



เขียนโปรแกรมภาษา C++ สำหรับผู้เริ่มต้น [Phase 2]



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

รู้จักกับอาร์เรย์



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>



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

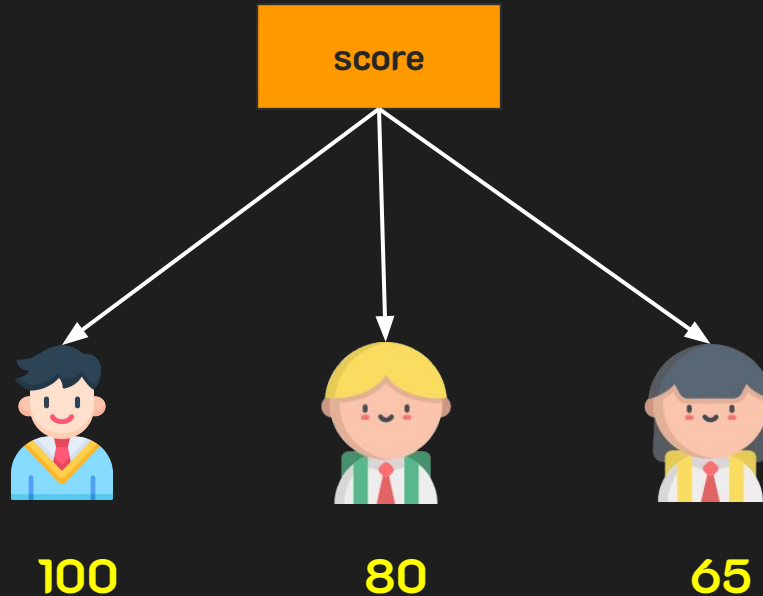


65



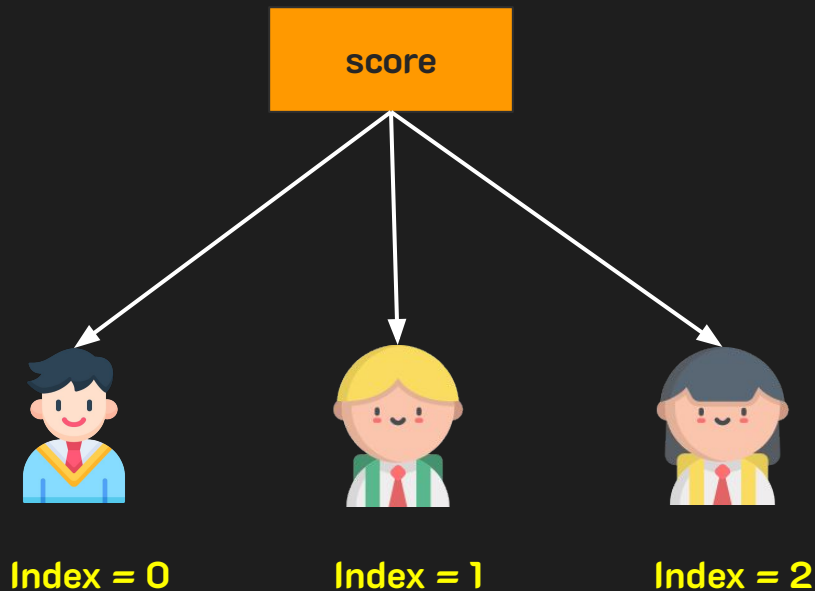


ตัวอย่างการสร้างตัวแปรอาร์เรย์





เข้าถึงสมาชิกในตัวแปรอาร์เรย์



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>



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

แบบไม่กำหนดขนาด

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

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

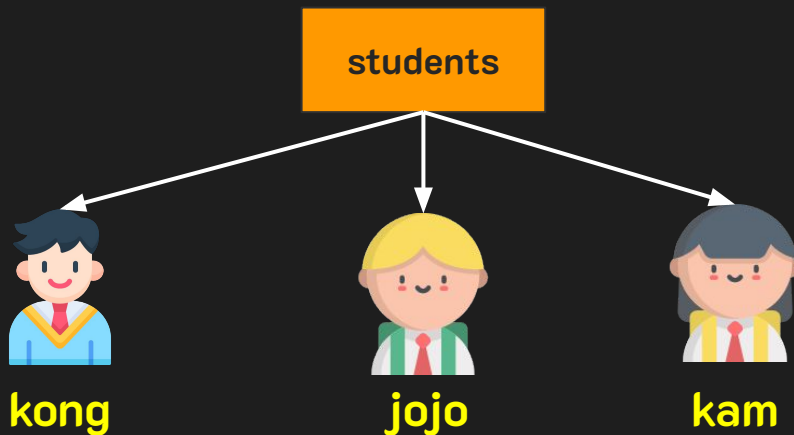


จัดการสมาชิกใน อาร์เรย์

การเข้าถึงสมาชิก



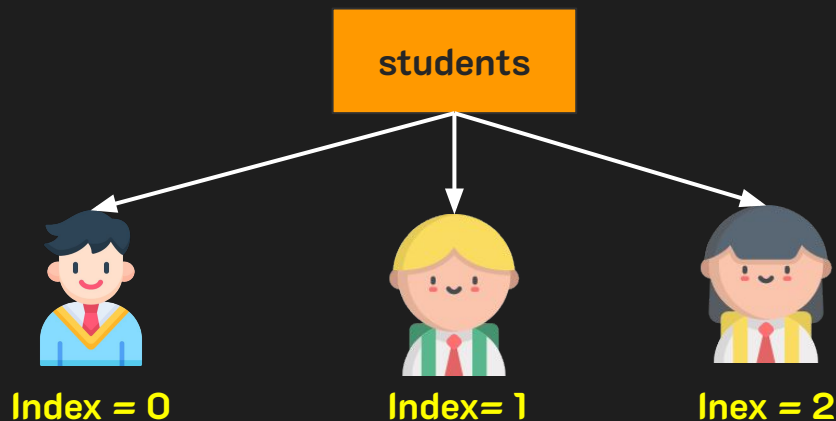
```
string students[3] = {"kong", "jojo", "kam"}
```



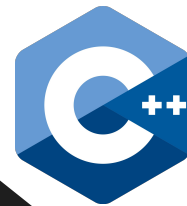
การเข้าถึงสมาชิก



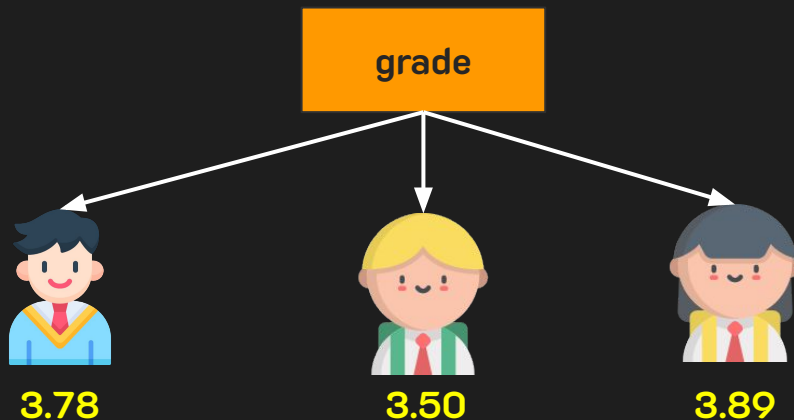
```
string students[3] = {"kong", "jojo", "kam"}
```



การเข้าถึงสมาชิก



```
double grade [3] = {3.78, 3.50, 3.89};
```





การเปลี่ยนแปลงข้อมูลสมาชิก Array

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

```
number[1] = 100;
```

```
string students [] = {"kong", "jojo"};
```

```
students [1] = "june";
```





การเข้าถึงสมาชิกด้วย For Loop

```
int number[] = {10, 20, 30, 40};  
for (int i = 0; i < 4; i++) {  
    // กระบวนการทำงาน  
}
```



Sizeof()



Sizeof

คือ ฟังก์ชันสำหรับดึงขนาดพื้นที่เก็บข้อมูลของ
ชนิดข้อมูลพื้นฐานที่สนใจ (Byte) มาใช้งาน



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>



ตารางแสดงขนาดเก็บข้อมูล

ชนิดข้อมูล	ขนาด (Byte)
boolean	1
int	4
long	4
float	4
double	8
char	1

จำนวนสมาชิกในอาร์เรย์



คำนวณจำนวนสมาชิก

$\text{sizeof}(\text{ชื่ออาร์เรย์}) / \text{sizeof}(\text{ชนิดข้อมูลที่เก็บในอาร์เรย์})$

```
int score[]={100,50,60}
```

```
int count = sizeof(score)/sizeof(score[0]) หรือ
```

```
int count = sizeof(score)/sizeof(int)
```



คำนวณจำนวนสมาชิก

- `int score[]={100,50,60}` // เก็บตัวเลขจำนวนเต็ม



คำนวณจำนวนสมาชิก

- `int score[]={100,50,60}` // เก็บตัวเลขจำนวนเต็ม
- `sizeof(int) = 4 byte`



คำนวณจำนวนสมาชิก

- `int score[]={100,50,60}` // เก็บตัวเลขจำนวนเต็ม
- `sizeof(int) = 4 byte`
- `sizeof(score) = 4 x 3 = 12 Byte`



คำนวณจำนวนสมาชิก

- `int score[]={100,50,60}` // เก็บตัวเลขจำนวนเต็ม
- `sizeof(int)` = 4 byte
- `sizeof(score)` = 4 x 3 = 12 Byte
- จำนวนสมาชิก = `sizeof(score) / sizeof(int)`



คำนวณจำนวนสมาชิก

- `int score[]={100,50,60}` // เก็บตัวเลขจำนวนเต็ม
- `sizeof(int)` = 4 byte
- `sizeof(score)` = 4 x 3 = 12 Byte
- จำนวนสมาชิก = 12 / 4



คำนวณจำนวนสมาชิก

- `int score[]={100,50,60}` // เก็บตัวเลขจำนวนเต็ม
- `sizeof(int)` = 4 byte
- `sizeof(score)` = 4 x 3 = 12 Byte
- จำนวนสมาชิก = 3



การเข้าถึงสมาชิกด้วย For Loop

```
int score[] = {100, 20, 30, 40};  
int count = sizeof(score)/sizeof(score[0])  
  
for (int i = 0; i < count ; 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



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

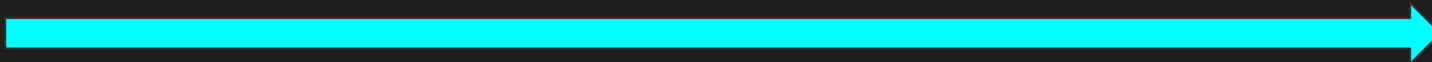


รูปแบบของอาร์เรย์ 2 มิติ

แถวที่ 0

แถวที่ 1

แถวที่ 2



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>



รูปแบบของอาร์เรย์ 2 มิติ

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



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>



รูปแบบของอาร์เรย์ 2 มิติ

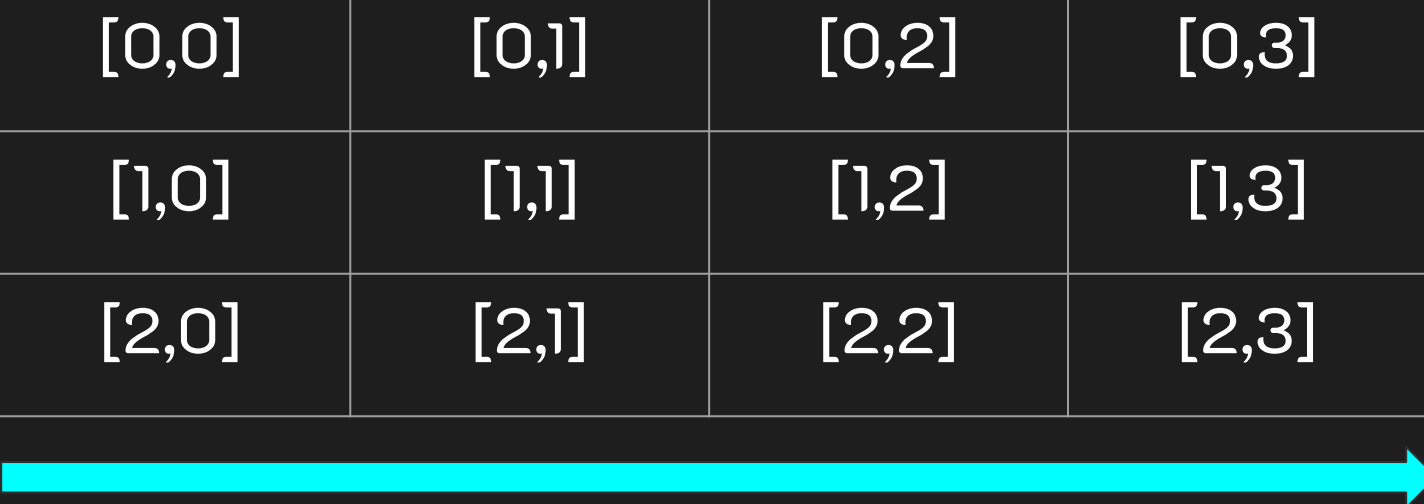
	คอลัมน์ที่ 0	คอลัมน์ที่ 1	คอลัมน์ที่ 2	คอลัมน์ที่ 3
แถวที่ 0				
แถวที่ 1				
แถวที่ 2				





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];`



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>



การสร้างอาร์เรย์ 2 มิติ

แบบกำหนดขนาดและค่าเริ่มต้น

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

ตัวอย่าง เช่น

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



การสร้างอาร์เรย์ 2 มิติ

แบบกำหนดขนาดและค่าเริ่มต้น

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

ตัวอย่าง เช่น

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

แถว 1



การสร้างอาร์เรย์ 2 มิติ

แบบกำหนดขนาดและค่าเริ่มต้น

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

ตัวอย่าง เช่น

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

แถว 1

{50,70,80,90},

แถว 2


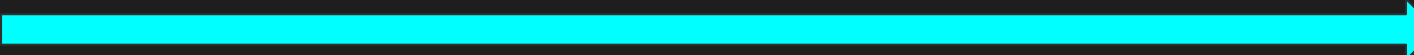
{100,99,60,55}

```
};
```



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



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

Nested Loop



Nested Loop

ในการเขียนโปรแกรมสามารถนำคำสั่งรูปแบบต่างๆ มาทำงานซ้อนกันได้เรียกว่า “ ลูปซ้อนลูป (Nested Loop)”



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>



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++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอกทำงาน 2 รอบ



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอกทำงาน 2 รอบ

Loop ในทำงาน 3 รอบ



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1

2



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1

2



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1

2



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1

2



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1

2



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

Loop ใน

1

2

3



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

2

Loop ใน

1

2

3



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

2

Loop ใน

1

2

3



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

2

Loop ใน

1

2

3

1



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

2

Loop ใน

1

2

3

1

2



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

2

Loop ใน

1

2

3

1

2

3



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

2

Loop ใน

1

2

3

1

2

3



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

2

Loop ใน

1

2

3

1

2

3



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

Loop นอก

1

2

Loop ใน

1

2

3

1

2

3



ตัวอย่าง Nested Loop

```
for(int i= 1; i<=2;i++){  
    cout<< i <<endl;  
    for(int j= 1; j<=3;j++){  
        cout<< j <<endl;  
    }  
}
```

จบการทำงาน!!

Loop นอก

1

2

Loop ใน

1

2

3

1

2

3

ฟังก์ชัน (Function)



ฟังก์ชัน (Function) คืออะไร

ชุดคำสั่งที่นำมาเขียนรวมกันเป็นกลุ่มเพื่อให้เรียกใช้งานตามวัตถุประสงค์ที่ต้องการและลดความซ้ำซ้อนของคำสั่งที่ใช้งานบ่อยๆ ฟังก์ชันสามารถนำไปใช้งานได้ทุกที่และแก้ไขได้ในภายหลัง ทำให้โค้ดในโปรแกรมมีระเบียบและใช้งานได้สะดวกมากยิ่งขึ้น





ประเภทของฟังก์ชัน

- **ฟังก์ชันมาตรฐาน (Standard Library Functions)** คือ ฟังก์ชันที่มีอยู่ในภาษา C++ ผู้ใช้สามารถเรียกใช้งานได้เลย เช่น `cout` , `cin` ที่ทำงานอยู่ในไลบรารี `iostream` เป็นต้น
- **ฟังก์ชันที่ผู้ใช้สร้างขึ้นเอง (User-Define Function)** คือ ฟังก์ชันที่ถูกสร้างขึ้นมาเพื่อวัตถุประสงค์ให้ทำงานตามที่ผู้ใช้ต้องการ



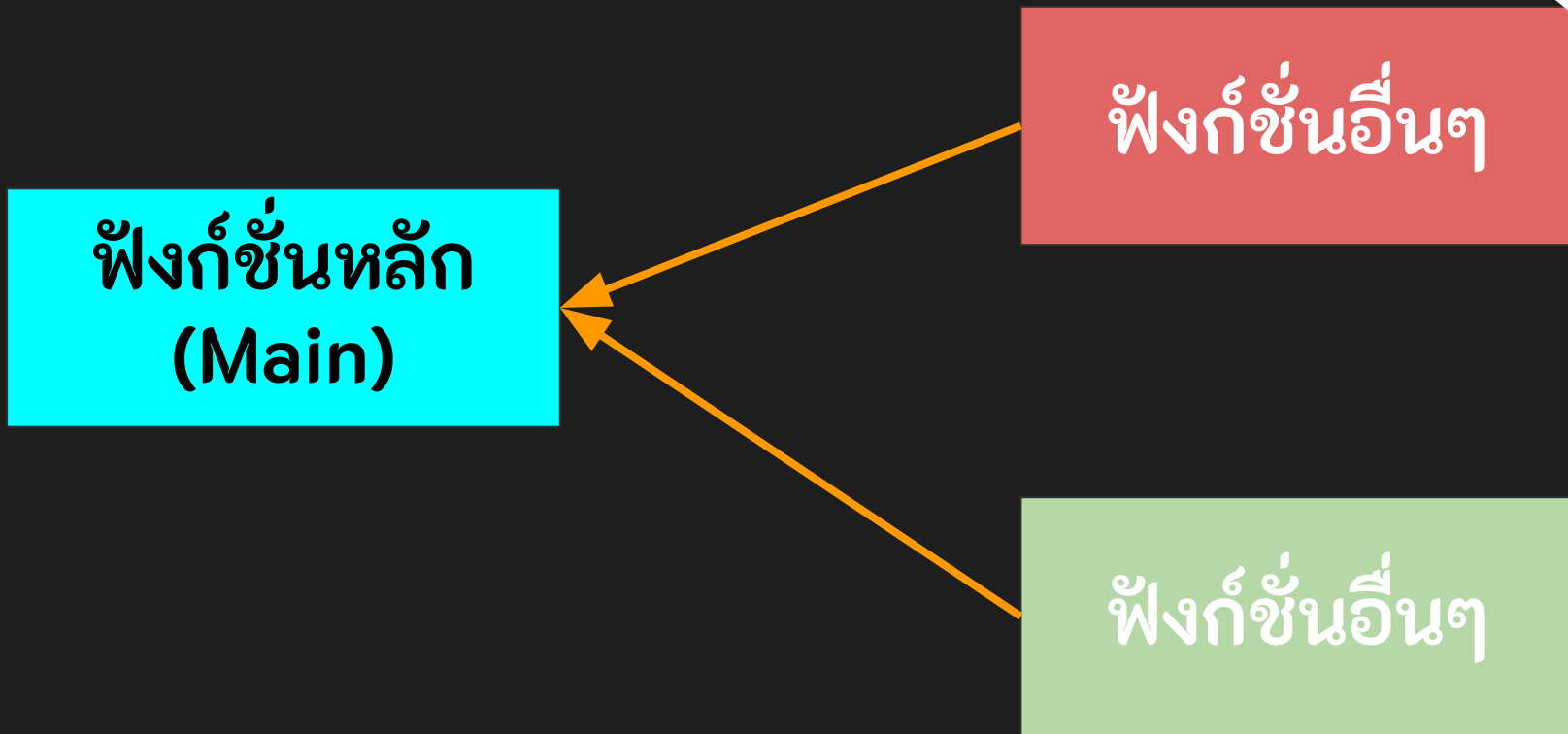


ฟังก์ชันเริ่มต้น (main)

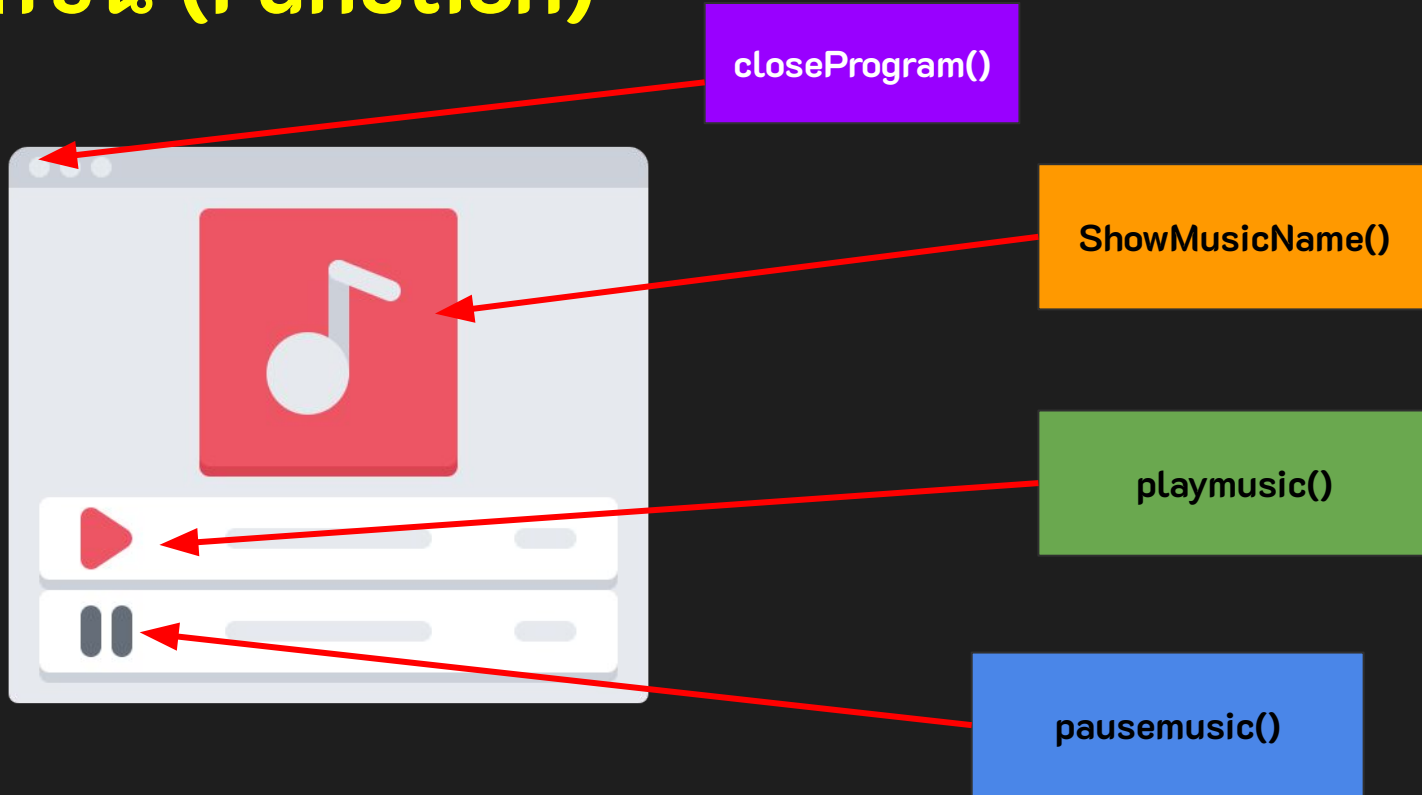
```
int main()
{
    //statement
}
```

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
```

```
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

```
ชื่อฟังก์ชัน ();
```


ฟังก์ชันแบบรับและส่งค่า



ฟังก์ชันแบบรับและส่งค่า

โครงสร้างคำสั่ง

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

การเรียกใช้งานฟังก์ชัน

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

ฟังก์ชันแบบกำหนดค่าเริ่มต้น



ฟังก์ชันแบบกำหนดค่าเริ่มต้น

เป็นการสร้างฟังก์ชันโดยกำหนดค่าเริ่มต้นให้กับพารามิเตอร์

โครงสร้างคำสั่ง

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

Function Overloading



กฎการตั้งชื่อฟังก์ชัน

- ชื่อฟังก์ชันต้องไม่ซ้ำกัน
- ชื่อฟังก์ชันสามารถตั้งเป็นตัวอักษรหรือตัวเลขได้
- ชื่อของฟังก์ชันต้องไม่ขึ้นต้นด้วยตัวเลข



Function Overloading

การสร้างฟังก์ชันที่มีชื่อเหมือนกันแต่สามารถรับ
พารามิเตอร์จำนวนต่างกันพร้อมคืนค่าที่แตกต่างกันได้



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

ขอบเขตตัวแปร



ขอบเขตตัวแปร

- **local variable** ตัวแปรที่ทำงานอยู่ในฟังก์ชันมีขอบเขตการทำงานตั้งแต่จุดเริ่มต้นไปจนถึงจุดสิ้นสุดของฟังก์ชัน
- **global variable** ตัวแปรที่ทำงานอยู่นอกฟังก์ชันมีขอบเขตการทำงานตั้งแต่จุดเริ่มต้นไปจนถึงจุดสิ้นสุดของไฟล์ที่ประกาศใช้





Local Variable

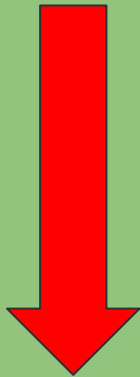
```
int main(){  
  
    int balance = 10000;  
  
}
```



Local Variable

```
int main(){
```

```
    int balance = 10000;
```



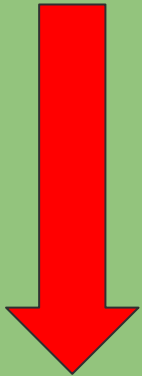
```
}
```

Local Variable



```
int main(){
```

```
    int balance = 10000;
```



```
}
```

```
void deposit (int amount){
```

```
    int value = amount;
```

```
}
```

```
void withdraw (int amount){
```

```
    int value = amount;
```

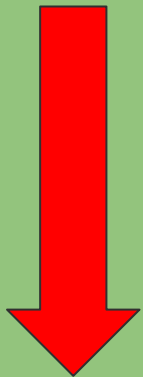
```
}
```

Local Variable



```
int main(){
```

```
    int balance = 10000;
```



```
}
```

```
void deposit (int amount){
```

```
    int value = amount;
```

```
}
```

```
void withdraw (int amount){
```

```
    int value = amount;
```

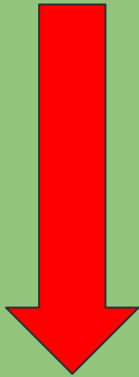
```
}
```



Local Variable

```
int main(){
```

```
    int balance = 10000;
```



```
}
```

```
void deposit (int amount){
```

```
    int value = amount;  
    balance+=value;
```

```
}
```

```
void withdraw (int amount){
```

```
    int value = amount;  
    balance-=value;
```

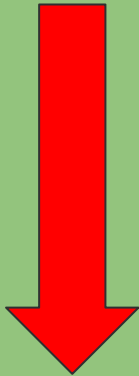
```
}
```

Local Variable



```
int main(){
```

```
    int balance = 10000;
```



```
}
```

```
void deposit (int amount){
```

```
    int value = amount;  
    balance+=value;
```

```
}
```



```
void withdraw (int amount){
```

```
    int value = amount;  
    balance-=value;
```

```
}
```



Global Variable



```
int balance = 1000;
int main(){
    .....
}

void deposit(int amount){
    .....
}
void withdraw(int amount){
    .....
}
```


Global Variable



```
int balance = 1000;
```

```
int main(){
```

```
.....
```

```
}
```

```
void deposit(int amount){
```

```
.....
```

```
}
```

```
void withdraw(int amount){
```

```
.....
```

```
}
```

Global Variable



```
int balance = 1000; //global variable
```

```
int main(){
```

```
.....
```

```
}
```

```
void deposit(int amount){
```

```
.....
```

```
}
```

```
void withdraw(int amount){
```

```
.....
```

```
}
```



Global Variable

```
int balance = 1000; // global variable
```

```
int main(){  
    cout<<balance<<endl;  
}
```

```
void deposit(int amount){  
    balance+=amount;  
}
```

```
void withdraw(int amount){  
    balance-=amount;  
}
```



Global Variable

```
int balance = 1000; // global variable
```

```
int main(){  
    cout<<balance<<endl;  
}
```

```
void deposit(int amount){  
    balance+=amount;  
}
```

```
void withdraw(int amount){  
    balance-=amount;  
}
```

รู้จักกับ Pointer



การสร้างตัวแปร

ตัวอย่าง

```
int number = 100;
```

ตัวแปร `number` เก็บค่าตัวเลขจำนวนเต็มคือเลข 100



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>



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



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>



การสร้างตัวแปร

ตัวอย่าง

```
int number = 100;
```

ตัวแปร number เก็บค่าตัวเลขจำนวนเต็มคือเลข 100

คำถาม : แล้วจะเข้าถึงค่าในหน่วยความจำได้อย่างไร ??



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>



การสร้างตัวแปร

ตัวอย่าง

```
int number = 100;
```

ตัวแปร number เก็บค่าตัวเลขจำนวนเต็มคือเลข 100

คำถาม : แล้วจะเข้าถึงค่าในหน่วยความจำได้อย่างไร ??

คำตอบ : ใช้พอยน์เตอร์ (Pointer)



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>



Pointer คืออะไร

พอยน์เตอร์ (Pointer) คือตัวแปรที่ใช้เก็บ
ตำแหน่งที่อยู่ของตัวแปรที่สนใจหรือค่าแอดเดรส
(Address) หน่วยความจำ ซึ่งมีประโยชน์อย่างมาก
สำหรับการเขียนโปรแกรมจัดการหน่วยความจำ



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

จัดการสตริง



จัดการสตริง

- การเชื่อมต่อสตริง (concatenation)
- การเข้าถึงตัวอักษรในสตริง
- เปรียบเทียบข้อความในสตริง



ฟังก์ชันจัดการสตริง



ฟังก์ชันจัดการสตริง

ชื่อฟังก์ชัน	ความหมาย
length()	หาความยาวสตริง
append()	เชื่อมต่อสตริง
empty()	เช็คค่าว่างในสตริง (0 = ไม่เป็นค่าว่าง , 1 = เป็นค่าว่าง)
compare()	เปรียบเทียบสตริง (0 = เหมือนกัน, 1 = ต่างกัน)



ฟังก์ชันจัดการสตริง

ชื่อฟังก์ชัน	ความหมาย
insert()	แทรกข้อความในสตริง
replace()	แทนที่ข้อความในสตริง

Enum



Enum คืออะไร

หมายถึง ตัวแปรที่เป็นรูปแบบตัวเลขจำนวนเต็ม ที่มีการตั้งชื่อเฉพาะขึ้นมาเพื่อเป็นตัวแทนของกลุ่มข้อมูล



การสร้าง Enum

```
enum ชื่อEnum {  
    key1, //ค่าเริ่มต้น = 0  
    key2,  
    ....  
}
```





การสร้าง Enum แบบกำหนดค่า

```
enum ชื่อEnum {  
    key1 = value1,  
    key2 = value2,  
    ....  
}
```



เวกเตอร์ (Vector)



เวกเตอร์ (Vector)

มีลักษณะการทำงานคล้ายอาร์เรย์แต่มีความ
สามารถในการปรับเปลี่ยนขนาดในการเก็บข้อมูล
สมาชิกได้ (Dynamic Size)



สรุปवेคเตอร์

1. ใช้เก็บกลุ่มของข้อมูล **ที่มีชนิดข้อมูลเดียวกัน**
2. ใช้ตัวแปรชื่อเดียวกัน
3. ใช้หมายเลขกำกับเพื่ออ้างอิงตำแหน่งของข้อมูล
4. **ขนาดการจัดเก็บข้อมูลมีความยืดหยุ่น**





การสร้างเวกเตอร์

นำเวกเตอร์เข้ามาทำงาน

```
#include <vector>
```

แบบไม่กำหนดค่าเริ่มต้น

```
vector<ชนิดข้อมูล> ชื่อตัวแปร
```

แบบกำหนดค่าเริ่มต้น

```
vector<ชนิดข้อมูล> ชื่อตัวแปร = {"สมาชิก"}
```




เข้าถึงสมาชิกในเวกเตอร์

```
vector<int> scores = {50,90,80,100};
```

50 (0)	99 (1)	80 (2)	100 (3)
--------	--------	--------	---------

```
scores.at(index);
```



ฟังก์ชันจัดการเวคเตอร์



ฟังก์ชันจัดการเวกเตอร์

ชื่อฟังก์ชัน	ความหมาย
size()	จำนวนสมาชิกในเวกเตอร์
at(index)	แสดงค่าในเวกเตอร์ ตำแหน่งที่ Index
push_back(element)	เพิ่มสมาชิกในเวกเตอร์



สตรัคเจอร์ (Structure)



สตรักเจอร์ (Structure)

คือ ข้อมูลแบบโครงสร้างที่นำเอาข้อมูลที่มีชนิดข้อมูลต่างกันมารวบรวมเข้าด้วยกัน แต่มีความสัมพันธ์ของข้อมูลแบบต่อกัน มาเก็บไว้ในโครงสร้างเดียวกัน

****เปรียบเทียบเหมือนกับสร้างชนิดข้อมูลขึ้นมาใช้งานเอง****



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>



การสร้างสตรัคเจอร์

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





การสร้างสตรัคเจอร์

```
struct student{  
    string name;  
    int score;  
};
```



```
student s1;  
s1.name = "KongRuksiam";  
s1.score = 90;
```



Type Aliases



Type Aliases คืออะไร

นอกจากชนิดข้อมูลพื้นฐานที่อยู่ในภาษา C++ เราสามารถตั้งชื่อหรือสร้างชนิดข้อมูลในรูปแบบของเราได้ เรียกว่า “**Type Aliases**” โดยใช้คีย์เวิร์ด **typedef** ซึ่งเป็นการนำชนิดข้อมูลเดิมของ C++ มากำหนดชื่อชนิดข้อมูลใหม่



Type Aliases คืออะไร

โครงสร้างคำสั่ง

```
typedef data_type ชื่อTypeAliases ;
```

ตัวอย่าง

- typedef int lek;
- typedef double lekjud;
- typedef string kum;

รู้จักกับ Generic



Generic คืออะไร

เป็นกระบวนการจัดการชนิดข้อมูลที่ระบุในฟังก์ชัน ให้มีความยืดหยุ่นตามการเรียกใช้งาน โดยรูปแบบการระบุชนิดข้อมูลนั้นจะเขียนในพื้นที่ `<>`



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

Generic Function



Generic Function

คือ การสร้างฟังก์ชันที่จัดการชนิดข้อมูลต่างกันได้

โครงสร้างคำสั่ง

```
template <class T>  
T funcName(T parameter){  
    //คำสั่งต่างๆ  
}
```



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>



Generic Function

คือ การสร้างฟังก์ชันที่จัดการชนิดข้อมูลต่างกันได้

โครงสร้างคำสั่ง

ระบุว่าเป็น Generic โดยให้ T เป็น
ตัวแทนของชนิดข้อมูลที่สนใจ

```
template <class T>
```

```
T funcName(T parameter){
```

```
//
```

ตัวแทนของชนิดข้อมูล

```
}
```



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>



Generic Function

การเรียกใช้งาน

- `funcName<type1>(type1 parameter)`
- `funcName<type2>(type2 parameter)`
- `funcName<type3>(type3 parameter)`
- `funcName<type4>(type4 parameter)`

