

# CE BOOK

## ce boost up



## สารบัญ

<b>วิธีดูนาฬิกา .....</b>	<b>E</b>
<b>ผังงาน (Flowchart) .....</b>	<b>1</b>
การเขียนผังงานที่ดี .....	1
ประโยชน์ของผังงาน .....	1
ข้อจำกัดของผังงาน .....	1
สัญลักษณ์ที่ใช้ในการเขียนผังงาน .....	1
รูปแบบของผังงาน .....	2
<b>รหัสเทียม (Pseudo Code) .....</b>	<b>3</b>
ประเภทรหัสเทียม (Pseudo Code Type) .....	3
Draft Code .....	3
Detailed Code .....	3
Simple Code .....	3
ตัวอย่าง .....	4
<b>ประวัติความเป็นมา .....</b>	<b>5</b>
<b>จุดเด่นของภาษา C .....</b>	<b>5</b>
<b>ข้อแตกต่างระหว่าง ภาษา C vs. C++ .....</b>	<b>5</b>
<b>โครงสร้างโปรแกรม .....</b>	<b>6</b>
เขตเดอร์ไฟล์ (Header File) .....	6
ฟังก์ชัน (Function) .....	6
ส่วนตัวโปรแกรม (Body) .....	6
<b>ตัวแปร (Variable) .....</b>	<b>7</b>
การตั้งชื่อไอเดียนติฟายเออร์ (Identifier Names) .....	7
การประกาศตัวแปร (Variable Declaration) .....	7
ชนิดของข้อมูล (Data Type) .....	8
ชนิดข้อมูลแบบจำนวนเต็ม (Integer Type) .....	8
ชนิดข้อมูลแบบจำนวนจริง (Real Floating Point Type) .....	9
ชนิดข้อมูลแบบตัวอักษร (Character Type) .....	9
ชนิดข้อมูลแบบสายอักษร (Group of Character Type) .....	9
Null Character และ Empty String .....	10
ประเภทตัวแปร (Variable Type) .....	10
การทำหนดค่าให้ตัวแปร (Variable Assignment) .....	11
<b>ค่าคงที่ (Constant) .....</b>	<b>11</b>
ระบุค่าโดยตรง (Literal Constants) .....	11
นิยามโดย #define (Defined Constants) .....	12
เก็บไว้ในตัวแปร (Memory Constants) .....	12
<b>การแปลงชนิดของข้อมูล (Data Type Conversion) .....</b>	<b>12</b>
Implicit Type Conversion .....	12
Explicit Type Conversion (Casting) .....	12
<b>การแสดงผลทางหน้าจอ (Display Data) .....</b>	<b>13</b>
1. ใช้ค่าสั่ง printf .....	13
การแสดงผลของข้อความ .....	13
การแสดงผลของค่าที่เก็บในตัวแปร .....	13
รหัสรูปแบบการแสดงผลข้อความ และการจัดรูปแบบการแสดงผลข้อมูลของค่าสั่ง printf() .....	15

การจัดพื้นที่ และการกำหนดการแสดงผลข้อมูลของค่าสั้ง printf() .....	15
2. ใช้ค่าสั้ง putchar() สำหรับแสดงแบบอักขระตัวเดียว (Character) .....	17
3. ใช้ค่าสั้ง puts() สำหรับแสดงข้อมูลแบบสายอักขระ (String) .....	17
<b>ตัวดำเนินการ (Operator).....</b>	<b>17</b>
ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operator).....	17
ตัวดำเนินการกำหนดค่า (Assignment Operator).....	18
ตัวดำเนินการยูนารี (Unary Operator).....	18
ตัวดำเนินการเปรียบเทียบ (Comparison Operator) .....	19
ตัวดำเนินการตรรกะ (Logical Operator).....	19
ตัวดำเนินการแบบมีเงื่อนไข (Conditional Operator).....	20
ตัวดำเนินการบอกขนาดชนิดข้อมูล (Sizeof Operator).....	20
ตัวดำเนินการระดับบิต (Bitwise Operator).....	20
Bitwise AND (&), Bitwise OR ( ), Bitwise Exclusive OR/XOR (^).....	20
Bitwise Shift Right (>>).....	21
Bitwise Shift Left (<<).....	21
Bitwise One's Complement (~) .....	21
ล่าดับความสำคัญของตัวดำเนินการ (Precedence) .....	22
<b>การรับข้อมูล (Input Data) .....</b>	<b>22</b>
1. การรับข้อมูลชนิดอักขระทีละตัวจากคีย์บอร์ดด้วยค่าสั้ง getch() และ getchar .....	22
2. การรับข้อมูลชนิดข้อความจากคีย์บอร์ดด้วยค่าสั้ง gets() .....	23
3. การรับข้อมูลทุกชนิดจากคีย์บอร์ดด้วยค่าสั้ง scanf() .....	23
การกำหนดล่าดับการรับข้อมูลของค่าสั้ง scanf().....	24
การกำหนดจำนวนตัวอักขระในการรับค่าของข้อมูลชนิดข้อความของค่าสั้ง scanf() .....	25
การนับจำนวนตัวอักขระที่รับค่าของข้อมูลด้วยค่าสั้ง scanf() .....	25
Regular Expression .....	25
<b>ควบคุมทิศทางการทำงานโปรแกรมด้วยค่าสั้งตัดสินใจ (Decision Control Statement).....</b>	<b>26</b>
ค่าสั้งตัดสินใจแบบเลือกทำ หรือไม่ทำด้วยค่าสั้ง if .....	26
ค่าสั้งตัดสินใจแบบสองทางเลือกด้วยค่าสั้ง if...else .....	26
ค่าสั้งตัดสินใจแบบหลายทางเลือกด้วยค่าสั้ง if...else if.....	26
ค่าสั้งตัดสินใจแบบหลายทางเลือกด้วยค่าสั้ง switch-case .....	27
<b>ควบคุมการทำงานของโปรแกรมด้วยค่าสั้งวนลูป (Repetition Control Statement).....</b>	<b>28</b>
วนลูปการทำงานด้วยจำนวนรอบที่แน่นอนด้วยค่าสั้ง for .....	28
วนลูปการทำงานเมื่อเงื่อนไขเป็นจริงด้วยค่าสั้ง while .....	29
วนลูปการทำงานอย่างน้อย 1 ครั้งด้วยค่าสั้ง do...while .....	29
ค่าสั้ง break และ continue .....	29
อธิบายลูป for อย่างละเอียด .....	30
<b>อาร์เรย์ (Array).....</b>	<b>31</b>
ตัวแปรอาร์เรย์ 1 มิติ (One Dimension Array) .....	31
ตัวแปรอาร์เรย์ 2 มิติ (2-Dimension Array) .....	31
ตัวแปรอาร์เรย์ 3 มิติ (3-Dimension Array) .....	31
การประกาศตัวแปรอาร์เรย์ .....	32
การกำหนดค่าข้อมูลให้ตัวแปรอาร์เรย์ .....	32
การอ้างถึงข้อมูลในตัวแปรอาร์เรย์ .....	33
<b>ฟังก์ชัน (Function).....</b>	<b>34</b>
ฟังก์ชันที่มาตราฐานที่ติดมากับภาษา C (Standard Library Function).....	34

math.h เกี่ยวกับการคำนวนทางคณิตศาสตร์ .....	34
string.h เกี่ยวกับข้อความ .....	35
ctype.h เกี่ยวกับตัวอักษรตัวเดียว .....	35
stdlib.h เกี่ยวกับการแปลงค่า string .....	35
dos.h เกี่ยวกับการติดต่อระบบปฏิบัติการ .....	35
conio.h เกี่ยวกับการแสดงผลทางจอภาพ .....	36
<b>ฟังก์ชันที่สร้างขึ้นเอง (User-Defined Functions) .....</b>	<b>36</b>
ฟังก์ชันที่ไม่มีการคืนค่ากลับและไม่มีการรับค่าพารามิเตอร์ (Void Functions with No Parameters) .....	36
ฟังก์ชันที่ไม่มีการคืนค่ากลับ แต่มีการรับค่าพารามิเตอร์ (Void Functions with Parameters) .....	37
ฟังก์ชันที่มีการคืนค่ากลับ แต่ไม่มีการรับค่าพารามิเตอร์ (Function Return Value with No Parameters) .....	37
ฟังก์ชันที่มีการคืนค่ากลับ และมีการรับค่าพารามิเตอร์ (Function Return Value with Parameters) .....	37
<b>พอยเตอร์ (Pointer) .....</b>	<b>40</b>
การประกาศตัวแปรประเภทพอยน์เตอร์ (Pointer Variable Declaration) .....	40
การทำหนดค่าให้ตัวแปรพอยน์เตอร์ (Pointer Variable Assignment) .....	40
การดำเนินการทางคณิตศาสตร์กับพอยน์เตอร์ (Pointer Arithmetics) .....	41
การอ้างถึงข้อมูลภายในตัวแปรพอยน์เตอร์ (Dereference Operator) .....	41
การใช้งานพอยน์เตอร์ .....	42
การใช้งานกับพอยน์เตอร์ 1 ตัว .....	42
การใช้งานกับพอยน์เตอร์หลายตัว .....	43
การใช้งานพอยน์เตอร์กับอาร์เรย์ .....	44
การใช้งานพอยน์เตอร์กับข้อความ .....	45
การใช้งานพอยน์เตอร์อ้างอิงพอยน์เตอร์ที่กำลังอ้างอิงถึงตัวแปรอื่น .....	46
พอยน์เตอร์กับฟังก์ชัน .....	46
การส่งค่าที่เก็บอยู่ในตัวแปรให้กับฟังก์ชัน (Pass by Value) .....	46
การส่งค่าที่อยู่ของตัวแปรให้กับฟังก์ชัน (Pass by Pointer) .....	46
การส่งการอ้างอิงถึงตัวแปรให้กับฟังก์ชัน (Pass by Reference) .....	46
<b>โครงสร้างข้อมูล (Structure) .....</b>	<b>47</b>
การนิยามโครงสร้างเพื่อใช้งาน (Structure Define) .....	47
การประกาศตัวแปรประเภทโครงสร้าง (Structure Variable Declaration) .....	47
การประกาศตัวแปรประเภทโครงสร้างหลายตัวโดยใช้อาร์เรย์ .....	48
การทำหนดค่าเริ่มต้นตอนประกาศตัวแปร การเข้าถึง และกำหนดค่าให้ตัวแปรประเภทโครงสร้าง .....	48
การใช้อาร์เรย์ภายในตัวแปรประเภทโครงสร้าง .....	48
โครงสร้างข้อนested โครงสร้าง (Nest Structure) .....	49
<b>ภาคผนวก .....</b>	<b>I</b>
คำศัพท์น่ารู้ .....	I
หมวดระบบคอมพิวเตอร์ .....	I
หมวดโครงสร้างภาษา .....	I
หมวดการแสดงผล และรับข้อมูล .....	II
หมวดนิพจน์ทางคณิตศาสตร์ .....	II
หมวดฟังก์ชันของภาษา C .....	III
หมวดคำสั่งเงื่อนไข .....	III
หมวดคำสั่งวนลูป .....	IV
ASCII TABLE .....	V
ผังอักษรและสกุลที่ไม่สามารถแสดงผล .....	V
ผังอักษรและสกุลที่สามารถแสดงผล .....	V

สรุปเนื้อหา By TK (Infographic for C Programming) .....	VI
ผังงาน (Flowchart) & โคดเทียน (Pseudo Code) .....	VI
ตัวแปร (Variable) & ค่าคงที่ (Constant) .....	VIII
การแสดงผลทางหน้าจอ (Display Data) .....	IX
การรับข้อมูล (Input Data) .....	X
รูปแบบที่ใช้บ่อย ๆ (Display Format-Input Format, Escape Sequence และ Flag) & การแปลงชนิดข้อมูล (Data Type Conversion) .....	XI
ตัวดำเนินการ (Operator) .....	XII
การทำงานด้วยการตัดสินใจด้วย if และ if-else (if and if-else – Decision Control Statement) .....	XIII
การทำงานด้วยการตัดสินใจด้วย switch-case (switch-case – Decision Control Statement) .....	XIV
การทำงานด้วยการวนลูปด้วย while และ do-while (while and do-while – Repetition Control Statement) ..	XV
การทำงานด้วยการวนลูปด้วย for (for – Repetition Control Statement) .....	XVI
ฟังก์ชัน (function) .....	XVII
อาร์เรย์ (Array) & ตัวแปรประเภทโครงสร้าง (Structure) .....	XVIII
ตัวแปรประเภทโครงสร้าง (Structure) – ต่อ .....	XIX
พอยน์เตอร์ (Pointer) .....	XX

## วิวัฒนาการ

น้อง ๆ หลายคนคงอาจจะสังสัยหัวข้อนี้ในสารบัญ วิวัฒนาการของภาษา C หรือพี่ เปป่า! มันคือ วิวัฒนาการของหนังสือเล่นน้อย ๆ เล่มนี้ยังไงหละ พากพีตั้งใจจะทำให้น้อง ๆ เอ้าไปศึกษา เตรียมความพร้อมก่อนเข้าเรียนในปี 1 ครับ :)

### CE57

เริ่มทำหนังสือนี้อย่างจริงจัง เพราะในค่าย CEBU#6 ที่รุ่นพากพีเข้ากันอะ หนังสือที่ปรับมาแจก "ไม่สิเรียกว่าชีท ตีกว่า พี่ ๆ เขายังไม่ได้ทำเป็นหนังสืออย่างจริงจังอะแหละ หลาย ๆ คนเลยได้มาระเกียบเรามากอง ๆ สุดท้ายพากพีก็ต้องไปหา Google เอาอยู่ดี พี่เลยตัดสินใจสร้างสิ่ง ๆ นี้ขึ้นมาครับ! ซึ่ง... มันก็ยังไม่เป็นหนังสือครอบครัว มีแค่เนื้อหา (พอยเดอร์ การเขียนไฟล์ก็ไม่มี) หน้าปก สารบัญ ภาคผนวกก็ยังไม่มี แต่จุดเด่นยังอะอีก สารภาพด้วยครับมีเวลาทำแค่ 1 สัปดาห์ 5555+ ขอภัยมาที่นี่อะ \/\ อายกรู้ความเป็นมาอะเอียดกว่านี้หรือ... ตามสิ แต่ไม่ออกครอบนะ อ้อ และพีขอส่งท้ายก่อนจากลา กันไปด้วยหนังสือส่วนสุดท้ายของเล่มกันครับ

#### พอยเดอร์ (Pointer)

พอกะอนน่อง.... แค่นี้ก็จะอ้วกแಡกดายแล้ว 5555 ยังเหลือเรื่องอ่าน-เขียนไฟล์นะถ้าอยากรู้เพิ่มเติม สำหรับหนังสือเล่นนี้พิมพ์ดิจิต หรือข้อมูลคิดพลาดตรงส่วนไหนพี่ ๆ ก็ขอภัยมา ณ ที่นี่อะ

มีตรงไหนสังสัยถามพี่ฯได้เนอะ อยากรู้โจทย์ก็ทักมาขอได้เหมือนกันมี source ให้ไปฟิก ขอให้โชค A สนูกับการอ่านหนังสือเล่นนี้ครับ :D

ACADEMIC TEAM - CE BOOST UP #7

~ 39 of 39 ~

[bit.ly/bookceboostup](http://bit.ly/bookceboostup)

Niranam...

### CE58

ในส่วนของปีนี้ ถ้าอยากรู้เรื่องราวให้กระจัง.... พากพี ๆ ก็พร้อมที่จะแตลงไข.. พี่ ๆ ได้سانต่อความตั้งใจของพี่ ๆ CE57 โดยน่าเนื้อหาจากค่าย CE BOOST UP #7 มาเพิ่มเติมส่วนพอยเดอร์ การเขียนโปรแกรมเชิงโครงสร้าง เพิ่มเติมส่วนเนื้อสรุปของหนังสือ รวมถึงหน้าปกอันละเอียด น่ารักเหมือนหน้าตาคนทำและคนอ่านด้วย โอ荷ว ๆ และคำศัพท์ที่ควรรู้ (ที่คาดว่าจะมีประโยชน์กับน้องหลาย ๆ ตอน เอ้ยคน มั่งนะ5555) สำหรับน้อง ๆ ที่เพิ่งฝึกหัดเขียนโปรแกรม เข้ามาอ่านแล้วอย่าเพิ่งห้อใจไปนะ ความพยายามไม่เคยทำร้ายสักคนที่ตั้งใจหัก ความมานะอุสาหะ และเวลาจะหลอมเราให้กล้ายเป็นคนที่เก่งเอง ขอแค่เราเก่งกว่าตัวเราในเวอร์ชันเมื่อวานก็พอแล้ว (น้อววว...เขียนเหมือนอ่านนิยายไปอีก) สำหรับน้อง ๆ ที่มีพื้นฐานมากบ้างแล้วน้า ก็เหมือนเป็นการทำทวนไปในตัว ฝึกฝนตัวเองต่อไปเรื่อยนะอุ่น ๆ ๆ เก้าเป็นกำลังใจให้น้าาา (ไม่เน้นยา สาระด้วย เน้นน่ารักนุญ ๆ ๆ)

หากมีเนื้อหาตรงไหนตกหล่นหรือเจอดูดพลาดตรงไหน หรือคิดว่าอยากรู้เพิ่มเติมส่วนเนื้อหานั้นสือตรงไหน สามารถติดต่อห้องไมค์พี่ ๆ ทุกคนได้เลยนะ ถ้าเลือกระหว่างการนอน กับการเรียน จะเลือกการนอน... "ไม่ใช่น้า" ในไข่ช้าง แต่ขอลาไปก่อนขอบคุณครับ

■Mayuyu■

### CE59

ครั้งนี้ก็ครั้งที่ 9 แล้ว ที่มีค่าย CE BOOST UP ค่ายนี้สืบทอดกันนานนานว่ากูกก ก รวมทั้งหนังสือเล่นนี้ด้วยก็ เช่นกัน ซึ่งปีนี้พี่แทบไม่ได้แกะอะไรหนังสือเลย เพราะพี่ฯปีที่แล้วเค้าทำไว้ตีมากแล้ว555555 พี่เองก็เคยเป็นน้องที่ใช้หนังสือเล่นนี้ จนตอนนี้มาเป็นพี่เองแล้ว ก็อยากรู้น้อง ๆ ได้รับในสิ่งที่พี่ปีก่อน ๆ ถ่ายทอดมาให้พี่ พี่หวังว่าหนังสือเล่นนี้ จะช่วยให้น้อง ๆ ได้เข้าใจภาษาซีที่เป็นภาษาพื้นฐานของเรามากขึ้น สามารถปรับใช้กับการใช้งานจริงได้ โชคดีกับการเรียนครับน้อง ๆ

สุดท่าน และมองเพื่อน

### CE60

สวัสดีครับน้องๆที่กำลังอ่านข้อความนี้ ค่ายครั้งนี้ก็เป็นครั้งที่ X แล้ว แซร์ 10 นั้นแหละ และพี่ก็เลยตั้งชื่อค่ายว่า CE BOOSTUP X เป็นไปจะเหลือท่องากเลยใช่ไหม ค่ายนี้ก็มีมาอย่างยาวนานแล้วนะครับยามากๆๆๆๆ จนพี่ก็จำไม่ได้เหมือนกัน ส่วนหนังสือเล่นนี้รุ่นพี่ของพากเราได้สร้างขึ้นมาซึ่งเป็นเนื้อหาที่ละเอียดในระดับนึง พี่ก็หวังว่าหนังสือเล่นนี้จะเป็นประโยชน์กับน้องๆและครับ โชคดีมีชัยในการเรียนครับ

ก็อดขอฟลิรัด และมองเพื่อน

## ผังงาน (Flowchart)

ผังงาน (Flowchart) คือ รูปแบบ หรือสัญลักษณ์ที่ใช้เขียนแทนคำอธิบาย ข้อความ หรือค่าพูดที่ใช้ในอัลกอริทึม เนื่องจาก การที่จะเข้าใจขั้นตอนได้ง่าย และตรงกันนั้นถ้าหากใช้ค่าพูด หรือข้อความจะทำได้ยากกว่าการใช้รูปภาพ หรือ สัญลักษณ์ที่เป็นสากล และคนทั่วโลกสามารถเข้าใจได้ง่าย

### การเขียนผังงานที่ดี

1. ใช้สัญลักษณ์ตามที่กำหนดไว้
2. ใช้ลูกศรแสดงทิศทางการทำงานของโปรแกรมจากนั้นลงล่าง หรือจากซ้ายไปขวา
3. คำอธิบายในภาพควรสั้นกระัดกระสินค้า และเข้าใจง่าย
4. ทุกแผนภาพต้องมีลูกศรแสดงทิศทางเข้า – ออก
5. “ไม่ควรโยงเส้นเชื่อมผังงานที่อยู่ใกล้มาก ๆ ควรใช้จุดเชื่อมต่อแทน”
6. ผังงานควรมีการทดสอบความถูกต้องของการทำงานก่อนนำไปเขียนโปรแกรม

### ประโยชน์ของผังงาน

1. ทำให้เข้าใจและแยกแยะปัญหาต่าง ๆ ได้ง่ายขึ้น
2. ผู้เขียนโปรแกรมสามารถมองเห็นลำดับการทำงาน รู้ว่าสิ่งใดควรทำก่อน สิ่งใดควรทำหลัง
3. สามารถหาข้อผิดพลาดของโปรแกรมได้ง่าย
4. ทำให้ผู้อื่นเข้าใจการทำงานได้ง่ายกว่าการดูจาก Source Code
5. ผู้อื่นสามารถเรียนรู้ และนำไปใช้เขียนเป็นภาษาที่ตนเองถนัดได้ง่าย

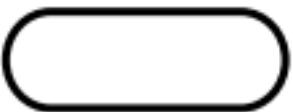
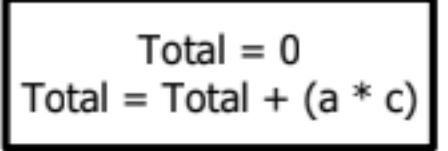
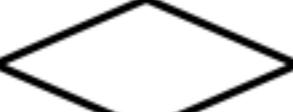
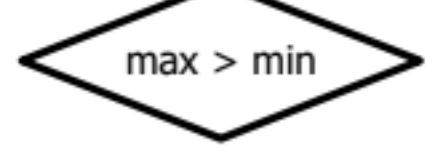
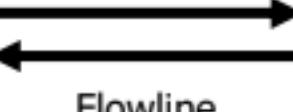
### ข้อจำกัดของผังงาน

คนที่เขียนโปรแกรมบางคนไม่นิยมเขียนผังงานก่อนเขียนโปรแกรม เพราะเห็นว่าเสียเวลา นอกเหนือนั้นยังมี ข้อจำกัดอื่น ๆ อีก ดัง

1. ผังงานเป็นการสื่อความหมายระหว่างบุคคลกับบุคคล มากกว่าที่จะสื่อความหมายระหว่างบุคคลกับเครื่อง เพราะผังงานไม่เขียนกับภาษาคอมพิวเตอร์ภาษาใดภาษาหนึ่ง ทำให้เครื่องไม่สามารถรับ และเข้าใจได้ว่าในผังงานนั้นต้องการ ให้ทำอะไร
2. ในบางครั้งเมื่อพิจารณาจากผังงาน จะไม่สามารถทราบได้ว่า ขั้นตอนการทำงานใดสำคัญกว่ากัน เพราทุก ๆ ขั้นตอน จะใช้รูปภาพ หรือสัญลักษณ์ในลักษณะเดียวกัน
3. การเขียนผังงานเป็นการลื้นเปลือง เพราะจะต้องใช้กระดาษ และอุปกรณ์อื่น ๆ เพื่อประกอบการเขียนภาพ

### สัญลักษณ์ที่ใช้ในการเขียนผังงาน

เป็นเครื่องมือ (Tools) ที่ใช้อธิบายรายละเอียดการทำงานตามขั้นตอนการทำงาน (Algorithm) แทนคำสั่งการ ทำงานของโปรแกรม โดยใช้สัญลักษณ์ (Symbol) แทนค่าสั่ง ที่นิยมใช้มีดังนี้

สัญลักษณ์แสดงขั้นตอนการทำงาน		
สัญลักษณ์	ความหมาย	ตัวอย่าง
	การกำหนดจุดเริ่มต้นของการทำงาน และแสดงจุดสิ้นสุดของ การทำงาน	 
	การแสดงรายละเอียดของการทำงาน และกระบวนการการทำงาน	 
	การแสดงรายละเอียดการเปรียบเทียบเงื่อนไขต่าง ๆ ใช้ในขั้นตอนที่มีการตัดสินใจว่าใช่ หรือไม่ใช่	
	การแสดงทิศทางความสัมพันธ์ของการทำงานในระบบงาน หรือ ลำดับงาน	
	การกำหนดจุดอ้างอิงในการเชื่อมต่อ ในหน้ากระดาษเดียวกัน ของการเขียนผังงานโครงสร้าง (Structured Flowchart)	

## สัญลักษณ์แสดงการรับ-ส่งข้อมูล

สัญลักษณ์	ความหมาย	ตัวอย่าง
Read or Write	<p><u>Read (รับค่า)</u> การรับค่าข้อมูล หรืออ่านข้อมูลเข้ามาโดยไม่ระบุอุปกรณ์รับข้อมูล (Input Device) โดยอาจรับค่าข้อมูลมาจากคีย์บอร์ด หรือจากไฟล์ข้อมูลก็ได้</p> <p><u>Write (แสดงผล)</u> การแสดงรายละเอียดข้อมูล หรือแสดงผลลัพธ์ของการประมวลผล โดยไม่ระบุอุปกรณ์การแสดงผล (Output Device) โดยอาจแสดงผลผ่านจอภาพ หรือเครื่องพิมพ์ (Printer) ก็ได้</p>	

### รูปแบบของผังงาน

ผังงานมีรูปแบบที่จำกัดอยู่ 3 รูปแบบ คือ

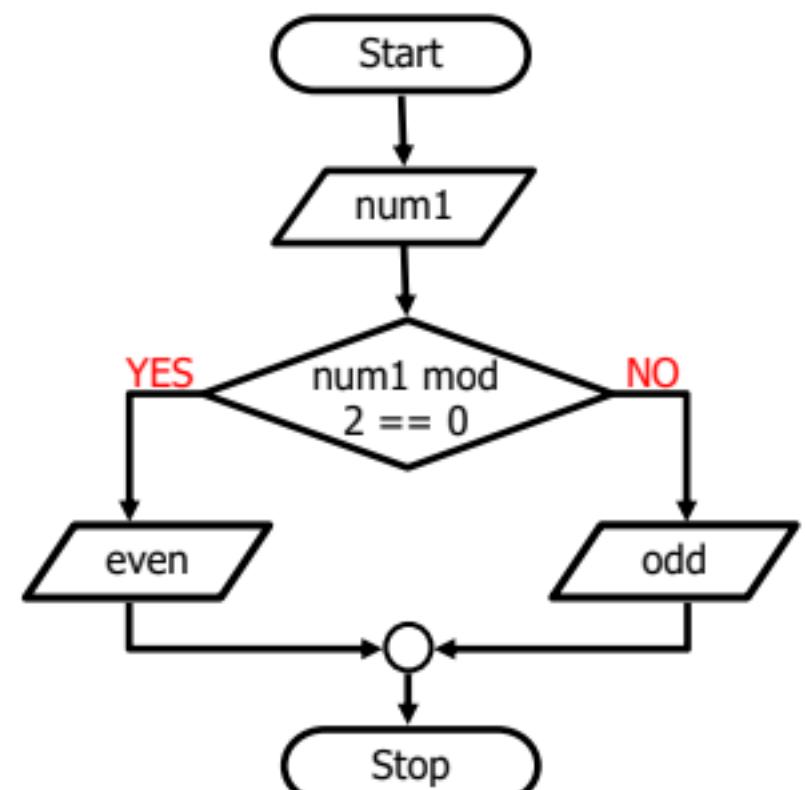
1. รูปแบบที่มีการกำหนดเงื่อนไขหรือให้เลือก (**Decision Structure**) : รูปแบบที่มีการสร้างเงื่อนไข สำหรับเลือกทำงาน โดยจะทำงานตามที่กำหนดหากตรงตามเงื่อนไข และถ้าหากไม่ตรงเงื่อนไขจะทำงานที่กำหนดตั้งไว้อีกอย่างหนึ่ง

```

1 #include<stdio.h>
2
3 main()
4 {
5     int num1;
6     scanf("%d", &num1);
7     if(num1 % 2 == 0)    printf("even");
8     else                  printf("odd");
9 }
```

ภาพตัวอย่างการเขียนโค้ด

ภาพตัวอย่างผลการรันโค้ด



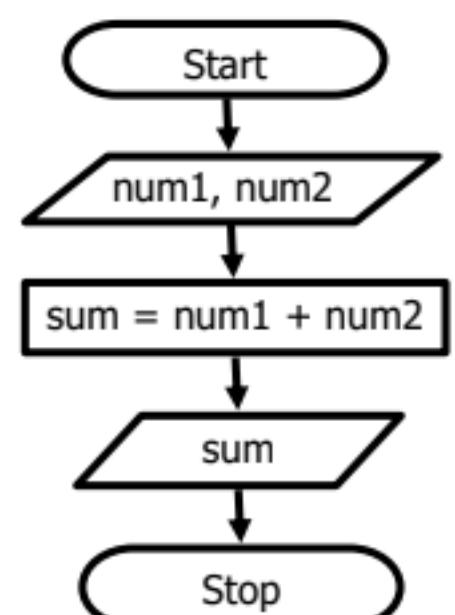
2. รูปแบบเรียงลำดับ (**Sequence Structure**) : การทำงานแบบเรียงลำดับตั้งแต่เดือนจนจบ เป็นรูปแบบง่าย ๆ ไม่มีการเปลี่ยนเที่ยบใด ๆ มีทิศทางเพียงทิศทางเดียว

```

1 #include<stdio.h>
2
3 main()
4 {
5     int num1, num2, sum;
6     scanf("%d%d", &num1, &num2);
7     sum = num1 + num2;
8     printf("%d", sum);
9 }
```

ภาพตัวอย่างการเขียนโค้ด

ภาพตัวอย่างผลการรันโค้ด



### 3. รูปแบบที่มีการทำงานแบบวนรอบ หรือ loop (Iteration Structure) : รูปแบบที่มีการทำงานซ้ำค่าสั่งเดิม โดยการทำงานนี้จะขึ้นอยู่กับเงื่อนไขที่กำหนดให้

```

1 #include<stdio.h>
2
3 main()
4 {
5     int num;
6     for(int i = 0; i < 10; i++){
7         printf("Hello World\n");
8     }
9 }
```

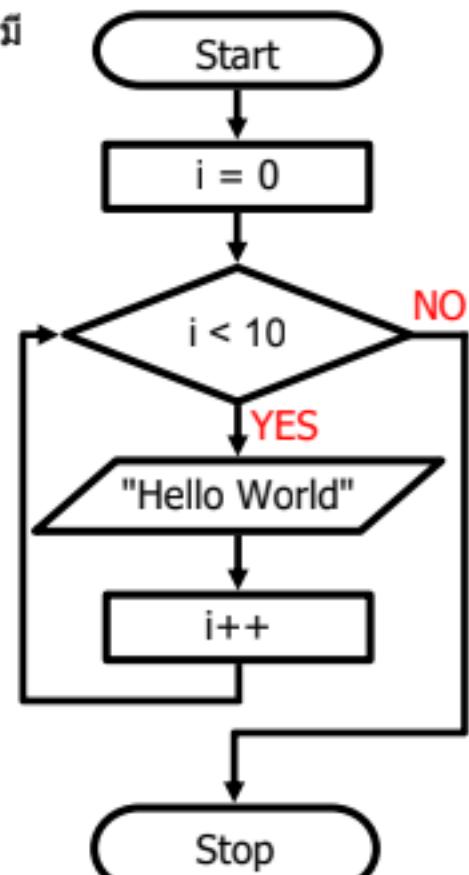
ภาพตัวอย่างการเขียนโค้ด

```

Hello World
Process exited after 0.01836
seconds with return value 0
Press any key to continue . . .

```

ภาพตัวอย่างผลการรันโค้ด



**ข้อสังเกต :** เส้นทุกเส้นต้อง “เชื่อมต่อ กัน และ มี ทิศทาง เสมอ” ดังนั้นทุกเส้นจะมีลูกศร และทุกเส้นจะต้องเชื่อมติดกันล่อง สัญลักษณ์

**ข้อสังเกต :** ลูกศรควรจะชี้ไปยังกล่องได้เพียง “เส้นเดียว” ดังนั้นหากมีมากกว่า 1 เส้นอาจใช้ IN-Page Connector ช่วย

**ข้อสังเกต :** ในสัญลักษณ์การตัดสินใจ (Decision) “สามารถใส่ได้แค่เงื่อนไขได้เท่านั้น” และเส้นที่ลากออกต้องมีเพียง แค่ 2 เส้นเท่าตามเงื่อนไขซึ่งมี 2 รูปแบบ คือ “Yes/True หรือ No/False” เท่านั้นหากเป็นเงื่อนไข if-else

## รหัสเทียม (Pseudo Code)

คือภาษาที่ใช้จำลองภาษาโปรแกรม โดยใช้ภาษาที่ไม่ใช่ภาษาโปรแกรมซึ่งไม่จำกัดภาษา และมีขั้นตอนที่แน่นอนกะทัดรัด เป็นเพียงการจำลองค่าสั่งจริงแบบย่อ ๆ ตามกระบวนการของโปรแกรมที่ต้องการสร้าง ซึ่งจะช่วยให้เห็นโครงสร้างโปรแกรมชัดเจนขึ้น สามารถหาจุดผิดของโปรแกรมได้ง่ายเมื่อมีปัญหา และเป็นตัวกำหนดงานเขียนโปรแกรมให้ทำงานได้ตามต้องการเมื่อมีการเขียนตามโค้ดเทียนที่สร้างไว้ หลักการเขียน Pseudo code ที่ต้องให้ในหนึ่งบรรทัด ควรมีค่าสั่งเดียว และย่อหน้าเพื่อแยกย่อยค่าสั่งให้ชัดเจน

### ประเภทรหัสเทียม (Pseudo Code Type)

#### Draft Code

เป็นการเขียนอย่างหยาบๆ ที่สุด ไม่ลงรายละเอียดอะไรมาก บอกว่ากำลังทำอะไรเป็นลำดับไป

#### Detailed Code

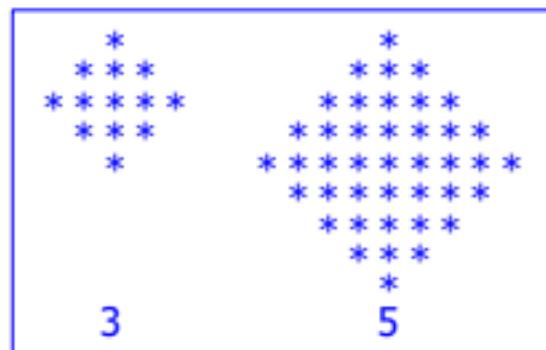
เป็นส่วนที่จะนำ draft code มาแตกรายละเอียดให้ชัดเจนว่ามีการทำงานตรงไหนอย่างไร โดย Detailed code จะทำกีรอนกีไว้จนกว่าจะได้การทำงานที่ชัดเจนที่สุดเพื่อแปลงเป็น simple code ต่อไป

#### Simple Code

เป็นส่วนที่มีความใกล้เคียงภาษาโปรแกรมมากที่สุด และมีรายละเอียดครบถ้วนที่สุด พร้อมที่จะนำไปแปลงไปเป็นภาษาโปรแกรมได้ทันที

## ตัวอย่าง

### ตัวอย่าง 1 Cr. พีดัน ไม้ม #CE57

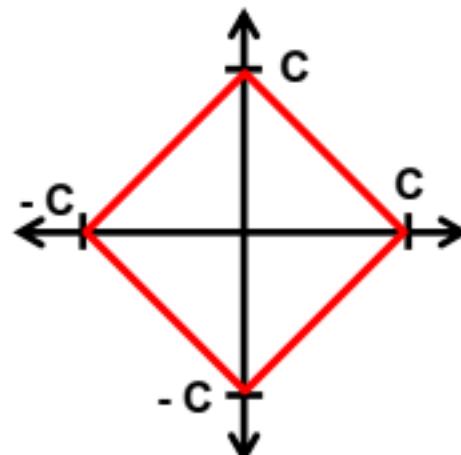


#### Draft Pseudo Code

- รับค่าตัวแปร N เพื่อสร้างรูป
- สร้างรูป \* จากพื้นที่ภายในกราฟ  $(x, y)$  โดยมีจุดยอดด้าน x,  $y = N - 1 = C$
- กำหนดเงื่อนไขที่จะแสดง \* และแสดงช่องว่าง

#### Detailed Pseudo Code

- รับค่าตัวแปร N
- กำหนดจุดยอดแกน  $x, y = N - 1 = C$



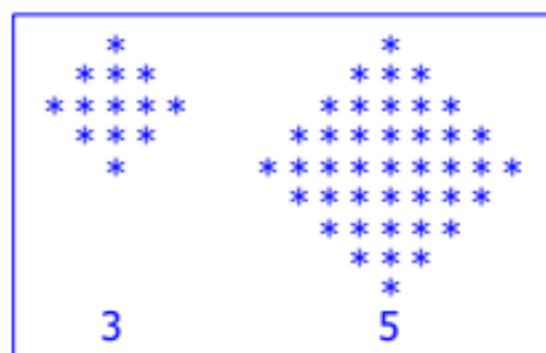
เงื่อนไขพื้นที่ภายในกราฟ คือ  
 $|y| \leq |x| - C$  แสดง \*

$|y| > |x| - C$  แสดง ช่องว่าง

#### Simple Pseudo Code

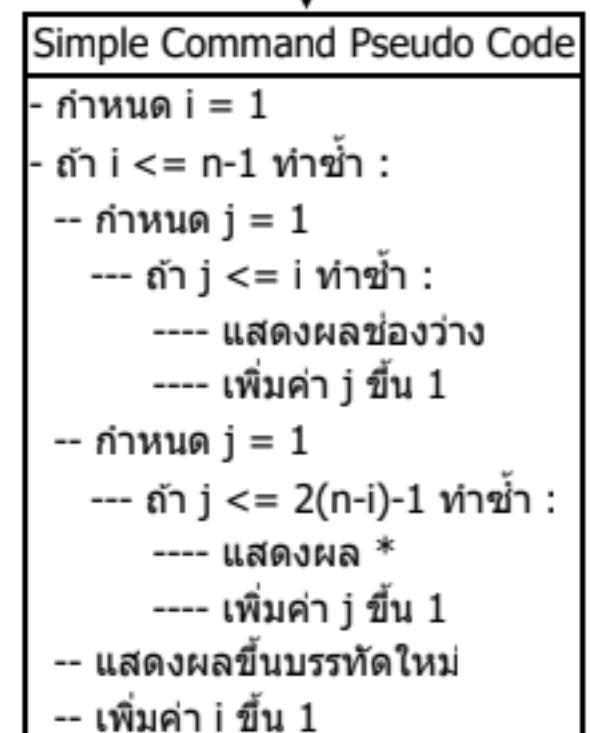
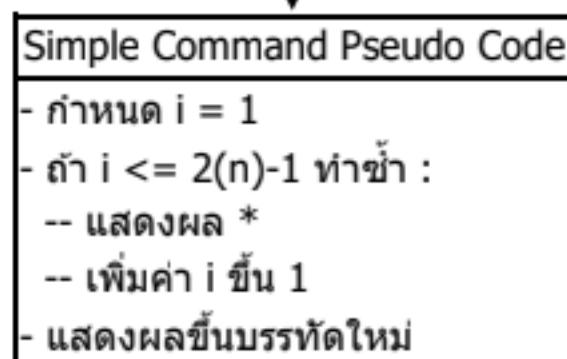
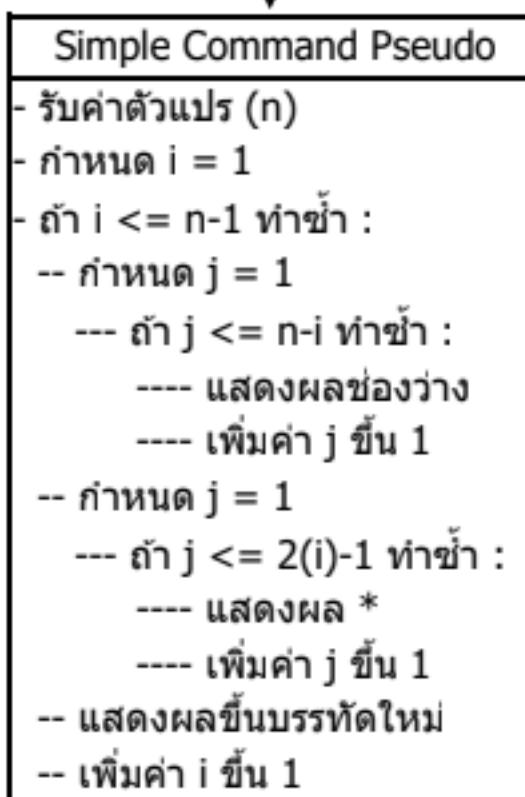
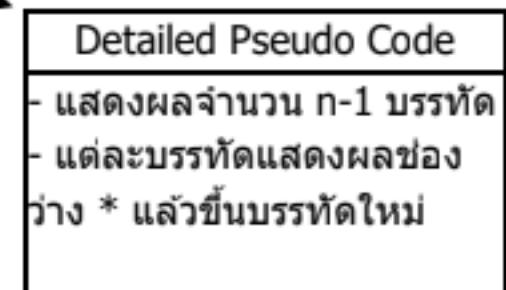
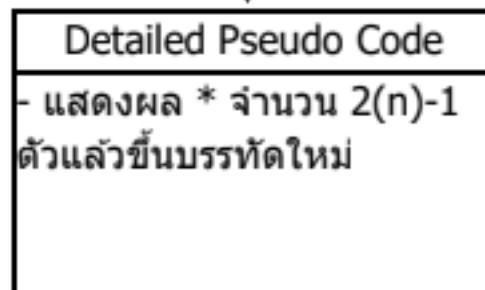
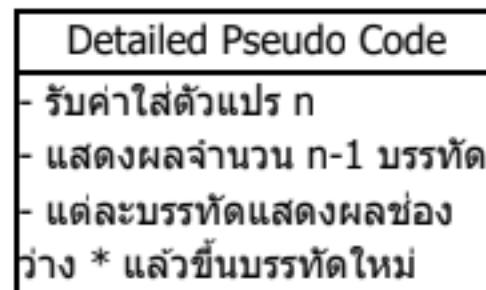
- รับค่าเก็บไว้ในตัวแปร N
- $N = N - 1$
- $y = N$
- ทำเงื่อนไขต่อไปนี้ข้ามเมื่อ  $y \geq -N$ 
  - $x = -N$
  - ทำเงื่อนไขต่อไปนี้ข้ามเมื่อ  $x \leq N$ 
    - เมื่อ  $|y| \leq |x| - N$  แสดง \*
    - นอกเหนือจากเงื่อนไข ก่อนหน้าแสดงช่องว่าง
    - $x = x + 1$
- $y = y - 1$

### ตัวอย่าง 2 Cr. พีกเณ #CE57



#### Draft Pseudo Code

- รับค่าใส่ตัวแปร n
- แสดงผล Δ บน
- แสดงผลบนรหักล่าง
- แสดงผล Δ ล่าง

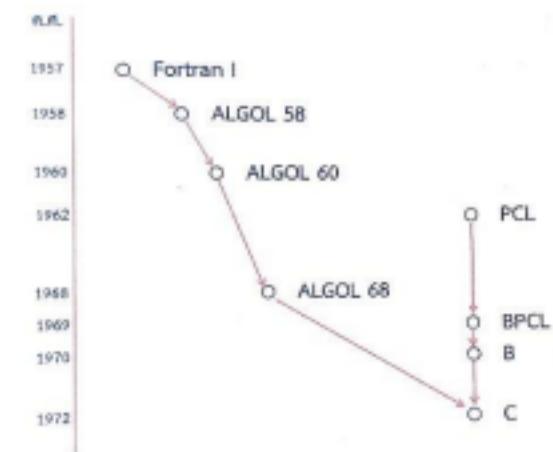


## ประวัติความเป็นมา

ภาษา C เป็นภาษาคอมพิวเตอร์ที่ได้พัฒนาขึ้นในปี ค.ศ. 1972 โดยเดนนิส ริชชี (Denis Ritchie) และ Bell Telephone Laboratories, Inc. (AT & T Bell Laboratories, 1984-1996 ในปัจจุบันคือ Nokia Bell Labs) ซึ่งภาษา C มีการพัฒนามาจากภาษา B ในช่วงแรกภาษา C ได้ถูกนำมาใช้เพื่อสร้างระบบปฏิบัติการ Unix หากน่าวินาการของภาษา C มาแสดงออกเป็นแผนภาพได้ดังนี้

ในปี 1978 เดนนิส ริชชี และนายเบรน เครนิกชาน (Denis Ritchie and Brian W. Kernighan) ได้แต่งหนังสือชื่อ "The C Programming Language" โดยนำเสนอกฎภาษา C ที่สามารถนำไปปรับใช้กับคอมพิวเตอร์ในรูปแบบต่าง ๆ ได้มากยิ่งขึ้น และทำให้ภาษา C ได้รับความนิยมอย่างมาก จนกระทั่งในปี ค.ศ. 1988 ได้มีการสร้างมาตรฐานของภาษา C ขึ้นมาในชื่อของ ANSI C ภายใต้ความร่วมมือระหว่างสถาบัน ANSI (American National Standards Institute) กับนายเดนนิส ริชชี และนายเบรน เครนิกชาน

ในปี 1990 องค์กรมาตรฐานสากล หรือ ISO (International Standards Organization) ได้ยอมรับมาตรฐานที่ได้สร้างขึ้นมาไว้ ภายใต้ชื่อ ANSI/ISO C



## จุดเด่นของภาษา C

- เป็นภาษามาตรฐาน การทำงานไม่ขึ้นกับฮาร์ดแวร์ ทำให้สามารถนำไปใช้ใน CPU รุ่นต่าง ๆ ได้
- เป็นภาษาระดับสูงที่ทำงานเหมือนภาษาระดับต่ำ สามารถทำงานแทนภาษา Assembly ได้
- มีแนวความคิดในการพัฒนาแบบ "โปรแกรมเชิงโครงสร้าง" (Structure Programming) จึงทำให้ภาษา C เป็นภาษาที่เหมาะสมสำหรับนักพัฒนาระบบ
- ความสามารถของคอมไพล์เตอร์ (Compiler) ในภาษา C มีประสิทธิภาพสูง ทำงานได้รวดเร็ว โดยใช้รหัสออบเจกต์ที่ลับ (Object) ทำให้เหมาะสมสำหรับการทำงานที่รวดเร็ว

**ข้อควรรู้ :** คอมไпал์เตอร์ (Compiler) จะทำหน้าที่แปลโค้ดที่เขียนทั้งหมดให้เป็นภาษาเครื่อง พร้อมกันนี้จะทำหน้าที่ตรวจสอบไวยากรณ์ของภาษา หากมีข้อผิดพลาดก็จะแจ้งให้เห็นถึงข้อผิดพลาด เมื่อกระบวนการทำงานของคอมไпал์เตอร์เสร็จสมบูรณ์ โปรแกรมจะสามารถใช้งานได้ ซึ่งจะแตกต่างกับหลักการทำงานของอินเทอร์เพเตอร์ (Interpreter) ซึ่งการทำงานจะแปลงภาษาโปรแกรม และประมาณผลค่าสั่งทีละค่าสั่ง

## ข้อแตกต่างระหว่าง ภาษา C vs. C++

เดิมโปรแกรมเมอร์ภาษา C จะสร้างโปรแกรมโดยคิดตาม step 1, 2, 3, ... ซึ่งมีข้อดีสำหรับโปรแกรมที่ไม่ซับซ้อนมากสามารถเขียนได้อย่างรวดเร็ว แต่นาน ๆ ไปโปรแกรมเริ่มจะมีความสามารถเพิ่มขึ้น สลับซับซ้อนกว่าเดิมซึ่งในระหว่างพัฒนา หรือแก้ไขปัญหาอาจสับสนได้ว่าเราจะต้องไปแก้ไขตรงจุดไหน จึงอาจทำให้เกิด Bug ได้ง่าย

จึงมีแนวคิดการเขียนโปรแกรมแบบ OOP ขึ้นมาคือให้แบ่งโปรแกรมออกเป็นชิ้น ๆ หรือเรียกว่าวัตถุ (Object) และพัฒนาแต่ละชิ้นมา จากนั้นจึงนำมาประกอบกันทีหลัง วิธีการนี้ทำให้ลดความสับสนเวลาพัฒนา หรือแก้ไขโปรแกรมลงมาได้มาก เพราะสามารถแก้ไขในส่วนเล็ก ๆ ขึ้นเดียวได้ ไม่ต้องแก้ไขทั้งโปรแกรม และนอกจากนี้เรายังสามารถนิวัตถุ (หรือส่วนที่แบ่งเป็นชิ้น ๆ) นำกลับมาใช้ใหม่ หรือนำไปใช้กับโปรแกรมอื่น ๆ ที่เราต้องการโดยไม่ต้องเสียเวลาเขียนใหม่ ประหยัดเวลาได้อีกมาก ถ้าจะเปรียบเหมือนกับการสร้างรถยนต์ แทนที่จะหล่อรถยนต์ทั้งคันทีเดียว (ภาษา C) ถ้าเสียต้องทำใหม่หมด หรือส่งไปซ่อมทั้งคัน แต่ถ้าใช้แนวคิดแบบ OOP แต่ละชิ้นทำขึ้นมาต่างหาก และจึงนำมาประกอบกันทีหลัง ถ้ามีปัญหาสามารถหาสาเหตุ และนำขึ้นส่วนใหม่มาเปลี่ยนได้ง่ายกว่า

**C :** สามารถแก้ไขโปรแกรมได้ง่ายกว่า และสามารถทำงานได้ไวกว่า C++ เนื่องจากโครงสร้างของไวยากรณ์ที่มีความซับซ้อนน้อยกว่า จึงเหมาะสมกับการเขียนไดร์เวอร์ควบคุมฮาร์ดแวร์, โปรแกรมที่ต้องการความเสถียรสูง, การเขียนโปรแกรมระดับ Kernel, ควบคุม Microcontroller

**C++ :** ถูกพัฒนาต่อยอดมาจาก C เพื่อทำให้การพัฒนาโปรแกรมใหญ่ ๆ และทำความเข้าใจเพื่อแก้ไขโปรแกรมได้สะดวกกว่าเดิมโดยเพิ่มในส่วนของ OOP (Object-Oriented Programming) เข้าไป C++ ยังสามารถเขียนโปรแกรมแบบ C ได้ จึงเหมาะสมกับการเขียนโปรแกรมใหญ่ ๆ ที่ต้องจัดการความซับซ้อนของโปรแกรม และข้อมูลที่มีจำนวนมากโดยที่โปรแกรมยังทำงานได้ความเร็ว

### ทำไมต้อง C ก่อน!

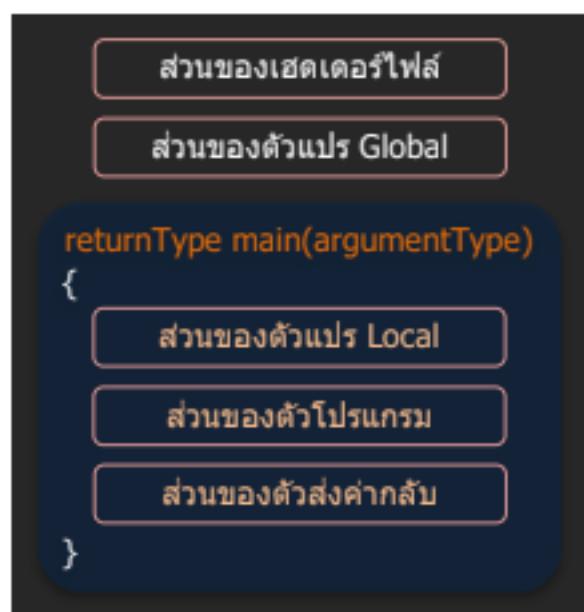
ถ้าเข้าใจโครงสร้างการเขียน C หากไปเขียนภาษาอื่น จะเข้าใจ หรือ Optimize ได้ง่ายขึ้น เพราะ C ไม่ค่อยมีตัวช่วย หรือ Library มากเท่าภาษาอื่นที่ง่ายกว่า เช่นภาษา Python ภาษา C จึงเหมาะสมสำหรับฝึก Logic การทำงานตั้งแต่ต้นจนจบ ... เขียนโปรแกรมเป็นโน้ตแล้วก็เขียนโปรแกรมได้ แนวคิดต้องได้ด้วย! ...

## โครงสร้างโปรแกรม

```

1 #include<stdio.h> #1
2
3 void function() #2
4 {
5     ← #3
6 }
7
8 main() #2
9 {
10}
11

```



### เศตเดอร์ไฟล์ (Header File)

เป็นส่วนที่เรียกใช้ไลบรารี ซึ่งจะขึ้นต้นด้วยเครื่องหมาย # เสมอ โดยเป็นการนำโคดในไฟล์เศตเดอร์ (.h) นั้น ๆ มาใส่ไว้ที่หัวของโปรแกรมที่เรากำลังเขียน คอมไพล์เลอร์จะทำในส่วนนี้เป็นส่วนแรก โดยสามารถเขียนได้ 2 แบบ คือ

```

||| #include<HeaderName>
||| #include "HeaderName"

```

ถ้าใช้เครื่องหมาย "... " คอมไпал์เลอร์จะหาไฟล์จากโฟลเดอร์ที่เก็บ Source Code ก่อน หลังจากนั้นจึงหาไฟล์ที่เก็บไว้กับตัวคอมไпал์เลอร์ แต่ถ้าใช้ <...> จะหาจากไฟล์ที่เก็บไว้กับตัวคอมไпал์เลอร์เพียงที่เดียวเท่านั้น

**ข้อควรระวัง :** " " ในใช้ตัวเดียวกันกับ " " (Double quote)

**ข้อสังเกต :** ไลบรารี คือส่วนที่เก็บค่าสั่งต่าง ๆ สำหรับการใช้งาน หากไม่ได้เรียกไลบรารีมาใช้อาจใช้บังคับสั่งไม่ได้ เช่น ถ้าต้องการได้ค่าของ 2 ยกกำลัง 3 จะใช้ค่าสั่งว่า pow(2, 3); หากไม่ได้ #include<math.h> ไว้ในส่วน Header จะทำให้คอมไпал์เลอร์ไม่รู้จักค่าสั่ง pow และขึ้น error เดือนอกมา

### ฟังก์ชัน (Function)

```

1 void function()
2 {
3
4 }
5
6 main() = int main()
7 {
8     return 0;
9 }
10
11
12 Process exited after 0.2029 seconds with return value 0
13 Press any key to continue . . .

```

เป็นส่วนการทำงานของโปรแกรม ซึ่งบังคับให้มืออย่างน้อย 1 ฟังก์ชัน คือ main() ซึ่งเป็นฟังก์ชันเริ่มการทำงานของโปรแกรม (ประมวลผลที่ฟังก์ชันนี้เป็นฟังก์ชันแรก) ขอบเขตของฟังก์ชันจะอยู่ภายในบล็อกซึ่งจะเริ่มต้นด้วยเครื่องหมาย { และสิ้นสุดด้วย } ฟังก์ชันส่วนมากจะมีส่วนส่งค่ากลับเมื่อทำงานจนฟังก์ชัน จะเห็นว่าเมื่อฟังก์ชันจะส่งค่ากลับจะต้องมีค่าสั่ง return

การประกาศไว้หน้าชื่อฟังก์ชันว่าจะส่งค่าประเภทไหนกลับ และใช้ค่าสั่ง return ใน การส่งค่ากลับมา เช่น void (ไม่ส่งค่ากลับ ทำให้ในฟังก์ชันไม่ต้องมีค่าสั่ง return) , int (ส่งค่าจำนวนเต็มกลับมา) , float (ส่งค่าทศนิยมกลับมา) , char (ส่งค่าตัวอักษรตัวเดียวกลับมา) เป็นต้น

**ข้อสังเกต :** ฟังก์ชัน main ถ้าไม่ได้ระบุค่าสั่งกลับจะมีค่า Default เป็นจำนวนเต็มที่มีค่าเท่ากับ 0 ถ้าโปรแกรมทำงานไม่เกิดข้อผิดพลาด

### ส่วนตัวโปรแกรม (Body)

จะอยู่ภายใต้ฟังก์ชันซึ่งประกอบด้วย 2 ส่วนคือ

- 3.1 ค่าสั่ง คือ ค่าที่กำหนดไว้ให้โปรแกรมทำงานตามที่ต้องการ ส่วนมากจะจบด้วยเครื่องหมาย ;
- 3.2 ตัวแปร คือ สัญลักษณ์ไข้แทนค่าต่าง ๆ ซึ่งค่าที่แทนสามารถเปลี่ยนเป็นค่าอื่นได้ โดยค่าที่นั้นจะเป็นตัวเลข ตัวหนังสือ หรือ object ก็ได้

1 main ()	=	main(){}	=	main(){}
2 {		}		

6 if(true){	=	if(true) {statement;}
7 statement;	=	
8 }		

**ข้อสังเกต :** โปรแกรมจะทำงานจากช้ายไปขวา จากนั้นลงล่าง ดังนั้นการเว้นวรรค หรือขีนบรรทัดใหม่ไม่มีผล

\* Statement โดยทั่วไปจะหมายถึง คำสั่งส่วนมากจะปิดท้ายด้วยเครื่องหมาย ; เช่น

**ข้อควรรู้ :** สามารถใช้คอมเม้นต์เพื่อทำให้คำสั่งบางคำสั่งที่เราไม่ต้องการกลับเป็นค่าอธิบาย หรือเพื่อไม่ให้ส่วนนั้นทำงาน สามารถทำได้ 2 วิธีคือ

1. ใช้ // เพื่อทำการคอมเม้นต์ข้อความด้านหลังสัญลักษณ์นี้ ครอบคลุมภายในบรรทัดนั้น
2. ใช้ /\* ... \*/ เพื่อทำการคอมเม้นต์ข้อความที่อยู่ด้านใน โดยสามารถใช้ได้หลายบรรทัด

**ข้อควรระวัง :** หากใช้ /\* ... \*/ ถ้าเจอกการปิดคอมเม้นต์ตัวแรกเมื่อไรข้อความหลังจากนั้นจะไม่ถูกคอมเม้นต์ เช่น /\*/\*/\* COMMENT \*/\*/ ตัว \*/ สืบเนื่องจะไม่ถูกคอมเม้นต์ และคอมไไฟล์จะเข้าใจว่าเป็นคำสั่ง ทำให้โปรแกรมเราเกิด Syntax Error ทำให้คอมไไฟล์ไม่ผ่านได้

```
1 #include<math.h>
2
3 main()
4 {
5     int x;
6     //statement1;
7     x = pow(2, 3);    //x=8
8     /*statement3;
9     statement4;
10    statement5;*/
11 }
```

## ตัวแปร (Variable)

คือ สัญลักษณ์ที่ใช้แทนตัวแหน่งบนหน่วยความจำ ใช้ในการเก็บค่าต่าง ๆ ในโปรแกรม ที่ต้องใช้สัญลักษณ์แทนเนื่องจากการอ้างถึงตัวแหน่งบนหน่วยความจำนั้นยากต่อการทำความเข้าใจ และการพัฒนาโปรแกรม ซึ่งผู้อ่านสามารถเรียกใช้ตัวแปรเพื่อใช้ในการค่าวน หรือเพื่อเก็บค่าได้ โดยกฎการตั้งชื่อให้กับ Identifier (ชื่อที่มีอยู่ในส่วนต่าง ๆ ของโครงสร้างโปรแกรมภาษา C) ซึ่งได้แก่ ตัวแปร พึ่งกชัน และเลเบลมีดังนี้

### การตั้งชื่อไอเดียนติฟายเออร์ (Identifier Names)

1. ชื่อประกอบด้วยตัวอักษรพิเศษไม่ได้ เช่น & # & เป็นต้น
2. ภายในชื่อประกอบด้วยช่องว่าง หรือแท็บไม่ได้
3. ในภาษา C / C++ เป็นแบบ Case-Sensitive คือตัวอักษรตัวใหญ่และตัวอักษรตัวเล็กถือเป็นคนละตัวกัน เช่น pom, Pom, PoM, POM จะถือว่าชื่อทั้งสี่ตัวนี้เป็นคนละตัวแปรกัน และคอมไไฟล์จะมองว่าตัวไม่สามารถตั้งชื่อตัวแปรได้เกิน 31 ตัวอักษร
4. สามารถประกอบด้วยตัวอักษร ตัวเลข และเครื่องหมาย Underscore (\_) แต่ตัวอักษรตัวแรกไม่สามารถเป็นตัวเลขได้ โดยในชื่อสามารถมีตัวเลขประกอบได้ เช่น \_Number, x1, \_708, pRayut44 เป็นต้น
5. ตั้งชื่อตัวแปรข้ากันไม่ได้ ไม่เช่นนั้นคอมไไฟล์จะไม่รู้ว่าตัวแปรไหนเป็นตัวไหน เช่นเดียวกันกับคำสั่งวน
6. ชื่อที่ตั้งขึ้นจะต้องไม่ซ้ำกับคำสั่ง (Reserved Word) ภายในโปรแกรม ซึ่งมีดังนี้

**การตั้งชื่อตัวแปร และพึ่งกชันให้เป็นสากล :** จะใช้หลัก Lower Camel Case ตัวแรกตัวเล็ก ถ้ามีค่าต่อ กันจะเป็นต้นตัวใหญ่ เช่น sphereRadius, getSphereRadius(), radius เป็นต้น

!!! แต่ตั้งเดิมภาษา C จะใช้ Snake Case เช่น sphere\_radius, get\_sphere\_radius() เป็นต้น

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

### การประกาศตัวแปร (Variable Declaration)

ก่อนที่จะตั้งชื่อให้ตัวแปร จะต้องทำการกำหนดชนิดของตัวแปร เพื่อให้โปรแกรมได้สำรองพื้นที่ในหน่วยความจำเพื่อเก็บข้อมูลตามที่เราต้องการ โดยสามารถทำได้ดังนี้

||| dataType varName;

โดยที่ **dataType** เป็นชนิดของข้อมูล เช่น **int x;**  
**varName** เป็นชื่อตัวแปร **char alphabet;**

ชื่อตัวแปร	หน่วยความจำ	ชื่อตัวแปร	หน่วยความจำ
num	5	num	5
ch	M	ch	M
str	Jade SPK	str	Jade SPK

**ข้อควรรู้ :** สามารถกำหนดค่าเริ่มต้นเข้าไปด้วยแต่ตอนประกาศตัวแปรได้เลย เช่น

```
int x = 8;
float y = 0.2;
char alphabet = 'J';
char name[] = "pravit";
```

เมื่อโปรแกรมทำงานถึงตรงที่ประกาศตัวแปรไว้ โปรแกรมจะทำการจ้องพื้นที่ในหน่วยความจำ ถ้ามีการกำหนดค่าให้ตัวแปร ค่าเหล่านั้นก็จะถูกเก็บลงในหน่วยความจำตามตำแหน่งที่ได้จ้องไว้ ดังรูปทางด้านข่าย

**ข้อสังเกต :** ตัวแปร ch ค่าที่เก็บลงหน่วยความจำ (กล่อง) จะง ๆ และไม่ใช่ M แต่เป็นตัวเลขรหัสแซลก์ของ M ซึ่งจะได้ศึกษาในส่วนต่อ ๆ ไป

## ชนิดของข้อมูล (Data Type)

ข้อมูลแต่ละชนิดมีขอบเขตความสามารถในการเก็บข้อมูลต่างกัน บางชนิดสามารถเก็บข้อมูลได้มากก็จะใช้หน่วยความจำมากตามไปด้วย จึงต้องคำนึงถึงความเหมาะสมในการใช้ตัวแปรชนิดต่าง ๆ และหากเก็บค่าของข้อมูลที่มีค่านากกว่าขอบเขตที่ชนิดของข้อมูลนั้นสามารถใช้เก็บค่าได้ หรือเก็บค่าในชนิดของข้อมูลที่ผิดประเภท จะสามารถทำให้โปรแกรมเกิดข้อผิดพลาดได้ โดยชนิดของข้อมูลในโปรแกรมจะมี 4 ประเภทหลัก ๆ คือจำนวนเต็ม (Integer Type), จำนวนจริง (Real floating-point Type), ตัวอักษร (Character Type) และสายอักษร (String Type)

### ชนิดข้อมูลแบบจำนวนเต็ม (Integer Type)

เป็นชนิดข้อมูลแบบจำนวนเต็ม ประกอบไปด้วยจำนวนเต็มบวก (1, 2, 3, 4, ...), จำนวนเต็มลบ (-1, -2, -3, -4, ...) และจำนวนเต็มศูนย์ (0) โดยมีรายละเอียดตามตารางด้านล่างดังนี้

ชนิดข้อมูล	ขนาด	ขอบเขต	อธิบายเพิ่มเติม (ชนิดข้อมูลที่เก็บ)
(signed) short	16 บิต	- $2^{15}+1$ ถึง $2^{15}-1$	จำนวนเต็มแบบสั้น คิดเครื่องหมาย
unsigned short	16 บิต	0 ถึง $2^{16}-1$	จำนวนเต็มแบบสั้น ไม่คิดเครื่องหมาย
(signed) int	16 บิต 32 บิต	- $2^{15}+1$ ถึง $2^{15}-1$ $-2^{31}+1$ ถึง $2^{31}-1$	จำนวนเต็ม คิดเครื่องหมาย
unsigned int	16 บิต 32 บิต	0 ถึง $2^{16}-1$ 0 ถึง $2^{32}-1$	จำนวนเต็ม ไม่คิดเครื่องหมาย
(signed) long	32 บิต	- $2^{31}+1$ ถึง $2^{31}-1$	จำนวนเต็มแบบยาว คิดเครื่องหมาย
unsigned long	32 บิต	0 ถึง $2^{32}-1$	จำนวนเต็มแบบยาว ไม่คิดเครื่องหมาย
(signed) long long	64 บิต	- $2^{63}+1$ ถึง $2^{63}-1$	จำนวนเต็มแบบยาวมาก คิดเครื่องหมาย
unsigned long long	64 บิต	0 ถึง $2^{64}-1$	จำนวนเต็มแบบยาวมาก ไม่คิดเครื่องหมาย

**ข้อสังเกต :** ถ้าเป็นตัวแปรแบบคิดเครื่องหมายสามารถจะ (signed) ตอนประกาศตัวแปรได้

**ข้อสังเกต :** ข้อมูลชนิดเดียวกันในภาษา C/C++ อาจจะมีความแตกต่างกันในเรื่องขนาด และขอบเขตได้ ซึ่งขึ้นอยู่กับระบบปฏิบัติการ และ Compiler ที่ใช้งาน เช่น ระบบปฏิบัติการ 8, 16 บิต ข้อมูลชนิด int จะมีขนาด 16 บิต (2 ไบต์) ระบบปฏิบัติการ 32, 64 บิต ข้อมูลชนิด int จะมีขนาด 32 บิต (4 ไบต์)

### การกำหนดค่าให้กับตัวแปรชนิดจำนวนเต็ม

- ต้องเป็นตัวเลขไม่มีจุดหรือเส้น หากกำหนดเป็นทศนิยมข้อมูลอาจมีการสูญหาย
- ห้ามใช้เครื่องหมาย , หรือช่องว่างคั่นระหว่างตัวเลข เช่น 1 000 000 หรือ 1,234 หรือ 12, 3 เป็นต้น
- ถ้าเป็นค่าบวกไม่จำเป็นต้องใส่เครื่องหมาย + นำหน้าค่า แต่ถ้าเป็นค่าลบต้องใส่เครื่องหมาย - นำค่าเสนออก
- ช่วงตัวเลขจำนวนเต็มควรอยู่ในช่วงชนิดข้อมูลนั้น ๆ
- สามารถใช้เครื่องหมาย Suffix ต่อท้ายค่าที่กำหนดให้ตัวแปรได้ โดยใช้ L ต่อท้ายชนิดข้อมูล long หรือใช้ U ต่อท้ายค่าที่เป็น unsigned (ใช้ตัวพิมพ์ใหญ่ หรือเล็กความหมายเหมือนกัน) เช่น

```
testInt = +1234; // int ใส่เครื่องหมาย + นำหน้าค่ายังทำงานปกติ ไม่ได้ผิดพลาด
testInt = 1234; /* int */
testLongInt = 123456789L; // long int
testUnsignedInt = 123456U; // unsigned int
testUnsignedLongInt = 123456789UL; // unsigned long int
```

## ชนิดข้อมูลแบบจำนวนจริง (Real Floating Point Type)

เป็นชนิดข้อมูลแบบตัวเลขทศนิยมที่สามารถถ่ายไปค่าจำนวนทางคณิตศาสตร์ได้ อาจมีจุดทศนิยม หรือไม่มีจุดทศนิยมก็ได้ โดยสามารถเขียนในรูปแบบเลขทศนิยม เช่น 1112.50, -250.0 หรอรูปแบบเลขทศนิยนยกกำลัง เช่น 3.456E+3 (มีค่า  $3.456 \times 10^3$  หรือ 3,456.0), 7.89E+4 (มีค่าเป็น  $7.89 \times 10^4$  หรือ 78,900.0) ความแม่นยำของทศนิยมในจำนวนจริง float 6 หลัก, double 17 หลัก และ long double 34 หลัก ซึ่งไม่สามารถใช้ได้ตามอธิบายในข้อควรรู้นี้[\*]

ชนิดข้อมูล	ขนาด	ขอบเขต	อธิบายเพิ่มเติม (ชนิดข้อมูลที่เก็บ)
float	32 บิต	$-3.4 \times 10^{-38}$ ถึง $3.4 \times 10^{38}$	เลขทศนิยม
double	64 บิต	$-1.79 \times 10^{-308}$ ถึง $1.79 \times 10^{308}$	เลขทศนิยม
long double	128 บิต	$-1.189 \times 10^{-4932}$ ถึง $1.189 \times 10^{4932}$	เลขทศนิยม

### การกำหนดค่าให้กับตัวแปรชนิดจำนวนจริง

จุดสังเกตจะพบว่า ชนิดข้อมูลแบบจำนวนจริงจะเป็นแบบคิดเครื่องหมาย (signed) เช่น โดยผู้อ่านจะสามารถกำหนดค่าตัวแปรโดยค่านึงถึงข้อกำหนดดังนี้

1. ควรจะเป็นค่าตัวเลขที่สามารถมีจุดทศนิยมได้ ถ้าหากกำหนดค่าเป็นจำนวนเดิมโปรแกรมจะทำการแปลงจากจำนวนเดิม (int) เป็นจำนวนจริงประเภท float ให้อัตโนมัติ (Implicit Conversion) เช่น testF = 15 ในตัวแปร testF จะเก็บเป็น 15.000000

2. ห้ามใช้เครื่องหมาย , หรือช่องว่างคั่นระหว่างตัวเลข เช่น 1 000 000.0 หรือ 1,234.03 หรือ 12, 3.5 เป็นต้น

3. กรณีเป็นค่าบวกไม่จำเป็นต้องใส่เครื่องหมาย + หน้าค่า แต่กรณีเป็นค่าลบต้องใส่เครื่องหมาย - หน้าค่า เช่น

4. การเขียนในรูปแบบใช้ตัวอักษร E ค่าที่ถูกกำหนดสามารถกำหนดได้ทั้งค่าวิกและค่าลบ

5. สามารถใช้เครื่องหมาย Suffix ต่อท้ายค่าที่กำหนดให้ตัวแปรได้ โดยใช้ ใช้ D ต่อท้ายค่าที่เป็น double หรือ L ต่อท้ายชนิดข้อมูล long double หรือใช้ F ต่อท้ายค่าที่เป็น float (ใช้ตัวพิมพ์ใหญ่ หรือพิมพ์เล็กความหมายเหมือนกัน) เช่น testFloat = 654.6F; // float

testDouble = 13.31D; // double

testLongDouble = 16.34L // long double

## ชนิดข้อมูลแบบตัวอักษร (Character Type)

char (Character) เป็นชนิดข้อมูลแบบอักขระตัวเดียวซึ่งมีขนาด 1 ไบต์ หรือ 8 บิต โดยอักขระในที่นี้เป็นได้ทั้งตัวอักษร (Alphabet), ตัวเลข (Digit), หรือสัญลักษณ์พิเศษ (Special symbols) ตัวแปร char จะเก็บข้อมูลเป็นตัวเลข (ASCII Code) โดยมีขอบเขตตั้งแต่ -128 ถึง 127

**ข้อสังเกต :** สามารถแปลงตัวอักษรให้เป็น ASCII Code ได้โดยการใส่ '' ครอบส่วนที่อักขระต้องการแปลงเป็นตัวเลข

```
1 #include<stdio.h>
2
3 main()
4 {
5     char test;
6
7     test = 'A'; //valid
8     printf("Char A = %c\n", test);
9     printf("ASCII A = %d\n", test);
10
11    test = 'ABC'; //invalid
12    printf("Char A = %c\n", test);
13    printf("ASCII A = %d", test);
14 }
```

## ชนิดข้อมูลแบบสายอักขระ (Group of Character Type)

ในภาษา C ไม่มีข้อมูลประเภท String แต่สามารถใช้แกลล่าดับ (Array) เข้ามาช่วย ชี้การกำหนดค่าจะใช้ " " ครอบสายอักขระที่ต้องการจะกำหนดค่าตั้งตัวอย่างด้านขวา

**ข้อควรรู้ :** การเก็บข้อความ (String) ใช้ตัวแปรประเภท char (ชนิดข้อมูลแบบตัวอักษร) มาต่อ กันเรื่อย ๆ จนได้เป็นข้อความ (สายของอักขระ) ซึ่งการนำตัวแปร หรอกลุ่มข้อมูลประเภทเดียวกันมาเรียงล่าดับกัน จะเรียกว่า แกลล่าดับ หรืออาร์เรย์ (Array)

```
1 #include<stdio.h>
2
3
4 main()
5 {
6     char test[] = "Hello";
7
8
9     printf("String : %s\n", test);
10    printf("test[0] = %c\n", test[0]);
11    printf("test[1] = %c\n", test[1]);
12    printf("test[2] = %c\n", test[2]);
13    printf("test[3] = %c\n", test[3]);
14    printf("test[4] = %c", test[4]);
15 }
```

## Null Character และ Empty String

ในการกำหนดค่าเริ่มต้นให้เป็นค่าว่างสามารถกำหนดให้เป็นตัว Null Character (\0) ซึ่งค่าของ Null Character ('\0') จะมีค่าเป็น 0 (ASCII Code) สำหรับตัวแปรข้อมูลชนิดสายอักขระถ้าหากไม่ได้กำหนดค่าเริ่มต้น (**char a;** หรือ **char b[5];**) ค่าที่เก็บจะเป็น "" โดยอัตโนมัติ

การกำหนดค่าเริ่มต้นของตัวแปรข้อมูลชนิดสายอักขระเป็นข้อความ (**char b[3] = "ab";**) อินเด็กซ์ (Index) ที่เหลือ คือตัวแปร **b[2]** จะถูกกำหนดค่าเป็น "" อัตโนมัติ และจะเรียกค่าที่เดินให้อัตโนมัตินี้ว่า Empty String / Zero Termination Character / Null Terminated String ปกติจะไม่สามารถกำหนด "" เองได้ แต่สามารถใช้ \0 แทน เช่น

```
||| char camp[12] = "CE Boost UP";
```

Index	0	1	2	3	4	5	6	7	8	9	10	11
Value	C	E		B	o	o	s	t		U	P	""

จากตาราง Index ที่ 2 และ 8 จะมีค่าเป็น " " คือช่องว่าง หรือ Space bar (ASCII Code = 32) ในใช้ Empty String และ Index ที่ 11 มีค่าเท่ากัน **"" ซึ่งเทียบเท่า '\0'** คือ Null Character (ASCII Code = 0)

**ข้อสังเกต :** หากประกาศตัวแปรแล้วใส่จำนวน Index น้อยกว่าข้อมูลจะทำให้เกิด Error ขึ้นได้ เช่น **char camp[5] = "CE Boost UP";** จะทำให้โปรแกรมแจ้งข้อผิดพลาด Array bounds overflow หรืออย่างอื่นขึ้นอยู่กับคอมไพล์เลอร์ที่ใช้

**NOTE:** บล็อก (block) หมายถึง ส่วนของ Code ที่อยู่ในระหว่างเครื่องหมาย { }

## ประเภทตัวแปร (Variable Type)

1. **ตัวแปรแบบโลคอล (Local Variable)** เป็นตัวแปรที่มีการประกาศไว้ที่ภายในฟังก์ชัน หรือภายในขอบเขตของบล็อกใด ๆ ซึ่งไม่สามารถเรียกใช้งานนอกฟังก์ชัน หรือภายนอกบล็อกที่ประกาศไว้ได้

2. **ตัวแปรแบบโกลบอล (Global Variable)** เป็นตัวแปรที่มีการประกาศไว้นอกฟังก์ชัน ซึ่งสามารถเรียกใช้งานในส่วนใดของโปรแกรมก็ได้

## ตัวอย่างโปรแกรม

```
1 #include<stdio.h>
2
3 int num = 10;
4
5 int testver (int x)
6 {
7     int n = 4, sum = 0;
8     sum = n * x;
9     return sum;
10 }
11
12 main()
13 {
14     int xnum;
15     printf("Global var. is %d\n", num);
16     xnum = testver(num);
17     xnum = xnum * n;
18     printf("Local var. is %d\n", x);
19 }
```

ผลลัพธ์โปรแกรม : คอมไพล์ไม่ผ่าน

เพราะบรรทัดที่ 17 ที่ทำให้เกิดการ Error ของโปรแกรม

บรรทัดที่ 3 : ประกาศตัวแปร **num** เป็น global variable ซึ่งทุกฟังก์ชันสามารถเรียกใช้ได้

บรรทัดที่ 7 : ประกาศตัวแปร **n** และ **sum** เป็น Local variable ซึ่งเจ้าของตัวแปร **n** และ **sum** มีฟังก์ชัน **testver()** เป็นเจ้าของ เพราะฉะนั้นจะมีแค่ **testver()** เท่านั้นที่ใช้ได้

บรรทัดที่ 14 : ประกาศตัวแปร **xnum** เป็น Local variable ซึ่งจะใช้ได้แค่ฟังก์ชัน **main()** เท่านั้น

บรรทัดที่ 16 : เป็นการเรียกใช้เรียกใช้ฟังก์ชัน **testver()** โดยส่งพารามิเตอร์เป็นเลขจำนวนเต็มเข้าไปในฟังก์ชัน โดยตัวเลขที่ส่งไปคือค่าของ **num** นั้นเอง และรอค่าที่ส่งกลับมาจากฟังก์ชัน **testver()** เพื่อกำหนดค่าให้ตัวแปร **xnum**

บรรทัดที่ 5 : ฟังก์ชัน **testver()** รับค่าเลขจำนวนเต็ม **num** ซึ่งมีค่าเท่ากับ 10 เข้ามาในฟังก์ชันจากนั้นนำไปคูณกับ **n** ที่เป็นตัวแปรโลคอลได้ผลเท่ากับ 40 และเก็บค่าไว้ที่ตัวแปร **sum** แล้วจึงส่งค่ากลับสู่ฟังก์ชัน **main()** ในบรรทัดที่ 9

บรรทัดที่ 17: พอดีบรรทัดนี้โปรแกรมจะเกิดปัญหาขึ้น เพราะเรียกใช้ตัวแปรแบบ Local ในฟังก์ชัน **testvar()** คือตัว **n** นั้นเอง โดยจะแสดงผล **Error : Undefined symbol 'n' in function main**

## การกำหนดค่าให้ตัวแปร (Variable Assignment)

### 1. กำหนดค่าตอนประกาศตัวแปร

||| `dataType varName = value;`

<b>โดยที่</b> <b>dataType</b>	เป็นชนิดของข้อมูล	เช่น <b>int</b> <code>x = -1150, 1112;</code>
<b>value</b>	เป็นค่าที่ต้องการกำหนดให้ตัวแปร <code>varName</code>	<b>float</b> <code>price = 12.50;</code> <b>char</b> <code>alphabet = 'A';</code> <b>char</b> <code>alphabet = 65;</code>

\* Array เช่น **char** `stat[] = "PASS";` หรือ **char** `stat[5] = "PASS";` หรือ **char** `stat[5] = {'P', 'A', 'S', 'S', '\0'};`

**ข้อสังเกต :** การประกาศตัวแปรสำหรับเก็บข้อความจะต้องเพื่อขนาดเพิ่ม 1 ตัวเพื่อสำหรับเก็บ `\0` ในตำแหน่งสุดท้าย

### 2. กำหนดค่าภายในฟังก์ชัน (ภายหลังจากการประกาศตัวแปร)

||| `left = right;`

จะใช้ตัวดำเนินการ = คือการนำค่าด้านขวา (`right`) ของตัวดำเนินการ = ที่ถูกกระทำเรียบร้อย มาใส่ในตัวถูกกระทำการทางด้านซ้าย (`left`) เช่น

```

x = -1150;           //นำค่า -1150 ใส่ตัวแปร x (ให้ x เป็นตัวแปรประเภท int)
price = 12.50;         //นำค่า 12.50 ใส่ตัวแปร price (ให้ price เป็นตัวแปรประเภท float)
alphabet = 'A'; หรือ alphabet = 65; //นำค่า 65 ใส่ตัวแปร alphabet (ให้ alphabet เป็นตัวแปรประเภท char)
number = 12 + 3 / 3;   //นำค่า 13 ใส่ตัวแปร number (ให้ number เป็นตัวแปรประเภท int)
code = 'A' + 2;        //นำค่า 67 ใส่ตัวแปร code (ให้ code เป็นตัวแปรประเภท char)
hex = 0x88;           //นำค่า 136 (เลขฐาน 16 เพราะมี 0x นำหน้า) ใส่ในตัวแปร hex (ให้ hex เป็นตัวแปรประเภท int)
cal = a + 5;          /* นำค่าที่ตัวแปร a เก็บเพิ่มค่าอีก 5 สมนติให้ a เก็บค่า 5 ดังนั้น
                           จะได้ค่า 10 ใส่ตัวแปร cal (ให้ cal เป็นตัวแปรประเภท short) */
stat[] = "PASS";       /* นำค่า 80 ใส่ตัวแปร stat[0], 65 ใส่ตัวแปร stat[1] และ 83 ใส่ตัวแปร
                           stat[2] และ stat[3] (ให้ stat[] เป็นตัวแปรประเภท char array) */

```

### 3. กำหนดค่าโดยผ่านค่าเข้าไปในฟังก์ชัน (ยังไม่กล่าวถึงในตอนนี้)

**ข้อควรรู้ :** ใช้ `#define` เปรียบเสมือนกำหนดค่าให้ตัวแปร

||| `#define new original`

<b>โดยที่</b> การใช้ค่าสั้นนี้ควรใช้ในส่วนของ Header เช่น <code>#define x 5</code>	<code>#define print printf</code>	
	<code>#define y (20/4)</code>	<code>#define NAME "google.com"</code>

**ข้อสังเกต :** การใช้ `#define` “ไม่ใช่การกำหนดค่าให้ตัวแปร” แต่เป็นการกำหนดความหมายให้ Complier รับรู้ว่าค่าที่ถูก `define (original)` คือเอาอะไรมาแทน (`new`) “การ `define` จึงคล้ายการกำหนดค่าให้ตัวแปร” ถ้าใช้ `#define` เปรียบเสมือนกำหนดค่าให้ตัวแปร ตัวแปรที่ถูก `#define` ไว้จะไม่สามารถเปลี่ยนแปลงค่าในฟังก์ชันได้ ๆ ได้อีก และการใช้ `#define` สามารถใช้กำหนดค่าสั้นใส่ในสิ่งที่ต้องการนำมาแทน (`new`) เพื่อใช้สิ่งที่กำหนดแทนค่าสั้นเดิม ๆ เหล่านั้นได้ เช่น `#define print printf` เพื่อใช้งาน `print` แทนค่าสั้น `printf` เป็นต้น

## ค่าคงที่ (Constant)

คือค่าข้อมูลชนิดใดชนิดหนึ่งที่ไม่มีการเปลี่ยนแปลงในขณะที่โปรแกรมทำงาน ตัวอย่างเช่น ค่า PI ซึ่งมีค่าเท่ากับ 3.14 เป็นต้น ซึ่งในภาษา C สามารถใช้งานได้ 3 รูปแบบ คือ ระบุค่าโดยตรง, นิยามโดย `#define` และเก็บไว้ในตัวแปร

### ระบุค่าโดยตรง (Literal Constants)

เป็นการกำหนดค่าคงที่เพื่อใช้งานโดยตรง โดยไม่มีการกำหนดค่าผ่านตัวแปรใด ๆ ทั้งสิ้น ตัวอย่างเช่น 'I', "I LOVE KMITL", 1, "\007" เป็นต้น

## นิยามโดย #define (Defined Constants)

เป็นการกำหนดค่าคงที่โดยการประกาศไว้ก่อนใช้งาน โดยมีรูปแบบการประกาศใช้งานค่าคงที่ดังนี้

||| `#define CONSTANTNAME value`

โดยที่ CONSTANTNAME คือ ชื่อของค่าคงที่ (ควรใช้เป็นตัวอักษรพิมพ์ใหญ่) เช่น `#define VAT 0.07`  
 Value คือ ค่าที่ต้องการกำหนดให้กับค่าคงที่ `#define TXT "HBD 2 Me"`  
`#define NEWLINE '\n'`  
`#define ONE 1`

## เก็บไว้ในตัวแปร (Memory Constants)

เป็นการกำหนดค่าคงที่ในรูปแบบของตัวแปร โดยมีรูปแบบการประกาศใช้งานค่าคงที่ดังนี้

||| `const DataType VARIABLENAME = value;`

โดยที่ DataType คือ ชนิดข้อมูลของค่าคงที่ เช่น `const float VAT = 0.07;`  
 VARIABLENAME คือ ชื่อของค่าคงที่ (ควรใช้เป็นตัวอักษรพิมพ์ใหญ่) `const int COUNT = 10;`  
 value คือ ค่าที่ต้องการกำหนดให้กับค่าคงที่ `const char CH = 'T';`

## การแปลงชนิดของข้อมูล (Data Type Conversion)

ข้อมูลต่าง ๆ จะสามารถดำเนินการได้จะต้องเป็นข้อมูลประเภทเดียวกันเท่านั้น หากต้องการนำข้อมูลที่ไม่ใช่ประเภทเดียวกันมาใช้ ก็จะต้องทำการแปลงข้อมูลให้เป็นชนิดเดียวกันก่อน ซึ่งสามารถแปลงได้ 2 วิธีคือ

### Implicit Type Conversion

การแปลงข้อมูลชนิดนี้ คอมไพล์เลอร์จะทำหน้าที่แปลงข้อมูลให้อัตโนมัติโดยแปลงชนิดข้อมูลที่มีนัยสำคัญต่ำ ไปเป็นชนิดข้อมูลเดียวกันที่มีนัยสำคัญสูงกว่าในชุดค่าสั้งนั้น ๆ โดยมีล่าดับนัยสำคัญดังนี้

ด้ำ `char > short > int > unsigned int > long int > unsigned long int > float > double > long double` สูง

เช่น • `int + long` : แปลง int ไปเป็น long (long มีนัยสำคัญสูงกว่า int)  
 • `char - float` : แปลง char ไปเป็น float (float มีนัยสำคัญสูงกว่า char)  
 • `float * long double` : แปลง float ไปเป็น long double (long double มีนัยสำคัญสูงกว่า int)  
 • `int / double` : แปลง int ไปเป็น double (double มีนัยสำคัญสูงกว่า int)  
 • `(short + long) * float` : จะทำในวงเล็บก่อน โดยแปลง short ไปเป็น long จากนั้นจะนำค่าที่ได้ไปแปลงเป็น float (วงเล็บมีล่าดับความสำคัญสูงที่สุดจึงต้องทำในวงเล็บก่อน)  
 • `(float - long) * short` : จะทำในวงเล็บก่อน โดยแปลง long ไปเป็น float จากนั้นจะนำค่า short มาแปลงเป็น float (ค่าที่ได้ในวงเล็บคือ float ซึ่งมีนัยสำคัญสูงกว่า short)

### Explicit Type Conversion (Casting)

การแปลงชนิดข้อมูลด้วยวิธีนี้คือ การแปลงชนิดข้อมูลโดยกำหนดได้ตามที่ผู้ใช้งานต้องการ สามารถทำได้โดยรูปแบบดังต่อไปนี้

||| `(DataType) ExpressionOrVariableName`

โดยที่ DataType คือ ชนิดข้อมูลปลายทาง  
 ExpressionOrVariableName คือ นิพจน์หรือตัวแปรที่ต้องการจะแปลงข้อมูล

ตัวอย่างการแปลงข้อมูลแบบ Explicit Type Conversion มีดังต่อไปนี้ กำหนดให้ `x` มีชนิดข้อมูลเป็น int  
`y` มีชนิดข้อมูลเป็น long int  
`z` มีชนิดข้อมูลเป็น float

```

z = (float) x           //แปลงชนิดข้อมูลตัวแปร x จาก int ไปเป็น float และนำผลลัพธ์ที่ได้เก็บลงตัวแปร z
x = (int) y           //แปลงชนิดข้อมูลตัวแปร y จาก long int ไปเป็น int และนำผลลัพธ์ที่ได้เก็บลงตัวแปร x
z = (float) (x + y)   /* จะทำ x + y ก่อน จากนั้นจะแปลงค่าที่ได้จาก x + y เป็น float และนำผลลัพธ์ที่ได้เก็บลงตัวแปร z */
z = (float) x / y     /* จะแปลงตัวแปร x เป็นชนิดข้อมูล float (Explicit Type Conversion) จากนั้นจะแปลงตัวแปร y เป็นข้อมูลชนิด float (Implicit Type Conversion) และนำผลลัพธ์ที่ได้เก็บลงตัวแปร z */

```

## การแสดงผลทางหน้าจอ (Display Data)

มีคำสั่งที่นิยมใช้แสดงผลอยู่ในสีเดอร์ไฟล์ stdio.h หลัก ๆ อยู่ 3 คำสั่งคือ

### 1. ใช้คำสั่ง printf

การแสดงผลของข้อความ

```
printf("TEXT");
```

โดยที่ text เป็นข้อความที่ต้องการจะแสดงผล

การแสดงผลของค่าที่เก็บในตัวแปร

```
printf("format_1format_2.....format_n", var_1, var_2, ..., var_n);
```

โดยที่ var\_n คือ ค่าที่มาจากการกำหนด หรือมาจากการตัวแปรที่ต้องการแสดงผล

format\_n คือ รูปแบบการแสดงผล ซึ่งต้องใช้ให้ตรงกับประเภทของ var\_n นั้น ๆ หรือรหัสค่าสั่งพิเศษ/รหัสควบคุม (Escape Sequence)

จากตัวอย่าง

```

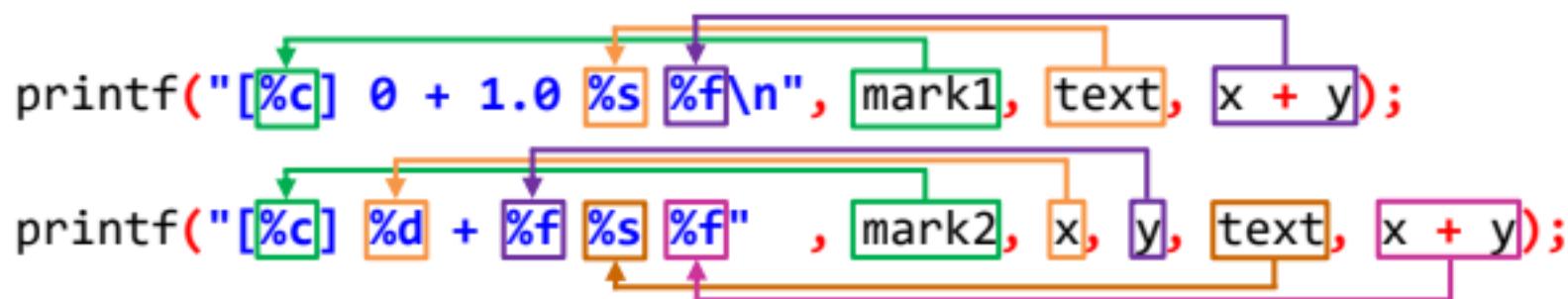
1 #include<stdio.h>
2
3 main()
4 {
5     int x = 0;
6     float y = 1.0;
7     char mark1 = 'B', mark2 = 'C';
8     char text[] = "is";
9     printf("[A] 0 + 1.0 is 1.000\n");
10    printf("[%c] 0 + 1.0 %s %f\n", mark1, text, x + y);
11    printf("[%c] %d + %f %s %f" , mark2, x, y, text, x + y);
12 }
```

ข้อ A เป็นการแสดงผลข้อความทั้งหมดจะได้ผลลัพธ์ที่อยู่ในเครื่องหมาย "" ออกมากันทั้งหมด และตอนสุดท้ายจะทำการขึ้นบรรทัดใหม่จาก \n ซึ่งเป็นรหัสค่าสั่งพิเศษ (Escape Sequence)

ข้อ B เป็นการแสดงผลข้อความพร้อมกับแสดงค่าในตัวแปร โดยจะแสดงข้อความ "[", แสดงค่าในตัวแปร mark1 (B), แสดงข้อความ "] 0 + 1.0 ", แสดงค่าในตัวแปร text (is), แสดงข้อความ " ", แสดงค่า x + y (1.000000) และทำการขึ้นบรรทัดใหม่

ข้อ C เป็นการแสดงผลข้อความพร้อมกับแสดงค่าในตัวแปร โดยจะแสดงข้อความ "[", แสดงค่าในตัวแปร mark2 (C), แสดงข้อความ "] ", แสดงค่าในตัวแปร x (0), แสดงข้อความ "+ ", แสดงค่าในตัวแปร y (1.000000), แสดงข้อความ " ", แสดงค่าในตัวแปร text (is), แสดงข้อความ " ", แสดงค่า x + y (1.000000)

**ข้อสังเกต :** ค่าที่แสดงผลออกมานะจะแสดงผลทุกตัวเหมือนในเครื่องหมาย "" ทุกประการยกเว้นรหัสค่าสั่งพิเศษ/รหัสควบคุมเช่น \t, \r และ format ในการแสดงผล เช่น %d %f ที่แสดงผลแตกต่างตามหน้าที่ ซึ่งสามารถแสดงข้อความและแสดงค่าจากตัวแปรได้ภายใน "" พร้อม ๆ กันหลายตัวได้ เช่น printf("text %d = %f", var1, var2);



```

1 #include<stdio.h>
2
3 main()
4 {
5     float x = 1.3;
6     printf("%.8f", x);
7     if(x == 1.3)    printf("x = 1.3");
8 }
```

**ข้อควรรู้ :** เนื่องจากคอมพิวเตอร์จึง ๆ แล้ว เก็บข้อมูลเป็นบิต ('0', '1') เท่านั้น คอมพิวเตอร์จึงใช้หลักการ IEEE 754 ในการแปลงเป็นเลขศนิยม จากตัวอย่างถ้าค่า 1.3 เมื่อถูกเก็บในหน่วยความจำ จึงมีค่าไม่เท่ากัน 1.3 ดังนั้นจึงไม่ควรใช้หลักนี้ในการเปรียบเทียบ

จากตัวอย่าง เมื่อสั่งแสดงผลเป็นจำนวนจริง ถ้าไม่ได้กำหนดการแสดงผลเพิ่มเติม จะแสดงหลักนี้โดยอัตโนมัติ และหากสั่งแสดงผลเลขศนิยมที่ถูกเก็บลงในหน่วยความจำอย่างละเอียด (หลักนี้หายไป) ก็จะแสดงผลแบบตัวแปรแต่ละประเภท จะพบว่าหลักนี้ในตัวแปรหลัง ๆ บางตัวมีค่าผิดเพี้ยน และถ้าสั่งแสดงหลักนี้โดยใช้รูปแบบการแสดงผลผิดประเภทค่าที่ได้ออกมาจะเป็น 0

```

1 #include<stdio.h>
2
3 main()
4 {
5     float PI = 3.14;
6     printf("%f\n", PI);
7     printf("%.10f\n", PI);
8     printf("%.10f\n", 1.50);
9     printf("%f\n", 10.0/3.0);
10    printf("%f\n", 10.0/6.0);
11    printf("%d\n", 10.0);
12 }
```

**ข้อควรรู้ :** หน้าต่างสีดำที่ได้หลังจาก Compile & Run จะเรียกว่าหน้า Console ซึ่งในคอมไฟเลอร์สมัยก่อนมีความกว้างสูงสุด 80 ตัวอักษร 25 บรรทัด หากแสดงผลเกิน 80 ตัวอักษรจะทำการขึ้นบรรทัดใหม่ให้เอง

**ข้อสังเกต :** หากนำข้อมูลชนิดหลักนี้เก็บไว้ในตัวแปรประเภทจำนวนเต็มจะถูกปัดเศษทั้ง 'ไม่ปัดขึ้น หรือปัดลง' ตามหลักคณิตศาสตร์

**ข้อสังเกต :** หากนำข้อมูลชนิดข้อความเก็บไว้ในตัวแปรประเภทอักขระ ตัวสุดท้ายของข้อความจะถูกเก็บลงตัวแปร

**ข้อสังเกต :** หากนำข้อมูลชนิดจำนวนเต็มมาดำเนินการ กันจะได้ผลลัพธ์เป็นข้อมูลจำนวนเต็มด้วยเหมือนกัน ถึงแม้จะกำหนดให้แสดงผลเป็นหลักนี้ วิธีแก้ไข คือใช้ (float)centimeter / (float)100 หรือ (float)centimeter / 100 หรืออีกวิธีหนึ่งคือแก้เป็น centimeter / 100.0

```

1 #include<stdio.h>
2
3
4 main()
5 {
6     int o = 11.2, a = 11.8;
7     int centimeter = 72;
8     char DE = 'Germany';
9     printf("%d %c %d\n", o, DE, a);
10    printf("Meter = %f", centimeter/100);
11 }
```

## รหัสรูปแบบการแสดงผลข้อความ และการจัดรูปแบบการแสดงผลข้อมูลของค่าสั่ง printf()

การแสดงผลด้วยค่าสั่ง printf() นั้นสามารถแสดงผลได้เกือบทุกชนิดข้อมูล โดยใช้รหัสรูปแบบการแสดงผลแทนค่าจากตัวแปร นิพจน์ หรือค่าคงที่ชนิดต่างๆ ในข้อความ รหัสรูปแบบการแสดงผลควรใช้ให้ถูกต้องตามชนิดของข้อมูลที่ต้องการ หากเลือกใช้ผิดประเภทอาจทำให้การแสดงผลผิดพลาดได้ และนอก จากนี้ยังมีรหัสคำสั่งพิเศษ หรือรหัสควบคุม (Escape Sequence) เพื่อใช้ควบคุม หรือแสดงผลในรูปแบบพิเศษ

รหัสคำสั่งพิเศษ (Escape Sequence)	ความหมาย
\b	เลื่อนเคอร์เซอร์โดยหลังไป 1 ตัวอักษร
\n	ขึ้นบรรทัดใหม่
\r	เลื่อนเคอร์เซอร์ไปข้างสุดของบรรทัด
\t	เว้นวรคุณครับ 1 แท็บ (8 ช่อง)
'	แสดงเครื่องหมาย ' (single quote)
"	แสดงเครื่องหมาย " (double quote)
\\"	แสดงเครื่องหมาย \
\0	Null

**ข้อควรรู้[\*]** : ในสามารถใช้ข้อมูลชนิด long double ได้เนื่องจากคอมไพล์เตอร์ (GCC) กำหนด long double มีขนาด 128 bits แต่ Runtime Library (Microsoft's) กำหนดให้ long double มีขนาด 64 bits

รหัสรูปแบบ	ชนิดข้อมูล
%c	ตัวอักษรหนึ่งตัว
%d	จำนวนเต็มชนิด int
%ld	จำนวนเต็มชนิด long
%e หรือ %E	จำนวนจริงแบบเอ็กซ์โพเนนเช่น
%f	จำนวนจริง float
%lf	จำนวนจริง double
%g หรือ %G	จำนวนจริง (General format)
%i	จำนวนเต็มชนิด int
%o	เลขฐานแปด
%p	พอยน์เดอร์
%s	ข้อความ
%u	จำนวนเต็มบวก (Unsigned)
%x หรือ %X	เลขฐานสิบหก (พิมพ์เล็ก/ใหญ่)
%hd	จำนวนเต็มชนิด short
%lo	จำนวนเต็มฐานแปดชนิด long
%hx	จำนวนเต็มฐานสิบหกชนิด short
%Lf	จำนวนจริงชนิด long double
%lu	จำนวนเต็มชนิด unsigned long

จากตัวอย่าง หากใส่ \ ตัวเดียวตัวคอมไพล์เตอร์จะเข้าใจว่าสามารถประกอบเป็นรหัสคำสั่งพิเศษ หรือรหัสควบคุม (Escape Sequence) ได้ หรือไม่ (% ก็เขียนกันจะเข้าใจว่าสามารถประกอบกันเป็นรหัสรูปแบบได้ หรือไม่) ถ้าหากเป็นรหัสคำสั่งพิเศษก็จะแสดงรหัสพิเศษออกมานะ เช่น ในบรรทัดที่ 5 จะมี (\E) และขึ้นบรรทัดใหม่ (\n) ถูกแสดงผลเป็นรหัสพิเศษ, ส่วนบรรทัดที่ 6 จะมี " (" และขึ้นบรรทัดใหม่ (\n) เป็นรหัสพิเศษที่ถูกแสดงผลออกมานะ และถ้าหากไม่ใส่ " ปิดท้ายจะทำให้ค่าสั่ง printf(" ); ขาด (" (ในครับ Syntax) และในทางกลับกันถ้าหากไม่ใช้รหัสพิเศษจะแสดงตัวอักษรนั้นออกมานะ เช่น ในบรรทัดที่ 7 A (\A) จะถูกแสดงผลออกมานะเป็นตัวอักษรตามปกติ หรือในบางคอมไพล์เตอร์จะแจ้งเตือน error ขึ้น [Warning] unknown escape sequence: '\A'

**ข้อควรรู้ :** จากตัวอย่างเมื่อสั่งแสดงผล % ตัวเดียวจะไม่เกิดอะไรขึ้นที่หน้าจอ ต้องใช้ %% ในการแสดงผล

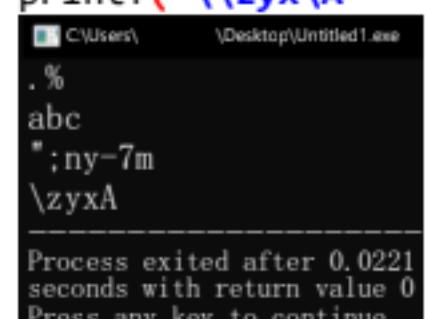
### การจัดพื้นที่ และการกำหนดการแสดงผลข้อมูลของค่าสั่ง printf()

บางครั้งการแสดงผลข้อมูลที่เราต้องการนั้น อาจจะต้องการจัดรูปแบบข้อความให้ชิดซ้ายหรือขวาหรือแม้แต่การแสดงข้อความตามจำนวนตัวอักษรที่กำหนด เช่น ต้องการแสดงทศนิยม 2 ตำแหน่ง, และแสดงข้อความไม่เกิน 10 อักษร เป็นต้น ซึ่งมีรูปแบบในการแสดงผลข้อมูลชนิดข้อความดังนี้

```
printf("%[m].[n]s", variable);
```

**โดยที่** [m] เป็นจำนวนช่องส่วนรับໃห้ในการแสดงข้อความ เช่น  
 [n] เป็นจำนวนตัวอักษรที่ต้องการแสดงผล  
 variable เป็นค่าข้อมูลหรือตัวแปรที่ต้องการแสดงผล

```
1 #include<stdio.h>
2 main()
3 {
4     printf("%..% \n");
5     printf("\Eabc \n");
6     printf("\";ny-7m\n");
7     printf("\\zyx\A ");
8 }
```



```
13 Process exited after 0.0221
seconds with return value 0
Press any key to continue .
```

ในการถีแสดงค่าจำนวนเต็ม มีรูปแบบการแสดงผลดังนี้

||| `printf("%[m]d", variable);`

โดยที่ [m] เป็นจำนวนช่องส่าหรับใช้ในการแสดงตัวเลข เช่น `printf("%8d", 12);`  
variable เป็นค่าข้อมูลหรือตัวแปรที่ต้องการแสดงผล

ในการถีแสดงค่าจำนวนจริง มีรูปแบบการแสดงผลดังนี้

||| `printf("%[m].[n]f ", variable);`

โดยที่ [m] เป็นจำนวนช่องส่าหรับใช้ในการแสดงตัวเลข เช่น `printf("%8f", 1.0);`  
[n] เป็นจำนวนหลักทศนิยมที่ต้องการแสดงผล เช่น `printf("%.2f", 2.5);`  
variable เป็นค่าข้อมูลหรือตัวแปรที่ต้องการแสดงผล เช่น `printf("%6.3f", 3.1);`

ข้อสังเกต : [m] และ [n] สามารถใส่ \* แทนได้ และกำหนดตัวเลขที่จะจัดรูป (เว้นช่องว่าง/กำหนดตำแหน่งทศนิยม)  
ไว้หลังส่วนของรูปแบบการแสดงผลได้ เช่น `printf("%5d", 10);` จะสามารถใช้ `printf("%*d", 5, 10);`  
`printf("%6.2f", 3.14);` จะสามารถใช้ `printf("%*.*f", 6, 2, 3.14);`

ในบางครั้งต้องการแสดงผลขิดข้าย หรือขิดขวาในส่วนแสดงผล หรืออาจจะต้องการกำหนดเครื่องหมายให้กับ  
ข้อมูลตัวเลข ภาษา C ได้กำหนดรูปแบบการแสดงผลต่าง ๆ เหล่านั้นไว้ดังนี้

ค่า flag	ความหมาย
-	ใช้กำหนดการแสดงผลทางจอกภาพให้ขิดข้าย ตามความกว้างของส่วนแสดงผลที่กำหนดไว้
+	ใช้กำหนดให้แสดงเครื่องหมาย + หรือ - หน้าค่าข้อมูลที่ต้องการแสดงผล
0	ใช้กำหนดให้แสดงเลข 0 หน้าค่าข้อมูลที่ต้องการแสดงผล
#	ใช้กำหนดให้แสดงเลข 0 หน้าค่าข้อมูลเลขฐานแปด, แสดง 0x และ 0X หน้าค่าข้อมูลเลขฐานสิบหก หรือใช้กำหนดให้แสดงผลเป็นเลขทศนิยม ในกรณีที่ข้อมูลที่ต้องการนั้นมีค่าเป็นจำนวนเต็ม หรือใช้ กำหนดให้แสดงผลข้อมูลเลขศูนย์ต่อท้าย ในกรณีที่ข้อมูลที่ต้องการนั้นเป็นเลขทศนิยมที่มีศูนย์ต่อท้าย

ตัวอย่าง การจัดรูปการถีแสดงผลของตัวเลขจำนวนเต็ม จำนวนจริง และสายอักขระ (ข้อความ)

```

1 #include<stdio.h>
2 main()
3 {
4     printf("Right justify : %10d\n" , 112);
5     printf("Left justify : %-10d\n\n", 112);
6
7     printf("Default minus : %d\n", -123);
8     printf("Default zero : %d\n", 0);
9     printf("Default plus : %d\n", 123);
10    printf("Display minus : %+d\n", -123);
11    printf("Display zero : %+d\n", 0);
12    printf("Display plus : %+d\n\n", 123);
13
14    printf("Default : %d\n" , 1234);
15    printf("Zero show : %010d\n\n", 1234);
16
17    printf("String : %10.6s", "long...text");
18    printf("\nFloat : %.*f\n", 5, 2, 3.14);
19    printf("Float : %8.3f\n\n", 12.38456);
20
21    printf("No# : %o %x %G\n", 100, 100, 10.2);
22    printf("Use# : %#o %#x %#G" , 100, 100, 10.2);
23 }
```

## 2. ใช้คำสั่ง putchar() สำหรับแสดงแบบอักขระตัวเดียว (Character)

||| `putchar(var);`

โดยที่ var เป็นตัวแปรชนิดอักขระ (char) ที่เก็บค่าเป็นตัวเลข ASCII ซึ่งแปลงอักขระเป็น ASCII โดยใช้ '' ครอบอักขระ

## 3. ใช้คำสั่ง puts() สำหรับแสดงข้อมูลแบบสายอักขระ (String)

||| `puts(var);`

โดยที่ var เป็นตัวแปรชนิดข้อความ หรือเป็นค่าข้อความภาษาในเครื่องหมาย " "

ข้อสังเกต : คำสั่ง putchar เมื่อแสดงผลเรียบร้อยแล้วจะไม่มีการขึ้นบรรทัดใหม่ แต่คำสั่ง puts จะขึ้นบรรทัดใหม่เมื่อแสดงผลเรียบร้อยแล้ว

```
1 #include<stdio.h>
2
3 main()
4 {
5     char ch = 'T';
6     putchar(ch);
7     putchar('A');
8     putchar(66);
9 }
10 TAB
11
12 Process exited after 0.02209
seconds with return value 0
13 Press any key to continue .
14
15
```

```
1 #include<stdio.h>
2
3 main()
4 {
5     char text[20] = "Thailand0.4";
6     puts(text);
7     puts(" ");
8     puts("**Edit Thailand4.0");
9 }
10 Thailand0.4
11
12 *Edit Thailand4.0
13
14
15 Process exited after 0.02509
seconds with return value 0
16 Press any key to continue . . .
```

## ตัวดำเนินการ (Operator)

ในการเขียนโปรแกรมทุก ๆ โปรแกรมจะต้องมีการประมวลผลเข้ามาเกี่ยวกับตัวอย่างเช่น แล้วลิ่งที่ทำให้เกิดการประมวลผลนั้นก็คือ ตัวดำเนินการ ซึ่งตัวดำเนินการพื้นฐานที่ควรรู้และทำความเข้าใจเบื้องต้นเป็นประเภทต่าง ๆ ได้ดังนี้

### ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operator)

มีการทำงานเหมือนกับการใช้งานทางคณิตศาสตร์ทั่วไป โดยมีดังนี้

ตัวดำเนินการ	ตัวอย่าง	ตัวดำเนินการ	ตัวอย่าง
+ บวก	$10 + 4 = 14$ $10 + 4.0 = 14.000000$ $'A' + 2 = 67$	- ลบ	$10 - 4 = 6$ $10 - 4.0 = 6.000000$
* คูณ	$10 * 4 = 40$ $10 * 4.0 = 40.000000$	/ หาร	$10 / 4 = 2$ $10 / 4.0 = 2.500000$ $-11 / 4 = -2$ $-11 / -4 = 2$
% หารเอาเศษ	$11 \% 4 = 3$ $4 \% 10 = 4$ $-11 \% 4 = -3$	เพิ่มเติม : นิพจน์ (Expression) คือ ข้อความหรือประโยคที่เขียนอยู่ในรูปสัญลักษณ์ โดยนำข้อมูล, ตัวแปร, พึงก์ชันหรือค่าคงที่มาสัมผันธ์กับตัวดำเนินการ (Operator) เช่น C + 1, A + B - (C - 7), A != B, A && B เป็นต้น	

ข้อควรรู้ : การทำงานของ % ในได้กับจำนวนเต็มเท่านั้น

การทำงานของ + , - , \* และ / ที่มีเฉพาะเลขจำนวนเต็ม

การทำงานของ + , - , \* และ / ที่มีเลขทศนิยมอยู่ด้วย

การทำงานของ + , - , \* และ / กับข้อมูลชนิดตัวอักขระ

จะได้ผลลัพธ์เป็นจำนวนเต็มเสมอ

จะได้ผลลัพธ์เป็นเลขจำนวนเต็มเสมอ

จะได้ผลลัพธ์เป็นเลขทศนิยมเสมอ (ตามกฎของการแปลงชนิดของข้อมูล)

จะทำโดยแปลงตัวอักขระเป็นค่า ASCII Code ก่อนจะนำไปประมวลผล

## ตัวดำเนินการกำหนดค่า (Assignment Operator)

ใช้สำหรับกำหนดค่าให้กับตัวแปรทางด้านซ้ายของตัวดำเนินการ โดยจะมีหลักการทำงานที่แตกต่างกันไปและกำหนดให้มีตัวแปรตั้งนี้ **ตัวแปร A และ B** เป็นขอนิดข้อมูลแบบตัวเลขจำนวนเต็ม กำหนดให้เท่ากับ 10 และ 4 ตามลำดับ **ตัวแปร C** เป็นข้อมูลชนิดตัวอักษร กำหนดให้เท่ากับ 'A' **ตัวแปร D** เป็นขอนิดข้อมูลแบบจำนวนเลขอctal กำหนดให้เท่ากับ 3.12 ดังนี้

ตัวดำเนินการ	ความหมาย
= : เท่ากับ	นำค่าตัวถูกกระทำทางด้านขวามาใส่ในตัวถูกกระทำทางด้านซ้าย <b>C = 'A'</b> $\rightarrow$ C = 'A' $\rightarrow$ C มีค่าเท่ากับ 65 (ASCII Code) <b>A = 10</b> $\rightarrow$ A = 10 $\rightarrow$ A มีค่าเท่ากับ 10
+ = : บวกเท่ากับ	นำค่าตัวถูกกระทำทางด้านซ้าย เท่ากับค่าตัวถูกกระทำทางด้านซ้ายบวกกับ ค่าตัวถูกกระทำทางด้านขวา <b>A += B</b> $\rightarrow$ A = A + B = 10 + 4 $\rightarrow$ A มีค่าเท่ากับ 14 <b>A += D</b> $\rightarrow$ A = A + D = 10 + 3.12 = 13.120000 $\rightarrow$ A มีค่าเท่ากับ 13 (เนื่องจาก A เป็นข้อมูลชนิดเลขจำนวนเต็ม) <b>A += C</b> $\rightarrow$ A = A + C = 10 + 65 $\rightarrow$ A มีค่าเท่ากับ 75 (การ + - * / % กับตัวอักษรจะแปลงเป็นค่า ASCII ซึ่งในที่นี้ 'A' มีค่า ASCII = 65) <b>D += A + B</b> $\rightarrow$ D = D + (A + B) = 3.12 + (10 + 4) = 3.12 + (14) $\rightarrow$ D = 17.120000 ดังนั้น D มีค่าเท่ากับ 17.12
- = : ลบเท่ากับ	นำค่าตัวถูกกระทำทางด้านซ้าย เท่ากับค่าตัวถูกกระทำทางด้านซ้ายลบกับ ค่าตัวถูกกระทำทางด้านขวา <b>A -= A</b> $\rightarrow$ A = A - A = 10 - 10 $\rightarrow$ A มีค่าเท่ากับ 0 <b>A -= A - B</b> $\rightarrow$ A = A - (A - B) = 10 - (10 - 4) $\rightarrow$ A มีค่าเท่ากับ 4
* = : คูณเท่ากับ	นำค่าตัวถูกกระทำทางด้านซ้าย เท่ากับค่าตัวถูกกระทำทางด้านซ้ายคูณกับ ค่าตัวถูกกระทำทางด้านขวาแสดงผล <b>D *= 2</b> $\rightarrow$ D = D * 2 = 3.12 * 2 = 6.240000 $\rightarrow$ D มีค่าเท่ากับ 6.24
/ = : หารเท่ากับ	จะนำค่าตัวถูกกระทำทางด้านซ้าย เท่ากับค่าตัวถูกกระทำทางด้านซ้ายหารกับ ค่าตัวถูกกระทำทางด้านขวา <b>A /= B / 2</b> $\rightarrow$ A = A / (B / 2) = 10 / (4 / 2) $\rightarrow$ A มีค่าเท่ากับ 5
% = : หารเอาเศษเท่ากับ	จะนำค่าตัวถูกกระทำทางด้านซ้าย เท่ากับเศษเหลือจากการหารระหว่าง ค่าตัวถูกกระทำทางด้านซ้าย กับค่าตัวถูกกระทำทางด้านขวา <b>A %= B</b> $\rightarrow$ A = A % B = 10 % 4 $\rightarrow$ A มีค่าเท่ากับ 2 <b>D = D - A % B</b> $\rightarrow$ D = D - (A % B) = 3.12 - (10 % 4) = 3.12 - (2) $\rightarrow$ D = 2.120000 (% สำคัญกว่า -) $\rightarrow$ D มีค่าเท่ากับ 1.12

## ตัวดำเนินการยูนารี (Unary Operator)

ตัวดำเนินการ	ความหมาย	
คือ การเพิ่ม ++ ค่าให้กับตัวแปรหนึ่งค่า	Postfix	<b>X = A++</b> $\rightarrow$ X = A $\rightarrow$ A = A + 1 $\rightarrow$ กำหนดค่าให้ X ก่อนแล้วค่อยเพิ่ม A 1 ค่า
	Prefix	<b>X = ++A</b> $\rightarrow$ A = A + 1 $\rightarrow$ X = A $\rightarrow$ เพิ่ม A ก่อน 1 ค่า จึงค่อยกำหนดค่าให้กับ X
คือ การลดค่า -- ให้กับตัวแปรหนึ่งค่า	Postfix	<b>X = A--</b> $\rightarrow$ X = A $\rightarrow$ A = A - 1 $\rightarrow$ กำหนดค่าให้ X ก่อนแล้วค่อยลด A 1 ค่า
	Prefix	<b>X = --A</b> $\rightarrow$ A = A - 1 $\rightarrow$ X = A $\rightarrow$ เพิ่ม A ก่อน 1 ค่า แล้วค่อยกำหนดค่าให้กับ X
+ เครื่องหมายบวก	Prefix	<b>A = +2</b> $\rightarrow$ A เท่ากับ 2 (กรณีเป็นค่านำจะใส่เครื่องหมาย + หรือไม่ใส่ก็ได้)
- เครื่องหมายลบ	Prefix	<b>A = -2</b> $\rightarrow$ A เท่ากับ -2 <b>A = -A</b> $\rightarrow$ A เท่ากับ 2 (เมื่อ isol หน้าตัวแปรได้ จะได้ผลลัพธ์ตรงข้าม)

```

1 #include<stdio.h>
2 main()
3 {
4     {
5         int a = 10, x = a;
6         printf("Before a++ a = %d, x = %d\n", a, x);
7         x = a++;
8         printf("a++      a = %d, x = %d\n", a, x);
9         printf("After a++ a = %d, x = %d\n\n", a, x);
10    }
11    {
12        int a = 10, x = a;
13        printf("Before ++a a = %d, x = %d\n", a, x);
14        x = ++a;
15        printf("++a      a = %d, x= %d\n", a, x);
16        printf("After ++a a = %d, x= %d", a, x);
17    }
18 }

```

C:\Users\ \Desktop\Untitled1.exe - \* x  
 Before a++ a = 10, x = 10  
 a++ a = 11, x = 10  
 After a++ a = 11, x = 10  
 Before ++a a = 10, x = 10  
 ++a a = 11, x = 11  
 After ++a a = 11, x = 11

Process exited after 0.02247  
 seconds with return value 0  
 Press any key to continue . . .

จากรูปนี้เราไปกำหนดค่าให้ตัวแปร x จากนั้น a จะมีค่าเพิ่ม 1 เป็น 11 แต่บรรทัดที่ 14 ค่า a จะมีค่าเพิ่ม 1 เป็น 11 ก่อน จากนั้นจึงนำค่า 11 กำหนดค่าให้ตัวแปร x

**ข้อสังเกต :** บางคนสงสัยว่าตัวแปรขึ้นเดียว (x และ a) ประกาศข้ามกันในฟังก์ชันได้ในนิพจน์ค่าตอบคือ “ไม่” แต่ในตัวอย่าง ตัวแปร x และ a ที่ประกาศไว้ในบรรทัดที่ 5 เป็นตัวแปร local ภายในปีกภาษาหรือบล็อก (block) ระหว่างบรรทัดที่ 4 และ 10 ภายในฟังก์ชัน main() อีกที หากโปรแกรมทำงานผ่านช่วงปีกภาษาหรือบล็อก (block) ปีกภาษีไปตัวแปร ก็จะไม่มีตัวแปร x และ a อีก หากต้องการใช้จึงต้องประกาศตัวแปรใหม่อีกรอบ

## ตัวดำเนินการเปรียบเทียบ (Comparison Operator)

เป็นตัวดำเนินการสำหรับเปรียบเทียบข้อมูลระหว่างตัวถูกกระทำทางด้านซ้าย และด้านขวาของตัวดำเนินการ ซึ่งผลลัพธ์จะมีค่าเป็นจริง (True) มีค่าเป็น 1 หรือเท็จ (False) มีค่าเป็น 0 เท่านั้นโดยมีการทำงานดังนี้

- ==** : ถ้าเท่ากันจะให้ผลลัพธ์เป็นจริง ถ้าไม่เท่าจะให้ผลลัพธ์เป็นเท็จ
- !=** : ถ้าไม่เท่ากันจะให้ผลลัพธ์เป็นจริง ถ้าเท่ากันจะให้ผลลัพธ์เป็นเท็จ
- >** : ถ้าซ้ายมากกว่าขวาจะให้ผลลัพธ์เป็นจริง ถ้าน้อยกว่าหรือเท่ากันจะให้ผลลัพธ์เป็นเท็จ
- >=** : ถ้าซ้ายมากกว่าหรือเท่ากันขวาจะให้ผลลัพธ์เป็นจริง ถ้าน้อยกว่าจะให้ผลลัพธ์เป็นเท็จ
- <** : ถ้าซ้ายน้อยกว่าขวาจะให้ผลลัพธ์เป็นจริง ถ้ามากกว่าหรือเท่ากันจะให้ผลลัพธ์เป็นเท็จ
- <=** : ถ้าซ้ายน้อยกว่าหรือเท่ากันขวาจะให้ผลลัพธ์เป็นจริง ถ้ามากกว่าจะให้ผลลัพธ์เป็นเท็จ

**ตัวอย่าง** กำหนดให้ A, B, C, D, E เป็นตัวแปรประเภท int เก็บค่า 3, 3, 4, 5 และ 6 ตามลำดับ กำหนดให้นิพจน์สีเขียวมีค่าความจริงเป็นจริง และนิพจน์สีแดงมีค่าความจริงเป็นเท็จ จะได้ผลลัพธ์ออกมานี้

A == A	A != A	A >= A	A <= A	A > A	A < A
A == B	A != B	A >= B	A <= B	A > B	A < B
A == C	A != C	A >= C	A <= C	A > C	A < C
A == D	A != D	A >= D	A <= D	A > D	A < D
A == E	A != E	A >= E	A <= E	A > E	A < E
		C >= A	C <= A	C > A	C < A

## ตัวดำเนินการตรรกะ (Logical Operator)

ใช้สำหรับกำหนดเงื่อนไขมากกว่า 1 เงื่อนไข ซึ่งผลลัพธ์ที่ได้จะได้จะมีแค่จริง หรือเท็จเท่านั้น

- && (และ)** : ใช้กำหนดเงื่อนไขในกรณีที่ต้องการให้นิพจน์ด้านซ้าย และด้านขวาให้เป็นจริงทั้งคู่จึงจะทำงานตามที่ต้องการ
- || (หรือ)** : ใช้กำหนดเงื่อนไข ในกรณีที่ต้องการให้นิพจน์ด้านซ้าย หรือด้านขวาด้านใดด้านหนึ่งให้เป็นจริงจึงจะทำงานตามที่ต้องการ
- !** (นิเสธ) : ใช้กำหนดเงื่อนไข ในกรณีที่ต้องการให้นิพจน์มีค่าความจริงตรงกันข้าม

## ตัวดำเนินการแบบมีเงื่อนไข (Conditional Operator)

ตัวดำเนินการชนิดนี้ ใช้สำหรับตรวจสอบเงื่อนไขของนิพจน์ว่ามีค่าความจริงเป็นจริง (True) หรือเป็นเท็จ (False) โดยมีรูปแบบการใช้งานดังนี้

||| Expression ? ValueTrue : ValueFalse

โดยที่ Expression คือ นิพจน์เงื่อนไข

ValueTrue คือ ค่าที่ได้กรณีที่เงื่อนไขเป็นจริง

ValueFalse คือ ค่าที่ได้กรณีที่เงื่อนไขเป็นเท็จ

เช่น Output = ( $i < 10$ ) ? 15 : 5;

X = ( $25 < 69$ ) ? 0 : 1;

Y = ( $7 < 5$ ) ? 0 : 1;

## ตัวดำเนินการบอกขนาดชนิดข้อมูล (Sizeof Operator)

ตัวดำเนินการชนิดนี้ใช้สำหรับหาขนาดชนิดของข้อมูลต่าง ๆ ที่ผู้อ่านต้องการทราบ โดยมีรูปแบบการใช้งานดังนี้

||| sizeof(data);

โดยที่ sizeof คือ ตัวดำเนินการบอกขนาดชนิดข้อมูล

data คือ ชนิดข้อมูล หรือตัวแปรที่ต้องการทราบขนาด

## ตัวดำเนินการระดับบิต (Bitwise Operator)

ในบางครั้งผู้อ่านอาจมีความจำเป็นที่จะต้องคำนวณในระดับบิต ซึ่งเป็นหน่วยข้อมูลที่เล็กที่สุด การทำงานในระดับบิตจะช่วยลดภาระการทำงานของ CPU เพราะ CPU จะประมวลผลที่ชนิดข้อมูลที่เป็นบิตเท่านั้น ซึ่งถ้าเป็นค่าน้ำณเป็นข้อมูลชนิดอื่น ๆ จะต้องทำการแปลงข้อมูลให้เป็นบิตก่อน ตัวดำเนินการระดับบิตเป็นตัวดำเนินการเพื่อใช้ในการจัดการข้อมูลในระดับบิต (bit) ซึ่งเป็นตัวดำเนินการพิเศษที่มีอยู่ในภาษา C ซึ่งมีดังนี้

ข้อควรรู้ : บิต (bit) เป็นหน่วยข้อมูลที่เล็กที่สุด โดยข้อมูลหนึ่งบิตจะมีสถานะได้เป็น 2 สถานะคือ 0 (ปิด) หรือ 1 (เปิด) หรือเรียกว่า ก่ออย่างว่า เลขฐานสองนั้นเอง

### Bitwise AND (&), Bitwise OR (|), Bitwise Exclusive OR/XOR (^)

p	q	p & q (Bitwise AND)	p   q (Bitwise OR)	p ^ q (Bitwise XOR)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

- p & q หมายถึง การเทียบบิตแบบ AND ระหว่าง p กับ q
- p | q หมายถึง การเทียบบิตแบบ OR ระหว่าง p กับ q
- p ^ q หมายถึง การเทียบบิตแบบ XOR ระหว่าง p กับ q

### ข้อควรรู้ :

การดำเนินการ AND จะสังเกตเห็นได้ว่าผลลัพธ์จะเป็น 1 ถ้าหากค่าตัวดำเนินการตัวที่หนึ่ง และค่าตัวดำเนินการตัวที่สองเป็น 1 ทั้งคู่ นอกนั้นให้ผลลัพธ์เป็น 0

การดำเนินการ OR จะสังเกตเห็นได้ว่า ผลลัพธ์จะเป็น 1 ถ้าหากค่าตัวดำเนินการตัวที่หนึ่ง หรือค่าตัวดำเนินการตัวที่สองตัวใดตัวหนึ่งเป็น 1 หรือเป็น 1 ทั้งสองตัว นอกนั้นให้ผลลัพธ์เป็น 0

การดำเนินการ XOR จะสังเกตเห็นได้ว่าผลลัพธ์จะเป็น 1 ถ้าหากค่าตัวดำเนินการตัวที่หนึ่ง และค่าตัวดำเนินการตัวที่สองมีค่าต่างกัน และได้ผลลัพธ์เป็น 0 เมื่อตัวดำเนินการทั้งสองมีค่าเหมือนกัน

```

1 #include<stdio.h>
2
3 main()
4 {
5     int x = 9;      // 9 : 0000 1001
6     int y = 7;      // 7 : 0000 0111
7     printf("x(%d) & y(%d) = %d\n", x, y, x & y);
8     /*          9 & 7 : 0000 0001           */
9
10    x = 12;        // 12 : 0000 1100
11    y = 5;        // 5 : 0000 0101
12    printf("x(%d) | y(%d) = %d\n", x, y, x | y);
13    /*          9 | 7 : 0000 1101           */
14
15    x = 10;        // 10 : 0000 1010
16    y = 3;        // 3 : 0000 0011
17    printf("x(%d) ^ y(%d) = %d\n", x, y, x ^ y);
18    /*          9 ^ 7 : 0000 1001           */
19 }
```

## Bitwise Shift Right (>>)

เป็นตัวดำเนินการสำหรับเลื่อนค่าบิตไปทางขวา โดยมีหลักการทำงานดังนี้ กำหนดให้  $x$  เป็นตัวถูกดำเนินการ และ  $y$  เป็นจำนวนการ Shift โดยที่  $x >> y$  หมายถึงเลื่อนบิตในตัวถูกดำเนินการ  $x$  ไปทางขวา  $y$  บิต ผลลัพธ์ที่ได้จากการ Shift Right จะได้เท่ากับ ผลหารของ  $x$  กับ  $2^y$

ก่อน Shift	1	1	0	1	0	1	0	1	0
หลัง Shift	0	1	1	0	1	0	1	0	1

**ข้อควรรู้ :** เนื่องจากไม่มีข้อมูลก่อนหน้าสำหรับใช้ในการเลื่อนบิตมาทางขวา ทำให้ บิตแรกหลังจาก shift จะถูกกำหนดค่าเป็น 0 และเมื่อทำการเลื่อนบิตไปทางขวา บิตสุดท้ายก่อน shift จะถูกตัดทิ้ง (ซ่องที่ไม่ลงสี)

จำนวนการ Shift	เลขที่ใช้หาร	ตัวดำเนินการ Shift
1	2	>>1
2	4	>>2
3	8	>>3
...	$2^y$	>>...
$y$	$2^y$	>>y

## Bitwise Shift Left (<<)

เป็นตัวดำเนินการสำหรับเลื่อนค่าบิตไปทางซ้าย โดยมีหลักการทำงานดังนี้ กำหนดให้  $x$  เป็นตัวถูกดำเนินการ และ  $y$  เป็นจำนวนการ Shift โดยที่  $x << y$  หมายถึงเลื่อนบิตในตัวถูกดำเนินการ  $x$  ไปทางซ้าย  $y$  บิต ผลลัพธ์ที่ได้จากการ Shift Left จะได้เท่ากับ ผลคูณของ  $x$  กับ  $2^y$

ก่อน Shift	1	1	0	1	0	1	0	1	0
หลัง Shift	1	0	1	0	1	0	1	0	0

**ข้อควรรู้ :** เนื่องจากไม่มีข้อมูลด้านหลังสำหรับใช้ในการเลื่อนบิตมาทางซ้าย ทำให้ บิตสุดท้ายหลังจาก shift จะถูกกำหนดค่าเป็น 0 และเมื่อทำการเลื่อนบิตไปทางซ้าย บิตแรกก่อน shift จะถูกตัดทิ้ง (ซ่องที่ไม่ลงสี)

จำนวนการ Shift	เลขที่ใช้คูณ	ตัวดำเนินการ Shift
1	2	<<1
2	4	<<2
3	8	<<3
...	$2^y$	<<...
$y$	$2^y$	<<y

## Bitwise One's Complement (~)

ค่าบิตเริ่มต้น (P)	หลังท่า Bitwise One's Complement (~P)
1	0
0	1

เป็นตัวดำเนินการสำหรับปรับค่าของบิตเป็นค่าตรงข้ามกันๆ คือ ปรับค่าบิต 1 (true) เป็นค่าบิต 0 (false) และ ปรับค่าบิต 0 (false) เป็นค่าบิต 1 (true)

```

1 #include<stdio.h>
2
3 main()
4 {
5     int input = 23;      // 22 : 0001 0111
6     int shift = 2;      // shift left = 2
7     printf("Left Shift : %d << %d = %d\n", input, shift, input << shift);
8     /*                      23 << 2 : 0101 1100                      */
9 }
10    input = 48;        // 48 : 0011 0000
11    shift = 3;        // shift left = 3
12    printf("Right Shift : %d >> %d = %d\n", input, shift, input >> shift);
13    /*                      48 >> 3 : 0000 0110                      */
14
15    input = 5;         // 10 : 0000 0101
16    printf("1'Complement : ~%d = %d", input, ~input);
17    /*                      ~5 : 1111 1010                      */
18 }
```

## ลำดับความสำคัญของตัวดำเนินการ (Precedence)

Category	Operator	Associativity
1. Prefix	( ) [ ] -> . ++ --	Left to Right
2. Unary	+ - ! ~ ++ -- (type) * & sizeof	Right to Left
3. Multiplicative	* / %	Left to Right
4. Additive	+ -	Left to Right
5. Shift	<< >>	Left to Right
6. Relational	< <= > >=	Left to Right
7. Equality	== !=	Left to Right

Category	Operator	Associativity
8. Bitwise AND	&	Left to Right
9. Bitwise XOR	^	Left to Right
10. Bitwise OR		Left to Right
11. Logic AND	&&	Left to Right
12. Logic OR		Left to Right
13. Conditional	? :	Right to Left
14. Assignment	= += -= *= /= %= > >= < <= &= ^=  =	Right to Left
15. Comma	,	Left to Right

**NOTE:** Left to Right หมายถึงลำดับการคิดเช่น  $x = 5 - 4 + 3$ ; คิดจากซ้ายไปขวา “ไม่ใช่ + สำคัญกว่า –”

## การรับข้อมูล (Input Data)

มีคำสั่งที่นิยมใช้ ซึ่งมีอยู่ในsheddeorไฟล์ stdio.h ที่สามารถใช้รับค่า 3 วิธีคือ

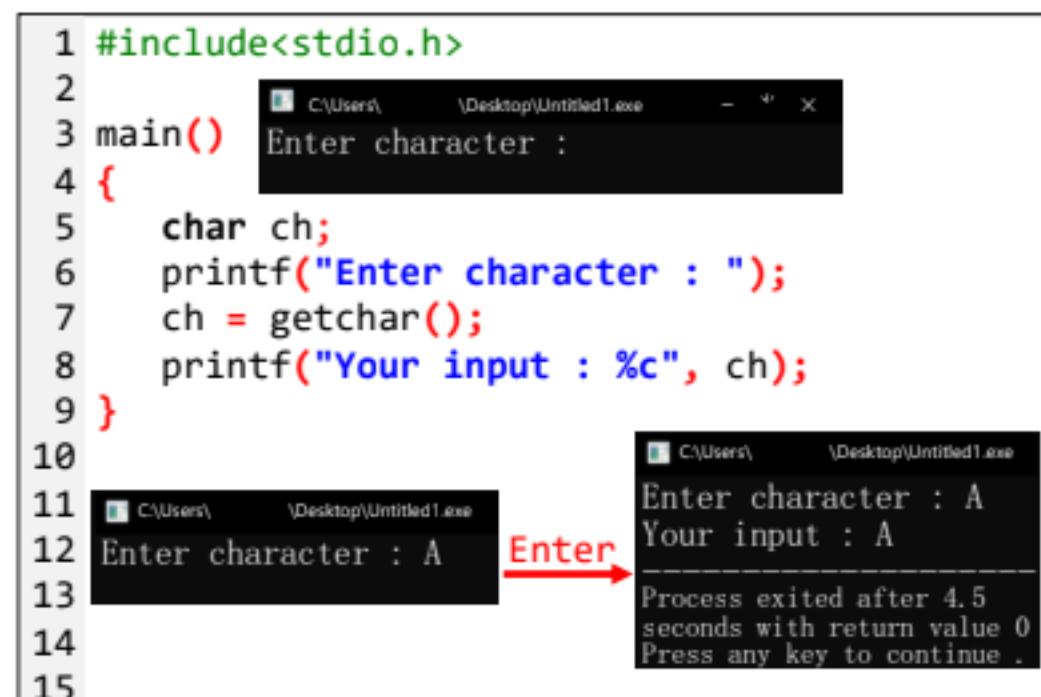
### 1. การรับข้อมูลชนิดอักขระที่ละตัวจากคีย์บอร์ดด้วยคำสั่ง getch() และ getchar()

```
ch = getch();
ch = getchar();
```

**โดยที่** ch เป็นตัวแปรชนิดอักขระ (char) สำหรับรับข้อมูลจากคีย์บอร์ด

คำสั่งทั้งสองเป็นการรับข้อมูลที่ละตัวอักขระ แต่จะแตกต่างกันที่การแสดงผลทางจอภาพโดย getch() จะรับข้อมูลเก็บไว้ที่ตัวแปรทันทีโดยไม่ต้องกดปุ่ม enter แต่ getchar() จะยังไม่รับข้อมูลเก็บไว้ที่ตัวแปรทันที ต้องมีการกดปุ่ม enter ก่อน

```
1 #include<stdio.h>
2 #include<conio.h>
3
4 main()
5 { Enter character :
6     char ch;
7     printf("Enter character : ");
8     ch = getch();
9     printf("Your input : %c", ch);
10 }
11
12 Enter character : Your input : A
13
14 Process exited after 1.146
15 seconds with return value 0
Press any key to continue . . .
```



```
1 #include<stdio.h>
2
3 main()
4 { Enter character :
5     char ch;
6     printf("Enter character : ");
7     ch = getchar();
8     printf("Your input : %c", ch);
9 }
10
11 Enter character : A
12 Enter character : A Enter
13
14
15
```

**ข้อสังเกต :** คำสั่ง getch() เป็นคำสั่งที่อยู่ในsheddeorไฟล์ในไลบรารี conio.h ในคอมไพเลอร์เก่า ๆ หากต้องการใช้ ให้ include เข้ามาด้วย

## 2. การรับข้อมูลนิดข้อความจากคีย์บอร์ดด้วยคำสั่ง gets()

||| `gets(str);`

โดยที่ str เป็นตัวแปรชนิดสายอักขระ (ข้อความ) สำหรับรับข้อมูลจากคีย์บอร์ด

```
1 #include<stdio.h>
2
3 main()
4 {
5     char str[30];
6     printf("Enter string : ");
7     gets(str);
8     printf("Your input : %s", str);
9 }
```

## 3. การรับข้อมูลทุกชนิดจากคีย์บอร์ดด้วยคำสั่ง scanf()

||| `scanf("format_1format_2.....format_n", &var_1, &var_2, ...., &var_n);`

โดยที่ var\_n คือ ตัวแปรที่ต้องการใช้สำหรับเก็บข้อมูล ซึ่งต้องตรงกับ format\_n ที่กำหนดไว้ format\_n คือ ส่วนกำหนดชนิดข้อมูลที่ต้องการรับจากคีย์บอร์ด

ข้อควรรู้ : หากรับข้อมูลนิดข้อความ และข้อมูลนิด Array ด้วยคำสั่ง scanf() ไม่จำเป็นต้องใช้เครื่องหมาย **&** นำหน้า ตัวแปรที่ใช้รับค่าข้อมูล

```
1 #include<stdio.h>
2
3 main()
4 {
5     float weight;
6     int age;
7     char gender, name[30];
8     printf("Enter gender (M or F) : ");
9     scanf("%c", &gender);
10    printf("Enter name : ");
11    scanf("%s", name);
12    printf("Enter age : ");
13    scanf("%d", &age);
14    printf("Enter weight : ");
15    scanf("%f", &weight);
16
17    printf("Name : [%c] %s\n", gender, name);
18    printf("Age : %d\n", age);
19    printf("Weight : %f", weight);
20 }
```

รหัสรูปแบบ	ชนิดข้อมูล
%c	ตัวอักษรหนึ่งตัว
%d	จำนวนเต็มชนิด int
%ld	จำนวนเต็มชนิด long
%e หรือ %E	จำนวนจริงแบบเอ็กซ์โพเนนต์
%f	จำนวนจริง float
%lf	จำนวนจริง double
%g หรือ %G	จำนวนจริง (General format)
%i	จำนวนเต็มชนิด int
%o	เลขฐานแปด
%s	ข้อความ
%u	จำนวนเต็มบวก (Unsigned)
%x หรือ %X	เลขฐานสิบหก (พิมพ์เล็ก/ใหญ่)
และอื่น ๆ เมื่อ mention format ใน printf	

จากตัวอย่าง เมื่อกรอกเพศ (M) แล้วกด Enter ....

จากนั้นใส่ชื่อ (JadeSPK) แล้วกด Enter ....

ใส่อายุ (9) แล้วกด Enter ....

และสุดท้ายกรอกน้ำหนัก (55.5) และกด Enter ...  
ข้อมูลที่กรอกเข้าไปก็จะถูกแสดงผลออกมาดังรูป

```
C:\Users\...\Desktop\Untitled1.exe
Enter gender (M or F) : M
Enter name : JadeSPK
Enter age : 9
Enter weight : 55.5
Name : [M] JadeSPK
Age : 9
Weight : 55.500000
Process exited after 18.2
seconds with return value 0
Press any key to continue . . .
```

```
1 #include<stdio.h>
2 main()
3 {
4     int age;
5     char gender, name[30];
6     printf("Enter name : ");
7     scanf("%s", name);
8     printf("Enter gender (M or F) : ");
9     scanf("%c", &gender);
10    if(gender == '\n');    printf("\n");
11    printf("Enter age : ");
12    scanf("%d", &age);
13 }
```

**ข้อสังเกต :** หากใช้การรับค่าตัวอักษรเดียว (char) ด้วย scanf ต่อจาก scanf อีกที (บรรทัดที่ 10) จะทำให้เกิด bug ขึ้น เนื่องจากอักษรตัวเดียวที่รับค่าไปจะเป็น \n คือการกด Enter เพื่อเสร็จสิ้นการรับค่าด้วย scanf ของค่าสั่งก่อนหน้า (บรรทัดที่ 7) โดยอาจจะแก้ด้วยวิธีตามรูปด้านล่างได้

```
1 #include<stdio.h>
2
3 main()
4 {
5     int age;
6     char gender, name[30];
7     printf("1. Enter name\n");
8     scanf("%s", name);
9     printf("2. Enter gender (M or F)\n");
10    scanf("%c", &gender);
11    if(gender == '\n')    printf("\n");
12    printf("Result : [%c] %s", gender, name);
```

```
C:\Users\...\Desktop\Untitled1.exe
1. Enter name
2. Enter gender (M or F)
JadeSPK
M
Result : [M] JadeSPK
Process exited after 8.799
seconds with return value 0
Press any key to continue . . .
```

**ข้อสังเกต :** การรับด้วย scanf สามารถรับหลายค่า ใส่ในแต่ละตัวแปรได้ภายในค่าสั่งเดียวเพียงกำหนด format และตัวแปรให้ตรงกัน เมื่อ input เข้าไป สามารถใช้ enter หรือเว้นวรรคในการส่งสิ้นสุดการรับค่าในแต่ละค่าได้

```
1 #include<stdio.h>
2
3 main()
4 {
5     int a, b;
6     scanf("%d%d", &a, &b); printf("1. %d %d\n", a, b);
7     scanf("%d-%d", &a, &b); printf("2. %d %d\n", a, b);
8     scanf("%d, %d", &a, &b); printf("3. %d %d\n", a, b);
9     scanf("%da%d", &a, &b); printf("4. %d %d" , a, b);
10 }
```

```
C:\Users\...\Desktop\Untitled1.exe
15 20
1. 15 20
20-90
2. 20 90
50,50
3. 50 50
10a10
4. 10 10
Process exited after 25.69
seconds with return value 0
Press any key to continue . . .
```

```
1 #include<stdio.h>
2
3 main()
4 {
5     int n;
6     scanf("%*d %d", &n);
7     printf("Output : %d", n);
8 }
```

### การกำหนดลำดับการรับข้อมูลของคำสั่ง scanf()

สามารถกำหนดลำดับการรับข้อมูลโดยใช้เครื่องหมาย \* ได้ หากลำดับของข้อมูลที่รับเข้ามาไม่เครื่องหมาย \* ข้อมูลที่รับเข้ามาในลำดับดังกล่าวจะไม่ถูกจัดเก็บลงตัวแปร ตัวอย่างต่อไปนี้

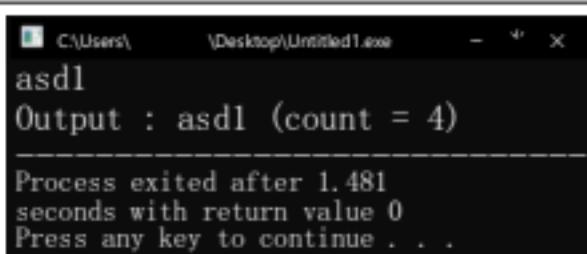
## การกำหนดจำนวนตัวอักษรในการรับค่าของข้อมูลนิดข้อความของคำสั่ง scanf()

สามารถกำหนดความยาวข้อความที่ต้องการรับสูงสุดได้โดยการใส่ตัวเลขหน้า `r` ใน `%r` ในส่วนของ `format` เช่น `%4s` เป็นต้น

```
1 #include<stdio.h>
2
3 main()
4 {
5     char str[20];
6     scanf("%4s", &str);
7     printf("Output : %s", str);
8 }
```

## การนับจำนวนตัวอักษรที่รับค่าของข้อมูลด้วยคำสั่ง scanf()

```
1 #include<stdio.h>
2
3 main()
4 {
5     int count;
6     char str[20];
7     scanf("%s%n", str, &count);
8     printf("Output : %s (count = %d)", str, count);
9 }
```



สามารถนับความยาวของข้อความที่รับเข้ามา เพื่อนำไปใช้กำหนดเงื่อนไขต่าง ๆ หรือทำอย่างอื่นได้

**ข้อสังเกต :** หากข้อมูลเป็นจำนวนเต็มก็สามารถนับจำนวนตัวอักษรได้เช่นกัน เช่น ใส่ 123 count จะนับได้ 3 ตัว

## Regular Expression

Regular expression หรือ Regex คือรูปแบบ หรือกลุ่มคำ (pattern) ที่กำหนดขึ้นเพื่อค้นหาข้อความ หรือตัวอักษรต่าง ๆ ว่าตรงตามเงื่อนไขของ pattern ที่กำหนดไว้หรือไม่ ซึ่งมักจะมีอยู่ในทุก ๆ ภาษาคอมพิวเตอร์ และมีความแตกต่างกันเพียงเล็กน้อย การใช้ Regex ส่วนมากในเล่มนี้จะกล่าวถึงการกำหนดรูปแบบการรับข้อมูลเอง (format\_n) สามารถทำได้โดยการใช้ Regular Expression เข้ามาช่วยซึ่งสามารถศึกษาเพิ่มเติมได้จาก <https://regexr.com> หรือ <https://www.tamemo.com/post/111/how-to-regular-expression> โดยภาษา C/C++ สามารถใช้ได้ 2 รูปแบบดังตารางด้านล่าง

รูปแบบ	ความหมาย
[ ]	เรียกว่า bracket expression หมายถึง กลุ่มของตัวอักษรในนี้เท่านั้นที่ต้องการ <u>หากเจอตัวอักษรนอกกลุ่มนี้จะทำการหยุดรับค่าทันทีสำหรับภาษา C/C++</u> สามารถใช้ – ช่วยในการนับตัวอักษรที่ต้องการเป็น range ได้ <ul style="list-style-type: none"> <li>• [abc] แมทช์กับ 'a', 'b', 'c'</li> <li>• [abcx-z] หรือ [a-cx-z] แมทช์กับ 'a', 'b', 'c', 'x', 'y', 'z'</li> <li>• [A-Za-z] แมทช์กับ 'a' ถึง 'z' และแมทช์กับอักษรตัวใหญ่ด้วย 'A' ถึง 'Z'</li> <li>* <u>[ก-݂]</u> สำหรับภาษาไทยซึ่งมีทั้ง พยัญชนะ สระ ตัวเลขไทยมากมาย ให้เริ่มด้วย 'ກ' ถึง '݂' จะได้ครบถ้วน</li> </ul>
[^ ]	เหมือนแคส [] แต่เปลี่ยนเป็น <u>ไม่เจออักษรในนี้แทน หากเจอตัวอักษรในกลุ่มนี้จะทำการหยุดรับค่าทันทีสำหรับภาษา C/C++</u>

เช่น	%[^\n] รับค่าเรียกจนกว่าเจอขึ้นบรรทัดใหม่	→ input : asduh[enter]	output : asduh
	%[^ab] รับค่าเรียกจนกว่าเจอ a หรือ b	→ input : eieiaty	output : eiei
	%[^a-c] รับค่าเรียกจนกว่าเจอ a, b หรือ c	→ input : wowcoasd	output : wow
	%[ab] รับค่าจะสั้นสุดเมื่อไม่พบอักษร a หรือ b	→ input : eieiatyAerb	output :
	%[A-C] รับค่าจะสั้นสุดเมื่อไม่พบอักษร A, B หรือ C	→ input : ABBCAaAC	output : ABBCA
	%[A-C0-6] รับค่าจะสั้นสุดเมื่อไม่พบอักษร A, B หรือ C และ 0-6	→ input : A0vh0Ba6z	output : A0

**ข้อสังเกต :** รหัสค่าสั่งพิเศษแต่ละตัวจะมีรหัสแอสกี (ASCII Code) เช่น new line หรือขึ้นบรรทัดใหม่ (ASCII Code = 10, LF), carriage return หรือกลับไปยังตัวแรกของบรรทัด (ASCII Code = 13, CR)

**ข้อสังเกต :** ในบางเครื่องรหัสค่าสั่งพิเศษในการขึ้นบรรทัดใหม่ (new line) ไม่ใช่ \n แต่จะมี \r เพิ่มเข้ามาเป็น \n\r

## ควบคุมทิศทางการทำงานโปรแกรมด้วยคำสั่งตัดสินใจ (Decision Control Statement)

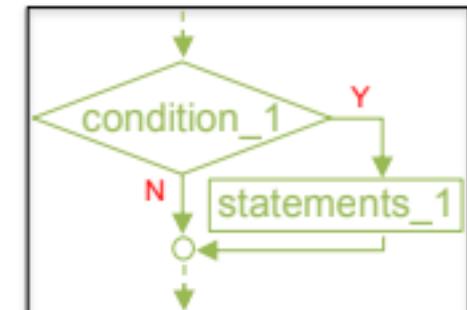
### กำหนดให้

**condition** เป็นตัวแปร หรือนิพจน์ที่เป็นเงื่อนไขของคำสั่ง  
**condition\_n** เป็นตัวแปร หรือนิพจน์ที่เป็นเงื่อนไขของคำสั่งที่ n ใช้กำหนดการตัดสินใจของโปรแกรม  
**statements\_1** เป็นชุดคำสั่งที่ต้องทำงานเมื่อเงื่อนไขที่ 1 เป็นจริง  
**statements\_2** เป็นชุดคำสั่งที่ต้องทำงานเมื่อเงื่อนไขที่ 2 เป็นจริง  
**statements\_n** เป็นชุดคำสั่งที่ต้องทำงานเมื่อเงื่อนไขที่ n เป็นจริง  
**Nstatements** เป็นชุดคำสั่งที่ต้องทำงานเมื่อเงื่อนไขที่กำหนดทั้งหมดเป็นเท็จ

### คำสั่งตัดสินใจแบบเลือกทำ หรือไม่ทำด้วยคำสั่ง if

```
if (condition_1)
{
  statements_1;
}
```

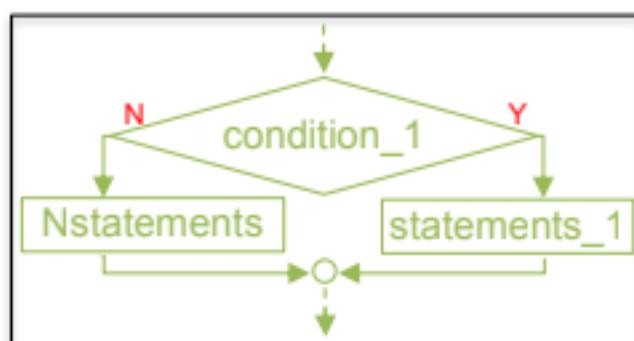
คำสั่ง if เป็นคำสั่งที่เราใช้กำหนดให้โปรแกรมตัดสินใจทำ หรือไม่ทำสิ่งใดสิ่งหนึ่ง โดยตรวจสอบเงื่อนไขที่กำหนดว่าเป็นจริง หรือเท็จ ถ้าเงื่อนไขที่กำหนดให้เป็นจริง (true) โปรแกรมจะทำงานที่ชุดคำสั่งที่อยู่ภายใต้คำสั่ง if แต่ถ้าเงื่อนไขที่กำหนดไว้เป็นเท็จ (false) โปรแกรมจะข้ามไปทำงานที่คำสั่งต่อไปทันที



**ข้อควรรู้ :** การกำหนดเงื่อนไขตามปกติ คือใช้นิพจน์มาทำการเปรียบเทียบค่ากันโดยใช้ Operator >, <, >=, <=, !=, == เช่น if (a>b) การดำเนินการเปรียบเทียบกันตามปกตินี้จะได้ผลลัพธ์สองแบบ คือเท็จ(False) จะได้ค่าเป็น 0 หาก และจริง (True) จะได้ค่าเป็น 1

**ข้อควรรู้ :** เงื่อนไขที่นิพจน์ไม่ได้กำหนดตามปกติ เช่น if (a = -10) ซึ่ง a = -10 เป็นการกำหนดค่าให้ตัวแปร a เป็น -10 → if (-10) หากเป็นเช่นนี้ค่าอื่น ๆ ที่ไม่ใช่ 0 จะถือว่าเป็นจริง (True) หมวด เช่น if (8), if (-9) จะถือว่าเป็นจริง และเข้าเงื่อนไขทำ statement ใน if ต่อ

### คำสั่งตัดสินใจแบบสองทางเลือกด้วยคำสั่ง if...else



คำสั่ง if...else เป็นคำสั่งที่เราใช้กำหนดให้โปรแกรมตัดสินใจเลือกทำคำสั่งอย่างใดอย่างหนึ่ง จาก 2 ทางเลือก โดยตรวจสอบเงื่อนไขที่กำหนดว่าเป็นจริง หรือเท็จ ถ้าเงื่อนไขที่กำหนดให้เป็นจริง (true) โปรแกรมจะทำงานที่ชุดคำสั่งที่อยู่ภายใต้ if แต่ถ้าเงื่อนไขที่กำหนดไว้เป็นเท็จ (false) โปรแกรมจะทำงานที่ชุดคำสั่งที่อยู่ภายใต้ else

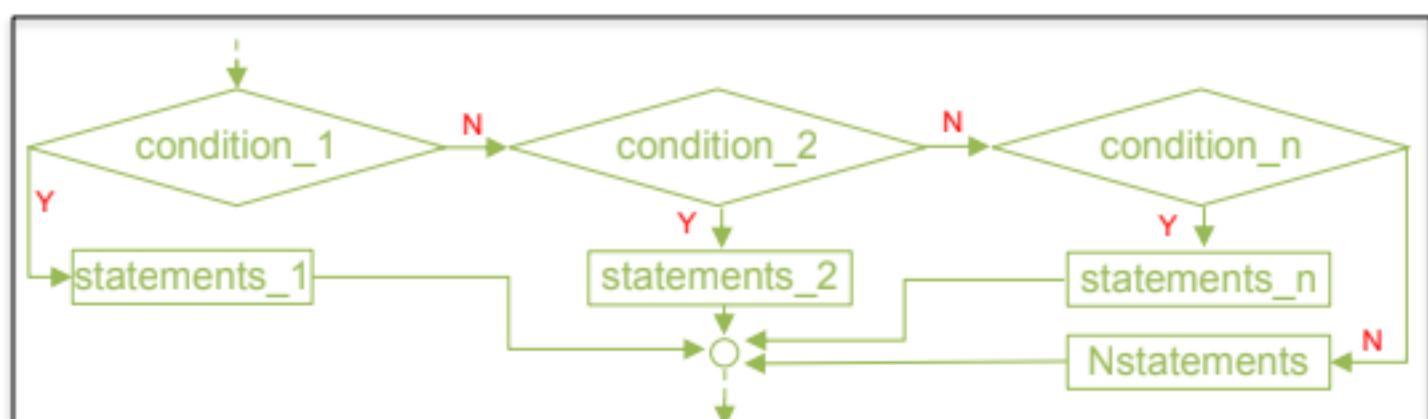
```
if (condition_1)
{
  statements_1;
}
else
{
  Nstatements;
}
```

### คำสั่งตัดสินใจแบบหลายทางเลือกด้วยคำสั่ง if...else if

```
if (condition_1)
{
  statements_1;
}
else if (condition_2)
{
  statements_2;
}
else if (condition_n)
{
  statements_n;
}
else
{
  Nstatements;
}
```

คำสั่ง if...else if เป็นคำสั่งที่เราใช้กำหนดให้โปรแกรมตัดสินใจเลือกทางหนึ่งจากทางเลือกที่มีมากกว่า 2 ทาง และแต่ละทางเลือกจะมีการกำหนดเงื่อนไขของแต่ละทางเลือกไว้ด้วย โดยโปรแกรมจะตรวจสอบเงื่อนไขที่ทางเลือกใหม่มีเงื่อนไขเป็นจริง (true) ก็จะทำงานที่ชุดคำสั่งภายในทางเลือกนั้น โดยไม่พิจารณาทางเลือกอื่นที่ไม่ได้ตรวจสอบอีก ในกรณีที่เงื่อนไขเป็นเท็จ (false) ให้ตรวจสอบเงื่อนไขถัดไป ในกรณีที่เงื่อนไขทั้งหมดเป็นเท็จให้โปรแกรมทำงานที่ชุดคำสั่ง else

**ข้อสังเกต :** if...else if ไม่จำเป็นต้องมี else มีแค่ if else-if else-if ..... else-if ก็ได้



**ข้อสังเกต :** เงื่อนไขที่ใช้เปรียบเทียบ หากใช้การเปรียบเทียบว่าเท่ากันหรือไม่ ให้ใช้เครื่องหมาย == ในการเปรียบเทียบ อย่าลืมสนับสนุนเครื่องหมาย = (การกำหนดค่า) หากใช้เครื่องหมาย = ดังในตัวอย่างเมื่อโปรแกรมทำงานถึงส่วน if คอมpileแล้วจะพิจารณาเงื่อนไข x = 1 แต่ x = 1 เป็นการกำหนดให้ x มีค่าเป็น 1 (จริง) ดังนั้นจะได้เงื่อนไข if (x) ซึ่งก็คือ if (1) หรือ if (true) ดังนั้นโปรแกรมจะเข้าเงื่อนไข if โดยไม่สนใจว่า x จะมีค่าอะไรมาก่อน

```
1 #include<stdio.h>
2
3 main()
4 {
5     int x = 0;
6     if(x = 1) printf("TRUE : x = %d", x);
7     else      printf("FALSE: x = %d", x);
8 }
```

```
1 #include<stdio.h>
2
3 main()
4 {
5     int x = 1;
6     if(x == 1) printf("TRUE : x = %d", x);
7     else      printf("FALSE: x = %d", x);
8     printf("\nif x == 0(false) print this");
9 }
```

**ข้อควรรู้ :** หาก statement ที่ตามหลัง if, else if หรือ else มีเพียงแค่ 1 statement ไม่จำเป็นต้องใส่ {} คลุม statement ก็ได้ **แต่ควรใส่ไว้** เพื่อกันลืมในการถือที่มีหลาย statement ตามหลัง if, else if หรือ else (ถ้าลืมใส่ปีกการตัวเงื่อนไขจะครอบคลุมเพียง statement เดียว)

จากตัวอย่างจะเห็นว่า x มีค่าเท่ากับ 1 จึงทำเงื่อนไขใน if ก็คือแสดงผล TRUE : x = 1 ออกมาและโปรแกรมก็จะหลุดออกจาก if-else ท่าค่าสั่ง printf("\nif x == 0(false) print this.."); ต่อ ซึ่งค่าสั่งนี้ไม่อยู่ใน if-else วิธีแก้ คือการใส่ปีกการคลุมค่าสั่งหลัง else ทั้งหมด

### คำสั่งตัดสินใจแบบหลายทางเลือกด้วยคำสั่ง switch-case

```
switch (condition)
{
    case constant_1 : statements_1;
                      break;
    case constant_2 : statements_2;
                      break;
    case constant_n : statements_n;
                      break;
    default         : Nstatements;
}
```

คำสั่ง switch-case เป็นคำสั่งตัดสินใจที่มีการทำงานเหมือนกับคำสั่ง if...else if คือเลือกทางใดทางหนึ่งจากทางเลือกที่มากกว่า 2 ทาง ในแต่ละทางเลือกจะมีการกำหนดเงื่อนไขของแต่ละทาง โดยตรวจสอบเงื่อนไขแต่ละทางเลือก หากพบว่าทางเลือกไหนมีเงื่อนไขเป็นจริง (true) ก็จะทำชุดคำสั่งภายในทางเลือกนั้น และไม่พิจารณาทางเลือกที่ยังไม่ได้ตรวจสอบอีก ในกรณีที่เงื่อนไขเป็นเท็จ (false) ให้ตรวจสอบเงื่อนไขถัดไป ในกรณีที่เงื่อนไขทั้งหมดเป็นเท็จ ให้โปรแกรมทำงานที่ชุดคำสั่ง default : สำหรับ flowchart จะเหมือนกับคำสั่งตัดสินใจแบบหลายทางเลือกด้วยคำสั่ง if...else if

<b>โดยที่</b>	<b>condition</b>	เป็นตัวแปร หรือนิพจน์ที่เป็นเงื่อนไขของคำสั่ง
	<b>constant_n</b>	เป็น <b>ตัวเลขที่เป็นค่าคงที่</b> ใช้ตรวจสอบกับตัวแปร หรือนิพจน์เงื่อนไข
	<b>statements_n</b>	เป็นชุดคำสั่งที่ต้องทำงานเมื่อตัวแปรหรือนิพจน์ที่ ก เป็นจริง
	<b>Nstatements</b>	เป็นชุดคำสั่งที่ต้องทำงานเมื่อตัวแปรหรือนิพจน์ทั้งหมดเป็นเท็จ

```
1 #include<stdio.h>
2
3 main()
4 {
5     int x = 65;
6     switch(x)
7     {
8         case 0 : printf("CASE 0\n");
9         case 1 : printf("CASE 1\n");
10        case 'A' : printf("CASE A\n");
11        case 'B' : printf("CASE B\n"); break;
12        case 2 : printf("CASE 2\n");
13        default : printf("Default\n");
14    }
15 }
```

**ข้อสังเกต :** ต้องใช้ค่าคงที่ในการแยกกรณีเท่านั้น จากบรรทัดที่ 10 และ 11 จะเห็นว่า 'A' คือการนำรหัสแสกนของ A ซึ่งมีค่าเท่ากับ 65 มาเป็นเงื่อนไขในการเช็ค และ 'B' คือการนำรหัสแสกนของ B ซึ่งมีค่าเท่ากับ 66 มาเป็นเงื่อนไขในการเช็ค

**ข้อสังเกต :** ถ้าไม่ได้ใส่คำสั่ง break เมื่อ switch-case เจอเงื่อนไขใน case ใดที่เป็นจริงแล้ว ก็จะทำ statement ใน case นั้น แล้วทำ statement ใน case ตัวล่างต่อไปเรื่อย ๆ จนกว่าจะเจอคำสั่ง break หากในบรรทัดที่ 11 ไม่ได้ใส่ คำสั่ง break ก็จะแสดงผล CASE 2 ขึ้นบรรทัดใหม่ Default ขึ้นบรรทัดใหม่ ต่อจาก CASE B ขึ้นบรรทัดใหม่

จากตัวอย่าง การเขียนโปรแกรมด้วยภาษา C  
 เกรด และกำหนดให้คะแนนมีค่าเป็น 75 เมื่อถึงส่วนเช็คเงื่อนไข จะทำการ เช็คเงื่อนไขใน if ก่อน (คะแนนมากกว่า หรือเท่ากับ 80 หรือไม่) ปรากฏว่าเป็น เท็จ จึงเช็คเงื่อนไข else if ต่อไป คือ คะแนนมากกว่า หรือเท่ากับ 75 แต่ น้อยกว่า 80 หรือไม่ หากให้เงื่อนไขนี้ เป็นจริง และทำค่าสั่งหลังเงื่อนไขนั้น เพียง 1 statement (แสดง B+)

**ข้อควรระวัง :** การกำหนดเงื่อนไขเป็น

ช่วงไม่สามารถกำหนด if( $75 < \text{score} < 80$ ) ได้ ต้องใช้ตัวดำเนินการทางตรรกศาสตร์ช่วย if(score > 75 && score < 80)

```

1 #include<stdio.h>
2 main()
3 {
4     int score = 75;
5     if (score >= 80) printf("A");
6     else if (score >= 75 && score < 80); printf("B+");
7     else if (score >= 70 && score < 75); printf("B");
8     else if (score >= 65 && score < 70); printf("C+");
9     else if (score >= 60 && score < 65); printf("C");
10    else if (score >= 55 && score < 60); printf("D+");
11    else if (score >= 50 && score < 55); printf("D");
12    else printf("F");
13 }

```

## ควบคุมการทำงานของโปรแกรมด้วยคำสั่งวนลูป (Repetition Control Statement)

วนลูปการทำงานด้วยจำนวนรอบที่แน่นอนด้วยคำสั่ง for

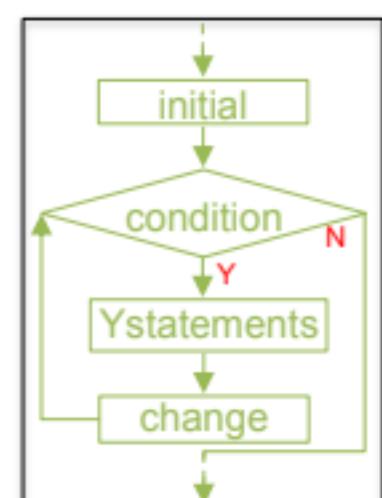
```

for (initial; condition; change)
{
    Ystatements;
}

```

คำสั่ง for เป็นคำสั่งวนซ้ำการทำงานเดิม ๆ ด้วย จำนวนรอบที่แน่นอน โดยที่โปรแกรมจะตรวจสอบ เงื่อนไขก่อนการทำงานทุกรอบ ถ้าเงื่อนไขเป็นจริงให้ ทำงานที่ชุดคำสั่งภายในลูปต่อไป เมื่อทำงานเสร็จจะ เพิ่ม หรือลดค่าตัวแปร และตรวจสอบเงื่อนไขใหม้อีกครั้ง ถ้าเงื่อนไขเป็นเท็จให้โปรแกรมออกจากลูปการทำงาน ไปทำงานที่คำสั่งถัดไปทันที

<b>โดยที่</b>	<b>initial</b>	เป็นส่วนกำหนดค่าเริ่มต้นของตัวแปรที่ใช้กำหนดเงื่อนไข
	<b>condition</b>	เป็นส่วนกำหนดเงื่อนไขการวนลูป สามารถใส่หลายเงื่อนไขได้ โดย ใช้ตัวดำเนินการแบบต่างๆ เช่น ช่วยกำหนดเงื่อนไขให้เป็นจริงหรือเท็จ
	<b>change</b>	เป็นส่วนการเปลี่ยนแปลงของตัวแปรหลังทำงานจนจบในแต่ละรอบ
	<b>Ystatements</b>	เป็นชุดคำสั่งที่ต้องทำเมื่อเงื่อนไขเป็นจริง



```

1 #include<stdio.h>
2 main()
3 {
4     int i = 0;
5     for(; i<=2; ) {
6         printf("%d", i);
7         i++;
8     }
9 }

```

= **for(; i<=2; ) {**  
**printf("%d", i++);** = **for(; i!=3; ) {**  
**printf("%d", i++);**

**ข้อสังเกต :** เงื่อนไขสามารถกำหนดได้หลายแบบแต่ผลลัพธ์ ที่ออกมายังไงก็ได้เนื่องจาก ส่วนรับในส่วนของ initial และ change ไม่จำเป็นต้องมีเสนอไป หรือสามารถกำหนดให้มีมากกว่า 1 ถ้า ในส่วนของ condition ก็สามารถมีมากกว่า 1 ได้ เช่นกัน โดยอาศัยตัวดำเนินการต่าง ๆ เช่นตามรูปด้านล่างนี้ แต่ ผู้อ่านควรเขียนให้ผู้อื่นเข้าใจโปรแกรมได้ง่าย

```

1 #include<stdio.h>
2 main()
3 {
4     for(int i=0; i<=2; i++) {
5         printf("%d", i);
6     }
7 }

```

```

1 #include<stdio.h>
2 main()
3 {
4     int x = 1, i;
5     for(i=0, x=2; i<2 || i==2; i++) {
6         printf("%d", i);
7     }
8 }

```

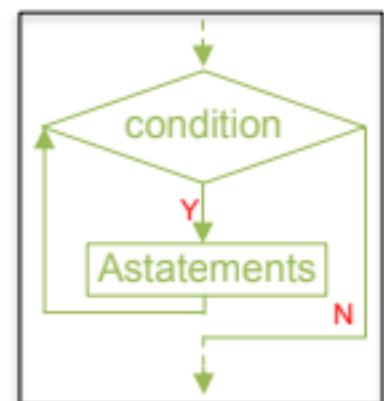
**ข้อควรระวัง :** บางคอมไไฟเลอร์เก่า ๆ บางตัวไม่สามารถประมวลผลคำสั่งในส่วน initial ได้เนื่องจากในตัวอย่างด้านข้าง แต่ถ้าประมวลผลได้ตัวแปรที่ถูกประกาศนั้นจะเป็นตัวแปรประเภท Local อยู่ภายใต้ลูป for ถ้าออกจาก ลูปตัวแปร i จะถูกลบออกจากหน่วยความจำเมื่อไม่ได้ประมวลผลไว้

## วนลูปการทำงานเมื่อเงื่อนไขเป็นจริงด้วยคำสั่ง while

```
while (condition)
{
    Astatements;
}
```

คำสั่ง while เป็นคำสั่งวนซ้ำการทำงานเดิมๆ ของโปรแกรม โดยโปรแกรมตรวจสอบเงื่อนไขก่อนการทำงานทุกครั้ง ถ้าเงื่อนไขที่กำหนดเป็นจริง โปรแกรมจะทำงานที่ชุดคำสั่งภายในลูป เมื่อโปรแกรมทำงานเสร็จ จะตรวจสอบเงื่อนไขใหม่อีกครั้ง ถ้าเงื่อนไขที่กำหนดเป็นเท็จ โปรแกรมจะออกจากลูปการทำงานไปทำงาน ที่คำสั่งถัดไปทันที

โดยที่ condition เป็นเงื่อนไขที่กำหนดให้ตรวจสอบก่อนทำงานภายในลูปทุกครั้ง  
Astatements เป็นชุดคำสั่งที่ต้องทำเมื่อเงื่อนไขเป็นจริง (หลังตรวจสอบเงื่อนไข)

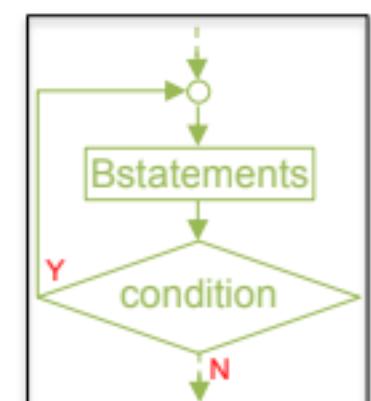


## วนลูปการทำงานอย่างน้อย 1 ครั้งด้วยคำสั่ง do...while

```
do
{
    Bstatements;
}while (condition);
```

คำสั่ง do..while เป็นคำสั่งวนซ้ำการทำงานเดิม ๆ โดยโปรแกรมจะทำงานชุดคำสั่งภายในลูปก่อน 1 รอบ จึงตรวจสอบเงื่อนไขที่กำหนด ถ้าเงื่อนไขที่กำหนดเป็นจริง จะกลับไปทำงานชุดคำสั่งภายในลูปอีกครั้ง และตรวจสอบเงื่อนไขที่กำหนดอีกครั้ง ถ้าเงื่อนไขที่กำหนดเป็นเท็จ โปรแกรมจะออกจากลูปการทำงานไป ทำงานที่คำสั่งถัดไปทันที

โดยที่ condition เป็นเงื่อนไขที่กำหนดให้ตรวจสอบหลังการทำงานภายในลูปทุกครั้ง  
Bstatements เป็นชุดคำสั่งที่ทำงานก่อนตรวจสอบเงื่อนไข



ข้อควรรู้ : ในกรณีที่จำนวนคำสั่งภายในลูปมากเกินไป ไม่จำเป็นต้องใส่เครื่องหมาย {} ก็ได้ แต่แนะนำให้ใส่เครื่องหมายปีกกา {} ทุกครั้ง

### คำสั่ง break และ continue

<pre> 1 #include&lt;stdio.h&gt; 2 3 main() 4 { 5     for(int i=0; i&lt;3; i++) { 6         for(int j=0; j&lt;9; j++) { 7             printf("i = %d, j = %d\n", i, j); 8             if(j == 2) break; 9         } 10    } 11 } 12   </pre>	<p>C:\Users\...\Desktop\Untitled1.exe</p> <pre> i = 0, j = 0 i = 0, j = 1 i = 0, j = 2 i = 1, j = 0 i = 1, j = 1 i = 1, j = 2 i = 2, j = 0 i = 2, j = 1 i = 2, j = 2   </pre> <p>Process exited after 0.0201 seconds with return value 0 Press any key to continue .</p>
---	--

จากรูปด้านบนเมื่อคอมไพล์แล้วทำงานถึงคำสั่ง break คอมไпал์จะทำการออกจากคำสั่งวนลูปปั๊บปั๊บที่กำลังทำงานอยู่ (เลิกทำคำสั่งวนลูป for ที่มีตัวแปร j กำหนด ซึ่งโดยปกติการที่ลูป j จะจบได้ j ต้องมีค่า  $\geq 9$  แต่ j มีค่าเท่ากับ 2 ตรงเงื่อนไข break ที่กำหนดไว้จึงออก

สำหรับในรูปด้านล่างเมื่อคอมไпал์แล้วทำงานถึงคำสั่ง continue คอมไpal์จะทำการวนลูปไปรอบนั้น (คำสั่งวนลูป for ที่กำลังทำงานในรอบที่ตัวแปร j มีค่าเท่ากับ 2 จะถูกข้ามไปยังรอบถัดไป (คือรอบที่ 4) หากเจอคำสั่ง continue)

<pre> 1 #include&lt;stdio.h&gt; 2 3 main() 4 { 5     for(int i=0; i&lt;3; i++) { 6         for(int j=0; j&lt;4; j++) { 7             if(j == 2) continue; 8             printf("i = %d, j = %d\n", i, j); 9         } 10    } 11 } 12   </pre>	<p>C:\Users\...\Desktop\Untitled1.exe</p> <pre> i = 0, j = 0 i = 0, j = 1 i = 0, j = 3 i = 1, j = 0 i = 1, j = 1 i = 1, j = 3 i = 2, j = 0 i = 2, j = 1 i = 2, j = 3   </pre> <p>Process exited after 0.0017 seconds with return value 0 Press any key to continue .</p>
--	--

ข้อควรระวัง : statement for และ while ไม่มีเครื่องหมาย ; ปิดท้าย หากใส่เครื่องหมาย ; จะเทียบเท่ากับดังรูป

```

1 #include<stdio.h>
2 main()
3 {
4     int i = 0;
5     for(int x=0; x<10; x++);
6     {
7         printf("Caution!");
8     }
9 }

```

ข้อควรระวัง : การกำหนดเงื่อนไขของลูป อาจทำให้เกิดลูปนั้นติด คือลูปที่วนไม่รู้จบเนื่องจากไม่ว่าลูปจะผ่านไปกี่รอบก็จะไม่มีวันผิดเงื่อนไข ที่กำหนด ไว้เหมือนตัวอย่างทางด้านข่าย หรือตามตัวอย่างต่อไปนี้ เช่น while(true) , while(1) , for(int x = 0; x >=0; x++) เป็นต้น

ตัวอย่างการใช้ลูปแต่ละประเภทในการแสดงผลตัวเลขตั้งแต่ 1 ถึง 6 โดยกำหนดเงื่อนไข และรูปแบบที่ใช้ต่างกันดังนี้

```

1 #include<stdio.h>
2 main()
3 {
4     int i = 0;
5     for(i=1; i<=6; i++)
6     {
7         printf("%d", i);
8     }
9 }
10

```

### อธิบายลูป for อย่างละเอียด

`for(i=0; i<6; i++)` เมื่อเริ่มต้นท่าลูป for จะทำงานในส่วนของ initial ก่อน โดยในที่นี้คือการกำหนดให้ i มีค่าเป็น 0 หลังจากนั้นจะเช็คเงื่อนไขในส่วน condition ว่าเป็นจริง หรือไม่ ถ้าเป็นจริงให้ทำการ statement ในลูป และเมื่อจบลูปจึงทำในส่วน change ต่อ

i = 0 loop#1	for(i=0; i<6; i++) { printf("%d", i); }	for(i=0; i<6; i++) { printf("%d", i); }	for(i=0; i<6; i++) { printf("%d", i); }
i = 1 loop#2	for(i=0; i<6; i++) { printf("%d", i); }	for(i=0; i<6; i++) { printf("%d", i); }	for(i=0; i<6; i++) { printf("%d", i); }
		⋮	
i = 5 loop#6	for(i=0; i<6; i++) { printf("%d", i); }	for(i=0; i<6; i++) { printf("%d", i); }	for(i=0; i<6; i++) { printf("%d", i); }
i = 6	for(i=0; i<6; i++) { printf("%d", i); }	จะท่าอย่างนี้ข้าไปข้ามจุดกว่าจะผิดเงื่อนไข (ลูปรอบแรก i=0 ลูปรอบ 2 i=1 .... ลูปรอบ 6 i=5 และเป็นลูปรอบสุดท้ายเนื่องจากก่อนจะขึ้นลูปรอบที่ 7 จะต้องเช็คเงื่อนไขก่อนเข้า แล้ว i มีค่าเป็น 6 ซึ่งจะผิดเงื่อนไข ผลคือจบลูป	

## อาร์เรย์ (Array)

ตัวแปรอาร์เรย์ เปรียบเสมือนการนำตัวแปรมาเรียงต่อกันหลาย ๆ ตัว โดยที่ทุกตัวมีชนิดข้อมูลแบบเดียวกัน มีชื่อ ตัวแปรเดียวกัน อาร์เรย์สามารถอ้างถึงข้อมูลแต่ละตัวที่เรียงต่อกันเป็นลำดับการโดยใช้ค่า索引 ซึ่งค่า索index นั้นตั้งกล่าวจะเรียกว่า อินเด็กซ์ (Index)

ตัวแปรอาร์เรย์ที่มีให้ใช้งานในภาษา C นั้นสามารถแยกได้ 2 แบบคือ ตัวแปรอาร์เรย์ 1 มิติ และตัวแปรอาร์เรย์ หลายมิติ (มากกว่า 1) ซึ่งในที่นี้จะยกล่าวถึงตัวแปรอาร์เรย์ 1 มิติ ตัวแปรอาร์เรย์ 2 มิติ และตัวแปรอาร์เรย์ 3 มิติเท่านั้น

### ตัวแปรอาร์เรย์ 1 มิติ (One Dimension Array)

เปรียบได้กับการนำตัวแปรมาเรียงต่อกันหลาย ๆ ตัวในลักษณะเป็น列ของข้อมูล ซึ่งสามารถจัดล่องตัวอย่างตัวแปรอาร์เรย์ 1 มิติ ข้อตัวแปร intEx1 เป็นตัวแปรชนิดจำนวนเต็มที่สามารถเก็บข้อมูลจำนวนเต็มได้ 6 ตัว ยกตัวอย่างดังนี้

Index	0	1	2	3	4	5
Value	5	8	1	3	9	2

- จากตัวอย่างจะเห็นได้ว่า ตัวแปรอาร์เรย์สามารถเก็บข้อมูลได้ 6 ตัว โดยที่
- |                  |   |
|------------------|---|
| ตัวแปรตัวแรก     | ค่า索index ที่ 0 มีค่าเท่ากับ 5 ซึ่งสามารถเขียนได้เป็น intEx1[0] = 5 |
| ตัวแปรตัวที่ 2   | ค่า索index ที่ 1 มีค่าเท่ากับ 8 ซึ่งสามารถเขียนได้เป็น intEx1[1] = 8 |
| ตัวแปรตัวที่ 3   | ค่า索index ที่ 2 มีค่าเท่ากับ 1 ซึ่งสามารถเขียนได้เป็น intEx1[2] = 1 |
| ตัวแปรตัวที่ 4   | ค่า索index ที่ 3 มีค่าเท่ากับ 3 ซึ่งสามารถเขียนได้เป็น intEx1[3] = 3 |
| ตัวแปรตัวที่ 5   | ค่า索index ที่ 4 มีค่าเท่ากับ 9 ซึ่งสามารถเขียนได้เป็น intEx1[4] = 9 |
| ตัวแปรตัวสุดท้าย | ค่า索index ที่ 5 มีค่าเท่ากับ 2 ซึ่งสามารถเขียนได้เป็น intEx1[5] = 2 |

### ตัวแปรอาร์เรย์ 2 มิติ (2-Dimension Array)

เปรียบได้กับการนำตัวแปรมาเรียงต่อกันหลาย ๆ ตัวในลักษณะของตารางข้อมูล ซึ่งสามารถจัดล่องตัวอย่างตัวแปร อาร์เรย์ 2 มิติ ข้อตัวแปร intEx2 เป็นตัวแปรชนิดจำนวนเต็มที่สามารถเก็บข้อมูลจำนวนเต็มได้ 15 ตัว ยกตัวอย่างดังนี้

Index					Value				
[0][0]	[0][1]	[0][2]	[0][3]	[0][4]	3	7	2	6	1
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]	7	1	9	2	6
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]	4	2	5	4	3

- จากรูปด้านบนจะเห็นได้ว่า ตัวแปรอาร์เรย์ที่มีขนาด 3 แถว 5 คอลัมน์ มีลักษณะคล้ายตาราง สามารถเก็บข้อมูลได้ 15 ตัว ตัวอย่างเช่น ตัวแปรแถวที่ 1 คอลัมน์ที่ 1 มีค่าเท่ากับ 3 ซึ่งสามารถเขียนได้เป็น intEx2[0][0] = 3 ตัวแปรแถวที่ 1 คอลัมน์ที่ 2 มีค่าเท่ากับ 7 ซึ่งสามารถเขียนได้เป็น intEx2[0][1] = 7 ตัวแปรแถวที่ 1 คอลัมน์ที่ 5 มีค่าเท่ากับ 1 ซึ่งสามารถเขียนได้เป็น intEx2[0][4] = 1 ตัวแปรแถวที่ 2 คอลัมน์ที่ 1 มีค่าเท่ากับ 7 ซึ่งสามารถเขียนได้เป็น intEx2[1][0] = 7 ตัวแปรแถวที่ 2 คอลัมน์ที่ 2 มีค่าเท่ากับ 1 ซึ่งสามารถเขียนได้เป็น intEx2[1][1] = 1 ตัวแปรแถวที่ 2 คอลัมน์ที่ 4 มีค่าเท่ากับ 2 ซึ่งสามารถเขียนได้เป็น intEx2[1][3] = 2 ตัวแปรแถวที่ 3 คอลัมน์ที่ 1 มีค่าเท่ากับ 4 ซึ่งสามารถเขียนได้เป็น intEx2[2][0] = 4 ตัวแปรแถวที่ 3 คอลัมน์ที่ 2 มีค่าเท่ากับ 2 ซึ่งสามารถเขียนได้เป็น intEx2[2][1] = 2 ตัวแปรแถวที่ 3 คอลัมน์ที่ 5 มีค่าเท่ากับ 3 ซึ่งสามารถเขียนได้เป็น intEx2[2][4] = 3

### ตัวแปรอาร์เรย์ 3 มิติ (3-Dimension Array)

เปรียบได้กับการนำตัวแปรมาเรียงต่อกันหลาย ๆ ตัวในลักษณะของกล่องข้อมูล ซึ่งสามารถจัดล่องตัวอย่างตัวแปร อาร์เรย์ 3 มิติ ข้อตัวแปร intEx3 เป็นตัวแปรชนิดจำนวนเต็มที่สามารถเก็บข้อมูลจำนวนเต็มได้ 24 ตัว ยกตัวอย่างดังนี้



จากรูปด้านบนจะเห็นได้ว่า ตัวแปรอาร์เรย์ที่มีขนาด 2 บล็อก 3 และ 4 คอลัมน์ สามารถเก็บข้อมูลได้ 24 ตัว ตัวอย่างเช่น ตัวแปรนล็อกที่ 1 แกวที่ 1 คอลัมน์ที่ 1 มีค่าเท่ากัน 9 ซึ่งสามารถเขียนได้เป็น `intEx3[0][0][0] = 9` ตัวแปรนล็อกที่ 1 แกวที่ 1 คอลัมน์ที่ 2 มีค่าเท่ากัน 2 ซึ่งสามารถเขียนได้เป็น `intEx3[0][0][1] = 2` ตัวแปรนล็อกที่ 1 แกวที่ 1 คอลัมน์ที่ 3 มีค่าเท่ากัน 1 ซึ่งสามารถเขียนได้เป็น `intEx3[0][0][2] = 1` ตัวแปรนล็อกที่ 1 แกวที่ 2 คอลัมน์ที่ 1 มีค่าเท่ากัน 8 ซึ่งสามารถเขียนได้เป็น `intEx3[0][1][0] = 8` ตัวแปรนล็อกที่ 1 แกวที่ 2 คอลัมน์ที่ 2 มีค่าเท่ากัน 5 ซึ่งสามารถเขียนได้เป็น `intEx3[0][1][1] = 5` ตัวแปรนล็อกที่ 1 แกวที่ 3 คอลัมน์ที่ 4 มีค่าเท่ากัน 7 ซึ่งสามารถเขียนได้เป็น `intEx3[0][2][3] = 7` ตัวแปรนล็อกที่ 2 แกวที่ 1 คอลัมน์ที่ 1 มีค่าเท่ากัน 7 ซึ่งสามารถเขียนได้เป็น `intEx3[1][0][0] = 7` ตัวแปรนล็อกที่ 2 แกวที่ 1 คอลัมน์ที่ 2 มีค่าเท่ากัน 1 ซึ่งสามารถเขียนได้เป็น `intEx3[1][0][1] = 1` ตัวแปรนล็อกที่ 2 แกวที่ 1 คอลัมน์ที่ 3 มีค่าเท่ากัน 4 ซึ่งสามารถเขียนได้เป็น `intEx3[1][0][2] = 4` ตัวแปรนล็อกที่ 2 แกวที่ 2 คอลัมน์ที่ 1 มีค่าเท่ากัน 3 ซึ่งสามารถเขียนได้เป็น `intEx3[1][1][0] = 3` ตัวแปรนล็อกที่ 2 แกวที่ 2 คอลัมน์ที่ 2 มีค่าเท่ากัน 7 ซึ่งสามารถเขียนได้เป็น `intEx3[1][1][1] = 7` ตัวแปรนล็อกที่ 2 แกวที่ 3 คอลัมน์ที่ 4 มีค่าเท่ากัน 2 ซึ่งสามารถเขียนได้เป็น `intEx3[1][2][3] = 2`

## การประกาศตัวแปรอาร์เรย์

||| `dataType varName[k][m][n];`

โดยที่ <b>dataType</b>	เป็นชนิดของข้อมูล	เช่น <b>int intNum[5];</b>
<b>varName</b>	เป็นชื่อตัวแปรอาร์เรย์	<b>char chPassword[4];</b>
<b>n</b>	เป็นขนาดคอลัมน์ของตัวแปรอาร์เรย์	<b>float fPrice[3][2];</b>
<b>m</b>	เป็นขนาดแถวของตัวแปรอาร์เรย์	<b>int intCount[3][5];</b>
<b>k</b>	เป็นขนาดบล็อก (หรือชั้น) ของตัวแปรอาร์เรย์	<b>int intNo[3][2][5];</b>

จากโค้ดตัวอย่างจะเห็นได้ว่าจะมีการกำหนดขนาดตัวแปรไว้ภายในเครื่องหมาย [ ] เช่น `int intCount[3][5]` คือ การประกาศตัวแปรชนิดจำนวนเต็มชื่อ `intCount` ขนาด 3 และ 5 คอลัมน์, `char chPassword[4]` คือการประกาศตัวแปรชนิดอักขระตัวเดียวชื่อ `chPassword` ขนาด 4 คอลัมน์ และ `int intNo[3][2][5]` คือการประกาศตัวแปรชนิดจำนวนเต็มชื่อ `intNo` ขนาด 3 บล็อก 2 และ 5 คอลัมน์ เป็นต้น

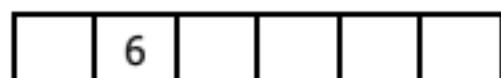
## การกำหนดค่าข้อมูลให้ตัวแปรอาร์เรย์

การกำหนดค่าให้กับตัวแปรแบบอาร์เรย์นั้น เราต้องกำหนดตำแหน่งอินเด็กซ์เพื่อบรุต้าแห่งนั่งของตัวแปรที่จะกำหนดค่า ยกตัวอย่างเช่น

`int intNum[5] = {5, 3, 4, 1, 7};`      `char chPassword[4] = {'P', 'a', 's', 's'};`  
`int intScore[2][4] = { {2, 4, 7, 6}, {9, 1, 3, 4} };`    `float fPrice[3][2] = {12.5, 17.8, 8.1, 45.9, 9.7, 86.5};`

ตัวอย่างที่ผ่านมาเป็นการประกาศตัวแปรร่วมกับกำหนดค่า ซึ่งเราอาจจะกำหนดค่าภายหลังการประกาศตัวแปรได้ เช่นกัน ซึ่งมีตัวอย่างดังนี้

จากตัวอย่างจะเห็นได้ว่า เรากำหนดค่าให้ตัวแปร `intNum` ในลำดับที่ 2 อินเด็กซ์ที่ 1 มีค่าเท่ากับ 6 ซึ่งได้ผลลัพธ์ดังนี้



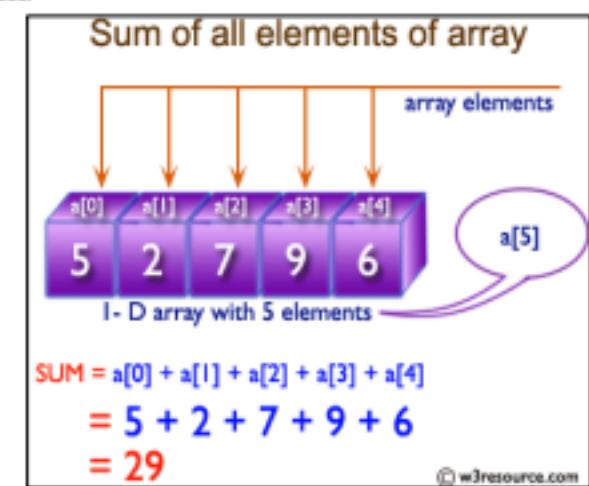
```
int intNum[5];
char chPassword[4];
float fPrice[3][2];

intNum[1] = 6;
chPassword[0] = 'Z';
fPrice[2][1] = 0.08;
```

## การอ้างถึงข้อมูลในตัวแปรอาร์เรย์

เนื่องมีการกำหนดค่าให้กับตัวแปรแล้ว ต่อไปก็ต้องรู้วิธีการอ่านข้อมูลจากตัวแปร โดยการระบุตำแหน่งข้อมูลที่ต้องการลงไป หรือที่เรียกว่า อินเด็กซ์ ซึ่งมีด้วยอย่างการใช้งานดังนี้

```
Ans      = intScore[2] * 4;
fSum     = fPrice[3][2] + fPrice[3][2];
intNum[0] = intNum[1] + intNum[2];
```



จากตัวอย่างด้านบน การเรียกใช้งานตัวแปรอาร์เรย์โดยระบุตำแหน่งข้อมูลภายในเครื่องหมาย [ ] เช่น Ans = intScore[2] \* 4; เป็นการกำหนดค่าให้กับตัวแปร Ans โดยมีค่ามาจากการคูณระหว่างค่าในตัวแปรอาร์เรย์ตำแหน่งที่ 3 กับจำนวนเต็ม 4 เป็นต้น

```
1 #include<stdio.h>
2
3 main()
4 {
5     int a[100];
6     int i, n, sum = 0;
7
8     printf("Input the number of elements = ");
9     scanf("%d", &n);
10
11    printf("Input %d elements\n", n);
12    for(i = 0; i < n; i++)
13    {
14        printf("elements %d : ", i);
15        scanf("%d", &a[i]);
16        sum += a[i];
17    }
18    printf("\nSum = %d\n", sum);
19    for(i = 0; i < n; i++) printf("a[%d] = %d\n", i, a[i]);
20 }
```

```
C:\Users\...\Desktop\Untitled1.exe
Input the number of elements = 5
Input 5 elements
elements 0 : 5
elements 1 : 2
elements 2 : 7
elements 3 : 9
elements 4 : 6
Sum = 29
a[0] = 5
a[1] = 2
a[2] = 7
a[3] = 9
a[4] = 6
-----
Process exited after 2.031
seconds with return value 0
Press any key to continue . . .
```

จากตัวอย่างด้านบน เป็นโปรแกรมคำนวณผลรวมของค่าที่รับเข้าไปใส่ในอาร์เรย์ และแสดงค่าในแต่ละ element โคดในบรรทัดที่ 8 – 9 จะเป็นการให้ผู้ใช้กรอกว่าจะเก็บค่ากี่ค่า เพื่อใช้สร้างอาร์เรย์ที่มี index ขนาดเท่ากับข้อมูลที่ใส่บรรทัดที่ 14 – 16 เป็นการรับค่าแต่ละค่าเข้ามาเก็บในอาร์เรย์แต่ละอินเด็กซ์ตั้งแต่ a[0] a[1] จนถึง a[n-1] และระหว่างนั้นถ้ารับค่าเข้ามาแล้วให้นำไปบวกเข้าไปเก็บไว้ในตัวแปร sum โดยใช้ค่าลั่ง sum += a[i] หรือ sum = sum + a[i] บรรทัดที่ 18 เป็นการแสดงผลบวกทั้งหมดจากตัวแปร sum

บรรทัดที่ 19 เป็นการแสดงค่าที่เก็บอยู่ในแต่ละอินเด็กซ์โดยใช้ลูปมาช่วย โดยกำหนดให้ลูปวนซ้ำเท่ากับจำนวนสมาชิกที่กรอกเข้ามา (n)

**ข้อควรระวัง :** การประกาศตัวแปร array ต้องประกาศให้พอด้วยจำนวนที่ใช้ ไม่นั้นจะเกิด error ขึ้น เช่น ประกาศอาร์เรย์ชื่อ a ไว้ 5 index ( int a[5]; ) แต่ตอนเรียกใช้ หรือกำหนดค่า a[5] จะมี....

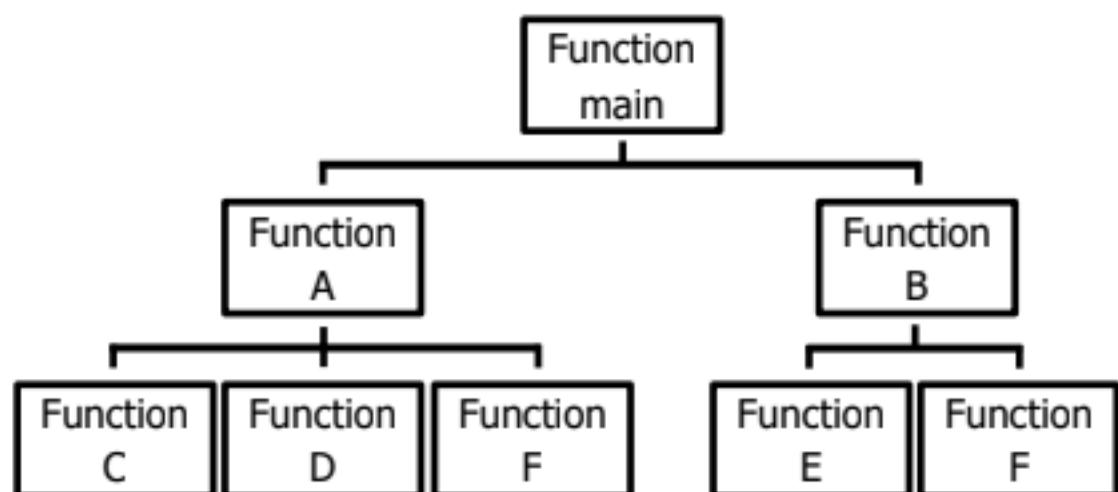
**ข้อสังเกต :** array นักจะใช้งานร่วมกับ for เพื่อให้ใช้งานได้ง่ายขึ้น เช่น แทนที่จะรับค่าให้อาร์เรย์ทีละตัว scanf("%d", &a[0]) scanf("%d", &a[1]) ... scanf("%d", &a[4]) เราจะรับค่าโดยใช้ for ช่วยเหมือนตัวอย่างด้านบน หรือจะใช้ช่วยในการแสดงผลแทนที่จะต้อง printf("%d", a[index]) ทุก index

**ข้อควรรับ :** array จะเริ่มต้นที่อินเด็กซ์เป็น 0 และตัวสุดท้ายเป็นอินเด็กซ์ที่ n-1 (หาก n เป็นจำนวนอินเด็กซ์ที่ประกาศไว้)

## ฟังก์ชัน (Function)

คือชุดคำสั่งที่รวมกันเป็นโปรแกรมย่อย ๆ ภายในเครื่องหมาย {} ถูกสร้างขึ้นมาเพื่อทำงานอย่างใดอย่างหนึ่ง และมีการตั้งชื่อของฟังก์ชันเพื่อให้สะท杵กต่อการเรียกใช้งานตามกฎการตั้งชื่อ โดยมีรูปแบบการเรียกใช้งานแตกต่างกัน คือจะมีการรับ หรือไม่รับข้อมูลจากโปรแกรมที่เรียกใช้งาน และจะมีการส่ง หรือไม่ส่งค่าข้อมูลออกจากฟังก์ชัน ซึ่งรูปแบบการใช้งานต่าง ๆ ของฟังก์ชันจะขึ้นอยู่กับหน้าที่ และเป้าหมายการทำงานของฟังก์ชันนั้น ๆ

การเขียนโปรแกรมภาษา C มีโครงสร้างประกอบด้วยฟังก์ชันการทำงาน โดยจะเริ่มต้นการทำงานที่ฟังก์ชัน main() เสมอ และฟังก์ชัน main() นั้นสามารถเรียกใช้ฟังก์ชันย่อยอื่น ๆ ได้ไม่ว่าจะเป็นฟังก์ชันที่ผู้ใช้งานสร้างขึ้นมาเอง (User-Defined Functions) หรือฟังก์ชันมาตรฐานที่ภาษา C ได้สร้างมาให้ (Standard Library Function) นอกจากนี้ในฟังก์ชันย่อยก็ยังสามารถที่จะเรียกใช้ฟังก์ชันย่อยอื่น ๆ ได้เช่นกัน



ตัวอย่างเช่น ฟังก์ชัน main() เรียกใช้งานฟังก์ชัน A และฟังก์ชัน B และฟังก์ชัน A เรียกใช้งาน C, ฟังก์ชัน D และฟังก์ชัน F ส่วนฟังก์ชัน B เรียกใช้งานฟังก์ชัน E และฟังก์ชัน F เป็นต้น

**ข้อควรรู้ :** การเขียนโปรแกรมเรียกใช้งานฟังก์ชันย่อย สามารถตรวจสอบการทำงาน และพัฒนาโปรแกรม "ได้ง่ายก็จริง แต่ผู้เขียนไม่แน่น่าให้แบ่งโปรแกรมย่อย ๆ เพื่อเรียกใช้งานข้อนักกันมากเกินไป" เนื่องจากจะทำให้โปรแกรมเกิดความขัดข้อง แก้ไขและพัฒนาได้ยาก

### ฟังก์ชันที่มาตรฐานที่ติดมาด้วยภาษา C (Standard Library Function)

ฟังก์ชันมาตรฐาน คือฟังก์ชันที่ผู้ใช้งานสามารถเรียกใช้งานจากไลบรารีของภาษาซึ่งได้ทันที เช่น ฟังก์ชันทางคณิตศาสตร์ ฟังก์ชันเกี่ยวกับสตริง ฟังก์ชันเกี่ยวกับการเปรียบเทียบ ฟังก์ชันเกี่ยวกับการแสดงผล และฟังก์ชันเกี่ยวกับวันเวลา เป็นต้น ซึ่งฟังก์ชันมาตรฐานนี้เป็นฟังก์ชันที่มีติดมาให้ เก็บไว้ใน header file ภาษา C (เก็บไว้ในแฟ้มที่มีนามสกุล \*.h ต่าง ๆ ) เมื่อต้องการใช้ฟังก์ชันใด จะต้องรู้ว่าฟังก์ชันนั้นอยู่ใน header file ใด จากนั้นจึงค่อยใช้ค่าสั่ง #include เข้ามาในส่วนตอนต้นของโปรแกรม จึงจะสามารถใช้ฟังก์ชันที่ต้องการได้ เช่น ฟังก์ชัน pow(x, y) จะถูกเก็บไว้ใน header file ที่ชื่อว่า math.h ดังนั้นจึงต้องใช้ค่าสั่ง #include แทรกอยู่ในส่วนตอนต้นของโปรแกรมหนีอฟังก์ชัน main( ) จึงจะสามารถเรียกใช้ฟังก์ชัน pow(x, y) มาใช้งานภายใต้โดยฟังก์ชันมาตรฐานเป็นฟังก์ชันที่บริษัทผู้ผลิต C compiler เขียนขึ้นเพื่อให้ผู้ใช้นำไปช่วยในการเขียนโปรแกรมทำให้การเขียนโปรแกรมสะดวก และง่ายขึ้น บางครั้งอาจเรียกฟังก์ชันมาตรฐานว่า "ไลบรารีฟังก์ชัน" (library functions)

#### math.h เกี่ยวกับการคำนวนทางคณิตศาสตร์

ฟังก์ชัน	คำอธิบาย
sin(x)	หาค่า sine ของมุม โดยที่ x เป็นมุมที่ต้องการหา มีหน่วยเป็นเรเดียน
cos(x)	หาค่า cosine ของมุม โดยที่ x เป็นมุมที่ต้องการหา มีหน่วยเป็นเรเดียน
tan(x)	หาค่า tangent ของมุม โดยที่ x เป็นมุมที่ต้องการหา มีหน่วยเป็นเรเดียน
sqrt(x)	หาค่ารากที่สอง โดยที่ x เป็นตัวแปร หรือค่าคงที่ ซึ่งเป็นจำนวนเต็มบวก หรือจำนวนเด็มศูนย์
pow(x, y)	หาค่ายกกำลัง โดยที่ x เป็นตัวแปรหรือค่าคงที่ ซึ่งเป็นเลขฐาน และเป็นจำนวนเต็มบวก หรือจำนวนเด็มศูนย์
pow10(x)	หาค่ายกกำลัง โดยที่เลขฐานเป็นเลข 10
log(n)	หาค่า logฐาน n โดยที่ n เป็นตัวแปร หรือค่าคงที่ ซึ่งเป็นจำนวนเต็มบวก หรือจำนวนเด็มศูนย์
log10(x)	หาค่า logฐาน 10 โดยที่ x เป็นตัวแปร หรือค่าคงที่ ซึ่งเป็นจำนวนเต็มบวก หรือจำนวนเด็มศูนย์
exp(x)	หาค่า e โดย e จะมีค่าประมาณ 2.718282
abs(x)	หาค่าสัมบูรณ์ของเลขจำนวนเต็ม
fabs(x)	หาค่าสัมบูรณ์ โดยที่ x เป็นตัวแปรหรือค่าคงที่

## string.h เกี่ยวกับข้อความ

ฟังก์ชัน	คำอธิบาย
strcpy(str1, str2)	คัดลอกข้อความจากตัวแปร str2 มาเก็บไว้ที่ตัวแปร str1
strcat(str1, str2)	เชื่อมต่อข้อความ โดยนำค่าตัวแปร str2 มาต่อท้ายตัวแปร str1 และเก็บค่าไว้ที่ตัวแปร str1
strchr(str, ch)	หาตำแหน่งของตัวอักษร ch ในสตริง str โดยหาตั้งแต่ตัวแรกจนกระทั่งพบ หรือจนกระทั่งหมด สตริง โดยจะเปรียบเทียบ null character ด้วย
strncpy(dest, src, n)	ทำการก็อปปี้สตริงจาก src ไปยัง dest ถ้า src เป็นจำนวน ก อักษร
strcmp(str1, str2)	เปรียบเทียบข้อมูลนิดสตริง str1 เทียบตัวแปร str2 ว่ามีค่ามากกว่า น้อยกว่า หรือเท่ากัน อย่างใดอย่างหนึ่ง ซึ่งการเปรียบเทียบสตริงจะใช้ค่ารหัส ASCII เปรียบเทียบที่ละตัวอักษร
strcmpi(str1, str2)	เปรียบเทียบข้อมูลนิดสตริง str1 เทียบตัวแปร str2 ว่ามีค่ามากกว่า น้อยกว่า หรือเท่ากัน อย่างใดอย่างหนึ่ง โดยไม่สนใจว่าเป็นตัวอักษรพิมพ์เล็ก หรือตัวอักษรพิมพ์ใหญ่ ซึ่งการเปรียบเทียบสตริงจะใช้ค่ารหัส ASCII เปรียบเทียบที่ละตัวอักษร
strlen(str)	หาความยาวข้อความ โดยที่ str เป็นตัวแปรชนิดข้อความหรือค่าคงที่
strupr(str)	แปลงข้อมูลในสตริง str จากตัวอักษรตัวพิมพ์เล็กให้เป็นอักษรตัวพิมพ์ใหญ่หมด
strlwr(str)	แปลงข้อมูลในสตริง str จากตัวอักษรตัวพิมพ์ใหญ่ให้เป็นอักษรตัวพิมพ์เล็กหมด
strstr(str, substr)	ค้นหาสับสตริงในสตริง โดยจะค้นดูข้อมูลใน str ว่ามีเหมือนในข้อมูลใน substr หรือไม่

## ctype.h เกี่ยวกับตัวอักษรตัวเดียว

ฟังก์ชัน	คำอธิบาย
isalnum(ch)	ตรวจสอบว่าข้อมูลที่อยู่ในตัวแปรมีค่าเป็นตัวอักษรหรือตัวเลข
isalpha(ch)	ตรวจสอบว่าข้อมูลที่อยู่ในตัวแปรมีค่าเป็นตัวอักษรหรือไม่
isdigit(ch)	ตรวจสอบว่าข้อมูลที่อยู่ในตัวแปรเป็นตัวเลข 0 ถึง 9 หรือไม่
islower(ch)	ตรวจสอบว่าข้อมูลที่อยู่ในตัวแปรเป็นตัวเล็กหรือไม่
isupper(ch)	ตรวจสอบว่าข้อมูลที่อยู่ในตัวแปรเป็นตัวใหญ่หรือไม่
isspace(ch)	ตรวจสอบดูค่าใน ch เป็นค่าแผลสก์โคลดของช่องว่างหรือไม่ ถ้าเป็นจริง จะให้ค่าไม่เท่ากับศูนย์ ถ้า เป็นเท็จ จะให้ค่าเท่ากับศูนย์
isxdigit(ch)	ตรวจสอบว่าค่าใน ch เป็นค่าแผลสก์โคลดของเลขฐานสิบหก (0-9, A-F หรือ a-f) หรือไม่ ถ้าเป็นจริง จะให้ค่าไม่เท่ากับศูนย์ ถ้า เป็นเท็จ จะให้ค่าเท่ากับศูนย์
tolower(ch)	เปลี่ยนตัวอักษรจากตัวอักษรตัวพิมพ์เล็กให้เป็นอักษรตัวพิมพ์ใหญ่หมด
toupper(ch)	เปลี่ยนตัวอักษรจากตัวอักษรตัวพิมพ์เล็กให้เป็นอักษรตัวพิมพ์ใหญ่หมด

## stdlib.h เกี่ยวกับการแปลงค่า string

ฟังก์ชัน	คำอธิบาย
atoi(s)	แปลงค่าข้อความ (string) เป็นตัวเลขจำนวนเต็ม (integer)
atof(s)	แปลงค่าข้อความ (string) เป็นตัวเลขจำนวนทศนิยม (float)
atol(s)	แปลงค่าข้อความ (string) เป็นตัวเลขจำนวนเต็ม (integer) ชนิด long integer
system ("cls");	ลบจอภาพ

## dos.h เกี่ยวกับการติดต่อระบบปฏิบัติการ

ฟังก์ชัน	คำอธิบาย
gettime()	ใช้ในการติดต่อเวลาของระบบปฏิบัติการ
getdate()	ใช้ในการติดต่อวันที่ของระบบปฏิบัติการ

## conio.h เกี่ยวกับการแสดงผลทางจอภาพ

ฟังก์ชัน	คำอธิบาย
getchar()	รับข้อมูล 1 อักขระ โดยการกด Enter
getche()	รับข้อมูล 1 อักขระ โดยไม่ต้องกด Enter
getch()	รับข้อมูล 1 อักขระในปุ่มพิเศษให้เห็นในการรับข้อมูล
putchar()	รับข้อมูล 1 อักขระออกทางจอภาพ
clrscr()	ลบจอภาพ

## ฟังก์ชันที่สร้างขึ้นเอง (User-Defined Functions)

เป็นฟังก์ชันที่ผู้ใช้งานเขียนโค้ดสร้างฟังก์ชันขึ้นมาใช้ อิงตามรูปแบบการสร้างฟังก์ชันของภาษา C เพื่อให้ทำงานอย่างใดอย่างหนึ่ง ซึ่งสามารถแบ่งรูปแบบการสร้างฟังก์ชันได้ 4 รูปแบบดังนี้

- ฟังก์ชันที่ไม่มีการคืนค่ากลับและไม่มีการรับค่าพารามิเตอร์ (Void Functions with No Parameters)
- ฟังก์ชันที่ไม่มีการคืนค่ากลับ แต่มีการรับค่าพารามิเตอร์ (Void Functions with Parameters)
- ฟังก์ชันที่มีการคืนค่ากลับ แต่ไม่มีการรับค่าพารามิเตอร์ (Function Return Value with No Parameters)
- ฟังก์ชันที่มีการคืนค่ากลับ และมีการรับค่าพารามิเตอร์ (Function Return Value with Parameters)

### ฟังก์ชันที่ไม่มีการคืนค่ากลับและไม่มีการรับค่าพารามิเตอร์ (Void Functions with No Parameters)

เป็นฟังก์ชันที่สร้างขึ้นโดยไม่มีการรับค่าข้อมูล (พารามิเตอร์) ใด ๆ จากฟังก์ชันที่เรียกใช้งาน และเมื่อฟังก์ชันทำงานเสร็จก็จะไม่มีการคืนค่าข้อมูลใด ๆ กลับไปให้ฟังก์ชันที่เรียกใช้งาน ซึ่งมีรูปแบบการสร้างฟังก์ชันดังนี้

```
void functionName(void) {
    statements;
}
```

จากรูปแบบฟังก์ชันดังกล่าว จึงสามารถสร้างฟังก์ชันใช้งานได้ดังต่อไปนี้

```
void printName(void) {
    printf("Programming Language");
}
```

**ข้อสังเกต :** คีย์เวิร์ด void หน้าชื่อฟังก์ชัน เป็นการกำหนดฟังก์ชันนั้นไม่มีการคืนค่าใด ๆ กลับไปยังฟังก์ชันที่เรียกใช้งาน และคีย์เวิร์ด void หลังชื่อฟังก์ชัน เป็นการกำหนดฟังก์ชันนั้นไม่มีการรับค่าพารามิเตอร์ใด ๆ จากฟังก์ชันที่เรียกใช้งาน

เมื่อได้เขียนฟังก์ชันขึ้นมาเพื่อใช้งานเองแล้ว จะยังไม่สามารถเรียกใช้งานฟังก์ชันดังกล่าว ได้จนกว่าจะมีการประกาศโดยให้ป้อนชื่อฟังก์ชันก่อนฟังก์ชัน main เพื่อให้คอมไพล์รู้จักฟังก์ชัน จากตัวอย่างโค้ดจะประกาศโดยให้มีดังนี้ **void printName(void);** ซึ่งการประกาศโดยให้มีรูปแบบดังนี้

```
void functionName(void);
```

หลังจากประกาศโดยให้มีชื่อฟังก์ชันเพื่อให้ฟังก์ชัน main รู้จักกับฟังก์ชันที่สร้างขึ้นแล้ว สามารถเรียกใช้งานฟังก์ชันดังกล่าว ได้จากการเรียกฟังก์ชัน main หรือจากฟังก์ชันอื่น ๆ ภายในโปรแกรมได้ จากตัวอย่างโค้ดสามารถเรียกใช้งานตัวฟังก์ชันได้ดังนี้ **printName();** โดยการเรียกใช้งานฟังก์ชันจะมีรูปแบบดังนี้

```
functionName();
```

จากรูปแบบการประกาศโดยให้มีชื่อฟังก์ชัน และการเรียกใช้งานฟังก์ชัน จึงสามารถสร้างฟังก์ชัน ใช้งานและเรียกใช้งานได้ตามตัวอย่างง่าย ๆ ทางด้านขวาดังนี้

**ข้อควรรู้ :** เมื่อจากโปรแกรมทำงานจากชัยไปขวา บนลงล่าง เมื่อรันโปรแกรม หากในฟังก์ชัน main เจอฟังก์ชันแปลง ๆ ที่คอมไಪล์ในรูปแบบจะมีการแจ้ง error ออกมา (คล้าย ๆ กับการใช้ตัวแปรแต่ยังไม่ได้ประกาศใช้ตัวแปร) จึงแก้ด้วยการประกาศให้คอมไಪล์รู้ว่ามีฟังก์ชันแปลง ๆ นือญ โดยการประกาศโดยให้มีชื่อฟังก์ชันที่มีการแสดงส่วนประกอบของฟังก์ชันว่ามีการคืนค่ากลับหรือไม่ มีพารามิเตอร์กี่ตัว ชื่อฟังก์ชันชื่ออะไร

```
#include<stdio.h>
void printName(void);
void main() {
    printName();
}
void printName(void) {
    printf("Programming Language");
}
```

## ฟังก์ชันที่ไม่มีการคืนค่ากลับ แต่มีการรับค่าพารามิเตอร์ (Void Functions with Parameters)

เป็นฟังก์ชันที่สร้างขึ้นโดยมีการรับค่าข้อมูล (พารามิเตอร์) จากฟังก์ชันที่เรียกใช้งาน และเมื่อฟังก์ชันทำงานเสร็จ จะไม่มีการคืนค่าข้อมูลใด ๆ กลับไปให้ฟังก์ชันที่เรียกใช้งาน ซึ่งมีรูปแบบการสร้างฟังก์ชันดังนี้

**กำหนดให้** **typeParameter\_n** เป็นชนิดข้อมูลที่ต้องการรับค่าจากฟังก์ชันที่เรียกใช้งาน  
**parameterName\_n** เป็นชื่อพารามิเตอร์ หรือตัวแปรตัวที่ *g* ที่ใช้รับข้อมูลจากฟังก์ชันที่เรียกใช้งาน  
 และต้องมีชนิดข้อมูลตรงกันกับ **typeParameter\_n**

```
void functionName(typeParameter_1 parameterName_1, ..., typeParameter_n parameterName_n)
{
    statements;
}
```

จากรูปแบบฟังก์ชันดังกล่าว จึงสามารถสร้างฟังก์ชันใช้งานได้ดังนี้ ตัวอย่างด้านขวา และการประกาศโปรแกรมโดยป้อนของฟังก์ชันดังกล่าวสามารถทำได้โดย **void calculate(int, int);** ซึ่งเป็นไปตามรูปแบบดังนี้

```
void calculate(int a, int b) {
    int ans;
    ans = a * b;
    printf("%d", ans);
}
```

```
void functionName(typeParameter_1 parameterName_1, ..., typeParameter_n
parameterName_n);
```

หลังจากการประกาศโปรแกรมโดยป้อนจะสามารถเรียกใช้งานตัวฟังก์ชันได้ดังนี้ **calculate(x, y);** โดยการเรียกใช้งานฟังก์ชันจะมีรูปแบบดังนี้

```
functionName(parameterName_1, ..., parameterName_n);
```

## ฟังก์ชันที่มีการคืนค่ากลับ แต่ไม่มีการรับค่าพารามิเตอร์ (Function Return Value with No Parameters)

เป็นฟังก์ชันที่สร้างขึ้นโดยไม่มีการรับค่าข้อมูล (พารามิเตอร์) ใด ๆ จากฟังก์ชันที่เรียกใช้งาน และ เมื่อฟังก์ชันทำงานเสร็จจะมีการคืนค่าข้อมูลกลับไปให้ฟังก์ชันที่เรียกใช้งาน ซึ่งมีรูปแบบการสร้างฟังก์ชันดังนี้

**กำหนดให้** **typeReturn** เป็นชนิดข้อมูลที่ต้องการคืนค่ากลับให้ฟังก์ชันที่เรียกใช้งาน  
**varReturn** เป็นตัวแปร หรือค่าที่ต้องการคืนค่ากลับให้ฟังก์ชันที่เรียก  
**variable** เป็นตัวแปรที่รับค่าข้อมูลที่ฟังก์ชันคืนค่ากลับมาให้ (สำหรับเก็บผลลัพธ์ของฟังก์ชัน)

```
typeReturn functionName(void)
{
    statements;
    return varReturn;
}
```

จากรูปแบบฟังก์ชัน จึงสามารถสร้างฟังก์ชันใช้งานได้ดังตัวอย่างด้านขวา และสามารถประกาศโปรแกรมโดยป้อนได้ดังนี้ **int calculate2(void);** โดยการประกาศโปรแกรมโดยป้อนจะมีรูปแบบตามด้านล่าง

```
int calculate2(void) {
    int a = 3, b = 5, c;
    c = a * b;
    return c;
}
```

```
typeReturn functionName(void);
```

หลังจากการประกาศโปรแกรมโดยป้อนจะสามารถเรียกใช้งานตัวฟังก์ชัน โดยเก็บผลการทำงานของฟังก์ชันไว้ที่ตัวแปร *ans* (สมมติมีการประกาศตัวแปร *ans* เป็นประเภท *int* เรียบร้อยแล้ว) ดังนี้ **ans = calculate2();** โดยการเรียกใช้งานฟังก์ชันจะมีรูปแบบดังนี้

```
variable = functionName();
```

## ฟังก์ชันที่มีการคืนค่ากลับ และมีการรับค่าพารามิเตอร์ (Function Return Value with Parameters)

เป็นฟังก์ชันที่สร้างขึ้นโดยมีการรับค่าข้อมูล (พารามิเตอร์) จากฟังก์ชันที่เรียกใช้งาน และ เมื่อฟังก์ชันทำงานเสร็จจะมีการคืนค่าข้อมูลกลับไปให้ฟังก์ชันที่เรียกใช้งาน ซึ่งมีรูปแบบการสร้างฟังก์ชันดังนี้

กำหนดให้

<b>typeReturn</b>	เป็นชนิดข้อมูลที่ต้องการคืนค่ากลับให้ฟังก์ชันที่เรียกใช้งาน
<b>typeParameter_n</b>	เป็นชนิดข้อมูลที่ต้องการรับจากฟังก์ชันที่เรียกใช้งาน
<b>parameterName_n</b>	เป็นชื่อพารามิเตอร์ หรือตัวแปรตัวที่ <i>n</i> ที่ใช้รับข้อมูลจากฟังก์ชันที่เรียกใช้งาน และต้องมีชนิดข้อมูลตรงกันกับ typeParameter_n
<b>varReturn</b>	เป็นตัวแปร หรือค่าที่ต้องการคืนค่ากลับให้ฟังก์ชันที่เรียก
<b>variable</b>	เป็นตัวแปรที่รับค่าข้อมูลที่ฟังก์ชันคืนค่ากลับมาให้ (สำหรับเก็บผลลัพธ์ของฟังก์ชัน)

```
typeReturn functionName(typeParameter_1 parameterName_1, ..., typeParameter_n
parameterName_n)
{
    statements;
    return varReturn;
}
```

```
float circleArea(float r) {
    return 3.14 * r * r;
}
```

จากรูปแบบฟังก์ชัน จึงสามารถสร้างฟังก์ชันใช้งานได้ดังตัวอย่างข้างต้น และสามารถประกาศโปรแกรมไทยปีได้ดังนี้ **float circleArea(float);** โดยการประกาศโปรแกรมไทยปีจะมีรูปแบบตามด้านล่าง

```
typeReturn functionName(typeParameter_1 parameterName_1, ..., typeParameter_n
parameterName_n);
```

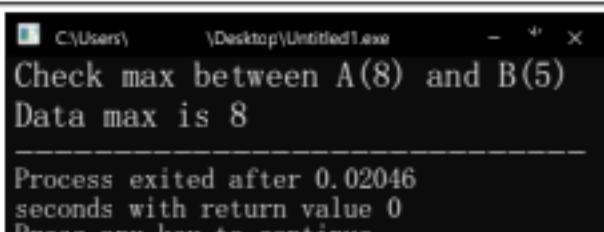
หลังจากการประกาศโปรแกรมไทยปีจะสามารถเรียกใช้งานตัวฟังก์ชัน โดยเก็บผลการทำงานของฟังก์ชันไว้ที่ตัวแปร area (สมมติมีการประกาศตัวแปร area เป็นประเภท float เรียนร้อยแล้ว) ดังนี้ **area = circleArea(radius);** ซึ่งการเรียกฟังก์ชันจะมีรูปแบบตามด้านล่าง

```
variable = functionName(parameterName_1, ..., parameterName_n);
```

**ข้อควรรู้ :** ในการรับส่งข้อมูลระหว่าง Argument กับ Parameter ต้องมีชนิดข้อมูลตรงกัน โดยที่ Argument คือตัวรับข้อมูลเข้ามาในฟังก์ชัน และ Parameter คือตัวส่งข้อมูลไปยังฟังก์ชัน



โปรแกรมนี้เป็นการเปรียบเทียบค่า 2 ค่า ระหว่าง 8 ที่เก็บอยู่ในตัวแปร a และ 5 ที่เก็บอยู่ในตัวแปร b โดยการส่งค่าผ่านพารามิเตอร์ num1 และ num2 เข้าไปในฟังก์ชัน ให้คำนวณอุปกรณ์และแสดงผลตามเงื่อนไขที่กำหนดไว้ โดยในบรรทัดที่ 3 เป็นการประกาศโปรแกรมไทยปี และบรรทัดที่ 12-17 เป็นฟังก์ชันที่รับค่าพารามิเตอร์มาเป็น int 2 ตัว และไม่มีการส่งค่าใด ๆ ออกมายังฟังก์ชัน (void) เมื่อเรียกใช้ฟังก์ชัน maxData

<pre> 1 #include&lt;stdio.h&gt; 2 3 void maxData(int, int); 4 5 main() 6 { 7     int a = 8, b = 5; 8     printf("Check max between A(%d) and B(%d)\n", a, b); 9     maxData(a, b); 10 } 11 12 void maxData(int num1, int num2) 13 { 14     if(num1 == num2)          printf("Data %d = %d", num1, num2); 15     else if(num1 &gt;= num2)    printf("Data max is %d", num1); 16     else                      printf("Data max is %d", num2); 17 }</pre>	
---	--

จากตัวอย่างด้านบน สามารถเขียนได้โดยไม่ต้องประกาศโปรแกรมไทยปีแบบตัวอย่างหน้าถัดไป โดยผลลัพธ์ที่ได้มีค่าเท่ากัน

```

1 #include<stdio.h>
2 void maxData(int num1, int num2)
3 {
4     if(num1 == num2)           printf("Data %d = %d", num1, num2);
5     else if(num1 > num2)      printf("Data max is %d", num1);
6     else                      printf("Data max is %d", num2);
7 }
8 main()
9 {
10    int a = 8, b = 5;
11    printf("Check max between A(%d) and B(%d)\n", a, b);
12    maxData(a, b);
13 }

```

ตัวอย่างต่อไป เป็นการสร้างฟังก์ชัน testReturn เพื่อนำมาใช้ในฟังก์ชัน main ฟังก์ชัน testReturn ถูกสร้างไว้ หลัง main จึงต้องประกาศໂປຣໂടໄທປเพื่อให้คอมไไฟเลอร์รู้จักว่า มีฟังก์ชัน testReturn อยู่ในโปรแกรม หากไม่ประกาศ หลังจากเริ่มรันโปรแกรมตัวໂປຣແກຣມจะໄລ່ທ່າງຈາກຂໍ້ໄປຂວາ ນັ້ນລົງລ່າງເນື້ອທ່າງທີ່ຝຶກໜັນ main ຈະເຊື່ອການເຮັດໃຫ້ ພຶກໜັນ testReturn ທີ່ຄວນໄພເລອຮີໄນ້ຮັບຈັກ (ຄລ້າຍ ໆ ກັນການໃຊ້ຕົວແປຣ ແຕ່ໄນ້ໄດ້ປະກາສວ່າຈະໃຫ້..)

ຝຶກໜັນ testReturn ທີ່ສ້າງຈະເປັນຝຶກໜັນທີ່ຮັບຄ່າຕົວເລີນ int ໄປ 1 ຕົວ (1 ພາຮາມີເຕେອຣ) ໂດຍໃນຝຶກໜັນຈະເປັນການ ນ້າຕົວເລີນໄປເຂົ້າເຂົ້າເວັນໄຂວ່າເລີນນີ້ ເປັນເລີນຈ່ານວຸນເຕັ້ນນັກ (Positive) ເປັນເລີນຈ່ານວຸນເຕັ້ນລົບ (Negative) ມີເປົ້າເປັນ ເລີນຈ່ານວຸນເຕັ້ນສູນຍົງ (Zero) ອາກດຽວຕາມເວັນໄທແສດງຂ້ອຄວາມກາຍໃນເວັນໄທນີ້ ແລະສັງຄ່າກໍລັບອອກນາເປັນອັກຂະຮັດ ແກ່ຂອງຈ່ານວຸນເຕັ້ນ (P, N, Z)

ໃນຝຶກໜັນຫລັກ (main) ຈະແສດງຄົງການເຮັດໃຫ້ຝຶກໜັນ testReturn ໃນຮູບແບບດ່າງ ໆ ເຊັ່ນ  
ໃນນຽທດທີ່ 7 ເປັນການນ້າຄ່າ 112 (ASCII ຂອງ p) ທີ່ໄດ້ຈາກການສັງຄ່າກໍລັບຂອງ testReturn(9) ກໍານົດລົງຕົວແປຣ test  
ຈາກນັ້ນນ້າຕົວແປຣ test ໄປແສດງຜລອອກທາງໜ້າຈອໃນນຽທດທີ່ 10

ໃນນຽທດທີ່ 8 ເປັນການເຮັດໃຫ້ຝຶກໜັນເພື່ອນ້າຄ່າທີ່ເທິ່ງອອກນາໄປແສດງຜລ ໂດຍຕຽງໄມ່ເກັນຜ່ານຕົວແປຣ ແຕ່

ໃນນຽທດທີ່ 13 ເປັນການເຮັດໃຫ້ຝຶກໜັນເພື່ອນ້າຄ່າທີ່ເທິ່ງອອກນາໜຶ່ງກີ້ວີ 110 (ASCII ຂອງ z) ໄປແສດງຜລໂດຍຕຽງໄມ່ເກັນ  
ຜ່ານຕົວແປຣ ແຕ່

<pre> 1 #include&lt;stdio.h&gt; 2 3 char testReturn(int); 4 5 main() 6 { 7     char test = testReturn(9); 8     testReturn(0); 9     printf("%c - Positive\n", test); 10    printf("%c - Negative\n", testReturn(-1)); 12 } 13 14 char testReturn(int num) 15 { 16     if(num &gt; 0) { 17         printf("print in testReturn - case if (%d)\n", num); 18         return 'p'; 19     } 20     else if(num &lt; 0) { 21         printf("print in testReturn - case else-if (%d)\n", num); 22         return 'n'; 23     } 24     else { 25         printf("print in testReturn - case else (%d)\n", num); 26         return 'z'; 27     } 28 } </pre>	<pre> C:\Users\...\Desktop\Untitled1.exe print in testReturn - case if (9) print in testReturn - case else(0) p - Positive print in testReturn - case else-if (-1) n - Negative  Process exited after 0.2046 seconds with return value 0 Press any key to continue . . </pre>
---	---

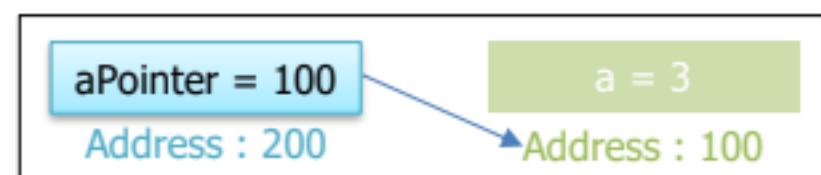
## พอยเตอร์ (Pointer)

พอยน์เตอร์เป็นรูปแบบตัวแปรประเภทหนึ่งที่ใช้อ้างอิง หรือที่เรียกว่า “จี้” (Point) โดยพอยน์เตอร์จะอ้างอิงไปยังตำแหน่ง (Address) ที่อยู่ของข้อมูล ซึ่งในที่นี่คือตำแหน่งของตัวแปรที่อยู่ภายใต้ความจำ การนำพอยน์เตอร์มาใช้งานสามารถทำให้โปรแกรมทำงานได้เร็วขึ้นอีกด้วย

	Address		Name
int y = 5;	0x600000	5	y
		:	
int x = 10;	0x59900F	10	x
		:	



ภาพการเก็บค่าของตัวแปรในหน่วยความจำ



ภาพการใช้ตัวแปรพอยน์เตอร์ (aPointer) อ้างอิงตัวแปรอื่น

### การประกาศตัวแปรประเภทพอยน์เตอร์ (Pointer Variable Declaration)

```
||| dataType *pointerName;
```

โดยที่ **dataType** เป็นชนิดของข้อมูล  
pointerName เป็นชื่อตัวแปรพอยน์เตอร์

เช่น **int \*this\_is\_a\_pointer;**  
**float \*eiei;**  
**char \*pt\_name;**

**ข้อสังเกต :** การสร้างตัวแปรพอยน์เตอร์ต้องใช้ \* นำหน้าชื่อตัวแปร และจะต้องประกาศประเภทข้อมูล (Data Type) ให้ตรงกับข้อมูลที่จะอ้างอิงถึง โดยตัวแปรพอยน์เตอร์จะเก็บเลขที่อยู่ (Address) ในหน่วยความจำของตัวแปรที่ซึ่งในรูปแบบของเลขฐาน 16 (Hexadecimal)

### การกำหนดค่าให้ตัวแปรพอยน์เตอร์ (Pointer Variable Assignment)

#### 1. กำหนดค่าตอนประกาศตัวแปร

```
||| dataType *pointerName = &varName;
```

โดยที่ **pointerName** เป็นชื่อตัวแปรพอยน์เตอร์  
**varName** เป็นตัวแปรที่ต้องการให้พอยน์เตอร์อ้างอิงถึง

เช่น **int \*this\_is\_a\_pointer = &myVariable;**  
**float \*eiei = &variable;**  
**char \*pt\_name = &name;**

#### 2. กำหนดค่าภายในฟังก์ชัน (ภายหลังจากการประกาศตัวแปร)

```
||| pointerName = &varName;
```

โดยที่ **pointerName** เป็นชื่อตัวแปรพอยน์เตอร์  
**varName** เป็นตัวแปรที่ต้องการให้พอยน์เตอร์อ้างอิงถึง

เช่น **eiei = &variable;**  
**pt\_name = &name;**

**ข้อควรรู้ :** การใส่ Ampersand (&) นำหน้าตัวแปรเพื่ออ้างอิงที่ถึงอยู่ (Address) บนหน่วยความจำของตัวแปรนั้น

```
1 #include<stdio.h>
2
3 main()
4 {
5     int a = 1, *aPointer;
6     aPointer = &a;
7 }
8
```

```
1 #include<stdio.h>
2
3 main()
4 {
5     int a = 1;
6     printf("Address a = %#x\n", &a);
7     printf("Value a = %d", a);
8 }
```

C:\Users\...\Desktop\Untitled1.exe  
Address a = 0x62fe1c  
Value a = 1  
Process exited after 0.1451  
seconds with return value 0  
Press any key to continue . . .

จากหัวข้อการประกาศตัวแปร และกำหนดค่าที่ผ่านมา สามารถยกตัวอย่างได้ด้านล่างด้านข้าง

## การดำเนินการทางคณิตศาสตร์กับพอยน์เตอร์ (Pointer Arithmetics)

การดำเนินการกับพอยน์เตอร์ในที่นี่ เป็นการใช้งานตัวดำเนินการทางคณิตศาสตร์กับพอยน์เตอร์ โดยใช้ตัวดำเนินการ `+`, `-`, `++` และ `--` ซึ่งผลที่ได้จะเป็นการเลื่อนตำแหน่งที่อยู่ (Address) ของพอยน์เตอร์ หรืออธิบายตามความหมายของตัวดำเนินการ คือการกำหนดให้พอยน์เตอร์ชี้ไปยังตำแหน่งที่อยู่ที่สูงขึ้น หรือชี้ไปยังตำแหน่งที่อยู่ที่ต่ำลง

```

1 #include<stdio.h>
2 main()
3 {
4     int a, b;
5     int *pt1 = &a, *pt2 = &b;
6     printf("pt1 = %p\n", pt1);
7     printf("pt2 = %p\n\n", pt2);
8     pt1--;
9     pt2++;
10    printf("pt1 = %p (pt1--;)\n", pt1);
11    printf("pt2 = %p (pt2++;)\n\n", pt2);
12    pt1 -= 2;
13    pt2 += 2;
14    printf("pt1 = %p (pt1 -= 2;)\n", pt1);
15    printf("pt2 = %p (pt2 += 2;)", pt2);
16 }

```

```

C:\Users\1\Desktop\Untitled1.exe - * x
pt1 = 0062FE0C
pt2 = 0062FE08

pt1 = 0062FE08 (pt1--)
pt2 = 0062FE0C (pt2++)

pt1 = 0062FE00 (pt1 -= 2)
pt2 = 0062FE14 (pt2 += 2)

Process exited after 0.02179
seconds with return value 0
Press any key to continue . .

```

	b	a				
Value						
Address	62FE00	62FE04	62FE08	62FE0C	62FE10	62FE14

^  
pt2      ^  
pt1

<<< เมื่อจบการ  
ทำงานบรรทัดที่ 5

	b	a				
Value						
Address	62FE00	62FE04	62FE08	62FE0C	62FE10	62FE14

^  
pt1      ^  
pt2

<<< เมื่อจบการ  
ทำงานบรรทัดที่ 9

	b	a				
Value						
Address	62FE00	62FE04	62FE08	62FE0C	62FE10	62FE14

^  
pt1      ^  
pt2

<<< เมื่อจบการ  
ทำงานบรรทัดที่ 13

## การอ้างถึงข้อมูลภายในตัวแปรพอยน์เตอร์ (Dereference Operator)

เพื่อเข้าถึงข้อมูลที่ตัวแปรพอยน์เตอร์อ้างอิง (ชี้) อยู่ จะใช้เครื่องหมาย `*` นำหน้าตัวแปรพอยน์เตอร์นั้น ดังรูปแบบด้านล่างนี้

III \*pointerName

จากตัวอย่างด้านล่างเป็นการยกตัวอย่างพอยน์เตอร์ พร้อมแสดงตำแหน่งที่อยู่ (Address) อ้างถึง

```

1 #include<stdio.h>
2 main()
3 {
4     int a = 30;
5     int b = a;
6     int *c;
7     c = &a;
8     printf("%d", *c);
9 }

```

เมื่อผ่านบรรทัดที่ 4 ในหน่วยความจำผู้เขียนหนังสือแสดงได้ดังนี้ (เลข Address ทั้งหมดอยู่ในรูปฐาน 16)

Name	a
Value	30
Address	0FFC
Value	30
Address	1000
Value	30
Address	1004
Value	30
Address	1008
Value	30
Address	100C

เมื่อจับบรรทัดที่ 5 และ 6 จะเป็นดังในตารางด้านข่าย และข่าวด้านล่างต้น

Name	<b>b</b>		<b>a</b>	
Value		30	30	
Address	0FFC	1000	1004	1008

Name	<b>c</b>	<b>b</b>	<b>a</b>	
Value		30	30	
Address	0FFC	1000	1004	1008

เมื่อจับบรรทัดที่ 7 จะได้ค่าต่าง ๆ ทั้งค่าที่เก็บอยู่ในตัวแปร และตำแหน่งบนหน่วยความจำดังนี้

Name	<b>c</b>	<b>b</b>	<b>a</b>	
Value	1008	30	30	
Address	0FFC	1000	1004	1008

และเมื่อจับบรรทัดที่ 8 จะแสดงค่าที่ตัวแปรพอยน์เตอร์ c อ้างถึงอยู่ ในที่นี่คือ 30 (ค่าที่อยู่ในตัวแปร a)

**ข้อควรรู้ :** Address ที่แสดงในตัวอย่างเป็นการสมมติ และตัวแปร int ในคอมไพเลอร์ของผู้เขียนมีขนาด 4 ไบต์ เลข Address หลังจากประกาศตัวแปรจึงขยับลดลงทีละ 4 (บางคอมไพเลอร์จะขยับขึ้น)

**ข้อควรระวัง :** หากไม่ใส่ \* นำหน้าชื่อตัวแปรจะเป็นการเรียกเลขที่อยู่ (Address) ในหน่วยความจำของตัวแปรนั้นแทนการเรียกค่าที่พอยน์เตอร์นั้น อ้างอิง (ชี้) อยู่ ตัวอย่างด้านขวา

```

1 #include<stdio.h>
2
3 main()
4 {
5     int a = 1;
6     int *aPointer = &a;
7     printf("%d\n", aPointer);
8     printf("%d", *aPointer);
9 }
```

## การใช้งานพอยน์เตอร์

สามารถใช้อ้างถึงข้อมูลภายในตัวแปรพอยน์เตอร์ตามที่กล่าวไว้ในหัวขอก่อนหน้า ซึ่งการอ้างถึงข้อมูลแบบนี้ แทนจะไม่ต่างกับการใช้งานตัวแปรปกติ แต่ส่วนมากจะใช้งานควบคู่ไปกับอาร์เรย์ หรืออีกกรณีที่ใช้งานกันมาก คือการเปลี่ยนแปลงการอ้างอิงของตัวแปรพอยน์เตอร์ ซึ่งเป็นการใช้งานเกี่ยวกับการจัดการหน่วยความจำ โดยเนื้อหาดังกล่าวจะไม่ลงรายละเอียดในเอกสารชุดนี้

## การใช้งานกับพอยน์เตอร์ 1 ตัว

จากตัวอย่างด้านล่างเป็นการใช้ตัวแปรพอยน์เตอร์ชื่อ numberPt เพื่อแสดงค่าที่เก็บอยู่ในตัวแปร number ซึ่งไม่เหมือนกับการแสดงผลของค่าตัวแปร number และสามารถแสดงความสัมพันธ์ระหว่างตัวแปร และที่อยู่ (Address) ได้ถูกต้อง

numberPt = 0X61FF18

Address = 0X61FF14

ภาพแสดงความสัมพันธ์ระหว่างตัวแปร และที่อยู่ (Address) หลังจนโปรแกรม

a = 5

```

1 #include<stdio.h>
2
3 main()
4 {
5     int number = 5;
6     int *numberPt; //Declaring pointer for int data type
7     numberPt = &number; //Point"(Make reference) to 'number' int variable
8     printf("numberPt point to address %p\n", numberPt); //Getting 'number' address
9     printf("number has value equal to %d", *numberPt);
10    //Getting 'number' value via 'numberPt'
11 }
```

## การใช้งานกับพอยน์เตอร์หลายตัว

### ตัวอย่างการใช้งานกับพอยน์เตอร์หลายตัวอ้างอิงถึงตัวแปรอื่น

```

1 #include<stdio.h>
2 main()
3 {
4     int a = 5, b = 10;
5     int *aPointer, *bPointer; //making references to int variables
6     printf("Use pointer to sum two value : a = %d, b = %d, a + b = %d", *aPointer,
7     *bPointer, *aPointer + *bPointer);
8 }
```

```

C:\Users\...\Desktop\Untitled1.exe
Use pointer to sum two value : a = 5, b = 10, a + b = 15
Process exited after 0.365 seconds with return value 0
Press any key to continue . . .

```

สำหรับตัวอย่างนี้มีการอ้างอิงตัวแปร 2 ตัว คือ a ด้วย aPointer และตัวแปร b ด้วย bPointer การแสดงผลลัพธ์ทำโดยใช้พอยน์เตอร์เข้าถึงค่าที่อ้างอิงโดยใช้ \* นำหน้าชื่อตัวแปรพอยน์เตอร์

aPointer = 0X620000

Address = 0X620004

a = 5

Address = 0X62000C

bPointer = 0X620004

Address = 0X620000

b = 10

Address = 0X620008

ภาพแสดงความสัมพันธ์ระหว่างตัวแปร และที่อยู่ (Address) หลังจบโปรแกรม

### ตัวอย่างการใช้พอยน์เตอร์หลายตัวอ้างอิงถึงตัวแปรตัวเดียวกัน

```

1 #include<stdio.h>
2 main()
3 {
4     int a = 5;
5     int *pointer1, *pointer2; //referencing same variables (a)
6     printf("pointer1 point to address %p has value of %d\n", pointer1, *pointer1);
7     printf("pointer2 point to address %p has value of %d\n", pointer2, *pointer2);
8     printf("\nChanging 'a' value with Pointer1 to 999\n");
9     *pointer1 = 999;
10    printf("pointer1 point to address %p has value of %d\n", pointer1, *pointer1);
11    printf("pointer2 point to address %p has value of %d\n", pointer2, *pointer2);
12 }
```

```

C:\Users\...\Desktop\Untitled1.exe
pointer1 point to address 0061FF1C has value of 5
pointer2 point to address 0061FF1C has value of 5

Changing 'a' value with Pointer1 to 999
pointer1 point to address 0061FF1C has value of 999
pointer2 point to address 0061FF1C has value of 999

Process exited after 0.3485 seconds with return value 0
Press any key to continue . . .

```

pointer1 = 0X61FF14

Address = 0X61FF18

a = 999

pointer2 = 0X61FF14

Address = 0X61FF14

Address = 0X61FF1C

ภาพแสดงความสัมพันธ์ระหว่างตัวแปร และที่อยู่ (Address) หลังจบโปรแกรม

ตัวอย่างนี้มีการใช้งานพอยน์เตอร์ 2 ตัวอ้างอิงถึงตัวแหน่งของตัวแปร a เมื่อมองกัน ข้อสังเกตคือ เมื่อค่าที่ถูกอ้างอิงถึงมีการเปลี่ยนแปลง ค่าที่พอยน์เตอร์อ้างอิงถึง ก็จะเปลี่ยนแปลงตามไปด้วยทุกตัว ลักษณะการ "ซื้้" ในตอนจบการทำงานจะดังรูปด้านขวา

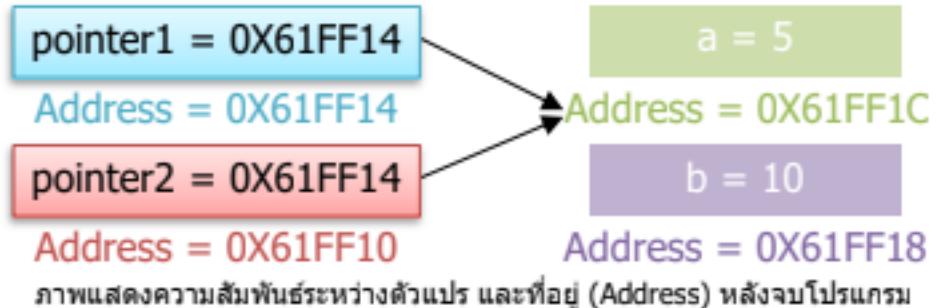
### ตัวอย่างการอ้างอิงพอยน์เตอร์อีกด้วยตัวหนึ่ง โดยใช้พอยน์เตอร์

```

1 #include<stdio.h>
2 main()
3 {
4     int a = 5, b = 10;
5     int *pointer1, *pointer2;
6     printf("pointer1 point to address %p has value of %d\n", pointer1, *pointer1);
7     printf("pointer2 point to address %p has value of %d\n", pointer2, *pointer2);
8     printf("\nAssign pointer2 = pointer1; \n ");
9     pointer2 = pointer1;
10    printf("pointer1 point to address %p has value of %d\n", pointer1, *pointer1);
11    printf("pointer2 point to address %p has value of %d\n", pointer2, *pointer2);
12 }
```

```
C:\Users\...\Desktop\Untitled1.exe
pointer1 point to address 0061FF1C has value of 5
pointer2 point to address 0061FF18 has value of 10

Assign pointer2 = pointer1;
pointer1 point to address 0061FF1C has value of 5
pointer2 point to address 0061FF1C has value of 5
Process exited after 0.1751 seconds with return value 0
Press any key to continue . . .
```



ตัวอย่างนี้มีหลักการทำงานใกล้เคียงกับตัวอย่างที่แล้ว แต่เปลี่ยนรูปแบบการอ้างอิงจากเดิม ใช้การอ้างอิงไปที่ตัวแปรโดยตรง โดยเป็นการอ้างอิงค่าที่อยู่ (Address) จากพอยน์เตอร์อื่นแทน ซึ่งในตัวอย่างเมื่อ pointer1 อ้างอิงไปถึงตัวแปร a และทำการดำเนินการ pointer2 = pointer1 ตัวพอยน์เตอร์ pointer2 ก็จะอ้างอิงไปที่ตัวแปร a เช่นกัน

### การใช้งานพอยน์เตอร์กับอาร์เรย์

การใช้งานพอยน์เตอร์คู่กับอาร์เรย์ นิยมใช้กันมากเนื่องจากพอยน์เตอร์สามารถใช้อ้างอิงตำแหน่งที่อยู่ของตัวแปร อาร์เรย์ได้ โดยปกติจะเป็นการอ้างอิงถึงตำแหน่งที่อยู่ของตัวแปรอาร์เรย์ตำแหน่งแรกเท่านั้น (อินเด็กซ์เป็น 0)

#### การกำหนดตัวแปรพอยน์เตอร์อ้างอิงไปยังอาร์เรย์

การประกาศให้พอยน์เตอร์อ้างอิงไปที่อาร์เรย์ตอนประกาศตัวแปรสามารถทำได้ดังนี้

```
dataType *pointerName = arrayName;
```

หรือสามารถกำหนดค่าให้พอยน์เตอร์หลังจากประกาศตัวแปรได้ดังนี้

```
pointerName = arrayName;
```

จะสังเกตได้ว่าไม่ต้องใส่ & หน้าตัวแปรที่ต้องการดึง Address ให้ตัวแปรพอยน์เตอร์ใช้สำหรับอ้างอิง เนื่องจาก ตัวแปรประเภทอาร์เรย์มีลักษณะคล้ายพอยน์เตอร์อยู่แล้ว และเมื่อกำหนดค่าในรูปแบบนี้ตัวแปรพอยน์เตอร์จะเริ่มอ้างอิง ในอินเด็กซ์ที่ 0 หากต้องการอ้างอิงที่อินเด็กซ์อื่น ๆ ให้กำหนดค่าดังรูปแบบนี้

```
dataType *pointerName = &arrayName[index];
```

```
pointerName = &arrayName[index];
```

การเข้าถึงค่าตัวแปรในแต่ละอินเด็กซ์สามารถทำได้ 2 รูปแบบดังนี้

```
*(pointerName + index);
```

```
pointerName[index];
```

**ข้อสังเกต :** การเข้าถึงในแบบแรกจะเป็นค้องมี ( ) เพื่อเลื่อนตำแหน่งของตัวแปรพอยน์เตอร์ ถ้าไม่มีวงเล็บพอยน์เตอร์ จะทำการอ้างอิงค่าที่ตัวแปรพอยน์เตอร์นั้นชี้อยู่ และทำการเพิ่มค่าให้กับตัวที่ถูกชี้แทนการเลื่อนตำแหน่งพอยน์เตอร์

จากตัวอย่างโค้ดด้านล่าง ทั้งสองโค้ดเป็นการใช้พอยน์เตอร์กับอาร์เรย์ ซึ่งผลลัพธ์ที่ได้มีค่าเท่ากัน และได้ตารางแสดงค่าที่เก็บอยู่ในตำแหน่งต่าง ๆ ตามที่อยู่ (Address)

	a[0]	a[1]	a[2]	a[3]	a[4]
Value	12	25	30	42	53
Address	62FDF0	62FDF4	62FDF8	62FDFA	62FE00
	aPointer	aPointer	aPointer	aPointer	aPointer

```
1 #include<stdio.h>
2 main()
3 {
4     int a[] = {12, 25, 30, 42, 53};
5     int *aPointer = a; //point to array (Don't need "&" in front of the array)
6     for(int i=0; i<5; i++)    printf("%d ", *(aPointer + i));
7 }
```

```
C:\Users\...\Desktop\Untitled1.exe
12 25 30 42 53
Process exited after 0.02372 seconds with return value 0
Press any key to continue . . .
```

```

1 #include<stdio.h>
2 main()
3 {
4     int a[] = {12, 25, 30, 42, 53};
5     int *aPointer = a; //point to array (Don't need "&" in front of the array)
6     for(int i=0; i<5; i++)    printf("%d ", aPointer[i]);
7 }

```

จากตัวอย่างโค้ดด้านล่างเป็นการนำพอยน์เตอร์หลัก ๆ ด้วย (Array of pointer) ให้อ้างอิงข้อมูล และเรียงกันเป็นกลุ่ม

```

1 #include<stdio.h>
2 main()
3 {
4     int man = 30, girl = 25, boy = 10;
5     int *arrayFromPt[3];
6     arrayFromPt[0] = &man;
7     arrayFromPt[1] = &girl;
8     arrayFromPt[2] = &boy;
9     for(int i=0; i<3; i++)    printf("arrayFromPt[%d] = %d\n", i, *arrayFromPt[i]);
10 }

```

**ข้อควรระวัง :** ควรระวังการใช้ ++ และ -- เพื่อเพิ่ม หรือลดค่าโดยใช้พอยน์เตอร์เพื่ออ้างอิงค่าของตัวแปรนั้น เพราะตัวดำเนินการ ++ และ -- มีลำดับความสำคัญของตัวดำเนินการ (Precedence) สูงกว่าพอยน์เตอร์ จากตัวอย่างด้านล่างจึงทำการเลื่อนตำแหน่งที่พอยน์เตอร์ขึ้นแทนการเพิ่มค่าตัวแปรที่พอยน์เตอร์อ้างอิงถึงอยู่ แก้ด้วยการใส่ () เพื่อให้พอยน์เตอร์ทำการอ้างอิงค่านั้นก่อนแล้วจึงค่อยเพิ่ม หรือลดค่า (\*ptA)++;

```

1 #include<stdio.h>
2 main()
3 {
4     int a = 1;
5     int *ptA = &a;
6     printf("a = %d, ptA = %d, *ptA = %d\n", a, ptA, *ptA);
7     *ptA++;
8     printf("a = %d, ptA = %d, *ptA = %d", a, ptA, *ptA);
9 }

```

นอกจากนี้ยังสามารถส่งผ่านอาร์เรย์ไปให้ฟังก์ชันโดยใช้พอยน์เตอร์ได้เช่นกัน ซึ่งอยู่ในหัวข้อต่อไป

### การใช้งานพอยน์เตอร์กับข้อความ

ในภาษาซีไม่มีตัวแปรประเภท string ดังนั้นการทำงานกับข้อความจึงเป็นการใช้สายอักขระ (Char Array หรือ Group of Character Type) แทน การใช้งานพอยน์เตอร์กับข้อความจึงแทนจะเหมือนการใช้งานกับอาร์เรย์ปกติได้เลย โดยความพิเศษของการใช้พอยน์เตอร์กับข้อความ คือการที่สามารถแสดงผลข้อความทั้งหมดผ่านการใช้ %s ได้ในทันที ตามตัวอย่างการใช้ด้านไป

```

1 #include<stdio.h>
2 #include<string.h>
3 main()
4 {
5     char message[] = "hello there!";
6     char *messagePointer = message;
7     printf("Print character one by one\n");
8     for(int i=0; i<strlen(message); i++) {
9         printf("%c", *(messagePointer + i));
10    }
11    printf("\nPrint as a string\n");
12    printf("%s", messagePointer); //Don't use '*' in front of the pointer
13 }

```

```

C:\Users\...\Desktop\Untitled1.exe
Print character one by one
hello there!
Print as a string
hello there!

```

## การใช้งานพอยน์เตอร์อ้างอิงพอยน์เตอร์ที่กำลังอ้างอิงถึงตัวแปรอื่น

พอยน์เตอร์สามารถนำมารอ้กันเองได้หลาย ๆ ชั้นได้เหมือนตัวอย่างด้านล่างนี้

```

1 #include<stdio.h>
2
3 main()
4 {
5     int man = 10;
6     int *ptA = &man;
7     int **ptB = &ptA;
8     int ***ptC = &ptB;
9     *(*(*ptC)) = 8888;
10    printf("Value of man = %d", man);
11 }
```

### พอยน์เตอร์กับฟังก์ชัน

#### การส่งค่าที่เก็บอยู่ในตัวแปรให้กับฟังก์ชัน (Pass by Value)

คือการส่งค่าหรือค่าที่เก็บในตัวแปรเป็นอาร์กิว เมนต์ส่งไปยังพารามิเตอร์ของฟังก์ชันปลายทาง หากมีการเปลี่ยนแปลงค่าในฟังก์ชันปลายทาง ค่าที่เปลี่ยนแปลงนั้นจะไม่ส่งผลต่อค่าในฟังก์ชันหลักหรือฟังก์ชันอื่น ๆ ซึ่งจะเปรียบเสมือนการคัดลอกข้อมูลไปใช้ในฟังก์ชัน โดยที่ข้อมูลเดิมจะยังไม่เปลี่ยนแปลง การใช้งานจะใช้ชื่อตัวแปรที่ต้องการส่งค่าได้เลย (ไม่มี \* หรือ & หน้าตัวแปร) ดังตัวอย่างด้านขวา

```

1 #include<stdio.h>
2
3 void function(int a, int b)
4 {
5
6 }
7
8 main()
9 {
10    function(10, 20);
11 }
```

#### การส่งค่าที่อยู่ของตัวแปรให้กับฟังก์ชัน (Pass by Pointer)

คือการใช้ค่าตำแหน่งของข้อมูล (Value Address) เป็นอาร์กิว เมนต์ส่งไปยังพารามิเตอร์ของฟังก์ชันปลายทาง (ในที่นี้ส่งค่าที่อยู่ของตัวแปร a และ b)

จากตัวอย่างด้านขวาภายในฟังก์ชันจะทำการสร้างพอยน์เตอร์ที่ชี้ไปที่เดียวกันกับค่าหนึ่งที่ได้รับมาเป็นอาร์กิวเมนต์ i และ j ถ้าหากมีการเปลี่ยนแปลงค่า i หรือ j ค่าตำแหน่งเดิมในฟังก์ชันหลักจะไม่ถูกเปลี่ยนแปลง แต่ถ้าหากทำการเปลี่ยนแปลงค่าที่พอยน์เตอร์ i และ j ข้อมูลจะส่งผลให้ตัวแปร a หรือ b เกิดการเปลี่ยนแปลงค่าตามด้วยนั่นเอง

```

1 #include<stdio.h>
2
3 void swapnum(int *i, int *j)
4 {
5     int temp = *i;
6     *i = *j;
7     *j = temp;
8 }
9
10 main()
11 {
12     int a = 10, b = 20;
13     swapnum(&a, &b);
14 }
```

#### การส่งการอ้างอิงถึงตัวแปรให้กับฟังก์ชัน (Pass by Reference)

คือการส่งการอ้างอิงถึงตัวแปรที่เป็นอาร์กิวเมนต์ไปยังพารามิเตอร์ของฟังก์ชันปลายทาง โดยการอ้างอิงจะเป็นการสร้างนามแฝงในกับตัวแปรนั้น ๆ เพื่อใช้ภายในฟังก์ชัน มืออยู่ในภาษา C++ เท่านั้น

```

1 #include<stdio.h>
2
3 void swapnum(int &i, int &j)
4 {
5     int temp = i;
6     i = j;
7     j = temp;
8 }
9
10 main() //Only C++
11 {
12     int a = 10, b = 20;
13     swapnum(a, b);
14 }
```

## โครงสร้างข้อมูล (Structure)

คือ “ตัวแปรประเภทโครงสร้าง” สามารถใช้ได้ในการจัดกลุ่มข้อมูลที่มีความเกี่ยวข้องกันไว้ด้วยกัน คือมีตัวแปรหลายตัว หลายชนิดอยู่一块ใน ยกตัวอย่างนักเรียนจะมีข้อมูลเช่น ชื่อ นามสกุล อายุ เกรด ห้องเรียน เลขที่ เพศ เป็นต้น คุณสมบัตินี้ถูกเอาไปต่อ�่อในเชิง Object Oriented Programming (OOP) เรียกว่า คลาส (Class) ซึ่งมีอยู่ในภาษาอื่นๆ เช่น C++, Java, Python เป็นต้น

### การนิยามโครงสร้างเพื่อใช้งาน (Structure Define)

```
struct StructName {
    dataType var_1;
    dataType var_2;
    :
    dataType var_n;
};
```

โดยที่ struct เป็นคีย์เวิร์ดในการนิยามโครงสร้างขึ้น (Structure)  
 StructName เป็นชื่อที่ใช้เรียกโครงสร้างที่นิยามขึ้น โดยการตั้งชื่อให้เป็นสากลจะชี้ชัดว่าพิมพ์ใหญ่  
 dataType เป็นชนิดของข้อมูลภายในโครงสร้างขึ้นมา  
 var\_1 ... var\_n เป็นชื่อตัวแปรภายในตัวแปรประเภทโครงสร้าง

### การประกาศตัวแปรประเภทโครงสร้าง (Structure Variable Declaration)

```
struct StructName {
    dataType var1;
    dataType var2;
    :
    dataType varn;
}struct_var_1, struct_var_2, ..., struct_var_n;
```

โดยที่ struct\_var\_1 ... struct\_var\_n เป็นชื่อตัวแปรประเภทโครงสร้าง

หรือสามารถนิยามของโครงสร้างไว้ก่อน และจึงค่อยประกาศตัวแปรประเภทโครงสร้างทีหลังดังรูปแบบด้านล่างนี้

```
struct StructName struct_var_1, struct_var_2, ..., struct_var_n;
```

จากโค้ดตัวอย่างด้านล่างเป็นการยกตัวอย่างการสร้าง และประกาศตัวตัวแปรประเภทโครงสร้าง โดยโครงสร้างดังกล่าวชื่อ School และมีตัวแปรประเภทโครงสร้างชื่อ student และ teacher ภายในโครงสร้างจะมีฟิลด์ข้อมูล (Data Field) คือตัวแปรหลายประเภทสำหรับเก็บข้อมูลต่างๆ เช่น ตัวแปร name เก็บชื่อผู้ใช้งานโดยให้ความยาวไม่เกิน 19 ตัวอักษร ตัวแปร age เก็บอายุ ผู้ใช้งาน เป็นต้น

```
1 struct School {
2     char name[20];
3     char lastName[20];
4     short age;
5     char gender;
6     char tel[11];
7     int id;
8 } student, teacher;
9
```

```
1 struct School {
2     char name[20];
3     char lastName[20];
4     short age;
5     char gender;
6     char tel[11];
7     int id;
8 };
9 struct School student, teacher;
```



struct  
School

Structure variable  
name  
student, teacher

Data Fields  
name  
lastName  
age  
gender  
tel  
id

ข้อสังเกต : สามารถใช้คีย์เวิร์ด **typedef** นำหน้าตอนนิยาม struct เพื่อลดรูปการประกาศตัวแปรประเภทโครงสร้างจากเดิมที่ใช้ struct StructName struct\_var; จะเหลือเพียง StructName struct\_var;

จากตัวอย่างด้านขวาเมื่อใช้คีย์เวิร์ด typedef ถ้าจะมองง่ายๆ ให้ถือว่า School หรือ Office เป็นตัวแปรประเภทนึงไปเลย ถ้าประกาศตัวแปรจึงใช้รูปแบบนี้ School student โดยไม่ต้องมี struct นำหน้า

```
1 typedef struct {
2     char name[50];
3     int id;
4 } School, Office;
5 School student, teacher;
6 Office ceo, employee;
```

## การประกาศตัวแปรประเภทโครงสร้างหลายตัวโดยใช้อาร์เรย์

สามารถสร้างตัวแปรประเภทโครงสร้างหลาย ๆ ตัวได้โดยใช้อาร์เรย์ได้ดังรูปแบบด้านล่าง

```
struct StructName {
    dataType var1;
    dataType var2;
    :
    dataType varn;
}struct_var[index];
```

โดยที่ index เป็นจำนวนอินเด็กซ์ของตัวแปรประเภทโครงสร้างที่ต้องการ

การกำหนดค่าเริ่มต้นตอนประกาศตัวแปร การเข้าถึง และกำหนดค่าให้ตัวแปรประเภทโครงสร้าง การเข้าถึงตัวแปรภายใน struct ให้ทำการเข้าถึงโดยใช้ Dot Notation ดังนี้

```
struct_var_1.var1;
```

โดยที่ var\_1 ... var\_n เป็นชื่อตัวแปรภายในตัวแปรประเภทโครงสร้าง

สำหรับการกำหนดค่าเริ่มต้นให้แต่ละตัวแปรภายในโครงสร้างสามารถกำหนดได้ดังนี้

```
struct_var_1 = {var1Value, var2Value, ..., varnValue};
```

โดยที่ var1Value ... varnValue เป็นค่าที่ต้องการกำหนดให้ตัวแปร var\_1 ... var\_n ภายในตัวแปรประเภทโครงสร้างตามลำดับ

การเข้าถึงค่านี้ผู้อ่านสามารถอ่านค่า และเปลี่ยนแปลงค่าได้เหมือนกับตัวแปรประเภทที่เข้าถึงได้ตามปกติ สมมติ var1 ที่กำหนดมีชนิดข้อมูลเป็นประเภทจำนวนเต็ม (int) ก็จะสามารถใช้ตัวดำเนินการทางคณิตศาสตร์ได้ตามปกติ ซึ่งสามารถยกตัวอย่างได้ดังนี้

```
1 #include<stdio.h>
2
3 struct Laptop {
4     char *brand[];
5     unsigned short ram;
6     unsigned short ssd;
7     unsigned int price;
8 };
9
10 main()
11 {
12     struct Laptop laptop = {"JadePC", 8,
13                             256, 20000};
14     printf("%s\n", laptop.brand);
15     printf("RAM = %d\n", laptop.ram);
16     printf("SSD = %d\n", laptop.ssd);
17     printf("%d.-", laptop.price);
18 }
```

```
1 #include<stdio.h>
2
3 struct Laptop {
4     char *brand[];
5     unsigned short ram;
6     unsigned short ssd;
7     unsigned int price;
8 };
9
10 main()
11 {
12     struct Laptop laptop;
13     laptop.brand = "JadePC";
14     laptop.ram = 8;
15     laptop.ssd = 256;
16     laptop.price = 20000;
17     printf("%s\n", laptop.brand);
18     printf("RAM = %d\n", laptop.ram);
19     printf("SSD = %d\n", laptop.ssd);
20     printf("%d.-", laptop.price);
21 }
```

จากตัวอย่างด้านบนจะนิยามโครงสร้างเป็น Global เพื่อใช้งานกับทุก ๆ พัฟ์ชัน โดยฝั่งข้างเป็นการกำหนดค่าเริ่มต้นพร้อมกับประกาศตัวแปรประเภทโครงสร้าง และแสดงผลข้อมูลภายในตัวแปรประเภทโครงสร้าง แต่ในฝั่งขวาเป็นการกำหนดค่าหลังจากประกาศตัวแปรประเภทโครงสร้าง และแสดงผลข้อมูลภายในตัวแปรประเภทโครงสร้างเหมือนเดิม

## การใช้อาร์เรย์ภายในตัวแปรประเภทโครงสร้าง

การใช้อาร์เรย์ภายในตัวแปรประเภทโครงสร้างสามารถใช้ได้เหมือนปกติ ยกตัวอย่างตัวอย่างดังรูปแบบด้านล่างนี้

```

struct StructName {
    dataType var1[index];
    dataType var2[index];
    ...
    dataType varn[index];
} struct_var_1, struct_var_2, ...., struct_var_n;

```

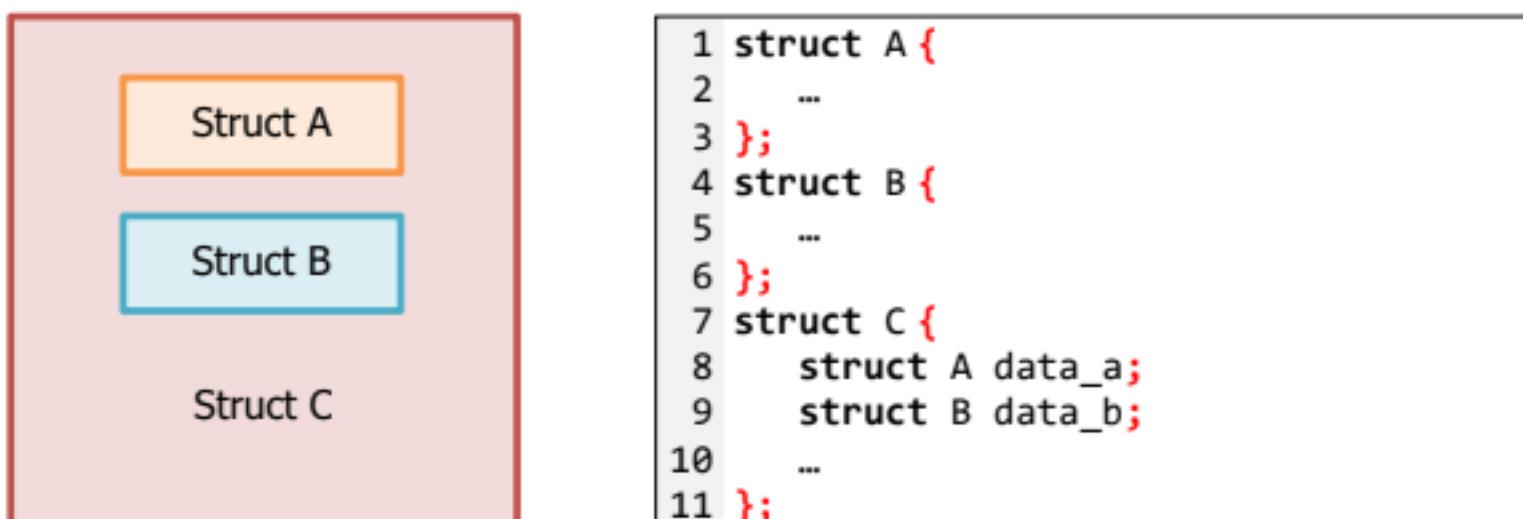
โดยที่ index เป็นจำนวนอินเด็กซ์ของตัวแปรภายในตัวแปรประเภทโครงสร้าง

และการเข้าถึงค่าในแต่ละอินเด็กซ์ที่ต้องการสามารถทำได้ดังนี้

```
struct_var_1.var1[index] ;
```

### โครงสร้างช้อนโครงสร้าง (Nest Structure)

ในภาษาซึ่งสามารถสร้างโครงสร้างช้อนเข้าไปในโครงสร้างได้อีกที่จะเรียกการกระทำนี้ว่า "Nest Structure" เช่น ประกาศโครงสร้าง C โดยภายในโครงสร้าง C ประกอบด้วยโครงสร้าง A และ B ซึ่งมีลักษณะดังรูป



#### ตัวอย่างโค้ด Nest Structure พร้อมรูปจำลองการเก็บข้อมูล

<pre> 1 struct Address { 2     int num; 3     char road[20]; 4     char district[20]; 5     char province[20]; 6 }; 7 struct Phone { 8     char home[10]; 9     char mobile[11]; 10}; 11 </pre>	<pre> 12 struct User { 13     char name[20]; 14     char surname[20]; 15     struct Address addr; 16     struct Phone tel; 17 }; 18 19 20 21 22 </pre>
---	--

name	surname	addr				tel	
		num	road	district	province	home	mobile

## ภาคผนวก

### คำศัพท์น่ารู้

#### หมวดระบบคอมพิวเตอร์

คำศัพท์	ความหมาย
<b>API</b>	ย่อมาจาก Application Programming Interface คือช่องทางการเชื่อมต่อ หรือติดต่อกันโปรแกรม ส่วนรับผู้เขียนโปรแกรมเรียกใช้งานโปรแกรม
<b>Client / Server</b>	การที่มีเครื่องผู้ให้บริการ (Server) และเครื่องผู้ใช้บริการ (Client) เชื่อมต่อกันอยู่ และเมื่อเครื่องผู้ใช้บริการได้มีการติดต่อร้องขอรับบริการจากเครื่องผู้ให้บริการ เครื่องผู้ให้บริการก็จะจัดการตามที่เครื่องผู้ขอใช้บริการร้องขอ หลังจากนั้นจึงส่งข้อมูลกลับไปให้
<b>Compiler</b>	โปรแกรมที่ทำหน้าที่ในการแปลง ภาษาระดับสูงให้เป็นภาษาเครื่องที่คอมพิวเตอร์เข้าใจ
<b>Computer Languages</b>	ภาษาที่ใช้ส่วนรับเขียน เพื่อสร้างโปรแกรมขึ้นมาเพื่อใช้งานบนเครื่องคอมพิวเตอร์
<b>Computer System</b>	ระบบที่พัฒนามาจาก 2 ส่วนใหญ่ ๆ คือ ฮาร์ดแวร์ และซอฟต์แวร์
<b>Hardware</b>	ส่วนประกอบของคอมพิวเตอร์ หรืออุปกรณ์ต่าง ๆ ที่ประกอบขึ้นเป็นเครื่องคอมพิวเตอร์ มีลักษณะเป็นโครงร่างสามารถมองเห็นด้วยตา และสัมผัสได้ (เป็นรูปธรรม) เช่น เครื่องพิมพ์ จอภาพ เม้าส์ คีย์บอร์ด เป็นต้น
<b>IDE</b>	ย่อมาจาก Integrated Development Environment เป็นแอปพลิเคชันซอฟต์แวร์ที่ค่อยอ่านวิเคราะห์ความต้องการให้กับนักเขียนโปรแกรม ส่วนรับการพัฒนาซอฟต์แวร์ เช่น VS Code
<b>Loader</b>	โปรแกรมที่ทำงานร่วมกับ Hardware มีหน้าที่โหลด Source Code ลงไปในหน่วยความจำ
<b>Personal Computer</b>	ระบบคอมพิวเตอร์ที่มีเครื่องคอมพิวเตอร์เครื่องเดียว และไม่ได้ทำการเชื่อมต่อกับเครื่องคอมพิวเตอร์เครื่องอื่น ๆ
<b>SDK</b>	ย่อมาจาก Software Development Kit คือชุดเครื่องมือที่เจาะจงในการเขียนโปรแกรมในอย่างโดยย่างหนัก จะรวมรวมเครื่องมือทุก ๆ อย่างในการเขียนโปรแกรมนั้น ๆ ไว้
<b>Software</b>	ชุดคำสั่ง หรือโปรแกรมที่ใช้ควบคุมการทำงานของเครื่องคอมพิวเตอร์ เขียนขึ้นโดยภาษาคอมพิวเตอร์จากนักเขียนโปรแกรม
<b>System development Life Cycle</b>	กระบวนการพัฒนาระบบงาน หรือระบบเทคโนโลยีสารสนเทศด้วย เพื่อช่วยแก้ปัญหาทางธุรกิจหรือตอบสนองความต้องการขององค์กร
<b>Time Sharing</b>	การเชื่อมต่อของคอมพิวเตอร์หลาย ๆ เครื่องมาต่อกันเครื่องคอมพิวเตอร์ศูนย์กลางที่ค่อยประมวลผลต่าง ๆ โดยใช้หลักการแบ่งเวลา และคอมพิวเตอร์หลาย ๆ เครื่องเหล่านั้นเรียกว่า Terminal

#### หมวดโครงสร้างภาษา

คำศัพท์	ความหมาย
<b>Compiled Language</b>	ภาษาที่มีคอมไพล์เลอร์ (Compiler) ทำหน้าที่ในการคอมไพล์ (Compile) หรือแปลงคำสั่งทั้งหมดในโปรแกรมให้เป็นภาษาเครื่อง (Machine Language) เพื่อให้เครื่องคอมพิวเตอร์น้ำค่าสั่งเหล่านั้นทำงานตามค่าสั่งด่อไป
<b>Constants</b>	ชนิดข้อมูลแบบค่าคงที่ ไม่สามารถเปลี่ยนแปลงในขณะที่โปรแกรมทำงานอยู่
<b>Flowchart</b>	ผังงาน มีไว้เพื่อให้ผู้ใช้ออกแบบขั้นตอนการทำงานของโปรแกรมก่อนเขียนโปรแกรม ซึ่งจะช่วยให้นักเขียนโปรแกรมเขียนโปรแกรมได้ง่ายขึ้น และไม่สับสน
<b>Global Variables</b>	ตัวแปรที่สามารถเรียกใช้ได้ทุกฟังก์ชันของโปรแกรม หากมีการประกาศตัวแปร
<b>Header Files</b>	ส่วนที่เก็บไลบรารีมาตรฐานของภาษา C ซึ่งจะถูกดึงเข้ามาร่วมกับโปรแกรมในขณะที่กำลังทำการคอมไابل์
<b>Local Variables</b>	ตัวแปรที่จะสามารถเรียกใช้ได้ภายในฟังก์ชันที่ประกาศตัวแปรนั้น
<b>Return Value</b>	เป็นค่าที่ส่งกลับ เมื่อฟังก์ชันดังกล่าวถูกเรียก
<b>Statements</b>	ประโยคที่เขียนขึ้นเพื่อให้โปรแกรมทำงาน ประกอบด้วยตัวแปร หรือนิพจน์ต่าง ๆ เพื่อใช้ประกาศตัวแปร กำหนดค่าเริ่มต้น กำหนดเงื่อนไข และใช้ในคำสั่งควบคุม นิพจน์ หรือตัวแปรที่เขียนขึ้นเป็นประโยคคำสั่งส่วนมากจะใช้เครื่องหมาย ; (Semicolon) ปิดท้ายคำสั่ง

## หมวดการแสดงผล และรับข้อมูล

คำศัพท์	ความหมาย
<b>Back-slash character</b>	ชุดค่าคงที่ของตัวอักษร ที่ใช้ในการควบคุมการแสดงผล และใช้ในการช่วยแสดงผลอักษรระบากตัวที่ไม่สามารถแสดงผลออกทางหน้าจอ
<b>getch</b>	คำสั่งในการรับค่าข้อมูลจากผู้ใช้เข้ามาทางคีย์บอร์ด 1 ตัวอักษร โดยที่เคอร์เซอร์จะไม่ขึ้นบรรทัดใหม่ และไม่ต้องกดปุ่ม Enter เมื่อป้อนครบทั้ง 1 ตัวอักษร นอกจานี้ตัวอักษรที่ป้อนลงไปจะไม่แสดงผลออกทางหน้าจอ
<b>getchar</b>	คำสั่งในการรับค่าข้อมูลจากผู้ใช้เข้ามาทางคีย์บอร์ด 1 ตัวอักษร จะต้องกดปุ่ม Enter เพื่อสิ้นสุดการรับข้อมูล ตัวอักษรที่ป้อนลงไปจะแสดงผลออกทางหน้าจอ และเคอร์เซอร์จะขึ้นบรรทัดใหม่
<b>getche</b>	เป็นฟังก์ชันที่ทำงานเหมือนฟังก์ชัน getch แต่จะแสดงตัวอักษรที่พิมพ์เข้าไป ออกทางจอภาพด้วย
<b>printf</b>	คำสั่งที่ใช้สำหรับแสดงค่าต่าง ๆ ออกทางจอภาพ
<b>scanf</b>	คำสั่งในการรับค่าจากผู้ใช้เข้ามาทางคีย์บอร์ด
<b>Standard Input File</b>	ไฟล์ที่ใช้เก็บข้อมูลจากคีย์บอร์ด และเรียงลำดับตามข้อมูลที่ได้รับมาจากผู้ใช้ ก่อนที่จะมีการนำไปใช้ หรือเรียกว่ายัง ๆ ว่าเป็น Buffered
<b>Standard Output File</b>	ไฟล์ที่ใช้เก็บข้อมูล ก่อนที่จะนำไปแสดงออกทางอุปกรณ์ต่าง ๆ ซึ่งจะเป็น Text File เมื่อต้องการแสดงข้อมูลนั้น จะทำการแปลงข้อมูลเป็นข้อความก่อนแล้วจึงค่อยแสดงออกทางจอภาพ

## หมวดนิพจน์ทางคณิตศาสตร์

คำศัพท์	ความหมาย
<b>Comparison Operator</b>	ตัวดำเนินการเปรียบเทียบ เป็นตัวดำเนินการสำหรับเปรียบเทียบข้อมูลระหว่างตัวถูกกระทำทางด้านซ้าย และด้านขวาของตัวดำเนินการ โดยผลลัพธ์ที่ได้จะมีค่าเป็นจริง หรือเท็จเท่านั้น
<b>Compound Assignments</b>	การกำหนดค่าแบบผสม เป็นการรวมของตัวดำเนินการได้ ๆ กับตัวดำเนินการกำหนดค่า (=) ที่ใช้สำหรับอัพเดตค่าให้กับตัวแปรโดยอ้างอิงจากค่าเดิมที่มีอยู่ เช่น การบวกเลขเพิ่มเข้าไปจากค่าเดิม หรือกล่าวอีกนัยหนึ่ง คือสามารถเป็นรูปแบบย่อในการใช้ตัวดำเนินการทางคณิตศาสตร์ เช่น $+=$ , $*=$ , $/=$ , $>=$ , $&=$ , $^=$
<b>Expressions</b>	นิพจน์ ข้อความ หรือประโยคที่เขียนอยู่ในรูปสัญลักษณ์ โดยนำข้อมูล ตัวแปร หรือฟังก์ชัน หรือค่าคงที่มาสัมพันธ์กับตัวดำเนินการอย่างใดอย่างหนึ่ง เช่น $2==2$ เป็นต้น
<b>Logical Operator</b>	ตัวดำเนินการทางด้านตรรกศาสตร์ ใช้สำหรับเชื่อมนิพจน์ เพื่อกำหนดเงื่อนไขมากกว่า 1 เงื่อนไข ซึ่งผลลัพธ์ที่ได้จะมีค่าเป็นจริง หรือเท็จเท่านั้น ซึ่งมีตัวดำเนินการดังนี้ $\&\&$ (AND), $\  \ $ (OR), $!$ (NOT/นิเสธ)
<b>Operand</b>	ตัวถูกกระทำ อาจเป็นค่าคงที่ ตัวแปร นิพจน์ หรือฟังก์ชันก็ได้ โดยจะถูกตัวดำเนินการด้วยตัวดำเนินการ หรือที่เรียกวันในภาษาพูดว่า "เครื่องหมาย" เช่นนิพจน์ $2+3 = 5$ โดย 2 และ 3 เป็น "ตัวถูกดำเนินการ" และ + เป็น "ส่วนตัวดำเนินการ"
<b>Operator</b>	ตัวดำเนินการซึ่งอาจเป็น การดำเนินการทางคณิตศาสตร์ การดำเนินการทางตรรกศาสตร์ หรืออื่น ๆ โดยที่ โอบีโอเรเตอร์มักจะเป็นเครื่องหมายหรือสัญลักษณ์พิเศษต่าง ๆ เช่น $+$ , $-$ , $*$ , $\%$ , $\&\&$
<b>Precedence</b>	ลำดับความสำคัญของตัวดำเนินการ โดยการทำงานจะเริ่มจากลำดับความสำคัญสูงไปยังต่ำสุด ซึ่ง () เรียกว่าวงเล็บ หรือ Identifier จะมีลำดับความสำคัญสูงสุด และ , เรียกว่าคอมม่า หรือ Comma จะมีลำดับความสำคัญต่ำสุดตามลำดับ
<b>Simple Assignments</b>	การกำหนดค่าแบบง่าย อย่างเช่นการกำหนดค่าสมการทางคณิตศาสตร์ทั่วไป
<b>Sizeof Operator</b>	ตัวดำเนินการชนิดนี้จะใช้สำหรับหาขนาดชนิดของข้อมูลชนิดของข้อมูลต่าง ๆ ที่ผู้อ่านต้องการทราบ โดยการใช้ <code>sizeof(DATA)</code> โดยที่ sizeof คือตัวดำเนินการบอกขนาด DATA คือชนิดข้อมูลหรือตัวแปรที่ต้องการทราบข้อมูล
<b>Unary Operator</b>	การดำเนินการทางคณิตศาสตร์โดยมีการกระทำกับ Operand เพียงตัวเดียว เพื่อทำการเพิ่มค่า ลดค่า หรือการสลับค่า (Inverting) ของข้อมูลชนิด Boolean

## หมวดฟังก์ชันของภาษา C

คำศัพท์	ความหมาย
<b>Abs</b>	ส่งค่ากลับเป็นค่า Absolute ของตัวเลขที่เข้ามาในฟังก์ชัน ฟังก์ชัน Absolute คือฟังก์ชันสำหรับหาค่าสัมบูรณ์ของตัวเลข ซึ่งจะได้ผลลัพธ์เป็นตัวเลขที่มีค่าเป็นบวก
<b>Ceil</b>	ฟังก์ชันที่จะส่งกลับเป็นค่าจำนวนเต็มที่มากกว่า หรือเท่ากับค่าที่ส่งไปให้
<b>Floor</b>	ฟังก์ชันที่จะส่งกลับเป็นค่าจำนวนเต็มที่น้อยกว่า หรือเท่ากับค่าที่ส่งไปให้
<b>Function</b>	ส่วนที่เก็บค่าสั่งต่าง ๆ ไว้ซึ่งในภาษา C จะบังคับให้มีฟังก์ชันอย่างน้อย 1 ฟังก์ชัน นั่นก็คือฟังก์ชัน main() และในโปรแกรม 1 โปรแกรมสามารถมีฟังก์ชันได้มากกว่า 1 ฟังก์ชัน
<b>Function Body</b>	ส่วนต่าง ๆ ที่ประกอบขึ้นเป็นฟังก์ชัน เช่น ตัวแปร และค่าสั่งในการทำงาน ส่วนทั้งหมดนี้จะโดยจะครอบด้วยสัญลักษณ์วงเล็บปีกกา
<b>Function Call</b>	เป็นการเรียกใช้ฟังก์ชัน โดยการเรียกใช้ฟังก์ชันจะต้องระบุชื่อของฟังก์ชัน และข้อมูลที่จะให้กับฟังก์ชันนั้น (Argument)
<b>Function Header</b>	เป็นส่วนหัวของฟังก์ชันซึ่งประกอบไปด้วย 3 ส่วนหลัก คือชื่อฟังก์ชัน (Function Name), พารามิเตอร์ (Parameter) และชนิดข้อมูลของค่าที่ส่งกลับ (Return Type)
<b>Parameter</b>	ส่วนที่มีการประกาศชนิดข้อมูล และชื่อของตัวแปรที่นำมารับค่า เพื่อส่งผ่านให้กับฟังก์ชัน อาจจะมี 1 ตัว หรือหลายตัว หรืออาจไม่มีเลยก็ได้ ขึ้นอยู่กับการใช้งานของฟังก์ชันนั้น ๆ ถ้าหากฟังก์ชันไม่มีพารามิเตอร์รับค่า อาจประกาศเป็น void หรือเว้นว่างไม่ประกาศก็ได้
<b>Pass by Reference</b>	การส่งค่าแบบอ้างอิง หรือการส่งค่าไปยังฟังก์ชันที่ถูกเรียกใช้โดยส่งเป็นนามแฝงของตัวแปรไป ซึ่งหากภายในฟังก์ชันมีการเปลี่ยนแปลงค่าพารามิเตอร์ ก็จะมีผลทำให้ค่าของอาร์กิวเมนต์ที่ต้นทางในโปรแกรมที่เรียกใช้เปลี่ยนแปลงไปด้วย
<b>Pass by Value</b>	การส่งค่าแบบกำหนดค่า หรือการคัดลอกค่าที่ผู้เรียกใช้ฟังก์ชันส่งให้ฟังก์ชัน โดยค่าที่ส่งมาจะถูกคัดลอกไปยังตัวแปรแบบ Local ในฟังก์ชัน โดยค่าต้นฉบับที่ผู้ใช้ส่งมาจะยังคงเดิม
<b>Prototype Declarations</b>	การประกาศฟังก์ชัน โดยส่วนมากจะประกาศอยู่ตรงส่วนบนสุดของโปรแกรม หรือก่อนส่วนการประกอบตัวแปรแบบ Global

## หมวดคำสั่งเงื่อนไข

คำศัพท์	ความหมาย
<b>And</b>	ใช้เชื่อมค่าความจริงของ 2 นิพจน์ หรือมากกว่า ผลลัพธ์จะเป็นจริงเมื่อนิพจน์ทั้ง 2 หรือทั้งหมดเป็นจริง และจะเป็นเท็จเมื่อมีอย่างน้อย 1 พจน์เป็นเท็จ ซึ่งใช้สัญลักษณ์ & หรือ &&
<b>else – if</b>	เป็นค่าสั่งสำหรับการสร้างเงื่อนไขแบบหลายทางเลือก โดยการเพิ่มเงื่อนไข else if เข้ามา นั่นหมายถึง เมื่อเช็คเงื่อนไขใน if และไม่ตรงเงื่อนไข จะมีการเช็คเงื่อนไขใน else if ถัดไป เรียงลำดับไปเรื่อย ๆ เมื่อตรงกับเงื่อนไขจะทำค่าสั่งในบล็อกนั้น และจบการทำงานแบบเงื่อนไขทันที (ไม่เช็คเงื่อนไขต่อ ๆ ไป ถ้าหากยังมี else if หรือ else ต่อ) สามารถใช้ร่วมกับค่าสั่ง else ซึ่งเป็นบล็อกของค่าสั่งที่จะทำงานเมื่อหากไม่มีเงื่อนไขใดใน if และ else if (ถ้ามี) เป็นจริง
<b>if</b>	เป็นค่าสั่ง 2 ทางเลือก ซึ่งถ้าเงื่อนไข (นิพจน์) เป็นจริงจะทำค่าสั่งที่กำหนดไว้ใน if และถ้าเงื่อนไข (นิพจน์) เป็นเท็จจะไม่ทำอะไร หลังจากนั้นทำค่าสั่งถัดกันมาหลังจาก if
<b>if... else</b>	เป็นค่าสั่ง 2 ทางเลือก ซึ่งถ้าเงื่อนไข (นิพจน์) เป็นจริงจะทำค่าสั่งที่กำหนดไว้ใน if และถ้าเงื่อนไข (นิพจน์) เป็นเท็จจะไปทำอีกค่าสั่งที่กำหนดไว้ใน else
<b>Logical Data</b>	ประเภทข้อมูลแบบตรรกะ
<b>Nested if</b>	เป็นค่าสั่ง if... else ที่มีค่าสั่ง if... else หรือค่าสั่ง if ข้อนกันอยุ่หลายชั้น
<b>Not</b>	ไม่สอด หรือค่าสั่งที่ใช้กลับค่าความจริง ซึ่งใช้สัญลักษณ์ !
<b>Or</b>	ใช้เชื่อมค่าความจริงของ 2 นิพจน์ หรือมากกว่า ผลลัพธ์จะเป็นจริงเมื่อมีอย่างน้อย 1 นิพจน์ เป็นจริง และเป็นเท็จเมื่อนิพจน์ทั้ง 2 หรือทั้งหมดเป็นเท็จ ซึ่งใช้สัญลักษณ์   หรือ
<b>switch</b>	เป็นค่าสั่งหลายทางเลือก ซึ่งเป็นค่าสั่งที่ใช้หลักการของ Nested if และค่าสั่งนี้จะมีตัวแปรสำหรับควบคุม (เปรียบได้กับเงื่อนไข) กับค่าสั่งถ้าหากตัวแปรนั้นตรงกับ case ที่กำหนดไว้

## หมวดคำสั่งวนลูป

คำศัพท์	ความหมาย
<b>break</b>	เป็นคำสั่งที่ให้โปรแกรมออกจากลูปทันที โดยไม่ท่าคำสั่งที่เหลือต่อ ซึ่งคำสั่ง break นี้สามารถใช้ได้กับลูปหลาย ๆ ลูปไม่ว่าจะเป็น while, do while, for และอื่น ๆ รวมถึงยังดองใช้ร่วมกับ switch ด้วย
<b>continue</b>	ถูกใช้เพื่อข้ามการทำงานในรอบปัจจุบัน ซึ่งจะไม่ท่าคำสั่งหลัง continue ที่ถูกเรียก และจะทำการเริ่มลูปเริ่มรอบถัดไปทันที
<b>do while</b>	เป็นลูปแบบ Post – Test Loop คือมีคำสั่งก่อนที่จะไปทำการตรวจสอบตัวควบคุมลูป (เงื่อนไข หรือนิพจน์)
<b>exit</b>	เป็นคำสั่งที่ใช้สำหรับการออกจากโปรแกรมย่อย (Sub หรือ Function) หรือออกจากกระบวนการช้า (Loop) จะใช้ร่วมกับเงื่อนไขที่ผู้ใช้ต้องการ
<b>for</b>	เป็นลูปแบบ Pretest Loop ที่ใช้นิพจน์ 3 นิพจน์สำหรับควบคุม โดยนิพจน์แรกเป็นนิพจน์เริ่มต้นส่วนมากใช้กำหนดค่าตัวควบคุม นิพจน์ที่ 2 เป็นเงื่อนไขในการตรวจสอบตัวควบคุมลูป (เงื่อนไข หรือนิพจน์) และนิพจน์ที่ 3 เป็นนิพจน์สำหรับเพิ่ม หรอลดค่าตัวควบคุมลูป
<b>Loop</b>	เป็นคำสั่งที่ใช้ควบคุมการทำงานของโปรแกรมแบบวนซ้ำ การทำงานเดินๆตามเงื่อนไขที่กำหนด เช่น การวนซ้ำการทำงานเดินเป็นจำนวน 10 รอบ ทำงานซ้ำ ๆ จนกว่าเงื่อนไขจะเป็นเท็จ
<b>Pretest</b>	ลูปประเภทหนึ่ง จะทำการตรวจสอบเงื่อนไขก่อนว่าเป็นจริง หรือเป็นเท็จ ถ้าเป็นจริงก็ให้เข้าไปท่าคำสั่ง หรือชุดคำสั่งต่อไป และเมื่อท่าคำสั่ง หรือชุดคำสั่งเสร็จ ก็จะกลับมาทำการตรวจสอบเงื่อนไขอีกรอบ และจะทำเช่นนี้ไปเรื่อย ๆ จนกว่าเงื่อนไขจะเป็นเท็จ ถึงจะจบการทำงานของลูป
<b>Post – Test</b>	ลูปประเภทหนึ่ง จะท่าคำสั่ง หรือชุดคำสั่งก่อน เมื่อเสร็จแล้วจึงจะมาตรวจสอบเงื่อนไขว่าเป็นจริง หรือเป็นเท็จ ถ้าเป็นจริงก็จะกลับไปท่าคำสั่ง หรือชุดคำสั่งเดินอีกรอบ และจะทำจนกว่าเงื่อนไขจะเป็นเท็จเช่นเดียวกัน
<b>while</b>	ลูป while นี้ใช้เงื่อนไขเป็นตัวควบคุมลูป (เงื่อนไข หรือนิพจน์) ซึ่งจะเป็นลูปแบบ Pretest Loop ซึ่งจะทำการตรวจสอบเงื่อนไขก่อนที่จะเข้าไปท่าคำสั่งในลูป

## ASCII TABLE

### ผังอักขระและสกุลที่ไม่สามารถแสดงผล

อักขระที่ปรากฏในตารางเป็นเพียงการแสดงว่า ณ ตำแหน่งนั้นมีรหัสตั้งกล่าวอยู่ ในใช้สัญลักษณ์ที่จะนำมาแสดงผลเป็นหลัก ซึ่งไขเป็นรหัสความคุณการพิมพ์บนเครื่องพิมพ์ หรือตัวแบ่งข้อมูลในลีบันทึกข้อมูลบางชนิด ( เช่น เทป )

BIN	DEC	HEX	CHAR	ความหมาย	BIN	DEC	HEX	CHAR	ความหมาย
0000 0000	0	00	(ว่าง)	NUL - null character	0001 0000	16	10	►	DLE - data link escape
0000 0001	1	01	☺	SOH - start of heading	0001 0001	17	11	◀	DC1 - device control one
0000 0010	2	02	☻	STX - start of text	0001 0010	18	12	↕	DC2 - device control two
0000 0011	3	03	♥	ETX - end of text	0001 0011	19	13	‼	DC3 - device control three
0000 0100	4	04	♦	EOT - end of transmission	0001 0100	20	14	¶	DC4 - device control four
0000 0101	5	05	♣	ENQ - enquiry	0001 0101	21	15	§	NAK - negative acknowledge
0000 0110	6	06	♠	ACK - acknowledge	0001 0110	22	16	▬	SYN - synchronous idle
0000 0111	7	07	•	BEL - bell	0001 0111	23	17	▬	ETB - end of trans. block
0000 1000	8	08	▣	BS - backspace	0001 1000	24	18	↑	CAN - cancel
0000 1001	9	09	○	HT - horizontal tabulation	0001 1001	25	19	↓	EM - end of medium
0000 1010	10	0A	▣	LF - line feed, new line	0001 1010	26	1A	→	SUB - substitute
0000 1011	11	0B	♂	VT - vertical tabulation	0001 1011	27	1B	←	ESC - escape
0000 1100	12	0C	♀	FF - form feed, new page	0001 1100	28	1C	∟	FS - file separator
0000 1101	13	0D	♪	CR - carriage return	0001 1101	29	1D	↔	GS - group separator
0000 1110	14	0E	♫	SO - shift out	0001 1110	30	1E	▲	RS - record separator
0000 1111	15	0F	☼	SI - shift in	0001 1111	31	1F	▼	US - unit separator
					0111 1111	127	7F	□	DEL - delete

### ผังอักขระและสกุลที่สามารถแสดงผล

BIN	DEC	HEX	CHAR	BIN	DEC	HEX	CHAR	BIN	DEC	HEX	CHAR
0010 0000	32	20	(ช่องว่าง)	0100 0000	64	40	@	0110 0000	96	60	'
0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0010 0100	36	24	\$	0100 0100	68	44	D	0110 0100	100	64	d
0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0010 0111	39	27	'	0100 0111	71	47	G	0110 0111	103	67	g
0010 1000	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
0010 1001	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
0010 1100	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
0010 1101	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
0010 1110	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
0010 1111	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
0011 0000	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
0011 0001	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q
0011 0010	50	32	2	0101 0010	82	52	R	0111 0010	114	72	r
0011 0011	51	33	3	0101 0011	83	53	S	0111 0011	115	73	s
0011 0100	52	34	4	0101 0100	84	54	T	0111 0100	116	74	t
0011 0101	53	35	5	0101 0101	85	55	U	0111 0101	117	75	u
0011 0110	54	36	6	0101 0110	86	56	V	0111 0110	118	76	v
0011 0111	55	37	7	0101 0111	87	57	W	0111 0111	119	77	w
0011 1000	56	38	8	0101 1000	88	58	X	0111 1000	120	78	x
0011 1001	57	39	9	0101 1001	89	59	Y	0111 1001	121	79	y
0011 1010	58	3A	:	0101 1010	90	5A	Z	0111 1010	122	7A	z
0011 1011	59	3B	;	0101 1011	91	5B	[	0111 1011	123	7B	{
0011 1100	60	3C	<	0101 1100	92	5C	\	0111 1100	124	7C	
0011 1101	61	3D	=	0101 1101	93	5D	]	0111 1101	125	7D	}
0011 1110	62	3E	>	0101 1110	94	5E	^	0111 1110	126	7E	~
0011 1111	63	3F	?	0101 1111	95	5F	_				



เป็นภาษาคอมพิวเตอร์ขั้นสูงที่พัฒนาขึ้น  
ในปี ค.ศ. 1972 โดย Denis Ritchie โดยที่  
ภาษา C มีการพัฒนามาจากภาษา B

# Flowchart

ผังงาน

เข้าใจได้ง่ายกว่า source code

“รู้ว่าสิ่งใดทำก่อน ทำหลัง”

หาข้อผิดพลาดได้ง่าย

คือ รูปแบบหรือลัญลักษณ์ที่ใช้แทนค่าอธิบายหรือคำพูด  
ที่ใช้ในอัลกอริทึม เพื่อให้คนอื่นสามารถเข้าใจได้ง่ายขึ้น

## ลัญลักษณ์ที่ใช้ในการเขียนผังงาน



ลัญลักษณ์แสดง  
จุดเริ่มต้นหรือจุดสิ้นสุด



ลัญลักษณ์ใน  
การสร้างเงื่อนไข



ใช้ในการประมวลผล



การนำเข้าหรือส่งออกข้อมูล  
โดยไม่ระบุสื่อ



ลัญลักษณ์จุดเชื่อมของ  
ผังงานในหน้าเดียวกัน



ลูกศรแสดง  
ทิศทางการทำงาน

# Pseudo Code

ภาษาที่ใช้จำลองภาษาโปรแกรม โดยจำลองคำสั่งจริงแบบย่อๆ ตามขั้นตอนที่ต้องการสร้าง  
ขึ้นแบ่งเป็น 3 ส่วน คือ

- 1. Draft code : เขียนคร่าวๆ ว่า “จะทำอะไร? ยังไง?”
- 2. Detailed code : นำ Draft code มาขยายความให้ละเอียด
- 3. Simple code : ใกล้เคียงกับภาษาโปรแกรมที่สุด พร้อมแปลไปใช้!

# โครงสร้างโปรแกรม

Console มีความกว้าง 80 ตัวอักษร 25 บรรทัด

หมายเหตุภาษา C++

หมายเหตุภาษา C

```
#include<file.h> _____ Preprocessor Directive •

type function_name (type); _____ Function Prototype Declaration •

type variable;  

เป็นตัวแปรที่สามารถใช้ที่ส่วนไหนของโปรแกรมก็ได

int main()
{
    เป็นตัวแปรที่สามารถใช้เฉพาะในฟังก์ชันที่ประกาศไว้
    type variable;
    statement; _____ ค่าสั่ง •
    return 0;
}

type function_name (type variable)  

{
    *การรับค่ามา (Parameter)
    type variable;
    statement;
    return (var);
}
```

Main Function

Function

**Header File** เป็นส่วนที่เรียกใช้ไลบรารี ซึ่งขึ้นต้นด้วย # เล่มโดยเป็นการนำโค้ดในไฟล์นั้นมาใช้

① ดึงค่าสั่งจากไฟล์เดอร์ที่เก็บ Source Code ก่อนแล้วค่อยหาจากที่เก็บไว้ในตัวคอมไฟล์เดอร์

**#include "headname.h"**

② ดึงค่าสั่งจากที่เก็บไว้ในตัวคอมไฟล์เดอร์ที่เดียวเท่านั้น

**#include <headname.h>**

หมายเหตุ

<b>#include&lt;stdio.h&gt;</b>	: ค่าสั่งทั่วไป
<b>#include&lt;conio.h&gt;</b>	: ค่าสั่งจัดการหน้าจอ
<b>#include&lt;math.h&gt;</b>	: ค่าสั่งเกี่ยวกับคณิตศาสตร์
<b>#include&lt;string.h&gt;</b>	: ค่าสั่งเกี่ยวกับข้อความ

# Constant



#define & Const :: กำหนดค่าคงที่ให้กับตัวแปร

#define identifier replacement

เบน #define PI 3.14

const type identifier = value;

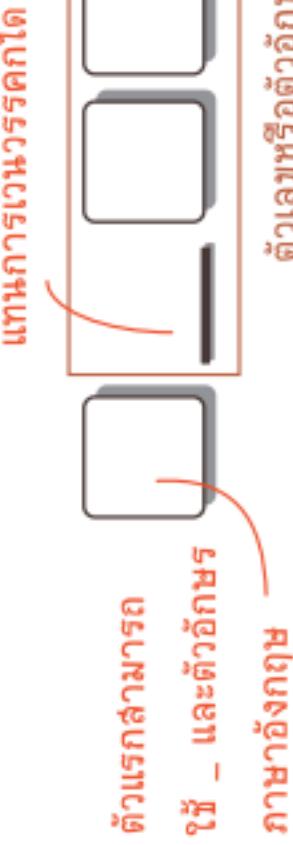
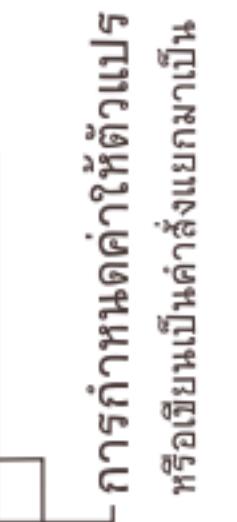
เบน const double PI = 3.1415926;

const char N = 'a';

หรือเขียนได้อีกรูปแบบ คือ

type identifier;

type identifier = value;



โดย ห้ามตั้งชื่อข้า กับคำสั่งว่า (Reserved Word)

auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	goto	if	int	long
register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void
volatile					

# Display Data

การแสดงผลทางหน้าจอ

## 1. ใช้คำสั่ง printf

การแสดงผลของข้อความ

```
printf("text");
```

การแสดงผลของค่าที่เก็บในตัวแปร

```
printf("format_1 format_2 ... format_n",var_1,var_2, ..., var_n);
```

more!

```
printf("%[m].[n]",variable);
```

การเว้นช่องใน Console

จำนวนที่แสดงออกมา

เช่น %3f คือ เลขทศนิยม 3 ตำแหน่ง

%4.5e คือ เว้นไป 4 ช่องแล้วเอาแค่ 5 ตัวอักษร

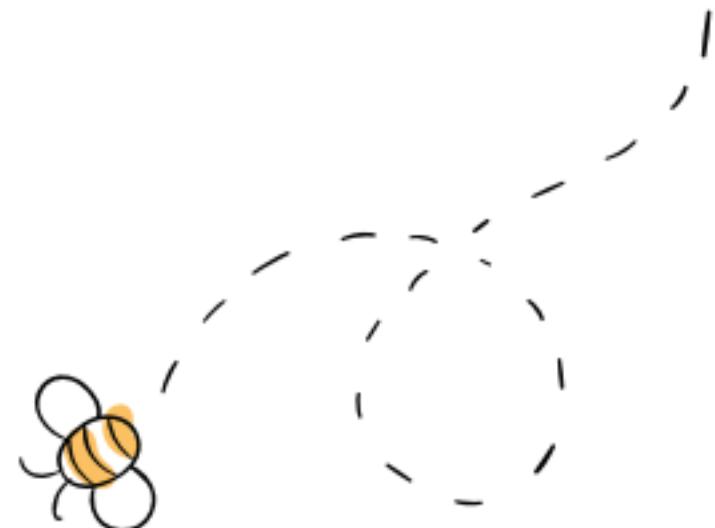
%2d คือ เว้นไป 2 ช่องของเลขจำนวนเต็ม

## 2. ใช้คำสั่ง putchar() สำหรับแสดงแบบอักขระตัวเดียว (Character)

```
putchar(var);
```

## 3. ใช้คำสั่ง puts() สำหรับแสดงข้อมูลแบบสายอักขระ (String)

```
puts(var);
```



# Input Data

การรับข้อมูล

## 1. การรับข้อมูลชนิดอักขระทีละตัวจากคีย์ด้วยคำสั่ง getch() และ getchar()

```
ch = getch()
```

```
ch = getchar()
```

“ch เป็นตัวแปรชนิดอักขระ (char) สำหรับรับข้อมูลจากคีย์บอร์ด

## 2. การรับข้อมูลชนิดข้อความจากคีย์บอร์ดด้วยคำสั่ง gets()

```
gets(str);
```

## 3. การรับข้อมูลทุกชนิดจากคีย์บอร์ดด้วยคำสั่ง scanf()

```
scanf("format_1 format_2 ... format_n",&var_1, &var_2, ..., &var_n);
```

ถ้ากรณีที่เป็นตัวแปรชนิดข้อความ ไม่ต้องใช้ &

### การกำหนดลำดับการรับข้อมูลของคำสั่ง scanf()

```
scanf("%*d %d",&n);
```

สามารถกำหนดลำดับการรับข้อมูลโดยใช้เครื่องหมาย \* ได้ หากลำดับของข้อมูลที่รับเข้ามาใช้เครื่องหมาย \* ข้อมูลที่รับเข้ามาในลำดับต่อไปจะไม่ถูกจัดเก็บลงตัวแปร

### การกำหนดจำนวนตัวอักขระในการรับค่าของข้อมูลชนิดข้อความของคำสั่ง scanf()

```
scanf("%4s",str);
```

สามารถกำหนดความยาวข้อความที่ต้องการรับสูงสุดได้ โดยการใส่ตัวเลขหน้า s ใน %s ในส่วนของ format

สามารถเปลี่ยนได้

### การนับจำนวนตัวอักขระที่รับค่าของข้อมูลด้วยคำสั่ง scanf()

```
scanf("%rs%n",str ,&count);
```

สามารถนับความยาวของข้อความที่รับเข้ามา เพื่อนำไปใช้กำหนดเงื่อนไขต่างๆ หรือทำอย่างอื่นได้

## Regular Expression



: หากเจออักขระนอกกลุ่มนี้ จะทำการหยุดการหยุดรับค่าทันที  
 เช่น [abc] คือ หา 'a', 'b', 'c'  
 [a-cx-z] คือ หา 'a', 'b', 'c', 'x', 'y', 'z'



: หากเจออักขระในกลุ่มนี้ จะทำการหยุดการหยุดรับค่าทันที

# การแปลงชนิดข้อมูล

# Data Type Conversion

## Explicit Type Conversion

แปลงชนิดข้อมูลก้าวหนึดตามที่ต้องการ

### (Type) ExpressionOrVariableName

```
float ans; ----- กำหนดค่าให้ตัวแปร  
ชื่อ num ชนิดข้อมูลเป็น ans  
int num = 20; ----- กำหนดค่าให้  
ตัวแปรชื่อ num ชนิดข้อมูลเป็น int  
ชื่อค่าเท่ากับ 20  
ans = (float)num; ----- เปลี่ยนข้อมูลจาก int เป็น float (ซึ่งมีค่าตือ 20.00)
```

แล้วนำค่ามาใส่ในตัวแปร ans

## Implicit Type Conversion

แปลงชนิดข้อมูลอัตโนมัติจากคำนวณโดยบัญชีโดยอัตโนมัติ

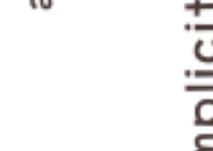
ชุดคำสั่ง

```
char > short > int > unsigned int > long int  
-----> float > double > long double
```

ถูก

# รูปแบบที่ใช้บ่อยๆ

ค่า flag	ความหมาย
-	ใช้กำหนดการและผลทางจรอภัยให้ดีข่าย ตามก้าวของล้านและจุดที่กำหนดไว้
+	ใช้กำหนดให้แสดงเดรีอห์มาย + หรือ - หน้าค่าว้อยลักษ์ต้องการแสดงผล
0	ใช้กำหนดให้แสดงลง 0 หากต้องการแสดงผล
#	ใช้กำหนดให้แสดงเลข 0, 0x และ 0X หน้าค่าว้อยลุลเครียร์และเครียร์หน้าสิบก้าว หรือให้กำหนดให้แสดงผลเป็นเล็กที่สิบก้าว ในกรณีที่ต้องการนั้นเมื่อเป็นจํานวนเต็ม หรือใช้กำหนดให้เป็นเล็กที่สิบก้าว เครียร์ตัวหนึ่งต่อท้าย ในการนี้ที่ต้องมีตัวอักษรที่ต้องการนั้นเป็นลูกขาก็ได้เช่นกัน



## รหัสรูปแบบ

## ชนิดข้อมูล

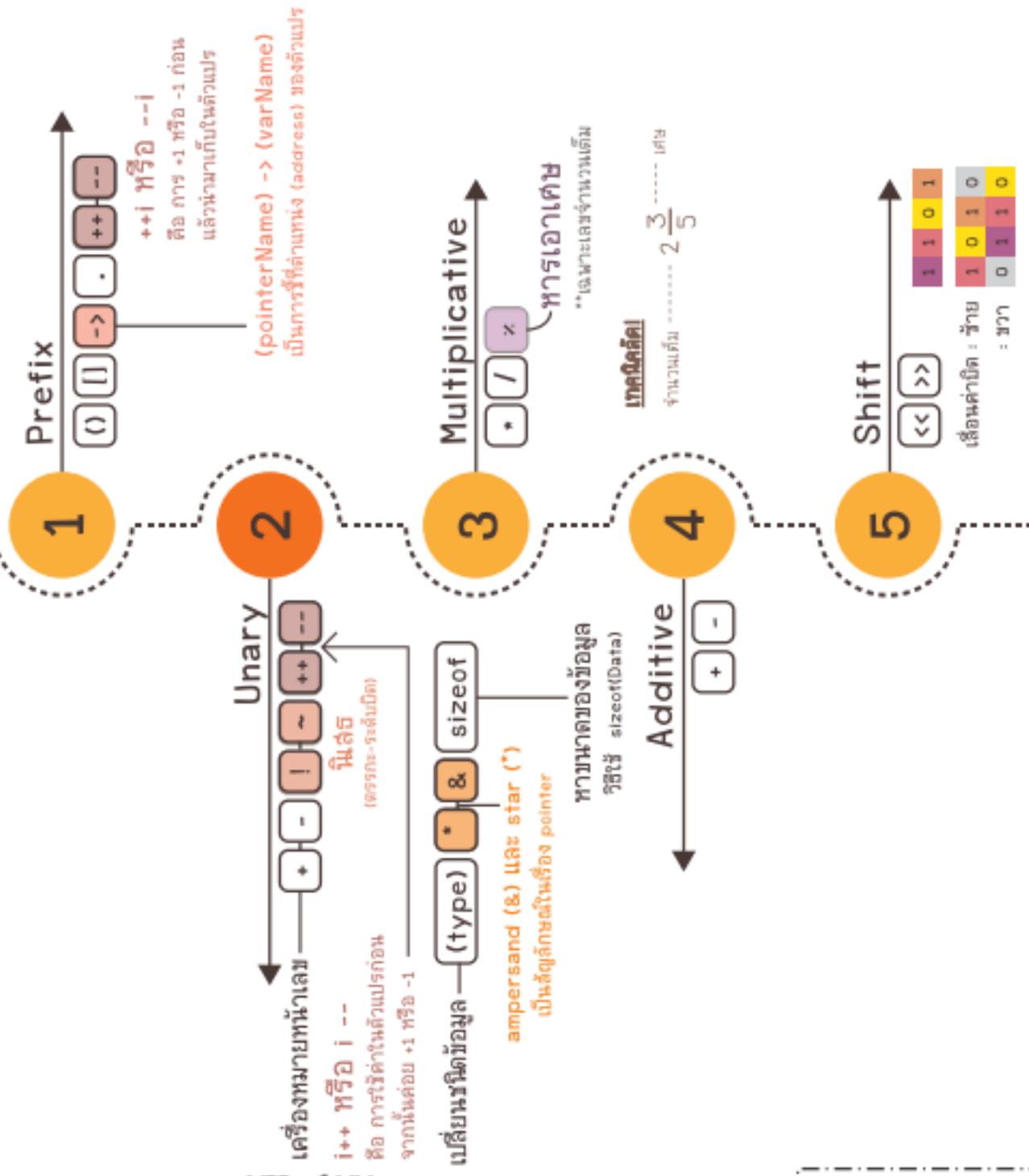
xc	ตัวอักษรระหว่างตัวอักษร
xd	จํานวนเต็มชนิด int
xf	จํานวนเต็มแบบเบิกโภภเนฑ์
xe	จํานวนจริงชนิด float
xf	จํานวนจริงชนิด float
gx	จํานวนจริงชนิด double
gf	จํานวนจริงชนิด double
gi	จํานวนเต็มชนิด int
go	จํานวนแบบตัวอักษร
gp	จํานวนเต็มบวก (Unsigned)
gx	จํานวนเต็มบวก
gxh	จํานวนเต็มบวก short
gxw	จํานวนเต็มฐานแบบชนิด long
gxL	จํานวนจริงชนิด long
gxu	จํานวนเต็มบวกชนิด short
gxv	จํานวนจริงชนิด long double
gxw	จํานวนเต็มชนิด unsigned long

\b	เลื่อนเคอร์เซอร์อยหลังไป 1 อักขระ
\n	รีบูร์ทต์ให้มี
\r	เลื่อนเคอร์เซอร์ไปช้าขึ้นอีกครั้ง Tab
\t	เว้นวรรค 8 ช่องหรือการ Tab
\o	Null
'	แสดงเครื่องหมาย ' (single quote)
"	แสดงเครื่องหมาย " (double quote)
\\	แสดงเครื่องหมาย \

# ตัวดำเนินการ Operator

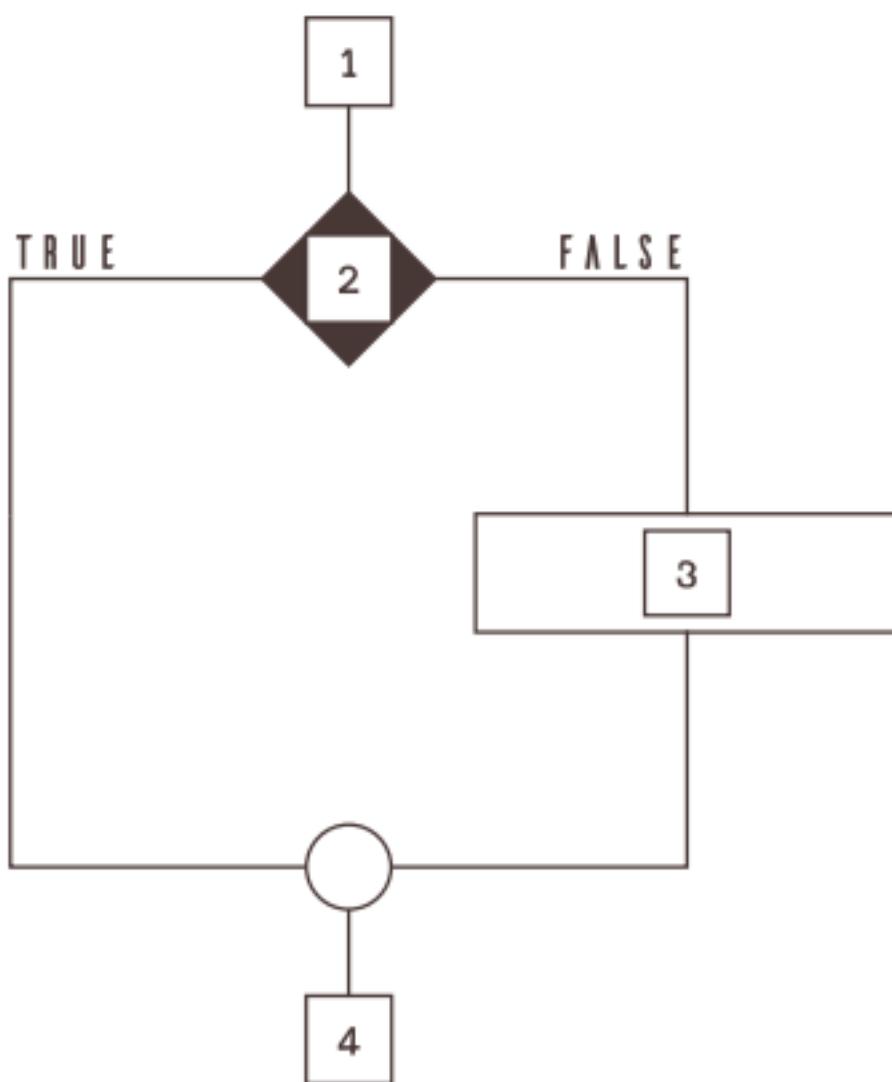
หมายเหตุ ● ตัวดำเนินการนี้จาก L  $\rightarrow$  R  
 ● ตัวดำเนินการนี้จาก R  $\rightarrow$  L

ลำดับความสำคัญของตัวดำเนินการ



# Decision Control Statement

การทำงานด้วยการตัดสินใจ



*if*

เป็นคำสั่งที่เราใช้กำหนดให้โปรแกรมตัดสินใจทำในสิ่งที่เป็นจริงหรือไม่ทำสิ่งใดเลย

```

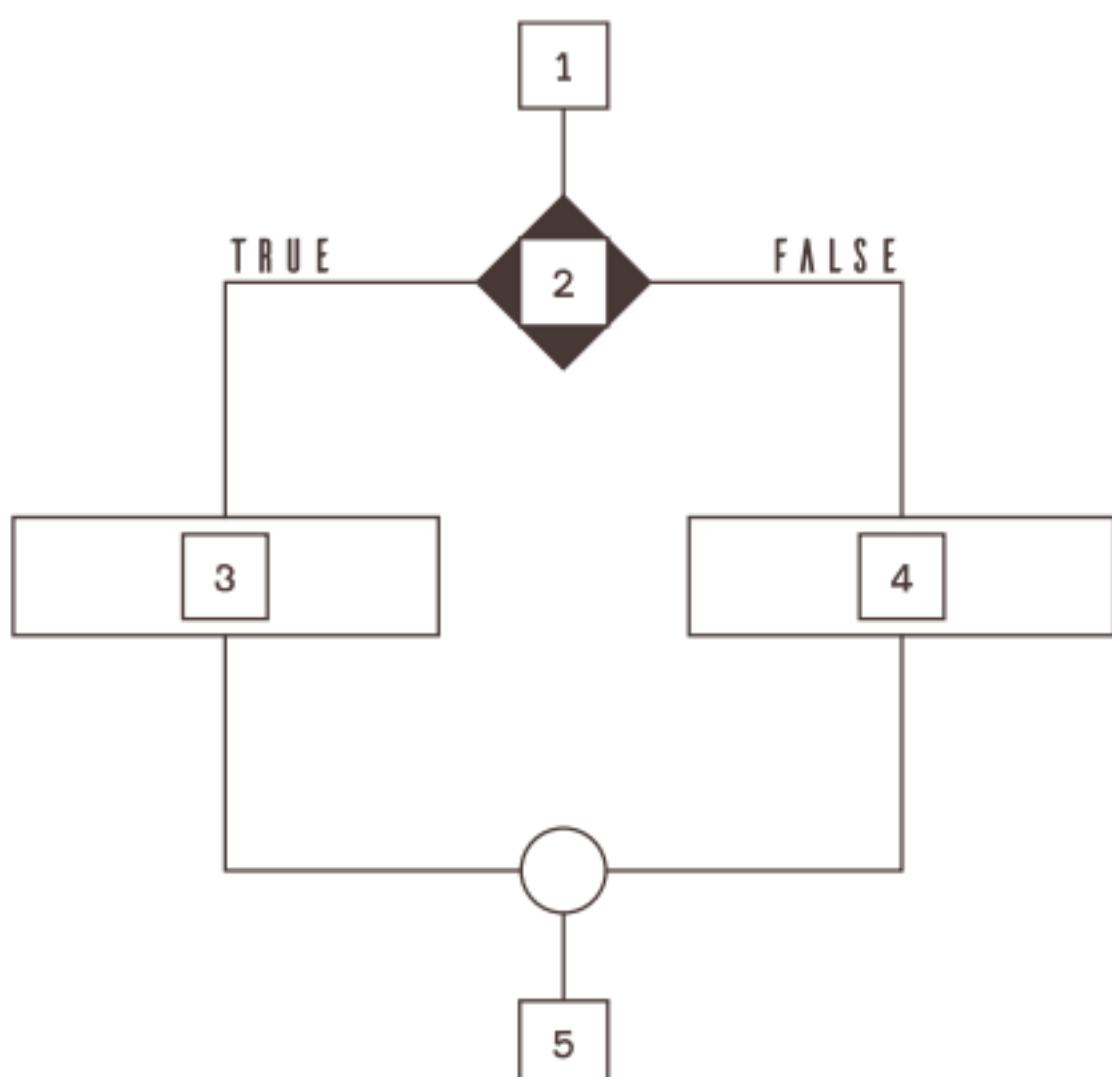
if (expression 2)
{
  statements 3;
}
  
```

*if - else*

เป็นคำสั่งที่เราใช้กำหนดให้โปรแกรมตัดสินใจทำในสิ่งที่เป็นจริงหรือทำในสิ่งที่เป็นเท็จ

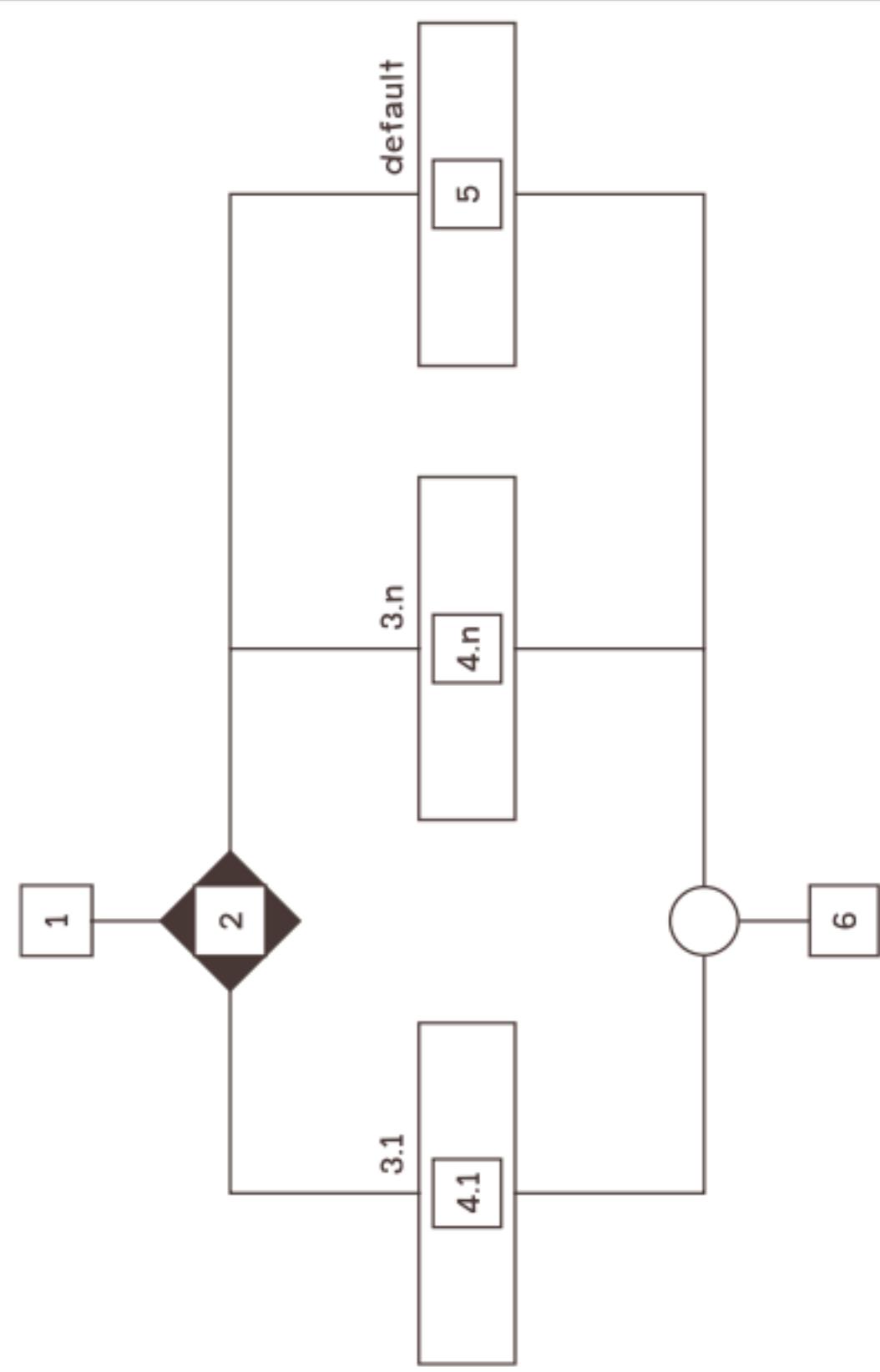
```

if (expression 2)
{
  statements 3;
  การทำงานผังเป็นจริง
}
else
{
  statements 4;
  การทำงานผังเป็นเท็จ
}
  
```



# Decision Control Statement

การทำงานด้วยการตัดสินใจ

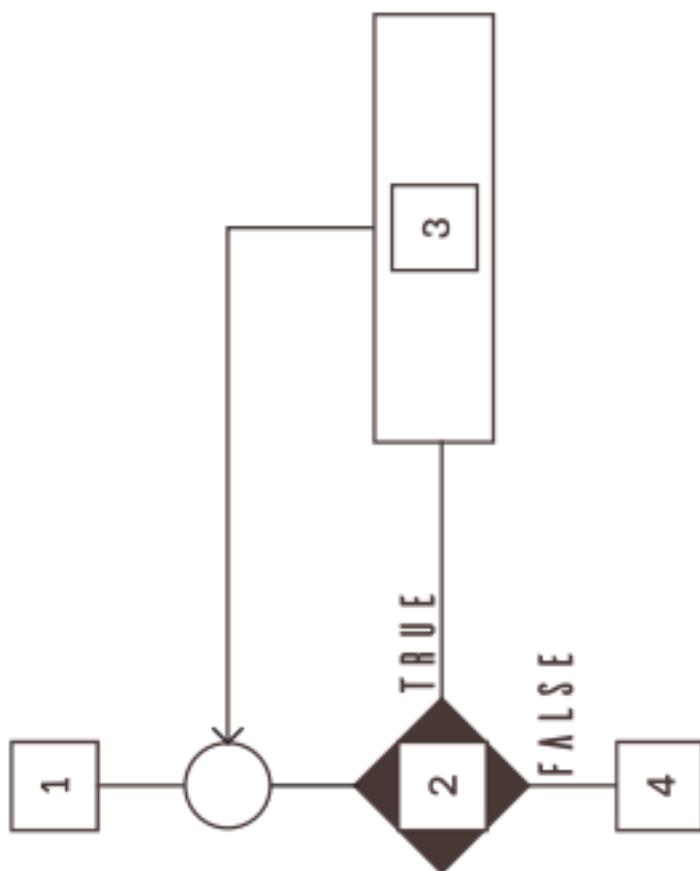


**switch - case**  
เป็นคำสั่งที่เราใช้กันหนาแน่นให้โปรแกรมตัดสินใจที่สามารถ  
เลือกได้มากกว่า 2 ทาง

```
switch (variable 2)
{
    case constant 3.1 : statement 4.1;
    break;
    case constant 3.n : statement 4.n;
    break;
    default : statement 5;
    break;
}
```

# Repetition Control Statement

การทำงานด้วยการวนลูป



*do - while*

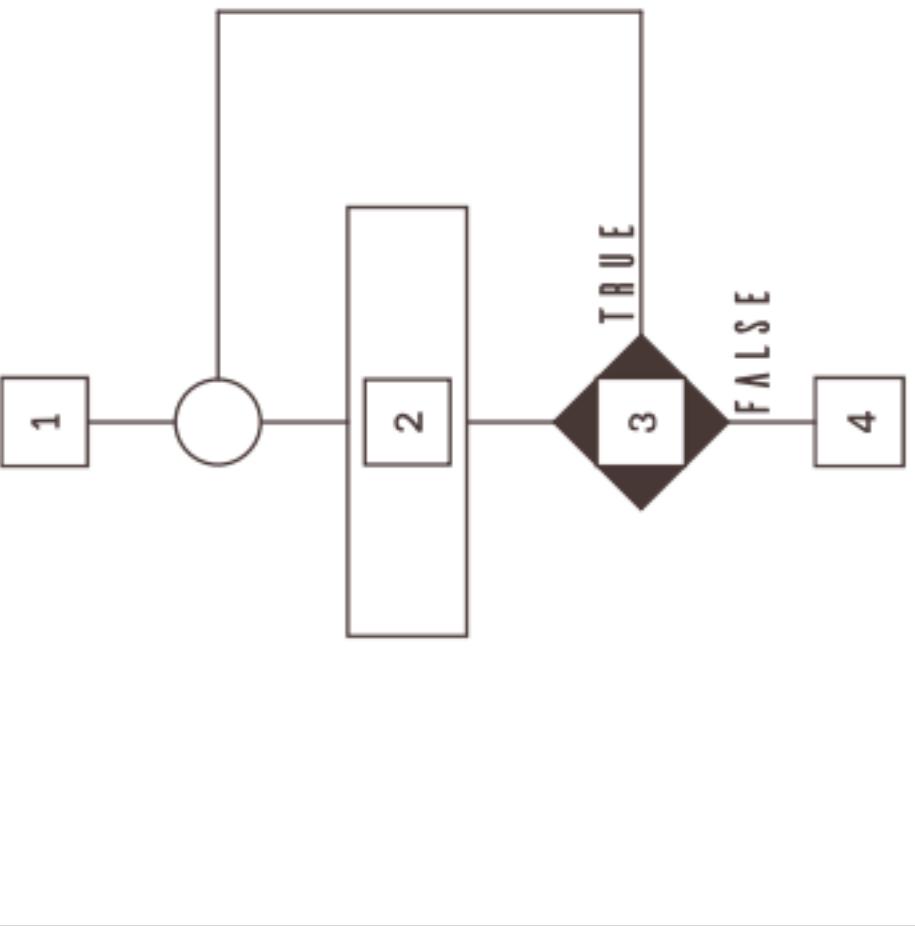
เป็นคำสั่งงานซ้ำโดยทำงาน 1 ครั้งแล้วค่อยตรวจสอบเงื่อนไข

```
do
{
    statements 2;
}
while (expression 3);
```

*while - loop*

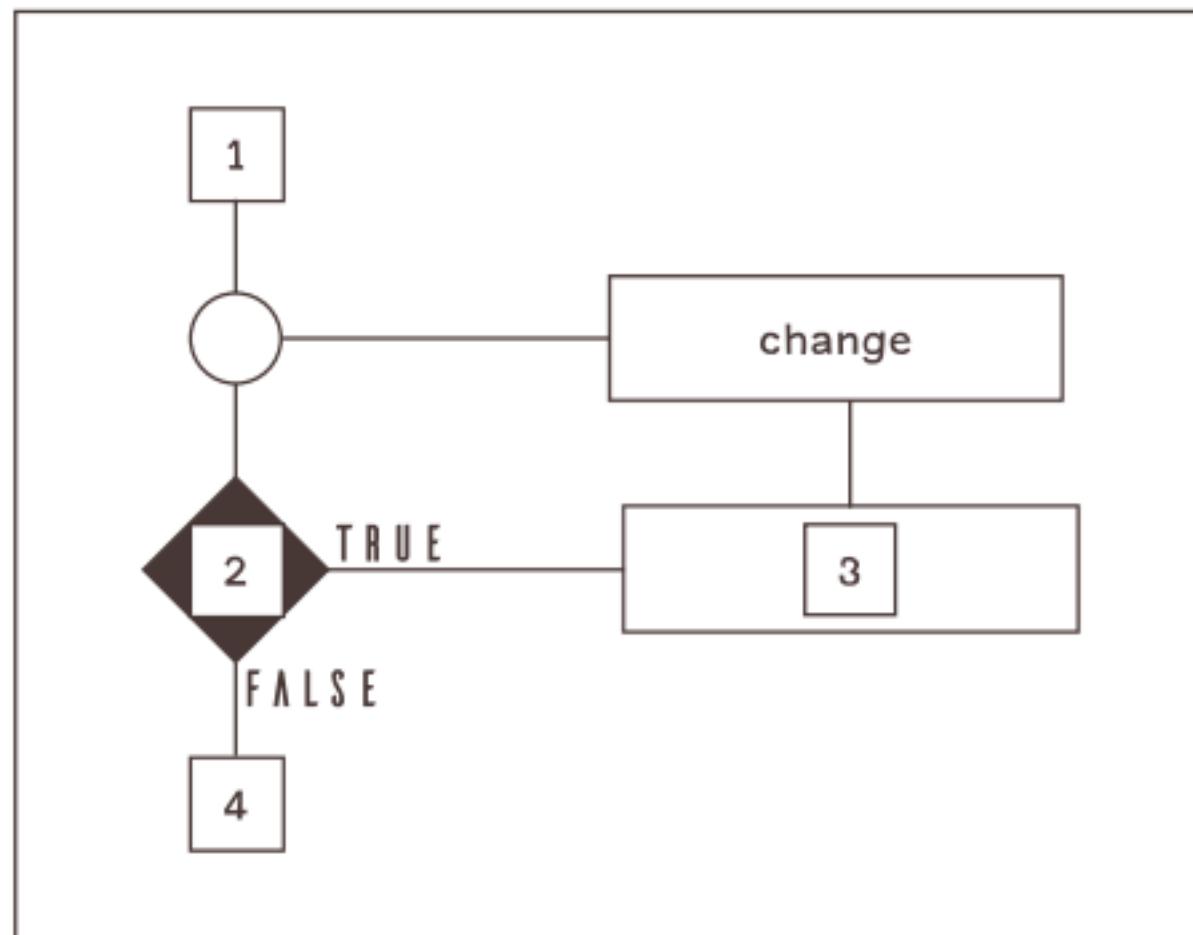
เป็นคำสั่งงานซ้ำโดยตรวจสอบเงื่อนไขก่อน ถ้าจริงจะทำงาน

```
while (expression 2)
{
    statements 3;
}
```



# Repetition Control Statement

การทำงานด้วยการวนลูป



*for loop*

เป็นคำสั่งวนซ้ำด้วยจำนวนรอบที่แน่นอน

```

for (initial; expression 2; change)
{
    เป็นค่าเริ่มต้นของตัวแปรที่จะเปลี่ยน
    ซึ่งจะประกาศตัวแปรตรงนี้หรือก่อนหน้าก็ได้

    statements 3;

}
  
```

หมาย!

คำสั่ง break : ออกจากลูปทันที

คำสั่ง continue : ข้ามการทำงานที่เหลือและเริ่มลูปใหม่ทันที



# โครงสร้างโปรแกรม

\*\*บางส่วนเท่านั้น

หมายเหตุเพิ่มเติม

```
int main()
{
    เป็นตัวแปรที่สามารถใช้เฉพาะในไฟล์ขั้นที่ประกาศไว้
    type variable;
    statement; ค่าสั่ง .
    return 0;
}
```

Main Function

```
type function_name (type variable)
{
    type variable;
    statement;
    return (var);
}
```

Function

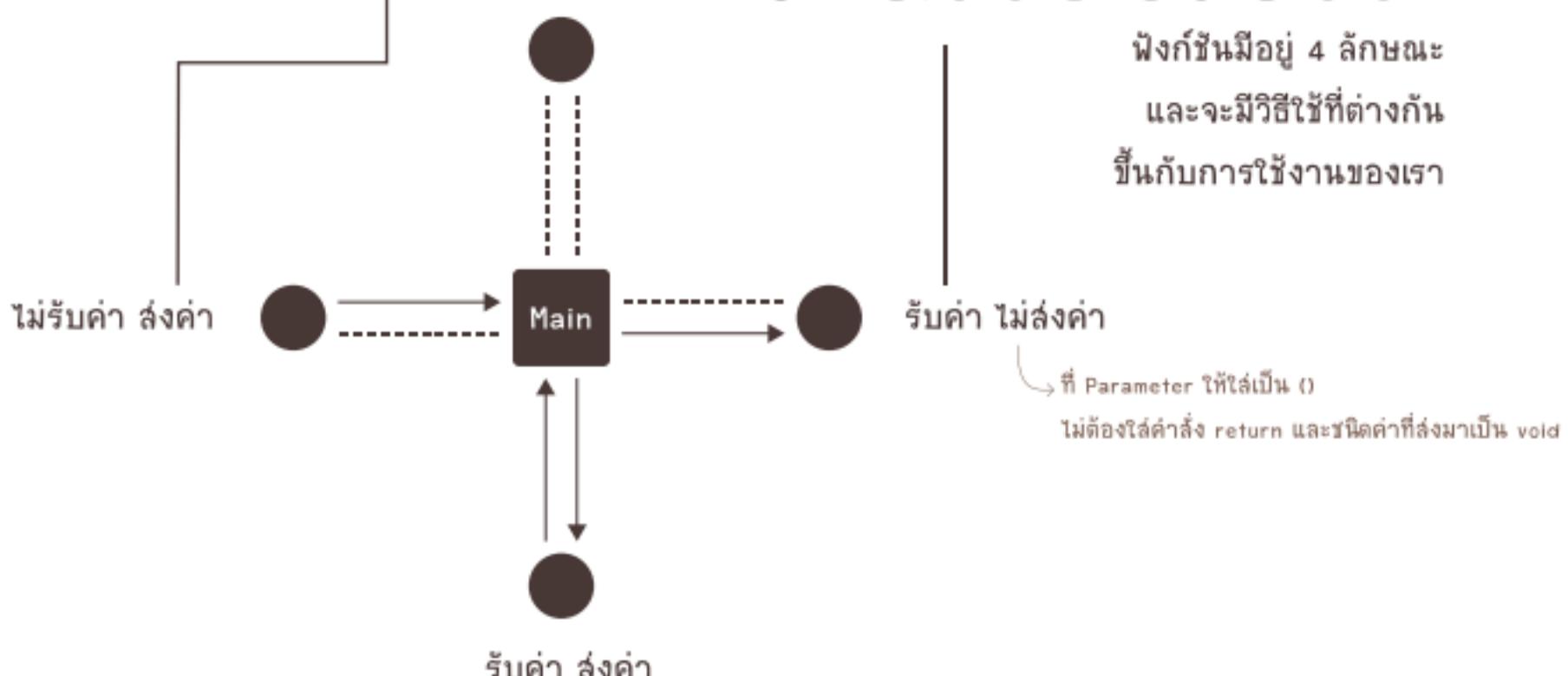
\*การรับค่ามา (Parameter)  
คือ การส่งค่าจากไฟล์ขั้นหลักที่เรียกใช้งาน  
มาสู่ในไฟล์ขั้นย่อย โดยที่จะส่งมาที่ค่าก็ได้  
โดยให้ครบตามจำนวนของ Parameter ตามที่  
กำหนดไว้ในไฟล์ขั้นย่อย

\*การส่งค่ากลับ (return)  
คือ การส่งค่าที่ผ่านการทำตามค่าสั่งให้  
ไฟล์ขั้นที่เราสร้างขึ้นแล้วถือว่าการนำค่าส่ง  
กลับมาที่ไฟล์ขั้นหลักเพื่อนำมาใช้ต่อ

ไม่รับค่า ไม่ส่งค่า

## Function ไฟล์ขั้น

ไฟล์ขั้นมีอยู่ 4 ลักษณะ  
และมีวิธีใช้ที่ต่างกัน  
ขึ้นกับการใช้งานของเรา



# Array

ตัวแปรชื่อเดียวกัน  
แต่สามารถเก็บได้หลายค่า

## ประกาศตัวแปรอาร์เรย์

```
type identifier[size];
```

- 1 มิติ : [ขนาด]
- 2 มิติ : [แถว][คอลัมน์]
- 3 มิติ : [บล็อก][แถว][คอลัมน์]



\*ถ้าเป็นข้อความจะมี 10 อักขระต่อท้ายเล่มอ  
ดังนั้น string ถือว่าเป็น array ประเภทหนึ่ง

## การอ้างถึงข้อมูลในตัวแปร

- ans = Avar[0] + 8;

เลือกตัวแปรพร้อมกับระบุ index ด้วย

- Avar[2] = (2 + ans) - 3;

## การกำหนดค่าข้อมูล

```
type identifier[size] = {var1, var 2, ..., var n};
```

# Structure

: ตัวแปรประเภทโครงสร้าง เป็นการจัดกลุ่มข้อมูลที่มีความเกี่ยวข้องไว้ด้วยกัน

### การประกาศ structure

```
struct STRUCT_NAME
{
    DataType var1;
    DataType var2;
    DataType var3;
};
```

### การประกาศ structure

```
typedef struct STRUCT_NAME{
    DataType var1;
    DataType var2;
}struct_var[index];
```

ส่วนใหญ่การประกาศ structure ไว้ในส่วนของ global เพราะมักจะถูกเรียกใช้งานจากหลาย scope



## การเข้าถึงตัวแปรภายใน struct

`struct_var.var1`

ใช้ dot notation

สามารถเปลี่ยนแปลงค่าและอ่านค่าได้  
เหมือนกับตัวแปรประเภทที่เข้าถึงได้ตามปกติ

`struct_var.var1[INDEX];`

เข้าถึงค่า ณ index ที่ต้องการ

`struct_var[INDEX].var1;`

เข้าถึงค่าภายในตัวแปรแต่ละตัวที่ index ใด ๆ

## การกำหนดค่าเริ่มต้นให้ตัวแปรของ structure

```
struct_var = {var_1, var_2, ..., var_n};
```

ข้อควรระวัง! ลำดับการกำหนดค่าของตัวแปร ไม่ เช่นนั้น การใช้งานตัวแปรย่อมผิดพลาดได้

## การประกาศตัวแปรของ structure

```
struct STRUCT_NAME{
    DataType var1;
    DataType var2;
    DataType var3;
} struct_var;
```

```
struct STRUCT_NAME{
    DataType var1;
    DataType var2;
    DataType var3;
};

struct STRUCT_NAME struct_var;
```

การใช้ array ภายในตัวแปรของ structure

สามารถได้เยี่ยนล้านๆ ได้ดังนี้

```
typedef struct STRUCT_NAME{
    DataType var1[index];
    DataType var2;
}struct_var;
```

```
typedef struct STRUCT_NAME{
    DataType var1;
    DataType var2;
};DefName;
DefName struct_var;
```

ข้อสังเกต! ใน structure ทุก ๆ ตัวจะใช้ชื่อตัวแปรภายในเหมือนกัน แต่ว่าค่าที่อยู่ภายในจะไม่เท่ากัน  
ขึ้นกับตัวแปรของ structure แต่ละตัว

# Pointer

เป็นรูปแบบตัวแปรประเภทหนึ่งที่ทำการ “ชี้” ไปยังตำแหน่งที่อยู่ (address) ของตัวแปรอื่นภายในหน่วยความจำ ซึ่งแสดงว่า pointer ไม่สามารถถูกกำหนดค่าที่จะใช้ได้โดยตรง แต่เป็นการอ้างอิงค่ากับตัวแปรอื่นที่มีอยู่แล้ว



## การประกาศตัวแปรประเภท pointer

- กำหนดค่าเริ่มต้นพร้อมการประกาศตัวแปร pointer

```
DataType *PointerName = &VarName;
```

หรือ 

```
DataType *PointerName = ArrayName;
```

 ถ้าเป็นแต่ละ index ต้องเขียนแบบนี้  
`&ArrayName[INDEX];`

- กำหนดค่าเริ่มต้นภายหลังการประกาศตัวแปร pointer

```
PointerName = &VarName;
```

หรือ

```
PointerName = ArrayName;
```

หมายเหตุ จะไม่มี “&” หน้าตัวแปร Array เพราะมีลักษณะเป็น pointer อยู่แล้ว  
 ซึ่งจะกำหนดให้ pointer ชี้เริ่มต้นที่ index 0 ก่อน

## การเข้าถึงค่าที่ตัวแปร pointer

```
*PointerName;
```

หรือ

การเข้าถึงค่าตัวแปรในแต่ละ index

หากไม่ใส่ \* ไว้ข้างหน้า  
 จะได้เป็นค่า address ที่  
 pointer อ้างอิงไป

```
PointerName[INDEX];
```

```
*(PointerName + INDEX);
```

ฟิ ( ) เพื่อเป็นการเปลี่ยนตำแหน่ง  
 การเขียนของ pointer ไม่ใช้การเพิ่ม  
 ค่าที่ pointer เม้าถึงอยู่

