

CE BOOK

CE Boost Up

สารบัญ

วิจัยและการ	E
ผังงาน (Flowchart)	1
การเขียนผังงานที่ดี	1
ประโยชน์ของผังงาน	1
ข้อจำกัดของผังงาน	1
สัญลักษณ์ที่ใช้ในการเขียนผังงาน	1
รูปแบบของผังงาน	2
รหัสเทียม (Pseudo Code)	3
ประเภทรหัสเทียม (Pseudo Code Type)	3
Draft Code	3
Detailed Code	3
Simple Code	3
ตัวอย่าง	4
ประวัติความเป็นมา	5
จุดเด่นของภาษา C	5
ข้อแตกต่างระหว่าง ภาษา C vs. C++	5
โครงสร้างโปรแกรม	6
ເຍືດເດວຣີໄຟລ໌ (Header File)	6
ຟັງກົນ (Function)	6
ສ່ວນຕົວໂປຣແກຣມ (Body)	6
ตັບແປຣ (Variable)	7
ການຕັ້ງຊື່ໄອເດີນຕິໄຟເວຼຣ (Identifier Names)	7
ການປະກາສຕັບແປຣ (Variable Declaration)	7
ໜົດຂອງຂໍ້ມູນ (Data Type)	8
ໜົດຂໍ້ມູນແບບຈຳນວນເຕັມ (Integer Type)	8
ໜົດຂໍ້ມູນແບບຈຳນວນຈິງ (Real Floating Point Type)	9
ໜົດຂໍ້ມູນແບບຕົວອັກຂະ (Character Type)	9
ໜົດຂໍ້ມູນແບບສາຍອັກຂະ (Group of Character Type)	9
Null Character ແລະ Empty String	10
ປະເທດຕັບແປຣ (Variable Type)	10
ການກຳນົດຄ່າໃຫ້ຕັບແປຣ (Variable Assignment)	11
ຄ່າคงຖື (Constant)	11
ຮະບຸຄ່າໂດຍຕຽງ (Literal Constants)	11
ນິຍາມໂດຍ #define (Defined Constants)	12
ເກັບໄວໃນຕັບແປຣ (Memory Constants)	12
ການແປ່ງນົດຂອງຂໍ້ມູນ (Data Type Conversion)	12
Implicit Type Conversion	12
Explicit Type Conversion (Casting)	12
ການແສດງຜລທາງໜ້າຈອ (Display Data)	13
1. ໃຊ້ຄໍາສັ່ງ printf	13
ການແສດງຜລຂອງຂໍ້ມູນ	13
ການແສດງຜລຂອງຄ່າທີ່ເກັບໃນຕັບແປຣ	13
ຮັບສຽງແບບການແສດງຜລຂໍ້ມູນ ແລະການຈັດຮູບແບບແສດງຜລຂໍ້ມູນຂອງຄໍາສັ່ງ printf()	15

การจัดพื้นที่ และการกำหนดการแสดงผลข้อมูลของคำสั่ง printf()	15
2. ใช้คำสั่ง putchar() สำหรับแสดงแบบอักขระตัวเดียว (Character)	17
3. ใช้คำสั่ง puts() สำหรับแสดงข้อมูลแบบสายอักขระ (String)	17
ตัวดำเนินการ (Operator).....	17
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
ลำดับความสำคัญของตัวดำเนินการ (Precedence)	22
การรับข้อมูล (Input Data)	22
1. การรับข้อมูลชนิดอักขระทีละตัวจากคีย์บอร์ดด้วยคำสั่ง getch() และ getchar	22
2. การรับข้อมูลชนิดข้อความจากคีย์บอร์ดด้วยคำสั่ง gets()	23
3. การรับข้อมูลทุกชนิดจากคีย์บอร์ดด้วยคำสั่ง scanf().....	23
การกำหนดลำดับการรับข้อมูลของคำสั่ง scanf().....	24
การกำหนดจำนวนตัวอักขระในการรับค่าของข้อมูลชนิดข้อความของคำสั่ง scanf()	25
การนับจำนวนตัวอักขระที่รับค่าของข้อมูลด้วยคำสั่ง scanf()	25
Regular Expression	25
ควบคุมทิศทางการทำงานโปรแกรมด้วยคำสั่งตัดสินใจ (Decision Control Statement).....	26
คำสั่งตัดสินใจแบบเลือกทำ หรือไม่ทำด้วยคำสั่ง if	26
คำสั่งตัดสินใจแบบสองทางเลือกด้วยคำสั่ง if...else	26
คำสั่งตัดสินใจแบบหลายทางเลือกด้วยคำสั่ง if...else if	26
คำสั่งตัดสินใจแบบหลายทางเลือกด้วยคำสั่ง switch-case	27
ควบคุมการทำงานของโปรแกรมด้วยคำสั่งวนลูป (Repetition Control Statement).....	28
วนลูปการทำงานด้วยจำนวนรอบที่แน่นอนด้วยคำสั่ง for	28
วนลูปการทำงานเมื่อเงื่อนไขเป็นจริงตัวยคำสั่ง while	29
วนลูปการทำงานอย่างน้อย 1 ครั้งด้วยคำสั่ง do...while	29
คำสั่ง break และ continue	29
อธิบายลูป for อย่างละเอียด	30
อาร์เรย์ (Array).....	31
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
ฟังก์ชัน (Function).....	34
ฟังก์ชันที่มาตราฐานที่ติดมากับภาษา C (Standard Library Function).....	34

math.h เกี่ยวกับการคำนวนทางคณิตศาสตร์	34
string.h เกี่ยวกับข้อความ	35
ctype.h เกี่ยวกับตัวอักษรระดับเดียว	35
stdlib.h เกี่ยวกับการแปลงค่า string	35
dos.h เกี่ยวกับการติดต่อระบบปฏิบัติการ	35
conio.h เกี่ยวกับการแสดงผลทางจอภาพ	36
ฟังก์ชันที่สร้างขึ้นเอง (User-Defined Functions)	36
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
พอยเตอร์ (Pointer)	40
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
การใช้งานพอยน์เตอร์อ้างอิงอิงพอยน์เตอร์ที่กำลังอ้างอิงตัวแปรอื่น	46
พอยน์เตอร์กับฟังก์ชัน	46
การส่งค่าที่เก็บอยู่ในตัวแปรให้กับฟังก์ชัน (Pass by Value)	46
การส่งค่าที่อยู่ของตัวแปรให้กับฟังก์ชัน (Pass by Pointer)	46
การส่งการอ้างอิงตัวแปรให้กับฟังก์ชัน (Pass by Reference)	46
โครงสร้างข้อมูล (Structure)	47
การนิยามโครงสร้างเพื่อใช้งาน (Structure Define)	47
การประกาศตัวแปรประเภทโครงสร้าง (Structure Variable Declaration)	47
การประกาศตัวแปรประเภทโครงสร้างหลายตัวโดยใช้อาร์เรย์	48
การทำหนดค่าเริ่มต้นตอนประกาศตัวแปร การเข้าถึง และกำหนดค่าให้ตัวแปรประเภทโครงสร้าง	48
การใช้อาร์เรย์ภายในตัวแปรประเภทโครงสร้าง	48
โครงสร้างซ้อนโครงสร้าง (Nest Structure)	49
ภาคผนวก	I
คำศัพท์ทั่วไป	I
หมวดระบบคอมพิวเตอร์	I
หมวดโครงสร้างภาษา	I
หมวดการแสดงผล และรับข้อมูล	II
หมวดนิพจน์ทางคณิตศาสตร์	II
หมวดฟังก์ชันของภาษา C	III
หมวดคำสั่งเงื่อนไข	III
หมวดคำสั่งวนลูป	IV
ASCII TABLE	V
ผังอักษรแอสกี้ที่ไม่สามารถแสดงผล	V
ผังอักษรแอสกี้ที่สามารถแสดงผล	V

สรุปเนื้อหา By TK (Infographic for C Programming)	VI
ผังงาน (Flowchart) & โคดเทียม (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

วิรัฒนาการ

น้อง ๆ หลายคนคงอาจจะสงสัยหัวข้อนี้ในสารบัญ วิรัฒนาการของภาษา C หรือพี่ เปป่า! มันคือ วิรัฒนาการของหนังสือเล่มน้อย ๆ เล่มนี้ยังไงหละ พากพีตั้งใจจะทำให้น้อง ๆ เอ้าไปศึกษา เตรียมความพร้อมก่อนเข้าเรียนในปี 1 ครับ :)

CE57

เริ่มทำหนังสือน้อยย่างจริงจัง เพราะในค่าย CEBU#6 ที่รุ่นพากพีเข้ากันอะ หนังสือที่ปรินต์มาแจก ไม่สิเรียกว่าชีท ดีกว่า พี่ ๆ เข้ายังไม่ได้ทำเป็นหนังสืออย่างจริงจังอะแหลก หลาย ๆ คนเลยได้มาลงกี๊เอามากอง ๆ สุดท้ายพากพีก็ต้องไปหา Google เอาอยู่ดี พี่เลยตัดสินใจสร้างสิ่ง ๆ นี้ขึ้นมาครับ! ซึ่ง... มันก็ยังไม่เป็นหนังสือหรอกครับ มีแค่เนื้อหา (พอยเตอร์ การเขียนไฟล์ก็ไม่มี) หน้าปก สารบัญ ภาคผนวกก็ยังไม่มี แคมป์ผิดยังเบอะอึก สารภาพด้วยครับมีเวลาทำแค่ 1 สัปดาห์ 5555+ ขอภัยมาที่นี้ยะ _/_ อยากรู้ความเป็นมาละเอียดกว่านี้หรือ... ถ้ามีสิ แต่ไม่บอกหรอกนะ อิอิ และพี่ขอส่งท้ายก่อนจากลากันไปด้วยหนังสือส่วนสุดท้ายของเล่มกันครับ

Niranam...

พอยเตอร์ (Pointer)

พอก่อนน้อง.... แคนนีก็จะอ้วกแตกตามแหลก 5555 ยังเหลือเรื่องอ่าน-เขียนไฟล์นะถ้าหากศึกษาเพิ่มเติม ถ้าหนังสือเล่มนี้พิมพ์ดี หรือข้อมูลพิດพลาดตรงส่วนไหนพี่ ๆ ก็ขออภัยมา ณ ที่นี้ยะ

มีตรงไหนสงสัยถามพี่ฯได้เนอะ อยากรู้โจทย์ก็หกมากขอได้เหมือนกันมี source ให้ไปฝึก ขอให้โชค A สนุกกับการอ่านหนังสือเล่มนี้ครับ :D

ACADEMIC TEAM - CE BOOST UP #7

~ 39 of 39 ~

bit.ly/bookceboostup

CE58

ในส่วนของปีนี้ ถ้าอยากรู้เรื่องราวให้กระจุ่ง.... พากพี ๆ ก็พร้อมที่จะแกล้งใจ.. พี่ ๆ ได้سانต่อความตั้งใจของพี่ฯ CE57 โดยนำเนื้อหาจากค่าย CE BOOST UP #7 มาเพิ่มเติมส่วนพอยเตอร์ การเขียนโปรแกรมเชิงโครงสร้าง เพิ่มเติมส่วนเนื้อสรุปของหนังสือ รวมถึงหน้าปกอันละเอียด น่ารักเหมือนหน้าตาคนทำและคนอ่านด้วย โอดหัว ๆ และคำศัพท์ที่ควรรู้ (ที่คาดว่าจะมีประโยชน์กับน้องหลาย ๆ ตน เอยกัน มั้งนะ5555) สำหรับน้อง ๆ ที่เพิ่งฝึกหัดเขียนโปรแกรม เข้ามาอ่านแล้วอย่าเพิ่งห้อใจไปนะ ความพยายามไม่เคยทำร้ายสักคนที่ตั้งใจหอ ความมานะอุสาหะ และกาลเวลาจะหลอมเราให้กล้ายเป็นคนที่เก่งเอง ขอแค่เราเก่งกว่าตัวเราในเวอร์ชันเมื่อวานก็พอแล้ว (น้อววว...เขียนเหมือนอ่านนิยายไปอึก) สำหรับน้อง ๆ ที่มีพื้นฐานมาบ้างแล้วน้าา ก็เหมือนเป็นการทบทวนไปในตัว ฝึกฝนตัวเองต่อไปเรื่อยๆ อย่าง ๆ ๆ เก้าเป็นกำลังใจให้น้าา (ไม่เน้นยาวยา สารเดียว เน้นน่ารักมุ้ย ๆ ๆ)

หากมีเนื้อหาตรงไหนตกหล่นหรือเจอบุคคลผิดพลาดตรงไหน หรือคิดว่าอย่างจะเพิ่มเติมส่วนเนื้อหานั้นสือตรงไหนสามารถติดต่อห้องไม่มีค์พี่ ๆ ทุกคนได้เลยนะ ถ้าเลือกรหัสการนอน กับการเรียน จงเลือกการนอน... ไม่ใช่ชั่ง แต่ขอลาไปก่อนขอบคุณครับ

Mayuyu

CE59

ครั้งนี้ก็ครั้งที่ 9 แล้ว ที่มีค่าย CE BOOST UP ค่ายนี้สืบทอดกันนานนานมีวากกาก รวมทั้งหนังสือเล่มนี้ด้วยก็เช่นกัน ซึ่งปีนี้พี่แทบไม่ได้แกะอะไรหนังสือเลย เพราะพี่ปีที่แล้วเค้าทำไว้ดีมากแล้ว555555 พี่เองก็เคยเป็นน้องที่ใช้หนังสือเล่มนี้ จนตอนนี้มาเป็นพี่เองแล้ว ก็อยากให้น้อง ๆ ได้รับในสิ่งที่พี่ก่อน ๆ ถ่ายทอดมาให้พี่ พี่หวังว่าหนังสือเล่มนี้ จะช่วยให้น้อง ๆ ได้เข้าใจภาษาซีที่เป็นภาษาพื้นฐานของเรามากขึ้น สามารถปรับใช้กับการใช้งานจริงได้ โชคดีกับการเรียนครับน้อง ๆ

สุดท่าน และมองเพื่อน

ผังงาน (Flowchart)

ผังงาน (Flowchart) คือ รูปแบบ หรือสัญลักษณ์ที่ใช้เขียนแทนคำอธิบาย ข้อความ หรือคำพูดที่ใช้ในอัลกอริทึม เนื่องจาก การที่จะเข้าใจขั้นตอนได้ง่าย และตรงกันนั้นถ้าหากใช้คอมพิวเตอร์ หรือข้อความจะทำได้ยากกว่าการใช้รูปภาพ หรือ สัญลักษณ์ที่เป็นสากล และคนทั่วโลกสามารถเข้าใจได้ง่าย

การเขียนผังงานที่ดี

1. ใช้สัญลักษณ์ตามที่กำหนดไว้
2. ใช้ลูกศรแสดงทิศทางการทำงานของโปรแกรมจากบนลงล่าง หรือจากซ้ายไปขวา
3. คำอธิบายในภาพควรสั้นกระชัดรัด และเข้าใจง่าย
4. ทุกแผนภาพต้องมีลูกศรแสดงทิศทางเข้า – ออก
5. ไม่ควรโยงเส้นเชื่อมผังงานที่อยู่ไกลมาก ๆ ควรใช้จุดเชื่อมต่อแทน
6. ผังงานควรมีการทดสอบความถูกต้องของการทำงานก่อนนำไปเขียนโปรแกรม

ประโยชน์ของผังงาน

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

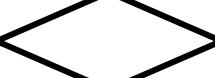
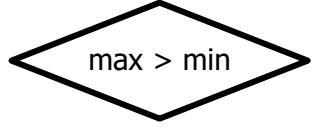
ข้อจำกัดของผังงาน

คนที่เขียนโปรแกรมบางคนไม่นิยมเขียนผังงานก่อนเขียนโปรแกรม เพราะเห็นว่าเสียเวลา นอกจานนั้นยังมี ข้อจำกัดอีก คือ

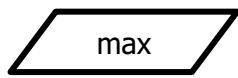
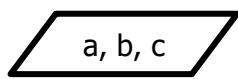
1. ผังงานเป็นการสื่อความหมายระหว่างบุคคลกับบุคคล มากกว่าที่จะสื่อความหมายระหว่างบุคคลกับเครื่อง เพาะผังงานไม่เข้ากับภาษาคอมพิวเตอร์ภาษาใดภาษาหนึ่ง ทำให้เครื่องไม่สามารถรับ และเข้าใจได้ว่าในผังงานนั้นต้องการ ให้ทำอะไร
2. ในบางครั้งเมื่อพิจารณาจากผังงาน จะไม่สามารถทราบได้ว่า ขั้นตอนการทำงานใดสำคัญกว่ากัน เพราะทุก ๆ ขั้นตอน จะใช้รูปภาพ หรือสัญลักษณ์ในลักษณะเดียวกัน
3. การเขียนผังงานเป็นการสื้นเปลือง เพราะจะต้องใช้กระดาษ และอุปกรณ์อื่น ๆ เพื่อประกอบการเขียนภาพ

สัญลักษณ์ที่ใช้ในการเขียนผังงาน

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

สัญลักษณ์แสดงขั้นตอนการทำงาน		
สัญลักษณ์	ความหมาย	ตัวอย่าง
 Start/Stop	การกำหนดจุดเริ่มต้นของการทำงาน และแสดงจุดสิ้นสุดของการทำงาน	 
 Process	การแสดงรายละเอียดของการทำงาน และกระบวนการทำงาน	<div style="border: 1px solid black; padding: 5px; width: fit-content;"> Total = 0 Total = Total + (a * c) </div>
 Decision	การแสดงรายละเอียดการเปลี่ยนเที่ยวนี้ไปต่าง ๆ ในขั้นตอนที่มีการตัดสินใจว่าใช้ หรือไม่ใช้	
 Flowline	การแสดงทิศทางความสัมพันธ์ของการทำงานในระบบงาน หรือ ลำดับงาน	
 IN-Page Connector	การกำหนดจุดอ้างอิงในการเชื่อมต่อ ในหน้ากระดาษเดียวกัน ของการเขียนผังงานโครงสร้าง (Structured Flowchart)	

สัญลักษณ์แสดงการรับ-ส่งข้อมูล

สัญลักษณ์	ความหมาย	ตัวอย่าง
 Read or Write	Read (รับค่า) การรับค่าข้อมูล หรืออ่านข้อมูลเข้ามาโดยไม่ระบุอุปกรณ์รับข้อมูล (Input Device) โดยอาจรับค่าข้อมูลมาจากคีย์บอร์ด หรือจากแฟ้มข้อมูลก็ได้ Write (แสดงผล) การแสดงรายละเอียดข้อมูล หรือแสดงผลลัพธ์ของการประมวลผล โดยไม่ระบุอุปกรณ์การแสดงผล (Output Device) โดยอาจแสดงผลผ่านจอภาพ หรือเครื่องพิมพ์ (Printer) ก็ได้	  

รูปแบบของผังงาน

ผังงานมีรูปแบบที่จำกัดอยู่ 3 รูปแบบ คือ

1. รูปแบบที่มีการกำหนดเงื่อนไขหรือให้เลือก (Decision Structure) : รูปแบบที่มีการสร้างเงื่อนไข สำหรับเลือกทำงาน โดยจะทำงานตามที่กำหนดหากตรงตามเงื่อนไข และถ้าหากไม่ตรงเงื่อนไขจะทำการที่กำหนดตั้งไว้ก่อนย่างหนึ่ง

```

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

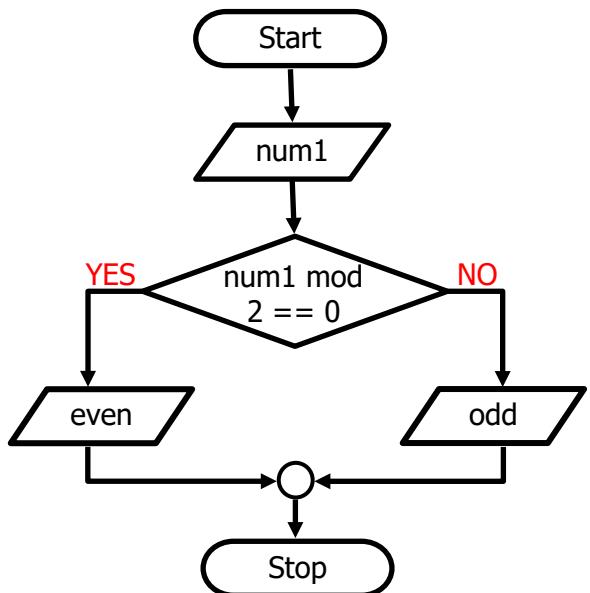
```

C:\Users\          \Desktop\Untitled1.exe - □ ×
5
odd
-----
Process exited after 0.7213
seconds with return value 0
Press any key to continue . . .
  
```

```

C:\Users\          \Desktop\Untitled1.exe - □ ×
4
even
-----
Process exited after 0.8413
seconds with return value 0
Press any key to continue . . .
  
```

ภาพตัวอย่างผลการรันโค้ด



2. รูปแบบเรียงลำดับ (Sequence Structure) : การทำงานแบบเรียงลำดับตั้งแต่ต้นจนจบ เป็นรูปแบบง่าย ๆ ไม่มีการเปรียบเทียบใด ๆ มีทิศทางเพียงทิศทางเดียว

```

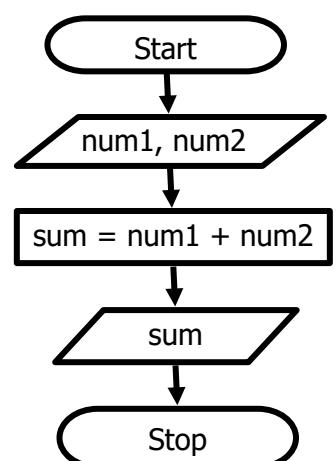
1 #include<stdio.h>
2
3 main()
4 {
5     int num1, num2, sum;
6     scanf("%d%d", &num1, &num2);
7     sum = num1 + num2;
8     printf("%d", sum);
9 }
  
```

ภาพตัวอย่างการเขียนโค้ด

```

C:\Users\          \Desktop\Untitled1.exe - □ ×
50
20
70
-----
Process exited after 2.776
seconds with return value 0
Press any key to continue . . .
  
```

ภาพตัวอย่างผลการรันโค้ด



3. รูปแบบที่มีการทำงานแบบวนรอบ หรือ loop (Iteration Structure) : รูปแบบที่มีการทำงานซ้ำๆ ค่าสั่งเดิม โดยการทำงานนี้จะขึ้นอยู่กับเงื่อนไขที่กำหนดให้

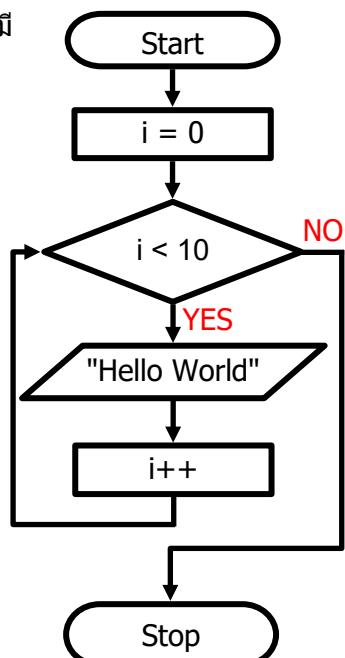
```
1 #include<stdio.h>
2
3 main()
4 {
5     int num;
6     for(int i = 0; i < 10; i++){
7         printf("Hello World\n");
8     }
9 }
```

```
C:\Users\ \Desktop\Untitled1.exe - □ ×
Hello World

-----
Process exited after 0.01836
seconds with return value 0
Press any key to continue . . .
```

ภาพตัวอย่างการเขียนโค้ด

ภาพตัวอย่างผลการรันโค้ด



ข้อสังเกต : เสน่ห์ทุกเส้นต้อง “เชื่อมต่อ กัน และมีทิศทางเสมอ” ดังนั้นทุกเส้นจะมีลูกศร และทุกเส้นจะต้องเชื่อมติดกันเป็นสัญลักษณ์

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

ข้อสังเกต : ในสัญลักษณ์การตัดสินใจ (Decision) “สามารถใส่ได้แค่เงื่อนไข่ใดเท่านั้น” และเล้นที่ลากออกต้องมีเพียงแค่ 2 เส้นเท่าตามเงื่อนไขซึ่งมี 2 รูปแบบ คือ “Yes/True หรือ No/False” เท่านั้นหากเป็นเงื่อนไข if-else

รหัสเทียม (Pseudo Code)

คือภาษาที่ใช้จำลองภาษาโปรแกรม โดยใช้ภาษาที่ไม่ใช้ภาษาโปรแกรมซึ่งไม่จำกัดภาษา และมีขั้นตอนที่แน่นอนจะทั้งหมด เป็นเพียงการจำลองคำสั่งจริงแบบย่อ ๆ ตามกระบวนการของโปรแกรมที่ต้องการสร้าง ซึ่งจะช่วยให้เห็นโครงสร้างโปรแกรมขั้นตอนนี้ สามารถหาจุดผิดของโปรแกรมได้ง่ายเมื่อมีปัญหา และเป็นตัวกำหนดงานเขียนโปรแกรมให้ทำงานได้ตามต้องการเมื่อมีการเขียนตามโค้ดเทียบกับที่สร้างไว้ หลักการเขียน Pseudo code ที่ดีควรให้ในหนึ่งบรรทัด ควรมีคำสั่งเดียว และย่อหน้าเพื่อแยกย่อคำสั่งให้ชัดเจน

ประเภทหัสเทียม (Pseudo Code Type)

Draft Code

เป็นการเขียนอย่างหยาบๆ ที่สุด ไม่ลงรายละเอียดอะไรมาก บอกว่ากำลังทำอะไรเป็นลำดับไป

Detailed Code

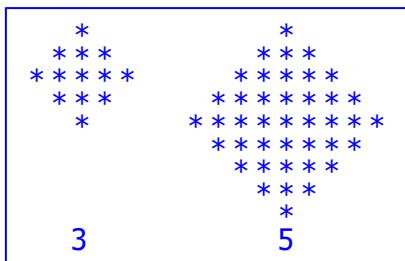
เป็นส่วนที่จะนำ draft code มาแตกรายละเอียดให้ชัดเจนว่ามีการทำงานตรงไหนอย่างไร โดย Detailed code จะทำกีรุบก็ได้จนกว่าจะได้การทำงานที่ชัดเจนที่สุดเพื่อแปลงเป็น simple code ต่อไป

Simple Code

เป็นส่วนที่มีความใกล้เคียงภาษาโปรแกรมมากที่สุด และมีรายละเอียดครบถ้วนที่สุด พร้อมที่จะนำไปแปลงเป็นภาษาโปรแกรมได้ทันที

ตัวอย่าง

ตัวอย่าง 1 Cr. พีตันไม้ #CE57

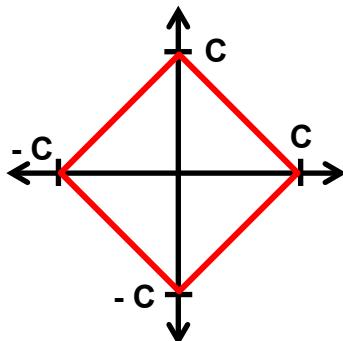


Draft Pseudo Code

- รับค่าตัวแปร N เพื่อสร้างรูป
- สร้างรูป * จากพื้นที่ภายในกราฟ (x, y) โดยมีจุดยอดตัดแกน x หรือ แกน y ที่ระยะห่างจากจุดกำกับเดิมเป็น $N - 1$
- กำหนดเงื่อนไขที่จะแสดง * และ แสดงช่องว่าง

Detailed Pseudo Code

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



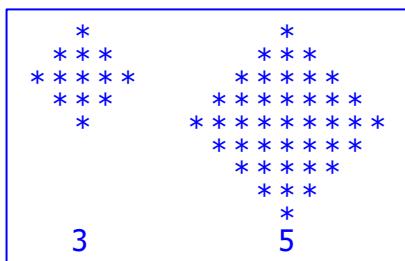
เงื่อนไขพื้นที่ภายในกราฟ คือ
 $|y| \leq |x| - C$ แสดง *

$|y| > |x| - C$ แสดง ช่องว่าง

Simple Pseudo Code

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

ตัวอย่าง 2 Cr. พีเคน #CE57



Draft Pseudo Code

- รับค่าใส่ตัวแปร n
- แสดงผล Δ บน
- แสดงผลบริหัตกลาง
- แสดงผล Δ ล่าง

Detailed Pseudo Code

- รับค่าใส่ตัวแปร n
- แสดงผลจำนวน $n-1$ บรรทัด
- แต่ละบรรทัดแสดงผลช่องว่าง * และขึ้นบรรทัดใหม่

Detailed Pseudo Code

- แสดงผล * จำนวน $2(n)-1$ ตัวแล้วขึ้นบรรทัดใหม่

Detailed Pseudo Code

- แสดงผลจำนวน $n-1$ บรรทัด
- แต่ละบรรทัดแสดงผลช่องว่าง * และขึ้นบรรทัดใหม่

Simple Command Pseudo

- รับค่าตัวแปร (n)
- กำหนด $i = 1$
- ถ้า $i \leq n-1$ ทำซ้ำ :
 - กำหนด $j = 1$
 - ถ้า $j \leq n-i$ ทำซ้ำ :
 - แสดงผลช่องว่าง
 - เพิ่มค่า j ขึ้น 1
 - กำหนด $j = 1$
 - ถ้า $j \leq 2(i)-1$ ทำซ้ำ :
 - แสดงผล *
 - เพิ่มค่า j ขึ้น 1
 - แสดงผลขึ้นบรรทัดใหม่
 - เพิ่มค่า i ขึ้น 1

Simple Command Pseudo

- กำหนด $i = 1$
- ถ้า $i \leq n-1$ ทำซ้ำ :
 - แสดงผล *
 - เพิ่มค่า i ขึ้น 1

Simple Command Pseudo

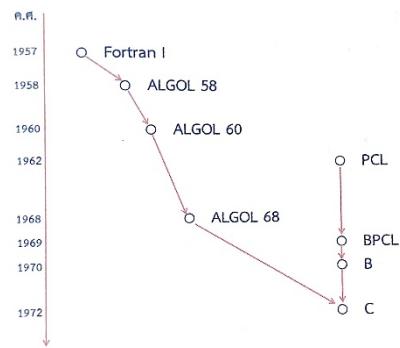
- กำหนด $i = 1$
- ถ้า $i \leq n-1$ ทำซ้ำ :
 - กำหนด $j = 1$
 - ถ้า $j \leq i$ ทำซ้ำ :
 - แสดงผลช่องว่าง
 - เพิ่มค่า j ขึ้น 1
 - กำหนด $j = 1$
 - ถ้า $j \leq 2(n-i)-1$ ทำซ้ำ :
 - แสดงผล *
 - เพิ่มค่า j ขึ้น 1
 - แสดงผลขึ้นบรรทัดใหม่
 - เพิ่มค่า i ขึ้น 1

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 หากนำรีวิวนานาการของภาษา C มาแสดงออกเป็นแผนภาพได้ดังนี้

ในปี 1978 เดนนิส ริชชี และนายเบรน เครนิกหาน (Denis Ritchie and Brian W. Kernighan) ได้แต่งหนังสือชื่อ "The C Programming Language" โดยนำเสนอภาษา 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 เครนิกหาน

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



จุดเด่นของภาษา C

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

ข้อควรรู้ : คอมไпал์เลอร์ (Compiler) จะทำหน้าที่แปลโค้ดที่เขียนทั้งหมดให้เป็นภาษาเครื่อง พร้อมกันนี้จะทำหน้าที่ตรวจสอบไวยากรณ์ของภาษา หากมีข้อผิดพลาดก็จะแจ้งให้เห็นถึงข้อผิดพลาด เมื่อกระบวนการทำงานของคอมไпал์เลอร์เสร็จสมบูรณ์ โปรแกรมจึงจะสามารถใช้งานได้ ซึ่งจะแตกต่างกับหลักการทำงานของอินเทอร์เพตเตอร์ (Interpreter) ซึ่งการทำงานจะแปลงภาษาโปรแกรม และประมวลผลคำสั่งทีละคำสั่ง

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 หรือเรียกว่า วัตถุ (Object) และพัฒนาแต่ละชิ้นมา จากนั้นจึงนำมาประกอบกันทีหลัง วิธีการนี้ทำให้ลดความสับสนเวลาพัฒนา หรือแก้ไขโปรแกรมลงมาได้มาก เพราะสามารถแก้ไขในส่วนเล็ก ๆ ขึ้นเดียวได้ ไม่ต้องแก้ไขทั้งโปรแกรม และนอกจากนี้สามารถนำร่วมกันได้ (Object-Oriented Programming) หรือส่วนที่แบ่งเป็นชิ้น ๆ นำกลับมาใช้ใหม่ หรือนำไปใช้กับโปรแกรมอื่น ๆ ที่เราต้องการโดยไม่ต้องเสียเวลาเขียนใหม่ ประหยัดเวลาได้อีกมาก ถ้าจะเปรียบเหมือนกับการสร้างรถยนต์ แทนที่จะหล่อรถยนต์ทั้งคันที่เดียว (ภาษา C) ถ้าเสียต้องทำใหม่หมด หรือส่งไปซ่อมทั้งคัน แต่ถ้าใช้แนวคิดแบบ OOP แต่ละชิ้นทำขึ้นมาต่างหาก และจึงนำมาประกอบกันทีหลัง ถ้ามีปัญหาสามารถหาสาเหตุ และนำชิ้นส่วนใหม่มาเปลี่ยนได้ง่ายกว่า

C : สามารถแก้ไขโปรแกรมได้ง่ายกว่า และสามารถทำงานได้ไวกว่า C++ เนื่องมาจากโครงสร้างของไวยากรณ์ที่มีความซับซ้อนน้อยกว่า จึงเหมาะสมกับการเขียนไดร์เวอร์ควบคุมชาร์ดแวร์, 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!

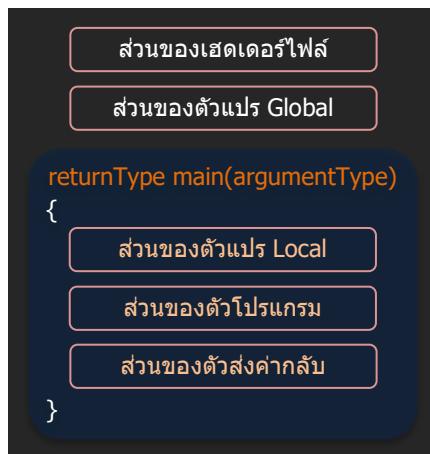
ถ้าเข้าใจโครงสร้างการเขียน C หากนำไปเขียนภาษาอื่น จะเข้าใจ หรือ Optimize ได้ง่ายขึ้น เพราะ C ไม่ค่อยมีดัวซ่วย หรือ Library มากเท่าภาษาอื่นที่ง่ายกว่า เช่นภาษา Python ภาษา C จึงเหมาะสมสำหรับฝึก Logic การทำงานตั้งแต่ต้นจนจบ ... เขียนโปรแกรมเป็นไปได้แค่เขียนโปรแกรมได้ แนวคิดต้องได้ด้วย! ...

โครงสร้างโปรแกรม

```

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

```



ヘดเดอร์ไฟล์ (Header File)

เป็นส่วนที่เรียกใช้ไลบรารี ซึ่งจะขึ้นต้นด้วยเครื่องหมาย # เสมอ โดยเป็นการนำโค้ดในไฟล์ヘดเดอร์ (.h) นั้น ๆ มาใส่ไว้ที่หัวของโปรแกรมที่เรากำลังเขียน คอมไพล์เวอร์จะทำในส่วนนี้เป็นส่วนแรก โดยสามารถเขียนได้ 2 แบบ คือ

```

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

```

ถ้าใช้เครื่องหมาย "... " คอมไпал์เวอร์จะหาไฟล์จากโฟลเดอร์ที่เก็บ Source Code ก่อน หลังจากนั้นจึงหาไฟล์ที่เก็บไว้กับตัวคอมไпал์เวอร์ แต่ถ้าใช้ <...> จะหาจากไฟล์ที่เก็บไว้กับตัวคอมไпал์เวอร์เพียงที่เดียวเท่านั้น

ข้อควรระวัง : " " ไม่ใช้ตัวเดียวกันกัน " " (Double quote)

ข้อสังเกต : ไลบรารี คือส่วนที่เก็บคำสั่งต่าง ๆ สำหรับการใช้งาน หากไม่ได้เรียกไลบรารีมาใช้อาจใช้งานคำสั่งไม่ได้ เช่น ถ้าต้องการได้ค่าของ 2 ยกกำลัง 3 จะใช้คำสั่งว่า pow(2, 3); หากไม่ได้ #include<math.h> ไว้ในส่วน Header จะทำให้คอมไпал์เวอร์ไม่รู้จักคำสั่ง pow และขึ้น error เดือนอกอกมา

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 ซึ่งบังคับให้มือย่างน้อย 1 ฟังก์ชัน คือ main() ซึ่งเป็นฟังก์ชันเริ่มการทำงานของโปรแกรม (ประมวลผลที่ฟังก์ชันนี้เป็นฟังก์ชันแรก) ขอบเขตของฟังก์ชันจะอยู่ภายในบล็อกซึ่งจะเริ่มต้นด้วยเครื่องหมาย { และสิ้นสุดด้วย } เครื่องหมาย } ฟังก์ชันส่วนมากจะมีส่วนส่งค่ากลับเมื่อทำงานจบ ฟังก์ชัน จะเห็นว่าเมื่อฟังก์ชันจะส่งค่ากลับจะต้องมีคำสั่ง return

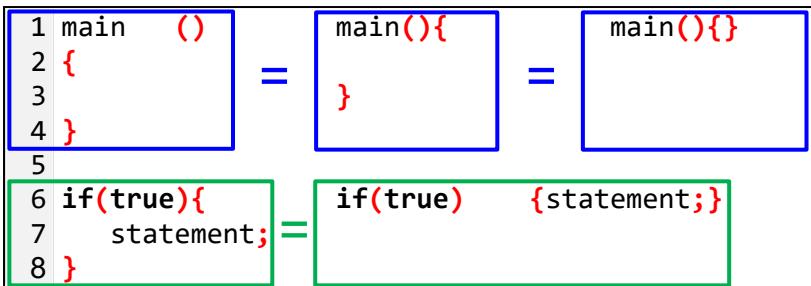
การประกาศไว้วันนี้ชื่อฟังก์ชันว่าจะส่งค่าประเภทไหนกลับ และใช้คำสั่ง return ในการส่งค่ากลับมา เช่น void (ไม่ส่งค่ากลับ ทำให้ในฟังก์ชันไม่ต้องมีคำสั่ง return) , int (ส่งค่าจำนวนเต็มกลับมา) , float (ส่งค่าทศนิยมกลับมา) , char (ส่งค่าตัวอักษรตัวเดียวกลับมา) เป็นต้น

ข้อสังเกต : ฟังก์ชัน main ถ้าไม่ได้ระบุค่าส่งกลับจะมีค่า Default เป็นจำนวนเต็มที่มีค่าเท่ากับ 0 ถ้าโปรแกรมทำงานไม่เกิดข้อผิดพลาด

ส่วนตัวโปรแกรม (Body)

จะอยู่ภายใต้ฟังก์ชันซึ่งประกอบด้วย 2 ส่วนคือ

- 3.1 คำสั่ง คือ คำที่กำหนดไว้ให้โปรแกรมทำงานตามที่ต้องการ ส่วนมากจะจบด้วยเครื่องหมาย ;
- 3.2 ตัวแปร คือ สัญลักษณ์ใช้แทนค่าต่าง ๆ ซึ่งค่าที่แทนสามารถเปลี่ยนเป็นค่าอื่นได้ โดยค่านั้นจะเป็นตัวเลข ตัวหนังสือ หรือ object ก็ได้



ข้อสังเกต : โปรแกรมจะทำงานจากช้วยไปขวา จากบนลงล่าง ดังนั้นการเร้นวรค หรือขึ้นบรรทัดใหม่ไม่มีผล

* Statement โดยทั่วไปจะหมายถึง คำสั่งส่วนมากจะปิดท้ายด้วยเครื่องหมาย ; เช่น

```

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 }

```

ข้อควรรู้ : สามารถใช้คอมเม้นต์เพื่อทำให้คำสั่งบางคำสั่งที่เราไม่ต้องการ gly เป็นคำอธิบาย หรือเพื่อไม่ให้ส่วนนั้นทำงาน สามารถทำได้ 2 วิธีคือ

- ใช้ // เพื่อทำการคอมเม้นต์ข้อความด้านหลังสัญลักษณ์นี้ ครอบคลุมภายในบรรทัดนั้น
- ใช้ /* ... */ เพื่อทำการคอมเม้นต์ข้อความที่อยู่ด้านใน โดยสามารถใช้ได้หลายบรรทัด

ข้อควรระวัง : หากใช้ /* ... */ ถ้าเจอการปิดคอมเม้นต์ตัวแรกเมื่อไรข้อความหลังจากนั้นจะไม่ถูกคอมเม้นต์ เช่น /*/*/* COMMENT *//_ตัว*/ สิ่งนี้จะไม่ถูกคอมเม้นต์ และคอมไพล์อรจะเข้าใจว่าเป็นคำสั่ง ทำให้โปรแกรมเราเกิด Syntax Error ทำให้คอมไпал์ไม่ผ่านได้

ตัวแปร (Variable)

คือ สัญลักษณ์ที่ใช้แทนตัวแหน่งบนหน่วยความจำ ใช้ในการเก็บค่าต่าง ๆ ในโปรแกรม ที่ต้องใช้สัญลักษณ์แทนเนื่องจากการอ้างถึงตัวแหน่งบนหน่วยความจำนั้นยกต่อการทำความเข้าใจ และการพัฒนาโปรแกรม ซึ่งผู้อ่านสามารถเรียกใช้ตัวแปรเพื่อใช้ในการคำนวณ หรือเพื่อเก็บค่าได้ โดยกฎการตั้งชื่อให้กับ Identifier (ชื่อที่มีอยู่ในส่วนต่าง ๆ ของโครงสร้างโปรแกรมภาษา C) ซึ่งได้แก่ ตัวแปร พิงกชัน และเลbel มีดังนี้

การตั้งชื่อไอเดียนติฟายเออร์ (Identifier Names)

- ชื่อประกอบด้วยตัวอักษรพิเศษไม่ได้ เช่น & @ # & เป็นต้น
- ภายในชื่อประกอบด้วยช่องว่าง หรือแท็บไม่ได้
- ในภาษา C / C++ เป็นแบบ Case-Sensitive คือตัวอักษรตัวใหญ่และตัวอักษรตัวเล็กถือเป็นคนละตัวกัน เช่น pom, Pom, PoM, POM จะถือว่าชื่อห้องสีตัวนี้เป็นคนละตัวแปรกัน และคอมไพล์อร์บางตัวไม่สามารถตั้งชื่อตัวแปรได้เกิน 31 ตัวอักษร
- สามารถประกอบด้วยตัวอักษร ตัวเลข และเครื่องหมาย Underscore (_) แต่ตัวอักษรตัวแรกไม่สามารถเป็นตัวเลขได้ โดยในชื่อสามารถมีตัวเลขประกอบได้ เช่น _Number, x1, _708, pRayut44 เป็นต้น
- ตั้งชื่อตัวแปรซ้ำกันไม่ได้ ไม่เช่นนั้นคอมไпал์อรจะไม่รู้ว่าตัวแปรไหนเป็นตัวไหน เช่นเดียวกันกับคำส่วน
- ชื่อที่ตั้งขึ้นจะต้องไม่ซ้ำกับคำส่วน (Reserved Word) ภายในโปรแกรม ซึ่งมีดังนี้

การตั้งชื่อตัวแปร และพิงกชันให้เป็นสากล : จะใช้หลัก Lower Camel Case ตัวแรกตัวเล็ก ถ้ามีคำต่อ กันจะขึ้นต้นตัวใหญ่ 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	num
ch	ch
str	str J a d e S P K

ข้อควรรู้ : สามารถกำหนดค่าเริ่มต้นเข้าไปด้วยตัวแปรได้เลย เช่น

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

เมื่อโปรแกรมทำงานถึงตรงที่ประกาศตัวแปรไว้ โปรแกรมจะทำการจดพื้นที่ในหน่วยความจำ ถ้ามีการกำหนดค่าให้ตัวแปร ค่าเหล่านั้นก็จะถูกเก็บลงในหน่วยความจำตามตำแหน่งที่ได้จัดไว้ ดังรูปทางด้านซ้าย

ข้อสังเกต : ตัวแปร ch ค่าที่เก็บลงหน่วยความจำ (กล่อง) จะเป็น M แต่เป็นตัวเลขรหัสแอสกี้ของ M ซึ่งจะได้ศึกษาในส่วนต่อไป

ชนิดของข้อมูล (Data Type)

ข้อมูลแต่ละชนิดมีขอบเขตความสามารถในการเก็บข้อมูลต่างกัน บางชนิดสามารถเก็บข้อมูลได้มากก็จะใช้หน่วยความจำมากตามไปด้วย จึงต้องคำนึงถึงความเหมาะสมในการใช้ตัวแปรชนิดต่างๆ และหากเก็บค่าของข้อมูลที่มีค่ามากกว่าขอบเขตที่ชนิดของข้อมูลนั้นสามารถใช้เก็บค่าได้ หรือเก็บค่าในชนิดของข้อมูลที่ผิดประเภท จะสามารถทำให้โปรแกรมเกิดข้อผิดพลาดได้ โดยชนิดของข้อมูลในโปรแกรมจะมี 4 ประเภทหลักๆ 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 เช่นข้ออยู่กับระบบปฏิบัติการ 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

- ต้องเป็นตัวเลขไม่มีจุดทศนิยม หากกำหนดเป็นทศนิยมข้อมูลอาจมีการสูญหาย
- ห้ามใช้เครื่องหมาย , หรือช่องว่างคั่นระหว่างตัวเลข เช่น 1 000 000 หรือ 1,234 หรือ 12, 3 เป็นต้น
- ถ้าเป็นค่าวากไม่จำเป็นต้องใส่เครื่องหมาย + นำหน้าค่า แต่ถ้าเป็นค่าลบต้องใส่เครื่องหมาย - นำค่าลบออก
- ช่วงตัวเลขจำนวนเต็มควรอยู่ในช่วงชนิดข้อมูลนั้นๆ
- 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)

เป็นชนิดข้อมูลแบบตัวเลขทศนิยมที่สามารถนำไปคำนวณทางคณิตศาสตร์ได้ อาจมีจุดทศนิยม หรือไม่มีจุดทศนิยมก็ได้ โดยสามารถเขียนในรูปแบบเลขทศนิยม เช่น 1112.50, -250.0 หรือรูปแบบเลขทศนิยมยกกำลัง เช่น 3.456E+3 (มีค่า 3.456×10^3 หรือ 3,456.0), 7.89E+4 (มีค่าเป็น 7.89×10^4 หรือ 78,900.0) ความแม่นยำของทศนิยมในจำนวนจริง float 6 หลัก, double 17 หลัก และ long double 34 หลัก ซึ่งไม่สามารถใช้ได้ตามอธิบายในข้อควรรู้[*]

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

การกำหนดค่าให้กับตัวแปรชนิดจำนวนจริง

จุดสังเกตจะพบว่า ชนิดข้อมูลแบบจำนวนจริงจะเป็นแบบคิดเครื่องหมาย (signed) เสมอ โดยผู้อ่านจะสามารถกำหนดค่าตัวแปรโดยคำนึงถึงข้อกำหนดดังนี้

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

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

3. กรณีเป็นค่าวากไม่จำเป็นต้องใส่เครื่องหมาย + หน้าค่า แต่กรณีเป็นค่าลบต้องใส่เครื่องหมาย - หน้าค่าเสมอ

4. การเขียนในรูปแบบใช้ตัวอักษร E ค่าที่ถูกกำหนดสามารถกำหนดได้ทั้งค่าวากและค่าลบ

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) เป็นชนิดข้อมูลแบบอักขระตัวเดียว ซึ่งมีขนาด 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 }
```

```
C:\Users\ \Desktop\Untitled1.exe
Char A = A
ASCII A = 65
Char A = C
ASCII A = 67
Process exited after 0.06027
seconds with return value 0
Press any key to continue . . .
```

```
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 }
```

```
C:\Users\ \Desktop\Untitled1.exe
String : Hello
test[0] = H
test[1] = e
test[2] = l
test[3] = l
test[4] = o
Process exited after 0.836
seconds with return value 0
```

ชนิดข้อมูลแบบสายอักขระ (Group of Character Type)

ในภาษา C ไม่มีข้อมูลประเภท String แต่สามารถใช้แคลบดับ (Array) เข้ามาช่วย ซึ่งการกำหนดค่าจะใช้ " " ครอบสายอักขระที่ต้องการจะกำหนดค่าตั้งตัวอย่างด้านขวา

ข้อควรรู้ : การเก็บข้อความ (String) ใช้ตัวแปรประเภท char (ชนิดข้อมูลแบบตัวอักษร) มาต่อ กันเรื่อยๆ จนได้เป็นข้อความ (สายของอักขระ) ซึ่งการนำตัวแปร หรือกลุ่มข้อมูลประเภทเดียวกันมาเรียงลำดับกัน จะเรียกว่า แคลบดับ หรืออาร์เรย์ (Array)

Null Character และ Empty String

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

การกำหนดค่าเริ่มต้นของตัวแปรข้อมูลนิดสายอักขระเป็นข้อความ (**char** b[3] = "ab";) อินเด็กซ์ (Index) ที่เหลือ คือตัวแปร b[2] จะถูกกำหนดค่าเป็น "" อัตโนมัติ และจะเรียกค่าที่เติมให้อัตโนมัตินี้ว่า 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) ไม่ใช่ Empty String และ Index ที่ 11 มีค่าเท่ากับ "" 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 หรือภายในขอบเขตของบล็อกได้ 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 กnum ซึ่งมีค่าเท่ากับ 10 เข้ามาในฟังก์ชันจากนั้นก็นำไปคูณกับ n ที่เป็นตัวแปรโลคอลได้ผลเท่ากับ 40 และเก็บค่าไว้ที่ตัวแปร sum และจึงส่งค่ากลับสู่ฟังก์ชัน main() ในบรรทัดที่ 9 นั้นเอง โดยจะแสดงผล **Error : Undefined symbol 'n' in function main**

การกำหนดค่าให้ตัวแปร (Variable Assignment)

1. กำหนดค่าตอนประกาศตัวแปร

||| `dataType varName = value;`

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

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

ข้อสังเกต : การประกาศตัวแปรสำหรับเก็บข้อความจะต้องเพื่อขนาดเพิ่ม 1 ตัวเสมอสำหรับเก็บ \0 ในตำแหน่งสุดท้าย

2. กำหนดค่าภายในฟังก์ชัน (ภายหลังจากการประกาศตัวแปร)

||| `left = right;`

จะใช้ตัวดำเนินการ = คือการนำค่าด้านขวา (right) ของตัวดำเนินการ = ที่ถูกกระทำเรียบร้อย มาใส่ในตัวถูกกระทำการด้านซ้าย (left) เช่น

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

<u>โดยที่</u> การใช้คำสั่งนี้ควรใช้ในส่วนของ 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)` คือเอาไว้มาแทน (`new`) “การ `define` จึงคล้ายการกำหนดค่าให้ตัวแปร” ถ้าใช้ `#define` เปรียบเสมือนกำหนดค่าให้ตัวแปร ตัวแปรที่ถูก `define` ไว้จะไม่สามารถเปลี่ยนแปลงค่าในฟังก์ชันได้ ๆ ได้อีก และการใช้ `#define` สามารถใช้กำหนดค่าสั่งใส่ในสิ่งที่ต้องการนำมาแทน (`new`) เพื่อใช้สิ่งที่กำหนดแทนค่าสั่งเดิม ๆ เหล่านั้นได้ เช่น `#define print printf` เพื่อใช้งาน `print` แทนค่าสั่ง `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"`
โดยที่ COUNT คือ ค่าที่ต้องการกำหนดให้กับค่าคงที่ `#define NEWLINE '\n'`
โดยที่ CH คือ ค่าที่ต้องการกำหนดให้กับค่าคงที่ `#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 คอมไพล์เวอร์จะทำหน้าที่แปลงข้อมูลให้อัตโนมัติโดยแปลงชนิดข้อมูลที่มีนัยสำคัญต่ำ ไปเป็นชนิดข้อมูลเดียวกันที่มีนัยสำคัญสูงกว่าในชุดคำสั่งนั้น ๆ โดยมีลำดับนัยสำคัญดังนี้

ต่ำ `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 (วงเล็บมีลำดับความสำคัญสูงที่สุดจึงต้องทำในวงเล็บก่อน)		
• (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

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

การแสดงผลทางหน้าจอ (Display Data)

มีคำสั่งที่นิยมใช้แสดงผลอยู่ในshedเตอร์ไฟล์ 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 หรือรหัสคำสั่งพิเศษ/รหัสควบคุม (Escape Sequence)

จากตัวอย่าง

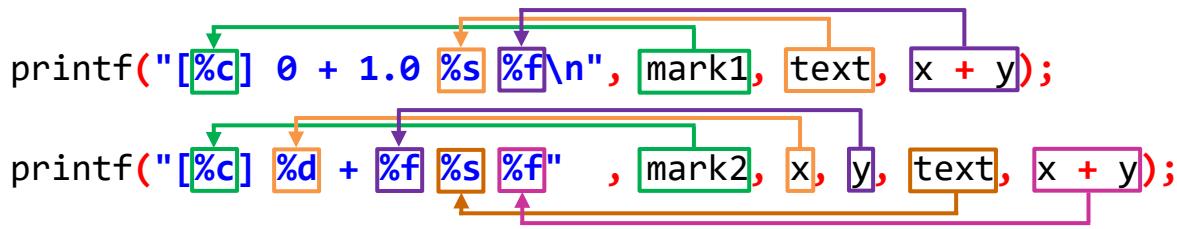
<pre>1 #include<stdio.h> 2 3 main() 4 { 5 int x = 0; 6 float y = 1.0; 7 char mark1 = 'B', mark2 = 'C'; 8 char text[] = "is"; 9 printf("[A] 0 + 1.0 is 1.00\n"); 10 printf("[%c] 0 + 1.0 %s %f\n", mark1, text, x + y); 11 printf("[%c] %d + %f %s %f" , mark2, x, y, text, x + y); 12 }</pre>	
---	--

ข้อ A เป็นการแสดงผลข้อความทั้งหมดจะได้ผลลัพธ์ที่อยู่ในเครื่องหมาย " " ออกมากันหมด และตอนสุดท้ายจะทำการขึ้นบรรทัดใหม่จาก \n ซึ่งเป็นรหัสคำสั่งพิเศษ (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 : ค่าที่แสดงผลออกมาจะแสดงผลทุกตัวเหมือนในเครื่องหมาย " " ทุกประการยกเว้นรหัสคำสั่งพิเศษ/รหัสควบคุมเช่น \t, \r และ format ในการแสดงผลเช่น %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 }

```

C:\Users\ \Desktop\Untitled1.exe
1. 29999995
Process exited after 0.02333 seconds with return value 0
Press any key to continue . . .

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

จากตัวอย่าง เมื่อสั่งแสดงผลเป็นจำนวนจริง ถ้าไม่ได้กำหนดการแสดงผลเพิ่มเติม จะแสดงทศนิยมออกมากทั้งหมด 6 ตำแหน่งตามความแม่นยำของตัวแปรประเภท float ถ้าสั่งแสดงทศนิยมเลขทศนิยมที่ได้จะถูกปัดตามหลักคณิตศาสตร์ให้อัตโนมัติ และหากสั่งแสดงผลเลขทศนิยมที่ถูกเก็บลงในหน่วยความจำอย่างละเอียด (ทศนิยมหลายตำแหน่ง) เกินความแม่นยำของตัวแปรแต่ละประเภท จะพบว่าทศนิยมในตำแหน่งหลัง ๆ บางตัวมีค่าผิดเพี้ยน และถ้าสั่งแสดงทศนิยมโดยใช้รูปแบบการแสดงผลผิดประเภทค่าที่ได้ออกมาจะเป็น 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", 10.0);
12 }
13

```

C:\Users\ \Desktop\Untitled1.exe
3.140000
3.1400001049
1.5000000000
3.333333
1.666667
0
Process exited after 0.02907 seconds with return value 0
Press any key to continue . . .

↑
MAX : 25 Rows / Page

← MAX : 80 Characters →

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

ข้อสังเกต : หากนำข้อมูลชนิดทศนิยมเก็บไว้ในตัวแปรประเภทจำนวนเด้มจะถูกปัดเศษทิ้ง ไม่ปัดขึ้น หรือปัดลงตามหลักคณิตศาสตร์

ข้อสังเกต : หากนำข้อมูลชนิดข้อความเก็บไว้ในตัวแปรประเภทอักขระ ตัวสุดท้ายของข้อความจะถูกเก็บลงตัวแปร

ข้อสังเกต : หากนำข้อมูลชนิดจำนวนเด้มมาดำเนินการกันจะได้ผลลัพธ์เป็นข้อมูลจำนวนเด้มด้วยเหมือนกัน ถึงแม้จะกำหนดให้แสดงผลเป็นทศนิยม วิธีแก้ไข คือใช้ (float)centimeter / (float)100 หรือ (float)centimeter / 100 หรืออธิบายหนึ่งคือแก้เป็น 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 }
12
13

```

C:\Users\ \Desktop\Untitled1.exe
11 y 11
Meter = 0.000000
Process exited after 0.02318 seconds with return value 0
Press any key to continue . . .

Line	Col	File	Message
8	12	C:\Users\ \Desktop\Untitled1.cpp	[Warning] character constant too long for its type [enabled by default]
		C:\Users\ \Desktop\Untitled1.cpp	In function 'int main()'
			[Warning] overflow in implicit constant conversion [-Woverflow]

รหัสรูปแบบการแสดงผลข้อความ และการจัดรูปแบบการแสดงผลข้อมูลของคำสั่ง printf()

การแสดงผลด้วยคำสั่ง printf() นั้นสามารถแสดงผลได้เกือบทุกชนิดข้อมูล โดยใช้รหัสรูปแบบการแสดงผลแทนค่าจากตัวแปร นิพจน์ หรือค่าคงที่ชนิดต่าง ๆ ในข้อความ รหัสรูปแบบการแสดงผลควรใช้ให้ถูกต้องตามชนิดของข้อมูลที่ต้องการ หากเลือกใช้ผิดประเภทอาจทำให้การแสดงผลผิดพลาดได้ และนอกจากนี้ยังมีรหัสคำสั่งพิเศษ หรือรหัสควบคุม (Escape Sequence) เพื่อใช้ควบคุม หรือแสดงผลในรูปแบบพิเศษ

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

ข้อควรรู้* : “ไม่สามารถใช้ข้อมูลชนิด 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	จำนวนจริงแบบเอ็กซ์โพเนนต์
%f	จำนวนจริง float
%lf	จำนวนจริง double
%g หรือ %G	จำนวนจริง (General format)
%i	จำนวนเต็มชนิด int
%o	เลขฐานแปด
%p	พอยน์เตอร์
%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 \ ตัวเดียวตัวคอมไพล์เตอร์จะเข้าคตัวอักษรถัดไปว่าสามารถประกอบเป็นรหัสคำสั่งพิเศษ หรือรหัสควบคุม (Escape Sequence) ได้ หรือไม่ (% ก็เช่นกันจะเข้าคตัวถัดไปว่าสามารถประกอบกันเป็นรหัสรูปแบบได้ หรือไม่) ถ้าหากเป็นรหัสคำสั่งพิเศษก็จะแสดงรหัสพิเศษออกมา เช่น ในบรรทัดที่ 5 จะมี (\E) และขึ้นบรรทัดใหม่ (\n) ถูกแสดงผลเป็นรหัสพิเศษ, ส่วนบรรทัดที่ 6 จะมี ("") และขึ้นบรรทัดใหม่ (\n) เป็นรหัสพิเศษที่ถูกแสดงผลออกมา และถ้าหากไม่ใส่ " ปิดท้ายจะทำให้คำสั่ง printf("); ขาด "(ไม่ครบ Syntax) และในทางกลับกันถ้าหากไม่ใช้รหัสพิเศษจะแสดงตัวอักษรนั้นออกมา เช่น ในบรรทัดที่ 7 A (\A) จะถูกแสดงผลออกมาเป็นตัวอักษรตามปกติ หรือในบางคอมไพล์เตอร์จะแจ้งเตือน error ขึ้น [Warning] unknown escape sequence: '\A'

ข้อควรรู้ : จากตัวอย่างเมื่อสั่งแสดงผล % ตัวเดียวจะไม่เกิดอะไรขึ้นที่หน้าจอ ต้องใช้ %% ในการแสดงผล

การจัดพื้นที่ และการกำหนดการแสดงผลข้อมูลของคำสั่ง printf()

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

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

โดยที่ [m] เป็นจำนวนช่องสำหรับใช้ในการแสดงข้อความ 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("\\zyxA ");
8 }
9 .
10 abc
11 ";
12 ny-7m
13 \zyxA
14 
```

C:\Users\...\Desktop\Untitled1.exe

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

ในการนี้แสดงค่าจำนวนเต็ม มีรูปแบบการแสดงผลดังนี้

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

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

ในการนี้แสดงค่าจำนวนจริง มีรูปแบบการแสดงผลดังนี้

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

โดยที่ [m] เป็นจำนวนของสำหรับใช้ในการแสดงตัวเลข เช่น printf("%8f", 1.0);
[n] เป็นจำนวนหลักทศนิยมที่ต้องการแสดงผล printf("%.2f", 2.5);
variable เป็นค่าข้อมูลหรือตัวแปรที่ต้องการแสดงผล 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 หรือขิดขวาในส่วนการแสดงผล หรืออาจจะต้องการกำหนดเครื่องหมายให้กับ
ข้อมูลตัวเลข ภาษา C ได้กำหนดรูปแบบการแสดงผลต่าง ๆ เหล่านั้นไว้ดังนี้

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

ตัวอย่าง การจัดรูปการณ์แสดงผลของตัวเลขจำนวนเต็ม 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 }
```

```
C:\Users\...\Desktop\Untitled1.exe
Right justify : 112
Left justify : 112
Default minus : -123
Default zero : 0
Default plus : 123
Display minus : -123
Display zero : +0
Display plus : +123
Default : 1234
Zero show : 0000001234
String : long..
Float : 3.14
Float : 12.385
No# : 144 64 10.2
Use# : 0144 0x64 10.2000
Process exited after 0.02583
seconds with return value 0
Press any key to continue . . .
```

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 หรือเป็นค่าข้อความภายในเครื่องหมาย " "

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
13 seconds with return value 0
14 Press any key to continue .
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 *Edit Thailand4.0
12
13
14
15
```

ตัวดำเนินการ (Operator)

ในการเขียนโปรแกรมทุก ๆ โปรแกรมจะต้องมีการประมวลผลเข้ามาเกี่ยวข้องด้วยเสมอ และสิ่งที่ทำให้เกิดการประมวลผลนั้นก็คือ 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, พึงกชัน หรือค่าคงที่มาสัมพันธ์กับตัวดำเนินการ (Operator) เช่น C + 1, A + B - (C - 7), A! = B, A && B เป็นต้น	

ข้อควรรู้ : การทำงานของ % ใช้ได้กับจำนวนเต็มเท่านั้น

การทำงานของ +, -, *, และ / ที่มีเฉพาะเลขจำนวนเต็ม

การทำงานของ +, -, *, และ / ที่มีเลขทศนิยมอยู่ด้วย

การทำงานของ +, -, *, และ / กับข้อมูลชนิดตัวอักขระ

จะได้ผลลัพธ์เป็นจำนวนเต็มเสมอ

จะได้ผลลัพธ์เป็นเลขจำนวนเต็มเสมอ

จะได้ผลลัพธ์เป็นเลขทศนิยมเสมอ (ตามกฎของ

การแปลงชนิดของข้อมูล)

จะทำโดยแปลงตัวอักขระเป็นค่า ASCII Code ก่อน
จะนำไปประมวลผล

ตัวดำเนินการกำหนดค่า (Assignment Operator)

ใช้สำหรับกำหนดค่าให้กับตัวแปรทางด้านซ้ายของตัวดำเนินการ โดยจะมีหลักการทำงานที่แตกต่างกันไปและกำหนดให้มีตัวแปรดังนี้ ตัวแปร A และ B เป็นชนิดข้อมูลแบบตัวเลขจำนวนเต็ม กำหนดให้เท่ากับ 10 และ 4 ตามลำดับ
ตัวแปร C เป็นข้อมูลชนิดตัวอักษร กำหนดให้เท่ากับ 'A'
ตัวแปร D เป็นชนิดข้อมูลแบบจำนวนเลขทศนิยม กำหนดให้เท่ากับ 3.12 ดังนี้

ตัวดำเนินการ	ความหมาย
= : เท่ากับ	นำค่าตัวถูกกระทำการทางด้านขวามาใส่ในตัวถูกกระทำการทางด้านซ้าย C = 'A' → C = 'A' → C มีค่าเท่ากับ 65 (ASCII Code) A = 10 → A = 10 → A มีค่าเท่ากับ 10
+ = : บวกเท่ากับ	นำค่าตัวถูกกระทำการทางด้านซ้าย เท่ากับค่าตัวถูกกระทำการทางด้านซ้ายบวกกับ ค่าตัวถูกกระทำการทางด้านขวา A += B → A = A + B = 10 + 4 → A มีค่าเท่ากับ 14 A += D → A = A + D = 10 + 3.12 = 13.120000 → A มีค่าเท่ากับ 13 (เนื่องจาก A เป็นข้อมูลชนิดเลขจำนวนเต็ม) A += C → A = A + C = 10 + 65 → A มีค่าเท่ากับ 75 (การ + - * / % กับตัวอักษรจะแปลงเป็นค่า ASCII ซึ่งในที่นี้ 'A' มีค่า ASCII = 65) D += A + B → D = D + (A + B) = 3.12 + (10 + 4) = 3.12 + (14) → D = 17.120000 ดังนั้น D มีค่าเท่ากับ 17.12
- = : ลบเท่ากับ	นำค่าตัวถูกกระทำการทางด้านซ้าย เท่ากับค่าตัวถูกกระทำการทางด้านซ้ายลบกับ ค่าตัวถูกกระทำการทางด้านขวา A -= A → A = A - A = 10 - 10 → A มีค่าเท่ากับ 0 A -= A - B → A = A - (A - B) = 10 - (10 - 4) → A มีค่าเท่ากับ 4
* = : คูณเท่ากับ	นำค่าตัวถูกกระทำการทางด้านซ้าย เท่ากับค่าตัวถูกกระทำการทางด้านซ้ายคูณกับ ค่าตัวถูกกระทำการทางด้านขวาแสดงผล D *= 2 → D = D * 2 = 3.12 * 2 = 6.240000 → D มีค่าเท่ากับ 6.24
/ = : หารเท่ากับ	จะนำค่าตัวถูกกระทำการทางด้านซ้าย เท่ากับค่าตัวถูกกระทำการทางด้านซ้ายหารกับ ค่าตัวถูกกระทำการทางด้านขวา A /= B / 2 → A = A / (B / 2) = 10 / (4 / 2) → A มีค่าเท่ากับ 5
% = : หารเอาเศษเท่ากับ	จะนำค่าตัวถูกกระทำการทางด้านซ้าย เท่ากับเศษเหลือจากการหารระหว่าง ค่าตัวถูกกระทำการทางด้านซ้าย กับค่าตัวถูกกระทำการทางด้านขวา A %= B → A = A % B = 10 % 4 → A มีค่าเท่ากับ 2 D = D - A % B → D = D - (A % B) = 3.12 - (10 % 4) = 3.12 - (2) → D = 2.120000 (% สำคัญกว่า -) → D มีค่าเท่ากับ 1.12

ตัวดำเนินการ unary (Unary Operator)

ตัวดำเนินการ	ความหมาย
คือ การเพิ่ม ++ ค่าให้กับตัว แปรนี้ค่า	Postfix X = A++ → X = A → A = A + 1 → กำหนดค่าให้ X ก่อนแล้วค่อยเพิ่ม A 1 ค่า
	Prefix X = ++A → A = A + 1 → X = A → เพิ่ม A ก่อน 1 ค่า จึงค่อยกำหนดค่าให้กับ X
คือ การลดค่า -- ให้กับตัวแปร นี้ค่า	Postfix X = A-- → X = A → A = A - 1 → กำหนดค่าให้ X ก่อนแล้วค่อยลด A 1 ค่า
	Prefix X = --A → A = A - 1 → X = A → เพิ่ม A ก่อน 1 ค่า และค่อยกำหนดค่าให้กับ X
+ เครื่องหมายบวก	Prefix A = +2 → A เท่ากับ 2 (กรณีเป็นค่าบวกจะใส่เครื่องหมาย + หรือไม่ใส่ก็ได้)
- เครื่องหมายลบ	Prefix A = -2 → A เท่ากับ -2 A = -A → A เท่ากับ 2 (เมื่อใส่ลบหน้าตัวแปรได้ ๆ 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
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 . . .

จากรูปนี้เราเห็นว่าตัวแปร x เป็น 10 จะถูกนำไปกำหนดค่าให้ตัวแปร x จากนั้น a จะมีค่าเพิ่ม 1 เป็น 11 แต่บรรทัดที่ 14 ค่า a จะมีค่าเพิ่ม 1 เป็น 11 ก่อน จากนั้นจึงนำค่า 11 กำหนดค่าให้ตัวแปร x

ข้อสังเกต : บางคนสงสัยว่าตัวแปรชื่อเดียว (x และ a) ประกาศซ้ำอีกในฟังก์ชันได้ไหม คำตอบคือ “ไม่” แต่ในตัวอย่าง ตัวแปร x และ a ที่ประกาศไว้ในบรรทัดที่ 5 เป็นตัวแปร local ภายในปีกกา หรือบล็อก (block) ระหว่างบรรทัดที่ 4 และ 10 ภายในฟังก์ชัน main() อีกที หากโปรแกรมทำงานผ่านช่วงปีกกา หรือบล็อก (block) ปีกนี้จะไปตัวแปร ก็จะไม่มีตัวแปร x และ a อีก หากต้องการใช้งานต้องประกาศตัวแปรใหม่อีกรอบ

ตัวดำเนินการเปรียบเทียบ (Comparison Operator)

เป็นตัวดำเนินการสำหรับเปรียบเทียบข้อมูลระหว่างตัวถูกกระทำการทางด้านซ้าย 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 กำหนดให้นิพจน์สีเขียวมีความจริงเป็นจริง และนิพจน์สีแดงมีความจริงเป็นเท็จ จะได้ผลลัพธ์ออกมาดังนี้

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 ซึ่งผลลัพธ์ที่ได้จะได้แค่จริง 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 = $(i < 10) ? 15 : 5;$

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

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

ตัวดำเนินการบอกรากขนาดชนิดข้อมูล (Sizeof Operator)

ตัวดำเนินการชนิดนี้ใช้สำหรับหาขนาดชนิดของข้อมูลต่าง ๆ ที่ผู้อ่านต้องการทราบ โดยมีรูปแบบการใช้งานดังนี้

||| sizeof(data);

โดยที่ sizeof คือ ตัวดำเนินการบอกรากขนาดชนิดข้อมูล

data คือ ชนิดข้อมูล หรือตัวแปรที่ต้องการทราบขนาด

ตัวดำเนินการระดับบิต (Bitwise Operator)

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

ข้อควรรู้ : มิต (bit) เป็นหน่วยข้อมูลที่เล็กที่สุด โดยข้อมูลหนึ่งบิตจะมีสถานะได้เป็น 2 สถานะคือ 0 (ปิด) หรือ 1 (เปิด) หรือเรียกว่าก็อย่างว่า เลขฐานสองนั่นเอง

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 จะสังเกตเห็นได้ว่าผลลัพธ์จะเป็น 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 จะได้เท่ากับผลหารของ x กับ 2^y

ก่อน Shift	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr></table>	1	1	0	1	0	1	0	1	0
1	1	0	1	0	1	0	1	0		
หลัง Shift	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr></table>	0	1	1	0	1	0	1	0	1
0	1	1	0	1	0	1	0	1		

ข้อควรรู้ : เนื่องจากไม่มีข้อมูลก่อนหน้าสำหรับใช้ในการเลื่อนบิตมาทางขวา ทำให้ บิตแรกหลังจาก shift จะถูกกำหนดค่าเป็น 0 และเมื่อทำการเลื่อนบิตไปทางขวา บิตสุดท้ายก่อน Shift จะถูกตัดทิ้ง (ช่องที่ไม่ลงสี)

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

ก่อน Shift	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr></table>	1	1	0	1	0	1	0	1	0
1	1	0	1	0	1	0	1	0		
หลัง Shift	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	1	0	1	0	1	0	1	0	0
1	0	1	0	1	0	1	0	0		

ข้อควรรู้ : เนื่องจากไม่มีข้อมูลด้านหลังสำหรับใช้ในการเลื่อนบิตมาทางซ้าย ทำให้ บิตสุดท้ายหลังจาก shift จะถูกกำหนดค่าเป็น 0 และเมื่อทำการเลื่อนบิตไปทางซ้าย บิตแรกก่อน Shift จะถูกตัดทิ้ง (ช่องที่ไม่ลงสี)

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

Bitwise One's Complement (~)

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

เป็นตัวดำเนินการสำหรับปรับค่าของบิตเป็นค่าตรงข้ามกล่าวคือ 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 หมายถึงลำดับการคิด เช่น $x = 5 - 4 + 3$; คิดจากซ้ายไปขวา ไม่ใช่ + สำคัญกว่า -

การรับข้อมูล (Input Data)

มีคำสั่งที่นิยมใช้ ชึ่งมีอยู่ในsheddeoreไฟล์ 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 Enter character : Your input : A
12 -----
13 Process exited after 1.146
14 seconds with return value 0
15 Press any key to continue . . .
```

```
1 #include<stdio.h>
2
3 main()
4 {
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
13 -----
14
15
```

ข้อสังเกต : คำสั่ง getch() เป็นคำสั่งที่อยู่ในsheddeoreไฟล์ในไลบรารี conio.h ในคอมไฟเลอร์เก่า ๆ หากต้องการใช้ ให้ include เข้ามาด้วย

2. การรับข้อมูลชนิดข้อความจากคีย์บอร์ดด้วยคำสั่ง 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 : หากรับข้อมูลชนิดข้อความ และข้อมูลชนิด Array ด้วยคำสั่ง scanf() ไม่จำเป็นต้องใช้เครื่องหมาย & นำหน้าตัวแปรที่ใช้รับค่าข้อมูล

```
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	จำนวนจริงแบบເລກຂໍໂພແນນດ໌
%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 (พิมพ์ເລັກ/ໃຫຍງ)
และอื่น ๆ เมื่อ input แบบ format ใน printf	

จากตัวอย่าง เมื่อกรอกเพศ (M) และกด Enter

จากนั้นใส่ชื่อ (JadeSPK) และกด Enter

ใส่อายุ (9) และกด 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 : หากใส่การรับค่าตัวอักษรระดับเดียว (char) ด้วย scanf ต่อจาก scanf อีกที (บรรทัดที่ 10) จะทำให้เกิด bug ขึ้น เนื่องจากอักษรตัวเดียวที่รับค่าไปจะเป็น \n คือการกด Enter เพื่อเสร็จสิ้นการรับค่าด้วย scanf ของคำสั่งก่อนหน้า (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     printf("2. Enter gender (M or F)\n");
9     scanf("%s %c", name, &gender);
10    if(gender == '\n')    printf("\\n");
11    printf("Result : [%c] %s", gender, name);
12 }
```

```
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 สามารถรับหlays ค่า ใส่ในแต่ละตัวแปรได้ภายในคำสั่งเดียวเพียงกำหนด 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 }
```

```
C:\Users\...\Desktop\Untitled1.exe
778
112
Output : 112
Process exited after 9.777
seconds with return value 0
Press any key to continue . . .
```

การกำหนดลำดับการรับข้อมูลของคำสั่ง scanf()

สามารถกำหนดลำดับการรับข้อมูลโดยใช้เครื่องหมาย * ได้ หากลำดับของข้อมูลที่รับเข้ามามีเครื่องหมาย * ข้อมูลที่รับเข้ามาในลำดับดังกล่าวจะไม่ถูกจัดเก็บลงตัวแปร ดังตัวอย่างต่อไปนี้

การกำหนดจำนวนตัวอักษรในการรับค่าของข้อมูลนิดข้อความของคำสั่ง scanf()

สามารถกำหนดความยาวข้อความที่ต้องการรับสูงสุดได้โดยการใส่ตัวเลขหน้า `s` ใน `%s` 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 ส่วนมากในลีมันจะกล่าวถึงการกำหนดรูปแบบการรับข้อมูลเอง (format_n) สามารถทำได้โดยการใช้ Regular Expression เข้ามาช่วยซึ่งสามารถศึกษาเพิ่มเติมได้จาก <https://regextester.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 – ช่วยในการตีที่อักษรที่ต้องการเป็น range ได้ <ul style="list-style-type: none"> • [abc] แมทช์กับ 'a', 'b', 'c' • [abcx-z] หรือ [a-cx-z] แมทช์กับ 'a', 'b', 'c', 'x', 'y', 'z' • [A-Za-z] แมทช์กับ 'a' ถึง 'z' และแมทช์กับอักษรตัวใหญ่ด้วย 'A' ถึง 'Z' * [ก-݄] <u>สำหรับภาษาไทยซึ่งมีทั้ง พยัญชนะ สระ ตัวเลข ไทยมากมาย ให้เริ่มด้วย 'ก' ถึง '݄' จะได้ครอบคลุมตัว</u>
[^]	เหมือนแคส [] แต่เปลี่ยนเป็น <u>ไม่เจออักษรในนี้แทน หากเจอตัวอักษรในกลุ่มนี้จะทำการหยุดรับค่าทันทีสำหรับภาษา C/C++</u>

เช่น	%[^ก] รับค่าเรียจจนกว่าเจอขั้นบรรทัดใหม่	→ input : asduh[enter]	output : asduh
	%[^ab] รับค่าเรียจจนกว่าเจอ a หรือ b	→ input : eieiaty	output : eiei
	%[^a-c] รับค่าเรียจจนกว่าเจอ a, b หรือ c	→ input : wowcoasd	output : wow
	%[ab] รับค่าจะสิ้นสุดเมื่อไม่พบอักษร a หรือ b	→ input : eieiatyAerb	output :
	%[A-C] รับค่าจะสิ้นสุดเมื่อไม่พบอักษร A, B หรือ C	→ input : ABBCAaAC	output : ABBCA
	%[A-C0-6] รับค่าจะสิ้นสุดเมื่อไม่พบอักษร A, B หรือ C และ 0-6	→ input : A0vh0Ba6z	output : A0

ข้อสังเกต : รหัสคำสั่งพิเศษแต่ละตัวจะมีรหัสแอสกี้ (ASCII Code) เช่น new line หรือขั้นบรรทัดใหม่ (ASCII Code = 10, LF), carriage return หรือกลับไปยังตัวแรกของบรรทัด (ASCII Code = 13, CR)

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

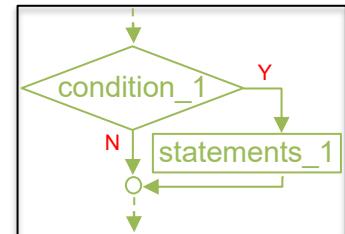
ควบคุมทิศทางการทำงานโปรแกรมด้วยคำสั่งตัดสินใจ (Decision Control Statement)

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;
}
```

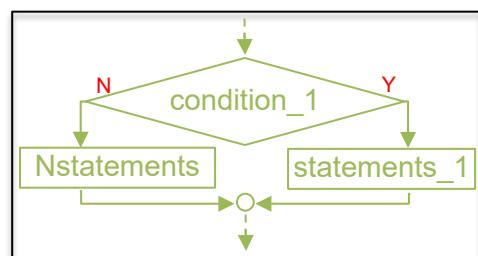
คำสั่ง if เป็นคำสั่งที่เราใช้กำหนดให้โปรแกรมตัดสินใจทำ หรือไม่ทำสิ่งใดสิ่งหนึ่ง โดยตรวจสอบเงื่อนไขที่กำหนดว่าเป็นจริง หรือเท็จ ถ้าเงื่อนไขที่กำหนดให้เป็นจริง (true) โปรแกรมจะทำงานที่ชุดคำสั่งที่อยู่ภายใต้คำสั่ง if แต่ถ้าเงื่อนไขที่กำหนดไม่เป็นเท็จ (false) โปรแกรมจะข้ามไปทำงานที่คำสั่งต่อไปทันที



ข้อควรรู้ : การกำหนดเงื่อนไขตามปกติ คือใช้นิพจน์มาทำการเปรียบเทียบค่ากันโดยใช้ Operator $>$, $<$, \geq , \leq , \neq เช่น $if (a>b)$ การดำเนินการเปรียบเทียบกันตามปกติจะได้ผลลัพธ์สองแบบ คือเท็จ(False) จะได้ค่าเป็น 0 หาก และจริง (True) จะได้ค่าเป็น 1

ข้อควรรู้ : เงื่อนไขที่นิพจน์ไม่ได้กำหนดตามปกติ เช่น $if (a = -10)$ ซึ่ง $a = -10$ เป็นการกำหนดค่าให้ตัวแปร a เป็น $-10 \rightarrow if (-10)$ 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 โดยตรวจสอบเงื่อนไขที่กำหนดว่า เป็นจริง หรือเท็จ ถ้าเงื่อนไขที่กำหนดให้เป็นจริง (true) โปรแกรมจะทำงานที่ชุดคำสั่งที่อยู่ภายใต้ if แต่ถ้าเงื่อนไขที่กำหนดไม่เป็นเท็จ (false) โปรแกรมจะทำงานที่ชุดคำสั่งที่อยู่ภายใต้คำสั่ง else

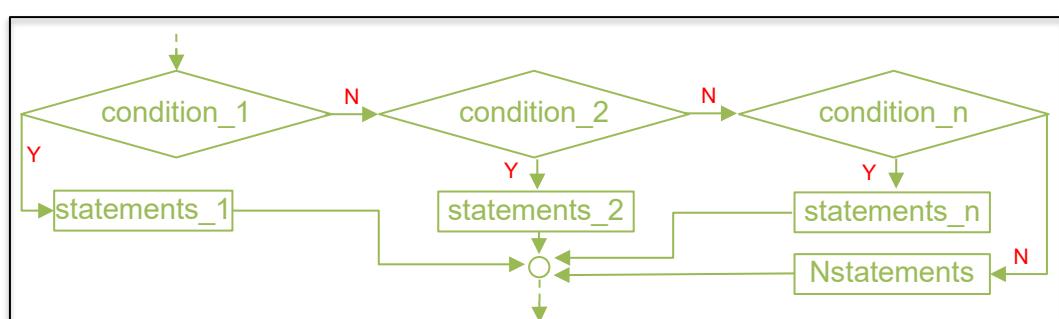
```
if (condition_1)
{
    statements_1;
}
else
{
    Nstatements;
}
```

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 และแต่ละทางเลือกจะมีการกำหนดเงื่อนไขของแต่ละทางเลือก ไว้ด้วย โดยโปรแกรมจะตรวจสอบเงื่อนไขทางเลือกใหม่มีเงื่อนไขเป็นจริง (true) ก็จะทำงานที่ชุดคำสั่งภายใต้ทางเลือกนั้น โดยไม่พิจารณาทางเลือกอื่นที่ไม่ได้ตรวจสอบอีก ในกรณีที่เงื่อนไขเป็นเท็จ (false) ให้ตรวจสอบเงื่อนไขต่อไป ในกรณีที่เงื่อนไขทั้งหมดเป็นเท็จ ให้โปรแกรมทำงานที่ชุดคำสั่ง else

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



ข้อสังเกต : เงื่อนไขที่ใช้เปรียบเทียบ หากใช้การเปรียบเทียบที่มากกว่าเดิม ให้ใช้เครื่องหมาย == ในการเปรียบเทียบ อย่างเช่นสนับสนุนเครื่องหมาย = (การกำหนดค่า) หากใช้เครื่องหมาย = ดังในตัวอย่างเมื่อโปรแกรมทำงานถึงส่วน if คอมpileอร์จะพิจารณาเงื่อนไข 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)
7     else
8 }
```

```
TRUE : x = 1
Process exited after 0.01769
seconds with return value 0
Press any key to continue . . .
```

```
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 }
```

```
TRUE : x = 1
if x == 0(false) print this
Process exited after 0.01736
seconds with return value 0
Press any key to continue . . .
```

จากตัวอย่างจะเห็นว่า x มีค่าเท่ากับ 1 จึงทำเงื่อนไขใน if ก็คือแสดงผล TRUE : x = 1 ออกมานะและโปรแกรมจะหลุดออกจาก if-else ทำคำสั่ง printf("\nif x == 0(false) print this.."); ต่อ ซึ่งคำสั่งนี้ไม่อยู่ใน if-else วิธีแก้ คือ การใส่ปีกภาคลุ่มคำสั่งหลัง 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 คือเลือกทางใดทางหนึ่งจากการทางเลือกที่มากกว่า 2 ทาง ในแต่ละทางเลือกจะมีการกำหนดเงื่อนไขของแต่ละทาง โดยตรวจสอบเงื่อนไขแต่ละทางเลือก หากพบว่าทางเลือกไหนมีเงื่อนไขเป็นจริง (true) ก็จะทำชุดคำสั่งภายในทางเลือกนั้น และไม่พิจารณาทางเลือกที่ยังไม่ได้ตรวจสอบอีก ในกรณีที่เงื่อนไขเป็นเท็จ (false) ให้ตรวจสอบเงื่อนไขถัดไป ในกรณีที่เงื่อนไขทั้งหมดเป็นเท็จ ให้โปรแกรมทำงานที่ชุดคำสั่ง default : สำหรับ flowchart จะเหมือนกับคำสั่งตัดสินใจแบบหลายทางเลือกด้วยคำสั่ง if...else if

โดยที่	condition	เป็นตัวแปร หรืออินพุตที่เป็นเงื่อนไขของคำสั่ง
	constant_n	เป็นตัวเลขที่เป็นค่าคงที่ใช้ตรวจสอบกับตัวแปร หรืออินพุตเงื่อนไข
	statements_n	เป็นชุดคำสั่งที่ต้องทำงานเมื่อตัวแปรหรืออินพุตที่ ก เป็นจริง
	Nstatements	เป็นชุดคำสั่งที่ต้องทำงานเมื่อตัวแปรหรืออินพุตทั้งหมดเป็นเท็จ

```
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 }
```

```
CASE A
CASE B
Process exited after 0.02122
seconds with return value 0
Press any key to continue . . .
```

ข้อสังเกต : ต้องใช้ค่าคงที่ในการแยกกรณีเท่านั้น จากรหัสที่ 10 และ 11 จะเห็นว่า 'A' คือการนำรหัสเอกสารก์ของ A ซึ่งมีค่าเท่ากับ 65 มาเป็นเงื่อนไขในการเช็ค และ 'B' คือการนำรหัสเอกสารก์ของ B ซึ่งมีค่าเท่ากับ 66 มาเป็นเงื่อนไขในการเช็ค

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

จากตัวอย่าง การเขียนโปรแกรมตัดเกรด และกำหนดให้คะแนนมีค่าเป็น 75 เมื่อถึงส่วนเช็คเงื่อนไข จะทำการเช็คเงื่อนไขใน if ก่อน (คะแนนมากกว่าหรือเท่ากับ 80 หรือไม่) ปรากฏว่าเป็นเท็จ จึงเช็คเงื่อนไข else if ต่อไป คือคะแนนมากกว่า หรือเท่ากับ 75 แต่น้อยกว่า 80 หรือไม่ ทำให้เงื่อนไขนี้เป็นจริง และทำการคำสั่งหลังเงื่อนไขนั้นเพียง 1 statement (แสดง B+)

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

```

1 #include<stdio.h>
2 main()
3 {
4     int score = 75;           B+
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

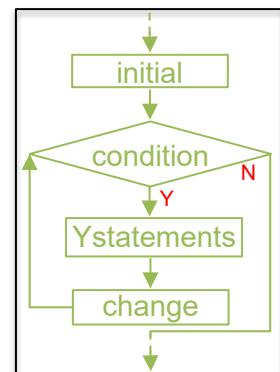
```

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

```

คำสั่ง for เป็นคำสั่งวนซ้ำการทำงานเดิม ๆ ด้วยจำนวนรอบที่แน่นอน โดยที่โปรแกรมจะตรวจสอบเงื่อนไขก่อนการทำงานทุกรอบ ถ้าเงื่อนไขเป็นจริงให้ทำงานที่ชุดคำสั่งภายในลูปต่อไป เมื่อทำงานเสร็จจะเพิ่ม หรือลดค่าตัวแปร และตรวจสอบเงื่อนไขใหม่อีกครั้ง ถ้าเงื่อนไขเป็นเท็จให้โปรแกรมออกจากลูปการทำงานไปทำงานที่คำสั่งถัดไปทันที

โดยที่	initial	เป็นส่วนกำหนดค่าเริ่มต้นของตัวแปรที่ใช้กำหนดเงื่อนไข
	condition	เป็นส่วนกำหนดเงื่อนไขในการวนลูป สามารถใส่หลายเงื่อนไขได้ โดยใช้ตัวดำเนินการแบบต่างๆ เช่น ช่วงกำหนดเงื่อนไขให้เป็นจริงหรือเท็จ
	change	เป็นส่วนการเปลี่ยนแปลงของตัวแปรหลังทำงานจนจบในแต่ละรอบ
	Ystatements	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 : เงื่อนไขสามารถกำหนดได้หลายแบบแต่ผลลัพธ์ที่ออกมามาได้เหมือนกัน สำหรับในส่วนของ initial และ change ไม่จำเป็นต้องมีเสมอไป หรือสามารถกำหนดให้มีมากกว่า 1 ก็ได้ ในส่วนของ condition ก็สามารถมีมากกว่า 1 ได้เช่นกัน โดยอาศัยตัวดำเนินการต่าง ๆ เช่นตามรูปด้านขวา แต่ผู้อ่านควรเขียนให้ผู้อื่นเข้าใจโปรแกรมได้ง่าย

```

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;           012
5     for(i=0, x=2; i<2 || i==2; i++) {
6         printf("%d", i);
7     }
8 }

```

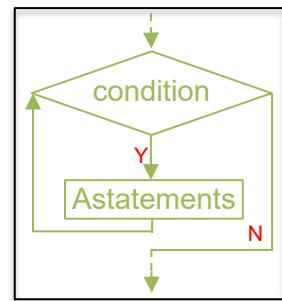
ข้อควรระวัง : บางคอมไพล์เตอร์เก่า ๆ บางตัวไม่สามารถประกาศตัวแปรในส่วน initial ได้เหมือนในตัวอย่างด้านซ้าย แต่ถ้าประกาศได้ตัวแปรที่ถูกประกาศนั้นจะเป็นตัวแปรประเภท Local อยู่ภายใต้ลูป for ถ้าออกจากลูปตัวแปร i จะถูกลบออกจากหน่วยความจำเมื่อมันไม่ได้ประกาศไว้

วนลูปการทำงานเมื่อเงื่อนไขเป็นจริงด้วยคำสั่ง while

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

คำสั่ง while เป็นคำสั่งวนซ้ำการทำงานเดิมๆ ของโปรแกรม โดยโปรแกรมตรวจสอบเงื่อนไขก่อนการทำงานทุกครั้ง ถ้าเงื่อนไขที่กำหนดเป็นจริง โปรแกรมจะทำงานที่ชุดคำสั่งภายในลูป เมื่อโปรแกรมทำงานเสร็จ จะตรวจสอบเงื่อนไขใหม่อีกครั้ง ถ้าเงื่อนไขที่กำหนดเป็นเท็จ โปรแกรมจะออกจากลูปการทำงานไปทำงานที่คำสั่งถัดไปทันที

โดยที่ condition เป็นเงื่อนไขที่กำหนดให้ตรวจสอบก่อนทำงานภายในลูปทุกครั้ง
Astatements เป็นชุดคำสั่งที่ต้องทำเมื่อเงื่อนไขเป็นจริง (หลังตรวจสอบเงื่อนไข)

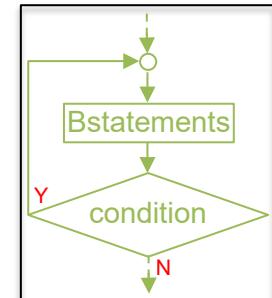


วนลูปการทำงานอย่างน้อย 1 ครั้งด้วยคำสั่ง do...while

```
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 : ในกรณีที่จำนวนคำสั่งภายในลูป for while do (ใน do-while) มีเพียงคำสั่งเดียว ไม่จำเป็นต้องใส่เครื่องหมาย {} ก็ได้ แต่แนะนำให้ใส่เครื่องหมายปีกกา {} ทุกครั้ง

คำสั่ง break และ continue

<pre>1 #include<stdio.h> 2 3 main() 4 { 5 for(int i=0; i<3; i++) { 6 for(int j=0; j<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> <hr/> <pre>Process exited after 0.0201 seconds with return value 0 Press any key to continue .</pre>
---	--

จากรูปด้านบนเมื่อคอมไพล์แล้วทำงานถึงคำสั่ง break คอมไпал์จะทำการออกจากคำสั่งวนลูปปัจจุบันที่กำลังทำงานอยู่ (เลิกทำคำสั่งวนลูป for ที่มีตัวแปร j กำหนด ซึ่งโดยปกติการที่ลูป j จะจบได้ j ต้องมีค่า ≥ 9 แต่ j มีค่าเท่ากับ 2 ตรงเงื่อนไข break ที่กำหนดไว้จึงออก

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

<pre>1 #include<stdio.h> 2 3 main() 4 { 5 for(int i=0; i<3; i++) { 6 for(int j=0; j<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> <hr/> <pre>Process exited after 0.0017 seconds with return value 0 Press any key to continue .</pre>
--	--

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 }
```

```
for(int x=0; x<10; x++) { } = for(int x=0; x<10; x++) { } printf("Caution!");
```

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

```
= while(x<10) { } = while(x<10) { } printf("Caution!");
```

ข้อควรระวัง : การกำหนดเงื่อนไขของลูป อาจทำให้เกิดลูปนั้นต่อได้ คือลูปที่วนไม่รู้จบเนื่องจากไม่ว่าลูปจะผ่านไปกี่รอบก็จะไม่มีวันผิดเงื่อนไข ที่กำหนด ไว้เหมือนตัวอย่างทางด้านซ้าย 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 }
```

```
123456
Process exited after 0.02169 seconds with return value 0
Press any key to continue . . .
```

```
= while(i<6)
{   printf("%d", ++i); } = while(i<6)
{   i++; printf("%d", i); } = i = 1;
do
{   printf("%d", i++); } while(i<=6);
```

อธิบายลูป for อย่างละเอียด

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

i = 0 loop#1	<code>for(i=0; i<6; i++)</code> { printf("%d", i); }	→	<code>for(i=0; i<6; i++)</code> { printf("%d", i); }	→	<code>for(i=0; i<6; i++)</code> { printf("%d", i); }
i = 1 loop#2	<code>for(i=0; i<6; i++)</code> { printf("%d", i); }	→	<code>for(i=0; i<6; i++)</code> { printf("%d", i); }	→	<code>for(i=0; i<6; i++)</code> { printf("%d", i); }
⋮	⋮	⋮	⋮	⋮	⋮
i = 5 loop#6	<code>for(i=0; i<6; i++)</code> { printf("%d", i); }	→	<code>for(i=0; i<6; i++)</code> { printf("%d", i); }	→	<code>for(i=0; i<6; i++)</code> { printf("%d", i); }
i = 6	<code>for(i=0; i<6; i++)</code> { printf("%d", i); }	จะทำอย่างนี้ข้าไปข้างมาจนกว่าจะผิดเงื่อนไข (ลูปรอบแรก i=0 ลูปรอบ 2 i=1 ลูปรอบ6 i=5 และเป็นลูปรอบสุดท้ายเนื่องจากก่อนจะเข้าลูปรอบที่ 7 จะต้องเช็คเงื่อนไขก่อนเข้า และ i มีค่าเป็น 6 ซึ่งจะผิดเงื่อนไข ผลคือจบลูป			

อาร์เรย์ (Array)

ตัวแปรอาร์เรย์ เปรียบเสมือนการนำตัวแปรมาเรียงต่อกันหลอย ๆ ตัว โดยที่ทุกตัวมีชนิดข้อมูลแบบเดียวกัน มีชื่อ ตัวแปรเดียวกัน อาร์เรย์สามารถอ้างถึงข้อมูลแต่ละตัวที่เรียงต่อกันเป็นลำดับการโดยใช้ตำแหน่งได้ ซึ่งตำแหน่งของข้อมูลดังกล่าวจะเรียกว่า อินเด็กซ์ (Index)

ตัวแปรอาร์เรย์ที่มีให้ในภาษา C นั้นสามารถแยกได้ 2 แบบคือ ตัวแปรอาร์เรย์ 1 มิติ และตัวแปรอาร์เรย์หลายมิติ (มากกว่า 1) ซึ่งในที่นี้จะยกล่าวถึงตัวแปรอาร์เรย์ 1 มิติ ตัวแปรอาร์เรย์ 2 มิติ และตัวแปรอาร์เรย์ 3 มิติเท่านั้น

ตัวแปรอาร์เรย์ 1 มิติ (One Dimension Array)

เปรียบได้กับการนำตัวแปรมาเรียงต่อกันหลอย ๆ ตัวในลักษณะเป็นแถวของข้อมูล ซึ่งสามารถจัดล่องตัวอย่างตัวแปรอาร์เรย์ 1 มิติ ชื่อตัวแปร intEx1 เป็นตัวแปรชนิดจำนวนเต็มที่สามารถเก็บข้อมูลจำนวนเต็มได้ 6 ตัว ยกตัวอย่างดังนี้

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

ตัวแปรตัวแรก	ตำแหน่งอินเด็กซ์ที่ 0 มีค่าเท่ากับ 5 ซึ่งสามารถเขียนได้เป็น intEx1[0] = 5
ตัวแปรตัวที่ 2	ตำแหน่งอินเด็กซ์ที่ 1 มีค่าเท่ากับ 8 ซึ่งสามารถเขียนได้เป็น intEx1[1] = 8
ตัวแปรตัวที่ 3	ตำแหน่งอินเด็กซ์ที่ 2 มีค่าเท่ากับ 1 ซึ่งสามารถเขียนได้เป็น intEx1[2] = 1
ตัวแปรตัวที่ 4	ตำแหน่งอินเด็กซ์ที่ 3 มีค่าเท่ากับ 3 ซึ่งสามารถเขียนได้เป็น intEx1[3] = 3
ตัวแปรตัวที่ 5	ตำแหน่งอินเด็กซ์ที่ 4 มีค่าเท่ากับ 9 ซึ่งสามารถเขียนได้เป็น intEx1[4] = 9
ตัวแปรตัวสุดท้าย	ตำแหน่งอินเด็กซ์ที่ 5 มีค่าเท่ากับ 2 ซึ่งสามารถเขียนได้เป็น intEx1[5] = 2

ตัวแปรอาร์เรย์ 2 มิติ (2-Dimension Array)

เปรียบได้กับการนำตัวแปรมาเรียงต่อกันหลอย ๆ ตัวในลักษณะของตารางข้อมูล ซึ่งสามารถจัดล่องตัวอย่างตัวแปรอาร์เรย์ 2 มิติ ชื่อตัวแปร 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

ตัวแปรตัวที่ 1 คอลัมน์ที่ 1 มีค่าเท่ากับ 3 ซึ่งสามารถเขียนได้เป็น intEx2[0][0] = 3
ตัวแปรตัวที่ 1 คอลัมน์ที่ 2 มีค่าเท่ากับ 7 ซึ่งสามารถเขียนได้เป็น intEx2[0][1] = 7
ตัวแปรตัวที่ 1 คอลัมน์ที่ 5 มีค่าเท่ากับ 1 ซึ่งสามารถเขียนได้เป็น intEx2[0][4] = 1
ตัวแปรตัวที่ 2 คอลัมน์ที่ 1 มีค่าเท่ากับ 7 ซึ่งสามารถเขียนได้เป็น intEx2[1][0] = 7
ตัวแปรตัวที่ 2 คอลัมน์ที่ 2 มีค่าเท่ากับ 1 ซึ่งสามารถเขียนได้เป็น intEx2[1][1] = 1
ตัวแปรตัวที่ 2 คอลัมน์ที่ 4 มีค่าเท่ากับ 2 ซึ่งสามารถเขียนได้เป็น intEx2[1][3] = 2
ตัวแปรตัวที่ 3 คอลัมน์ที่ 1 มีค่าเท่ากับ 4 ซึ่งสามารถเขียนได้เป็น intEx2[2][0] = 4
ตัวแปรตัวที่ 3 คอลัมน์ที่ 2 มีค่าเท่ากับ 2 ซึ่งสามารถเขียนได้เป็น intEx2[2][1] = 2
ตัวแปรตัวที่ 3 คอลัมน์ที่ 5 มีค่าเท่ากับ 3 ซึ่งสามารถเขียนได้เป็น intEx2[2][4] = 3

ตัวแปรอาร์เรย์ 3 มิติ (3-Dimension Array)

เปรียบได้กับการนำตัวแปรมาเรียงต่อกันหลอย ๆ ตัวในลักษณะของกล่องข้อมูล ซึ่งสามารถจัดล่องตัวอย่างตัวแปรอาร์เรย์ 3 มิติ ชื่อตัวแปร intEx3 เป็นตัวแปรชนิดจำนวนเต็มที่สามารถเก็บข้อมูลจำนวนเต็มได้ 24 ตัว ยกตัวอย่างดังนี้



จากรูปด้านบนจะเห็นได้ว่า ตัวแปรอาร์เรย์ที่มีขนาด 2 บล็อก 3 และ 4 คอลัมน์ สามารถเก็บข้อมูลได้ 24 ตัว ตัวอย่างเช่น ตัวแปรบล็อกที่ 1 伟大ที่ 1 คอลัมน์ที่ 1 มีค่าเท่ากับ 9 ซึ่งสามารถเขียนได้เป็น `intEx3[0][0][0] = 9` ตัวแปรบล็อกที่ 1 伟大ที่ 1 คอลัมน์ที่ 2 มีค่าเท่ากับ 2 ซึ่งสามารถเขียนได้เป็น `intEx3[0][0][1] = 2` ตัวแปรบล็อกที่ 1 伟大ที่ 1 คอลัมน์ที่ 3 มีค่าเท่ากับ 1 ซึ่งสามารถเขียนได้เป็น `intEx3[0][0][2] = 1` ตัวแปรบล็อกที่ 1 伟大ที่ 2 คอลัมน์ที่ 1 มีค่าเท่ากับ 8 ซึ่งสามารถเขียนได้เป็น `intEx3[0][1][0] = 8` ตัวแปรบล็อกที่ 1 伟大ที่ 2 คอลัมน์ที่ 2 มีค่าเท่ากับ 5 ซึ่งสามารถเขียนได้เป็น `intEx3[0][1][1] = 5` ตัวแปรบล็อกที่ 1 伟大ที่ 3 คอลัมน์ที่ 4 มีค่าเท่ากับ 7 ซึ่งสามารถเขียนได้เป็น `intEx3[0][2][3] = 7` ตัวแปรบล็อกที่ 2 伟大ที่ 1 คอลัมน์ที่ 1 มีค่าเท่ากับ 7 ซึ่งสามารถเขียนได้เป็น `intEx3[1][0][0] = 7` ตัวแปรบล็อกที่ 2 伟大ที่ 1 คอลัมน์ที่ 2 มีค่าเท่ากับ 1 ซึ่งสามารถเขียนได้เป็น `intEx3[1][0][1] = 1` ตัวแปรบล็อกที่ 2 伟大ที่ 1 คอลัมน์ที่ 3 มีค่าเท่ากับ 4 ซึ่งสามารถเขียนได้เป็น `intEx3[1][0][2] = 4` ตัวแปรบล็อกที่ 2 伟大ที่ 2 คอลัมน์ที่ 1 มีค่าเท่ากับ 3 ซึ่งสามารถเขียนได้เป็น `intEx3[1][1][0] = 3` ตัวแปรบล็อกที่ 2 伟大ที่ 2 คอลัมน์ที่ 2 มีค่าเท่ากับ 7 ซึ่งสามารถเขียนได้เป็น `intEx3[1][1][1] = 7` ตัวแปรบล็อกที่ 2 伟大ที่ 3 คอลัมน์ที่ 4 มีค่าเท่ากับ 2 ซึ่งสามารถเขียนได้เป็น `intEx3[1][2][3] = 2`

การประกาศตัวแปรอาร์เรย์

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

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

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 เราต้องกำหนดตำแหน่งอินเด็กซ์เพื่อบุตำแหน่งของตัวแปรที่จะกำหนดค่า 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 ซึ่งเราอาจจะกำหนดค่าภายในเดียวกัน ซึ่งมีตัวอย่างดังนี้

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

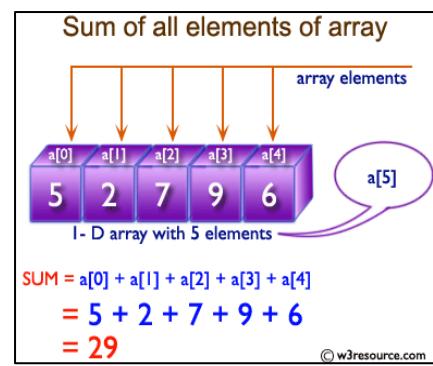
	6			
--	---	--	--	--

```
int intNum[5];  
char chPassword[4];  
float fPrice[3][2];  
  
intNum[1] = 6;  
chPassword[0] = 'Z';  
fPrice[2][1] = 0.08;
```

การอ้างถึงข้อมูลในตัวแปรอาร์เรย์

เมื่อมีการกำหนดค่าให้กับตัวแปรแล้ว ต่อไปก็ต้องรู้วิธีการอ่านข้อมูลจากตัวแปร โดยการระบุตำแหน่งข้อมูลที่ต้องการลงไป หรือที่เรียกว่า อินเด็กซ์ ซึ่งมีด้วยการใช้งานดังนี้

```
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 โดยมีค่ามาจากการผลคูณระหว่างค่าในตัวแปรอาร์เรย์ตำแหน่งที่ 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 โคลด์ในบรรทัดที่ 8 – 9 จะเป็นการให้ผู้ใช้กรอกว่าจะเก็บค่ากี่ค่า เพื่อใช้สร้างอาร์เรย์ที่มี index ขนาดเท่ากับข้อมูลที่ใส่บรรทัดที่ 14 – 16 เป็นการรับค่าแต่ละค่าเข้ามาเก็บในอาร์เรย์แต่ละอินเด็กซ์ตั้งแต่ a[0] a[1] จนถึง a[n-1] และระหว่างนั้นถ้ารับค่าเข้ามาแล้วให้นำไปบวกเข้าไปเก็บไว้ในตัวแปร sum โดยใช้ค่าสั่ง 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....

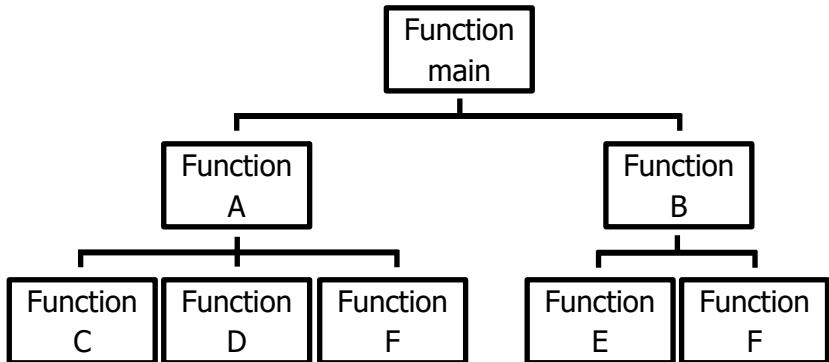
ข้อสังเกต : array มักจะใช้งานร่วมกับ for เพื่อให้ใช้งานได้ง่ายขึ้น เช่น แทนที่จะรับค่าให้อาร์เรย์ทีละตัว scanf("%d", &a[0]) scanf("%d", &a[1]) ... scanf("%d", &a[4]) เราจะรับค่าโดยใช้ for ช่วยเหมือนด้วยการอ่านตัวอย่างด้านบน หรือจะใช้ช่วยในการแสดงผลแทนที่จะต้อง printf("%d", a[index]) ทุก index

ข้อควรรู้ : array จะเริ่มต้นที่อินเด็กซ์เป็น 0 และตัวสุดท้ายเป็นอินเด็กซ์ที่ n-1 (ให้ n เป็นจำนวนอินเด็กซ์ที่ประกาศไว้)

ฟังก์ชัน (Function)

คือชุดคำสั่งที่รวมกันเป็นโปรแกรมย่อย ๆ ภายในเครื่องหมาย {} ถูกสร้างขึ้นมาเพื่อทำงานอย่างใดอย่างหนึ่ง และมีการตั้งชื่อของฟังก์ชันเพื่อให้สะท้อนต่อการเรียกใช้งานตามกฎการตั้งชื่อ โดยมีรูปแบบการเรียกใช้งานแตกต่างกัน คือจะมีการรับ หรือไม่รับข้อมูลจากโปรแกรมที่เรียกใช้งาน และจะมีการส่ง หรือไม่ส่งค่าข้อมูลออกจากฟังก์ชัน ซึ่งรูปแบบการใช้งานต่าง ๆ ของฟังก์ชันจะขึ้นอยู่กับหน้าที่ และป้าหมายการทำงานของฟังก์ชันนั้น 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 แต่ผู้เขียนไม่แนะนำให้แบ่งโปรแกรมย่อย ๆ เพื่อเรียกใช้งานข้อนกันมากเกินไป เนื่องจากจะทำให้โปรแกรมเกิดความซับซ้อน แก้ไขและพัฒนาได้ยาก

ฟังก์ชันที่มาตรฐานที่ติดมากับภาษา C (Standard Library Function)

ฟังก์ชันมาตรฐาน คือฟังก์ชันที่ผู้ใช้สามารถเรียกใช้งานจากไลบรารีของภาษาซึ่งได้ทันที เช่น ฟังก์ชันทางคณิตศาสตร์ ฟังก์ชันเกี่ยวกับสตริง ฟังก์ชันเกี่ยวกับการเปรียบเทียบ ฟังก์ชันเกี่ยวกับการแสดงผล และฟังก์ชันเกี่ยวกับวันเวลา เป็นต้น ซึ่งฟังก์ชันมาตรฐานนี้เป็นฟังก์ชันที่มีติดมาให้ เก็บไว้ใน header file ภาษา C (เก็บไว้ในแฟ้มที่มีนามสกุล *.h ต่าง ๆ) เมื่อต้องการใช้ฟังก์ชันใด จะต้องรู้ว่าฟังก์ชันนั้นอยู่ใน header file ใด จากนั้นจึงค่อยใช้คำสั่ง #include เช่น ในส่วนตอนต้นของโปรแกรม จึงจะสามารถใช้ฟังก์ชันที่ต้องการได้ เช่น ฟังก์ชัน pow(x, y) จะถูกเก็บไว้ใน header file ที่ชื่อว่า math.h ดังนั้นจึงต้องใช้คำสั่ง #include แทรกอยู่ในส่วนตอนต้นของโปรแกรมหนึ่งอีกฟังก์ชัน main() จึงจะสามารถเรียกใช้ฟังก์ชัน pow(x, y) มาใช้งานภายในโปรแกรมนี้ได้ โดยฟังก์ชันมาตรฐานเป็นฟังก์ชันที่บริษัทผู้ผลิต 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 logฐานสิบ 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 หรือจนกระทั่งหมดสตริง โดยจะเปรียบเทียบ null character ด้วย
strncpy(dest, src, n)	ทำการก็อปปี้สตริงจาก src ไปยัง dest ถ้า src เป็นจำนวน n อักษร
strcmp(str1, str2)	เปรียบเทียบข้อมูลชนิดสตริง str1 เทียบตัวแปร str2 ว่ามีค่ามากกว่า น้อยกว่า หรือเท่ากันอย่างใดอย่างหนึ่ง ซึ่งการเปรียบเทียบสตริงจะใช้ค่ารหัส ASCII เปรียบเทียบที่ลับตัวอักษร
strcmpl(str1, str2)	เปรียบเทียบข้อมูลชนิดสตริง str1 เทียบตัวแปร str2 ว่ามีค่ามากกว่า น้อยกว่า หรือเท่ากันอย่างใดอย่างหนึ่ง โดยไม่สนใจเป็นตัวอักษรพิมพ์เล็ก หรือตัวอักษรพิมพ์ใหญ่ซึ่งการเปรียบเทียบสตริงจะใช้ค่ารหัส ASCII เปรียบเทียบที่ลับตัวอักษร
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

cctype.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)	ตรวจสอบคุณค่าใน ch เป็นค่าแอสกี้โค้ดของช่องว่างหรือไม่ ถ้าเป็นจริง จะให้ค่าไม่เท่ากับศูนย์ ถ้าเป็นเท็จ จะให้ค่าเท่ากับศูนย์
isxdigit(ch)	ตรวจสอบว่าค่าใน ch เป็นค่าแอสกี้โค้ดของเลขฐานสิบหก (0-9, A-F หรือ a-f) หรือไม่ ถ้าเป็นจริง จะให้ค่าไม่เท่ากับศูนย์ ถ้าเป็นเท็จ 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 อักขระ ใน模式ให้เห็นในการรับข้อมูล
putchar()	รับข้อมูล 1 อักขระออกทางจอภาพ
clrscr()	ลบจอภาพ

ฟังก์ชันที่สร้างขึ้นเอง (User-Defined Functions)

เป็นฟังก์ชันที่ผู้ใช้งานเขียนโดยสร้างฟังก์ชันขึ้นมาใช้ ဝิธามรูปแบบการสร้างฟังก์ชันของภาษา C เพื่อให้ทำงานอย่างโดยย่างหนึ่ง 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 และเมื่อฟังก์ชันทำงานเสร็จจะไม่มีการคืนค่าข้อมูลใด ๆ กลับไปให้ฟังก์ชันที่เรียกใช้งาน ซึ่งมีรูปแบบการสร้างฟังก์ชันดังนี้

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

จากรูปแบบฟังก์ชันดังกล่าว จึงสามารถสร้างฟังก์ชันใช้งานได้ดังตัวอย่างด้านล่าง

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

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

เมื่อได้เขียนฟังก์ชันขึ้นมาเพื่อใช้งานเองแล้ว จะยังไม่สามารถเรียกใช้งานฟังก์ชันดังกล่าว ได้จนกว่าจะมีการประกาศโปรโตไทป์ของฟังก์ชันก่อนฟังก์ชัน main เพื่อให้คอมไพล์เลอร์รู้จักฟังก์ชัน จากตัวอย่างโค้ดจะประกาศโปรโตไทป์ได้ดังนี้ **void printName(void);** ซึ่งการประกาศโปรโตไทป์จะมีรูปแบบดังนี้

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

หลังจากประกาศโปรโตไทป์ของฟังก์ชันเพื่อให้ฟังก์ชัน main รู้จักกับฟังก์ชันที่สร้างขึ้นแล้ว สามารถเรียกใช้งานฟังก์ชันดังกล่าวได้จากฟังก์ชัน main หรือจากฟังก์ชันอื่น ๆ ภายในโปรแกรมได้ จากตัวอย่างโค้ดสามารถเรียกใช้งานตัวฟังก์ชันได้ดังนี้ **printName();** โดยการเรียกใช้งานฟังก์ชันจะมีรูปแบบดังนี้

```
functionName();
```

จากรูปแบบการประกาศโปรโตไทป์ของฟังก์ชัน และการเรียกใช้งานฟังก์ชัน จึงสามารถสร้างฟังก์ชัน ใช้งานและเรียกใช้งานได้ตามตัวอย่างง่าย ๆ ทางด้านขวาดังนี้

ข้อควรรู้ : เนื่องจากโปรแกรมทำงานจากซ้ายไปขวา บนลงล่าง เมื่อรันโปรแกรม หากในฟังก์ชัน main เจอฟังก์ชันแปลก ๆ ที่คอมไพล์ร่วมกันจะมีการแจ้ง error ออกมา (คล้าย ๆ กับการใช้ตัวแปรแต่ยังไม่ได้ประกาศให้ตัวแปร) จึงแก้ตัวยกประกาศให้คอมไпал์ร์รู้ว่ามีฟังก์ชันแปลก ๆ นือญ โดยการประกาศโปรโตไทป์ ซึ่งเป็นการแสดงส่วนประกอบของฟังก์ชันว่ามีการคืนค่ากลับหรือไม่ 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 n ที่ใช้รับข้อมูลจากฟังก์ชันที่เรียกใช้งาน
 และต้องมีชนิดข้อมูลตรงกันกับ typeParameter_n

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

จากรูปแบบฟังก์ชันดังกล่าว จึงสามารถสร้างฟังก์ชันใช้งานได้ดังตัวอย่างด้านขวา และการประกาศໂປຣໂടໄທปีของฟังก์ชันดังกล่าวสามารถทำได้โดย **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_1);
```

หลังจากการประกาศໂປຣໂടໄທปีจะสามารถเรียกใช้งานตัวฟังก์ชันได้ดังนี้ **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 และสามารถประกาศໂປຣໂটໄທปีได้ดังนี้ **int calculate2(void);** โดยการประกาศໂປຣໂটໄທปีจะมีรูปแบบตามด้านล่าง

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

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

หลังจากการประกาศໂປຣໂಟໄທปีจะสามารถเรียกใช้งานตัวฟังก์ชัน โดยเก็บผลการทำงานของฟังก์ชันไว้ที่ตัวแปร ans (สมมติว่าประกาศตัวแปร ans เป็นประเภท int เรียบร้อยแล้ว) ดังนี้ **ans = calculate2();** โดยการเรียกใช้งานฟังก์ชันจะมีรูปแบบดังนี้

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

ฟังก์ชันที่มีการคืนค่ากลับ และมีการรับค่าพารามิเตอร์ (Function Return Value with Parameters)

เป็นฟังก์ชันที่สร้างขึ้นโดยมีการรับค่าข้อมูล (พารามิเตอร์) จากฟังก์ชันที่เรียกใช้งาน และเมื่อฟังก์ชันทำงานเสร็จจะมีการคืนค่าข้อมูลกลับไปให้ฟังก์ชันที่เรียกใช้งาน ซึ่งมีรูปแบบการสร้างฟังก์ชันดังนี้

กำหนดให้**typeReturn****typeParameter_n**

parameterName_n

varReturn

variable

เป็นชนิดข้อมูลที่ต้องการคืนค่ากลับให้ฟังก์ชันที่เรียกใช้งาน

เป็นชนิดข้อมูลที่ต้องการรับจากฟังก์ชันที่เรียกใช้งาน

เป็นชื่อพารามิเตอร์ หรือตัวแปรตัวที่ *n* ที่ใช้รับข้อมูลจากฟังก์ชันที่เรียกใช้งาน

และต้องมีชนิดข้อมูลตรงกันกับ typeParameter_n

เป็นตัวแปร หรือค่าที่ต้องการคืนค่ากลับให้ฟังก์ชันที่เรียก

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 จึงสามารถสร้างฟังก์ชันໃใช้งานได้ดังต่อไปนี้
และสามารถประกาศໂປຣໂടໄທປີໄດ້ດັ່ງນີ້ **float circleArea(float);**
ໂດຍການປະກາສົມໂປຣໂടໄທປີຈະມີຮູບແບບຕາມດ້ານລ່າງ

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

หลังจากการປະກາສົມໂປຣໂടໄທປີຈະສາມາດເຮັດວຽກໃຫ້ງານດ້ານຝຶກໜັກ ໂດຍເກັບຜົນການທ່ານຂອງຝຶກໜັກໄວ້ທີ່ດ້ານແປຣ
area (ສົມດີມີການປະກາສົມດ້ານແປຣ area ເປັນປະເກດ float ເຮັດວຽກແລ້ວ) ດັ່ງນີ້ **area = circleArea(radius);** ຈຶ່ງ
ການເຮັດວຽກຝຶກໜັກຈະມີຮູບແບບຕາມດ້ານລ່າງ

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

ข้อควรรู้ : ໃນການຮັບສົ່ງຂອ່ມງວດຮ່ວມມືກ Argument ກັບ Parameter
ດ້ານນີ້ມີຂໍ້ມູນຕາມດ້ານລ່າງ
ໃນຝຶກໜັກ ແລ້ວ Parameter ຕື່ມາດ້ວຍສົ່ງຂອ່ມງວດໄປຢັ້ງຝຶກໜັກ



ໂປຣແກນນີ້ເປັນການເປົ້າມີຄ່າ 2 ຄ່າ ຮະຫວ່າງ 8 ທີ່ເກັບອຸ່ນໃນດ້ານແປຣ a ແລ້ວ 5 ທີ່ເກັບອຸ່ນໃນດ້ານແປຣ b ໂດຍການສົ່ງ
ຄ່າຜ່ານພາຣາມີເຕືອນໄງ້ num1 ແລ້ວ num2 ເຂົ້າໄປໃນຝຶກໜັກ ໃຫ້ຄໍານາວນອອກມາແລະແສດງຜົນດຳເນີນໃຫ້ທີ່ກຳນົດໄວ້ ໂດຍໃນ
ບຣທັດທີ່ 3 ເປັນການປະກາສົມໂປຣໂടໄທປີ ແລ້ວບຣທັດທີ່ 12-17 ເປັນຝຶກໜັກທີ່ຮັບຄ່າພາຣາມີເຕືອນໄງ້ເປັນ int 2 ຕ້າ ແລ້ວໄມ່ມີ
ການສົ່ງຄ່າໄດ້ ຈຶ່ງອອກມາຈາກຝຶກໜັກ (void) ເມື່ອເຮັດວຽກໃຫ້ຝຶກໜັກ maxData

<pre>1 #include<stdio.h> 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 >= num2) printf("Data max is %d", num1); 16 else printf("Data max is %d", num2); 17 }</pre>	
---	--

ຈາກດ້ານນີ້ສາມາດເຂົ້າມີໂປຣໂടໄທປີແບບດ້ານນີ້ໄດ້ໂດຍໃນການປະກາສົມໂປຣໂടໄທປີ

```

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 จึงต้องประกาศໂປຣໂടີໄທປີເພື່ອໃຫ້ຄວາມໄພເລອຮົງຈັກວ່າ ມີຝັ້ງຂັ້ນ testReturn ອູ່ຢູ່ໃນໂປຣແກຣມ ລາກໄມ່ປະກາສ ທີ່ຈະເຊື່ອການເຮັດວຽກໃຫ້ ເພື່ອການເຮັດວຽກໃຫ້ ພັດທະນາການຈຳກັດກຳກັບການໃຫ້ຕົວແປຣ ແຕ່ໄມ່ໄດ້ປະກາສວ່າຈະໃຫ້..)

ຝັ້ງຂັ້ນ testReturn ທີ່ສ່ວັງຈະເປັນຝັ້ງຂັ້ນທີ່ຮັບຄ່າຕົວເລີນ int ໄປ 1 ຕ້າ (1 ພາຣາມິເຕେຣີ) ໂດຍໃນຝັ້ງຂັ້ນຈະເປັນການ ນໍາຕົວເລີນໄປເຂົ້າເຫັນວ່າເລີນນີ້ ເປັນເລີນຈຳນວນເຕີມບວກ (Positive) ເປັນເລີນຈຳນວນເຕີມລົມ (Negative) ຮີ້ວີ້ເປັນ ເລີນຈຳນວນເຕີມສຸນຍົງ (Zero) ລາຍລະອຽດກ່ອນການເຮັດວຽກໃຫ້ແລ້ວ ແລ້ວກ່ອນການສົ່ງຄ່າກັບລັບອອກມາເປັນອັກຂະຮັດ ແກ່ກຂອງຈຳນວນເຕີມ (P, N, Z)

ໃນຝັ້ງຂັ້ນທີ່ 7 ເປັນການນຳຄ່າ 112 (ASCII ຂອງ p) ທີ່ໄດ້ຈຳການສ່າງກັບຄ່າກັບລັບຂອງ testReturn(9) ກໍາທັນດລັງຕົວແປຣ test ຈາກນັ້ນນໍາຕົວແປຣ test ໄປແສດງຜລອອກທາງໜ້າຈາກໃນບຣ້າທີ່ 10

ໃນບຣ້າທີ່ 8 ເປັນການເຮັດວຽກຝັ້ງຂັ້ນເພື່ອນຳຄ່າທີ່ຮີ້ເທີຣົນອອກມາໄປແສດງຜລ ໂດຍຕຽບໄມ່ເກັບຜ່ານຕົວແປຣ ໃຊ້
ໃນບຣ້າທີ່ 13 ເປັນການເຮັດວຽກຝັ້ງຂັ້ນເພື່ອນຳຄ່າທີ່ຮີ້ເທີຣົນອອກມາຂຶ້ນກີ້ວ່າ 110 (ASCII ຂອງ n) ໄປແສດງຜລໄດ້ຕຽບໄມ່ເກັບ
ຜ່ານຕົວແປຣ ໃຊ້

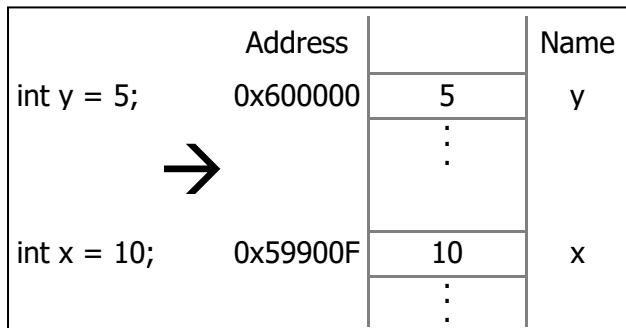
```

1 #include<stdio.h>
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 > 0) {
17         printf("print in testReturn - case if (%d)\n", num);
18         return 'p';
19     }
20     else if(num < 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 }

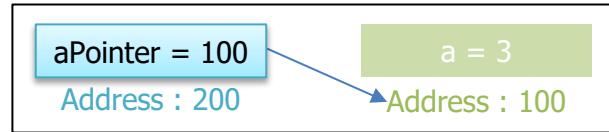
```

พอยเตอร์ (Pointer)

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



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 : การสร้างตัวแปรพอยน์เตอร์ด้องใช้ * 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 สามารถยกตัวอย่างໂຄด้อย่างง่ายได้ตามตัวอย่างด้านล่าง

การดำเนินการทางคณิตศาสตร์กับพอยน์เตอร์ (Pointer Arithmetics)

การดำเนินการกับพอยน์เตอร์ในที่นี้ 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\          \Desktop\Untitled1.exe
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)

เพื่อเข้าถึงข้อมูลที่ตัวแปรพอยน์เตอร์อ้างถึง (ชี้) อยู่ จะใช้เครื่องหมาย * นำหน้าตัวแปรพอยน์เตอร์นั้น ตั้งรูปแบบด้านล่างนี้

III *pointerName

จากตัวอย่างด้านล่างเป็นการยกตัวอย่างพอยน์เตอร์ พร้อมแสดงตำแหน่งที่อยู่ (Address) อย่างละเอียด

```

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				
Address	0FFC	1000	1004	1008

เมื่อจับบรรทัดที่ 5 และ 6 จะเป็นตั้งในตารางด้านข่าย และข่าวตามลำดับ

Name	b	a		
Value		30	30	
Address	0FFC	1000	1004	1008

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

เมื่อจับบรรทัดที่ 7 จะได้ค่าต่าง ๆ ทั้งค่าที่เก็บอยู่ในตัวแปร และตำแหน่งบนหน่วยความจำดังนี้

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

และเมื่อจับบรรทัดที่ 8 จะแสดงค่าที่ตัวแปรพอยน์เตอร์ C อ้างถึงอยู่ ในที่นี่คือ 30 (ค่าที่อยู่ในตัวแปร a)

ข้อควรรู้ : Address ที่แสดงในตัวอย่างเป็นการสมมติ และตัวแปร int ในคอมฯไฟล์ของผู้เขียนมีขนาด 4 ไบต์ เลข Address หลังจากประกาศตัวแปรจึงขับลดลงทีละ 4 (บางคอมฯไฟล์จะขับขึ้น)

ข้อควรระวัง : หากไม่ใส่ * นำหน้าชื่อตัวแปรจะ เป็นการเรียกเลขที่อยู่ (Address) ในหน่วยความจำ ของตัวแปรนั้นแทนการเรียกค่าที่พอยน์เตอร์นั้น อ้างอิง (ชี้) อยู่ ดังตัวอย่างด้านขวา

```

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 หรืออีกรูปแบบหนึ่งที่ใช้งานกันมาก คือการเปลี่ยนแปลงการอ้างอิงของตัวแปรพอยน์เตอร์ ซึ่งเป็นการใช้งานเกี่ยวกับการจัดการหน่วยความจำ โดยเนื้อหาดังกล่าวจะไม่ลงรายละเอียดในเอกสารชุดนี้

การใช้งานกับพอยน์เตอร์ 1 ตัว

จากตัวอย่างด้านล่าง เป็นการใช้ตัวแปรพอยน์เตอร์ชื่อ numberPt เพื่อแสดงค่าที่เก็บอยู่ในตัวแปร number ซึ่งไม่เหมือนกับการแสดงผลของค่าตัวแปร number และสามารถแสดงความสัมพันธ์ระหว่างตัวแปร และที่อยู่ (Address) ได้ดังรูปด้านขวา

numberPt = 0X61FF18

Address = 0X61FF18

a = 5

Address = 0X61FF18

ภาพแสดงความสัมพันธ์ระหว่างตัวแปร และที่อยู่ (Address) หลังจนโปรแกรม

```

1 #include<stdio.h>
2
3 main()
4 {
5     int number = 5;
6     int *numberPt; //Declaring pointer for int data type
7     numberPt = &number; //Pointing 'number' to 'numberPt'
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 }
```

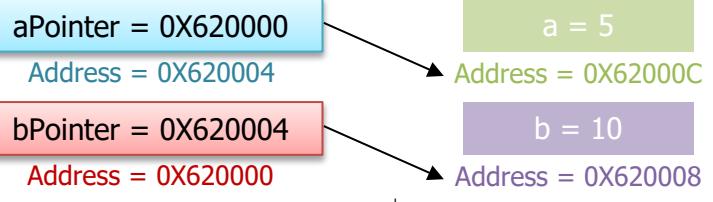
การใช้งานกับพอยน์เตอร์หลายตัว

ตัวอย่างการใช้งานกับพอยน์เตอร์หลายตัวอ้างอิงถึงตัวแปรอื่น

```
1 #include<stdio.h>
2 main()
3 {
4     int a = 5, b = 10;
5     int *aPointer = &a, *bPointer = &b; //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 a ด้วย aPointer และตัวแปร b ด้วย bPointer การแสดงผลลัพธ์ท่าโดยใช้พอยน์เตอร์เข้าถึงค่าที่อ้างอิงโดยใช้ * นำหน้าชื่อตัวแปรพอยน์เตอร์



ภาพแสดงความสัมพันธ์ระหว่างตัวแปร และที่อยู่ (Address) หลังจบโปรแกรม

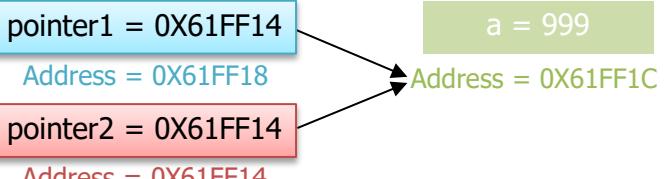
ตัวอย่างการใช้พอยน์เตอร์หลายตัวอ้างอิงถึงตัวแปรตัวเดียวกัน

```
1 #include<stdio.h>
2 main()
3 {
4     int a = 5;
5     int *pointer1 = &a, *pointer2 = &a; //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 . . .
```



ภาพแสดงความสัมพันธ์ระหว่างตัวแปร และที่อยู่ (Address) หลังจบโปรแกรม

ตัวอย่างนี้มีการใช้งานพอยน์เตอร์ 2 ตัวอ้างอิงถึงตัวแหน่งของตัวแปร a เมื่อมองกัน ข้อสังเกตคือ เมื่อค่าที่ถูกอ้างอิงถึงมีการเปลี่ยนแปลง ค่าที่พอยน์เตอร์อ้างอิงถึง ก็จะเปลี่ยนแปลงตามไปด้วยทุกด้วย ลักษณะการ "ขี้" ในตอนจบการทำงานจะดังรูปด้านขวา

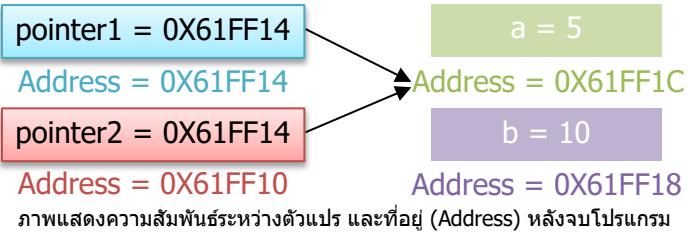
ตัวอย่างการอ้างอิงพอยน์เตอร์อีกตัวหนึ่ง โดยใช้พอยน์เตอร์

```
1 #include<stdio.h>
2 main()
3 {
4     int a = 5, b = 10;
5     int *pointer1 = &a, *pointer2 = &b;
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 นิยมใช้กันมากเนื่องจากพอยน์เตอร์สามารถใช้อ้างอิงตำแหน่งที่อยู่ของตัวแปรอาร์เรย์ได้ โดยปกติจะเป็นการอ้างอิงถึงตำแหน่งที่อยู่ของตัวแปรอาร์เรย์ตำแหน่งแรกเท่านั้น (อินเด็กซ์เป็น 0)

การกำหนดตัวแปรพอยน์เตอร์อ้างอิงไปยังอาร์เรย์

การประกาศให้พอยน์เตอร์อ้างอิงไปที่อาร์เรย์ตอนประกาศตัวแปรสามารถทำได้ดังนี้

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

หรือสามารถกำหนดค่าให้พอยน์เตอร์หลังจากประกาศตัวแปรได้ดังนี้

```
||| pointerName = arrayName;
```

จะสังเกตได้ว่าไม่ต้องใส่ & หน้าตัวแปรที่ต้องการดึง Address ให้ตัวแปรพอยน์เตอร์ใช้สำหรับอ้างอิง เนื่องจากตัวแปรประเภทอาร์เรย์มีลักษณะคล้ายพอยน์เตอร์อยู่แล้ว และเมื่อกำหนดค่าในรูปแบบนี้ตัวแปรพอยน์เตอร์จะเริ่มอ้างอิงในอินเด็กซ์ที่ 0 หากต้องการอ้างอิงที่อินเด็กซ์อื่น ๆ ให้กำหนดค่าดังรูปแบบนี้

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

การเข้าถึงค่าตัวแปรในแต่ละอินเด็กซ์สามารถทำได้ 2 รูปแบบดังนี้

```
||| *(pointerName + index);
||| pointerName[index];
```

ข้อสังเกต : การเข้าถึงในแบบแรกจำเป็นต้องมี () เพื่อเลื่อนตำแหน่งของตัวแปรพอยน์เตอร์ ถ้าไม่มีวงเล็บพอยน์เตอร์ จะทำการอ้างอิงค่าที่ตัวแปรพอยน์เตอร์นั้นชี้อยู่ และทำการเพิ่มค่าให้กับตัวที่ถูกชี้แทนการเลื่อนตำแหน่งพอยน์เตอร์

จากตัวอย่างโค้ดด้านล่างทั้งสองโค้ดเป็นการใช้พอยน์เตอร์กับอาร์เรย์ ซึ่งผลลัพธ์ที่ได้มีค่าเท่ากัน และได้ตารางแสดงค่าที่เก็บอยู่ในตัวแหน่งต่าง 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				

```
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 }

```

จากตัวอย่างโค้ดด้านล่างเป็นการนำพอยน์เตอร์หล่าย 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 จากตัวอย่างด้านล่างจึงทำการเลื่อนตำแหน่งที่พอยน์เตอร์ซึ่งแทนการเพิ่มค่าด้วยการใส่ () 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 }

```

นอกจากระบบสามารถส่งผ่านอาร์เรย์ไปให้ฟังก์ชันโดยใช้พอยน์เตอร์ได้เช่นกัน ซึ่งอยู่ในหัวข้อต่อ ๆ ไป

การใช้งานพอยน์เตอร์กับข้อความ

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

```

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!
Process exited after 0.1053
seconds with return value 0
Press any key to continue . .

```

การใช้งานพอยน์เตอร์อ้างอิงพอยน์เตอร์ที่กำลังอ้างอิงถึงตัวแปรอื่น

พอยน์เตอร์สามารถนำมารอ้องกันเองได้หลาย 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)

จากตัวอย่างด้านขวาภายในฟังก์ชันจะทำการสร้างพอยน์เตอร์ที่ชี้ไปที่เดียวกันกับตำแหน่งที่ได้รับมาเป็นอาร์กิวเมนต์ i และ j ถ้าหากมีการเปลี่ยนแปลงค่า i หรือ j ค่าตำแหน่งเดิมในฟังก์ชันหลักจะไม่ถูกเปลี่ยนแปลง แต่ถ้าหากทำการเปลี่ยนแปลงค่าที่พอยน์เตอร์ i และ j ชื่อยุ่งจะส่งผลให้ตัวแปร 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 มือญี่ปุ่นภาษา 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 หล่ายชนิดอยู่ข้างใน ยกตัวอย่างนักเรียนจะมีข้อมูลเช่น ชื่อ นามสกุล อายุ เกรด ห้องเรียน เลขที่ เพศ เป็นต้น คุณสมบัตินี้ถูกเอาไปต่อ�่อในเชิง Object Oriented Programming (OOP) เรียกว่า คลาส (Class) ซึ่งมีอยู่ในภาษาอื่น ๆ 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 โดยการตั้งชื่อให้เป็นสากลจะชี้แจงค่าด้วยตัวพิมพ์ใหญ่
 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 โดยโครงสร้างดังลักษณะ School และมีตัวแปรประเภทโครงสร้างชื่อ student และ teacher ภายในโครงสร้างจะมีฟิลด์ข้อมูล (Data Field) คือตัวแปรหลายประเภทสำหรับเก็บข้อมูลต่าง ๆ 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;
```

```
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 ถ้าจะมองง่าย ๆ ให้ถือว่า School หรือ Office เป็นตัวแปรประเภทหนึ่งไปเลย 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 ตัวได้โดยใช้อาร์เรย์ช่วยได้ดังรูปแบบด้านล่าง

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

โดยที่ index เป็นจำนวนอินเดกซ์ของตัวแปรประเภทโครงสร้างที่ต้องการ

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

การเข้าถึงค่านั้นผ่านสามารถอ่านค่า 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 เพื่อใช้งานกันทุก ๆ พัฒนาโดยผู้เขียนเป็นการกำหนดค่าเริ่มต้นพร้อมกับประกาศตัวแปรประเภทโครงสร้าง และแสดงผลข้อมูลภายใต้ตัวแปรประเภทโครงสร้าง แต่ในผู้เขียนเป็นการกำหนดค่าหลังจากประกาศตัวแปรประเภทโครงสร้าง และแสดงผลข้อมูลภายใต้ตัวแปรประเภทโครงสร้างเหมือนเดิม

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

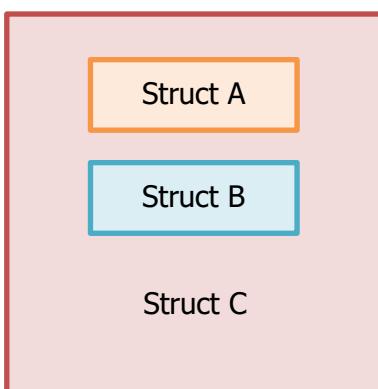
```

||| struct_var_1.var1[index] ;

```

โครงสร้างข้อนestedโครงสร้าง (Nest Structure)

ในภาษาซีสามารถสร้างโครงสร้างข้อนestedเข้าไปในโครงสร้างได้อีกที่จะเรียกว่า “Nest Structure” เช่น ประกาศโครงสร้าง C โดยภายในโครงสร้าง C ประกอบด้วยโครงสร้าง A และ B ซึ่งมีลักษณะดังรูป



```

1 struct A {
2   ...
3 };
4 struct B {
5   ...
6 };
7 struct C {
8   struct A data_a;
9   struct B data_b;
10 ...
11 };

```

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
API	ย่อมาจาก Application Programming Interface คือช่องทางการเชื่อมต่อ หรือติดต่อกับโปรแกรม สำหรับผู้เขียนโปรแกรมเรียกใช้งานโปรแกรม
Client / Server	การที่มีเครื่องผู้ให้บริการ (Server) และเครื่องผู้ใช้บริการ (Client) เชื่อมต่อกันอยู่ และเมื่อเครื่องผู้ใช้บริการได้มีการติดต่อร้องขอบริการจากเครื่องผู้ให้บริการ เครื่องผู้ให้บริการก็จะจัดการตามที่เครื่องผู้ขอใช้บริการร้องขอ หลังจากนั้นจึงส่งข้อมูลกลับไปให้
Compiler	โปรแกรมที่ทำหน้าที่ในการแปลง ภาษาระดับสูงให้เป็นภาษาเครื่องที่คอมพิวเตอร์เข้าใจ
Computer Languages	ภาษาที่ใช้สำหรับเขียน เพื่อสร้างโปรแกรมขึ้นมาเพื่อใช้งานบนเครื่องคอมพิวเตอร์
Computer System	ระบบที่พัฒนามาจาก 2 ส่วนใหญ่ ๆ คือ ฮาร์ดแวร์ และซอฟต์แวร์
Hardware	ส่วนประกอบของคอมพิวเตอร์ หรืออุปกรณ์ต่าง ๆ ที่ประกอบขึ้นเป็นเครื่องคอมพิวเตอร์ มีลักษณะเป็นโครงร่างสามารถมองเห็นด้วยตา และสัมผัสได้ (เป็นรูปธรรม) เช่น เครื่องพิมพ์ จอภาพ เม้าส์ คีย์บอร์ด เป็นต้น
IDE	ย่อมาจาก Integrated Development Environment เป็นแอปพลิเคชันซอฟต์แวร์ที่ค่อยอ่านวิเคราะห์ความแตกต่างกันนักเขียนโปรแกรม สำหรับการพัฒนาซอฟต์แวร์ เช่น VS Code
Loader	โปรแกรมที่ทำงานร่วมกับ Hardware มีหน้าที่โหลด Source Code ลงไว้ในหน่วยความจำ
Personal Computer	ระบบคอมพิวเตอร์ที่มีเครื่องคอมพิวเตอร์เครื่องเดียว และไม่ได้ทำการเชื่อมต่อกับเครื่องคอมพิวเตอร์เครื่องอื่น ๆ
SDK	ย่อมาจาก Software Development Kit คือชุดเครื่องมือที่จะช่วยในการเขียนโปรแกรมในอย่างใดอย่างหนึ่ง รวบรวมเครื่องมือทุก ๆ อย่างในการเขียนโปรแกรมนั้น ๆ ไว้
Software	ชุดคำสั่ง หรือ โปรแกรมที่ใช้ควบคุมการทำงานของเครื่องคอมพิวเตอร์ เขียนขึ้นโดยภาษาคอมพิวเตอร์จากนักเขียนโปรแกรม
System development Life Cycle	กระบวนการพัฒนาระบบงาน หรือระบบเทคโนโลยีสารสนเทศด้วย เพื่อช่วยแก้ปัญหาทางธุรกิจหรือตอบสนองความต้องการขององค์กร
Time Sharing	การเชื่อมต่อของคอมพิวเตอร์หลาย ๆ เครื่องมาต่อกับเครื่องคอมพิวเตอร์ศูนย์กลางที่ค่อยประมวลผลต่าง ๆ โดยใช้หลักการแบ่งเวลา และคอมพิวเตอร์หลาย ๆ เครื่องเหล่านั้นเรียกว่า Terminal

หมวดโครงสร้างภาษา

คำศัพท์	ความหมาย
Compiled Language	ภาษาที่มีคอมไพล์เลอร์ (Compiler) หน้าที่ในการคอมไพล์ (Compile) หรือแปลงคำสั่งทั้งหมดในโปรแกรมให้เป็นภาษาเครื่อง (Machine Language) เพื่อให้เครื่องคอมพิวเตอร์นำคำสั่งเหล่านั้นทำงานตามคำสั่งต่อไป
Constants	ชนิดข้อมูลแบบคงที่ ไม่สามารถเปลี่ยนแปลงในขณะที่โปรแกรมทำงานอยู่
Flowchart	ผังงาน มีไว้เพื่อให้นักเขียนโปรแกรมเขียนโปรแกรมได้ง่ายขึ้น และไม่สับสน
Global Variables	ตัวแปรที่สามารถเรียกใช้ได้ทุกฟังก์ชันของโปรแกรม หากมีการประกาศตัวแปร
Header Files	ส่วนที่เก็บไฟล์าร์มาราตรฐานของภาษา C ซึ่งจะถูกดึงเข้ามาร่วมกับโปรแกรมในขณะที่กำลังทำการคอมไพล์
Local Variables	ตัวแปรที่จะสามารถเรียกใช้ได้ภายในฟังก์ชันที่ประกาศตัวแปรนั้น
Return Value	เป็นค่าที่ส่งกลับ เมื่อฟังก์ชันดังกล่าวถูกเรียก
Statements	ประโยคที่เขียนขึ้นเพื่อให้โปรแกรมทำงาน ประกอบด้วยตัวแปร หรือนิพจน์ต่าง ๆ เพื่อใช้ประกาศตัวแปร กำหนดค่าเริ่มต้น กำหนดเงื่อนไข และใช้ในคำสั่งควบคุม นิพจน์ หรือตัวแปรที่เขียนขึ้นเป็นประโยคคำสั่งส่วนมากจะใช้เครื่องหมาย ; (Semicolon) ปิดท้ายคำสั่ง

หมวดการแสดงผล และรับข้อมูล

คำศัพท์	ความหมาย
Back-slash character	ชุดค่าคงที่ของตัวอักษร ที่ใช้ในการควบคุมการแสดงผล และใช้ในการช่วยแสดงผลอักขระบางตัวที่ไม่สามารถแสดงผลออกทางหน้าจอ
getch	คำสั่งในการรับค่าข้อมูลจากผู้ใช้เข้ามาทางคีย์บอร์ด 1 ตัวอักษร โดยที่เคอร์เซอร์จะไม่ขึ้นบรรทัดใหม่ และไม่ต้องกดปุ่ม Enter เมื่อป้อนครบ 1 ตัวอักษร นอกจากนี้ตัวอักษรที่ป้อนลงไปจะไม่แสดงผลออกทางหน้าจอ
getchar	คำสั่งในการรับค่าข้อมูลจากผู้ใช้เข้ามาทางคีย์บอร์ด 1 ตัวอักษร จะต้องกดปุ่ม Enter เพื่อสั่งสุดการรับข้อมูล ตัวอักษรที่ป้อนลงไปจะแสดงผลออกทางหน้าจอ และเคอร์เซอร์จะขึ้นบรรทัดใหม่
getche	เป็นฟังก์ชันที่ทำงานเหมือนฟังก์ชัน getch แต่จะแสดงตัวอักษรที่พิมพ์เข้าไป ออกทางจอภาพด้วย
printf	คำสั่งที่ใช้สำหรับแสดงค่าต่าง ๆ ออกทางจอภาพ
scanf	คำสั่งในการรับค่าจากผู้ใช้เข้ามาทางคีย์บอร์ด
Standard Input File	ไฟล์ที่ใช้เก็บข้อมูลจากคีย์บอร์ด และเรียงลำดับตามข้อมูลที่ได้รับมาจากผู้ใช้ ก่อนที่จะมีการนำไปใช้ หรือเรียกว่ายัง ๆ ว่าเป็น Buffered
Standard Output File	ไฟล์ที่ใช้เก็บข้อมูล ก่อนที่จะนำไปแสดงออกทางอุปกรณ์ต่าง ๆ ซึ่งจะเป็น Text File เมื่อต้องการแสดงข้อมูลนั้น จะทำการแปลงข้อมูลเป็นข้อความก่อนแล้วจึงค่อยแสดงออกทางจอภาพ

หมวดนิพจน์ทางคณิตศาสตร์

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

หมวดฟังก์ชันของภาษา C

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

หมวดคำสั่งเงื่อนไข

คำศัพท์	ความหมาย
And	ใช้เชื่อมค่าความจริงของ 2 นิพจน์ หรือมากกว่า ผลลัพธ์จะเป็นจริงเมื่อนิพจน์ทั้ง 2 หรือทั้งหมดเป็นจริง และจะเป็นเท็จเมื่อมีอย่างน้อย 1 พจน์เป็นเท็จ ซึ่งใช้สัญลักษณ์ & หรือ &&
else – if	เป็นคำสั่งสำหรับการสร้างเงื่อนไขแบบหลายทางเลือก โดยการเพิ่มเงื่อนไข else if เข้ามาหนึ่งหมายเลขถึง เมื่อเช็คเงื่อนไขใน if และไม่ตรงเงื่อนไข จะมีการเช็คเงื่อนไขใน else if ถัดไปเรียงลำดับไปเรื่อย ๆ เมื่อตรงกับเงื่อนไขจะทำการคำสั่งในบล็อกนั้น และจบการทำงานแบบเงื่อนไขทันที (ไม่เช็คเงื่อนไขต่อ ๆ ไป ถ้าหากยังมี else if หรือ else ต่อ) สามารถใช้ร่วมกับคำสั่ง else ซึ่งเป็นบล็อกของคำสั่งที่จะทำงานเมื่อหากไม่มีเงื่อนไขใดใน if และ else if (ถ้ามี) เป็นจริง
if	เป็นคำสั่ง 2 ทางเลือก ซึ่งถ้าเงื่อนไข (นิพจน์) เป็นจริงจะทำการคำสั่งที่กำหนดไว้ใน if และถ้าเงื่อนไข (นิพจน์) เป็นเท็จจะไม่�行 หลังจากนั้นทำการคำสั่งถัดกันมาหลังจาก if
if... else	เป็นคำสั่ง 2 ทางเลือก ซึ่งถ้าเงื่อนไข (นิพจน์) เป็นจริงจะทำการคำสั่งที่กำหนดไว้ใน if และถ้าเงื่อนไข (นิพจน์) เป็นเท็จจะไปทำการคำสั่งที่กำหนดไว้ใน else
Logical Data	ประเภทข้อมูลแบบตรรกะ
Nested if	เป็นคำสั่ง if... else ที่มีคำสั่ง if... else หรือคำสั่ง if ซ้อนกันอยู่หลายชั้น
Not	นิสเช็ค หรือคำสั่งที่ใช้กลับค่าความจริง ซึ่งใช้สัญลักษณ์ !
Or	ใช้เชื่อมค่าความจริงของ 2 นิพจน์ หรือมากกว่า ผลลัพธ์จะเป็นจริงเมื่อมีอย่างน้อย 1 นิพจน์ เป็นจริง และเป็นเท็จเมื่อนิพจน์ทั้ง 2 หรือทั้งหมดเป็นเท็จ ซึ่งใช้สัญลักษณ์ หรือ
switch	เป็นคำสั่งหลายทางเลือก ซึ่งเป็นคำสั่งที่ใช้หลักการของ Nested if และคำสั่งนี้จะมีตัวแปรสำหรับควบคุม (เปรียบได้กับเงื่อนไข) กับคำสั่งถ้าหากตัวแปรนั้นตรงกับ case ที่กำหนดไว้บัน

หมวดคำสั่งวนลูป

คำศัพท์	ความหมาย
break	เป็นคำสั่งที่ให้โปรแกรมออกจากลูปทันที โดยไม่ทำการคำสั่งต่อ ซึ่งคำสั่ง break นี้สามารถใช้ได้กับลูปหลาย ๆ ลูปไม่ว่าจะเป็น while, do while, for และอื่น ๆ รวมถึงยังต้องใช้ร่วมกับ switch ด้วย
continue	ถูกใช้เพื่อข้ามการทำงานในรอบปัจจุบัน ซึ่งจะไม่ทำการคำสั่งหลัง continue ที่ถูกเรียก และจะทำการเริ่มลูปเริ่มรอบถัดไปทันที
do while	เป็นลูปแบบ Post – Test Loop คือมีคำสั่งก่อนที่จะไปทำการตรวจสอบตัวควบคุมลูป (เงื่อนไข หรือนิพจน์)
exit	เป็นคำสั่งที่ใช้สำหรับการออกจากโปรแกรมย่อย (Sub หรือ Function) หรือออกจากภารณ์ช้า (Loop) จะใช้ร่วมกับเงื่อนไขที่ผู้ใช้ต้องการ
for	เป็นลูปแบบ Pretest Loop ที่ใช้นิพจน์ 3 นิพจน์สำหรับควบคุม โดยนิพจน์แรกเป็นนิพจน์เริ่มต้นส่วนมากใช้กำหนดค่าตัวควบคุม นิพจน์ที่ 2 เป็นเงื่อนไขในการตรวจสอบตัวควบคุมลูป (เงื่อนไข หรือนิพจน์) และนิพจน์ที่ 3 เป็นนิพจน์สำหรับเพิ่ม หรือลดค่าตัวควบคุมลูป
Loop	เป็นคำสั่งที่ใช้ควบคุมการทำงานของโปรแกรมแบบวนซ้ำ การทำงานเดิมๆตามเงื่อนไขที่กำหนด เช่น การวนซ้ำการทำงานเดิมเป็นจำนวน 10 รอบ ทำงานซ้ำ ๆ จนกว่าเงื่อนไขจะเป็นเท็จ
Pretest	ลูปประเภทหนึ่ง จะทำการตรวจสอบเงื่อนไขก่อนว่าเป็นจริง หรือเป็นเท็จ ถ้าเป็นจริงก็ให้เข้าไปทำการคำสั่ง หรือชุดคำสั่งต่อไป และเมื่อทำการคำสั่ง หรือชุดคำสั่งเสร็จ ก็จะกลับมาทำการตรวจสอบเงื่อนไขอีกรอบ และจะทำเช่นนี้ไปเรื่อย ๆ จนกว่าเงื่อนไขจะเป็นเท็จ ถึงจะจบการทำงานของลูป
Post – Test	ลูปประเภทหนึ่ง จะทำการคำสั่ง หรือชุดคำสั่งก่อน เมื่อเสร็จแล้วจึงจะมาตรวจสอบเงื่อนไขว่าเป็นจริง หรือเป็นเท็จ ถ้าเป็นจริงก็จะกลับไปทำการคำสั่ง หรือชุดคำสั่งเดิมอีกรอบ และจะทำจนกว่าเงื่อนไขจะเป็นเท็จ เช่นเดียวกัน
while	ลูป while นี้ใช้เงื่อนไขเป็นตัวควบคุมลูป (เงื่อนไข หรือนิพจน์) ซึ่งจะเป็นลูปแบบ Pretest Loop ซึ่งจะทำการตรวจสอบเงื่อนไขก่อนที่จะเข้าไปทำการคำสั่งในลูป

ASCII TABLE

ผังอักขระและสกีที่ไม่สามารถแสดงผล

อักขระที่ปรากฏในตารางเป็นเพียงการแสดงว่า ณ ตำแหน่งนั้นมีรหัสดังกล่าวอยู่ ไม่ใช่สัญลักษณ์ที่จะนำมาแสดงผลเป็นหลัก ซึ่งไข้เป็นรหัสควบคุมการพิมพ์บนเครื่องพิมพ์ หรือตัวแบ่งข้อมูลในสื่อบันทึกข้อมูลบางชนิด (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

ผังอักขระและสกีที่สามารถแสดงผล

BIN	DEC	HEX	CHAR
0010 0000	32	20	(ช่องว่าง)
0010 0001	33	21	!
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	\$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29)
0010 1010	42	2A	*
0010 1011	43	2B	+
0010 1100	44	2C	.
0010 1101	45	2D	-
0010 1110	46	2E	.
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6
0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9
0011 1010	58	3A	:
0011 1011	59	3B	;
0011 1100	60	3C	<
0011 1101	61	3D	=
0011 1110	62	3E	>
0011 1111	63	3F	?

BIN	DEC	HEX	CHAR
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O
0101 0000	80	50	P
0101 0001	81	51	Q
0101 0010	82	52	R
0101 0011	83	53	S
0101 0100	84	54	T
0101 0101	85	55	U
0101 0110	86	56	V
0101 0111	87	57	W
0101 1000	88	58	X
0101 1001	89	59	Y
0101 1010	90	5A	Z
0101 1011	91	5B	[
0101 1100	92	5C	\
0101 1101	93	5D]
0101 1110	94	5E	^
0101 1111	95	5F	_

BIN	DEC	HEX	CHAR
0110 0000	96	60	`
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f
0110 0111	103	67	g
0110 1000	104	68	h
0110 1001	105	69	i
0110 1010	106	6A	j
0110 1011	107	6B	k
0110 1100	108	6C	l
0110 1101	109	6D	m
0110 1110	110	6E	n
0110 1111	111	6F	o
0111 0000	112	70	p
0111 0001	113	71	q
0111 0010	114	72	r
0111 0011	115	73	s
0111 0100	116	74	t
0111 0101	117	75	u
0111 0110	118	76	v
0111 0111	119	77	w
0111 1000	120	78	x
0111 1001	121	79	y
0111 1010	122	7A	z
0111 1011	123	7B	{
0111 1100	124	7C	
0111 1101	125	7D	}
0111 1110	126	7E	~



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

Flowchart — ผังงาน —

คือ รูปแบบหรือสัญลักษณ์ที่ใช้แทนคำอธิบายหรือคำพูด
ที่ใช้ในอัลกอริทึม เพื่อให้คนอื่นสามารถเข้าใจได้ง่ายขึ้น

สัญลักษณ์ที่ใช้ในการเขียนผังงาน



สัญลักษณ์แสดง
จุดเริ่มต้นหรือจุดสิ้นสุด



สัญลักษณ์ใน
การสร้างเงื่อนไข



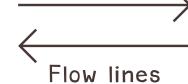
ใช้ในการประมวลผล



การนำเข้าหรือส่งออกข้อมูล
โดยไม่ระบุสืบ



สัญลักษณ์จุดเชื่อมของ
ผังงานในหน้าเดียวกัน



ลูกศรแสดง
ทิศทางการทำงาน

Pseudo Code

ภาษาที่ใช้จำลองภาษาโปรแกรม โดยจำลองคำสั่งจริงแบบย่อๆ ตามขั้นตอนที่ต้องการสร้าง
ซึ่งแบ่งเป็น 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

ส่วนหัวโปรแกรม

ส่วนตัวบ่งคุณ

`#include<file.h>`

Preprocessor Directive •

`type function_name (type);`

Function Prototype Declaration •

`type variable;`

เป็นตัวแปรที่สามารถใช้ที่ล้วนในหนึ่งของโปรแกรมก็ได้

`int main()`

{

เป็นตัวแปรที่สามารถใช้เฉพาะในฟังก์ชันที่ประกาศไว้

`type variable;`

`statement;` คำสั่ง •

`return 0;`

}

Main Function

`type function_name (type variable)`

{

`type variable;`

`statement;`

`return (var);`

}

*การรับค่ามา (Parameter)

คือ การส่งค่าจากฟังก์ชันหลักที่เรียกใช้งาน
มาสู่ในฟังก์ชันย่อย โดยที่จะส่งมา กี่ค่าก็ได้
โดยให้ครบตามจำนวนของ Parameter ตามที่
กำหนดไว้ในฟังก์ชันย่อย

Function

Header File

เป็นล้วนที่เรียกใช้ไลบรารี ซึ่งขึ้นต้นด้วย # เเละอโดยเป็นการนำโค้ดในไฟล์นั้นมาใช้

1. ดึงคำสั่งจากจากไฟล์เดอร์ที่เก็บ Source Code ก่อนแล้วค่อยหาจากที่เก็บไว้ในตัวคอมไฟล์

`#include "headname.h"`

2. ดึงคำสั่งจากที่เก็บไว้ในตัวคอมไฟล์ที่เดียวเท่านั้น

`#include <headname.h>`

ตัวอย่าง

`#include<stdio.h>` : คำสั่งที่นำไป

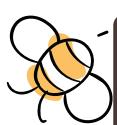
`#include<conio.h>` : คำสั่งจัดการหน้าจอ

`#include<math.h>` : คำสั่งเกี่ยวกับคณิตศาสตร์

`#include<string.h>` : คำสั่งเกี่ยวกับข้อความ



ค่าคงที่ Constant



#define identifier replacement

```
#define PI 3.14
```

```
#define Gravity 10
```

const type identifier = value;

```
const double PI = 3.1415926;
```

```
const char N = 'a';
```

หรือเขียนได้รูปแบบนี้



*ถ้าต้องการภาษาอังกฤษต้องเลิกและตัวให้ญี่ปุ่นกว่า “แต่ต่างกัน”
ภาษาอังกฤษ

โดย ห้ามตั้งชื่อข้าว กับคำสlang (Reserved Word)

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

ตัวคงที่

สี 2 ลักษณะ คือ Local V. และ Global V.

ซึ่งทางสองลักษณะนี้ มีการใช้งานกันต่างกันเล็กน้อย

การประมวลผลตัวแปร

```
type identifier;
type identifier = value;
```

การกำหนดค่าให้ตัวแปร

หรือที่บันทึกค่าสั่งแบ่งมาเป็น

identifier = value;

**อ่านจากขวาไปซ้าย

- ข้อมูลชนิดว่างเปล่า : void
- ข้อมูลชนิดจานวนเต็ม : short, int, long int
ex. 1, 20, -6
- ข้อมูลชนิดจาริจ : float, double, long double
ex. 12.600000, -46.000000
- ข้อมูลชนิดอักขระ : char
ex. 'A', '2', 'S'
(แปลงตัวอักษรเป็น ASCII Code 1)
- ข้อมูลชนิดตรรกะ : bool
ex. True, False

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

จำนวนที่แสดงออกมาก
 เช่น `%3f` คือ เลขทศนิยม 3 ตำแหน่ง

`%4.5s` คือ เว้นไป 4 ช่องและเอาแค่ 5 ตัวอักษร

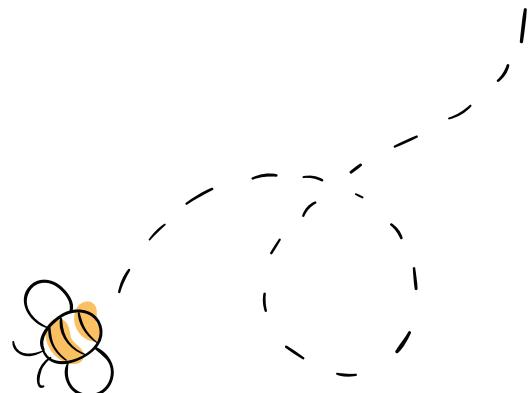
`%2d` คือ เว้นไป 2 ช่องของเลขจำนวนเต็ม

2. ใช้คำสั่ง putchar() สำหรับแสดงแบบอักขระตัวเดียว (Character)

```
putchar(var);
```

3. ใช้คำสั่ง puts() สำหรับแสดงข้อมูลแบบล้ายอักขระ (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 * ข้อมูลที่รับ
เข้ามาในลำดับต่อๆ กันจะไม่ถูกจัดเก็บลงตัวแปร

การกำหนดจำนวนตัวอักขระในการรับค่าของข้อมูลชนิดข้อความของคำสั่ง scanf()

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

สามารถกำหนดความยาวข้อความที่ต้องการรับสูงสุดได้
โดยการใส่ตัวเลขหน้า s ใน %s ในส่วนของ format

สามารถเปลี่ยนได้

การนับจำนวนตัวอักขระที่รับค่าของข้อมูลด้วยคำสั่ง scanf()

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

สามารถนับความยาวของข้อความที่รับเข้ามา เพื่อนำไปใช้กำหนด
เงื่อนไขต่างๆ 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



Conversion Data Type

Explicit Type Conversion

แบบชนิดที่จะมุ่งทำให้คนติดตามกีฬา

(Type) ExpressionOrVariableName

**float ans; ----- ការណែនាំថាទ្វបរ
 ដើម្បី កុំពេញលោកស្រីបង្កើត**
**int num = 20; ----- ការណែនាំថាអ្នក
 ត្រូវបានផ្តល់លទ្ធផលជានួយ**

សម្រាប់ការពាក្យ 20

```
ans = (float)num; ----- แปลงข้อมูลจาก int เป็น float (รูปนี้ค่าต้อง 20.00)
```

နေပါတ်မှာ အမြန် မြန်မာ လိပ်စာမျက်နှာများကို ဖော်ပြန်ခဲ့ရန် အကြောင်း ဖြစ်ပါသည်။

Implicit Type Conversion

ՃՐԱԾԵՍ ՖԻՇՄԱՆ ՄԱԿԱՀԱՅ ԱՌԵՎ ՀԵՎԱԴՐԱՎՈՐ ՄԱԿԱՀԱՅ ԱՌԵՎ ՀԵՎԱԴՐԱՎՈՐ

កុបេបាតីខ្លួនបែក		ការងារមាមការ	កំណត់ flag
	ឯកសារនៃការងារដែលបានរកចុចិត្ត	-	ឱ្យការងារដែលបានរកចុចិត្ត ធានាក្នុងទីតាំងសាន្តរោគសាធារណការ
	ឯកសារនៃការងារដែលបានរកចុចិត្ត	+	ឱ្យការងារដែលបានរកចុចិត្ត ធានាក្នុងទីតាំងសាន្តរោគសាធារណការ
0	ឯកសារនៃការងារដែលបានរកចុចិត្ត	0	ឯកសារនៃការងារដែលបានរកចុចិត្ត

ଶ୍ରୀମଦ୍ଭଗବତ

卷之三

ຄະນະຄວາມ

၁ ក្នុងការបង្កើតរបស់សាសនាថ្មី តើអ្វីដែលត្រូវបានធ្វើឡើង

Explicit Type Conversion

แบบนี้ก็จะได้รับการอนุมัติตามที่ต้องการ

(Type) ExpressionOrVariableName

၁၂၅

~ XI of XX ~

ຮັກສ່ຽງແບບ	ຍືນດີຂອງມູນຄຸນ	ຄວາມການຍາຍ
<code>>C</code>	ຕັ້ງຄົງກະທິ່ນເຫັນຕີ່ງ ຈຳນວນເຕີມກົມືດ int	
<code>>d</code>	ຈຳນວນເຕີມກົມືດ long	
<code>>ld</code>	ຈຳນວນເຕີມກົມືດ long ຈຳນວນເຕີມແບບເລືອກໃໝ່ນັ້ນຕີ່	
<code>>e ທີ່ອ >E</code>	ຈຳນວນຈົບງານຕີ່ໂດ float ຈຳນວນຈົບງານຕີ່ໂດ double	
<code>>f</code>	ຈຳນວນຈົບງານຕີ່ໂດ double	
<code>>lf</code>	ຈຳນວນຈົບງານຕີ່ໂດ General format	
<code>>g ທີ່ອ >G</code>	ຈຳນວນເຕີມກົມືດ int ເລີນຈົບງານແບດ	
<code>>i</code>	ພວຍນີ້ໂຕອ່ຽວ	
<code>>o</code>	ກົດຄວາມ	
<code>>p</code>	ຈຳນວນເຕີມມັກວາ (Unsigned)	
<code>>q</code>	ເລີນຈົບງານສຶກກ	
<code>>n</code>	ຈຳນວນເຕີມກົມືດ short	
<code>>hd</code>	ຈຳນວນເຕີມຮູນແມດກົມືດ long	
<code>>lo</code>	ຈຳນວນເຕີມຮູນແມດກົມືດ short	
<code>>hx</code>	ຈຳນວນຈົບງານສຶກກົມືດ short	
<code>>Lf</code>	ຈຳນວນຈົບງານຕີ່ໂດ long double	
<code>>lu</code>	ຈຳນວນເຕີມກົມືດ unsigned long	
		ຄ້າການທີ່ໄປ
<code>\b</code>	ເລືອນເຄອງຮັກສ່ຽງກົມືດທີ່ໄປ 1 ຢັກຮັບ	
<code>\t</code>	ກື່ນປຣຖິຕິກົມືດ	
<code>\r</code>	ເລືອນເຕອກຮັກສ່ຽງປາກ່າງໜ້າໄສດີ	
<code>\t</code>	ເງິນຮຽດ 8 ຂອງທີ່ກົກກາຣີ Tab	
<code>\0</code>	Null	
<code>\'</code>	ແສດຖາວີ່ຈົບງານຍາຍ “ (single quote)	
<code>\”</code>	ແສດຖາວີ່ຈົບງານຍາຍ “ (double quote)	
<code>\ </code>	ແສດຈະເຕີ່ຈົບງານຍາຍ \	

Full Version [A0] : bit.ly/infoCCEBU

၁၂၅

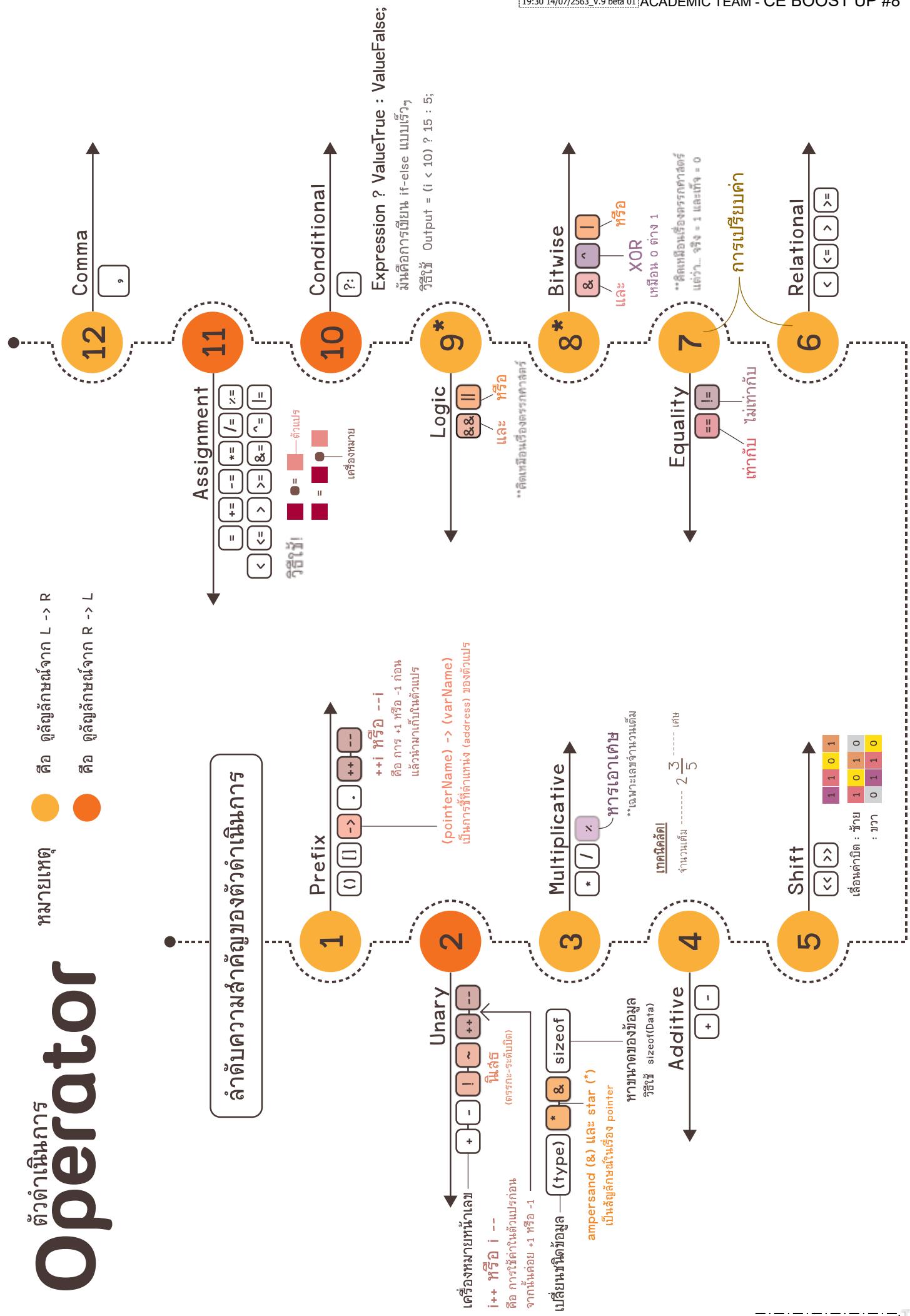
char > short > int > unsigned int > long int

- ० -

bit.ly/bookceboostup

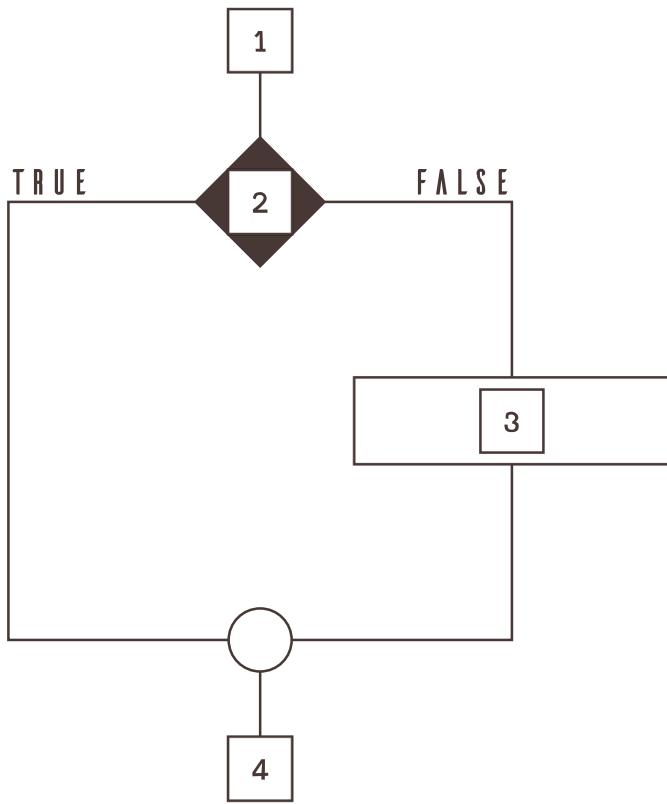
Operator ຕັ້ງດຳເນື້ອກາຮົງ

	គីវា ទូតែស្ម័គមជាមុនការ L -> R
	គីវា ទូតែស្ម័គមជាមុនការ R -> L



Decision Control Statement

การทำงานด้วยการตัดสินใจ



if

เป็นคำสั่งที่เราใช้กำหนดให้โปรแกรมตัดสินใจทำในสิ่งที่เป็นจริงหรือไม่ทำสิ่งใดเลย

```

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

if - else

เป็นคำสั่งที่เราใช้กำหนดให้โปรแกรมตัดสินใจทำในสิ่งที่เป็นจริงหรือทำในสิ่งที่เป็นเท็จ

```
if (expression 2)
```

```
{
```

```
    statements 3;
```

การทำงานผังเป็นจริง

```
}
```

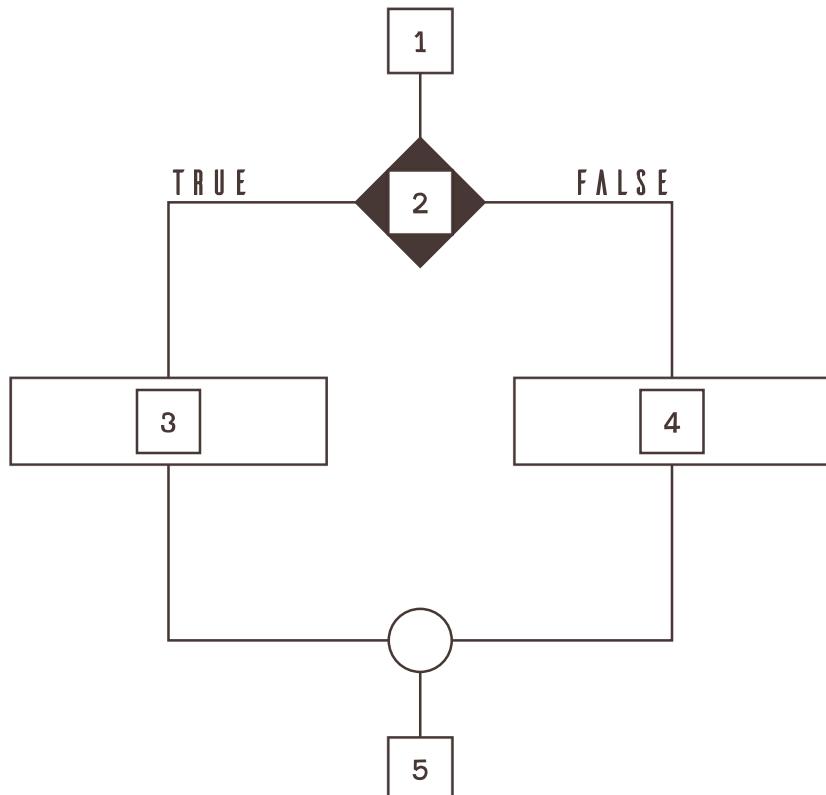
```
else
```

```
{
```

```
    statements 4;
```

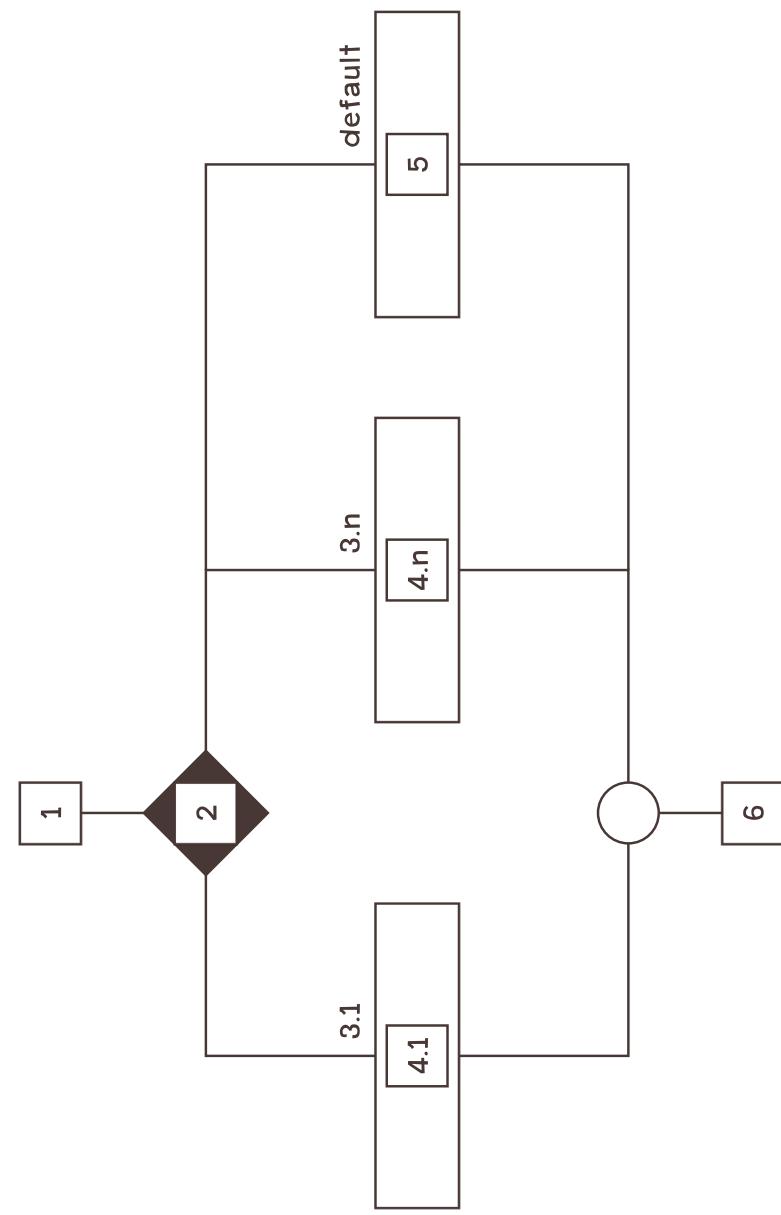
การทำงานผังเป็นเท็จ

```
}
```



Decision Control Statement

การกำหนดเงื่อนไขด้วยการตัดสินใจ

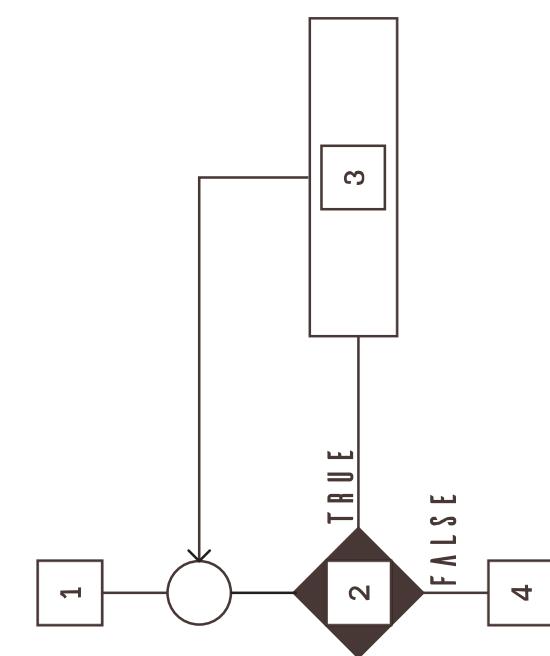


switch - case
เป็นคำสั่งที่เราใช้กำหนดให้โปรแกรมตัดสินใจที่สถานะรุก
เลือกได้ตามกฎกว่า 2 ทาง

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

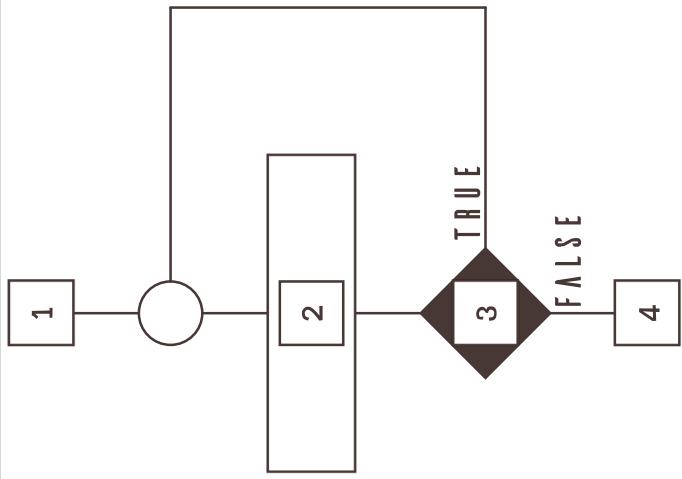
Repetition Control Statement

การกำหนดขั้นตอนการวนซ้ำ



do - while
ແນະຄໍາສຳງວ້າໄດ້ປໍການ 1 ດຽວແລ້ວຄໍປໍຕຽຈສົກປ່ອນໆໆ

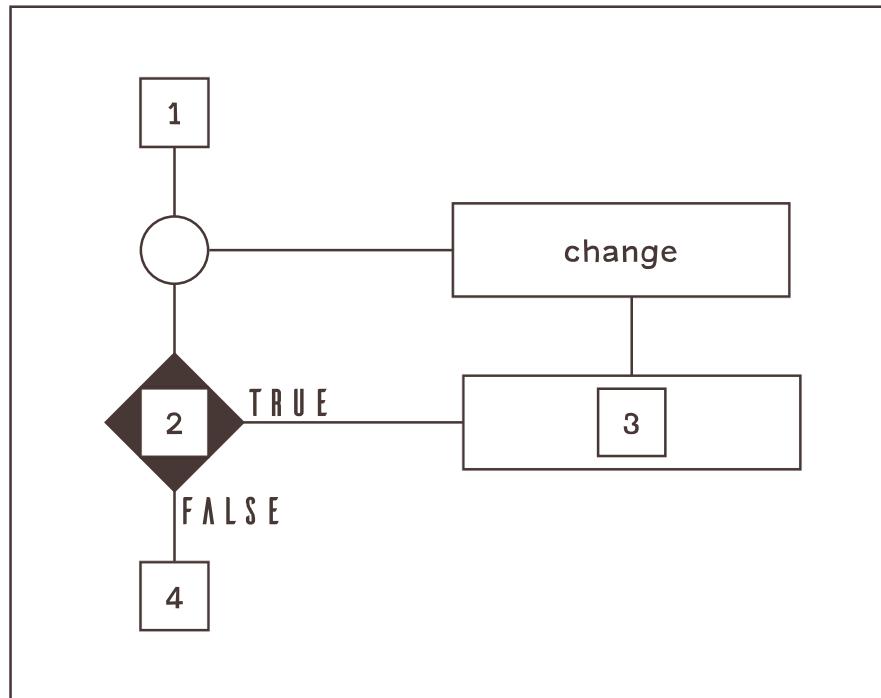
```
do
{
    statements 2;
}
while (expression 3);
```



while - loop
ແນະຄໍາສຳງວ້າໄດ້ປໍຕຽຈສົກເສື່ອນໄປກ່ອນ ຄ້າຈີງຈະກຳການ
while (expression 2)
{
 statements 3;
}

Repetition Control Statement

การทำงานด้วยการวนลูป



for loop

เป็นคำสั่งวนซ้ำด้วยจำนวนรอบที่แน่นอน

```

for (initial; expression 2; change)
{
    เป็นค่าเริ่มต้นของตัวแปรที่จะเปลี่ยน
    ซึ่งจะประกาศตัวแปรตรงนี้หรือก่อนหน้าก็ได้
    statements 3;
}
  
```

more!

คำสั่ง **break** : ออกจากการวนลูปทันที

คำสั่ง **continue** : ข้ามการทำงานที่เหลือและเริ่มลูปใหม่ทันที



โครงสร้างโปรแกรม

**บางส่วนเท่านั้น

```
int main()
{
    เป็นตัวแปรที่สามารถใช้เฉพาะในฟังก์ชันที่ประกาศไว้
    type variable;
    statement; คำสั่ง .
    return 0;
}
```

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

Main Function

Function

*การรับค่ามา (Parameter)
คือ การส่งค่าจากฟังก์ชันหลักที่เรียกใช้งาน
มาสู่ในฟังก์ชันย่อย โดยที่จะส่งมาถึงค่าก็ได้
โดยให้ครบตามจำนวนของ Parameter ตามที่
กำหนดไว้ในฟังก์ชันย่อย

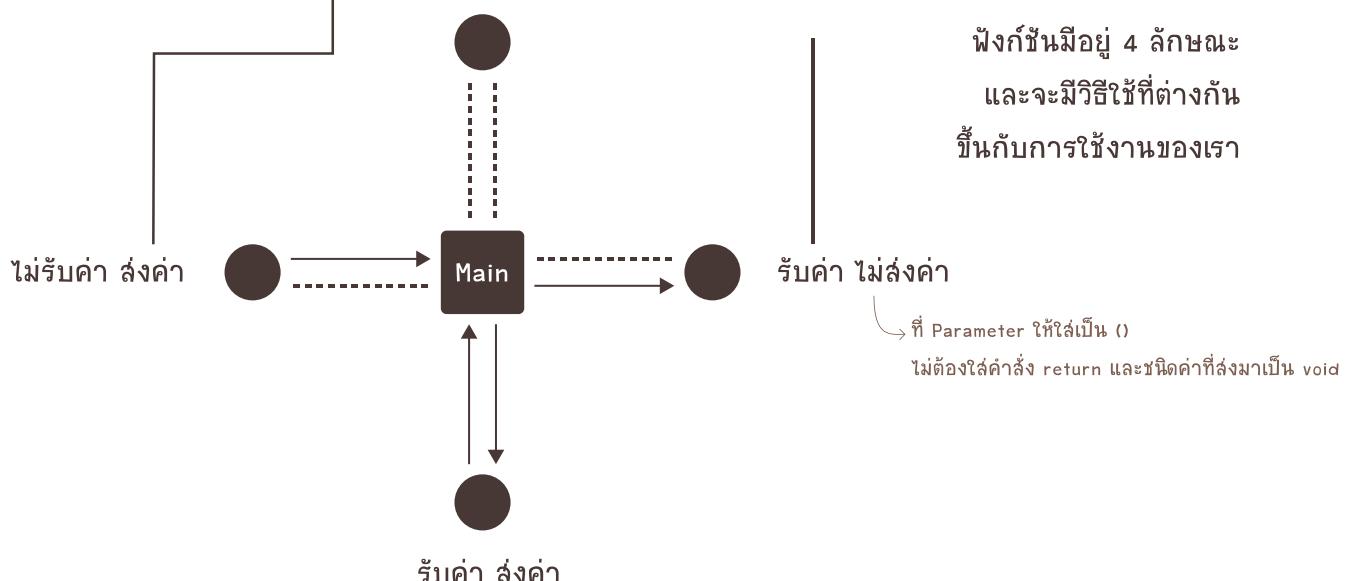
*การส่งค่ากลับ (return)

คือ การส่งค่าที่ผ่านการทำตามค่าสั่งให้
ฟังก์ชันที่เราสร้างขึ้นแล้วต้องการนำค่าลง
กลับมาที่ฟังก์ชันหลักเพื่อนำมาใช้ต่อ

ไม่รับค่า ไม่ส่งค่า

Function ฟังก์ชัน

ฟังก์ชันมีอยู่ 4 ลักษณะ
และจะมีวิธีใช้ที่ต่างกัน
ขึ้นกับการใช้งานของเรา



ที่ Parameter ให้ใส่เป็น ()

ไม่ต้องใส่ค่าสั่ง return และชนิดค่าที่ส่งมาเป็น void

Array

ตัวแปรชื่อเดียวกัน
แต่สามารถเก็บได้หลายค่า

ประกาศตัวแปรอาร์เรย์

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

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



*ถ้าเป็นข้อความจะมี 10 อักต่อท้ายเล่ม
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 ได้

การกำหนดค่าเริ่มต้นให้ตัวแปรของ structure

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

ข้อควรระวัง! ลำดับการกำหนดค่าของตัวแปร ไม่ เช่น นี้ 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

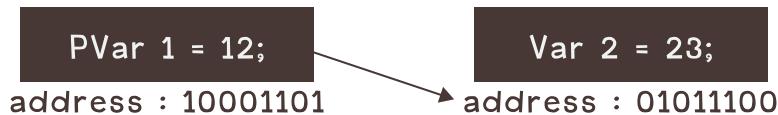
สามารถได้เยี่ยนลื้นๆ ได้ดังนี้

```
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;
```

หากไม่ใส่ * ไว้ข้างหน้า
 จะได้เป็นค่า address ที่
 pointer อ้างอิงไป

หรือ

การเข้าถึงค่าตัวแปรในแต่ละ index

```
PointerName[INDEX];
```

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

มี () เพื่อเป็นการเปลี่ยนตำแหน่ง
 การเขียนของ pointer ไม่ใช่การเพิ่ม
 ค่าที่ pointer เข้าถึงอยู่

