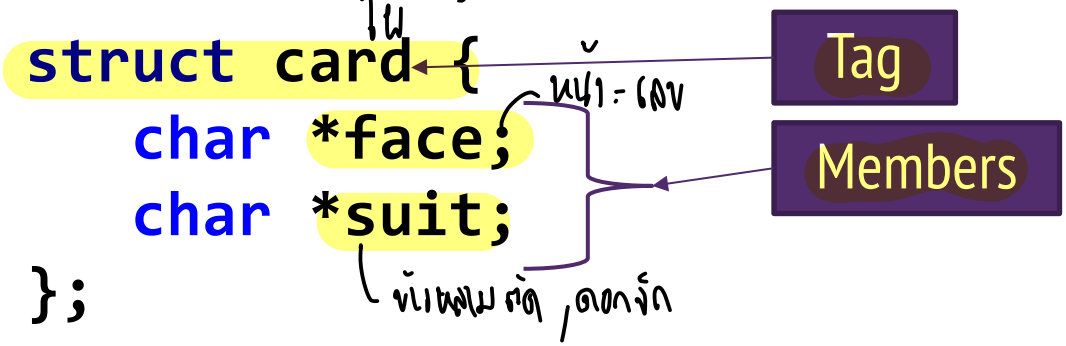
C Programming for Engineers

Structures, Unions

ICEN 360– Spring 2017

Prof. Dola Saha

Structure

- Collections of related variables under one name.
- Variables of may be of different data types.
- 

- Keyword struct introduces the structure definition.
- Members of the same structure type must have unique names, but two different structure types may contain members of the same name without conflict.

Structure Declaration

- ```
  struct employee {
      char firstName[20];
      char lastName[20];
      unsigned int age;
      char gender;
      double hourlySalary;
  };
  ```
  กำหนด struct
- ```
  struct employee employee1, employee2;
  ```
  ประกาศตัวแปร
- ```
  struct employee employees[100];
  ```
- ```
  struct employee {
      char firstName[20];
      char lastName[20];
      unsigned int age;
      char gender;
      double hourlySalary;
  } employee1, employee2, *employeePtr;
  ```
  ใช้ก็ได้แบบ type def

Structure Tag

- The structure tag name is optional.
- If a structure definition does not contain a structure tag name, variables of the structure type may be declared *only* in the structure definition—*not* in a separate declaration.

Self Reference

- *A structure cannot contain an instance of itself.*
- A variable of type `struct employee` cannot be declared in the definition for `struct employee`.
- A pointer to `struct employee`, may be included.
- For example,

```
struct employee2 {
    char firstName[20];
    char lastName[20];
    unsigned int age;
    char gender;
    double hourlySalary;
    struct employee2 person; // ERROR
    struct employee2 *ePtr; // pointer
};
```

ไม่ใช้ pointer

- `struct employee2` contains an instance of itself (person), which is an error.

Storage in Memory

- Structures may *not* be compared using operators == and !=, because ไม่สมาเก เปรียบเทียบได้
  - structure members are not necessarily stored in consecutive bytes of memory.
- Computers may store specific data types only on certain memory boundaries such as half-word, word or double-word boundaries.
- A word is a standard memory unit used to store data in a computer—usually 2 bytes or 4 bytes.

Storage in Memory

```
struct example {
    char c;
    int i;
} sample1, sample2;
```

Byte

0	1	2	3	4	5	6	7
01100001				00000000	00000000	00000000	01100001

Possible storage, but machine dependant

Initialization

```
struct card {
    char *face;
    char *suit;
};
```

```
struct card aCard = {"Three", "Hearts"};
```

- If there are fewer initializers in the list than members in the structure,
  - the remaining members are automatically initialized to 0
  - or NULL if the member is a pointer.
- Assignment Statement of same struct type
  - `struct card aCard1 = aCard2;`

Accessing Structure Members

- the structure member operator (.)—also called the dot operator
  - `printf("%s", aCard.suit); // displays Hearts`
- the structure pointer operator (->)—also called the arrow operator.
  - `cardPtr = &aCard;`
  - `printf("%s", cardPtr->suit); // displays Hearts`
  - Following are equivalent
    - `cardPtr->suit`
    - `(*cardPtr).suit`

Example

```
 4  #include <stdio.h>
 5
 6  // card structure definition
 7  struct card {
 8     char *face; // define pointer face
 9     char *suit; // define pointer suit
10  };
11
12  int main(void)
13  {
14     struct card aCard; // define one struct card variable
15
16     // place strings into aCard
17     aCard.face = "Ace";
18     aCard.suit = "Spades";
19
20     struct card *cardPtr = &aCard; // assign address of aCard to cardPtr
21
22     printf("%s%s%s\n%s%s%s\n%s%s%s\n", aCard.face, " of ", aCard.suit,
23        cardPtr->face, " of ", cardPtr->suit,
24        (*cardPtr).face, " of ", (*cardPtr).suit);
25  }
```

```
Ace of Spades
Ace of Spades
Ace of Spades
```

Structure with Function

- Structures may be passed to functions by
  - passing individual structure members
  - by passing an entire structure
  - by passing a pointer to a structure.
- Functions can return
  - individual structure members
  - an entire structure
  - a pointer to a structure

typedef

- The keyword `typedef` is a way to create synonyms (or aliases) for previously defined data types.
- Names for structure types are often defined with `typedef` to create shorter type names.
- Example:
  - `typedef struct card Card;`

    `Card` is a synonym for type `struct card`.
- Example:
  - ```
    typedef struct {
       char *face;
       char *suit;
    } Card;
    ```
  - `Card myCard, *myCardPtr, deck[52];`
    ```

Card Shuffling Example (1)

```
1  // Fig. 10.3: fig10_03.c
2  // Card shuffling and dealing program using structures
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6
7  #define CARDS 52
8  #define FACES 13
9
10 // card structure definition
11 struct card {
12    const char *face; // define pointer face
13    const char *suit; // define pointer suit
14 };
15
16 typedef struct card Card; // new type name for struct card
17
18 // prototypes
19 void fillDeck(Card * const wDeck, const char * wFace[],
20    const char * wSuit[]);
21 void shuffle(Card * const wDeck);
22 void deal(const Card * const wDeck);
23
```

Card Shuffling Example (2)

```
24  int main(void)
25  {
26     Card deck[CARDS]; // define array of Cards
27
28     // initialize array of pointers
29     const char *face[] = { "Ace", "Deuce", "Three", "Four", "Five",
30        "Six", "Seven", "Eight", "Nine", "Ten",
31        "Jack", "Queen", "King"};
32
33     // initialize array of pointers
34     const char *suit[] = { "Hearts", "Diamonds", "Clubs", "Spades"};
35
36     srand(time(NULL)); // randomize
37
38     fillDeck(deck, face, suit); // load the deck with Cards
39     shuffle(deck); // put Cards in random order
40     deal(deck); // deal all 52 Cards
41  }
42
```

Card Shuffling Example (3)

```
43  // place strings into Card structures
44  void fillDeck(Card * const wDeck, const char * wFace[],
45     const char * wSuit[])
46  {
47     // loop through wDeck
48     for (size_t i = 0; i < CARDS; ++i) {
49        wDeck[i].face = wFace[i % FACES];
50        wDeck[i].suit = wSuit[i / FACES];
51     }
52  }
53
54  // shuffle cards
55  void shuffle(Card * const wDeck)
56  {
57     // loop through wDeck randomly swapping Cards
58     for (size_t i = 0; i < CARDS; ++i) {
59        size_t j = rand() % CARDS;
60        Card temp = wDeck[i];
61        wDeck[i] = wDeck[j];
62        wDeck[j] = temp;
63     }
64  }
65
```

Card Shuffling Example (4)

```
66  // deal cards
67  void deal(const Card * const wDeck)
68  {
69     // loop through wDeck
70     for (size_t i = 0; i < CARDS; ++i) {
71        printf("%5s of %-8s%s", wDeck[i].face , wDeck[i].suit ,
72           (i + 1) % 4 ? "  " : "\n");
73     }
74  }
```

Card Shuffling Example (5)

```
Three of Hearts     Jack of Clubs     Three of Spades      Six of Diamonds
 Five of Hearts    Eight of Spades    Three of Clubs     Deuce of Spades
 Jack of Spades     Four of Hearts    Deuce of Hearts      Six of Clubs
Queen of Clubs    Three of Diamonds  Eight of Diamonds     King of Clubs
 King of Hearts    Eight of Hearts    Queen of Hearts    Seven of Clubs
Seven of Diamonds   Nine of Spades     Five of Clubs     Eight of Clubs
  Six of Hearts   Deuce of Diamonds    Five of Spades     Four of Clubs
Deuce of Clubs     Nine of Hearts    Seven of Hearts     Four of Spades
  Ten of Spades    King of Diamonds     Ten of Hearts     Jack of Diamonds
 Four of Diamonds   Six of Spades     Five of Diamonds     Ace of Diamonds
  Ace of Clubs     Jack of Hearts      Ten of Clubs     Queen of Diamonds
  Ace of Hearts     Ten of Diamonds    Nine of Clubs      King of Spades
  Ace of Spades    Nine of Diamonds   Seven of Spades    Queen of Spades
```

ไม่สามารถกำหนดค่าได้

Structure Example (preview)

- This declaration introduces the type struct fraction (both words are required) as a new type.
- C uses the period (.) to access the fields in a record.
- You can copy two records of the same type using a single assignment statement, however == does not work on structs (see note link).

```
struct fraction {
    int numerator;
    int denominator;          // can't initialize
};

struct fraction f1, f2;       // declare two fractions
f1.numerator = 25;
f1.denominator = 10;
f2 = f1;                      // this copies over the whole struct
```

Structure Declarations

ประกาศ แบบ มี ชื่อ หรือ ไม่มี ก็ได้

struct tag {member_list} variable_list;

```
struct S {
   int a;
   float b;
} x;
```

Declares x to be a structure having two members, a and b. In addition, the structure tag S is created for use in future declarations.

```
struct {
int a;
float b;
} z;
```

Omitting the tag field; cannot create any more variables with the same type as z

```
struct S {
int a;
float b;
};
```

Omitting the variable list defines the tag S for use in later declarations

```
struct S y;
```

Omitting the member list declares another structure variable y with the same type as x

ใช้ตอน struct อยู่ใน file อื่น

```
struct S;
```

Incomplete declaration which informs the compiler that S is a structure tag to be defined later

Structure Declarations (cont)

- So tag, member_list and variable_list are all optional, but cannot all be omitted; at least two must appear for a complete declaration.

```
struct  {
    int a;
    char b;
    float c;
} x;
```

```
struct  {
    int a;
    char b;
    float c;
} y[20], *z;
```

Single variable x contains 3 members

เป็น คน คือ อัน

Structs on the left are treated different by the compiler
DIFFERENT TYPES
i.e. z = &x is ILLEGAL

An array of 20 structures (y); and
A pointer to a structure of this type (z)

More Structure Declarations

- The TAG field
  - Allows a name to be given to the member list so that it can be referenced in subsequent declarations
  - Allows many declarations to use the same member list and thus create structures of the same type

```
struct  SIMPLE {
   int a;
   char b;
   float c;
} ;
```

Associates tag with member list; does not create any variables

So ➔ struct SIMPLE x;
struct SIMPLE y[20], *z;

Now x, y, and z are all the same kind of structure

Incomplete Declarations

- Structures that are mutually dependent
- As with self referential structures, at least one of the structures must refer to the other only through pointers
- So, which one gets declared first???

ประกาศเพื่อจะได้ A ใช้

```
struct B;

struct A  {
      struct  B   *partner;
      /* etc */
} ;

struct B  {
      struct  A    *partner;
      /* etc */
} ;
```

incomplete ต้องเป็น pointer

- **Declares an identifier to be a structure tag**
- **Use this tag in declarations where the size of the structure is not needed (pointer!)**
- **Needed in the member list of A**

- **Doesn't have to be a pointer**

ไม่จำเป็นต้องเป็น pointer

Initializing Structures

- Missing values cause the remaining members to get default initialization… whatever that might be!

```
typedef struct  {
    int       a;
    char      b;
    float     c;
}  Simple;

struct INIT_EX  {
    int       a;
    short     b[10];
    Simple  c;
}  x = {  10,
          { 1, 2, 3, 4, 5 },
          { 25, 'x', 1.9 }
};
```

What goes here (hint in blue below)?

```
struct INIT_EX   y = { 0 , {10,  20, 30, 40, 50,
                            60, 70, 80, 90, 100 },
                          { 1000, 'a', 3.14 }
                    } ;
```

Name all the variables and their initial values:

```
y.a = 0
y.b[0] = 10;  y.b[1] = 20; y.b[2] = 30; etc
y.c.a = 1000;
y.c.b = 'a';
y.c.c = 3.14;
```

Structures as Function arguments

- Legal to pass a structure to a function similar to any other variable but often inefficient

```
/* electronic cash register individual
transaction receipt */
#define PRODUCT_SIZE  20;
typedef struct  {
    char    product[PRODUCT_SIZE];
    int     qty;
    float   unit_price;
    float   total_amount;
}  Transaction;
```

- **Function call:**
  - print_receipt(current_trans);
  - Copy by value copies 32 bytes to the stack which can then be discarded later
- **Instead…**
  - (Transaction *trans)
  - trans->product // fyi: (*trans).product
  - trans->qty
  - trans->unit_price
  - trans->total_amount
  - print_receipt(&current_trans);
  - void print_receipt(Transaction *trans)

```
void print_receipt (Transaction trans)  {
  printf("%s\n, trans.product);
  printf(%d @ %.2f total %.2f\n", trans.qty, trans.unit_price, trans.total_amount);
}
```

Struct storage issues

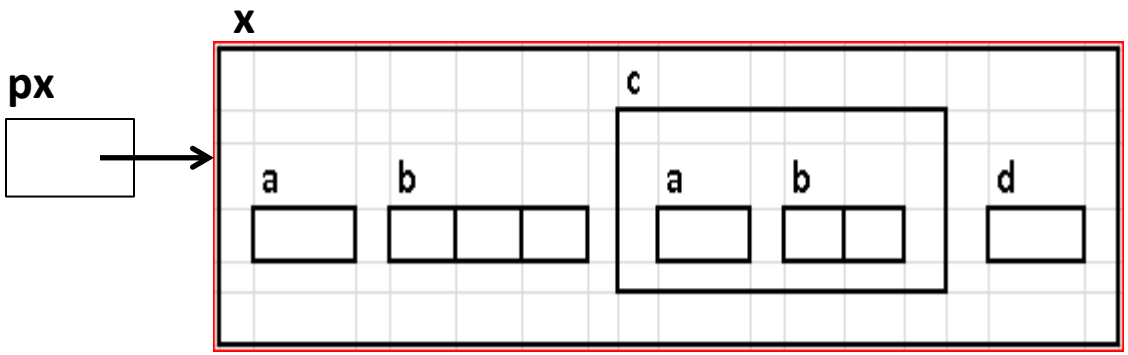
- A struct declaration consists of a list of fields, each of which can have any type. The total storage required for a struct object is the sum of the storage requirements of all the fields, plus any **internal padding**.

Structure memory (again)

- What does memory look like?

```
typedef struct  {
    int       a;
    short    b[2];
}  Ex2;

typedef struct EX  {
    int       a;
    char     b[3];
    Ex2       c;
    struct    EX  *d;
}  Ex;
```



Given the following declaration, fill in the above memory locations:

```
Ex  x = {  10, "Hi", { 5 , { -1, 25 }  } , 0 };
Ex *px = &x;
```

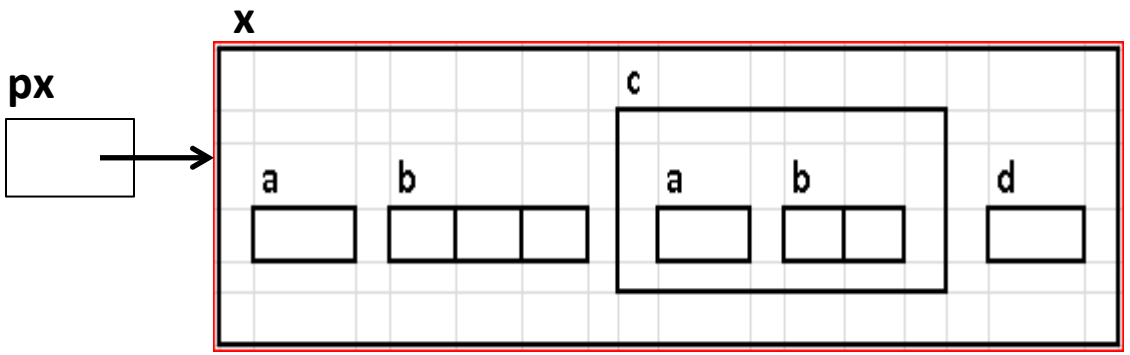
pre-condition = กำหนดค่าเริ่มต้น
post-condition = ให้ดูหมั่นใจว่าผลลัพธ์จะเป็นอะไร

Structure memory (again)

- What does memory look like?

```
typedef struct  {
    int       a;
    short    b[2];
}  Ex2;

typedef struct EX  {
    int       a;
    char      b[3];
    Ex2       c;
    struct    EX  *d;
}  Ex;
```

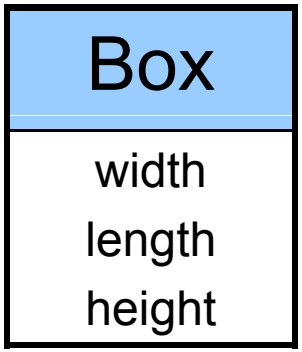


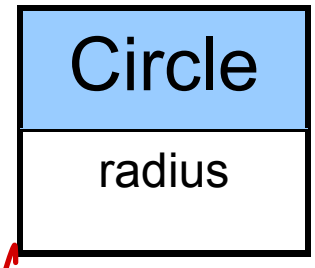
Given the following declaration, fill in the above memory locations:

```
Ex  x = {  10, “Hi”, { 5 , { -1, 25 }  } , 0 };
Ex *px = &x;
```

structs

Aggregating associated data into a single variable





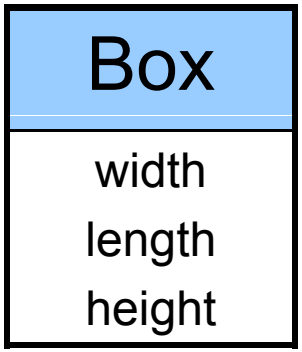

```
int main()
{
        Box mybox;
        Circle c;

        mybox.width = 10;
        mybox.length = 30;
        mybox.height = 10;
        c.radius = 10;
}
```

user-define type

abstraction = การดึงเฉพาะสิ่งสำคัญ รู้ว่าเป็นอะไร แต่ไม่ต้องรู้ว่าทำงานยังไง

The idea



I want to describe a box. I need variables for the width, length, and height.

ใช้ เป็นก้อน เดียวกัน

I can use three variables, but wouldn't it be better if I had a single variable to describe a box?

That variable can have three parts, the width, length, and height.

Another Example

```
struct bankRecordStruct
{
   char name[50];
   float balance;
};

struct bankRecordStruct billsAcc;
```

You can use mixed data types within the struct (int, float, char [])

Accessing values

```
struct bankRecordStruct
{
   char name[50];
   float balance;
};

struct bankRecordStruct billsAcc;

printf("My balance is: %f\n", billsAcc.balance);

float bal = billsAcc.balance;
```

Access values in a struct using a period: "."

Assign Values using Scanf()

```
struct BankRecord
{
    char name[50];
    float balance;
};

int main()
{
   struct BankRecord newAcc;  /* create new bank record */

   printf("Enter account name: ");
   scanf("%50s", newAcc.name);
   printf("Enter account balance: ");
   scanf("%d", &newAcc.balance);
}
```

Copy via =

You can set two struct type variables equal to each other and each element will be copied

```
struct Box { int width, length, height; };

int main()
{
    struct Box b, c;
    b.width = 5; b.length=1; b.height = 2;
    c = b;          // copies all elements of b to c
    printf("%d %d %d\n", c.width, c.length, c.height);
}
```

Passing Struct to a function

- You can pass a struct to a function. All the elements are copied
- If an element is a pointer, the pointer is copied **but not** what it points to!

```
int myFunction(struct Person p)
{
…
}
```

Using Structs in Functions

Write a program that

- Prompts the user to enter the dimensions of a 3D box and a circle
- Prints the volume of the box and area of the circle

Sample run:

```
Enter the box dimensions (width,length,height): 1 2 3
Enter the radius of the circle: 0.8

Box volume = 6
Circle area = 2.01
```

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

struct Box { int width, height , length; };

int GetVolume(struct Box b)
{
     return b.width * b.height * b.length;
}

int main()
{
     struct Box b;

     printf("Enter the box dimensions (width length height): ");
     scanf("%d %d %d", &b.width, &b.length, &b.height);

     printf("Box volume = %d\n", GetVolume(b));
}
```

ส่งเป็น struct ป้องกันการเปลี่ยนแปลงของข้อมูล

* ถ้าส่งเป็น pointer จะสามารถเปลี่ยนแปลงค่าได้

Note: == Comparison doesn't work

```
struct Box { int width, length, height; };

int main()
{
        struct Box b, c;
        b.width = 5; b.length=1; b.height = 2;
        c = b;
        if (c == b)        /* Error when you compile! */
                printf("c and b are identical\n");
        else
                printf("c and b are different\n");
} t
```

เปรียบเทียบ ไม่ได้

Error message: invalid operands to binary == (have 'Box' and 'Box')

Create your own equality test

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

struct Box { int width, height , length; };

int IsEqual(struct Box b, struct Box c)
{
   if (b.width==c.width &&
       b.length==c.length &&
       b.height==c.height)
       return 1;
   else
       return 0;
}
```

```
struct Box b, c;
b.width = 5; b.length=1; b.height = 2;
c = b;

if (IsEqual(b,c))
    printf("c and b are identical\n");
else
    printf("c and b are different\n");
```

Arrays of structs

You can declare an array of a structure and manipulate each one

```
typedef struct
{
  double radius;
  int x;
  int y;
  char name[10];
} Circle;

Circle circles[5];
```

Size of a Struct: sizeof

```
typedef struct
{
  double radius;      /* 8 bytes */
  int x;              /* 4 bytes */
  int y;              /* 4 bytes */
  char name[10];      /* 10 bytes */
} Circle;
```

↳ ต้องมองเป็น 12

```
printf("Size of Circle struct is %d\n",
        sizeof(Circle));
```

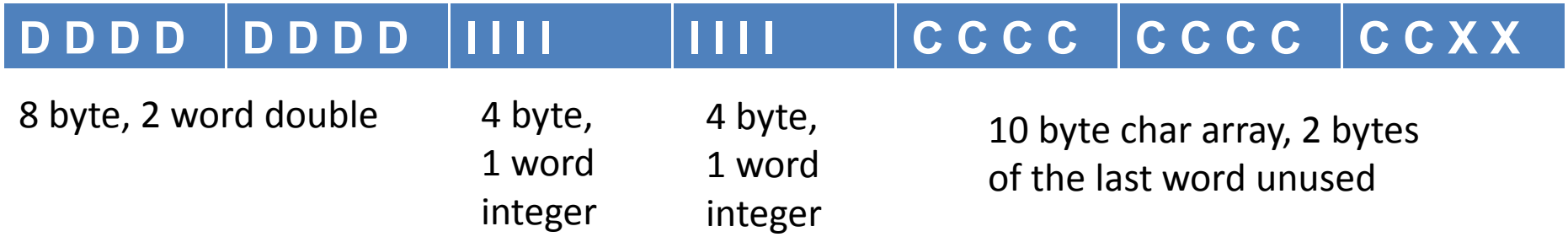
Size of a Struct

8 + 4 + 4 + 10 = 26

- But sizeof() reports 28 bytes!!!

Most machines require alignment on 4-byte boundary (a word)

- last word is not filled by the char (2 bytes used, 2 left over)



Arrays of Structures

- The converse of a structure with arrays:
- Example:

```
struct  entry {
   char  fname [10] ;
   char  lname [12] ;
   char phone [8] ;
} ;
struct entry list [1000];
```

ประกาศตัวแปร มี struct 1000

- This creates a list of 1000 identical entry(s).
- Assignments:

```
list [1] = list [6];
strcpy (list[1].phone, list[6].phone);
list[6].phone[1] = list[3].phone[4] ;
```

An Example

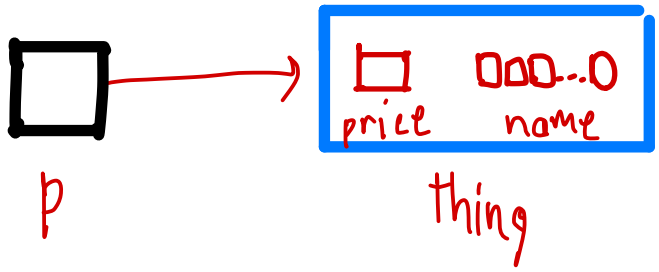
```
#include <stdio.h>
struct entry {
    char fname [20];
    char lname [20];
    char phone [10];
} ;
```

```
int main() {
   struct entry list[4]; → ประกาศเป็นอาร์เรย์
   int i;
   for (i=0; i < 4; i++) {
      printf ("\nEnter first name: ");
      scanf  ("%s", list[i].fname);
      printf ("Enter last name: ");
      scanf  ("%s", list[i].lname);
      printf ("Enter phone in 123-4567 format: ");
      scanf  ("%s", list[i].phone);
   }
   printf ("\n\n");
   for (i=0; i < 4; i++) {
      printf ("Name: %s %s", list[i].fname, list[i].lname);
      printf ("\t\tPhone: %s\n", list[i].phone);
   }
}
```

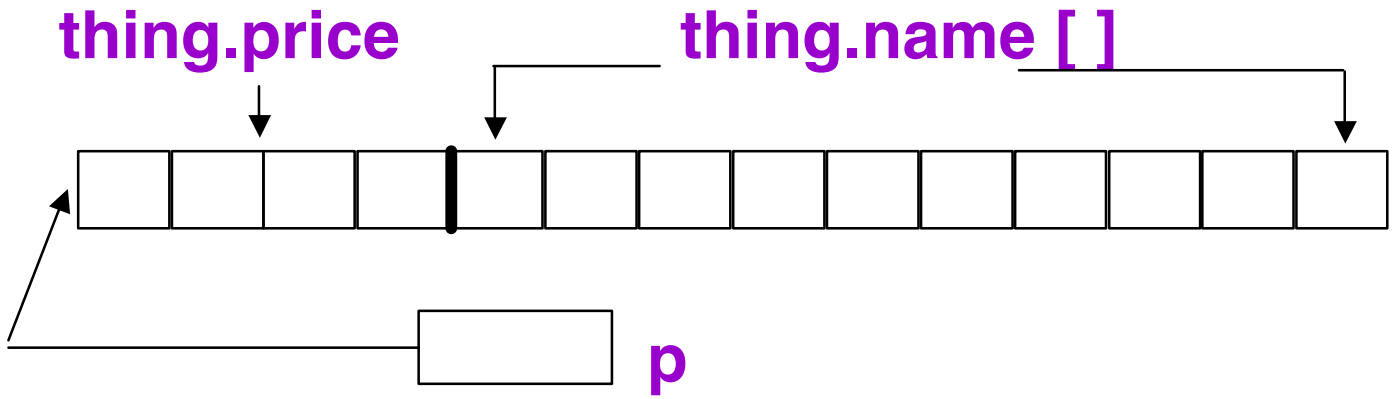
ใส่ Index ด้วย

Pointers to Structures

```
struct part {
   float price ;
   char name [10] ;
} ;
struct part *p , thing;
p = &thing;
/* The following three statements are equivalent *
thing.price = 50;
(*p).price = 50;   /* () around *p is needed */
p -> price = 50;
```

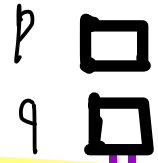


Pointers to Structures



- p is set to point to the first byte of the struct variable

Pointers to Structures




```
struct part * p, *q;
p = (struct part *) malloc( sizeof(struct part) );
q = (struct part *) malloc( sizeof(struct part) );
p -> price = 199.99 ;
strcpy( p -> name, "hard disk" );
(*q) = (*p);
q = p;
free(p);
free(q); /* This statement causes a problem !!!
            Why? */
```

Pointers to Structures

- You can allocate a structure array as well:

```
{
    struct part *ptr;
    ptr = (struct part *) malloc(10 * sizeof(struct part) );
    for( i=0; i< 10; i++)
    {
        ptr[ i ].price = 10.0 * i;
        sprintf( ptr[ i ].name, "part %d", i );
    }
    ……
    free(ptr);
}
```

จะมี 10 ก้อน เข้าถึง ได้ด้วย index

Pointers to Structures

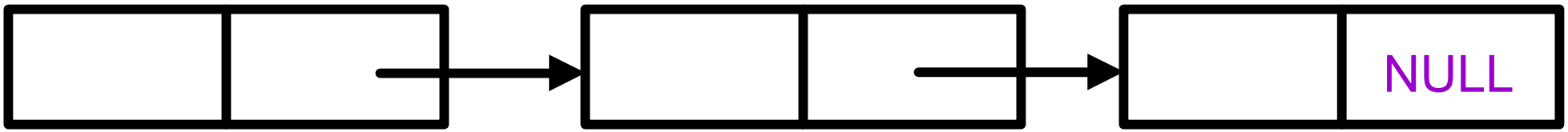
- You can use pointer arithmetic to access the elements of the array:

```
{
    struct part *ptr,  *p;
    ptr = (struct part *) malloc(10 * sizeof(struct part) );
    for( i=0, p=ptr; i< 10; i++, p++) ยก pointer ไปเรื่อยๆ
    {
          p -> price = 10.0 * i;
          sprintf( p -> name, "part %d", i );
    }
    ……
    free(ptr);
}
```

Pointer as Structure Member

```
struct node{
    int data;
    struct node *next;
};
struct node a,b,c;
a.next = &b;
b.next = &c;
c.next = NULL;
```

```
a.data = 1;
a.next->data = 2;
/* b.data =2 */
a.next->next->data = 3;
/* c.data = 3 */
c.next = (struct node *)
    malloc(sizeof(struct
    node));
……
```



Assignment Operator vs. memcpy

- This assign a struct to another

```
{
   struct part a,b;
   b.price = 39.99;
   b.name = "floppy";
   a = b;
}
```

- Equivalently, you can use memcpy

```
#include <string.h>
……
{
    struct part a,b;
    b.price = 39.99;
    b.name = "floppy";
    memcpy(&a,&b,sizeof(part));
}
```

copy จาก b ไป a ต้องบอกขนาดด้วย

Array Member vs. Pointer Member

```
struct book {
   float price;
   char name[50];
};
```

size of = 56

```
int main()
{
   struct book a,b;
   b.price = 19.99;
   strcpy(b.name, "C handbook");
   a = b;
   strcpy(b.name, "Unix
   handbook");
   puts(a.name);
   puts(b.name);
}
```

ชื่อ a จะไม่เปลี่ยนตาม b

Array Member vs. Pointer Member

```
struct book {
    float price;
    char *name;
};
```

ไม่ รู้เลย เท่าไร

```
int main()
{
    struct book a,b;
    b.price = 19.99;
    b.name = (char *) malloc(50);
    strcpy(b.name, "C handbook");
    a = b;
    strcpy(b.name, "Unix handbook");
    puts(a.name);
    puts(b.name);
    free(b.name);
}
```

b เปลี่ยน a เปลี่ยน

A function called strdup() will do the malloc() and strcpy() in one step for you!

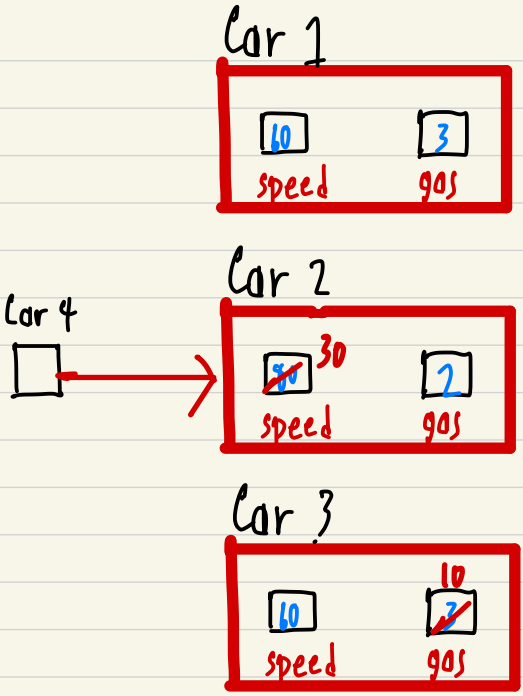
Passing Structures to Functions (1)

- Structures are passed by value to functions
  - The parameter variable is a local variable, which will be assigned by the value of the argument passed.
  - Unlike Java.
- This means that the structure is copied if it is passed as a parameter.
  - This can be inefficient if the structure is big.
    - In this case it may be more efficient to pass a pointer to the struct.
- A struct can also be returned from a function.

Passing Structures to Functions (2)

```
struct book {
    float price;
    char abstract[5000];
};
void print_abstract( struct
    book *p_book)
{
    puts( p_book->abstract );
};
```

```
struct pairInt {
    int min, max;
};
struct pairInt min_max(int x,int y)
{
    struct pairInt pair;
    pair.min = (x > y) ? y : x;
    pair.max = (x > y) ? x : y;
    return pairInt;
}
int main(){
    struct pairInt result;
    result = min_max( 3, 5 );
    printf("%d<=%d", result.min,
    result.max);
}
```



Car 1

speed = 60

gas = 3

Car 2

speed = 30

gas = 2

Car 3

speed = 60

gas = 10

Car 4

speed = 30

gas = 2