

ฟังก์ชัน (Function)



ฟังก์ชัน ?

ฟังก์ชัน หรือ โปรแกรมย่อย คือ สิ่งที่สามารถถูกเรียกใช้งานซ้ำ ๆ ได้จากโปรแกรมส่วนอื่น ๆ ตัวอย่างเช่น

- ฟังก์ชัน printf
- ฟังก์ชัน scanf

```
1  #include <stdio.h>
2
3  int main(void) {
4      int n;
5      scanf( "%d", &n );
6      printf( "Hello World.\n" );
7      return 0;
8  }
```

ข้อดีของฟังก์ชัน

- ทำให้เกิดการแบ่งโค้ดเป็นส่วน ๆ ที่มีเป้าหมายการทำงานชัดเจน
- เรียกใช้งานซ้ำได้ และลดความซ้ำซ้อน
- อ่านเข้าใจ และแก้ไขได้ง่าย
- การทำงานเป็นอิสระ

ประเภทของฟังก์ชัน

- ฟังก์ชันมาตรฐาน (Standard Function) เป็นฟังก์ชันที่ Compiler มีมาให้อยู่แล้ว ถูกจัดเป็นหมวดหมู่ในไลบรารีต่าง ๆ ในการใช้งานต้องเรียกใช้ Include Directives
- ฟังก์ชันที่ผู้เขียนโปรแกรมสร้างขึ้น (User-defined Function) เป็นฟังก์ชันที่ผู้เขียนโปรแกรมสร้างขึ้นมาใช้งานเองตามต้องการ

ฟังก์ชันมาตรฐาน (Standard Function)

ฟังก์ชันการคำนวณทางคณิตศาสตร์
`#include <math.h>`

ฟังก์ชัน	คำอธิบาย
<code>sin(x)</code>	หาค่า sine ของ x โดย x เป็นเรเดียน
<code>sqrt(x)</code>	หารากที่ 2 ของ x
<code>pow(x,y)</code>	หาค่าของ x ยกกำลัง y
<code>log10(x)</code>	หา logarithm ฐาน 10 ของ x
<code>abs(x)</code>	ค่าสัมบูรณ์ของ x

ฟังก์ชันมาตรฐาน (Standard Function)

ฟังก์ชันสำหรับข้อความ – #include <string.h>

ฟังก์ชัน	คำอธิบาย
strcpy(str1, str2)	คัดลอกข้อความจาก str2 ไปเก็บที่ str1
strcat(str1, str2)	ต่อข้อความใน str1 ด้วย str2
strncat(str1, str2, n)	ต่อข้อความใน str1 ด้วย str2 จำนวน n อักษร
strcmp(str1, str2)	เปรียบเทียบตัวอักษรในข้อความ (case sensitive) ถ้า str1 = str2 จะได้ 0
strlen(str)	หาความยาวข้อความ

โปรแกรมการเรียกใช้ฟังก์ชันมาตรฐาน

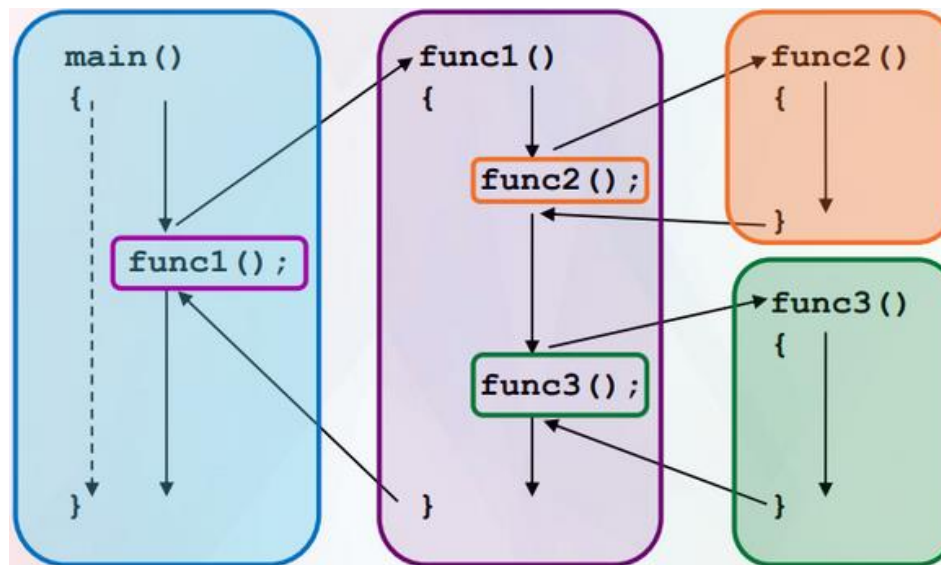
```
1  #include <stdio.h>
2  #include <string.h>
3  int main()
4  {
5      char str1[20]="Hello";
6      char str2[20];
7      int lenStr2;
8      strcpy(str2, str1);
9      strcat(str2, " C Programming.");
10     printf("%s\n", str2);
11     printf("str2 is string length = %d", strlen(str2));
12     return 0;
13 }
14
```

output

```
Hello C Programming.
str2 is string length = 20
-----
Process exited after 0.03667 seconds
Press any key to continue . . .
```

ฟังก์ชันที่สร้างขึ้น (User-defined Function)

- สร้างครั้งเดียว เรียกใช้ได้หลายครั้ง
- ตัวแปรที่ประกาศในฟังก์ชัน มีขอบเขตการใช้งานอยู่ในฟังก์ชันนั้นๆ
- ฟังก์ชันที่สร้างขึ้น จะถูกเรียกใช้ด้วย main หรือฟังก์ชันอื่นได้



โครงสร้างของฟังก์ชัน

```
return_type function-name(type para-1, ... , type para-n)
{
    local variable-declarations ;

    statement-1 ;
    statement-2 ;
    ...
    statement-n ;

    return (value) ;
}
```

โครงสร้างของฟังก์ชัน

```
return_type function-name(type para-1, ... , type para-n)
```

```
{
```

```
    local variable-declarations ;
```

```
    statement-1 ;
```

```
    statement-2 ;
```

```
    ...
```

```
    statement-n ;
```

```
    return (value) ;
```

```
}
```

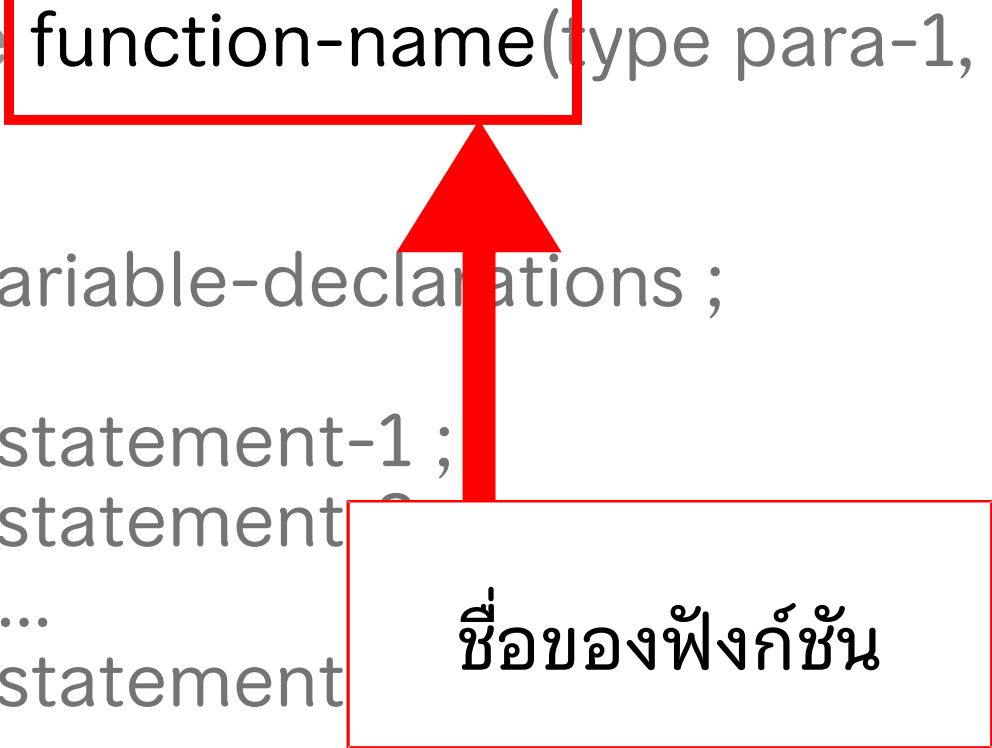
- ไม่มีการคืนค่า เช่น void
- มีการคืนค่า เช่น int , float

โครงสร้างของฟังก์ชัน

```
return_type function-name(type para-1, ... , type para-n)
{
    local variable-declarations ;

    statement-1 ;
    statement-2 ;
    ...
    statement-n ;

    return (value) ;
}
```



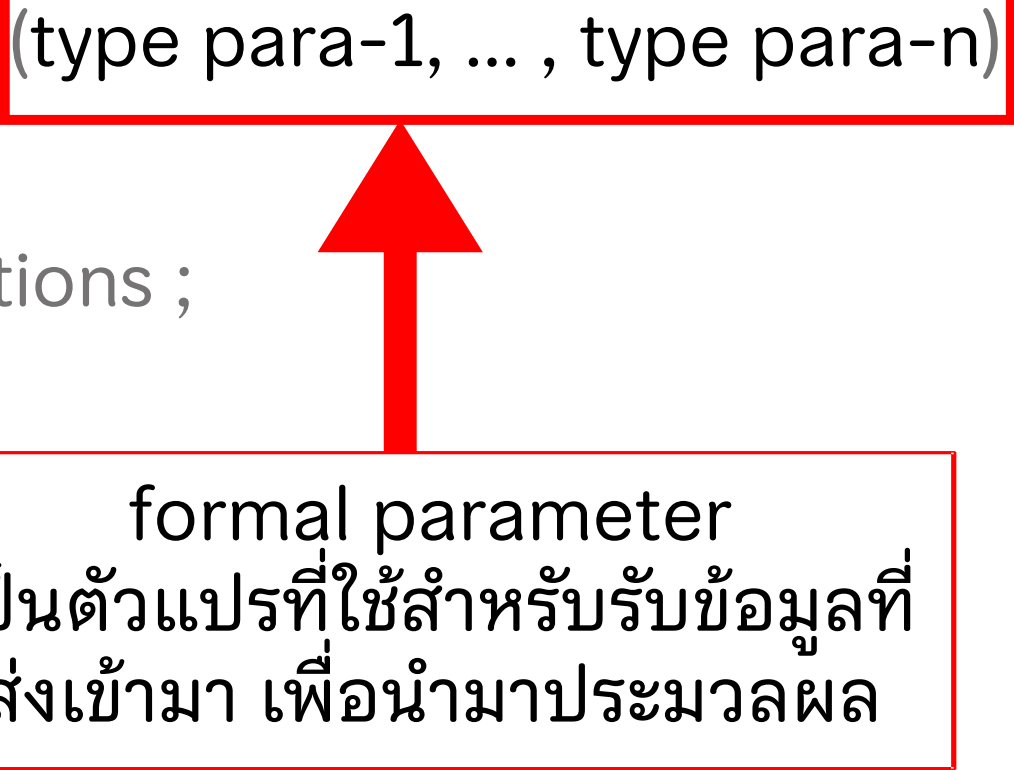
ชื่อของฟังก์ชัน

โครงสร้างของฟังก์ชัน

```
return_type function-name(type para-1, ... , type para-n)
{
    local variable-declarations ;

    statement-1 ;
    statement-2 ;
    ...
    statement-n ;

    return (value) ;
}
```



formal parameter
เป็นตัวแปรที่ใช้สำหรับรับข้อมูลที่
ส่งเข้ามา เพื่อนำมาประมวลผล

โครงสร้างของฟังก์ชัน

```
return_type function-name(type para-1, ... , type para-n)
```

```
{
```

```
    local variable-declarations ;
```

```
    statement-1 ;
```

```
    statement-2 ;
```

```
    ...
```

```
    statement-n
```

```
    return (value) ;
```

```
}
```

เป็นการสร้างตัวแปรเพื่อใช้งาน
ภายในฟังก์ชัน

โครงสร้างของฟังก์ชัน

```
return_type function-name(type para-1, ... , type para-n)  
{
```

```
    local variable-declarations ;
```

```
        statement-1 ;  
        statement-2 ;  
        ...  
        statement-n ;
```

```
    return (value) ;
```

```
}
```

ชุดคำสั่งที่ภายใน
ฟังก์ชัน



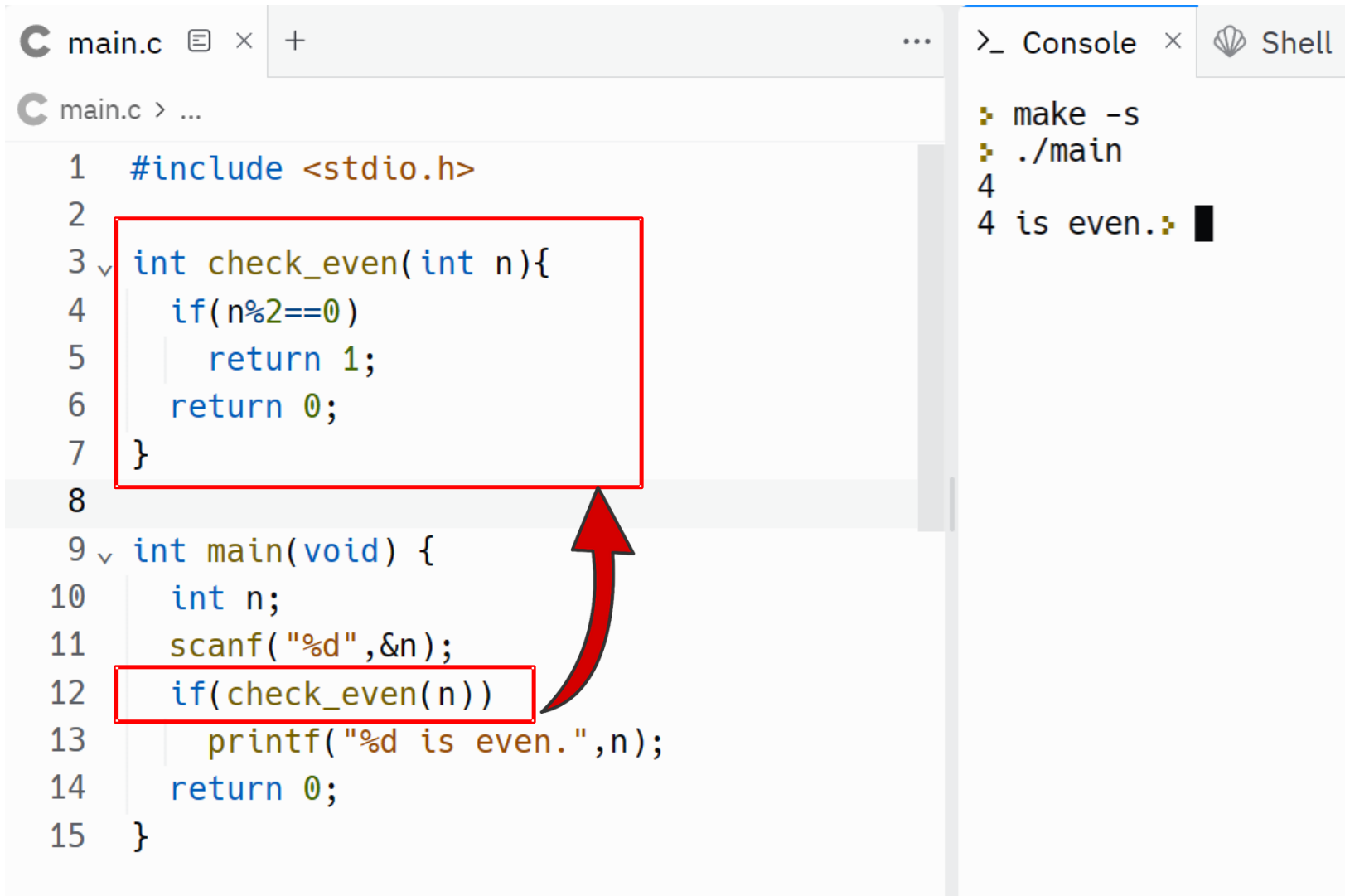
โครงสร้างของฟังก์ชัน

```
return_type function-name(type para-1, ... , type para-n)
{
    local variable-declarations ;

    statement-1 ;
    statement-2 ;
    ...
    statement-n ;
    return (value) ;
}
```

เป็นคำสั่งที่ใช้สำหรับคืนค่าออก
จากฟังก์ชัน **หากฟังก์ชันไม่มี
การคืนค่า ไม่ต้องใช้คำสั่งนี้

ตัวอย่างฟังก์ชัน



```
C main.c [icon] × + ...
C main.c > ...

1  #include <stdio.h>
2
3  int check_even(int n){
4      if(n%2==0)
5          return 1;
6      return 0;
7  }
8
9  int main(void) {
10     int n;
11     scanf("%d",&n);
12     if(check_even(n))
13         printf("%d is even.",n);
14     return 0;
15 }
```

>_ Console × Shell

```
➤ make -s
➤ ./main
4
4 is even.➤ █
```


ตัวอย่างการสร้างและเรียกใช้ฟังก์ชัน

ในการสร้างและเรียกใช้ฟังก์ชัน สามารถทำได้ดังนี้

1

C main.c > ...

```
1  #include <stdio.h>
2
3  int check_even(int n){
4      if(n%2==0)
5          return 1;
6      return 0;
7  }
8
9  int main(void) {
10     int n;
11     scanf("%d",&n);
12     if(check_even(n))
13         printf("%d is even.",n);
14     return 0;
15 }
```

ตัวอย่างการสร้างและเรียกใช้ฟังก์ชัน

ในการสร้างและเรียกใช้ฟังก์ชัน สามารถทำได้ดังนี้

C main.c > f main

2

```
1  int check_even(int n);
2  #include <stdio.h>
3
4  int main(void) {
5      int n;
6      scanf("%d",&n);
7      if(check_even(n))
8          printf("%d is even.",n);
9      return 0;
10 }
11
12 int check_even(int n){
13     if(n%2==0)
14         return 1;
15     return 0;
16 }
```

ต้นแบบฟังก์ชัน

- เราเรียกฟังก์ชันที่ถูกประกาศไว้ว่า ต้นแบบของฟังก์ชัน (Function Prototype)

- วิธีประกาศต้นแบบ คือ
ชนิดข้อมูล ชื่อฟังก์ชัน(พารามิเตอร์);
เช่น

`int max(int x, int y);`

`double findCircleArea(double radius);`

`int add_number(int x, int y);`

ตัวอย่างการสร้างและเรียกใช้ฟังก์ชัน

ในการสร้างและเรียกใช้ฟังก์ชัน สามารถทำได้ดังนี้

C main.c > f main

```
1 int check_even(int n);
```

Prototype

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

```
3
```

```
4 int main(void) {
```

```
5     int n;
```

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

```
7     if(check_even(n))
```

```
8         printf("%d is even.",n);
```

```
9     return 0;
```

```
10 }
```

```
11
```

```
12 int check_even(int n){
```

```
13     if(n%2==0)
```

```
14         return 1;
```

```
15     return 0;
```

```
16 }
```

Definition

ประโยชน์ของการประกาศและนิยามฟังก์ชันแยกกัน

C main.c > f main

```
1  int check_even(int n);
2  #include <stdio.h>
3
4  int main(void) {
5      int n;
6      scanf("%d",&n);
7      if(check_even(n))
8          printf("%d is even.",n);
9      return 0;
10 }
11
12 int check_even(int n){
13     if(n%2==0)
14         return 1;
15     return 0;
16 }
```

ทำให้เราสามารถ
นิยามฟังก์ชันไว้ด้าน
ใต้ของจุดที่ทำการ
เรียกใช้ได้

จุดเรียกใช้ฟังก์ชัน

ฟังก์ชัน

รูปแบบการรับ-ส่งค่าฟังก์ชัน

ฟังก์ชันสามารถรับ-ส่งค่าได้ 4 รูปแบบดังนี้

- ไม่มีการรับ-ส่งค่าใด ๆ
ตัวอย่างเช่น `void function_age (void)`
- รับค่าเข้าอย่างเดียว
ตัวอย่างเช่น `void function_age (int n)`
- ส่งค่ากลับอย่างเดียว
ตัวอย่างเช่น `float function_age (void)`
- รับค่าและส่งค่ากลับ
ตัวอย่างเช่น `float function_age (float n)`

ตัวอย่างฟังก์ชันที่ไม่มีการรับ-ส่งค่าใด ๆ

C main.c > f main

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

output

```
> make -s
> ./main
Hello World.
> □
```

ตัวอย่างฟังก์ชันที่รับค่าเข้าอย่างเดียว

C main.c > f main

```
1  #include <stdio.h>
2
3  void hello(char n[]){
4      printf("Hello %s.\n",n);
5  }
6
7  int main(void) {
8      char name[] = "C Programming";
9      hello(name);
10     return 0;
11 }
```

output

```
> make -s
> ./main
Hello C Programming.
> █
```


ตัวอย่างฟังก์ชันที่ส่งค่าออกอย่างเดียว

C main.c > f main

```
1  #include <stdio.h>
2
3  √ int getAge(void){
4      int age = 15;
5      return age;
6  }
7
8  √ int main(void) {
9      int age;
10     printf("My name is Computer.\n");
11     age = getAge();
12     printf("Age : %d\n",age);
13     return 0;
14 }
```

output

```
➤ make -s
➤ ./main
My name is Computer.
Age : 15
➤ █
```

ตัวอย่างฟังก์ชันที่มีการรับและส่งค่า

C main.c > f getSum

```
1  #include <stdio.h>
2
3  int getSum(int x,int y){
4      int sum = 0;
5      sum = x+y;
6      return sum;
7  }
8
9  int main(void) {
10     int n1,n2,s;
11     scanf("%d %d",&n1,&n2);
12     s = getSum(n1,n2);
13     printf("Sum is %d\n",s);
14     return 0;
15 }
```

output

```
> make -s
> ./main
10 20
Sum is 30
> █
```

กฎของการเรียกใช้ฟังก์ชัน

C main.c > ...

```
1  #include <stdio.h>
2
3  int A(int x) {
4      if(x > 0)
5          return 1;
6      else
7          return 0;
8  }
9
10 int B(int x) {
11     return !A(x);
12 }
13
14 int main(void) {
15     int x;
16     scanf("%d", &x);
17     printf("Positive Integer = %d\n", A(x));
18     printf("Negative Integer = %d\n", B(x));
19     return 0;
20 }
```



ฟังก์ชัน main เรียกใช้ A ได้



ฟังก์ชัน main เรียกใช้ B ได้



ฟังก์ชัน B เรียกใช้ A ได้



ฟังก์ชัน A เรียกใช้ B ได้หรือไม่?

กฎของการเรียกใช้ฟังก์ชัน

- ฟังก์ชัน main สามารถเรียกใช้ฟังก์ชันย่อยที่สร้างขึ้นได้
- ฟังก์ชันย่อยที่สร้างขึ้น สามารถเรียกใช้ฟังก์ชันย่อยอื่น ๆ ได้
- ทั้งฟังก์ชัน main และฟังก์ชันย่อย เรียกฟังก์ชันอื่นมากกว่าหนึ่งก็ได้
- ฟังก์ชันย่อยจะเรียกฟังก์ชันมาตรฐานได้ เช่น scanf และ printf เป็นต้น
- ฟังก์ชันย่อยสามารถเรียกใช้ฟังก์ชันตัวเองได้

พัก 15 นาที



พารามิเตอร์ (Parameter)

ฟังก์ชันที่มีการรับค่า สามารถแบ่งย่อยได้ตามลักษณะของพารามิเตอร์ที่ส่งให้ฟังก์ชัน ดังนี้

- **Pass by Value** เป็นการสำเนาค่าของ Actual parameter ไปใส่ไว้ใน Formal Parameter ดังนั้น หากค่าของ Formal Parameter เปลี่ยนไป **จะไม่มีผลกับค่า**ใน Actual Parameter
- **Pass by Reference** เป็นการส่งตำแหน่งหน่วยความจำของ Actual parameter ให้กับ Formal Parameter ทำให้ Formal Parameter **เสมือนเป็นตัวแปรเดียวกัน**กับ Actual Parameter

พารามิเตอร์ Pass by Value

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* run this program using the console */
5 int power(int x){
6     x=x*x;
7     return x;
8 }
9
10 int main(int argc, char *argv[])
11 {
12     int x,pw;
13     scanf("%d",&x);
14     printf("x before call power = %d\n",x);
15     pw = power(x);
16     printf("pw = %d\n",pw);
17     printf("x after call power = %d\n",x);
18     return 0;
19 }
```

เรียกใช้ฟังก์ชัน power
และส่งค่าตัวแปร x จาก
main ไปในตัวแปร x
ของ power

Workshop การใช้งานฟังก์ชันแบบ Pass by Value

ปัญหา : Three Number

จงเขียนโปรแกรมรับตัวเลขจำนวนเต็ม a , b และ c ที่ไม่ซ้ำกัน และ $-1000 < a, b, c < 1000$

ผลลัพธ์

บรรทัดแรก พิมพ์ตัวเลขที่มีค่ามากที่สุด
บรรทัดที่สอง พิมพ์ตัวเลขที่มีค่าน้อยที่สุด

ตัวอย่าง

ตัวอย่างที่ 1		ตัวอย่างที่ 2	
ข้อมูลเข้า	ผลลัพธ์	ข้อมูลเข้า	ผลลัพธ์
42	514	100	100
-29	-29	25	25
514		49	

พารามิเตอร์ Pass by Reference

Pass by Reference เป็นการส่งตำแหน่งหน่วยความจำของ Actual parameter ให้กับ Formal Parameter ทำให้ Formal Parameter เสมือนเป็นตัวแปรเดียวกันกับ Actual Parameter

องค์ความรู้พื้นฐานที่จำเป็นต่อการทำความเข้าใจ Pass by Reference

- ตำแหน่งหน่วยความจำ (Memory Address)
- ตัวแปร Pointer

ตัวอย่างฟังก์ชันรับค่าแบบพารามิเตอร์ Pass by Reference

C main.c > ...

```
1  #include <stdio.h>
2
3  void changeA(int *b){
4      *b=*b+100;
5  }
6
7  int main(void) {
8      int a=10;
9      printf("Before change a = %d\n",a);
10     changeA(&a);
11     printf("After change a = %d\n",a);
12     return 0;
13 }
```

output

```
> make -s
> ./main
Before change a = 10
After change a = 110
> □
```

ชนิดของตัวแปรในฟังก์ชัน

ชนิดของตัวแปรแยกตามขอบเขตการทำงาน

- **ตัวแปรชนิด Global Variable** เป็นตัวแปรที่ประกาศไว้ นอกฟังก์ชัน อยู่ส่วนหัวโปรแกรม สามารถใช้ได้ทุกที่ ทุกฟังก์ชันในโปรแกรม
- **ตัวแปรชนิด Local Variable** ถูกสร้างขึ้นภายในฟังก์ชัน การเปลี่ยนแปลงจะมีผลภายในฟังก์ชันเท่านั้น หากมีชื่อซ้ำกับ Global variable จะถือว่าเป็นคนละตัวแปรกัน ตัวแปรประเภทนี้ เมื่อโปรแกรมออกจากฟังก์ชันจะถูกทำลาย

ตัวแปรชนิด Global Variable

C main.c > ...

```
1  #include <stdio.h>
2  int num = 10;
3
4  void changeNum(void){
5      num = num + 20;
6  }
7  int main(void) {
8      printf("Before: num = %d\n", num);
9      changeNum();
10     printf("After: num = %d\n", num);
11     return 0;
12 }
```

ตัวแปร num เป็น Global Variable

output

```
> make -s
> ./main
Before: num = 10
After: num = 30
> □
```

ตัวแปรชนิด Local Variable

C main.c > f main

```
1  #include <stdio.h>
2  ✓ int sum(int num1,int num2){
3      int sum;
4      sum=num1+num2;
5      return sum;
6  }
7  ✓ int main(void) {
8      int x,y;
9      scanf("%d", &x);
10     scanf("%d", &y);
11     printf("Sum is %d\n",sum(x,y));
12     return 0;
13 }
```

ตัวแปร sum เป็น Local Variable

ตัวแปร x,y เป็น Local Variable

การส่งอาร์เรย์เป็นพารามิเตอร์ให้กับฟังก์ชัน

Array มีขนาดใหญ่เล็กแค่ไหนนั้น ขึ้นอยู่กับผู้เขียนโปรแกรมเป็นผู้กำหนด ดังนั้นในหลายภาษาคอมพิวเตอร์ (รวมถึงภาษา C) กำหนดให้กลไกการส่ง Array เข้าไปทำงานในฟังก์ชันอยู่ในรูปแบบ Pass by Reference ทั้งนี้เนื่องจาก

- ประหยัดพื้นที่หน่วยความจำ (ไม่ต้องจองหน่วยความจำขึ้นอีก 1 ชุดที่มีขนาดเท่ากับ Array ที่เป็น Actual Parameter)
- ลดการประมวลผล CPU ในการสำเนาข้อมูลจาก Actual สู่ Formal Parameter
- ลดการประมวลผล CPU ในการส่งค่า Array ทั้งชุดออกจากฟังก์ชัน (กรณีที่ต้องการส่ง Array ที่ส่งเข้าไปกลับมาประมวลผลต่อ)

ตัวอย่าง ฟังก์ชันที่รับค่าเป็นอาร์เรย์

C main.c > ...

```
1  #include <stdio.h>
2  void exchangeYen(float yen[]){
3      int i;
4      printf("***Yen After exchange Thai Baht***\n");
5      for(i=0;i<3;i++){
6          printf("%.2f\n",yen[i]*4.09);
7      }
8  }
9
10 int main(void) {
11     float baht[3]={50,100,200};
12     int i;
13     printf("***Thai Baht Before exchange Yen***\n");
14     for(i=0;i<3;i++){
15         printf("%.2f\n",baht[i]);
16     }
17     exchangeYen(baht);
18     return 0;
19 }
```

พารามิเตอร์เป็นชนิด Array

output

```
50.00
100.00
200.00
***Yen After exchange Thai Baht***
204.50
409.00
818.00
```

ส่งพารามิเตอร์ Array ให้ฟังก์ชัน

ฟังก์ชันที่รับค่าเป็นอาร์เรย์ Pass by Reference

พารามิเตอร์เป็นชนิด Pointer

```
1 #include <stdio.h>
2 void print(int* arr,int n)
3 {
4     int i;
5     for (i = 0; i < n; i++)
6         printf("%d\t",arr[i]);
7 }
8
9 void add10(int* arr,int n)
10 {
11     int i;
12     for (i = 0; i < n; i++)
13         arr[i]+=10;
14 }
15
16 int main()
17 {
18     int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
19     int n = sizeof(arr) / sizeof(arr[0]);
20     printf("*****Before call add10*****\n");
21     print(arr,n);
22     add10(arr,n);
23     printf("\n*****After call add10*****\n");
24     print(arr,n);
25     return 0;
26 }
```

output

*****Before call add10*****

1	2	3	4	5	6	7	8
*****After call add10*****							
11	12	13	14	15	16	17	18

Process exited after 0.008557 seconds with return value 0
Press any key to continue . . .

ส่งพารามิเตอร์ Array ให้ฟังก์ชัน

ฟังก์ชันเรียกซ้ำ (Recursive Function)

- เป็นฟังก์ชันที่สร้างขึ้นเอง โดยภายในฟังก์ชันจะมีการเรียกชื่อของตัวเองซ้ำ ๆ
- เป็นฟังก์ชันที่ใช้เมื่อคำตอบสามารถหาแบบต่อเนื่องกัน
- ฟังก์ชันแบบเรียกตัวเองแบบนี้ จะต้องมีจุดที่ให้ออกจากฟังก์ชันได้

ฟังก์ชันเรียกซ้ำ (Recursive Function)

ตัวอย่างเช่น

$$N! = N \times (N-1) \times (N-2) \times \dots \times 3 \times 2 \times 1$$

recursive จะถูกเรียกซ้ำจนเมื่อ $N=0$ จะได้ว่า $0! = 1$

เช่น แสดงการหา $5! = ?$

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

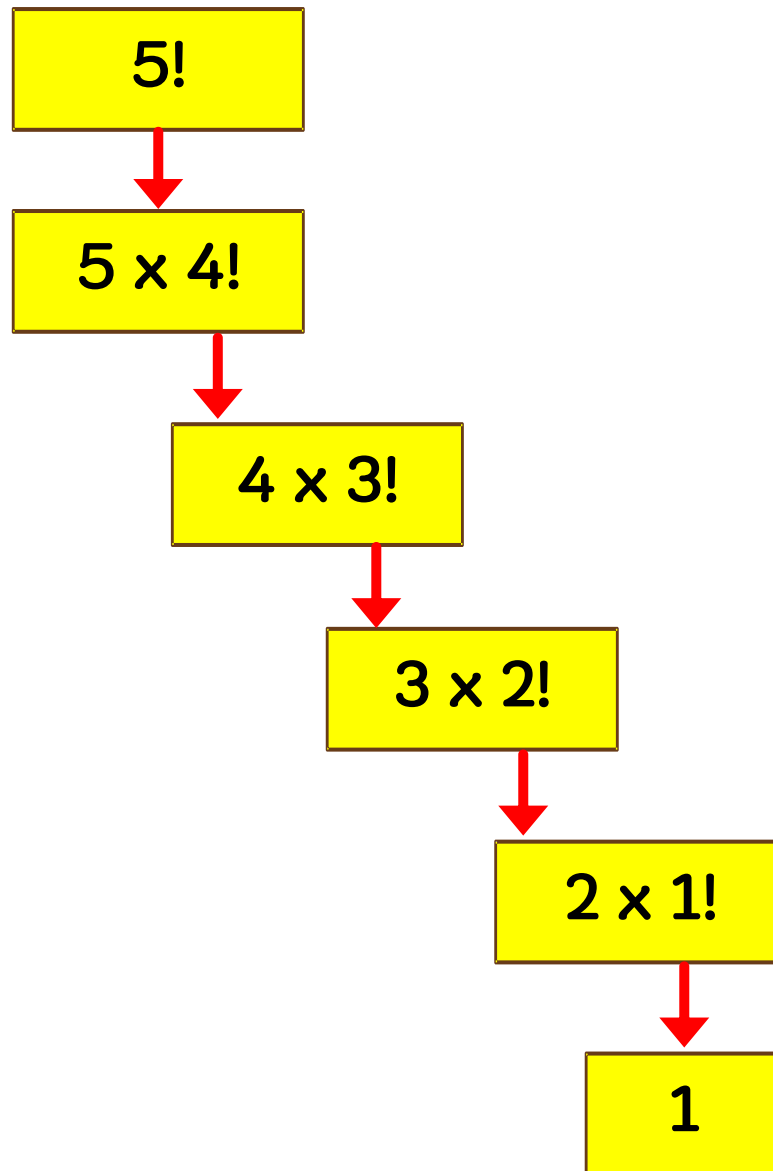
$$2! = 2 \times 1!$$

$$1! = 1 \times 0!$$

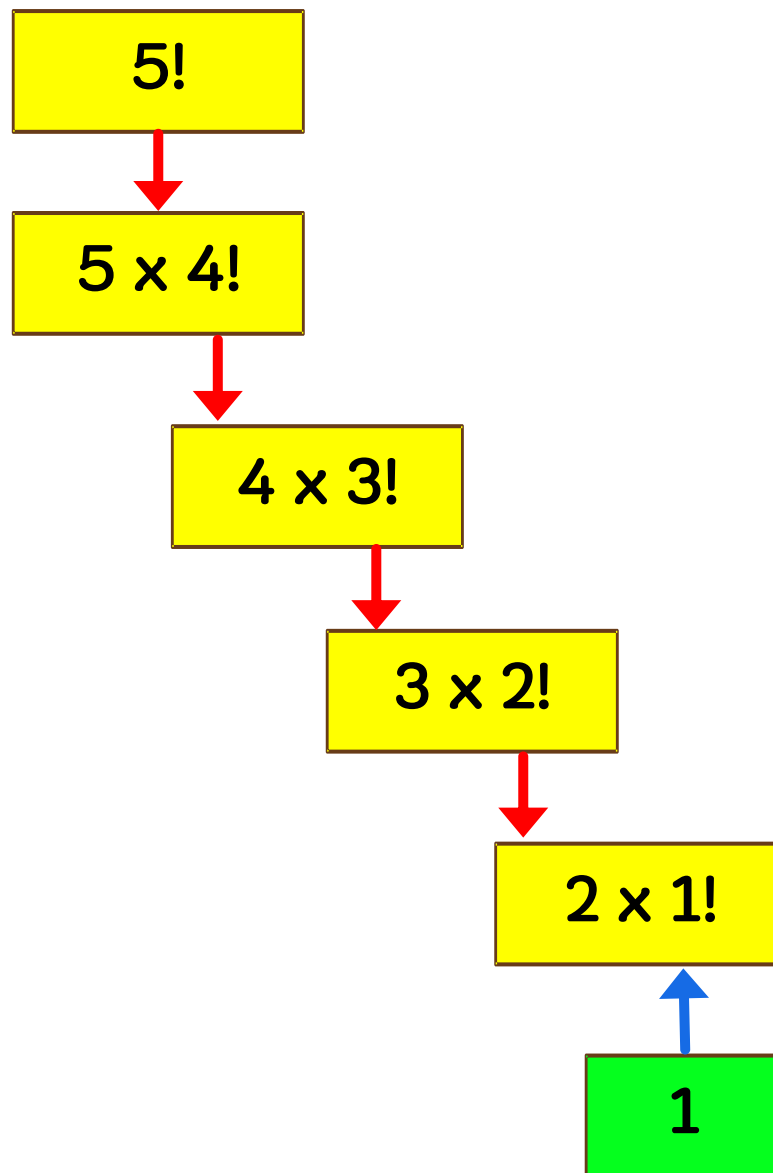
สามารถเขียนได้เป็น

$$N! = N \times (N-1)!$$

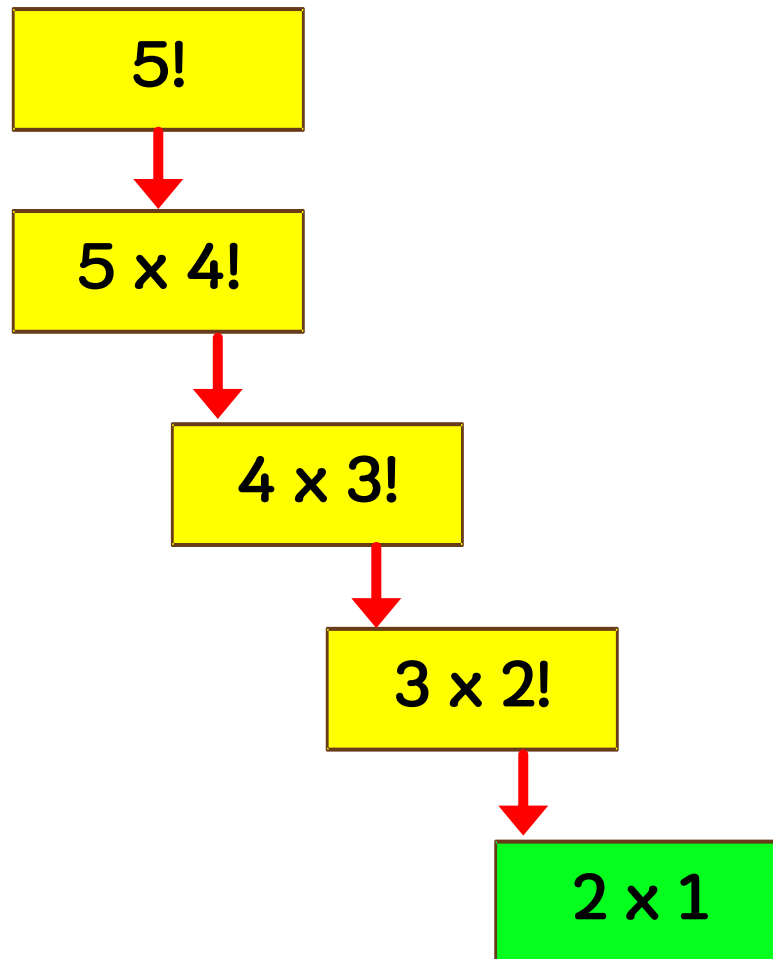
การคืนค่าของฟังก์ชัน Recursive Function



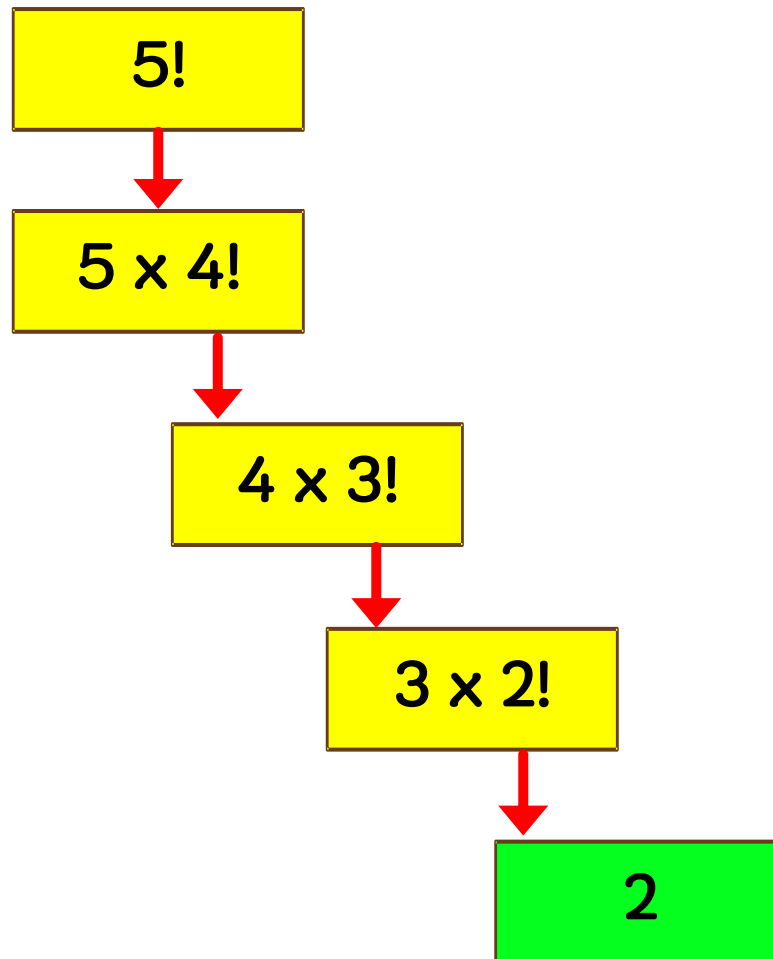
การคืนค่าของฟังก์ชัน Recursive Function



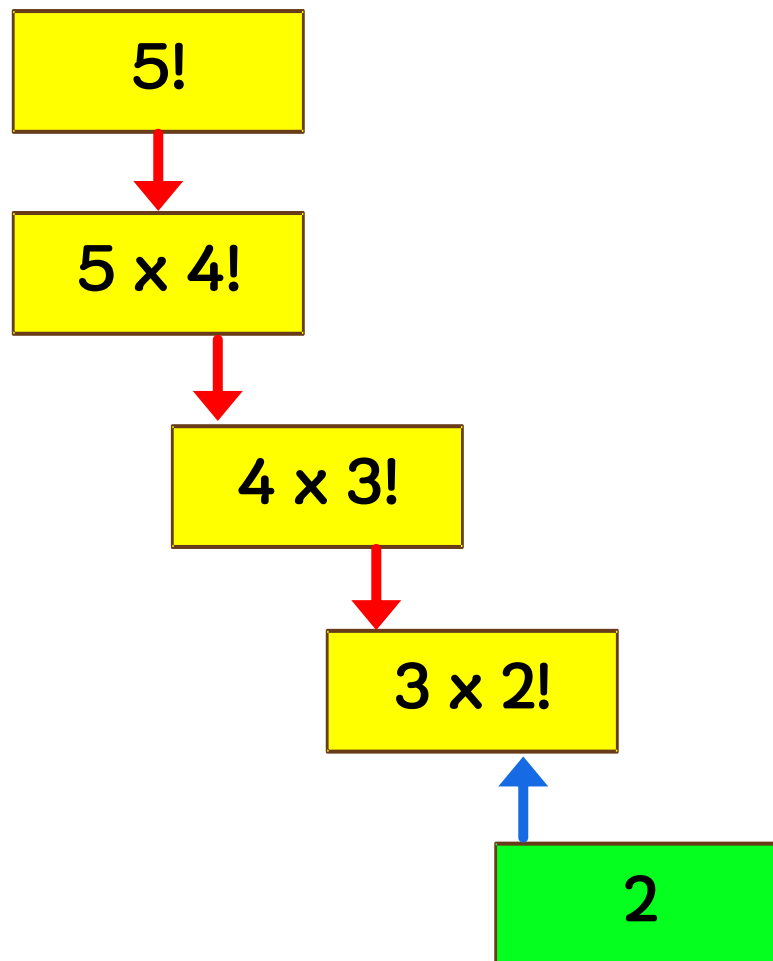
การคืนค่าของฟังก์ชัน Recursive Function



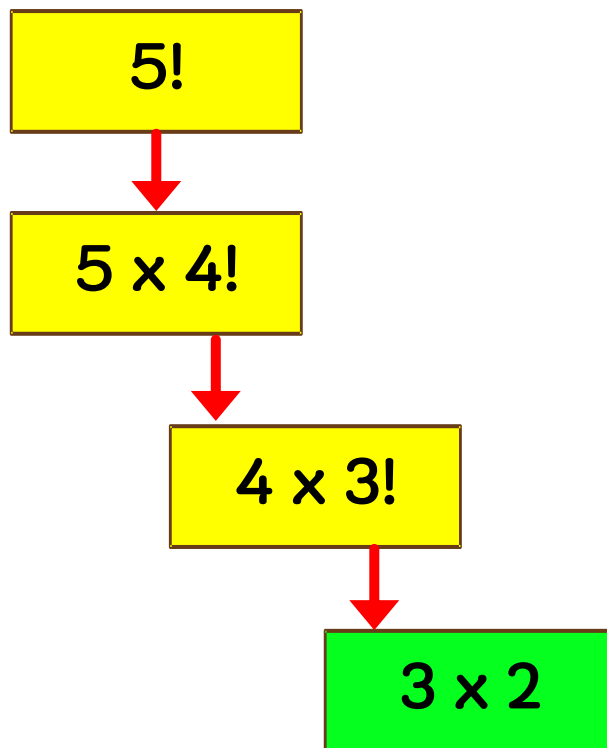
การคืนค่าของฟังก์ชัน Recursive Function



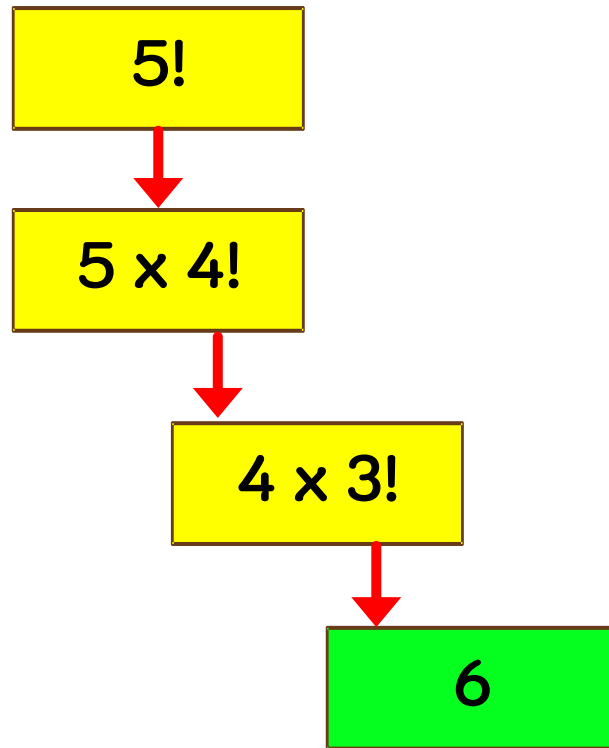
การคืนค่าของฟังก์ชัน Recursive Function



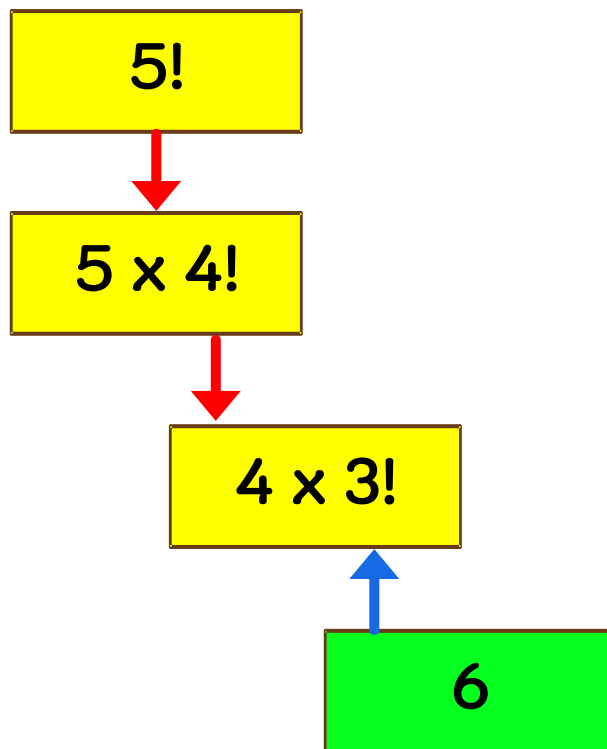
การคืนค่าของฟังก์ชัน Recursive Function



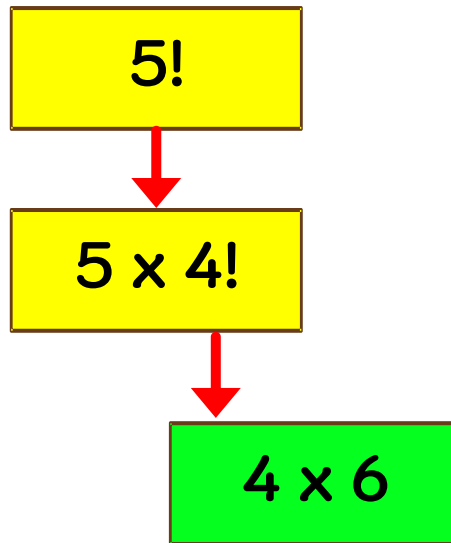
การคืนค่าของฟังก์ชัน Recursive Function



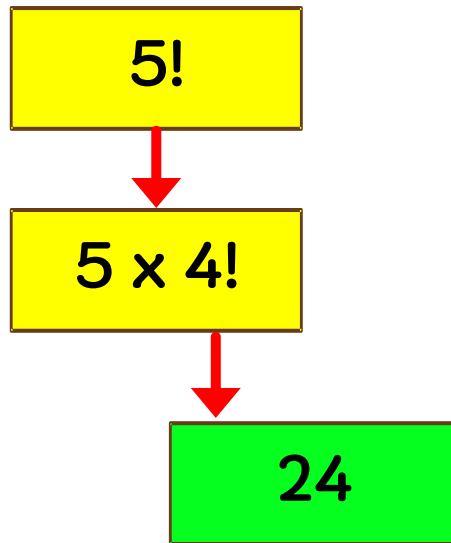
การคืนค่าของฟังก์ชัน Recursive Function



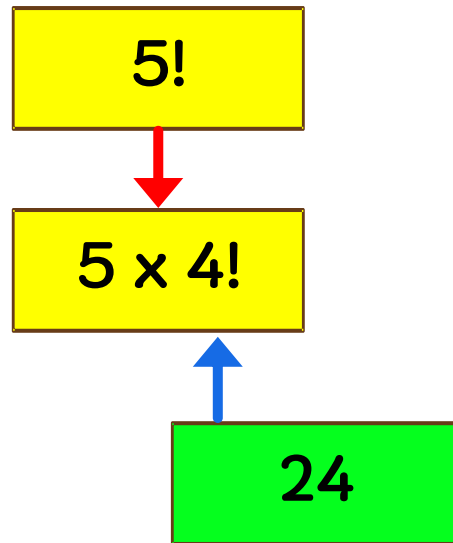
การคืนค่าของฟังก์ชัน Recursive Function



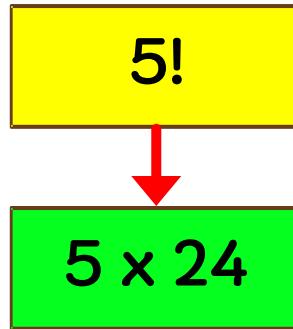
การคืนค่าของฟังก์ชัน Recursive Function



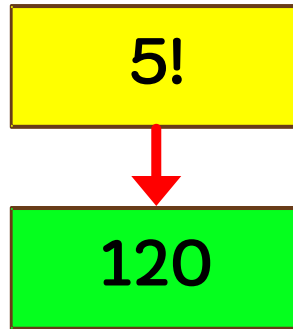
การคืนค่าของฟังก์ชัน Recursive Function



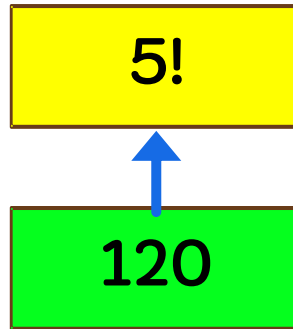
การคืนค่าของฟังก์ชัน Recursive Function



การคืนค่าของฟังก์ชัน Recursive Function



การคืนค่าของฟังก์ชัน Recursive Function



การคืนค่าของฟังก์ชัน Recursive Function

5!

=

120

ตัวอย่างฟังก์ชัน Recursive Function

output

```
5
Result is 120
-----
Process exited after 2.856
Press any key to continue .
```

```
1  #include <stdio.h>
2
3  int factorial(int n)
4  {
5      if(n==1)
6          n=1;
7      else
8          n=n*factorial(n-1);
9      return n;
10 }
11
12 int main()
13 {
14     int n,result=0;
15     scanf("%d",&n);
16     result = factorial(n);
17     printf("Result is %d",result);
18     return 0;
19 }
20
```

เรียกฟังก์ชันตัวเองซ้ำ

เรียกใช้งานฟังก์ชันจาก main

Workshop การใช้งาน Recursive Function

ปัญหา : recursive number

จงเขียนโปรแกรมรับตัวเลขจำนวนเต็ม n ที่ $2 < n < 100$ และเรียกใช้ฟังก์ชันแบบ Recursive เพื่อทำการหาผลบวก

ผลลัพธ์

มี 1 บรรทัด เป็นผลลัพธ์จากการเรียกใช้ฟังก์ชัน

ตัวอย่าง

ตัวอย่างที่ 1		ตัวอย่างที่ 2		ตัวอย่างที่ 3	
ข้อมูลเข้า	ผลลัพธ์	ข้อมูลเข้า	ผลลัพธ์	ข้อมูลเข้า	ผลลัพธ์
5	15	50	1275	100	5050

?



สวัสดี

