ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ คณะวิทยาศาสตร์ประยุกต์
มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

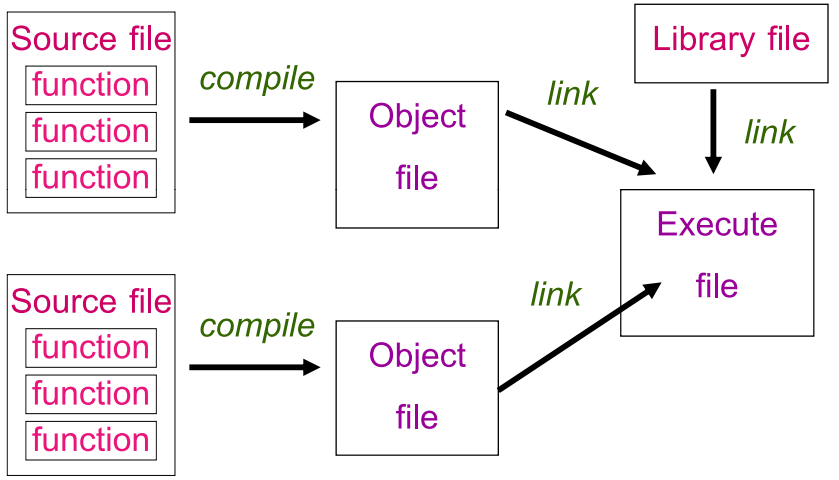
Computer Programming II
การเขียนโปรแกรมคอมพิวเตอร์2
LECTURE#7 ฟังก์ชัน (Function)

อ.สถิตย์ ประสมพันธ์

ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ

KMUTNB

ขั้นตอนการสร้างโปรแกรมด้วยภาษา C

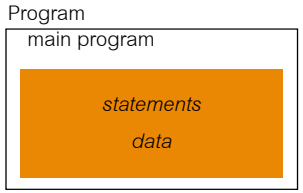


ฟังก์ชันคืออะไร

- **ฟังก์ชัน(function)** คือ ส่วนของโปรแกรมที่ทำงานเสร็จสิ้นภายในตัวเอง มีลักษณะเหมือนกับโปรแกรมย่อย ที่รวมอยู่ในโปรแกรมหลักอีกทีหนึ่ง
- ใช้หลักการแบ่งแยกและรวม(*Divide and Conquer)*
  - แบ่งการทำงานออกเป็นส่วนเล็กๆ ที่ทำงานเสร็จสมบูรณ์ในตัว
  - มีการทดสอบและแก้ไขส่วนเล็กๆนี้
  - ประกอบส่วนเล็กๆนี้ ขึ้นมาเป็นโปรแกรมขนาดใหญ่ที่สมบูรณ์ในขั้นตอนสุดท้าย

Unstructured Programming

รูปแบบ : โปรแกรมเขียนเรียงไปเรื่อยๆตามลำดับการประมวลผล



ข้อเสีย : เมื่อโปรแกรมเริ่มมีขนาดใหญ่ ทำให้ยากต่อการดูแลหรือปรับปรุงเปลี่ยนแปลงเช่น

- การหาจุดผิดพลาด ทำได้ยากขึ้น
- เกิดการใช้ตัวแปรซ้ำซ้อนได้ง่ายขึ้น

Unstructured Programming

```
#include <stdio.h>
main()
{
 int first, second, third;
 printf("\n F(X) = 3X + 10 if X > 0\n");
 printf("\n F(X) = 10 if X = 0\n");
 printf("\n Enter 3 values\n");
 scanf("%d %d %d", &first, &second, &third);
 if (first > 0)
    printf("F(%d) is %d\n", first, 3*first + 10);
 else
    printf("F(%d) is 10\n", first);
 if (second > 0)
    printf("F(%d) is %d\n", second, 3*second + 10);
else
    printf("F(%d) is 10\n", second);
if (third > 0)
    printf("F(%d) is %d\n", third, 3*third + 10);
else
    printf("F(%d) is 10\n", third);
}
```

Code program ซ้ำซ้อน

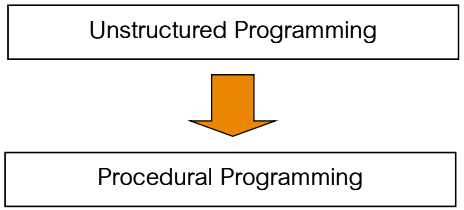
คำนวณ first

คำนวณ second

คำนวณ third

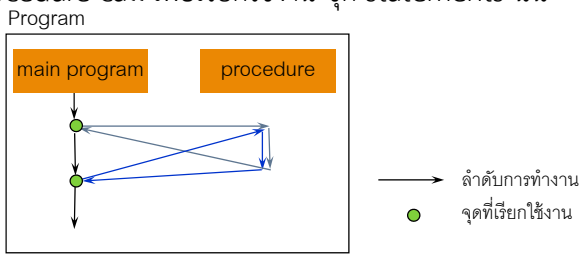
เหตุผลของการใช้ฟังก์ชัน

โปรแกรมเริ่มซับซ้อนมีขนาดใหญ่ มีการเปลี่ยนแปลงโครงสร้างของโปรแกรม



Procedural Programming

รูปแบบ : ชุด statements สามารถรวมไว้ในที่ที่หนึ่งแยกจากส่วนของ main และมีการใช้ procedure call เพื่อเรียกใช้งาน ชุด statements นั้น

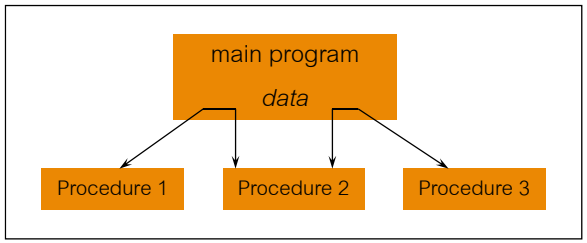


ข้อดี :

- มีการซ่อนรายละเอียดซับซ้อนไว้ใน Procedure ทำให้โปรแกรมหลักง่ายในการทำความเข้าใจ
- ลดความซ้ำซ้อนของส่วนโปรแกรมที่เหมือนๆกันไว้ใน Procedure เดียวกัน
- การหาข้อผิดพลาดของโปรแกรมง่ายขึ้น โดยแยกหาระหว่างส่วน main กับ procedures ใดๆ

Procedural Programming

การทำงาน main เรียกใช้งาน procedure โดยอาจมีการส่งค่าให้กับการเรียกแต่ละครั้ง เราอาจมองได้ว่า main เป็นตัวประสานงานระหว่างหลายๆ procedures และเป็นส่วนคอยควบคุมข้อมูลหลัก



สรุป Procedural Programming มีหนึ่งโปรแกรมหลักที่ประกอบไปด้วยส่วนย่อยๆ เรียกว่า procedures หรือฟังก์ชัน

Procedural Programming

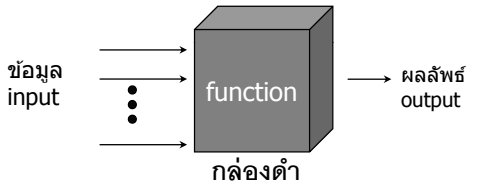
```
#include <stdio.h>
void get_Fx(int x);
main()
{
 int first, second, third;
 printf("\n F(X) = 3X + 10 if X > 0\n");
 printf("\n F(X) = 10 if X = 0\n");
 printf("\n Enter 3 values\n");
 scanf("%d %d %d", &first, &second, &third);
 get_Fx(first);
 get_Fx(second);
 get_Fx(third);
}

void get_Fx(int x)
{
 if (x > 0)
   printf("F(%d) is %d\n", x, (3*x) + 10);
 else
   printf("F(%d) is 10\n", x);
}
```

- รวมการทำงานแบบเดียวกันไว้ด้วยกัน
- เปลี่ยนแปลงตัวแปร x ในการเรียกใช้งานแต่ละครั้ง

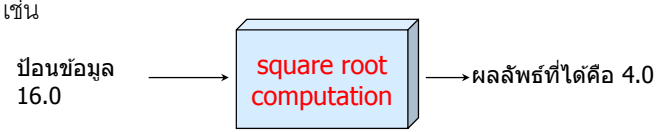
ฟังก์ชันในภาษา C

ฟังก์ชัน หรือ Procedure คือ

- ชุดของ statements ที่ทำงานอย่างใดอย่างหนึ่ง และมีชื่อเรียก
- ส่วนอื่นของโปรแกรมสามารถเรียกใช้งานฟังก์ชันได้

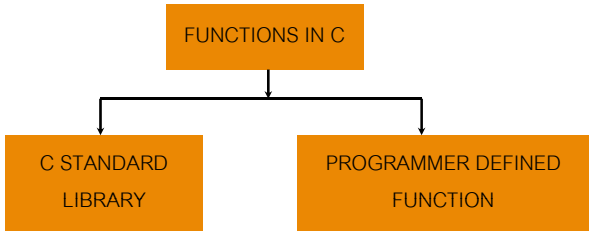


เช่น



ฟังก์ชันในภาษา C

- โปรแกรมภาษา C ประกอบไปด้วยหนึ่งฟังก์ชัน (main) หรือมากกว่า
- แต่ละฟังก์ชันประกอบไปด้วยหนึ่ง statement หรือมากกว่า
- ภาษา C แบ่งฟังก์ชันเป็น 2 แบบ
  - ฟังก์ชันมาตรฐานใน C
  - ฟังก์ชันที่สร้างโดยผู้เขียนโปรแกรม



ฟังก์ชันมาตรฐานใน C (C Standard Library)

- ฟังก์ชันที่ผู้ผลิต C compiler เป็นผู้เขียนขึ้น โดยเก็บไว้ใน C library
- ประกอบด้วยฟังก์ชันเกี่ยวกับ
  - disk I/O (input/output), standard I/O ex. printf(), scanf(), ...
  - string manipulation ex. strlen(), strcpy(), ...
  - mathematics ex. sqrt(), sin(), cos(), ...
- สามารถเรียกใช้งานได้เลย แต่ต้องมีการ include header file ที่นิยามฟังก์ชันนั้นๆไว้ เช่น

  จะใช้ printf(), scanf() ต้องเขียน **#include <stdio.h>**

  จะใช้ sqrt(), sin(), cos() ต้องเขียน **#include <math.h>**

ฟังก์ชันมาตรฐานทางคณิตศาสตร์

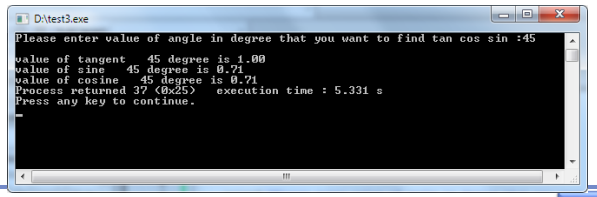
- ถ้าต้องการใช้ฟังก์ชันมาตรฐานทางคณิตศาสตร์จะต้องนำเข้า Standard Library Files 2ไฟล์ คือ
  - math.h และ stdlib.h
- ฟังก์ชันทางคณิตศาสตร์ เป็นฟังก์ชันที่ใช้ทางการคำนวณทางคณิตศาสตร์ ปกติอยู่ใน math.h ผลลัพธ์ ที่ได้จากฟังก์ชันกลุ่มนี้เป็นข้อมูลประเภท double ดังนั้นตัวแปรที่ใช้จึงเป็นพวกที่มีชนิดเป็น double

ฟังก์ชันมาตรฐานทางคณิตศาสตร์

ชื่อฟังก์ชัน	หน้าที่
sin(x)	ใช้คำนวณหาค่าของ sine โดย x มีค่าของมุมในหน่วย เรเดียน
cos(x)	ใช้หาค่า cosine โดย x มีหน่วยเป็นเรเดียน(radian)
tan(x)	ใช้หาค่า tangent โดย x มีหน่วยเป็นเรเดียน(radian)
sqrt(x)	ใช้หาค่ารากที่สองของ x โดย x เป็นตัวเลขหรือตัวแปรที่ไม่ติดลบ
exp(x)	ใช้หาค่า ex โดย e มีค่าประมาณ 2.718282
pow(x,y)	ใช้หาค่า x y
log(x)	ใช้หาค่า log ฐาน e เรียกว่า natural logarithm โดย x เป็นตัวเลขหรือตัวแปรที่ไม่ติดลบ
log10(x)	ใช้หาค่า log ฐาน 10 โดย x เป็นตัวเลขหรือตัวแปรที่ไม่ติดลบ
ceil(x)	ใช้ในการปัดเศษทศนิยมของ x เมื่อ x เป็นเลขทศนิยม
floor(x)	ใช้ในการตัดเศษทศนิยมของ x ทิ้งเมื่อ x เป็นเลขทศนิยม
fabs(x)	ใช้ในการหาค่าสัมบูรณ์ของค่าคงที่หรือตัวแปรที่มีทศนิยม โดยเป็นบวกหรือลบก็ได้

ฟังก์ชันมาตรฐานทางคณิตศาสตร์

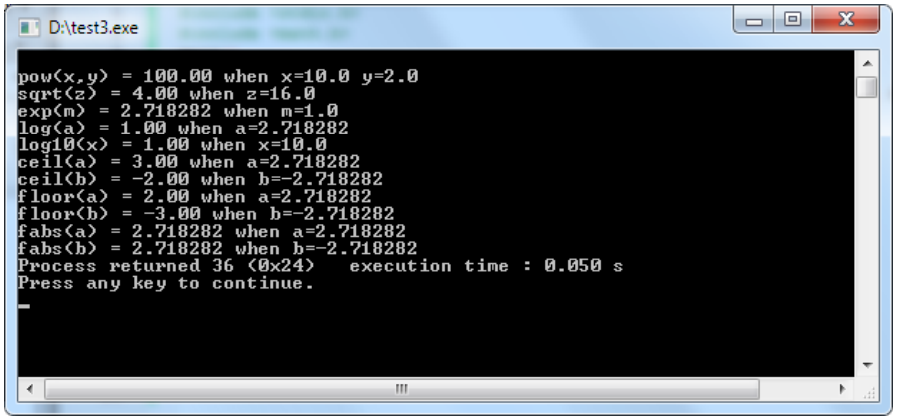
```
#include <stdio.h>
#include <math.h>
main()
{
    float deg , angle, pi = 3.141592654;

    printf("Please enter value of angle in degree that you want to find
tan cos sin :");
    scanf("%f",&deg);
    angle = deg * pi / 180;
    printf("\nvalue of tangent %4.0f degree is %4.2f ",deg,tan(angle));
    printf("\nvalue of sine %4.0f degree is %4.2f ",deg,sin(angle));
    printf("\nvalue of cosine %4.0f degree is %4.2f ",deg,cos(angle));

}
```



ฟังก์ชันมาตรฐานทางคณิตศาสตร์

```
#include <stdio.h>
#include <math.h>
main()
{

   double x = 10.0, y = 2.0 ,z = 16.0,a = 2.718282,

          b = -2.718282 , m=1.0;

    printf("\npow\(x,y\) = %4.2f when x=10.0 y=2.0", pow(x,y));
    printf("\nsqrt\(z\) = %4.2f when z=16.0", sqrt(z));
    printf("\nexp\(m\) = %4.6f when m=1.0",exp(m));
    printf("\nlog\(a\) = %4.2f when a=2.718282",log(a));
    printf("\nlog10\(x\) = %4.2f when x=10.0",log10(x));
    printf("\nceil\(a\) = %4.2f when a=2.718282",ceil(a));
    printf("\nceil\(b\) = %4.2f when b=-2.718282",ceil(b));
    printf("\nfloor\(a\) = %4.2f when a=2.718282",floor(a));
    printf("\nfloor\(b\) = %4.2f when b=-2.718282",floor(b));
    printf("\nfabs\(a\) = %4.6f when a=2.718282" ,fabs(a));
    printf("\nfabs\(b\) = %4.6f when b=-2.718282" ,fabs(b));
}
```

ฟังก์ชันมาตรฐานทางคณิตศาสตร์



ฟังก์ชันที่จัดการเกี่ยวกับตัวอักษร(character functions) #include <ctype.h>

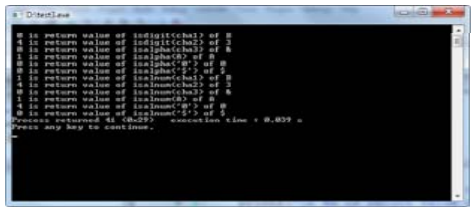
ชื่อฟังก์ชัน	หน้าที่
isalnum(cha)	ตรวจสอบว่าข้อมูลในตัวแปร เป็นตัวอักขระหรือตัวเลขหรือไม่
isalpha(cha)	ตรวจสอบว่าข้อมูลในตัวแปร เป็นตัวอักขระหรือไม่
isdigit(cha)	ตรวจสอบว่าข้อมูลในตัวแปรฟังก์ชัน เป็นตัวเลขหรือไม่
islower(cha)	ตรวจสอบว่าตัวอักขระในตัวแปร cha เป็นตัวพิมพ์เล็กหรือไม่
isupper(cha)	ตรวจสอบว่าตัวอักขระในตัวแปร cha เป็นตัวพิมพ์ใหญ่หรือไม่
tolower(cha)	เปลี่ยนตัวพิมพ์ใหญ่ที่เก็บอยู่ในตัวแปรให้เป็นตัวพิมพ์เล็ก
toupper(cha)	เปลี่ยนตัวพิมพ์เล็กที่เก็บอยู่ในตัวแปรให้เป็นตัวพิมพ์ใหญ่
isspace(cha)	ตรวจสอบว่าข้อมูลในตัวแปร cha เป็น whitespace หรือไม่ whitespace ได้แก่ space ,tab ,vertical tab ,formfeed ,carriage retun ,newline
isxdigit(cha)	ตรวจสอบว่าข้อมูลในตัวแปร cha เป็น เลขฐานสิบหก (คือ 0-9 , A-F , a – f) หรือไม่

ฟังก์ชันที่จัดการเกี่ยวกับตัวอักษร(character functions)

ชื่อฟังก์ชัน	หน้าที่
gotoxy(x,y);	อยู่ใน conio.h ใช้สั่งให้เคอร์เซอร์เคลื่อนที่ไปตามตำแหน่งที่ระบุ โดย x คือ ตำแหน่งของสดมภ์บนจอภาพ ส่วน y คือตำแหน่งแถวบนจอภาพนับจากบนลงล่าง
clreol();	เป็นฟังก์ชันอยู่ใน conio.h ใช้ลบข้อความตั้งแต่ตำแหน่งที่เคอร์เซอร์อยู่ไปจนจบบรรทัด
delline();	อยู่ในconio.h ใช้ลบข้อความทั้งบรรทัดที่เคอร์เซอร์อยู่ไปจนจบบรรทัดและเลื่อนข้อความในบรรทัดล่างขึ้นมา
insline();	เป็นฟังก์ชันอยู่ในconio.h ใช้แทรกบรรทัดว่าง 1 บรรทัดใต้บรรทัดที่เคอร์เซอร์อยู่
system(“dos command”);	เป็นฟังก์ชันอยู่ในstdlib.h ใช้เรียกคำสั่งของ dos ขึ้นมาทำงาน เช่นคำสั่ง cls dir date time
abort();	ฟังก์ชันที่อยู่ใน <stdlib.h> ใช้ ยกเลิกการทำงานของโปรแกรมทันทีไม่ว่าจะทำงานสำเร็จหรือไม่ และมีข้อความ Abnomal program termination แสดงทางจอภาพ
abs(x) ;	ฟังก์ชันที่อยู่ใน <stdlib.h> ใช้หาค่าสัมบูรณ์ของ x โดย x ต้องเป็นจำนวนเต็ม
labs(x);	ฟังก์ชันที่อยู่ใน <stdlib.h> ใช้หาค่าสัมบูรณ์ของ x โดย x ต้องเป็นlong int
atoi(s);	ฟังก์ชันที่อยู่ใน <stdlib.h> ใช้เปลี่ยนข้อความให้เป็นเลขจำนวนเต็ม
atol(s);	ฟังก์ชันที่อยู่ใน <stdlib.h> ใช้เปลี่ยนข้อความให้เป็น long integer
atof(s);	ฟังก์ชันที่อยู่ใน <stdlib.h> ใช้เปลี่ยนข้อความให้เป็น floating point

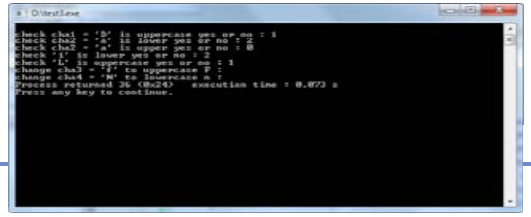
ฟังก์ชันที่จัดการเกี่ยวกับตัวอักษร(character functions)

```
#include <stdio.h>
#include <ctype.h>
main()
{
    char cha1 = 'B' ,cha2 ='3',cha3= '&';
    printf("\n %d is return value of isdigit\(cha1\) of %c",isdigit(cha1),cha1);
    printf("\n %d is return value of isdigit\(cha2\) of %c ",isdigit(cha2),cha2);
    printf("\n %d is return value of isalpha\(cha3\) of %c ",isalpha(cha3),cha3);
    printf("\n %d is return value of isalpha\(A\) of %c ",isalpha('A'),'A');
    printf("\n %d is return value of isalpha\('0'\) of %c ",isalpha('0'),'0');
    printf("\n %d is return value of isalpha\('$'\) of %c ",isalpha('$'),'$');
    printf("\n %d is return value of isalnum\(cha1\) of %c",isalnum(cha1),cha1);
    printf("\n %d is return value of isalnum\(cha2\) of %c ",isalnum(cha2),cha2);
    printf("\n %d is return value of isalnum\(cha3\) of %c ",isalnum(cha3),cha3);
    printf("\n %d is return value of isalnum\(A\) of %c ",isalnum('A'),'A');
    printf("\n %d is return value of isalnum\('0'\) of %c ",isalnum('0'),'0');
    printf("\n %d is return value of isalnum\('$'\) of %c ",isalnum('$'),'$');
}
```



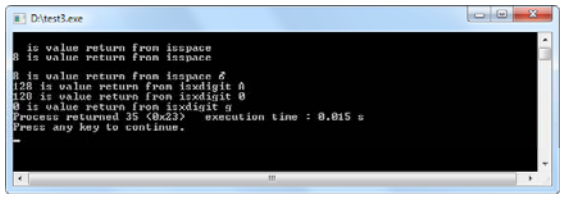
ฟังก์ชันที่จัดการเกี่ยวกับตัวอักษร(character functions)

```
#include <stdio.h>
#include <ctype.h>
main()
{
    char cha1 = 'D' , cha2 = 'a' ,cha3 = 'f' , cha4 = 'N' ;
    printf("\ncheck cha1 = 'D' is uppercase yes or no : %d ",isupper(cha1));
    printf("\ncheck cha2 = 'a' is lower yes or no : %d ",islower(cha2));
    printf("\ncheck cha2 = 'a' is upper yes or no : %d ",isupper(cha2));
    printf("\ncheck 'i' is lower yes or no : %d ",islower('i'));
    printf("\ncheck 'L' is uppercase yes or no : %d ",isupper('L'));
    printf("\nchange cha3 = 'f' to uppercase %c : ", toupper(cha3));
    printf("\nchange cha4 = 'N' to lowercase %c : ", tolower(cha4));
}
```



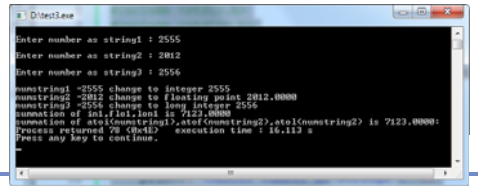
ฟังก์ชันที่จัดการเกี่ยวกับตัวอักษร(character functions)

```
#include <stdio.h>
#include <ctype.h>
main()
{
    char cha1 ='\r',cha2= '\n',cha3='\v' ,cha4 ='A';
    printf("\n%d is value return from isspace %c ",isspace(cha1),cha1);
    printf("\n%d is value return from isspace %c ",isspace(cha2),cha2);
    printf("\n%d is value return from isspace %c ",isspace(cha3),cha3);
    printf("\n%d is value return from isxdigit %c ",isxdigit(cha4),cha4);
    printf("\n%d is value return from isxdigit %c ",isxdigit('0'),'0');
    printf("\n%d is value return from isxdigit %c ",isxdigit('g'),'g');

}
```



ฟังก์ชันที่จัดการเกี่ยวกับตัวอักษร(character functions)

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    char numstring1[10],numstring2[10],numstring3[10];
    int in1;
    float flo1;
    long lon1;
    printf("\nEnter number as string1 : ");
    scanf("%s",numstring1);
    printf("\nEnter number as string2 : ");
    scanf("%s",numstring2);
    printf("\nEnter number as string3 : ");
    scanf("%s",numstring3);
    in1 = atoi(numstring1); flo1 = atof(numstring2); lon1=atol(numstring3);
    printf("\nnumstring1 =%s change to integer %d ",numstring1,in1);
    printf("\nnumstring2 =%s change to floating point %4.4f ",numstring2,flo1);
    printf("\nnumstring3 =%s change to long integer %d ",numstring3,lon1);
    printf("\nsummation of in1,flo1,lon1 is %6.4f ",in1+flo1+lon1);
    printf("\nsummation of atoi(numstring1),atof(numstring2),atol(numstring2) is
%6.4lf:",atoi(numstring1)+atof(numstring2)+atol(numstring3));
}
```



การใช้งานฟังก์ชันมาตรฐานใน C

โปรแกรมหาค่ารากที่สอง โดยใช้ฟังก์ชัน sqrt() ใน math.h

ต้นแบบ(prototype) : **double sqrt(double num)**

ให้ค่า $\sqrt{num}$

```
#include <stdio.h>
#include <math.h>
int main()
{       printf("%.2f\n", sqrt(16.0));
        printf("%.2f\n", sqrt(sqrt(16.0)));


}
```

```
4.00
2.00
```

การใช้งานฟังก์ชันมาตรฐานใน C

โปรแกรมแสดง 10 ยกกำลัง 1 ถึง 5 โดยเรียกใช้ฟังก์ชัน pow() ใน math.h

ต้นแบบ double pow(double base, double exp);

ให้ค่า $base^{exp}$

```
#include <stdio.h>
#include <math.h>
int main()
{
    double x=10.0, y =1.0;

    do {
        printf("%.2f\n", pow(x,y));
        y++;
    } while (y < 6);
}
```

```
10.00
100.00
1000.00
10000.00
100000.00
```

หารากที่ 2 => ยกกำลัง 0.5

หารากที่ 5 => ยกกำลัง ?

สร้างฟังก์ชันด้วยตนเอง

การสร้างฟังก์ชันมี 2 ส่วนคือ

1. **ต้นแบบฟังก์ชัน (function prototype)**
2. **นิยามฟังก์ชัน (function definition)**

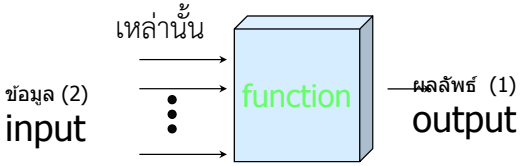
ต้นแบบฟังก์ชันจะมีหรือไม่ ขึ้นกับตำแหน่งที่เรานิยามฟังก์ชันในโปรแกรม

- ถ้าฟังก์ชันนิยามก่อน main (in-line function) ไม่จำเป็นต้องมีต้นแบบฟังก์ชัน
- ถ้าฟังก์ชันนิยามหลัง main จำเป็นต้องมีการประกาศต้นแบบฟังก์ชันก่อนฟังก์ชัน main

Function prototype

ต้นแบบของฟังก์ชันเป็นตัวบอกให้คอมไพเลอร์รู้ถึง:

1. ชนิดข้อมูลที่จะส่งค่ากลับ (1)
2. จำนวนพารามิเตอร์ที่ฟังก์ชันต้องการ (2)
3. ชนิดของพารามิเตอร์แต่ละตัว รวมทั้งลำดับของพารามิเตอร์เหล่านั้น



Note เป็นไปได้ที่ฟังก์ชันจะไม่มีการส่งค่ากลับ และ ไม่ต้องการข้อมูล input

Function prototype

- รูปแบบ

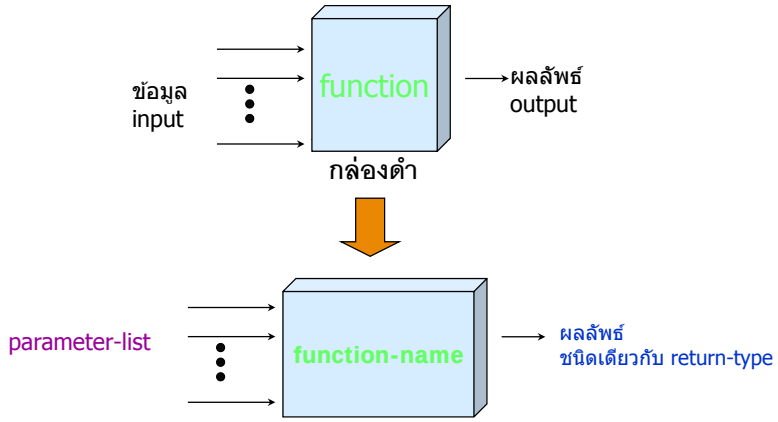
return-type function-name (parameter-list) ;

- return-type หรือชนิดข้อมูลที่จะส่งค่ากลับได้แก่ void, int, double, char, ...
- **function-name** ชื่อของฟังก์ชัน
- parameter-list จำนวนพารามิเตอร์ที่ฟังก์ชันต้องการ
  - แต่ละพารามิเตอร์ประกอบด้วย ชนิดตัวแปรและชื่อตัวแปร
  - แต่ละพารามิเตอร์แยกด้วยเครื่องหมาย “ , ”

ตัวอย่าง int **add** (int a, int b) **;**

Function prototype

- เทียบกับตอนเป็นกล่องดำ



นิยามฟังก์ชัน (function definition)

- นิยามฟังก์ชันเป็นการเขียนรายละเอียดการทำงานของฟังก์ชันนั้นๆ
- ประกอบด้วยส่วนของ header และ algorithm
  - header จะมีการเขียนเหมือน ต้นแบบฟังก์ชัน แต่ไม่มี ;
  - algorithm เขียนอยู่ภายใน { }
- รูปแบบ

```
return-type function-name (parameter-list)
{
   รายละเอียดการทำงาน
}
```

ถ้า return-type ไม่ใช่ void ต้องมีการส่งค่ากลับโดยใช้คำสั่ง return ตามด้วยค่าที่จะส่งกลับ

นิยามฟังก์ชัน (function definition)

- ตัวอย่างฟังก์ชันที่มี return

```
int add (int a, int b)
{
      int result;
      result = a+b;
      return result;
}
```

- ตัวอย่างฟังก์ชันที่ไม่มี return

```
void  checkresult (int result)
{
   if(result < 0)
         printf("result is negative");
   if(result == 0)
         printf("result is zero");
   if(result > 0)
         printf("result is positive")
}
```


ชื่อต่างๆเกี่ยวกับฟังก์ชัน

```
#include <stdio.h>
int square(int x);
int main()
{
  int a=2,b;
  b = square(a);
  printf("b = %d \n",b);
  return 0;
}
int square(int x)
{
  int y;
  y = x*x;
  return y;
}
```

รูปแบบของฟังก์ชัน

int , char , float , double ฯลฯ

```
ชนิดข้อมูลที่คืนค่า ชื่อฟังก์ชัน ( การประกาศตัวแปร )
    {
        การประกาศตัวแปรภายในฟังก์ชัน;
        คำสั่ง;
        return  (ค่าข้อมูลที่ต้องการส่งค่ากลับ);
    }
```

รูปแบบของฟังก์ชัน

```
void   ชื่อฟังก์ชัน ( การประกาศตัวแปร )
    {
        การประกาศตัวแปรภายในฟังก์ชัน;
        คำสั่ง;
    }
```

ตัวอย่างการสร้างฟังก์ชัน

- โปรแกรมเพื่อบวกเลขสองจำนวนที่รับจากผู้ใช้ และแสดงผลการคำนวณ

สามารถแบ่งการทำงานเป็นงานย่อยได้ดังนี้

1.รับข้อมูล 2 จำนวนจากผู้ใช้

2.บวกเลข 2 จำนวนแล้วเก็บผลลัพธ์

3.แสดงผลลัพธ์ของการทำงาน

ตัวอย่างการสร้างฟังก์ชัน

จะได้ว่าโปรแกรมประกอบด้วยฟังก์ชัน 4 ฟังก์ชันคือ

ฟังก์ชันหลัก

ฟังก์ชันการรับข้อมูล

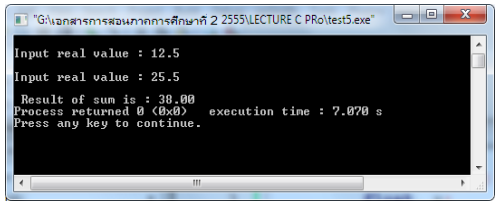
ฟังก์ชันในการบวกเลข

ฟังก์ชันแสดงผลลัพธ์

ตัวอย่างการสร้างฟังก์ชัน

แสดงการทำงานของโปรแกรมการบวกเลขจำนวนจริง 2 จำนวนที่รับจากผู้ใช้

```
#include <stdio.h>
float  InputDouble ( ) {
        float  x;
        printf ( "\nInput real value : " );
        scanf ( "%f", &x );
        return  x ;
}
float  SumDouble ( float  x,  float  y ) {
        return  x + y ;
}

void  PrintOut ( float  x ) {
        printf ( "\n Result of sum is : %.2f",  x );
}
int  main (  ) {
        float   a1, a2, sumVal;
         a1  =  InputDouble( );
         fflush(stdin);
        a2  =  InputDouble( );
        sumVal  =  SumDouble ( a1, a2 );
        PrintOut ( sumVal );
        return 0;
}
```



การเรียกใช้ฟังก์ชัน

การเรียกใช้ฟังก์ชันที่มีการคืนค่า จะใช้รูปแบบดังต่อไปนี้

ค่าที่รับ = ฟังก์ชัน (อาร์กิวเมนต์)

ค่าที่ถูกคืนมาจากการทำงานของฟังก์ชั่น

เรียกใช้งานฟังก์ชันได้อย่างไร

BOSS อาจเป็นโปรแกรม main หรือเป็นฟังก์ชันอื่น

WORKER คือฟังก์ชันที่ถูกเรียกใช้งาน

ฟังก์ชันแบ่งตามการรับส่งข้อมูล

แบบที่ 1 - ฟังก์ชันที่ไม่รับผ่านค่า และไม่ส่งผ่านค่ากลับ

```
void func1();
```

แบบที่ 2 - ฟังก์ชันที่มีการรับผ่านค่า แต่ไม่ส่งผ่านค่ากลับ

```
void func2(int a);
```

แบบที่ 3 - ฟังก์ชันที่มีการรับผ่านค่า และส่งผ่านค่ากลับ

```
int func3(int a);
```

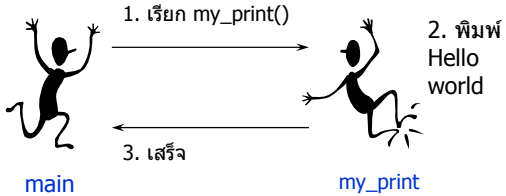
แบบที่ 4 - ฟังก์ชันที่ไม่รับผ่านค่า แต่มีการส่งผ่านค่ากลับ

```
int func4();
```

ฟังก์ชันแบ่งตามการรับส่งข้อมูล

- ฟังก์ชันที่ไม่รับผ่านค่า และไม่ส่งผ่านค่ากลับ

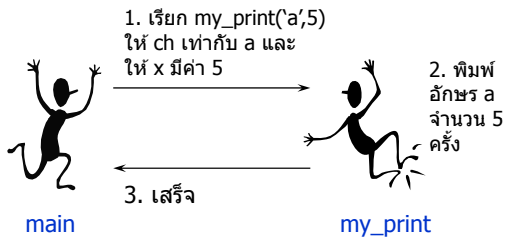
```
#include <stdio.h>
void my_print();

void main()
{
   my_print();
}

void my_print()
{
   printf("Hello world");
}
```



ฟังก์ชันแบ่งตามการรับส่งข้อมูล

- ฟังก์ชันที่มีการรับผ่านค่า แต่ไม่ส่งผ่านค่ากลับ

```
#include <stdio.h>
void my_print(char ch, int x);

void main()
{
   my_print('a', 5);
}

void my_print(char ch, int x)
{
   while (x > 0)
   {
      printf("%c", ch);
      x--;
   }
}
```

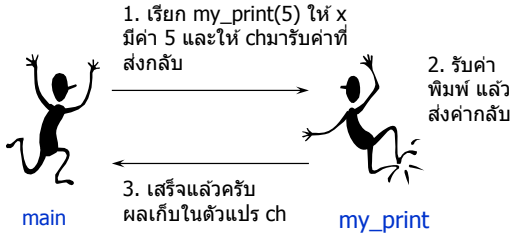


Note :
ฟังก์ชันต้องการพารามิเตอร์ 2 ตัว ตอนเรียกใช้งานต้องให้อากิวเมนต์ให้ถูกต้องตามจำนวนและชนิดที่ต้องการ

ฟังก์ชันแบ่งตามการรับส่งข้อมูล

- ฟังก์ชันที่มีการรับผ่านค่า และส่งผ่านค่ากลับ

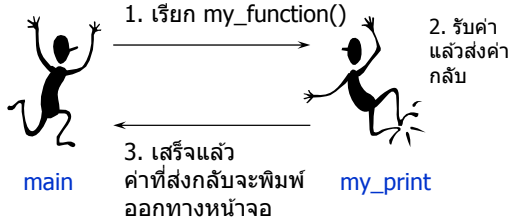
```
#include <stdio.h>
char my_print(int x);
void main()
{
   char ch;
   ch = my_print(5);
   printf("%c", ch);
}
char my_print(int x)
{
   char lch;
   scanf("%c", &lch);
   while (x > 0)
   {
      printf("%c", lch);
      x--;
   }
  return lch;
}
```



ฟังก์ชันแบ่งตามการรับส่งข้อมูล

- ฟังก์ชันที่ไม่รับผ่านค่า แต่มีการส่งผ่านค่ากลับ

```
#include <stdio.h>
int my_function();
void main()
{
   printf("%d", my_function());
}
int my_function()
{
   int a;
   scanf("%d", &a);
   return a*5;
}
```



การส่งค่าพารามิเตอร์ (Parameter Passing)

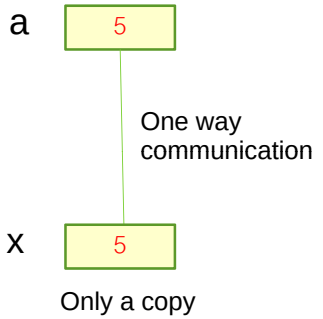
-การส่งโดยค่า Pass by Value

- เป็นการส่งค่าพารามิเตอร์จากโปรแกรมที่เรียกไปยังโปรแกรมย่อยเพียงด้านเดียว ไม่มีการส่งค่ากลับ
- ค่าที่ส่งไปยังจะถูกคัดลอกไปไว้ที่หน่วยความจำใหม่โปรแกรมย่อยใช้ทำงาน จึงไม่มีผลต่อตัวพารามิเตอร์ที่เรียกใช้

การส่งค่าพารามิเตอร์ (Parameter Passing)

-การส่งโดยค่า Pass by Value

```
#include <stdio.h>
void func(int x);
main()
{
 int a=5;
 func(a)
 printf("%d\n",a);
 return 0;
}

void func(int x)
{
   x=x+3;
   return;
}
```



การส่งค่าพารามิเตอร์ (Parameter Passing)

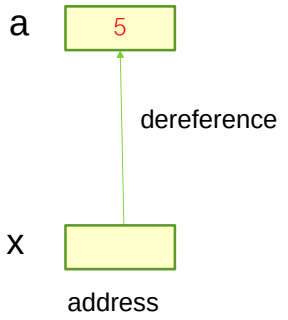
-การส่งโดยที่อยู่ของตัวแปร Pass by Reference

- เป็นการผ่านค่าที่อยู่ของตัวแปรในโปรแกรมหลักไปยังโปรแกรมย่อย
- เมื่อโปรแกรมย่อยทำงานจึงทำให้ค่าในโปรแกรมหลักเปลี่ยนแปลง

การส่งค่าพารามิเตอร์ (Parameter Passing)

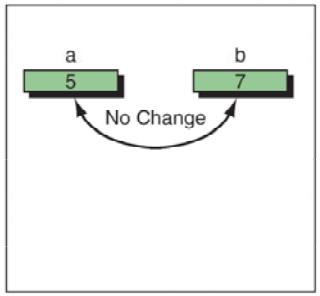
-การส่งโดยที่อยู่ของตัวแปร Pass by Reference

```
#include <stdio.h>
void func(int *x);
main()
{
 int a=5;
 func(&a)
 printf("%d\n",a);
 return 0;
}

void func(int *x)
{
   *x=*x+3;
   return;
}
```

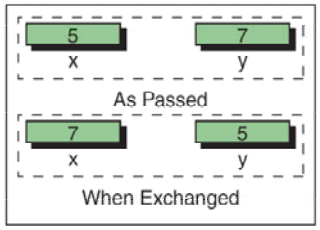


การส่งค่าพารามิเตอร์ (Parameter Passing)

```
// Function Declarations
void exchange (int x, int y);

int main (void)
{
  int a = 5;
  int b = 7;
  exchange (a, b);
  printf("%d %d\n", a, b);
  return 0;
} // main
```



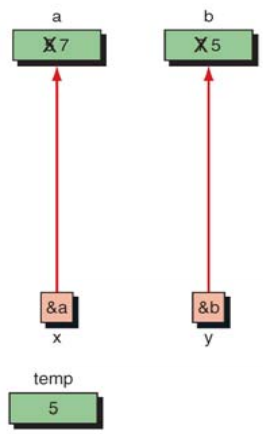

```
void exchange (int x, int y)
{
  int temp;

  temp = x;
  x    = y;
  y    = temp;
  return;
} // exchange
```



การส่งค่าพารามิเตอร์ (Parameter Passing)

```
// Function Declaration
void exchange (int*, int*);

int main (void)
{
  int a = 5;
  int b = 7;

  exchange (&a, &b);
  printf("%d %d\n", a, b);
  return 0;
} // main
```

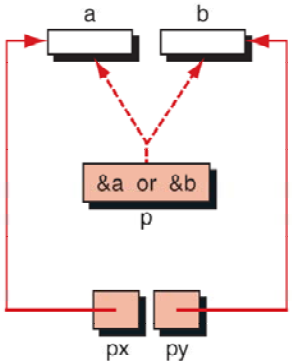
```
void exchange (int* px, int* py)
{
  int temp;

  temp = *px;
  *px  = *py;
  *py  = temp;
  return;
} // exchange
```



การส่งค่าพารามิเตอร์ (Parameter Passing)

```
// Prototype Declarations
int* smaller (int* p1, int* p2);

int main (void)
  …
  int  a;
  int  b;
  int* p;
  …
  scanf ( "%d %d", &a, &b );
  p = smaller (&a, &b);
  …
```

```
int* smaller (int* px, int* py)
{
  return (*px < *py ? px : py);
} // smaller
```



สรุปการเรียกใช้งานฟังก์ชัน

- ฟังก์ชันที่ไม่มีการส่งค่ากลับ สามารถเรียกชื่อฟังก์ชันได้เลย
- โดยต้องให้อากิวเมนต์ให้ถูกต้อง

  เช่น my_print(), my_print('a',5);

```
void my_print()
{
   printf("Hello world");
}
```

```
void my_print(char ch, int x)
{
   while (x > 0)
   {
      printf("%c", ch);
      x--;
   }
}
```

สรุปการเรียกใช้งานฟังก์ชัน

- ฟังก์ชันที่มีการส่งค่ากลับ ให้มองเสมือนว่าเป็นตัวแปรตัวหนึ่งที่มีชนิดเดียวกับชนิดของข้อมูลที่ส่งค่ากลับ วิธีเรียกใช้งาน
  - เอาตัวแปรมารับค่า เช่น ch = my_print(5);
  - ใช้เป็นส่วนหนึ่งในนิพจน์ เช่น printf("%d", my_function());
  - ใช้เป็นอาร์กิวเมนต์ของฟังก์ชันอื่น เช่น มีฟังก์ชัน int add(int a, int b);

    int x = add(add(1,2),4);

    คำนวณ add(1,2) ก่อน แล้วผลลัพธ์ที่ได้ ให้เป็นอาร์กิวเมนต์ของ add(... ,4)

โจทย์ตัวอย่าง

จงเขียนโปรโตไทป์ให้ฟังก์ชันชื่อ script ที่มีพารามิเตอร์ 3 ตัว

1 จำนวนช่องว่างที่ต้องการให้แสดงเมื่อเริ่มบรรทัด

2 ตัวอักษรที่ต้องการให้แสดงหลังจากช่องว่างในพารามิเตอร์แรก

3 จำนวนครั้งที่ต้องการให้แสดงตัวอักษรในพารามิเตอร์ที่ 2

โดยฟังก์ชัน script ไม่มีการส่งค่ากลับ

void script(int space, char c, int time);

โจทย์ตัวอย่าง

เขียนรายละเอียดการทำงานของฟังก์ชัน script ในข้อที่แล้ว ฟังก์ชัน script มีพารามิเตอร์ 3 ตัว

- 1 จำนวนช่องว่างที่ต้องการให้แสดงเมื่อเริ่มบรรทัด
- 2 ตัวอักษรที่ต้องการให้แสดงหลังจากช่องว่างในพารามิเตอร์แรก
- 3 จำนวนครั้งที่ต้องการให้แสดงตัวอักษรในพารามิเตอร์ที่ 2

โดยฟังก์ชัน script ไม่มีการส่งค่ากลับ

```
void script(int space, char c, int time)
{   int i;
    for(i=0; i< space; i++)
        printf(" ");
    for(i=0; i< time; i++)
        printf("%c", c);
}
```


ตัวอย่าง

เขียนโปรแกรมสำหรับหาค่าศักย์ไฟฟ้าซึ่งมีสมการดังนี้

V = I*R

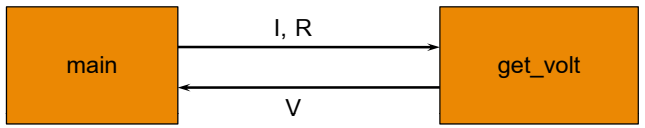
V คือ ค่าศักย์ไฟฟ้า , I คือ ค่ากระแสไฟฟ้า ส่วน R คือ ค่าความต้านทาน และทั้งสามค่านี้เป็นจำนวนจริง โดยกำหนดให้ส่วนที่คำนวณ

ค่า V อยู่ในฟังก์ชัน get_volt

ค่า I และ R รับจากผู้ใช้

ส่วนที่แสดงผลลัพธ์ของค่า V ให้อยู่ในฟังก์ชัน main

```
#include <stdio.h>

float get_volt(float I,float R);
int main()
{   float i,r,v;
    printf("Enter  I  : ");
    scanf("%f", &i);
    printf("Enter  R  : ");
    scanf("%f", &r);
    v = get_volt(i,r);
    printf("Volt = %f \n",v);
    return 0;
}

float get_volt(float I,float R)
{
  float V = I*R;
  return V;
}
```



ตัวอย่าง

เขียนโปรแกรมสำหรับหาค่า F(x) ซึ่งมีสมการดังนี้

$F(x) = x^2 + 2x + 1$ ถ้า x มีค่าไม่ต่ำกว่า 0

$= 0$ ถ้า x มีค่าต่ำกว่า 0

กำหนดให้ส่วนที่ใช้ในการคำนวณค่า F(x) อยู่ใน ฟังก์ชัน get_fx

ค่า x รับจากผู้ใช้ แสดงผลลัพธ์ของค่า F(x) อยู่ใน ฟังก์ชัน main (x และ F(x) เป็นจำนวนเต็ม)

```c
#include <stdio.h>

int get_fx(int a);

int main()
{
    int x,fx;
    printf("Enter  x  : ");
    scanf("%d", &x);
    fx = get_fx(x);
    printf("f(%d) = %d \n",x,fx);
    return 0;
}
int get_fx(int a)
{
    if(a>=0)
        return (a*a)+(2*a)+1;
    else
        return 0;
}
```

ฟังก์ชันแบบเรียกตัวเอง (recursive function)

- ฟังก์ชันแบบเรียกตัวเองคือ ฟังก์ชันที่มีการเรียกตัวเองโดยให้พารามิเตอร์ที่แตกต่างกันออกไปเช่น การหา Factorial หรือการหา Fibonacci

ข้อควรระวัง :
ฟังก์ชันแบบเรียกตัวเอง จำเป็นจะต้องมี if statement เพื่อใช้ในการตัดสินใจว่าฟังก์ชันจะเรียกตัวเองต่อไป หรือ หยุดเพื่อส่งค่ากลับ

```c
#include<stdio.h>
int factorial(int x);
int main()
{
  int y = factorial(3);
  printf("3! = %d", y);
  return 0;
}
int factorial(int x)
{
  if(x <= 1)
    return 1;
  else
    return  x* factorial(x-1);
}
```


ตัวอย่าง

- จงเขียนฟังก์ชันสำหรับหาค่า F(x) ซึ่งมีสมการดังนี้

F(x) = 0 ถ้า x เท่ากับ 0

= 1 ถ้า x เท่ากับ 1

= F(x-1) + F(x-2) ถ้า x มากกว่า 1

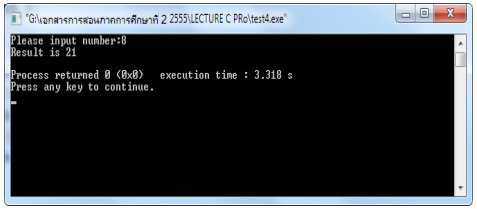
- กำหนดให้ x และ F(x) เป็นจำนวนเต็ม และให้ตั้งชื่อฟังก์ชันว่า fib

(อนุกรม Fibonacci)

- Fibonacci numbers: 1 1 2 3 5 8 13 21 34 55 89 144 233 ...

ฟังก์ชันแบบเรียกตัวเอง (recursive function)

```c
#include <stdio.h>
int fib(int x);
int main()
{
  int n, result=0;
  printf("Please input number:");
  scanf("%d",&n);
  result=fib(n);
  printf("Result is %d\n",result);
  return 0;
}
int fib(int x)
{
  if(x == 0)
    return 0;
  else if(x == 1)
    return 1;
  else if(x > 1)
    return fib(x-1) + fib(x-2);
}
```

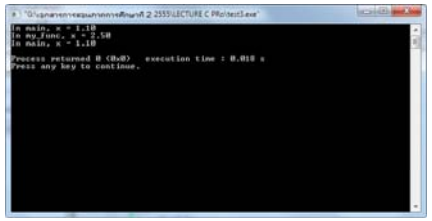


ขอบเขตของตัวแปรในเรื่องของฟังก์ชัน

- เมื่อมีการเรียกใช้งานฟังก์ชันจะมีการจองพื้นที่หน่วยความจำสำหรับตัวแปรที่ต้องใช้ภายในฟังก์ชันนั้น
- เมื่อสิ้นสุดการทำงานของฟังก์ชันก็จะมีการคืนพื้นที่หน่วยความจำส่วนนั้นกลับสู่ระบบ การใช้งานตัวแปรแต่ละตัวจะมีขอบเขตของการใช้งานขึ้นอยู่กับตำแหน่งที่ประกาศตัวแปรนั้น
  - ตัวแปรภายใน (local variable) – ตัวแปรที่ประกาศในบล็อค (ภายในเครื่องหมาย { }) ของฟังก์ชันใดๆ จะรู้จักเฉพาะในฟังก์ชันนั้นๆ
  - ตัวแปรภายนอก (global variable) - ตัวแปรที่ประกาศนอกฟังก์ชันใดๆ (รวมถึงฟังก์ชัน main) จะรู้จักในทุกฟังก์ชันที่เขียนถัดจากการประกาศตัวแปร

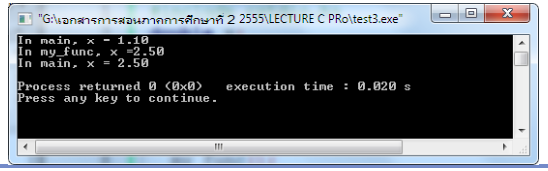
ขอบเขตของตัวแปรภายในฟังก์ชัน

```c
#include <stdio.h>
void my_func();
int main()
{
  double x = 1.1;
  printf("In main, x = %.2f \n",x);
  my_func();
  printf("In main, x = %.2f \n",x);
  return 0;
}
void my_func()
{
  double x;
  x = 2.5;
  printf("In my_func, x = %.2f \n",x);
}
```



ขอบเขตของตัวแปรภายนอกฟังก์ชัน

```c
#include <stdio.h>
double x;
void my_func();
int main()
{
  x = 1.1;
  printf("In main, x = %.2f \n",x);
  my_func();
  printf("In main, x = %.2f \n",x);
  return 0;
}
void my_func()
{
  x = 2.5;
  printf("In my_func, x =%.2f \n",x);
}
```



สรุปขอบเขตของตัวแปร

```c
#define MAX 950
void one(int anarg,
         double second)
{
  int onelocal;
  ...
}
#define LIMIT 200
int fun_two(int one, char anarg)
{   int localvar;
  ...
}

int main(void)
{   int localvar;
  ...
}
```

ตัวแปร	รู้จักใน one	รู้จักใน fun_two	รู้จักใน main
MAX	✔	✔	✔
anarg(int)	✔	✘	✘
second	✔	✘	✘
onelocal	✔	✘	✘
LIMIT	✘	✔	✔
one (parameter)	✘	✔	✘
anarg(char)	✘	✔	✘
localvar(fun_two)	✘	✔	✘
localvar(main)	✘	✘	✔

ฟังก์ชัน	รู้จักใน one	รู้จักใน fun_two	รู้จักใน main
one (function)	✔	✘	✔
fun_two	✘	✔	✔