

# C Programming : Pointer & Memory

## Instructor

Pathorn Tengkiattrakul, [pathorn.teng@gmail.com](mailto:pathorn.teng@gmail.com)

# Today's topics

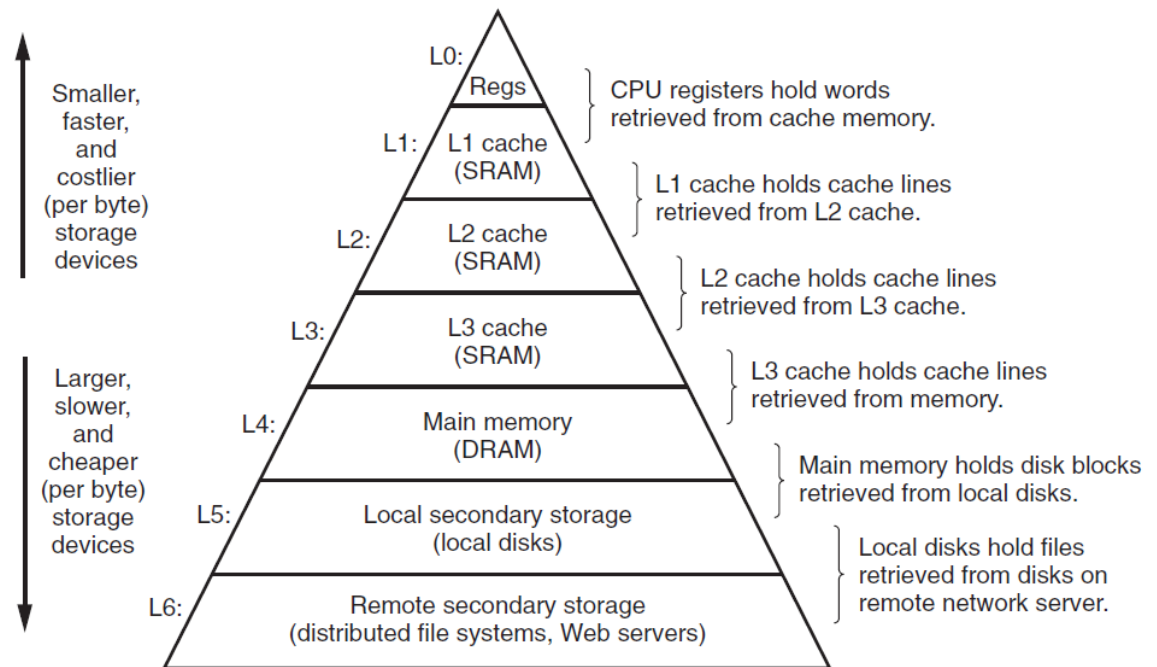
- Memory management
- Pointer
- Pointer & Function
- Pointer & Array
- Memory allocation
- Pointer to pointer
- Review

# อะไรและทำไมต้อง main memory

- Main memory หรือ Ram เป็นหน่วยความจำชั่วคราวของคอมพิวเตอร์ใช้ในการเพิ่มประสิทธิภาพการทำงานของเครื่อง
- ลองจินตนาการถึงหากเราต้องการทำรายงานในห้องสมุด
  - หนังสือในห้องสมุด = (ข้อมูลใน harddisk)
  - หนังสือที่เราสนใจจะมาทำรายงานเอามาไว้บนโต๊ะ = (ข้อมูลใน ram)

# การเก็บข้อมูลของคอมพิวเตอร์

- ชนิดของแหล่งเก็บข้อมูลในคอมพิวเตอร์
- Register (L0)
- Cache (L1,L2,L3)
- Ram (L4)
- Harddisk (L5)



# Memory size

- ขนาดของ main memory ขึ้นอยู่กับคุณลักษณะของเครื่อง เช่น 4Gb 8Gb (Gigabytes)
- หน่วยย่อยที่สุดของ memory คือ bit นั่นคือ 1 bit เก็บข้อมูลได้ 0 หรือ 1 เท่านั้น
- แต่โดยส่วนมากเรานิยมใช้หน่วย bytes ในการระบุขนาดของ main memory 1 byte = 8 bits

# ขนาดของตัวแปรชนิดต่างๆ (byte)

C declaration	32-bit	64-bit
char	1	1
short int	2	2
int	4	4
long int	4	8
long long int	8	8
char *	4	8
float	4	4
double	8	8

# Memory 4Gb

FFFFFFFF
FFFFFFFE
FFFFFFFD
.....
00000002
00000001
00000000

แทนด้วยเลขฐาน 16 (1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)

# การเก็บข้อมูลในหน่วยความจำ

- ตัวแปรชนิด char ใช้พื้นที่ในการเก็บ 1 byte เก็บข้อมูลได้  $2^8 = 256$
- ดังนั้นเมื่อ char เช่น 0x1A (16) เก็บอยู่ใน memory

Address	Value
00000000	0x1A

- ตัวแปรชนิด int ใช้พื้นที่ในการเก็บ 4 bytes เก็บข้อมูลได้  $2^{32}$  (สี่พันล้านกว่า)
- ดังนั้นเมื่อ int เช่น 0x01F52A17 เก็บอยู่ใน memory

Address	Value
00000000	01
00000001	F5
00000002	2A
00000003	17

Address	Value
00000000	17
00000001	2A
00000002	F5
00000003	01



# Big Endian vs Little Endian

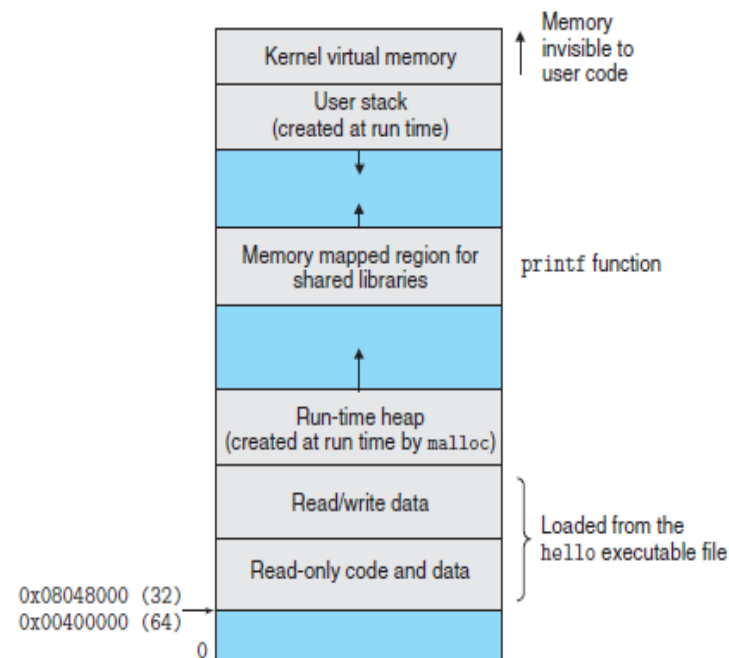
- ลำดับการเก็บข้อมูลของตัวแปรขึ้นอยู่กับชนิดของเครื่องคอมพิวเตอร์
- Big Endian machine เก็บ byte มีความสำคัญสุด (มีค่ามาสุด) ไว้ที่ address แรก
- Little Endian machine เก็บ byte ที่มีความสำคัญน้อยสุด (มีค่าน้อยสุด) ไว้ที่ address แรก

Address (Big)	Value
00000000	01
00000001	F5
00000002	2A
00000003	17

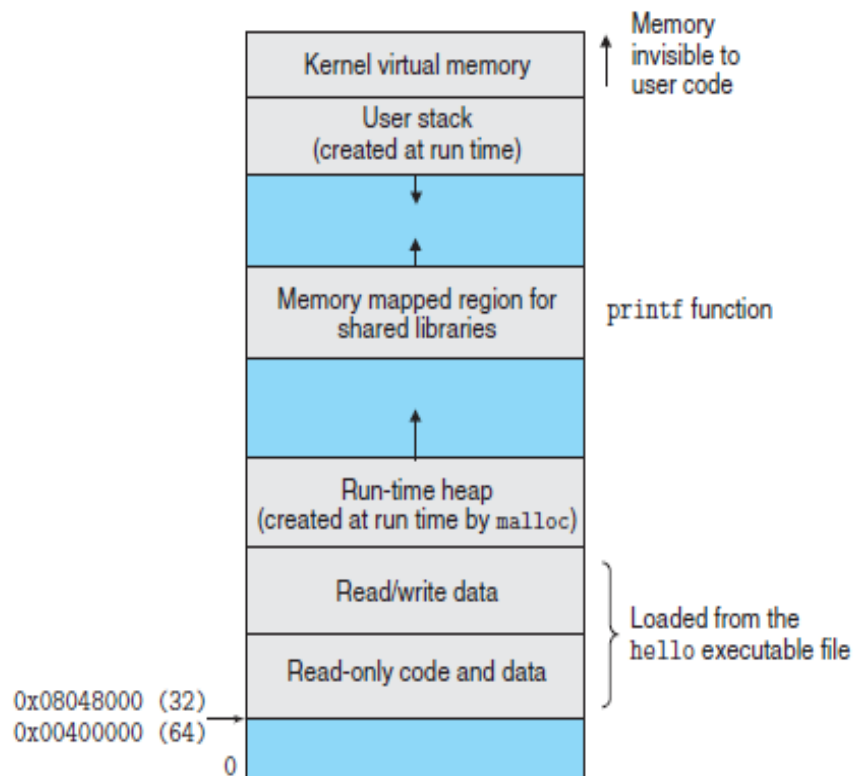
Address (Little)	Value
00000000	17
00000001	2A
00000002	F5
00000003	01

# กระบวนการทำงานของโปรแกรม

- ผู้ใช้งานเรียกใช้โปรแกรม (run โปรแกรมโดยใช้ command line, ดับเบิลคลิกโปรแกรม หรือกด run บน IDE (เช่น dev c))
- หน่วยประมวลผลโหลดโปรแกรมลง memory โดยโครงสร้างของ virtual memory จะมีดังนี้



# โครงสร้าง virtual memory ของโปรแกรม



- **Read/write code and data** -> เก็บข้อมูลโปรแกรม เช่น machine code, global variable, static variable
- **Run-time heap** -> เป็นพื้นที่ที่จะถูกใช้เมื่อมีการเรียกใช้ฟังก์ชัน malloc/calloc ในการจอง memory
- **Shared Libraries** -> เก็บ code library เช่น code ของฟังก์ชัน printf
- **User stack** -> เก็บค่าของ local variable และ parameter ของฟังก์ชัน

## คอมพิวเตอร์ 4 bits (word size)

- Computer 4 bits เป็นคอมพิวเตอร์ในยุคโบราณที่ใช้เลข 4 bits ในการกำหนด address ของ main memory
- ดังนั้น จะมี address ที่ไม่ซ้ำกันได้ทั้งหมดกี่ address?
- $2^4 = 16$  address ที่ไม่ซ้ำกัน
- 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010  
1011 1100 1101 1110 1111
- ดังนั้นปริมาณ main memory ที่ computer 4 bits จะรับได้คือ 16bytes

# คอมพิวเตอร์ 32bits และ 64 bits (word size)

- ระบบปฏิบัติการไม่ว่าจะเป็น Windows หรือ Linux ต่างมีสิ่งที่เรียกว่า word size ซึ่งเป็นสิ่งบ่งบอกว่าคอมพิวเตอร์จะประมวลผลข้อมูลที่มีขนาดเท่าไร
- ระบบปฏิบัติการปัจจุบันนิยม word size ขนาด 32bits และ 64bits เป็นหลัก ดังนั้นจำนวน address ใน memory ที่ไม่ซ้ำกัน
- $2^{32} = 4294967296$  bytes หรือ 4Gb
- $2^{64} = 1.8446744073709551616 * 10^{19}$

# การเก็บข้อมูลใน main memory

- โดยปกติในภาษา C เมื่อเราทำการประกาศตัวแปรใหม่ เช่น `int a = 10;` เมื่อโปรแกรมถูกทำงานก็จะทำการจองพื้นที่ 4 bytes ใน main memory (ram) ให้กับตัวแปร a
- ดังนั้นค่า 10 ก็จะถูกเก็บไว้ในพื้นที่แห่งหนึ่งใน main memory โดยพื้นที่แห่งนั้นก็จะมีเลขกำกับไว้ (memory address)

value	10	Null	Null	Null
address	1000	1004	1008	1016

# Pointer (ตัวชี้)

- Pointer เป็นตัวแปรชนิดหนึ่งที่ใช้เก็บ address ของหน่วยความจำแทนที่จะเก็บข้อมูลตรง ๆ

```
#include<stdio.h>
main(){
    int a = 10;
    int *b;
    b = &a;
    printf("b = %u, *b = %d",b,*b);
}
```

	*b = 1000			
value	10	Null	Null	Null
address	1000	1004	1008	1016
	b = 1000			

# เครื่องหมายที่เกี่ยวข้องกับ pointer

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

```
main(){
```

```
    int a = 10;
```

ประกาศตัวแปร pointer

```
    int *b;
```

b ชี้ไปที่ address ที่เก็บข้อมูลของตัวแปร a

```
    b = &a;
```

```
    printf("*b = %d", *b);
```

เรียกดูข้อมูลจาก address ที่ pointer b ชี้

```
}
```



# Casting pointer

- เราจะสามารถใช้ pointer ของตัวแปรชนิดหนึ่งชี้ไปยังตัวแปรอีกชนิดได้
- เรียกการทำ casting pointer (ไม่ควรทำหากไม่จำเป็น)

```
#include<stdio.h>
main(){
    int a = 10;
    char *b;
    b = (char *) &a;
    printf("*b = %d", *b);
}
```

# void\* pointer

- void\* pointer เป็น pointer ที่ใช้ในการเก็บค่า address เป็นหลัก ไม่ควรทำการ dereference

```
#include<stdio.h>
main(){
    int a = 10, *c;
    void *b;
    b = (void *) &a;
    c = b;
    printf("*c = %d", *c);
}
```

## ตัวอย่าง Pointer (ต่อ)

```
#include<stdio.h>
main() {
    int a = 10;
    int *b;
    b = &a;
    *b = *b++;
    printf("a = %d",a);
    printf("b = %u, &a = %u, *b = %d, a = %d, b, &a, *b, a);
}
```

# ฟังก์ชัน size of

- sizeof() เป็นฟังก์ชันที่ใช้ในการตรวจสอบขนาดของตัวแปร

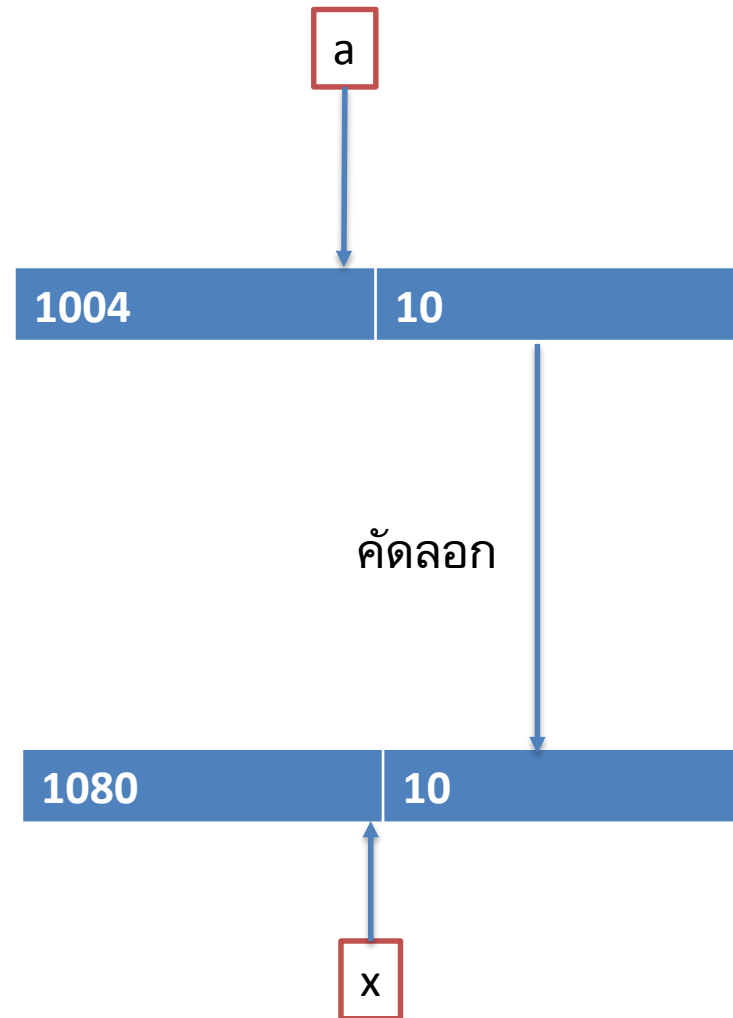
```
#include <stdio.h>
int main(){
    int a;
    float b;
    double c;
    char d;
    printf("Size of int: %d bytes\n",sizeof(a));
    printf("Size of float: %d bytes\n",sizeof(b));
    printf("Size of double: %d bytes\n",sizeof(c));
    printf("Size of char: %d byte\n",sizeof(d));
    return 0;
}
```

# Pointer & Function

- การส่ง parameter ไปยังฟังก์ชันแบ่งออกเป็นสองแบบ pass-by-value และ pass-by-reference
- Pass-by-value คือการเรียกใช้งานฟังก์ชันที่เราคุ้นเคย นั่นคือส่งค่า ๆ หนึ่งไปให้กับฟังก์ชันประมวลผล
- Pass-by-reference คือการเรียกใช้งานฟังก์ชันโดยการส่ง address ของตัวแปรในไปให้แทนการส่งค่าปกติ

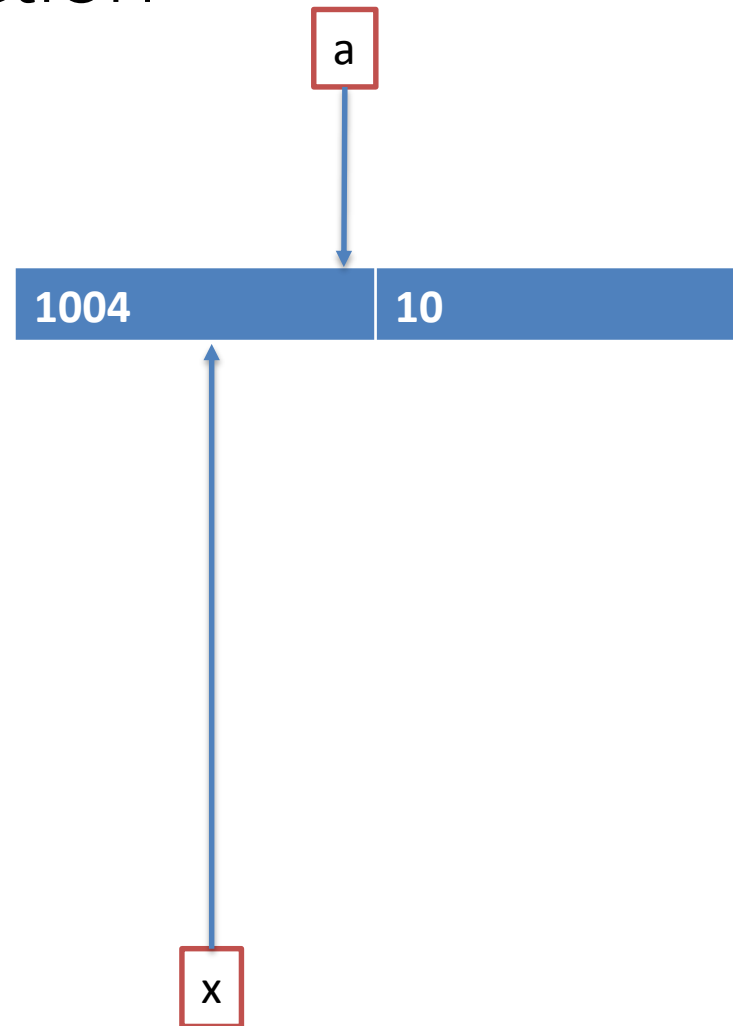
# Pass-by-value function

```
#include<stdio.h>
int double_value(int x);
main(){
    int a = 10;
    a = double(a);
}
int double_value(int x){
    return x*2;
}
```



# Pass-by-reference function

```
#include<stdio.h>
int double_value(int *x);
main(){
    int a = 10;
    double(&a);
}
int double_value(int *x){
    return *x*2;
}
```



# ตัวอย่าง pass-by-reference

```
#include<stdio.h>
void sort(int *number);
main(){
    int list[3] = {5,4,2};
    sort(list);
}
int* sort(int *number){
    int i,j,temp;
    for(i=0;i<3;i++){
        for(j=i+1;j<3;j++){
            if(number[i] > number[j])
                temp = number[i];
                number[i] = number[j];
                number[j] = temp;
        }
    }
    return number;
}
```



# ตัวอย่าง pass-by-reference

```
#include<stdio.h>
int* double_value(int value);
main(){
    int x = 10;
    int *pointer;
    pointer = double_value(x);
    printf("pointer = %d",pointer);
}
int* double_value(int x){
    x = x*2;
    return &x;
}
```

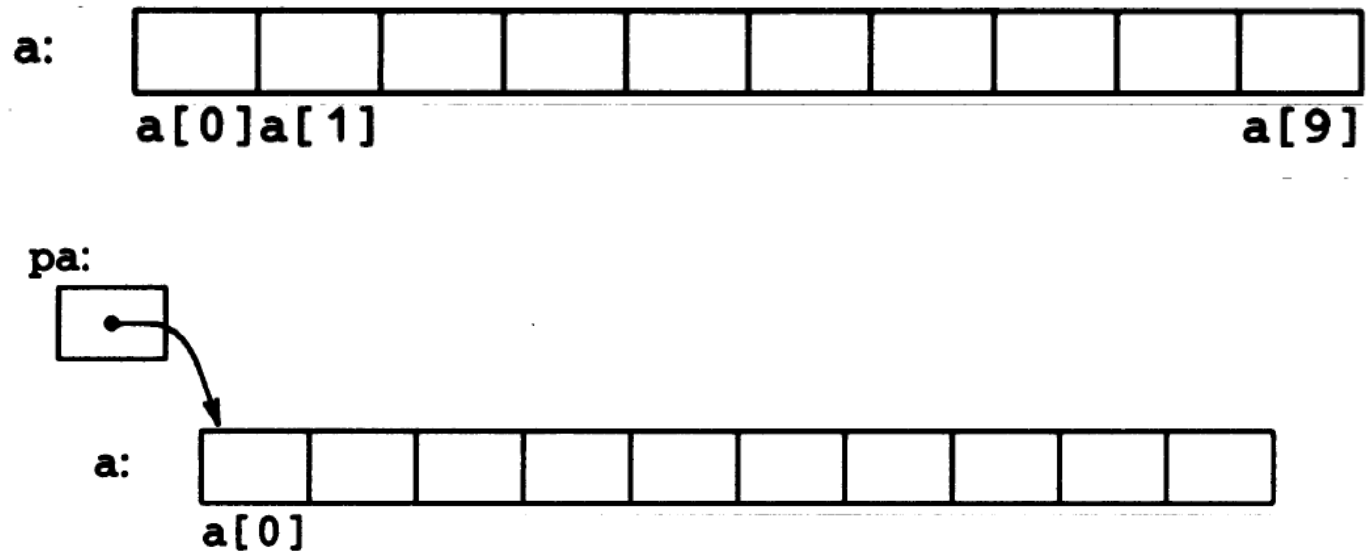
# Pointer & Array

Pointer และ Array มีความสัมพันธ์ใกล้เคียงกันมากจนอาจจะเรียกได้ว่า array นั้นทำหน้าที่เหมือนเป็น pointer ชนิดหนึ่ง

```
int a[10];
```

```
int *pa;
```

```
pa = a;
```



# Pointer Arithmetic

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

```
main(){
```

```
    int a[10];
```

```
    int *pa;
```

```
    a[0] = 0;
```

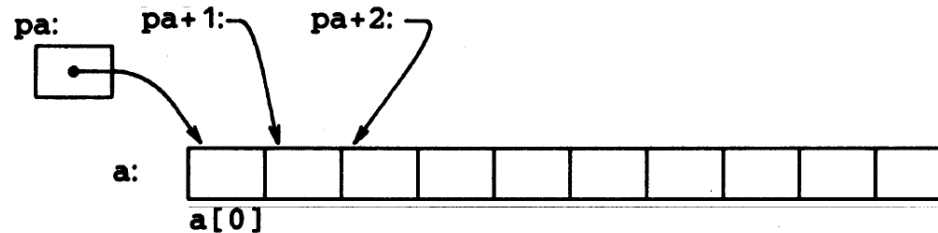
```
    a[1] = 1;
```

```
    a[2] = 2;
```

```
    pa = a;
```

```
    printf("*pa = %d, *(pa+1) = %d, *(pa+2) = %d", *pa, *(pa+1), *(pa+2));
```

```
}
```



การบวกตัวแปรชนิด pointer จะทำการบวกเป็นจำนวนขนาดของชนิดตัวแปรที่  
ตัวแปร pointer ชี้ไป เช่น `int +4`, `char + 1`, `double + 8` เป็นต้น

# Pointer & Array (ต่อ)

```
#include<stdio.h>
main(){
    int a[10];
    a[0] = 0;
    a[1] = 1;
    a[2] = 2;
    printf("*a = %d, *(a+1) = %d, *(a+2) = %d",*a, *(a+1), *(a+2));
}
```

# Array & memory

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

```
main(){
```

```
    int a[2][2];
```

```
    int *p;
```

```
    a[0][0] = 1;
```

```
    a[0][1] = 3;
```

```
    a[1][0] = 2;
```

```
    a[1][1] = 4;
```

```
    p = a;
```

```
    printf("(p+1) = %d",*(p+1));
```

```
}
```

Address	Value
0000	1
0004	2
0008	3
00012	4

# Memory allocation (การจองหน่วยความจำ)

- ตัวแปรชนิด pointer เมื่อถูกสร้างขึ้นมาจะมีค่าแบบสุ่ม นั่นคือ address ที่ชี้ไปไม่ได้ถูกจองไว้สำหรับโปรแกรมนี้ อาจจะทำให้เกิด segmentation fault (จะอธิบายในภายหลัง)
- malloc() และ calloc() เป็นสองฟังก์ชันที่ใช้ในการจองพื้นที่ในหน่วยความจำสำหรับการเก็บข้อมูล
- malloc() จองพื้นที่โดยไม่ได้ทำการ zerolized (ทำให้ค่าในพื้นที่ที่จองเป็น 0 ทั้งหมด) แต่ calloc() จะทำการ zerolized พื้นที่ก่อน

# ตัวอย่าง malloc & calloc

```
#include<stdio.h>
main(){
    int *a,*b,*c;
    a = malloc(4);
    printf("*a = %d",*a);
    b = calloc(4);
    printf("*b = %d",*b);
    c = malloc(sizeof(int) * 10);
    c[0] = 100; c[1] = 200; c[2] = 300;
    printf("c[0] = %d, *(c+1) = %d, c[2] = %d",c[0],*(c+1),c[2]);
}
```

# Free memory

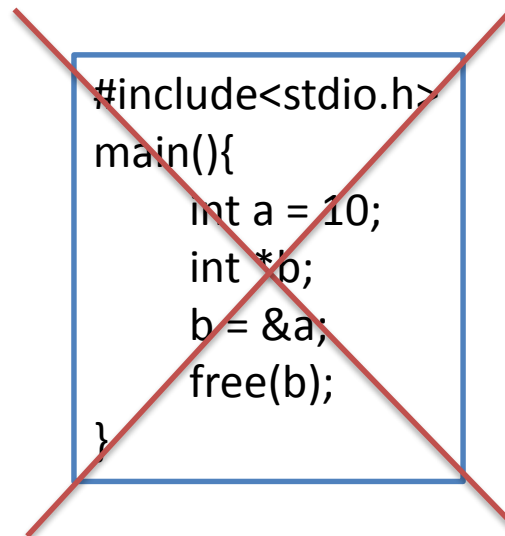
เมื่อได้ใช้ฟังก์ชัน malloc หรือ calloc ในการจองพื้นที่ของ main memory แล้ว เมื่อเลิกใช้งานผู้ใช้งานต้องทำการยกเลิกการจอง (free) memory ด้วย ฟังก์ชัน free() ในการปล่อย memory ที่ถูกจองไว้คืนให้กับ operating system

```
#include<stdio.h>
main(){
    int *a;
    a = malloc(sizeof(int));
    free(a);
}
```



# Do not free memory of variable

- หากไม่ได้ใช้ฟังก์ชัน malloc/calloc ในการจองพื้นที่
- อย่า free memory



```
#include<stdio.h>
main(){
    int a = 10;
    int *b;
    b = &a;
    free(b);
}
```

# Pointer to Pointer

- เราสามารถใช้ pointer ใช้ไปยัง pointer ซ้อนกันได้

```
#include<stdio.h>
main(){
    int **a;
    int *b;
    int c = 5;
    b = &c;
    a = &b;
    printf("**a = %d",**a)
}
```

	Address	Value
a →	1012	1008
b →	1008	1004
c →	1004	5

## Value summary

&a = 1012	b = ?	&c = ?
a = 1008	*b = ?	c = ?
*a = 1004	&b = ?	
**a = 5		

# Function Pointer

- Pointer function เป็นการนำเอา pointer ชี้ไปยังที่อยู่ฟังก์ชัน
- มีประโยชน์ในการเขียนโปรแกรมหลายอย่าง เช่น โปรแกรมคำนวณน้ำหนักเหมาะสม
- เราสามารถสร้าง function pointer ในการเรียกใช้ฟังก์ชันที่ต่างกันโดยการเขียน code เพียงครั้งเดียวได้

# การประกาศ function pointer

(return\_type) (\*function\_pointer\_name)(parameter\_type);

```
#include<stdio.h>
int double_value(int x);
int (*function_pointer)(int);
main(){
    int a = 10;
    function_pointer = &double_value;
    a = function_pointer(a); // (*function_pointer)(a);
    printf("a = %d",a);
}
int double_value(int x){
    return x*2;
}
```

# ตัวอย่างฟังก์ชันคำนวณน้ำหนักเหมาะสม

```
#include<stdio.h>
int man_weight(int height);
int woman_weight(int height);
int (*suitable_weight)(int);
main(){
    char gender = 'm';
    int height = 170;
    if(gender == 'm') suitable_weight = &man_weight;
    else if(gender == 'f') suitable_weight = &women_weight;

    printf("suitable weight = %d\n",suitable_weight(height));
}
int man_weight(int height){
    return 0.9*(height-100);
}
int women_weight(int height){
    return 0.8*(height-100);
}
```

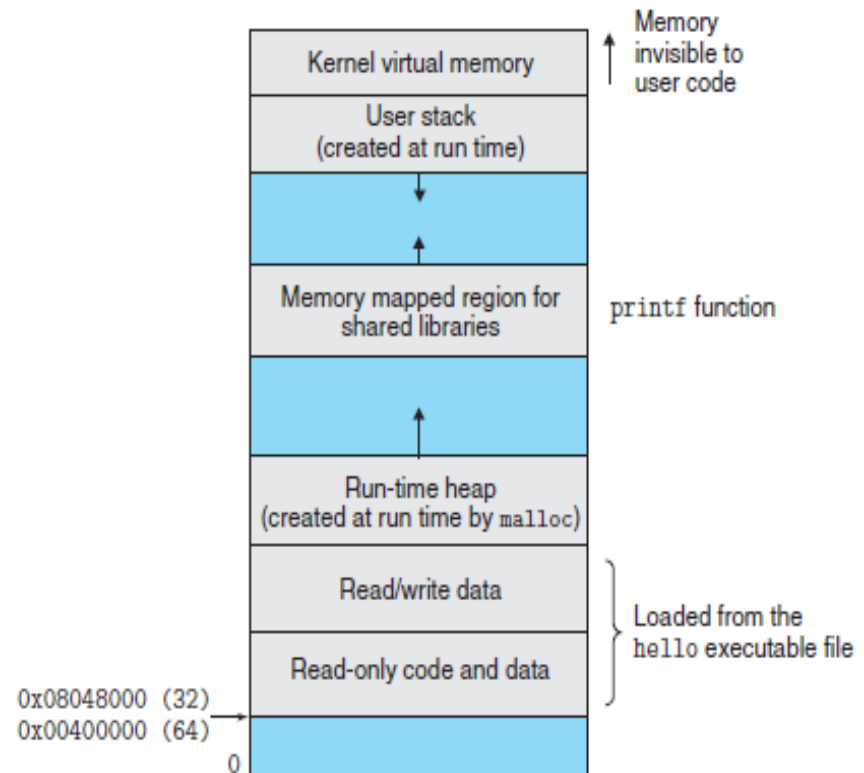
# Segmentation fault

- Segmentation fault เป็น error ที่นักเขียนโปรแกรมภาษา C ที่ใช้งาน pointer ทุกคนคุ้นเคย
- Segmentation fault เกิดจากที่ CPU พยายามจะเข้าอ่าน address ใน memory ที่ไม่ได้ถูกจองไว้สำหรับโปรแกรมนี้
- ส่วนใหญ่มักจะเกิดความผิดพลาดของ pointer

```
Segmentation fault (core dumped).
```

# Segmentation fault (ต่อ)

```
#include<stdio.h>
main(){
    int *a;
    a = (int *)0xffffffff;
    *a = 1;
}
```



# Let's review

สมมติให้ d เริ่มใช้ memory  
ตั้งแต่ address 1000

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

```
main(){
```

```
    int **a,**b,**c,*d;
```

```
    int e[5] = {1,2,3,4,5};
```

```
    *a = &e[3];
```

```
    b = e;
```

```
    c = (e+1);
```

```
    d = e;
```

```
    **a = 10;
```

```
    b[4] = 7;
```

```
    *(d+1) = 5;
```

```
    c++;
```

```
}
```

1	2	3	4	5
1000	1004	1008	1012	1016
e[0]	E[1]	E[2]	E[3]	E[4]

ตัวแปร	ค่า
a	-
*a	1012
**a	10

ตัวแปร	ค่า
b	1000
*b	1
**b	-

ตัวแปร	ค่า
c	1008
*c	3
**c	-

ตัวแปร	ค่า
d	1000
*d	1
**d	-

ตัวแปร	ค่า
e	1000
e[0]	1
e[1]	5

ตัวแปร	ค่า
e[2]	3
e[3]	7
e[4]	5



# Let's review (Endianness)

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

```
main(){
```

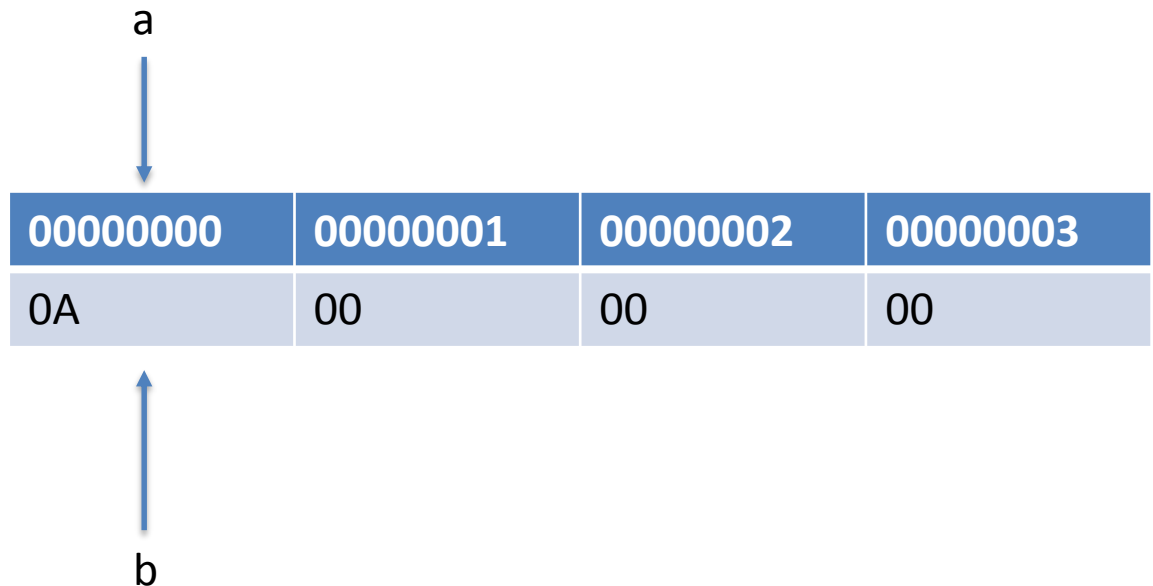
```
    int a = 10;
```

```
    char *b;
```

```
    b = (char *)a;
```

```
    printf("%d",*b);
```

```
}
```



# Main argument

- เมื่อเริ่มเรียกใช้ program ผู้ใช้งานสามารถส่ง argument ไปให้กับฟังก์ชัน main ได้ เพราะฟังก์ชัน main มีการรับ argument อยู่สองตัว คือ argv และ argc
- argv จะเป็น int ที่บอกจำนวน argument ที่ส่งมา
- argc จะเป็น array ของ string มีขนาดไม่แน่นอนขึ้นอยู่กับจำนวน argument ที่ส่งมา

# ตัวอย่างโปรแกรม echo

```
#include<stdio.h>

main(int argv, char *argc[]){

    int i = 0;

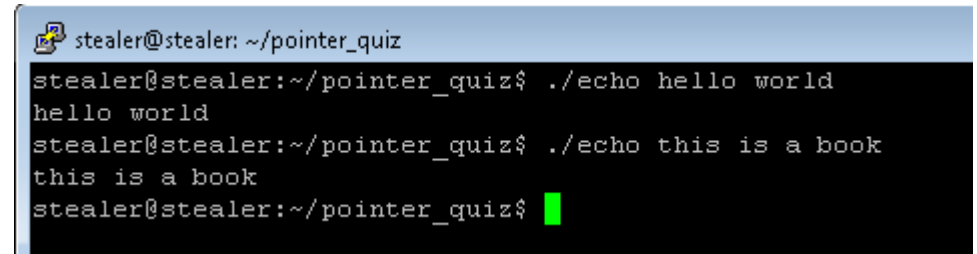
    for(i=1;i<argv;i++){

        printf("%s ",argc[i]);

    }

    printf("\n");

}
```

A terminal window with a blue title bar showing the command prompt 'stealer@stealer: ~/pointer\_quiz'. The user enters './echo hello world' and the output is 'hello world'. Then the user enters './echo this is a book' and the output is 'this is a book'. The prompt is followed by a green cursor.

```
stealer@stealer: ~/pointer_quiz
stealer@stealer:~/pointer_quiz$ ./echo hello world
hello world
stealer@stealer:~/pointer_quiz$ ./echo this is a book
this is a book
stealer@stealer:~/pointer_quiz$
```

Q&A