

# Structure data type

## ข้อมูลแบบโครงสร้าง

Declaring Structure Types

Creating Structure Variables

Using Structure Variables

Arrays Of Structures

Passing Structures To Function

Self-referential Structures

Linear Linked List

## การประกาศข้อมูลแบบโครงสร้าง (Declaring Structure Types)

- การประกาศข้อมูลแบบโครงสร้าง เป็นเพียงการประกาศหรือกำหนดโครงสร้างของข้อมูล เพื่อที่จะใช้เป็นชนิดของตัวแปร
- การประกาศข้อมูลแบบโครงสร้าง ทำได้โดยใช้คำเฉพาะ struct โดยมีรูปแบบดังนี้

```
struct ชื่อโครงสร้าง { รายละเอียดของสมาชิกแต่ละตัว };
```

- ตัวอย่าง

```
struct date {  
    int day;  
    int month;  
    int year;  
};  
struct student {  
    char id[8];  
    char name[25];  
    int mark;  
    char grade;  
};
```

## ข้อมูลแบบโครงสร้าง (structure data)

- ระเบียบข้อมูล (Record) ได้แก่ ฟิวด์หรือเซตข้อมูลตั้งแต่ 1 ฟิวด์ขึ้นไปที่มีความสัมพันธ์เกี่ยวข้งกันแล้วมีความหมายที่หมายถึงรายละเอียดของสิ่งที่น่าสนใจ เช่น ชื่อ เลขประจำตัว ยอดขายของพนักงาน 1 คน รวมกันเป็นข้อมูลของพนักงานขาย 1 ระเบียบ หรือ 1 เรคอร์ด
- ข้อมูลแบบโครงสร้างหมายถึง กลุ่มของข้อมูลที่มีความสัมพันธ์กัน โดยข้อมูลเหล่านี้อาจมีชนิดข้อมูลที่แตกต่างกัน มีชื่อที่ใช้อ้างถึงข้อมูลแต่ละตัวต่างกัน แต่ข้อมูลกลุ่มนี้จะอยู่ภายใต้ชื่อกลุ่มเดียวกัน
- สมาชิกหรือข้อมูลแต่ละตัวในโครงสร้างหนึ่งๆ เปรียบเสมือน เซตข้อมูลในแต่ละระเบียบ นั่นเอง

## การสร้างตัวแปรข้อมูลแบบโครงสร้าง (Creating Structure Variables)

- โครงร่างของข้อมูลที่ประกาศ ทำหน้าที่เสมือนเป็น ชนิดข้อมูล
- การประกาศหรือสร้างตัวแปรที่มีชนิดข้อมูลเป็นแบบโครงสร้าง มีรูปแบบดังนี้
  - struct ชื่อโครงสร้าง รายชื่อตัวแปร;
  - struct ชื่อโครงสร้าง {รายละเอียดของสมาชิกแต่ละตัว} รายชื่อตัวแปร;
  - struct ชื่อโครงสร้าง ตัวแปร = {ค่าคงที่ ที่มีชนิดข้อมูลตรงกับชนิดของแต่ละเซตข้อมูล};
- ตัวอย่าง

```
struct date today, tomorrow;  
struct card {  
    int pips;  
    char suits;  
} x;  
struct date yesterday = {18, 1, 2553};
```

# การใช้ตัวแปรข้อมูลแบบโครงสร้าง

5

## (Using Structure Variables)

- การอ้างถึงข้อมูลของตัวแปรแบบโครงสร้าง ต้องอ้างให้ถึงค่าของสมาชิกในโครงสร้างหรือค่าของแต่ละเขตข้อมูล
- โดยใช้ตัวดำเนินการ . (dot)

*structure variable . member name*

*ชื่อตัวแปรแบบโครงสร้าง. ชื่อฟิลด์หรือชื่อเขตข้อมูล*

```

struct card {
    int    pips;
    char   suits;
} x;

x.pips = 8;
x.suits = 'D'; /* สมมติให้ไฟหน้าข้าวหลามตัด แทนด้วยตัว D */
printf("x.pips = %d\nx.suits = %c\n", x.pips, x.suits );

```



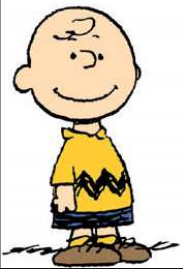
7

จงเขียนโปรแกรมรับรายละเอียดของนักศึกษา ซึ่งประกอบไปด้วย

- รหัส
- ชื่อ-สกุล
- คะแนน

แล้วแสดงรายละเอียดทั้งหมด พร้อมเกรดที่ควรจะได้ จากคะแนนที่ป้อน

ถ้าคะแนนที่ได้ < 50	ได้เกรด F
50-60	D
61-74	C
75-84	B
85 up	A



7

# ตัวอย่างการอ้างถึงหรือแสดงค่าตัวแปรแบบโครงสร้าง

6

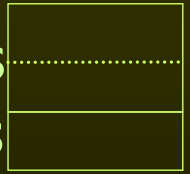
```

struct card {
    int    pips;
    char   suits;
} x, y;

x.pips = 4;
puts("Please enter your desired suits");
scanf("%c", &x.suits);
printf("your card is %c\n", x.suits);

y.pips = x.pips + 8;
y.suits = x.suits + 3;
printf("y = { %d, %c } ", y.pips, y.suits);

```



```

#include <stdio.h>
#include <conio.h>
struct student {
    char id[8];
    char name[25];
    int mark;
    char grade;
};

void main () {
    struct student stu;
    stu.grade = ' '; /* กำหนดค่าเริ่มต้นให้ฟิลด์เกรดเป็นช่องว่าง */
    puts("enter student information : ");
    printf ("Student ID "); gets (stu.id);
    printf ("      Name : "); gets(stu.name);
    printf ("      mark : "); scanf("%d", &stu.mark);
    stu.grade =
        stu.mark<50 ? 'F' : (stu.mark<61? 'D' : (stu.mark<75? 'C' :
(stu.mark<85?'B' : 'A')));
}

```



8

## การใช้พอยน์เตอร์ของข้อมูลแบบโครงสร้าง

- การประกาศตัวแปรพอยน์เตอร์ของข้อมูลแบบโครงสร้าง ทำได้ด้วยตนเองเดียวกับการประกาศพอยน์เตอร์ของข้อมูลพื้นฐาน

- struct tag-name \* pointer variable ;**

- การอ้างถึงข้อมูลสมาชิกของโครงสร้างเมื่อใช้พอยน์เตอร์ ทำได้ 2 วิธี
  - โดยใช้ตัวดำเนินการ ->

*Pointer structure variable -> member name*

*ชื่อตัวแปรพอยน์เตอร์ของโครงสร้าง -> ชื่อฟิลด์หรือชื่อเขตข้อมูล*

- ✓ โดยใช้ ตัวดำเนินการ \*

*(\* Pointer structure variable) . member name*

*(\* ชื่อตัวแปรพอยน์เตอร์ของโครงสร้าง) . ชื่อฟิลด์หรือชื่อเขตข้อมูล*

- ✓ **struct card x, \*p = &x;**

p->pips = 8;

(\*p).suits = 'D'; /\* สมมติให้ไพ่หน้าชาวหลามตัด แทนด้วยตัว D \*/

printf("x.pips = %d\nx.suits = %c\n", (\*p).pips, p->suits );

## การส่งผ่านค่าข้อมูลแบบโครงสร้างให้กับฟังก์ชัน

- ♦ **Call by value:** พารามิเตอร์ของฟังก์ชันอาจจะเป็นข้อมูลเพียงเขต (field) เดียว หรือเป็นข้อมูลทั้งระเบียน ก็ได้

```
void display_stu (struct student x) {
    printf("Student Id : %s\n", x.id);
    printf("    name : %s\n", x.name);
    printf("  birthdate : %0d/%0d/%0d\n", x.birthdate.day,
        x.birthdate.month, x.birthdate.year);
    printf("registration : %0d/%0d/%0d\n", x.registration.day,
        x.registration.month, x.registration.year);
    printf(" grade : %.2f\n", x.grade);
}
:
struct student stu;
display_stu(stu);
```

## Example

```
struct student {
    char id[8];
    char name[25];
    int mark;
    char grade;
};

:
struct student temp, *p=&temp;
temp.grade = 'A';
strcpy(temp.name, "W. Jumpa");
strcpy(temp.id, "2305171");
```

Expression	Equivalent exp.	Conceptual Value
temp.grade	p->grade (*p).grade	char
temp.name		char *
temp.id		
(*p).id		

## การส่งผ่านค่าข้อมูลแบบโครงสร้างให้กับฟังก์ชัน

- ♦ **Call by reference**

```
void input_stu (struct student *x) {
    gets(x->id);
    gets(x->name);
    scanf("%d/%d/%d", &x->birthdate.day,
        &x->birthdate.month, &x->birthdate.year);
    scanf("%d/%d/%d", &x->registration.day,
        &x->registration.month, &x->registration.year);
    x->grade = 0.0;
}

void main () {
    struct student stu;
    input_stu(&stu);
    display_stu(stu);
}
```

## แถวลำดับของข้อมูลแบบโครงสร้าง

- การประกาศตัวแปรชุดที่มีสมาชิกเป็นข้อมูลแบบโครงสร้าง ทำได้ด้วยทำนองเดียวกับการประกาศตัวแปรชุดของข้อมูลพื้นฐาน

```
struct card c, deck[52];
```

```
#define heart 'A'
#define diamond 'D'
```

```
:
deck[0].pips = 1;
deck[0].suits = diamond;
```

```
for (i = heart; i <= diamond; i++)
  for (j = 0; j < 13; j++) {
    deck[(i-heart)*13 + j].pips = j+1;
    deck[(i-heart)*13 + j].suits = i;
  }
```

## การอ้างถึงข้อมูลแบบโครงสร้างที่กำลังประกาศ

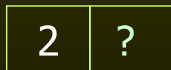
### Self-referential structures

```
struct node {
    int data;
    struct node * next;
} x, y, z;
```

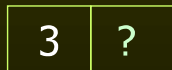
```
x.data = 1;      y.data = 2;      z.data = 3;
```



x



y



z

```
x.next = &y;      y.next = &z;      z.next = NULL;
```



x



y



z

## โครงสร้างของข้อมูลแบบโครงสร้าง

- สมาชิกของข้อมูลแบบโครงสร้าง อาจมีชนิดข้อมูลเป็นแบบโครงสร้างได้

```
struct date {
    int day;
    int month;
    int year;
};

struct student {
    char id[8];
    char name[25];
    struct date birthdate;
    struct date registration;
    float gpa;
} stu, temp;

stu.birthdate.day = 14;
stu.birthdate.month = 12;
scanf("%d/%d/%d", &stu.registration.day, &stu.registration.month,
    &stu.registration.year);
```

## Linear Linked List

- Dynamic allocation

- malloc()

- ฟังก์ชันนี้ ทำหน้าที่จัดสรรเนื้อที่ในหน่วยความจำที่ละบล็อคให้กับตัวแปร

- Include : <alloc.h> และ <stdlib.h>

- Prototype : void \* malloc( size\_t );

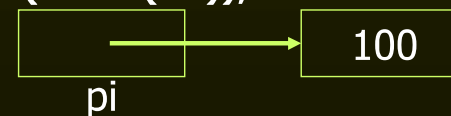
- Argument : จำนวนไบต์ที่ต้องการรับการจัดสรร

- Return : a pointer to a void if successful, Null if not.

- int \*pi;

```
pi = malloc(sizeof(int));
```

```
*pi = 100;
```



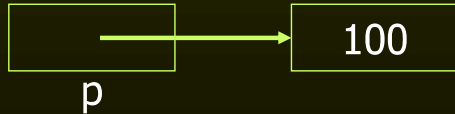
## Linear Linked List (cont.)

- **Dynamic release**
- free()
- ทำหน้าที่คืนเนื้อที่ในหน่วยความจำที่ถูกจัดสรรโดย malloc() ให้กับระบบ
- Include : <alloc.h> และ <stdlib.h>
- Prototype : void \* free( void \* );
- Argument : พอยน์เตอร์ของเนื้อที่ที่ถูกจัดสรรโดย malloc()
- Return : void ไม่มีการส่งค่ากลับ

• **int \*pi;**

**pi = malloc(sizeof(int));**

**\*pi = 100;**



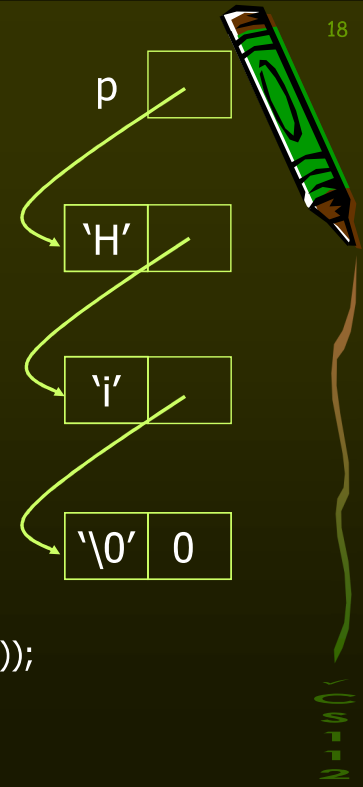
**free(pi);**

17

## ตัวอย่าง สร้างลิงค์ลิสต์

```
#include <stdio.h>
#include <alloc.h>
#include <stdlib.h>
}
struct node {
    char data;
    struct node * next;
};
void main () {
    struct node *p;

    p = malloc(sizeof(struct node));
    p->data = 'H';
    p->next = malloc(sizeof(struct node));
    p->next->data = 'i';
    p->next->next = malloc(sizeof(struct node));
    p->next->next->data = '\0';
    p->next->next->next = NULL;
}
```



18

## Renaming type name with typedef

- typedef ชื่อชนิดข้อมูลเดิม ชื่อชนิดข้อมูลใหม่

19

## Unions

- **union** หมายถึง กลุ่มของตัวแปรต่างชนิดกัน แต่ใช้เนื้อที่หน่วยความจำเดียวกัน
- ตัวแปรที่เป็น **union** จะสามารถเก็บค่าข้อมูลต่างชนิดกันในพื้นที่หน่วยความจำเดียวกัน
- โครงสร้างแบบนี้ต้องระบุหรือประกาศไว้ก่อนใช้ เพื่อที่ตัวแปลภาษาจะได้ทราบว่า พื้นที่นั้นจะเก็บข้อมูลชนิดใดบ้าง
- รูปแบบการประกาศ  
`union tag-name { share space member list };`
- ตัวอย่าง  

```
union u_sample {
    char c_value;
    int i_value;
    long l_value;
};
```

20

# การประกาศตัวแปรให้เป็น union

- รูปแบบการประกาศ

```
union tag-name variable-list;
```

- ตัวอย่าง

```
union u_sample x;
```

- การประกาศตัวแปรพร้อมระบุการเป็น union

```
union tag-name {share space member list} variable-list;
```

```
union u_sample {
```

```
    char c_value;
    int  i_value;
    long l_value;
```

```
} x, u;
```

- การใช้เนื้อที่ในหน่วยความจำ คอมไพเลอร์จะสำรองไว้เท่าขนาดของข้อมูลที่ใหญ่ที่สุด

## ตัวอย่าง

- การใช้ union พบได้บ่อยๆ ได้แก่การให้ union เป็น สมาชิกหนึ่งในข้อมูลแบบโครงสร้าง เช่น

```
struct tab {
```

```
    char *name;
    int  type;
    union {
        int    i;
        float  f;
        char   c;
```

```
    } data;
```

```
} table;
```

```
table.data.c = '0';
```

```
printf("table.data.i = %d\n", table.data.i);
```

# การอ้างถึงหรือเรียกใช้ตัวแปรแบบ union

- รูปแบบ

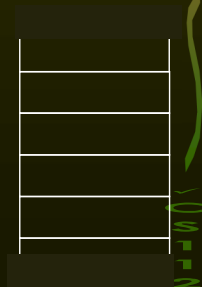
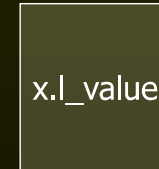
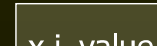
```
union-variabl . member-name
```

- ตัวอย่าง

```
union u_sample {
```

```
    char c_value;
    int  i_value;
    long l_value;
```

```
} x, u;
```



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

```
void main() {
```

```
    union share {
        char ch;
        int  i;
        char str[10];
    } var;
```

```
    printf("Size of union var is %d\n", sizeof(var));
```

```
    var.ch = '#'; printf("%c ", var.ch);
```

```
    var.i = 57; putchar(var.ch);
```

```
    puts(var.str);
```

```
}
```

ผลจากการรันโปรแกรมจะได้

Size of union var is 10

# 99

← cursor