



เขียนโปรแกรมภาษา C

สำหรับผู้เริ่มต้น

แนะนำเนื้อหา

ภาษา C คือ ภาษาคอมพิวเตอร์ใช้สำหรับพัฒนาโปรแกรม โดยผู้พัฒนาต้องเรียนรู้โครงสร้างภาษา C ว่ามีโครงสร้างการเขียนอย่างไรเพื่อสั่งการให้คอมพิวเตอร์นั้นทำงานตามวัตถุประสงค์ที่ต้องการ เพื่อให้ผู้พัฒนาโปรแกรมนั้นสามารถสื่อสารกับคอมพิวเตอร์ได้ต้องอาศัยส่วนที่เรียกว่า **ตัวแปลภาษา**

ตัวแปลภาษา

ตัวแปลภาษาเปรียบเสมือนกับล่ามทำหน้าที่แปลงโค้ดภาษาคอมพิวเตอร์ที่มนุษย์เขียนขึ้น (ภาษา C) ไปเป็นภาษาที่เครื่องคอมพิวเตอร์เข้าใจว่าต้องการให้ทำงานอะไร



ประเภทของตัวแปลภาษา

ในปัจจุบันตัวแปลภาษาแบ่งออกเป็น 2 ประเภท

- คอมไพเลอร์ (Compiler)
- อินเตอร์พรีเตอร์ (Interpreter)

ภาษา C จัดเป็นภาษาคอมพิวเตอร์ที่ใช้ตัวแปลภาษาแบบคอมไพเลอร์

	ข้อดี	ข้อเสีย
Compiler	<ul style="list-style-type: none"> ทำงานได้เร็ว เนื่องจากจะทำการแปลคำสั่งทั้งหมดในครั้งเดียว แล้วจึงทำงานตามคำสั่งของโปรแกรมในภายหลัง 	<ul style="list-style-type: none"> เมื่อเกิดข้อผิดพลาดขึ้นจะตรวจสอบหาข้อผิดพลาดได้ยาก เพราะทำการแปลคำสั่งทีเดียวทั้งโปรแกรม
Interpreter	<ul style="list-style-type: none"> แปลคำสั่งทีละบรรทัด ทำให้หาข้อผิดพลาดของโปรแกรมได้ง่าย เนื่องจากแปลคำสั่งทีละบรรทัด สามารถสั่งให้โปรแกรมทำงานเฉพาะจุดได้ ไม่เสียเวลาการแปลคำสั่งเป็นเวลานาน 	<ul style="list-style-type: none"> ช้า เนื่องจากทำงานทีละบรรทัด

โครงสร้างคำสั่งภาษา C

```
#include <stdio.h>

int main()
{
    printf("Hello World");
    return 0;
}
```

องค์ประกอบพื้นฐานของภาษา C

- ไฟล์ที่เก็บโค้ดภาษา C จะมีนามสกุลไฟล์ .c
- ฟังก์ชัน `main()` คือ ฟังก์ชันพิเศษ กลุ่มคำสั่งที่อยู่ในฟังก์ชันนี้ จะทำงานโดยอัตโนมัติในตอนเริ่มต้นเสมอ

องค์ประกอบพื้นฐานของภาษา C

- **ขอบเขต (Block)** ใช้สัญลักษณ์ {} เพื่อบอกขอบเขตการทำงานของ
ของกลุ่มคำสั่งว่ามีจุดเริ่มต้นและสิ้นสุดที่ตำแหน่งใด
- **เครื่องหมายสิ้นสุดคำสั่ง** ใช้สัญลักษณ์ ;
- **คำอธิบาย (Comment)** ใช้สัญลักษณ์ // หรือ /* */

โครงสร้างคำสั่งภาษา C

คำสั่งของภาษา C ที่เขียนในไฟล์นามสกุล .c จะถูก
เรียกมาจากส่วนที่เรียกว่าไลบรารี (Library)



โครงสร้างคำสั่งภาษา C

ไลบรารี (Library) ประกอบด้วย 3 องค์ประกอบ ได้แก่

- **เครื่องหมาย # (Preprocessor)** คือการกำหนดให้ตัวแปลภาษา C ทราบว่าต้องแปลความหมายของไลบรารีอะไร
- **ไคเรกทีฟ (Directive)** คำสั่งที่ต่อจาก # คือการ include หรืออ้างอิงไฟล์ไลบรารี ที่อยู่ระหว่างเครื่องหมาย < กับ >
- **Header File** หมายถึงชื่อไลบรารีที่ต้องการอ้างอิง เช่น `stdio.h`

โครงสร้างคำสั่งภาษา C

```
#include <stdio.h>

int main()
{
    printf("Hello World");
    return 0;
}
```

โครงสร้างคำสั่งภาษา C

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

นำคำสั่งพื้นฐานที่อยู่ในไลบรารี stdio เข้ามาทำงาน เช่น กลุ่มคำสั่งที่ต้องการแสดงผลออกทางจอภาพ (printf) เป็นต้น

โครงสร้างคำสั่งภาษา C

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

```
int main()
```

```
{
```

```
    printf("Hello World"); //แสดงผลข้อความ Hello World ออกทางจอภาพ
```

```
    return 0;
```

```
}
```

หมายเหตุ (Comment)

จุดประสงค์

- อธิบายหน้าที่หรือความหมายของโค้ดที่เขียน
- ยกเลิกโค้ดชั่วคราว ส่งผลให้ตัวแปลภาษาไม่สนใจโค้ดในบรรทัดที่ถูกทำหมายเหตุ



หมายเหตุ (Comment)

วิธีที่ 1 โดยใช้เครื่องหมาย Slash (/) ใช้ในการอธิบายคำสั่งสั้นๆ
ในรูปแบบบรรทัดเดียว

วิธีที่ 2 เขียนคำอธิบายไว้ในเครื่องหมาย /* ... */ ใช้ในการอธิบายคำสั่ง
ยาวๆหรือแบบหลายบรรทัด

แสดงผลข้อมูล (Output)

คือคำสั่งสำหรับใช้แสดงผลข้อมูลออกจากจอภาพทั้งข้อมูลที่อยู่ในรูปแบบ ตัวเลขและตัวอักษรหรือผลลัพธ์จากการประมวลผล มีโครงสร้างคำสั่งดังนี้

```
printf("Format String",List Of Data)
```


แสดงผลข้อมูล (Output)

- **Format String** คือ ชุดข้อความพิเศษสำหรับกำหนดรูปแบบการแสดงผลข้อมูล
- **List Of Data** คือ รายการข้อมูลต่างๆที่ต้องการแสดงผล ซึ่งสามารถแสดงข้อมูลได้มากกว่า 1 รายการโดยคั่นด้วยเครื่องหมายคอมม่า (,)

แสดงผลข้อมูล (Output)

Format String	คำอธิบาย
%d	ข้อมูลตัวเลขจำนวนเต็ม
%f	ข้อมูลตัวเลขที่มีจุดทศนิยม
%c	ข้อมูลตัวอักษร
%s	ข้อมูลข้อความ มีลักษณะเป็นชุดของตัวอักษร ตัวเลข หรือ อักขระพิเศษ อยู่ในพื้นที่เครื่องหมาย Double Quote (“ ”)

แสดงผลข้อมูล (Output)

อักขระควบคุมการแสดงผล	คำอธิบาย
\n	ขึ้นบรรทัดใหม่
\t	เว้นช่องว่างในแนวนอน

ตัวแปรและชนิดข้อมูล

ตัวแปร (Variable) คือ ชื่อที่ถูกนิยามขึ้นมาเพื่อใช้เก็บค่าข้อมูลสำหรับนำไปใช้งานในโปรแกรม โดยข้อมูลประกอบด้วย ข้อความ ตัวเลข ตัวอักษร หรือผลลัพธ์จากการประมวลผลข้อมูลค่าที่เก็บในตัวแปร สามารถเปลี่ยนแปลงค่าได้

ชนิดข้อมูลพื้นฐาน (Data Type)

ชนิดข้อมูล	คำอธิบาย
boolean	ค่าทางตรรกศาสตร์ (True = 1 / False = 0)
int	ตัวเลขที่ไม่มีจุดทศนิยม
float	ตัวเลขที่มีจุดทศนิยม
char	ตัวอักษร (ใช้เครื่องหมาย ‘ ’)
string	กลุ่มตัวอักษรหรือข้อความ (ใช้เครื่องหมาย “ ”)

การสร้างตัวแปร

- ชนิดข้อมูล ชื่อตัวแปร ;
- ชนิดข้อมูล ชื่อตัวแปร = ค่าเริ่มต้น;
- ชนิดข้อมูล ชื่อตัวแปร1 , ชื่อตัวแปร2 ;

ให้นำค่าทางขวามือของเครื่องหมาย = ไปเก็บไว้ในตัวแปรที่อยู่ด้านซ้ายมือ

กฎการตั้งชื่อ

- ขึ้นต้นด้วยตัวอักษร A-Z หรือ a-z หรือ _ เครื่องหมายขีดเส้นใต้เท่านั้น
- อักษรตัวแรกห้ามเป็นตัวเลข
- ตัวพิมพ์เล็ก-พิมพ์ใหญ่มีความหมายต่างกัน (Case Sensitive)
- ห้ามใช้อักขระพิเศษมาประกอบเป็นชื่อตัวแปร เช่น {}, %, ^ และช่องว่าง เป็นต้น
- ไม่สามารถประกาศชื่อเดียวกัน แต่มีชนิดข้อมูล 2 ชนิดได้
- ไม่ซ้ำกับคำสงวนในภาษา C

คำสงวนในภาษา C (Keywords)

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

ค่าคงที่ (Constant)

มีลักษณะการใช้งานคล้ายกับตัวแปร แต่ค่าคงที่
คือค่าที่ไม่สามารถเปลี่ยนแปลงได้ ตอนประกาศใช้งาน
ค่าคงที่จะต้องมีการประกาศค่าเริ่มต้นเสมอ

ประเภทของค่าคงที่ (Constant)

- Literal Constant
- Defined Constant
- Memory Constant

Literal Constant

คือ ค่าคงที่ซึ่งเป็นข้อมูลที่แน่นอน ไม่จำเป็นต้องมีตัวแปรมารองรับ สามารถกำหนดเข้าไปในโปรแกรมได้เลย เช่น

```
printf("Hello %s", "KongRuksiam")
```

```
printf(" 2 x 2 = %d" , 4)
```

Defined Constant

คือ ค่าคงที่ซึ่งประกาศไว้ที่ส่วนหัวของโปรแกรมใน
ลักษณะ Preprocessing Directives เช่น

```
#define MAX_VALUE 100
```

```
int main(){
```

```
    printf("MAX = %d",MAX_VALUE)
```

```
}
```

Memory Constant

เป็นการกำหนดค่าคงที่ให้ตัวแปร ส่งผลให้ตัวแปรที่ถูกกำหนดค่านั้น ไม่สามารถแปรเปลี่ยนค่าได้ ตลอดการทำงานของโปรแกรม ตัวอย่าง เช่น

```
const float PI = 3.14 ;
```

```
const int SIZE = 10;
```

การรับข้อมูล (Input)

คือ คำสั่งสำหรับรับค่าผ่านทางคีย์บอร์ดและเก็บค่าดังกล่าวลงในตัวแปรที่มีโครงสร้างคำสั่ง ดังนี้

`scanf("Format String",List Of Address)`

การรับข้อมูล (Input)

- **Format String** คือ ชุดข้อความพิเศษสำหรับกำหนดรูปแบบการรับข้อมูลจากผู้ใช้
- **List Of Address** คือ ตำแหน่งของตัวแปรในหน่วยความจำ โดยทั่วไปจะอยู่ในรูปของตัวแปรที่มีเครื่องหมาย & นำหน้า เช่น ตำแหน่งของตัวแปร name คือ &name

การรับข้อมูล (Input)

Format String	คำอธิบาย
%d	เก็บข้อมูลตัวเลขจำนวนเต็ม
%f	เก็บข้อมูลตัวเลขที่มีจุดทศนิยม
%c	เก็บข้อมูลตัวอักษร
%s	เก็บข้อมูลกลุ่มตัวอักษร (ข้อความ)

ตัวดำเนินการ (Operator)

กลุ่มของเครื่องหมายหรือสัญลักษณ์ที่ใช้ในการเขียนโปรแกรม

$$A+B$$

- ตัวดำเนินการ (Operator)
- ตัวถูกดำเนินการ (Operand)

ตัวดำเนินการทางคณิตศาสตร์

Operator	คำอธิบาย
+	บวก
-	ลบ
*	คูณ
/	หาร
%	หารเอาเศษ

ตัวดำเนินการเพิ่มค่าและลดค่า

Operator	รูปแบบการเขียน	ความหมาย
++ (Prefix)	++a	เพิ่มค่าให้ a ก่อน 1 ค่าแล้วนำไปใช้
++ (Postfix)	a++	นำค่าปัจจุบันใน a ไปใช้ก่อนแล้วค่อยเพิ่มค่า
-- (Prefix)	--b	ลดค่าให้ b ก่อน 1 ค่าแล้วนำไปใช้
-- (Postfix)	b--	นำค่าปัจจุบันใน b ไปใช้ก่อนแล้วค่อยลดค่า

Compound Assignment

Assignment	รูปแบบการเขียน	ความหมาย
<code>+=</code>	<code>x+=y</code>	<code>x=x+y</code>
<code>-=</code>	<code>x-=y</code>	<code>x=x-y</code>
<code>*=</code>	<code>x*=y</code>	<code>x=x*y</code>
<code>/=</code>	<code>x/=y</code>	<code>x=x/y</code>
<code>%=</code>	<code>x%=y</code>	<code>x=x%y</code>

ตัวดำเนินการเปรียบเทียบ

ผลการเปรียบเทียบจะได้คำตอบ คือ True (1) หรือ False (0)

Operator	คำอธิบาย
==	เท่ากับ
!=	ไม่เท่ากับ
>	มากกว่า
<	น้อยกว่า
>=	มากกว่าเท่ากับ
<=	น้อยกว่าเท่ากับ

ลำดับความสำคัญของตัวดำเนินการ

ลำดับที่	เครื่องหมาย	ลำดับการทำงาน
1	()	
2	++ , --	ซ้ายไปขวา
3	* , / , %	ซ้ายไปขวา
4	+ , -	ซ้ายไปขวา
5	< , <= , > , >=	ซ้ายไปขวา
6	== , !=	ซ้ายไปขวา
7	&& (AND)	ซ้ายไปขวา
8	(OR)	ซ้ายไปขวา
9	= , += , -= , *= , /= , %=	ขวาไปซ้าย

ลำดับความสำคัญของตัวดำเนินการ

ลำดับที่	เครื่องหมาย	ลำดับการทำงาน
1	()	
2	++ , --	ซ้ายไปขวา
3	* , / , %	ซ้ายไปขวา
4	+ , -	ซ้ายไปขวา
5	< , <= , > , >=	ซ้ายไปขวา
6	== , !=	ซ้ายไปขวา
7	&& (AND)	ซ้ายไปขวา
8	(OR)	ซ้ายไปขวา
9	= , += , -= , *= , /= , %=	ขวาไปซ้าย

กรณีศึกษา (Case Study)

1. $2+8 \times 9 = ?$
2. $10 - 4+2 = ?$
3. $10 \times (2+5) = ?$
4. $5 \times 2 - 40 / 5 = ?$
5. $7 + 8 / 2 + 25 = ?$

กรณีศึกษา (Case Study)

1. $2+8\times 9 = 74$

2. $10 - 4+2 = 8$

3. $10 \times (2+5) = 70$

4. $5 \times 2 - 40 / 5 = 2$

5. $7 + 8 / 2 + 25 = 36$

โครงสร้างควบคุม (Control Structure)

คือ กลุ่มคำสั่งที่ใช้ควบคุมการทำงานของโปรแกรม

- แบบลำดับ (Sequence)
- แบบมีเงื่อนไข (Condition)
- แบบทำซ้ำ (Loop)

โครงสร้างควบคุม (Control Structure)

คือ กลุ่มคำสั่งที่ใช้ควบคุมการทำงานของโปรแกรม

- แบบลำดับ (Sequence)
- แบบมีเงื่อนไข (Condition)
- แบบทำซ้ำ (Loop)

แบบมีเงื่อนไข (Condition)

กลุ่มคำสั่งที่ใช้ตัดสินใจในการเลือกทำงานตามเงื่อนไขต่างๆ ภายในโปรแกรม

- if
- Switch..Case

รูปแบบคำสั่งแบบเงื่อนไขเดียว

If Statement

เป็นคำสั่งที่ใช้กำหนดเงื่อนไขในการตัดสินใจทำงานของโปรแกรม ถ้าเงื่อนไขเป็นจริงจะทำตามคำสั่งต่างๆที่กำหนดภายใต้เงื่อนไขนั้นๆ

รูปแบบคำสั่งแบบเงื่อนไขเดียว

```
if(condition){ //เงื่อนไข  
    คำสั่งเมื่อเงื่อนไขเป็นจริง ;  
}
```

โจทย์ปัญหา

คำนวณคะแนนสอบวิชาคอมพิวเตอร์ของนักเรียนในห้องโดย
มีคะแนนเต็ม 100 คะแนน ต้องการอยากรทราบว่านักเรียนคนใด
สอบผ่านบ้างใช้เกณฑ์วัดผลดังนี้

- คะแนนตั้งแต่ 50 คะแนนขึ้นไป => สอบผ่าน



If...Else Statement

```
if(เงื่อนไข){  
    คำสั่งเมื่อเงื่อนไขเป็นจริง ;  
}  
else{  
    คำสั่งเมื่อเงื่อนไขเป็นเท็จ ;  
}
```


โจทย์ปัญหา

คำนวณคะแนนสอบวิชาคอมพิวเตอร์ของนักเรียนในห้องโดย
มีคะแนนเต็ม 100 คะแนน ต้องการอยากรทราบว่านักเรียนคนใด
สอบผ่านบ้างใช้เกณฑ์วัดผลดังนี้

- คะแนนตั้งแต่ 50 คะแนนขึ้นไป => สอบผ่าน
- คะแนนน้อยกว่า 50 คะแนน => สอบไม่ผ่าน



รูปแบบคำสั่งแบบหลายเงื่อนไข

```
if(เงื่อนไขที่ 1){  
    คำสั่งเมื่อเงื่อนไขที่ 1 เป็นจริง ;  
}else if(เงื่อนไขที่ 2){  
    คำสั่งเมื่อเงื่อนไขที่ 2 เป็นจริง ;  
}else if(เงื่อนไขที่ 3){  
    คำสั่งเมื่อเงื่อนไขที่ 3 เป็นจริง ;  
}else{  
    คำสั่งเมื่อทุกเงื่อนไขเป็นเท็จ ;  
}
```

if..else แบบลดรูป (Ternary Operator)

```
if(เงื่อนไข){  
    คำสั่งเมื่อเงื่อนไขเป็นจริง ;  
}  
else{  
    คำสั่งเมื่อเงื่อนไขเป็นเท็จ ;  
}
```

ตัวแปร = เงื่อนไข ? คำสั่งเมื่อเงื่อนไขเป็นจริง : คำสั่งเมื่อเงื่อนไขเป็นเท็จ;

โจทย์ปัญหา : โปรแกรมตัดเกรดอย่างง่าย

คำนวณเกรดจากคะแนนสอบของนักเรียน
โดยมีเกณฑ์วัดผลดังนี้ คือ

- คะแนนตั้งแต่ 50 ขึ้นไป ได้เกรด A
- น้อยกว่า 50 คะแนน ได้เกรด F



ตัวดำเนินการทางตรรกศาสตร์

Operator	คำอธิบาย
&& (AND)	และ
(OR)	หรือ
! (NOT)	ไม่

ตัวดำเนินการทางตรรกศาสตร์

A	!A	A	B	A && B	A B
true	false	false	false	false	false
false	true	false	true	false	true
		true	false	false	true
		true	true	true	true

โจทย์ปัญหา : เกณฑ์การรับสมัครงาน

AND (และ) , &&

ผู้สมัครเป็นเพศชาย **และ** มีอายุ 20 ปีเป็นต้นไป (ผ่านเกณฑ์)

OR (หรือ) , ||

ผู้สมัครเป็นเพศชาย **หรือ** มีอายุ 20 ปีเป็นต้นไป (ผ่านเกณฑ์)

NOT (ไม่) , !

ผู้สมัคร**ไม่ได้**เป็นเพศชาย **และ** มีอายุ 20 ปีเป็นต้นไป

ตัวดำเนินการทางตรรกศาสตร์

- ให้ $A =$ เงื่อนไขที่ 1 (เพศ)
- ให้ $B =$ เงื่อนไขที่ 2 (อายุ)



ตัวดำเนินการทางตรรกศาสตร์

A	!A	A	B	A && B	A B
true	false	false	false	false	false
false	true	false	true	false	true
		true	false	false	true
		true	true	true	true

ตัวดำเนินการทางตรรกศาสตร์

เพศ (A)	!A	A	อายุ (B)	A && B	A B
ชาย	หญิง	หญิง	<20	ไม่ผ่านเกณฑ์	ไม่ผ่านเกณฑ์
หญิง	ชาย	หญิง	>=20	ไม่ผ่านเกณฑ์	ผ่านเกณฑ์
		ชาย	<20	ไม่ผ่านเกณฑ์	ผ่านเกณฑ์
		ชาย	>=20	ผ่านเกณฑ์	ผ่านเกณฑ์

โจทย์ปัญหา : โปรแกรมตัดเกรดอย่างง่าย

คำนวณเกรดจากคะแนนสอบของนักเรียน คะแนนเต็ม 100 คะแนน โดยมีเกณฑ์วัดผลดังนี้ คือ

- คะแนน 80 - 100 ได้เกรด A
- คะแนน 70 - 79 ได้เกรด B
- คะแนน 60 - 69 ได้เกรด C
- คะแนน 0 - 59 คะแนน ได้เกรด F
- ป้อนค่าอื่น ได้เกรด N (Input Error)



Nested-If

```
if(เงื่อนไขหลัก){  
    if(เงื่อนไขย่อยที่ n ){  
        คำสั่งเมื่อเงื่อนไขย่อยที่ n เป็นจริง ;  
    }  
}
```

โจทย์ปัญหา : ใช้บริการแอปธนาคาร

- ผู้ใช้บริการต้อง Login เข้าสู่ระบบ โดยป้อนข้อมูลดังนี้
 - username : member
 - password : 1234
- กรณีที่ป้อนข้อมูลถูกต้องจะสามารถใช้บริการธนาคารได้
 - ป้อนหมายเลข 1 : ถอนเงิน (withdraw)
 - ป้อนหมายเลข 2 : ฝากเงิน (deposit)
 - ป้อนหมายเลขอื่น : ออกจากระบบ
- กรณีที่ป้อนข้อมูลไม่ถูกต้องระบบจะแจ้งว่าไม่พบบัญชี

แบบมีเงื่อนไข (Condition)

กลุ่มคำสั่งที่ใช้ตัดสินใจในการเลือกทำงานตามเงื่อนไขต่างๆ ภายในโปรแกรม

- if
- Switch..Case

Switch..Case

Switch เป็นคำสั่งที่ใช้กำหนดเงื่อนไขคล้ายๆ กับ if แต่จะเลือกเพียงหนึ่งทางเลือกออกมาทำงานโดยนำค่าในตัวแปรมากำหนดเป็นทางเลือกผ่านคำสั่ง case (ตัวแปรควบคุม)

รูปแบบคำสั่ง

```
switch (ค่าที่เก็บในตัวแปรควบคุม) {
```

```
    case ค่าที่ 1 : คำสั่งที่ 1;
```

```
        break;
```

```
    case ค่าที่ 2 : คำสั่งที่ 2;
```

```
        break;
```

```
    .....
```

```
    case ค่าที่ N : คำสั่งที่ N;
```

```
        break;
```

```
    default : คำสั่งเมื่อไม่มีค่าที่ตรงกับที่ระบุใน case
```

```
}
```

*****คำสั่ง break**

จะทำให้โปรแกรมกระโดด
ออกไปทำงานนอกคำสั่ง switch
ถ้าไม่มีคำสั่ง break โปรแกรมจะทำ
คำสั่งต่อไปเรื่อยๆ จนจบการทำงาน

โจทย์ปัญหา : ใช้บริการธนาคาร

ป้อนหมายเลขเพื่อใช้บริการ

- หมายเลข 1 : เปิดบัญชีใหม่ (create account)
- หมายเลข 2 : ถอนเงิน (withdraw)
- หมายเลข 3 : ฝากเงิน (deposit)
- หมายเลขอื่น : ข้อมูลไม่ถูกต้อง (Invalid)

โครงสร้างควบคุม (Control Structure)

คือ กลุ่มคำสั่งที่ใช้ควบคุมการทำงานของโปรแกรม

- แบบลำดับ (Sequence)
- แบบมีเงื่อนไข (Condition)
- แบบทำซ้ำ (Loop)

โครงสร้างควบคุม (Control Structure)

คือ กลุ่มคำสั่งที่ใช้ควบคุมการทำงานของโปรแกรม

- แบบลำดับ (Sequence)
- แบบมีเงื่อนไข (Condition)
- แบบทำซ้ำ (Loop)

แบบทำซ้ำ (Loop)

กลุ่มคำสั่งที่ใช้ในการวนรอบ (Loop) โปรแกรมจะทำงานไปเรื่อยๆจนกว่าเงื่อนไขที่กำหนดไว้จะเป็นเท็จ จึงจะหยุดทำงาน

- While
- For
- Do..While

แบบทำซ้ำ (Loop)

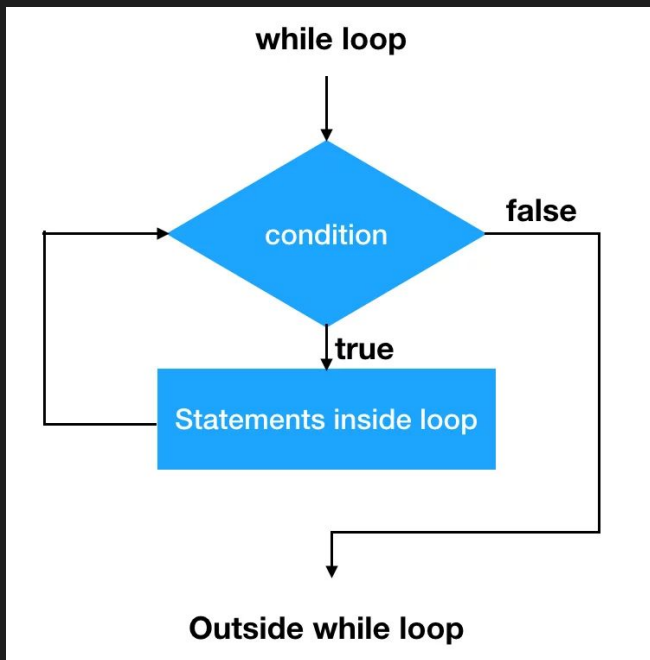
กลุ่มคำสั่งที่ใช้ในการวนรอบ (Loop) โปรแกรมจะทำงานไปเรื่อยๆจนกว่าเงื่อนไขที่กำหนดไว้จะเป็นเท็จ จึงจะหยุดทำงาน

- While
- For
- Do..While

While Loop

จะทำงานตามคำสั่งภายใน while ไปเรื่อยๆเมื่อเงื่อนไขที่กำหนดเป็นจริง

```
while (เงื่อนไข){  
    คำสั่งที่จะทำซ้ำเมื่อเงื่อนไขเป็นจริง ;  
}
```



1. เช็คเงื่อนไขถ้าเป็นจริงให้
ทำคำสั่งซ้ำใน Statement
2. ถ้าเป็นเท็จให้ออกจาก Loop

```
while(condition){  
    //statement  
}
```

Output

แสดงความ "Hello C"
จำนวน 3 ครั้ง


```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

- Hello C

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

- Hello C

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

- Hello C , count=2

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

- Hello C , **count=2**


```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

- Hello C , count=2

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

- Hello C , count=2

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

- Hello C , count=2
- Hello C

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

- Hello C , count=2
- Hello C

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

- Hello C , count=2
- Hello C , count=3

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

- Hello C , count=2
- Hello C , **count=3**

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

- Hello C , count=2
- Hello C , count=3

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

- Hello C , count=2
- Hello C , count=3


```
int count = 1;
while(count<=3){
    printf("Hello C\n");
    count++;
}
```

Output

- Hello C , count=2
- Hello C , count=3
- Hello C

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

- Hello C , count=2
- Hello C , count=3
- Hello C

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

- Hello C , count=2
- Hello C , count=3
- Hello C , count=4

```
int count = 1;
while(count<=3){
    printf("Hello C\n");
    count++;
}
```

Output

- Hello C , count=2
- Hello C , count=3
- Hello C , **count=4**

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

- Hello C , count=2
- Hello C , count=3
- Hello C , count=4

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

Output

- Hello C , count=2
- Hello C , count=3
- Hello C , count=4

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

//จบโปรแกรม

Output

- Hello C , count=2
- Hello C , count=3
- Hello C , count=4

```
int count = 1;  
while(count<=3){  
    printf("Hello C\n");  
    count++;  
}
```

//จบโปรแกรม

Output

- Hello C
- Hello C
- Hello C

แบบทำซ้ำ (Loop)

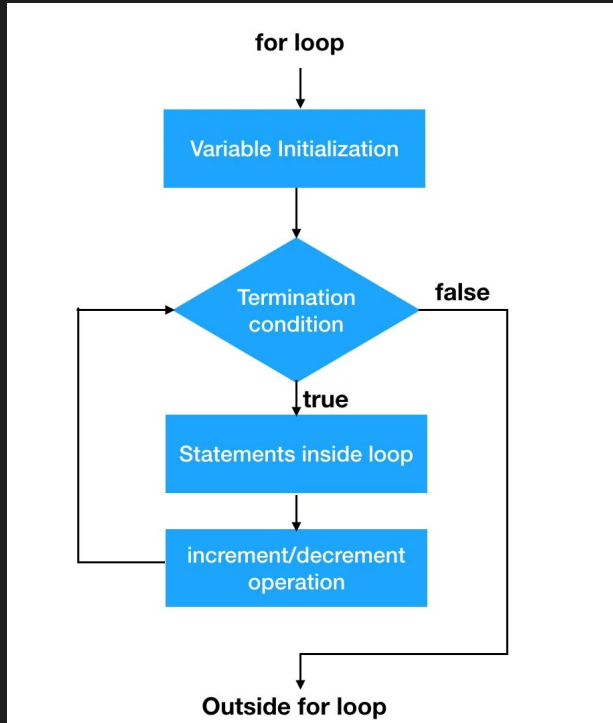
กลุ่มคำสั่งที่ใช้ในการวนรอบ (Loop) โปรแกรมจะทำงานไปเรื่อยๆจนกว่าเงื่อนไขที่กำหนดไว้จะเป็นเท็จ จึงจะหยุดทำงาน

- While
- For
- Do..While

For Loop

เป็นรูปแบบการซ้ำที่ใช้ในการตรวจสอบเงื่อนไขการทำงาน มีการกำหนดค่าเริ่มต้นและเปลี่ยนค่าไปพร้อมๆกัน เมื่อเงื่อนไขในคำสั่ง for เป็นจริงก็จะทำงานตามคำสั่งที่แสดงไว้ภายในคำสั่ง for ไปเรื่อยๆ

```
for(ค่าเริ่มต้นของตัวแปร; เงื่อนไข; เปลี่ยนแปลงค่าตัวแปร) {  
    คำสั่งเมื่อเงื่อนไขเป็นจริง;  
}
```



<https://cdn.journaldev.com/wp-content/uploads/2017/10/java-for-loop.png.webp>

1. กำหนดค่าเริ่มต้น
2. เช็คเงื่อนไขถ้าเป็นจริงให้ทำ
คำสั่งซ้ำใน Statement
3. ถ้าเป็นเท็จให้ออกจาก Loop

```
for(ค่าเริ่มต้นของตัวแปร; เงื่อนไข; เปลี่ยนแปลงค่าตัวแปร){  
    printf("Hello C\n");  
}
```

Output

แสดงข้อความ "Hello C" จำนวน 3 ครั้ง

```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

Output

แสดงข้อความ "Hello C" จำนวน 3 ครั้ง

```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```



```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

- Hello C

```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

- Hello C

```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

- Hello C , count = 2

```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

- Hello C , count = 2

```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

- Hello C , count = 2

```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

- Hello C , count = 2
- Hello C

```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

- Hello C , count = 2
- Hello C

```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

- Hello C , count = 2
- Hello C , count = 3


```
for(int count = 1; count <= 3; count++){  
    printf("Hello C\n");  
}
```

- Hello C , count = 2
- Hello C , count = 3

```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

- Hello C , count = 2
- Hello C , count = 3

```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

- Hello C , count = 2
- Hello C , count = 3
- Hello C

```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

- Hello C , count = 2
- Hello C , count = 3
- Hello C

```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

- Hello C , count = 2
- Hello C , count = 3
- Hello C , count = 4

```
for(int count = 1; count <= 3; count++){  
    printf("Hello C\n");  
}
```

- Hello C , count = 2
- Hello C , count = 3
- Hello C , count = 4

```
for(int count = 1; count <= 3; count++){  
    printf("Hello C\n");  
}
```

- Hello C , count = 2
- Hello C , count = 3
- Hello C , count = 4

```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

- Hello C , count = 2
- Hello C , count = 3
- Hello C , count = 4

//จบโปรแกรม


```
for(int count = 1; count <=3;count++){  
    printf("Hello C\n");  
}
```

- Hello C
- Hello C
- Hello C

แบบทำซ้ำ (Loop)

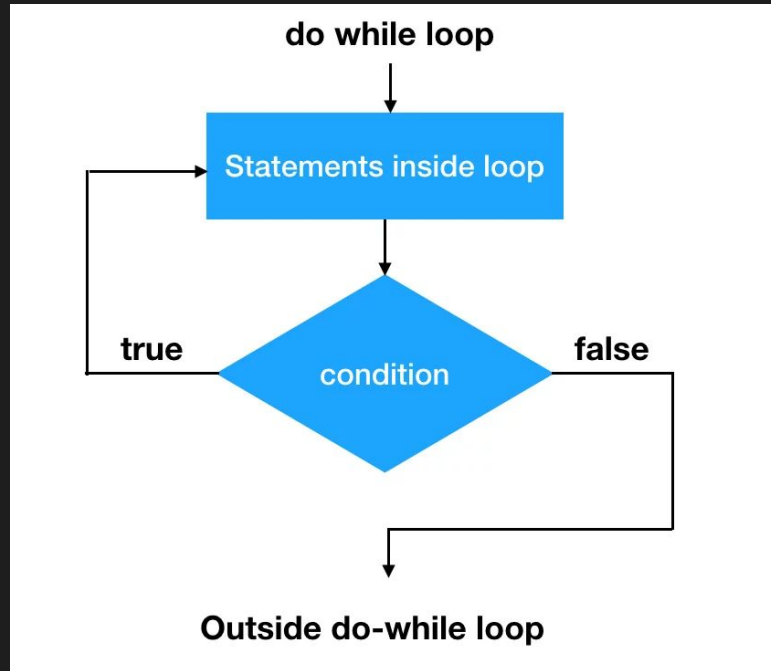
กลุ่มคำสั่งที่ใช้ในการวนรอบ (Loop) โปรแกรมจะทำงานไปเรื่อยๆจนกว่าเงื่อนไขที่กำหนดไว้จะเป็นเท็จ จึงจะหยุดทำงาน

- While
- For
- Do..While

Do...While Loop

โปรแกรมจะทำงานตามคำสั่งอย่างน้อย 1 รอบ เมื่อทำงานเสร็จจะ
มาตรวจสอบเงื่อนไขที่คำสั่ง while ถ้าเงื่อนไขเป็นจริงจะวนกลับขึ้นไปทำ
งานที่คำสั่งใหม่อีกรอบ แต่ถ้าเป็นเท็จจะหลุดออกจากลูป

```
do {  
    คำสั่งต่างๆ เมื่อเงื่อนไขเป็นจริง;  
} while(เงื่อนไข);
```



1. ทำงานคำสั่งใน Statement
2. เช็คเงื่อนไขถ้าเป็นจริงให้กลับไปทำซ้ำใน Statement
3. ถ้าเป็นเท็จให้ออกจาก Loop

คำสั่งที่เกี่ยวข้องกับ Loop

- **break** ถ้าโปรแกรมพบคำสั่งนี้จะหลุดจากการทำงานในลูปทันที เพื่อไปทำคำสั่งอื่นที่อยู่นอกลูป
- **continue** คำสั่งนี้จะทำให้หยุดการทำงานแล้วย้อนกลับไปเริ่มต้นการทำงานที่ต้นลูปใหม่

ข้อแตกต่างและการใช้งาน Loop

- For ใช้ในกรณีรู้จำนวนรอบที่ชัดเจน
- While ใช้ในกรณีที่ไม่รู้จำนวนรอบ
- Do..while ใช้ในกรณีที่ต้องการให้ลองทำก่อน 1 รอบ
แล้วทำซ้ำไปเรื่อยๆ ตราบเท่าที่เงื่อนไขเป็นจริง

หาผลรวมตัวเลข

- จำนวนตัวเลข : 5 จำนวน
- รับค่าตัวเลขผ่านแป้นพิมพ์
- แสดงผลรวมตัวเลข 5 จำนวน (total)

หาผลรวมตัวเลข (ไม่จำกัดจำนวน)

- จำนวนตัวเลข : ไม่จำกัด
- รับค่าตัวเลขผ่านแป้นพิมพ์
- ถ้าป้อนเลขน้อยกว่าหรือเท่ากับ 0 ให้จบการทำงาน
- แสดงผลรวมตัวเลขตามจำนวนที่ป้อน (total)

Nested Loop

ในการเขียนโปรแกรมสามารถนำคำสั่ง
รูปแบบต่างๆ ให้มาทำงานซ้อนกันได้เรียกว่า
“ ลูปซ้อนลูป (Nested Loop) ”

Nested Loop

โครงสร้างคำสั่ง (For Loop)

```
for(ค่าเริ่มต้นของตัวแปร; เงื่อนไข; เปลี่ยนแปลงค่าตัวแปร){  
    for(ค่าเริ่มต้นของตัวแปร; เงื่อนไข; เปลี่ยนแปลงค่าตัวแปร){  
          
    }  
}
```

Nested Loop

โครงสร้างคำสั่ง (For Loop)

Loop นอก

```
for(ค่าเริ่มต้นของตัวแปร; เงื่อนไข; เปลี่ยนแปลงค่าตัวแปร){  
    for(ค่าเริ่มต้นของตัวแปร; เงื่อนไข; เปลี่ยนแปลงค่าตัวแปร){  
          
    }  
}
```

Nested Loop

โครงสร้างคำสั่ง (For Loop)

```
Loop นอก → for(ค่าเริ่มต้นของตัวแปร; เงื่อนไข; เปลี่ยนแปลงค่าตัวแปร){  
    Loop ใน → for(ค่าเริ่มต้นของตัวแปร; เงื่อนไข; เปลี่ยนแปลงค่าตัวแปร){  
        }  
    }  
}
```

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอกทำงาน 2 รอบ

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอกทำงาน 2 รอบ

Loop ในทำงาน 3 รอบ

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```


ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

2

Loop ใน

1

2

3

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

2

Loop ใน

1

2

3

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

2

Loop ใน

1

2

3

1

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

2

Loop ใน

1

2

3

1

2

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

2

Loop ใน

1

2

3

1

2

3

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

2

Loop ใน

1

2

3

1

2

3

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

2

Loop ใน

1

2

3

1

2

3

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

Loop นอก

1

2

Loop ใน

1

2

3

1

2

3

ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    printf("%d\n",i);  
    for(int j= 1; j<=3;j++){  
        printf("%d\n",j);  
    }  
}
```

จบการทำงาน!!

Loop นอก

1

2

Loop ใน

1

2

3

1

2

3

ฟังก์ชัน getchar และ putchar

การรับและแสดงผลข้อมูลแบบตัวอักษรนอกจากจะใช้ฟังก์ชัน scanf() และ printf() แล้ว ยังมีฟังก์ชันเฉพาะที่ใช้สำหรับรับข้อมูลแบบตัวอักษร คือ

- **getchar()** คือ ฟังก์ชันสำหรับรับข้อมูล 1 ตัวอักษรจากคีย์บอร์ด
- **putchar()** คือ ฟังก์ชันสำหรับแสดงผลข้อมูล 1 ตัวอักษรออกทางจอภาพ

ฟังก์ชัน gets และ puts

การรับและแสดงผลชุดข้อความ (String) นอกจากจะใช้ฟังก์ชัน scanf() และ printf() แล้ว ยังมีฟังก์ชันเฉพาะที่ใช้จัดการข้อความ คือ

- **gets()** คือ ฟังก์ชันสำหรับรับข้อมูลชุดข้อความจากคีย์บอร์ด
- **puts()** คือ ฟังก์ชันสำหรับแสดงผลชุดข้อความออกทางจอภาพ

รู้จักกับอาร์เรย์

ข้อจำกัดของชนิดข้อมูลพื้นฐาน

การประกาศตัวแปรแต่ละครั้ง ตัวแปร 1 ตัวสามารถเก็บข้อมูลได้แค่ 1 ค่าเท่านั้น เช่น

```
int score1 = 100
```

```
int score2 = 80
```

```
int score3 = 65
```

ตัวอย่างการสร้างตัวแปรแบบปกติ

```
int score1 = 100
```

```
int score2 = 80
```

```
int score3 = 65
```

score1



100

score2



80

score3



65

ข้อจำกัดของชนิดข้อมูลพื้นฐาน

“ ถ้าอยากเก็บเลข 10 ค่าต้องทำอะไร ต้องประกาศ
ตัวแปรจำนวน 10 ตัวแปร หรือไม่ ? ”



ตัวอย่างการสร้างตัวแปรแบบปกติ

```
int score1 = 100
```

```
int score2 = 80
```

```
int score3 = 65
```

```
int score4 = 80
```

```
int score5 = 90
```

```
int scoreN = xx
```

score1



100

score2



80

score3



65

อาร์เรย์ คืออะไร

1. ชุดของตัวแปรที่อยู่ในรูปลำดับใช้เก็บค่าข้อมูลให้อยู่ในกลุ่มเดียวกัน โดยข้อมูลภายในอาร์เรย์จะถูกเก็บในตำแหน่งที่ต่อเนื่องกัน
2. เป็นตัวแปรที่ใช้ในการเก็บข้อมูลที่มีลำดับที่ต่อเนื่อง ซึ่งข้อมูลมีค่าได้หลายค่าโดยใช้ชื่ออ้างอิงได้เพียงชื่อเดียว และใช้หมายเลขกำกับ (**index**) ให้กับตัวแปรเพื่อจำแนกความแตกต่างของค่าตัวแปรแต่ละตัว

คุณสมบัติของอาร์เรย์

1. ใช้เก็บกลุ่มของข้อมูล
2. ข้อมูลที่อยู่ในอาร์เรย์จะเรียกว่าสมาชิก หรือ อิลิเมนต์ (element)
3. แต่ละอิลิเมนต์ (element) จะเก็บค่าข้อมูล (value) และ อินเด็กซ์ (Index)
4. Index หมายถึงคีย์ของอาร์เรย์ใช้อ้างอิงตำแหน่งของ element เริ่มต้นที่ 0
5. สมาชิกในอาร์เรย์ต้องมีชนิดข้อมูลเหมือนกัน
6. สมาชิกในอาร์เรย์จะถูกค้นด้วยเครื่องหมายคอมม่า



ตัวอย่างการสร้างตัวแปรแบบปกติ

```
int score1 = 100
```

```
int score2 = 80
```

```
int score3 = 65
```

score1



100

score2



80

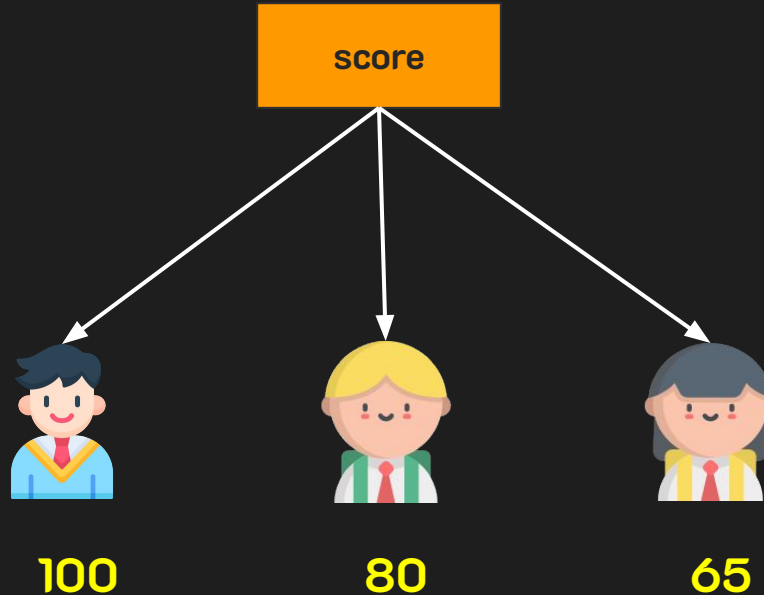
score3



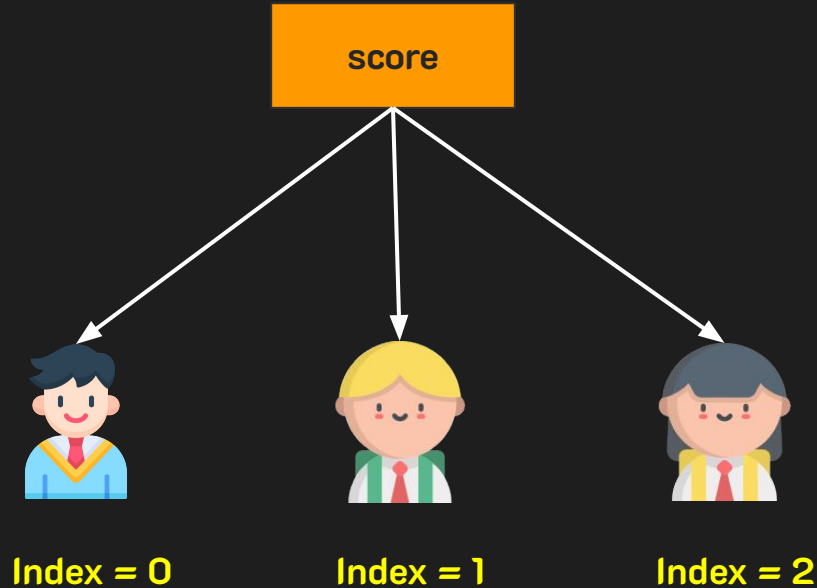
65



ตัวอย่างการสร้างตัวแปรอาร์เรย์



เข้าถึงสมาชิกในตัวแปรอาร์เรย์



สรุปอาร์เรย์

1. ใช้เก็บกลุ่มของข้อมูล ที่มีชนิดข้อมูลเดียวกัน
2. ใช้ตัวแปรชื่อเดียวกัน
3. ใช้หมายเลขกำกับเพื่ออ้างอิงตำแหน่งของข้อมูลในอาร์เรย์
4. มีขนาดที่แน่นอนไม่สามารถปรับเปลี่ยนขนาดได้



การสร้างอาร์เรย์ (Array)

การสร้างอาร์เรย์

แบบกำหนดขนาด

ชนิดข้อมูล ชื่อตัวแปร[ขนาด]; **//ขนาดต้องเป็นตัวเลขจำนวนเต็ม**
เช่น `int score [3];`

แบบกำหนดขนาดและค่าเริ่มต้น

ชนิดข้อมูล ชื่อตัวแปร [ขนาด] = {สมาชิก,...};
เช่น `int score [3] = {100,90,70};`

การสร้างอาร์เรย์

```
int score [4] = {100,90,70,80};
```

100	90	70	80
-----	----	----	----

```
int score [10] = {100,90,70,80};
```

100	90	70	80	0	0	0	0	0	0
-----	----	----	----	---	---	---	---	---	---

การสร้างอาร์เรย์

```
int score [4] = {100,90,70,80};
```

100	90	70	80
-----	----	----	----

```
int score [10] = {100,90,70,80};
```

100	90	70	80	0	0	0	0	0	0
-----	----	----	----	---	---	---	---	---	---

การสร้างอาร์เรย์

แบบไม่กำหนดขนาด

ชนิดข้อมูล ชื่อตัวแปร[] = {สมาชิก,...};
เช่น `int score [] = {100,90,70,80};`

100

90

70

80

จัดการสมาชิกใน อาร์เรย์

การเข้าถึงสมาชิก

```
int score [3] = {100,90,80};
```

100	90	80
-----	----	----

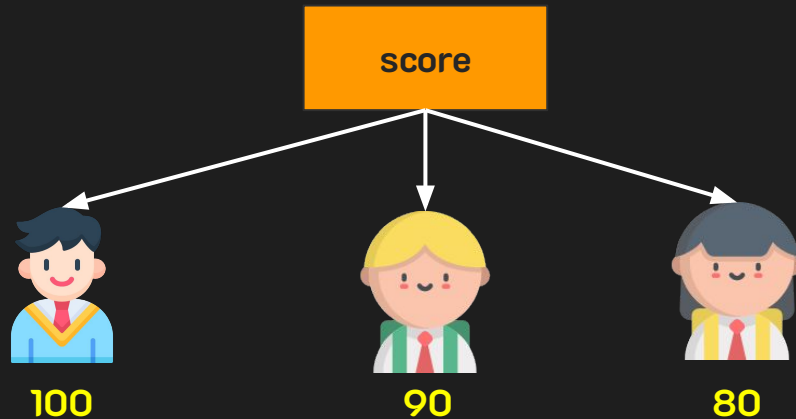
การเข้าถึงสมาชิก

```
int score [3] = {100,90,80};
```

100 (0)	90 (1)	80 (2)
---------	--------	--------

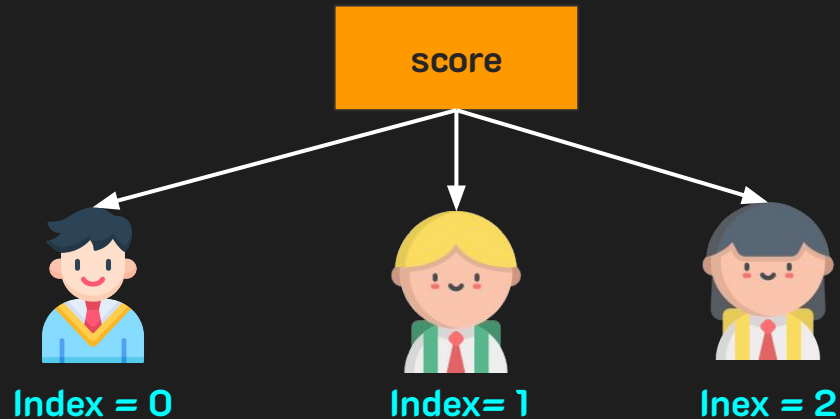
การเข้าถึงสมาชิก

```
int score [3] = {100, 90, 80};
```



การเข้าถึงสมาชิก

```
int score [3] = {100,90,80};
```



การเปลี่ยนแปลงข้อมูลสมาชิก Array

```
int number[] = {10, 20, 30, 40};
```

```
number[2] = 100;
```

```
char vowels[] = {'A','F','I','O','U'}
```

```
vowels[1]='E';
```

การเข้าถึงสมาชิกด้วย For Loop

```
int number[] = {10, 20, 30};  
for (int i = 0; i < 3; i++) {  
    // กระบวนการทำงาน  
}
```

อาร์เรย์ 2 มิติ

อาร์เรย์ 2 มิติ

- อาร์เรย์ที่มีข้อมูลสมาชิกภายในเป็นอาร์เรย์ (Array ซ้อน Array)
- มีโครงสร้างเป็นรูปแบบแถว (แนวนอน) และคอลัมน์ (แนวตั้ง)



รูปแบบของอาร์เรย์ 1 มิติ

```
int number [] = {10, 20, 30, 40};
```

10	20	30	40
----	----	----	----



Array 1 มิติ

รูปแบบของอาร์เรย์ 2 มิติ

แถวที่ 0

--	--	--	--

แถวที่ 1

--	--	--	--

แถวที่ 2

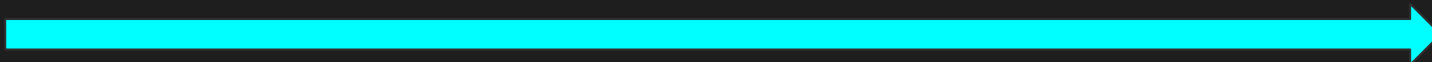
--	--	--	--

รูปแบบของอาร์เรย์ 2 มิติ

แถวที่ 0

แถวที่ 1

แถวที่ 2



รูปแบบของอาร์เรย์ 2 มิติ

	คอลัมน์ที่ 0	คอลัมน์ที่ 1	คอลัมน์ที่ 2	คอลัมน์ที่ 3
แถวที่ 0				
แถวที่ 1				
แถวที่ 2				



การเข้าถึงสมาชิกในอาร์เรย์ 2 มิติ

	คอลัมน์ที่ 0	คอลัมน์ที่ 1	คอลัมน์ที่ 2	คอลัมน์ที่ 3
แถวที่ 0	[0,0]	[0,1]	[0,2]	[0,3]
แถวที่ 1	[1,0]	[1,1]	[1,2]	[1,3]
แถวที่ 2	[2,0]	[2,1]	[2,2]	[2,3]



สร้างอาร์เรย์ 2 มิติ

การสร้างอาร์เรย์ 2 มิติ

แบบกำหนดขนาด

ชนิดข้อมูล ชื่อตัวแปร[ขนาดแถว][ขนาดคอลัมน์];
เช่น `int score [2][4];`

การสร้างอาร์เรย์ 2 มิติ

แบบกำหนดขนาดและค่าเริ่มต้น

ชนิดข้อมูล ชื่อตัวแปร[ขนาดแถว][ขนาดคอลัมน์] = {สมาชิก,...};

ตัวอย่าง เช่น

```
int numbers [2][4]={  
    {50,70,80,90},  
    {100,99,60,55}  
};
```

การสร้างอาร์เรย์ 2 มิติ

แบบกำหนดขนาดและค่าเริ่มต้น

ชนิดข้อมูล ชื่อตัวแปร[ขนาดแถว][ขนาดคอลัมน์] = {สมาชิก,...};

ตัวอย่าง เช่น

```
int numbers [2][4]={
```

แถว 1 → {50,70,80,90},
 {100,99,60,55}

```
};
```

การสร้างอาร์เรย์ 2 มิติ

แบบกำหนดขนาดและค่าเริ่มต้น

ชนิดข้อมูล ชื่อตัวแปร[ขนาดแถว][ขนาดคอลัมน์] = {สมาชิก,...};

ตัวอย่าง เช่น

```
int numbers [2][4]={
```

แถว 1 → {50,70,80,90},

แถว 2 → {100,99,60,55}

```
};
```

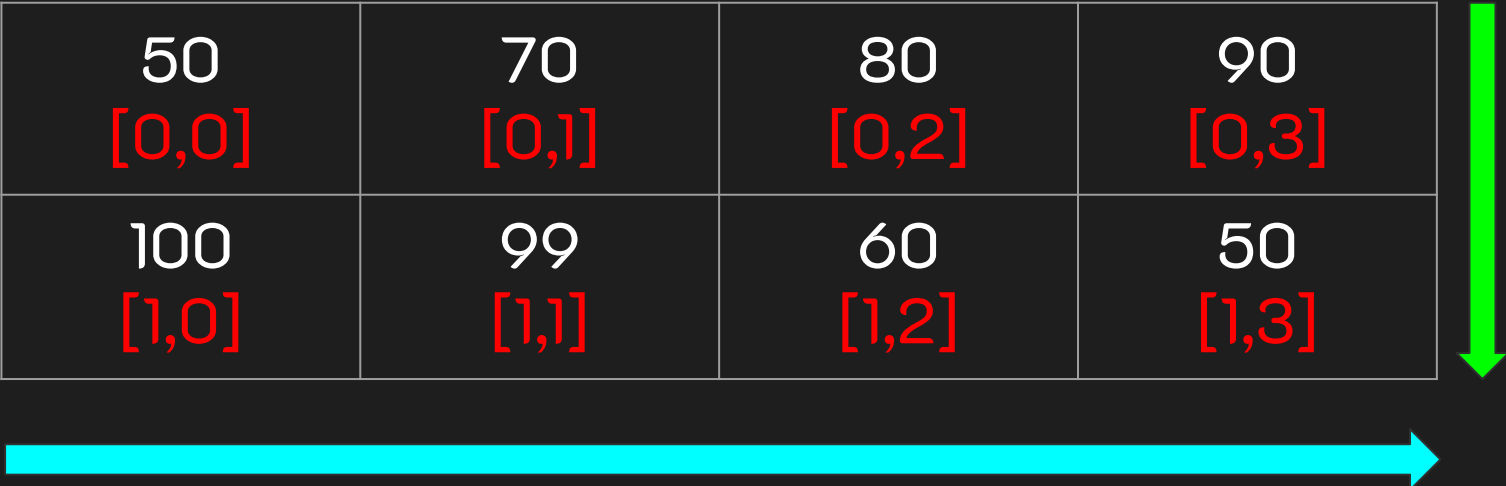
โครงสร้างของอาร์เรย์ 2 มิติ

	คอลัมน์ที่ 0	คอลัมน์ที่ 1	คอลัมน์ที่ 2	คอลัมน์ที่ 3
แถวที่ 0	50	70	80	90
แถวที่ 1	100	99	60	50



การเข้าถึงสมาชิกอาร์เรย์ 2 มิติ

	คอลัมน์ที่ 0	คอลัมน์ที่ 1	คอลัมน์ที่ 2	คอลัมน์ที่ 3
แถวที่ 0	50 [0,0]	70 [0,1]	80 [0,2]	90 [0,3]
แถวที่ 1	100 [1,0]	99 [1,1]	60 [1,2]	50 [1,3]



การเปลี่ยนแปลงค่าในอาร์เรย์ 2 มิติ

โครงสร้างคำสั่ง

ชื่อตัวแปร[แถว][คอลัมน์] = กำหนดค่า

score [0][1] = 99

score [1][3] = 80

ฟังก์ชัน (Function)

ฟังก์ชัน (Function) คืออะไร

ชุดคำสั่งที่นำมาเขียนรวมกันเป็นกลุ่มเพื่อให้เรียกใช้งานตาม
วัตถุประสงค์ที่ต้องการและลดความซ้ำซ้อนของคำสั่งที่ใช้งานบ่อย

ฟังก์ชันสามารถนำไปใช้งานได้ทุกที่และแก้ไขได้ในภายหลัง
ทำให้โค้ดในโปรแกรมมีระเบียบและใช้งานได้สะดวกมากยิ่งขึ้น

ประเภทของฟังก์ชัน

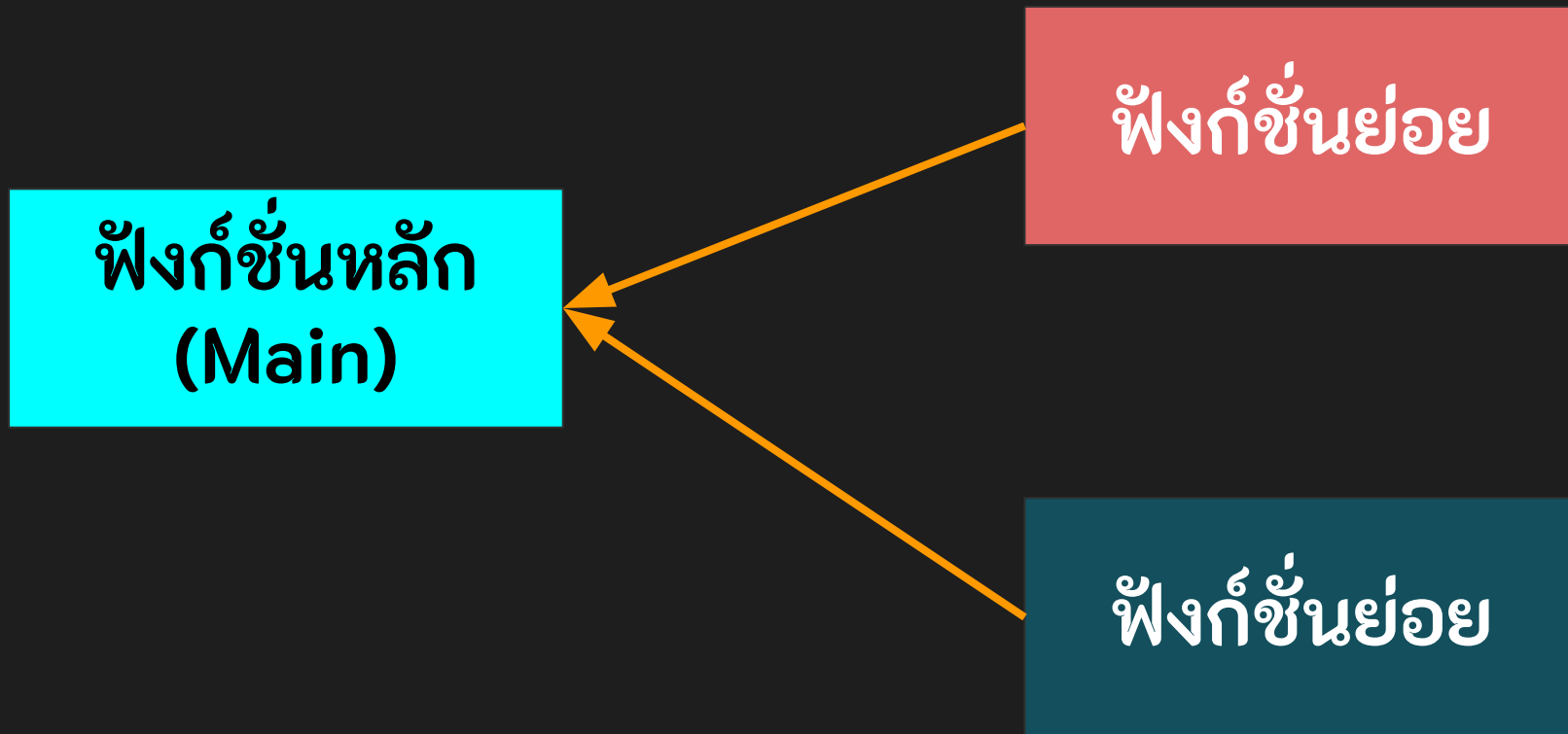
- **ฟังก์ชันมาตรฐาน (Standard Library Functions)** คือ ฟังก์ชันที่มีอยู่ในภาษา C ผู้ใช้สามารถเรียกใช้งานได้เลย เช่น printf() , scanf() ที่ทำงานอยู่ในไลบรารี หรือ Header File เช่น stdio.h เป็นต้น
- **ฟังก์ชันที่ผู้ใช้สร้างขึ้นเอง (User-Define Function)** คือ ฟังก์ชันที่ถูกสร้างขึ้นมาเพื่อวัตถุประสงค์ให้ทำงานตามที่ใช้ต้องการ

ฟังก์ชันหลัก (main)

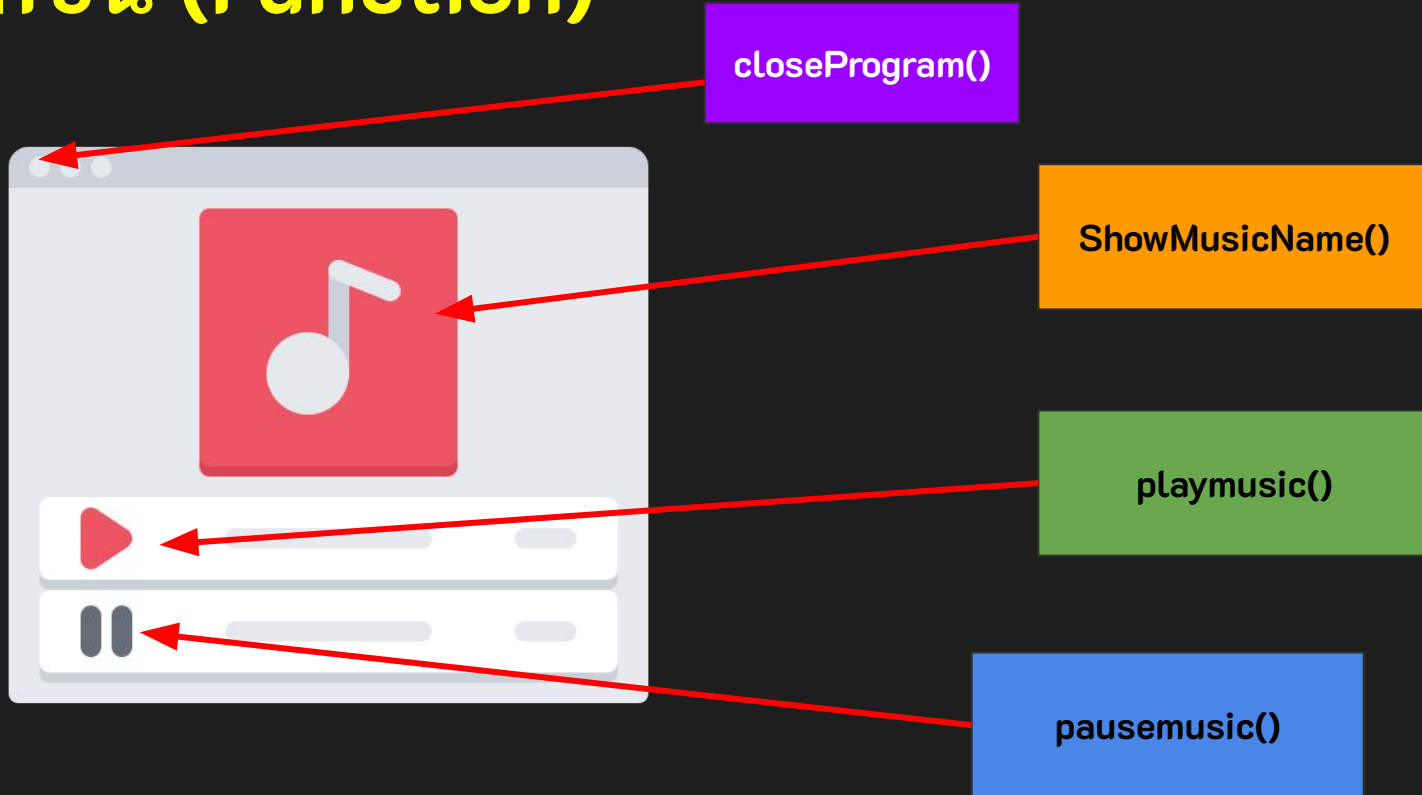
```
int main()  
{  
  
    //statement  
  
}
```

ฟังก์ชัน **main()** คือ ฟังก์ชันพิเศษกลุ่มคำสั่งที่อยู่ในฟังก์ชันนี้จะถูกสั่งให้ทำงานโดยอัตโนมัติเป็นลำดับแรกเสมอ

ฟังก์ชัน (Function)



ฟังก์ชัน (Function)



รูปแบบฟังก์ชัน

- ฟังก์ชันแบบปกติ (Void Function)
- ฟังก์ชันแบบมีพารามิเตอร์ (Parameter Function)
- ฟังก์ชันแบบมีค่าส่งกลับ (Return Function)
- ฟังก์ชันแบบรับและส่งค่า

กฎการตั้งชื่อฟังก์ชัน

- ชื่อฟังก์ชันต้องไม่ซ้ำกัน
- ชื่อฟังก์ชันสามารถตั้งเป็นตัวอักษรหรือตัวเลขได้
- ชื่อของฟังก์ชันต้องไม่ขึ้นต้นด้วยตัวเลข

วิธีสร้างฟังก์ชัน

1. นิยามชื่อฟังก์ชันก่อนกำหนดโครงสร้าง

```
func_name(); //นิยามชื่อฟังก์ชัน (function prototype)
```

```
int main(){  
}
```

```
func_name(){} //กำหนดโครงสร้างการทำงานหลังฟังก์ชัน main
```

วิธีสร้างฟังก์ชัน

2. นิยามชื่อฟังก์ชันพร้อมกำหนดโครงสร้าง

// นิยามชื่อพร้อมกำหนดโครงสร้างคำสั่ง (เขียนอยู่ด้านบน main เท่านั้น)

```
func_name(){ }
```

```
int main(){  
}
```


ฟังก็ชื่นแบบปกติ

ฟังก์ชันที่ไม่มีการรับและส่งค่า (void)

โครงสร้างคำสั่ง

```
void ชื่อฟังก์ชัน(){  
    // คำสั่งต่างๆ  
}
```

การเรียกใช้งานฟังก์ชัน

```
ชื่อฟังก์ชัน ();
```

ฟังก์ชันแบบมีพารามิเตอร์

ฟังก์ชันแบบมีพารามิเตอร์ (Parameter)

โครงสร้างคำสั่ง

```
void ชื่อฟังก์ชัน(parameter1,parameter2,...){  
    // กลุ่มคำสั่งต่างๆ  
}
```

การเรียกใช้งานฟังก์ชัน

```
ชื่อฟังก์ชัน (argument1,argument2,...);
```

ฟังก์ชันแบบมีพารามิเตอร์ (Parameter)

โครงสร้างคำสั่ง

```
void ชื่อฟังก์ชัน(parameter1,parameter2,...){  
    // กลุ่มคำสั่งต่างๆ  
}
```

การเรียกใช้งานฟังก์ชัน

ชื่อฟังก์ชัน (argument1,argument2,...);

- อาร์กิวเมนต์ คือ ตัวแปรหรือค่าที่ต้องการส่งมาให้กับฟังก์ชัน (ตัวแปรส่ง)
- พารามิเตอร์ คือ ตัวแปรที่ฟังก์ชันสร้างไว้สำหรับรับค่าที่จะส่งเข้ามาให้กับฟังก์ชัน (ตัวแปรรับ)

ฟังก็ชื่นแบบมีค่าส่งกลับ

ฟังก์ชันแบบมีค่าส่งกลับ (Return)

โครงสร้างคำสั่ง

```
type ชื่อฟังก์ชัน(){  
    return ค่าที่จะส่งออกไป (อ้างอิงตามชนิดข้อมูล)  
}
```

การเรียกใช้งานฟังก์ชัน

ตัวแปรที่รับค่าจากฟังก์ชัน = ชื่อฟังก์ชัน ();

ฟังก์ชันแบบรับและส่งค่า

ฟังก์ชันแบบรับและส่งค่า

โครงสร้างคำสั่ง

```
type ชื่อฟังก์ชัน(parameter1,parameter2,...){  
    return ค่าที่จะส่งออกไป (อ้างอิงตามชนิดข้อมูล)  
}
```

การเรียกใช้งานฟังก์ชัน

ตัวแปรที่รับค่าจากฟังก์ชัน = ชื่อฟังก์ชัน(argument1,argument2..);

ขอบเขตตัวแปร

ขอบเขตตัวแปร

- **Local variable** ตัวแปรที่ประกาศอยู่ภายในฟังก์ชันมีขอบเขตการทำงานตั้งแต่จุดเริ่มต้นไปจนถึงจุดสิ้นสุดของฟังก์ชันจะถือได้ว่าฟังก์ชันนั้นเป็นเจ้าของตัวแปรนั้น ฟังก์ชันอื่นจะไม่สามารถเรียกใช้งานตัวแปรนี้ได้

ขอบเขตตัวแปร

- **Global variable** ตัวแปรที่ประกาศอยู่นอกฟังก์ชันมีขอบเขตการทำงานตั้งแต่จุดเริ่มต้นไปจนถึงจุดสิ้นสุดของไฟล์ที่ประกาศใช้ นั่นหมายถึงตัวแปรดังกล่าวนี้เป็นสาธารณะ ไม่มีฟังก์ชันใดเป็นเจ้าของ ทุกฟังก์ชันสามารถเรียกใช้งานตัวแปรนี้ได้

รู้จักกับ Pointer

การสร้างตัวแปร

ตัวอย่าง

```
int number = 100;
```

ตัวแปร number เก็บค่าตัวเลขจำนวนเต็มคือเลข 100



การสร้างตัวแปร

ตัวอย่าง

```
int number = 100;
```

ตัวแปร number เก็บค่าตัวเลขจำนวนเต็มคือเลข 100

คำถาม : ตัวแปร number และค่า 100 ถูกเก็บไว้ที่ใด ???

การสร้างตัวแปร

ตัวอย่าง

```
int number = 100;
```

ตัวแปร number เก็บค่าตัวเลขจำนวนเต็มคือเลข 100

คำถาม : ตัวแปร number และค่า 100 ถูกเก็บไว้ที่ใด ???

คำตอบ : เก็บไว้ที่หน่วยความจำ

การสร้างตัวแปร

ตัวอย่าง

```
int number = 100;
```

ตัวแปร number เก็บค่าตัวเลขจำนวนเต็มคือเลข 100

คำถาม : แล้วจะเข้าถึงค่าในหน่วยความจำได้อย่างไร ??

การสร้างตัวแปร

ตัวอย่าง

```
int number = 100;
```

ตัวแปร number เก็บค่าตัวเลขจำนวนเต็มคือเลข 100

คำถาม : แล้วจะเข้าถึงค่าในหน่วยความจำได้อย่างไร ??

คำตอบ : ใช้พอยน์เตอร์ (Pointer)

Pointer คืออะไร

พอยน์เตอร์ (Pointer) คือตัวแปรที่ใช้เก็บ**ตำแหน่งที่อยู่**ของตัวแปรที่สนใจหรือค่าแอดเดรส (Address) หน่วยความจำ ซึ่งมีประโยชน์อย่างมากสำหรับการเขียนโปรแกรมจัดการหน่วยความจำ

โครงสร้างพื้นที่หน่วยความจำ

ตัวแปร	ค่าในตัวแปร	แอดเดรส (Address)
a	10	0x6ffe3c
b	20	0x6ffe38
c	'A'	0x6ffe30

โครงสร้างพื้นที่หน่วยความจำ

ตัวแปร	ค่าในตัวแปร	แอดเดรส (Address)
a	10	0x6ffe3c
b	20	0x6ffe38
c	'A'	0x6ffe30

Address คือ ตำแหน่งที่เก็บข้อมูลในหน่วยความจำ เป็นรูปแบบเลขฐาน 16

การสร้างตัวแปร Pointer

การสร้างตัวแปร Pointer

โครงสร้างคำสั่ง

ชนิดข้อมูล *ตัวแปรพอยน์เตอร์;

การสร้างตัวแปร Pointer

ตัวอย่าง

```
int number = 10;
```

```
int *p1 = &number;
```

```
char letter = 'A';
```

```
char *p2 = &letter;
```


การสร้างตัวแปร Pointer

ตัวอย่าง

```
int number = 10;
```

```
int *p1 = &number;
```

```
char letter = 'A';
```

```
char *p2 = &letter;
```

การสร้างตัวแปร Pointer

ตัวอย่าง

```
int number = 10;
```

```
int *p1 = &number;
```

```
char letter = 'A';
```

```
char *p2 = &letter;
```

* คือ ตำแหน่งแอดเดรสในหน่วย
ความจำที่พอยน์เตอร์ชี้อยู่

& คือ ค่าแอดเดรสของตัวแปร

การสร้างตัวแปร Pointer

ตัวอย่าง

```
int number = 10;
```

```
int *p1 = &number;
```

```
char letter = 'A';
```

```
char *p2 = &letter;
```

ตัวแปร p1 คือ ตัวแปร pointer ที่ชี้ไปที่แอดเดรสของตัวแปรที่เป็นรูปแบบ int

ตัวแปร p2 คือ ตัวแปร pointer ที่ชี้ไปที่แอดเดรสของตัวแปรที่เป็นรูปแบบ char

สตรัคเจอร์ (Structure)

สตรัคเจอร์ (Structure)

คือ ข้อมูลแบบโครงสร้างที่นำเอาข้อมูลที่มีชนิดข้อมูลต่างกันมารวบรวมเข้าด้วยกัน แต่มีความสัมพันธ์ของข้อมูลแบบต่อกัน มาเก็บไว้ในโครงสร้างเดียวกัน

****เปรียบเทียบเสมือนกับสร้างชนิดข้อมูลขึ้นมาใช้งานเอง****




การสร้างสตรัคเจอร์

```
struct ชื่อสตรัคเจอร์ {  
    ชนิดข้อมูลตัวที่ 1 ตัวแปรที่ 1 ;  
    ชนิดข้อมูลตัวที่ 2 ตัวแปรที่ 2 ;  
    ....  
}
```



การสร้างสตรัคเจอร์

```
struct user{  
    char name[20];  
    char gender;  
    int age;  
};
```



```
struct user emp1;  
strcpy(emp1.name,"kong");  
emp1.gender='M';  
emp1.age = 30;
```