Fundamental Programming I

Exercises v1.1

CONTENTS

[This document should be used in conjunction with PPS]

Exercise01/1 – C Review

My Power

จงเขียนโปรแกรมที่ทำหาค่าของเลขยกกำลัง

รูปแบบการแสดงผล

```
base: <5>
power: <2>
= 25
```

หมายเหตุ ให้คำนวณโดยการใช้ลูป ห้ามใช้ฟังก์ชันทางคณิตศาสตร์จากไลบรารีใดๆ

Code Template

```
//main.c
#include<stdio.h>
int main(){
    printf("base: ");
    printf("power: ");

    return 0;
}
```

Test Input

```
4
3
```

Test Output

```
base:power:=64
```

Number Counter

จงเขียนโปรแกรมรับค่าตัวเลขจากผู้ใช้ ต่อเนื่องไปจนกว่าผู้ใช้จะใส่เลข 0 โดยเมื่อหยุดรับข้อมูล โปรแกรมจะแสดงจำนวนของตัวเลขบวก และจำนวนของตัวเลขลบ โดยไม่ถือว่า 0 ที่ใส่ไว้ตัวสุดท้ายเป็นข้อมูล
โดยให้แสดงผลดังนี้

```
N01: <2>
N02: <-5>
N03: <7>
N04: <0>
Positive=2
Negative=1
```

Code Template

```
//main.c
#include<stdio.h>

int main(){
    printf("N%02d: ");
    printf("Positive=");
    printf("Negative=");

    return 0;
}
```

Test Input

```
5
6
9
-1
-7
-3
5
0
```

Test Output

```
N01:N02:N03:N04:N05:N06:N07:N08:Positive=4
Negative=3
```

High Low

จงเขียนโปรแกรมรับค่าตัวเลขจากผู้ใช้จำนวน 10 ตัว แล้วหาค่าสูงสุดและต่ำสุดของตัวเลขเหล่านั้น
รูปแบบการแสดงผลเป็นดังนี้

```
N01: <8>
N02: <10>
N03: <2>
...
N10: <7>
Max=15
Min=2
```

Code Template

```
//main.c
#include<stdio.h>

int main(){
    printf("N%02d: ");
    printf("Max=");
    printf("Min=");

    return 0;
}
```

Test Input

```
5
6
9
3
-2
10
55
3
1
0
```

Test Output

```
N01:N02:N03:N04:N05:N06:N07:N08:N09:N10:Max=55
Min=-2
```

Prime Number

รับตัวเลข 1 ตัว สมมติให้เป็นค่า x แล้วคำนวนว่า มีตัวเลขที่เป็นจำนวนเฉพาะอยู่กี่ตัว จาก 1 ถึง x

รูปแบบการแสดงผล

```
X: 15
=6
```

หมายเหตุ: 1 ไม่นับเป็นจำนวนเฉพาะ ตย.ข้างต้นจึงมีเลขจำนวนเฉพาะ 6 ตัว คือ 2 3 5 7 11 13

Code Template

```
//main.c
#include<stdio.h>

int main(){
    printf("X:");

    return 0;
}
```

Test Input

```
300
```

Test Output

```
X:=62
```

Leap Year

รับค่าพารามิเตอร์เป็นหมายเลขปี แล้วแสดงผลลัพธ์ออกมาว่าเป็นปีอธิกสุรทินหรือไม่
ปีอธิกสุรทินคือปีที่เดือนกุมภาพันธ์มี 29 วัน โดยเป็นปีที่หารด้วย 4 ลงตัว แต่หากปีนั้นลงท้ายด้วยหนึ่งร้อย ต้องหารด้วย 400 ลงตัว)

```
Enter: 2000
=Yes
```

หากไม่ใช่ให้แสดงว่า No
ทั้งนี้ การคำนวณว่าเป็นปีอธิกสุรทินหรือไม่ ให้เขียนฟังก์ชันโดยมีต้นแบบดังต่อไปนี้

```
int is_leap_year(int y);
```

คืนค่า 1 หากใช่ และ 0 หากไม่ใช่

Code Template

```
//main.c
#include<stdio.h>

int is_leap_year(int y);

int main(){

    return 0;
}
```

Test Input

```
3000
```

Test Output

```
Enter:=No
```

Number Components

รับตัวเลขจำนวนเต็มไม่เกิน 5 หลัก เพื่อกระจายตัวเลขตามหน่วย

รูปแบบการแสดงผล

```
Enter: 1503
=1000+500+3
```

โดยการกระจายตัวเลขให้ใช้ฟังก์ชัน

```
void display_components(int n);
```

Code Template

```
//main.c
#include<stdio.h>

void display_components(int n);

int main(){
    printf("Enter:");

    return 0;
}
```

Test Input

```
32010
Test Output
Enter:=30000+2000+10
...
```

Char Twist

รับตัวอักษรหนึ่งตัว หากเป็นตัวหนังสือ a-z หรือ A-Z ให้แสดงผลเป็นตัวหนังสือนั้น แต่จากตัวเล็กเป็นตัวใหญ่ จากตัวใหญ่เป็นตัวเล็ก โดยโปรแกรมจะวนรับค่า จนกว่าผู้ใช้จะใส่ตัวหนังสือที่ไม่ใช่ตัวหนังสือภาษาอังกฤษ

รูปแบบการแสดงผล

```
Char: <B>
=b
Char: <z>
=Z
Char: <*>
End.
```

หมายเหตุ การแปลงตัวหนังสือให้ใช้ฟังก์ชัน `char_twist` ซึ่งมีต้นแบบของฟังก์ชันดังนี้

```
char char_twist(char c);
```

Code Template

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

char char_twist(char c);

int main(){

    return 0;
}
```

Test Input

```
c
C
g
%
```

Test Output

```
Char:=C
Char:=c
Char:=G
Char:End.
```

Char Next Twist

เขียนโปรแกรม เพื่อรับตัวอักษรด้วยฟังก์ชัน `getchar()` จนกว่าจะเจอ `EOF` โดยแสดงผลตัวหนังสือที่รับเข้าไปตามเงื่อนไขนี้

- หากเป็นตัวหนังสือ ให้แสดงเป็นตัวหนังสือตัวถัดไป (ตัวถัดไปของ z คือ a)
- หากเป็นตัวหนังสือ ให้เปลี่ยนตัวเล็กเป็นตัวใหญ่ ตัวใหญ่เป็นตัวเล็ก
- หากเป็นอักขระอื่นๆ ให้แสดงตัวอักษรนั้นได้เลย

โดยโปรแกรมจะต้องใช้ฟังก์ชันข้างล่าง

```
char char_next(char c); //ตัวหนังสือตัวถัดไป

char char_twist(char c);  //ตัวหนังสือ case ตรงข้าม
```

และเรียกใช้ฟังก์ชันทั้งสองในลักษณะ cascade กล่าวคือ

```
char_twist(char_next(c));
```

หรือ

```
char_next(char_twist(c));
```

รูปแบบการแสดงผล

```
<hEllO *worLd>
IfMMp *XPSmE
<EOF>
```

หมายเหตุ `EOF`

บน Linux, Mac กด `Ctrl+d` ได้ทันที

บน Windows กด `Ctrl+z` ในบรรทัดว่างๆ แล้วกด Enter

Code Template

```
//main.c
#include<stdio.h>

char char_next(char c);
char char_twist(char c);

int main(){

    return 0;
}
```

Test Input

```
hEllO *worLd
zZc!
```

Test Output

```
IfMMp *XPSmE
AaD!
```

Upper Values

รับตัวเลขจำนวนเต็ม 10 ตัว แล้วแสดงผลเฉพาะตัวเลขที่มีค่าสูงกว่าค่าเฉลี่ย

รูปแบบการแสดงผล

```
N01: <7>
N02: <3>
...
N10: <6>
=7 9 10 6
```

(สมมติให้ ค่าเฉลี่ยของตัวเลขทั้งหมดคือ 5.5)

Code Template

```
//main.c
#include<stdio.h>

int main(){
    printf("N%02d: ");

    return 0;
}
```

Test Input

```
3
5
9
8
2
0
6
7
1
4
```

Test Output

```
N01: N02: N03: N04: N05: N06: N07: N08: N09: N10: =5 9 8 6 7
```

Reverse String

รับข้อความหนึ่งข้อความจากผู้ใช้ ขนาดไม่เกิน 127 ตัวอักษร แล้วให้แสดงข้อความย้อนหลัง

รูปแบบการแสดงผล

```
Enter: <Hello world>
=dlrow olleH
```

หมายเหตุ

** ให้เรียกใช้ฟังก์ชัน `strlen` เพื่อหาขนาดของข้อความได้

** การรับค่าจากผู้ใช้โดยใช้ `scanf` จะต้องระบุ format เป็น `%[^\n]s`

Code Template

```
//main.c
#include<stdio.h>

int main(){
    printf("Enter: ");

    return 0;
}
```

Test Input

```
The Definitive Guide to Grails, Second Edition
```

Test Output

```
Enter: =noitidE dnoceS ,sliarG ot ediuG evitinifeD ehT
```

Array Adding

รับข้อมูลตัวเลขจำนวนเต็ม 3 ชุด ชุดละ 5 ตัว แล้วเรียกใช้ฟังก์ชัน `array_adding` เพื่อบวกตัวเลขชุดที่ 2 และ 3 ไปในตัวเลขชุดที่ 1

รูปแบบการแสดงผล

```
S1-01: <5>
S1-02: <7>
...
S3-04: <2>
S3-05: <1>
= 9 15 7 12 10
```

โดยฟังก์ชัน `array_adding` มีต้นแบบดังนี้

```
void array_adding(int[] a, int[] b, int len);
```

** `a` คืออาเรย์หลัก

** `b` คืออาเรย์ที่จะถูกนำมาค่าสมาชิกไปบวกกับอาเรย์หลัก

** `len` จำนวนของสมาชิกที่ถูกประมวลผล

ข้อสังเกตุ หลังการเรียกใช้ฟังก์ชัน `a` เปลี่ยนแปลง แต่ `b` ยังคงเหมือนเดิม

ทั้งนี้การรับตัวเลขจากผู้ใช้ ให้ใช้ฟังก์ชัน `array_input` ที่กำหนดไว้ให้ อย่างเหมาะสม

Code Template

```
//main.c
#include<stdio.h>

void array_adding(int a[], int b[], int len);
void array_input(int a[], int len, int set);

int main(){

    return 0;
}
void array_input(int a[], int len, int set){
    int i;
    for(i = 0; i < len; i++){
        printf("S%d-%02d: ", set, i + 1);
        scanf("%d", &a[i]);
    }
}
```

Test Input

```
4
5
6
1
3
2
6
0
8
-1
3
5
12
7
6
```

Test Output

```
S1-01: S1-02: S1-03: S1-04: S1-05: S2-01: S2-02: S2-03: S2-04: S2-05:
S3-01: S3-02: S3-03: S3-04: S3-05: =9 16 18 16 8
```

Print Minus

รับค่าตัวเลขจำนวนเต็มจำนวน 6 ตัว ให้พิมพ์ค่าตัวเลขที่ติดลบทีละ 1 ตัว โดยการเรียกใช้ฟังก์ชัน `print_minus` ที่กำหนดให้

รูปแบบการแสดงผล

```
N01: <5>
N02: <4>
..
N06: <0>
=
-2
-3
```

หมายเหตุ ห้ามแก้ไขฟังก์ชัน `print_minus`

Code Template

```
//main.c
#include<stdio.h>

int print_minus(int a[], int len);

int main(){
    printf("N%02d: ");

    return 0;
}
int print_minus(int a[], int len){
    int i;
    for(i = 0; i < len; i++){
        if(a[i] < 0){
            printf("%d\n", a[i]);
            return i;
        }
    }
    return -1;
}
```

Test Input

```
5
-4
3
2
-1
0
```

Test Output

```
N01: N02: N03: N04: N05: N06: N07: N08: =
-4
-1
```

Matrix Determinant

สร้างอาเรย์สองมิติขนาด 3x3 เพื่อแทนเมตริกซ์ขนาด 3x3 โดยให้ค่าสมาชิกทุกตัวเป็น 0 แล้วรับข้อมูลสมาชิกในอาเรย์จากผู้ใช้ โดยผู้ใช้สามารถกำหนดค่าของสมาชิกของเมตริกซ์ได้โดยการใส่ตัวเลข 3 ตัว ต่อการรับค่าหนึ่งครั้ง คือ แถว หลัก ข้อมูล ตามลำดับ

รับค่าข้อมูลจนกว่าผู้ใช้จะใส่ข้อมูล แถว เป็น -1 เมื่อเสร็จสิ้นการรับค่ามูล ให้หาค่า determinant ของเมตริกซ์ดังกล่าว โดยหาก A มีสมาชิกเป็น

```
a b c
d e f
g h i
```

```
det (A) = aei + bfg + cdh – afh – bdi – ceg
```

ทั้งนี้การหาค่า determinate ให้กระทำผ่านฟังก์ชัน

```
int detA(int a[][3]);
```

ตัวอย่างการแสดงผล

```
Enter: <2 1 7>
Enter: <1 0 8>
...
Enter: <-1>
=12
```

Code Template

```
//main.c
#include<stdio.h>

int detA(int a[][3]);

int main(){

    return 0;
}
```

Test Input

```
0 0 -2
0 1 2
0 2 -3
2 0 2
1 0 -1
1 1 1
2 2 -1
1 2 3
-1
```

Test Output

```
Enter: Enter: Enter: Enter: Enter: Enter: Enter: Enter: Enter: =18
```

Exercise02 – Intro to Pointers

Pointer-101

รับตัวเลขหนึ่งตัวจากผู้ใช้ หาค่าเลขยกกำลังสองของตัวเลขดังกล่าว โดยใช้ฟังก์ชัน `sq` อย่างเหมาะสม

`int sq(int *data);`

รูปแบบการแสดงผล

```
Enter: <3>
= 9
```

Code Template

```
//main.c
#include<stdio.h>

int sq(int*);

int main(){
  int a;
  printf("Enter: ");

  printf("=");
  return 0;
}
```

Test Input

```
4
```

Test Output

```
Enter:=16
...
```

Sum with Pointer

รับตัวเลขจากผู้ใช้ จนกว่าผู้ใช้จะใส่ตัวเลข 0 นำตัวเลขทั้งหมดมาหาค่าผลรวม โดยใช้ฟังก์ชัน `psum` อย่างเหมาะสม

`void psum(int* m, int* data);`

โดยฟังก์ชัน `psum` จะนำค่าที่ `data` ชี้อยู่ มาบวกลงในค่าที่ m ชี้อยู่

รูปแบบการแสดงผล

```
N: <3>
N: <7>
N: <6>
N: <0>
= 16
```

Code Template

```
//main.c
#include<stdio.h>

void psum(int* m, int* data);

int main(){
  printf("N:");


  printf("= ");
  return 0;
}
Test Input
7
2
5
3
0
```

Test Output

```
N:N:N:N:N:=17
```

Max with Pointer

รับตัวเลขจำนวน 5 ตัว แล้วพิมพ์ค่าตัวเลขที่มีค่ามากที่สุด ทั้งนี้ในการเปรียบเทียบตัวเลข จะต้องใช้ฟังก์ชัน `max` แทนการเปรียบเทียบตัวเลข

```
int* max(int* a, int *b);
```

ฟังก์ชัน จะคืนค่าพอยเตอร์ที่ชี้ไปยังข้อมูลที่มีค่ามากกว่า

หมายเหตุ

1. ห้ามเก็บตัวเลขไว้ในอาเรย์หรือเทียบเท่า
2. ให้ตั้งสมมติฐานว่า ผู้ใช้จะไม่ใส่ตัวเลขติดลบ (ค่าที่น้อยที่สุดคือ 0)

รูปแบบการแสดงผล

```
N1: <2>
N2: <7>
N3: <8>
N4: <0>
N5: <1>
=8
```

Code Template

```
//main.c
#include<stdio.h>
#define NUM 5
int* max(int* a, int *b);

int main(){
  int i;

  for(i = 0; i < NUM; i++){
    printf("N%d: ", i+1);
  }

  printf("=");
  return 0;
}

int* max(int* a, int *b){

}
```

Test Input

```
1
3
9
6
4
```

Test Output

```
N1:N2:N3:N4:N5:=9
```

Find Max with Pointer

รับค่าจำนวนเต็ม จำนวน 5 ตัว เก็บไว้ในอาเรย์ แล้วใช้ฟังก์ชัน `find_max` ในการค้นหาค่าที่มีค่ามากที่สุด
ทั้งนี้ฟังก์ชัน `find_max` มีต้นแบบของฟังก์ชันดังนี้

```
int* find_max(int *p, int len);
```

=> `p` จุดเริ่มต้นของอาเรย์

=> `len` จำนวนสมาชิกของอาเรย์ p

=> คืนค่า เป็น พอยเตอร์อ้างอิงไปยังสมาชิกที่มีค่ามากที่สุด

**ห้ามใช้เครื่องหมาย [] ในฟังก์ชัน `find_max`

รูปแบบการแสดงผล

```
N01: <3>
N02: <8>
N03: <0>
N04: <9>
N05: <1>
= 9
```

Code Template

```
//main.c
#include<stdio.h>
#define LEN 5
int* find_max(int *p, int len);
int main(){
    int d[LEN], i;
    int *max_ptr;
    for(i = 0; i < LEN; i++){
        printf("N%02d: ", i + 1);
        scanf("%d", &d[i]);
    }

    //use find max appropriately
    printf("=%d\n", *max_ptr);
    return 0;
}

//define find max here!
int* find_max(int *p, int len){

}
```

Test Input

```
5
6
2
1
8
```

Test Output

```
N01: N02: N03: N04: N05: = 8
```

Scoped Find Max

รับค่าจำนวนเต็ม จำนวน 10 ตัว เก็บไว้ในอาเรย์ หลังจากนั้น รับค่าจำนวนเต็ม 2 ตัว เพื่อกำหนดช่วงของข้อมูลที่ต้องการหาค่ามากที่สุด โดยให้ช่วงข้อมูลเป็นค่า 0-9

แล้วใช้ฟังก์ชัน `find_max` ในการค้นหาค่าที่มีค่ามากที่สุดของช่วงที่กำหนด

ทั้งนี้ฟังก์ชัน `find_max` มีต้นแบบของฟังก์ชันดังนี้

```
int* find_max(int *p, int *q);
```

=> `p` จุดเริ่มต้นของอาเรย์

=> `q` จุดสิ้นสุดของอาเรย์

=> คืนค่า เป็น พอยเตอร์อ้างอิงไปยังสมาชิกที่มีค่ามากที่สุด

**ห้ามใช้เครื่องหมาย [] ในฟังก์ชัน `find_max`

รูปแบบการแสดงผล

```
N01: <10>
N02: <8>
N03: <0>
N04: <7>
...
N09: <1>
N10: <4>
Scope: <4 8>
= 9
```

ข้อสังเกตุ แม้ว่า 10 จะมีค่ามากที่สุดในข้อมูลตัวอย่าง แต่คำตอบคือ 9 เพราะ 10 ไม่อยู่ในขอบเขต 4-8

Code Template

```
//main.c
#include<stdio.h>
#define LEN 10

int* find_max(int *p, int *q);

int main(){
    int d[LEN], i, m, n;
    int *max_ptr;

    for(i = 0; i < LEN; i++){
        printf("N%02d: ", i + 1);
        scanf("%d", &d[i]);
    }
    printf("Scope: ");
    scanf("%d %d", &m, &n);

    //use find max appropriately


    printf("=%d\n", *max_ptr);
    return 0;
}

//define find max here!
int* find_max(int *p, int *q){

}
```

Test Input

```
1
12
5
6
9
5
3
10
5
0
3 9
```

Test Output

```
N01:N02:N03:N04:N05:N06:N07:N08:N09:N10:Scope:=10
...
```

Reset Zero with Pointer

รับตัวเลขจำนวนเต็มจำนวน 10 ตัว เก็บไว้ในอาเรย์ หลังจากนั้นใช้ฟังก์ชัน `find_neg` เพื่อหาตำแหน่งของตัวเลขที่ติดลบ แล้วเปลี่ยนค่าตัวเลขติดลบให้เป็น 0

ต้นแบบของฟังก์ชัน `find_neg`

```
int* find_neg(int *p, int *q);
```

=> `p` จุดเริ่มต้นของอาเรย์

=> `q` จุดสิ้นสุดของอาเรย์

=> คืนค่า พอยเตอร์อ้างอิงไปยังตัวเลขติดลบ ตัวแรกที่เจอ แต่จะคืนค่าเป็น NULL หากไม่มีตัวเลขเหลืออยู่ หรือ `p >= q`

โดยห้ามใช้เครื่องหมาย [] ในฟังก์ชัน `find_neg`

รูปแบบการแสดงผล

```
N01: <3>
N02: <-1>
N03: <-2>
...
N10: <8>
= 3 0 0 2 1 0 7 9 0 8
```

Code Template

```
//main.c
#include<stdio.h>
#define LEN 10
int* find_neg(int *p, int *q);

int main(){
    int d[LEN], i;

    for(i = 0; i < LEN; i++){
        printf("N%02d: ", i + 1);
        scanf("%d", &d[i]);
    }

    //use find_neg appropriately

    printf("= ");
    for(i = 0; i < LEN; i++){
        printf("%d ", d[i]);
    }
    printf("\n");
    return 0;
}


int* find_neg(int *p, int *q){

}
```

Test Input

```
4
-1
0
2
3
-2
7
-3
9
1
```

Test Output

```
N01:N02:N03:N04:N05:N06:N07:N08:N09:N10:=4 0 0 2 3 0 7 0 9 1
```

Line Distance

รับข้อมูลจุดพิกัด `x,y` 2 จุดจากผู้ใช้ เพื่อคำนวณหาระยะห่างระหว่างจุดสองจุด
โดยกำหนดให้มีการนิยามและเรียกใช้ฟังก์ชันข้างล่างอย่างเหมาะสม

```
float line_distance(Point p1, Point p2);
```

** หาค่าระยะห่างระหว่างจุดสองจุด

```
Point point_input();
```

** รับค่าจุดจากผู้ใช้ (ไม่มีการ `printf` ในฟังก์ชันนี้)

รูปแบบการแสดงผล

```
P1: <3 6>
P2: <0 0>
=6.71
```

Code Template

```
//main.c
#include<stdio.h>

struct point{
    int x;
    int y;
};
typedef struct point Point;

Point point_input();
float line_distance(Point p1, Point p2);

int main(){
    printf("P1: ");
    printf("P2: ");
    printf("=%.2f");
    return 0;
}
```

Test Input

```
5 7
1 4
```

Test Output

```
P1: P2: =5.00
```

Employee Salary

รับข้อมูลของพนักงานจำนวน 5 คน โดยข้อมูลของพนักงานประกอบด้วย ชื่อ และ เงินเดือน
เมื่อรับข้อมูลเสร็จสิ้นแล้ว ให้แสดงชื่อของผู้ได้รับเงินเดือนสูงสุด และแสดงเงินเดือนรวมของพนักงานทั้งบริษัท
รูปแบบการแสดงผล

```
E01: <John Doe/3000>
...
E05: <Marry D/4000>
Max=Marry D/4000
Total=16000
```

โดยมีการใช้ฟังก์ชันต่อไปนี้อย่างเหมาะสม

```
Employee input_employee();
```

**รับค่าข้อมูลของพนักงาน (ไม่มีการ `printf`)

```
int find_max(Employee e[], int len);
```

**หา `index` ของพนักงานที่มีเงินเดือนสูงสุด

```
int find_total(Employee e[], int len);
```

**หาเงินเดือนรวมของพนักงานทั้งบริษัท

Code Template

```
//main.c
#include<stdio.h>

typedef struct {
    char name[128];
    int salary;
} Employee;

Employee input_employee();
int find_max(Employee e[], int len);
int find_total(Employee e[], int len);

int main(){
    printf("Max=");
    printf("Total=");
    return 0;
}

Employee input_employee(){
    Employee e;
    char delim;
    scanf(" %[^/]s", e.name);
    scanf("%c%d", &delim, &e.salary);
    return e;
}
```

Test Input

```
John Doe/3000
Kent J/9000
Marry D/4000
Jenny K/2000
Farm S/1000
```

Test Output

```
E01: E02: E03: E04: E05: Max=Kent J/9000
Total=19000
```

Employee Salary Dynamic

รับข้อมูลของพนักงานจำนวน 5 คน โดยข้อมูลของพนักงานประกอบด้วย ชื่อ และ เงินเดือน
เมื่อรับข้อมูลเสร็จสิ้นแล้ว ให้แสดงเงินเดือนรวมของพนักงานทั้งบริษัท

รูปแบบการแสดงผล

```
E01: <John Doe/3000>
...
E05: <Marry D/4000>
Total=16000
```

โดยมีการใช้ฟังก์ชันต่อไปนี้อย่างเหมาะสม

```
Employee* input_employee();
```

**รับค่าข้อมูลของพนักงาน (ไม่มีการ `printf`) มีการใช้ `malloc` เพื่อจองหน่วยความจำสำหรับเก็บข้อมูลพนักงานใหม่

```
int find_total(Employee e[], int len);
```

**หาเงินเดือนรวมของพนักงานทั้งบริษัท

Code Template

```
//main.c
#include<stdio.h>

#define NUM 5

typedef struct {
    char name[128];
    int salary;
} Employee;

Employee *input_employee();
int find_total(Employee e[], int len);

int main(){
    Employee *employees[NUM];

    printf("Max=");
```

```
    printf("Total=");

    //free employees
    return 0;
}

Employee *input_employee(){
    Employee *e;

    return e;
}

int find_total(Employee e[], int len){

}
```

Test Input

```
John Doe/3000
Kent J/9000
Marry D/4000
Jenny K/2000
Farm S/1000
```

Test Output

```
Total=19000
...
```

Adding

เขียนฟังก์ชันเพื่อรับค่าตัวเลข จนกว่าผู้ใช้จะใส่เลข 0 โดยทุกครั้งที่รับตัวเลข โปรแกรมจะแสดงผลรวมของตัวเลขทั้งหมดโดยแยกแสดงค่าบวก และค่าลบ ทุกครั้ง

```
N01: <4>
=4/0
N02: <-3>
=4/-3
N03: <7>
=11/-3
...
N08: <9>
=42/-12
N09: <0>
42-12=30
```

ทั้งนี้จะต้องการใช้ฟังก์ชัน adding อย่างเหมาะสม

```
void adding(int *pos_ptr, int *neg_ptr, int v);
```

=> `pos_ptr` เป็นพอยเตอร์อ้างอิงไปยังตัวเก็บผลรวมของค่าบวก

=> `neg_ptr` เป็นพอยเตอร์อ้างอิงไปยังตัวเก็บผลรวมด้านลบ

=> `v` ค่าที่ต้องประมวลผล

Code Template

```
//main.c
#include<stdio.h>
void adding(int *pos_ptr, int *neg_ptr, int v);

int main(){
    int pos = 0, neg = 0;

    return 0;
}
```

Test Input

```
3
-1
2
8
-5
7
0
```

Test Output

```
N01: =3/0
N02: =3/-1
N03: =5/-1
N04: =13/-1
N05: =13/-6
N06: =20/-6
N07: 20-6=14
...
```

Line Distance with Pointer

รับข้อมูลจุดพิกัด `x,y` 2 จุดจากผู้ใช้ เพื่อคำนวณหาระยะห่างระหว่างจุดสองจุด

โดยกำหนดให้มีการนิยามและเรียกใช้ฟังก์ชันข้างล่างอย่างเหมาะสม

```
void point_input(Point *p);
```

** รับค่าจุดจากผู้ใช้ (ไม่มีการ `printf` ในฟังก์ชันนี้)

```
float line_distance(Point *p1, Point *p2);
```

** หาค่าระยะห่างระหว่างจุดสองจุด

รูปแบบการแสดงผล

```
P1: <3 6>
P2: <0 0>
=6.71
```

Code Template

```
//main.c
#include<stdio.h>

void point_input(Point *p);
float line_distance(Point *p1, Point *p2);

int main(){

    return 0;
}
```

Test Input

```
5 7
1 4
```

Test Output

```
P1: P2: =5.00
...
```

Complex Number

จำนวนเชิงซ้อน เป็นจำนวนที่ประกอบด้วย ส่วนจริง และส่วนจินตภาพ

จงเขียนโปรแกรมเพื่อรับจำนวนเชิงซ้อนสองตัว แล้วแสดงค่าผลบวกของจำนวนดังกล่าว

โดยการหาผลบวก ให้เขียนโดยใช้ฟังก์ชัน

```
Complex add_complex(Complex a, Complex b);
```

รูปแบบการแสดงผล

```
C-A: <3 7>
C-B: <5 1>
=8+8i
```

Code Template

```
//main.c
#include<stdio.h>
typedef struct {
    int real;
    int img;  //imagine
} Complex;

Complex add_complex(Complex a, Complex b);

int main(){
    Complex a, b, result;

    printf("=%d%+di\n", result.real, result.img);
    return 0;
}
```

Test Input

```
7 3 5 -7
```

Test Output

```
C-A: C-B: =12-4i
...
```

Complex Number Pointer

จงเขียนโปรแกรมเพื่อรับจำนวนเชิงซ้อนสองตัว แล้วแสดงค่าผลบวกของจำนวนดังกล่าว

โดยการหาผลบวก ให้เขียนโดยใช้ฟังก์ชัน

```
void add_complex(Complex *rptr, Complex *aptr, Complex *bptr);
```

=> `aptr` พอยเตอร์อ้างอิงไปยังจำนวนเชิงซ้อน a

=> `bptr` พอยเตอร์อ้างอิงไปยังจำนวนเชิงซ้อน b

=> `rptr` พอยเตอร์อ้างอิงไปยังจำนวนเชิงซ้อนผลลัพธ์ของ a + b

รูปแบบการแสดงผล

```
C-A: <3 7>
C-B: <5 1>
=8+8i
```

Code Template

```
//main.c
#include<stdio.h>
typedef struct {
    int real;
    int img;  //imagine
} Complex;

void add_complex(Complex *rptr, Complex *aptr, Complex *bptr);
```

```
int main(){
    Complex a, b, result;

    printf("=%d%+di\n", result.real, result.img);
    return 0;
}
```

Test Input

```
7 3
5 -7
```

Test Output

```
C-A: C-B: =12-4i
```

Basic Pointer to String

รับข้อความจากหน้าจอหนึ่งข้อความ แล้วทำการเปลี่ยนข้อความดังกล่าวให้เป็นตัวใหญ่ทั้งหมด

การเปลี่ยนตัวอักษรให้เป็นตัวใหญ่ ให้ใช้ฟังก์ชัน

```
void to_cap(char *cptr);
```

รูปแบบการแสดงผล

```
Enter: <Hello World>
= HELLO WORLD
```

Code Template

```
//main.c
#include<stdio.h>
#include<string.h>

void to_cap(char *cptr);

int main(){
    char s[256];
    int len = -1, i;

    printf("Enter: ");
    scanf("%[^\n]s", s);

    len = strlen(s);

    for(i = 0; i < len; i++){
        //your code here
    }

    printf("=%s\n", s);
    return 0;
}
```

Test Input

```
I am Not In Love
```

Test Output

```
Enter: =I AM NOT IN LOVE
```

Complex Numbers

จงเขียนโปรแกรมเพื่อรับจำนวนเชิงซ้อน 5 ตัว บันทึกไว้ในอาเรย์ แล้วแสดงค่าผลบวกของจำนวนดังกล่าว โดยใช้ฟังก์ชันที่กำหนดให้อย่างเหมาะสม

```
void input_complex(Complex *cptr);
```

รับข้อมูลจำนวนเชิงซ้อนหนึ่งตัว

```
void top_complex(Complex *rptr, Complex *cptr)
```

เพิ่มค่าจำนวนเชิงซ้อน `c` ไปยังจำนวนเชิงซ้อน `r` เช่น ในกรณี `r` มีค่าเป็น 2+3i และ `c` เป็น 1+2i หลังการประมวลผล `r` จะมีค่าเป็น 3+5i

```
void sum_complex(Complex *rptr, Complex cs[], int len);
```

หาผลรวมของสมาชิกของอาเรย์ `cs` จำนวน `len` ตัว

รูปแบบการแสดงผล

```
C01: <3 7>
...
C05: <2 -1>
=15+12i
```

Code Template

```
//main.c
#include<stdio.h>
#define LEN 5

typedef struct {
    int real;
    int img;  //imagine
} Complex;

void input_complex(Complex *cptr);
void top_complex(Complex *rptr, Complex *cptr);
void sum_complex(Complex *rptr, Complex cs[], int len);

int main(){
    Complex a[LEN], result;

    printf("=%d%+di\n", result.real, result.img);
    return 0;
}
```

Test Input

```
7 3
5 2
-2 1
-1 -1
0 0
```

Test Output

```
C01: C02: C03: C04: C05:=9+5i
```

Exercise05 - File

Read Binary File

จงเขียนโปรแกรมที่ทำการอ่านไฟล์ไบนารี ชื่อ `employee.bin`
โดยไฟล์จะเริ่มต้นจากตัวเลขจำนวนเต็ม บอกจำนวนข้อมูลพนักงานที่บันทึกไว้ในไฟล์ จากนั้นประมวลผลข้อมูลให้อยู่ในรูปของ `struct employee` ที่กำหนดให้ แล้วทำการแสดงข้อมูลของพนักงานทุกคน พร้อมแสดงผลเงินเดือนรวม
*** ทำการแสดงผลจากการอ่านออกทางหน้าจอ สำหรับการอ่านไฟล์นั้นจะต้องทำการกดปุ่ม L-Test ก่อน เพื่อ download ไฟล์มาเก็บไว้ในเครื่องโดยอัตโนมัติ แล้วค่อยทำการรันโปรแกรม ***
รูปแบบการแสดงผล

```
John Doe:3000.0
Mark Ken:2300.0
Sucy Merc:2000.0
=7300.0
```

Code Template

```
//main.c
#include <stdio.h>

struct employee {
  char name[128];
  float salary;
};

typedef struct employee Employee;

int main() {
  int num;
  Employee e;
  float total = 0.0;
  FILE *fp;

  printf("%s:%.1f\n", e.name, e.salary);
  printf("=%.1f\n", total);
  return 0;
}
```

Test Input

Test Output

```
John Doe:3000.0
Mark Ken:2300.0
Sucy Merc:2000.0
=7300.0
```

Random Access

เขียนโปรแกรม เพื่อดึงตัวเลขจำนวนเต็มที่ถูกบันทึกไว้ในไฟล์ไบนารี `num.dat` โดยโปรแกรมจะถูกเขียนโดยไม่ทราบล่วงหน้าว่ามีตัวเลขจำนวนกี่ตัวในไฟล์ดังกล่าว และไม่มีการอ่านข้อมูลทั้งหมดขึ้นสู่อาเรย์ โปรแกรมจะรับข้อมูลจากผู้ใช้ ว่า ต้องการทราบตัวเลขลำดับใด (ให้ลำดับแรกคือ 1) หลังจากนั้น จะแสดงตัวเลขที่อยู่ ณ. ตำแหน่งที่กำหนด
โปรแกรมจะหยุดการทำงานเมื่อผู้ใช้ใส่ตำแหน่งเป็น 0 หรือค่าติดลบ ทั้งนี้ให้ถือว่า ผู้ใช้จะไม่ใส่ตำแหน่งที่ไม่มีอยู่จริงในไฟล์
ตัวอย่างการแสดงผล

```
Enter: <1>
= 40
Enter: <372>
= 68
Enter: <250>
= 12
Enter: <0>
Done.
```

ทำการแสดงผลจากการอ่านออกทางหน้าจอ สำหรับการอ่านไฟล์นั้นจะต้องทำการกดปุ่ม L-Test ก่อน เพื่อ download ไฟล์มาเก็บไว้ในเครื่องโดยอัตโนมัติ แล้วค่อยทำการรันโปรแกรม

Code Template

```
//main.c
#include<stdio.h>

int main(){
    FILE *fp;
    int d;

    printf("Enter: ");
    printf("= %d\n", d);
    printf("Done.\n");

    return 0;
}
```

Test Input

```
3
777
256
43
0
```

Test Output

```
Enter:=79
Enter:=91
Enter:=80
Enter:=9
Enter:Done.
```

Exercise06 – Intro to Linked List

String to List

จงรับข้อความ 1 บรรทัดจากผู้ใช้ ทำการแปลงเป็น Linked List อย่างเหมาะสม โดยใช้ฟังก์ชัน `string_to_list` ที่ปรากฏใน slides

```
LNP string_to_list(char s[]);
```

เรียกใช้ฟังก์ชัน `all_cap` เพื่อเปลี่ยนตัวอักษรทุกตัวใน list ให้เป็นตัวอักษรตัวใหญ่

```
void all_cap(LNP cptr);
```

ทำการแสดงข้อความผลลัพธ์ จากการเข้าถึงสมาชิกใน Linked List แบบ `iterative`

รูปแบบการแสดงผล

```
Enter: Hello world
=HELLO WORLD
*ในข้อนี้จะใช้ gets แทน scanf
```

Code Template

```
//main.c
#include<stdio.h>

struct listnode {
  char data;
  struct listnode *nextptr;
};

typedef struct listnode  LISTNODE;
typedef LISTNODE *LNP;

LNP string_to_list(char s[]);
void all_cap(LNP cptr);

int main(){
  LNP head = NULL;
  char line[128];

  printf("Enter: ");
  gets(line);



  return 0;
}

LNP string_to_list(char s[]){
  LNP head = NULL, tail;
  int i;
```

```
  if (s[0] != '\0') {
    head = malloc(sizeof(LISTNODE));
    head->data = s[0];
    tail = head;
    for (i=1; s[i] != '\0'; i++){
      tail->nextptr = malloc(sizeof(LISTNODE));
      tail = tail->nextptr;
      tail->data = s[i];
    }
    tail->nextptr = NULL; /* list end */
  }
  return head;
}
```

Test Input

```
I love you.
```

Test Output

```
Enter: =I LOVE YOU.
```

Linked List

จากโค้ดที่กำหนดให้ ฟังก์ชัน `main` มีการสร้าง linked list ซึ่งมีสมาชิกเป็นโหนดที่มีเลขจำนวนเต็ม 1 ตัว เป็นข้อมูล ค่าข้อมูลดังกล่าวคือ 0
รับข้อมูลตัวเลขจากผู้ใช้ เพื่อนำไปสร้างโหนด แล้วต่อท้ายลิสต์ที่มีอยู่เดิม จนกว่าผู้ใช้จะใส่ตัวเลขค่าลบ
จงเขียนฟังก์ชันต่อไปนี้

```
LNP append_node(LNP tail, int v);
```

-- เพิ่ม node ให้ลิสต์โดยต่อจากหาง โดยเขียนฟังก์ชันนี้ภายใต้สมมติฐานว่า tail ไม่มีค่าเป็น NULL
-- คืนค่า tail ใหม่

```
int length(LNP head);
```

-- หาจำนวนสมาชิกในลิงค์ลิสต์ (เขียนเป็น recursive)

```
void print_list(LNP head);
```

-- พิมพ์สมาชิกทั้งหมด เรียงตามลำดับ (เขียนเป็น recursive)

```
void print_reverse_list(LNP head);
```

-- พิมพ์สมาชิกทั้งหมด จากหลังมาหน้า (เขียนเป็น recursive)

```
int sum(LNP head);
```

-- หาผลบวกของสมาชิดทุกตัวในลิสต์ (เขียนเป็น recursive)

```
void free_list(LNP *head);
```

-- free สมาชิกทั้งหมด และทำให้ลิสต์ว่างเปล่า (เขียนเป็น iterative)

เพื่อประมวลผลข้อมูล และแสดงผลข้อความ ตามรูปแบบข้างล่าง

```
Enter: <2>
Enter: <7>
Enter: <8>
Enter: <-1>
---Result---
Len=4
List=0 2 7 8
Reverse=8 7 2 0
Sum=17
```

Code Template

```
//main.c
#include<stdio.h>

LNP append_node(LNP tail, int v);
int length(LNP head);
void print_list(LNP head);
void print_reverse_list(LNP head);
int sum(LNP head);
void free_list(LNP *head);

int main(){
    LNP head, tail;
    int v;

    head = tail =(LNP)malloc(sizeof(LN));
    tail->v = 0;
    tail->next = NULL;

    //iteratively append list node, DO NOT forget to update the tail
    printf("Enter: ");
    scanf("%d", &v);

    printf("---Result---\n");

    //process your list here!
    printf("Len=");
    printf("List=");
    printf("Reverse=");
    printf("Sum=");

    //free your list here!

    return 0;
}
```

Test Input

```
7
2
6
9
4
2
3
5
8
-1
```

Test Output

```
Enter: Enter: Enter: Enter: Enter: Enter: Enter: Enter: Enter: Enter:
---Result---
Len=10
List=0 7 2 6 9 4 2 3 5 8
Reverse=8 5 3 2 4 9 6 2 7 0
Sum=46
```

Exercise07 – More Linked List

Linked List Polygon

เขียนโปรแกรมเพื่อรับจุด `x,y` เพื่อสร้าง Polygon (รูปหลายเหลี่ยม) โดยโปรแกรมจะรับจุดไปจนกว่าผู้ใช้จะป้อนจุดเริ่มต้นอีกครั้ง (รูปปิด) หลังจากนั้น ให้แสดงผลเส้นรอบรูป จำนวน 2 ครั้ง โดยครั้งที่ 1 จะคำนวณด้วยวิธี iterative ส่วนครั้งที่ 2 ให้คำนวณด้วยวิธี recursive

โดยให้มีการใช้ฟังก์ชันดังต่อไปนี้อย่างเหมาะสม

```
int next_point(Polygon *gon, Point *p);
```

เพิ่มจุด p เข้าไปในลิงค์ลิสต์ gon หากจุด p ไม่ใช่จุดเริ่มต้น โดยฟังก์ชันจะคืนค่า 1 หากการเพิ่มสำเร็จ และคืน 0 ในทางตรงกันข้าม

```
float iter_perim(Polygon gon);
```

หาเส้นรอบวงด้วยวิธี iterative

```
float recur_perim(Point *start, Polygon gon);
```

หาเส้นรอบวงด้วยวิธี recursive

รูปแบบการแสดงผล

```
------
P01: <1, 1>
P02: <3, 7>
...
P07: <1, 1>
iterative = 50.70
recursive = 50.70
```

หมายเหตุ: ให้ถือว่า ผู้ใช้จะป้อนข้อมูล **Polygon** ถูกต้อง (**valid polygon**) นศ. ไม่ต้องเพิ่มการตรวจสอบเงื่อนไข

Code Template

```
//main.c
#include<stdio.h>
#include<math.h>

struct point {
   int x;
   int y;
};
typedef struct point Point;

struct listnode {
   Point p;
   struct listnode *next;
};
```

```
typedef struct listnode LN;
typedef struct listnode* LNP;
typedef struct listnode* Polygon;

int next_point(Polygon *gon, Point *p);
float distance(Point *p1, Point *p2);
float iter_perim(Polygon gon);
float recur_perim(Point *start, Polygon gon);

int main(){
   Polygon gon = NULL;
   Point p;
   int i = 0;
   do{
      printf("P%02d: ", ++i);
      scanf(" %d %d", &p.x, &p.y);
   }while(next_point(&gon, &p));

   printf("iterative = %.2f\n", iter_perim(gon));
   printf("recursive = %.2f\n", recur_perim(&gon->p, gon));
   return 0;
}

int next_point(Polygon *gon, Point *p){

}

float distance(Point *p1, Point *p2){
   float xdiff = p1->x - p2->x;
   float ydiff = p1->y - p2->y;
   return sqrt(xdiff*xdiff + ydiff*ydiff);
}

float iter_perim(Polygon gon){
   float perim = 0.0;

   return perim;
}

float recur_perim(Point *start, Polygon gon){

}
```

Test Input

```
5 5
6 8
12 3
7 8
5 5
```

Test Output

```
P01: P02: P03: P04: P05: iterative = 21.65
recursive = 21.65
```

Exercise08 - Stack

Reverse-with-stack

รับข้อมูลเลขจำนวนเต็ม จนกว่า ผู้ใช้จะใส่เลข 0 หรือติดลบ ไปข้อมูล push ใส่ลงไปใน stack
เมื่อเสร็จสิ้นการรับข้อมูล ทำการ pop ข้อมูลออกมาจาก stack เพื่อพิมพ์ข้อมูลย้อนหลัง
ทั้งนี้ ให้ใช้ linked list ในการ implement โครงสร้าง stack และการ push จะทำการ `insert` ที่หัว และการ `pop` ทำการ `remove` ที่หัว

รูปแบบการแสดงผล

```
============================
N01: <5>
N02: <3>
...
N09: <2>
N10: <-1>
= 2 ... 3 5
```

Code Template

```
main.c
#include<stdio.h>
struct listnode{
   int data;
   struct listnode *next;
};

typedef struct listnode LN;
typedef LN *LNP;

void push(LNP *sptr, int v);
int pop(LNP *sptr);

int main(){
   LNP stack = NULL;

   printf("N%02d: ");

   printf("= ");
   printf("%d ");
   printf("\n");

   return 0;
}
```

Test Input

```
3
6
7
```

```
8
1
4
2
9
3
5
-1
```

Test Output

```
N01:N02:N03:N04:N05:N06:N07:N08:N09:N10:N11:=5 3 9 2 4 1 8 7 6 3
```

Exercise09 - Tree

Tree Basics

จงเขียนโปรแกรมอย่างเหมาะสม เพื่อประมวลผล tree ที่สร้างขึ้น โดยมีสมาชิก เป็น 4 2 5 7 1 ตามลำดับ (โค้ดในการสร้าง tree เขียนไว้แล้วใน main) เพื่อแสดงผลลัพธ์ดังนี้

```
Height = 3
In-order = 1 2 4 5 7
Pre-order = 4 2 1 5 7
Post-order = 1 2 7 5 4
Total-nodes = 5
Leaf-nodes = 2
```

Code Template

```
//main.c
#include <stdio.h>
#include "integer_tree.h"
int tree_height(TREE t);
void print_in_order(TREE t);
void print_pre_order(TREE t);
void print_post_order(TREE t);
int total_nodes(TREE t);
int leaf_nodes(TREE t);

int main()
{
    TREE t = NULL;
    insert_node(&t, 4);
    insert_node(&t, 2);
    insert_node(&t, 5);
    insert_node(&t, 7);
    insert_node(&t, 1);

    return 0;
}
```

Code Template

```
integer_tree.h
#include <stdio.h>
struct treenode{
	struct treenode *leftptr;
	int data;
	struct treenode *rightptr;
	};
typedef struct treenode TREENODE;
typedef TREENODE *TREE;

TREE mkEmpty(void) { return NULL;}
int isEmptyTree(TREE t) { return t == NULL; }
TREE mkTree(int x,TREE leftT, TREE rightT) {
	TREE t;
	t = malloc(sizeof(TREENODE));
	t->data = x;
	t->leftptr = leftT;
	t->rightptr = rightT;
	return t;
}
void insert_node(TREE *tp, int value) {
	if (*tp == NULL) {
		*tp = malloc(sizeof(TREENODE));
		(*tp)->data = value;
		(*tp)->leftptr = NULL;
		(*tp)->rightptr = NULL;
	}else if (value < (*tp)->data )
		insert_node(&((*tp)->leftptr), value);
	else if (value > (*tp)->data )
		insert_node(&((*tp)->rightptr), value);
	else
		printf("duplicate node\n");
}
```

Test Input

-

Test Output

```
Height = 3
In-order = 1 2 4 5 7
Pre-order = 4 2 1 5 7
Post-order = 1 2 7 5 4
Total-nodes = 5
Leaf-nodes = 2
```