

สารบัญ

Flowchart.....	1
ผังงาน (Flowchart).....	1
การเขียนผังงานที่ดี.....	1
ประโยชน์ของผังงาน.....	1
ข้อจำกัดของผังงาน.....	1
สัญลักษณ์ที่ใช้ในการเขียนผังงาน.....	1
รูปแบบของผังงาน.....	2
Pseudo Code	3
ประวัติความเป็นมา	6
จุดเด่นของภาษา C	6
ข้อแตกต่างระหว่าง ภาษา C vs. C++	6
โครงสร้างโปรแกรม	7
1. Header File.....	7
2. Function.....	7
3. ส่วนตัวโปรแกรม.....	7
ตัวแปร	8
การตั้งชื่อ	8
การประกาศตัวแปร.....	8
ชนิดของข้อมูล (Data Type)	9
ชนิดข้อมูลแบบจำนวนเต็ม (Integer Type).....	9
ชนิดข้อมูลแบบจำนวนจริง (Real floating-point type)	10
ชนิดข้อมูลแบบตัวอักษร (Character type)	10
ชนิดข้อมูลแบบสายอักษร (String type)	10
Null Character และ Empty String.....	11
ประเภทตัวแปร.....	11
การกำหนดค่าให้ตัวแปร	12
ค่าคงที่ (Constant)	12
ระบุค่าโดยตรง (Literal Constants).....	12
นิยามโดย #define (Defined Constants)	13
เก็บไว้ในตัวแปร (Memory Constants)	13
การแปลงชนิดของข้อมูล (Data Type Conversion)	13
Implicit Type Conversion	13
Explicit Type Conversion (Casting)	13
การแสดงผลทางหน้าจอ (Display Data)	14
1. ใช้คำสั่ง printf.....	14
การแสดงผลของข้อความ.....	14
การแสดงผลของค่าที่เก็บในตัวแปร	14
รหัสรูปแบบการแสดงผลข้อความ และการจัดรูปแบบแสดงผลข้อมูลของคำสั่ง printf()	16
การจัดพื้นที่ และการกำหนดการแสดงผลข้อมูลของคำสั่ง printf()	16
2. ใช้คำสั่ง putchar() สำหรับแสดงแบบอักขระตัวเดียว (Character)	18
3. ใช้คำสั่ง puts() สำหรับแสดงข้อมูลแบบสายอักษร (String)	18
ตัวดำเนินการ	18
ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operator)	18

ตัวดำเนินการกำหนดค่า (Assignment Operator)	19
ตัวดำเนินการยูนารี (Unary Operator)	19
ตัวดำเนินการเปรียบเทียบ (Comparison Operator)	20
ตัวดำเนินการตรรกะ (Logical Operator)	20
ตัวดำเนินการแบบมีเงื่อนไข (Conditional Operator)	21
ตัวดำเนินการบอกขนาดชนิดข้อมูล (Sizeof Operator)	21
ตัวดำเนินการระดับบิต (Bitwise Operator)	21
Bitwise AND (&), Bitwise OR (), Bitwise Exclusive OR/XOR (^)	21
Bitwise Shift Right (>>)	22
Bitwise Shift Left (<<)	22
Bitwise One's Complement (~)	22
ลำดับความสำคัญของตัวดำเนินการ	23
การรับข้อมูล (Input Data)	23
1. การรับข้อมูลชนิดอักขระที่ละตัวจากคีย์บอร์ดด้วยคำสั่ง getch() และ getchar	23
2. การรับข้อมูลชนิดข้อความจากคีย์บอร์ดด้วยคำสั่ง gets()	24
3. การรับข้อมูลทุกชนิดจากคีย์บอร์ดด้วยคำสั่ง scanf()	24
การกำหนดลำดับการรับข้อมูลของคำสั่ง scanf()	25
การกำหนดจำนวนตัวอักขระในการรับค่าของข้อมูลชนิดข้อความของคำสั่ง scanf()	26
การนับจำนวนตัวอักขระที่รับค่าของข้อมูลด้วยคำสั่ง scanf()	26
Regular Expression.....	26
ควบคุมทิศทางการทำงานโปรแกรมด้วยคำสั่งตัดสินใจ (Decision Control Statement).....	27
คำสั่งตัดสินใจแบบเลือกทำ หรือไม่ทำด้วยคำสั่ง if	27
คำสั่งตัดสินใจแบบสองทางเลือกด้วยคำสั่ง if...else	27
คำสั่งตัดสินใจแบบหลายทางเลือกด้วยคำสั่ง if...else if	27
คำสั่งตัดสินใจแบบหลายทางเลือกด้วยคำสั่ง switch-case	28
ควบคุมการทำงานของโปรแกรมด้วยคำสั่งวนลูป (Repetition Control Statement).....	29
วนลูปการทำงานด้วยจำนวนรอบที่แน่นอนด้วยคำสั่ง for	29
วนลูปการทำงานเมื่อเงื่อนไขเป็นจริงด้วยคำสั่ง while	30
วนลูปการทำงานอย่างน้อย 1 ครั้งด้วยคำสั่ง do...while	30
คำสั่ง break และ continue	30
อธิบายลูป for อย่างละเอียด	31
Array	32
ตัวแปรอาร์เรย์ 1 มิติ (One Dimension Array)	32
ตัวแปรอาร์เรย์ 2 มิติ (2-Dimension Array)	32
ตัวแปรอาร์เรย์ 3 มิติ (3-Dimension Array)	32
การประกาศตัวแปรอาร์เรย์	33
การกำหนดค่าข้อมูลให้ตัวแปรอาร์เรย์	33
การอ้างถึงข้อมูลในตัวแปรอาร์เรย์	34
ฟังก์ชัน (Function)	35
ฟังก์ชันที่สร้างขึ้นเอง (User-Defined Functions)	35
ฟังก์ชันที่ไม่มีการคืนค่ากลับและไม่มีการรับค่าพารามิเตอร์ (Void Functions with No Parameters)	35
ฟังก์ชันที่ไม่มีการคืนค่ากลับ แต่มีการรับค่าพารามิเตอร์ (Void Functions with Parameters)	36
ฟังก์ชันที่มีการคืนค่ากลับ แต่ไม่มีการรับค่าพารามิเตอร์ (Function Return value with No Parameters)	36
ฟังก์ชันที่มีการคืนค่ากลับ และมีการรับค่าพารามิเตอร์ (Function Return Value with Parameters)	37
พอยเตอร์ (Pointer)	39

ภาคผนวก	A
คำศัพท์น่ารู้	A
หมวดระบบคอมพิวเตอร์	A
หมวดโครงสร้างภาษา	A
หมวดการแสดงผล และรับข้อมูล	B
หมวดนิพจน์ทางคณิตศาสตร์	B
หมวดฟังก์ชันของภาษา C	B
หมวดคำสั่งเงื่อนไข	C
หมวดคำสั่งวนลูป	C
ASCII TABLE	D
ผังอักษรและสกีที่ไม่สามารถแสดงผล	D
ผังอักษรและสกีที่สามารถแสดงผล	D

วิัฒนาการ

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

CE57

เริ่มทำหนังสือเนื้อหาจริงจัง เพราะในค่าย CE BOOST UP #6 ที่รุ่นพากพีเข้ากันอะ หนังสือที่ปรินต์มาแจก ไม่สีเรียกว่าซีทีก่าว พี่ ๆ เขายังไม่ได้ทำเป็นหนังสืออย่างจริงจังอะแหละ หลาย ๆ คนเลยได้มาลงทะเบียนมากอง ๆ สุดท้ายพากพีก็ต้องไปหา Google เอาอยู่ดี พี่เลยตัดสินใจสร้างสิ่ง ๆ นี้ขึ้นมาครับ! ซึ่ง... มันก็ยังไม่เป็นหนังสือหรือครับ มีแค่เนื้อหา (พอยเตอร์ การเขียนไฟล์ก็ยังไม่มี) หน้าปก สารบัญ ภาคผนวกก็ยังไม่มี แต่จะผิดยังเบอะอึก สารภาพด้วยครับมีเวลาทำแค่ 1 สัปดาห์ 5555+ ขออภัยมาที่นี่อะ _/_ อย่ากรุ้วความ

เป็นมาละเอียดกว่านี้หรือ ตามสิ ไม่บอกหรอก อ้อ และพี่ขอส่งท้ายจากลากันไปด้วยหนังสือส่วนสุดท้ายดันครับ

Niranam...

พอยเตอร์ (Pointer)

พอกะอน้อง.... แค่นี้ก็จะอ้วกแตกตายแล้ว 5555 ยังเหลือเรื่องอ่าน-เขียนไฟล์นะถ้าอยากรีบศึกษาเพิ่มเติม ถ้านั้งสือเล่มนี้พิมพ์สด หรือข้อมูลผิดพลาดตรงส่วนไหนพี่ ๆ ก็ขออภัยมา ณ ที่นี่อะ

มีตรงไหนสงสัยถามพี่ ๆ ได้เนอะ อยากทำโจทย์ก็ทักมาขอได้เหมือนกันมี source ให้ไปฝึก ขอให้โชค A สนุกกัน การอ่านหนังสือเล่มนี้ครับ :D

ACADEMIC TEAM - CE BOOST UP #7

~ 39 of 39 ~

bit.ly/bookceboostup

CE58

Flowchart

ผังงาน (Flowchart)

ผังงาน (Flowchart) คือ รูปแบบหรือสัญลักษณ์ ที่ใช้เขียนแทนคำอธิบาย ข้อความหรือค่าพุดที่ใช้ในอัลกอริทึม เนื่องจาก การที่จะเข้าใจขั้นตอนได้ง่ายและตรงกันนั้น หากใช้ค่าพุดหรือข้อความจะทำได้ยากกว่าการใช้รูปภาพหรือ สัญลักษณ์ ที่เป็นสากลและคนทั่วโลกสามารถเข้าใจได้ง่าย

การเขียนผังงานที่ดี

1. ใช้สัญลักษณ์ตามที่กำหนดไว้
2. ใช้ลูกศรแสดงทิศทางการทำงานของโปรแกรมจากบนลงล่าง หรือจากซ้ายไปขวา
3. คำอธิบายในภาพควรสั้นกระัดกระสินค้า และเข้าใจง่าย
4. ทุกแผนภาพต้องมีลูกศรแสดงทิศทางเข้า – ออก
5. ไม่ควรพยายามเขียนผังงานที่อยู่ไกลมากๆ ควรใช้จุดเชื่อมต่อแทน
6. ผังงานควรมีการทดสอบความถูกต้องของการทำงานก่อนนำไปเขียนโปรแกรม

ประโยชน์ของผังงาน

1. ทำให้เข้าใจและแยกแยะปัญหาต่างๆ ได้ง่ายขึ้น
2. ผู้เขียนโปรแกรมสามารถเห็นลำดับการทำงาน รู้ว่าสิ่งใดควรทำก่อน สิ่งใดควรทำหลัง
3. สามารถหาข้อผิดพลาดของโปรแกรมได้ง่าย
4. ทำให้ผู้อื่นเข้าใจการทำงานได้ง่ายกว่าการดูจาก source code
5. ผู้อื่นสามารถเรียนรู้ และนำไปเขียนเป็นภาษาที่ตนเองถนัดได้ง่าย

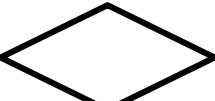
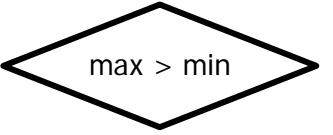
ข้อจำกัดของผังงาน

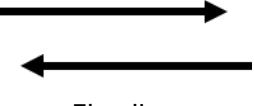
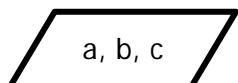
คนที่เขียนโปรแกรมบางคนไม่นิยมเขียนผังงานก่อนเขียนโปรแกรม เพราะ เห็นว่าเสียเวลา นอกจานนั้นยังมี ข้อจำกัดอีก คือ

1. ผังงานเป็นการสื่อความหมายระหว่างบุคคลกับบุคคลมากกว่าที่จะสื่อความหมายระหว่างบุคคลกับเครื่อง เพราะ ผังงานไม่เขียนกับภาษาคอมพิวเตอร์ภาษาใดภาษาหนึ่ง ทำให้เครื่องไม่สามารถรับและเข้าใจได้ว่าในผังงานนั้นต้องการให้ ทำอะไร
2. ในบางครั้ง เมื่อพิจารณาจากผังงาน จะไม่สามารถทราบได้ว่า ขั้นตอนการทำงานใดสำคัญกว่ากัน เพราะทุกๆ ขั้นตอน จะใช้รูปภาพหรือสัญลักษณ์ในลักษณะเดียวกัน
3. การเขียนผังงานเป็นการสิ้นเปลือง เพราะจะต้องใช้กระดาษและอุปกรณ์อื่นๆ เพื่อประกอบการเขียนภาพ

สัญลักษณ์ที่ใช้ในการเขียนผังงาน

เป็นเครื่องมือ (Tools) ที่ใช้อธิบายรายละเอียดการทำงานตามขั้นตอนการทำงาน (Algorithm) แทนคำสั่งการทำงานของโปรแกรม โดยใช้สัญลักษณ์ (Symbol) แทนคำสั่ง ที่นิยมใช้มีดังนี้

สัญลักษณ์แสดงขั้นตอนการทำงาน		
สัญลักษณ์	ความหมาย	ตัวอย่าง
 Start/Stop	การกำหนดจุดเริ่มต้นของการทำงาน และแสดงจุดสิ้นสุด ของการทำงาน	 
 Process	การแสดงรายละเอียดของการทำงานและกระบวนการทำงาน	$\text{Total} = 0$ $\text{Total} = \text{Total} + (\text{a} * \text{c})$
 Decision	การแสดงรายละเอียดการเปรียบเทียบเงื่อนไขต่างๆ ใช้ใน ขั้นตอนที่มีการตัดสินใจว่า ใช้ หรือ ไม่ใช้	

	การแสดงทิศทางความสัมพันธ์ของการทำงานในระบบงานหรือลำดับงาน	
	การกำหนดจุดอ้างอิงในการเชื่อมต่อ ในหน้ากระดาษเดียวกันของการเขียนผังงานโครงสร้าง (Structured Flowchart)	
สัญลักษณ์แสดงการรับ-ส่งข้อมูล		
สัญลักษณ์	ความหมาย	ตัวอย่าง
	<p><u>Read(รับค่า)</u> การรับค่าข้อมูลหรืออ่านข้อมูลเข้ามาโดยไม่ระบุอุปกรณ์รับข้อมูล (Input Device) โดยอาจรับค่าข้อมูลมาจาก คีย์บอร์ด หรือจากแฟ้มข้อมูลก็ได้</p> <p><u>Write(แสดงผล)</u> การแสดงรายละเอียดข้อมูลหรือแสดงผลลัพธ์ของการประมวลผล โดยไม่ระบุอุปกรณ์การแสดงผล (Output Device) โดยอาจแสดงผลผ่านจอภาพหรือเครื่องพิมพ์ (Printer) ก็ได้</p>	  

รูปแบบของผังงาน

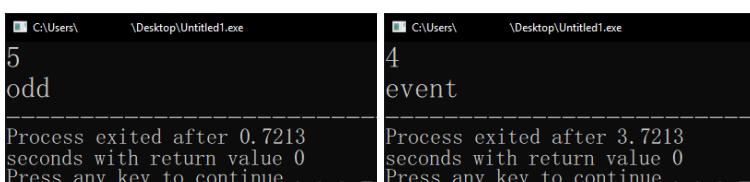
ผังงาน มีรูปแบบที่จำกัดอยู่ 3 รูปแบบ คือ

1. รูปแบบที่มีการกำหนดเงื่อนไขหรือให้เลือก (Decision Structure) : รูปแบบที่มีการสร้างเงื่อนไขเพื่อเลือกทำงาน โดยจะทำงานตามทางที่เลือกหากตรงเงื่อนไข

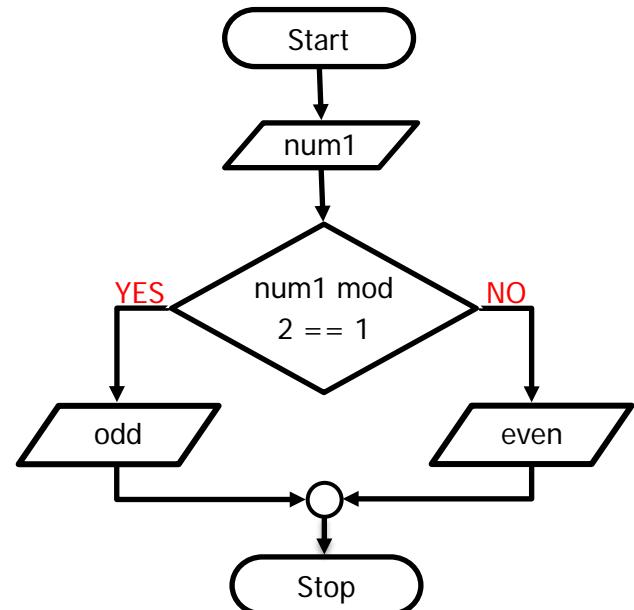
```

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

ภาพตัวอย่างการเขียนโค้ด



ภาพตัวอย่างผลการรันโค้ด



2. รูปแบบเรียงลำดับ (Sequence Structure) : การทำงานแบบเรียงลำดับตั้งแต่ต้นจนจบ เป็นรูปแบบง่ายๆ ไม่มีการเปรียบเทียบใดๆ 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 }
10
```

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

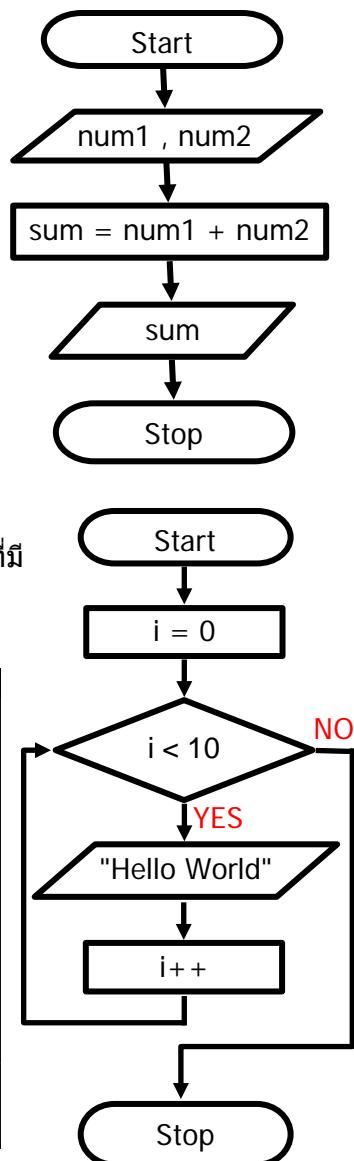
3. รูปแบบที่มีการทำงานวนรอบ หรือ loop (Iteration Structure) : รูปแบบที่มีการทำงานซ้ำค่าสั่งเดิม โดยการทำงานนี้จะขึ้นอยู่กับเงื่อนไขที่กำหนดให้

```
1 #include<stdio.h>
2
3 main()
4 {
5     int num1;
6     for(int i = 0; i < 10; i++){
7         printf("Hello World\n");
8     }
9 }
```

ภาพตัวอย่างการเขียนโค้ด

ภาพตัวอย่างผลการรันโค้ด >

```
C:\Users\ \Desktop\Untitled1.exe
Hello World
Process exited after 0.01836
seconds with return value 0
Press any key to continue . . .
```



ข้อสังเกต : เส้นทุกเส้นต้อง “เชื่อมต่อกัน และมีทิศทางเสมอ” ดังนั้นทุกเส้นจะมีลูกศร และทุกเส้นจะต้องเชื่อมติดกันล่อง สัญลักษณ์

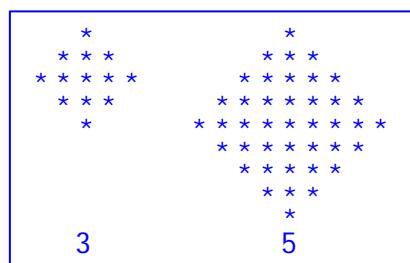
ข้อสังเกต : ลูกศรควรจะชี้ไปยังกล่องได้เพียง “เส้นเดียว” ดังนั้นหากมีมากกว่า 1 เส้นอาจจะใช้ IN-Page Connector

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

Pseudo Code

คือ ภาษาที่ใช้จำลองภาษาโปรแกรม โดยใช้ภาษาที่ไม่ใช้ภาษาโปรแกรมซึ่งไม่จำกัดภาษา และมีขั้นตอนที่แน่นอนจะทัดรัด เป็นเพียงการจำลองคำสั่งจริงแบบย่อ ๆ ตามกระบวนการของโปรแกรมที่ต้องการสร้าง ซึ่งจะช่วยให้เห็นโครงสร้างโปรแกรมชัดเจนขึ้น สามารถหาจุดผิดของโปรแกรมได้ง่ายเมื่อมีปัญหา และเป็นตัวกำหนดงานเขียนโปรแกรมให้ทำงานได้ตามต้องการเมื่อการเขียนตามโค้ดเที่ยมที่สร้างไว้ หลักการเขียน Pseudo code ที่ดีควรให้ในหนึ่งบรรทัด ควรมีคำสั่งเดียว และย่อหน้าเพื่อย่อคำสั่งให้ชัดเจน Pseudo code แบ่งออกเป็น 3 ส่วนหลักๆ

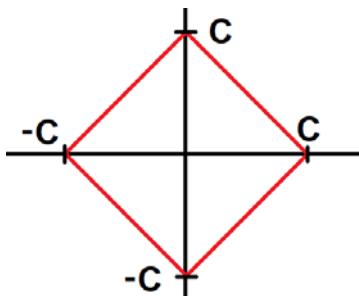
1. Draft code : เป็นการเขียนอย่างหยาบๆ ที่สุด ไม่ลงรายละเอียดอะไรมาก บอกว่ากำลังทำอะไรเป็นลำดับไป
2. Detailed code : เป็นส่วนที่จะนำ draft code มาแต่รายละเอียดให้ชัดเจนว่ามีการทำงานตรงไหนอย่างไร detailed code จะทำกี่รอบก็ได้จนกว่าจะได้การทำงานที่ชัดเจนที่สุดเพื่อแปลงเป็น simple code ต่อไป
3. Simple code : เป็นส่วนที่มีความใกล้เคียงภาษาโปรแกรมมากที่สุดและรายละเอียดครบถ้วนที่สุด พร้อมที่จะนำไปแปลงเป็นภาษาโปรแกรมได้ทันที

ตัวอย่าง 1 Cr. พีตันไม์ #CE57Draft Pseudo Code

- รับค่าตัวแปร N เพื่อสร้างรูป
- สร้างรูป * จากพื้นที่ภายในกราฟ (x, y) โดยมีจุดยอดตัดแกน x หรือแกน y ที่ระยะห่างจากจุดกำเนิด $N - 1$
- กำหนดเงื่อนไขที่แสดง * และแสดงช่องว่าง

Detailed Pseudo Code

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

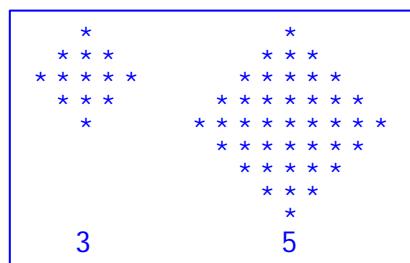


เงื่อนไขพื้นที่ภายในกราฟ คือ
 $|y| \leq |x| - C$ แสดง *

$|y| > |x| - C$ แสดง ช่องว่าง

Simple Pseudo Code

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

ตัวอย่าง 2 Cr. พีเกม #CE57Draft Pseudo Code

- รับค่าใส่ตัวแปร n
- แสดงผล Δ บน
- แสดงผลบรรทัดกลาง
- แสดงผล Δ ล่าง

Detailed Pseudo Code

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

Simple Pseudo Code

- 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

Detailed Pseudo Code

- แสดงผล * จำนวน $2n-1$ ตัวแล้วขึ้นบรรทัดใหม่

Simple Pseudo Code

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

Detailed Pseudo Code

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

Simple Pseudo Code

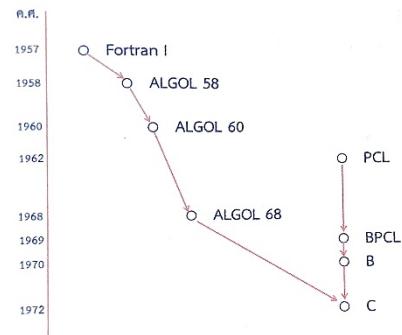
- รับค่าตัวแปร (n)
- ถ้า $i \leq n-1$ ทำซ้ำ :
 - กำหนด $j = 1$
 - ถ้า $j \leq i$ ทำซ้ำ :
 - แสดงผลช่องว่าง
 - เพิ่มค่า j ขึ้น 1
 - กำหนด $j = 1$
 - ถ้า $j \leq 2(n-i)-1$ ทำซ้ำ :
 - แสดงผล *
 - เพิ่มค่า j ขึ้น 1
 - แสดงผลขึ้นบรรทัดใหม่
 - เพิ่มค่า i ขึ้น 1

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 ที่สามารถนำไปรับใช้กับคอมพิวเตอร์ในรูปแบบต่าง 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 การทำงานไม่ขึ้นกับฮาร์ดแวร์ ทำให้สามารถนำไปใช้ใน CPU รุ่นต่าง ๆ ได้
- เป็นภาษาระดับสูงที่ทำงานเหมือนภาษาระดับต่ำ สามารถทำงานแทนภาษา Assembly ได้
- มีแนวความคิดในการพัฒนาแบบ “โปรแกรมเชิงโครงสร้าง” (Structure Programming) ทำให้ภาษา C เป็นภาษาที่เหมาะสมสำหรับนำมาพัฒนาระบบ
- ความสามารถของคอมไพล์เลอร์ (Compiler) ในภาษา C มีประสิทธิภาพสูง ทำงานได้รวดเร็ว โดยใช้รหัสออบเจกต์ (Object) ที่สั้น ทำให้เหมาะสมสำหรับการทำงานที่รวดเร็ว

ข้อควรรู้ : คอมไпал์เลอร์ (Compiler) จะทำหน้าที่แปลโคดที่เขียนทั้งหมดให้เป็นภาษาเครื่อง พร้อมกันนี้จะทำหน้าที่ตรวจสอบไวยากรณ์ของภาษา หากมีข้อผิดพลาดก็จะแจ้งให้เห็นถึงข้อผิดพลาด เมื่อกระบวนการทำงานของคอมไпал์เลอร์เสร็จสมบูรณ์ โปรแกรมจึงจะสามารถใช้งานได้ ซึ่งแตกต่างกับหลักการทำงานของ อินเทอร์พريเตอร์ (Interpreter) ซึ่งการทำงานจะแปลงภาษาโปรแกรม และประมวลผลคำสั่งทีละคำสั่ง

ข้อแตกต่างระหว่าง ภาษา C vs. C++

ทำไมต้อง C ก่อน!

ถ้าเข้าใจโครงสร้างการเขียน C หากไปเขียนภาษาอื่น จะเข้าใจ หรือ Optimise ได้ง่ายขึ้น เพราะ C ไม่ค่อยมีตัวช่วย หรือ Library มากเท่าภาษาอื่นที่ง่ายกว่า เช่น ภาษา Python ภาษา C จึงเหมาะสมสำหรับฝึก Logic การทำงานตั้งแต่ต้นจนจบ ... เขียนโปรแกรมเป็นไปไข่డีเจียน โปรแกรมได้ แนวคิดต้องได้ด้วย ...

เดิมโปรแกรมเมอร์ภาษา C จะสร้างโปรแกรมโดยคิดตาม step 1, 2, 3, ... ซึ่งก็มีข้อดีสำหรับโปรแกรมที่ไม่ซับซ้อนมากสามารถเขียนได้อย่างรวดเร็ว แต่นาน ๆ ไปโปรแกรมเริ่มจะมีความสามารถเพิ่มขึ้น สลับซับซ้อนกว่าเดิมซึ่งในระหว่างพัฒนา หรือแก้ไขปัญหา อาจสับสนได้ว่าเราจะต้องไปแก้ไขตรงจุดไหน จึงอาจทำให้เกิด Bug ได้ง่าย

จึงมีแนวคิดการเขียนโปรแกรมแบบ OOP ขึ้นมาคือให้แบ่งโปรแกรมออกเป็นชิ้น ๆ หรือเรียกว่า วัตถุ (Object) และพัฒนาแต่ละชิ้นมา จากนั้น จึงนำมาประกอบกันทีหลัง วิธีการนี้ทำให้ลดความสับสนเวลาพัฒนาหรือแก้ไข สามารถนำร่วมกันที่หลัง วิธีการนี้ทำให้ลดความสับสนเวลาพัฒนาหรือแก้ไขในส่วนเล็ก ๆ ขึ้นเดียวได้ ไม่ต้องแก้ไขทั้งโปรแกรม และนอกจากนี้เรายังสามารถนำร่วมกันที่หลัง หรือส่วนที่แบ่งเป็นชิ้นๆ นำกลับมาใช้ใหม่ หรือนำไปใช้กับโปรแกรมอื่น ๆ ที่เราต้องการโดยไม่ต้องเสียเวลาเขียนใหม่ ประหยัดเวลาได้อีกมาก ถ้าจะเปรียบเหมือนกับการสร้างรถยนต์ แทนที่จะหล่อรถยนต์ทั้งคันที่เดียว (ภาษาซี) เวลาเสียก์ต้องทำใหม่หมด หรือส่งไปปั้นใหม่ทั้งคัน แต่ถ้าใช้แนวคิดแบบ OOP แต่ละชิ้นทำขึ้นมาต่างหาก และจึงนำมาประกอบกันทีหลัง เวลาเมื่อยกมาแล้วนำชิ้นส่วนใหม่มาเปลี่ยนได้ง่ายกว่า

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

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

โครงสร้างโปรแกรม

```

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

```

C:\Users\ \Desktop\Untitled1.exe
Process exited after 0.02409 seconds with return value 0
Press any key to continue . . .



1. Header File

เป็นส่วนที่เรียกใช้ไลบรารี ซึ่งจะขึ้นต้นด้วยเครื่องหมาย # เสมอ โดยเป็นการนำโค้ดในไฟล์ヘッเดอร์ (.h) นั้น ๆ มาใส่ไว้ที่หัวของโปรแกรมที่เรากำลังเขียน คอมไพล์เวอร์จะทำในส่วนนี้เป็นส่วนแรก สามารถเขียนได้ 2 แบบ คือ

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

ถ้าใช้เครื่องหมาย "..." คอมไпал์เวอร์จะหาไฟล์จากโฟลเดอร์ที่เก็บ Source code ก่อนแล้วจึงหาไฟล์ที่เก็บไว้กับตัวคอมไпал์เวอร์ แต่ถ้าใช้ <...> จะหาจากไฟล์ที่เก็บไว้กับตัวคอมไпал์เวอร์เพียงที่เดียวเท่านั้น

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

2. Function

```

1 void function()
2 {
3
4 }
5
6 main() = 6 int main()
7 { 7 {
8
9 } 8
9 } return 0;

```

C:\Users\ \Desktop\Untitled1.exe
Process exited after 0.02092 seconds with return value 0
Press any key to continue . . .

เป็นส่วนการทำงานของโปรแกรม ซึ่งบังคับให้มืออย่างน้อย 1 พังก์ชัน คือ main() ซึ่งเป็นพังก์ชันเริ่มการทำงานของโปรแกรม(ประมวลผลที่พังก์ชันนี้เป็นพังก์ชันแรก) ขอบเขตของพังก์ชันจะเริ่มต้นด้วยเครื่องหมาย { และสิ้นสุดด้วยเครื่องหมาย } พังก์ชันส่วนมากจะมีส่วนส่งค่ากลับเมื่อทำงานจบพังก์ชัน จะเห็นว่าเมื่อพังก์ชันจะส่งค่ากลับจะต้องมีคำสั่ง return

การประกาศไว้ว่าชื่อพังก์ชันว่าจะส่งค่าประเภทไหนกลับ และใช้คำสั่ง return ในการส่งค่ากลับมา เช่น void (ไม่ส่งค่ากลับทำให้ในพังก์ชันไม่ต้องมีคำสั่ง return) , int (ส่งค่าจำนวนเต็มกลับมา) , float (ส่งค่าทางศนิยมกลับมา) , char (ส่งค่าตัวอักษรตัวเดียวกลับมา) เป็นต้น

ข้อสังเกต : พังก์ชัน main ถ้าไม่ไดระบุค่าส่งกลับจะมีค่า Default เป็นจำนวนเต็มที่มีค่าเท่ากับ 0 ถ้าโปรแกรมทำงานไม่เกิดข้อผิดพลาด

3. ส่วนตัวโปรแกรม

จะอยู่ภายใต้พังก์ชันซึ่งประกอบด้วย 2 ส่วนคือ

- 3.1 คำสั่ง คือ คำที่กำหนดไว้ให้โปรแกรมทำงานตามที่ต้องการ ส่วนมากจะจบด้วยเครื่องหมาย ;
- 3.2 ตัวแปร คือ สัญลักษณ์ใช้แทนค่าต่าง ๆ ซึ่งค่าที่แทนสามารถเปลี่ยนค่าอื่นได้ โดยค่านั้นจะเป็นตัวเลข ตัวหนัง หรือ object ก็ได้

```
1 main()
2 {
3
4 }
```

=

```
1 main(){}
2
3 }
4
```

=

```
1 main(){}
2
3
4
```

```
5
6 if(true){
7     statement;
8 }
```

=

```
5
6 if(true)    {statement;}
7
8
```

ข้อควรรู้ : สามารถใช้คอมเม้นต์เพื่อทำให้คำสั่งบางคำสั่งที่เราไม่ต้องการ
กลับเป็นคำอธิบาย เพื่อไม่ให้ส่วนนั้นทำงาน สามารถทำได้ 2 วิธีคือ

1. ใช้ // เพื่อทำการคอมเม้นต์ข้อความด้านหลังสัญลักษณ์นี้ โดยใช้ได้
แค่ภายในบรรทัดนั้น

2. ใช้ /* ... */ เพื่อทำการคอมเม้นต์ข้อความที่อยู่ด้านใน โดยสามารถ
ใช้ได้หลายบรรทัด

ข้อควรระวัง : หากใช้ /* ... */ ถ้าเจอกับการปิดคอมเม้นต์ตัวแรกเมื่อไร
ข้อความหลังจากนั้นจะไม่ถูกคอมเม้นต์ เช่น /*/*/* COMMENT *//*/ตัว
*/ สีน้ำเงินจะไม่ถูกคอมเม้นต์ และคอมไไฟเลอร์จะเข้าใจว่าเป็นคำสั่ง ทำ
ให้โปรแกรมเราเกิด Syntax Error ทำให้คอมไไฟล์ไม่ผ่านได้

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

ตัวแปร

คือ สัญลักษณ์ที่ใช้แทนตำแหน่งบนหน่วยความจำ ใช้ในการเก็บค่าต่าง ๆ ในโปรแกรม ที่ต้องใช้สัญลักษณ์แทน
เนื่องจากการอ้างถึงตำแหน่งบนหน่วยความจำนั้นยากต่อการทำความเข้าใจ และการพัฒนาโปรแกรม โดยเราสามารถ
เรียกใช้ตัวแปรเพื่อใช้ในการคำนวณหรือเพื่อกำกับค่าได้ ซึ่งกฎการตั้งชื่อให้กับตัวแปรมีดังนี้

การตั้งชื่อ

- ชื่อประกอบด้วยตัวอักษรพิเศษไม่ได้ เช่น & @ # & เป็นต้น
- ภายใต้ชื่อประกอบด้วยช่องว่าง หรือแท็บไม่ได้
- สามารถประกอบด้วยตัวอักษร ตัวเลข และเครื่องหมาย Underscore (_) แต่ตัวอักษรตัวแรกไม่สามารถขึ้นต้น
ด้วยตัวเลขได้ โดยในชื่อสามารถมีตัวเลขประกอบได้ เช่น _Number, x1, _708, pRayut44 เป็นต้น
- ในภาษา C/C++ เป็นแบบ Case-Sensitive คือ ตัวอักษรตัวใหญ่และตัวอักษรตัวเล็กถือเป็นคนละตัวกัน เช่น
pom, Pom, PoM, POM จะถือว่าชื่อห้องสีตัวนี้เป็นคนละตัวแปรกัน และคอมไไฟเลอร์บางตัวไม่สามารถตั้งชื่อตัวแปรได้เกิน
31 ตัวอักษร
- ตั้งชื่อตัวแปรช้ากันไม่ได้ ไม่เช่นนั้นคอมไไฟเลอร์จะไม่รู้ว่าตัวแปรไหนเป็นตัวไหน เช่นเดียวกันกับคำส่วน
- ชื่อที่ตั้งขึ้นจะต้องไม่ช้ากันคำส่วน (Reserved Word) ภายในโปรแกรม ซึ่งมีดังนี้

หลักการตั้งชื่อตัวแปร และฟังก์ชันให้เป็น
หลักสากล : ตัวแรกตัวเล็ก ค่าต่อ กัน(ถ้ามี)
ชื่อต้นตัวใหญ่ เช่น book, numberOfBook()
getNumberOfBook() 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

ก่อนที่จะตั้งชื่อให้ตัวแปร จะต้องทำการกำหนดชนิดของตัวแปร เพื่อให้โปรแกรมได้สำรองพื้นที่ในหน่วยความจำ
เพื่อกำกับข้อมูลตามที่เราต้องการ โดยสามารถทำได้ดังนี้

||| type varName

โดยที่	type	เป็นชนิดของข้อมูล	เช่น	int x;
	varName	เป็นชื่อตัวแปร		char alphabet;

ชื่อตัวแปร หน่วยความจำ	ชื่อตัวแปร หน่วยความจำ
num	5
ch	M
str	J a d e S P K

ข้อควรรู้ : สามารถกำหนดค่าเริ่มต้นเข้าไปตั้งแต่ตอนประกาศตัวแปรได้เลย เช่น

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

เมื่อโปรแกรมทำงานถึงตรงที่ประกาศตัวแปรไว้ โปรแกรมจะทำการจดพื้นที่ในหน่วยความจำ ถ้ามีการกำหนดค่าให้ตัวแปร ค่าเหล่านั้นก็จะถูกเก็บลงในหน่วยความจำตามตำแหน่งที่ได้จ้องไว้ ดังรูปทางด้านซ้าย

ข้อสังเกต : ตัวแปร ch ค่าที่เก็บลงหน่วยความจำ(กล่อง) จริง ๆ แล้วไม่ใช่ M แต่เป็นตัวเลขรหัสแอกซ์ของ M ซึ่งจะได้ศึกษาในหน้าต่อ ๆ ไป

ชนิดของข้อมูล (Data Type)

ข้อมูลแต่ละชนิดมีขอบเขตความสามารถในการเก็บข้อมูลต่างกัน บางชนิดสามารถเก็บข้อมูลได้มากก็จะใช้หน่วยความจำมากตามไปด้วย จึงต้องคำนึงถึงความเหมาะสมในการใช้ตัวแปรชนิดต่าง ๆ และหากเก็บค่าของข้อมูลที่มีค่ามากกว่าขอบเขตที่ชนิดของข้อมูลนั้นสามารถใช้เก็บค่าได้ หรือเก็บค่าในชนิดของข้อมูลที่ผิดประเภท จะสามารถทำให้โปรแกรมเกิดข้อผิดพลาดได้ โดยชนิดของข้อมูลในโปรแกรมจะมี 4 ประเภทหลัก ๆ คือ

1. แบบจำนวนเต็ม (Integer type)
2. แบบจำนวนจริง (Real floating-point type)
3. แบบตัวอักษร (Character type)
4. แบบสายอักษร (String type)

ชนิดข้อมูลแบบจำนวนเต็ม (Integer Type)

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

ชนิดข้อมูล	ขนาด	ขอบเขต		อธิบายเพิ่มเติม (ชนิดข้อมูลที่เก็บ)
(signed) short	16 บิต	-32,768	ถึง 32,767	จำนวนเต็มแบบสั้น คิดเครื่องหมาย
unsigned short	16 บิต	0	ถึง 65,535	จำนวนเต็มแบบสั้น ไม่คิดเครื่องหมาย
(signed) int	32 บิต	-2,147,483,648	ถึง 2,147,483,649	จำนวนเต็ม คิดเครื่องหมาย
unsigned int	32 บิต	0	ถึง 4,294,967,296	จำนวนเต็ม ไม่คิดเครื่องหมาย
(signed) long	32 บิต	-2,147,483,648	ถึง 2,147,483,649	จำนวนเต็มแบบยาว คิดเครื่องหมาย
unsigned long	32 บิต	0	ถึง 4,294,967,296	จำนวนเต็มแบบยาว ไม่คิดเครื่องหมาย

ข้อสังเกต : ถ้าเป็นตัวแปรแบบคิดเครื่องหมายสามารถละ (signed) ตอนประกาศตัวแปรได้

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

การกำหนดค่าให้กับตัวแปรชนิดจำนวนเต็ม

1. ต้องเป็นตัวเลขไม่มีจุดทศนิยม หากกำหนดเป็นทศนิยมข้อมูลอาจมีการสูญหาย
2. ห้ามใช้เครื่องหมาย , หรือช่องว่างคั่นระหว่างตัวเลข เช่น 1,234 ซึ่งถือว่าผิด
3. ถ้าเป็นค่าว่างไม่จำเป็นต้องใส่เครื่องหมาย + นำหน้าค่า แต่ถ้าเป็นค่าลบต้องใส่เครื่องหมาย - นำหน้าค่าเสมอ
4. ช่วงตัวเลขจำนวนเต็มควรอยู่ในช่วงชนิดข้อมูลนั้น ๆ
5. สามารถใช้เครื่องหมาย Suffix ต่อท้ายค่าที่กำหนดให้ตัวแปรได้ โดยใช้ L ต่อท้ายชนิดข้อมูล long หรือใช้ P ต่อท้ายค่าที่เป็น unsigned (ใช้ตัวพิมพ์ใหญ่หรือเล็กความหมายเหมือนกัน) เช่น

```
testInt = 1234;           /* int */      testInt = -1234;          /* int */
testLongInt = 123456789L; /* long int */ testUnsignedInt = 123456U; /* unsigned int */
testUnsignedLongInt = 123456789UL; // unsigned long int
testInt = +1234;          // int ถ้าใส่เครื่องหมาย + นำหน้าค่าโปรแกรมก็ยังทำงานได้ไม่ผิดพลาด
```

ชนิดข้อมูลแบบจำนวนจริง (Real floating-point type)

Floating point เป็นชนิดข้อมูลแบบตัวเลขทศนิยมที่สามารถนำไปคำนวณทางคณิตศาสตร์ได้ อาจมีจุดทศนิยมหรือไม่มีจุดทศนิยมก็ได้ โดยสามารถเขียนในรูปแบบเลขทศนิยม เช่น 1112.50, -250.0 เป็นต้น 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) Float นิยมดูทศนิยมแค่ 7 หลัก (จำนวนหลักที่จะมีความถูกต้องคือ 7.225 หลัก) Double 15 หลัก (15.955) Long Double 34 หลัก (34.016)

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 ชนิดข้อมูลแบบจำนวนจริงจะเป็นแบบ signed (คิดเครื่องหมาย) เสมอ ซึ่งผู้อ่านสามารถกำหนดค่าตัวแปรโดยคำนึงถึงข้อกำหนดดังนี้

1. ควรจะเป็นค่าตัวเลขที่สามารถมีจุดทศนิยมได้
2. ห้ามใช้เครื่องหมาย , หรือช่องว่างคันระหว่างตัวเลข เช่น 1,234.03
3. กรณีเป็นค่าบวกไม่จำเป็นต้องใส่เครื่องหมาย + หน้าค่า แต่กรณีเป็นค่าลบต้องใส่เครื่องหมาย - หน้าค่าเสมอ
4. การเขียนในรูปแบบใช้ตัวอักษร E ค่าที่ถูกกำหนดสามารถกำหนดได้ทั้งค่าบวกและค่าลบ
5. สามารถใช้เครื่องหมาย Suffix ต่อท้ายค่าที่กำหนดให้ตัวแปรได้ โดยใช้ L ต่อท้ายชนิดข้อมูล long double หรือใช้ D ต่อท้ายค่าที่เป็น double หรือใช้ F ต่อท้ายค่าที่เป็น float (ใช้ตัวพิมพ์ใหญ่หรือเล็กความหมายเหมือนกัน)

```
เช่น testFloat = 654.6F;           // float
      testDouble = 13.31D;          // double
      testLongDouble = 16.34L     // long double
```

ชนิดข้อมูลแบบตัวอักษร (Character type)

Char (Character) เป็นชนิดข้อมูลแบบอักษรตัวเดียว มีขนาด 1 ไบต์ หรือ 8 บิต โดยการแปลงตัวอักษรเพื่อกำกับ โดยเอาอักษร, ตัวเลข (Digit) หรือสัญลักษณ์พิเศษ (Special symbols) ใส่ใน ''

ข้อสังเกต : ตัวแปรประเภท char จะเก็บตัวอักษรเปรียบเสมือนเป็นเลขที่อยู่ในรูปแบบรหัสแอลกอริทึม (ASCII Code)

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

ชนิดข้อมูลแบบสายอักษร (String type)

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

ในภาษา C ไม่มีข้อมูลประเภท String แต่สามารถใช้ Array (แ夸ลำดับ) เข้ามาช่วยซึ่งสามารถกำหนดค่าจะใช้ " " ครอบสายอักษรที่ต้องการดึงตัวอย่างด้านข้าง

ข้อควรรู้ : การเก็บข้อความ (String) จะใช้ตัวแปรประเภท char (ชนิดข้อมูลแบบตัวอักษร) มาต่อ กันเรื่อย ๆ จนได้เป็นข้อความ(สายของอักษร) ซึ่งการนำตัวแปร/กลุ่มข้อมูลประเภทเดียวกันเรียงลำดับกันเรียกว่า Array

Null Character และ Empty String

ในการกำหนดค่าเริ่มต้นให้เป็นค่าว่างสามารถกำหนดให้เป็นตัว Null Character (\0) ซึ่งค่าของ Null Character ('\0') จะมีค่าเป็น 0 สำหรับตัวแปรข้อมูลชนิดสายอักขระถ้าหากไม่ได้กำหนดค่าเริ่มต้น (`char a;` หรือ `char b[5];`) ค่าที่เก็บจะเป็น "" โดยอัตโนมัติ หรือการกำหนดค่าเริ่มต้นของตัวแปรข้อมูลชนิดสายอักขระเป็นข้อความ (`char b[3] = "ab";`) อินเด็กซ์ที่เหลือ (`a[2]`) จะถูกกำหนดค่าเป็น "" อัตโนมัติ และจะเรียกค่าที่เดิมให้อัตโนมัตินี้ว่า Empty String / Zero Termination Character / Null Terminated String ปกติจะไม่สามารถกำหนด "" เอง แต่สามารถใช้ \0 แทนได้ 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	""

จากตารางจะเห็นได้ว่าช่อง Index ที่ 2 และ 8 จะมีค่าเป็น "" คือ ช่องว่าง/Space bar (ASCII Code = 40) ในชุด Empty String และ Index ที่ 11 และ 12 มีค่าเท่ากัน **"" ซึ่งเทียบเท่า '\0'** คือ Null Character (ASCII Code = 0)

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

NOTE : block หมายถึง ส่วนของ Code ที่อยู่ในระหว่างเครื่องหมาย { }

ประเภทตัวแปร

- ตัวแปรแบบโลคอล (Local Variable) เป็นตัวแปรที่มีการประกาศไว้ที่ภายในฟังก์ชัน หรือภายในขอบเขตของ block ใด ๆ ซึ่งไม่สามารถเรียกใช้งานนอกฟังก์ชัน หรือภายนอก block ที่ประกาศไว้ได้
- 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 num is %d\n",num);
16     xnum = testver(num);
17     xnum = xnum * n;
18     printf("Local var is %d\n",x);
19 }
```

ผลลัพธ์โปรแกรม : คอมไพล์ไม่ผ่าน

เพราบรรทัดที่ 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 : เป็นการเรียกใช้เรียกใช้ Function testver() โดยส่งพารามิเตอร์เป็นเลขจำนวนเต็มเข้าไปในฟังก์ชันโดยตัวที่ส่งไปคือค่าของ num นั้นเอง

บรรทัดที่ 5 : ฟังก์ชัน testver() รับค่าเลขจำนวนเต็ม num ซึ่งมีค่าเท่ากับ 10 เข้ามาในฟังก์ชันจากนั้นก็นำไปคูณกับ n ที่เป็นตัวแปรโลคอลได้ผลเท่ากับ 40 และเก็บค่าไว้ที่ตัวแปร sum และค่อยส่งค่ากับสู่ฟังก์ชัน main() ในบรรทัดที่ 14

บรรทัดที่ 17: พอดีตรงนี้โปรแกรมก็เกิดปัญหา เพราะ เรียกใช้ตัวแปรแบบ Local ของฟังก์ชัน testvar() คือตัว n นั้นเองครับ โดยมันจะแสดงว่า Error : Undefined symbol 'n' in function main

การกำหนดค่าให้ตัวแปร

1. กำหนดค่าต่อน嫖妓ตัวแปร

||| **type** varName = value;

โดยที่ value เป็นค่าที่ต้องการกำหนดใส่ตัวแปร varName เช่น **int** x = -1150, 1112;
float price = 12.50;
char alphabet = 'A'; หรือ **char** alphabet = 65;
* Array เช่น **char** stat[] = "PASS"; หรือ **char** stat[5] = "PASS"; หรือ **char** stat[5] = {'P', 'A', 'S', 'S', '\0'};

ข้อสังเกต : การประการด้วยเครื่องสำอางค์ที่มีส่วนผสมของสารเคมี เช่น น้ำหอม น้ำยาล้างหน้า น้ำยาล้างตา เป็นต้น อาจทำให้เกิดอาการแพ้ได้

2. กำหนดค่าในฟังก์ชัน

```
||| left = right;
```

```

จะใช้ตัวดำเนินการ = คือการนำค่าตัวที่ถูกกระทำจากด้านขวา(right) มาใส่ในตัวถูกกระทำทางด้านซ้าย(left)
เช่น x = -1150; //นำค่า -1150 ใส่ตัวแปร x (ให้ x เป็นตัวแปรประเภท int)
price = 12.50; //นำค่า 12.50 ใส่ตัวแปร price (ให้ price เป็นตัวแปรประเภท float)
alphabet = 'A'; หรือ alphabet = 65; //นำค่า 65 ใส่ตัวแปร alphabet (ให้ alphabet เป็นตัวแปรประเภท char)
number = 12 + 3 / 3 ; //นำค่า 13 ใส่ตัวแปร number (ให้ number เป็นตัวแปรประเภท int)
code = 'A' + 2; //นำค่า 67 ใส่ตัวแปร code (ให้ code เป็นตัวแปรประเภท char)
hex = 0x88; //นำค่า 136 (เลขฐาน16 เพื่อมี 0x นำหน้า) ใส่ในตัวแปร hex (ให้ hex เป็นตัวแปรประเภท int)
cal = a + 5; /* นำค่าที่ตัวแปร a เก็บอยู่บวก 5 สมมติให้ a เก็บค่า 5 ดังนั้นจะได้ค่า 10
                ใส่ตัวแปร cal (ให้ cal เป็นตัวแปรประเภท short) */
stat[] = "PASS"; หรือ stat[5] = "PASS"; /* นำค่า 80 ใส่ตัวแปร stat[0], 65 ใส่ตัวแปร stat[1] และ 83 ใส่ใน
                ตัวแปร stat[2] และ stat[3] (stat[] เป็นตัวแปรประเภท char array) */

```

3. กำหนดค่าโดยผ่านค่าเข้าไปในฟังก์ชัน (ยังไม่กล่าวถึงในตอนนี้)

ข้อควรร์ : ใช้ `#define` เปรียบเสมือนกำหนดค่าให้ตัวแปร

||| #define new original

ข้อสังเกต : การใช้ `#define` “ไม่ใช้การกำหนดค่าให้ตัวแปร” แต่เป็นการให้ความหมายให้ Complier รับรู้ว่าค่าที่ถูก `define` (original) คือเอาไว้มาแทน (new) “การ `define` จึงคล้ายการกำหนดค่าให้ตัวแปร” ถ้าใช้กำหนดค่าให้กับตัว `#define` ตัวแปรที่ถูกกำหนดไว้ (new) จะไม่สามารถเปลี่ยนแปลงในฟังก์ชันได้ ได้อีก และการใช้ `#define` สามารถใช้กำหนดค่าสั้นๆในสิ่งที่ต้องการนำมาแทน (new) เพื่อใช้สิ่งที่กำหนดแทนคำสั้นๆนั้นได้ (new) เช่น `#define print printf`

ค่าคงที่ (Constant)

คือค่าข้อมูลชนิดใดชนิดหนึ่งที่ไม่มีการเปลี่ยนแปลงในขณะที่โปรแกรมทำงาน ตัวอย่างเช่น ค่า Pi ซึ่งมีค่าเท่ากับ 3.14 เป็นต้น ซึ่งในภาษา C สามารถใช้งานได้ 3 รูปแบบ คือ ระบุค่าโดยตรง, นิยามโดย #define และ เก็บไว้ในตัวแปร

ระบุค่าโดยตรง (Literal Constants)

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

นิยามโดย #define (Defined Constants)

เป็นการกำหนดค่าคงที่โดยการประกาศไว้ก่อนใช้งาน โดยมีรูปแบบการประกาศใช้งานค่าคงที่ดังนี้

||| `#define ConstantName value`

โดยที่ ConstantName คือ ชื่อของค่าคงที่
Value คือ ค่าที่ต้องการกำหนดให้กับค่าคงที่

เช่น `#define VAT 0.07`
`#define TXT "Welcome to Thailand"`
`#define NEWLINE '\n'`
`#define ONE 1`

เก็บไว้ในตัวแปร (Memory Constants)

เป็นการกำหนดค่าคงที่ในรูปแบบของตัวแปร โดยมีรูปแบบการประกาศใช้งานค่าคงที่ดังนี้

||| `const DataType VARIABLENAME = value;`

โดยที่ DataType คือ ชนิดข้อมูลของค่าคงที่
VARIABLENAME คือ ชื่อของค่าคงที่ควรใช้เป็นตัวอักษรพิมพ์ใหญ่
value คือ ค่าที่ต้องการกำหนดให้กับค่าคงที่

เช่น `const float VAT = 0.07;`
`const int COUNT = 10;`
`const char CH = 'T';`

การแปลงชนิดของข้อมูล (Data Type Conversion)

ข้อมูลต่าง ๆ จะสามารถดำเนินการได้จะต้องเป็นข้อมูลประเภทเดียวกันเท่านั้น หากต้องการนำข้อมูลที่ไม่ใช่ประเภทเดียวกันมาใช้ ก็จะต้องทำการแปลงข้อมูลให้เป็นชนิดเดียวกันก่อน ซึ่งสามารถแปลงได้ 2 วิธีคือ

Implicit Type Conversion

การแปลงข้อมูลชนิดนี้ คอมไพล์เวอร์จะทำหน้าที่แปลงข้อมูลให้อัตโนมัติโดยแปลงชนิดข้อมูลที่มีนัยสำคัญต่ำ ไปเป็นชนิดข้อมูลเดียวกันที่มีนัยสำคัญสูงกว่าในชุดคำสั่งนั้น ๆ โดยมีลำดับนัยสำคัญดังนี้

ต่อ `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 การแปลงชนิดข้อมูลโดยกำหนดได้ตามที่ตนต้องการ สามารถทำได้โดยรูปแบบดังต่อไปนี้

||| `(DataType) ExpressionOrVariableName`

โดยที่ DataType คือ ชนิดข้อมูลปลายทาง
ExpressionOrVariableName คือ นิพจน์หรือตัวแปรที่ต้องการจะแปลงข้อมูล

ตัวอย่างการแปลงข้อมูลแบบ Explicit Type Conversion มีตั้งต่อไปนี้ **กำหนดให้**

- x มีชนิดข้อมูลเป็น int
- y มีชนิดข้อมูลเป็น long int
- z มีชนิดข้อมูลเป็น float

```

z = (float) x          //แปลงชนิดข้อมูลตัวแปร x จาก int    ไปเป็น float และนำผลลัพธ์ที่ได้เก็บลงตัวแปร z
x = (int) y          //แปลงชนิดข้อมูลตัวแปร y จาก long int ไปเป็น int และนำผลลัพธ์ที่ได้เก็บลงตัวแปร x
z = (float) (x + y)  /* จะทำ x + y ก่อน จากนั้นจะแปลงค่าที่ได้จาก x + y เป็น float และนำผลลัพธ์ที่ได้เก็บลง
                      ตัวแปร z */
z = (float) x / y    /* 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_1 format_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

```

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

ข้อ A เป็นการแสดงผลข้อความทั้งหมดจะได้ผลลัพธ์ที่อยู่ในเครื่องหมาย " " ออกมาก้าวหน้า 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 (B), แสดงข้อความ "] " , แสดงค่าในตัวแปร x (0) , แสดงข้อความ " + " , แสดงค่าในตัวแปร y (1.000000), แสดงข้อความ " " , แสดงค่าในตัวแปร text (is), แสดงข้อความ " " , แสดงค่า x + y (1.000000) และทำการขึ้นบรรทัดใหม่

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

```
printf("[%c] 0 + 1.0 %s %f\n", mark1, text, x+y);
printf("[%c] %d + %f %s %f\n", mark2, x, y, text, x+y);
```

```
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 }
```

จากตัวอย่าง เมื่อสั่งแสดงผลเป็นจำนวนจริง ถ้าไม่ได้กำหนดการแสดงผลอะไรเพิ่มเติมจะแสดงทศนิยมอกรมาทั้งหมด 6 ตัวแห่งนั้น ถ้าสั่งแสดงทศนิยมโดยตำแหน่งทศนิยมตัวเลขที่ได้จะถูกบีดตามหลักคณิตศาสตร์ให้อัดโน้มติด และหากสั่งแสดงผลทศนิยมที่ถูกเก็บลงในหน่วยความจำ ออกมากลายตำแหน่งจะพบว่าทศนิยมในตำแหน่งหลังๆบางตัวมีค่าผิดเพี้ยน และ ถ้าสั่งแสดงทศนิยมโดยใช้รูปแบบการแสดงผลผิดประเภทค่าที่ได้ออกมาจะเป็น 0

ข้อควรรู้ : เนื่องจากคอมพิวเตอร์จึงแล้วเก็บข้อมูลเป็นมิติ (0, 1) เท่านั้น คอมพิวเตอร์จึงใช้หลักการ IEEE 754 ในการแปลงเป็นเลขทศนิยม จากตัวอย่างถ้าค่า 1.3 เมื่อถูกเก็บในหน่วยความจำจะมีค่าไม่เท่ากับ 1.3 ดังนั้นจึงไม่ควรใช้ทศนิยมในการเปรียบเทียบ

```
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 }
```

ข้อควรรู้ : หน้าต่างสีดำที่ได้หลังจาก Compile & Run จะเรียกว่าหน้า Console ซึ่งมีความกว้าง 80 ตัวอักษร 25 บรรทัด ในคอมไฟเลอร์สมัยก่อนๆ หากแสดงผลเกิน 80 ตัวอักษรจะทำการขึ้นบรรทัดใหม่ให้อ่อง

ข้อสังเกต : หากนำข้อมูลชนิดทศนิยมเก็บไว้ในตัวแปรประเภทจำนวนเต็มจะถูกบีดเศษทิ้ง ไม่บีดชี้น-ลง ตามหลักคณิตศาสตร์

ข้อสังเกต : หากนำข้อมูลชนิดข้อความเก็บไว้ในตัวแปรประเภทอักษร ตัวสุดท้ายของข้อความจะถูกเก็บลงตัวแปร

ข้อสังเกต : หากนำข้อมูลชนิดจำนวนเต็มมาดำเนินการกับจะได้ผลลัพธ์เป็นข้อมูลจำนวนเต็มด้วยเหมือนกัน ถึงแม้จะกำหนดให้แสดงผลเป็นทศนิยม วิธีแก้ไขคือการใช้ (float) centimeter / (float)100 หรือ (float)centimeter / 100 หรืออีกวิธีหนึ่งคือแก้เป็น centimeter / 100.0

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

รหัสรูปแบบการแสดงผลข้อความ และการจัดรูปแบบการแสดงผลข้อมูลของคำสั่ง printf()

การแสดงผลด้วยคำสั่ง printf() นั้นสามารถแสดงผลได้เกือบทุกชนิดข้อมูล โดยใช้รหัสรูปแบบการแสดงผลแทนค่าจากตัวแปร นิพจน์ หรือค่าคงที่ชนิดต่าง ๆ ในข้อความ รหัสรูปแบบการแสดงผลควรใช้ให้ถูกต้องตามชนิดของข้อมูลที่ต้องการ หากเลือกใช้ผิดประเภทอาจทำให้การแสดงผลผิดพลาดได้ ซึ่งมีรหัสรูปแบบการแสดงผลที่ควรรู้ดังนี้

อักขระพิเศษ	ความหมาย
\b	เลื่อนเคอร์เซอร์โดยหลังไป 1 ตัวอักขระ
\n	ขึ้นบรรทัดใหม่
\r	เลื่อนเคอร์เซอร์ไปทางข่ายสุด
\t	เว้นวรรคจนครบ 1 แท็บ (8 ช่อง)
\'	แสดงเครื่องหมาย ' (single quote) code
\"	แสดงเครื่องหมาย " (double quote)
\\\	แสดงเครื่องหมาย \
%%	แสดงเครื่องหมาย %

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

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	จำนวนเต็มสิบหกแปดชนิด short
%Lf	จำนวนจริงชนิด long double
%lu	จำนวนเต็มชนิด unsigned long

จากตัวอย่าง เมื่อสั่งแสดงผล % ตัวเดียวจะไม่เกิดอะไรขึ้นที่หน้าจอ ต้องใช้ %% ในการแสดงผล ส่วน \ ถ้าใส่ตัวเดียว จะทำให้หลัง \ เป็นข้อความที่จะแสดงผลจนกว่าจะเจอเครื่องหมาย " ที่ไม่ติดตัว \

การจัดพื้นที่ และการกำหนดการแสดงผลข้อมูลของคำสั่ง printf()

บางครั้งการแสดงผลข้อมูลที่เราต้องการนั้น อาจจะต้องการจัดรูปแบบข้อความให้ชิดข้ายหรือขวาหรือแม้แต่การแสดงข้อความตามจำนวนตัวอักขระที่กำหนด เช่น ต้องการแสดงทศนิยม 2 ตำแหน่ง, แสดงข้อความไม่เกิน 10 อักขระ เป็นต้น ซึ่งมีรูปแบบในการแสดงผลข้อมูลชนิดข้อความดังนี้

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

โดยที่ [m] เป็นจำนวนช่องสำหรับใช้ในการแสดงข้อความ เช่น printf("%4s", "ab");
 [n] เป็นจำนวนตัวอักขระที่ต้องการแสดงผล
 variable เป็นค่าข้อมูลหรือตัวแปรที่ต้องการแสดงผล printf("%.3s", "abcdef");
 printf("%5.3s", "abcdef");

ในการนี้แสดงค่าจำนวนเต็ม มีรูปแบบการแสดงผลดังนี้

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

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

ในการนี้แสดงค่าจำนวนจริง มีรูปแบบการแสดงผลดังนี้

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

โดยที่ [m] เป็นจำนวนช่องสำหรับใช้ในการแสดงตัวเลข 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);

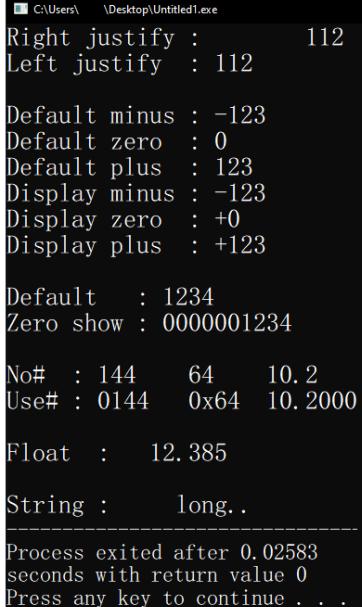
ในการแสดงผลการทำงานของโปรแกรม บางครั้งที่เรียกต้องการกำหนดให้การแสดงผลนั้น ๆ ชิดซ้ายหรือชิดขวา ในส่วนแสดงผล บางครั้งอาจจะต้องการกำหนดเครื่องหมายให้กับค่าข้อมูล ซึ่งภาษา C ได้กำหนดรูปแบบการแสดงผล ดังนี้ ไว้ดังนี้

ค่า flag	ความหมาย
-	ใช้กำหนดการแสดงผลทางจอภาพให้ชิดซ้าย ตามความกว้างของส่วนแสดงผลที่กำหนดไว้
+	ใช้กำหนดให้แสดงเครื่องหมาย + หรือ - หน้าค่าข้อมูลที่ต้องการแสดงผล
0	ใช้กำหนดให้แสดงเลข 0 หน้าค่าข้อมูลที่ต้องการแสดงผล
#	ใช้กำหนดให้แสดงเลข 0, 0x และ 0X หน้าค่าข้อมูลเลขฐานแปดและเลขฐานลิบหก หรือใช้กำหนดให้แสดงผลเป็นเลขทศนิยม ในกรณีที่ข้อมูลที่ต้องการนั้นมีค่าเป็นจำนวนเต็ม หรือใช้กำหนดให้แสดงผลข้อมูลเลขศูนย์ต่อท้าย ในกรณีที่ข้อมูลที่ต้องการนั้นเป็นเลขทศนิยมที่มีศูนย์ต่อท้าย

ตัวอย่าง การจัดรูปการณ์แสดงผลของตัวเลขจำนวนเต็ม จำนวนจริง และสายอักขระ (ข้อความ)

```

1 #include<stdio.h>
2
3 main()
4 {
5     printf("Right justify : %10d\n" , 112);
6     printf("Left justify : %-10d\n\n", 112);
7
8     printf("Default minus : %d\n" , -123);
9     printf("Default zero : %d\n" , 0);
10    printf("Default plus : %d\n" , 123);
11    printf("Display minus : %+d\n" , -123);
12    printf("Display zero : %+d\n" , 0);
13    printf("Display plus : %+d\n\n" , 123);
14
15    printf("Default : %d\n" , 1234);
16    printf("Zero show : %010d\n\n" , 1234);
17
18    printf("No# : %o    %x    %G\n" , 100, 100, 10.2);
19    printf("Use# : %#o   %#x   %#G\n\n" , 100, 100, 10.2);
20
21    printf("Float : %8.3f\n\n" , 12.38456);
22
23    printf("String : %10.6s" , "long....string");
24 }
```



2. ใช้คำสั่ง putchar() สำหรับแสดงแบบอักขระตัวเดียว (Character)

||| putchar(var);

โดยที่ var เป็นตัวแปรชนิดอักขระ (char) หรือเป็นค่าอักขระภายในเครื่องหมาย '' หรือตัวเลข

3. ใช้คำสั่ง puts() สำหรับแสดงข้อมูลแบบสายอักขระ (String)

||| puts(var);

โดยที่ var เป็นตัวแปรชนิดข้อความ หรือเป็นค่าข้อความภายในเครื่องหมาย " "

ข้อสังเกต : คำสั่ง putchar เมื่อ
แสดงผลเรียบร้อยแล้วจะไม่มีการ
ขึ้นบรรทัดใหม่ แต่คำสั่ง puts จะ
ขึ้นบรรทัดใหม่เมื่อแสดงผล
เรียบร้อยแล้ว

```
1 #include<stdio.h>
2
3 main()
4{
5     char ch = 'T';
6     putchar(ch);
7     putchar('A');
8     putchar(66);
9 }
10
11 TAB
```

```
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
11 Thailand0.4
12 *Edit Thailand4.0
13
```

ตัวดำเนินการ

ในการเขียนโปรแกรมทุก ๆ โปรแกรมจะต้องมีการประมวลผลเข้ามาเกี่ยวกับข้อความ แล้วสิ่งที่ทำให้เกิดการประมวลผลนั้นก็คือ 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 ข้อความหรือประโยค ที่เขียนอยู่ในรูปสัญลักษณ์ โดยนำข้อมูล, ตัวแปร, พึงกัน หรือค่าคงที่มาสัมพันธ์กับตัวดำเนินการ (Operator) เช่น C + 1, A + B - (C - 7), A! = B, A && B เป็นต้น	

ข้อควรรู้ : การทำงานของ % ใช้ได้กับจำนวนเต็มเท่านั้น

การทำงานของ +, -, * และ / ที่มีเฉพาะเลขจำนวนเต็ม จะได้ผลลัพธ์เป็นจำนวนเต็มเสมอ

การทำงานของ +, -, * และ / ที่มีเลขทศนิยมอยู่ด้วย จะได้ผลลัพธ์เป็นเลขจำนวนเต็มเสมอ (ตามกฎของ

การแปลงชนิดข้อมูล)

การทำงานของ +, -, * และ / กับข้อมูลชนิดตัวอักขระ จะทำโดยแปลงตัวอักขระเป็นค่า ASCII Code ก่อน

จะได้ผลลัพธ์เป็นจำนวนเต็มเสมอ

จะได้ผลลัพธ์เป็นเลขจำนวนเต็มเสมอ

จะได้ผลลัพธ์เป็นเลขทศนิยมเสมอ (ตามกฎของ

การแปลงชนิดข้อมูล)

จะทำโดยแปลงตัวอักขระเป็นค่า ASCII Code ก่อน

จะนำไปประมวลผล

ตัวดำเนินการกำหนดค่า (Assignment Operator)

ใช้สำหรับกำหนดค่าให้กับตัวแปรทางด้านซ้ายของตัวดำเนินการ โดยจะมีหลักการทำงานที่แตกต่างกันไปและกำหนดให้มี ตัวแปรตั้งนี้ 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	จะนำค่าตัวถูกกระทำการทางด้านขวามาใส่ในตัวถูกกระทำการทางด้านซ้าย $C = 'A' \rightarrow C = 'A' \rightarrow C \text{ มีค่าเท่ากับ 'A' (65)}$ $A = 10 \rightarrow A = 10 \rightarrow A \text{ มีค่าเท่ากับ 10}$
+ = : บวกเท่ากับ	จะนำค่าตัวถูกกระทำการทางด้านซ้าย เท่ากับค่าตัวถูกกระทำการทางด้านซ้ายบวกกับค่าตัวถูกกระทำการทางด้านขวา $A += B \rightarrow A = A + B = 10 + 4 \rightarrow A \text{ มีค่าเท่ากับ 14}$ $A += D \rightarrow A = A + D = 10 + 3.12 \rightarrow A \text{ มีค่าเท่ากับ 13}$ (เนื่องจาก A เป็นข้อมูลชนิดเลขจำนวนเต็ม) $A += C \rightarrow A = A + C = 10 + 65 \rightarrow A \text{ มีค่าเท่ากับ 75}$ (การ + - * / % กับตัวอักษรจะแปลงเป็นค่า ASCII ซึ่งในที่นี้ 'A' มีค่า ASCII = 65) $D += A + B \rightarrow D = D + (A + B) \rightarrow D = 3.12 + (10 + 4) \text{ ดังนั้น D มีค่าเท่ากับ 17.12}$
- = : ลบเท่ากับ	จะนำค่าตัวถูกกระทำการทางด้านซ้าย เท่ากับค่าตัวถูกกระทำการทางด้านซ้ายลบกับค่าตัวถูกกระทำการทางด้านขวา $A -= A \rightarrow A = A - A = 10 - 10 \rightarrow A \text{ มีค่าเท่ากับ 0}$ $A -= A - B \rightarrow A = A - (A - B) = 10 - (10 - 4) \rightarrow A \text{ มีค่าเท่ากับ 4}$
* = : คูณเท่ากับ	จะนำค่าตัวถูกกระทำการทางด้านซ้าย เท่ากับค่าตัวถูกกระทำการทางด้านซ้ายคูณกับค่าตัวถูกกระทำการทางด้านขวาแสดงผล $D *= 2 \rightarrow D = D * 2 = 3.12 * 2 \rightarrow D \text{ มีค่าเท่ากับ 6.24}$
/ = : หารเท่ากับ	จะนำค่าตัวถูกกระทำการทางด้านซ้าย เท่ากับค่าตัวถูกกระทำการทางด้านซ้ายหารกับค่าตัวถูกกระทำการทางด้านขวา $A /= B / 2 \rightarrow A = A / (B / 2) = 10 / (4 / 2) \rightarrow A \text{ มีค่าเท่ากับ 5}$
% = : หารเอาเศษเท่ากับ	จะนำค่าตัวถูกกระทำการทางด้านซ้าย เท่ากับเศษเหลือจากการหารระหว่าง ค่าตัวถูกกระทำการทางด้านซ้าย กับค่าตัวถูกกระทำการทางด้านขวา $A \%= B \rightarrow A = A \% B = 10 \% 4 \rightarrow A \text{ มีค่าเท่ากับ 2}$ $D = D - A \% B \rightarrow D = D - (A \% B) = 10 - (10 \% 4) \rightarrow D \text{ มีค่าเท่ากับ 1.12}$ (% มีลำดับความสำคัญสูงกว่า -)

ตัวดำเนินการยูนารี (Unary Operator)

ตัวดำเนินการ	ความหมาย	
คือ การเพิ่ม ++ ค่าให้กับตัว แปรนึงค่า	Postfix	$X = A++ \rightarrow X = A \rightarrow A = A + 1 \rightarrow \text{กำหนดค่าให้ } X \text{ ก่อนแล้วค่อยเพิ่ม } A \text{ 1 ค่า}$
	Prefix	$X = ++A \rightarrow A = A + 1 \rightarrow X = A \rightarrow \text{เพิ่ม } A \text{ ก่อน 1 ค่า จึงค่อยกำหนดค่าให้กับ } X$
คือ การลดค่า -- ให้กับตัวแปร นึงค่า	Postfix	$X = A-- \rightarrow X = A \rightarrow A = A - 1 \rightarrow \text{กำหนดค่าให้ } X \text{ ก่อนแล้วค่อยลด } A \text{ 1 ค่า}$
	Prefix	$X = --A \rightarrow A = A - 1 \rightarrow X = A \rightarrow \text{เพิ่ม } A \text{ ก่อน 1 ค่า แล้วค่อยกำหนดค่าให้กับ } X$
+ เครื่องหมายบวก	Prefix	$A = +2 \rightarrow A \text{ เท่ากับ 2 (กรณีเป็นค่าบวกจะใส่เครื่องหมาย + หรือไม่ใส่ก็ได้)}$
- เครื่องหมายลบ	Prefix	$A = -2 \rightarrow A \text{ เท่ากับ -2}$ $A = -A \rightarrow A \text{ เท่ากับ 2 (เมื่อใส่ลบหน้าตัวแปรได้ ๆ จะได้ผลลัพธ์ตรงข้าม)}$

```

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

```

จากรูปบรรทัดที่ 7 ค่า a (10) จะถูกใส่ในตัวแปร x จากนั้น a จะมีค่าเพิ่ม 1 เป็น 11 แต่บรรทัดที่ 14 ค่า a จะมีค่าเพิ่ม 1 เป็น 11 ก่อน จากนั้นจึงนำค่า 11 ใส่ใน x

ข้อสังเกต : บางคนสงสัยว่าตัวแปรชื่อเดียว (x และ a) ประกาศซ้ำอีกในฟังก์ชันได้ไหม คำตอบ คือ “ไม่” แต่ในตัวอย่าง ตัวแปร x และ a ที่ประกาศไว้ในบรรทัดที่ 5 เป็นตัวแปร local ภายใน block (ปิกก้า) ระหว่างบรรทัดที่ 4 และ 10 ภายในฟังก์ชัน main() อีกที หากโปรแกรมทำงานผ่านช่วง block ปิกกานี้ไป ตัวแปร ก็จะไม่มีตัวแปร x และ a อีก หากต้องการจึงต้องประกาศตัวแปรใหม่อีกรอบ

ตัวดำเนินการเปรียบเทียบ (Comparison Operator)

เป็นตัวดำเนินการสำหรับเปรียบเทียบข้อมูลระหว่างตัวถูกกระทำทางด้านซ้ายและด้านขวาของตัวดำเนินการ ซึ่งผลลัพธ์จะมีค่าเป็นจริง (True) มีค่าเป็น 1 หรือเท็จ (False) มีค่าเป็น 0 เท่านั้นโดยมีการทำงานดังนี้

==	: ถ้าเท่ากันจะให้ผลลัพธ์เป็นจริง ถ้าไม่เท่าจะให้ผลลัพธ์เป็นเท็จ
!=	: ถ้าไม่เท่ากันจะให้ผลลัพธ์เป็นจริง ถ้าเท่ากันจะให้ผลลัพธ์เป็นเท็จ
>	: ถ้าซ้ายมากกว่าขวาจะให้ผลลัพธ์เป็นจริง ถ้าน้อยกว่าหรือเท่ากับจะให้ผลลัพธ์เป็นเท็จ
>=	: ถ้าซ้ายมากกว่าหรือเท่ากับขวาจะให้ผลลัพธ์เป็นจริง ถ้าน้อยกว่าจะให้ผลลัพธ์เป็นเท็จ
<	: ถ้าซ้ายน้อยกว่าขวาจะให้ผลลัพธ์เป็นจริง ถ้ามากกว่าหรือเท่ากับจะให้ผลลัพธ์เป็นเท็จ
<=	: ถ้าซ้ายน้อยกว่าหรือเท่ากับขวาจะให้ผลลัพธ์เป็นจริง ถ้ามากกว่าจะให้ผลลัพธ์เป็นเท็จ

ตัวอย่าง กำหนดให้ A, B, C, D, E เป็นตัวแปรประเภท int เก็บค่า 3, 3, 4, 5 ตามลำดับ กำหนดให้นิพจน์สีเขียวมีค่าความจริงเป็นจริง และนิพจน์สีแดงมีค่าความจริงเป็นเท็จ จะได้ผลลัพธ์ออกมารูปดังนี้

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

ตัวดำเนินการตรรกะ (Logical Operator)

ใช้สำหรับกำหนดเงื่อนไขมากกว่า 1 เงื่อนไข ซึ่งผลลัพธ์ที่ได้จะได้จะมีแค่จริง หรือเท็จเท่านั้น

&& (และ)	: ใช้กำหนดเงื่อนไข ในกรณีที่ต้องการให้นิพจน์ด้านซ้ายและด้านขวาให้เป็นจริงทั้งคู่จึงจะทำงานตามที่ต้องการ
 (หรือ)	: ใช้กำหนดเงื่อนไข ในกรณีที่ต้องการให้นิพจน์ด้านซ้ายหรือด้านขวาด้านใดด้านหนึ่งให้เป็นจริงจึงจะทำงานตามที่ต้องการ
! (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 = $(i < 10) ? 15 : 5;$
 $X = (25 < 100) ? 0 : 1;$
 $Y = (7 < 5) ? 0 : 1;$

ตัวดำเนินการบอกรากขนาดชนิดข้อมูล (Sizeof Operator)

ตัวดำเนินการชนิดนี้ใช้สำหรับหาขนาดชนิดของข้อมูลต่าง ๆ ที่ผู้อ่านต้องการทราบ โดยมีรูปแบบการใช้งานดังนี้

||| sizeof(Data)

โดยที่ sizeof คือ ตัวดำเนินการบอกรากขนาดชนิดข้อมูล
 Data คือ ชนิดข้อมูลหรือตัวแปรที่ต้องการทราบขนาด

ตัวดำเนินการระดับบิต (Bitwise Operator)

ในบางครั้งผู้อ่านอาจมีความจำเป็นที่จะต้องคำนวณในระดับบิต ซึ่งเป็นหน่วยข้อมูลที่เล็กที่สุด การทำงานในระดับบิตจะช่วยลดภาระการทำงานของ CPU เพราะ CPU จะประมวลผลที่ชนิดข้อมูลที่เป็นบิตเท่านั้น ซึ่งถ้าเป็นข้อมูลชนิดอื่น ๆ จะต้องแปลงข้อมูลให้เป็นบิตก่อน ตัวดำเนินการระดับบิต เป็นตัวดำเนินการเพื่อใช้ในการจัดการข้อมูลในระดับบิต (bit) เป็นตัวดำเนินการพิเศษที่มีอยู่ในภาษา C ซึ่งมีดังนี้

ข้อควรรู้ : บิต (bit) เป็นหน่วยข้อมูลที่เล็กที่สุด โดยข้อมูลหนึ่งบิตจะมีสถานะได้เป็น 2 สถานะคือ 0 (ปิด) หรือ 1 (เปิด) หรือเรียกว่าก่อป่าย่างว่า เลขฐานสองนั่นเอง

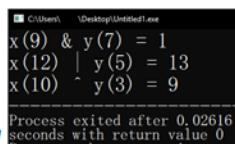
Bitwise AND (&), Bitwise OR (|), Bitwise Exclusive OR/XOR (^)

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

```

1 #include<stdio.h>
2
3 main()
4{
5     x = 9;          // 9 : 0000 1001
6     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", x, y, x^y);
18    /*          9 ^ 7 : 0000 1001           */
19}

```



ข้อควรรู้ :

การดำเนินการ AND จะสังเกตเห็นได้ว่าผลลัพธ์จะเป็น 1 ถ้าหากค่าตัวดำเนินการตัวที่หนึ่งและค่าตัวดำเนินการตัวที่สองเป็น 1 ทั้งคู่ นอกนั้นให้ผลลัพธ์เป็น 0

การดำเนินการ OR จะสังเกตเห็นได้ว่าผลลัพธ์จะเป็น 1 ถ้าหากค่าตัวดำเนินการตัวที่หนึ่งหรือค่าตัวดำเนินการตัวที่สองตัวใดตัวหนึ่งเป็น 1 หรือเป็น 1 ทั้งสองตัว นอกนั้นให้ผลลัพธ์เป็น 0

การดำเนินการ XOR ผู้อ่านจะสังเกตเห็นได้ว่าผลลัพธ์จะเป็น 1 ถ้าหากค่าตัวดำเนินการตัวที่หนึ่งและค่าตัวดำเนินการตัวที่สองมีค่าต่างกัน และได้ผลลัพธ์เป็น 0 เมื่อตัวดำเนินการทั้งสองมีค่าเหมือนกัน

Bitwise Shift Right (>>)

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

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

ข้อควรรู้ : หากเลื่อนบิตไปขวา บิตแรกหลัง Shift จะไม่มีตัวที่จะถูก Shift มาจะถูกกำหนดค่าให้เป็น 0 แทน และบิตสุดท้ายก่อน Shift หลังจากโอน Shift จะถูกตัดค่าทิ้ง (ช่องที่ไม่ลงสี)

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

Bitwise Shift Left (<<)

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

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

ข้อควรรู้ : หากเลื่อนบิตไปซ้าย บิตสุดท้ายหลัง Shift จะไม่มีตัวที่จะถูก Shift มาจะถูกกำหนดค่าให้เป็น 0 แทน และบิตแรกก่อน Shift หลังจากโอน Shift จะถูกตัดค่าทิ้ง (ช่องที่ไม่ลงสี)

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

Bitwise One's Complement (~)

ค่าบิตเริ่มต้น (P)	หลังทำ Bitwise One's Complement ($\sim P$)
1	0
0	1

เป็นตัวดำเนินการสำหรับปรับค่าของบิตเป็นค่าตรงข้าม กล่าวคือ ปรับค่าบิต 1 (true) เปลี่ยนเป็นค่าบิต 0 (false) และ ปรับค่าบิต 0 (false) เปลี่ยนเป็นค่าบิต 1 (true)

```

1 #include<stdio.h>
2
3 main()
4 {
5     int input = 23;      // 23 : 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 right = 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

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

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

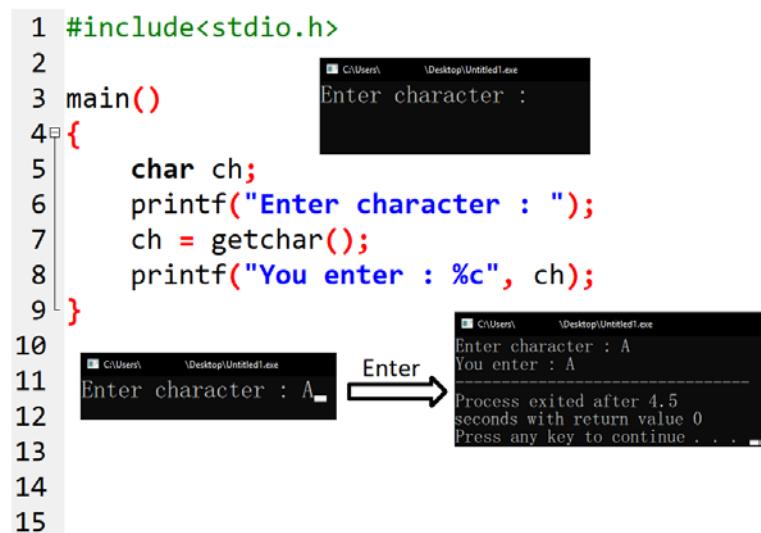
```
||| ch = getch();
```

```
||| 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 {
6     char ch;
7     printf("Enter character : ");
8     ch = getch();
9     printf("You enter : %c", ch);
10}
11
12 Enter character : You enter : A
13
14 Process exited after 1.146 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 = getch();
8     printf("You enter : %c", ch);
9 }
10
11 Enter character : You enter : A
12
13
14
15
```

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

2. การรับข้อมูลนิดข้อความจากคีย์บอร์ดด้วยคำสั่ง gets()

||| gets(str);

โดยที่ str เป็นตัวแปรชนิดสายอักขระ (ข้อความ) สำหรับรับข้อมูลจากคีย์บอร์ด

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

3. การรับข้อมูลทุกชนิดจากคีย์บอร์ดด้วยคำสั่ง scanf()

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

โดยที่ var_n คือ ตัวแปรที่ต้องการใช้สำหรับเก็บข้อมูล ซึ่งต้องตรงกับ format_n ที่กำหนดไว้ format_n คือ ส่วนกำหนดชนิดข้อมูลที่ต้องการรับจากคีย์บอร์ด

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

```
1 #include<stdio.h>
2
3 main()
4 {
5     char name[30];char gender;
6     int age;
7     float weight;
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 \nWeight : %f", age, weight);
19 }
```

จากตัวอย่าง

รหัสรูปแบบ	ชนิดข้อมูล
%c	ตัวอักขระหนึ่งตัว
%d	จำนวนเต็มชนิด int
%ld	จำนวนเต็มชนิด long
%e หรือ %E	จำนวนจริงแบบເອັກຫຼີໂພເນດ
%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	เลขฐานສິບຫກ (พิมพ์ເລັກ/ໃຫຍ່)
และอื่น ๆ เมื่อ format ใน printf	

```
C:\Users\ \Desktop\Untitled1.exe
Enter gender(M or F) : M
Enter name :
ปี蛾กรอกເພີ່ມ (M) ແລ້ວກັດ Enter ....
C:\Users\ \Desktop\Untitled1.exe
Enter gender(M or F) : M
Enter name : JadeSPK
Enter age :
C:\Users\ \Desktop\Untitled1.exe
Enter gender(M or F) : M
Enter name : JadeSPK
Enter age :
Enter weight :
ໃສ່ອາຍຸ (9) ແລ້ວກັດ Enter ....
```

```
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 . . .
```

ແລະສຸດທ້າຍກຣອກນ້າໜັກ
(55.5) ແລ້ວກັດ Enter ...
ຂ້ອມູລທີກຣອກເຂົ້າໄປກີຈະຖຸກ
ແສດງຜລອອກນາດັ່ງຮູບ

ຂ້ອສັງເກດ : ລາກໃສ່ກຣອກຄໍາຕັ້ງອັກຂະຕັ້ງເດືອຍ (char) ດ້ວຍ scanf ຕ່ອຈາກ scanf ອຶກທີ (ບຣທັດທີ 10) ຈະທ່າໃຫ້ເກີດ bug!
ເນື່ອງຈາກອັກຂະຕັ້ງເດືອຍທີ່ຮັບຄໍາໄປຈະເປັນ \n ຄືກຣອກກົດ Enter ເພື່ອເສົ້າງສິນກຣອກຄໍາຕັ້ງຢ່າງ scanf ໃນຄໍາສັ່ງກ່ອນໜ້າ
(ບຣທັດທີ 8)

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

ຈະຈະແກດວ່າວິທີຕາມຮູປດ້ານລ່າງນີ້

```
1 #include<stdio.h>
2
3 main()
4 {
5     char name[30];char gender;
6     int age;
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("Enter name : [%c] %s", gender, name);
12 }
```

ຂ້ອສັງເກດ : ກຣອກຄໍາຕັ້ງຢ່າງ scanf ສາມາຄົວໜ້າລາຍຄ່າ ໄສີໃນແຕ່ລະຕັ້ງແປຣໄດ້ກາຍໃນຄໍາສັ່ງເດືອຍເພີ່ມກໍານົດ format ແລ້ວຕັ້ງແປຣໃຫ້ຕຽບກັນ ເມື່ອ input ເຂົ້າໄປ ສາມາຄໃຫ້ enter ອີ່ວ່ານວຽກໃນກຣອກຄໍາໃນແຕ່ລະຄ່າໄດ້

```
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\n", a, b);
10 }
```

ຂ້ອຄວຽກ : ທ່າງກຣອກຂ້ອມູລຕ່ອກນ້າໜັກ ທັງສາມາຄ ກໍານົດຕັ້ງອັກຊ່າເພື່ອໃຫ້ຄົ່ນຮ່ວງກຣອກຄໍາຕັ້ງຢ່າງ input ຕັ້ງຕ່ອໄປໄດ້ຕາມຮູປ

ການກໍານົດລຳດັບກຣອກຂ້ອມູລຂອງຄໍາສັ່ງ scanf()

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

ສາມາຄກໍານົດລຳດັບກຣອກຂ້ອມູລໂດຍໃໝ່ເຄື່ອງໝາຍ *
ໄດ້ ພ້າກລາດັບຂອງຂ້ອມູລທີ່ຮັບເຂົ້າມາໃໝ່ເຄື່ອງໝາຍ * ຂ້ອມູລທີ່
ຮັບເຂົ້າມາໃນລຳດັບດັ່ງກ່າວຈະໄມ່ຖຸກຈັດເກັບລົງຕັ້ງແປຣ ດັ່ງຕົວຢ່າງ
ຕ່ອໄປນີ້

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

ในบางครั้งที่ต้องการกำหนดรูปแบบการรับข้อมูลเอง (format_n) สามารถทำได้โดยการใช้ Regular Expression เข้ามาช่วยซึ่งสามารถศึกษาเพิ่มเติมได้จาก <https://www.tamemo.com/post/111/how-to-regular-expression> หรือ <https://regexr.com> โดยภาษา C/C++ สามารถใช้ได้ 2 รูปแบบดังต่อไปนี้

รูปแบบ	ความหมาย
[]	เรียกว่า bracket expression หมายถึง กลุ่มของตัวอักษรในนี้เท่านั้นที่ต้องการ หากเจอตัวอักษรนอกกลุ่มนี้จะทำการหยุดรับค่าทันทีสำหรับภาษา C/C++ สามารถใช้ – ช่วยในการนับที่อักษรที่ต้องการเป็น 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' และแมทช์กับอักษรตัวใหญ่ 'A' ถึง 'Z' * [ก-݂] <u>สำหรับภาษาไทย</u> ซึ่งมีทั้ง พยัญชนะ สระ ตัวเลขไทยมากมาย ให้เริ่มด้วย 'ก' ถึง '݂' จะได้ครอบทุกตัว
[^]	เหมือนแคส [] แต่เปลี่ยนเป็น <u>ไม่</u> เจอตัวอักษรในนี้แทน หากเจอตัวอักษรในกลุ่มนี้จะทำการหยุดรับค่าทันทีสำหรับภาษา C/C++

เช่น	<code>%[^\\n]</code> รับค่าเรียกๆจนกว่าเจอข้อบรรทัดใหม่	\rightarrow input : asduh <code>[enter]</code>	output : asduh
	<code>%[^ab]</code> รับค่าเรียกๆจนกว่าเจอ a หรือ b	\rightarrow input : eiei <code>a</code> ty	output : eiei
	<code>%[^a-c]</code> รับค่าเรียกๆจนกว่าเจอ a, b หรือ c	\rightarrow input : wow <code>c</code> oasd	output : wow
	<code>%[ab]</code> รับค่าจะสิ้นสุดเมื่อไม่พบอักษร a, b	\rightarrow input : eiei <code>a</code> tyAerb	output :
	<code>%[A-C]</code> รับค่าจะสิ้นสุดเมื่อไม่พบอักษร A, B หรือ C	\rightarrow input : ABBCA <code>a</code> AC	output : ABBCA
	<code>%[A-C0-6]</code> รับค่าจะสิ้นสุดเมื่อไม่พบอักษร A, B หรือ C และ 0-6	\rightarrow input : A0 <code>v</code> h0Ba6z	output : A0

ข้อสังเกต : ในบางเครื่องรหัสคำสั่งพิเศษในการขึ้นบรรทัดใหม่ (new line) ไม่ใช่ `\n` แต่จะเป็น `\r\n` หรือ `\r`

ข้อสังเกต : รหัสคำสั่งพิเศษแต่ละตัวจะมีรหัสแอ็คชัน (ASCII Code) เช่น ขึ้นบรรทัดใหม่ (new line) รหัสแอ็คชัน = 10

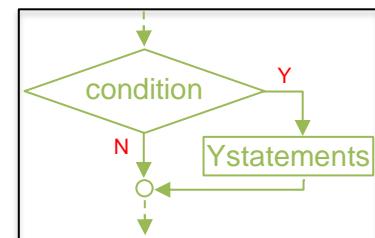
ควบคุมทิศทางการทำงานโปรแกรมด้วยคำสั่งตัดสินใจ (Decision Control Statement)

กำหนดให้	condition	เป็นตัวแปรหรือนิพจน์ที่เป็นเงื่อนไขของคำสั่ง
	condition_n	เป็นเงื่อนไขที่ <i>ก</i> ที่ใช้กำหนดการตัดสินใจของโปรแกรม
	statements_1	เป็นชุดคำสั่งที่ต้องทำงานเมื่อเงื่อนไขที่กำหนดเป็นจริง
	statements_2	เป็นชุดคำสั่งที่ต้องทำงานเมื่อเงื่อนไขที่กำหนดเป็นเท็จ
	statements_n	เป็นชุดคำสั่งที่ต้องทำงานเมื่อเงื่อนไขที่กำหนดที่ <i>ก</i> เป็นจริง
	Ystatements	เป็นชุดคำสั่งที่ต้องทำงานเมื่อเงื่อนไขที่กำหนดเป็นจริง
	Nstatements	เป็นชุดคำสั่งที่ต้องทำงานเมื่อเงื่อนไขที่กำหนดทั้งหมดเป็นเท็จ

คำสั่งตัดสินใจแบบเลือกทำ หรือไม่ทำด้วยคำสั่ง if

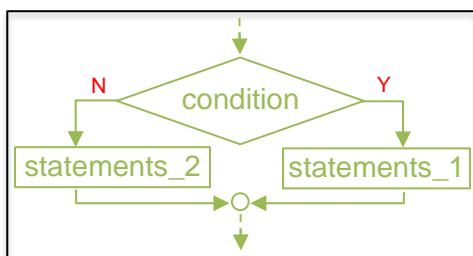
```
if (condition)
{
    Ystatements;
}
```

คำสั่ง if เป็นคำสั่งที่เราใช้กำหนดให้โปรแกรมตัดสินใจ ทำหรือไม่ทำสิ่งใดสิ่งหนึ่ง โดยตรวจสอบเงื่อนไขที่กำหนดว่าเป็นจริงหรือเท็จ ถ้าเงื่อนไขที่กำหนดให้เป็นจริง (true) โปรแกรมจะทำงานที่ชุดคำสั่งที่อยู่ภายใต้คำสั่ง if แต่ถ้าเงื่อนไขที่กำหนดไว้ เป็นเท็จ (false) โปรแกรมจะข้ามไปทำงานที่คำสั่งต่อไปทันที



ข้อควรรู้ : การกำหนดเงื่อนไขตามปกติ คือใช้นิพจน์มาทำการเปรียบเทียบค่ากันโดยใช้ Operator > < >= <= != == เช่น if(a>b) การดำเนินการเปรียบเทียบกันตามปกนี้จะได้ผลลัพธ์สองแบบ คือ เท็จ (False) จะได้ค่าเป็น 0 หาก และจริง(True) จะได้ค่าเป็น 1 นิพจน์ที่ใส่เป็นเงื่อนไขไม่ได้กำหนดตามปกติ เช่น if(a = -10) ซึ่ง a = -10 เป็นการกำหนดค่าตัวแปร a จะมีค่าเป็น -10 -> if(-10) หากเป็นเช่นนี้ค่าอื่น ๆ ที่ไม่ใช่ 0 จะถือว่าเป็นจริง(True) หมวด เช่น if(8), if(-9) จะถือว่าเป็นจริงและเข้าเงื่อนไขใน if

คำสั่งตัดสินใจแบบสองทางเลือกด้วยคำสั่ง if...else



คำสั่ง if...else เป็นคำสั่งที่เราใช้กำหนดให้โปรแกรมตัดสินใจเลือกทำคำสั่งอย่างใดอย่างหนึ่ง จาก 2 ทางเลือก โดยตรวจสอบเงื่อนไขที่กำหนดว่าเป็นจริงหรือเท็จ ถ้าเงื่อนไขที่กำหนดให้เป็นจริง (true) โปรแกรมจะทำงานที่ชุดคำสั่งที่อยู่ภายใต้คำสั่ง if แต่ถ้าเงื่อนไขที่กำหนดไว้เป็นเท็จ (false) โปรแกรมจะทำงานที่ชุดคำสั่งที่อยู่ภายใต้คำสั่ง else

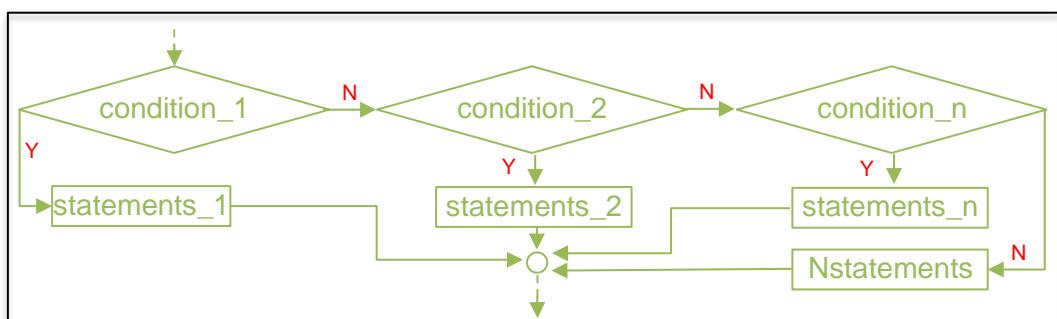
```
if (condition)
{
    statements_1;
}
else
{
    statements_2;
}
```

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 ไวด้วย โดยโปรแกรมจะตรวจสอบเงื่อนไขที่ลงทะเบียนเลือกใหม่เงื่อนไขเป็นจริง (true) ก็จะทำงานที่ชุดคำสั่งภายใต้ทางเลือกนั้น โดยไม่พิจารณาทางเลือกอื่นที่ไม่ได้ตรวจสอบอีก ในกรณีที่เงื่อนไขเป็นเท็จ (false) ให้ตรวจสอบเงื่อนไขถัดไป ในกรณีที่เงื่อนไขทั้งหมดเป็นเท็จให้โปรแกรมทำงานที่ชุดคำสั่ง else

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



ข้อสังเกต : เจื่อนไขที่ใช้เปรียบเทียบ หากใช้การเปรียบเทียบว่าเท่ากันหรือไม่ ให้ใช้เครื่องหมาย == ในการเปรียบเทียบ อย่างสับสนกับเครื่องหมาย = (การกำหนดค่า) หากใช้เครื่องหมาย = ดังในตัวอย่างเมื่อโปรแกรมทำงานถึงส่วน if คอมpileอร์จะพิจารณาเงื่อนไข x = 1 แต่ x = 1 เป็นการกำหนดให้ x มีค่าเป็น 1 (จริง) ดังนั้นจะได้เจื่อนไข if(x) ซึ่งก็คือ if(1) หรือ if(true) ดังนั้นโปรแกรมจะเข้าเงื่อนไข if โดยไม่สนใจว่า x จะมีค่าอะไรก่อน

```
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 }
```

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 คือการใส่ปีกภาคลุ่มคำสั่งหลัง else ห้างหนด

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;
}
```

โดยที่	condition	เป็นตัวแปรหรือนิพจน์ที่เป็นเงื่อนไขของคำสั่ง
	constant_n	เป็นค่าคงที่ที่ใช้ตรวจสอบกับตัวแปรหรือนิพจน์เงื่อนไข
	statements_n	เป็นชุดคำสั่งที่ต้องทำงานเมื่อตัวแปรหรือนิพจน์ที่ ก เป็นจริง
	Nstatements	เป็นชุดคำสั่งที่ต้องทำงานเมื่อตัวแปรหรือนิพจน์ห้างหนดเป็นเท็จ

```
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 }
```

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

ข้อควรรู้ : หาก statement ที่ตามหลัง if, else if หรือ else มีเพียงแค่ 1 statement ไม่จำเป็นต้องใส่ {} คลุม statement ที่ได้ **แต่ควรใส่ไว้เพื่อกันล้มในกรณีที่มีหลายstatement** ตามหลัง if, else if หรือ else (ถ้าลืมใส่ปีกการตัวเดี่ยวเงื่อนไขจะครอบคลุมเพียง statement เดียว)

คำสั่ง switch-case เป็นคำสั่งตัดสินใจที่มีการทำงานเหมือนกับคำสั่ง if...else if คือ เลือกทางใดทางหนึ่งจากทางเลือกที่มากกว่า 2 ทาง ในแต่ละทางเลือกจะมีการกำหนดเงื่อนไขของแต่ละทาง โดยตรวจสอบเงื่อนไขแต่ละทางเลือก หากพบว่าทางเลือกไหนมีเงื่อนไขเป็นจริง (true) ก็จะทำชุดคำสั่งภายในทางเลือกนั้นและไม่พิจารณาทางเลือกที่ยังไม่ได้ตรวจสอบอีก ในกรณีที่เงื่อนไขเป็นเท็จ (false) ให้ตรวจสอบเงื่อนไขถัดไป ในกรณีที่เงื่อนไขห้างหนดเป็นเท็จ ให้โปรแกรมทำงานที่ชุดคำสั่ง default : สำหรับ flowchart จะเหมือนกับคำสั่งตัดสินใจแบบหลายทางเลือกด้วยคำสั่ง if...else if

ข้อสังเกต : ต้องใช้ค่าคงที่ในการแยกกรณีห้างหนน จากบรรทัดที่ 10 และ 11 จะเห็นว่า 'A' คือการเอกสารหัสแอกซ์ของ A ซึ่งมีค่าเท่ากับ 65 มาเป็นเงื่อนไขในการเช็ค และเอา 'B' คือการเอกสารหัสแอกซ์ของ B ซึ่งมีค่าเท่ากับ 66 มาเป็นเงื่อนไขในการเช็ค

ข้อสังเกต : หากไม่ได้ใส่คำสั่ง break เมื่อ switch-case เจอเงื่อนไขใน case ใดที่เป็นจริงแล้ว ก็จะทำ statement ใน case นั้น และทำ statement ใน case ด้านล่างต่อไปเรื่อยๆ จนกว่าจะเจอคำสั่ง break หากในบรรทัดที่ 11 ไม่ได้ใส่ คำสั่ง break ก็จะแสดงผล CASE 2 ขึ้นบรรทัดใหม่ Default ขึ้นบรรทัดใหม่ ต่อจาก CASE B ขึ้นบรรทัดใหม่

จากตัวอย่าง เป็นการเขียนโปรแกรมตัดเกรด และกำหนดให้คะแนนมีค่าเป็น 75 เมื่อถึงช่วงเช็คเงื่อนไข จะทำการเช็คเงื่อนไขใน if ก่อน (คะแนนมากกว่าหรือเท่ากับ 80 หรือไม่) ปรากฏว่าเป็นเท็จจึงเช็คเงื่อนไข else if ต่อไป คือคะแนนมากกว่าหรือเท่ากับ 75 แต่น้อยกว่า 80 หรือไม่ ทำให้เงื่อนไขนี้เป็นจริง และทำการคำสั่งหลัง เงื่อนไขนั้นเพียง 1 statement (แสดง B+)

```

1 #include<stdio.h>
2
3 main()
4 {
5     int score = 75;
6     if (score >= 80)
7         printf("A");
8     else if (score >= 75 && score < 80)
9         printf("B+");
10    else if (score >= 70 && score < 75)
11        printf("B");
12    else if (score >= 65 && score < 70)
13        printf("C+");
14    else if (score >= 60 && score < 65)
15        printf("C");
16    else if (score >= 55 && score < 60)
17        printf("D+");
18    else if (score >= 50 && score < 55)
19        printf("D");
20    else
21        printf("F");
22 }

```

ควบคุมการทำงานของโปรแกรมด้วยคำสั่งวนลูป (Repetition Control Statement)

วนลูปการทำงานด้วยจำนวนรอบที่แน่นอนด้วยคำสั่ง for

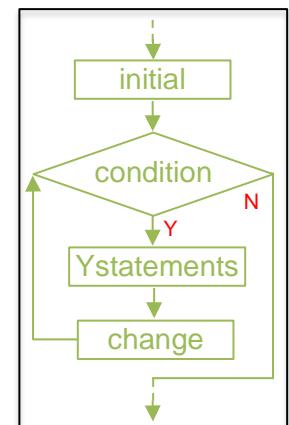
```

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

```

คำสั่ง for เป็นคำสั่งวนซ้ำการทำงานเดิมๆ ด้วยจำนวนรอบที่แน่นอน โดยที่โปรแกรมจะตรวจสอบเงื่อนไขก่อนการทำงานทุกรอบ ถ้าเงื่อนไขเป็นจริงให้ทำงานที่ชุดคำสั่งภายในลูปต่อไป เมื่อทำงานเสร็จจะเพิ่ม หรือลดค่าตัวแปร และตรวจสอบเงื่อนไขใหม่อีกครั้ง ถ้าเงื่อนไขเป็นเท็จให้โปรแกรมออกจากลูปการทำงานไป ทำงานที่คำสั่งถัดไปทันที

<u>โดยที่</u>	initial	เป็นส่วนกำหนดค่าเริ่มต้นของตัวแปรที่ใช้กำหนดเงื่อนไข
	condition	เป็นส่วนกำหนดเงื่อนไขการวนลูป สามารถใส่หลายเงื่อนไขได้ โดยใช้ตัวดำเนินการแบบต่างๆ เช่น ช่วยกำหนดเงื่อนไขให้เป็นจริงหรือเท็จ
	change	เป็นส่วนการเปลี่ยนแปลงของตัวแปรหลังทำงานจนจบในลักษณะ
	Ystatements	เป็นชุดคำสั่งที่ต้องทำเมื่อเงื่อนไขเป็นจริง



```

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

```

```

main()
{
    int i=0;
    for(; i<=2;){
        printf("%d", i++);
    }
}

```

```

main()
{
    int i=0;
    for(; i!=3;){
        printf("%d", i++);
    }
}

```

ข้อสังเกต : เงื่อนไขสามารถกำหนดได้หลายแบบแต่ผลลัพธ์ที่ออกมามาได้เหมือนกัน และในส่วนของ initial และ change ไม่จำเป็นต้องมีเสมอไป หรือสามารถกำหนดให้มีมากกว่า 1 ก็ได้ ในส่วนของ condition ก็สามารถมีมากกว่า 1 ได้เช่นกันโดยอาศัยตัวดำเนินการต่างๆ เช่น ตามรูปตัวอย่างด้านขวา แต่ควรเขียนให้ผู้อื่นเข้าใจโปรแกรมได้ง่าย

```

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

```

```

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

```

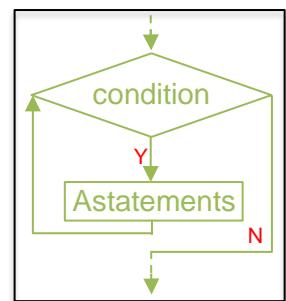
ข้อควรระวัง : บาง Complier เก่า ๆ บางตัวไม่สามารถประกาศตัวแปรในส่วน initial ได้เหมือนในตัวอย่างด้านข้าง แต่ถ้าประกาศได้ตัวแปรที่ถูกประกาศนั้นจะเป็นตัวแปรประเภท Local อยู่ภายใต้ลูปตัวแปร จึงถูกlobber ออกจากหน่วยความจำเมื่อตอนที่ไม่ได้ประกาศไว้

วนลูปการทำงานเมื่อเงื่อนไขเป็นจริงด้วยคำสั่ง while

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

คำสั่ง while เป็นคำสั่งวนซ้ำการทำงานเดิมๆ ของโปรแกรม โดยโปรแกรมตรวจสอบเงื่อนไขก่อน การทำงานทุกครั้ง ถ้าเงื่อนไขที่กำหนดเป็นจริง โปรแกรมจะทำงานที่ชุดคำสั่งภายในลูป เมื่อโปรแกรมทำงาน เสร็จจะตรวจสอบเงื่อนไขใหม่อีกครั้ง ถ้าเงื่อนไขที่กำหนดเป็นเท็จ โปรแกรมจะออกจากลูปการทำงานไปทำงาน ที่คำสั่งถัดไปทันที

โดยที่ condition เป็นเงื่อนไขที่กำหนดให้ตรวจสอบก่อนทำงานภายในลูปทุกครั้ง
Astatements เป็นชุดคำสั่งที่ต้องทำเมื่อเงื่อนไขเป็นจริง (หลังตรวจสอบเงื่อนไข)

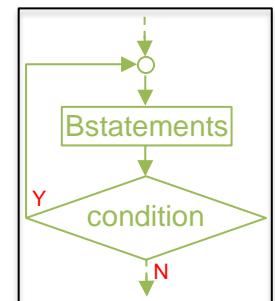


วนลูปการทำงานอย่างน้อย 1 ครั้งด้วยคำสั่ง do...while

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

คำสั่ง do..while เป็นคำสั่งวนซ้ำการทำงานเดิมๆ โดยโปรแกรมจะทำงานชุดคำสั่งภายในลูปก่อน 1 รอบ จึงตรวจสอบเงื่อนไขที่กำหนด ถ้าเงื่อนไขที่กำหนดเป็นจริง ให้กลับไปทำงานชุดคำสั่งภายในลูปอีกครั้ง และตรวจสอบเงื่อนไขที่กำหนดอีกครั้ง ถ้าเงื่อนไขที่กำหนดเป็นเท็จ โปรแกรมจะออกจากลูปการทำงานไป ทำงานที่คำสั่งถัดไปทันที

โดยที่ condition เป็นเงื่อนไขที่กำหนดให้ตรวจสอบหลังการทำงานภายในลูปทุกครั้ง
Bstatements เป็นชุดคำสั่งที่ทำงานก่อนตรวจสอบเงื่อนไข



ข้อควรรู้ : ในกรณีที่จำนวนคำสั่งภายในลูปมากเกินไป อาจจะไม่จำเป็นต้องใส่เครื่องหมาย {} ก็ได้ แต่แนะนำให้ใส่เครื่องหมายปีกๆ {} ทุกครั้ง

คำสั่ง break และ continue

```
1 #include<stdio.h>
2
3 main()
4 {
5     for(int i = 0; i<3; i++)
6     {
7         for(int j = 0; j<9; j++)
8         {
9             printf("i = %d, j = %d\n", i, j);
10            if(j == 2) break;
11        }
12    }
13 }
```

```
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
```

```
1 #include<stdio.h>
2
3 main()
4 {
5     for(int i = 0; i<3; i++)
6     {
7         for(int j = 0; j<4; j++)
8         {
9             if(j == 2) continue;
10            printf("i = %d, j = %d\n", i, j);
11        }
12    }
13 }
```

```
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
```

```
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
```

Process exited after 0.00172 seconds with return value 0
Press any key to continue .

ในรูปทางด้านข่ายเมื่อคอมไพล์แล้วทำงานถึงคำสั่ง break คอมไพล์จะทำการออกจากคำสั่งวนลูปปัจจุบันที่กำลังทำงานอยู่ (เลิกทำคำสั่งวนลูป for ที่มีตัวแปร j กำหนด ซึ่งโดยปกติการที่ลูป j จะจบได้ j ต้องมีค่า $>= 9$ แต่เมื่อ j มีค่าเท่ากับ 2 ตรงเงื่อนไข break ที่กำหนดไว้จึงออกลูป)

ในรูปด้านขวาเมื่อคอมไпал์แล้วทำงานถึงคำสั่ง continue คอมไпал์จะทำการข้ามลูปในรอบนั้น (คำสั่งวนลูป for ที่กำลังทำงานในรอบที่ตัวแปร j มีค่าเท่ากับ 2 จะถูกข้ามไปยังรอบถัดไป(คือรอบที่ 3) หากเจอคำสั่ง continue)

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

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

Caution!

Process exited after 0.01793 seconds with return value 0
Press any key to continue .

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

```

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

```

ข้อควรระวัง : ระวังการกำหนดเงื่อนไขของลูป อาจทำให้เกิดลูปนันต์ได้ คือลูปที่วนไม่รู้จบเนื่องจากไม่ว่าลูปจะผ่านไปกี่รอบก็จะไม่มีวันผิดเงื่อนไขที่กำหนด ไว้เหมือนตัวอย่างทางด้านซ้าย หรือตามตัวอย่างต่อไปนี้ เช่น while(true) , while(1) , for(int x = 0; x >=0; x++) เป็นต้น

จากตัวอย่างด้านล่าง เป็นการใช้ลูปแต่ละประเภทในการไล่แสดงผลตัวเลขตั้งแต่ 1 ถึง 6 โดยมีการกำหนดเงื่อนไข และรูปแบบที่ใช้ต่างกันดังนี้

```

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

```

C:\Users\1\Desktop\Untitled1.exe
123456
Process exited after 0.0236 seconds with return value 0
Press any key to continue . . .

```

int i=0;
while(i<6)
{
    printf("%d",i++);
}
```

```

int i=0;
while(i<6)
{
    printf("%d",i++);
    i++;
}
```

```

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

อธิบายลูป for อย่างละเอียด

```

for(i=0; i<6;i++)
{
    printf("%d",i);
}
```

เมื่อเริ่มต้นทำลูป for จะทำงานในส่วนของ initial ก่อนในที่นี่คือการกำหนดให้ i มีค่าเป็น 0 หลังจากนั้นจะเช็คเงื่อนไขในส่วน condition ว่าเป็นจริงหรือไม่ ถ้าเป็นจริงให้ทำ statement ในลูปและเมื่อจบลูปปึงทำในส่วน change ต่อ

i = 0
loop#1

for(i=0; i<6;i++)
{
 printf("%d",i);
}

for(i=0; i<6;i++)
{
 printf("%d",i);
}

i = 1
loop#2

for(i=0; i<6;i++)
{
 printf("%d",i);
}

for(i=0; i<6;i++)
{
 printf("%d",i);
}

i = 2
loop#3

for(i=0; i<6;i++)
{
 printf("%d",i);
}

for(i=0; i<6;i++)
{
 printf("%d",i);
}

i = 3
loop#4

for(i=0; i<6;i++)
{
 printf("%d",i);
}

for(i=0; i<6;i++)
{
 printf("%d",i);
}

i = 4
loop#5

for(i=0; i<6;i++)
{
 printf("%d",i);
}

for(i=0; i<6;i++)
{
 printf("%d",i);
}

i = 5
loop#6

for(i=0; i<6;i++)
{
 printf("%d",i);
}

for(i=0; i<6;i++)
{
 printf("%d",i);
}

จะทำอย่างนี้เข้าไปข้างมาจนกว่าจะผิดเงื่อนไข (ลูปรอบแรก i=0 ลูปรอบ 2 i=1 ลูปรอบ 6 i=5 และเป็นลูปรอบสุดท้ายเนื่องจากก่อนจะขึ้นลูปรอบที่ 7 จะต้องเช็คเงื่อนไขก่อนเข้า แล้ว i มีค่าเป็น 6 ซึ่งจะผิดเงื่อนไข ผลคือจบลูป

Array

ตัวแปรอาร์เรย์ เปรียบเสมือนการนำตัวแปรมาเรียงต่อกันหลายๆตัว โดยที่ทุกตัวมีชนิดข้อมูลเดียวกัน มีชื่อตัวแปรเดียวกัน และสามารถอ้างถึงตำแหน่งข้อมูลแต่ละตัวที่เรียงต่อกันด้วยลำดับการจัดเรียง ซึ่งเรียกตำแหน่งข้อมูลแต่ละตัวว่า อินเด็กซ์ (Index)

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

ตัวแปรอาร์เรย์ 1 มิติ (One Dimension Array)

เปรียบได้กับการนำตัวแปรมาเรียงต่อกันหลายๆตัวในลักษณะของ列าข้อมูล ซึ่งสามารถจำลองตัวอย่างตัวแปรอาร์เรย์ 1 มิติ ชื่อตัวแปร intEx1 เป็นตัวแปรชนิดจำนวนเต็มที่สามารถเก็บข้อมูลจำนวนเต็มได้ 6 ตัว ยกตัวอย่างดังนี้

0	1	2	3	4	5
→					
5	8	1	3	9	2

จากตัวอย่างจะเห็นได้ว่า ตัวแปรอาร์เรย์สามารถเก็บข้อมูลได้ 6 ตัว โดยที่

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)

เปรียบได้กับการนำตัวแปรมาเรียงต่อกันหลายๆตัวในลักษณะของตารางข้อมูล ซึ่งสามารถจำลองตัวอย่างตัวแปรอาร์เรย์ 2 มิติ ชื่อตัวแปร intEx2 เป็นตัวแปรชนิดจำนวนเต็มที่สามารถเก็บข้อมูลจำนวนเต็มได้ 15 ตัว ยกตัวอย่างดังนี้

[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]

3	7	2	6	1
7	1	9	2	6
4	2	5	4	3

จากรูปด้านบนจะเห็นได้ว่า ตัวแปรอาร์เรย์ที่มีขนาด 3 แถว 5 คอลัมน์ มีลักษณะคล้ายตาราง สามารถเก็บข้อมูลได้ 15 ตัว ตัวอย่างเช่น ตัวแปรแรกที่ 1 คอลัมน์ที่ 1 มีค่าเท่ากับ 5 ซึ่งสามารถเขียนได้เป็น intEx2[0][0] = 3

ตัวแปรแรกที่ 1 คอลัมน์ที่ 2 มีค่าเท่ากับ 1 ซึ่งสามารถเขียนได้เป็น intEx2[0][1] = 7

ตัวแปรแรกที่ 1 คอลัมน์ที่ 5 มีค่าเท่ากับ 2 ซึ่งสามารถเขียนได้เป็น intEx2[0][4] = 1

ตัวแปรแรกที่ 2 คอลัมน์ที่ 1 มีค่าเท่ากับ 7 ซึ่งสามารถเขียนได้เป็น intEx2[1][0] = 7

ตัวแปรแรกที่ 2 คอลัมน์ที่ 2 มีค่าเท่ากับ 1 ซึ่งสามารถเขียนได้เป็น intEx2[1][1] = 1

ตัวแปรแรกที่ 2 คอลัมน์ที่ 4 มีค่าเท่ากับ 2 ซึ่งสามารถเขียนได้เป็น intEx2[1][3] = 2

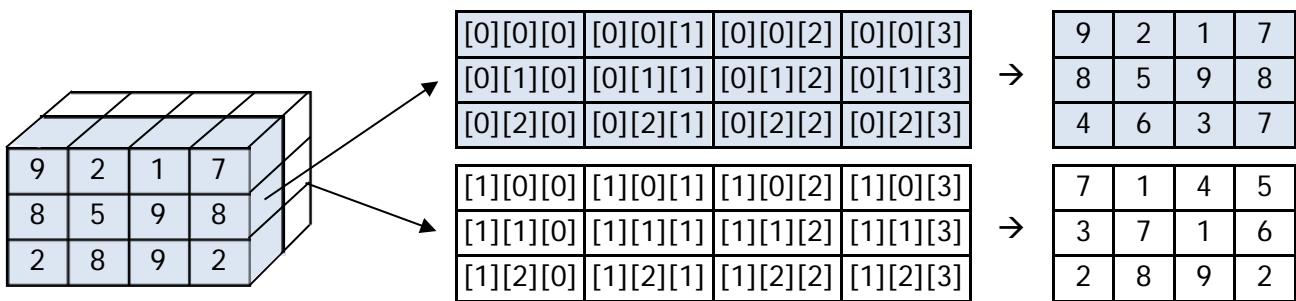
ตัวแปรแรกที่ 3 คอลัมน์ที่ 1 มีค่าเท่ากับ 3 ซึ่งสามารถเขียนได้เป็น intEx2[2][0] = 4

ตัวแปรแรกที่ 3 คอลัมน์ที่ 2 มีค่าเท่ากับ 4 ซึ่งสามารถเขียนได้เป็น intEx2[2][1] = 2

ตัวแปรแรกที่ 3 คอลัมน์ที่ 5 มีค่าเท่ากับ 5 ซึ่งสามารถเขียนได้เป็น intEx2[2][4] = 3

ตัวแปรอาร์เรย์ 3 มิติ (3-Dimension Array)

เปรียบได้กับการนำตัวแปรมาเรียงต่อกันหลายๆตัวในลักษณะของกล่องข้อมูล ซึ่งสามารถจำลองตัวอย่างตัวแปรอาร์เรย์ 3 มิติ ชื่อตัวแปร intEx3 เป็นตัวแปรชนิดจำนวนเต็มที่สามารถเก็บข้อมูลจำนวนเต็มได้ 24 ตัว ยกตัวอย่างดังนี้



จากรูปด้านบนจะเห็นได้ว่า ตัวแปรอาร์เรย์ที่มีขนาด 2 บล็อก 3 และ 4 คอลัมน์ สามารถเก็บข้อมูลได้ 24 ตัว ตัวอย่างเช่น

- ตัวแปรบล็อกที่ 1 แกรที่ 1 คอลัมน์ที่ 1 มีค่าเท่ากับ 9 ซึ่งสามารถเขียนได้เป็น intEx3[0][0][0] = 9
- ตัวแปรบล็อกที่ 1 แกรที่ 1 คอลัมน์ที่ 2 มีค่าเท่ากับ 2 ซึ่งสามารถเขียนได้เป็น intEx3[0][0][1] = 2
- ตัวแปรบล็อกที่ 1 แกรที่ 1 คอลัมน์ที่ 3 มีค่าเท่ากับ 1 ซึ่งสามารถเขียนได้เป็น intEx3[0][0][2] = 1
- ตัวแปรบล็อกที่ 1 แกรที่ 2 คอลัมน์ที่ 1 มีค่าเท่ากับ 8 ซึ่งสามารถเขียนได้เป็น intEx3[0][1][0] = 8
- ตัวแปรบล็อกที่ 1 แกรที่ 2 คอลัมน์ที่ 2 มีค่าเท่ากับ 5 ซึ่งสามารถเขียนได้เป็น intEx3[0][1][1] = 5
- ตัวแปรบล็อกที่ 1 แกรที่ 3 คอลัมน์ที่ 4 มีค่าเท่ากับ 7 ซึ่งสามารถเขียนได้เป็น intEx3[0][2][3] = 7
- ตัวแปรบล็อกที่ 2 แกรที่ 1 คอลัมน์ที่ 1 มีค่าเท่ากับ 7 ซึ่งสามารถเขียนได้เป็น intEx3[1][0][1] = 7
- ตัวแปรบล็อกที่ 2 แกรที่ 1 คอลัมน์ที่ 2 มีค่าเท่ากับ 1 ซึ่งสามารถเขียนได้เป็น intEx3[1][0][0] = 1
- ตัวแปรบล็อกที่ 2 แกรที่ 1 คอลัมน์ที่ 3 มีค่าเท่ากับ 4 ซึ่งสามารถเขียนได้เป็น intEx3[1][0][2] = 4
- ตัวแปรบล็อกที่ 2 แกรที่ 2 คอลัมน์ที่ 1 มีค่าเท่ากับ 3 ซึ่งสามารถเขียนได้เป็น intEx3[1][1][0] = 3
- ตัวแปรบล็อกที่ 2 แกรที่ 2 คอลัมน์ที่ 2 มีค่าเท่ากับ 7 ซึ่งสามารถเขียนได้เป็น intEx3[1][1][1] = 7
- ตัวแปรบล็อกที่ 2 แกรที่ 3 คอลัมน์ที่ 4 มีค่าเท่ากับ 2 ซึ่งสามารถเขียนได้เป็น intEx3[1][2][3] = 2

การประกาศตัวแปรอาร์เรย์

||| **type varName[k][m][n]**

โดยที่ type เป็นชนิดของข้อมูล

varName เป็นชื่อตัวแปรอาร์เรย์
n เป็นขนาดคอลัมน์ของตัวแปรอาร์เรย์
m เป็นขนาดแถวของตัวแปราร์เรย์
k เป็นขนาดแกรของตัวแปราร์เรย์

เช่น **int intNum[5];**
char chPassword[4];
float fPrice[3][2];
int intNo[3][2][5];
int intCount[3][5];

จากตัวอย่างโค้ดจะเห็นได้ว่าจะมีการกำหนดขนาดตัวแปรไว้ด้วยเครื่องหมาย [] เช่น การประกาศตัวแปรชนิดจำนวนเต็มชื่อ intCount ขนาด 3 และ 5 คอลัมน์ เป็นต้น

การกำหนดค่าข้อมูลให้ตัวแปรอาร์เรย์

การกำหนดค่าให้กับตัวแปรแบบอาร์เรย์นั้น เราต้องกำหนดต่าแทนงอินเด็กซ์เพื่อบรุตำแหน่งของตัวแปรที่จะกำหนดค่า 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 ซึ่งเราอาจจะกำหนดค่าภายหลังการประกาศตัวแปรได้เช่นกัน ซึ่งมีตัวอย่างดังนี้

จากตัวอย่างจะเห็นได้ว่า เรากำหนดค่าให้ตัวแปร intNum ในลำดับที่ 2 อินเด็กซ์ที่ 1 มีค่าเท่ากับ 6 ซึ่งได้ผลลัพธ์ดังนี้

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

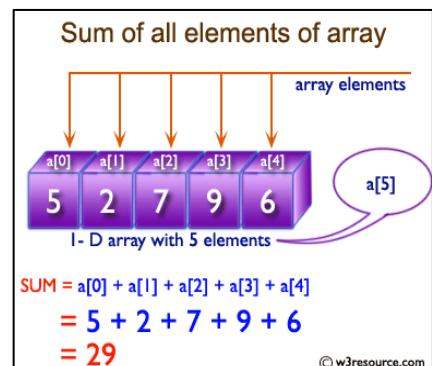
```
int intNum[5];  
char chPassword[4];  
float fPrice[3][2];  
  
intNum[1] = 6;  
chPassword[0] = 'Z';  
fPrice[2][1] = 0.08;
```

การอ้างถึงข้อมูลในตัวแปรอาร์เรย์

เมื่อมีการกำหนดค่าให้กับตัวแปรแล้ว ต้องไปก็ต้องรู้วิธีการอ่านข้อมูลจากตัวแปร 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] * 3; เป็นการกำหนดค่าให้กับตัวแปร Ans โดยมีค่ามาจากการคูณระหว่างค่าในตัวแปรอาร์เรย์ตำแหน่งที่ 3 กับจำนวนเต็ม 4 เป็นต้น



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

Input the number of elements = 5

Input 5 elements

element 0 : 5

element 1 : 2

element 2 : 7

element 3 : 9

element 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 เป็นโปรแกรมที่คำนวณผลรวมของค่าที่รับเข้าไปใส่ในอาร์เรย์

โค้ดในบรรทัดที่ 8 9 จะเป็นการให้ผู้ใช้กรอกว่าจะเก็บค่ากี่ค่า เพื่อใช้สร้างอาร์เรย์ที่มี index ขนาดเท่ากับข้อมูลที่ใส่ บรรทัดที่ 14 – 16 เป็นการรับค่าแต่ละค่าเข้ามาเก็บในอาร์เรย์แต่ละอินเด็กซ์ตั้งแต่ a[0] a[1] จนถึง a[n-1] และระหว่างนั้นถ้ารับค่าเข้ามาแล้วให้นำไปบวกเข้าไปเก็บไว้ในตัวแปร sum โดยใช้ค่าสั่ง sum = sum + a[i]

บรรทัดที่ 18 เป็นการแสดงผลบวกทั้งหมดจากตัวแปร sum

บรรทัดที่ 19 เป็นการแสดงค่าที่เก็บอยู่ในแต่ละอินเด็กซ์โดยใช้ลูปมาช่วย โดยกำหนดให้ลูปวนช้าเท่ากับจำนวนสมาชิกที่กรอกเข้ามา (n)

ข้อควรระวัง : การประกาศตัวแปร array ต้องประกาศให้พอด้วยจำนวนที่ใช้ "ไม่นั้นจะเกิด error ขึ้น เช่น ประกาศอาร์เรย์ชื่อ a ไว้ 5 index (int a[5];) แต่ตอนเรียกใช้หรือกำหนดค่า a[5] จะบ้ม...."

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

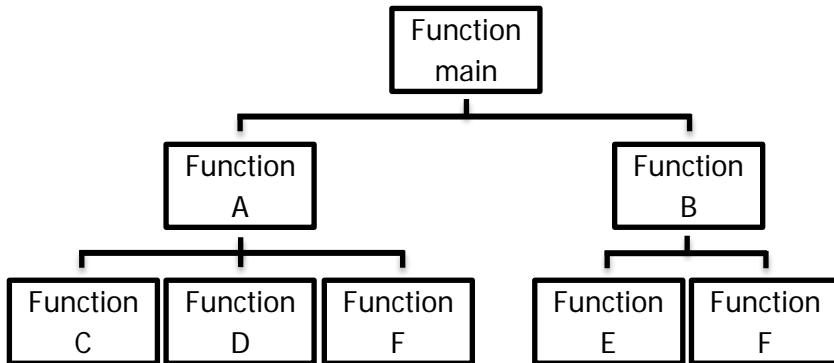
ข้อควรรู้ : array จะเริ่มต้นที่อินเด็กซ์เป็น 0 และตัวสุดท้ายเป็นอินเด็กซ์ที่ n-1 (ให้ n เป็นจำนวนอินเด็กซ์ที่ประกาศไว้)

ฟังก์ชัน (Function)

คือชุดคำสั่งที่รวมกันเป็นโปรแกรมย่อยๆ ภายในเครื่องหมาย {} ถูกสร้างขึ้นมาเพื่อทำงานอย่างโดยย่างหนึง และมีการตั้งชื่อของฟังก์ชันเพื่อให้สังวรกต่อการเรียกใช้งานตามกฎการตั้งชื่อ โดยมีรูปแบบการเรียกใช้งานแตกต่างกันคือ จะมีการรับหรือไม่รับข้อมูลจากโปรแกรมที่เรียกใช้งาน และจะมีการส่งหรือไม่ส่งค่าจ้มูลออกจากฟังก์ชัน ซึ่งรูปแบบการใช้งานต่างๆ ของฟังก์ชันจะขึ้นอยู่กับหน้าที่และเป้าหมายการทำงานของฟังก์ชันนั้นๆ

ในการเขียนโปรแกรมภาษา C

มีโครงสร้างประกอบด้วยฟังก์ชันการทำงาน โดยเริ่มต้นการทำงาน ฟังก์ชัน main() เสมอ และฟังก์ชัน main() นั้นสามารถเรียกใช้ฟังก์ชันย่อยอื่นๆ ได้ไม่ว่าจะเป็นฟังก์ชันที่ผู้อ่านสร้างขึ้นมาเอง (User-Defined Functions) หรือฟังก์ชันมาตรฐานที่ภาษา C ได้สร้างมาให้แล้ว (Standard Library Function) นอกจากนี้ในฟังก์ชันย่อยก็ยังสามารถที่จะเรียกใช้ฟังก์ชันย่อยอื่นๆ 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 สามารถตรวจสอบการทำงานและพัฒนาโปรแกรม ได้ง่ายก็จริง แต่ผู้เขียนไม่แนะนำให้แบ่งโปรแกรมย่อยๆ เพื่อเรียกใช้งานข้อนักกันมากเกินไป เนื่องจากจะทำให้โปรแกรมเกิดความซับซ้อน แก้ไขและพัฒนาได้ยาก

ฟังก์ชันที่สร้างขึ้นเอง (User-Defined Functions)

เป็นฟังก์ชันที่ผู้อ่านเขียนโดยสร้างฟังก์ชันขึ้นมาใช้งานเอง อิงตามรูปแบบการสร้างฟังก์ชันของภาษา C เพื่อให้ทำงานอย่างโดยย่างหนึง ซึ่งสามารถแบ่ง รูปแบบการสร้างฟังก์ชันได้ 4 รูปแบบดังนี้

1. ฟังก์ชันที่ไม่มีการคืนค่ากลับและไม่มีการรับค่าพารามิเตอร์ (Void Functions with No Parameters)
2. ฟังก์ชันที่ไม่มีการคืนค่ากลับ แต่มีการรับค่าพารามิเตอร์ (Void Functions with Parameters)
3. ฟังก์ชันที่มีการคืนค่ากลับ แต่ไม่มีการรับค่าพารามิเตอร์ (Function Return Value with No Parameters)
4. ฟังก์ชันที่มีการคืนค่ากลับ และมีการรับค่าพารามิเตอร์ (Function Return Value with Parameters)

ฟังก์ชันที่ไม่มีการคืนค่ากลับและไม่มีการรับค่าพารามิเตอร์ (Void Functions with No Parameters)

เป็นฟังก์ชันที่สร้างขึ้นโดยไม่มีการรับค่าข้อมูล (พารามิเตอร์) ใด ๆ จากฟังก์ชันที่เรียกใช้งาน และเมื่อฟังก์ชันทำงานเสร็จจะไม่มีการคืนค่าข้อมูลใด ๆ กลับไปให้ฟังก์ชันที่เรียกใช้งาน ซึ่งมีรูปแบบการสร้างฟังก์ชันดังนี้

นิยาม functionName เป็นชื่อฟังก์ชันที่ต้องการสร้าง
statements เป็นชุดคำสั่งภายในฟังก์ชัน

```

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

จากรูปแบบฟังก์ชัน เราสามารถสร้างฟังก์ชันใช้งานได้ดังตัวอย่างต่อไปนี้

```

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

ข้อสังเกต : คีย์เวิร์ด void หน้าชื่อฟังก์ชัน เป็นการกำหนดฟังก์ชันนั้นไม่มีการคืนค่าใด ๆ กลับไปยังฟังก์ชันที่เรียกใช้งาน คีย์เวิร์ด void หลังชื่อฟังก์ชัน เป็นการกำหนดฟังก์ชันนั้นไม่มีการรับค่าพารามิเตอร์ใด ๆ จาก ฟังก์ชันที่เรียกใช้งาน

เมื่อเราได้เขียนฟังก์ชันขึ้นมาเพื่อใช้งานเองแล้วนั้น แต่หากยังไม่สามารถเรียกใช้งานฟังก์ชันตั้งกล่าว ได้จะนกว่า จะมีการประกาศโปรแกรมไทยปีของฟังก์ชันก่อนฟังก์ชัน main เพื่อให้คอมไพล์รู้จักฟังก์ชัน จากตัวอย่างโค้ดจะประกาศ โปรแกรมหาได้ดังนี้ `void PrintName (void);` ซึ่งการประกาศโปรแกรมหาปีจะมีรูปแบบตามด้านล่าง

||| `void functionName (void);`

เมื่อเราประกาศโปรแกรมหาปีของฟังก์ชันเพื่อให้ฟังก์ชัน main รู้จักกับฟังก์ชันที่เราได้สร้างขึ้นแล้ว เราสามารถเรียกใช้งานฟังก์ชันดังกล่าวได้จากฟังก์ชัน main หรือจากฟังก์ชันอื่น ๆ ภายในโปรแกรมได้ จากตัวอย่างโค้ดจะเรียกใช้งานด้วย ฟังก์ชันได้ดังนี้ `PrintName();` ซึ่งการเรียกฟังก์ชันจะมีรูปแบบตามด้านล่าง

||| `functionName ();`

จากรูปแบบการประกาศโปรแกรมหาปีของฟังก์ชันและการเรียกใช้งานฟังก์ชัน เราสามารถสร้างฟังก์ชัน ใช้งานและเรียกใช้งานได้ด้วย ตัวอย่างง่าย ๆ ทางด้านขวาดังนี้

ข้อควรรู้ : เนื่องจากโปรแกรมทำงานจากซ้ายไปขวา บนลงล่าง เมื่อ รันโปรแกรม หากในฟังก์ชัน main เจอฟังก์ชันแปลง ๆ ที่คอมไпал์ร์ไม่รู้จักจะแจ้ง error ออกมานะ จึงแก้ด้วยการประกาศให้คอมไпал์ร์รู้ว่ามีฟังก์ชันแปลง ๆ นือยุนนะ โดยรูปแบบเป็นเหมือนที่ประกาศไว้ใน โปรแกรมหาปี ส่วนตัวฟังก์ชันแปลง ๆ นี้จะอธิบายส่วนประกอบข้างในอยู่ ในส่วนอื่นของโปรแกรมหลังฟังก์ชัน main (คล้าย ๆ กับการใช้ตัวแปรแต่ยังไม่ได้ประกาศใช้ตัวแปร)

```
#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 ก็ไม่มีการคืนค่าข้อมูลใด ๆ กลับไปให้ฟังก์ชันที่เรียกใช้งาน ซึ่งมีรูปแบบการสร้างฟังก์ชันดังนี้

||| `void functionName (typeParameter_1 varName_1, ..., typeParameter_n varName_n){
{
 statements;
}`

โดยที่ `typeParameter_n` เป็นชนิดข้อมูลที่ต้องการรับจากฟังก์ชันที่เรียกใช้งาน `varName_n` เป็นชื่อตัวแปรที่ใช้รับข้อมูลจากฟังก์ชันที่เรียกใช้งาน

จากรูปแบบฟังก์ชัน เราสามารถสร้างฟังก์ชันใช้งานได้ดังตัวอย่างต่อไปนี้

```
void Calculate(int A, int B){  
    int Ans;  
    Ans = A * B;  
    printf("%d", Ans);  
}
```

จากตัวอย่างโค้ดจะประกาศโปรแกรมหาปีได้ดังนี้ `void Calculate(int, int);` ซึ่งการประกาศโปรแกรมหาปีจะมีรูปแบบตามด้านล่าง

||| `void functionName (typeParameter_1, ..., typeParameter_n);`

โดยที่ `typeParameter_n` เป็นชนิดข้อมูลที่ต้องการรับจากฟังก์ชันที่เรียกใช้งาน

เมื่อเราประกาศโปรแกร addCriterionของฟังก์ชันเพื่อให้ฟังก์ชัน main รู้จักกับฟังก์ชันที่เราได้สร้างขึ้นแล้ว เราสามารถเรียกใช้งานฟังก์ชันดังกล่าวได้จากฟังก์ชัน main หรือจากฟังก์ชันอื่น ๆ ภายในโปรแกรมได้ จากตัวอย่างโค้ดจะเรียกใช้งานด้วย ฟังก์ชันได้ดังนี้ `Calculate(X, Y);` ซึ่งการเรียกฟังก์ชันจะมีรูปแบบตามด้านล่าง

||| `functionName (varName_1, ..., varName_n);`

โดยที่ `varName_n` เป็นชื่อตัวแปรที่ใช้รับข้อมูลจากฟังก์ชันที่เรียกใช้งาน ซึ่งต้องมีชนิดข้อมูลเดียวกันกับชนิดข้อมูล ที่ฟังก์ชันต้องการ

ฟังก์ชันที่มีการคืนค่ากลับ แต่ไม่มีการรับค่าพารามิเตอร์ (Function Return value with No Parameters)

เป็นฟังก์ชันที่สร้างขึ้นโดยไม่มีการรับค่าข้อมูล (พารามิเตอร์) ใด ๆ จากฟังก์ชันที่เรียกใช้งาน และ เมื่อฟังก์ชันทำงานเสร็จจะมีการคืนค่าข้อมูลกลับไปให้ฟังก์ชันที่เรียกใช้งาน ซึ่งมีรูปแบบการสร้างฟังก์ชันดังนี้

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

โดยที่ **typeReturn**
varNameReturn

เป็นชนิดข้อมูลที่ต้องการคืนค่ากลับให้ฟังก์ชันที่เรียกใช้งาน
เป็นชื่อตัวแปรที่คืนค่ากลับให้ฟังก์ชันที่เรียก

จากรูปแบบฟังก์ชัน เราสามารถสร้างฟังก์ชันใช้งานได้ดังต่อไปนี้

```
int Calculate (void) {
    int A, B, C;
    A = 3;
    B = 5;
    C = A * B;
    return C;
}
```

จากตัวอย่างโค้ดจะประกาศโปรโตไทป์ได้ดังนี้ **[int Calculate ()]** ซึ่งการประกาศโปรโตไทป์จะมีรูปแบบตามด้านล่าง

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

โดยที่ **typeReturn** เป็นชนิดข้อมูลที่ต้องการคืนค่ากลับให้ฟังก์ชันที่เรียกใช้งาน

เมื่อเราประกาศโปรโตไทป์ของฟังก์ชันเพื่อให้ฟังก์ชัน main รู้จักกับฟังก์ชันที่เราได้สร้างขึ้นแล้ว เราสามารถเรียกใช้งานฟังก์ชันดังกล่าวได้จากฟังก์ชัน main หรือจากฟังก์ชันอื่น ๆ ภายในโปรแกรมได้ จากตัวอย่างโค้ดจะได้การเรียกใช้งานตัวฟังก์ชัน โดยเก็บผลการทำงานของฟังก์ชันไว้ที่ตัวแปร Ans ดังนี้ **[Ans = Calculate ()]** ซึ่งการเรียกฟังก์ชันจะมีรูปแบบตามด้านล่าง

```
varNameReturn = functionName();
```

โดยที่ varNameReturn เป็นชื่อตัวแปรที่รับค่าข้อมูลที่ฟังก์ชันคืนค่ากลับมาให้ เรียกอีกอย่างว่าผลลัพธ์ของฟังก์ชันฟังก์ชันที่มีการคืนค่ากลับ และมีการรับค่าพารามิเตอร์ (Function Return Value with Parameters)

เป็นฟังก์ชันที่สร้างขึ้นโดยมีการรับค่าข้อมูล (พารามิเตอร์) จากฟังก์ชันที่เรียกใช้งาน และเมื่อฟังก์ชันทำงานเสร็จจะมีการคืนค่าข้อมูลกลับไปให้ฟังก์ชันที่เรียกใช้งาน ซึ่งมีรูปแบบการสร้างฟังก์ชันดังนี้

```
typeReturn functionName (typeParameter_1 varName_1, ..., typeParameter_n varName_n){
{
    statements;
    return varNameReturn;
}
```

โดยที่ varName_n

เป็นชื่อตัวแปรที่ใช้รับข้อมูลจากฟังก์ชันที่เรียกใช้งาน และต้องมีชนิด ข้อมูลเดียวกันกับชนิดข้อมูลที่ฟังก์ชันต้องการ

typeReturn

เป็นชนิดข้อมูลที่ต้องการคืนค่ากลับให้ฟังก์ชันที่เรียกใช้งาน

typeParameter_n

เป็นชนิดข้อมูลที่ต้องการรับจากฟังก์ชันที่เรียกใช้งาน

varNameReturn

เป็นชื่อตัวแปรที่คืนค่ากลับให้ฟังก์ชันที่เรียก

จากรูปแบบฟังก์ชัน เราสามารถสร้างฟังก์ชันใช้งานได้ดังตัวอย่างต่อไปนี้

```
float Calculate(int R){
    float X;
    X = 3.14 * R * R;
    return X;
}
```

จากตัวอย่างโค้ดจะประกาศโปรโตไทป์ได้ดังนี้ **float Calculate (int);** ชึ่งการประกาศโปรโตไทป์จะมีรูปแบบตามด้านล่าง

```
||| typeReturn functionName (typeParameter_1, ..., typeParameter_n);
```

โดยที่ **typeReturn** เป็นชนิดข้อมูลที่ต้องการคืนค่ากลับให้ฟังก์ชันที่เรียกใช้งาน
typeParameter_n เป็นชนิดข้อมูลที่ต้องการรับจากฟังก์ชันที่เรียกใช้งาน

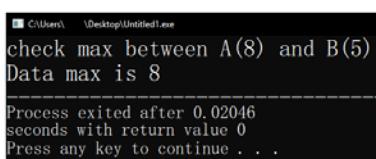
เมื่อเราประกาศโปรโตไทป์ของฟังก์ชันเพื่อให้ฟังก์ชัน main รู้จักกับฟังก์ชันที่เราได้สร้างขึ้นแล้ว เราสามารถเรียกใช้งานฟังก์ชันดังกล่าวได้จากฟังก์ชัน main หรือจากฟังก์ชันอื่น ๆ ภายในโปรแกรมได้ จากตัวอย่างโค้ดจะได้การเรียกใช้งานตัวฟังก์ชัน โดยเก็บผลการทำงานของฟังก์ชันไว้ที่ตัวแปร Area ดังนี้ **Area = Calculate (radius);** ชึ่งการเรียกฟังก์ชันจะมีรูปแบบตามด้านล่าง

```
||| varNameReturn = functionName(varName_1, ..., varName_n);
```

โดยที่ **varNameReturn** เป็นชื่อตัวแปรที่รับค่าข้อมูลที่ฟังก์ชันคืนค่ากลับมาให้ เรียกอีกอย่างว่าผลลัพธ์ของฟังก์ชัน **varName_n** เป็นชื่อตัวแปรที่ใช้รับข้อมูลจากฟังก์ชันที่เรียกใช้งาน และต้องมีชนิด ข้อมูลเดียวกันกับชนิด ข้อมูลที่ฟังก์ชันต้องการ

ตัวอย่าง

```
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 }
```



บรรทัดที่ 3 เป็นการประกาศโปรโตไทป์ หากไม่ต้องการประกาศต้องประกาศให้นำฟังก์ชัน maxData เขียนไว้ก่อนฟังก์ชัน main

บรรทัดที่ 12-17 เป็นฟังก์ชันที่รับค่าพารามิเตอร์ มาเป็น int 2 ตัว และไม่มีการส่งค่าใดๆอกมาจากฟังก์ชัน (void) เมื่อเรียกใช้

โปรแกรมนี้เป็นการเปรียบเทียบค่า 2 ค่า ระหว่าง 8 และ 5 โดยการส่งค่าผ่านเข้าไปในฟังก์ชัน (ค่า a=8, b=5) ให้คำนวนอุณหภูมิและแสดงผลตามเงื่อนไขที่กำหนดไว้

ข้อควรรู้ : ในการรับส่งข้อมูลระหว่าง Argument กับ Parameter ต้องมีชนิดข้อมูลตรงกัน

Argument คือ ตัวรับข้อมูลเข้ามาในฟังก์ชัน

Parameter คือ ตัวส่งข้อมูลไปยังฟังก์ชัน



```
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 }
```

จากตัวอย่างด้านบน สามารถ

เขียนได้แบบตัวอย่างด้านขวาได้ โดย

ผลลัพธ์ที่ได้มีค่าเท่ากัน

```
C:\Users\...\Desktop\Untitled1.exe
check max between A(8) and B(5)
Data max is 8
Process exited after 0.02429
seconds with return value 0
Press any key to continue . . .
```

```

1 #include<stdio.h>
2
3 char testReturn(int);
4
5 main()
6 {
7     char test = testReturn(9);
8     printf("%c - Positive\n\n",test);
9
10    printf("%c - Negative\n\n",testReturn(-1));
11
12    testReturn(0);
13
14 }
15
16 char testReturn(int num)
17 {
18     if(num > 0)
19     {
20         printf("print in testReturn - case if (%d)\n", num);
21         return 'p';
22     }
23     else if(num < 0)
24     {
25         printf("print in testReturn - case else-if (%d)\n", num);
26         return 'n';
27     }
28     else
29     {
30         printf("print in testReturn - case else (%d)\n", num);
31         return 'z';
32     }
33 }

```

```

C:\Users\...\Desktop\Untitled1.exe
print in testReturn - case if (9)
p - Positive

print in testReturn - case else-if (-1)
n - Negative

print in testReturn - case else (0)

Process exited after 0.02386
Seconds with return value 0
Press any key to continue . . .

```

จากตัวอย่าง เป็นการสร้างฟังก์ชัน `testReturn` แต่นำตัวรายละเอียดฟังก์ชันไว้หลัง `main` จึงต้องประกาศprototype ให้ปีเพื่อให้คอมไพล์ตรวจสอบว่า มีฟังก์ชัน `testReturn` อยู่ในโปรแกรม หากไม่ประกาศพอโปรแกรมทำงานไม่จากข้างไปข้าว บนลงล่างเข้าฟังก์ชัน `main` ก็จะเจอกับการเรียนกใช้ฟังก์ชัน `testReturn` ซึ่งคอมไพล์ร์ไม่รู้จัก (คล้ายๆกับการใช้ตัวแปร แต่ไม่ได้ประกาศว่าจะใช้..)

ฟังก์ชัน `testReturn` ที่สร้างจะเป็นฟังก์ชันที่รับค่าตัวเลข `int` ไป 1 ตัว (1 พารามิเตอร์) โดยในฟังก์ชันจะเป็นการนำตัวเลขไปเข้าเงื่อนไขเพื่อเช็คว่าเลขนั้น เป็นจำนวนเต็มบวก จำนวนเต็มลบ หรือจำนวนเต็ม 0 หากเข้าเงื่อนไขแต่ละเงื่อนไขจะให้แสดงว่าเข้าเงื่อนไขอะไร แล้วใน () แสดงเลขที่กรอกเข้ามา พร้อมทั้งส่งค่าออกจากการฟังก์ชันเป็น `char` (รหัส ASCII) ตามเงื่อนไข

ในฟังก์ชันหลัก (`main`) จะแสดงถึงการเรียกใช้ฟังก์ชันแบบต่าง ๆ ส่วนมากจะนำฟังก์ชันมาใช้ได้ 2 รูปแบบคือ

1. ใช้เพื่อให้ฟังก์ชัน `testReturn` ทำงานเฉยๆ > เพื่อที่จะแสดงผล หรือดำเนินการอื่นๆภายในฟังก์ชันแล้วจบการทำงานเหมือนในบรรทัดที่ 12
2. ใช้เพื่อนำค่าที่ได้จากการส่งกลับไปใช้ประโยชน์ > อย่างเช่นในบรรทัดที่ 7 เป็นการนำค่า 112 (ASCII ของ `p`) กำหนดลงตัวแปร `test` และนำตัวแปร `test` ไปแสดงผลออกทางหน้าจอ หรือในบรรทัดที่ 10 เป็นการเรียกฟังก์ชันเพื่อรับค่าที่รีเทิร์นออกมาไปแสดงผลเลย

พอยเตอร์ (Pointer)

พอยเตอร์.... แค่นี้ก็จะอ้วกแตกตายแล้ว 5555 ยังเหลือเรื่องอ่าน-เขียนไฟล์นะถ้าอยากรู้ศึกษาเพิ่มเติม ถ้าหนังสือเล่มนี้พิมพ์ผิด หรือข้อมูลผิดพลาดตรงส่วนไหนพี่ ๆ ก็ขออภัยมา ณ ที่นี่จะ

มีตรงไหนสงสัยถามพี่ ๆ ได้เนอะ อย่างทำโจทย์ก็ทักมาขอได้เหมือนกันมี source ให้ไปฝึก ขอให้โชค A สนุกกับการอ่านหนังสือเล่มนี้ครับ :D

ภาคผนวก

คำศัพท์น่ารู้

หมวดระบบคอมพิวเตอร์

คำศัพท์	ความหมาย
Computer System	ระบบที่พัฒนามาจาก 2 ส่วนใหญ่ ๆ คือ ฮาร์ดแวร์ และซอฟต์แวร์ งานได้กำหนดไว้
Hardware	ส่วนประกอบของคอมพิวเตอร์ที่สามารถจับต้องได้
Software	โปรแกรมที่ทำงานอยู่ในระบบคอมพิวเตอร์ ซึ่งจะทำงานตามที่ผู้ใช้งานได้กำหนดไว้
Personal Computer	ระบบคอมพิวเตอร์ที่มีเครื่องคอมพิวเตอร์เพียงเครื่องเดียว และไม่ได้ทำการเชื่อมต่อกับเครื่องคอมพิวเตอร์เครื่องอื่น ๆ
Time-sharing	การเชื่อมต่อของคอมพิวเตอร์หลาย ๆ เครื่องมาต่อกันเครื่องคอมพิวเตอร์ศูนย์กลางโดยคอมพิวเตอร์เหล่านั้นเรียกว่า Terminal การทำงานของระบบคอมพิวเตอร์แบบนี้ คือ การประมวลผลที่อยู่ที่เครื่องคอมพิวเตอร์ศูนย์กลางเพียงเครื่องเดียว โดย Terminal ทุกเครื่องจะส่งคำสั่งที่ต้องการประมวลผลมาที่เครื่องคอมพิวเตอร์ศูนย์กลาง เพราะการประมวลผลที่เครื่องคอมพิวเตอร์ศูนย์กลางจะต้องมีการแบ่งเวลาในการประมวลคำสั่งต่าง ๆ ที่ส่งมาจากการ Terminal ทุกเครื่อง
Client/Server	เครื่องคอมพิวเตอร์เครื่องหนึ่ง ที่ทำหน้าที่เป็น Server อยู่แล้วจัดการทรัพยากรของระบบทั้งหมด และมีเครื่อง Clients ต่อเข้าที่เครื่อง Server โดยใช้ทรัพยากรต่าง ๆ ที่เครื่อง Server มีอยู่ ตามประสิทธิภาพของผู้ใช้แต่ละคน และการประมวลผลจะไม่ทำอยู่บนเครื่อง Server แต่จะประมวลผลที่เครื่อง Clients แต่ละเครื่องเอง และอาจจะส่งข้อมูลต่าง ๆ ที่ไปเก็บที่เครื่อง Server
Computer Languages	ภาษาที่ใช้สำหรับเขียน เพื่อสร้างโปรแกรมขึ้นมาเพื่อใช้งานบนเครื่องคอมพิวเตอร์
Compiler	โปรแกรมที่ทำหน้าที่ในการแปลง ภาษาระดับสูงให้เป็นภาษาเครื่อง
Loader	โปรแกรมที่ทำหน้าที่โหลดตัว Source Code ลงไปในหน่วยความจำ
System development Life Cycle	กระบวนการพัฒนาโปรแกรม หรือระบบขึ้นมาใช้งาน

หมวดโครงสร้างภาษา

คำศัพท์	ความหมาย
Compiled Language	ภาษาที่มีคอมไพล์เลอร์ (Compiler) ทำหน้าที่ในการคอมไพล์ (Compile) หรือแปลงคำสั่งทั้งหมดในโปรแกรมให้เป็นภาษาเครื่อง (Machine Language) เพื่อให้เครื่องคอมพิวเตอร์นำคำสั่งเหล่านั้นทำงานตามคำสั่งต่อไป
Header Files	ส่วนที่เก็บไลบรารีมาตรฐานของภาษา C ซึ่งจะถูกดึงเข้ามาร่วมกับโปรแกรมในขณะที่กำลังทำการคอมไพล์
Global Variables	ตัวแปรที่สามารถเรียกใช้ได้ทุกฟังก์ชันของโปรแกรม
Functions	ส่วนที่เก็บคำสั่งต่าง ๆ ไว้ซึ่งในภาษา C จะบังคับให้มีฟังก์ชันอย่างน้อย 1 ฟังก์ชัน นั่นก็คือฟังก์ชัน Main() และในโปรแกรม 1 โปรแกรมสามารถมีฟังก์ชันได้มากกว่า 1 ฟังก์ชัน
Local Variables	ตัวแปรที่จะสามารถเรียกใช้ได้ ภายในฟังก์ชันที่ประกาศตัวแปรนั้น
Return Value	เป็นส่วนที่บอกให้รู้ว่า ฟังก์ชันนี้จะมีการส่งค่าอะไรกลับไปให้กับฟังก์ชันที่เรียกฟังก์ชันนั้น
Identifier	เป็นกฎการตั้งชื่อตัวแปรและฟังก์ชัน
Constants	ชนิดข้อมูลแบบค่าคงที่ ผู้ใช้จะไม่สามารถเปลี่ยนแปลงค่าของตัวแปรนั้น ในขณะที่โปรแกรมทำงานอยู่
Statements	คำสั่งต่าง ๆ ที่ประกอบขึ้นจนเป็นตัวโปรแกรม
Flowchart	ผังงาน มีไว้เพื่อให้ผู้ใช้ออกแบบขั้นตอนการทำงานของโปรแกรมก่อนที่จะลงมือเขียนโปรแกรม ซึ่งจะช่วยให้ผู้ใช้เขียนโปรแกรมได้ง่ายขึ้น และไม่สับสน

หมวดการแสดงผล และรับข้อมูล

คำศัพท์	ความหมาย
Standard Input File	ไฟล์ที่ใช้เก็บข้อมูลก่อน ที่ได้รับจากผู้ใช้ผ่านทางคีย์บอร์ดเข้าไป
Standard Output File	ไฟล์ที่ใช้เก็บข้อมูลก่อน ที่จะนำไปแสดงออกทางอุปกรณ์ต่าง ๆ
Printf	เป็นคำสั่งที่ใช้สำหรับแสดงค่าต่าง ๆ ออกทางจอภาพ
Back-slash character	ชุดค่าคงที่ของตัวอักษร ที่จะใช้ในการควบคุมการพิมพ์และแสดงเครื่องหมายบางอย่างที่ไม่สามารถพิมพ์เครื่องหมายนั้นตรง ๆ
Scanf	คำสั่งในการรับค่าข้อมูลจากผู้ใช้เข้ามาทางคีย์บอร์ด
Getch	คำสั่งในการรับค่าข้อมูลจากผู้ใช้เข้ามาทางคีย์บอร์ด 1 ตัวอักษร เครื่อร์เชอร์จะไม่ขึ้นบรรทัดใหม่ และไม่ต้องกดปุ่ม Enter เมื่อป้อนครบ 1 ตัวอักษร นอกจกนี้ตัวอักษรที่ป้อนลงไปจะไม่แสดงผลออกทางจอภาพ
Getche	เป็นฟังก์ชันที่ทำงานเหมือนฟังก์ชัน getch แต่จะแสดงตัวอักษรที่พิมพ์เข้าไป ออกทางจอภาพด้วย
Getchar	คำสั่งในการรับค่าข้อมูลจากผู้ใช้เข้ามาทางคีย์บอร์ด 1 ตัวอักษร ต้องกดปุ่ม Enter เพื่อสิ้นสุดการรับข้อมูล ตัวอักษรที่ป้อนลงไปจะแสดงผลออกทางจอภาพ และเครื่อร์เชอร์จะขึ้นบรรทัดใหม่

หมวดนิพจน์ทางคณิตศาสตร์

คำศัพท์	ความหมาย
Expressions	นิพจน์ต่าง ๆ
Operand	ตัวถูกดำเนินการ หรือตัวแปรต่าง ๆ เป็นต้น
Operator	ตัวดำเนินการ เป็นเครื่องหมายต่าง ๆ
Precedence	ลำดับความสำคัญของตัวดำเนินการ
Simple Assignments	การกำหนดค่าแบบง่าย
Compound Assignments	การกำหนดค่าแบบผสม

หมวดฟังก์ชันของภาษา C

คำศัพท์	ความหมาย
Function	เป็นตัวโปรแกรมส่วนหนึ่ง ที่นำไปเขียนแยกไว้ ถ้าจะใช้จะต้องมีการเรียกใช้ โดยในฟังก์ชันจะมีพารามิเตอร์ที่ให้รับค่าจากฟังก์ชันที่เรียกใช้ และอาจจะมีค่าที่ส่ง หรือ Return Value
Function Header	เป็นส่วนหัวของฟังก์ชันซึ่งประกอบไปด้วย 3 ส่วนหลัก คือ ชนิดข้อมูลของค่าที่ส่งกลับ (Return Type), ชื่อฟังก์ชัน (Function Name) และรายชื่อพารามิเตอร์
Function Body	ส่วนที่ประกอบขึ้นเป็นฟังก์ชัน การประกาศตัวแปรแบบ Local และชุดคำสั่งต่าง ๆ ของฟังก์ชันนั้น
Prototype Declarations	การประกาศฟังก์ชัน โดยจะประกาศอยู่ตรงส่วนบนสุดของโปรแกรม หรือก่อนส่วนการประกอบตัวแปรแบบ Global
Function Call	เป็นการเรียกใช้ฟังก์ชัน
Pass by Value	การส่งค่าแบบกำหนดค่า
Pass by Reference	การส่งค่าแบบอ้างอิง
Abs	ฟังก์ชันที่ส่งกลับค่า Absolute ของตัวเลขที่รับเข้ามา Absolute คือ ค่าทางบวกของค่านั้นโดยชนิดพารามิเตอร์เป็นตัวเลขจำนวนเต็มหรือ Integer
Ceil	ฟังก์ชันที่จะหาค่าจำนวนเต็มที่มากกว่า หรือเท่ากับค่าที่ส่งไปให้
Floor	ฟังก์ชันที่จะหาค่าจำนวนเต็มที่น้อยกว่า หรือเท่ากับค่าที่ส่งไปให้

หมวดคำสั่งเงื่อนไข

คำศัพท์	ความหมาย
Logical Data	ข้อมูลตรรกะ
Not	คำสั่งที่ใช้กลับค่าความจริง
And	ใช้เชื่อม 2 เงื่อนไข หรือมากกว่า ผลลัพธ์จะเป็นจริงเมื่อเงื่อนไขทั้ง 2 หรือทั้งหมดเป็นจริง และจะให้เป็นเท็จเมื่อมีอย่างน้อย 1 เงื่อนไขเป็นเท็จ
Or	ใช้เชื่อม 2 เงื่อนไข หรือมากกว่า ผลลัพธ์จะเป็นจริงเมื่อมีอย่างน้อย 1 เงื่อนไขเป็นจริง และ เป็นเท็จเมื่อเงื่อนไขทั้ง 2 หรือทั้งหมดเป็นเท็จ
if... else	เป็นคำสั่ง 2 ทางเลือก ซึ่งถ้าเงื่อนไขเป็นจริงจะทำคำสั่งที่กำหนดไว้ แต่ถ้าเงื่อนไขเป็นเท็จ จะไปทำอีกคำสั่ง
if	เป็นคำสั่ง 2 ทางเลือกันหนึ่งซึ่งถ้าเงื่อนไขเป็นจริงจะทำคำสั่งกำหนดไว้ แต่ถ้าเงื่อนไข เป็นเท็จเป็นจะไม่ทำอะไร
Nested if	เป็นคำสั่ง if... else ที่มีคำสั่ง if... else หรือคำสั่ง if ข้อนอยู่ด้านในอีกด้านหนึ่ง
switch	เป็นคำสั่งหลายทางเลือก ซึ่งเป็นคำสั่งที่แปลงมาจากคำสั่ง Nested if คำสั่งนี้จะมีตัวแปรหนึ่งที่ใช้หาว่าคำสั่งไหน
case	เป็นส่วนหนึ่งของคำสั่ง Switch คือเป็น
else – if	เป็นคำสั่งที่ใช้แทนคำสั่ง Switch นั้นจะต้องเป็นชนิดข้อมูลที่เป็นจำนวนเต็มเท่านั้น แต่ถ้า เมื่อไรที่ต้องใช้ตัวแปรที่เป็นชนิดข้อมูลที่คนไม่สามารถใช้คำสั่ง Switch ได้แต่ ภาษา C ก็ได้มีคำสั่งอีกคำสั่งหนึ่งที่เป็นคำสั่งหลายทางเลือกและสามารถใช้ได้กับ ชนิดข้อมูลทุกประเภทคำสั่งนั้นก็คือ else – if ซึ่งชุดคำสั่งเหมือนกับคำสั่ง if...else แต่ต่างกันตรงที่ในคำสั่ง else ใช้ต่อด้วยคำสั่ง if ได้เลย

หมวดคำสั่งวนลูป

คำศัพท์	ความหมาย
Loop	การวนทำคำสั่งเดิมหลาย ๆ ครั้ง
Pretest	ลูปประเพณีนี้ จะทำการตรวจสอบเงื่อนไขก่อนว่าเป็นจริง หรือเป็นเท็จ ถ้าเป็นจริงก็ให้ เข้าไปทำคำสั่งหรือชุดคำสั่งต่อไป และเมื่อทำคำสั่งหรือชุดคำสั่งเสร็จแล้วก็จะกลับมาทำการ ตรวจสอบเงื่อนไขอีก ครั้งและจะทำเช่นนี้ไปเรื่อย ๆ จนกว่าเงื่อนไขจะเป็นเท็จ ก็จบการ ทำงานของลูป
Post – Test	ลูปประเพณีนี้ จะทำคำสั่งหรือชุดคำสั่งก่อน เมื่อเสร็จแล้วถึงจะมาตรวจสอบเงื่อนไขว่า เป็นจริงหรือเป็นเท็จ ถ้าเป็นจริงก็จะกลับไปทำคำสั่งหรือชุดคำสั่งเดิมอีกครั้ง และจะทำ จนกว่าเงื่อนไขจะเป็นเท็จเขียนเดียวกัน
while	ลูป while นี้ใช้เงื่อนไขเป็นตัวควบคุมลูป และเป็นลูปแบบ Pretest Loop ซึ่งจะทำการ ตรวจสอบเงื่อนไขก่อนที่จะเข้าไปทำคำสั่งในลูป
for	คำสั่ง for นั้นจะเป็นลูปแบบ Pretest Loop ที่ใช้นิพจน์ 3 นิพจน์ นิพจน์แรกเป็นการ กำหนดค่า นิพจน์ที่ 2 เป็นเงื่อนไขในการตรวจสอบตัวควบคุมลูป
do...while	คำสั่ง do...while เป็นลูปแบบ Post – Test Loop ลูปแบบนี้จะมีคำสั่งก่อนที่จะไปทำการ ตรวจสอบตัวควบคุมลูป
break	เป็นคำสั่งในการกระโดดออกจากชุดคำสั่ง Switch และมันก็สามารถนำมาใช้ในคำสั่งวนลูป ได้เหมือนกันเพื่อใช้กระโดดออกจากลูปใน กรณีต่าง ๆ ได้
continue	คำสั่งนี้จะไม่ได้กระโดดออกจากลูปเลย แต่จะกระโดดคำสั่งอื่น ๆ ในลูปไปทำการ ตรวจสอบตามนิพจน์เลย

ASCII TABLE

ผังอักขระและสกีที่ไม่สามารถแสดงผล

อักขระที่ปรากฏในตารางเป็นเพียงการแสดงว่า ณ ตำแหน่งนั้นมีรหัสตั้งกล่าวอยู่ ไม่ใช่สัญลักษณ์ที่จะนำมาแสดงผลเป็นหลัก ซึ่งใช้เป็นรหัสควบคุมการพิมพ์บนเครื่องพิมพ์ หรือตัวแบ่งข้อมูลในสื่อบันทึกข้อมูลบางชนิด (เช่นเทป)

BIN	DEC	HEX	CHA	ความหมาย
0000 0000	0	00	(ว่าง)	NUL - null character
0000 0001	1	01	☺	SOH - start of heading
0000 0010	2	02	☻	STX - start of text
0000 0011	3	03	♥	ETX - end of text
0000 0100	4	04	♦	EOT - end of transmission
0000 0101	5	05	♣	ENQ - enquiry
0000 0110	6	06	♠	ACK - acknowledge
0000 0111	7	07	•	BEL - bell
0000 1000	8	08	▣	BS - backspace
0000 1001	9	09	○	HT - horizontal tabulation
0000 1010	10	0A	▣	LF - line feed
0000 1011	11	0B	♂	VT - vertical tabulation
0000 1100	12	0C	♀	FF - form feed
0000 1101	13	0D	♪	CR - carriage return
0000 1110	14	0E	♫	SO - shift out
0000 1111	15	0F	☼	SI - shift in

BIN	DEC	HEX	CHA	ความหมาย
0001 0000	16	10	▶	DLE - data link escape
0001 0001	17	11	◀	DC1 - device control one
0001 0010	18	12	↑	DC2 - device control two
0001 0011	19	13	!!	DC3 - device control three
0001 0100	20	14	¶	DC4 - device control four
0001 0101	21	15	§	NAK - negative
0001 0110	22	16	—	SYN - synchronous idle
0001 0111	23	17	↑	ETB - end of transmission
0001 1000	24	18	↑	CAN - cancel
0001 1001	25	19	↓	EM - end of medium
0001 1010	26	1A	→	SUB - substitute
0001 1011	27	1B	←	ESC - escape
0001 1100	28	1C	∟	FS - file separator
0001 1101	29	1D	↔	GS - group separator
0001 1110	30	1E	▲	RS - record separator
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	~