกใช้ฟังก์ชันดังกล่าวเท่านั้น และเมื่อฟังก์ชันเสร็จลินการทำงาน ค่าของตัวแปรก็จะถูกยกเลิกไป แล้วเมื่อเรียกใช้ฟังก์ชันนั้นอีกครั้ง ตัวแปรและค่าของมันก็จะถูกกำหนดขึ้นมาใหม่เป็นเช่นนี้เรื่อยไป อย่างไรก็ตาม หากเราต้องการนำค่าของตัวแปรแบบ local ที่ประกาศขึ้นในฟังก์ชัน กลับมาใช้งานอีกครั้งเมื่อเรียกฟังก์ชันนี้ในครั้งต่อไป ก็สามารถทำได้ เช่นกัน โดยสร้างเป็นตัวแปรแบบสแตติก ด้วยการวางคีย์เวิร์ด static ไว้หน้าชื่อตัวแปร

**static** ชนิดข้อมูล ชื่อตัวแปร

เช่น static int x หรือ static float y = 1.23 เป็นต้น อย่างไรก็ตาม ค่าของตัวแปรแบบสแตติก จะมีผลแค่ในระหว่างรันโปรแกรมเท่านั้น ไม่ได้จดเก็บไว้ตลอด หลังจากจบการทำงานไปแล้ว ค่าของตัวแปรก็จะถูกยกเลิกไป ซึ่งขอให้เราดูเพิ่มเติมจากตัวอย่างต่อไปนี้

**ตัวอย่าง 9-7** เปรียบเทียบระหว่างตัวแปรแบบสแตติก และแบบที่ไม่ใช่สแตติก (non-static) ที่ถูกสร้างไว้ในฟังก์ชัน โดยเราจะเพิ่มค่าตัวแปรขึ้นทีละ 1 เพื่อดูว่ามันสามารถนำค่าเดิมกลับมาใช้ใหม่ได้หรือไม่

```
#include <stdio.h>

int increment1();
int increment2();

void main() {
    for (int i = 1; i <= 5; i++) {
        printf(
            "\nloop %d non-static value = %d",
            i, increment1()
        );

        printf(
            "\nloop %d static value = %d",
            i, increment2()
        );

        putchar('\n');
    }
}
```

```

int increment1() {
    int value = 0;
    return ++value;
}

int increment2() {
    static int value = 0;
    return ++value;
}

```

```

loop 1 non-static value = 1
loop 1 static value = 1

loop 2 non-static value = 1
loop 2 static value = 2

loop 3 non-static value = 1
loop 3 static value = 3

loop 4 non-static value = 1
loop 4 static value = 4

loop 5 non-static value = 1
loop 5 static value = 5

```

จากผลลัพธ์จะเห็นว่า ตัวแปรแบบที่ไม่ใช่สแตติก ค่าจะคงเดิมไปตลอด เพราะถูกสร้างใหม่ทุกครั้งที่เรียกใช้งาน แต่ตัวแปรแบบสแตติก ค่าจะเพิ่มขึ้นไปเรื่อยๆ เพราะสามารถนำค่าเดิมกลับมาใช้ได้นั่นเอง

## การเรียกฟังก์ชันแบบ Recursion

นอกจากการที่ฟังก์ชันใดๆ จะไปเรียกฟังก์ชันอื่นๆ ขึ้นมาทำงานแล้ว ก็ยังสามารถเรียกตัวมันเองขึ้นมาทำงานได้เช่นกัน ซึ่งก็ทำเช่นเดียวกับการเรียกฟังก์ชันอื่น เพียงแต่เป็นการเรียกซึ่งของมันเท่านั้น ดังแนวทางต่อไปนี้

```

void repeat() {
    ...
    repeat();
}

```

การที่ฟังก์ชันเรียกตัวมันเองขึ้นมาทำงานแบบซ้ำๆ เราเรียกว่า Recursion ความจริงแล้วเราสามารถนำวิธีการแบบ Recursion ไปประยุกต์ใช้งานได้อย่างหลายแบบ แต่สำหรับในที่นี้ จะเน้นให้เรารู้จักการนำวิธีการแบบ Recursion มาเพื่อใช้เรียกฟังก์ชันตัวมันเองเพื่อการรับข้อมูลจากผู้ใช้งานกว่าจะใส่ค่าได้ตรงตามเงื่อนไขที่ต้องการ แต่ก่อนอื่น ขอให้ลองพิจารณาการเรียกฟังก์ชันแบบไม่ใช้วิธี Recursion ดังนี้

```

// โค้ดนี้ ยังไม่ใช่ Recursion

int scan_int() {
    int x;
    printf("enter int >>");
    scanf("%d", &x);
    return x;
}

```

```

}

void f1() {
    int a = scan_int();
    //ถ้าผลลัพธ์ไม่ตรงตามเงื่อนไข
    if (a < 1 || a > 10) {
        puts("error! (please enter 1-10)");
        a = scan_int(1, 10);
    } else {
        puts("thanks!");
    }
}

void f2() {
    int a = scan_int();
    //ถ้าผลลัพธ์ไม่ตรงตามเงื่อนไข
    if (a < 10 || a > 20) {
        puts("error! (please enter 10-20)");
        a = scan_int(10, 20);
    } else {
        puts("thanks!");
    }
}

```

จากโค้ด เราให้ฟังก์ชัน `scan_int()` ส่งผลลัพธ์กลับมาทันที ไม่ว่าจะใส่ข้อมูลเป็นอย่างไร ก็ตาม แล้วมาตรวจสอบที่ฟังก์ชันซึ่งเป็นผู้เรียกใช้งาน และถ้ามีหลายฟังก์ชันที่จำเป็นต้องเรียก `scan_int()` เพื่อรับข้อมูลเหมือนๆ กัน การตรวจสอบก็ต้องทำซ้ำๆ ในทุกฟังก์ชัน เช่นใน `f1()` และ `f2()` ดังโค้ดที่ผ่านมา จึงเป็นการทำงานที่ซ้ำซ้อนซึ่งเราควรแก้ไขใหม่ โดยกำหนดวิธีการตรวจสอบไว้ใน `scan_int()` โดยตรง ถ้าไม่อยู่ในเงื่อนไขที่ต้องการ ก็เรียกตัวมันเองขึ้นมาทำงานแบบ Recursion ดังตัวอย่างต่อไปนี้

**ตัวอย่าง 9-8** เป็นการนำโค้ดที่แล้ว มาแก้ไขโดยให้ฟังก์ชัน รับพารามิเตอร์เป็นช่วงข้อมูลที่ต้องการ และหากข้อมูลที่รับเข้ามาไม่อยู่ในช่วงนี้ ก็จะเรียกตัวมันเองขึ้นมาทำงานแบบ Recursion

```

#include <stdio.h>

int scan_int(char msg[], int min, int max);
void f1();
void f2();

void main() {

```

```

enter 1 - 10 >>11
enter 1 - 10 >>-1
enter 1 - 10 >>5
in f1() value = 5

enter 10 - 20 >>0
enter 10 - 20 >>-9
enter 10 - 20 >>13
in f2() value = 13

```

```

f1();
f2();
}

int scan_int(char msg[], int min, int max) {
    int x;
    printf(msg);
    scanf("%d", &x);
    if (min <= x && x <= max) {
        return x;
    } else {
        scan_int(msg, min, max);
    }
}

void f1() {
    putchar('\n');
    int x = scan_int("enter 1 - 10 >>", 1, 10);
    printf("in f1() value = %d\n", x);
}

void f2() {
    putchar('\n');
    int x = scan_int("enter 10 - 20 >>", 10, 20);
    printf("in f2() value = %d\n", x);
}

```

**ตัวอย่าง 9-9** ตัวอย่างนี้เราจะนำความรู้จากหลาย ๆ เรื่องที่เราศึกษาไปในบทนี้มาใช้ร่วมกัน โดยสมมติว่ามีเงินคงเหลือในบัญชีจำนวนหนึ่ง ซึ่งเราจะสร้างฟังก์ชันต่าง ๆ ดังนี้

- ฟังก์ชัน `select_menu()` สำหรับเลือกเมนู ถ้าเลือกรายการซึ่งไม่อยู่ในช่วงที่ต้องการให้เรียกตัวมันเองขึ้นมาทำงานแบบ Recursion เพื่อเลือกใหม่จนกว่าจะถูกต้อง
- ฟังก์ชัน `deposit()` และ `withdraw()` สำหรับการฝากและถอนเงินตามลำดับ
- ฟังก์ชัน `get_amount()` สำหรับการรับข้อมูลจำนวนเงินทางคีย์บอร์ด ต้องอยู่ในหลักการที่เป็นໄได้ มีฉะนั้น จะเรียกตัวมันเองขึ้นมาทำงานแบบ Recursion
- ฟังก์ชัน `update_balance()` สำหรับปรับปรุงยอดคงเหลือโดยรับค่าเป็นจำนวนเงินที่ฝากหรือถอน

```

#define _POSIX_C_SOURCE 200809L
#include <locale.h>
#include <stdio.h>
#include <stdlib.h>

//prototype
short select_menu();
int get_amount(char msg[100]);
void deposit();
void withdraw();
void update_balance(int amount);

void main() {
    setlocale(LC_ALL, "");
    update_balance(0);
    select_menu();
}

short select_menu() {
    short m;
    printf("\n0: exit, 1: deposit, 2: withdraw");
    printf("\nselect menu no. >>");
    scanf("%hd", &m);

    if (m == 0) {
        exit(0);
    } else if (m == 1) {
        deposit();
    } else if (m == 2) {
        withdraw();
    } else {
        select_menu();
    }
}

void deposit() {
    int amount = get_amount("amount to deposit >>");
    update_balance(amount);
}

void withdraw() {
    int amount = get_amount("amount to withdraw >>");
    update_balance(-amount);
}

int get_amount(char msg[100]) {
    int amount;

```

```

balance = 1,000
0: exit, 1: deposit, 2: withdraw
select menu no. >>1
amount to deposit >>800

balance = 1,800
0: exit, 1: deposit, 2: withdraw
select menu no. >>2
amount to withdraw >>400

balance = 1,400
0: exit, 1: deposit, 2: withdraw
select menu no. >>0

```

```

printf(msg);
scanf("%d", &amount);
if (amount >= 0) {
    return amount;
} else {
    get_amount(msg);
}
}

void update_balance(int amount) {
    static int balance = 1000;
    if (amount < 0 && -amount > balance) {
        printf("\ninsufficient balance!\n");
    } else {
        balance += amount;
        printf("\nbalance = %'d\n", balance);
    }
    select_menu();
}

```

เนื้อหาในบทนี้ เราได้เรียนรู้ทั้งการสร้างและเรียกใช้ฟังก์ชัน รวมถึงข้อกำหนดสำคัญต่าง ๆ ที่เกี่ยวข้องทั้งหมด ซึ่งฟังก์ชันถือเป็นองค์ประกอบที่สำคัญอย่างยิ่งในการเขียนโปรแกรม ที่จะช่วยลดความยุ่งยากและยุ่งเหยิงในการเขียนโค้ดที่ซ้ำซ้อนกันลงมาได้ แล้วยังส่งผลให้โปรแกรมมีขนาดลดลง และสามารถทำงานได้เร็วขึ้นอีกด้วย



```
    = modifier_ob.modifiers.new("MIRROR")
    object_to_mirror_ob = mirror_ob
    mod.mirror_object = mirror_ob
    if len(mod.mirror_object.modifiers) > 0:
        for mod in mod.mirror_object.modifiers:
            if mod.type == "MIRROR_X":
                mod.use_x = True
                mod.use_y = False
                mod.use_z = False
            elif mod.type == "MIRROR_Y":
                mod.use_x = False
                mod.use_y = True
                mod.use_z = False
            elif mod.type == "MIRROR_Z":
                mod.use_x = False
                mod.use_y = False
                mod.use_z = True
    else:
        mod.type = "MIRROR_X"
        mod.use_x = True
        mod.use_y = False
        mod.use_z = False
    # add the end -add back the deselected
    # objects
    if len(context.selected_objects) < 2:
        print("please select exactly two objects, one to mirror to the selected object")
        return {'FINISHED'}
```

#### OPERATOR CLASSES

```
class MirrorOperator(bpy.types.Operator):
    """mirror to the selected object"""
    bl_idname = "object.mirror"
    bl_label = "Mirror"
    bl_options = {'REGISTER', 'UNDO'}
```

```
    def execute(self, context):
        if len(context.selected_objects) < 2:
            print("please select exactly two objects, one to mirror to the selected object")
            return {'FINISHED'}
```





## วาร์เรย์และสตริง (เพิ่มเติม)

หากมีตัวแปรที่จัดเก็บข้อมูลเป็นจำนวนมาก การอ้างถึงตัวแปรที่ลະตัวก็จะเกิดความยุ่งยาก ซึ่งเราอาจเปลี่ยนมาใช้วิธีการสร้างตัวแปรที่สามารถเก็บรายการข้อมูลได้มากกว่า 1 ค่า ซึ่งเรียกว่าวาร์เรย์ โดยรายการข้อมูลดังกล่าวที่เรามารถเข้าถึงด้วยลูป for หรือ while รวมถึงฟังก์ชันที่เกี่ยวข้องเพื่อดำเนินการกับมันได้

### ลักษณะพื้นฐานของวาร์เรย์

ถ้าเราจะรับข้อมูลที่เป็นตัวเลขจากผู้ใช้ผ่านทางคีย์บอร์ดทั้งหมด 10 จำนวน เพื่อจะนำไปใช้งานอื่นๆ ต่อไป เรา ก็อาจต้องสร้างตัวแปร 10 ตัวเพื่อรับในแต่ละค่า และเนื่องจากการรับค่า ในแต่ละครั้ง จะเก็บในตัวแปรคนละตัว ดังนั้น จึงอาจไม่สะดวกที่จะใช้ลูป เช่น

```
int n1;
printf("number 1 >>");
scanf("%d", &n1);

int n2;
printf("number 2 >>");
scanf("%d", &n2);

int n3;
printf("number 3 >>");
scanf("%d", &n3);
...
int n10;
printf("number 10 >>");
scanf("%d", &n10);
```

100	200	300	...	...	900	1000
n	n2	n3	...	...	n9	n10

จากโค้ดที่ผ่านมา เราจะมองเห็นความยุ่งยากได้อย่างชัดเจน เพราะนอกจากจะต้องมีตัวแปรเป็นจำนวนมากแล้ว ก็ยังต้องเขียนโค้ดซ้ำซ้อนกัน และขาดความยืดหยุ่นต่อการเปลี่ยนแปลง แก้ไขในอนาคตอีกด้วย แต่ถ้าเราเปลี่ยนมาจัดเก็บในรูปแบบของชุดข้อมูลที่มีเลขลำดับต่อเนื่องกัน สำหรับการอ้างอิง ก็จะใช้ลูปเพื่อเข้าถึงข้อมูลดังกล่าวได้ เช่น (โค้ดโดยลังเขปเท่านั้น ให้ดูผ่านๆ ไปก่อน)

```
int n[10];
for (int i = 0; i < 10; i++) {
    printf("\nnumber %d >>", i + 1);
    scanf("%d", &n[i]);
}
```

100	200	300	...	...	900	1000
n[0]	n[1]	n[2]	...	...	n[8]	n[9]

จากโค้ด เราจัดเก็บข้อมูลทั้งหมดด้วยตัวแปร `n` เพียงตัวเดียว โดยใช้เลขลำดับสำหรับ การอ้างถึงข้อมูลแต่ละค่าในรูปแบบ `n[x]` ดังนั้น หากใช้ร่วมกับลูป `for` ก็สามารถนำตัวนับของลูป (`i`) มาระบุเลขเป็นลำดับดังกล่าวได้ จึงจัดการได้ง่ายขึ้นและมีความยืดหยุ่นมากกว่า ซึ่งลักษณะ การจัดเก็บข้อมูลเช่นนี้เรารู้ว่า **อาร์เรย์ (Array)** โดยหลักการสำคัญที่ควรรู้จักในเบื้องต้นคือ

- ชุดของข้อมูลที่เก็บในอาร์เรย์จะประกอบไปด้วยข้อมูลอยู่  $n$  ที่เรียกว่าสมาชิก (element) เช่น จากภาพที่แล้ว ตัวเลข 100, 200, ... คือสมาชิกแต่ละตัวของอาร์เรย์
- ตัวเลขที่ระบุเอาไว้ในวงเล็บ [] ตอนประกาศอาร์เรย์ เป็นการกำหนดจำนวนสมาชิก สูงสุดที่มีหรือจัดเก็บได้ภายในอาร์เรย์นั้น เช่นจากโค้ดที่ผ่านมา เรากำหนดเป็น `int[10]` แสดงว่ามีสมาชิกได้สูงสุดไม่เกิน 10 ตัว
- สมาชิกแต่ละตัวในอาร์เรย์ จะมีเลขลำดับ (index หรือ subscript) ที่ต่อเนื่องกันไป โดยเริ่มจาก 0, 1, 2, ... ไปจนลิ้นสุดอาร์เรย์ ซึ่งสมาชิกตัวแรกต้องมีลำดับเป็น 0 เสมอ
- การเข้าถึงสมาชิกแต่ละตัวในอาร์เรย์จะต้องใช้เลขลำดับเป็นตัวกำหนด เช่น `n[0]` หมายถึงสมาชิกตัวแรก หรือ `n[9]` หมายถึงสมาชิกตัวที่ 10 ( เพราะตัวแรกลำดับเป็น 0 )

อย่างไรก็ตาม ลักษณะของอย่างของอาร์เรย์ในภาษา C อาจแตกต่างไปจากภาษาอื่นๆ อญี่ บัง ซึ่งรายละเอียดปลีกย่อยอื่นๆ เราจะได้เรียนรู้เพิ่มเติมในลำดับต่อไป



### หมายเหตุ

ในช่วงแรกๆ จะเน้นการใช้เฉพาะอาร์เรย์ชนิดตัวเลขและอักขระไปก่อน ล้วนอาร์เรย์ของสตริงจะ กล่าวถึงในช่วงหลังๆ เนื่องจากต้องใช้วิธีการนางอย่างที่แตกต่างออกไป ซึ่งเราควรใช้พื้นฐานของ อาร์เรย์แบบตัวเลขไปก่อน

## การสร้างและกำหนดค่าสมาชิกของอาร์เรย์

อาร์เรย์ ก็คือตัวแปรอีกแบบหนึ่ง ซึ่งในบทที่ผ่านๆ มาనั้น เรายังได้ใช้มานบ้างแล้วนั้นคือ อาร์เรย์ของอักขระ (char array) เพื่อจัดเก็บข้อมูลแบบสตริง แต่อย่างไรก็ตาม เพื่อการเรียนรู้ที่ครบสมบูรณ์ยิ่งขึ้น ในที่นี้จะย้อนกลับไปศึกษาหลักการตั้งแต่เริ่มแรกอีกครั้ง เพราะมีลักษณะที่สำคัญหลายอย่างที่ยังไม่ได้กล่าวถึง โดยการสร้างหรือประกาศตัวแปรอาร์เรย์มีรูปแบบพื้นฐาน ดังนี้

ชนิดข้อมูล ชื่ออาร์เรย์[ขนาด]

ทั้งชนิดข้อมูล และชื่ออาร์เรย์ ก็เหมือนกับการประกาศตัวแปร ส่วนขนาดหรือจำนวน สมาชิกให้เขียนไว้ในวงเล็บ [] และต้องเป็นเลขจำนวนเต็มมากเท่านั้น เช่น

```
int nums[10];
float data[5 * 3];           //data[25]
unsigned short items[99 + 1]; //items[100]
char vowels[5];

double x[3], y;             //y เป็นตัวแปรธรรมด้า
long m[5], n[10];

int size = 20;
int arr[size];              //arr[20]

const unsigned short len = 50;
float list[len * 2];        //list[100]

int a[-3];      //Error เพราะจำนวนสมาชิกต้องเป็นจำนวนเต็มมาก
float b[5.5]; //Error เพราะจำนวนสมาชิกต้องเป็นจำนวนเต็มมาก
```

หลังจากที่สร้างอาร์เรย์ขึ้นมาแล้ว ลำดับต่อไปเราต้องกำหนดค่าของสมาชิกแต่ละตัวภายใน อาร์เรย์ และเนื่องจากตัวแปรอาร์เรย์แต่ละตัว สามารถจัดเก็บสมาชิกได้หลายตัวตามขนาดที่เรา กำหนดในตอนประกาศอาร์เรย์ ทั้งนี้การอ้างถึงสมาชิกย่อยๆ ของอาร์เรย์ เราจะใช้วงเล็บ [] วาง ต่อท้ายชื่อตัวแปรเพื่อบุลเล็กลำดับของสมาชิกในรูปแบบดังนี้

ชื่ออาร์เรย์[เลขลำดับ]

เลขลำดับ (index หรือ subscript) จะต้องมีค่าต่อเนื่องกันโดยอยู่ระหว่าง 0 ถึง (ขนาด อาร์เรย์ - 1) เช่น ถ้าประกาศอาร์เรย์เป็น int a[10] แสดงว่าเลขลำดับจะต้องอยู่ระหว่าง 0 ถึง (10 - 1) หรือ 0 - 9 เป็นต้น ส่วนการกำหนดค่าของสมาชิกของอาร์เรย์เราก็แค่อ้างถึงสมาชิก ในลำดับที่ต้องการแล้วกำหนดค่าด้วยเครื่องหมาย = เช่นเดียวกับตัวแปร เช่น

```
int a[5];      //มีสมาชิกสูงสุด 5 ตัว (เลขลำดับ 0 - 4)
a[0] = 10;     //กำหนดค่าให้กับสมาชิกตัวที่ 1
a[1] = 15;     //กำหนดค่าให้กับสมาชิกตัวที่ 2
a[2] = 25;     //กำหนดค่าให้กับสมาชิกตัวที่ 3
a[3] = 30;     //กำหนดค่าให้กับสมาชิกตัวที่ 4
a[4] = 45;     //กำหนดค่าให้กับสมาชิกตัวที่ 5

a[5] = 50;     //แม้มิได้เกิดข้อผิดพลาด แต่ข้อมูลไม่ได้ถูกจัดเก็บ
                // เพราะลำดับสมาชิกที่ถูกต้องของอาร์เรย์นี้คือ 0 - 4
```

การกำหนดค่าของสมาชิก จะต้องพิจารณาชนิดข้อมูลให้สอดคล้องกัน เช่นเดียวกับตัวแปร เช่น หากเราประกาศเป็นอาร์เรย์ชนิด int สมาชิกแต่ละตัวจะต้องเป็นเลขจำนวนเต็ม มิฉะนั้นอาจ เกิดข้อผิดพลาด หรือถูกแปลง (cast) เป็นจำนวนเต็มโดยอัตโนมัติ (ถ้าเป็นไปได้) อย่างไรก็ตาม เราไม่จำเป็นต้องกำหนดค่าของสมาชิกให้ครบถ้วน แต่อาจกำหนดเพียงบางลำดับ ส่วนสมาชิก ในลำดับที่เราไม่ได้กำหนด จะมีค่าเป็น 0 โดยอัตโนมัติสำหรับอาร์เรย์ชนิดตัวเลข หรือเป็นอักขระ ว่าง (null character) สำหรับอาร์เรย์ชนิดอักขระ เช่น

```
float f[5];
f[0] = 3.141;
f[3] = 0.357;
//f[1] = f[4] = 0.0
char letters[3];
letters[2] = 'c';
//letters[0] = letters[1] = letters[2] = ''
```

ที่ผ่านมานั้น เราแยกขั้นตอนการประกาศและกำหนดค่าของอาร์เรย์ออกจากกัน จึงอาจ ดูยุ่งยากเล็กน้อย แต่เราสามารถร่วบทั้ง 2 กรณีให้เหลือเพียงขั้นตอนเดียว นั่นคือกำหนดค่า ของสมาชิกแต่ละตัวด้วยแต่ขั้นตอนการประกาศ ในรูปแบบดังนี้

ชนิดข้อมูล ชื่ออาร์เรย์[] = { สมาชิก_0, สมาชิก_1, ... }; หรือ ชนิดข้อมูล ชื่ออาร์เรย์[จำนวน] = { สมาชิก_0, สมาชิก_1, ... };
---

- กรณีนี้ เราจะใช้วงเล็บ {} ในการกำหนดสมาชิกเริ่มแรกให้อาร์เรย์
- กรณีที่เราไม่ระบุขนาดในวงเล็บ [] จำนวนสมาชิกที่กำหนดไว้ใน {} จะเป็นตัวบ่งชี้ขนาดของอาร์เรย์โดยอัตโนมัติ เช่น ถ้าในวงเล็บ {} เรากำหนดข้อมูลไว้ 3 ค่า แสดงว่าอาร์เรย์นั้นมีขนาดเป็น 3
- กรณีที่เราระบุขนาดในวงเล็บ [] ขนาดของอาร์เรย์จะเท่ากับตัวเลขที่ระบุนี้
- ถ้าแม้จะระบุขนาดในวงเล็บ [] แต่ก็ไม่จำเป็นที่เราต้องกำหนดค่าของสมาชิกจนครบทุกตัว โดยสมาชิกในลำดับที่เราไม่ได้กำหนด จะมีค่าเป็น 0 โดยอัตโนมัติสำหรับอาร์เรย์ชนิดตัวเลข หรือเป็นอักขระว่าง สำหรับอาร์เรย์ชนิดอักขระเหมือนเช่นเคย

```

int odds[] = { 1, 3, 5 };           //อาร์เรย์มีขนาดเป็น 3
//odds[0] = 1, odds[1] = 3, odds[2] = 5

short evens[] = { 2, 4, 6, 8 };    //อาร์เรย์มีขนาดเป็น 7
//evens[0] = 2, evens[1] = 4, evens[2] = 6, evens[3] = 8
//evens[4] = evens[5] = evens[6] = 0

```

- ถ้าแม้จะกำหนดค่าด้วยวิธีนี้ แต่หากเราไม่สามารถกำหนดค่าของสมาชิกแบบเรียงลำดับต่อเนื่องกันได้ อาจเลือกกำหนดเพียงบางค่าในรูปแบบ
  $\{ [ \text{เลขลำดับ} ] = \text{ค่าที่กำหนด}, \dots \}$   
 ส่วนลำดับที่ไม่ได้กำหนดค่าก็เป็นไปตามหลักการเดิม ดังแนวทางต่อไปนี้

```

int a[] = { 1, 3, [4]=11, [7]=99 };
//เนื่องจากเลขลำดับสูงสุดคือ 7 ดังนั้น a จะมีขนาดเป็น 8
//โดยลำดับที่เราไม่ได้กำหนด (2, 3, 5, 6) จะมีค่าเป็น 0
//a = { 1, 3, 0, 0, 11, 0, 0, 99 };

short b[7] = { 2, 4, [3]=10, [5]=100 };
//อาร์เรย์มีขนาดเป็น 7 ลำดับที่ไม่ได้กำหนดจะมีค่าเป็น 0
//b = { 2, 4, 0, 10, 0, 100, 0 }

float c[6] = { [3]=3.3, 0.75, -1.99, [1]=1.1 };
//0.75 และ -1.99 อยู่ด้วยกันในลำดับ 3 จึงเป็นลำดับ 4 และ 5
//c = { 0.0, 1.1, 0, 3.3, 0.75, -1.99 }

double d[4] = { 3.141, [5]=5.55};   //Error
//d = { 3.141, 0.0, 0.0, 0.0 }
//การประกาศ d[4] แสดงว่าอาร์เรย์มีขนาดเป็น 4 (ลำดับ 0 - 3)
//ดังนั้น การกำหนดค่า [5]=5.55 จึงเกินขนาดของอาร์เรย์
//ทำให้เกิดข้อผิดพลาด

```

```

int e[5] = { [4]=44, 108, 1009, [1]=11, 711 };
//711 ตามหลังลำดับที่ 1 จึงเป็นลำดับที่ 2
//ตามปกติ 108 และ 1009 อยู่หลังลำดับ [4] จึงควรเป็นลำดับ 5, 6
//แต่การที่เราประยุกต์ e[5] แสดงว่ามีสมาชิกได้ 5 ตัว (ลำดับ 0 - 4)
//ดังนั้นลำดับที่ 5, 6 จึงไม่มีผล
//e = { 0, 11, 711, 0, 44 }

char vowels[5] = { 'a', 'e', [4]='u' };
//vowels = { 'a', 'e', '', '', 'u' };

```

สมาชิกของอาร์เรย์ที่กำหนดเอาไว้แล้ว สามารถเปลี่ยนแปลงแก้ไขในภายหลังได้ ซึ่งก็ทำ เช่นเดียวกับการกำหนดค่าแบบปกติ ดังแนวทางดังต่อไปนี้

```

int nums[] = { 7, 11, 108, 1009 };
//อาร์เรย์นี้มีสมาชิกได้สูงสุด 6 ตัว (ลำดับ 0 - 5)
//แต่เรากำหนดค่าเริ่มต้นเพียง 4 ตัว
//สมาชิกที่ไม่ได้ระบุจะมีค่าเป็น 0

nums[1] = 101;      //แก้ไขค่าเดิม (11) เป็น 101
nums[3] = 1900;     //แก้ไขค่าเดิม (1009) เป็น 1900
nums[4] = 2023;     //แก้ไขค่าเดิม (0) เป็น 2023
nums[6] = 6699;     //ข้อมูลไม่ถูกจัดเก็บ เพราะเกินขอบเขต

```

สำหรับตัวอย่างการนำไปใช้งานในลักษณะต่าง ๆ ให้ดูร่วมกับหัวข้อต่อไป

## การเข้าถึงสมาชิกของอาร์เรย์

หลังจากที่เราได้เรียนรู้วิธีการสร้างอาร์เรย์ในเบื้องต้นมาแล้ว ลำดับต่อไปเราจะศึกษาถึงวิธีการเข้าถึงสมาชิกของอาร์เรย์เพื่ออ่านค่ามาใช้งาน โดยมีแนวทางดังต่อไปนี้

## การอ่านค่าสมาชิกของอาร์เรย์

การอ่านค่าสมาชิกของอาร์เรย์ เราต้องอ้างถึงสมาชิกเป้าหมายที่ต้องการอ่านในรูปแบบเดียวกับกำหนดค่านั้นคือ ชื่ออาร์เรย์[เลขลำดับ] จากนั้นก็สามารถนำมาแสดงผลด้วย printf() หรือใช้งานอื่น ๆ ตามต้องการ แต่อย่างไรก็ตาม เราต้องคำนึงถึงเลขลำดับที่เป็นไปได้ของอาร์เรย์นั้น คือ 0 ถึง (ขนาด - 1) ดังที่กล่าวมาในหัวข้อที่แล้ว ทั้งนี้ หากระบุเลขลำดับที่ไม่อยู่ในขอบเขตอาจเกิดข้อผิดพลาด หรือได้ผลลัพธ์ที่ไม่ถูกต้อง เช่น

```

int a[] = {7, 11, 108, 1009};
printf("%d", a[0]);           //7
printf("%d", a[2]);           //108

float b[] = { 12.34, 56.78, 90.10 };
printf("%0.2f", b[1]);        //56.78
printf("%g", b[2]);           //90.1
printf("%0.2f", b[3]);        //ได้ผลลัพธ์ที่ไม่ถูกต้อง

```

ถ้าอาร์เรย์มีสมาชิกเป็นจำนวนมาก และเราต้องการอ่านค่าสมาชิกทุกตัว การอ้างถึงทีละตัวก็อาจเกิดความยุ่งยาก ดังนั้น เราสามารถนำลูป for มาใช้แทนได้ เนื่องจากลูปแบบนี้ต้องมีตัวนับ เช่น for (int i = 0; ...) จึงสามารถใช้ตัวนับของลูป มากำหนดเป็นลำดับสมาชิกของอาร์เรย์ ได้พอดี เช่น arr[i] และถ้าทราบขนาดของอาร์เรย์ก็จะเข้าถึงสมาชิกได้ทั้งหมด เช่น

```

int evens[] = { 2, 4, 6, 8, 10 };
for (int i = 0; i < 5; i++) {
    printf("%d ", evens[i]);
}

int size = 5;
int nums[] = {10, 20, 30, 40, 50};
int sum = 0;
for (int i = 0; i < size; i++) {
    sum += nums[i];
}

```

**ตัวอย่าง 10-1** สมมติว่าเรามีข้อมูลยอดขายลินค้า 5 ชนิด ที่จัดเก็บในแบบอาร์เรย์ ให้เราแสดงผลยอดขายลินค้าเหล่านั้นในแบบคล้ายกับตารางพร้อมกราฟเปรียบเทียบยอดขาย

```

#include <stdio.h>

void main() {
    //สมมติว่าเป็นยอดขายลินค้าแต่ละชนิด
    int p[] = {8, 4, 9, 2, 7};

    //ส่วนหัวตาราง
    printf(
        "\n%7s\t%5s\t%-10s",
        "product", "sales", "chart"
    );
}

```

product	sales	chart
1	8	*****
2	4	***
3	9	*****
4	2	**
5	7	*****

```

//วนลูปเท่ากับจำนวนลินค้าหรือสมาชิกในอาร์เรย์
for (int i = 0; i < 5; i++) {
    //แสดงข้อมูล 2 คอลัมน์แรก
    printf("\n%7d\t%5d", (i + 1), p[i]);

    //แสดงกราฟ โดยให้ * แทนสินค้า 1 ชิ้น
    //ซึ่งจะวนลูปเท่ากับจำนวนลินค้าที่ขายได้
    for (int j = 1; j <= p[i]; j++) {
        printf("*");
    }
}

putchar('\n');
}

```

**ตัวอย่าง 10-2** การทอยลูกเต๋าแต่ละครั้ง จะขึ้นหน้าได้หน้าหนึ่งระหว่าง 1 - 6 ในตัวอย่างนี้ เราจะทดสอบว่า ถ้าทอยลูกเต๋าจำนวน 1000 ครั้ง แต่ละหน้าจะขึ้นกี่ครั้ง โดยลำดับหน้าที่ขึ้นในการทอยแต่ละครั้งเราเก็บรวมไว้ในแบบอาร์เรย์ และเนื่องจากลำดับหน้าเป็นเลขเรียงกันตั้งแต่ 1 - 6 จึงสามารถนำมาเป็นลำดับอาร์เรย์ได้เลย ส่วนการทอยเราจะใช้วิธีการวนลูปตามจำนวนครั้ง ที่จะทอย โดยในแต่ละลูป จะสร้างเลขสุ่มระหว่าง 1 - 6 เพื่อใช้แทนเลขหน้าที่ทอยได้ ซึ่งถ้าได้หน้า 1 ก็ใช้นำไปเป็นเลขลำดับในการบวกเพิ่มหน้านั้นในอาร์เรย์ เช่น

ถ้าได้หน้า 3      faces[3] += 1  
 ถ้าได้หน้า 5      faces[5] += 1  
 ถ้าได้หน้า 3      faces[3] += 1  
 ถ้าได้หน้า 1      faces[1] += 1  
 ...

เมื่อวนลูปครบจำนวนครั้งในการทอยลูกเต๋าแล้ว ก็แสดงผลในแบบที่คล้ายกับตาราง

```

#include <stdio.h>
#include <time.h>
#include <stdlib.h>

void main() {
    //แม้ลูกเต่าจะมี 6 หน้า (1 - 6)
    //แต่เพื่อป้องกันความลับสน
    //จะให้ลำดับอาร์เรย์ตรงกับเลขหน้าของลูกเต่า
    //โดยให้อาร์เรย์มีสมาชิกจากลำดับ 0 - 6
    //แต่ลำดับ 0 เราจะไม่ใช้งาน
    int faces[7] = { };
}

```

face	frequency
1	166
2	159
3	146
4	182
5	156
6	191

```

srand(time(0) + rand());
int r;

//rand_min_max = min + rand() % (max - min + 1)
for (int i = 1; i <= 1000; i++) {
    r = 1 + rand() % (6 - 1 + 1);

    //ตัวได้หน้าใด ก็ให้เพิ่มจำนวนหน้านั้นไปอีก 1
    faces[r]++;
}

printf("\n%-4s\t\t%-9s\n", "face", "frequency");
for (int i = 1; i <= 6; i++) {
    printf("%4d\t\t%9d\n", i, faces[i]);
}
}

```

**ตัวอย่าง 10-3** ตัวอย่างนี้ จะรับข้อมูลทางคีย์บอร์ดเป็นคะแนนของนักเรียนจำนวนหนึ่ง จากนั้นพิจารณาให้เกรดโดยยึดเอาผู้ได้คะแนนสูงสุด (max) เป็นเกณฑ์ ดังนี้

- ได้เกรด A ถ้าคะแนน  $\geq max - 10$
- ได้เกรด B ถ้าคะแนน  $\geq max - 20$
- ได้เกรด C ถ้าคะแนน  $\geq max - 30$
- ได้เกรด D ถ้าคะแนน  $\geq max - 40$
- ได้เกรด F ถ้าคะแนน  $< max - 40$

หลักการคิดคือ เราต้องใช้ลูป for เพื่อรับข้อมูลให้ครบทุกคนก่อน โดยนำค่าไปเก็บไว้ใน อาร์เรย์ จากนั้นหาคะแนนสูงสุดที่เก็บในอาร์เรย์ ต่อไปก็ใช้ลูปเพื่อเข้าถึงคะแนนของแต่ละคน แล้วนำมาหาผลต่างแล้วเทียบเป็นเกรดตามหลักการที่ตั้งเอาไว้ โดยแนวทางการเขียนโค้ดเป็นดังนี้

```

#include <stdio.h>
#include <stdlib.h>

void main() {
    int num_students = 6;
    int score[num_students];
    int max = 0;

    putchar('\n');

    for (int i = 0; i < num_students; i++) {
        printf("student #%d score >>", i + 1);

```

```

student #1 score >>65
student #2 score >>43
student #3 score >>72
student #4 score >>68
student #5 score >>81
student #6 score >>54

student #1 grade: B
student #2 grade: D
student #3 grade: A
student #4 grade: B
student #5 grade: A
student #6 grade: C

```

```

scanf("%d", &score[i]);

//ถ้าคะแนนไม่อยู่ในขอบเขตที่ถูกต้อง
//ให้วนลูป เพื่อรับคะแนนใหม่ (ไม่เพิ่มค่าตัวนับ)
if (score[i] <= 0 || score[i] >= 100) {
    i--;
    continue;
}

if (score[i] > max) {
    max = score[i];
}
}

int d;
char g;

for (int i = 0; i < num_students; i++) {
    d = max - score[i];
    if (d <= 10) g = 'A';
    else if (d <= 20) g = 'B';
    else if (d <= 30) g = 'C';
    else if (d <= 40) g = 'D';
    else g = 'F';

    printf("\nstudent #%-d grade: %c", (i+1), g);
}
putchar('\n');
}

```

## การหาขนาดของอาร์เรย์

เราทราบไปแล้วว่า การอ่านค่าของอาร์เรย์นั้นจำเป็นต้องระบุเลขลำดับให้อยู่ในช่วงหรือขอบเขตที่เป็นไปได้ของอาร์เรย์นั้น มีฉะนั้นอาจเกิดข้อผิดพลาด หรือได้ผลลัพธ์ที่ไม่ตรงตามต้องการ อย่างไรก็ตาม สำหรับลำดับเริ่มต้น เราต้องรู้แล้วว่าต้องเริ่มจาก 0 เสมอ แต่ลำดับของสมาชิกตัวสุดท้าย จะขึ้นกับขนาดของอาร์เรย์ (จำนวนสมาชิก) ซึ่งในบางกรณี เราไม่ทราบล่วงหน้าถึงขนาดที่แน่นอนของอาร์เรย์ จึงจำเป็นต้องตรวจสอบในขณะรัน แต่ในภาษา C ไม่มีฟังก์ชันในการตรวจสอบขนาดที่แท้จริงของอาร์เรย์ได้ คงมีเพียงฟังก์ชัน `sizeof()` ซึ่งเป็นการตรวจสอบขนาดการจัดเก็บข้อมูลเป็นไบต์ (byte) เช่น

```

int a[5];
printf("%d", sizeof(a));           //20

```

จากโค้ดผู้อ่านอาจสงสัยว่า ทำไมผลลัพธ์จึงเป็น 20 แทนที่จะเป็น 5 ทั้งนี้ เพราะ sizeof() วัดขนาดเป็นไปต่อตังที่กล่าวมาแล้ว ไม่ใช่การนับจำนวนสมาชิก ซึ่งเราสามารถนำค่าที่ได้จาก sizeof() ไปคำนวณหาจำนวนสมาชิกได้ ดังที่จะกล่าวถึงในลำดับต่อไป แต่ก่อนอื่น อย่างให้เราทำความเข้าใจก่อนว่า การที่เราประกาศอาร์เรย์พร้อมระบุขนาดของสมาชิก เช่น int a[5] เป็นการกำหนดขนาดพื้นที่สำหรับจัดเก็บข้อมูลและต้องล้มพันธ์กับชนิดข้อมูลอีกด้วย ซึ่งจากบทที่ 3 เราได้ทราบไปแล้วว่า ข้อมูลแต่ละชนิดจะใช้พื้นที่จัดเก็บแตกต่างกัน เช่น ชนิด int ใช้พื้นที่ทั้งหมด 4 ไบต์ ทั้งนี้ หากเราประกาศอาร์เรย์เป็น int a[5] แสดงว่าตัวแปร a สามารถกำหนดสมาชิกได้ 5 ตัว และแต่ละตัวมีขนาด 4 ไบต์ (เพราะเป็นชนิด int) ดังนั้น ขนาดพื้นที่ในการจัดเก็บข้อมูลรวมเมื่อเราประกาศอาร์เรย์ int a[5] จึงเป็น  $4 * 5 = 20$  ดังโค้ดที่ผ่านมานั้นเอง

และเพื่อความเข้าใจที่ชัดเจนยิ่งขึ้น ในตารางด้านไป ผู้เขียนจะแสดงขนาดพื้นที่การจัดเก็บของข้อมูลแต่ละชนิดที่นำเสนอไว้ พร้อมตัวอย่างผลลัพธ์ที่ได้ถ้าเราอ่านขนาดด้วย sizeof() เมื่อประกาศอาร์เรย์ที่มีจำนวนสมาชิกแตกต่างกัน ดังนี้

type	size (bytes)	type a[1]: sizeof(a)	type b[2]: sizeof(b)	type c[5]: sizeof(c)
short	2	2	$2 * 2 = 4$	$2 * 5 = 10$
int	4	4	$4 * 2 = 8$	$4 * 5 = 20$
long	4	4	$4 * 2 = 8$	$4 * 5 = 20$
long long	8	8	$8 * 2 = 16$	$8 * 5 = 40$
float	4	4	$4 * 2 = 8$	$4 * 5 = 20$
double	8	8	$8 * 2 = 16$	$8 * 5 = 40$
char	1	1	$1 * 2 = 2$	$1 * 5 = 5$

จากตาราง ถ้าเราคำนวณส่วนมาเขียนเป็นโค้ด ก็จะมีลักษณะดังนี้ เช่น

```
int a[2];
printf("%d", sizeof(a)); //8
//ชนิด int สมาชิก 1 ตัว ใช้พื้นที่เก็บ 4 ไบต์
//ดังนั้น สมาชิก 2 ตัวใช้พื้นที่เก็บ 8 ไบต์

short b[2];
printf("%d", sizeof(b)); //4
//ชนิด short สมาชิก 1 ตัว ใช้พื้นที่เก็บ 2 ไบต์
//ดังนั้น สมาชิก 2 ตัวใช้พื้นที่เก็บ 4 ไบต์
```

```
double c[5];
printf("%d", sizeof(c)); //40
//ชนิด double สมาชิก 1 ตัว ใช้พื้นที่เก็บ 8 ไบต์
//ดังนั้น สมาชิก 5 ตัวใช้พื้นที่เก็บ 40 ไบต์
```

อย่างไรก็ตาม การหาขนาดพื้นที่จัดเก็บข้อมูลจาก `sizeof(array)` ซึ่งได้ค่าเป็นจำนวนไบต์ ไม่ใช่จำนวนสมาชิก จึงไม่สามารถนำไปกำหนดขอบเขตของสมาชิกได้ แต่เราต้องใช้วิธีการคำนวณแบบย้อนกลับ กล่าวคือ ผลลัพธ์ที่ได้จาก `sizeof(array)` เกิดจาก

$$\text{sizeof(array)} = \text{ขนาดของสมาชิกแต่ละตัว} * \text{จำนวนสมาชิก}$$

$$\text{ดังนั้น จำนวนสมาชิก} = \text{sizeof(array)} / \text{ขนาดของสมาชิกแต่ละตัว}$$

ในการเรียกใช้ `sizeof` ขนาด (พื้นที่จัดเก็บข้อมูล) ของสมาชิกแต่ละตัวจะเท่ากันหมด และเท่ากับขนาดพื้นที่จัดเก็บของชนิดข้อมูลนั้น เช่น

```
int a[3] => sizeof(a[0]) = sizeof(a[1]) = sizeof(a[2]) = sizeof(int)
```

ดังนั้น วิธีที่ง่ายสุดในการหาจำนวนสมาชิกของอาร์เรย์อาจใช้สูตร ดังนี้

```
num_elements = sizeof(array) / sizeof(array[0])
```

สาเหตุที่ใช้ `array[0]` เพราะลำดับอาร์เรย์เริ่มจาก 0 เสมอ ซึ่งถ้าเป็นอาร์เรย์ที่มีอยู่จริงและไม่ว่าอาร์เรย์จะมีขนาดเท่าใดก็ตาม ย่อมต้องมีลำดับที่ 0 อย่างแน่นอน เช่น

```
int a[] = {1, 3, 5, 7, 9};
int n_a = sizeof(a) / sizeof(a[0]); //5

short b[10] = {5, 5, 5};
int n_b = sizeof(b) / sizeof(b[0]); //10

float c[5] = { [3] = 3.141 };
int n_c = sizeof(c) / sizeof(c[0]); //5
//ถึงอย่างไรก็ต้องมี c[0] เพราะสมาชิกที่ขาดหายไป
//จะถูกกำหนดค่าเป็น 0 โดยอัตโนมัติ ดังที่เราเรียนรู้มาแล้ว
```



### หมายเหตุ

ตามหลักการที่ถูกต้องของภาษา C นั้น ผลลัพธ์ที่ได้จาก `sizeof()` จะเป็นค่าหรือข้อมูลชนิด `size_t` หรือพจน์ที่เทียบได้กับ `unsigned int` ซึ่งเป็นเลขจำนวนเต็มบวก แต่เพื่อป้องกันความลับสนหรือไม่ให้ดูยุ่งยากเกินไป ในที่นี้ จะใช้ชนิด `int` ที่เราคุ้นเคยทดแทนไปก่อน

**ตัวอย่าง 10-4** จากอาร์เรย์ที่กำหนดให้หาค่าต่ำสุดและสูงสุด ซึ่งก็ใช้แนวทางเดิมดังที่เราเคยทำกันมาในบทก่อน ๆ ที่รับข้อมูลจากคีย์บอร์ด เพียงแต่คราวนี้เปลี่ยนเป็นการอ่านค่าจากอาร์เรย์แทน โดยดึงค่าต่ำสุดและสูงสุดให้เท่ากับสมาชิกตัวแรกของอาร์เรย์เอาไว้ก่อน จากนั้นวนลูปตามจำนวนสมาชิกในอาร์เรย์ ซึ่งในแต่ละลูป หากสมาชิกตัวนั้นมีค่าน้อยกว่า ค่าต่ำสุดเดิม ก็กำหนดให้เป็นค่าต่ำสุดแทนค่าเดิม และถ้ามีค่ามากกว่าค่าสูงสุดเดิม ก็กำหนดให้เป็นค่าสูงสุดแทนค่าเดิม ทั้งนี้เมื่อวนลูปจนครบจำนวนสมาชิก ก็จะได้ค่าต่ำสุดและสูงสุดตามต้องการ นอกจากนี้ ก็ให้เพิ่มการนับจำนวนเลขคู่และเลขคี่ในอาร์เรย์นั้นด้วยว่ามีอย่างละกี่จำนวน

```
#include <stdio.h>

void main() {
    int a[] = {555, 7, 11, -101, 108, -1009, 999};
    int size = sizeof(a) / sizeof(a[0]);

    //เริ่มแรก ให้ค่าต่ำสุดและสูงสุดเท่ากับสมาชิกตัวแรกไว้ก่อน
    int min = a[0], max = min;
    int odds = 0, evens = 0;

    for (int i = 0; i < size; i++) {
        //ถ้าสมาชิกตัวนี้ น้อยกว่าค่าต่ำสุดเดิม
        //ก็กำหนดให้เป็นค่าต่ำสุดแทนค่าเดิม
        if (a[i] < min) {
            min = a[i];
        }

        //ถ้าสมาชิกตัวนี้ มากกว่าค่าสูงสุดเดิม
        //ก็กำหนดให้เป็นค่าสูงสุดแทนค่าเดิม
        if (a[i] > max) {
            max = a[i];
        }

        if (a[i] % 2 == 0) {
            evens++;
        } else {
            odds++;
        }
    }

    printf("\nmin = %d, max = %d", min, max);
    printf("\nodds = %d, evens = %d\n", odds, evens);
}
```

min = -1009, max = 999  
 odds = 6, evens = 1

## การใช้อาร์เรย์เป็นพารามิเตอร์ของฟังก์ชัน

เราสามารถใช้อาร์เรย์เป็นพารามิเตอร์ของฟังก์ชันได้เช่นเดียวกับข้อมูลพื้นฐานทั่วไป แต่ก็มีลักษณะบางอย่างที่แตกต่างจากอาร์เรย์ในแบบตัวแปรปกติ ซึ่งสามารถสรุปแนวทางได้ดังนี้

- ในส่วนพารามิเตอร์ของฟังก์ชัน เราต้องกำหนดค่าให้กับพารามิเตอร์แบบข้อมูลพื้นฐาน นั่นคือ ระบุทั้งชนิดข้อมูลแล้วตามด้วยชื่อพารามิเตอร์และกิ่งเล็บ [] พร้อมตัวเลข ระบุขนาดสูงสุดของอาร์เรย์ เช่น

```
void func(int data[10]) {
    for (int i = 0; i < 10; i++) {
        printf("%d ", data[i]);
    }
    ...
}
```

- หากเราไม่ระบุขนาดอาร์เรย์ที่เป็นพารามิเตอร์ จะมีปัญหางานอย่างคือ เราไม่สามารถตรวจสอบขนาดหรือจำนวนสมาชิกของอาร์เรย์ที่เป็นพารามิเตอร์ได้ด้วยฟังก์ชัน sizeof() เช่น โค้ดต่อไปนี้จะเกิดข้อผิดพลาด

```
void func(int data[]) {
    int sz = sizeof(data) / sizeof(data[0]); //Error
    //แต่หากเราไม่ทราบขนาดของอาร์เรย์
    //อาจเกิดปัญหานในการใช้งานขั้นตอนต่อไป
    ...
}
```

- แต่หากเราจะระบุขนาดเป็นตัวเลขที่แน่นอนเหมือนกับโค้ดก่อนนี้ จะขาดความยืดหยุ่น เพราะอาร์เรย์จะมีจำนวนสมาชิกเกินกว่านั้นไม่ได้ ซึ่งเราอาจแก้ปัญหาโดยให้มีพารามิเตอร์อีกด้วยสำหรับบ่งชี้ขนาดของอาร์เรย์ เช่น

```
//ให้พารามิเตอร์ size เป็นขนาดของพารามิเตอร์ data
void func(int data[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", data[i]);
    }
    ...
}
```

```

void main() {
    int a[] = {7, 11, 108, 1009};
    // ก่อนเรียกใช้ฟังก์ชัน เราต้องอ่านขนาดของอาร์เรย์
    // เพื่อกำหนดเป็นอาร์กิวเม้นต์ตัวที่สอง
    int sz = sizeof(a) / sizeof(a[0]);
    func(a, sz);
    ...
}

```

**ตัวอย่าง 10-5** แนวทางการกำหนดฟังก์ชันที่มีพารามิเตอร์เป็นอาร์เรย์ โดยฟังก์ชันนี้จะเป็นการตรวจสอบว่ามีข้อมูลอยู่ในอาร์เรย์หรือไม่

```

#include <stdio.h>
#include <stdbool.h>

bool in_num_array(float nums[], int size, float x) {
    for (int i = 0; i < size; i++) {
        if (nums[i] == x) {
            return true;
        }
    }

    return false;
}

void main() {
    float a[] = {-1, 3.141, -0, -5, 0.357, 99};
    int sz = sizeof(a) / sizeof(a[0]);

    putchar('\n');
    in_num_array(a, sz, -1) ?
        printf("%d is in array\n", -1) :
        printf("%d is not in array\n", -1);

    in_num_array(a, sz, 0.0) ?
        printf("%.1f is in array\n", 0.0) :
        printf("%.1f is not in array\n", 0.0);

    in_num_array(a, sz, 69) ?
        printf("%d is in array\n", 69) :
        printf("%d is not in array\n", 69);
}

```

-1 is in array  
0.0 is in array  
69 is not in array

## อาร์เรย์แบบ 2 มิติ

อาร์เรย์ที่เราได้ศึกษาในหัวข้อต่างๆ ที่ผ่านมานั้น จดอยู่ในประเภทอาร์เรย์ 1 มิติ (One-Dimensional Array) คือสามารถแต่ละตัวจะมีเพียงค่าเดียว ไม่ได้เป็นอาร์เรย์ที่ซ้อนกันลงไป ลักษณะในหัวข้อนี้จะกล่าวถึงอาร์เรย์เพิ่มเติมอีกแบบหนึ่ง นั่นคือ อาร์เรย์แบบ 2 มิติ (Two-Dimensional Array) โดยอาร์เรย์แบบนี้สามารถแต่ละตัวของมันจะเป็นอาร์เรย์ด้วย หรือเป็นอาร์เรย์ชุดของอาร์เรย์ (Array-of-Array) นั่นเอง โดยมีรายละเอียดที่นำเสนอในดังนี้

### ลักษณะของอาร์เรย์ 2 มิติ

ลักษณะของอาร์เรย์ 2 มิติอาจจะซับซ้อนอยู่บ้าง ดังนั้น จึงขออธิบายแนวคิดพื้นฐานในการจัดเก็บข้อมูลของอาร์เรย์ชนิดนี้ก่อนดังต่อไปนี้

สมมติว่าเราเก็บข้อมูลของจำนวนลินค้า 3 ชนิดที่ขายได้ในรอบสัปดาห์ ดังตาราง (ในที่นี้จะเอาชนิดลินค้าเป็นหลัก) ซึ่งถ้าเราพิจารณาในรูปแบบอาร์เรย์ 1 มิติ จะได้ดังนี้

สินค้านิดที่ 1				สินค้านิดที่ 2				สินค้านิดที่ 3																											
<table border="1"> <tr> <td>100</td><td>101</td><td>...</td><td>106</td></tr> <tr> <td>[0]</td><td>[1]</td><td>...</td><td>[6]</td></tr> </table>				100	101	...	106	[0]	[1]	...	[6]	<table border="1"> <tr> <td>200</td><td>201</td><td>...</td><td>206</td></tr> <tr> <td>[0]</td><td>[1]</td><td>...</td><td>[6]</td></tr> </table>				200	201	...	206	[0]	[1]	...	[6]	<table border="1"> <tr> <td>300</td><td>301</td><td>...</td><td>306</td></tr> <tr> <td>[0]</td><td>[1]</td><td>...</td><td>[6]</td></tr> </table>				300	301	...	306	[0]	[1]	...	[6]
100	101	...	106																																
[0]	[1]	...	[6]																																
200	201	...	206																																
[0]	[1]	...	[6]																																
300	301	...	306																																
[0]	[1]	...	[6]																																
อาทิตย์	จันทร์	...	เสาร์	อาทิตย์	จันทร์	...	เสาร์	อาทิตย์	จันทร์	...	เสาร์																								

```
int p1[] = {100, 101, ..., 106}; //สินค้านิดที่ 1
int p2[] = {200, 201, ..., 206}; //สินค้านิดที่ 2
int p3[] = {300, 301, ..., 306}; //สินค้านิดที่ 3
```

แต่อาร์เรย์แบบ 1 มิติ จะมีความยุ่งยากในกรณีที่เราจะต้องเข้าไปจัดการข้อมูลลินค้าแต่ละชนิดจนครบ เช่น ถ้าต้องการหาผลรวมของจำนวนยอดขายลินค้าทุกชนิดในแต่ละวัน จะต้องเขียนโค้ดดังนี้

```
int sum_p1 = 0;
for (int i = 0; i < 7; i++) {
    sum_p1 += p1[i];
}

int sum_p2 = 0;
for (int i = 0; i < 7; i++) {
    sum_p2 += p2[i];
}
...
```

ถ้าเรามีลินค้าเป็นร้อยชนิดก็ต้องทำข้าแบบนี้ไปจนครบ จึงเกิดความยุ่งยากและขาดความยืดหยุ่น ดังนั้น เราจึงจะเปลี่ยนวิธีการใหม่ โดยให้รายการลินค้าแต่ละชนิดเป็นอาร์เรย์ และให้จำนวนที่ขายได้ในแต่ละวันก็เป็นอาร์เรย์ด้วย ดังลักษณะดังนี้

```
int p1[] = {100, 101, ..., 106};      //สินค้าชนิดที่ 1
int p2[] = {200, 201, ..., 206};      //สินค้าชนิดที่ 2
int p3[] = {300, 301, ..., 306};      //สินค้าชนิดที่ 3
int sales[] = {p1, p2, p3};
```

<b>sales[0]</b>	<b>sales[1]</b>	<b>sales[2]</b>
100    101    ...    106	200    201    ...    206	300    301    ...    306
[0]    [1]    ...    [6]	[0]    [1]    ...    [6]	[0]    [1]    ...    [6]

`sales[0]` อ้างถึง `p1`, `sales[1]` อ้างถึง `p2`, และ `sales[2]` อ้างถึง `p3` ดังนั้น จึงสามารถอ้างถึงลินค้าแต่ละชนิดโดยใช้ลูป

```
for (int i = 0; i < 3; i++) {
    ...
    sales[i] ...
    //ใช้ตัวนับของลูปในการอ้างถึงลินค้าแต่ละชนิด เช่น
    //sales[0] ใช้อ้างถึงลินค้า p1
}
```

อย่างไรก็ตามเนื่องจากยอดขายลินค้าแต่ละชนิดก็เป็นอาร์เรย์ด้วย ดังนั้น ในอาร์เรย์จึงมีสมาชิก 2 อย่างคือ รายการลินค้าแต่ละชนิด กับยอดขายของลินค้าแต่ละชนิดในแต่ละวัน โดยเราต้องใช้วงลูป [] จำนวน 2 อัน ในการอ้างถึงสมาชิกของอาร์เรย์ในลักษณะนี้ คือ

`sales[ชนิดลินค้า][ยอดขายลินค้า]`

`sales[0][0] = 100`  
เป็นยอดขายลินค้าชนิดที่ 1 (ลำดับในอาร์เรย์คือ 0)  
ในวันอาทิตย์ (ลำดับเป็น 0)

`sales[0][1] = 101`  
เป็นยอดขายลินค้าชนิดที่ 1 ในวันจันทร์ (ลำดับเป็น 1)

`sales[0][6] = 106`  
เป็นยอดขายลินค้าชนิดที่ 1 ในวันเสาร์ (ลำดับเป็น 6)

...

```

sales[1][0] = 200
เป็นยอดขายสินค้าชนิดที่ 2 (ลำดับในอาร์เรย์คือ 1)

sales[1][1] = 201
เป็นยอดขายสินค้าชนิดที่ 2 ในวันจันทร์
...
sales[2][0] = 300
เป็นยอดขายสินค้าชนิดที่ 3 ในวันอาทิตย์

sales[2][6] = 306
เป็นยอดขายสินค้าชนิดที่ 3 ในวันเสาร์

```

จากที่กล่าวมาทั้งหมด ก็หวังว่าผู้อ่านจะพอมองเห็นลักษณะของอาร์เรย์ 2 มิติได้ชัดเจน ยิ่งขึ้น ซึ่งเราสามารถสรุปลักษณะหลักเกณฑ์ที่สำคัญของอาร์เรย์ 2 มิติได้ดังนี้

- อาร์เรย์ 2 มิติคืออาร์เรย์ที่วางช้อนกัน ซึ่งสมาชิกแต่ละตัวก็จะเป็นอาร์เรย์ด้วย ดังนั้น เราจึงอาจเรียกอีกอย่างว่าเป็นอาร์เรย์ของอาร์เรย์ (Array-of-Array)
- ตัวแปรของอาร์เรย์ 2 มิติจะกำหนดด้วยวงเล็บต่อท้าย 2 อัน เช่น `data[][]` โดย อันแรกหรืออาร์เรย์ชั้นนอกใช้กำหนดเลขลำดับของสมาชิกแต่ละอาร์เรย์ย่อย และ อันที่สองใช้กำหนดลำดับสมาชิกภายในอาร์เรย์ย่อยหรืออาร์เรย์ชั้นใน
- การสร้างอาร์เรย์ 2 มิติกำหนดถูกต้องกับอาร์เรย์ 1 มิติ แต่ต้องใช้วงเล็บ 2 อัน โดยชนิด ของมันจะยึดตามข้อมูลที่จะเก็บในหรืออาร์เรย์ชั้นใน ดังแนวทางต่อไปนี้

```

int sales[3][6];
//อาร์เรย์ชั้นนอกมีสมาชิก 3 ตัว อาร์เรย์ชั้นในมีสมาชิก 6 ตัว

double locations[5][2];
//อาร์เรย์ชั้นนอกมีสมาชิก 5 ตัว อาร์เรย์ชั้นในมีสมาชิก 2 ตัว

char days[12][31];
//อาร์เรย์ชั้นนอกมีสมาชิก 12 ตัว อาร์เรย์ชั้นในมีสมาชิก 31 ตัว

int rows, cols;
float table[rows][cols];
//ขนาดของอาร์เรย์ ชี้ไปกับค่าของตัวแปร rows และ cols

```

- นอกจากนี้ เราอาจสร้างอาร์เรย์ 2 มิติพร้อมกับกำหนดค่าเริ่มต้นให้กับมัน ในลักษณะดังนี้

```

int a[2][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}};
//อาร์เรย์ชั้นนอกมีสมาชิก 2 ตัว อาร์เรย์ชั้นในมีสมาชิกอันละ 4 ตัว

double b[3][2] = {
    {1.111, 2.222}, {3.333, 4.444}, {5.555, 6.666}
};
//อาร์เรย์ชั้นนอกมีสมาชิก 3 ตัว อาร์เรย์ชั้นในมีสมาชิกอันละ 2 ตัว

float f[5][10] = {{1.23, 4.56}, {3.141}, [4]={7.7, 9.9}}
//เราอาจกำหนดสมาชิกเพียงบางส่วน (ส่วนที่ไม่กำหนดจะเป็น 0)

char c[4][2] = {
    {'Y', 'N'}, {'L', 'R'}, {'A', 'Z'}, {'R', 'X'}
};
//อาร์เรย์ชั้นนอกมีสมาชิก 4 ตัว อาร์เรย์ชั้นในมีสมาชิกอันละ 2 ตัว
//การนีซ้อมูลชนิดสตริง จะกล่าวถึงในหัวข้อต่อๆ ไป

```

- การอ้างถึงสมาชิกของอาร์เรย์ 2 มิติจะต้องใช้วงเล็บ 2 อัน โดยอันแรกใช้อ้างถึงลำดับของอาร์เรย์อันที่ 1 และอีกอันก็ใช้อ้างถึงลำดับของอาร์เรย์อันที่ 2 เช่น

```

int a[2][3];
a[0][0] = 10;
a[0][1] = 20;
a[0][2] = 30;
a[1][0] = 40; a[1][1] = 50; a[1][2] = 60;

double d[3][2] = {
    {1.111, 2.222}, {3.333, 4.444}, {5.555, 6.666}
};
printf("%g", d[0][0]);      //1.111
printf("%g", d[0][1]);      //2.222
printf("%g", d[1][1]);      //4.444
printf("%g", d[2][0]);      //5.555

double x = d[1][0];        //x = 3.333
double y = d[2][1];        //y = 6.666

```

## การหาขนาดของอาร์เรย์ 2 มิติ

เนื่องจากอาร์เรย์ 2 มิติเป็นอาร์เรย์ช้อนกัน ดังนั้น การหาขนาดหรือจำนวนสมาชิกของอาร์เรย์ ก็แบ่งเป็น 2 ลักษณะคือ

- จำนวนอาร์เรย์ย่อย
- จำนวนสมาชิกของแต่ละอาร์เรย์ย่อย

การหาขนาดหรือจำนวนสมาชิกของอาร์เรย์ 2 มิติ ก็ใช้หลักการพื้นฐานจากอาร์เรย์ 1 มิติ นั่นคือ ต้องคำนวณจากขนาดในหน่วยจำนวนไปต์ ซึ่งขอให้ดูแนวทางจากโค้ดต่อไปนี้

```
int a[3][5] = {{1,2,3,4}, {5,6,7,8}, {9,10}};
//ถึงแม้ว่าจะกำหนดสมาชิกไม่ครบทั้งหมด
//แต่ส่วนที่ขาดหายไป ก็ถูกนำไปคำนวณขนาดเหมือนเดิม

int sizeof_a = sizeof(a);
//คือจำนวนไปต์ของทั้งอาร์เรย์ ซึ่งเท่ากับ  $4 * 3 * 5 = 60$ 
//โดย 4 คือขนาดของชนิด int

int sizeof_a_0 = sizeof(a[0]);
//เนื่องจากอาร์เรย์ 2 มิติ ขนาดของอาร์เรย์ย่อย จะเท่ากับทั้งหมด
//แม้ในแต่ละอาร์เรย์ย่อยจะกำหนดจำนวนสมาชิกไม่เท่ากันก็ตาม
//โดยส่วนที่ขาดหายไป จะถูกเติมเลข 0 ให้จนครบ
//ดังนั้น sizeof(a[0]) =  $4 * 5 = 20$ 
//เนื่องจากแต่ละอาร์เรย์ย่อยมีสมาชิก 5 ตัว
//และแต่ละตัวมีขนาด 4 ไปต์

int num_subarr = sizeof(a) / sizeof(a[0]);
//เป็นการหาจำนวนอาร์เรย์ย่อย =  $60 / 20 = 3$ 

int num_el_of_subarr = sizeof(a[0]) / sizeof(a[0][0]);
//เป็นการหาว่าแต่ละอาร์เรย์ย่อยมีสมาชิกกี่ตัว =  $20 / 4 = 5$ 
//โดย sizeof(a[0][0]) = 4 เพราะเป็นชนิด int
```

การหาขนาดของอาร์เรย์ดังกล่าวนี้ จะเป็นประโยชน์สำหรับกรณีที่เราไม่สามารถระบุขนาดที่แน่นอนให้แก่อาร์เรย์เอาไว้ล่วงหน้าได้ แต่หากเป็นกรณีที่รู้ค่าແນ้นอนอยู่แล้วก็ไม่จำเป็นต้องกำหนดจำนวนให้เสียเวลาปกติได้

## การเข้าถึงสมาชิกในอาร์เรย์โดยใช้ลูป

การเข้าถึงสมาชิกย่อย ๆ ทุกตัวในแต่ละอาร์เรย์แบบ 2 มิติ เราต้องใช้ลูป for ช้อนกัน โดยลูปชั้นนอกใช้อังกฤษลำดับของอาร์เรย์ย่อย ส่วนลูปชั้นในใช้อังกฤษลำดับสมาชิกแต่ละตัวของแต่ละอาร์เรย์ย่อย

**ตัวอย่าง 10-6** แนวทางการเข้าถึงสมาชิกของอาร์เรย์แบบ 2 มิติ

```
#include <stdio.h>

void main() {
    int arr[3][6] = {
        {1, 3, 5, 7}, {2, 4, 6, 8, 10, 12}, {11, 22, 33}
    };
    //ความจริงขนาดของอาร์เรย์ เรากำหนดไว้ล่วงหน้า หรือรู้อยู่แล้ว
    //แต่ในที่นี้จะลองคำนวณเพื่อทบทวนสิ่งที่เราได้เรียนรู้มา

    int num_subarr = sizeof(arr) / sizeof(arr[0]);           //3
    int num_el_of_subarr = sizeof(arr[0]) / sizeof(arr[0][0]);

    //ลูป i ใช้เข้าถึงแต่ละอาร์เรย์ย่อย
    for (int i = 0; i < num_subarr; i++) {

        //ลูป j ใช้เข้าถึงสมาชิกแต่ละตัวในอาร์เรย์ย่อย
        printf("arr[%d] => ", i);
        for (int j = 0; j < num_el_of_subarr; j++) {
            printf("%d ", arr[i][j]);
        }

        putchar('\n');
    }
}
```

```
arr[0] => 1 3 5 7 0 0
arr[1] => 2 4 6 8 10 12
arr[2] => 11 22 33 0 0 0
```

## อาร์เรย์ของสตริง

ในบทที่ 4 เราได้เรียนรู้กันไปบ้างแล้วว่า สตริงก็คืออาร์เรย์ของอักขระ (char array) ซึ่งการกำหนดตัวแปรชนิดสตริง เราต้องสร้างเป็นอาร์เรย์ชนิด char เช่น

```
char str[100] = "Hello World";
char msg[] = "I can C";
```

ในเมื่อสตริงมันเป็นอาร์เรย์อยู่แล้ว ดังนั้น หากเราจะจัดเก็บข้อมูลชนิดสตริงในแบบอาร์เรย์ ต้องสร้างเป็นอาร์เรย์ชนิด char ในแบบ 2 มิติ ในรูปแบบดังนี้

```
char arr_name[n][max_len] = {"str_1", "str_2", "str_3", ..., "str_n"}
```

- **n** คือจำนวนสมาชิก หรือจำนวนสตริงสูงสุดที่มีได้ในอาร์เรย์
- **max\_len** คือความยาวสูงสุดของสตริงภายในอาร์เรย์ ต้องกำหนดให้ครอบคลุมสตริงอันที่จะยาวที่สุด

แนวทางการลร้างอาร์เรย์ของสตริงทั้งแบบที่ถูกต้องและแบบที่ผิดพลาด เช่น



```
char colors[3][20] = {"red", "green", "blue"};
char animals[5][10] = {"ant", "bug", "cat", "dog"};
//หากจำนวนสตริงไม่ครบตามขนาดที่กำหนด
//ก็จะเติมสตริงว่าง "" ลงไปแทน

char a[3][10] = {"one", "two", "three", "four"};
//ผิดพลาด เพราะในอาร์เรย์มีสมาชิก 4 ตัว
//ซึ่งเกินจำนวนตามที่กำหนดไว้คือ 3

char countries[3][10] = {
    "Thailand", "Japan", "United Kingdom"
};
//ผิดพลาด เพราะคำว่า United Kingdom
//มีจำนวนอักขระเกิน 10 ตัว ตามค่าที่กำหนดไว้
```

หากเราต้องการอ่านหรือเข้าถึงสตริงแต่ละตัวในอาร์เรย์ ให้อ้างถึงแบบอาร์เรย์ 1 มิติ เพื่อ อ่านทั้งคำ แต่ถ้าเราเข้าถึงแบบอาร์เรย์ 2 มิติจะเป็นการอ่านอักขระย่อยๆ ในแต่ละคำ รวมถึง ลักษณะที่นำเสนอในอื่นๆ เช่น

```
char colors[3][20] = {"red", "green", "blue"};

char a[10], b[20];
strcpy(a, colors[0]);           //a = "red"
strcpy(b, colors[1]);           //b = "green"
printf("%s", colors[2]);        //blue
puts(colors[0]);               //red

colors[2] = "yellow";           //error
strcpy(colors[2], "yellow");    //ok

char c = colors[0][0];          //r (red)
char d = colors[1][0];          //g (green)
char e = colors[1][3];          //e (green)
printf("%c", colors[1][4]);     //n (green)
```

```

int size = sizeof(colors[0]) / sizeof(colors[0][0]);
for (int i = 0; i < size; i++) {
    printf("%s ", colors[i]);
}

```

และเนื่องจากสตริงคืออาร์เรย์ของอักขระ ดังนั้น หากเราต้องการเข้าถึงแต่ละอักขระในสตริง ก็สามารถอ้างถึงด้วยเลขลำดับในแบบอาร์เรย์ 1 มิติ เช่น

```

char str[] = "thailand";
char a = str[0];           //t
char b = str[1];           //h
putchar(str[2]);          //a
printf("%c", str[3]);      //i

```

หากเราต้องการนับจำนวนอักขระในสตริง อาจใช้ฟังก์ชัน **strlen()** ซึ่งฟังก์ชันนี้อยู่ใน **เดอร์ string.h เช่น**

```

...
#include <string.h>
...
char str[2][20] = {"hello", "world"};
char sd[] = "sawasdee";
int n1 = strlen("hi");        //2
int n2 = strlen(sd);         //8
int n3 = strlen(str[0]);     //5

char vowels[] = "aeiou";
for (int i = 0; i < strlen(vowels); i++) {
    putchar(vowels[i]);
}

int x = 38014257;
char strnum[10];
sprintf(strnum, "%d", x);    //แปลงตัวเลขเป็นสตริง
for (int i = 0; i < strlen(strnum); i++) {
    printf("%c ", strnum[i]);   //3 8 0 1 4 2 5 7
}

```

สำหรับรายละเอียดของฟังก์ชัน **sprintf()** สามารถย้อนกลับไปทบทวนได้จากบทที่ 4 และลิ้งหนึ่งที่เรารายงานอยู่บ่อยๆ คือแปลงจากอักขระตัวเลข '0' - '9' ไปเป็นตัวเลข 0 - 9 ที่สามารถคำนวณได้ อาจใช้วิธีการง่ายๆ แบบได้แบบหนึ่งคือ

```

int a = '1' - '0';      //a = 1 (int)
int b = '2' - '0';      //b = 2 (int)
int c = '3' - 48;       //c = 3 (int)
int d = '4' - 48;       //d = 4 (int)

```

เราอาจลบอักษรตัวเลขด้วย '0' หรือ 48 ก็ได้ เพราะ ASCII ของ '0' คือ 48 ซึ่งเราสามารถย้อนกลับไปทบทวนข้อมูลชนิด char ได้จากบทที่ 3

**ตัวอย่าง 10-7** เป็นการแสดงชื่อวันในรอบสัปดาห์ (Sunday - Saturday) พร้อมชื่อสีประจำวันนั้น โดยเราจะเก็บชื่อวันและสีไว้ในอาร์เรย์ และรับค่าเป็นตัวเลขลำดับของวันในรอบสัปดาห์ (1 - 7) ทางคีย์บอร์ด จากนั้นนำตัวเลขนี้ไปใช้เป็นลำดับในการอ่านค่าจากอาร์เรย์

```

#include <stdio.h>

void main() {
    char daynames[7][20] = {
        "sunday", "monday", "tuesday", "wednesday",
        "thursday", "friday", "saturday"
    };

    char colors[7][20] = {
        "red", "yellow", "pink", "green",
        "orange", "blue", "purple"
    };

    int dayofweek;

    printf("\nEnter day of week (1 - 7) >>");
    scanf("%d", &dayofweek);

    if (dayofweek >= 1 && dayofweek <= 7) {
        //เนื่องจากลำดับของอาร์เรย์เริ่มจาก 0
        //แต่ลำดับวันเริ่มจาก 1 ดังนั้น ต้องลบออก 1
        //เพื่อให้ตรงกับลำดับของอาร์เรย์
        dayofweek -= 1;
        printf("\nDay name: %s", daynames[dayofweek]);
        printf("\nColor: %s\n", colors[dayofweek]);
    } else {
        puts("Error!");
    }
}

```

enter day of week (1 - 7) >>4  
 day name: wednesday  
 color: green

**ตัวอย่าง 10-8** เป็นการแสดงคำอ่านของตัวเลขแบบเรียงตัวในภาษาอังกฤษ เช่น 2309 อ่านว่า two three zero nine เป็นต้น โดยเก็บรายการคำอ่านในไว้อาร์เรย์ และให้ผู้ใช้ใส่ค่าตัวเลขทางคีย์บอร์ด จากนั้น เรานำไปคัดแยกตัวเลขออกจากกัน และนำตัวเลขไปใช้เป็นลำดับเพื่ออ่านคำอ่านของเลขตัวนั้นจากอาร์เรย์

```
#include <stdio.h>
#include <string.h>
```

```
void main() {
    char words[10][20] = {
        "zero", "one", "two", "three", "four",
        "five", "six", "seven", "eight", "nine"
    };

    long num;
    char strnum[20], text[100], digit;
    int d;

    printf("\nenter +int number >>");
    scanf("%ld", &num);

    sprintf(strnum, "%ld", num);

    for (int i = 0; i < strlen(strnum); i++) {
        d = strnum[i] - '0';
        //แปลงจากอักษรตัวเลขไปเป็นตัวเลข
        //เพื่อใช้เป็นลำดับของอาร์เรย์

        strcat(text, words[d]);
        strcat(text, " ");
    }

    puts(text);
}
```

```
enter +int number >>1081009
one zero eight one zero zero nine
```

**ตัวอย่าง 10-9** เป็นแนวทางการสร้างอาร์เรย์ช้อนกัน โดยจะแบ่งเป็น 2 อาร์เรย์ ซึ่งอันแรกจะเก็บข้อมูลชื่อทวีป และอีกอันใช้จัดเก็บข้อมูลชื่อประเทศของแต่ละทวีป ซึ่งทั้งสองอาร์เรย์นี้ ต้องวางลำดับให้สัมพันธ์กัน จากนั้นใช้ลูป for เพื่ออ่านชื่อทวีปและแต่ละประเทศในทวีปนั้นมาแสดงผล

```
#include <stdio.h>

void main() {
    //อาร์เรย์สำหรับเก็บชื่อของ 3 ทวีป
    //โดยชื่อแต่ละทวีปยาวไม่เกิน 20 อักษร
    char continents[3][20] = {
        "Asia", "Europe", "North America"
    };

    //อาร์เรย์สำหรับข้อมูลของ 3 ทวีป
    //แต่ละทวีปมีไม่เกิน 4 ประเทศ
    //ชื่อแต่ละประเทศยาวไม่เกิน 30 อักษร
    //ต้องวางอาร์เรย์ที่เก็บชื่อประเทศ
    //ให้ตรงกับลำดับทวีปในอาร์เรย์อันแรก
    char countries[3][4][30] = {
        {"Thailand", "Japan", "India", "Qatar"}, 
        {"UK", "France", "Germany", "Italy"}, 
        {"USA", "Canada", "Mexico"}
    };

    //แสดงผลโดยลูปชั้นนอกวนตามจำนวนทวีป
    for (int i = 0; i < 3; i++) {
        //แสดงชื่อทวีป
        printf("\n%s: ", continents[i]);

        //ลูปชั้นในแสดงชื่อแต่ละประเทศในทวีปนั้นๆ
        for (int j = 0; j < 4; j++) {
            (j > 0) ? printf(", ") : printf("");
            printf("%s", countries[i][j]);
        }
    }

    putchar('\n');
}
```

Asia: Thailand, Japan, India, Qatar
Europe: UK, France, Germany, Italy
North America: USA, Canada, Mexico,

**ตัวอย่าง 10-10** เราจะรับเวลาทางคีย์บอร์ด โดยเวลาที่ถูกต้องจะอยู่ในรูปแบบ ชั่วโมง:นาที:วินาที ซึ่งเป็นข้อมูลชนิดสตริง และทำการตรวจสอบดังนี้

- ค่าที่รับเข้ามาอยู่ในรูปแบบของเวลา n คือ hh:mm:ss หรือไม่ โดยอักขระในตำแหน่งที่ 2 และ 5 ต้องเป็นเครื่องหมาย :
- ค่าชั่วโมงคืออักขระในตำแหน่งที่ 0, 1 ต้องอยู่ระหว่าง 0 - 23
- ค่านาทีคืออักขระในตำแหน่งที่ 3, 4 ต้องอยู่ระหว่าง 0 - 59
- ค่าวินาทีคืออักขระในตำแหน่งที่ 6, 7 ต้องอยู่ระหว่าง 0 - 59

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <stdlib.h>

void main() {
    char time[9];

    printf(
        "\nEnter time %s",
        "(format hh:mm:ss e.g 10:09:45) >>"
    );
    scanf("%s", time);

    //ความยาว (จำนวนอักขระทั้งหมด) ต้องเท่ากับ 8
    bool len8 = strlen(time) == 8;

    //ตำแหน่งที่ 2 และ 5 ต้องเป็น :
    bool colon1 = time[2] == ':';
    bool colon2 = time[5] == ':';

    //ตรวจสอบว่ารูปแบบถูกต้องทั้งหมดหรือไม่
    if (!len8 || !colon1 || !colon2) {
        printf(
            "\ninvalid time format %s",
            "(hh:mm:ss)"
        );
    } else {
        //แยกค่าชั่วโมง นาที วินาที เป็นสตริง (อาร์เรย์อักขระ)
        char h[] = { time[0], time[1], '\0' };
        char m[] = { time[3], time[4], '\0' };
        char s[] = { time[6], time[7], '\0' };
    }
}
```

```

// แปลงจากสตริงเป็นตัวเลขเพื่อการตรวจสอบ
int hour = atoi(h);
int minute = atoi(m);
int second = atoi(s);

bool valid_hour = hour >= 0 && hour <= 23;
bool valid_minute = minute >= 0 && minute <= 59;
bool valid_second = second >= 0 && second <= 59;
if (valid_hour && valid_minute && valid_second) {
    printf("\ntime: %s\n", time);
    printf("hour: %d\n", hour);
    printf("minute: %d\n", minute);
    printf("second: %d\n", second);
} else {
    printf("\nvalue out of range");
}
}

putchar('\n');
}

```

```

enter time (format hh:mm:ss e.g 10:09:45) >>24:00:01
value out of range
-----
```

```

enter time (format hh:mm:ss e.g 10:09:45) >>12 34 56
-----
```

```

invalid time format (hh:mm:ss)
-----
```

```

enter time (format hh:mm:ss e.g 10:09:45) >>01:23:45
-----
```

```

time: 01:23:45
hour: 1
minute: 23
second: 45
-----
```

ลิ้งที่เราได้เรียนรู้กันไปในบทนี้ ทั้งข้อมูลแบบอาร์เรย์และสตริง ซึ่งต่างก็เป็นองค์ประกอบ  
สำคัญของภาษา C ที่ต้องใช้งานกันไปตลอด ดังนั้น เรายังคงศึกษาหลักการต่างๆ ให้มีความเข้าใจ  
ที่ดีพอสมควร ทั้งนี้ก็เพื่อประโยชน์ต่อการเรียนรู้เนื้อหาในบทต่อๆ ไป



# จ บ

## การเข้าถึงตำแหน่งตัวอักษรโดยใช้พอยน์เตอร์

จ บ กษณะที่สำคัญอย่างหนึ่งของภาษา C คือ การเข้าถึงตำแหน่งที่จัดเก็บข้อมูลโดยใช้พอยน์เตอร์ (pointer) ซึ่งในบางกรณี คือเป็นทางเลือกที่เราสามารถนำพอยน์เตอร์มาใช้ทดแทนวิธีการแบบปกติได้ แต่บางกรณีที่จำเป็นอย่างยิ่งที่เราจะต้องใช้วิธีการแบบพอยน์เตอร์ เพราะถูกกำหนดเอาไว้ล่วงหน้าแล้ว เช่น พารามิเตอร์ของฟังก์ชันต่างๆ ที่มากับภาษา C ซึ่งมักจะอยู่ในแบบของพอยน์เตอร์เป็นส่วนมาก ทั้งนี้ เราสามารถนำพอยน์เตอร์มาใช้งานได้ในหลายลักษณะ เช่น การใช้ร่วมกับตัวแปรพื้นฐานทั่วไป และโดยเฉพาะอย่างยิ่งการใช้ร่วมกับอาร์เรย์และสตริง จะถือเป็นเป้าหมายหลักของการใช้พอยน์เตอร์เลยก็ว่าได้

### หลักการเบื้องต้นของพอยน์เตอร์

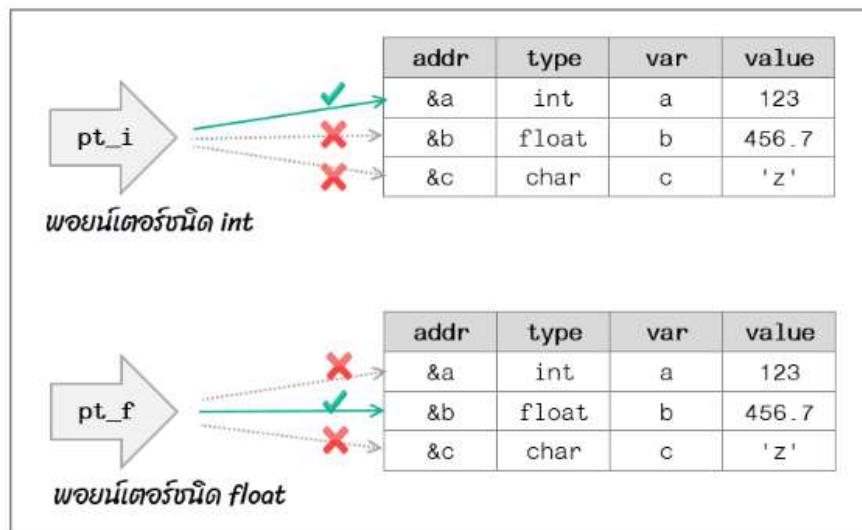
ในภาษา C จะมีวิธีการที่สำคัญอย่างหนึ่งในการเข้าถึงและการจัดการข้อมูล นั่นคือ การใช้พอยน์เตอร์ (pointer) ซึ่งลักษณะพื้นฐานของมันคือ จะซื้อไปยังตำแหน่งข้อมูล (ตัวแปร) หรืออาจเป็นการซื้อไปยังพอยน์เตอร์ด้วยกันเองก็ได้ จึงช่วยให้การประมวลผลมีความรวดเร็วยิ่งขึ้น และถือเป็นจุดเด่นที่สำคัญอย่างหนึ่งของภาษา C

อย่างไรก็ตาม ผู้เริ่มต้นศึกษา มักลับสนใจระหว่างพอยน์เตอร์และตัวแปร ดังนั้น ในหัวข้อนี้ เราจะเริ่มเรียนรู้จากหลักการที่สำคัญในเบื้องต้นของพอยน์เตอร์โดยใช้รูปภาพประกอบ ส่วนรายละเอียดทั้งหมดจะกล่าวถึงในหัวข้อต่อๆ ไป

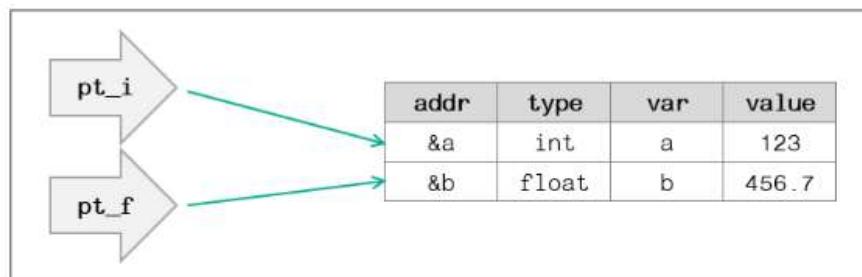
- ตามปกตินั้น ตัวแปรที่เราสร้างขึ้น จะมีตำแหน่ง (address) ของมันเองที่ไม่ซ้ำกันเลย พร้อมกับชนิดข้อมูลและค่าที่มันจัดเก็บ ซึ่งในที่นี้จะสมมติให้เป็นรูปภาพแบบง่ายๆ (สำหรับตำแหน่งของตัวแปร ความจริงจะเป็นรหัสที่ซับซ้อนกว่านี้ แต่ในที่นี้จะแทนด้วยลักษณ์แบบง่ายๆ เพื่อประกอบความเข้าใจเบื้องต้นเท่านั้น)

addr	type	var	value
&a	int	a	123
&b	float	b	456.7
&c	char	c	'z'

- พอยน์เตอร์ จะมีลักษณะบางอย่างคล้ายกับตัวแปรทั่วไป ซึ่งเราต้องกำหนดชนิดข้อมูลให้กับมัน เช่นเดียวกับตัวแปร ทั้งนี้ พอยน์เตอร์จะต้องชี้ไปยังตำแหน่งตัวแปรที่เป็นชนิดเดียวกันเท่านั้น เช่น พอยน์เตอร์ชนิด int จะต้องชี้ไปยังตำแหน่งของตัวแปรชนิด int เท่านั้น จะชี้ไปยังชนิด float หรือ char ไม่ได้



- เมื่อพอยน์เตอร์ชี้ไปที่ตำแหน่งของตัวแปรใด ในลำดับต่อไป เราถึงสามารถเข้าถึงและจัดการกับค่าของตัวแปรนั้นผ่านทางพอยน์เตอร์ได้ เช่นเดียวกับที่เราจัดการผ่านตัวแปรโดยตรง



```
// แนวทางของトイดโดยสังเขปเท่านั้น จากภาพข้างบน
*pt_i = a = 123
*pt_f = b = 456.7
*pt_i + 1 = a + 1 = 123 + 1 = 124
*pt_f * 10 = b * 10 = 456.7 * 10 = 4567.0
```

ที่กล่าวมาแล้ว เป็นเพียงหลักการพื้นฐานเพียงบางส่วนของพอยน์เตอร์เท่านั้น ซึ่งนอกจากนี้ยังมีรายละเอียดปลีกย่อยที่เรารู้รู้เพิ่มเติมอีกมาก ตามที่เราจะได้เรียนรู้ในหัวข้อต่อๆ ไป

## การประกาศตัวแปรประเภทพอยน์เตอร์

พอยน์เตอร์ จัดเป็นตัวแปรอีกประเภทหนึ่ง ดังนั้น จึงมีลักษณะพื้นฐานบางอย่างที่คล้ายคลึงกัน โดยเฉพาะอย่างยิ่งการประกาศตัวแปรประเภทนี้ขึ้นมา ก่อนการใช้งาน ซึ่งมีวิธีการให้เลือกใช้ดังนี้

รูปแบบการประกาศตัวแปรประเภทพอยน์เตอร์คือ

```
ชนิดข้อมูล *ชื่อพอยน์เตอร์
หรือ ชนิดข้อมูล * ชื่อพอยน์เตอร์
หรือ ชนิดข้อมูล* ชื่อพอยน์เตอร์
```

- ชนิดข้อมูล ก็เช่นเดียวกับของตัวแปร เพื่อกำหนดว่าพอยน์เตอร์นี้จะนำไปที่ข้อมูลหรือตัวแปรชนิดใด เช่น int, float, long, char
  - \* เพื่อให้พอยน์เตอร์แตกต่างจากตัวแปรธรรมดา เราจะใช้เครื่องหมาย \* เป็นตัวปั้งชี้ว่าเป็นตัวแปรพอยน์เตอร์ ซึ่งในขั้นตอนการประกาศ อาจกำหนดเครื่องหมายไว้หน้าชื่อพอยน์เตอร์ หรือหลังชนิดข้อมูล หรือระหว่างชนิดข้อมูลกับชื่อพอยน์เตอร์ก็ได้
  - ชื่อพอยน์เตอร์ ใช้หลักการเดียวกับชื่อตัวแปร สำหรับในหนังสือเล่มนี้ ผู้เขียนจะให้ชื่อพอยน์เตอร์ขึ้นต้นด้วยคำว่า pt เพื่อให้ดูแตกต่างจากตัวแปรธรรมดาและลังเกตได้ง่าย
- แนวทางการประกาศตัวแปรประเภทพอยน์เตอร์ในเบื้องต้น เช่น

```
int *pt_i1;
int* pt_i2;
int * pt_i3;
float *pt_f;
char *pt_ch;
long* pt_ln;
```

ถ้ามีพอยน์เตอร์ที่เป็นชนิดข้อมูลเดียวกัน สามารถประกาศรวมในคำลั่งเดียวกันได้ โดยคั่นด้วยเครื่องหมาย , เช่นเดียวกับตัวแปร แต่ต้องใช้รูปแบบที่ระบุเครื่องหมาย \* ไว้หน้าตัวแปรทุกตัวเท่านั้น เช่น

```
int *pt_i1, *pt_i2, *pt_i3;
float *pt_f1, *pt_f2;
```

เราสามารถประกาศตัวแปรธรรมดาร่วมกับพอยน์เตอร์ได้ ถ้าเป็นชนิดข้อมูลเดียวกัน แต่เฉพาะตัวแปรที่ระบุ \* นำหน้าเท่านั้นที่เป็นพอยน์เตอร์ เช่น

```
int *pt_i1, *pt_i2, num, value;
float a, b, c, *d, *e, f;
```

หากเราใช้วิธีแบบการเขียนเครื่องหมาย \* ไว้หลังชนิดข้อมูลแล้วระบุตัวแปรหลายตัว ผลที่ได้คือ เฉพาะตัวแปรตัวแรกเท่านั้นที่จัดเป็นพอยน์เตอร์ ส่วนตัวแปรอื่นๆ ที่เขียนตามหลังจะเป็นตัวแปรธรรมดา เช่น

```
int* pt, n, v;           //เฉพาะ pt ที่เป็นพอยน์เตอร์
                          //ส่วน n, v เป็นตัวแปรธรรมดานิด int

float* a, b, c, d;       //เฉพาะ a ที่เป็นพอยน์เตอร์
char * x, y, z;          //เฉพาะ x ที่เป็นพอยน์เตอร์

long* p, *t, *r;         //ทั้งหมดเป็นพอยน์เตอร์
```

## การกำหนดตำแหน่งให้พอยน์เตอร์ซึ่ไป

การประกาศตัวแปรพอยน์เตอร์ดังที่เราทำไปในหัวข้อที่แล้ว เป็นเพียงการสร้างตัวแปรแบบว่างๆ ที่ยังไม่มีค่าหรือการซึ่ไปที่ตำแหน่งใดๆ จึงไม่สามารถนำไปใช้งานได้ คล้ายกับการที่เราประกาศตัวแปรขึ้นมาแล้ว แต่ยังไม่ได้กำหนดค่าให้กับมันนั่นเอง ดังนั้น ในลำดับต่อไป เราจะมาเรียนรู้วิธีการระบุตำแหน่งให้กับมันว่าจะต้องซึ่ไปที่ใด โดยมีหลักการดังนี้

ตามปกตินั้น เมื่อเราต้องการอ้างถึง "ตำแหน่ง" ของตัวแปร เราจะวางเครื่องหมาย & ไว้หน้าชื่อตัวแปร เช่น

```
int a, b;
float c;

printf("\n&a => %d ", &a); //เช่น 6422044
printf("\n&b => %d", &b); //เช่น 6422040
printf("\n&c => %d", &c); //เช่น 6422036
```

เราทราบแล้วว่า พอยน์เตอร์ต้องซื้อไปที่ตำแหน่งของตัวแปร ดังนั้น ค่าที่จะกำหนดให้แก่ พอยน์เตอร์จะต้องเป็นตำแหน่งของตัวแปร ซึ่งมีรูปแบบพื้นฐานให้เลือกใช้ 2 วิธีคือ

### วิธีที่ 1 กำหนดค่าหลังประกาศพอยน์เตอร์ไว้ก่อนแล้ว ให้ใช้รูปแบบดังนี้

ชื่อพอยน์เตอร์ = &ตัวแปร

- ชื่อพอยน์เตอร์ ก็ตามที่เราได้ประกาศไว้ล่วงหน้าแล้ว แต่กรณีการกำหนดค่าวิธีนี้ ไม่ต้องมีเครื่องหมาย \* นำหน้าชื่อพอยน์เตอร์
- &ชื่อตัวแปร ก็คือตัวแปรที่เราประกาศไว้ล่วงหน้าแล้ว โดยต้องมีเครื่องหมาย & ข้างหน้าตัวแปรเสมอ เพราะเป็นการอ้างถึงตำแหน่งของมัน (หากระบุแค่ชื่อตัวแปรจะเป็นการอ้างถึงค่าของมัน)
- ทั้งพอยน์เตอร์และตัวแปร ที่จะใช้ร่วมกันต้องเป็นชนิดข้อมูลเดียวกันเท่านั้น เช่น ตัวแปรก็เป็นชนิด int ก็ต้องใช้พอยน์เตอร์ชนิด int เพื่อซื้อไปยังตัวแปรดังกล่าว

```
int a, b;
float c, d;
int *pt_a, *pt_b;
float *pt_c, *pt_d;

pt_a = &a;
pt_b = &b;
pt_c = &c;

*pt_d = &d; //error เพราะมี * นำหน้าชื่อพอยน์เตอร์
```

### วิธีที่ 2 กำหนดค่าในขั้นตอนการประกาศพอยน์เตอร์ ให้ใช้รูปแบบดังนี้

ชนิดข้อมูล \*ชื่อพอยน์เตอร์ = &ตัวแปร  
หรือ ชนิดข้อมูล\* ชื่อพอยน์เตอร์ = &ตัวแปร

- วิธีนี้ ต้องมีเครื่องหมาย \* ข้างหน้าชื่อพอยน์เตอร์ หรือตามหลังชนิดข้อมูล เพราะเป็นการประกาศพอยน์เตอร์
- ต้องมีเครื่องหมาย & ข้างหน้าตัวแปรเช่นเคย เพราะเป็นการอ้างถึงตำแหน่งของมัน

- หากจะใช้วิธีนี้ เราต้องประกาศตัวแปรเอาไว้ก่อนที่จะประกาศพอยน์เตอร์

```
int a, b, c;
int *pt_a = &a;
int *pt_b = &b;
int* pt_c = &c;
```

## การอ่านและกำหนดค่าของตัวแปรบằngกับพอยน์เตอร์ซึ่ง

เมื่อเรากำหนดให้พอยน์เตอร์ซึ่งไปที่ตำแหน่งของตัวแปรได้ดังหัวข้อที่ผ่านมา ก็สามารถเข้าถึงค่าของตัวแปรในตำแหน่งนั้นได้เช่นเดียวกับการเข้าถึงผ่านตัวแปรนั้นโดยตรง ไม่ว่าจะเป็นการอ่าน การกำหนดค่า หรือการแก้ไข ก็สามารถทำผ่านพอยน์เตอร์ได้ โดยลักษณะสำคัญที่เราควรรู้จักมีดังนี้

หากเราต้องการเข้าถึงข้อมูลหรือค่าของตัวแปรที่พอยน์เตอร์ซึ่งอยู่ ก็ให้อ้างถึงโดยระบุชื่อพอยน์เตอร์และต้องมีเครื่องหมาย \* อยู่ข้างหน้า เช่น

```
int x = 10, y;
int *pt;
pt = &x;           //ซึ่งไปที่ตำแหน่งของ x

y = *pt;          //อ่านค่าที่พอยน์เตอร์ซึ่งอยู่

printf("\nx = %d", x);      //10
printf("\n*pt = %d", *pt); //10
printf("\ny = %d", y);      //10

*pt += 1;          //เพิ่ยงเท่ากับ x += 1
printf("\nx = %d", x);      //11
printf("\n*pt = %d", *pt); //11

x *= 3;
printf("\nx = %d", x);      //33
printf("\n*pt = %d", *pt); //33

y = *pt + x;        //y = 33 + 33 = 66
```

ถ้าเราเปลี่ยนตำแหน่งที่พอยน์เตอร์ซึ่งจากตำแหน่งของตัวแปรหนึ่ง ไปซึ่งตำแหน่งของอีกตัวแปรหนึ่ง การเชื่อมโยงกับตัวแปรเดิมจะถูกยกเลิกและไม่มีผลต่อ ก็ เช่น

```

int x = 10, y = 20;
int *pt = &x; //ชี้ที่ตำแหน่งของ x

printf("*pt = %d", *pt); //10

pt = &y; //เปลี่ยนไปชี้ที่ตำแหน่งของ y
*pt += 5;
printf("x = %d", x); //10
printf("y = %d", y); //25
printf("*pt = %d", *pt); //25

```

นอกจากการอ้างถึงตำแหน่งผ่านพอยน์เตอร์แล้ว เราอาจอ้างถึงตำแหน่งผ่านตัวแปร ก็ได้โดยเขียนเครื่องหมาย & ไว้หน้าตัวแปร ก็จะหมายถึงตำแหน่งของตัวมันเอง แล้วถ้าเรานำเครื่องหมาย \* ไปวางไว้ข้างหน้า & เช่น \*&a ก็จะเทียบเท่ากับอ้างถึงผ่านพอยน์เตอร์ เช่น

```

int x = 10, y;
pt = &x
printf("&x = %d", &x); //6422040
printf("*&x = %d", *&x); //10

y = *&x + 5;
printf("y = %d", y); //10 + 5 = 15

*&x *= 2;
printf("*&x = %d", *&x); //10 * 2 = 20

```

pt = &x  
 \*pt = x  
 \*&x = x

นอกจากการซื้อโดยระบุค่าเป็นตำแหน่งของตัวแปรแล้ว หากเรามีหลายพอยน์เตอร์และเป็นชนิดข้อมูลเดียวกัน บางที่เราราจให้พอยน์เตอร์อันหนึ่งซึ่งเป็นพอยน์เตอร์อันหนึ่งก็ได้ แล้วพอยน์เตอร์ทั้งสองก็จะอ้างถึงตำแหน่งของตัวแปรเดียวกัน และมีผลต่อกันเหมือนเช่นเคย

```

int x = 10;
int *pt1, *pt2;

pt1 = &x;
pt2 = pt1; //pt2 = &x;

*pt1 += 5;
printf("\n*x = %d", x); //15
printf("\n*pt1 = %d", *pt1); //15
printf("\n*pt2 = %d", *pt2); //15

```

```

x = *pt1 + *pt2;
printf("\n*x = %d", x);           //30
printf("\n*pt1 = %d", *pt1);      //30
printf("\n*pt2 = %d", *pt2);      //30

```

การใช้พอยน์เตอร์เพื่อซึ่ปยังตำแหน่งของตัวแปรตามที่กล่าวถึงในหัวข้อต่างๆ ดังที่ผ่านมา เราอาจรู้สึกว่ามันไม่มีประโยชน์หรือความจำเป็นใด ๆ ที่จะต้องไปทำเช่นนั้นเลย ในเมื่อเราดำเนินการผ่านตัวแปรโดยตรงก็ได้ ซึ่งเราจะเริ่มมองเห็นข้อแตกต่างเมื่อนำใช้ร่วมกับฟังก์ชันในบางกรณี และเห็นคุณค่าที่แท้จริงของมันเมื่อนำไปใช้ร่วมกับอาร์เรย์ ตามที่จะกล่าวถึงในหัวข้อต่อ ๆ ไป

## การใช้พอยน์เตอร์ร่วมกับฟังก์ชัน

การนำพอยน์เตอร์มาใช้ร่วมกับฟังก์ชัน จะแบ่งออกเป็น 2 ลักษณะ นั่นคือ การรับพารามิเตอร์เข้ามาแบบพอยน์เตอร์ และการส่งข้อมูลกลับออกไปจากฟังก์ชันในแบบพอยน์เตอร์ โดยมีรายละเอียดดังต่อไปนี้

### การรับพารามิเตอร์ในแบบพอยน์เตอร์

โดยทั่วไป เราจะรับพารามิเตอร์แบบพอยน์เตอร์เข้ามาในฟังก์ชันเมื่อต้องการให้การเปลี่ยนแปลงที่เกิดขึ้นกับพารามิเตอร์นั้น ส่งผลไปถึงค่าของตัวแปรเดียวที่อยู่นอกฟังก์ชันโดยอัตโนมัติ หรือในภาษาคอมพิวเตอร์อื่น ๆ อาจเรียกว่า “reference” นั่นคือ การผ่านค่าพารามิเตอร์แบบอ้างอิง (pass by reference) นั่นคือ ทั้งในและนอกฟังก์ชันจะอ้างอิงตำแหน่งเดียวกัน โดยมีหลักการพื้นฐานดังนี้

- ที่ตัวฟังก์ชัน ให้กำหนดพารามิเตอร์ที่จะเป็นแบบพอยน์เตอร์ เช่นเดียวกับการประกาศตัวแปรแบบพอยน์เตอร์นั่นคือ อาจจะเครื่องหมายไวหน้าชื่อพารามิเตอร์ หรือหลังชนิดข้อมูล หรือระหว่างกลาง ทั้งนี้ในแต่ละฟังก์ชัน จะมีพารามิเตอร์ที่เป็นแบบพอยน์เตอร์จำนวนกี่ตัวก็ได้
- ภายในฟังก์ชัน หากเราต้องการอ้างถึงค่าของพารามิเตอร์ ก็ต้องมีเครื่องหมาย \* นำหน้าชื่อพารามิเตอร์ตามหลักการเดิม เช่น

```

//ฟังก์ชันสำหรับลดค่าเลขจำนวนเต็มลงไป 1 ค่า (decrement)
void dec(int *num) {    //หรือ (int* num)
    *num -= 1;
    //ต้องมีเครื่องหมาย * นำหน้า
    //เพราะอ้างถึงค่าที่พอยน์เตอร์ซึ่ง
}

```

- เมื่อเรียกใช้ฟังก์ชัน หากพารามิเตอร์ตัวใดที่เป็นพอยน์เตอร์ จะต้องกำหนดตำแหน่งของตัวแปรที่จะส่งค่าให้กับมันโดยระบุเครื่องหมาย & นำหน้าชื่อตัวแปร ดังกล่าว เช่น

```
int x = 100;
printf("\nbefore: x = %d", x); //100

dec(&x); //ระบุตำแหน่งของตัวแปร x

printf("\nafter: x = %d", x); //99
```

```
void dec(int *num) {
    ...
}

void main() {
    int x = 100;
    dec(&x);
}
```

จากโค้ด จะเห็นว่า ภายในฟังก์ชัน dec() เราได้ทำการเปลี่ยนค่าตัวแปร (ดำเนินการผ่านพอยน์เตอร์) โดยไม่ได้ส่งผลลัพธ์กลับออกมานอกจากฟังก์ชัน แล้วเมื่อเรียกฟังก์ชันโดยส่งตำแหน่งของตัวแปรให้กับมัน ค่าของตัวแปรดังกล่าวที่อยู่นอกฟังก์ชัน dec() จะเปลี่ยนตามโดยอัตโนมัติ เพราะทั้งในและนอกฟังก์ชันต่างก็อ้างอิงไปยังตำแหน่งเดียวกัน ค่าของมันจึงตรงกัน ทั้งนี้ หากเราใช้วิธีการแบบปกติธรรมดा (ไม่ใช้พอยน์เตอร์) จะต้องใช้สร้างฟังก์ชันแบบส่งผลลัพธ์กลับ แล้วเมื่อเรียกฟังก์ชันก็ต้องมีตัวแปรไปรับค่า ดังการ porównเทียบสองข้อไปนี้

ไม่ใช้พอยน์เตอร์	ใช้พอยน์เตอร์
<pre>int dec(int n) {     n -=1;     return n; } ... int x = 100; x = dec(x); printf("x = %d", x); //99</pre>	<pre>void dec(int *n) {     *n -= 1; } ... int x = 100; dec(&amp;x); printf("x = %d", x); //99</pre>

ในภาษา C มีฟังก์ชันจำนวนมากที่รับพารามิเตอร์แบบพอยน์เตอร์ ตัวอย่างที่ชัดเจนที่สุด ซึ่งเราใช้งานกันมาตั้งแต่ก็คือ scanf() นั่นเอง โดยเราต้องส่งตำแหน่งของตัวแปร ใช้เก็บผลลัพธ์ให้กับมัน เช่น

```
float num;
printf("enter number >>");
scanf("%f", &num);
```

```
// ส่งตำแหน่งตัวแปรที่ใช้เก็บผลลัพธ์ให้กับมัน
// พิงก์ชันใน scanf() จะเก็บค่าลงในตำแหน่งของตัวแปรนั้น
// ซึ่งหลังเรียก scanf() จะส่งผลมาถึงตัวแปร กบ m โดยอัตโนมัติ
```

และเพื่อความเข้าใจที่ชัดเจนยิ่งขึ้น เราลองมาดูตัวอย่างเพิ่มเติมเกี่ยวกับการรับค่าพารามิเตอร์ที่เป็นพอยน์เตอร์

**ตัวอย่าง 11-1** เป็นการสร้างฟังก์ชันสลับค่าของตัวแปรชนิดตัวเลข โดยให้มีพารามิเตอร์เป็นแบบพอยน์เตอร์ ซึ่งเมื่อเรียกใช้งาน เรายังแค่ส่งตำแหน่งของตัวแปรมาให้แก่ฟังก์ชันนี้ตามเดิม

```
#include <stdio.h>

void swap_num(float* n1, float* n2) {
    float temp;
    temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}

void main() {
    float x = 66, y = 99;

    printf("\nbefore: x = %g, y = %g", x, y);

    swap_num(&x, &y);

    printf("\nafter: x = %g, y = %g\n", x, y);
}
```

before: x = 66, y = 99  
after: x = 99, y = 66

### การส่งค่าแบบพอยน์เตอร์กลับจากฟังก์ชัน

ไม่ใช่แค่เพียงการรับพารามิเตอร์ในแบบพอยน์เตอร์ แต่เรายังสามารถส่งข้อมูลกลับออกจากการฟังก์ชันในแบบพอยน์เตอร์ได้เช่นกัน โดยมีข้อกำหนดที่สำคัญดังนี้

- ที่ตัวฟังก์ชัน ให้วางเครื่องหมาย \* ไว้หน้าชื่อฟังก์ชัน หรือตามหลังชนิดข้อมูลที่จะส่งกลับ เช่น

```
int *func1(..) {
    ...
}
```

หรือ

```
int* func2(...) {
    ...
}
```

- เนื่องจากเป็นการส่งค่ากลับแบบพอยน์เตอร์ซึ่งต้องซื้อไปยังตำแหน่งของตัวแปร ดังนั้นค่าที่จะส่งกลับต้องเก็บไว้ในตัวแปร จะระบุค่าข้อมูลโดยตรงไม่ได้ ( เพราะไม่สามารถบ่งชี้ตำแหน่งของมันได้ ) เช่น

```
int *func1(...) {
    ...
    return &x; //OK ส่งตำแหน่งของ x คืนไป
}

int *func2(...) {
    ...
    return 123; //error ระบุตำแหน่งไม่ได้
}
```

- แต่จากเรื่องของเขตของตัวแปรในบทที่ 9 เราทราบแล้วว่า ตัวแปรแบบ local ( หรือ auto ) ที่ประกาศขึ้นภายในฟังก์ชัน จะถูกยกเลิกเมื่อลิ้นสุดฟังก์ชัน ดังนั้น หากเราประกาศตัวแปรธรรมดابนแบบ local ขึ้นในฟังก์ชันเพื่อเก็บผลลัพธ์ก่อนจะส่งตำแหน่งมันออกໄไปตามหลักการที่กล่าวมาในหัวข้อที่แล้ว จะเกิดปัญหาคือ เมื่อลิ้นสุดฟังก์ชันแล้วตัวแปรที่เก็บผลลัพธ์ถูกยกเลิก จะไม่สามารถอ้างถึงตำแหน่งของตัวแปรนั้นจากนอกฟังก์ชันได้ ซึ่งเราสามารถแก้ปัญหานี้ได้หลายแนวทาง แต่วิธีการพื้นฐานที่สุดคือ กำหนดให้ตัวแปรที่จะเก็บผลลัพธ์เป็นแบบสแตติก เนื่องจากตัวแปรแบบนี้ จะยังไม่ถูกยกเลิกแม้จะลิ้นสุดฟังก์ชันแล้วก็ตาม เพราะอาจต้องนำกลับมาใช้ใหม่เมื่อเรียกฟังก์ชันครั้งต่อไป นั่นแสดงว่า ถ้าเราใช้ตัวแปรชนิดนี้เก็บผลลัพธ์ที่จะส่งคืนจากฟังก์ชัน ก็สามารถอ้างถึงตำแหน่งของมันได้ตลอด

```
int *func(...) {
    static int x; //ให้ตัวแปรที่เก็บผลลัพธ์เป็น static
    ...
    return &x; //ส่งตำแหน่งคืนไปตามหลักการเดิม
}
```

- การเรียกใช้ฟังก์ชันที่คืนค่าแบบพอยน์เตอร์ ตัวแปรที่ใช้รับค่า จะต้องเป็นพอยน์เตอร์ ชนิดเดียวกับที่ส่งคืนมา ดังโค้ดต่อไปนี้

```
void main() {
    int *n = func(...);
    //ตัวแปรที่รับค่าต้องเป็นพอยน์เตอร์เช่นกัน
    ...
}
```

```
int *func(...) {
    static int x;
    ...
    return &x;
}

void main() {
    int *n = func(...);
    printf("result = %d", *n);
}
```

- สำหรับการส่งค่ากลับเป็นข้อมูลชนิดสตริงซึ่งเป็นอาร์เรย์ของอักขระ จะมีข้อกำหนดบางอย่างที่แตกต่างจากที่กล่าวมานี้ ซึ่งความจริงรายละเอียดเกี่ยวกับการใช้พอยน์เตอร์ร่วมกับอาร์เรย์จะอยู่ในหัวข้อต่อ ๆ ไปของบทนี้ แต่เพื่อความต่อเนื่องจึงขอวิธีการส่งข้อมูลกลับจากฟังก์ชันแบบพอยน์เตอร์มาแนะนำให้ทราบล่วงก่อน ดังนี้
  - ในส่วนของฟังก์ชันกำหนดชนิดข้อมูลส่งกลับเป็น char และใช้เครื่องหมาย \* เพื่อบ่งชี้ว่าจะส่งค่ากลับแบบพอยน์เตอร์ตามเดิม เช่น char \*get\_string(...)
  - ภายในฟังก์ชัน เราสามารถส่งค่าสตริงกลับไปโดยตรงเลยก็ได้ หรือจะระบุแค่ชื่อตัวแปร (ไม่ต้องมีเครื่องหมาย \* นำหน้า) หรือจะส่งค่าจากตัวแปรโดยวาง \*& ไว้หน้าชื่อตัวแปร (ตัวแปรต้องเป็น static เมื่อตอนเดิม)
  - ในส่วนที่เรียกใช้ฟังก์ชัน ให้สร้างตัวแปรแบบ char array เพื่อรับค่าจากฟังก์ชัน แล้วใช้ฟังก์ชัน strcpy() เพื่อนำค่าจากฟังก์ชันไปกำหนดให้แก่ตัวแปรที่รับค่า

```
char *get_string(...) {
    static result[20] = "";
    ...
    //เลือกส่งค่ากลับแบบใดแบบหนึ่ง
    //return "abcd";          สามารถส่งค่าที่ไม่เก็บในตัวแปรกลับไปได้
    //return result;          ส่งค่าจากตัวแปรโดยตรง (ไม่มี * นำหน้า)
    //return *&result;        ส่งค่าของตัวแปรกลับไปโดยตรง
}

void main() {
    char str[20];
    strcpy(str, get_string());
}
```

แต่ขอให้พึงระวังเอาไว้ว่า ตัวแปรแบบสแตติก สามารถนำค่าเดิมกลับมาใช้ได้ ดังนั้น กายในฟังก์ชัน ต้องตรวจสอบให้รอบคอบว่า มีการนำค่าเดิมของตัวแปรกลับมาใช้ซ้ำหรือไม่ มี ฉะนั้นอาจก่อให้เกิดข้อผิดพลาดได้

**ตัวอย่าง 11-2** เป็นการสร้างฟังก์ชันที่ส่งค่ากลับในแบบพอยน์เตอร์ ซึ่งดัดแปลงมาจาก ตัวอย่างเดิมที่เราเคยทำมาในบทก่อน ๆ นี้ โดยเป็นลักษณะการฝากและถอนเงินในบัญชี ที่ประกอบด้วยฟังก์ชัน `balance()` ซึ่งจะคืนค่าเป็นยอดเงินคงเหลือในแบบพอยน์เตอร์ และวิธีฟังก์ชัน `deposit()` และ `withdraw()` สำหรับเพิ่มและหักเงินจากบัญชี ตามลำดับ ทั้งนี้ต้องอ่านค่ายอดคงเหลือเดิมจาก `balance()` มาใช้เพื่ออัปเดตค่าจากเดิม ซึ่งทั้งหมดนี้เป็นเพียงหลักการแบบง่าย ๆ ที่เราน่าจะพิจารณาอยู่แล้ว จึงขอให้ดูวิธีจากโค้ดได้เลย

```
#define _POSIX_C_SOURCE 200809L
#include <locale.h>
#include <stdio.h>

float *balance() {
    static float b;
    return &b;
}

void deposit(float amount) {
    printf("\ndeposit: %'g", amount);
    if (amount > 0) {
        float *b = balance();
        *b += amount;
        printf("\nbalance: %'g\n", *b);
    }
}

void withdraw(float amount) {
    float *b = balance();
    printf("\nwithdraw: %'g", amount);
    if (amount > 0 && amount < *b) {
        *b -= amount;
        printf("\nbalance: %'g\n", *b);
    } else {
        printf("\ninsufficient balance!\n");
    }
}
```

```

void main() {
    setlocale(LC_ALL, "");
    deposit(1000);
    deposit(500);
    withdraw(800);
    withdraw(1000);
    deposit(1200);
    withdraw(900);

    putchar('\n');
}

```

```

deposit: 1,000
balance: 1,000

deposit: 500
balance: 1,500

withdraw: 800
balance: 700

withdraw: 1,000
insufficient balance!

deposit: 1,200
balance: 1,900

withdraw: 900
balance: 1,000

```

**ตัวอย่าง 11-3** เป็นการสร้างฟังก์ชันที่ส่งค่ากลับในแบบข้อมูลชนิดสตริง โดยฟังก์ชันดังกล่าวจะรับเลขโดด 0 - 9 เข้ามาทางพารามิเตอร์ และเรากำหนดให้เป็นคำอ่านในภาษาอังกฤษของเลขตัวนั้น เช่น 1 อ่านว่า one เป็นต้น และส่งผลลัพธ์กลับไปในแบบสตริง ส่วนที่ main() เราจะให้ผู้ใช้ใส่ตัวเลขระหว่าง 0 - 9 ทางคีย์บอร์ด และต้องมีการตรวจสอบด้วยว่าตรงตามเงื่อนไขหรือไม่ และถ้าส่งให้แก่ฟังก์ชันเพื่อรับสตริงที่เป็นคำอ่านของเลขตัวนั้น

```

#include <stdio.h>
#include <string.h>
#include <ctype.h>

char *num_to_text(int digit) {
    static char result[20] = "";

    char text[10][20] = {
        "zero", "one", "two", "three", "four",
        "five", "six", "seven", "eight", "nine"
    };

    if (digit >= 0 && digit <= 9) {
        strcpy(result, text[digit]);
    }

    return result;
}

void main() {
    char d_c, w[20];
    int d_i;

    printf("\nEnter 1 digit >>");
    d_c = getchar();
}

```

```

enter 1 digit >>7
7 pronounce seven
-----
enter 1 digit >>3
3 pronounce three

```

```

if (isdigit(d_c)) {
    //แปลงอักษรเป็นตัวเลข โดยหาความแตกต่างของรหัส
    d_i = d_c - 48;

    //เรียกฟังก์ชันพร้อมส่งตัวเลขไปให้
    //แล้วนำผลลัพธ์ไปเก็บตัวแปร
    strcpy(w, num_to_text(d_i));

    printf("\n%d pronounce %s\n", d_i, w);
} else {
    printf("\nplease enter 1 digit\n");
}
}

```

## การใช้พอยน์เตอร์ร่วมกับอาร์เรย์

ในบทที่ผ่านมาแล้ว เราได้เรียนรู้เกี่ยวกับอาร์เรย์กันมาแล้ว ซึ่งเราใช้วิธีการเข้าถึงสมาชิกด้วยเลขลำดับหรรมดา แต่ยังไก่ตาม สำหรับในภาษา C นั้นเรายังสามารถจัดการกับอาร์เรย์ได้อีกรูปแบบนั่นก็คือการใช้พอยน์เตอร์ และเนื่องจากรายละเอียดปลีกย่อยระหว่างพอยน์เตอร์และอาร์เรย์มีค่อนข้างมาก ดังนั้น จึงขอแยกอธิบายเป็นหัวข้อย่อยไปตามลำดับ ดังนี้

### การใช้คำแนะนำของสมาชิกด้วยพอยน์เตอร์

หากเราต้องการใช้พอยน์เตอร์เพื่อเข้าถึงสมาชิกของอาร์เรย์ ก็เริ่มจากการสร้างทั้งตัวแปรพอยน์เตอร์และอาร์เรย์ขึ้นมาก่อน ตามแนวทางเดิมทั้งหมด จากนั้นก็จะดำเนินการในรูปแบบดังนี้

ชื่อพอยน์เตอร์ = &ชื่ออาร์เรย์ [ เลขลำดับสมาชิก ]
---

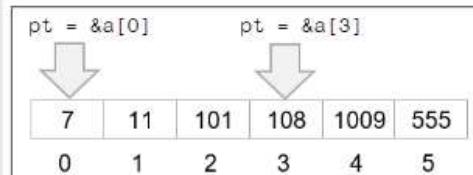
กรณีที่เราต้องการเข้าไปที่สมาชิกตัวแรกของอาร์เรย์ อาจจะบุคคลชื่ออาร์เรย์โดยไม่จำเป็นต้องมีเครื่องหมาย & และเลขลำดับก็ได้ ส่วนหลักการพื้นฐานต่างๆ ก็เหมือนเดิม จึงขอให้ดูจากโค้ดต่อไปนี้เลย

```

int *pt, a[] = {7, 11, 101, 108, 1009, 555};

pt = &a[0];
//ชี้ไปที่ตำแหน่งของสมาชิกลำดับ 0
//หรือเขียนเป็น
//pt = a;

```



```
//หรือถ้าต้องการประกาศพอยน์เตอร์พร้อมกำหนดค่า
//int *pt = &a[0];           แบบที่ 1
//int *pt = a;               แบบที่ 2

pt = &a[3];
//ซึ่งไปที่ตำแหน่งของสมาชิกลำดับ 3
```

ถ้าต้องการอ้างค่าของสมาชิกในลำดับที่พอยน์เตอร์ชี้อยู่ เช่นการอ่านหรือกำหนดค่า ก็ให้ระบุผ่าน \*ชื่อพอยน์เตอร์ เช่นเดิม ดังแนวทางต่อไปนี้ (ใช้พอยน์เตอร์และอาร์เรย์จากໂຄດที่แล้ว)

```
pt = &a[0];
int a0 = *pt;           //a0 = 7

pt = &a[5];
printf("a5 = %d", *pt); //555

print("a[1] = %d", *&a[1]); //a1 = 11

pt = &a[5];
*pt = 500;              //แก้ไขค่าเดิมเป็น 500

int x = *pt + *pt;      //x = 500 + 500
```

## การเลื่อนตำแหน่งของพอยน์เตอร์

การเลื่อนตำแหน่งของพอยน์เตอร์ที่ซึ่งไปยังสมาชิกของอาร์เรย์ สามารถทำได้ง่ายมาก เพียงแค่นำตัวเลขจำนวนเต็มซึ่งเป็นระยะที่เราจะเลื่อนไปมาหากหรือลบออกจากตำแหน่งเดิมที่มันชี้อยู่ ก็จะเป็นการเลื่อนไปยังตำแหน่งใหม่ทันที ซึ่งลักษณะเช่นนี้จะใช้ได้เฉพาะกับพอยน์เตอร์ที่ซึ่งไปยังอาร์เรย์เท่านั้น แต่ใช้กับพอยน์เตอร์ที่ซึ่งไปยังตัวแปรธรรมดาก็ไม่ได้ (จะได้ผลลัพธ์ที่ไม่ถูกต้อง) ดังแนวทางต่อไปนี้

```
int *pt, a[] = {7, 11, 101, 108, 1009, 555, 69, 96};

pt = &a[0];           //int *pt = &a[0]
//pt = a;
pt = pt + 1;
//เลื่อนตำแหน่งจากเดิมไปอีก 1
//ดังนั้น พอยน์เตอร์จะซึ่งไปที่ตำแหน่ง &a[0+1] => &a[1]

pt += 2;
//เลื่อนตำแหน่งจากเดิมไปอีก 2
```

```
// ดังนั้น พอยน์เตอร์จะชี้ไปที่ตำแหน่ง &a[1+2] => &a[3]

pt += 4;
// พอยน์เตอร์จะชี้ไปที่ตำแหน่ง &a[3+4] => &a[7]

pt -= 2;
// พอยน์เตอร์จะชี้ไปที่ตำแหน่ง &a[7-2] => &a[5]

pt -= 5;
// พอยน์เตอร์จะชี้ไปที่ตำแหน่ง &a[5-5] => &a[0]

// pt += 10;
// เกินขอบเขตของอาร์เรย์ อาจได้ค่าที่ผิดพลาด
```

ขออีกว่า การเลื่อนตำแหน่งดังที่กล่าวมานี้ ใช้ได้ผลเฉพาะกับพอยน์เตอร์ที่ชี้ไปยังอาร์เรย์เท่านั้น แต่ไม่สามารถนำไปใช้กับพอยน์เตอร์ที่ชี้ไปยังตัวแปรธรรมดائدี

#### ตัวอย่าง 11-4 แนวทางการเข้าถึงสมาชิกของอาร์เรย์โดยใช้พอยน์เตอร์

```
#include <stdio.h>

void main() {
    int nums[] = {80, 23, 61, 72, 34, 56};
    int n = sizeof(nums) / sizeof(nums[0]);

    for (int *pt = &nums[0]; pt < &nums[n]; pt += 1) {
        printf("%d ", *pt);
    }
}
// ผลลัพธ์: 80 23 61 72 34 56
```

### การใช้พอยน์เตอร์ร่วมกับเครื่องหมาย ++ และ --

เครื่องหมาย ++ (Increment) จะเป็นการเพิ่มค่าของตัวแปรจากเดิมไปอีก 1 และเครื่องหมาย -- (Decrement) จะเป็นการลดค่าของตัวแปรจากเดิมลงไป 1 ซึ่งเราได้เรียนรู้และใช้งานกันมาบ่อยๆ แล้ว ทั้งนี้ เราสามารถนำเครื่องหมายทั้งสองอันนี้มาใช้กับพอยน์เตอร์ได้เช่นเดียวกับตัวแปร แต่บางกรณีเรามักตีความหมายในแบบผิดๆ จันได้ผลลัพธ์ที่คลาดเคลื่อนดังนี้ ในหัวข้อนี้ จึงจะแนะนำแนวทางการเพิ่มหรือลดค่าตัวแปรที่ถูกชี้ด้วยพอยน์เตอร์โดยใช้เครื่องหมาย ++ และ -- ซึ่งหลักการบางอย่างที่เราควรรู้เพิ่มเติม มีดังนี้

- การใช้เครื่องหมาย ++ และ -- ร่วมกับใช้กับพอยน์เตอร์ จะได้ผลเฉพาะกรณีที่ซึ้งไปยังอาร์เรย์เท่านั้น แต่หากเป็นพอยน์เตอร์ที่ซึ้งไปยังตัวแปรธรรมด้า อาจให้ผลลัพธ์ที่ผิดพลาด
- การใช้เครื่องหมาย ++ และ -- ร่วมกับพอยน์เตอร์อาจเกิดผล 2 อย่างคือ
  - ค่า** ของสมาชิกในอาร์เรย์ที่พอยน์เตอร์ซึ้งไป จะเพิ่มขึ้นหรือลดลง 1 ค่า (ซึ้งกับเครื่องหมายที่เลือกและตำแหน่งที่วางเครื่องหมาย)
  - ตำแหน่ง** (ลำดับ) ที่พอยน์เตอร์จะซึ้งไปยังสมาชิกในอาร์เรย์ จะเพิ่มขึ้นหรือลง 1 ลำดับ (ซึ้งกับเครื่องหมาย)
- ถ้าวาง ++ และ -- ไว้หลังพอยน์เตอร์ จะมีผลดังภาพและได้ดังนี้ไป

$*pt++ \left\{ \begin{array}{l} 1 \quad *pt \quad \text{จำนวนค่าสมาชิกที่พอยน์เตอร์ซึ้งไป} \\ 2 \quad pt++ \quad \text{เดือนตำแหน่งที่บันทึกคำนับถัดไป} \end{array} \right.$

<pre>int *pt; int a[] = {7, 11, 108, 1009}; pt = a; //&amp;a[0]</pre>	
<pre>int x = *pt++;</pre>	วาง ++ ไว้หลังพอยน์เตอร์ ดังนั้น การอ่านค่า (*pt) จะเกิดขึ้นก่อน โดยค่าจะถูกกำหนดให้แก่ x และเลื่อนพอยน์เตอร์ไปซึ้งที่ลำดับถัดไป (pt++)
<pre>printf("x = %d", x); //7</pre>	x จะเก็บค่าที่ได้จาก *pt
<pre>printf("*pt = %d", *pt); //11</pre>	เนื่องจาก pt++ พอยน์เตอร์จะเลื่อนไปยังลำดับถัดไป ซึ่งจากเดิมซึ้งที่ &a[0] ไปซึ้งที่ &a[1] ดังนั้น ค่าที่ได้จาก *pt จึงเป็นค่าของ a[1]
<pre>*pt++;</pre>	การอ่านค่า (*pt) จะเกิดขึ้นก่อน แต่ไม่ได้ถูกนำไปใช้งานใดๆ จากนั้นเลื่อนพอยน์เตอร์ไปซึ้งที่ลำดับถัดไป (pt++)
<pre>printf("**pt = %d", *pt); //108</pre>	เนื่องจาก pt++ พอยน์เตอร์จะเลื่อนไปจากเดิมซึ้งที่ &a[1] ไปซึ้งที่ &a[2] ดังนั้น ค่าที่ได้จาก *pt จึงเป็นค่าของ a[2]
<pre>*pt--;</pre>	การอ่านค่า (*pt) จะเกิดขึ้นก่อน แต่ไม่ได้ถูกนำไปใช้งานใดๆ จากนั้นเลื่อนพอยน์เตอร์ย้อนกลับไปซึ้งที่ลำดับก่อนนี้ (pt--)
<pre>printf("**pt = %d", *pt); //11</pre>	เนื่องจาก pt-- พอยน์เตอร์จะเลื่อนไปจากเดิมซึ้งที่ &a[2] กลับไปซึ้งที่ &a[1] ดังนั้น ค่าที่ได้จาก *pt จึงเป็นค่าของ a[1]

- ถ้าว่าง ++ และ -- ไว้หน้าตัวแปรพอยน์เตอร์ แต่อยู่หลัง \* จะมีผลดังภาพและโค้ดไป

**\*++pt** {

- 1    **++pt**    เลื่อนตำแหน่งไปยังสมาชิกคำนับถัดไป
- 2    **\*pt**    อ่านค่าสมาชิกที่พอยน์เตอร์ชี้ไป

<pre>int *pt; int a[] = {7, 11, 108, 1009}; pt = a; //&amp;a[0]</pre>	
<b>int x = *++pt;</b>	วง ++ ไว้หน้าพอยน์เตอร์ ดังนั้น การเลื่อนพอยน์เตอร์ไปชี้ที่ลำดับถัดไป (++pt) จะเกิดขึ้นก่อน ซึ่งจากเดิมชี้ที่ &a[0] ไปชี้ที่ &a[1] และการอ่านค่า (*pt) จะทำให้หลังโดยค่าจะถูกกำหนดให้แก่ x
<b>printf("x = %d", x); //11</b>	x จะเก็บค่าที่ได้จาก *pt ซึ่งชี้ไปที่ &a[1]
<b>printf("**pt = %d", *pt); //11</b>	เนื่องจาก pt ชี้ไปที่ &a[1] ค่าที่ได้จาก *pt จึงเป็นค่าของ a[1]
<b>*++pt;</b>	การเลื่อนพอยน์เตอร์ไปชี้ที่ลำดับถัดไป (++pt) จะเกิดขึ้นก่อน ซึ่งจากเดิมชี้ที่ &a[1] ไปชี้ที่ &a[2] และการอ่านค่า (*pt) จะทำให้หลัง แต่ไม่ได้ถูกนำไปใช้งานใดๆ
<b>printf("**pt = %d", *pt); //108</b>	เนื่องจาก pt++ พอยน์เตอร์จะเลื่อนไปจากเดิมชี้ที่ &a[1] ไปชี้ที่ &a[2] ดังนั้น ค่าที่ได้จาก *pt จึงเป็นค่าของ a[2]
<b>--*pt;</b>	การเลื่อนพอยน์เตอร์ไปชี้ที่ลำดับก่อนนี้ (--pt) จะเกิดขึ้นก่อน ซึ่งจากเดิมชี้ที่ &a[2] ไปชี้ที่ &a[1] และการอ่านค่า (*pt) จะทำให้หลัง แต่ไม่ได้ถูกนำไปใช้งานใดๆ
<b>printf("**pt = %d", *pt); //11</b>	เนื่องจาก pt-- พอยน์เตอร์จะเลื่อนไปจากชี้ที่ &a[2] กลับไปชี้ที่ &a[1] ดังนั้น ค่าที่ได้จาก *pt จึงเป็นค่าของ a[1]

- ถ้าว่าง ++ และ -- ไว้หน้า \* และตัวแปรพอยน์เตอร์ จะมีผลดังภาพและโค้ดถัดไป

**++\*pt** {

- 1    เพิ่มค่าของสมาชิกที่พอยน์เตอร์ชี้ไปในขณะนั้นไปอีก 1 โดยพอยน์เตอร์ยังชี้ที่เดิม
- 2    ดึงค่าใหม่ของสมาชิกหลังการเพิ่มกลับไป และอัปเดตค่าของสมาชิกในตำแหน่งนั้นเป็นค่าใหม่

int *pt; int a[] = {7, 11, 108, 1009}; pt = a; //&a[0]	
int x = ++*pt;  printf("x = %d", x); //8	pt ชี้ที่ &a[0] ดังนั้น *pt = 7 เมื่อ ++ จึงกลายเป็น 8 แล้วค่อยกำหนดให้แก่ x และค่าของ a[0] ก็กลายเป็น 8 เช่นกัน
printf("*pt = %d", *pt); //8	
++*pt;	พอยน์เตอร์ยังชี้ไปที่ &a[0] ซึ่งเดิม *pt = 8 แล้วหากเพิ่ม 1 กลายเป็น 9 และค่าของ a[0] ก็กลายเป็น 9 เช่นกัน
printf("*pt = %d", *pt); //9	
--*pt;	พอยน์เตอร์ยังชี้ไปที่ &a[0] ซึ่งเดิม *pt = 9 และลดลง 1 กลายเป็น 8 และค่าของ a[0] ก็กลายเป็น 8 เช่นกัน
printf("*pt = %d", *pt); //8	

**ตัวอย่าง 11-5** แนวทางการเข้าถึงสมาชิกของอาร์เรย์ด้วยลูปแบบ while ร่วมกับการเลื่อนพอยน์เตอร์ด้วยเครื่องหมาย ++

```
#include <stdio.h>

void main() {
    int nums[] = {80, 23, 61, 72, 34, 56};
    int n = sizeof(nums) / sizeof(nums[0]);
    int *pt = nums, sum = 0;

    while (pt < &a[n]) {
        sum += *pt++;
    }

    printf("sum = %d", sum);
}
//ผลลัพธ์: 326
```

## การใช้พอยน์เตอร์กับสตริง

หากเราจัดการสตริงในแบบอาร์เรย์ของอักขระดังที่ผ่านมา จะพบกับความยุ่งยากในบางกรณี แต่หากเปลี่ยนมาใช้วิธีการแบบพอยน์เตอร์ อาจช่วยลดความยุ่งยากลงได้ดังรายละเอียดต่อไปนี้

## การใช้พอยน์เตอร์แทนตัวแปรแบบสตริง

ตามปกตินั้น ตัวแปรที่เราใช้เก็บข้อมูลในแบบสตริงนั้นต้องเป็นอาร์เรย์ของอักขระ ซึ่งตัวแปรแบบนี้มีข้อบกพร่องในบางกรณี ดังนี้

```
// ต้องกำหนดค่าในชั้นตอนการประกาศตัวแปร
char str1[] = "Hello";           //ok

// ถ้าทำแบบนี้จะเกิดข้อผิดพลาด
char str2[];
str2 = "Hello";                 //error

// ถ้าจะแก้ไขค่าของตัวแปร ต้องใช้ strcpy()
strcpy(str1, "Sawasdee");      //ok
str1 = "Bye";                  //error

// การส่งค่าแบบสตริงกลับจากฟังก์ชัน
// ต้องกำหนดตัวแปรที่เก็บผลลัพธ์เป็น static char array
char *get_str() {
    static char result[100];
    ...
    return result;
}

// นอกจากนี้ การรับค่าจากฟังก์ชันที่คืนค่าเป็นสตริง
// ก็ต้องใช้ strcpy เช่นกัน ดังนี้
...
char str[20];
str = get_str();                //error
strcpy(str, get_str());         //ok
```

ถ้าเราเปลี่ยนมาสร้างตัวแปรแบบพอยน์เตอร์แทนการใช้รูปแบบอาร์เรย์ของอักขระ จะแก้ปัญหาดังที่กล่าวมาได้ โดยวิธีนี้เราแค่สร้างตัวแปรพอยน์เตอร์ชนิด char (ไม่ต้องมีวงเล็บ []) ต่อท้ายตัวแปร) และต่อไปก็สามารถกำหนดและแก้ไขด้วยเครื่องหมาย = เมื่อไหร่ก็ได้ โดยไม่จำเป็นต้องใช้ strcpy() นอกจากนี้ การส่งผลลัพธ์กลับ ก็ไม่ต้องใช้ static char array อีก เช่น

```
// อาจกำหนดค่าในชั้นตอนการประกาศตัวแปรก็ได้
char *str1 = "Hello";           //ok

// หรือจะกำหนดค่าในภายหลังด้วยเครื่องหมาย = ก็ได้
char *str2;
str2 = "Hello";                 //ok
```

```

//สามารถแก้ไขค่าของตัวแปรในภายหลังด้วยเครื่องหมาย =
str1 = "Sawasdee";           //ok
str2 = "Bye";                //ok

//การส่งค่าแบบสติงกลับจากฟังก์ชัน ตัวแปรที่เก็บผลลัพธ์
//สามารถกำหนดตัวแปรอยู่เตอร์ชนิด char ธรรมด้า
char *get_str() {
    char *result;
    ...
    return result;
}

//สามารถรับค่าจากฟังก์ชันที่คืนค่าเป็นสติง
//ด้วยเครื่องหมาย = ได้เลย
char *str1;
str1 = get_str();           //ok

char *str2 = get_str();     //ok

```

อย่างไรก็ตาม ในกรณีที่เราต้องการนำสตริงมาเชื่อมต่อกับสตริงในตัวแปรซึ่งต้องใช้ `strcat()` จะต้องประกาศตัวแปรในแบบอาร์เรย์ของอักขระตามเดิมจึงจะใช้งานได้ หากใช้แบบ พอยน์เตอร์จะไม่ได้ผลตามต้องการ เช่น

```

char *str1 = "";
char str2[100] = "";
strcat(str1, "test");      //ไม่เกิดผล
strcat(str2, "test");     //ok

```

## การใช้พอยน์เตอร์ร่วมกับอาร์เรย์ของสตริงในแบบ 2 มิติ

กรณีที่เราสร้างอาร์เรย์ของสตริงในแบบอาร์เรย์ 2 มิติ ก็มีปัญหาหรือความยุ่งยากในการ แก้ไขค่าของสมาชิกที่ต้องใช้ `strcpy()` รวมถึงไม่สามารถเพิ่มจำนวนสมาชิกเกินกว่าขนาดเริ่มต้น เช่น

```

//รูปแบบการประกาศพร้อมกำหนดค่าเริ่มต้น
char countries[][][20] = {"thailand", "japan", "korea"};

//ถ้าแค่ประกาศแล้วค่าภายใน ต้องใช้ strcpy
char colors[3][20];
//colors[0] = "red";           //error
strcpy(colors[0], "red");    //ok
strcpy(colors[1], "green");   //ok

```

```
//เพิ่มสมาชิกเกินขนาดเริ่มต้นไม่ได้
strcpy(colors[4], "black");      //error
strcpy(countries[4], "china");   //error
```

แต่ถ้าเปลี่ยนมาใช้วิธีการแบบพอยน์เตอร์ ก็สามารถแก้ปัญหาหรือลดความยุ่งยากลงไปได้ชึ่งวิธีนี้ เราต้องสร้างตัวแปรแบบพอยน์เตอร์ให้มีชนิด char แล้วใช้งานใน [] เพียงอันเดียวต่อท้ายชื่อตัวแปร ซึ่งจะกำหนดสมาชิกตั้งแต่ตอนประกาศ หรือจะกำหนดภายหลังด้วยเครื่องหมาย = ก็ได้ นอกจากนี้ยังสามารถเพิ่มจำนวนสมาชิกใหม่ แม้จะเกินขนาดที่กำหนดไว้ตั้งแต่เริ่มแรกก็ตาม เช่น

```
//รูปแบบการประกาศพร้อมกำหนดค่าเริ่มต้น
char *countries[3] = {"thailand", "japan", "korea"};

//สามารถประกาศตัวแปรล่วงหน้า
//แล้วกำหนดหรือแก้ไขค่าสมาชิกในภายหลังด้วยเครื่องหมาย =
char *colors[3];
colors[0] = "red";           //ok
colors[1] = "green";         //ok

//สามารถเพิ่มสมาชิกเกินขนาดเริ่มต้นได้
colors[4] = "black";         //ok
countries[4] = "china";     //ok
countries[5] = "vietnam";    //ok

//เราสามารถหาขนาดหรือจำนวนสมาชิกได้ตามปกติ
int size = sizeof(colors) / sizeof(colors[0]);
```

**ตัวอย่าง 11-6** จากตัวอย่าง 10-2 ในบทที่ 10 ซึ่งเราเก็บชื่อวันในรอบสัปดาห์ (Sunday - Saturday) พร้อมชื่อลีปประจำวันนั้น ไว้ในอาร์เรย์แบบ 2 มิติ และรับค่าเป็นตัวเลขลำดับของวันในรอบสัปดาห์ (1 - 7) ทางคีย์บอร์ด จากนั้นนำตัวเลขนี้ไปใช้เป็นลำดับในการอ่านค่าจากอาร์เรย์ ในตัวอย่างนี้ เราจะแก้ไข มาจัดเก็บในแบบพอยน์เตอร์ ล้วนขั้นตอนอื่นๆ ก็เหมือนเดิม

```
#include <stdio.h>

void main() {
    char *daynames[7] = {
        "sunday", "monday", "tuesday", "wednesday",
        "thursday", "friday", "saturday"
    };
}
```

```
enter day of week (1 - 7) >>3
day name: tuesday
color: pink
```

```
enter day of week (1 - 7) >>7
day name: saturday
color: purple
```

```

char *colors[] = {
    "red", "yellow", "pink", "green",
    "orange", "blue", "purple"
};

int dayofweek;

printf("\nEnter day of week (1 - 7) >>");
scanf("%d", &dayofweek);

if (dayofweek >= 1 && dayofweek <= 7) {
    dayofweek -= 1;
    printf("\nDay name: %s", daynames[dayofweek]);
    printf("\nColor: %s\n", colors[dayofweek]);
} else {
    puts("Error!");
}

```

**ตัวอย่าง 11-7** จากตัวอย่าง 11-3 ซึ่งเราสร้างฟังก์ชันเพื่อรับค่าเป็นคำอ่านของเลขโดด 0 - 9 โดยจะคืนค่าในแบบสตริง และเมื่อเรียกฟังก์ชันนี้ เราต้องใช้ strcpy เพื่อนำผลลัพธ์ไปเก็บในตัวแปร จึงยุ่งยากเล็กน้อย ดังนั้น ในตัวอย่างนี้ เราจะนำมาปรับปรุงให้ง่ายขึ้นโดยสร้างตัวแปรแบบพอยน์เตอร์ เพื่อรับค่าที่ส่งกลับจากฟังก์ชันด้วยเครื่องหมาย = ได้เลย ส่วนขั้นตอนอื่นๆ ก็มีการปรับเปลี่ยนมาใช้แบบพอยน์เตอร์เช่นกัน ซึ่งขอให้ดูจากโค้ดได้เลย (ผลลัพธ์เหมือนเดิมทั้งหมด)

```

#include <stdio.h>
#include <string.h>
#include <ctype.h>

char *num_to_text(int digit) {
    char *result = "";

    char *text[] = {
        "zero", "one", "two", "three", "four",
        "five", "six", "seven", "eight", "nine"
    };

    if (digit >= 0 && digit <= 9) {
        result = text[digit];
    }
}

```

```

    return result;
}

void main() {
    char d_c;
    int d_i;

    printf("\nEnter 1 digit >>");
    d_c = getchar();

    if (isdigit(d_c)) {
        //แปลงอักษรเป็นตัวเลข โดยหาความแตกต่างของรหัส
        d_i = d_c - 48;

        //เรียกฟังก์ชันพร้อมส่งตัวเลขไปให้
        //แล้วนำผลลัพธ์ไปเก็บตัวแปร
        char *w = num_to_text(d_i);

        printf("\n%d pronounce %s\n", d_i, w);
    } else {
        printf("\nplease enter 1 digit\n");
    }
}

```

**ตัวอย่าง 11-8** เป็นการสร้างฟังก์ชันเพื่อตรวจสอบว่ามีสตริงอยู่ในอาร์เรย์หรือไม่ โดยจะรับพารามิเตอร์เป็นอาร์เรย์ของสตริง ขนาดของอาร์เรย์ และสตริงที่จะตรวจสอบ จากนั้นกวนลูปตามขนาดของอาร์เรย์เพื่อตรวจสอบลมาซิกที่จะรายการ ว่าเท่ากับสตริงที่รับเข้ามาทางพารามิเตอร์ หรือไม่ โดยส่งค่ากลับในแบบบูลีน

```

#include <stdio.h>
#include <stdbool.h>

bool in_str_array(char *str_array[],
                   int size,
                   char *find_str)
{
    char *pt;

    for (int i = 0; i < size; i++) {
        pt = str_array[i];
        if (pt == find_str) {
            return true;
        }
    }
}

```

durian is in array  
Durian is not in array

```

    }

    return false;
}

void main() {
    char *str[] = {
        "apple", "banana", "coconut", "durian"
    };
    int sz = sizeof(str) / sizeof(str[0]);

    putchar('\n');

    char *f1;

    f1 = "durian";
    in_str_array(str, sz, f1) ?
        printf("%s is in array\n", f1) :
        printf("%s is not in array\n", f1);

    f1 = "Durian";
    in_str_array(str, sz, f1) ?
        printf("%s is in array\n", f1) :
        printf("%s is not in array\n", f1);
}

```

สิ่งที่เราได้เรียนรู้ในบทนี้เป็นการเข้าถึงตำแหน่งที่จัดเก็บข้อมูลโดยใช้พอยน์เตอร์ ซึ่งเป็นลักษณะที่สำคัญอย่างหนึ่งของภาษา C ทั้งนี้ เราสามารถนำพอยน์เตอร์มาใช้งานได้ในหลายลักษณะ เช่น การใช้ร่วมกับตัวแปรพื้นฐานทั่วไป และโดยเฉพาะอย่างยิ่งการใช้ร่วมกับอาร์เรย์ และสตริง ซึ่งถือเป็นเป้าหมายหลักของการใช้พอยน์เตอร์



# 12

## รวมตัวอย่างโค้ดเพิ่มเติม ชุดที่ 2

หน้านี้ จะเป็นการรวบรวมตัวอย่างโค้ดชุดที่ 2 โดยใช้พื้นฐานความรู้ระหว่างบทที่ 9-11 เป็นหลัก ซึ่งก็จะมีทั้งตัวอย่างที่ง่ายและค่อนข้างยากไปจนถึงลำบาก ผู้อ่านแต่ละคนที่อาจมีพื้นฐานการเขียนโปรแกรมในระดับที่แตกต่างกัน ทั้งนี้ หากตัวอย่างไหนยากเกินกว่าที่เราจะเข้าใจได้ ก็อาจข้ามไปก่อน ซึ่งเมื่อเราฝึกการเรียนรู้ในระดับหนึ่งแล้ว ค่อยย้อนกลับมาทบทวนในภายหลังก็ได้

### ตัวอย่าง 12-1 ตารางเกม Tic Tac Toe

ถ้าเรามีอาร์เรย์ของอักษรระเบบ 2 มิติที่สามารถประกอบด้วย 'X' และ 'O' เท่านั้น โดยเราจะนำอาร์เรย์ดังกล่าวมาสร้างตารางเลียนแบบเกม Tic Tac Toe ดังนี้

```
#include <stdio.h>

void main() {
    char xo[3][3] = [
        {'X', 'O', 'O'}, //row 1
        {'O', 'X', 'X'}, //row 2
        {'X', 'O', 'X'} //row 3
    };

    char *row_sep = "\n-----+\n";
    //เส้นแบ่งระหว่างแถว

    printf(row_sep);

    for (int row = 0; row < 3; row++) {
        for (int col = 0; col < 3; col++) {
            printf("| %c ", xo[row][col]);
        }
        printf("|\n");
    }
}
```

+	-	+	-	+	-	+
	x		o		o	
+	-	+	-	+	-	+
	o		x		x	
+	-	+	-	+	-	+
	x		o		x	
+	-	+	-	+	-	+

```

    //อักษรจะ ' | ' ใช้เป็นเลื่อนขอบด้านซ้าย
}
printf(" | ");
//ขอบของคอลัมน์สุดท้ายของแต่ละແກ່ວ

printf(row_sep);
}
}

```

### ตัวอย่าง 12-2 หมายเลข 6 หลักของสลากกินแบ่ง

ในตัวอย่าง 8-12 ของบทที่ 8 เราได้เขียนโค้ดเพื่อสมมติว่าเป็นการแสดงหมายเลขของสลากกินแบ่งแต่ละใบ ที่ประกอบด้วยเลข 6 หลัก และด้านล่างของเลขแต่ละตัวจะเป็นอักษรย่อ 3 ตัวของคำอ่านในภาษาอังกฤษของเลขด้านนั้น ดังภาพ



ในตัวอย่างดังกล่าว เราใช้วิธีการสุ่มตัวเลข 0 - 9 เพื่อกำหนดเป็นเลขของแต่ละหลักแล้ว ใช้คำลั่ง switch-case เพื่อเทียบเป็นตัวย่อคำอ่านในภาษาอังกฤษ แต่ในตัวอย่างนี้ เราจะเก็บตัวย่อไว้ในอาร์เรย์ แล้วนำเลขแต่ละหลักที่สุ่มได้ มาใช้เป็นลำดับเพื่ออ่านค่าจากอาร์เรย์ ส่วนหลักการอื่นๆ ก็เหมือนเดิม

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

void main() {
    char *text[] = {
        " ZER ", " ONE ", " TWO ", " THR ", " FOR ",
        " FIV ", " SIX ", " SEV ", " EGT ", " NIN "
    };

    char line1[100] = ""; //บรรทัดที่ 1 (ตัวเลข)
    char line2[100] = ""; //บรรทัดที่ 2 (ตัวย่อของคำอ่าน)
    char str_digit[10];

    srand(time(0));
    rand();
    int x;

```

8	7	2	0	2	9
EGT	SEV	TWO	ZER	TWO	NIN

```

for (int i = 0; i <= 6; i++) {
    x = rand() % 10;
    sprintf(str_digit, " %d ", x);
    strcat(line1, str_digit);
    strcat(line2, text[x]);
}

printf("\n%s\n%s\n", line1, line2);
}

```

**ตัวอย่าง 12-3 พังก์ชันสำหรับสร้างอาร์เรย์เลขสุ่มตามเงื่อนไข**

บางครั้ง เราต้องการสร้างเลขสุ่มมากกว่า 1 จำนวน แต่หากจะสร้างทีละจำนวน ก็ต้องเขียนโค้ดซ้ำซ้อนกันหลายครั้ง ดังนั้น ในตัวอย่างนี้ เราจะสร้างฟังก์ชันที่ให้ผลลัพธ์เป็นอาร์เรย์ของเลขสุ่มซึ่งสามารถระบุพารามิเตอร์เพื่อกำหนดเงื่อนไขของเลขสุ่มได้คือ

- ขนาดหรือจำนวนตัวเลขที่ต้องการ
- ค่าต่ำสุดของเลขแต่ละตัว
- ค่าสูงสุดของเลขแต่ละตัว
- เลขแต่ละตัวเป็นชนิด int

แต่จากปัญหางานประการของการคืนค่ากลับจากฟังก์ชันในแบบอาร์เรย์ที่เราได้เรียนรู้มาแล้วจากบทที่ 10 ในที่นี้เราจะจัดทำโดยให้มีพารามิเตอร์ตัวหนึ่งเป็นแบบพอยน์เตอร์เพื่อซึ่ไปยังผลลัพธ์ จากนั้นในส่วนที่เรียกใช้ฟังก์ชันก็กำหนดอาร์เรย์ผลลัพธ์ที่จะใช้คู่กับพอยน์เตอร์ดังกล่าวเพื่อนำค่าไปใช้งาน (รายละเอียดส่วนนี้สามารถย้อนกลับไปดูได้จากบทที่ 10)

```

#include <stdio.h>
#include <time.h>
#include <stdlib.h>

void array_rand(int size, int min, int max, int *result) {
    srand(time(0));
    //ถ้าค่าต่ำสุดมากกว่าค่าสูงสุด ให้สลับค่ากัน
    //เพื่อป้องคงให้มีผลลัพธ์เกิดขึ้นตามเดิม
    if (min > max) {
        int temp = min;
        min = max;
        max = temp;
    }

    for (int i = 0; i < size; i++) {
        result[i] = min + rand() % (max - min + 1);
    }
}

```

result array: 69 34 25 11 97

```

    }

void main() {
    int size = 5, min = 10, max = 100;
    int a[size];

    array_rand(size, min, max, a); //&a[0]

    printf("\nresult array: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", a[i]);
    }

    putchar('\n');
}

```

#### ตัวอย่าง 12-4 พังก์ชันจัดเรียงอาร์เรย์แบบย้อนกลับ (array reverse)

การจัดเรียงอาร์เรย์แบบย้อนกลับจะเป็นพื้นฐานไปสู่การกระทำอื่น ๆ กับข้อมูลที่มีลักษณะเป็นอาร์เรย์ได้ เช่น การจัดเรียงลดลงแบบย้อนกลับ หรือการจัดเรียงอาร์เรย์จากมากไปหาน้อย หรือจากน้อยไปมาก ถ้าอาร์เรย์มีการจัดเรียงในทิศทางตรงกันข้ามเอาไว้แล้ว เป็นต้น ซึ่งหลักการโดยลังเขปของการเรียงอาร์เรย์แบบย้อนกลับคือ เอาตัวแรกไปกำหนดเป็นตัวสุดท้ายของผลลัพธ์ ( $r$ ) และตัวถัดไปก็กำหนดเป็นตัวรองสุดท้าย เป็นเช่นนี้เรื่อยๆ ไปจนครบทุกตัว ดังภาพ

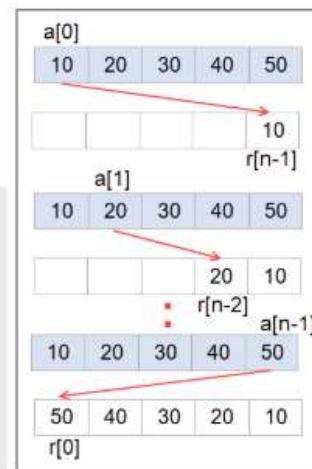
พังก์ชันที่จะสร้างขึ้นในตัวอย่างนี้ จะรับพารามิเตอร์เป็นอาร์เรย์ที่ต้องเรียงย้อนกลับ ขนาด และพารามิเตอร์ที่เป็นพอยน์เตอร์เพื่อชี้ไปยังผลลัพธ์ที่ได้

```

#include <stdio.h>
#include <string.h>

void array_reverse(float arr[],
                   int size,
                   float *result)
{
    int n = size - 1;
    for (int i = 0; i < size; i++) {
        result[i] = arr[n];
        n--;
    }
}

```



```

void array_print(float arr[], int size){
    for (int i = 0; i < size; i++) {
        printf("%g ", arr[i]);
    }
}

void main() {
    float a[] = {11, 7.5, -4, 9.9};
    int size = sizeof(a) / sizeof(a[0]);

    float r[size];
    array_reverse(a, size, r);

    printf("\nbefore reverse: ");
    array_print(a, size);

    printf("\n\nafter reverse: ");
    array_print(r, size);

    putchar('\n');
}

```

before reverse: 11 7.5 -4 9.9  
 after reverse: 9.9 -4 7.5 11

#### ตัวอย่าง 12-5 พงก์ชันจัดเรียงสตริงแบบย้อนกลับ (string reverse)

เนื่องจากสตริงก็คืออาร์เรย์ของอักขระ ดังนั้น การเรียงสตริงแบบย้อนกลับก็ใช้หลักการเดียวกับการเรียงอาร์เรย์ย้อนกลับที่นำเสนอนในตัวอย่างที่แล้ว เพียงแค่เปลี่ยนชนิดข้อมูลเท่านั้น เอง ซึ่งขอให้ดูจากโค้ดได้เลย

```

#include <stdio.h>
#include <string.h>

char *str_reverse(char *str) {
    static char result[] = "";
    int len = strlen(str);
    int n = len - 1;
    for (int i = 0; i < len; i++) {
        result[i] = str[n];
        n--;
    }

    return result;
}

void main() {
}

```

before: string reverse  
 after: esrever gnirts

```

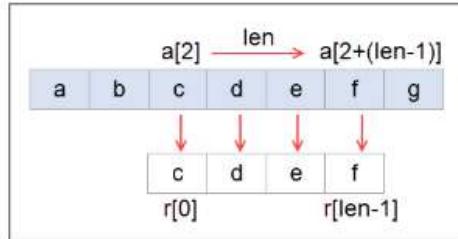
char *s = "string reverse";
char *r = str_reverse(s);

printf("\nbefor: %s", s);
printf("\nafter: %s\n", r);
}

```

**ตัวอย่าง 12-6 พังก์ชันคัดลอกส่วนของสตริงแบบกำหนดความยาว**

ในตัวอย่างนี้ เราจะลองสร้างพังก์ชันเพื่ออ่านหรือคัดลอกบางส่วนของสตริง โดยกำหนดลำดับเริ่มต้นในการคัดลอกและความยาวหรือจำนวนอักขระที่ต้องการซึ่งหลักการโดยสังเขปมีดังนี้



- พังก์ชันที่สร้างขึ้นจะรับค่าพารามิเตอร์ที่ประกอบด้วย สตริงหลัก ตำแหน่งเริ่มต้น และความยาวที่ต้องการ
- ต้องใช้ลูป for เพื่อเข้าถึงอักขระทีละตัว โดยเริ่มจากลำดับที่ระบุ (รับค่าเข้ามาทางพารามิเตอร์) และกำหนดขอบเขตการวนลูปโดยบวกตำแหน่งเริ่มต้นกับความยาว
- อ่านอักขระในช่วงขอบเขตที่กำหนดไปจัดเก็บไว้ในอาร์เรย์ (สตริง) ผลลัพธ์ (r) เมื่อครบแล้วก็ล่งค่ากลับออกไป

```

#include <stdio.h>
#include <string.h>
#include <stdio.h>
#include <string.h>

char *substr_len(char *str, int start, int len) {
    if (len <= 0) {
        return "";
    }

    static char result[] = "";
    int r = 0;
    int stop = start + len;

    for (int i = start; i < stop; i++) {
        result[r] = str[i];
        r++;
    }

    return result;
}

```

```

}

void main() {
    char *h = "abcdefghijkl";
    char *s = substr_len(h, 2, 5);

    printf("%s", s);
}
//ผลลัพธ์ cdefg

```

**ตัวอย่าง 12-7 พังก์ชันคัดลอกส่วนสตริงแบบกำหนดช่วงอักษร**

ในตัวอย่างนี้ เราจะปรับเปลี่ยนจากตัวอย่างที่แล้วโดยสร้างพังก์ชันเพื่อคัดลอกหรืออ่านส่วนของสตริงโดยกำหนดลำดับเริ่มต้นและลำดับสิ้นสุดตามต้องการ (start, stop) ซึ่งหลักการส่วนใหญ่ก็เหมือนกับตัวอย่างที่แล้ว แต่ลิ่งที่จะพิจารณาเพิ่มเติมในตัวอย่างนี้คือ ตามปกติค่า start จะน้อยกว่า stop ซึ่งเป็นการคัดลอกจากซ้ายไปขวา แต่เราจะมีทางเลือกให้ผู้เรียนใช้พังก์ชันสามารถคัดลอกแบบย้อนกลับจากขวาไปซ้ายโดยระบุ start > stop ซึ่งรายละเอียดต่างๆ ให้ดูจากโค้ดต่อไปนี้

```

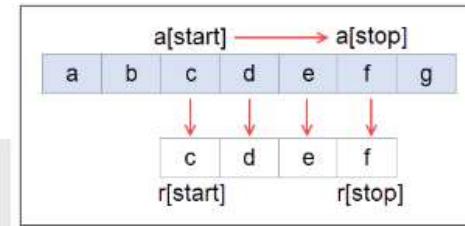
#include <stdio.h>
#include <string.h>

char *substr_range(char *str, int start, int stop) {
    int len = strlen(str);
    if (start > len) {
        return "";
    }

    static char result[100] = "";
    int r = 0;

    if (start < stop) { //คัดลอกจากซ้ายไปขวา
        for (int i = start; i <= stop; i++) {
            result[r] = str[i];
            r++;
        }
    } else { //คัดลอกจากขวาไปซ้าย
        for (int i = start; i >= stop; i--) {
            result[r] = str[i];
            r++;
        }
    }
}

```



```

    return result;
}

void main() {
    char *str = "abcdefghijkl";
    char *s = substr_range(str, 5, 2);

    printf("%s", s);
}

//ผลลัพธ์: fedc

```

**ตัวอย่าง 12-8 พังก์ชันตรวจสอบส่วนขึ้นต้นหรือลงท้ายของสตริง**

สตริงบางอย่าง ส่วนขึ้นต้นหรือลงท้ายมีข้อกำหนดที่แน่นอน เช่น หากเป็น URL หรือ ตำแหน่งของเว็บเพจควรขึ้นต้นด้วย http หรือ https เป็นต้น ดังนั้น ในตัวอย่างนี้ เราจะลองสร้างฟังก์ชันเพื่อตรวจสอบว่าสตริงนั้น ขึ้นต้น (start with) หรือลงท้าย (end with) ด้วยคำที่ระบุ หรือไม่ ซึ่งหลักการก็ไม่มีอะไรยุ่งยาก โดยหากจะตรวจสอบส่วนขึ้นต้น ก็ต้องเทียบอักษรในช่วงต้นของลำดับที่ตรงกันว่าเหมือนกันทั้งหมดหรือไม่ แต่ หากจะตรวจสอบส่วนลงท้าย ก็เทียบอักษรในช่วงท้ายแบบตัวต่อตัวเช่นกันดังแนวทางในภาพข้างมือ

a	b	c	d			
[0]	[1]	[2]	[3]			
a	b	c	d	e	f	g

d	e	f	g			
[n-4]	[n-3]	[n-2]	[n-1]			
a	b	c	d	e	f	g

```

#include <stdio.h>
#include <stdbool.h>
#include <string.h>

bool str_startswith(char *mainstr, char *substr) {
    int len = strlen(substr);

    for (int i = 0; i < len; i++) {
        if (substr[i] != mainstr[i]) {
            return false;
        }
    }

    return true;
}

bool str_endswith(char *mainstr, char *substr) {
    int main_len = strlen(mainstr);
    int sub_len = strlen(substr);
    int start = main_len - sub_len;

```

```

int j = 0;
for (int i = start; i < main_len; i++) {
    if (mainstr[i] != substr[j]) {
        return false;
    }
    j++;
}

return true;
}

void main() {
    char *url = "https://www.developerthai.com";
    char *protocol = "https";
    char *domain = ".co.th";

    printf("\n%s\n", url);

    if (str_startswith(url, protocol)) {
        printf("\nstart with: %s", protocol);
    } else {
        printf("\ndoesn't start with: %s", protocol);
    }

    if (str_endswith(url, domain)) {
        printf("\nend with: %s", domain);
    } else {
        printf("\ndoesn't end with: %s", domain);
    }

    putchar('\n');
}

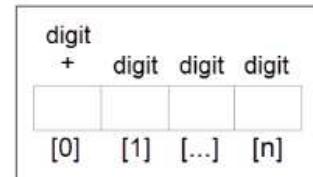
```

https://www.developerthai.com  
start with: https  
doesn't end with: .co.th

### ตัวอย่าง 12-9 พังก์ชันตรวจสอบว่าเป็นเลขจำนวนเต็มหรือไม่

ในบางกรณี เราจำเป็นต้องรับข้อมูลเข้ามาในขณะรันโปรแกรม เช่น การรับผ่านทางคีย์บอร์ด ถ้าเรามีข้อกำหนดว่าข้อมูลดังกล่าวต้องเป็นจำนวนเต็มเท่านั้น จึงจะดำเนินการต่อไปได้ แต่ถ้ายังไงก็ตาม ถ้าผู้ใช้ใส่ข้อมูลที่ไม่เป็นตัวเลขและเรานำไปใช้งานทันทีโดยไม่มีการตรวจสอบ ก็อาจเกิดข้อผิดพลาดจนโปรแกรมต้องหยุดทำงานลงกลางครั้นก็เป็นได้ ดังนั้น ในหัวข้อนี้เราจะลองสร้างฟังก์ชันเพื่อใช้ตรวจสอบข้อมูลที่เป็น String Number ว่าอยู่ในรูปแบบที่เป็นเลขจำนวนเต็มบวก (unsigned int) หรือไม่ เช่น "345" ถือว่าเป็นเลขจำนวนเต็มบวก ในขณะที่ "-555" หรือ "3.14" หรือ "1,234" ไม่ถือว่าเป็นเลขจำนวนเต็มบวก โดยมีหลักการพอลังเขปคือ

- ค่าที่เราตรวจสอบต้องอยู่ในรูปแบบสตริง โดยเราจะใช้วิธีการแบบอาร์เรย์เพื่อเข้าถึงอักขระทีละตัวด้วยลูป for
- ในแต่ละลูป เราจะอ่านอักขระในลำดับนั้นมา และใช้ฟังก์ชัน isdigit() ตรวจสอบว่าเป็นเลข 0 - 9 หรือไม่ ทั้งนี้ ถ้ามีอักขระตัวใดตัวหนึ่งที่ไม่ใช่ 0 - 9 ก็แสดงว่าไม่ใช่เลขจำนวนเต็มบางอย่างแน่นอน
- แต่มีข้อยกเว้นอย่างหนึ่งคือ กรณีที่เป็นเลขจำนวนบวก เราสามารถเขียนเครื่องหมาย + กำกับไว้ข้างหน้าได้ เช่น +10 แม้โดยทั่วไปเราจะไม่นิยมเขียนแบบนี้ แต่ก็เป็นสิ่งที่ทำได้ และในภาษา C ก็รองรับการเขียนแบบนี้ เช่นกัน ดังนั้น เราต้องเพิ่มเติมข้อยกเว้นเข้าไปอีก นั่นคือ การตรวจสอบที่อักขระตัวแรก (ลำดับ 0) ถ้าเป็นเลข 0 - 9 หรือเครื่องหมาย + ก็ให้ถือว่าดำเนินการนั้นเป็นตัวเลข ส่วนลำดับอื่นๆ ต้องเป็น 0 - 9 ตามเดิม



```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
```

```
12345 is unsigned int
0 is unsigned int
-100 is not unsigned int
+1 is unsigned int
3.143 is not unsigned int
```

```
bool is_uint(char *str_num) {
    int len = strlen(str_num);
    char c;

    for (int i = 0; i < len; i++) {
        c = str_num[i];

        //ดำเนินการ อาจเป็น 0 - 9 หรือ + ก็ได้
        if (i == 0 && (isdigit(c) || c == '+')) {
            continue;
        }

        if (!isdigit(c)) {
            return false;
        }
    }

    return true;
}

void main() {
    char *a[] = {
        "12345", "0", "-100",
```

```

    "+1", "3.143", "007"
};

char *str;

for (int i = 0; i < 5; i++) {
    str = (is_uint(a[i])) ? "is" : "is not";
    printf("\n%s %s unsigned int", a[i], str);
}

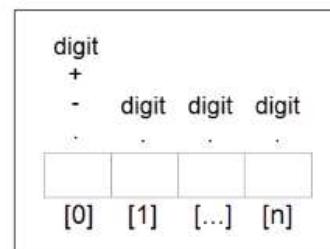
putchar('\n');
}

```

### ตัวอย่าง 12-10 พิ้งก์ชันตรวจสอบว่าเป็นตัวเลขหรือไม่

ในตัวอย่างที่แล้ว เราตรวจสอบเพียงแค่ว่า String Number นั้น อยู่ในรูปแบบของจำนวนเต็มบวกหรือไม่ จึงมีขอบเขตที่จำกัด เพราะข้อมูลที่เรานำมาใช้งาน อาจเป็นจำนวนทศนิยม ทั้งค่าวากและลบ ดังนั้น ในตัวอย่างนี้ เราจะนำมาปรับปรุงการตรวจสอบจากตัวอย่างที่แล้ว โดยให้รองรับค่าที่เป็นตัวเลข (number) โดยลักษณะที่จะถูกจัดว่าเป็นตัวเลข ได้แก่ จำนวนเต็มบวก จำนวนที่มีทศนิยม จำนวนที่ติดลบ เช่น -10 และจำนวนที่ขึ้นต้นด้วยเครื่องหมาย + เช่น +100 ซึ่งสามารถใช้หลักการบางส่วนจากตัวอย่างที่แล้ว แต่การตรวจสอบที่จะเพิ่มเข้ามาคือ

- อักษรตัวแรก อาจเป็น - หรือ + หรือ . ก็ได้
- อักษรตัวถัดไป อาจเป็นตัวเลข หรือ . ก็ได้
- กรณีของข้อมูลที่เป็นทศนิยม จะมี . ได้เพียงอันเดียว เท่านั้น ซึ่งเราต้องเพิ่มการตรวจสอบว่ามี . อยู่ในข้อมูล กี่ครั้ง ถ้ามีมากกว่า 1 อันก็ไม่จัดว่าเป็นตัวเลข



เพื่อลดความยุ่งยาก ในที่นี้จะไม่รองรับการเขียนแบบ exponential (e) แต่ถ้าผู้อ่านเข้าใจ หลักการต่างๆ ของตัวอย่างนี้พอแล้ว จะให้รองรับด้วยก็ได้ โดยเขียนโค้ดเพิ่มเติมอีกเล็กน้อย

```

#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>

bool is_number(char *str_num) {
    int len = strlen(str_num);
    char ch;
    bool found_dot = false;

```

```

-0.0 is a number
-.001 is a number
127.0.0 is not a number
+1 is a number
++1 is not a number
256GB is not a number
5-- is not a number
4x4 is not a number
99. is a number

```

```

for (int i = 0; i < len; i++) {
    ch = str_num[i];

    //ตัวหนึ่งแรก อาจเป็นตัวเลข, -, + หรือ .
    //ถ้าเป็น . ให้กำหนดสถานะว่าพบ . แล้ว
    //โดยจะใช้ล้ำหนึ่งตรวจสอบในตัวหนึ่งต่อๆ ไป
    //ซึ่งจะพบ . อีกไม่ได้
    if (i == 0) {
        if (isdigit(ch) || ch == '-' || ch == '+') {
            continue;
        } else if (ch == '.') {
            found_dot = true;
            continue;
        } else {
            return false;
        }
    } else {
        if (isdigit(ch)) {
            continue;

            //ถ้าพบ . ที่ตัวหนึ่งได้ และยังไม่เคยพบมาก่อน
            //ให้กำหนดสถานะว่าพบ . แล้ว
            //โดยจะใช้ล้ำหนึ่งตรวจสอบในตัวหนึ่งต่อๆ ไป
            //ซึ่งจะพบ . อีกไม่ได้
        } else if (ch == '.' && !found_dot) {
            found_dot = true;
            continue;
        } else {
            return false;
        }
    }
}

return true;
}

void main() {
    char *a[] = {
        "12345", "5,555", "00", "-100", "3.143",
        ".357", "-0.0", "-.001", "127.0.0", "+1",
        "++1", "256GB", "5--", "4x4", "99."
    };

    int size = sizeof(a) / sizeof(a[0]);
    char *str;

    for (int i = 0; i < size; i++) {
        str = (is_number(a[i])) ? "is" : "is not";
}

```

```

        printf("\n%s %s a number", a[i], str);
    }
    putchar('\n');
}

```

### ตัวอย่าง 12-11 กฎการตั้งรหัสผ่าน (1)

ถ้าเรามีกฎในการตั้งรหัสผ่านว่า ต้องประกอบด้วย a - z หรือ A - Z หรือ 0 - 9 เท่านั้นและจำนวนอักษรทั้งหมดรวมกันต้องอยู่ระหว่าง 6 - 20 ตัว ซึ่งเราจะเข้าถึงอักษรที่ลະตัวในรหัสผ่านแล้วใช้ฟังก์ชัน `isalnum()` ของภาษา C เพื่อตรวจสอบว่าเป็น a - z หรือ A - Z หรือ 0 - 9 หรือไม่ถ้ามีตัวใดที่ไม่ใช่ ก็ถือว่าไม่ตรงตามข้อกำหนดที่ตั้งเอาไว้ โดยในตัวอย่างนี้ให้เราสรุปรหัสผ่านทางคีย์บอร์ด แล้วตรวจสอบตามหลักการที่กล่าวมา

```

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdbool.h>
#include <stdlib.h>

void main() {
    char pw[20];
    printf("\nEnter 6-20 chars password");
    printf("\nAccept only a-z, A-Z, 0-9 >>");
    gets(pw);

    int len = strlen(pw);
    if (len < 6 || len > 20) {
        printf(
            "\nError! Password must has 6-20 chars\n"
        );
        exit(0);
    }

    bool pw_valid = true;
    for (int i = 0; i < len; i++) {
        if (!isalnum(pw[i])) {
            pw_valid = false;
            break;
        }
    }

    (pw_valid) ?
        printf("\npassword is valid\n") :
        printf("enter 6-20 chars password\n");
        accept only a-z, A-Z, 0-9 >>p@$$w0rd
        error! password contains invalid char

```

enter 6-20 chars password  
 accept only a-z, A-Z, 0-9 >>p@\$\$w0rd  
 error! password contains invalid char

enter 6-20 chars password  
 accept only a-z, A-Z, 0-9 >>qwerty456  
 password is valid

```

    printf(
        "\nerror! password contains invalid char\n"
    );
}

```

### ตัวอย่าง 12-12 กฎการตั้งรหัสผ่าน (2)

ถ้าเรามีกฎในการตั้งรหัสผ่านว่าเป็นต้นเช่นเดียวกับตัวอย่างที่แล้ว และมีข้อกำหนดเพิ่มเข้ามาอีกบางส่วน คือ

- ต้องประกอบด้วย a - z หรือ A - Z หรือ 0 - 9 เท่านั้น
- จำนวนอักษรทั้งหมดรวมกันต้องอยู่ระหว่าง 6 - 20 ตัว
- ต้องประกอบด้วยตัวพิมพ์เล็ก (a - z) อย่างน้อย 2 ตัว
- ต้องประกอบด้วยตัวพิมพ์ใหญ่ (A - Z) อย่างน้อย 2 ตัว
- ต้องประกอบด้วยตัวเลข (0 - 9) อย่างน้อย 2 ตัว

ซึ่งเราใช้หลักการพื้นฐานเช่นเดียวกับตัวอย่างที่แล้ว โดยเข้าถึงอักษรที่ละตัวในรหัสผ่านแล้วใช้ฟังก์ชัน `isalnum()` ของภาษา C เพื่อตรวจสอบว่าเป็น a - z หรือ A - Z หรือ 0 - 9 หรือไม่ พร้อมกันนี้ก็มีตัวนับจำนวนอักษรและตัวเลข ด้วย เพื่อนำไปตรวจสอบว่าครบตามจำนวนที่กำหนดเอาไว้หรือไม่

```

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdbool.h>
#include <stdlib.h>

void main() {
    char pw[20];
    printf("\nEnter 6-20 chars for password");
    printf("\nAccept only a-z, A-Z, 0-9 >>");
    gets(pw);

    int len = strlen(pw);
    if (len < 6 || len > 20) {
        printf("\nError! pswd must contain 6-20 chars\n");
        exit(0);
    }

    int lower = 0, upper = 0, digit = 0;
    for (int i = 0; i < len; i++) {
        if (islower(pw[i])) lower++;
        if (isupper(pw[i])) upper++;
        if (isdigit(pw[i])) digit++;
    }

    if (lower < 2 || upper < 2 || digit < 2) {
        printf("err! pswd must contain 2+ upper letters\n");
    } else {
        printf("enter 6-20 chars for password\n");
        printf("Accept only a-z, A-Z, 0-9 >>qwertY\n");
        printf("err! pswd must contain 2+ digits\n");
    }
}

enter 6-20 chars for password
Accept only a-z, A-Z, 0-9 >>qwertY
err! pswd must contain 2+ digits

```

```

enter 6-20 chars for password
Accept only a-z, A-Z, 0-9 >>qwertY
err! pswd must contain 2+ digits

```

```

enter 6-20 chars for password
Accept only a-z, A-Z, 0-9 >>qwertY
err! pswd must contain 2+ digits

```

```

enter 6-20 chars for password
Accept only a-z, A-Z, 0-9 >>qwertY01
ok! pswd is valid

```

```

bool pw_valid = true;
char c;
int n_lw = 0, n_up = 0, n_dg = 0;

for (int i = 0; i < len; i++) {
    c = pw[i];
    if (!isalnum(c)) {
        pw_valid = false;
        break;
    } else if (islower(c)) {
        n_lw++;
    } else if (isupper(c)) {
        n_up++;
    } else if (isdigit(c)) {
        n_dg++;
    }
}

if (!pw_valid) {
    printf("\nerr! pswd contains invalid char\n");
} else if (n_lw < 2) {
    printf(
        "\nerr! pswd must contain 2+ lower letters\n"
    );
} else if (n_up < 2) {
    printf(
        "\nerr! pswd must contain 2+ upper letters\n"
    );
} else if (n_dg < 2) {
    printf(
        "\nerr! pswd must contain 2+ digits\n"
    );
} else {
    printf("\nok! pswd is valid\n");
}
}

```

**ตัวอย่าง 12-13 หาค่า checksum ของ ISBN-13**

หมายเลข ISBN-13 ของหนังสือแต่ละเล่ม จะประกอบด้วยเลข 13 หลัก เช่น 9786160833979 โดยตัวสุดท้าย (ขวาสุด) เรียกว่าค่า checksum ซึ่งมีสูตรในการคำนวณคือ

$$10 - (d_1 + 3d_2 + d_3 + 3d_4 + d_5 + 3d_6 + d_7 + 3d_8 + d_9 + 3d_{10} + d_{11} + 3d_{12}) \% 10$$

### ขั้นตอนการคิดโดยสังเขปมีดังนี้

- เราต้องรับค่า ISBN ทางคีย์บอร์ดเป็นเลข 12 หลักแรก
- วนลูป 12 ครั้งเพื่ออ่านตัวเลข ISBN โดยในแต่ละลูปให้อ่านตัวเลขของตำแหน่งนั้นมา
- ถ้าเป็นตัวเลขในหลักคู่ เช่น 2, 4, ... ให้คูณด้วย 3 ก่อนบวกเพิ่มในผลรวม แต่หากเป็นเลขในหลักคี่ เช่น 1, 3, ... ให้บวกเลขนั้นเข้ากับผลรวมได้เลย
- เมื่อได้ผลรวมของเลขครบทุกหลัก ให้หารแบบเอาค่าที่เหลือด้วย 10
- นำผลหารไปลบออกจาก 10 ถ้าได้ค่า checksum เป็น 10 ให้แทนด้วยเลข 0 แต่ถ้าเป็นเลขอื่นๆ ก็ใช้เป็นค่า checksum ได้เลย
- เมื่อนำค่า checksum ไปต่อท้ายเลข ISBN 12 หลักแรกที่เราใช้คำนวณ ก็ได้ค่าเป็น ISBN-13 ตามต้องการ



```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

void main() {
    char isbn[13];
    printf("\nEnter 12 first-digits of isbn >>");
    gets(isbn);

    int len = strlen(isbn);
    if (len != 12) {
        printf("\nerr! invalid isbn\n");
        exit(0);
    }

    char c;
    int d, sum = 0;

    for (int i = 0; i < len; i++) {
        c = isbn[i];
        //ถ้าไม่ใช่ตัวเลข ให้หยุดทำงาน
        if (!isdigit(c)) {
            printf("\nerr! invalid isbn\n");
            exit(0);
        }
    }
}
```

enter 12 first-digits of isbn >>978616083397

ISBN-13: 9786160833979

enter 12 first-digits of isbn >>978616084515

ISBN-13: 9786160845156

```

d = c - 48;      //char => int

//เลขหลักคี่ ให้นำงเข้ากับผลรวมได้เลย
if ((i + 1) % 2 != 0) {
    sum += d;
}

//เลขหลักคู่ ให้คูณ 3 ก่อนนำงเข้ากับผลรวม
else {
    sum += 3 * d;
}

}

//หาค่า checksum โดยนำผลรวมของเลขทุกหลัก
//มาหารແບນເຂາค່າທີ່ເຫຼືອດ້ວຍ 10
int checksum = 10 - (sum % 10);

//ຕ້າໄດ້ຄ່າທີ່ເຫຼືອເປັນ 10 ໃຫ້ຄ່າ checksum ເປັນ 0
//ຕ້າໄດ້ເລີ້ມອື່ນໆ ໃຫ້ໃຊ້ເປັນຄ່າ checksum ໄດ້ເລີຍ
checksum = (checksum == 10) ? 0 : checksum;

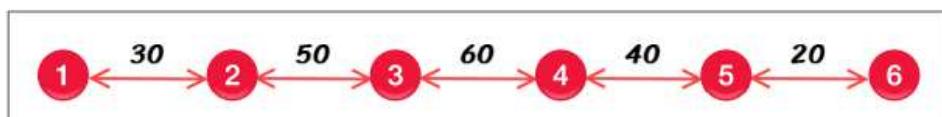
//ນຳ checksum ໄປຕ່ອທ້າຍ ISBN 12 ລັກແຮກໃຫ້ເປັນ ISBN-13
printf("\nISBN-13: %s%d\n", isbn, checksum);
}

```

การทดสอบ ให้ผู้อ่านลองใส่ค่าโดยลงนามหมายเลข ISBN จำนวน 12 หลักแรกของหนังสือ เล่มใดก็ได้ที่มีอยู่แล้ว (เพื่อจะได้เปรียบเทียบความถูกต้อง) หลังจากนั้นทดสอบ ให้ดูว่าหลักที่ 13 ของผลลัพธ์ที่ได้ ตรงกับที่ปรากฏบนหนังสือเล่มนั้นหรือไม่

#### ตัวอย่าง 12-14 ราคาตัวรถไฟ

สมมติว่า ราคาตัวโดยสารรถไฟระหว่างสถานี 1, 2, ..., 6 เป็นดังนี้



ให้เรารับค่าเป็นสถานีต้นทางและปลายทาง และคำนวณราคาตัวรถไฟของช่วงดังกล่าวว่า เป็นเท่าใด เช่น ถ้าสถานีต้นทางคือ 2 และสถานีปลายทางคือ 5 แสดงว่าราคาตัวเท่ากับ  $50 + 60 + 40 = 150$  โดยใช้แนวทางดังต่อไปนี้

- แม้จะทำได้หลายวิธี แต่เพื่อให้สอดคล้องกับเนื้อหาที่เราได้ศึกษามาในบทก่อน ๆ นี้ เราจะใช้วิธีการจัดเก็บข้อมูลราคาตัวรถไฟในแบบอาร์เรย์ โดยให้สมาชิกแต่ละตัวเป็นราคาตัวจากสถานีนั้นไปยังสถานีถัดไป เช่น

```
ticketPrices = { 0, 30, 50, 60, 40, 20 };
//หมายความว่า จากสถานี 0 => 1 ราคา 0 บาท
//ซึ่งไม่มีสถานี 0 อยู่จริง แต่ทำให้ตรงหลักการของอาร์เรย์
//จากสถานี 1 => 2 ราคา 30 บาท
//จากสถานี 2 => 3 ราคา 50 บาท
//...
//จากสถานี 5 => 6 ราคา 20 บาท
```

- เราจะรับค่าของสถานีต้นทาง (origin) และปลายทาง (destination) เป็นตัวเลข 1 - 6 และใช้ตัวเลขนี้ไปกำหนดช่วงของลูป เพื่อหาผลรวมค่าโดยสารระหว่างสถานี เช่น

```
#เช่น จากสถานี 3 ไป 5 แสดงว่าต้องคิดระหว่างสถานะ (3 => 4) + (4 => 5)
#ticket_prices[3] + ticket_prices[4] = 60 + 40 = 100 บาท

for (int i = origin; i <= destination; i++) {
    total += ticket_prices[i];
}
```

- แต่ผู้โดยสารอาจนั่งจากสถานีในลำดับย้อนกลับก็ได เช่น จากสถานะ 4 => 1 ซึ่งค่าตัวก็จะเท่ากับระหว่างสถานี 1 => 4 โดยเราแค่สลับหมายเลขสถานีจากต้นทางเป็นปลายทาง และจากปลายทางเป็นต้นทาง และคิดค่าตัวด้วยลูปเดิมได้เลย

```
#include <stdio.h>

void main() {
    //printf("\n(1)--30--(2)--50--(3)--60--");
    //printf("(4)--40--(5)--20--(6)\n");

    //int ticket_prices[] = {
    //    0, 30, 50, 60, 40, 20
    //};

    int ticket_prices[] = {[1]=30, 50, 60, 40, 20};

    int st1, st2;
```

```
enter station 1 (1 - 6) >>5
enter station 2 (1 - 6) >>1

station 5 => 1
total ticket price: 200
```

```

putchar('\n');
//รับหมายเลขสถานีต้นทาง โดยให้วนลูบ
//จนกว่าผู้ใช้จะใส่ค่าที่อยู่ในช่วง 1 - 6
while (1) {
    printf("enter station 1 (1 - 6) >>");
    scanf("%d", &st1);
    if (st1 >= 1 && st1 <= 6) {
        break;
    }
}

//รับหมายเลขสถานีปลายทาง โดยให้วนลูบ
//จนกว่าผู้ใช้จะใส่ค่าที่อยู่ในช่วง 1 - 6
while (1) {
    printf("enter station 2 (1 - 6) >>");
    scanf("%d", &st2);
    if (st2 >= 1 && st2 <= 6) {
        break;
    }
}

printf("\nstation %d => %d", st1, st2);

int tmp;

//ถ้าใส่หมายเลขสถานีปลายทาง น้อยกว่า ต้นทาง
// เช่น ต้นทาง = 5, ปลายทาง = 2
// ก็ให้สลับหมายเลข เพื่อให้นำไปใช้กันลูบ
// ในขั้นตอนต่อไปได้เลย โดยไม่ต้องแยกคิด
// เพราะถึงอย่างไร ค่าตัวไปหรือกลับจะเท่ากันอยู่แล้ว
if (st2 < st1) {
    tmp = st1;
    st1 = st2;
    st2 = tmp;
}

//หาผลรวมตัวโดยสารระหว่างสถานีต้นทาง ถึง ปลายทาง
int total = 0;
for (int i = st1; i <= st2; i++) {
    total += ticket_prices[i];
}

printf("\ntotal ticket price: %d\n");
}

```

**ตัวอย่าง 12-15** วันสำคัญในรอบปี

ในตัวอย่างนี้ เราจะทดลองการจัดเก็บและอ่านค่าจากอาร์เรย์ที่มากกว่า 1 มิติ โดยจะเป็นข้อมูลของวันสำคัญหลากหลายล้วนในรอบปี เช่น วันขึ้นปีใหม่ วันวาเลนไทน์ เป็นต้น ซึ่งวิธีการต่างๆ ขอให้ดูจากโค้ดได้เลย

```
#include <stdio.h>
#include <stdlib.h>

void main() {
    char *month_names[] = {
        [1] = "January", "February", "March", "April",
        "May", "June", "July", "August",
        "September", "October", "November", "December"
    };

    char imp_days[][3][50] = {
        // วันที่, เดือน, ชื่อวันสำคัญ
        {"1", "1", "New Year Day"},
        {"14", "2", "Valentine's Day"},
        {"14", "3", "Pi Day"},
        {"1", "4", "April Fool's Day"},
        {"1", "5", "Labour Day"},
        {"31", "10", "Halloween Day"},
        {"25", "12", "Christmas Day"},
        {"31", "12", "New Year's Eve"}
    };

    int size = sizeof(imp_days) / sizeof(imp_days[0]);

    char *day, *month, *imp_dayname;
    int m;

    for (int i = 0; i < size; i++) {
        day = imp_days[i][0];
        // วันที่ เราแค่แสดงผล จึงไม่จำเป็นต้องแปลงเป็นตัวเลข

        m = atoi(imp_days[i][1]);
        // เดือน เราต้องใช้เป็นเลขจำนวนเพื่ออ่านชื่อเดือนจากอาร์เรย์
        // จึงต้องแปลงจากสตริงเป็นตัวเลข

        month = month_names[m];
        imp_dayname = imp_days[i][2];
    }
}
```

1 January: New Year Day
14 February: Valentine's Day
14 March: Pi Day
1 April: April Fool's Day
1 May: Labour Day
31 October: Halloween Day
25 December: Christmas Day
31 December: New Year's Eve

```

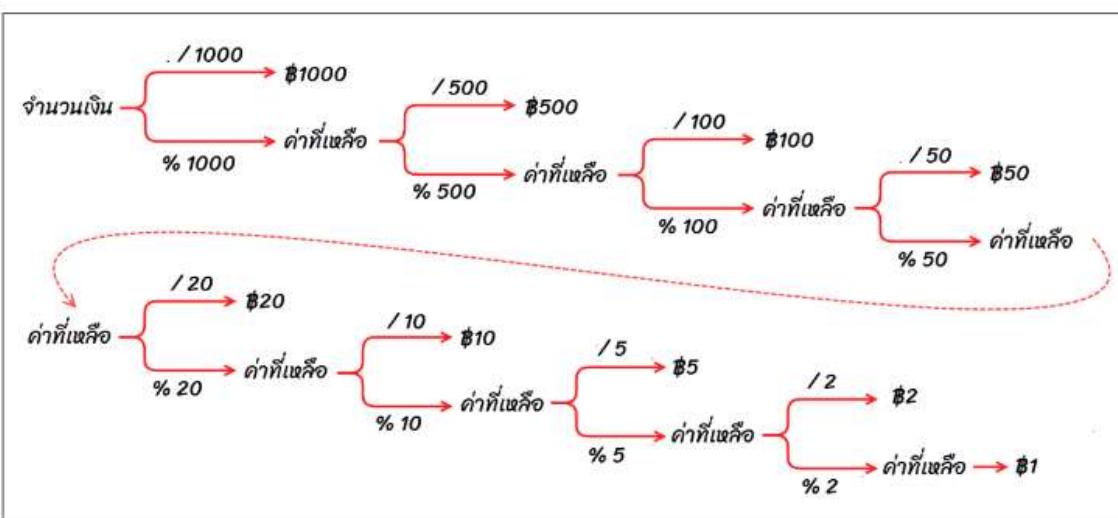
        printf("\n%s %s: %s", day, month, imp_dayname);
    }

    putchar('\n');
}

```

### ตัวอย่าง 12-16 ชนิดธนบัตรและเหรียญตามจำนวนเงิน

ในตัวอย่างนี้ เราจะรับข้อมูลตัวเลขทางคีย์บอร์ด โดยสมมติว่าเป็นจำนวนเงินที่ต้องจ่ายออกไป และตรวจสอบว่าจำนวนเงินดังกล่าว ต้องประกอบด้วยธนบัตรและเหรียญชนิดใดบ้าง โดยให้เริ่มจากค่ามากที่สุดที่เป็นไปได้เสมอ คือ 1000, 500, ..., 1, 50 ล้านคร์ และ 25 ล้านคร์ ซึ่งเราใช้หลักการเดียวกับตัวอย่างชนิดธนบัตรที่กดได้จากตู้ ATM ที่เคยทำมา แต่กรณีนี้จะขับช้อนมากยิ่งขึ้น ขอให้ดูขั้นตอนจากภาพต่อไปนี้



จากภาพ ธนบัตรและเหรียญทุกชนิดจะคำนวณด้วยหลักการเดียวกัน ซึ่งหากเราจะแยกดำเนินการกับธนบัตรและเหรียญทีละชนิด ก็ต้องเขียนโค้ดซ้ำซ้อนกัน ดังนั้น เราจะใช้วิธีการจัดเก็บชนิดธนบัตรและเหรียญไว้ในอาร์เรย์ และวนลูปเพื่อนำค่าจากอาร์เรย์ไปคำนวณแล้วนำค่าที่เหลือไปคำนวณในลูปต่อไปเรื่อยๆ เพื่อหาชนิดธนบัตรหรือเหรียญที่มีค่าลดลงมาตามลำดับ

แต่อย่างไรก็ตาม ปัญหาที่เราต้องเจอก็คือ หากกำหนดอาร์เรย์เป็นชนิด int จะไม่สามารถเก็บค่า 0.5 และ 0.25 ได้ แต่หากให้เป็นอาร์เรย์ชนิด float จะมีปัญหาเมื่อเราใช้เครื่องหมาย % เพื่อหาค่าที่เหลือ เพราะเครื่องหมายนี้ใช้กับจำนวนเต็มเท่านั้น ซึ่งเราจะแก้ไขโดยการจัดเก็บเฉพาะธนบัตรและเหรียญที่เป็นจำนวนเต็มคือ 1000 บาทมานถึง 1 บาท ส่วนเหรียญ 0.25 และ 0.5 จะเป็นໄไปได้มากสุดแค่อย่างละ 1 เหรียญ จะใช้วิธีแยกເອົາສ່ວນທຄນິຍມຂອງจำนวนເງິນທີ່ຕ້ອງຈ່າຍ (ຄໍານີ້) ໄປทำการคำนวณຕ່າງໆກຳໃຫຍງໃຫຍງ

```

#include <stdio.h>
#include <stdlib.h>

void main() {
    //bank note array (เฉพาะจำนวนเต็ม)
    //50 และ 25 สถาบันแบงค์ที่หลัง
    int bna[] = {
        1000, 500, 100, 50, 20, 10, 5, 2, 1
    };

    float amount;
    printf("\namount to pay >>");
    scanf("%f", &amount);

    if (amount <= 0) {
        printf("\nerr! amount must be greater than 0\n");
        exit(0);
    }

    int size = sizeof(bna) / sizeof(bna[0]);

    //เอาส่วนจำนวนเต็มไปคำนวณชนิด 1000 บาท - 1 บาท
    int amount_int = (int)amount;

    int bn;          //bank note
    int remainder = amount_int;

    for (int i = 0; i < size; i++) {
        bn = (int)(remainder / bna[i]);

        if (bn != 0) {
            printf("\nb%d: %d", bna[i], bn);
        }

        remainder %= bna[i];
    }

    //เอาส่วนเศษนิยมไปคำนวณชนิด 25 และ 50 สถาบัน
    float fraction = amount - amount_int;

    if (fraction >= 0.5) {
        printf("\nb0_5: 1");
        fraction -= 0.5;
    }

    if (fraction >= 0.25) {
        printf("\nb0_25: 1");
    }
}

```

amount to pay >>2998.75
b1000: 2
b500: 1
b100: 4
b50: 1
b20: 2
b5: 1
b2: 1
b1: 1
b0_5: 1
b0_25: 1

```

    }
    putchar( '\n' );
}

```

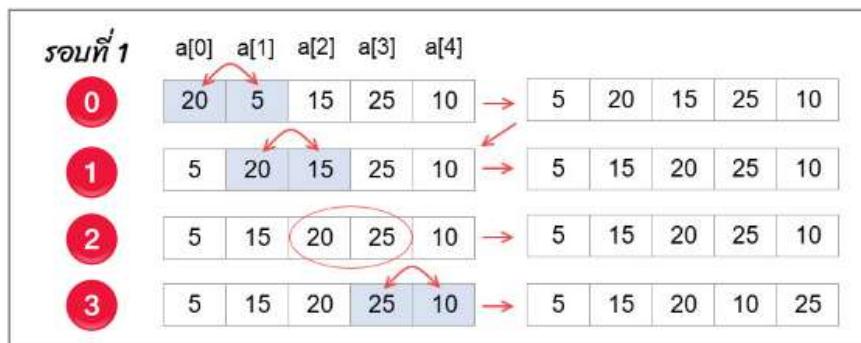
**ตัวอย่าง 12-17 การเรียงลำดับอาร์เรย์แบบ Bubble Sort**

การเรียงลำดับข้อมูลในอาร์เรย์มีให้เลือกหลายวิธี แต่ในที่นี้จะแนะนำเพียง 2 วิธีคือ Bubble Sort และ Selection Sort ซึ่งน่าจะเข้าใจได้ง่ายกว่าวิธีอื่นๆ สำหรับในตัวอย่างนี้จะกล่าวถึง การเรียงลำดับแบบ Bubble Sort เพื่อเรียงข้อมูลชนิดตัวเลขจากน้อยไปหามาก แม้การเขียนโค้ด จะไม่ได้ยืดยาว แต่เราควรรู้จักหลักการในเบื้องต้น ซึ่งจะช่วยให้เราเข้าใจโค้ดได้ง่ายและรวดเร็ว ขึ้น ทั้งนี้ วิธีการจัดเรียงแบบ Bubble Sort คือ การเปรียบเทียบระหว่าง ข้อมูลในตำแหน่งแรก กับตำแหน่งถัดไปที่อยู่ติดกัน ถ้าค่าในตำแหน่งแรกมากกว่า ให้สลับค่ากัน และเลื่อนจุดเปรียบเทียบไปยังลำดับต่อไปเรื่อยๆ จนลินสุดอาร์เรย์ และวนกลับทำซ้ำจนกว่าจะครบตามจำนวนข้อมูล ในอาร์เรย์ ซึ่งการเรียงลำดับแบบนี้ ข้อมูลจะถูกสลับค่าและขยายตำแหน่งไปเรื่อยๆ เปรียบเสมือน พองอากาศ (Bubble) ที่อยู่ในน้ำ ซึ่งมันจะค่อยๆ ลอยขึ้นสู่ผิวน้ำ

สมมติว่าเรามีอาร์เรย์ที่ประกอบด้วยสมาชิกดังนี้

```
int a[] = { 20, 5, 15, 25, 10 };
```

ถ้าเราจะเรียงลำดับจากน้อยไปหามากแบบ Bubble Sort ก็มีหลักการดังนี้



**รอบที่ 1** เราต้องเปรียบเทียบระหว่างสมาชิกในแต่ละลำดับกับสมาชิกตัวถัดไป ถ้าการจัดเรียงไม่ถูกต้อง ให้สลับตำแหน่ง ซึ่งจากภาพข้างบน สามารถอธิบายเพิ่มเติมได้ดังนี้ (หมายเลขอารบิกจากลำดับของตำแหน่งหลักที่ใช้เปรียบเทียบ)

- 0) a[0] > a[1] (20 > 5) ให้สลับค่า
- 1) a[1] > a[2] (20 > 15) ให้สลับค่า

2)  $a[2] < a[3]$  ( $20 < 25$ ) ไม่ต้องสลับ (เรียงถูกแล้ว)

3)  $a[3] > a[4]$  ( $25 > 10$ ) ให้สลับค่า



**รอบที่ 2** ข้อมูลเริ่มแรกจะต่อเนื่องจากรอบที่ 1 และเราກ็ทำเช่นเดียวกับรอบที่ 1 แต่ไม่จำเป็นต้องทำไปจนถึงสมาชิกตัวสุดท้าย (ในที่นี้คือ  $a[4]$  เพราะในรอบแรกนั้น เราขยับและสลับตำแหน่งต่อเนื่องกันมาเรื่อยๆ ดังนั้น สมาชิกตัวสุดท้ายในรอบที่ 1 ย่อมเป็นตัวที่มีค่ามากที่สุดอยู่แล้ว ซึ่งการกระทำในรอบที่ 2 คือ

0)  $a[0] < a[1]$  ( $5 < 15$ ) ไม่ต้องสลับค่า (เรียงถูกแล้ว)

1)  $a[1] < a[2]$  ( $15 < 20$ ) ไม่ต้องสลับค่า (เรียงถูกแล้ว)

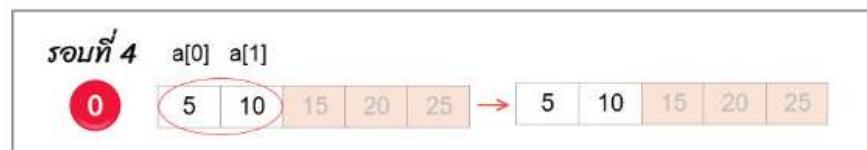
2)  $a[2] > a[3]$  ( $20 > 10$ ) ให้สลับค่า



**รอบที่ 3** ทำเช่นเดียวกับรอบที่ผ่านมา แต่ไม่จำเป็นต้องทำไปจนถึง 2 ลำดับสมาชิกสุดท้าย (ในที่นี้คือ  $a[3]$  และ  $a[4]$ ) เพราะในรอบก่อนนี้ เราขยับและสลับตำแหน่งต่อเนื่องกันมาเรื่อยๆ ดังนั้น สมาชิก 2 ตัวสุดท้ายย่อมจัดเรียงไว้ถูกต้องแล้ว ซึ่งการกระทำในรอบที่ 3 คือ

0)  $a[0] < a[1]$  ( $5 < 15$ ) ไม่ต้องสลับค่า (เรียงถูกแล้ว)

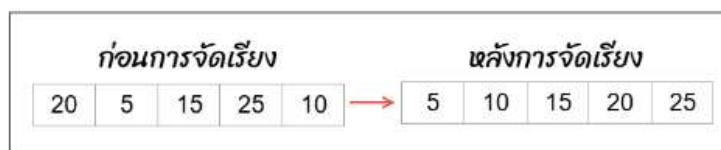
1)  $a[1] > a[2]$  ( $15 > 10$ ) ให้สลับค่า



**รอบที่ 4** ทำเช่นเดียวกับรอบที่ผ่านมา แต่ไม่จำเป็นต้องทำไปจนถึง 3 ลำดับสมาชิก สุดท้าย (ในที่นี้คือ  $a[2]$ ,  $a[3]$  และ  $a[4]$ ) เพราะในรอบก่อนนี้ เราขยับและสลับตำแหน่งต่อเนื่อง กันมาเรื่อยๆ ดังนั้น สมาชิก 3 ตัวสุดท้ายย่อมจัดเรียงไว้ถูกต้องแล้ว ซึ่งการกระทำในรอบที่ 4 คือ

- 0)  $a[0] < a[1]$  ( $5 < 10$ ) ไม่ต้องสลับค่า (เรียงถูกแล้ว)

หลังจากการจัดเรียงครบทุกรอบตามที่กล่าวมา ข้อมูลจะถูกจัดเรียงจากน้อยไปมากในลักษณะดังภาพ



สำหรับในโค้ดถัดไป จะสร้างเป็นฟังก์ชันการจัดเรียงชื่อ `bubble_sort()` โดยรับพารามิเตอร์ เป็นพอยน์เตอร์ชนิด `float` เพื่อให้ใช้ได้กับทั้งข้อมูลเลขจำนวนเต็มและเลขทศนิยม

```
#include <stdio.h>

void bubble_sort(float *a, int size) {
    float temp;

    // ลูปซ้ำนокวนตามจำนวนสมาชิกในอาร์เรย์
    for (int i = 0; i < size; i++) {

        // ลูปซ้ำใน วนรอบจากสมาชิกตัวแรก
        // ถึง ผลิต่ำของจำนวนสมาชิก กับ ลำดับของลูปซ้ำนอก
        // (จำนวนตัวสุดท้ายก็ได้ แต่ไม่มีประโยชน์และทำให้ช้า)
        for (int j = 0; j < (size - i) - 1; j++) {

            // ถ้าค่าสมาชิกในลำดับลูปนั้น มากกว่า ลำดับถัดไป
            // ให้สลับตำแหน่ง
            if (a[j] > a[j + 1]) {
                temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }
}
```

```

void array_print(float arr[], int size){
    for (int i = 0; i < size; i++) {
        printf("%g ", arr[i]);
    }
}

void main() {
    float arr[] = {-1, 0, 9, -999, 0, -0, 99};
    int size = sizeof(arr) / sizeof(arr[0]);

    printf("\nbefore: ");
    array_print(arr, size);

    printf("\nafter: ");
    bubble_sort(arr, size);
    array_print(arr, size);

    putchar('\n');

    float arr2[] = { .357, -3.141, 7.11, 10, -0.0, -100};
    size = sizeof(arr2) / sizeof(arr2[0]);

    printf("\nbefore: ");
    array_print(arr2, size);

    printf("\nafter: ");
    bubble_sort(arr2, size);
    array_print(arr2, size);

    putchar('\n');
}

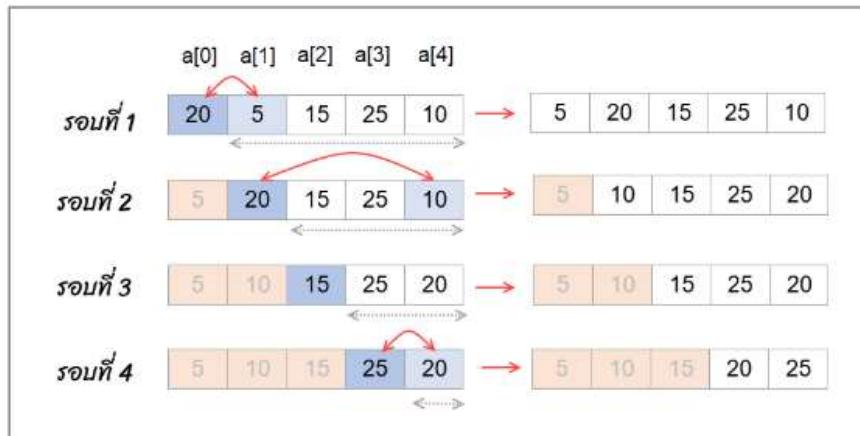
```

before: -1 0 9 -999 0 0 99  
after: -999 -1 0 0 0 9 99

before: 0.357 -3.141 7.11 10 -0 -100  
after: -100 -3.141 -0 0.357 7.11 10

### ตัวอย่าง 12-18 การเรียงลำดับอาร์เรย์แบบ Selection Sort

หลักการเรียงลำดับแบบ Selection Sort คือ จะวนลูปชั้นนอกตามจำนวนสมาชิก แล้วในแต่ละลูปจะอ่านค่าสมาชิกในรอบนั้นมาใช้เป็นตัวเทียบ และวนลูปชั้นในโดยเริ่มจากลำดับถัดไปจากลำดับในลูปชั้นนอก เพื่อค้นหาสมาชิกที่มีค่าน้อยที่สุด และนำมาสลับตำแหน่งกับตัวเทียบทองลูปชั้นนอก และทำเช่นนี้ไปเรื่อยๆ จนกว่าจะครบทั้งหมด ซึ่งการคัดเลือกสมาชิกที่มีค่าน้อยที่สุดมาสลับตำแหน่งนี้เอง จึงเป็นที่มาของชื่อ Selection Sort ทั้งนี้เพื่อความเข้าใจที่ชัดเจนยิ่งขึ้น ขอให้ดูเพิ่มเติมจากภาพและคำอธิบายถัดไป



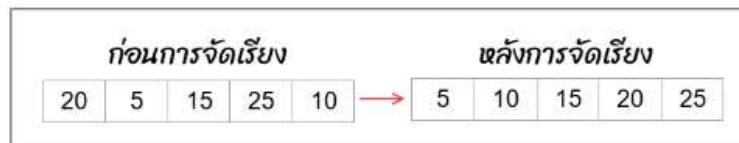
**รอบที่ 1** ให้สมาชิกลำดับ 0 (20) เป็นตัวเทียบ แล้วค้นหาตัวที่มีค่าน้อยสุดที่อยู่ถัดจากตัวเทียบ (ในภาพที่แล้ว ลูกศรเล่นประคือของเขตการค้นหาค่าน้อยที่สุดในรอบนั้น) ซึ่งจะเห็นว่าในช่วงนั้น 5 มีค่าน้อยที่สุด และมีค่าน้อยกว่าตัวเทียบคือ 20 จึงสลับตำแหน่งกัน

**รอบที่ 2** ให้สมาชิกลำดับ 1 (20) เป็นตัวเทียบ แล้วค้นหาตัวที่มีค่าน้อยสุดที่อยู่ถัดจากตัวเทียบ ซึ่งจะเห็นว่าในช่วงนั้น 10 มีค่าน้อยที่สุด และมีค่าน้อยกว่าตัวเทียบคือ 20 จึงสลับตำแหน่งกัน

**รอบที่ 3** ให้สมาชิกลำดับ 2 (15) เป็นตัวเทียบ แล้วค้นหาตัวที่มีค่าน้อยสุดที่อยู่ถัดจากตัวเทียบ ซึ่งจะเห็นว่าในช่วงนั้น 20 มีค่าน้อยที่สุด แต่มีค่ามากกว่าตัวเทียบคือ 15 จึงไม่ต้องสลับตำแหน่ง

**รอบที่ 4** ให้สมาชิกลำดับ 3 (25) เป็นตัวเทียบ แล้วค้นหาตัวที่มีค่าน้อยสุดที่อยู่ถัดจากตัวเทียบ ซึ่งจะเห็นว่าในช่วงนั้น 20 มีค่าน้อยที่สุด และมีค่าน้อยกว่าตัวเทียบคือ 25 จึงสลับตำแหน่งกัน

หลังจากการจัดเรียงครบถ้วนตามที่กล่าวมา ข้อมูลจะถูกจัดเรียงจากน้อยไปมากในลักษณะดังภาพ



```

#include <stdio.h>

void selection_sort(float *a, int size) {
    int temp, cmp, min, min_index;
    //ความจริงไม่ต้องมีตัวแปร cmp เลยก็ได้
    //โดยใช้ a[i] เพียงอย่างเดียว
    //แต่เพื่อให้ตรงกับคำอธิบายที่ผ่านมา
    //จึงมีตัวแปร cmp เข้ามาร่วมด้วย

    for (int i = 0; i < size; i++) {
        cmp = a[i]; //ตัวเทียบในแต่ละรอบ

        //อุปชั้นค้นหาสมาชิกที่มีค่าน้อยที่สุด
        //ในตำแหน่งทั้งหมดที่อยู่ด้านหลังตัวเทียบ
        min = cmp;
        for (int j = i; j < size; j++) {
            if (a[j] < min) {
                min = a[j];
                min_index = j;
            }
        }

        //ถ้าค่าน้อยที่สุดในช่วงนั้น
        //น้อยกว่าค่าของตัวเทียบ ให้สลับกัน
        if (min < cmp) { //cmp = a[i]
            temp = a[i];
            a[i] = min;
            a[min_index] = temp;
        }
    }
}

void array_print(float arr[], int size){
    for (int i = 0; i < size; i++) {
        printf("%g ", arr[i]);
    }
}

void main() {
    float arr[] = {-1, 0, 9, -999, 0, -0, 99};
    int size = sizeof(arr) / sizeof(arr[0]);

    printf("\nbefore: ");
    array_print(arr, size);
}

```

```

printf("\nafte: ");
selection_sort(arr, size);
array_print(arr, size);

putchar('\n');

float arr2[] = { .357, -3.141, 7.11, 10, -0.0, -100 };
size = sizeof(arr2) / sizeof(arr2[0]);

printf("\nbefor: ");
array_print(arr2, size);

printf("\nafter: ");
selection_sort(arr2, size);
array_print(arr2, size);

putchar('\n');
}

```

before: -1 0 9 -999 0 0 99  
after: -999 -1 0 0 0 9 99

before: 0.357 -3.141 7.11 10 -0 -100  
after: -100 -3.141 -0 0.357 7.11 10

### ตัวอย่าง 12-19 เกมทายคำศัพท์ (1)

การสร้างเกมทายคำศัพท์ภาษาอังกฤษซึ่งดัดแปลงจากเกม Hangman โดยจะรับข้อมูลทางคีย์บอร์ด ดังแนวทางด่อไปนี้

1. เราต้องสร้างรายการของคำศัพท์สำหรับให้ทาย โดยเก็บไว้ในอาร์เรย์ และความจำนำวนมากพอสมควร
2. สุ่มตัวอย่างคำจากอาร์เรย์ ขึ้นมา 1 คำ
3. แสดงตัวอักษรให้ผู้ใช้เห็นในรูปแบบ \*\*\*\*\* โดยมีจำนวน \* เท่ากับจำนวนตัวอักษรในคำที่จะให้ทาย (คำที่สุ่มขึ้นมา) เช่น หากสุ่มได้คำว่า tiger ก็แทนด้วย \*\*\*\*\*
4. ให้ผู้ใช้พิมพ้อักษรลงไป 1 ตัว แล้วเราตรวจสอบว่าอักษรนั้นมีอยู่ในคำที่ให้ทายหรือไม่ ถ้ามี ให้นำตัวอักษรนั้นไปแทนที่เครื่องหมาย \* ในตำแหน่งที่ตรงกัน เช่น ถ้าใส่ตัว i ก็จะกลายเป็น \*i\*\*\* เพื่อแจ้งให้ผู้เล่นทราบว่ามีตัวอักษรตำแหน่งใดที่ถูกแล้ว และยังเหลือตัวอักษรที่ตำแหน่งใดอีก ก่อนการรับข้อมูลครั้งต่อไป
5. ทำซ้ำในขั้นตอนข้อ 4 ไปเรื่อยๆ จนกว่าจะใส่ตัวอักษรได้ครบทุกตัว จึงจะเป็นผู้ชนะ

```

#include <stdio.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>

void main() {
    char *vocabs[] = {
        "anaconda", "snake", ...
    };
    int size = sizeof(vocabs) / sizeof(vocabs[0]);

    //สร้างเลขสุ่ม เพื่อนำไปเป็นลำดับ
    //ในการเลือกคำศัพท์ที่จะให้ทาย
    srand(time(0));
    rand();
    int r = rand() % size;

    char *g_vocab = vocabs[r]; //คำศัพท์ที่จะให้ทาย

    int len = strlen(g_vocab);

    char guessed[len];
    //ใช้แทนตัวอักษรในที่คำแทนงที่ทายไปแล้ว
    //และที่ยังเหลือ เช่น *i*e*

    //เริ่มต้น ให้แทนทุกตัวอักษรในคำที่ให้ทาย
    //ด้วย * เช่น tiger => *****
    for (int i = 0; i < len; i++) {
        strcat(guessed, "*");
    }

    printf("\nvocab to guess %s\n", guessed);

    //ให้รับตัวอักษรจากผู้ใช้ ถ้ามีในคำที่ให้ทาย
    //ก็นำไปแทนที่ในคำแทนงนั้น
    //และวนลูปไปเรื่อยๆ จนกว่าจะครบตัวอักษร
    char g_char;
    while (1) {
        printf("\nEnter char >>");
        g_char = getchar();
        getchar(); //clear \n

        //วนลูปเพื่อตรวจสอบว่าตัวอักษรที่รับจากผู้ใช้
        //มีในคำที่ให้ทายหรือไม่ ถ้ามีก็แทนที่ลงในคำแทนงนั้น
        for (int i = 0; i < len; i++) {
            if (g_vocab[i] == g_char) {
                guessed[i] = g_char;
            }
        }
    }
}

```

```

vocab to guess *****

enter char >>a
****a

enter char >>e
*e**a

enter char >>t
*t***a

enter char >>b
*eb*a

enter char >>z
zeb*a

enter char >>r
zebra

you win!

```

```

        }

    }

    //แสดงผลให้เห็นว่ามีตัวอักษรในคำແນ່ງໃດທີ່ຖາຍຄູກແລ້ວ
    //ແລະຍັງເຫຼືອໃນຕຳແໜ່ງໃດອີກ ເຊັ່ນ t*g**
    printf("%s\n", guessed);

    //ດັ່ງທາຍຄູກຄຽນທຸກຕົວອັກສອນ ຕີ່ວ່າເປັນຜູ້ຂະນະ
    //ໃຫ້ອອກຈາກລູບເພື່ອຫຼຸດເກມ
    if (strcmp(g_vocab, guessed) == 0) {
        break;
    }
}

printf("\nyou win!\n");
}

```

#### ตัวอย่าง 12-20 เกมไทยคำศัพท์ (2)

เกมไทยคำศัพท์ในตัวอย่างที่แล้ว เราให้ทายไปเรื่อยๆ จนกว่าจะครบถ้วนทุกตัวอักษร บางกรณี อาจทำให้เกมยืดเยื้อ ซึ่งเราอาจเพิ่มข้อกำหนดว่าจะใส่ตัวอักษรได้ไม่เกินกี่ครั้ง (ทายได้กี่ครั้ง) ซึ่งในที่นี้จะใช้วิธีนับจำนวนตัวอักษรในคำที่ให้ทาย โดยจำนวนครั้งที่ทายได้จะไม่เกิน 2 เท่าของจำนวนตัวอักษร เช่น หากคำนั้นมี 5 ตัวอักษร ก็ให้ทายได้ไม่เกิน 10 ครั้ง เป็นต้น ซึ่งหลักการล้วนใหญ่ ก็เหมือนเดิม เพียงแค่เพิ่มขั้นตอนการนับจำนวนครั้งที่ทายไปแล้ว และตรวจสอบก่อนการรับตัวอักษรครั้งต่อไป ถ้าเกินจำนวนครั้งที่กำหนดก็ให้ถือว่าแพ้ในเกมนั้น (ขอแสดงคำอธิบายเฉพาะส่วนที่เป็นการนับและตรวจสอบจำนวนครั้งของการทาย)

```

#include <stdio.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>

void main() {
    char *vocab[] = {
        "anaconda", "snake", ...
    };
    int size = sizeof(vocab) / sizeof(vocab[0]);

    srand(time(0));
    rand();
    int r = rand() % size;
}

```

```

char *g_vocab = vocabs[r];
int len = strlen(g_vocab);
char guessed[len];

for (int i = 0; i < len; i++) {
    strcat(guessed, "*");
}

//หาได้สูงสุดไม่เกิน 2 เท่าของจำนวนตัวอักษรในคำที่ให้หา
int max_guess = 2 * len, count_guess = 1;
char g_char;

printf("\nvocab to guess %s", guessed);
printf("\nmax guess: %d times\n", max_guess);

while (1) {
    printf("\nguess # %d", count_guess);
    printf("\nenter char >>");
    g_char = getchar();
    getchar();      //clear \n

    for (int i = 0; i < len; i++) {
        if (g_vocab[i] == g_char) {
            guessed[i] = g_char;
        }
    }
    printf("%s\n", guessed);

    //ถ้าหาดูครบถ้วนทุกตัวอักษร
    if (strcmp(g_vocab, guessed) == 0) {
        break;

        //ถ้าหาครบจำนวนครั้งที่กำหนดเอาไว้
        //แต่ยังไม่ครบทุกตัวอักษร ให้ถือว่าแพ้ และหยุดทำงาน
    } else if (count_guess == max_guess) {
        printf(
            "\ngame over! you guessed more than %d times\n",
            max_guess
        );
        exit(0);
    }

    //นับจำนวนครั้งที่หาเพิ่มขึ้นเรื่อยๆ
    count_guess++;
}

printf("\nyou win!\n");
}

```

```

vocab to guess *****
max guess: 10 times

guess #1
enter char >>a
****

guess #2
enter char >>e
****e

guess #3
enter char >>i
****e

guess #4
enter char >>t
****e

guess #5
enter char >>p
****e

```

```

guess #6
enter char >>s
***se

guess #7
enter char >>o
*o*se

guess #8
enter char >>g
*o*se

guess #9
enter char >>h
ho*se

guess #10
enter char >>x
ho*se

game over! you guessed more than 10 times

```

ตัวอย่างโค้ดหลักหลายรูปแบบที่นำเสนอในบทนี้ นอกจากจะเป็นการบทหวานเนื้อหาสำคัญ ให้เกิดความเข้าใจมากยิ่งขึ้นแล้ว ยังช่วยให้เราได้รู้จักรากนิกาย พลิกแพลงในลักษณะต่างๆ แม้ บางกรณีอาจต้องใช้ไหวพริบและประสบการณ์พอสมควร แต่ทั้งหมดนี้ ก็อยู่บนพื้นฐานที่เราได้ เรียนรู้มาแล้วทั้งหมด







## สตรัคเจอร์ ญูเนียน และอี็นเนม

ภาษา C มีทางเลือกในการกำหนดโครงสร้างข้อมูลได้หลายแบบ และลิ้งที่เราจะได้เรียนรู้ในบทนี้คือ โครงสร้างข้อมูลแบบ สตรัคเจอร์ ญูเนียน และอี็นเนม เมื่อโดยทั่วไปเราจะใช้สตรัคเจอร์เป็นหลัก แต่ก็ควรเรียนรู้โครงสร้างชนิดอื่นๆ เอาไว้บ้าง เพราะอาจจำเป็นต้องนำไปใช้งานในบางสถานการณ์ ทั้งนี้โครงสร้างข้อมูลจะช่วยให้เราสามารถกำหนดและจัดองค์ประกอบของข้อมูลอันเดียวกันได้ง่ายและสะดวกรวดเร็วยิ่งขึ้น ซึ่งจะเป็นประโยชน์อย่างมากสำหรับการนำไปประยุกต์ใช้งานในระดับที่สูงขึ้นไป ดังรายละเอียดที่จะกล่าวถึงในบทนี้

### โครงสร้างข้อมูลแบบสตรัคเจอร์ (Structure)

ข้อมูลบางอย่างอาจมีองค์ประกอบปลีกย่อยมากกว่า 1 อย่าง จึงจะครบสมบูรณ์เพียงพอที่จะนำไปใช้งานอีก ได้ เช่น จุดพิกัดทางเรขาคณิต อาจต้องมีทั้งค่าในแนวแกน X และ Y หรือข้อมูลของปฏิทินก็ต้องมีทั้งค่า วันที่ เดือน ปี เป็นต้น นอกจากนี้ยังมีกรณีอื่นๆ อีกมากมาย ซึ่งเราจะได้เรียนรู้เพิ่มเติมกันไปเรื่อยๆ ทั้งนี้ หากเราต้องการจัดเก็บข้อมูลของลิ้งที่มีองค์ประกอบมากกว่า 1 ตามปกติ เราต้องสร้างตัวแปรแยกกัน เช่น

```
//ข้อมูลของจุดพิกัด 1 จุด
float x = 10.5, y = 15.25;

//ตัวมีหลายจุด ก็ต้องสร้างหลายตัวแปร เช่น
float p1_x = 10.5, p1_y = 15.25;
float p2_x = 7, p2_y = 11;
float p3_x = 108, p3_y = 100.9;
```

จากลักษณะของความยุ่งยากในการสร้างตัวแปรแยกตามองค์ประกอบย่อยๆ ของข้อมูลดังที่กล่าวมา ในภาษา C จึงมีวิธีการกำหนดโครงสร้างของข้อมูล (Structure) ที่มีองค์ประกอบปลีกย่อยอีก 1 ลงไปได้อีก โดยใช้รูปแบบดังนี้

```
struct ชื่อโครงสร้างข้อมูล {
    ชนิดข้อมูล ชื่อสมาชิก_1;
    ชนิดข้อมูล ชื่อสมาชิก_2;
    ชนิดข้อมูล ชื่อสมาชิก_3;
    ...
};
```

- **struct** (มาจากคำว่า Structure) เป็นคำส่วนในภาษา C ที่ใช้กำหนดโครงสร้างของข้อมูล (ต่อไปเพื่อป้องกันความลับสนับสนุนคำว่า struct บางครั้งอาจจะเรียกโครงสร้างของข้อมูลแบบทั่วศัพท์ว่า สรุกเจอร์)
- ชื่อโครงสร้างข้อมูล มีหลักการคล้ายการตั้งชื่อตัวแปร แต่ชื่อนี้จะถูกจัดเป็นชนิดข้อมูล (type) ที่เราสร้างขึ้นมาเอง ไม่ใช่ตัวแปร
- ชนิดข้อมูล และชื่อสมาชิก เป็นการกำหนดองค์ประกอบอย่างๆ ของโครงสร้างข้อมูล อันนั้นว่ามีอะไรบ้าง โดยองค์ประกอบแต่ละอย่าง ก็มีหลักการคล้ายกับการประกาศตัวแปรที่ต้องระบุทั้งชนิดข้อมูลและชื่อสมาชิก
- การประกาศสมาชิกทั้งหมดของสรุกเจอร์ต้องอยู่ภายใต้ลักษณะ {} และต้องปิดท้ายด้วยเครื่องหมาย ; เช่นเดียวกับคำสั่งอื่นๆ ในภาษา C

นอกจากรูปแบบการประกาศสรุกเจอร์ดังที่กล่าวมานี้แล้ว เราสามารถปรับเปลี่ยนบางอย่างให้แตกต่างไปจากนี้ได้ ซึ่งจะกล่าวเพิ่มเติมในภายหลัง สำหรับแนวทางการกำหนดสรุกเจอร์ นี้ดังนี้

```
struct point {           //ประกาศสรุกเจอร์ชื่อ point
    float x;            //สมาชิก (องค์ประกอบ) ของ point
    float y;            //สมาชิก (องค์ประกอบ) ของ point
};

struct calendar {        //ประกาศสรุกเจอร์ชื่อ calendar
    int day;
    int month;
    int year;
};

struct person {
    char *name;
    int age;
    char *address;
    char *gender;
};
```

กรณีที่สมาชิกมีชนิดข้อมูลเหมือนกัน สามารถเขียนรวมไว้ในชนิดข้อมูลเดียวกันได้โดยคั่นด้วยเครื่องหมาย , เช่นเดียวกับการประกาศตัวแปร เช่น

```
struct point {
    float x, y;
};

struct calendar { int day, month, year; };

struct person {
    char *name, *address, *gender;
    int age;
};
```

## การกำหนดและอ่านข้อมูลจากสตรัคเจอร์

โครงสร้างข้อมูลหรือสตรัคเจอร์ที่เรากำหนดขึ้นมาดังในหัวข้อที่แล้ว จะถูกจัดเป็นชนิดข้อมูลแบบใหม่ที่เราสร้างขึ้นมาเอง (ไม่ใช่ตัวแปร) ดังนั้น เมื่อจะนำไปใช้งาน จะต้องประกาศตัวแปรให้มีชนิดข้อมูลเป็นชื่อสตรัคเจอร์ที่เรากำหนดขึ้น เช่น

```
struct point {
    float x, y;
};

struct calendar {
    int day, month, year;
};

// แนวทางการประกาศตัวแปรเพื่อใช้อ้างถึง struct
struct point p;           // p คือ ตัวแปรชนิด point
struct calendar cal;     // cal คือตัวแปรชนิด calendar
```

การประกาศตัวแปรในกรณีนี้ ต้องระบุคำว่า struct นำหน้าชื่อโครงสร้างอันนั้นเสมอ เพราะไม่ใช่ชนิดข้อมูลพื้นฐานทั่วไป และในลำดับต่อไป หากเราต้องการกำหนดค่าให้กับสมาชิกของโครงสร้าง ให้ระบุชื่อตัวแปรและชื่อสมาชิกในรูปแบบดังนี้

ชื่อตัวแปร.ชื่อสมาชิก

```
struct point {
    float x, y;
};
```

```

struct point p;
p.x = 10;           //กำหนดค่าให้กับสมาชิก x
p.y = 20.5;         //กำหนดค่าให้กับสมาชิก y

struct person {
    char *name, *address, *gender;
    int age;
};

struct person ps;
ps.name = "Tom Jerry";
ps.address = "...";
ps.gender = "male";
ps.age = 40;

```

การกำหนดค่าสมาชิกที่ละเอียดอ่อนๆ ทำให้เราเห็นว่ามันดูยืดยาวไปหน่อย อาจเปลี่ยนมาใช้วิธีการกำหนดค่าพร้อมกับขั้นตอนการประกาศตัวแปรของโครงสร้าง โดยระบุค่าของสมาชิกไว้ในวงเล็บ {} เช่นเดียวกับค่าของอาร์เรย์ เช่น

```

struct point p = { 10, 20.5 };

struct person ps = {
    "Tom Jerry", "...USA", "male", 40
};

```

การกำหนดค่าด้วยวิธีการในโค้ดข้างบนนี้ แต่ละค่าที่เราระบุ จะถูกนำไปกำหนดให้แก่ สมาชิกตามลำดับที่เราประกาศไว้ในสตรัคเจอร์ ดังนั้น หากเราวางลำดับผิด การจัดเก็บข้อมูลก็เกิดปัญหา หรือมีข้อผิดพลาดเกิดขึ้น หากชนิดข้อมูลของสมาชิกไม่สอดคล้องกับค่าที่กำหนดให้กับมัน ซึ่งเราอาจแก้ปัญหาโดยการระบุชื่อสมาชิกกำกับไว้ด้วย โดยชื่อดังกล่าวให้วางจุด(.) เอาไว้ข้างหน้า ทั้งนี้ เราจะระบุชื่อกำกับสมาชิกทุกตัวหรือเพียงบางตัวก็ได้ กรณีที่เราระบุชื่อกำกับทั้งหมด การเรียงลำดับอาจไม่ตรงกับที่กำหนดไว้ในสตรัคเจอร์ก็ได้ เช่น

```

struct point p = { .x=10, .y=20.5 };

struct calendar cal = { .year=2022, .month=8, .day=26 };

struct person ps = {
    "Tom Jerry", .address=".USA",
    "male", .age=40
};

```

หากไม่จำเป็นต้องกำหนดค่าให้กับสมาชิกจนครบถ้วน แต่สามารถเลือกกำหนดเพียงบางตัว ซึ่งตัวที่เหลือจะมีค่าตามค่าเดิมของชนิดข้อมูลนั้น หรือเราจะไปกำหนดค่าของตัวที่เหลือเพิ่มเติมทีหลังก็ได้ แต่สิ่งที่ต้องระวังคือ หากเราระบุชื่อกับสมาชิกเพียงบางตัว ควรเรียงลำดับให้ตรงกับการจัดเรียงในสตรัคเจอร์ มิฉะนั้นอาจเกิดข้อผิดพลาดในการจับคู่ได้ เช่น หากเป็นชนิดตัวเลขก็จะเป็น 0

```
struct calendar cal = { .year=2022 };
cal.month = 8;
cal.day = 26;
```

นอกจากนี้ เราสามารถลดขั้นตอนให้ลั้นลง โดยรวมการประกาศสตรัคเจอร์และตัวแปรเข้าไว้ด้วยกัน ดังรูปแบบต่อไปนี้

```
struct ชื่อสตรัคเจอร์ {
    ...
} ตัวแปร;
```

วิธีนี้ เราจะประกาศตัวแปรที่ใช้อ้างถึงสตรัคเจอร์อันนั้นโดยวางต่อท้ายบล็อก {} แล้วในลำดับต่อไป ก็ใช้งานผ่านตัวแปรดังกล่าวได้เลย เช่น

```
struct point_float {
    float x, y;
} p;

p.x = 10;
p.y = 30.25;

struct point_int {
    int x, y;
} a, b, c;

a.x = 1;
a.y = 2;
b.x = 5;
c.y = 6;
...

struct calendar {
    int day, month, year;
} cal = { .day=26, .month=8, 2022};
```

```

struct rgb {
    int red, green, blue;
}                                     //ไม่มี ; ปิดท้าย
c1 = {100, 150, 100}, //คันแนตเลตัวแปรด้วย ,
c2 = {0, 0, 255},
c3 = {200, 50, 150}; //ตัวแปรสุดท้าย ปิดท้ายด้วย ;

```

เมื่อเราต้องการอ่านค่าสมาชิกของสตรีกเจอร์ ก็อ้างถึงในแบบเดียวกับการทำหนดค่านั้น คือ ชื่อตัวแปร.ชื่อสมาชิก เช่น

```

struct calendar {
    int day, month, year;
} cal = { .day=26, .month=8, 2022};

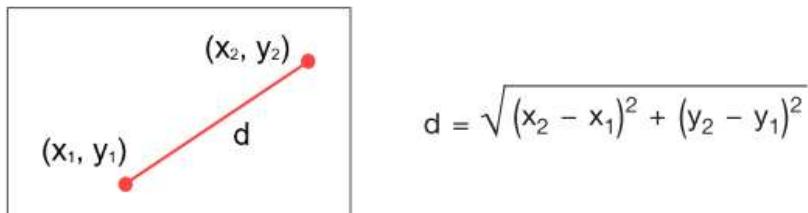
int d = cal.day;

char *month_names[] = { "January", ..., "December" };
char *m = month_name[cal.month - 1];

printf("Today is %d %s %d", d, m, cal.year);

```

**ตัวอย่าง 13-1** สูตรในการคำนวณหาระยะห่างระหว่างจุด 2 จุดคือ



ในตัวอย่างนี้ เราจะสร้างสตรีกเจอร์ของจุดดังแนวทางการใช้ประกอบการอธิบายที่ผ่านมา และนำมาใช้เพื่อกำหนดจุด 2 จุด จากนั้นคำนวณหาระยะห่างระหว่างจุดเหล่านั้นตามสูตรข้างบน

```

#include <stdio.h>
#include <math.h>

void main() {
    struct point {
        float x, y;
    }
    p = {6, 6},
    q = {9, 10};
}

```

```

float dx = q.x - p.x;
float dy = q.y - p.y;

float distance = sqrt(pow(dx, 2) + pow(dy, 2));

printf(
    "\ndistance between (%g, %g) and (%g, %g) is %g\n",
    p.x, p.y, q.x, q.y, distance
);
}

```

distance between (6, 6) and (9, 10) is 5

**ตัวอย่าง 13-2** ในตัวอย่างนี้ จะเป็นการกำหนดสตรัคเจอร์ของปฏิทิน (วัน เดือน ปี) ดังที่ใช้ประกอบการอธิบายที่ผ่านมา โดยรับค่า วัน เดือน ปี ทางคีย์บอร์ดแล้วนำไปกำหนดให้กับสมาชิกแต่ละตัวของสตรัคเจอร์ปฏิทิน จากนั้นให้ตรวจสอบว่า ค่าที่รับเข้ามาอยู่ในขอบเขตที่เป็นไปได้หรือไม่ เช่น วันที่อยู่ในขอบเขตของเดือนนั้นหรือไม่ ถ้าเป็นเดือนที่ 2 (กุมภาพันธ์) ให้ตรวจสอบว่า เป็นปีอธิกสุรทินหรือไม่ ถ้าไม่ใช่วันที่จะต้องไม่เกิน 28 แต่ถ้าใช่ก็ได้สูงสุดถึงวันที่ 29 ซึ่งหลักการหักหนดเหล่านี้ก็เป็นเรื่องเดิมที่เราเคยตรวจสอบในตัวอย่างของบทก่อนๆ นี้มาแล้ว

```

#include <stdio.h>
#include <stdlib.h>

void main() {
    struct calendar {
        int day, month, year;
    } cal;

    printf(
        "\nEnter day month year\n%s",
        "e.g 20 8 2022 >>"
    );
    scanf("%d %d %d", &cal.day, &cal.month, &cal.year);

    if (cal.day < 1 || cal.day > 31) {
        printf("\nDay must be between 1 - 31\n");
        exit(0);
    } else if (cal.month < 1 || cal.month > 12) {
        printf("\nMonth must be between 1 - 12\n");
        exit(0);
    } else if (cal.year < 1 || cal.year > 9999) {
        printf("\nYear must be between 1 - 9999\n");
        exit(0);
    }
}

```

```

//เดือนที่ 4, 6, 9 และ 11 ไม่ได้ไม่เกิน 30 วัน
if (cal.month==4 || cal.month==6 ||
    cal.month==9 || cal.month==11) {

    if (cal.day > 30) {
        printf("\nthis month has 30 days\n");
        exit(0);
    }
}
//ถ้าเป็นเดือนกุมภาพันธ์ ต้องตรวจสอบว่าเป็นปีอิกสูตรหรือไม่
//ถ้าไม่ใช้ก็มิได้ไม่เกิน 28 วัน แต่ถ้าใช้ก็มิได้ไม่เกิน 29 วัน
else if (cal.month == 2) {
    if ((cal.year%400 == 0) ||
        (cal.year%100 != 0 && cal.yea%4 == 0)) {

        if (cal.day > 29) {
            printf("\n... has 29 days\n");
            exit(0);
        }
    } else if (cal.day > 28) {
        printf("\n... has 28 days\n");
        exit(0);
    }
}

printf("\ncalendar ok!\n");
}

```

```

enter day month year
e.g 20 8 2022 >>31 13 2022
month must be between 1 - 12
-----
enter day month year
e.g 20 8 2022 >>29 2 2023
this month of this year has 28 days

```

## การสร้างอาร์เรย์ของสตริงจ่อร์

กรณีที่เราจำเป็นต้องแยกจัดเก็บข้อมูลชนิดสตริงจ่อร์ออกเป็นหลาย ๆ ชุด ก็อาจต้องสร้างตัวแปรแยกตามชุดข้อมูล เช่น หากเรามี 3 ชุดพิกัด ก็ต้องสร้าง 3 ตัวแปรเพื่อใช้แทนแต่ละชุด ดังในหัวข้อที่ผ่านมา ซึ่งบางครั้ง เรามีวิธีการเข้าถึงสมาชิกในรูปแบบเดียวกัน เราจึงอาจต้องเขียนโค้ดช้าๆ ในลักษณะเดียวกัน เช่น

```

struct point { int x, y; }
p1 = {5, 10},
p2 = {15, 20},
p3 = {30, 35};

printf("point 1: (%d, %d)\n", p1.x, p2.y);
printf("point 2: (%d, %d)\n", p2.x, p2.y);
printf("point 3: (%d, %d)\n", p3.x, p3.y);

```

เราอาจแก้ปัญหาความยุ่งยากดังที่กล่าวมาด้วยการสร้างตัวแปรให้เป็นแบบอาร์เรย์ ซึ่งใช้ตัวแปรเพียงตัวเดียวเพื่อกีบชุดข้อมูลทั้งหมด ดังแนวทางต่างๆ ต่อไปนี้

```

struct point { int x, y; };
struct point p[3];
p[0].x = 5;
p[0].y = 10;
p[1].x = 15;
//...
struct point2 {
    int x, y;
} pos[3];

pos[0].x = 5;
pos[0].y = 10;
pos[1].x = 15;
pos[2].y = 30;

struct point3 {
    int x, y;
} ps[] = {
    {5, 10}, {15, 20}, {25, 30}
};

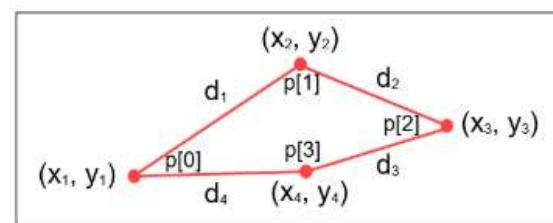
```

การเข้าถึงสมาชิกของโครงสร้างอาร์เรย์ ก็ต้องระบุเลขลำดับในอาร์เรย์แล้วตามด้วยชื่อสมาชิกเป้าหมาย ในรูปแบบ

ตัวแปร [ เลขลำดับ ].ชื่อสมาชิก

เช่น  $p[0].x$ ,  $p[0].y$ ,  $p[1].x$  เป็นต้น ซึ่งสามารถดูเพิ่มเติมได้จากตัวอย่างต่อไปนี้

**ตัวอย่าง 13-3** จากตัวอย่าง 13-1 ซึ่งเป็นการคำนวณหาระยะทางระหว่าง 2 จุด และในตัวอย่างนี้ เราจะสร้างให้มีมากกว่า 2 จุดในแบบอาร์เรย์ และคำนวณหาระยะทางรวมทั้งหมดระหว่างจุดเหล่านั้นคล้ายกับเส้นล้อมรอบ (perimeter) ของรูปหลายเหลี่ยม (polygon) ที่มีความยาวของแต่ละด้านไม่สม่ำเสมอ โดยวิธีในการคำนวณ เราக්ใช้สูตรเดิมเพื่อหาระยะห่างระหว่าง 2 จุด และขยายไปเรื่อยๆ จนถึงจุดสุดท้าย ก็คำนวณหาระยะห่างจุดนั้นกับจุดเริ่มต้น ดังในภาพ



```

#include <stdio.h>
#include <math.h>

void main() {
    const int size = 4;

    struct point {
        float x, y;
    }
    p[] = {
        {0, 0}, {3, 4}, {10, 15}, {13, 6}
    };

    float dx, dy, pm = 0;

    for (int i = 0; i < size; i++) {
        //ถ้าไม่ใช่จุดสุดท้าย
        //ให้คิดระยะระหว่างจุดนั้นกับจุดถัดไป
        if (i < (size - 1)) {
            dx = p[i].x - p[i+1].x;
            dy = p[i].y - p[i+1].y;
        }
        //ถ้าเป็นจุดสุดท้าย
        //ให้คิดระยะระหว่างจุดนั้นกับจุดแรก
        else {
            dx = p[i].x - p[0].x;
            dy = p[i].y - p[0].y;
        }

        pm += sqrt((dx * dx) + (dy * dy));
        //pm += sqrt(pow(dx, 2) + pow(dy, 2));
    }

    printf("\nperimeter = %g\n", pm);
}
//ผลลัพธ์ perimeter = 41.8431

```

**ตัวอย่าง 13-4** ในตัวอย่างนี้ เราจะสร้างสตรัคเจอร์ที่เป็นโครงสร้างอย่างง่ายของข้อมูลลินค์จากนั้นให้ผู้ใช้ใส่ข้อมูลต่างๆ เข้ามาเองทางคีย์บอร์ด โดยให้มีลินค์มากกว่า 1 รายการ ดังนั้นเราต้องจัดเก็บข้อมูลในแบบอาร์เรย์ และก็นำไปแสดงผลในลักษณะที่คล้ายกับตาราง

```

#include <stdio.h>

void main() {
    const int size = 5;

```

```

struct product {
    int id;
    char name[20];
    float price;
    int remaining;
} pd[size];

//รับข้อมูลสินค้าทางคีย์บอร์ด
printf(
    "\nEnter product data\n%s\n",
    "name price remaining",
    "e.g. jean 900 5\n\n"
);

for (int i = 0; i < size; i++) {
    pd[i].id = i + 1;
    printf("product #%-d >>", pd[i].id);
    scanf(
        "%s %f %d",
        pd[i].name, &pd[i].price, &pd[i].remaining
    );
}

//แสดงผลในแบบตาราง
printf(
    "\n%3s\t%-15s\t%5s\t%10s\n%s%s",
    "id", "name", "price", "remaining",
    "-----",
    "-----"
);

for (int i = 0; i < size; i++) {
    printf(
        "\n%3d\t%-15s\t%5g\t%10d",
        pd[i].id, pd[i].name, pd[i].price,
        pd[i].remaining
    );
}
putchar('\n');
}

```

```

enter product data
name price remaining
e.g. jean 900 5

product #1 >>shirt 350 15
product #2 >>shoes 600 8
product #3 >>shorts 299 10
product #4 >>jean 1200 0
product #5 >>polo 300 20

```

<b>id</b>	<b>name</b>	<b>price</b>	<b>remaining</b>
1	shirt	350	15
2	shoes	600	8
3	shorts	299	10
4	jean	1200	0
5	polo	300	20

## การใช้พอยน์เตอร์ร่วมกับสตรักเจอร์

นอกจากการอ้างถึงสมาชิกของสตรักเจอร์ผ่านตัวแปรโดยตรงแล้ว เราสามารถใช้วิธีการของพอยน์เตอร์ได้อีก 1 ทางเลือก ซึ่งอาจเป็นประโยชน์กับการใช้งานในระดับที่สูงขึ้น โดยมีแนวทางดังต่อไปนี้

- เราต้องกำหนดสตรักเจอร์แล้วก็ตัวแปรสำหรับอ้างอิงไปตามปกติ
- สร้างตัวแปรแบบพอยน์เตอร์ โดยให้มีชนิดเป็นสตรักเจอร์ที่เราต้องการซึ่งด้วยพอยน์เตอร์
- ต้องให้พอยน์เตอร์ซึ่งไปที่ตำแหน่ง (address) ของตัวแปรสตรักเจอร์ เมื่อันกับการซึ่งตำแหน่งตัวแปรตามปกติ

```
struct point {
    int x, y
} pos = { 10, 20 };

struct point *pt;           //พอยน์เตอร์ของโครงสร้างชนิด point
pt = &pos;                  //ซึ่งไปที่ตำแหน่งของตัวแปร pos
```

- การอ้างถึงสมาชิกผ่านพอยน์เตอร์ใช้ arrow operator (->) แทนจุด หรืออาจใช้จุด เมื่อันเดิมแต่ต้องเขียนในรูปแบบ (\*pointer).member เช่น

```
pt->x = 50;                //กำหนดหรือแก้ไขค่า x
pt->y = 55;                //กำหนดหรือแก้ไขค่า y

int x = pt->x;            //อ่านค่า x
printf("y: %d", pt->y); //อ่านค่า y

//หรือแบบใช้จุด
int y = (*pt).y;
printf("x: %d", (*pt).x);
```

สำหรับตัวอย่างการใช้พอยน์เตอร์ร่วมกับสตรักเจอร์ ให้ดูร่วมกับหัวข้อถัดไป

## การใช้สตรักเจอร์ร่วมกับฟังก์ชัน

นอกจากการใช้สตรักเจอร์ในแบบตัวแปรทั่วไปแล้ว เราสามารถใช้สตรักเจอร์ร่วมกับฟังก์ชันได้เช่นกัน ทั้งการใช้เป็นพารามิเตอร์ และการส่งข้อมูลกลับออกจากฟังก์ชัน ซึ่งมีแนวทางดังต่อไปนี้

## การใช้สตรัคเจอร์เพื่อเป็นพารามิเตอร์

ในการนี้ที่เราต้องการให้พารามิเตอร์ของฟังก์ชันเป็นข้อมูลชนิดสตรัคเจอร์ ก็มีแนวทางโดยสังเขปดังนี้

- เราควรกำหนดสตรัคเจอร์ไว้นอกฟังก์ชัน เพื่อให้ฟังก์ชันต่างๆ รับรู้การมีอยู่ของสตรัคเจอร์ชนิดนั้นได้
- ที่พารามิเตอร์ของฟังก์ชัน ต้องระบุคำว่า struct พิริมชื่อ (ชนิด) แล้วก็ตามด้วยชื่อพารามิเตอร์ เช่นเดียวกับการประกาศตัวแปรประเภทสตรัคเจอร์

```
struct person {
    char *name;
    int age;
    char *gender;
};

void show_person_info(struct person ps) {
    printf(
        "name: %s, gender: %s, age: %d",
        ps.name, ps.gender, ps.age
    );
}
```

- ส่วนที่เรียกใช้ฟังก์ชัน ก็สร้างตัวแปรของสตรัคเจอร์ชนิดนั้น พร้อมกำหนดค่าให้กับมัน แล้วนำมาเป็นอาร์กิวเมนต์ของฟังก์ชัน

```
void main() {
    struct person p = { "Tony Stark", age:50, "male" };
    show_person_info(p);
}
```

## การส่งค่าแบบสตรัคเจอร์กลับจากฟังก์ชัน

หากเราจะส่งค่าแบบสตรัคเจอร์กลับจากฟังก์ชัน อาจทำในรูปแบบข้อมูลพื้นฐานทั่วไปก็ได้ นั่นคือ กำหนดคำว่า struct ตามด้วยชนิดของสตรัคเจอร์ แล้วก็ชื่อฟังก์ชัน เช่น

```
struct point dosomething(...){
    struct point p = ....
    ...
    return p;
}
```

```
void main() {
    struct point pn = { 12, 34 };
    struct point p_result = dosomething(pn);
    int x = p_result.x;
    ...
}
```

หรืออีกแบบคือ ถ้าในฟังก์ชันมีการเปลี่ยนแปลงค่าของสมาชิก หากเราต้องการให้มันส่งผลมาถึงสตรักเจอร์อันนั้นที่อยู่นอกฟังก์ชัน ให้กำหนดพารามิเตอร์เป็นแบบพอยน์เตอร์ เพื่อให้มันซึ่ไปยังตำแหน่งของข้อมูล หรือเป็นการผ่านค่าแบบ pass by reference นั้นเอง ซึ่งก็ใช้หลักการเดียวกับพอยน์เตอร์ที่เป็นพารามิเตอร์ธรรมดาดังที่เราได้ศึกษามาแล้ว ดังแนวทางด้านไปนี้

```
void func(struct point *pt) {
    pt->x +=1;
    ...
}

void main() {
    struct point p = { 20, 30 };
    func(&p);
    int x = p.x; //21
    ...
}
```

**ตัวอย่าง 13-5** ในตัวอย่างนี้ เราจะสร้างสตรักเจอร์ที่ใช้กำหนดโครงสร้างของตัวเลขนั้นคือ ส่วนจำนวนเต็ม (int part) และส่วนเศษ (fraction part) จากนั้นจะสร้างฟังก์ชัน 2 อันเพื่อใช้ร่วมกับสตรักเจอร์นี้

- ฟังก์ชันแรก รับพารามิเตอร์เป็นสตรักเจอร์แล้วเพิ่มค่าส่วนจำนวนเต็มไป 1 แล้วคืนค่ากลับเป็นสตรักเจอร์เช่นเดิม
- ฟังก์ชันที่สอง รับพารามิเตอร์เป็นสตรักเจอร์เช่นกัน แต่เป็นแบบพอยน์เตอร์ จากนั้นลดค่าส่วนจำนวนเต็มลง 1 โดยไม่คืนค่ากลับ เพราะเป็นพอยน์เตอร์ ซึ่งจะล่งผลถึงส่วนที่อยู่นอกฟังก์ชันโดยอัตโนมัติ

```
#include <stdio.h>

struct number {
    int intpart;
    float fractpart;
```

```

};

struct number inc(struct number n) {
    n.intpart +=1;
    return n;
}

void dec(struct number *pt) {
    pt->intpart -= 1;
}

void main() {
    struct number num = {.intpart=10, .fractpart=0.50};
    printf(
        "\nthen: number = %g",
        num.intpart + num.fractpart
    );

    struct number num2 = inc(num);
    printf(
        "\nafter call inc(): number = %g",
        num2.intpart + num2.fractpart
    );

    dec(&num);
    printf(
        "\nafter call dec(): number = %g\n",
        num.intpart + num.fractpart
    );
}
}

```

then: number = 10.5  
after call inc(): number = 11.5  
after call dec(): number = 9.5

## การสร้างสตรัคเจอร์ซ้อนกัน

สมาชิกของสตรัคเจอร์นอกจากจะเป็นชนิดข้อมูลพื้นฐานทั่วไป เช่น int, float, char แล้ว ยังอาจเป็นแบบสตรัคเจอร์ด้วยกันก็ได้ ซึ่งลักษณะเช่นนี้จึงเหมือนกับการสร้างสตรัคเจอร์ซ้อนกัน นั่นเอง (Nested Structure) โดยมีแนวทางดังต่อไปนี้

สตรัคเจอร์ย่อยที่จะนำไปซ่อนหรือเป็นสมาชิกของสตรัคเจอร์หลักอื่น อาจสร้างไว้ล่วงหน้า ตามวิธีการปกติ แล้วที่สตรัคเจอร์หลักก็นำสตรัคเจอร์ย่อยมากำหนดเป็นสมาชิกดังนี้

```
// สตรัคเจอร์ย่อยที่สามารถนำไปซ่อนในสตรัคเจอร์อื่น
struct location {
    double latitude, longitude;
```

```

};

//สตอร์กเจอร์หลักที่จะนำสตอร์กเจอร์อื่นมาเป็นสมาชิก
struct map {
    char *type;
    int zoom;
    struct location loc;
    //สตอร์กเจอร์ย่อยต้องระบุชื่อตัวแปร
    //เพื่อใช้อ้างถึงสมาชิกของมัน
};

```

การกำหนดสตอร์กเจอร์ย่อยแยกไว้ต่างหากมีข้อดีคือ เราสามารถนำสตอร์กเจอร์อันนั้น (location) ไปใช้งานอย่างอิสระโดยไม่ขึ้นกับสตอร์กเจอร์หลัก (map) ก็ได้ แต่หากเรามีข้อกำหนดว่า ต้องใช้ location เป็นส่วนหนึ่งของ map เท่านั้น ก็ให้นำโครงสร้างของ location มาวางช้อนใน map เช่น

```

struct map {
    char *type;
    int zoom;
    struct location {
        double latitude, longitude;
    } loc;
    //loc คือตัวแปรที่ใช้อ้างถึงสมาชิกของ location
};

```

เมื่อนำสตอร์กเจอร์ที่มีสมาชิกเป็นสตอร์กเจอร์ไปใช้งาน ก็ทำตามหลักการเดิม เพียงแต่หากต้องการอ้างถึงสมาชิกของสตอร์กเจอร์ย่อย ให้ระบุผ่านชื่อตัวแปรของสตอร์กเจอร์ย่อยลงไปด้วย เช่น

```

struct map mp;
mp.type = "satellite";
mp.zoom = 16;
mp.loc.latitude = 50.67890;           //*****
mp.loc.longitude = 10.23456;          //*****

struct map mp2 = {
    "satellite", .zoom=17,
    .loc={ 50.55555, 60.66666 }
};

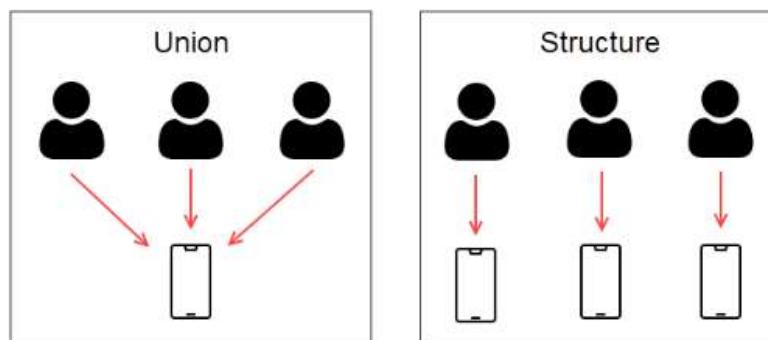
```

## โครงสร้างข้อมูลแบบบัญเนียน (Union)

บัญเนียน (Union) เป็นโครงสร้างข้อมูลอีกแบบหนึ่ง ที่เหมือนกับสตรัคเจอร์เกือบทั้งหมด แต่มีข้อแตกต่างที่สำคัญในบางกรณี นั่นคือ

บัญเนียน	สตรัคเจอร์
กำหนดโครงสร้างด้วยคำว่า union	กำหนดโครงสร้างด้วยคำว่า struct
ใช้ตำแหน่งในการเข้าถึงค่าของสมาชิกร่วมกันเพียงตำแหน่งเดียว (น่าจะเป็นที่มาของคำว่า union)	ตำแหน่งการเข้าถึงค่าของสมาชิกแยกกันในสมาชิกแต่ละตัว
ขนาดการจดพื้นที่หน่วยความจำ เท่ากับ สมาชิกที่มีขนาดใหญ่ที่สุด (มีรายละเอียดเพิ่มเติม)	ขนาดการจดพื้นที่หน่วยความจำ เท่ากับ ผลรวมของสมาชิกทุกตัว

ในเมื่องต้น ขอให้ลองเปรียบเทียบว่า บัญเนียน ก็คล้ายกับการที่สมาชิกในครอบครัวต้องใช้ลิ้งของบางอย่างร่วมกัน เช่น สมมติว่าครอบครัวนั้นมีโทรศัพท์เพียงเครื่องเดียว ทุกคนจะต้องใช้ร่วมกัน ซึ่งในขณะใดขณะหนึ่ง จะมีเพียงสมาชิกคนใดคนหนึ่งเท่านั้นที่ใช้โทรศัพท์ได้ ในขณะที่สตรัคเจอร์ ก็เหมือนกับการที่สมาชิกในครอบครัวที่มีลิ้งนั้นเป็นของตนเองทั้งหมด เช่น มีโทรศัพท์คงเหลือ ดังนั้น จึงสามารถใช้ของใครของมันได้เลย



เนื่องจากข้อกำหนดของบัญเนียนส่วนใหญ่ก็เหมือนกับสตรัคเจอร์ จึงไม่อนำลักษณะพื้นฐานมากล่าวข้าม แต่จะเน้นการเปรียบเทียบให้เห็นเฉพาะลิ้งที่แตกต่างกับสตรัคเจอร์เท่านั้น ดังรายละเอียดต่อไปนี้

- การกำหนดโครงสร้างแบบบัญเนียน ก็ใช้วิธีการเดียวกับสตรัคเจอร์ เพียงแต่เปลี่ยนจากคำว่า struct ไปเป็น union ส่วนลักษณะปลิกย่ออยู่อื่นๆ ที่น่าสนใจ เช่น
  - ◎ หากเป็นข้อมูลชนิดสตริง ควรประกาศตัวแปร (ชื่อสมาชิก) แบบ char array (หากใช้แบบพอยน์เดอร์อาจมีปัญหา)

- ◎ หากใช้วิธีกำหนดค่าเริ่มต้นของสมาชิกด้วยวงเล็บ {} ควรระบุชื่อสมาชิกกำกับ เอาไว้ด้วย

```

union point {
    float x, y;
} pnt;
pnt.x = 20.25; pnt.y = 50.55;

union calendar {
    int day, month, year;
} cal = {.day=1, .month=9, .year=2022 };

union product {
    char name[30];           //ข้อมูลชนิดสตริงให้ใช้ char array
    float price;
    int remaining;
    char size;
};

union product pd = {
    .name="T-Shirt", .price=399.0,
    .remaining=10, .size='M'
};

```

- การอ่านค่าในแต่ละขณะ จะได้ค่าของสมาชิกตัวล่าสุดที่เรากำหนดค่าให้กับมันเท่านั้น ทั้งนี้เพราะสมาชิกของยูเนี่ยนใช้พื้นที่หน่วยความจำร่วมกันดังที่กล่าวมาแล้ว และจะมีเพียงสมาชิกตัวใดตัวหนึ่งที่เข้าถึงได้ในแต่ละขณะ (ปกติจะเป็นตัวล่าสุดที่กำหนดค่าให้กับมัน) ดังการเปรียบเทียบในโค้ดต่อไปนี้

```

union person {
    char name[30];
    int age;
    char gender; // 'm', 'f'
}
ps = { .name="Tom Jerry", .age=30, .gender='m' };
//แสดงว่าค่าล่าสุดที่กำหนดคือ gender = 'm'

printf("name: %s\n", ps.name);      //m
printf("age: %d\n", ps.age);        //109 (ascii(m))
printf("gender: %c\n", ps.gender);  //m

ps.age = 35;
printf("name: %s\n", ps.name);      //#
printf("age: %d\n", ps.age);        //35

```

```

printf("gender: %c\n", ps.gender); //#

strcpy(ps.name, "Tony Stark");
//เราประการชื่อสมाचิกแบบ char array จึงต้องกำหนดค่าด้วย strcpy()

printf("name: %s\n", ps.name); //Tony Stark
printf("age: %d\n", ps.age); //2037280596
printf("gender: %c\n", ps.gender); //T

```

- การจองขนาดพื้นที่หน่วยความจำสำหรับญี่ปุ่น จะเริ่มจาก 1 ไปต่อไปเป็น 2 ไปต่อไปนั้นถ้าไม่พอ ก็จะเพิ่มครั้งละ 4 ไปเรื่อยๆ เป็น 8, 12, 16, 20, 24, ... และในแต่ละขณะ จะมีสมाचิกเพียงตัวเดียวเท่านั้นที่เข้าใช้พื้นที่นี้ได้ ดังนั้น ขนาดพื้นที่หน่วยความจำที่ต้องจองจึงเท่ากับขนาดของสมाचิกที่ใหญ่ที่สุด และเนื่องจากขนาดพื้นที่จะเพิ่มครั้งละ 4 ไปต่อไป หากขนาดสมाचิกที่ใหญ่ที่สุดไม่ใช่เลขที่ 4 หารลงตัว มันจะเลือกค่าถัดไปที่ 4 หารลงตัวมาให้แทน เช่น หากสมाचิกที่ใหญ่สุดมีขนาด 22 ไปต่อไปไม่ใช่เลขที่ 4 หารลงตัว มันจึงเลือกขนาดถัดไปที่มากกว่า 22 และ 4 หารลงตัวให้แทนนั่นคือ 24 หรือหากสมाचิกใหญ่สุดมีขนาด 39 ซึ่ง 4 หารไม่ลงตัว มันจะใช้ขนาดถัดไปที่ 4 หารลงตัวคือ 40

```

union demo {           //size
    char c;           //1
    char str[2];      //2 *
} dm;
printf("size: %d", sizeof(dm)); //2

union test {          //size
    double d;         //8 *
    float f;          //4
} ts;
printf("size: %d", sizeof(ts)); //8

union product {        //size
    char name[30];   //30 *
    float price;     //4
    int remaining;   //4
    char size;        //1
} pd;
printf("size: %d", sizeof(pd)); //32

union person {         //size
    char name[1];    //40
    char address[150]; //150 *
}

```

```

short age;           //2
long income;        //4
} ps;
printf("size: %d", sizeof(ps)); //152

```

ลิ้งที่ควรพึงระวังมากที่สุดในการใช้โครงสร้างแบบบัญเนี้ยนคือ ค่าที่อ่านได้ในแต่ละขณะ จะเป็นค่าของสมาชิกตัวล่าสุดที่เรากำหนดค่าให้กับมัน เนื่องจากการใช้พื้นที่หน่วยความจำร่วมกัน นั่นเอง สำหรับการใช้งานอื่นๆ ของบัญเนี้ยน ส่วนใหญ่ก็คล้ายกับสตรีกเจอร์ ไม่ว่าจะเป็น การใช้งาน ในแบบอาร์เรย์ การใช้ร่วมกับพอยน์เตอร์และฟังก์ชัน ดังนั้น จึงขอกล่าวถึงบัญเนี้ยนแค่เพียงเท่านี้

## ชุดข้อมูลแบบอีนนัม (Enum)

อีนนัม (มาจากคำว่า Enumeration) เป็นการสร้างชุดค่าคงที่ซึ่งเป็นคำในแบบสตริงเพื่อใช้แทนตัวเลข โดยมีรูปแบบพื้นฐานดังนี้

```

enum ชื่ออีนนัม {
    ชื่อสมาชิก_1 = เลขจำนวนเต็ม,
    ชื่อสมาชิก_2 = เลขจำนวนเต็ม,
    ชื่อสมาชิก_3 = เลขจำนวนเต็ม,
    ...
};

```

- enum เป็นคำส่วนในภาษาซี เพื่อปงชี้ว่าเป็นชุดข้อมูลแบบ enumeration
- ชื่ออีนนัม คือชื่อชุดข้อมูลดังกล่าว ซึ่งก็คล้ายกับชื่อสตรีกเจอร์และบัญเนี้ยน
- ชื่อสมาชิก=เลขจำนวนเต็ม เป็นการกำหนดสมาชิกของอีนนัมในแบบสตริง ซึ่งปงชี้ว่า สตริงหรือคำนั้นใช้แทนเลขจำนวนใด เช่น one=1 หมายถึง คำว่า one จะใช้แทนหรือ มีค่าเท่ากับ 1 โดยมีลักษณะที่ควรรู้เพิ่มเติมคือ
  - ต้องกำหนดสมาชิกไว้บล็อกหรือวงเล็บ {} เช่นเดียวกับสตรีกเจอร์และบัญเนี้ยน
  - ชื่อสมาชิก ให้เขียนคำนั้นในแบบสตริงลงโดย แต่ ไม่ ต้องมีเครื่องหมาย " " ครอบเหมือนสตริงทั่วไป
  - ชื่อสมาชิกมีหลักการเหมือนกับการตั้งชื่อตัวแปร เช่น ต้องเป็นตัวอักษร a - z หรือ A - Z หรือเครื่องหมาย \_ หรือเลข 0 - 9 แต่ห้ามขั้นต้นด้วยตัวเลข จะเป็นอักษรอะไรก็ได้จากนั้น เช่น ช่องว่าง ไม่ได้
  - ตัวเลขที่เป็นค่าของสมาชิก ต้องเป็นเลขจำนวนเต็มเท่านั้น (ติดลบได้)

- ◎ ถ้าไม่ระบุค่าตัวเลขให้กับสมาชิกของอีนนัม มันจะมีค่าเริ่มจาก 0 แล้วเพิ่มขึ้นทีละ 1 ไปตามลำดับ เช่น 0, 1, 2, 3, .... คล้ายกับลำดับของอาร์เรย์
- ◎ ค่าตัวเลขของสมาชิก จะเป็นค่าคงที่ไปตลอด ไม่สามารถแก้ไขในภายหลังได้

เมื่อนำไปใช้งาน ให้เราระบุเฉพาะชื่อสมาชิกที่ต้องการ โดยไม่ต้องระบุชื่ออีนนัม ซึ่งจะได้ค่าเป็นตัวเลขที่เรากำหนดให้สมาชิกอันนั้น เช่น ถ้าในอีนนัมกำหนด  $ha=5$  สามารถนำไปคำนวณโดยเขียนเป็น  $10 + ha$  จะได้ผลลัพธ์เป็น  $10 + 5 = 15$  ดังแนวทางต่อไปนี้

```
enum product {
    iPhone = 30000,
    iPad = 20000,
    MacBook = 80000
};

printf(
    "total price: %d",
    (iPhone + iPad + MacBook)
);

enum thaityextnum {
    ha=5, hok=6, jed=7, pad=8, kao=9, sib=10
};
//เนื่องจากค่าของสมาชิกจัดเรียงกันแบบเพิ่มทีละ 1
//ดังนั้น อาจเขียนตัวเลขเฉพาะค่าสมาชิกตัวแรกก็ได้ เช่น
//enum thaityextnum { ha=5, hok, jed, pad, kao, sib };
int a = hok + sib;      //a = 6 + 10 = 16

enum boolean { false=0, true=1 };
//เทียบเท่ากัน
//enum boolean { false, true };
if (true) {
    printf("You win!");
}
```

โดยทั่วไป เราจะพบเห็นการใช้อีนนัมได้ไม่บ่อยนัก จึงขอแนะนำให้รู้จักเพียงเท่านี้ก่อนอย่างไรก็ตาม เมื่อเราเขียนภาษา C ในระดับที่ซับซ้อนยิ่งขึ้น ก็อาจมีบางสถานการณ์ที่สามารถนำอีนนัมไปใช้งานร่วมด้วยก็เป็นได้

สิ่งที่เราได้เรียนรู้ในบทนี้ก็คือ โครงสร้างข้อมูลแบบ สตรัคเจอร์ ยูเนี่ยน และอีนนัม แต่โดยทั่วไปเราจะใช้สตรัคเจอร์เป็นหลัก ทั้งนี้โครงสร้างข้อมูลจะช่วยให้เราสามารถกำหนดและจัดองค์ประกอบแยกย่อยของข้อมูลอันเดียวกันได้ง่ายและสะดวกรวดเร็วมากขึ้น ซึ่งจะเป็นประโยชน์อย่างยิ่งสำหรับการนำไปประยุกต์ใช้งานในระดับที่สูงขึ้นไป

```
    = modifier_ob.modifiers.new("MIRROR")
    object_to_mirror_ob = mirror_ob
    mod.mirror_object = mirror_ob
    if len(mirror_ob.modifiers) == 0:
        mod.name = "MIRROR_X"
        mod.use_x = True
        mod.use_y = False
        mod.use_z = False
    elif len(mirror_ob.modifiers) == 1:
        mod.name = "MIRROR_Y"
        mod.use_x = False
        mod.use_y = True
        mod.use_z = False
    else:
        mod.name = "MIRROR_Z"
        mod.use_x = False
        mod.use_y = False
        mod.use_z = True

    # add the end -add back the deselected
    # objects
    if len(selected) == 1:
        selected[0].select = 1
    else:
        selected[0].select = 0
        context.scene.objects.active = modifier
        modifier.select = 1
        for obj in selected[1:]:
            obj.select = 1
            obj.select = 0
        context.selected_objects[0] = modifier
        selected[0].select = 1
        selected[1].select = 1
        print("please select exactly two objects, one object is not None")
```

operator\_classes = [Operator]  
"mirror to the selected object"""  
"select.mirror\_mirror\_x"  
"mirror X"





## การเขียนและอ่านไฟล์

นื้อหาในบทนี้จะกล่าวถึงวิธีการอ่านและเขียนข้อมูลลงในไฟล์ทั้งแบบ Text File ซึ่งเป็นไฟล์ข้อความธรรมดา และการนำโครงสร้างข้อมูล (Structure) ที่เราได้เรียนรู้ในบทที่แล้ว มาเขียนลงในไฟล์แบบไบนาเรีย (Binary File) ซึ่งการเขียน การเพิ่มข้อมูลใหม่ การอ่านทั้งไฟล์ หรือการเลือกอ่านเพียงบางส่วน จะช่วยให้เราสามารถจัดการกับข้อมูลได้อย่างเป็นระบบและมีประสิทธิภาพมากยิ่งขึ้น

### การเปิดและปิดไฟล์

ไฟล์ (File) เป็นชื่อของส่วนของการจัดเก็บข้อมูล ที่เราสามารถดำเนินการบางอย่างกับมันได้ เช่น การสร้างไฟล์ใหม่ การเขียน อ่าน หรือปรับปรุงแก้ไขข้อมูล เป็นต้น ทั้งนี้ ไม่ว่าเราจะทำสิ่งใดกับไฟล์ ก็มีขั้นตอนพื้นฐานหลักๆ ที่ต้องทำเหมือนกันคือ

1. เปิดไฟล์ (Open File)
2. ดำเนินการกับไฟล์ตามต้องการ (Perform Operations) เช่น อ่าน เขียน เป็นต้น
3. ปิดไฟล์ (Close File)

จากขั้นตอนที่กล่าวมา จะเห็นว่า สิ่งที่เราต้องดำเนินการเป็นอย่างแรกก็คือ การเปิดไฟล์ ตามรายละเอียดที่จะกล่าวถึงในหัวข้อนี้ แต่อย่างไรก็ตาม การเปิดไฟล์มีลักษณะคล้ายอยู่หลายอย่างที่เราต้องรู้เพิ่มเติมหรือดำเนินการควบคู่กันไปด้วย ซึ่งจะแยกอธิบายเป็นหัวข้อย่อยๆ ดังต่อไปนี้

### การระบุตำแหน่งไฟล์

การระบุตำแหน่งไฟล์เป้าหมายที่เราจะอ่านหรือเขียนข้อมูล หรือเรียกว่า พาธ (path) จะต้องเขียนในแบบสตริง แต่อย่างไรก็ตาม เนื่องจากเราใช้เครื่องหมาย \ ในการแบ่งหรือคั่นระหว่างไดเรกทอรี เช่น c:\data\test และภาษา C ก็ใช้ \ สำหรับการกำหนดอักษรพิเศษบางตัวเหมือนกัน

เช่น \t, \t จึงอาจส่งผลให้เกิดข้อผิดพลาดในบางกรณี ดังนั้น เพื่อหลีกเลี่ยงปัญหาดังกล่าว เราอาจกำหนดพาธโดยใช้เครื่องหมาย \ สองอันคือ \\ เพื่อคั่นระหว่างไดเรกทอรี เช่น

```
char *path1 = "c:\\c-lang\\chapter14\\sample.txt";
char *path2 = "c:\\users\\admin\\documents\\sample.txt";
```

ในการนี้ไฟล์นั้นจัดเก็บอยู่ในตำแหน่งหรือโฟลเดอร์เดียวกับไฟล์ของโค้ดภาษา C ที่เราใช้ติดต่อกับไฟล์ ก็อาจจะบุคคลซึ่งไฟล์โดยไม่จำเป็นต้องระบุไดเรกทอรีก็ได้ เช่น ถ้าโค้ดเก็บอยู่ที่

c:\c-lang\chapter14\example.c

และไฟล์เป้าหมายที่เราจะอ่านหรือเขียนข้อมูลอยู่ที่

c:\c-lang\chapter14\data.txt

การกำหนดพาธ ก็อาจจะบุคคลซึ่งไฟล์ ดังนี้

```
char *path = "data.txt";
```

อย่างไรก็ตาม ถ้าเราสร้างไดเรกทอรีหรือโฟลเดอร์ปั้ลกิย่อยช้อนลงไปจากตำแหน่งที่จัดเก็บไฟล์ของโค้ด จะต้องระบุตำแหน่งไดเรกทอรีอย่าง ๆ เหล่านั้นที่อยู่ถัดจากตำแหน่งที่จัดเก็บไฟล์ของโค้ดลงไปด้วย เช่น ถ้าโค้ดเก็บอยู่ที่

c:\c-lang\chapter14\example.c

และไฟล์เป้าหมายที่เราจะอ่านหรือเขียนข้อมูลอยู่ที่

c:\c-lang\chapter14\files\text\data.txt

การกำหนดพาธ ก็อาจจะบุคคลซึ่งไฟล์ที่อยู่ถัดจากไฟล์ของโค้ด ดังนี้

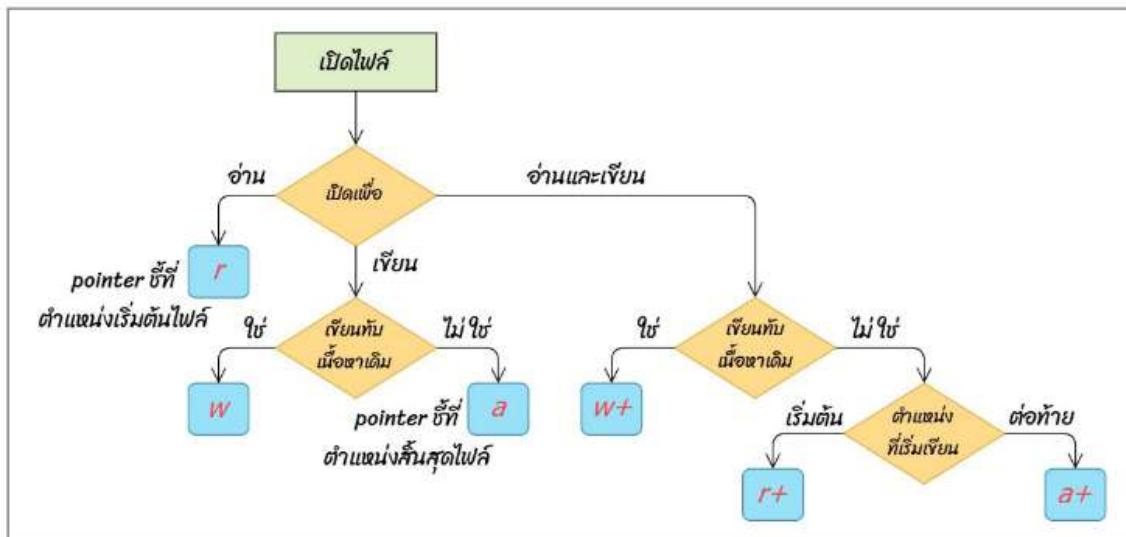
```
char *path = "files\\text\\data.txt";
```

## โหมดในการเปิดไฟล์

การเปิดไฟล์มีหลายวัตถุประสงค์ เช่น อ่านอย่างเดียว เขียนอย่างเดียว ทั้งอ่านและเขียน หรือ เพื่อเพิ่มข้อมูลต่อท้าย ดังนั้น ในการเปิดไฟล์ที่จะกล่าวถึงในหัวข้อถัดไป ก็มีลิสต์หนึ่งที่เราจะเป็นต้องกำหนดลงไปด้วย นั่นก็คือ โหมด (Mode) ในการเปิดไฟล์ หรือคล้ายกับการบ่งชี้ว่าเราจะเปิดไฟล์เพื่อทำลิสต์ใดนั้นเอง โดยโหมดการเปิดไฟล์ในเบื้องต้นมีให้เลือกดังนี้

r	สำหรับการอ่านอย่างเดียว และต้องมีไฟล์อยู่ก่อนแล้ว
w	เปิดไฟล์เพื่อการเขียนอย่างเดียว ถ้าไฟล์นั้นยังไม่มีจะถูกสร้างขึ้นใหม่ แต่ถ้ามีอยู่แล้ว เนื้อหาในไฟล์เดิมจะถูกเขียนทับ
a	เปิดเพื่อเพิ่มเนื้อหาต่อท้าย (append) ไฟล์อย่างเดียว ถ้าไฟล์นั้นยังไม่มีจะถูกสร้างขึ้นใหม่
r+	สำหรับการอ่านและเขียน แต่ต้องมีไฟล์อยู่ก่อนแล้ว
w+	เปิดไฟล์เพื่อการเขียนและอ่าน ถ้าไฟล์นั้นยังไม่มีจะถูกสร้างขึ้นใหม่ แต่ถ้ามีอยู่แล้ว เนื้อหาในไฟล์เดิมจะถูกเขียนทับ
a+	เปิดเพื่ออ่านและเพิ่มน้ำหนาต่อท้าย (append) ไฟล์ ถ้าไฟล์นั้นยังไม่มีจะถูกสร้างขึ้นใหม่ แต่เนื่องจากโหมดนี้ pointer จะอยู่ที่ท้ายไฟล์ ดังนั้น หากอ่านเนื้อหา ก็อาจได้เพียงค่าว่างๆ ยกเว้นเราจะเลื่อน pointer ย้อนกลับ (ตูรยละเอียดเพิ่มเติมในหัวข้อด้านไป)

โหมดที่เป็นการอ่านโดยตรง (r) ต้องมีไฟล์เป้าหมายอยู่ก่อนแล้ว มิฉะนั้นจะเกิดข้อผิดพลาด แต่โหมดที่เน้นการเขียนเป็นหลัก (w และ a) ถ้าไม่มีไฟล์เป้าหมายที่ระบุ จะถูกสร้างให้ใหม่ ทั้งนี้ หากเราไม่แน่ใจว่าจะเปิดไฟล์ในโหมดใด ก็ลองใช้หลักการพิจารณาดังแผนภาพด้านไปนี้



## การเปิดไฟล์

การเปิดไฟล์จะต้องใช้ฟังก์ชัน `fopen()` ซึ่งมีรูปแบบพื้นฐานดังนี้

```
fopen(file, mode)
```

- file ก็คือชื่อไฟล์เป้าหมายที่เราต้องการเปิด ทั้งนี้หากไฟล์ไม่ได้จัดเก็บในโฟลเดอร์เดียวกับไฟล์ของโคด เราจำเป็นต้องระบุตำแหน่งลงมาด้วย ดังหลักการที่ได้กล่าวมาแล้ว
- mode ก็คือวัตถุประสงค์ในการเปิดไฟล์ตามที่กล่าวไปในหัวข้อที่แล้ว

เมื่อเราเปิดไฟล์ด้วย `fopen()` เนื้อหาของไฟล์จะถูกนำไปจัดเก็บบนหน่วยความจำเพื่อรอจัดการตามวัตถุประสงค์ เช่น อ่านหรือเขียน เป็นต้น ซึ่งเราจำเป็นต้องใช้พอยน์เตอร์เพื่อเข้าถึงตำแหน่งที่จัดเก็บเนื้อหาของไฟล์ดังกล่าว ดังนั้น ลิ้งที่เราจำเป็นต้องทำควบคู่ไปกับการเรียกฟังก์ชัน `fopen()` ก็คือการสร้างตัวแปรพอยน์เตอร์ชนิด `FILE` (เขียนตัวพิมพ์ใหญ่ทั้งหมด) เพื่อเข้าถึงเนื้อหาของไฟล์ที่เปิดด้วย `fopen()` เช่น

```
char *path = "readme.txt";
FILE *fpt = fopen(path, "r");
//fpt ชี้ไปยังตำแหน่งที่จัดเก็บเนื้อหาของไฟล์

//หรือเขียนเป็น
//FILE *fpt = fopen("readme.txt", "r");
```

สำหรับ荷ดในการเปิดไฟล์ ถึงแม้จะเป็นแบบอักษรตัวเดียว ก็ต้องเขียนในแบบสตริง (ใช้เครื่องหมาย " ") แต่อย่างไรก็ตาม ถ้าไฟล์ที่เราอ้างถึงไม่มีอยู่จริง หรืออาจเขียนผิด พังก์ชัน `fopen()` จะคืนค่า `NULL` ซึ่งอาจเกิดข้อผิดพลาดในการใช้งานขึ้นก็ได้ โดยเฉพาะอย่างยิ่งเมื่อเปิดด้วย荷ด `r` หรือ `r++` ดังนั้น เพื่อป้องกันข้อผิดพลาดดังที่กล่าวมา เมื่อเปิดไฟล์ด้วย `fopen()` เราควรตรวจสอบด้วยว่ามันคืนค่าเป็น `NULL` หรือไม่ ถ้าใช่ก็แจ้งข้อผิดพลาด ก็อาจต้องหยุดทำงานแต่ถ้าไม่ใช่แล้วค่อยดำเนินการต่อไป เช่น

```
FILE *fpt = fopen("readme.txt", "r");
if (fpt == NULL) {
    puts("error! can't open file");
    exit(0);
}

/* หรืออาจเขียนให้กระชับขึ้นเป็น
if ((fpt = fopen("readme.txt", "r")) == NULL) {
    puts("error! can't open file");
    exit(0);
}
*/
```



### หมายเหตุ

การเปิดไฟล์ในโหมด w(+) หรือ a(+) เฉพาะกรณีที่ชื่อไฟล์ (รวมทั้งส่วนขยาย เช่น .txt) ที่เราระบุไม่มีอยู่ก่อนเท่านั้น ไฟล์จะจะถูกสร้างขึ้นใหม่ แต่หากได้รึกหรือหรือไฟล์เดอร์ที่ระบุไม่มีอยู่ก่อน ก็จะเกิดข้อผิดพลาดโดยไม่มีสิ่งใดถูกสร้างขึ้นใหม่ เช่น ถ้าระบุตำแหน่งเป็น

```
FILE *fpt = fopen("files\\data\\test.txt", "w");
```

สมมติว่าไฟล์เดอร์ files และ/หรือ data ไม่มีอยู่ก่อน ก็จะเกิดข้อผิดพลาด โดยไม่มีการสร้างไฟล์เดอร์และไฟล์ที่ระบุขึ้นมาใหม่

## การเปิดไฟล์

หากเปิดไฟล์สำเร็จ ต่อไปเมื่อเราจะจัดการกับเนื้อหาของไฟล์ ก็ต้องอ้างอิงผ่านตัวแปรพอยน์เตอร์ (ในโค้ดที่ผ่านมาคือ fpt) และหลังจากที่เปิดไฟล์และดำเนินการต่างๆ จนเสร็จล้วนแล้ว ก็ควรจะปิดไฟล์ดังกล่าวด้วยฟังก์ชัน fclose() ดังโค้ดต่อไปนี้

```
//open file
FILE *fpt;
if ((fpt = fopen("readme.txt", "r")) == NULL) {
    puts("error! can't open file");
    exit(0);
}

//perform operation
//...

//close file
fclose(fpt);
```

หลังจากที่ไฟล์ถูกปิดไปแล้ว หากเราจะเข้าถึงไฟล์ครั้งต่อไป จะต้องเปิดใหม่ด้วยคำสั่ง fopen() ตามหลักการเดิมทั้งหมด

## การเขียนไฟล์

เพื่อให้เราสามารถเข้าถึงและจัดการไฟล์ตามที่จะกล่าวถึงในหัวข้อต่อไปได้ เราจะเรียนรู้วิธีการสร้างและเขียนเนื้อหาลงในไฟล์เป็นลำดับแรก และเพื่อให้ไฟล์ที่ใช้ทดสอบการเขียนและอ่านเนื้อหาร่วมกับบทนี้ อยู่เป็นสัดส่วนไม่ประปนกับไฟล์ของโค้ด ก็จะสร้างไฟล์เดอร์ชื่อ files เพื่อจัดเก็บไฟล์เหล่านั้น โดยสร้างเพิ่มเข้ามาในไฟล์เดอร์ chapter14 ซึ่งอันนี้ เราต้องสร้างเองเอาไว้ล่วงหน้า สำหรับการเขียนไฟล์ มีแนวทางดังต่อไปนี้

> chapter13
▼ chapter14
> files
C example14-1.c
C example14-2.c

- ถ้าไฟล์นั้นยังไม่มีอยู่ก่อน เราอาจกำหนดโหมดในการเปิดไฟล์เป็น w(+) หรือ a(+) ก็ได้ผลเหมือนกัน
- ถ้าไฟล์นั้นยังมีอยู่ก่อนแล้ว เราต้องเลือกโหมด ตามลักษณะที่จะเขียนคือ (สามารถย้อนกลับไปดูได้จากแผนภาพในหัวข้อ โหมดการเปิดไฟล์ ดังที่กล่าวมาแล้ว)
  - ถ้าต้องการ เขียนทับ เนื้อหาเดิม (ถ้ามีเนื้อหาเดิมในไฟล์ จะหายไปทั้งหมด) ให้เปิดด้วยโหมด w หรือ w+
  - ถ้าต้องการ เขียนต่อท้าย เนื้อหาเดิม (เนื้อหาเดิมในไฟล์จะยังคงอยู่) ให้เปิดด้วย a หรือ a+
  - ถ้าต้องการ เขียนก่อน เนื้อหาเดิม (เนื้อหาเดิมในไฟล์จะยังคงอยู่) ให้เปิดด้วย r+
- ตำแหน่งในการเขียน เราอาจปรับเปลี่ยนโดยการย้ายตำแหน่งพอยน์เตอร์ ตามที่จะกล่าวถึงในลำดับต่อๆ ไป

การเขียนไฟล์ในเบื้องต้น เราอาจเลือกใช้ฟังก์ชันอันไดอันหนึ่งคือ

<b>fprintf()</b>	เขียนไฟล์โดยกำหนดชนิดข้อมูลที่จะเขียนได้
<b>fputs()</b>	เขียนข้อมูลชนิดสตริงลงในไฟล์
<b>fputc()</b>	เขียนไฟล์โดยกำหนดข้อมูลทีละ 1 อักขระ

แต่ละฟังก์ชันมีข้อกำหนดปลีกย่อยที่เราต้องรู้เพิ่มเติมอีกหลายอย่าง จึงขอแยกอธิบายเป็นหัวข้อย่อย ๆ ดังนี้

## ฟังก์ชัน fprintf()

มีรูปแบบพารามิเตอร์และรายละเอียดเพิ่มเติมดังนี้

```
fprintf(file_pointer, str_format, data_list)
```

- file\_pointer ก็คือตัวแปรพอยน์เตอร์ชนิด FILE ที่เราได้จาก fopen() นั่นเอง
- str\_format ก็คือสัญลักษณ์ที่ใช้แทนชนิดข้อมูลที่จะเขียนลงในไฟล์ เช่น %d สำหรับเลขจำนวนเต็ม, %f หรือ %g สำหรับเลขทศนิยม, %c สำหรับอักขระ 1 ตัว หรือ %s สำหรับสตริง เป็นต้น
- data\_list ก็คือข้อมูลที่เราจะแทรกลงใน str\_format เช่นเดียวกับ printf()

แนวทางการเขียนโค้ดในเบื้องต้น เช่น (ดูโค้ดที่สมบูรณ์ในตัวอย่างต่อๆ ไป)

```
FILE *fpt = fopen("...", "w");
...
char *str = "hello";
fprintf(fpt, "%s", str);
fprintf(fpt, "%s\n", " world");
fprintf(fpt, "%d\n", 12345);
fprintf(fpt, "pi = %g\n", 3.141);
fprintf(fpt, "%c", 'z');
...
fclose(fpt);
```

**ตัวอย่าง 14-1** แนวทางการเขียนข้อมูลหลายชนิดลงในไฟล์โดยใช้ฟังก์ชัน fprintf()

```
#include <stdio.h>
#include <stdlib.h>

void main() {
    FILE *fpt;
    char *path = "files\\ex14-1.txt";

    //ถึงแม้ว่าจะเปิดเพื่อเขียน ซึ่งหากไม่มีไฟล์อยู่ก่อน จะถูกสร้างใหม่
    //แต่บางกรณีอาจเกิดข้อผิดพลาด เช่น ชื่อไฟล์ซ้ำกับที่มีอยู่แล้ว
    //ดังนั้น ควรทำการตรวจสอบตามปกติก่อนใช้งาน
    if ((fpt = fopen(path, "w")) == NULL) {
        printf("\nerror! can't open file\n");
        exit(0);
    }

    char *names[] = {
        "T-Shirt", "Polo", "Trousers", "Slacks", "Jeans"
    };

    float princes[] = {
        300, 250, 450, 500, 1000
    };

    char sizes[] = {
        'M', 'S', 'S', 'L', 'M'
    };

    fprintf(fpt, "name,price,size");
    for (int i = 0; i < 5; i++) {
```

```

        fprintf(fpt,
            "\n%s,%g,%c",
            names[i], princes[i], sizes[i]
        );
    }

fclose(fpt);
}

```

หลังจากรันทดสอบโค้ดดังที่กล่าวมา หากเราเปิดไฟล์ที่เขียนข้อมูลลงไปจะปรากฏเนื้อหาในไฟล์ดังภาพ

```

C example14-1.c ex14-1.txt
chapter14 > files > ex14-1.txt
1 name,price,size
2 T-Shirt,300,M
3 Polo,250,S
4 Trouzers,450,S
5 Slacks,500,L
6 Jeans,1000,M

```

## ฟังก์ชัน fputs()

มีรูปแบบพารามิเตอร์และรายละเอียดเพิ่มเติมดังนี้

```
fputs(str, file_pointer);
```

str คือข้อมูลที่เราจะเขียนลงในไฟล์ ซึ่งต้องกำหนดเป็นชนิดสตริงเท่านั้น สำหรับแนวทางการเขียนโค้ดในเบื้องต้น เช่น

```

FILE *fpt = fopen("...", "w");
...
char *a = "apple";
fputs(a, fpt);
fputs("banana\n", fpt);
fputs("coconut\n", fpt);
fputs("durian\n", fpt);
fputs("12345\n", fpt);
fputs("z", fpt);

fclose(fpt);

```

การนำไปใช้งาน ให้ดูร่วมกับตัวอย่างของหัวข้อถัดไป

## ฟังก์ชัน fputc()

มีรูปแบบพารามิเตอร์และรายละเอียดเพิ่มเติมดังนี้

```
fputc(chr, file_pointer);
```

chr คืออักขระที่เราจะเขียนลงในไฟล์ ซึ่งต้องกำหนดเป็นชนิด char เท่านั้น หรือเป็นเลขรหัส ASCII ของอักขระที่ต้องการเขียน ส่วนแนวทางการเขียนโค้ดในเบื้องต้น เช่น

```
FILE *fpt = fopen("...", "w");
...
char a = 'a';
fputc(a, fpt);
fputc('b', fpt);
fputc(99, fpt); //เขียนอักขระที่มีรหัสเป็น 99 (ตัว c)
fputc('\n', fpt);

char *str = "defg";
for (int i = 0; i < strlen(str); i++) {
    fputc(str[i], fpt);
    //fputc('\n', fpt);
}

fclose(fpt);
```

**ตัวอย่าง 14-2** แนวทางการเขียนข้อมูลชนิดสตริงและอักขระลงในไฟล์ด้วยฟังก์ชัน fputs() และ fputc() โดยรับข้อมูลทางคีย์บอร์ด

```
#include <stdio.h>
#include <stdlib.h>

void main() {
    FILE *fpt;
    char *path = "files\\ex14-2.txt";

    if ((fpt = fopen(path, "w")) == NULL) {
        printf("\nerror! can't open file\n");
        exit(0);
    }

    char name[100], email[150];

    printf("\nEnter name email");
    printf("\ne.g. John John@test.net\n\n");

    for (int i = 1; i <= 5; i++) {
        printf("person # %d >>", i);
        scanf("%s%s", name, email);
    }
}
```

```

        if (i > 1) fputc('\n', fpt);
        fputs(name, fpt);
        fputc(':', fpt);
        fputs(email, fpt);

    }

fclose(fpt);
}

```

enter name email  
e.g. john john@test.net

person #1 >>elon elon@musk.com  
person #2 >>bill bill@gates.net  
person #3 >>tim tim@cook.org  
person #4 >>mark mark@zuck.info  
person #5 >>jeff jeff@bezos.biz

example14-2.c ex14-2.txt

chapter14 > files > ex14-2.txt

1	elon:elon@musk.com
2	bill:bill@gates.net
3	tim:tim@cook.org
4	mark:mark@zuck.info
5	jeff:jeff@bezos.biz

**ตัวอย่าง 14-3** เปรียบเทียบการเขียนด้วยโหมด a และ r+ โดยในตอนแรกจะเขียนข้อมูลลงในไฟล์ด้วยโหมด w ก่อน จากนั้นจำเป็นต้องปิดไฟล์ เพื่อเปิดใหม่ในโหมดอื่น

```

#include <stdio.h>
#include <stdlib.h>

void main() {
    FILE *fpt;
    char *path = "files\\ex14-3.txt";

    if ((fpt = fopen(path, "w")) == NULL) {
        printf("\nerror! can't open file\n");
        exit(0);
    }

    fputs("one\ntwo\nthree\nfour", fpt);
    fclose(fpt); //ต้องปิดไฟล์ก่อนจะเปิดในโหมดอื่น

    fpt = fopen(path, "a");
    fputs("\nfive", fpt);
    fclose(fpt);

    fpt = fopen(path, "r+");
    fputs("six", fpt);
    fclose(fpt);
}

```

example14-3.c ex14-3.txt

chapter14 > files > ex14-3.txt

1	six ← เขียนด้วยโหมด r+
2	two
3	three
4	four
5	five ← เขียนด้วยโหมด a

จากผลลัพธ์จะเห็นว่า โหมด a จะเขียนต่อท้ายเนื้อหาเดิม (พอยน์เตอร์ซึ่งทำหน่งลืนสุดไฟล์) ส่วนโหมด r+ จะเขียนที่ตำแหน่งเริ่มต้นไฟล์ และเขียนทับเนื้อหาเดิมบางส่วนที่อยู่ในช่วงแรกๆ

## การอ่านไฟล์

หากเราต้องการอ่านเนื้อหาของไฟล์เพื่อนำไปใช้งานอื่นๆ ต่อไป แม้จะมีทางเลือกในการเปิดไฟล์ได้หลายโหมด แต่ก็มีลักษณะปลีกย่อยที่แตกต่างกัน ซึ่งขอมากล่าวโดยสรุปอีกครั้ง คือ

- ถ้าเปิดด้วยโหมด r จะอ่านได้อย่างเดียว
- ถ้าเปิดด้วยโหมด r+ จะทำได้ทั้งอ่านและเขียน
- ถ้าเปิดด้วยโหมด w+ จะทำได้ทั้งอ่านและเขียน ถ้ามีการเขียน เนื้อหาเดิมภายในไฟล์ จะหายไปหมด เพราะถูกเขียนทับ (ถ้าจะเปิดโหมดนี้ ควรอ่านเนื้อหาเดิมก่อนแล้วค่อยเขียนเนื้อหาใหม่ทับลงไป)
- ถ้าเปิดด้วยโหมด a+ จะทำได้ทั้งอ่านและเขียน ถ้ามีการเขียน เนื้อหาใหม่จะถูกนำไปเขียนต่อท้ายเนื้อหาเดิม (เนื้อหาเดิมยังคงอยู่)

ในการอ่านเนื้อหาของไฟล์ พังก์ชันสำคัญที่เรา尼ยมใช้งานเป็นส่วนใหญ่คือ

<b>fgets()</b>	อ่านข้อมูลจากไฟล์จนบรรทัด หรือตามจำนวนอักขระที่ระบุ หรือจนลืนสุดไฟล์ ขึ้นกับว่าเงื่อนไขใดจะเกิดขึ้นก่อน
<b>fgetc()</b>	อ่านข้อมูลจากไฟล์ทีละ 1 อักขระ
<b>fscanf()</b>	อ่านข้อมูลตามรูปแบบที่กำหนด

แต่ละพังก์ชันมีข้อกำหนดปลีกย่อยที่เราต้องรู้เพิ่มเติมอีกหลายอย่าง จึงขอแยกอธิบายเป็นหัวข้อย่อยๆ ดังนี้

### พังก์ชัน fgets()

มีรูปแบบและข้อกำหนดที่สำคัญดังนี้

```
fgets(file_pointer, (จำนวนอักขระสูงสุด + 1), ตัวแปรที่เก็บผลลัพธ์)
```

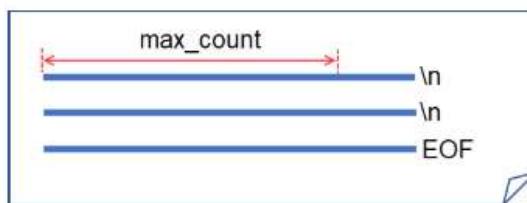
- ขอบเขตการอ่านข้อมูลของ fgets() จะขึ้นกับเงื่อนไขอย่างไรบ้างคือ

- จำนวนอักขระสูงสุด (max count)

(max count)

- จุดลิ้นสุดบรรทัด (\n)

- จุดลิ้นสุดไฟล์ (EOF: End Of File)



- เงื่อนไขดังที่กล่าวมาขึ้นกับว่าอันใดเกิดขึ้นก่อน เช่น หากเนื้อหาในไฟล์มีหลายบรรทัด และแต่ละบรรทัดมีจำนวน 50 อักขระ ถ้าเรากำหนดจำนวนอักขระสูงสุดเป็น 30 ก็จะอ่านได้ 30 อักขระแรกของบรรทัดที่ 1 (เงื่อนไขนี้เกิดขึ้นก่อน) แต่หากกำหนดจำนวนอักขระสูงสุดเป็น 80 ซึ่งเกินกว่าจำนวนอักขระใน 1 บรรทัด ก็จะอ่านได้แค่บรรทัดแรก (หรือเจอ \n ก่อนนั้นเอง) แต่ถ้าในไฟล์มีบรรทัดเดียว (ไม่มี \n ที่จุดลิ้นสุดไฟล์) และกำหนดจำนวนอักขระสูงสุดมากกว่าจำนวนอักขระทั้งหมด ก็จะอ่านได้เป็นเนื้อหาทั้งหมดที่มีในไฟล์ (เจอ EOF เป็นอันดับแรก)
- จำนวนอักขระสูงสุด เป็นตัวบ่งชี้ว่า เราจะอ่านข้อมูลสูงสุดกี่อักขระ และในการกำหนดค่าให้บวกเพิ่มอีก 1 (สำหรับ \0 ปิดท้ายสตริง) เช่น หากเราต้องการ 20 อักขระ ให้กำหนดอาร์กิวเมนต์เป็น 21 ทั้งนี้ หากเรากำหนดจำนวนอักขระเกินกว่าที่มีอยู่จริง ก็จะได้เท่าที่มี โดยไม่เกิดข้อผิดพลาด
- ตัวแปรที่เก็บผลลัพธ์ ต้องสร้างไว้ล่วงหน้า ให้เป็นชนิดสตริง (char array) ห้ามใช้ตัวแปรพอยน์เตอร์

แนวทางการอ่านข้อมูลด้วย fgets() ในเบื้องต้น เช่น

```

FILE *fpt = fopen("...", "r");
...
int max_chars = 50;      //ต้องการสูงสุด 50 อักขระ
char chr[max_chars + 1];

fgets(fpt, (max_chars + 1), chr);

int num_ch = 21;
//จะได้สูงสุดไม่เกิน 20 อักขระ
//แต่การเขียนแบบนี้ อาจทำให้เราเข้าใจผิดว่าจะได้ 21 อักขระ

char str[num_ch];
fgets(fpt, num_ch, str);

```

**ตัวอย่าง 14-4** จากไฟล์ ex14-2.txt ที่เราได้สร้างและเขียนเนื้อหาของไฟล์เอาไว้ในตัวอย่าง 14-2 เราจะทดลองอ่านเนื้อหาทั้งหมดที่ลงทะเบียนจากไฟล์ดังกล่าวด้วยฟังก์ชัน fgets()

```
#include <stdio.h>
#include <stdlib.h>

void main() {
    FILE *fpt;
    char *path = "files\\ex14-2.txt";

    if ((fpt = fopen(path, "r")) == NULL) {
        puts("\nerr! no such file or dir\n");
        exit(0);
    }

    //ถ้าต้องการอ่านที่ลงทะเบียน ให้กำหนดจำนวนอักขระสูงสุด
    //โดยคาดว่าให้เกินจำนวนอักขระที่มีในแต่ละบรรทัด
    //เพื่อให้เจอ \n ก่อนเงื่อนไขอื่น
    int max_len = 200;
    char lines[max_len + 1];

    putchar('\n');
    while (fgets(lines, max_len + 1, fpt) != NULL) {
        printf("%s", lines);
        //แต่ละบรรทัดที่อ่านได้จะรวม \n ต่อท้ายด้วย
        //ยกเว้นบรรทัดสุดท้ายอาจไม่มี \n (ขึ้นกับการเขียนไฟล์)

        //ถ้าจะตัด \n ท้ายบรรทัดออกให้กำหนดโค้ดดังนี้
        //remove trailing newline
        /*
        last_char = strlen(lines) - 1;
        if (lines[last_char] == '\n') {
            lines[last_char] = '\0';
        }
        */
    }
    putchar('\n');

    fclose(fpt);
}
```

elon:elon@musk.com
bill:bill@gates.net
tim:tim@cook.org
mark:mark@zucker.info
jeff:jeff@bezos.biz

## ฟังก์ชัน fgetc()

มีรูปแบบและข้อกำหนดที่น่าสนใจดังนี้

```
char c = fgetc(file_pointer)
```

- ตัวแปรที่เก็บผลลัพธ์ ต้องเป็นชนิดอักขระ (char) เพราะฟังก์ชันนี้จะคืนค่ากลับมาเป็นอักขระ 1 ตัว
- เมื่อเราเรียกฟังก์ชัน fgetc() แต่ละครั้ง มันจะคืนค่าเป็นอักขระ 1 ตัวในตำแหน่งที่พอยน์เตอร์ชี้อยู่ และเลื่อนพอยน์เตอร์ไปอีกหนึ่งอักขระ ดังนั้น เมื่อเราเรียก fgetc() ครั้งต่อๆ ไป ก็จะได้อักขระตัวถัดไปเรื่อยๆ
- เมื่อพอยน์เตอร์เลื่อนไปจนถึงจุดสิ้นสุดไฟล์ มันจะคืนค่าคงที่ EOF (ต้องเขียนด้วยตัวพิมพ์ใหญ่ทั้งหมด) ซึ่งมาจากคำว่า End Of File โดยค่าคงที่นี้ จัดเป็นชนิดอักขระ
- ถ้าเราต้องการอ่านเนื้อหาของไฟล์ต่อเนื่องกัน อาจใช้ลูปเพื่ออ่านอักขระทีละตัวมาเรียงต่อกันเป็นสตริง (อาร์เรย์) หรือแสดงผล ถ้าต้องการเนื้อหาทั้งหมดไฟล์ ก็อ่านไปจนกว่าจะเจอค่า EOF ซึ่งเป็นจุดสิ้นสุดไฟล์
- หรือวิธีที่ 2 คือใช้ฟังก์ชัน feof(file\_pointer) ตรวจสอบว่าในขณะนั้นพอยน์เตอร์ชี้อยู่ที่จุดสิ้นสุดไฟล์หรือไม่ ถ้าใช่จะคืนค่าจริงในแบบบูลีน (true หรือ 1)

แนวทางการอ่านข้อมูลด้วย fgetc() ในเบื้องต้น เช่น

```
FILE *fpt = fopen("...", "r");
//สมมติว่าเนื้อหาในไฟล์คือ ab\nodefghij
...
//ถ้าต้องการอ่านอักขระทีละตัว
char c = fgetc(fpt); //c = 'a'
c = fgetc(fpt); //c = 'b'
c = fgetc(fpt); //c = '\n'
...
c = fgetc(fpt); //c = 'j'
c = fgetc(fpt); //c = EOF

//ถ้าต้องการอ่านเนื้อหาทั้งหมด
//การใช้ลูป while จะหมายความที่สุด
//โดยเรียกฟังก์ชันไปจนกว่าได้ผลลัพธ์เป็น EOF
char ch;
while (1) {
    ch = fgetc(fpt);
```

```

if (ch == EOF) {
    break;
}
printf("%c", ch);
}

// สามารถเขียนแบบร่วมรัดได้เป็น
while ((ch = fgetc(fpt)) != EOF) {
    printf("%c", ch);
}

// หรืออีกวิธีคือ ตรวจสอบจุดลิ้นสุดไฟล์ด้วยฟังก์ชัน feof()
while (!feof(fpt)) { // ถ้ายังไม่ลิ้นสุดไฟล์ ก็อ่านไปเรื่อยๆ
    printf("%c", fgetc(fpt));
}

```

**ตัวอย่าง 14-5** การนับจำนวนบรรทัดและหาความยาว (จำนวนอักขระ) ของไฟล์ ทั้งนี้ การนับจำนวนบรรทัดก็คือ การนับจำนวนครั้งที่มันเจอ \n เพื่อขึ้นบรรทัดใหม่นั่นเอง ส่วนการนับจำนวนอักขระ ก็แค่นับเพิ่มทุกครั้งที่เรียก fgetc() แล้วได้อักขระที่ไม่ใช่ \n และ EOF (ซึ่งว่าง ก็นับเป็นอักขระ)

```

#include <stdio.h>
#include <stdlib.h>

void main() {
    FILE *fpt;
    char *path = "files\\ex14-1.txt";

    if ((fpt = fopen(path, "r")) == NULL) {
        puts("\nerror! no such file\n");
        exit(0);
    }

    char c;
    int num_lines = 0, num_chars = 0;

    while ((c = fgetc(fpt))) {
        if (c == '\n') {
            num_lines++;
        } else if (c == EOF) {
            num_lines++;
            break;
        } else {

```

```

        num_chars++;
    }
}

printf(
    "\nlines: %d\nchars: %d\n",
    num_lines, num_chars
);
}

```

lines: 6  
chars: 76

## การเลื่อนตำแหน่งของพอยน์เตอร์

บางครั้ง เราจำเป็นต้องอ่านข้อมูลหลายครั้ง โดยในแต่ละครั้งอาจอ่านข้อมูลไปจนสิ้นสุดไฟล์ หรือตำแหน่งใดๆ กายในไฟล์ ทั้งนี้หากเราอ่านไปจนถึง EOF จะอ่านต่อไม่ได้อีก หากต้องการอ่านอีกรอบ จะต้องเลื่อนพอยน์เตอร์กลับไปที่ตำแหน่งเริ่มต้นไฟล์ด้วยฟังก์ชันอันได้อันหนึ่งดังต่อไปนี้

- rewind(file\_pointer)
- fseek(file\_pointer, 0, SEEK\_SET)
- fseek(file\_pointer, 0, 0)

ฟังก์ชันและการกำหนดอาร์กิวเม้นต์ดังกล่าวเป็นการเลื่อนพอยน์เตอร์ไปยัง จุดเริ่มต้นไฟล์ ล้วนการเลื่อนไปยังตำแหน่งอื่นๆ จะกล่าวถึงในภายหลัง

```

FILE *fpt = fopen("...", "r");
...
while ((ch = fgetc(fpt)) != EOF) {
    printf("%c", ch);
}
printf("%c", fgetc(fpt)); //ไม่มีผล เพราะสิ้นสุดไฟล์แล้ว

rewind(fpt);           //กลับไปที่ตำแหน่งเริ่มต้น
printf("%c", fgetc(fpt)); //ได้อักขระตัวที่ 1
printf("%c", fgetc(fpt)); //ได้อักขระตัวที่ 2

fseek(fpt, 0, 0);      //กลับไปที่ตำแหน่งเริ่มต้น
printf("%c", fgetc(fpt)); //ได้อักขระตัวที่ 1

```

ถ้าต้องการเลื่อนไปยังจุดสิ้นสุดไฟล์อาจกำหนดแบบใดแบบหนึ่งคือ

- fseek(file\_pointer, 0, SEEK\_END)
- fseek(file\_pointer, 0, 2)

ยังมีฟังก์ชันเกี่ยวกับตำแหน่งที่นำสั่นใจอีกอย่างคือ `ftell(file_pointer)` ซึ่งใช้ตรวจสอบว่า ในขณะนั้นพอยน์เตอร์ซื้อยู่ที่ตำแหน่งหรือลำดับที่เท่าไร เช่น

```
FILE *fpt = fopen("...", "r");
...
int pos = ftell(fpt);
printf("%d\n", pos); //0

char c = fgetc(fpt);
printf("%d\n", ftell(fpt)); //1

c = fgetc(fpt);
printf("%d\n", ftell(fpt)); //2

//ถ้าพอยน์เตอร์เลื่อนไปถึงจุดสิ้นสุดไฟล์
//ftell() จะคืนค่าเป็นตำแหน่งสุดท้ายของไฟล์นั้น
```

## ฟังก์ชัน `fscanf()`

รูปแบบและข้อกำหนดที่เราจำเป็นต้องทราบเกี่ยวกับฟังก์ชันนี้คือ

<code>fscanf(file_pointer, รูปแบบ, ตัวแปรที่เก็บผลลัพธ์)</code>
---

- รูปแบบ ก็กำหนดด้วยสัญลักษณ์ตามชนิดข้อมูลที่เก็บในไฟล์ เช่นเดียวกับฟังก์ชัน `scanf()` เช่น `%d` สำหรับเลขจำนวนเต็ม หรือ `%s` สำหรับสตริง เป็นต้น
- ตัวแปรที่เก็บผลลัพธ์ ต้องสัมพันธ์กับสัญลักษณ์ที่กำหนดรูปแบบเช่นเดียวกับ `scanf()`
- ฟังก์ชัน `fscanf()` มากใช้กับไฟล์ที่มีแบบแผนการจัดเก็บข้อมูลที่แน่นอน และอาจดูยุ่งยากเล็กน้อย ซึ่งผู้เขียนขอสรุปหลักการที่น่าจะช่วยให้เราเข้าใจได้ง่ายขึ้น ดังนี้
  - ไฟล์ที่จะอ่านด้วย `fscanf()` ในแต่ละแถว ควรแบ่งข้อมูลออกเป็นคอลัมน์อย่างๆ และใช้รูปแบบการจัดเก็บเหมือนกันทุกแถว ดังแนวทางในภาพ
  - ในแต่ละแถว ต้องมีลิ่งที่ใช้แบ่งข้อมูลแต่ละคอลัมน์ออกจากกัน ซึ่งผู้เขียนแนะนำให้แบ่งด้วยแท็บ (`\t`) จะลำบากน้อยที่สุด ทั้งนี้หากเราใช้อักษรอะไรอย่างอื่น เช่น , อาจถูกมองว่าเป็นส่วนหนึ่งของข้อมูลในแต่ละคอลัมน์ ซึ่งต้องกำหนดรูปแบบการคัดแยกที่ซับซ้อนมากขึ้น แต่หากใช้ซองว่าง อาจมีปัญหากับบางคำที่มีซองว่างรวมอยู่ด้วย เช่น หากในคอลัมน์นั้นเป็นคำว่า `hello world` ก็อาจถูกตีความหมายว่า `hello` และ `world` อยู่คุณละคอลัมน์ เป็นต้น

abdef 1234 3.141 M ghijk 5555 -100.9 S xyz 108 1009 X ...
--

- สมมติว่าข้อมูลที่เราเก็บในไฟล์มีลักษณะดังในภาพที่แล้ว โดยแต่ละคอลัมน์แบ่งด้วย \t ดังนั้น แนวทางการกำหนดรูปแบบสัญลักษณ์ในการคัดแยกจะเป็นดังภาพ (ต้องมี \t ปิดท้ายแต่ละແຖา)
- เราต้องสร้างตัวแปรมาใช้ในการจัดเก็บข้อมูลในแต่ละคอลัมน์ เช่นเดียวกับรับข้อมูลทางคีย์บอร์ดด้วย scanf()

%s\t%d\t%f\t%c\n
abdef 1234 3.141 M
ghijk 5555 -100.9 S
xyz 108 1009 X

สำหรับแนวทางการอ่านข้อมูลจากไฟล์ด้วย fscanf() ให้ดูจากตัวอย่างต่อไปนี้

**ตัวอย่าง 14-6** จากไฟล์ที่เราใช้ประกอบในตัวอย่าง 14-1 ซึ่งใช้ , เป็นตัวแบ่งข้อมูลระหว่างคอลัมน์ เราจะปรับเปลี่ยนโดยการสร้างไฟล์ขึ้นมาใหม่ โดยใช้ \t แบ่งคอลัมน์ และลองอ่านข้อมูลจากแต่ละคอลัมน์แสดงผลด้วยฟังก์ชัน fscanf() ดังหลักการที่ได้กล่าวมาแล้ว

```
#include <stdio.h>
#include <stdlib.h>

void main() {
    FILE *fpt;
    char *path = "files\\ex14-6.txt";

    if ((fpt = fopen(path, "w+")) == NULL) {
        printf("\nerror! can't open file\n");
        exit(0);
    }

    char p[][3][30] = {
        {"T-Shirt", "300", "M"},
        {"Polo", "250", "S"},
        {"Trousers", "450", "S"},
        {"Slacks", "500", "L"},
        {"Jeans", "1000", "M"}
    };

    int n = sizeof(p) / sizeof(p[0]);

    for (int i = 0; i < n; i++) {
        fprintf(fpt, "%s\t%s\t%s\n",
            p[i][0], p[i][1], p[i][2]
        );
    }
}
```

```

// หลังการเขียนไฟล์ พอย่านเตอร์จะชี้ที่ท้ายไฟล์
// หากเราต้องการอ่านข้อมูลทั้งหมด
// ต้องเลื่อนพอยน์เตอร์กลับไปที่จุดเริ่มต้นของไฟล์
rewind(fpt);

char name[20], size;
int price;

while (!feof(fpt)) {
    fscanf(fpt, "%s\t%d\t%c\n", name, &price, &size);
    printf(
        "\nname: %s \t price: %d \t size: %c",
        name, price, size
    );
}

putchar('\n');
fclose(fpt);
}

```

name: T-Shirt	price: 300	size: M
name: Polo	price: 250	size: S
name: Trousers	price: 450	size: S
name: Slacks	price: 500	size: L
name: Jeans	price: 1000	size: M

**ตัวอย่าง 14-7** จากตัวอย่าง 14-2 ซึ่งเรารับข้อมูลทางคีย์บอร์ดแล้วนำไปเขียนลงในไฟล์ แต่กรณีดังกล่าวเรากำหนดขอบเขตหรือจำนวนครั้งในการรับข้อมูลໄว้แบบตายตัว จึงขาดความยืดหยุ่น ในตัวอย่างนี้ เราจะนำมารับปruzโดยให้รับข้อมูลทางคีย์บอร์ดได้เรื่อยๆ จนกว่าผู้ใช้จะกด **<ctrl + z>** จึงจะถือว่ายกเลิกการใส่ข้อมูล ซึ่งสามารถอธิบายเพิ่มเติมได้ดังนี้

- ในระบบ Windows จะถือว่า **<ctrl + z>** คือจุดสิ้นสุดการรับข้อมูล (ในระบบ MacOS/Linux ใช้ **<ctrl + d>**)
- เราสามารถใช้ฟังก์ชัน **feof()** เพื่อตรวจสอบจุดสิ้นสุดการรับข้อมูลทางคีย์บอร์ดได้ โดยกำหนดอาร์กิวเมนต์เป็นคำว่า **stdin** (มาจาก standard input) เช่น **feof(stdin)** โดยหากค่าที่รับเข้ามาเป็น **<ctrl + z>** ซึ่งเทียบเท่ากับ **eof** จะได้ค่าเป็นจริง

แม้หัวข้อนี้จะเป็นเรื่องการอ่านค่าจากไฟล์ แต่เพื่อให้ต่อเนื่องกับการใช้ฟังก์ชัน **feof()** ที่ได้เรียนรู้ในหัวข้อนี้ จึงนำเสนอด้วยตัวอย่างไว้ที่นี่

```

#include <stdio.h>
#include <stdlib.h>

void main() {
    FILE *fpt;
    char *path = "files\\ex14-6.txt";

```

```

if ((fpt = fopen(path, "w")) == NULL) {
    printf("\nerror! can't open file\n");
    exit(0);
}

char name[100], email[150];

printf("\nEnter name email");
printf("e.g. john john@test.info");
printf("\npress <ctrl+z> to stop\n\n");

//ใช้ลูป while เพื่อรับค่าเข้ามาเรื่อยๆ
//จนกว่าค่าที่รับเข้ามายจะเป็น <ctrl + z>
//ซึ่งเทียบเท่ากับ eof นั่นเอง
int n = 1;
while (1) {
    printf("person #%-d >>", n);
    scanf("%s%s", name, email);

    //ถ้าอินพุตเป็น <ctrl + z> ให้หยุดรับข้อมูล
    if (feof(stdin)) {
        break;
    }

    if (n > 1) fputc('\n', fpt);

    //เขียนข้อมูลที่รับเข้ามาลงในไฟล์
    fputs(name, fpt);
    fputc(':', fpt);
    fputs(email, fpt);

    n++;
}

fclose(fpt);
}

```

enter name email  
e.g. john john@test.info  
press <ctrl+z> to stop

person #1 >>jim jim@test.com  
person #2 >>jane jane@example.net  
person #3 >>joe joe@demo.org  
person #4 >>^Z ← กด <ctrl+z>  
c:\c-lang\chapter14| เพื่อยุดรับข้อมูล

C example14-7.c ex14-7.txt X  
chapter14 > files > ex14-7.txt

1	jim:jim@test.com
2	jane:jane@example.net
3	joe:joe@demo.org

## เก็บข้อมูลสตรักเจอร์ตัวย่อไฟล์ใบหน้า

ในหัวข้อที่ผ่านๆ มาแล้ว เราจัดข้อมูลในไฟล์เป็นแบบอักขระโดยตรง หรือเรียกว่า Text File หรือไฟล์ข้อความ และนอกจากรูปแบบนี้ยังมีไฟล์อีกรูปแบบหนึ่งซึ่งเรียกว่าไฟล์ใบหนารี (Binary File) เป็นการจัดเก็บข้อมูลในรูปแบบที่เราไม่สามารถอ่านเข้าใจได้ เพราะไม่ได้เก็บเป็นอักขระโดยตรง แม้เราจะสามารถใช้ไฟล์ใบหนารีได้หลายลักษณะ แต่สำหรับในที่นี้จะเน้นการจัดเก็บข้อมูลที่มีโครงสร้างที่แน่นอน (สตรักเจอร์) ลงในไฟล์ใบหนารี อย่างไรก็ตาม เนื่องจากวิธีดำเนินการกับไฟล์ชนิดนี้จะแตกต่างไปจากไฟล์ข้อความ ดังนั้น จึงมีสิ่งที่เราควรรู้จักในเบื้องต้นก่อนจะเข้าสู่ขั้นตอนการเขียนและอ่านไฟล์ในลำดับต่อไป ดังนี้

### โหมดในการเปิดไฟล์ใบหนารี

การเปิดและปิดไฟล์ใบหนารีก็ใช้ฟังก์ชัน fopen() และ fclose() เช่นเดียวกับไฟล์ข้อความ แต่สิ่งที่แตกต่างกันคือ การระบุโหมดสำหรับเปิดไฟล์ ซึ่งในการนี้ของไฟล์ใบหนารี มีโหมดให้เลือกดังนี้

<b>rb</b>	สำหรับการอ่านอย่างเดียว และต้องมีไฟล์อยู่ก่อนแล้ว
<b>rb+</b>	หรือ r+b สำหรับการอ่านและเขียน แต่ต้องมีไฟล์อยู่ก่อนแล้ว
<b>wb</b>	เปิดไฟล์เพื่อการเขียนอย่างเดียว ถ้าไฟล์นั้นยังไม่มีจะถูกสร้างขึ้นใหม่ แต่ถ้ามีอยู่แล้ว เนื้อหาในไฟล์เดิมจะถูกเขียนทับ
<b>wb+</b>	หรือ w+b เปิดไฟล์เพื่อการเขียนและอ่าน ถ้าไฟล์นั้นยังไม่มีจะถูกสร้างขึ้นใหม่ แต่ถ้ามีอยู่แล้ว เนื้อหาในไฟล์เดิมจะถูกเขียนทับ
<b>ab</b>	เปิดเพื่อเพิ่มเนื้อหาต่อท้าย (append) ไฟล์อย่างเดียว ถ้าไฟล์นั้นยังไม่มีจะถูกสร้างขึ้นใหม่
<b>ab+</b>	หรือ a+b เปิดเพื่ออ่านและเพิ่มเนื้อหาต่อท้าย (append) ไฟล์ ถ้าไฟล์นั้นยังไม่มีจะถูกสร้างขึ้นใหม่ แต่เนื่องจากโหมดนี้ pointer จะอยู่ที่ท้ายไฟล์ ดังนั้น หากอ่านเนื้อหาก็อาจจะได้เพียงคำว่างๆ ยกเว้นเราจะเลื่อน pointer ย้อนกลับ (ดูรายละเอียดเพิ่มเติมในหัวข้อถัดไป)

ขั้นตอนหลักๆ ก็เหมือนกับการเปิดไฟล์ข้อความ ต่างกันที่การระบุโหมดให้สอดคล้องกับไฟล์ชนิดนี้ และโดยทั่วไปแล้ว เรามักกำหนดส่วนขยายของไฟล์ใบหนารีเป็น .bin หรือไม่ก็ .dat (แต่ไม่ใช่ข้อบังคับ เราสามารถใช้ส่วนขยายอย่างอื่นก็ได้)

```

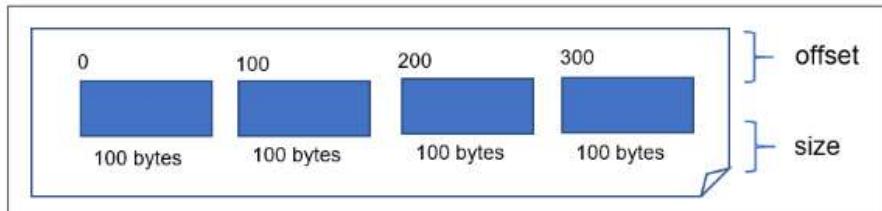
FILE *fpt;
char *path = "files\\example.dat";

if ((fpt = fopen(path, "rb+")) == NULL) {
    printf("\nerror! no such file or dir\n");
    exit(0);
}
...
fclose(fpt);           //ปิดไฟล์

```

### การจัดแบ่งข้อมูลแบบ Record

จากที่กล่าวไปแล้วว่า ในที่นี่เราจะเน้นการจัดเก็บข้อมูลสตรัคเจอร์ลงในไฟล์เป็นบinaire เป็นหลัก และจากบทที่ผ่านมา เราถูกใจเรียนรู้ไปแล้วว่า ข้อมูลแบบสตรัคเจอร์นั้น จะมีขนาดเท่ากับสมาชิกอยู่ๆ รวมกัน ดังนั้น ถ้าสมมติว่า เราสร้างสตรัคเจอร์หลายๆ ชุดจากโครงสร้างอันเดียวกัน แล้วนำมาเก็บลงในไฟล์ เราอาจจะเรียกข้อมูลในแต่ชุดว่า เรคคอร์ด (Record) หรือบล็อกข้อมูล (Block of Data) และเนื่องจากเราสร้างข้อมูลแต่ละเรคคอร์ด จากสตรัคเจอร์ที่มีโครงสร้างเดียวกัน ดังนั้น จึงมีขนาดเท่ากันไปตลอด ดังภาพ (สมมติว่าแต่ละเรคคอร์ดมีขนาดเป็น 100 ไบต์)



จากภาพ ถ้าเราต้องการข้อมูลเรคคอร์ดแรก เราต้องเลื่อนพอยน์เตอร์ไปยังตำแหน่งเริ่มต้น หรือถ้าต้องการข้อมูลเรคคอร์ดที่ 3 ก็ต้องเลื่อนพอยน์เตอร์ไปยังลำดับไบต์ที่ 200 เป็นต้น

### การเขียนข้อมูลสตรัคเจอร์ลงในไฟล์เป็นบinaire

พงก์ชั้นที่น่าจะเหมาะสมที่สุดสำหรับการเขียนข้อมูลสตรัคเจอร์ลงในไฟล์เป็นบinaire ก็คือ `fwrite()` ซึ่งมีรูปแบบดังนี้

```
fwrite(ตำแหน่งที่จะเขียนข้อมูล, ขนาดข้อมูล, จำนวนเรคคอร์ด, พอยน์เตอร์)
```

- ตำแหน่งที่จะเขียนข้อมูล ข้อมูลนั้นอาจเป็นสตรัคเจอร์หรืออาร์เรย์ก็ได้ แต่ในที่นี้จะเน้นเฉพาะสตรัคเจอร์ โดยระบุค่าเป็นตำแหน่งที่จัดเก็บข้อมูลนั้น (มี & ข้างหน้าตัวแปร)

- ขนาดข้อมูล เราสามารถตรวจสอบขนาดข้อมูลสตรัคเจอร์ที่เราจะเขียนโดยใช้ฟังก์ชัน `sizeof()` ซึ่งจะคืนค่ามาเป็นตัวเลขชนิด `size_t` (ปกติใช้วัดขนาดของเจ็กต์)
- จำนวนเรคอร์ด โดยทั่วไปเรามักจะเขียนเพียงอันเดียว ดังนั้น ส่วนมากจึงกำหนดค่าเป็น 1
- พอยน์เตอร์ ก็คือ file pointer ที่ได้จาก `fopen()` นั้นเอง

สำหรับแนวทางโดยทั่วไปคือ อันดับแรกเราต้องจัดเตรียมสตรัคเจอร์เอาไว้ก่อน จากนั้น ก็กำหนดค่าให้กับมัน แต่โดยส่วนใหญ่เรามักจะมีข้อมูลหลายชุด ดังนั้น เราอาจใช้ตัวแปรแบบ อาร์เรย์เพื่อจัดเก็บชุดสตรัคเจอร์ที่จะเขียน และขั้นตอนการเขียน หากเป็นชุดข้อมูลที่มีหลาย เรคอร์ด เราสามารถใช้ลูป `for` เพื่อเขียนด้วย `fwrite()` ทีละเรคอร์ดไปจนครบ ทั้งนี้ เมื่อเรา เขียนแต่ละเรคอร์ดเสร็จแล้ว พอยน์เตอร์จะขยับไปท้ายเรคอร์ดที่เพิ่งเขียนโดยอัตโนมัติ ดังนั้น จึงสามารถเขียนข้อมูลเรคอร์ดถัดไปได้ทันที โดยไม่จำเป็นต้องคำนวณตำแหน่งไปต่อหรือ `offset` เพื่อลั่งเลื่อนพอยน์เตอร์ ส่วนลักษณะปลีกย่อยอื่นๆ ให้ดูจากตัวอย่างต่อไปนี้

**ตัวอย่าง 14-8 การเขียนชุดข้อมูลสตรัคเจอร์ลงในไฟล์ในนารีดังหลักการที่กล่าวมาแล้ว**

```
#include <stdio.h>
#include <stdlib.h>

struct product {
    char name[30];
    float price;
    int remaining;
};

void main() {
    FILE *fpt;
    char *path = "files\\ex14-8.dat";

    if ((fpt = fopen(path, "wb")) == NULL) {
        printf("\nerror! can't open file\n");
        exit(0);
    }
    struct product pd[] = {
        {"jean", 1000, 10},
        {"shirt", 345, 5},
        {"jacket", 555.50, 8},
        {"slacks", 456, 7},
        {"trousers", 499, 0}
    }
    fwrite(pd, sizeof(struct product), 5, fpt);
}
```

```

};

int num_pd = sizeof(pd) / sizeof(pd[0]);

//ขนาดของสตรัคเจอร์ (เรคคอร์ด)
size_t size = sizeof(struct product);

for (int i = 0; i < num_pd; i++) {
    fwrite(&pd[i], size, 1, fpt);
}

fclose(fpt);
}

```

เมื่อรันไฟล์นี้ หากไม่มีข้อผิดพลาด ก็จะได้ไฟล์ ex14-8.dat เพิ่มเข้ามา แต่เนื่องจากเป็นไฟล์แบบไบนารี เราจึงไม่สามารถอ่านด้วยเครื่องมือของ VS Code เหมือนกับไฟล์ข้อความได้

### การอ่านข้อมูลสตรัคเจอร์จากไฟล์ไบนารี

การอ่านข้อมูลจากไฟล์ไบนารี ส่วนใหญ่เราเตรียมใช้ฟังก์ชัน  `fread()` ซึ่งมีรูปแบบดังนี้

```
fread(ตำแหน่งที่เก็บข้อมูล, ขนาด, จำนวน, พอยน์เตอร์)
```

- **ตำแหน่งที่เก็บข้อมูล** ก็คือตำแหน่งตัวแปร (ชนิด `struct`) ที่จะเก็บข้อมูลผลลัพธ์นั้นเอง โดยต้องเขียนเครื่องหมาย `&` กำกับที่หน้าตัวแปร
- **ขนาด** ก็คือขนาดเรคคอร์ดหรือจำนวนไบต์ที่เราจะอ่านข้อมูลนั้นเอง ซึ่งเราสามารถใช้ฟังก์ชัน  `sizeof()` เพื่อหาขนาดสตรัคเจอร์ (เรคคอร์ด) แล้วนำมากำหนดเป็นอาร์กิวเมนต์อันนี้ได้
- **จำนวน** ก็คือจำนวนเรคคอร์ดที่จะอ่าน โดยทั่วไปเรามักอ่านเพียงเรคคอร์ดเดียว ดังนั้น ส่วนมากจึงกำหนดค่าเป็น 1

ตามปกตินั้น เราต้องเลื่อนพอยน์เตอร์ไปยังจุดเริ่มต้นของเรคคอร์ดที่ต้องการอ่านข้อมูลแล้วค่อยอ่านด้วย  `fread()` แต่ถ้าเป็นการอ่านทุกเรคคอร์ด เมื่อเราอ่านเรคคอร์ดแรกเสร็จ พอยน์เตอร์จะเลื่อนไปที่ท้ายเรคคอร์ดนั้นโดยอัตโนมัติ (หรือจุดเริ่มต้นเรคคอร์ดถัดไป) ดังนั้น จึงสามารถอ่านเรคคอร์ดถัดจากนั้นได้ทันที และการอ่านเรคคอร์ดที่เหลือก็เป็นแบบเดียวกัน ด้วยเหตุนี้ เราจึงอ่านค่าด้วย  `fread()` แบบต่อเนื่องได้เลย โดยไม่จำเป็นต้องล็อกเลื่อนพอยน์เตอร์ให้ยุ่งยาก (ยกเว้นจะอ่านจากเรคคอร์ดเป้าหมายแบบเจาะจง)

ข้อมูลผลลัพธ์ที่อ่านได้จะถูกเก็บไว้ในตัวแปรที่เราระบุเป็นอาร์กิวเมนต์ และผลลัพธ์ที่คืนจากฟังก์ชัน `read()` จะเป็นขนาดของเรคอร์ดที่อ่านได้ (ชนิด `size_t`) ทั้งนี้ ถ้าเราจะอ่านค่าของทุกเรคอร์ด ก็สามารถใช้ลูป `while()` เพื่อเรียก `read()` ไปเรื่อยๆ จนกว่าจะถึง EOF เช่นเดียวกับไฟล์ข้อความ ดังแนวทางในตัวอย่างต่อไปนี้

**ตัวอย่าง 14-9** การอ่านข้อมูลสตอรักเจอร์จากไฟล์ในนารีด้วย `read()` ที่เราได้เขียนไว้ในตัวอย่างที่แล้ว โดยนำมาแสดงผลแบบง่ายๆ

```
#include <stdio.h>
#include <stdlib.h>

struct product {
    char name[30];
    float price;
    int remaining;
};

void main() {
    FILE *fpt;
    char *path = "files\\ex14-8.dat";

    if ((fpt = fopen(path, "rb")) == NULL) {
        printf("\nerror! no such file or dir\n");
        exit(0);
    }

    size_t size = sizeof(struct product);
    struct product pd;

    while (!feof(fpt)) {
        size_t result = fread(&pd, size, 1, fpt);
        if (result == 0) {
            break;
        }

        printf(
            "\nname: %s -- price: %g -- remaining: %d",
            pd.name, pd.price, pd.remaining
        );
    }
    putchar('\n');
    fclose(fpt);
}
```

name: jean -- price: 1000 -- remaining: 10  
 name: shirt -- price: 345 -- remaining: 5  
 name: jacket -- price: 555.5 -- remaining: 8  
 name: slakcs -- price: 456 -- remaining: 7  
 name: trousers -- price: 499 -- remaining: 0

**ตัวอย่าง 14-10** เพื่อเพิ่มความเข้าใจเรื่องการเขียนและอ่านข้อมูลสตรีกเจอร์ที่จัดเก็บในไฟล์ใบนำร่องให้มากยิ่งขึ้น ในตัวอย่างนี้ เราจะให้มีทั้งขั้นตอนการเขียนและอ่านข้อมูลแบบต่อเนื่องกันโดยจะแสดงผลลัพธ์ในแบบที่คล้ายกับตาราง ส่วนหลักการต่างๆ ก็เหมือนเดิมทั้งหมด

<i>name</i>	<i>age</i>	<i>position</i>	<i>salary</i>	
% - 20 s \t % 3 s \t % - 15 s \t % 10 s				
name	age	position	salary	
-----				
Harry Potter	40	Manager	80,000	
Tom Jerry	35	Programmer	45,000	
% - 20 s \t % 3 d \t % - 15 s \t % ' 10 d				

```
#define _POSIX_C_SOURCE 200809L
#include <locale.h>
#include <stdio.h>
#include <stdlib.h>

struct employee {
    char name[20];
    int age;
    char position[15];
    int salary;
};

void main() {
    setlocale(LC_ALL, "");
    FILE *fpt;
    char *path = "files\\ex14-10.dat";

    if ((fpt = fopen(path, "wb+")) == NULL) {
        printf("\nerror! can't open file\n");
        exit(0);
    }

    struct employee emp[] = {
        {"Harry Potter", 40, "Manager", 80000},
        {"Tom Jerry", 35, "Programmer", 45000},
        {"Tony Stark", 44, "Engineer", 55000},
        {"James Bond", 38, "Human Resource", 30000},
        {"Snow White", 30, "Accountant", 35000},
        {"Alice Wonderland", 25, "Officer", 18000}
    };

    int num_pd = sizeof(emp) / sizeof(emp[0]);
}
```

```

size_t size = sizeof(struct employee);

for (int i = 0; i < num_pd; i++) {
    fwrite(&emp[i], size, 1, fpt);
}

rewind(fpt);
struct employee em;

//แสดงผลในแบบตาราง
printf(
    "\n%-20s\t%3s\t%-15s\t%10s\n%s%s",
    "name", "age", "position", "salary",
    "-----",
    "-----"
);

while (!feof(fpt)) {
    size_t result = fread(&em, size, 1, fpt);
    if (result == 0) {
        break;
    }

    printf(
        "\n%-20s\t%3d\t%-15s\t%10d",
        em.name, em.age, em.position, em.salary
    );
}

putchar('\n');

fclose(fpt);
}

```

name	age	position	salary
Harry Potter	40	Manager	80,000
Tom Jerry	35	Programmer	45,000
Tony Stark	44	Engineer	55,000
James Bond	38	Human Resource	30,000
Snow White	30	Accountant	35,000
Alice Wonderland	25	Officer	18,000

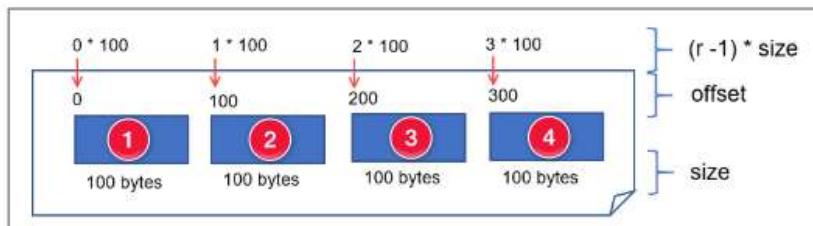
## การเข้าถึงข้อมูลแบบเจาะจง

ในหัวข้อที่ผ่านมานั้น เราอ่านหรือเข้าถึงข้อมูลในไฟล์แบบต่อเนื่องไปเรื่อยๆ ตั้งแต่ เรคคอร์ดแรกจนสิ้นสุดไฟล์ พอยน์เตอร์จะถูกเลื่อนไปยังจุดเริ่มต้นของเรคคอร์ดถัดไปโดยอัตโนมัติ ดังที่ได้กล่าวไปบ้างแล้วในหัวข้อที่ผ่านมา แต่หากเราต้องการอ่านข้อมูลจากเรคคอร์ดลำดับใด ลำดับหนึ่งแบบเจาะจง ต้องเลื่อนพอยน์เตอร์ไปยังจุดเริ่มต้นของเรคคอร์ดเป้าหมายด้วยฟังก์ชัน `fseek()` ซึ่งมีรูปแบบดังนี้

<code>fseek(พอยน์เตอร์, ขนาด, จุดเทียบตำแหน่ง)</code>
---

- ขนาด ก็คือตำแหน่งที่เราจะเลื่อนไป โดยเราคิดเป็นจำนวนไบต์ หรือจะเลื่อนพอยน์เตอร์ไปยังไบต์ที่เท่าใดนั้นเอง ซึ่งในกรณีนี้ เราเก็บข้อมูลสตรัคเจอร์ที่แต่ละเรคคอร์ด มีขนาดเท่ากันหมด ดังนั้น ตำแหน่งที่เราจะเลื่อนไปจึงคำนวนได้จาก

(เลขลำดับเรคคอร์ด - 1) \* (ขนาดของแต่ละเรคคอร์ด)



- จุดเดียบตำแหน่ง เป็นตัวกำหนดว่า การเลื่อนพอยน์เตอร์ไปตามขนาดที่ระบุนั้น จะเทียบกับตำแหน่งใด ซึ่งมีให้เลือก 3 ค่าคือ
  - SEEK\_SET (หรือ 0) เทียบกับตำแหน่งเริ่มต้นไฟล์
  - SEEK\_CUR (หรือ 1) เทียบกับตำแหน่งปัจจุบันที่พอยน์เตอร์ชี้อยู่
  - SEEK\_END (หรือ 2) เทียบกับตำแหน่งลิ้นสุดไฟล์แล้วเลื่อนย้อนกลับมา

เช่น ถ้ากำหนดค่าเป็น `fseek(fpt, 3 * rec_size, SEEK_SET)` หมายถึงเลื่อนพอยน์เตอร์ไปยังเรคคอร์ดที่ 4 นับจากจุดเริ่มต้นไฟล์ และหลังจากนั้น เราจึงสามารถอ่านค่าด้วย `read()` ได้เลย และขั้นตอนต่อจากนั้นก็เหมือนกับหัวข้อที่แล้วทั้งหมด โดยขอให้ดูเพิ่มเติมจากตัวอย่างต่อไปนี้

**ตัวอย่าง 14-11** จากไฟล์ใบนำร่องที่สร้างไว้ในตัวอย่างที่แล้ว ในตัวอย่างนี้เราจะรับค่าทางคีย์บอร์ด เป็นเลขลำดับเรคคอร์ดที่ต้องอ่าน จากนั้นนำไปกำหนดให้แก่ `fseek()` เพื่อเลื่อนพอยน์เตอร์ไปยังเรคคอร์ดดังกล่าว และอ่านข้อมูลของเรคคอร์ดนั้นมาแสดงผล เช่นเดียวกับตัวอย่างที่ผ่านมา

```
#define _POSIX_C_SOURCE 200809L
#include <locale.h>
#include <stdio.h>
#include <stdlib.h>

struct employee {
    char name[20];
    int age;
    char position[15];
    int salary;
};

int main() {
    // Your code here
}
```

```

void main() {
    setlocale(LC_ALL, "");
    FILE *fpt;
    char *path = "files\\ex14-10.dat";
    if ((fpt = fopen(path, "rb")) == NULL) {
        printf("\nerror! no such file or dir\n");
        exit(0);
    }

    size_t size = sizeof(struct employee);
    struct employee em;

    int r;
    printf("\nEnter record order to read >>");
    scanf("%d", &r);

    fseek(fpt, (r-1) * size, SEEK_SET);

    size_t result = fread(&em, size, 1, fpt);
    if (result > 0) {
        printf(
            "\n%-20s\t%3s\t%-15s\t%10s\n",
            "name", "age", "position", "salary",
            "-----",
            "-----"
        );
        printf(
            "\n%-20s\t%3d\t%-15s\t%10d\n",
            em.name, em.age, em.position, em.salary
        );
    }
}

fclose(fpt);
}

```

enter record order to read >>4			
name	age	position	salary
James Bond	38	Human Resource	30,000
<hr/>			
enter record order to read >>1			
name	age	position	salary
Harry Potter	40	Manager	80,000

เนื้อหาที่เราได้เรียนรู้กันไปในบทนี้ ทั้งการอ่านและเขียนข้อมูลลงในไฟล์แบบ Text File และการนำโครงสร้างข้อมูล (Structure) ที่เราได้เรียนรู้ในบทที่แล้ว มาเขียนลงในไฟล์แบบไบนาเรีย (Binary File) ซึ่งการเขียน การเพิ่มข้อมูลใหม่ การอ่านทั้งไฟล์ หรือการเลือกอ่านเพียงบางส่วน จะช่วยให้เราสามารถจัดการกับข้อมูลได้อย่างเป็นระบบและเบี่ยบมากยิ่งขึ้น



# 15

## พรีโพรเซสเซอร์และมาโคร

จํา

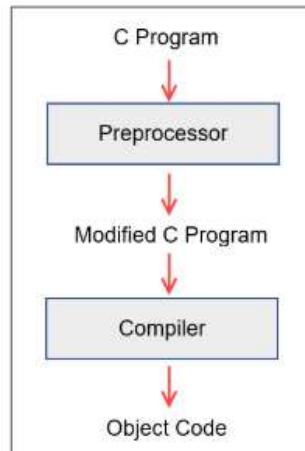
กษณะพื้นฐานของภาษา C ที่เราควรรู้จักเพิ่มเติมก็คือ พรีโพรเซสเซอร์ (Preprocessor) และมาโคร (Macro) ซึ่งใช้กำหนดส่วนที่จะถูกดำเนินการล่วงหน้า เพื่อให้ส่วนนี้มีผลไปถึงได้ที่อยู่ในลำดับถัดไป โดยพรีโพรเซสเซอร์ บางส่วนอาจถูกกำหนดมาให้แล้วพร้อมกับภาษา C แต่นอกจากนี้ เราสามารถสร้างขึ้นมาใช้งานเองได้ ดังรายละเอียดที่จะได้เรียนรู้กันในบทนี้

### เกี่ยวกับพรีโพรเซสเซอร์และไดเรกทีฟ

พรีโพรเซสเซอร์ (Preprocessor) เป็นชุดคำสั่งบางส่วนที่ต้องถูกดำเนินการล่วงหน้าก่อนที่จะนำไปใช้งานที่ส่วนอื่น ๆ โดยจะกำหนดในแบบคำสั่งพิเศษที่เรียกว่า Preprocessing Directive หรือเรียกสั้น ๆ ว่า ไดเรกทีฟ (Directive) ดังหลักการในภาพ

โดยทั่วไปแล้ว เราจะแบ่งไดเรกทีฟออกเป็น 4 กลุ่มหลัก ๆ คือ

- **File Inclusion** เป็นไดเรกทีฟสำหรับการอ้างอิงไฟล์ เอเดเครอร์ ซึ่งก็คือ #include ที่เราใช้กันมาตลอดนั่นเอง
- **Macro Definition** เป็นไดเรกทีฟสำหรับกำหนดค่าคงที่รวมถึงการเขียนคำสั่งย่อที่เราเรียกว่า มาโคร (Macro) ซึ่งไดเรกทีฟกลุ่มนี้ประกอบด้วย #define และ #undef
- **Conditional Compilation** เป็นไดเรกทีฟสำหรับกำหนดเงื่อนไขการทำงาน ซึ่งประกอบด้วย #if, #else, #elif, #endif, #ifdef และ #ifndef
- **Other Directives** เป็นไดเรกทีฟส่วนที่เหลืออีกบางส่วน ได้แก่ #error, #line และ #pragma



โดยทั่วไปแล้ว เราสามารถใช้งานได้เร็วทีฟในกลุ่ม File Inclusion และ Macro Definition เป็นส่วนใหญ่ ส่วนกลุ่มอื่นๆ อาจพบเห็นได้ไม่นบยนัก ซึ่งรายละเอียดการใช้ได้เร็วทีฟแต่ละแบบ จะกล่าวถึงในหัวข้อต่างๆ ต่อจากนี้



### หมายเหตุ

เนื่องจากได้เร็วทีฟในกลุ่ม File Inclusion ชื่อก็คือ #include เราได้ผ่านการใช้งานกันมาตลอดตั้งแต่ที่แรกและต่อเนื่องมาในทุกบทของหนังสือเล่มนี้ โดยวิธีการหลักๆ ก็เพียงแค่ระบุชื่อไฟล์เดอร์หรือเมื่อนเขียนเคย จึงไม่มีอะไรที่จำเป็นต้องรู้เพิ่มเติมอีก ดังนั้น จึงไม่ขอนำได้เร็วทีฟ #include มากล่าวถึงช้าอีก

## กลุ่มได้เร็วทีฟ Macro Definition

มาโคร (Macro) เป็นสัญลักษณ์สำหรับการทำหน้าค่าคงที่หรือกลุ่มคำสั่งเพื่อดำเนินการบางอย่าง ซึ่งเมื่อเราอ้างถึงสัญลักษณ์ดังกล่าว ค่าคงที่หรือคำสั่งที่สร้างไว้ร่วมกันก็จะถูกดำเนินการทันที ซึ่งวิธีการสร้างมาโครนั้น ให้ใช้ได้เร็วทีฟ #define ในรูปแบบดังนี้

```
#define ชื่อมาโคร ค่าคงที่หรือคำสั่ง
```

- #define เป็นได้เร็วทีฟที่บ่งชี้ว่าเป็นการสร้างมาโคร
- ชื่อมาโคร ก็ใช้หลักการเดียวกับการตั้งชื่อตัวแปร ซึ่งโดยทั่วไปเรานิยมเขียนชื่อมาโครเป็นตัวพิมพ์ใหญ่ทุกตัวอักษร เพื่อให้แตกต่างจากตัวแปรทั่วไป (แต่ไม่ใช่ข้อนั้น เราจะเขียนด้วยตัวพิมพ์อย่างไรก็ได้)
- ค่าคงที่หรือคำสั่ง เราอาจกำหนดค่าที่แน่นอนเป็นตัวเลข สดิง หรืออักษร (ค่านี้จะแก้ไขในภายหลังด้วยวิธีการแบบปกติไม่ได้) หรืออาจกำหนดเป็นคำสั่งเพื่อดำเนินการบางอย่าง เช่น การคำนวน การแสดงผล หรืออื่นๆ
- ไม่ ต้องใช้เครื่องหมาย = การกำหนดค่าให้กับมาโคร แต่ให้เว้นช่องว่างหลังชื่อมาโครแล้วระบุค่าหรือคำสั่งต่อจากชื่อของมันได้เลย
- มาโครแต่ละอันต้องเขียนให้จบในบรรทัดเดียว และเมื่อสิ้นสุดมาโคร ไม่ ต้องเขียนเครื่องหมาย ; ปิดท้ายเมื่อคำสั่งทั่วไป

ค่าของมาโครที่เรากำหนด ไม่สามารถแก้ไขในภายหลังด้วยการเปลี่ยนค่าแบบปกติได้ยกเว้นเราจะยกเลิกมาโครอันเดิมด้วยได้เร็วทีฟ #undef และสร้างมาโครชื่อเดิมด้วย #define พร้อมกำหนดค่าใหม่ตามต้องการ เช่น

```
#define RATE 0.5

void main() {
    printf("rate = %g\n", RATE);      //0.5

    RATE = 0.75;                    //Error แก้ไขค่าของมาโครไม่ได้

    #undef RATE                      //ยกเลิกมาโครเดิม
    #define RATE 0.75                //กำหนดมาโครขึ้นมาใหม่

    printf("rate = %g\n", RATE);      //0.75
}
```

สำหรับการกำหนดมาโครแบบค่าคงที่ ซึ่งเราเคยศึกษามานั้งแล้วในบทที่ 3 แต่ขอนำมา  
กล่าวทบทวนอีกครั้ง และขอให้ดูเพิ่มเติมจากตัวอย่างต่อไปนี้

---

**ตัวอย่าง 15-1** แนวทางการกำหนดมาโครแบบค่าคงที่และการนำค่าไปใช้งาน

```
#include <stdio.h>

#define PI 3.14          //มาโครชื่อ PI มีค่าเป็น 3.14
#define VAT 7
#define TRUE 1
#define FALSE 0
#define ERR_NOFILE "error, file not found"

void main() {
    #undef PI
    #define PI 3.14159

    printf("pi = %g\n", PI);

    int q = 5, p = 100;
    float total = (p * q) * (1 + VAT/100.0);
    printf("total inc. vat = %g\n", total);

    if (TRUE) {
        puts("Yes! TRUE = 1\n");
    } else {
        puts("No! TRUE != 1\n");
    }

    FILE *fpt;
    fpt = fopen("xxx.txt","r");
```

```

if (fpt == NULL) {
    puts(ERR_NOFILE);
}

```

```

pi = 3.14159
total inc. vat = 535
Yes! TRUE = 1
error, file not found

```

นอกจากการทำหน้าโครงเป็นค่าคงที่แล้ว เราอาจกำหนดเป็นคำสั่งเพื่อดำเนินการบางอย่างก็ได้ โดยมีลิสต์ที่เราควรรู้ในเบื้องต้นคือ

- ให้เขียนคำสั่งเหล่านั้นต่อจากชื่อมาโครได้เลย
- ถ้ามีหลายคำสั่งย่อย ให้คั่นแต่ละคำสั่งด้วยเครื่องหมาย ;
- ถ้ามีหลายคำสั่งย่อย คำสั่งแรกห้ามคืนค่าเป็น void
- ต้องเขียนทุกคำสั่งไว้บรรทัดเดียวกันทั้งหมด แต่ในกรณีที่ไม่สามารถทำเช่นนั้นได้ และหากต้องการแยกบางส่วนไปขึ้นบรรทัดใหม่ ให้วางเครื่องหมาย \ ปิดท้ายบรรทัดที่จุดซึ่งต้องการนำໂโคดไปขึ้นบรรทัดใหม่

### ตัวอย่าง 15-2 การกำหนดมาโครแบบคำสั่งย่อยๆ คล้ายกับฟังก์ชัน

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define HELLO printf("Hello")

#define COUNT printf("one\t"); \
            printf("two\t"); \
            printf("three")

#define NOFILE printf("file not found\n"); \
            exit(0)

#define SEED srand(time(0))
#define RAND_1_100      1 + rand() % 100

void main() {
    putchar('\n');
    HELLO;

    putchar('\n');
    COUNT;
}

```

```

SEED;
int r = RAND_1_100;
printf("\nrandom num: %d\n", r);

FILE *fpt;
fpt = fopen("xxx.txt","r");
if (fpt == NULL) {
    NOFILE;
}
}

```

```

Hello
one      two      three
rand num: 42
file not found

```

คำสั่งของมาโครสามารถรับพารามิเตอร์จากส่วนที่เรียกใช้มาโครได้ เช่นเดียวกับฟังก์ชัน แล้วในส่วนคำสั่งของมาโครก็นำพารามิเตอร์ดังกล่าวไปใช้งานได้ในแบบเดียวกัน โดยหลักการ ในเบื้องต้นคือ

- ถ้าต้องการรับพารามิเตอร์ ให้กำหนดวงเล็บ ( ) ต่อท้ายชื่อมาโคร
- กำหนดพารามิเตอร์ในวงเล็บ ( ) ถ้ามีหลายตัวให้คั่นด้วยเครื่องหมาย ,
- ระบุเฉพาะพารามิเตอร์เท่านั้น แต่ไม่ต้องกำหนดชนิดข้อมูลของพารามิเตอร์

**ตัวอย่าง 15-3** การกำหนดมาโครแบบคำสั่งย่อโดยรับพารามิเตอร์จากส่วนที่เรียกใช้มาโครคล้ายกับฟังก์ชัน

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define SQUARE(n) n * n
#define ODD_EVEN(n) (n % 2 == 0) ? "even" : "odd"
#define SEED srand(time(0))
#define RAND(min, max) min + rand() % (max - min + 1)
#define SWAP(a, b) a = a + b; b = a - b; a = a - b;
/*
a = 3, b = 5
a = a + b = 3 + 5 = 8
b = a - b = 8 - 5 = 3
a = a - b = 8 - 3 = 5
*/
void main() {
    printf("\n9^2 = %d", SQUARE(9));
    printf("\n99 is %s", ODD_EVEN(99));
}

```

```

9^2 = 81
99 is odd
rand(20, 30): 27
before swapping: a = 108, b = -1009
after swapping: a = -1009, b = 108

```

```

SEED;
printf("\nrand(20, 30): %d", RAND(20, 30));

int a = 108, b = -1009;
printf("\nbefore swapping: a = %d, b = %d", a, b);
SWAP(a, b);
printf("\nafter swapping: a = %d, b = %d\n", a, b);
}

```

## ค่าคงที่ในกลุ่ม Predefined Macro

นอกจากมาโครที่เรากำหนดค่าหรือวิธีดำเนินการขึ้นมาเองแล้ว ยังมีมาโครอีกส่วนหนึ่งซึ่งเป็นค่าที่มีอยู่แล้วในภาษา C หรือเป็นมาโครที่ถูกสร้างมาให้ล่วงหน้าแล้วนั่นเอง โดยส่วนใหญ่จะเป็นข้อมูลที่เรามีโอกาสได้นำไปใช้งานอยู่บ่อยๆ เช่นอันที่นำสนใจดังนี้

<u>FILE</u>	คือชื่อไฟล์ของโค้ดที่กำลังถูกประมวลผลในขณะนั้น
<u>LINE</u>	หมายเลขบรรทัดที่กำลังประมวลผล
<u>DATE</u>	ค่าวันเดือนปีปัจุบันในรูปแบบ ชื่อเดือน (3 ตัวแรก) วันที่ ปี.ค. เช่น Jan 10 2023
<u>TIME</u>	เวลาขณะนั้นในรูปแบบ ชั่วโมง:นาที:วินาที

เราสามารถนำค่าเหล่านี้ไปใช้งานได้ทันที เช่นเฉพาะ LINE เท่านั้นจะได้ค่าเป็นตัวเลข ส่วนตัวอื่นๆ จะได้ค่าเป็นสตริงทั้งหมด

**ตัวอย่าง 15-4** แนวทางการใช้มาโครในกลุ่ม Predefine ซึ่งถูกกำหนดมาให้ล่วงหน้าแล้วในภาษา C

```

#include <stdio.h>

void main() {
    printf("\nthis file is: %s", __FILE__);
    printf("\nthis line is %d", __LINE__);
    printf("\ntoday is %s", __DATE__);
    printf("\ncurrent time is %s\n", __TIME__);
}

```

```

this file is: example15-4.c
this line is 5
today is Sep 5 2022
current time is 15:32:35

```

**ตัวอย่าง 15-5** เป็นการแสดงให้ดูของไฟล์ที่เราทำการประมวลผลขณะนั้น โดยเปิดไฟล์เป้าหมายจากมาโคร `_FILE_` และอ่านเนื้อหาไปแสดงผลดังที่เราเคยทำกันมาในบทที่แล้ว

```
#include <stdio.h>

void main() {
    FILE *fpt;
    fpt = fopen(_FILE_, "r");
    char c;

    putchar('\n');
    while ((c = fgetc(fpt)) != EOF) {
        printf("%c", c);
    }
    putchar('\n');

    fclose(fpt);
}
```

```
#include <stdio.h>

void main() {
    FILE *fpt;
    fpt = fopen(_FILE_, "r");
    char c;

    putchar('\n');
    while ((c = fgetc(fpt)) != EOF) {
        printf("%c", c);
    }
    putchar('\n');

    fclose(fpt);
}
```

## กลุ่มไดเร็กทีฟ Conditional Compilation

ไดเร็กทีฟในกลุ่ม Conditional ใช้ในการกำหนดเงื่อนไข ซึ่งหากเป็นการใช้งานระดับพื้นฐานทั่วไป เราอาจพบเห็นไดเร็กทีฟกลุ่มไดไม่น้อยนัก แต่ถึงอย่างไรก็จะแนะนำให้รู้จักแนวทาง เอาไว้บ้าง เพื่อประโยชน์ต่อการใช้งานในระดับที่สูงขึ้นต่อไป โดยไดเร็กทีฟในกลุ่มนี้ประกอบด้วย

<code>#if</code>	ตรวจสอบเงื่อนไขเช่นเดียวกับคำสั่ง <code>if</code>
<code>#elif</code>	ตรวจสอบเงื่อนไขอื่นๆ เพิ่มเติมเช่นเดียวกับ <code>else if</code>
<code>#else</code>	กำหนดทางเลือกในการทำงานถ้าไม่ตรงกับเงื่อนไขที่ตรวจสอบด้วย <code>#if</code> หรือ <code>#elif</code>

#ifdef	ตรวจสอบว่าได้กำหนด (define) ตัวแปร หรือมาโครที่ระบุเป็นเงื่อนไขหรือไม่ และกำหนดวิธีการทำงานถ้าได้กำหนดเอาไว้
#ifndef	ตรวจสอบว่าไม่ได้กำหนด (not define) ตัวแปร หรือมาโครที่ระบุเป็นเงื่อนไขหรือไม่ และกำหนดวิธีการทำงานถ้าไม่ได้กำหนดเอาไว้
#endif	ลิ้นสุดขอบเขตคำลั่งของ #if

ได้เร็กทิฟกลุ่มนี้ที่แสดงในตารางที่กล่าวมา จะต้องระบุเงื่อนไขทั้งหมด ยกเว้นเพียง #else เท่านั้น แนวทางการใช้งานโดยสังเขป เช่น

**ตัวอย่าง 15-6** แนวทางการใช้ไดร์กทิฟในกลุ่ม Conditional Compilation

```
#include <stdio.h>

#define DEFAULT_VALUE 1

void main() {
    #ifdef DEFAULT_VALUE
        printf("\ndefault value defined!\n");
    #else
        #define DEFAULT_VALUE 1
    #endif

    printf("\ndefault value is %d\n", DEFAULT_VALUE);
}
```

ลิ้งที่เราได้เรียนรู้กันไปในบทนี้ เป็นลักษณะพื้นฐานของภาษา C ที่เราควรรู้จักเพิ่มเติม ทั้ง การใช้ไดร์กทิฟแบบต่างๆ รวมถึงการเขียนมาโคร ซึ่งใช้กำหนดส่วนที่จะถูกดำเนินการล่วงหน้า เพื่อให้ส่วนนี้มีผลไปถึงโค้ดที่อยู่ในลำดับถัดไป ซึ่งในทางปฏิบัติ เราอาจนำไปใช้งานได้ในบางกรณี ดังนั้น จึงจำเป็นต้องเรียนรู้เอาไว้บ้าง เมื่อดูส่วนใหญ่เราจะใช้พรีprocessor ที่ถูกกำหนดมาให้ ล่วงหน้าแล้วพร้อมกับภาษา C ก็ตาม แต่อย่างน้อยเราก็รู้หลักการและที่มาของมัน หรือบางครั้ง เรายังสามารถสร้างชื่นมาใช้งานเองได้



# 16

## รวมตัวอย่างโค้ดเพิ่มเติม ชุดที่ 3

แบบนี้ เป็นการรวบรวมตัวอย่างโค้ดชุดที่ 3 โดยใช้พื้นฐานความรู้ระหว่างบทที่ 13 - 15 เป็นหลัก แต่ถึงอย่างไรก็ต้องนำความรู้พื้นฐานบางส่วนจากบทก่อนๆ นี้มาใช้งานร่วมด้วย เพื่อเป็นการทบทวนเนื้อหาในเรื่องที่สำคัญให้เกิดความเข้าใจมากยิ่งขึ้น และช่วยให้เรามองเห็นแนวทางการนำความรู้เหล่านั้นไปใช้งานในรูปแบบที่แตกต่างออกไป

### ตัวอย่าง 16-1 ไฟล์บันทึกคำคุณ

ในตัวอย่างนี้ เราจะให้ผู้ใช้ใส่คำคุณที่เป็นภาษาอังกฤษเข้ามาทางคีย์บอร์ด จากนั้นนำไปบันทึกลงในไฟล์แบบข้อความ (Text File) โดยจะวนลูปเพื่อรับคำคุณไปเรื่อยๆ จนกว่าผู้ใช้จะกด <ctrl+z> จึงจะถือว่าลิ้นสุดการใส่ข้อมูล

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define NOFILE "can't open file"; exit(0)

void main() {
    FILE *fpt;
    fpt = fopen("files\\ex16-1.txt", "a+");
    if (fpt == NULL) {
        NOFILE;
    }

    printf(
        "\npress ctrl+z to cancel\n%s",
        "-----\n"
    );
}
```

```

char quote[150];
int n = 1;
while (1) {
    printf("\nenter quote >>");
    gets(quote);

    //ถ้ากด <ctrl+z> ให้หยุดรับข้อมูล
    if (feof(stdin)) {
        break;
    }

    fputs(quote, fpt);
    fputc('\n', fpt);

    n++;
}

printf("\n--- list of quotes ---\n");

//เลื่อนพอยโนเตอร์กลับไปยังจุดเริ่มต้น เพื่ออ่านไฟล์
rewind(fpt);

//ต้ายังไม่ลื้นสุดไฟล์ ก็อ่านไปเรื่อยๆ
while (!feof(fpt)) {
    printf("%c", fgetc(fpt));
}

fclose(fpt);
}

```

```

press ctrl+z to cancel
-----
enter quote >>well begun is half done
enter quote >>done is better than perfect
enter quote >>fake it until you make it
enter quote >>no pain no gain
enter quote >>the biggest risk is not taking any risk
enter quote >>every situation in life is temporary
enter quote >>the journey of thousand miles begins with a first step

```

```

enter quote >>^Z
--- list of quotes ---
well begun is half done
done is better than perfect
fake it until you make it
no pain no gain
the biggest risk is not taking any risk
every situation in life is temporary
the journey of thousand miles begins with a first step

```

### ตัวอย่าง 16-2 สร้าง Log File บันทึกเหตุการณ์

เราจะเลียนแบบการสร้าง Log File แบบง่ายๆ ที่ข้อมูลในแต่ละบรรทัดจะประกอบด้วยค่าวันเดือนปีและเวลาในขณะนั้น แล้วตามด้วยเหตุการณ์ที่เกิดขึ้น ซึ่งเราจะวนลูปเพื่อรับค่าทางคีย์บอร์ดที่เป็นเหตุการณ์ที่เกิดขึ้นในแต่ละขณะ จากนั้นก็เขียนลงในไฟล์ จนกว่าผู้ใช้จะกด <ctrl+z> จึงจะยกเลิก แต่ถ้ายังไม่กด <ctrl+z> ลิ้งที่เราต้องพิจารณาเพิ่มเติมคือ เราจะได้ค่าวันเวลาจากไหน ซึ่งจากบทที่ผ่านมา เราจะได้ทราบว่าในภาษา C มีมาโครที่ชื่อ \_\_DATE\_\_ และ \_\_TIME\_\_ แต่ปัญหาคือ ค่าที่ได้จากมาโครทั้ง 2 อันนี้จะคงที่ตลอด (เป็นค่าที่อ่านมาเก็บไว้เมื่อเริ่มรันโปรแกรม) และจะไม่อัปเดตตามวันเวลาจริงที่เปลี่ยนแปลงไปในขณะที่รัน จนกว่าเราจะรันครั้งต่อไป ซึ่งไม่สอดคล้องกับลักษณะของ Log File ที่จะบันทึกวันเวลาจริงที่เกิดเหตุการณ์ดังนั้น ในตัวอย่างนี้ เราจะกำหนดวันเวลาด้วยวิธีที่ยังไม่เคยกล่าวถึงในหนังสือเล่มนี้มาก่อน แต่ไม่มีอะไรถูกยาก นั่นคือ

- รับค่าวันเวลาปัจจุบันด้วยฟังก์ชัน `time()` จะได้ค่าเป็นข้อมูลชนิด `time_t`
- แปลงค่าชนิด `time_t` ที่ได้จาก `time()` เป็นรูปแบบสติงด้วยฟังก์ชัน `ctime()`
- ค่าที่ได้จาก `ctime()` จะอยู่ในรูปแบบ ชื่อวัน ชื่อเดือน วันที่ ชั่วโมง:นาที:วินาที ปี.ค.ศ. โดยชื่อวันและเดือนจะเป็นอักษร 3 ตัวแรก เช่น Sat Dec 31 12:30:59 2022
- ค่าที่ได้จาก `ctime()` จะมี \n ต่อท้าย แต่เนื่องจากเราต้องการเขียนข้อความซึ่งเป็นเหตุการณ์ที่เกิดขึ้นให้อยู่บรรทัดเดียวกับวันเวลา ดังนั้น ต้องตัด \n ที่ต่อท้ายวันเวลาออกไป หากว่าเราเคยทำมาแล้วในเรื่องการอ่านไฟล์ หรือดูจากโค้ดได้เลย

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

#define NOFILE "can't open file"; exit(0)

void main() {
    FILE *fpt;
    fpt = fopen("files\\ex16-2.txt", "a+");
    if (fpt == NULL) {
        NOFILE;
    }

    printf(
        "\npress ctrl+z to cancel\n%s",
        "-----\n"
    )
}
```

```

);

char str[150];
int n = 1;
time_t tm;
char *t, last_char;

while (1) {
    printf("\nwhat's happening? >>");
    gets(str);

    if (feof(stdin)) {
        break;
    }

    time(&tm);
    t = ctime(&tm);           // มี \n ต่อท้าย

    // ตัด \n ท้ายสตริงวันเวลาโดยอักขระตัวนี้จะเป็นตัวสุดท้าย
    // ดังนั้น เรายังตรวจสอบตำแหน่งจากความยาวของสตริง
    // ต่อไปก็ลบออก 1 เพราะตำแหน่งอักขระเริ่มจาก 0
    // และเปลี่ยนอักขระในตำแหน่งนั้น (\n) เป็นอักขระว่าง (\0)
    last_char = strlen(t) - 1;
    t[last_char] = '\0';

    fprintf(fpt, "%s %s \n", t, str);

    n++;
}

fclose(fpt);
}

```

press **ctrl+z** to cancel

---

what's happening? >>check email  
 what's happening? >>drink coffee  
 what's happening? >>video conference  
 what's happening? >>have lunch  
 what's happening? >>^Z

**C example16-2.c** ex16-2.txt x

chapter16 > files > ex16-2.txt

1	Fri Sep 09 11:53:44 2022	check email
2	Fri Sep 09 11:54:30 2022	drink coffee
3	Fri Sep 09 11:56:06 2022	video conference
4	Fri Sep 09 12:00:23 2022	have lunch
5		

### ตัวอย่าง 16-3 สร้างไฟล์คล้ายกับเรียงเบอร์

เป็นการสร้างเลขสุ่มสำหรับการออกรางวัลที่ 1 - 5 รวมถึงรางวัลเลขหน้าและเลขท้าย จากนั้นเขียนลงในไฟล์ คล้ายกับเรียงเบอร์ ซึ่งแต่ละรางวัลมีจำนวนหมายเลขอ้างนี้

- รางวัลที่ 1 มีจำนวน 1 หมายเลขอ้าง
- รางวัลเลขท้าย 2 ตัว มีจำนวน 1 หมายเลขอ้าง
- รางวัลเลขหน้า 3 ตัว และรางวัลเลขท้าย 3 ตัว มีรางวัลละ 2 หมายเลขอ้าง
- รางวัลที่ 2 มีจำนวน 5 หมายเลขอ้าง
- รางวัลที่ 3 มีจำนวน 10 หมายเลขอ้าง
- รางวัลที่ 4 มีจำนวน 50 หมายเลขอ้าง
- รางวัลที่ 5 มีจำนวน 100 หมายเลขอ้าง

ในแต่ละรางวัลนั้น จะใช้หลักการเดียวกันคือ สร้างเลขสุ่มที่เป็นเลขโดด (0 - 9) แล้วนำมาวางต่อ กันจนครบตามจำนวนหลักของรางวัลนั้นๆ เช่น รางวัลที่ 1 ซึ่งประกอบด้วยเลข 6 หลัก ก็สร้างเลขโดด 6 ตัวแล้วนำมาวางต่อ กัน เป็นต้น และรางวัลอื่นๆ ก็ทำแบบเดียวกัน ดังนั้น เพื่อไม่ให้เราต้องเขียนโค้ดซ้ำซ้อนกัน เราจะแยกชั้นตอนการสุ่มเลขเพื่ออกรางวัลมาสร้างเป็นฟังก์ชันไว้ต่างหาก โดยรับค่าพารามิเตอร์ที่บ่งชี้ว่ารางวัลนั้นมีเลขกี่หลัก และจำนวนกี่หมายเลขอ้างรายละเอียดต่างๆ ให้เพิ่มเติมจากโค้ดได้เลย

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#define NOFILE "can't open file"; exit(0)

FILE *fpt;

//prototype
void random_lotto(int len, int num_prizes);

void main() {
    fpt = fopen("files\\ex16-3.txt", "w");
    if (fpt == NULL) {
        NOFILE;
    }

    fprintf(fpt, "%s", "\n#1st prize \n");
    random_lotto(6, 1);
}
```

```

fprintf(fpt, "%s", "\n#last 2-digit prize \n");
random_lotto(2, 1);

fprintf(fpt, "%s", "\n#first 3-digit prize \n");
random_lotto(3, 2);

fprintf(fpt, "%s", "\n#last 3-digit prize \n");
random_lotto(3, 2);

fprintf(fpt, "%s", "\n#2nd prize \n");
random_lotto(6, 5);

fprintf(fpt, "%s", "\n#3rd prize \n");
random_lotto(6, 10);

fprintf(fpt, "%s", "\n#4th prize \n");
random_lotto(6, 50);

fprintf(fpt, "%s", "\n#5th prize \n");
random_lotto(6, 100);

fclose(fpt);

printf("\ncheck lotto numbers in files\\ex16-3.txt\n");
}

//ฟังก์ชันสำหรับสร้างเลขสุ่มเพื่อออกรางวัล
//โดยพารามิเตอร์ len คือจำนวนหลักของรางวัลนั้น
//และ num_prizes คือจำนวนหมายเลขของรางวัลนั้น
void random_lotto(int len, int num_prizes) {
    srand(time(0) + rand());

    int digit;      //เลขโดดที่สุ่มได้

    //ลูป for ซึ่งวนอก วนตามจำนวนหมายเลขของรางวัลนั้น
    for (int i = 1; i <= num_prizes; i++) {

        //ลูป for ซึ่งในเพื่อสร้างเลขสุ่ม 0 - 9
        //จนครบจำนวนหลักของรางวัลนั้นๆ
        //แล้วก็นำมาระบบต่อกันจนครบ
        //จากนั้นก็เขียนลงในไฟล์
        for (int j = 0; j < len; j++) {
            digit = rand() % 10;
            fprintf(fpt, "%d", digit);
        }

        //เว้นระยะระหว่างหมายเลขเท่ากับ 1 แท็บ
    }
}

```

```

fputc('\t', fpt);

//บางรางวัลอาจมีจำนวนเกิน 10 หมายเลขอ
//เราจะแสดงเพียงและไม่เกิน 10 หมายเลข
//ดังนั้น หากเขียนจนครบจำนวนครั้งเป็นเลขเต็ม 10
//ก็ให้ขึ้นบรรทัดใหม่
if (i % 10 == 0 && i < num_prizes) {
    fputc('\n', fpt);
}

fputc('\n', fpt);
}

```

C example16-3.c ex16-3.txt X

chapter16 > files > ex16-3.txt

```

1
2 #1st prize
3 542484
4
5 #last 2-digit prize
6 11
7
8 #first 3-digit prize
9 014 610
10
11 #last 3-digit prize
12 696 151
13
14 #2nd prize
15 055627 712087 870683 337481 894315
16
17 #3rd prize
18 330756 430356 347688 424772 963226 792557 317606 515362 436292 285048
19
20 #4th prize
21 589069 267116 743717 369640 620103 787601 404951 618689 144462 525111
22 226934 385496 120020 626913 118044 483953 647923 799463 381437 983307
23 309976 926939 431542 945849 036425 725399 436132 411561 372311 433653
24 912908 162995 199308 896526 692208 090468 652556 458236 506410 564270
25 916572 887838 262984 316523 268243 674400 871412 961899 931530 121499
26
27 #5th prize
28 424074 348176 036863 569580 197657 721610 655067 454481 425732 474685
29 679089 457159 383135 190271 255020 231588 389326 092012 724206 658053
30 326324 974594 149708 950377 291819 185053 139931 208526 746123 375558
31 794548 444281 167154 267326 583278 539016 494367 462270 295655 203087
32 932954 596700 916202 742852 287172 518425 004644 708686 326676 990188
33 563447 251607 393990 717778 776042 230619 187128 459010 556947 912483
34 650495 750244 134516 426007 424980 776310 272036 438678 995320 080379
35 122702 931755 960437 067958 057284 925038 003811 376074 188345 336146
36 233004 984832 417254 591186 812976 160882 542917 959428 231650 682781
37 933724 324026 912026 583184 879962 021838 629766 567634 839902 178048
38

```

**ตัวอย่าง 16-4 กำหนดข้อมูลทางคีย์บอร์ดให้แก่สตรีกเจอร์**

ตัวอย่างนี้จะเป็นการทบทวนเกี่ยวกับสตรีกเจอร์ โดยจะเป็นโครงสร้างข้อมูลของนักเรียน ซึ่งค่าที่เราต้องใส่ประกอบด้วย ชื่อ และคะแนนในบางวิชา เข้ามาทางคีย์บอร์ด แต่เนื่องจาก โดยทั่วไปแล้ว นักเรียนจะมีมากกว่า 1 คน ดังนั้น เราต้องจัดเก็บข้อมูลในแบบอาร์เรย์ของ สตรีกเจอร์ ซึ่งจะรับค่าไปเรื่อยๆ จนกว่าผู้ใช้จะกด <ctrl+z> และเราก็อ่านค่าจากอาร์เรย์ไปแสดง ผลแบบธรรมดा

```
#include <stdio.h>

//prototype
char check_grade(int score);

//สตรีกเจอร์ข้อมูลนักเรียน
struct student {
    int id;
    char name[30];
    int math, sci, eng;      //คะแนนในบางวิชา
} std[] = {};

void main() {
    int num_students = 5;

    printf(
        "\nEnter name math sci eng score\n%s",
        "e.g >>John 75 69 88\n\n"
    );

    struct student s;

    //วนลูปเพื่อรับข้อมูลไปเรื่อยๆ จนกว่าจะกด <ctrl+z>
    int i = 0;
    while (1) {
        printf("student #%d data >>", (i + 1));

        scanf(
            "%s%d%d%d",
            std[i].name,
            &std[i].math,
            &std[i].sci,
            &std[i].eng
        );
    }
}
```

```
if (feof(stdin)) { //ถ้ากด <ctrl+z>
    break;
}

//ค่า id ให้ใช้ลำดับของอูป เพื่อลดความยุ่งยาก
std[i].id = i + 1;

i++;
}

i = 0;
while (1) {
    s = std[i];

    if (s.id == 0) {
        break;
    }

    printf(
        "\nid: %d -- name: %s -- ",
        s.id, s.name
    );

    //นอกจากราการแสดงตัวเลขคะแนนแต่ละวิชาแล้ว
    //เรายังต้องส่งคะแนนไปยังฟังก์ชันที่ใช้ตรวจสอบเกรด
    //เพื่อนำกลับมาแสดงผลอีกด้วย
    printf(
        "math: %d(%c) -- sci: %d(%c) -- eng: %d(%c)",
        s.math, check_grade(s.math),
        s.sci, check_grade(s.sci),
        s.eng, check_grade(s.eng)
    );

    i++;
}

putchar('\n');
}

//ฟังก์ชันสำหรับตรวจสอบว่า
//ระดับคะแนนที่รับเข้ามาทางพารามิเตอร์
//ควรจะได้เกรดอะไร
char check_grade(int score) {
    char g;
```

```

if (score >= 80) g = 'A';
else if (score >= 70) g = 'B';
else if (score >= 60) g = 'C';
else if (score >= 50) g = 'D';
else if (score >= 1) g = 'F';
else g = '?';

return g;
}

```

```

enter name math sci eng score
e.g >>John 75 69 88

student #1 data >>Tom 45 59 61
student #2 data >>Jerry 87 64 92
student #3 data >>Ben 65 79 88
student #4 data >>Bean 33 52 75
student #5 data >>^Z

id: 1 -- name: Tom -- math: 45(F) -- sci: 59(D) -- eng: 61(C)
id: 2 -- name: Jerry -- math: 87(A) -- sci: 64(C) -- eng: 92(A)
id: 3 -- name: Ben -- math: 65(C) -- sci: 79(B) -- eng: 88(A)
id: 4 -- name: Bean -- math: 33(F) -- sci: 52(D) -- eng: 75(B)

```

### ตัวอย่าง 16-5 เขียนข้อมูลสตรัคเจอร์ลงในไฟล์ใบหนารี

จากตัวอย่างที่แล้ว เรารับข้อมูลสตรัคเจอร์ทางคีย์บอร์ดแล้วก็เพียงแค่แสดงผลออกไป และในตัวอย่างนี้ เราจะปรับปรุงจากตัวอย่างที่แล้ว โดยหลังจากที่รับข้อมูลเข้ามาแล้ว ก็นำไป เขียนลงในไฟล์ใบหนารี แต่ยังไม่มีการแสดงผลเพื่อให้เราบทวนแค่พื้นฐานไปก่อน ซึ่งขั้นตอนต่างๆ ก็เหมือนกับที่เราได้เรียนรู้มาแล้วจากบทที่ 14 (ตัวอย่างนี้ แค่รับข้อมูลแล้วเขียนลงในไฟล์ ส่วน การแสดงผล จะกล่าวถึงในตัวอย่างถัดไป เพื่อลดความซับซ้อน)

```

#include <stdio.h>
#include <stdlib.h>

#define NOFILE "can't open file"; exit(0)

struct student {
    int id;
    char name[30];
    int math, sci, eng;
} std[] = {};

void main() {
    FILE *fpt;

```

```
fpt = fopen("files\\ex16-5.dat", "wb");
if (fpt == NULL) {
    NOFILE;
}

printf(
    "\nenter name math sci eng score\n%s",
    "e.g >>John 75 69 88\n\n"
);

struct student s;

//วนลูปเพื่อรับข้อมูลไปเรื่อยๆ จนกว่าจะกด <ctrl+z>
int i = 0;
while (1) {
    printf("student # %d data >>", (i + 1));
    scanf(
        "%s%d%d%d",
        std[i].name,
        &std[i].math,
        &std[i].sci,
        &std[i].eng
    );

    if (feof(stdin)) { //ถ้ากด <ctrl+z>
        break;
    }

    //ค่า id ให้ใช้ลำดับของลูป เพื่อลดความยุ่งยาก
    std[i].id = i + 1;

    i++;
}

//ขนาดของสตริง (เรคคอร์ด)
size_t size = sizeof(struct student);

//เขียนข้อมูลแต่ละเรคคอร์ดลงในไฟล์
i = 0;
while (1) {
    if (std[i].id == 0) {
        break;
    }

    fwrite(&std[i], size, 1, fpt);
```

```

        i++;
    }

    printf("\ndata saved\n");

    fclose(fpt);
}

```

```

enter name math sci eng score
e.g >>John 75 69 88

student #1 data >>Tom 45 59 61
student #2 data >>Jerry 87 64 92
student #3 data >>Ben 65 79 88
student #4 data >>Bean 33 52 75
student #5 data >>^Z

data saved

```

### ตัวอย่าง 16-6 แสดงตารางข้อมูลสตรกเจอร์จากไฟล์ใบนารี

จากตัวอย่างที่แล้ว เราอ่านข้อมูลสตรกเจอร์ทางคีย์บอร์ดแล้วเขียนลงในไฟล์ใบนารี และในตัวอย่างนี้ เราจะอ่านข้อมูลจากไฟล์ดังกล่าวมาแสดงผลในแบบตาราง ซึ่งต้องมีการจัดรูปแบบ หรือกำหนดขนาดความกว้างของข้อมูลแต่ละค่า การจัดเรียงจึงมีลักษณะคล้ายกับตาราง ซึ่งความจริงลักษณะการแสดงผลเช่นนี้ เรายังเคยทำในบางตัวอย่างของบทที่ผ่านๆ มาแล้ว แต่คราวนี้จะซับซ้อนกว่าเดิมเล็กน้อย โดยพื้นฐานบางส่วนก็ได้แก่ ลักษณะของข้อมูลที่ต้องการ ตัวอย่างที่ผ่านมา ดังนั้น จึงขอเน้นอธิบายเฉพาะส่วนการจัดรูปแบบตารางเท่านั้น ดังภาพ

<i>id</i>	<i>name</i>	<i>math</i>	<i>grade</i>	<i>science</i>	<i>grade</i>	<i>english</i>	<i>grade</i>
<code>%4s \t% -15s \t%10s \t%4s \t%10s \t%4s \t%10s \t%4s</code>							
<i>id</i>	<i>name</i>	<i>math</i>	<i>grade</i>	<i>science</i>	<i>grade</i>	<i>english</i>	<i>grade</i>
1	Tom	45	F	59	D	61	C
2	Jerry	87	A	64	C	92	A
<code>%4d \t% -15s \t%10d \t%4c \t%10d \t%4c \t%10d \t%4c</code>							

คอลัมน์ที่เป็นการแสดงเกรด ซึ่งไม่ได้จัดเก็บข้อมูลลงในไฟล์ เรายังคงใช้ช่วงสำหรับการตรวจสอบแยกไว้ต่างหาก เมื่อกับตัวอย่างที่ 16-4 นั่นเอง

```

#include <stdio.h>
#include <stdlib.h>

#define NOFILE "can't open file"; exit(0)

//prototype
char check_grade(int score);

struct student {
    int id;
    char name[30];
    int math, sci, eng;
}

```

```
};

void main() {
    FILE *fpt;
    fpt = fopen("files\\ex16-5.dat", "rb");
    if (fpt == NULL) {
        NOFILE;
    }

    struct student std;
    size_t size = sizeof(struct student);

    //แสดงส่วนหัวของตาราง
    printf(
        "\n%4s\t%-15s\t%10s\t%4s\t%10s\t%4s\t%10s\t%4s\n%s%s%s",
        "id", "name", "math", "grade",
        "science", "grade", "english", "grade",
        "-----",
        "-----",
        "-----"
    );

    //แสดงส่วนข้อมูลของตาราง
    while (!feof(fpt)) {
        size_t result = fread(&std, size, 1, fpt);

        if (result == 0) {
            break;
        }

        printf(
            "\n%4d\t%-15s\t%10d\t%4c\t%10d\t%4c\t%10d\t%4c",
            std.id,
            std.name,
            std.math, check_grade(std.math),
            std.sci, check_grade(std.sci),
            std.eng, check_grade(std.eng)
        );
    }

    putchar('\n');
    fclose(fpt);
}

char check_grade(int score) {
    char g;
```

```

if (score >= 80) g = 'A';
else if (score >= 70) g = 'B';
else if (score >= 60) g = 'C';
else if (score >= 50) g = 'D';
else if (score >= 1) g = 'F';
else g = '?';

return g;
}

```

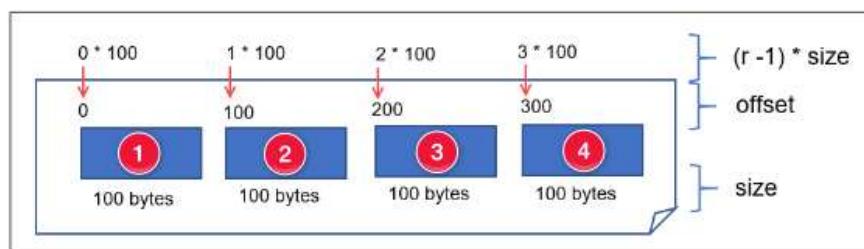
id	name	math	grade	science	grade	english	grade
1	Tom	45	F	59	D	61	C
2	Jerry	87	A	64	C	92	A
3	Ben	65	C	79	B	88	A
4	Bean	33	F	52	D	75	B

### ตัวอย่าง 16-7 จัดการข้อมูลสตรัคเจอร์ในไฟล์ใบหนารี

จากตัวอย่างนี้ จะเป็นการนำตัวอย่างตั้งแต่ 16-4 จนถึง 16-6 มาประยุกต์เข้าด้วยกัน ซึ่งเป็นการกำหนดสตรัคเจอร์ของข้อมูลนักเรียนแล้วเก็บในไฟล์ใบหนารีเช่นเดิม โดยเราต้องเพิ่มข้อมูลเข้าไปเองทางคีย์บอร์ด และวิธีเลือกแสดงผลได้ว่าจะแสดงผลทุกเรคอร์ด หรือจะแสดงเพียงเรคอร์ดอันใดอันหนึ่งด้วยการระบุค่า id ของเรคอร์ดนั้น ดังนั้น จึงมีขั้นตอนมากกว่าตัวอย่างที่ผ่านมา ซึ่งลักษณะคล้ายอยู่ที่เรารู้เพิ่มเติมสำหรับตัวอย่างนี้คือ

- ตัวอย่างนี้ จะมีเมนูให้เลือกซึ่งผู้ใช้ต้องเลือกโดยใส่ตัวเลขทางคีย์บอร์ด อย่างโดยอย่างหนึ่งคือ
    - แสดงข้อมูลทุกเรคอร์ด
    - แสดงข้อมูลของเรคอร์ดที่มีค่า id ตามที่ระบุ
    - เพิ่มข้อมูลใหม่
    - หรือค่าอื่นๆ นอกเหนือจาก 1-3 เป็นการหยุดทำงาน
  - ในตัวอย่างนี้ หากยังไม่มีข้อมูลเก็บอยู่ในไฟล์มาก่อน เริ่มแรกเรายังเพิ่มข้อมูลลงในไฟล์ โดยเลือกเมนูหมายเลข 3 จากนั้นก็กำหนดค่าให้มีอ่อนตัวอย่างที่ผ่านมา แต่ลิ่งที่เพิ่มเข้ามาคือ ให้กำหนดทั้งชื่อและนามสกุล จากนั้นทั้งชื่อและนามสกุลจะถูกนำไปเชื่อมต่อให้เป็นสมาชิกค่าเดียวกัน
- 1) show all      2) show by id  
 3) add new      otherwise exit  
 select >>
- 1) show all      2) show by id  
 3) add new      otherwise exit  
 select >>  
  
 enter firstname lastname math sci eng score  
 e.g >>Tom Jerry 75 69 88  
  
 student #1 data>>Elon Musk 80 99 75

- ขั้นตอนการเพิ่มข้อมูล เรานำข้อมูลที่รับเข้ามาเขียนลงในไฟล์ด้วยฟังก์ชัน `fwrite()` ตามเดิม โดยจะเป็นการเพิ่มต่อท้ายข้อมูลเดิม แต่สิ่งที่ต้องดำเนินการเพิ่มเติมคือ ข้อมูลแต่ละเรคอร์ดต้องมีคอลัมน์หรือสมาชิกที่เป็นค่า `id` ซึ่งค่านี้ เราไม่รับทางคีย์บอร์ด แต่จะกำหนดค่าเป็นตัวเลขลงไปให้เองโดยอัตโนมัติ และเพื่อให้เป็นตัวเลขที่ต่อเนื่องจากเรคอร์ดก่อนนี้ เราจะตรวจสอบว่าเรคอร์ดสุดท้ายมีค่า `id` เป็นเท่าไรแล้วหากเพิ่มไปอีก 1 จากนั้นก็นำตัวเลขนี้ไปเป็นค่า `id` ของเรคอร์ดใหม่ที่จะเพิ่มลงไป
- การแสดงข้อมูล ก็ทำในแบบตาราง เช่นเดียวกับตัวอย่างที่ผ่านมา แต่เนื่องจากเรามี 2 ทางเลือกคือ แสดงข้อมูลทั้งหมด กับแสดงเฉพาะเรคอร์ดที่มีค่า `id` ตามที่ระบุ ซึ่งการแสดงข้อมูลทั้งหมดเราก็อ่านตั้งแต่เริ่มต้นไปจนถึงสุดไฟล์ตามหลักการเดิม แต่หากแสดงตามค่า `id` ซึ่งในที่นี้ค่า `id` จะตรงกับลำดับเรคอร์ด ดังนั้น เราก็ใช้เทคนิคเดียวกับตัวอย่างสุดท้ายของบทที่ 14 นั่นคือ ต้องเลื่อนพอยน์เตอร์ไปยังเรคอร์ดเป้าหมายก่อน โดยนำค่า `id` ไปคูณกับขนาดของเรคอร์ด จากนั้นก็อ่านข้อมูลมาแสดงเพียงเรคอร์ดเดียว



สำหรับขั้นตอนปลีกย่อยอื่นๆ ส่วนใหญ่เป็นเรื่องเดิมที่ใช้ประกอบในตัวอย่างก่อนๆ นี้ ทั้งหมด ซึ่งสามารถดูเพิ่มเติมจากโค้ดต่อไปนี้ได้เลย

```
#include <stdio.h>
#include <stdlib.h>

#define NOFILE "can't open file"; exit(0)

FILE *fpt;
const char *path = "files\\ex16-7.dat";

//prototypes
char check_grade(int score);
void show_menu();
void show_data(int id);
void add_new();
```

```

struct student {
    int id;
    char name[30];
    int math, sci, eng;
} std;

void main() {
    fpt = fopen(path, "ab+");
    if (fpt == NULL) {
        NOFILE;
    }

    show_menu();
}

//ฟังก์ชันสำหรับเลือกเมนู
void show_menu() {
    int m;
    printf(
        "\n%s\n%s\n%s",
        "1) show all\t2) show by id",
        "3) add new\totherwise exit",
        "select >>"
    );
    scanf("%d", &m);

    if (m == 1) {
        show_data(0);
    } else if (m == 2) {
        //ถ้าเลือกแสดงผลตามค่า id
        //ให้รับค่า id ทางคีย์บอร์ด
        int id;
        printf("enter id >>");
        scanf("%d", &id);
        show_data(id);
    } else if (m == 3) {
        add_new();
    } else {
        fclose(fpt);
        exit(0);
    }
}

//ฟังก์ชันแสดงข้อมูล โดยรับค่าพารามิเตอร์เป็นค่า id
//ถ้าค่า id เป็น 0 แสดงทั้งหมด
//มีฉะนั้น แสดงข้อมูลของเรccord ที่มีค่า id ตามที่ระบุ
void show_data(int id) {

```

```

size_t size = sizeof(struct student);

//ส่วนหัวตาราง
printf(
    "\n%4s\t%-15s\t%10s\t%4s\t%10s\t%4s\t%10s\t%4s\n%s%s",
    "id", "name", "math", "grade", "science",
    "grade", "english", "grade",
    "-----",
    "-----"
);

//ถ้า id = 0 คือแสดงทุกเรคอร์ด
//ให้เลื่อนพอยน์เตอร์ไปจุดเริ่มต้นของไฟล์
if (id == 0) {
    rewind(fpt);
}
//ถ้าระบุค่า id (รับเข้ามาทางพารามิเตอร์)
//ให้เลื่อนพอยน์เตอร์ไปจุดเริ่มต้นของเรคอร์ดนั้น
else {
    fseek(fpt, (id-1) * size, SEEK_SET);
}

//อ่านข้อมูลไปแสดงผล
while (!feof(fpt)) {
    size_t result = fread(&std, size, 1, fpt);
    if (result == 0) {
        break;
    }

    printf(
        "\n%4d\t%-15s\t%10d\t%4c\t%10d\t%4c\t%10d\t%4c",
        std.id,
        std.name,
        std.math, check_grade(std.math),
        std.sci, check_grade(std.sci),
        std.eng, check_grade(std.eng)
    );
}

//ถ้าระบุค่า id เป็นการแสดงเพียงเรคอร์ดเดียว
//ก็ออกจากลูป เพื่อไม่ต้องแสดงเรคอร์ดต่อไปอีก
if (id != 0) {
    break;
}
}

putchar('\n');

```

```

    show_menu();
}

//ฟังก์ชันสำหรับเพิ่มข้อมูลใหม่
void add_new() {
    size_t size = sizeof(struct student);
    int max_id;

    //เริ่มจากการตรวจสอบค่า id ของเรccordสุดท้าย
    //เพื่อนำไปกำหนดเป็น id ของเรccordที่จะเพิ่มใหม่
    rewind(fpt);
    while (!feof(fpt)) {
        size_t result = fread(&std, size, 1, fpt);

        if (result == 0) {
            break;
        }

        max_id = std.id;
    }

    //รับค่าทั้งชื่อและนามสกุล
    char fname[10], lname[10];
    printf(
        "\nEnter firstname lastname math sci eng score\n%s",
        "e.g >>Tom Jerry 75 69 88\n\t\t"
    );

    printf("student #d data >>", (max_id + 1));
    scanf(
        "%s%s%d%d%d",
        fname, lname,
        &std.math,
        &std.sci,
        &std.eng
    );

    //นำชื่อและนามสกุลมาเขียนต่อเป็นค่าเดียวกัน
    sprintf(std.name, "%s %s", fname, lname);

    //เพิ่มค่า id ของเรccordสุดท้ายที่ได้มา
    //เพื่อเป็น id ของเรccordใหม่
    std.id = max_id + 1;

    //เลื่อนพอยน์เตอร์ไปยังท้ายไฟล์ เพื่อเขียนต่อท้ายข้อมูลเดิม
    fseek(fpt, 0, SEEK_END);
}

```

```
//เขียนข้อมูลลงในไฟล์
fwrite(&std, size, 1, fpt);

show_menu();
}

char check_grade(int score) {
    char g;

    if (score >= 80) g = 'A';
    else if (score >= 70) g = 'B';
    else if (score >= 60) g = 'C';
    else if (score >= 50) g = 'D';
    else if (score >= 1) g = 'F';
    else g = '?';

    return g;
}
```

1) show all    2) show by id  
 3) add new    otherwise exit  
 select >>3  
 enter firstname lastname math sci eng score  
 e.g >>Tom Jerry 75 69 88  
 student #1 data >>Ben Ten 56 78 90

1) show all    2) show by id  
 3) add new    otherwise exit  
 select >>1

id	name	math	grade	science	grade	english	grade
1	Ben Ten	56	D	78	B	90	A

1) show all    2) show by id  
 3) add new    otherwise exit  
 select >>3  
 enter firstname lastname math sci eng score  
 e.g >>Tom Jerry 75 69 88  
 student #2 data >>Snow White 59 77 64

1) show all    2) show by id  
 3) add new    otherwise exit  
 select >>1

id	name	math	grade	science	grade	english	grade
1	Ben Ten	56	D	78	B	90	A
2	Snow White	59	D	77	B	64	C

```

1) show all      2) show by id
3) add new      otherwise exit
select >>2
enter id >>1

```

id	name	math	grade	science	grade	english	grade
1	Ben Ten	56	D	78	B	90	A

เนื้อหาต่างๆ ที่กล่าวมาทั้งหมดในหนังสือเล่มนี้ แม้จะไม่ครอบคลุมในทุกรายี แต่ก็น่าจะเป็นพื้นฐานให้ผู้อ่านสามารถเรียนรู้ด้วยตนเองได้ รวมทั้งเป็นแนวทางสำหรับการศึกษาเพิ่มเติมจากแหล่งข้อมูลอื่นๆ เพื่อเสริมทักษะการเขียนโปรแกรมให้เพิ่มพูนมากยิ่งขึ้น จนสามารถก้าวไปสู่ระดับผู้เชี่ยวชาญในการเขียนโปรแกรมด้วยภาษา C





# การเขียนโปรแกรม

# ภาษา C

## ขั้นพื้นฐาน

**ภาษา C** เป็นหนึ่งในภาษาคอมพิวเตอร์ที่มีผู้ใช้งานเป็นจำนวนมากในปัจจุบัน เนื่องจากลักษณะโครงสร้างที่ไม่ซับซ้อนจนเกินไป จึงเรียนรู้ได้ไม่ยาก รวมทั้งความเร็วในการประมวลผล และความสามารถที่โดดเด่นในอีกหลาย ๆ ด้าน ดังนั้น สถาบันการศึกษาส่วนใหญ่จึงมักเลือกภาษา C ในการเรียนการสอนสำหรับวิชาการเขียนโปรแกรมระดับเริ่มต้น และแม้ว่าภาษา C จะถูกสร้างมานานมากแล้ว แต่ซอฟต์แวร์หลัก ๆ ที่เราใช้งานกันอยู่ทุกวันนี้ ล้วนถูกสร้างจากพื้นฐานของภาษา C แทนทั้งสิ้น เช่น ระบบปฏิบัติการต่าง ๆ โปรแกรมด้านฐานข้อมูลและธุรกิจอุตสาหกรรม การคำนวณทางวิทยาศาสตร์ Game Engine และอื่น ๆ อีกมากมาย ดังนั้น การเขียนโปรแกรมด้วยภาษา C จึงเป็นสิ่งสำคัญอย่างหนึ่งที่ผู้ต้องการก้าวสู่สายงานด้านการพัฒนาซอฟต์แวร์ระดับมืออาชีพควรต้องเรียนรู้เอาไว้

### เนื้อหาในหนังสือประกอบด้วย :

- บทที่ 1 : การจัดเตรียมเครื่องมือ
- บทที่ 2 : การเขียนโค้ดภาษา C ในเบื้องต้น
- บทที่ 3 : ชนิดข้อมูลและตัวแปร
- บทที่ 4 : สตริง การรับและแสดงผลข้อมูล
- บทที่ 5 : ตัวเลขและการคำนวณ
- บทที่ 6 : การเปลี่ยนเทียบและกำหนดเงื่อนไข
- บทที่ 7 : การทำซ้ำแบบวนรอบ
- บทที่ 8 : รวมตัวอย่างโค้ดเพิ่มเติม ชุดที่ 1

- บทที่ 9 : การสร้างและใช้งานฟังก์ชัน
- บทที่ 10 : อาร์เรย์และสตริง (เพิ่มเติม)
- บทที่ 11 : การซึ่ดແղ່ງด้วยພອຍືນເຕົກ
- บทที่ 12 : รวมตัวอย่างโค้ดเพิ่มเติม ชุดที่ 2
- บทที่ 13 : ສຕັກເຈອົງ ຍູ້ເນີຍ ແລະ ເອັນນັມ
- บทที่ 14 : การเขียนและอ่านໄຟລ໌
- บทที่ 15 : ພຣີໂພເຮສເຊອົງ ແລະ ມາໂຄຣ
- บทที่ 16 : รวมตัวอย่างโค้ดเพิ่มเติม ชุดที่ 3



www.se-ed.com

sbc.fans

SE-ED Publisher

#### พร้อมจำหน่ายในรูปแบบ

- |  |   |
|--|---|
| <input checked="" type="checkbox"/> e-book (PDF) | <input type="checkbox"/> audiobooks                     |
| <input type="checkbox"/> e-book (EPUB)           | <input type="checkbox"/> audio CD / MP3                 |
| <hr/>  |   |
| <input checked="" type="checkbox"/> ปกอ่อน       | <input type="checkbox"/> LARGE PRINT (ตัวอักษรขนาดใหญ่) |

ISBN 978-616-08-4751-8

9 78616 0847518  
239 บาท