บทที่ 5
ตัวแปรกลุ่ม
(Organinzed Variables)

การรวมกลุ่มตัวแปรสามารถกระทำได้ 2 วิธีคือ รวมแบบอะเรย์ (array) กรณีตัวแปรเหล่านั้นเป็นตัวแปรแบบเดียวกันกับรวมแบบโครงสร้างตัวแปร (structure) กรณีตัวแปรเหล่านั้นเป็นตัวแปรต่างแบบกัน

5.1 อะเรย์

การรวมตัวแปรแบบเดียวกันหลาย ๆ ตัวเป็นกลุ่มที่ใช้ชื่อร่วมกันหรืออะเรย์มีรูปไวยากรณ์ดังนี้คือ

```
type variables-name [size] ;
```

คำว่า type หมายถึงชนิดของตัวแปรอาจเป็น int, char, float, double หรือแบบใดก็ได้ variables-name คือชื่อตัวแปรขณะที่ size หมายถึงขนาดของอะเรย์หรือจำนวนตัวแปรที่มารวมกันเป็นอะเรย์ ที่จริง size ก็คือ dimension ข้อที่พึงสังเกตก็คือ ดัชนี (index) หรือเลขที่ของตัวแปรจะเริ่มที่เลข 0 เช่น ถ้าเราระบุอะเรย์เป็น

```
int iarr[5] ;
```

แสดงว่าเราต้องการรวมตัวแปรแบบ int ชื่อ iarr หมายเลข 0 คือ iarr[0] หมายเลข 1 คือ iarr[1] หมายเลข 2 คือ iarr[2] หมายเลข 3 คือ iarr[3] และหมายเลข 4 คือ iarr[4] ไว้เป็นกลุ่มก้อนเดียวกัน

สำหรับการกำหนดค่าเริ่มต้นของแต่ละตัวแปรในอะเรย์โดยเฉพาะเมื่อเป็นอะเรย์ของตัวแปรภายนอก เราก็สามารถกระทำได้ โดยเพิ่มข้อความลงไปในรูปไวยากรณ์เดิม ซึ่งอาจทำได้ 2 วิธี เช่น ถ้าเรากำหนดให้ตัวแปร iarr เป็นตัวแปร static แบบ int ซึ่งมีจำนวนทั้งสิ้น 5 ตัวโดยที่ iarr[0] มีค่าเริ่มต้นเป็น 5 iarr[1] มีค่าเริ่มต้นเป็น 6 iarr[2] มีค่าเริ่มต้นเป็น 7 iarr[3] มีค่าเริ่มต้นเป็น 8 และ iarr[4] มีค่าเริ่มต้นเป็น 9 เราจะพบรูปไวยากรณ์ของอะเรย์นี้ดังนี้คือ

```
1.  static int iarr[5] = { 5, 6, 7, 8, 9} หรือ
2.  static int iarr[ ] = { 5, 6, 7, 8, 9 }
```

ขอให้สังเกตว่าในวิธีที่ 2 นั้นเราไม่ระบุขนาดของอะเรย์ หรือจำนวนตัวแปร iarr ซึ่งก็ไม่ถือว่าผิดไวยากรณ์ เพราะรูปนี้คอมไพเลอร์จะรู้เองและสามารถทราบขนาด (size) ของอะเรย์ได้ด้วยการนับจำนวนค่าเริ่มต้นเมื่อนับจำนวนค่าเริ่มต้นได้เท่าไรก็จะเตรียม address ไว้ให้แก่แต่ละตัวแปร ซึ่งก็มีผลตรงกันกับวิธีที่ 1 ที่เรากำหนดขนาดของอะเรย์ไว้ก่อน เพื่อแจ้งให้คอมไพเลอร์ทราบเพื่อจะได้เตรียม address ไว้ให้ตัวแปรต่อไป

ตัวอย่างอะเรย์ในภาษาอื่นปรากฏดังนี้

BASIC

```
DIM IARR(5)                                    definition
IARR(1)                                        reference
. . .
IARR(5)
```

FORTRAN

```
DIMENSION IARR(5)                              definition
IARR(1)                                        reference
. . .
IARR(5)
```

PASCAL

```
VAR
     IARR: ARRAY[0..4] OF INTEGER;             definition
     IARR[0]                                   reference
     . . .
     IARR[4]
```

PL/I

```
DECLARE IARR(0:4) FIXED BIN(15,0);             definition
IARR(0)                                        reference
. . .
IARR(4)
```

COBOL

```
01 TABLE-OF-INT                                definition
     05 IARR PICTURE 99999 OCCURS 5 TIMES.
IARR(1)                                        reference
. . .
IARR(5)
```

Element	Value
iarr[2]	**31**
iarr[3]	**18**
iarr[4]	**22**

นอกจากการจัดกลุ่มตัวแปรดังตัวอย่างที่ผ่านมาแล้ว เรายังสามารถจัดกลุ่มตัวแปรแบบเดียวกัน แต่มีอยู่หลายกลุ่มไว้ในอะเรย์เดียวกันได้ โดยเพิ่มตัวเลขแสดงจำนวนกลุ่มเข้าไปในรูปไวยากรณ์เดิม เช่น **int** ia [5][7] หรือ **static int** ia [5][7] ซึ่ง

ซึ่งแสดงว่าเรารวมตัวแปรแบบ int ชื่อ ia[0], ia[1], ..., ia[6]รวม 5 กลุ่มเข้าด้วยกันทั้งนี้ [7] หมายถึงขนาดของกลุ่มขณะที่ [5] หมายถึงจำนวนกลุ่ม และการกำหนดค่าเริ่มต้นก็ยังคงใช้รูปไวยากรณ์คล้ายกับรูปที่มีตัวแปรกลุ่มเดียว เช่น

```
static int ia[6][5]={
    {1,2,3,4,5},
    {6,7,8,9,10},
    {11,12,13,14,15},
    {16,17,18,19,20},
    {21,22,23,24,25},
    {26,27,28,29,30}
    };
```

แสดงว่าเรารวมกลุ่มตัวแปรภายนอกชนิด static ที่เป็นตัวแปรแบบ int ชื่อ ia จำนวน 6 กลุ่ม ๆ ละ 5 ตัว คือ ia[0], ..., ia[4] ไว้ด้วยกัน โดยมีค่าเริ่มต้นต่างกัน ซึ่งคอมไพเลอร์จะจัดแอดเดรสและค่าเริ่มต้นในส่วนความจำของแต่ละตัวแปรไว้ดังภาพ ขอให้สังเกตว่า

ia[1][0] หมายถึงตัวแปร ia[0] ของแถว (กลุ่ม)ที่ 2 มีค่าเริ่มต้นเท่ากับ 6

ia[1] [4] หมายถึงตัวแปร ia[4] ของแถว (กลุ่ม)ที่ 2 มีค่าเริ่มต้นเท่ากับ **10**

ia[5] [0] หมายถึงตัวแปร ia[0] ของแถวที่ 6 มีค่าเริ่มต้นเท่ากับ 26

ia[5][4] หมายถึงตัวแปร ia[4] ของแถวที่ 6 มีค่าเริ่มต้นเท่ากับ **30**

รายละเอียดเกี่ยวกับแอดเดรสและมูลค่าเริ่มต้นในส่วนความจำปรากฏดังนี้

	ADDRESS	VALUE IN MEMORY
ia[0][0]	1000	1
ia[0][1]	1002	2
ia[0][2]	1003	3
ia[0][3]	1004	4
ia[0][4]	1005	5
ia[1][0]	1006	6
. . .		
ia[2][0]	1010	11
. . .		
ia[3][0]	1015	16
. . .		
ia[4][0]	1020	21
. . .		
ia[5][0]	1025	26
. . .		
ia[5][4]	1029	30

ที่จริงแล้วจุดโคออร์ดิเนต หรือตำแหน่งของสมาชิกในอะเรย์ในภาษา C ก็มีลักษณะเช่นเดียวกันกับในภาษาอื่น เพียงแต่ดัชนี หรือเลขที่เริ่มจาก 0 เป็นต้นไป มิใช่เริ่มจาก 1 เช่น - ia [5][4] เราอาจอ่านในรูปอะเรย์ได้เป็นสมาชิกของอะเรย์ชื่อ ia ณ แถวที่ 6 สดมภ์ที่ 5 มีค่าเท่ากับ 30 ดังนี้เป็นต้น

สำหรับอะเรย์ของตัวแปรภายนอกชนิด static ของ character ซึ่งทำหน้าที่แบบเดียวกันกับ string constant ก็มีรูปไวยากรณ์ รวมทั้งการกำหนดค่าเริ่มต้นเช่นเดียวกัน เช่น string constant คือ

```
static char carr [ 1 = "ABCD" ;
```

แสดงว่าเรากำหนดให้ carrเป็นอะเรย์ของตัวแปร static โดยตัวแปรหนึ่ง ๆ ของอะ-เรย์นี้เป็น character variable มีค่าเริ่มต้นดังนี้

carr [0]มีค่าเริ่มต้นเท่ากับ 'A' ซึ่งมีค่าตามรหัสแอสกี้เท่ากับ 65
carr [1] มีค่าเริ่มต้นเท่ากับ 'B' ซึ่งมีค่าตามรหัสแอสกี้เท่ากับ 66
carr [2] มีค่าเริ่มต้นเท่ากับ 'C' ซึ่งมีค่าตามรหัสแอสกี้เท่ากับ 67
carr [3] มีค่าเริ่มต้นเท่ากับ 'D' ซึ่งมีค่าตามรหัสแอสกี้เท่ากับ 68

หากเราเขียน string constant ข้างต้นเป็นรูปอะเรย์พร้อมค่าเริ่มต้น จะปรากฏไวยา-กรณ์ดังนี้คือ

```
static char carr [5] = { 'A', 'B', 'C', 'D', '\0' };
```

'\0' หมายถึง NUL characterใช้สำหรับปิดท้ายเพื่อแสดงว่าจบสตริงแล้ว ภาพแสดงแอดเดรสและค่าเริ่มต้นปรากฏดังนี้

Diagram 5.1 MEMORY LAYOUT OF CARR

ELEMENT	ADDRESS	VALUE	ASCII VALUE IN MEMORY
carr[0]	1000	'A'	65
carr[1]	1001	'B'	66
carr[2]	1002	'C'	67
carr[3]	1003	'D'	68
carr[4]	1004	'\0'	0

ขอให้สังเกตว่าในภาษา Cจะไม่ระบุจำนวนcharacterในสตริง แต่จะใช้ NUL character เป็นตัวบ่งชี้ว่า สตริงจบลงที่ใด หมายความว่าเมื่อคอมไพเลอร์พบ '\0' ก็แปลว่าสตริงมีความยาวถึงอักขระสุดท้ายก่อน NUL character อักขระที่พ้น NUL character ไปแม้

ว่าจะมีการระบุขนาดของอะเรย์ไว้ หรือไม่ก็ตามให้ถือว่ามีค่าเป็น 0 (ถ้าหากว่าสตริงยาวเกิน 1 บรรทัด ถ้าจะต่อบรรทัดให้ปิดท้ายบรรทัดเก่าด้วย\ (เรียกว่า backslash) แล้วกด ↵) เช่น

```
static char carr[8] = "ABCDE";
```

แสดงว่าเรากำหนดค่าเริ่มต้นให้แก่ตัวแปรในอะเรย์ carrไว้เพียง 5 ตัว ที่เหลืออีก 3 ตัว มิได้กำหนดค่าเริ่มต้นไว้ให้ ดังนั้นตัวแปร carr[5] carr[6] และ carr[7] จึงมีค่าเริ่มต้นเท่ากับ 0 ซึ่งการกำหนดค่าเริ่มต้นตามแบบ string constant เช่นนี้ จะมีผลเดียวกับการกำหนดอะเรย์ดังนี้

```
static char carr[8]={'A','B','C','D','E',',0',0, 0}
```

การรับ-ส่งอะเรย์ไปมาระหว่างโปรแกรมเรียก (อาจหมายถึง call function, call expression, call routine) กับฟังก์ชันนั้นเราสามารถใช้ชื่ออะเรย์เป็นอาร์กิวเมนต์ในการเรียกฟังก์ชั่น เพราะมีผลเสมือน main ( )หรือโปรแกรมเรียกส่งแอดเดรสแรกสุดของอะเรย์ไปยังฟังก์ชัน (เรียกว่า call by reference) ซึ่งในกรณีเช่นนี้ อาร์กิวเมนต์ของฟังก์ชั่นจึงต้องจัดอาร์กิวเมนต์ของฟังก์ชั่นเองไว้ในรูปอะเรย์ หรือ pointer เช่นถ้ากำหนดให้อาร์กิวเมนต์ของฟังก์ชั่นเป็นอะเรย์จะพบว่า (ดูตัวอย่าง)

```
func (abc)
char abc[ 1 ;
   {
   . . .
    }
```

หมายความว่าฟังก์ชั่นชื่อ func มีอาร์กิวเมนต์ชื่อ abc และเรากำหนดให้อาร์กิวเมนต์ abc

เป็นอะเรย์ของตัวแปรแบบ char ขอให้สังเกตขนาดของอะเรย์ abc[ ] ว่าเราปล่อยว่างมิได้กำหนดค่าของขนาดเป็นตัวเลข

หรือ
```
func (abcd)
int abcd [ ][5] ;
    {
    ....
    }
```

หมายความว่าฟังก์ชันชื่อ func มีอาร์กิวเมนต์ชื่อ abcd และเรากำหนดให้ abdc เป็นอะ-เรย์ของตัวแปรแบบ int ที่มีตัวแปรแถวละ 6 ตัวแต่มิได้กำหนดจำนวนแถว

การที่เราเปิดขนาด (size) ของอะเรย์ให้ว่างไว้ ก็เพราะเราไม่อาจทราบได้อย่างแน่ชัดว่าฟังก์ชัน **func** จะรับอาร์กิวเมนต์ที่ส่งมาจากภายนอกกี่ตัว เว้นแต่เราจะกำหนดขนาดของสตริงไว้อย่างชัดเจน (ด้วยการปิดด้วย NUL character)

5.2 โครงสร้าง (structure)

โครงสร้างหมายถึงกลุ่มของตัวแปรหรือการรวมกลุ่มของตัวแปรต่างแบบเอาไว้ในที่เดียวกัน โดยตัวแปรแต่ละตัวในโครงสร้างถือว่าเป็นตัวแปรเดี่ยว ๆ 1 ตัว ที่จริงโครงสร้างก็คือระเบียน (record) หนึ่งนั่นเอง รูปไวยากรณ์ของโครงสร้างปรากฏดังนี้

```
struct tag-type {
        variable declaration ;
        variable declaration ;
            ....
        }
        variable-name  ;
```

เช่น

```
struct {
   char month [10];
   int day ;
   int year ;
   }
   holiday ;
```

เป็นการกำหนดให้ตัวแปรชื่อ holiday เป็นโครงสร้างที่ประกอบด้วยตัวแปรชื่อ month เป็นอะเรย์ขนาด 10 อักขระของตัวแปรแบบ char ตัวแปรชื่อ day เป็นตัวแปรแบบ int และตัวแปรชื่อ year เป็นตัวแปรแบบ int โดยไม่กำหนด tag-type

จะเห็นว่าเราสามารถกำหนดลักษณะของตัวแปรได้เช่นเดียวกับที่ศึกษาผ่านมา ขอให้สังเกตว่าเรามีตัวแปรหลายแบบรวมกันอยู่ในโครงสร้างของตัวแปรชื่อ holiday ตัวอย่างโครงสร้างในภาษาอื่นปรากฏดังนี้

BASIC

Nothing comparable

FORTRAN

Nothing comparable

PASCAL

```
TYPE                                                    definition
     DATE = RECORD
            MONTH : ARRAY[0..9] OF CHAR;
            DAY : INTEGER;
            YEAR : INTEGER
     END;
VAR
    HOLIDAY : DATE;
    HOLIDAY.YEAR                                        reference to element
```

PL/I

```
DECLARE 1 HOLIDAY,                                      definition
        5 MONTH CHAR(10),
        5 DAY   FIXED BIN(15,0),
        5 YEAR FIXED BIN(15,0);
HOLIDAY.YEAR                                            reference to element
```

COBOL

```
01 HOLIDAY                                              definition
       05 MONTH PICTURE XXXXXXXXXX.
       05 DAY     PICTURE 99.
       05 YEAR    PICTURE 9999.
HOLIDAY.YEAR                                            reference to element
```

จากโครงสร้างมีสิ่งที่ควรทราบดังนี้

1. เนื่องจากโครงสร้างเป็นตัวแปรที่ประกอบไปด้วยตัวแปรหลายตัว หลาย - แบบ การอ้างอิงตัวใดตัวหนึ่งหรือหลายตัวจำเป็นต้องมีวิธีเรียกหรือวิธีอ้างอิง วิธีดังกล่าวเรียกว่า member operator หรือ dot operator ซึ่งใช้จุดหรือเครื่องหมายมหัพภาค

(คือ .) ตามด้วยตัวแปรตัวใดตัวหนึ่งในโครงสร้าง เช่น

holiday.month	อ้างถึงตัวแปรในโครงสร้างที่ชื่อ month ซึ่งเป็น charactor array ขนาด 10 อักขระ
holiday.month[0]	อ้างถึงตัวแปรชื่อ month ตัวแรก
holiday.day	อ้างถึงตัวแปรชื่อ day ซึ่งเป็นตัวแปรแบบ int
holiday.year	อ้างถึงตัวแปรชื่อ year ซึ่งเป็นตัวแปรแบบ int
holiday.day++	เพิ่มค่าตัวแปรชื่อ day ที่อ้างถึงครั้งละ 1
holiday.manth[0]='C'	ใส่ค่า 'C' ลงในที่ของสมาชิกตัวแรกของอะเรย์ชื่อ month ที่อ้างถึง

แอดเดรสของตัวแปรชื่อ month, day และ year ในโครงสร้างชื่อ holiday ปรากฏในส่วนความจำดังนี้ (สมมุติเริ่มที่แอดเดรส 1000)

MEMBER OF HOLIDAY	ADDRESS
month	500
day	510
year	512

2. tag-type หรือ structure tag

ความจริงแล้ว tag-type นี้เราจะกำหนดหรือไม่ก็ได้ (optional) แต่ถ้ากำหนดขึ้นก็เป็นการกำหนดให้โครงสร้างนั้นเป็นรูปแบบที่โครงสร้างอื่นสามารถยืม หรืออาศัย

รูปแบบได้ถ้าต้องการ เช่น

```
struct date {
    char month [10] ;
    int day ;
    int year ;
    } ;
```

เป็นการระบุว่าโครงสร้างตัวแปรนี้มี tag-type (เทียบได้กับนามสกุล) ชื่อ date มีสมาชิกภายในคือ month day และ year

ถ้าตัวแปรอื่นเช่นตัวแปรชื่อ dayofyr (ย่อมาจาก day of year) มี tag-type ชื่อ date เหมือนกันคือ

```
struct date dayofyr ;
```

ชี้ว่าสมาชิกของตัวแปร dayofyr คือ dayofyr.month, dayofyr.day และ dayofyr.year แสดงว่าโครงสร้าง dayofyrจะอ้างถึงหรืออ้างอิงถึงตัวแปรเดียวกันกับโครงสร้างที่มี tag-type เดียวกัน

การรับ-ส่งโครงสร้างและกำหนดค่าจะกระทำตรง ๆ มิได้ (ยกเว้น UNIX C) ต้องกระทำโดยทางอ้อม (เรียกว่า indirection access) คือ เรียกฟังก์ชันด้วยอาร์กิวเมนต์ที่เป็นแอดเดรสของโครงสร้าง เช่น &dayofyr และส่งค่าคืน (return (value) จากฟังก์ชันด้วย pointer ส่วนการกำหนดค่าหรือสำเนาค่าของโครงสร้างหนึ่งไปบนอีกโครงสร้างหนึ่งเราก็ทำตรง ๆ คือสำเนาทั้งโครงสร้างในคราวเดียวเลยไม่ได้ แต่ต้องกำหนดค่า(assign)ไปคราวละสมาชิกจนครบทุกสมาชิกของโครงสร้าง

tag-type เป็นสิ่งที่มีประโยชน์ เพราะช่วยให้เราประหยัดเวลาคือสามารถยืมรูปแบบหรือแม่พิมพ์ของโครงสร้างที่เคยนิยามไว้แล้วไปใช้ได้ โดยเพียงแต่อ้างชื่อ tag-type ก็เพียงพอที่จะแจ้งให้คอมไพเลอร์ทราบแล้วว่า เราต้องการให้ตัวแปรที่เราสนใจมีรูปหรือพิมพ์เดียวกับใครโดยไม่จำเป็นต้องกำหนดลักษณะตัวแปรนั้น ลองดูตัวอย่างต่อไปนี้

```
struct terminal{
   char clear ; /* clear screen */
   int width ; /* character per line */
   int line ; /* number of lines */
  };
```

โครงสร้างนี้มี tag-type ชื่อ terminal คำว่า terminal มิใช่ชื่อของโครงสร้าง (ในที่นี้เรามิได้ตั้งชื่อโครงสร้างไว้ ถ้าเราประสงค์จะให้โครงสร้างใดมีรูปแบบหรือพิมพ์เดียวกับโครงสร้างนี้เราสามารถอ้างเอาชื่อ terminal ไปใช้ได้ โดยไม่ต้องกำหนดโครงสร้างใหม่ที่ซ้ำกันกับโครงสร้างนี้ให้เสียเวลา เช่น ถ้าเราประสงค์จะให้ตัวแปรชื่อ crt_1 และ crt_2 มีโครงสร้างพิมพ์เดียวกับโครงสร้างข้างบนเราสามารถกำหนดได้สั้น ๆ ดังนี้คือ

```
struct terminal crt_1, crt_2 ;
```

ก็แปลว่าตัวแปรชื่อ crt_1 และ crt_2 มีโครงสร้างแบบเดียวกับ terminal กรณีนี้ถ้าเราไม่ชอบใช้ tag-type เราสามารถกำหนดโครงสร้างของ crt_1 และ crt_2 ได้เป็น

```
struct {
   char clear ;
   int width ;
   int line ;
  } crt_1 ;
```

```
struct {
   char clear ;
   int width ;
   int line ;
  } crt_2 ;
```

ขอให้สังเกตว่าถ้าเราฉลาดเราควรใช้วิธียืมรูปแบบคือใช้เป็น `struct terminal crt_1, crt_2;` เพราะประหยัดเวลากว่า

3. การกำหนดค่าเริ่มต้นให้แก่โครงสร้าง

การกำหนดค่าเริ่มต้นของโครงสร้างเราสามารถกระทำได้ 2 วิธี คือ กำหนดโดยมี tag-type กับโดยไม่มี tag-type การกำหนดโดยมี tag-type นั้นเราสามารถกำหนดค่าเริ่มต้นได้ทันที โดยไม่ต้องกำหนดลักษณะ หรือกล่าวถึงตัวแปรทั้งหลายในกลุ่มเดียวกัน ทั้งนี้เพราะการระบุ tag-type ไว้แสดงว่า เรากำลังใช้โครงสร้างแบบเดียวกับที่เคยกำหนดไว้แล้วก่อนหน้านั้น เช่น

```
static struct date holiday ={ '\001', 1, 1984):  หรือ
static struct date holiday = { "JANUARY", 1, 1984};
```

แสดงว่าเราต้องการใช้โครงสร้างเดียวกันกับตัวแปรที่มี tag-type ชื่อว่า date ขอให้สังเกตว่า '\001' ก็คือเลข 1 ในฐาน 10 แสดงว่าเรากำหนดให้ตัวแปร char month มีค่าเริ่มต้นเท่ากับ '\001' ก็คือเดือน 1 หรือกำหนดเป็น JANUARY เลยก็ได้

อีกวิธีหนึ่งคือการกำหนดค่าเริ่มต้นโดยไม่มี tag-type กรณีเช่นนี้เราจำเป็นต้องกำหนดลักษณะตัวแปรไปด้วยในขณะเดียวกัน เช่น ถ้าต้องการกำหนดค่าเริ่มต้นของตัวแปร dayofyr โดยไม่อ้างอิง tag-type ชื่อ date เราอาจกระทำได้ดังนี้

```
static struct {
    char  month[10] ;
    int day ;
    int year ;
  } dayofyr = ("JANUARY", 1, 1984 }
```

อนึ่งเนื่องจากตัวแปร month เป็นตัวแปรแบบ char ดังนั้นเราจึงสามารถกำหนดค่าให้แก่ตัวแปรนี้ได้ 2 วิธีคือ กำหนดเป็น string constantเช่นกำหนดเป็น "JANUARY" หรือกำหนดเป็น character constant คือ { 'J', 'A', 'N', 'U', 'A', 'R', 'Y', '\0' } หรือ 'J', 'A', 'N', 'U', 'A', 'R', 'Y', '\0' , 0, 0 ก็ได้ (ขอให้สังเกตเลข 0 สองตัวหลังของวิธีที่ 2 นี้ เราใส่ 0 ลงในในที่ของ month[8] กับ month[9] ขณะที่ '\0' เป็น NUL character หรือ NUL terminater เพื่อมิให้ month[8] หมายถึง 1 และ month[9] หมายถึง 1984

ถ้าเรากำหนดค่าเริ่มต้นของตัวแปร holiday เป็น

static struct date holiday = { 'J', 'A', 'N', 'U', 'A', 'R', 'Y', '\0', 0, 0, 1, 1984 } ค่าเริ่มต้นเหล่านี้จะปรากฏในส่วนความจำ ดังนี้ (สมมุติแอดเดรสแรกคือหมายเลข 1000)

VARIABLE	ADDRESS	VALUE
holiday.month[0]	1000	'J'
holiday.month[1]	1001	'A'
holiday.month[2]	1002	'N'
holiday.month[3]	1003	'U'
holiday.month[4]	1004	'A'
holiday.month[5]	1005	'R'
holiday.month[6]	1006	'Y'
holiday.month[7]	1007	'\0'
holiday.month[8]	1008	0
holiday.month[9]	1009	0
holiday.day	1010	1
holiday.year	1012	1984

ลองพิจารณาตัวอย่างต่อไปนี้ซึ่งแสดงการใช้ประโยชน์ของโครงสร้าง

```
struct {
    char clear ;
    int width ;
    int lines;
 } crt = { '\014', 80, 24 } ;
main ( )
 {
   puts ("this is a test of the clear screen".) ;
   puts ("press any key to continue :") ;
   getchar ( ) ;
   putchar (crt.clear) ; /* use puts for string */
   puts ("the screen should have been cleared.") ;
 }
```

ฟังก์ชัน puts ใช้สำหรับพิมพ์สตริงให้ปรากฏบนจอภาพ ในที่นี้ฟังก์ชัน puts จะพิมพ์ข้อความในเครื่องหมายคำพูดคู่ว่า this is a test of the clear screen และ press any key to continue เมื่อเรากดปุ่มใด (ก็ได้) ตามคำสั่งซึ่งเรียกโดยฟังก์ชัน getchar ( ) (ฟังก์ชัน getchar ( ) รับอักขระที่ส่งเข้าทางแป้นพิมพ์) ฟังก์ชัน putchar ( ) จะพิมพ์ค่าของ crt.clear คือ '\014' ลงบนจอภาพ character '\014' เป็น non-printable claracter โดยเป็นเลขฐาน 8 แปลว่า vertical tab', (คือเลื่อนบรรทัด)

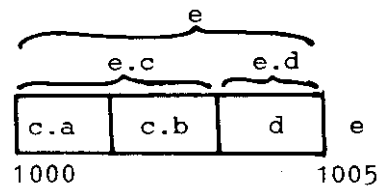
โครงสร้างต่าง ๆ นั้นเราสามารถจัดให้เป็นโครงสร้างของโครงสร้าง อะเรย์ของโครงสร้างและสามารถใช้ชื่อร่วมกันในระหว่างสมาชิกของใครสร้างต่าง ๆ ที่มีลักษณะเดียวกันได้ กล่าวคือ

ก. ถ้าเราประสงค์จะกำหนดโครงสร้างของโครงสร้าง ให้กำหนดโครงสร้างแรกขึ้นก่อน แล้วกำหนดโครงสร้างใหม่ที่ผนวกโครงสร้างเดิมไว้ภายใน เช่น สร้างโครงสร้างที่มี tag-type ชื่อ scดังต่อไปนี้ ขอให้สังเกตลักษณะการเก็บค่าในส่วนความจำ (สมมุติที่แอดเดรสเริ่มจาก 1000)

```
struct sc {
    int a ;
    int b ;
    } ;
```

ถ้าต้องการสร้างโครงสร้างชื่อ e ที่ผนวกเอาโครงสร้างที่มี tag-type ชื่อ sc ไว้ภายในให้กระทำดังนี้

```
struct {
    struct sc c ;
    int d ;
    } e ;
```



จะเห็นว่าในโครงสร้างใหม่นี้เราสามารถอ้างอิงหรืออ้างถึง (reference) ตัวแปรได้ 2 ครั้ง คือ e.d กับ e.c โดยที่ e.c สามารถอ้างถึงต่อไปได้อีกเป็น e.c.a กับ e.c.b ทั้งนี้เพราะตัวแปร c เป็นโครงสร้างที่มี tag-type เป็น sc

ขอให้สังเกตว่า เราจะใช้ tag-type (หรือ structure-tag) ก็ต่อเมื่อเราประสงค์จะให้ตัวแปรต่าง ๆ ที่เราสนใจมีโครงสร้างลักษณะเดียวกัน

ข. ถ้าเราประสงค์จะกำหนดอะเรย์ของโครงสร้าง ให้กระทำคล้าย ๆ กับโครงสร้างของโครงสร้างคือให้กำหนดโครงสร้างแรกขึ้นก่อนแล้วให้กำหนดตัวแปรอะเรย์ที่อ้างอิง tag-type เดียวกัน ขอให้สังเกตการใช้ประโยชน์ของ tag-type ในกรณีข้อ ก และ

ช นี้ ตัวอย่างเช่น

```
struct date {
     char month [10] ;
     int day ;
     int year ;
     } ;
```

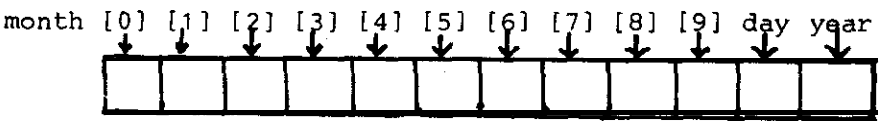
เป็นโครงสร้างที่มี tag-type ชื่อ date ถ้าเราต้องการสร้างอะเรย์ของโครงสร้างนี้สำหรับใน 12 เดือนให้อ้าง tag-type คือ dateแล้วกำหนดโครงสร้างที่มีตัวแปรเป็นอะเรย์ดังนี้

```
struct date someday 1121 ;
```

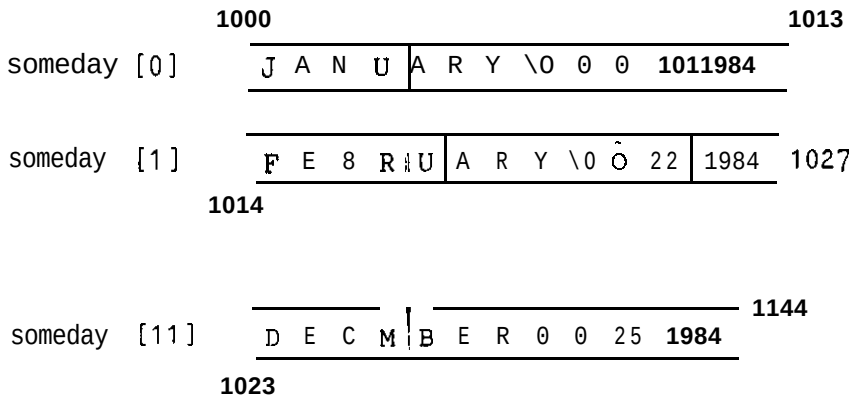
หมายความว่าเรากำหนดให้ตัวแปร someday เป็นอะเรย์ของโครงสร้างที่มี tag-type เป็น date ที่กำหนดไว้ข้างบน และเราสามารถกำหนดค่าเริ่มต้นของตัวแปร someday ได้ดังนี้

```
static struct date someday [12] = {
        ("JANUARY", 10, 1984 } ,
        { "FEBRUARY", 22, 1934 } ,

        { "DECEMBER", 25, 1984 }

     }
```

โครงสร้างของตัวแปรที่มี tag-type ชื่อ date ปรากฏดังนี้



สมมุติเริ่มที่แอดเดรส 1000 จะปรากฏอะเรย์ของโครงสร้างข้างตนดังนี้



5.3 การทำงานของโครงสร้างและอะเรย์

ตัวแปรโครงสร้างนั้นเราสร้างขึ้นเพื่อให้สามารถใช้สมาชิกของโครงสร้างหรือตัวโครงสร้างเองทำหน้าที่เป็นอาร์กิวเมนต์ในกรณีที่ต้องการเรียกฟังก์ชันมาทำงาน (โดยมากแล้วคอมไพเลอร์ทั่ว ๆ ไปยกเว้นกรณี UNIX C และ C ในเครื่องระดับมินิและเมนเฟรมจะไม่ยอมรับ-ส่งตัวโครงสร้างเป็นอาร์กิวเมนต์) วิธีปฏิบัติสำหรับเรียกใช้ฟังก์ชันก็คือ ส่งแอดเดรสของโครงสร้าง (ใช้ & operator) หรือของสมาชิกของโครงสร้าง (ใช้ dot operator และ & operator) ไปยังฟังก์ชัน

ลองดูตัวอย่างต่อไปนี้ที่ใช้โครงสร้าง crt สั่งชลอ (pause)การพิมพ์ข้อมูลลงบนจอภาพไว้ชั่วคราว ภายหลังจากพิมพ์ข้อมูลหรือข้อความจำนวนหนึ่ง (ตามตัวอย่างคือ 22 บรรทัด) ลงบนจอภาพเพื่อเราจะได้ตรวจทานข้อมูลว่าถูกต้องหรือไม่ เมื่อเห็นว่าถูกต้องค่อยพิมพ์ต่อโดยล้างจอภาพเดิมเสียก่อน ดังนี้

```
struct {
   char clear ;
   int width ;
   int lines ;
 } crt = { ' 014', 80, 24 };
main ( )
  {
   char c ;
   int num-lines ;
   num-line=1 ;
   for (; ;) {
         if (num-lines= =1)
             putchar (crt.clear) ;
         puts ("print something on the screen");
```

คำอธิบายประกอบ

โครงสร้างชื่อ crt กำหนดให้ clear =

'\014' (ล้างจอ) จอหนึ่งจุข้อความ 24 บรรทัด

บรรทัดละ 80 อักขระ

ให้ for เป็น infinite loop

สั่งล้างจอโดยสั่ง crt.clear ไปให้ Putchar ( )

ไปให้ ส่งแอดเดรสของ num-line ไปให้ pause ( )

แล้วเก็บในที่ชื่อ c ถ้าส่ง # มาให้เพื่อตัดออก

นอก for loop

ให้ so-for เป็น pointer ค่าของ *so-for คือค่าบนแอดเดรสของ num-lines

crt.lines-2 = 24-2

ไม่ส่งอะไรออกไป

รับข้อมูลเป็นอักขระผ่านแป้นพิมพ์

reset line counter to 1

```
        nun-lines+= 1 ;
        c=pause (& mm-lines) ;
        if(c= = '#')
               break ;
    }
 }
/*function to pause a crt display */
/*when screen becomes filled */
pause (so-for)
int  *  so-for
 {
     char c ;
     if (*so-for  !  crt.lines-2
           return (0) ;
     printf ("\n\t\t press any key to
            continue or # to end.") ;
     c = getchar ( ) ;
     • so-far=1 ;
     return (c) ;
 }
```

โปรแกรมหลักเริ่มด้วยการกำหนดให้ num-lines = 1 และ for loop เป็น infinite loop โดยที่ถ้า num-lines = = 1 ; ให้ล้างจอ คำสั่งล้างจอคือ putchar (crt.clear) ค่าอาร์กิวเมนต์ที่ส่งให้ putchar คือ '\014' ซึ่งเป็นรหัสแอสกี้ที่สั่งให้ล้างจอ แล้วพิมพ์ข้อความคราวละบรรทัด (80 อักขระ) ทุกครั้งที่พิมพ์เสร็จ 1 บรรทัด จะถามไปที่ฟังก์ชั่น - pause ( ) ว่าข้อความเต็มจอหรือยัง ถ้ายังไม่เต็มจอ (22 บรรทัด) ฟังก์ชั่น pause ( )

จะส่ง 0 (คือไม่ส่งอะไร) ไปให้ main ( ) ซึ่งเราก็มีสิทธิพิมพ์ข้อความต่าง ๆ ไปได้เรื่อย ๆ แต่ถ้าเต็มจอแล้ว pause ( ) จะส่งค่าเป็น 2 ลักษณะตามที่เราสั่ง คือ ส่ง character constant '#' ที่แปลว่าหยุดงานคือตัดออกจาก infinite loop หรือส่งเป็น pointer คือ * so-for=1 ชี้ไปที่ num-lines=1 ซึ่งแปลว่าล้างจอ แล้วพิมพ์จอต่อไป

ขอให้สังเกตว่า main ( ) จะเรียกใช้ฟังก์ชั่น pause ( ) โดยแจ้งให้ - pause ( ) ตามไปรับของที่อยู่ในบ้านหรือแอดเดรสของ num-lines ขอให้สังเกตอีกด้วยว่าสิ่งที่ส่งไปให้ pause ( ) ก็คือ line counter คือ num-lines หากมีคำถามว่าทำไมจึงไม่ส่ง crt.lines เป็นอาร์กิวเมนต์ไปยัง pause ( ) เพื่อจะได้ตรวจว่าเต็มจอหรือยังก็ตอบได้ว่า crt เป็นโครงสร้างนอก main ( )จึงมีลักษณะเป็น external storage class ซึ่งฟังก์ชั่นทั้งปวงในไฟล์มีสิทธิเรียกใช้เพราะเป็นสาธารณะ (glabal) อยู่แล้ว และ pause ( ) ก็หยิบมาใช้ได้เองตามที่เห็นว่าจำเป็น ขอให้สังเกตคำสั่งว่า if (* so-far ! = crt.lines-2 return (0) ; ซึ่งหากจะให้ส่ง crt.lines เป็นอาร์กิว-เมนต์เราก็สามารถทำได้ด้วยการกำหนดให้ crtเป็นโครงสร้างภายใน (privacy) ของ main ( ) ดังนี้

```
main ( )
 {
   char c ;
   int  num-lines  ;
   struct{ /*structure declared within main ( ) */
            char  clear  ;
            int width ;
            int  lines  ;
  }crt ;
            /*and initialized within  main( ) */
```

```
    crt.clear = '\014' ;
    crt.width = 80 ;
    crt.lines = 24 ;
    num-lines = 1 ;
    for (; ;) {
        if (num-lines = =1) ;
              putchar (crt.clear) ;
        puts ("print something on the screen") ;
        num-lines + = 1 ;
        c = pause (& num-lines, crt.lines) ;
        if (c = = '#')
               break ;
        }

/*function to pause a crt display */
/*when screen becomes filled */
pause (so_far, max)
int *so_far, max ;
{
     char c ;
     if (*so_far ! = max_2)
         return (0) ;
     printf ("\n \t \t press any key to continue.") ;
     c = getchar ( ) ;
     *so_far = 1 ;
     return (c) ;
}
```

กรณีนี้เรากำหนดให้โครงสร้างชื่อ crt เป็นโครงสร้างภายใน (internal หรือ local) ของ main ( ) แต่เนื่องจาก pause ( ) จะต้องทราบว่าจะต้องแสดงข้อความทั้งสิ้นกี่บรรทัด คือ จะต้องทราบจำนวนบรรทัดคือ lines และพบว่า lines เป็นสมาชิกของ crt ที่เป็นโครงสร้างส่วนตัวของ main ( ) การที่ pause ( ) ต้องการทราบจำนวนบรรทัดจึงเป็นความจำเป็นที่ main ( ) จะต้องส่งอาร์กิวเมนต์ชื่อ crt.lines และ num_lines ไปให้ pause ( ) ขอให้สังเกตในฟังก์ชัน main ( ) จะพบว่าข้อมูลในแอดเดรสของ num-lines นั้นเปลี่ยนค่าไปเรื่อย ๆ เพราะ num-lines เป็น counter และเมื่อnum-lineมีค่าถึง 22 เราจะต้องล้างจอคือเปลี่ยนค่า (alter) ให้ num_lines = 1 (ดูคำสั่ง * so_far=1 ใน pause ( )) การที่เราต้องกรใช้ฟังก์ชันเรียก (ในที่นี้คือ pause ( )) ไปเปลี่ยนค่า (alter) ของตัวแปรใน main ( ) เป็นเหตุผลที่ชี้ให้เห็นถึงความจำเป็นที่เราจำเป็นต้องใช้ pointer ลองดูค่าในแอดเดรสของ crt.lines ซึ่งเรากำหนดให้มีค่าคงที่เท่ากับ 24 นั้น จะพบว่าเราไม่เคยคิดที่จะเปลี่ยนแปลงค่านี้เป็นอย่างอื่นเลย ดังนี้การรับส่งอาร์กิวเมนต์กรณีของ crt. lines จึงมิใช่รับส่งในรูป pointer แต่เป็นการส่งแบบ call by value ตามธรรมดาที่ฟังก์ชัน pause ( ) มิได้เปลี่ยนค่าในแอดเดรสของ crt.lines แต่ประการใด

ข้อสังเกตอีกประการหนึ่งก็คือการกำหนดค่าเริ่มต้นของสมาชิกของโครงสร้างในกรณีที่โครงสร้างเป็นโครงสร้างส่วนตัวของฟังก์ชันก็คือ ถ้าโครงสร้างใดมิใช่โครงสร้างภายนอก แต่เป็นโครงสร้างส่วนตัวของฟังก์ชันใดการกำหนดค่าเริ่มต้นของสมาชิกโครงสร้างให้กำหนดไว้ในลักษณะเดียวกันกับการกำหนดกำหนดค่าเริ่มต้นให้แก่ตัวแปรปกติธรรมดาทั่วไป

สำหรับการใช้อะเรย์ในโครงสร้าง เช่นกรณีที่สมาชิกของโครงสร้างเป็นอะเรย์หรือตั้งชื่อโครงสร้างเองเป็นอะเรย์ทั้งสองประการนั้นให้ลองศึกษาจากตัวอย่างต่อไปนี้

อะเรย์ของโครงสร้างก็คือกลุ่มของโครงสร้างแบบเดียวกันที่เรียงลำดับ หรือจัดกลุ่มกันไว้ เวลาเรานึกถึงโครงสร้างขอให้นึกถึงแปลนบ้านที่ประกอบไปด้วยห้องเล็กบ้างห้องใหญ่บ้าง ห้องเล็กก็คือพื้นที่ห้องขนาดเล็กที่เหมาะสำหรับเก็บของน้อย ห้องใหญ่เหมาะสำหรับเก็บของมาก การกำหนดชนิดของตัวแปรจึงเป็นเรื่องของการจัดที่คือวาดแปลนให้เหมาะสมที่จะเก็บตัวแปรที่เปลืองที่เก็บหรือใช้ห้องเล็กใหญ่ไม่เท่ากันไว้อย่างเหมาะสม ถ้าเราปลูกบ้านตามแปลนนี้เรียงต่อ ๆ กันแบบบ้านจัดสรร ลักษณะดังกล่าวคืออะเรย์ของโครงสร้างนั่นเอง

ลองดูโครงสร้างต่อไปนี้ซึ่งเป็นอะเรย์ของโครงสร้างสำหรับจ่ายเงินเดือนให้แก่พนักงาน แต่ก่อนที่จะดูตัวอย่างโปรแกรมเรื่องนี้ขอให้ทำความรู้จักกันโครงสร้าง และอะเรย์ของโครงสร้างรวมทั้งความรู้ที่จำเป็นเกี่ยวกับการจัดเงินและการใช้ชนิดของธนบัตรและเหรียญดังนี้

โครงสร้างที่จะใช้อ้างอิงคือ

```
struct   bill-count{
     char name [NAMSIZ] ;
     int wage;
     int curncy [8] ;
 } totals [MAXSIZ] ;
   int denom [8] = { 50000, 10000, 2000, 1000, 500, 100, 50, 1 }
```

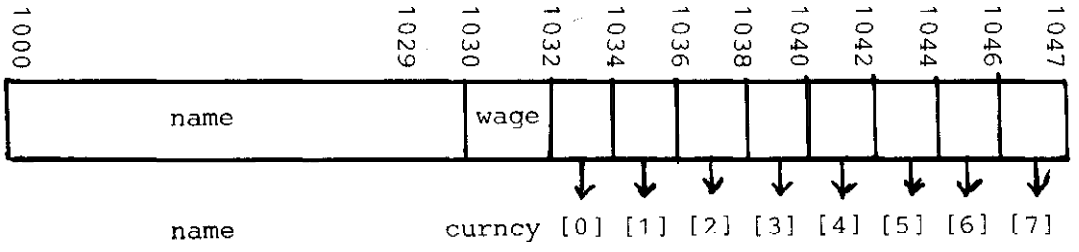
โครงสร้างนี้คือโครงสร้างชื่อ totals ขอให้สังเกตว่าเรากำหนดให้ totals เป็นอะเรย์คือ totals [ ] แสดงว่านี่คืออะเรย์ของโครงสร้าง สมาชิกของอะเรย์นี้จะมีด้วยกันทั้งสิ้นเท่ากับ MAXSIZ โครงสร้าง MAXSIZ คือ symbolic constant ที่เรานิยามด้วยคอมมานด์ # define เช่นถ้าเรากำหนดให้ MAXSIZ เท่ากับ 50 แสดงว่าเรากำหนด

ให้มีอะเรย์ชื่อ totals ประกอบด้วยโครงสร้างเป็นสมาชิกรวมทั้งสิ้น 50 โครงสร้างคือ โครงสร้าง totals [0], totals [1], ..., totals [49] เป็นต้น โครงสร้างเหล่านี้มี structure tag ชื่อ bill-count สมาชิกของโครงสร้างประกอบไปด้วยตัวแปร 3 ตัว คือตัวแปร char ชื่อ name มีความยาวเท่ากับ NAMSIZ อักขระ ขอให้สังเกตว่า name เป็นอะเรย์เรียกว่า character array ใช้สำหรับเก็บชื่อพนักงาน ตัวแปร int ชื่อ wage (ใช้ที่ 2 อักขระ) ใช้สำหรับเก็บข้อมูลค่าจ้างพนักงานตามชื่อในที่ชื่อ name และตัวแปรชื่อ curncy เป็น integer array ประกอบด้วยตัวแปร int 8 ตัว (แต่ละตัวใช้ที่ 2 อักขระ) ใช้สำหรับเก็บจำนวน ธนบัตรหรือเหรียญชนิดต่าง ๆ ที่จ่ายให้พนักงาน ในที่นี้สมมุติว่าเราใช้ชนิดของธนบัตรและเหรียญรวม 8 ชนิดคือ ธนบัตรฉบับ 500, 100, 20, 10 และเหรียญ 5, 1, 50, .01 เราจึงกำหนดให้ตัวแปรชื่อ denom (denom ย่อมาจาก denomination อัตราเงิน) เป็น integer array คือ denom [8] และกำหนดค่าเริ่มต้นของสมาชิกของ สมาชิกของ demon เป็นหน่วยสตางค์คือ

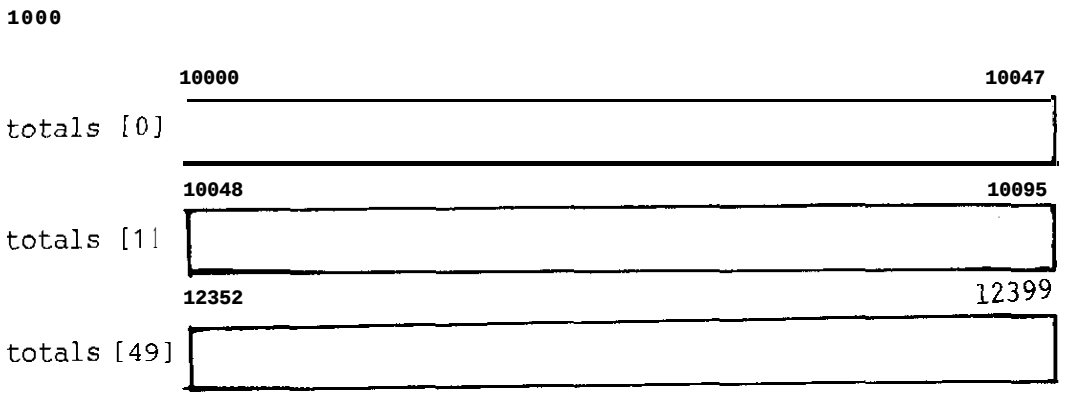
denom [8] ={ 50000, 10000, 2000, 1000, 500, 100, 50, 1 }

หมายความว่า denom [0]= 50000 สตางค์ = 50000/100 บาท เป็นต้น

สมมุติเรานิยามให้ NAMSIZ เท่ากับ **30** และ MAXSIZ เท่ากับ **50** จะปรากฏอะเรย์ของโครงสร้างทั้งสิ้น **50** โครงสร้างคือโครงสร้าง total [0], total [1], ..., total [49] แต่ละโครงสร้างมีแบบ (template) ดังนี้ สมมุติเริ่มที่แอดเดรส 1000



ดังนั้นอะเรย์ของโครงสร้าง total จึงปรากฏดังไดอะแกรมต่อไปนี้ สมมุติเริ่มที่แอดเดรส



แต่ละโครงสร้างจะบรรจุข้อมูลของพนักงานได้ 1 คน คือ ชื่อและนามสกุล - (ยาวไม่เกิน 30 อักขระ) อัตราค่าจ้างหรือเงินเดือนและจำนวนธนบัตร และเหรียญชนิดต่าง ๆ ที่เราได้รับมาในซองเงินเดือน

ลองดูโปรแกรมเรื่องการจ่ายเงินเดือนด้วยเงินสดโดยละเอียดจากโปรแกรมต่อไปนี้ รายละเอียดประกอบคำสั่งจะเขียนบรรยายไว้ท้ายโปรแกรม

```
#define MAXSIZ 50 /*set maximum number of employees */
#define NAMSIZ 30 /*set max character in name */
struct bill-count!
    char name [NAMSIZ];
    int wage ;
    int curncy [8];
```

```
} total [MAXSIZ] ;
int denom [8] ={ 50000, 10000, 2000, 1000, 500, 100, 50, 1 } ;
main ( )
{
      char string (NAMAIZ], c ;
      int x, i, j, part ;
      float sum ;
      for (i=0 ; i<NAMSIZ ; i ++) {
          getfld("\n name(30 char max, #=end):",NAMSIZ-1,
            totals [i]. name) ;
          if(totals [il. name [0]==='#')
              break ;
          totals [i]. wage=part=getsum(" n enter wage;");
          for (j = 0 , j<8 ; ++j){
              sum = part ;
              totals [j]. curncy [j] = sum/denom [j] ;
              part % = denom [j] ;
           }
    }
    printf(' \n wage t$500 t$100 t$20 t$10 t$5 t$1 t-50 t.01");
    for (j=0; j<8; j++)
        denom [j] = 0 ;
    for (j=0 ; j<i, j++){
        printf("\n %s\t%d\n;t", totals[j].name, totals[j].wage);
        for(x=0 ; x<8 ; x++) {
           printf("%7d", tatols [j].curncy [x]) ;
           denom[x] + = totals[j].curncy [x] ;
         }
```

```
        printf ("\n\n");
        for (j=1; j<78; j++)
              printf ("-");
        printf ("\n\n totals : ");
        for (j=0; j<8, j++)
             printf ("%7d", denom[j]);
}
getfld (prompt, biggest, individ)
char individ [ I, *prompt ;
int biggest ;
{
        char c ;
        int i ;
        printf ("%s", prompt) ;
        i = o ;
        while ( ( c=getchar ( ) )!= '\n' && i<biggest) {
              individ [i] = c ;
              i++;
        }
        individ [i] = '\0' ;
}
getsum (prompt)
char * prompt ;
{
        char wage [NAMSIZ] , c ;
        int i ;
        printf ("%s", prompt) ;
        i = 0 ;
```

```
while ((c = getchar ( )) ! = ' n' && i<NAMSIZ-1)
        if ( c! = '.') {
            wage [i] = c ;
            i++ ;
         }
 }
wage [i] = '\0' ;
return (atoi (wage) ) ;
}
```

โปรแกรมนี้เริ่มด้วยการนิยามให้ totals [ ] เป็นอะเรย์ของโครงสร้าง ดังรายละเอียดที่อธิบายมาแล้วในตอนต้น

ภายหลังจากที่เราได้กำหนดลักษณะตัวแปรที่จำเป็นแล้วโปรแกรม main ( ) จะเริ่มสั่งให้ฟังก์ชั่น getfld ( ) ทำงานโดยรับข้อมูลให้แก่สมาชิกแรกของโครงสร้าง-totals [0] โดย main ( )จะส่งแอดเดรสของ totals [0] name ไปให้ getfld ( ) แอดเดรสของ totals[0] name ก็คือแอดเดรสของสมาชิกของโครงสร้าง totals[0] ชื่อ name ฟังก์ชั่น getfld ( ) จะรับอักขระต่าง ๆ ที่เป็นชื่อพนักงานเข้ามาทีละอักขระ ด้วยฟังก์ชั่น getchar ( ) โดยนำอักขระเหล่านั้นไปเก็บไว้ในอะเรย์ individ [ ] และรับอักขระเข้าไปเก็บในอะเรย์เรื่อยไปตราบเท่าที่เรายังไม่กด ↵ (หมายถึง cariage return ในโปรแกรมคือ '\n') หรือจำนวนอักขระยังไม่ครบ NAMSIZ-1 อักขระ แล้วปิดท้ายด้วย NUL คือ '\0' เพื่อให้สามารถพิมพ์ชื่อในรูป string ได้ เมื่อรับชื่อแล้วก็ส่งการควบคุมคืน main ( ) แล้วตรวจดูว่าอักขระแรกของของ name เป็น # (pound sign) หรือไม่ ถ้าใช่ก็จะตัดออกจาก for loop (แสดงว่าเราเลิกบันทึกข้อมูล) ถ้าไม่

ใช่ # ก็จะส่งการควบคุมไปสู่คำสั่งต่อไปคือสั่งให้กรอกข้อมูลเกี่ยวกับค่าจ้างของพนักงานชื่อนั้นด้วยคำสั่ง totals [i].wage = part = getsum ( ) ;

ขอให้สังเกตคำสั่ง totals [i].wage = part = getsum ( ) ;ว่าเป็นคำสั่งที่มีลักษณะของ multiple assignment ซึ่งภาษา C ยอมให้มีได้เสมอ คำสั่งข้างต้นมีผลเหมือนคำสั่งว่า

```
part = getsum ( ) ;
totals [i].wage = part ;
```

ซึ่งเราสามารถกำหนดให้คำสั่งหนึ่งประกอบไปด้วยกี่ assignment ก็ได้

ลองดูที่คำสั่ง totals [i].wage = part = getsum ( ) ; จะพบว่าเราเรียกใช้ฟังก์ชั่น getsum ( ) ให้ทำงานแล้วส่งผลมาเก็บที่ total [i].wage เมื่อพิจารณาฟังก์ชั่น getsum ( ) จะพบว่าเรากำหนดให้ wage เป็น character array ขนาด NAMSIZ อักขระโดยรับข้อมูลเข้าในรูป ASCII string เก็บไว้ในอะเรย์ชื่อ wage เมื่อเรากด ↵ (คือ RETERN) ก็จะมี NUL มาปิดท้าย string จากนั้นจึงแปลง ASCII string เป็น integer ด้วยฟังก์ชั่น atoi ( ) atoi ( ) คือฟังก์ชั่นที่แปลง ascii itointeger แปลงเสร็จก็ส่งผลไปเก็บที่ part ใน main ( ) แล้ว main ( ) ก็สั่งให้ใส่ค่าจาก part ลงใน totals [i].wage (คือในที่ชื่อ wage ของโครงสร้าง totals [i] ต่อไป !

จากนั้นการทำงานจะมาทำที่ for loop ในคือ

```
for (j = 0, j<8; j++) {
    sum = part ;
    totals [i].curncy [i] = sum/denom [j] ;
    part % = denom [j] ;

 }
```

for loop นี้จะทำงานโดยใส่ค่าของค่าจ้างใน part ลงใน sum แล้วจัดการแปลงค่าจ้างเป็นธนบัตรชนิดต่าง ๆ และเหรียญชนิดต่าง ๆ ด้วยการนำหน่วยเงินหน่วยที่ j (j=0, 1, ..., 7) ไปหารค่าจ้างผลหารจะปรากฏออกมาเป็นจำนวนธนบัตรชนิดต่าง ๆ และเหรียญชนิดต่าง ๆ แล้วเก็บจำนวนเหล่านั้นลงในอะเรย์ชื่อ curncy [ ] ของโครงสร้าง totals [i] การหารใช้วิธี modulo division คือ part % = denom [j] ; ซึ่งก็คือ

```
part = part % denom [j] ;
```

ตัวอย่างเช่น นายแดงได้รับค่าส่งจ้าง 8985 บาท for loop นี้จะแปลงเงินจำนวนนี้เป็นธนบัตรชนิดต่าง ๆ และเหรียญชนิดต่าง ๆ รวม 8 ชนิด (คือ loop วิ่ง 8 รอบ) ดังนี้

รอบที่ 1 898500/50000 = 17 เศษ 48500 คือธนบัตรฉบับละ 500 บาทรวม 17 ฉบับ

รอบที่ 2 48500/10000 = 4 เศษ 8500 คือธนบัตรฉบับละ 100 บาทรวม 4 ฉบับ

รอบที่ 3 8500/2000 = 4 เศษ 500 คือธนบัตรฉบับละ 20 บาทรวม 4 ฉบับ

รอบที่ 4 500/1000 = 0 เศษ 50 คือไม่มีธนบัตรฉบับละ 10 บาท

รอบที่ 5 500/500 = 1 เศษ 0 คือเหรียญ 5 บาทรวม 1 เหรียญ

รอบที่ 6 0/100 = 0 เศษ 0 คือไม่มีเหรียญ 1 บาท

รอบที่ 7 0/50 = 0 เศษ 0 คือไม่มีเหรียญ 50 สตางค์

รอบที่ 8 0/1 = 0 เศษ 0 คือไม่มีเหรียญ 1 สตางค์

ก็แปลว่าฝ่ายการเงินจ่ายเงินเดือนให้นายแดงทั้งสิ้น 8985 บาท โดยจ่ายเป็นธนบัตรฉบับละ 500 บาทรวม 17 ฉบับ ฉบับละ 100 บาท รวม 4 ฉบับ ฉบับละ 20 บาท รวม 4 ฉบับ และเหรียญ 5 บาทอีก 1 เหรียญรวม 8985 บาทพอดีไม่มีธนบัตรฉบับละ 10 บาท และเหรียญอื่น ๆ

จากนั้นโปรแกรมจะเปลี่ยนค่า i เป็น 1 เพื่อทำงานรอบต่อไป (ตามตัวอย่างนี้ i = 0 คือนายแดง) แล้วเปลี่ยนค่า i ไปเรื่อย ๆ จนถึง i = 49 คือพนักงานคนที่ 50

ส่วนท้ายของ main ( ) จะทำการรวมจำนวนธนบัตรชนิดต่าง ๆ และเหรียญชนิดต่าง ๆ ที่จำเป็นต้องใช้เพื่อจ่ายเงินเดือนให้แก่พนักงานทั้ง 50 คน

ขอให้ผู้อ่านทดลองวิ่งโปรแกรมนี้ดูเองโดยกำหนดข้อมูลต่าง ๆ เอง เพื่อให้เกิดความรู้ ความเข้าใจที่ลึกซึ้งขึ้น

ก่อนที่จะผ่านตอนนี้ไปขอทำความเข้าใจกับคำสั่ง return ในภาษา C เพื่อให้เข้าใจและสั่งงานได้ถูกต้อง

คำสั่ง return ก็คือคำสั่งที่ใช้สั่งย้ายการควบคุมคืนไปยังจุดเรียก คำสั่งนี้คือ return ; ซึ่งทำหน้าที่เช่นเดียวกับคำสั่ง RETURN ในภาษาเบสิค ต่างกันเล็กน้อย ตรงที่คำสั่ง return ; ในภาษา C ดูจะเป็นส่วนเกินเพราะถึงไม่สั่งว่า return ; ฟังก์ชันต่าง ๆ ก็จะย้ายการควบคุมคืนสู่จุดเรียกโดยอัตโนมัติอยู่แล้วเมื่อคอมไพเลอร์อ่านพบ วงเล็บปีกกาปิด (}) ที่อยู่ท้ายฟังก์ชั่น เช่นฟังก์ชัน pause-1 ( ) และ pause-2 ( ) ต่อไปนี้ต่างก็ส่งการควบคุมคืนไปยังจุดเรียกเหมือนกันคือ

```
pause-1 ( )
 {
   int c ;
   c = getchar ( ) ;
   return ;
 }
```

```
pause-2 ( )
 {
    int c ;
    c = getchar ( ) ;
 }
```

ขอให้สังเกตว่า pause-2 ( ) ไม่มีคำสั่ง return ;

แต่ในภาษา c มิใช่มีเพียงคำสั่ง return ; เท่านั้น แต่ยังมีคำสั่ง return ( ) อีกคำสั่งหนึ่งซึ่งทำงานต่างกัน คำสั่ง return ( ) ; จะส่งผลลัพธ์ที่อยู่ในวงเล็บเล็กคืนไปยังจุดเรียกเช่น

```
pause-I ( )
 {
   int c ;
   c = getchar ( ) ;
   return (c) ;
 }
```

จะส่งอักขระในที่ชื่อ c ที่รับจากแป้นพิมพ์ด้วยคำสั่ง c = getchar ( ) ; ไปยังจุดที่เรียกด้วยเหตุนี้จึงกล่าวว่า return (0) ; ก็คือไม่ส่งอะไรไปได้จุดเรียก

รูปไวยากรณ์ของคำสั่ง return คือ return exp ; ตามตัวอย่างข้างบน exp คือ (c) หากไม่ประสงค์จะใช้วงเล็บโดยเขียนเป็น return c ก็ไม่ผิด

5.4 Unions

ยูเนียนคือพื้นที่ในส่วนความจำที่เรากำหนดขึ้นเพื่อให้สามารถ เรียกหรืออ้างอิงใช้ได้หลายวิธี หรือนัยหนึ่งก็คือพื้นที่ที่ยอมให้ตัวแปรหนึ่ง ๆ มีความสามารถ รับค่าข้อมูลได้หลายแบบในขณะเดียวกัน ทั้งนี้เพื่อเป็นการประหยัดพื้นที่ รูปไวยากรณ์ของยูเนียนใกล้เคียงกับรูปไวยากรณ์ของโครงสร้างมาก รูปไวยากรณ์ปรากฏดังนี้

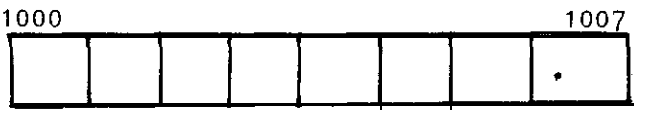
```
union tag_type {
      variable declaration ;
      variable declariation ;

      }
      variable name ;
```

คอมไพเลอร์จะอ่านโปรแกรมเมื่อพบคำว่า union จะอ่านชนิดของตัวแปรแล้วจัดเตรียมเนื้อที่ในส่วนความจำไว้ให้ใหญ่พอที่ตัวแปรที่ใช้พื้นที่มากที่สุด (largest item) ของกลุ่มจะบรรจุลงได้(แสดงว่าขนาดของ union ก็คือขนาดของตัวแปรที่ใหญ่ที่สุด หรือใช้พื้นที่มากที่สุด การเรียกหรือการอ้างถึงตัวแปรใดใน union ให้ปฏิบัติเช่นเดียวกับโครงสร้าง ข้อที่ควรระวังก็คือ union เป็นตัวแปรหรือพื้นที่ที่อนุญาตให้ตัวแปรมากแบบผ่านเข้ามาอยู่ หรือเก็บไว้ได้ เมื่อเก็บตัวแปรแบบใดไว้ก็ขอให้เราจำไว้ด้วย เพราะถ้าเรียกใช้ผิดก็จะส่งค่าให้ผิด

ตัวอย่างเช่น

```
union {
   char  t-char  ;
   int t_int ;
   float t_float ;
   double t-double ;
   } u ;
```

เป็นการเตรียมพื้นที่ในส่วนความจำให้แก่ตัวแปรชื่อ u ซึ่งอาจถูกส่งให้เก็บค่าที่ส่งมาจากฟังก์ชันตัวแปร u สามารถรับค่าข้อมูลได้หลายแบบคือแบบ char, int, float และ double เมื่อคอมไพเลอร์อ่านพบคำว่า union จะอ่านต่อไปว่าเรากำหนดชนิดของตัวแปรไว้เป็นแบบใดบ้าง ขอให้สังเกตว่าเราใช้ชื่อตัวสมาชิกของ u ว่าเป็น t แบบเดียวกัน แต่หากเราประสงค์จะใช้ชื่ออื่นก็ได้ ตามตัวอย่างแสดงว่าเราให้ u เก็บค่าที่ส่งมาจากฟังก์ชั่นได้ 4 แบบคือ char หรือ int หรือ float หรือ double ก็ได้ เมื่อเป็นเช่นนี้คอมไพเลอร์จะเตรียมที่ไว้ 8 ไบท์ (หรือ 64 บิท) ตามจำนวนขนาดพื้นที่ของ double integer ดังภาพ (สมมุติเริ่มที่ address1000)



ตัวแปร union u

การเรียกค่าจากพื้นที่ดังกล่าวไปใช้ให้ใช้ dot operator หรือ member opertor เช่นเรียกว่า u.t-char, u.t-int, u.t-float หรือ u.t-double การเรียกเราต้องทราบด้วยว่าขณะนั้น u เก็บค่าแบบใดไว้หากเก็บ double เอาไว้แต่เราเรียกเป็น u.t-char u จะส่งค่าให้เพียง 1 ไบท์ ซึ่งก็รับ-ส่งค่ากันได้ แต่ไม่ถูกต้องแม่นยำ แปลว่าเราต้องจำไว้เองว่าก่อนหน้านี้ได้ส่งอะไรไปเก็บไว้ใน u จะได้เรียกใช้หรือส่งค่าต่อไปที่อื่นได้ถูกต้อง อย่างไรก็ตามเรามี operator ชื่อ sizeof ช่วยแก้ปัญหานี้ sizeof จะช่วยนับจำนวนไบท์ในพื้นที่ของ union variable ทำให้ทราบได้ว่าขณะนี้มีตัวแปรแบบใดถูกเก็บไว้ในที่นั้น แบบทั่ว ๆ ไปของ opertor นี้คือ

```
sizeof (unknown)
```

เช่น

```
x = sizeof (y)
```

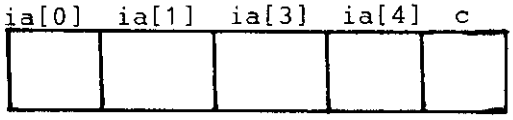
จะนับจำนวนไบท์ของตัวแปร union ชื่อ y นับได้เท่าไรก็จะใส่ค่าลงในที่ของตัวแปร x ทำให้เราพอจะทราบได้ว่าขณะนี้ตัวแปร y เก็บค่าแบบใดเอาไว้ โดยเปรียบเทียบกับขนาด - (size) ของตัวแปรแต่ละแบบที่เรารู้จักดีแล้ว

อนึ่งตัวแปร union นั้นเราไม่อาจกำหนดค่าเริ่มต้นได้ การโอเปอเรตตัวแปร union เข้าด้วยกันกระทำได้ด้วยการเรียกหรืออ้างอิงแอดเดรสเท่านั้น ในภาษา C ของ UNIX นั้นเราจะรับส่งและกำหนดค่าได้เช่นเดียวกับตัวแปรโครงสร้าง นอกจากนี้เรายังกำหนดเป็น union ของโครงสร้าง union ของอะเรย์และ union ของ union ได้โดยปฏิบัติเช่นเดียวกันกับการทำโครงสร้างของโครงสร้าง อะเรย์ของโครงสร้าง เช่น

กำหนดให้โครงสร้างหนึ่งมี tag-type ชื่อ ss ประกอบด้วยสมาชิกคือ int array 5 ตัวและ char อีก 1 ตัวคือ

```
struct ss {
     int ia[5] ;
     char c ;
      } ;
```

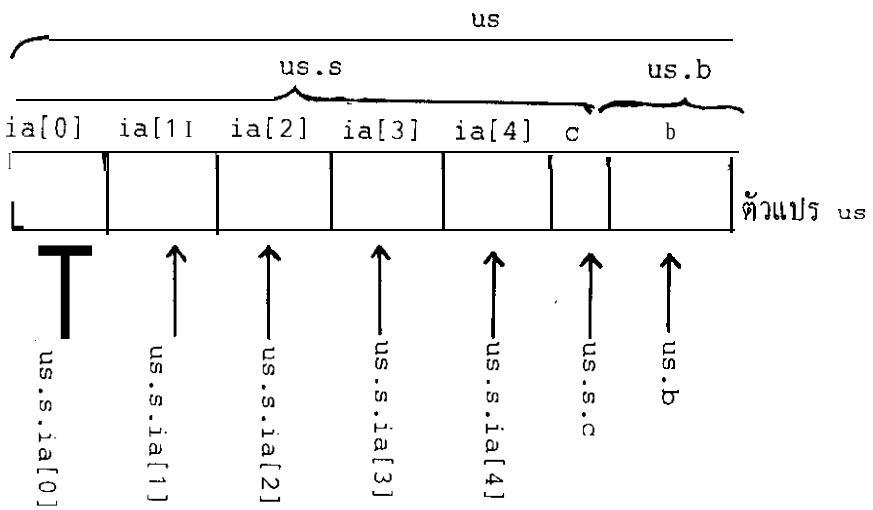
พื้นที่ในส่วนความจำของตัวแปรนี้จะปรากฏดังภาพ ขอให้สังเกตว่า ia เป็น int จึงใช้พื้นที่ 2 ไบท์



ถ้าใช้ us เป็นตัวแปร union ที่ประกอบไปด้วยโครงสร้างชื่อที่มี tag-type ชื่อ ss และตัวแปร b ที่เป็น float int (ใช้พื้นที่ 4 ไบท์) เราสามารถกำหนดตัวแปร us ได้ดังนี้

```
union ua{
    struct ss s ;
    float b ;
    } us ;
```

ภาพของ union ชื่อ us ปรากฏดังนี้ สมมุติว่าเริ่มต้นที่แอดเดรส 1000



ภาพแสดงแอดเดรสและขนาด (size) หรือ length ของแต่ละตัวแปรที่เป็นสมาชิกตัวแปร us ปรากฏดังนี้

VARIABLE	TYPE	ADDRESS	LENGTH
us	union	1000	15
us.s	structure	1000	11
us.s.ia	array	1000	10
us.s.ia[0]	integer	1000	2
us.s.ia[1]	integer	1002	2
us.s.ia[2]	integer	1004	2
us.s.ia[3]	integer	1006	2
us.s.ia[4]	integer	1008	2
us.s.c	character	1010	1
us.s.b	float	1011	4

5.5 ตัวอย่างฟังก์ชัน

ต่อไปนี้เป็นตัวอย่างของฟังก์ชันในภาษา C ขอให้ระลึกว่าการทำงานในภาษา C นั้นเราให้วิธีเชื่อมโยงฟังก์ชันเข้าด้วยกันด้วยการรับส่งด้วย argument ฟังก์ชันทั้งหลายอาจนิยามขึ้นใหม่เรียก definition function หรือเป็นฟังก์ชันใน compiler library โดยมากแล้วฟังก์ชันจะอยู่ใน compiler library การทำงานภาษา C จึงทำได้รวดเร็วมากเพราะสาเหตุนี้ และขอให้ระลึกว่าตัวแปรกลุ่มนั้นเรารับส่งค่าโดยวิธีที่เรียกว่า call by refernce

ลองพิจารณาฟังก์ชันต่อไปนี้

1) ฟังก์ชัน sumarr ใช้รวมค่าสมาชิกของอะเรย์เข้าด้วยกัน

```
double ●    mwr(%n)
/* sums a float array */
float a[]; /* array to be summed */
int n;     /* size of array */
  {
  double sum=0.0;
  int i;
  for (i = 0;i<n;i++)
       {
       sum = sum + a[i];
       t
  return sum;
  t
```

2) ฟังก์ชัน invert ใช้ย้อนทวนลำดับที่สมาชิกของอะเรย์

```
invert(a,num)
/* inverts the sequence of numbers in an array *I.
int a[ ]; /* array to be inverted */
int num; /* number of elements in array */
    {
    int i, j, temp;
    for (i=0, j=num-1;i<j;i++, j--)
        {
        temp=a[i];
        a[i]=a[j];
        a[j]=temp;
        }
    return 0;
    }
```

3) ฟังก์ชัน match ใช้เปรียบเทียบสตริงอักขระต่ออักขระ แล้วแจ้งผลลัพธ์ออกมา

```
match(instr,mstr)
/* matches mstr against instr
      returns -1 or index of match */
char instr[ ]; /* String to be indexed into */
char mstr [ ]; /* String to be matched */
    {
    int ret;
    register int i, j,mcnt, ncnt;
    ncnt=strlen(instr);
    mcnt=strlen(mstr);
    ret= -1;
    /* starting with each character */
    for (i=0;i<=(ncnt-mcnt);i++)
        {
        /* do the comparison against current string */
        for (j=0;j<mcnt;j++)
            {
            if (mstr[j]!=instr[i+j]) break:
            t
        I* if completed all loops, then a match */
        if (j==mcnt)
            {
            ret=i;
            break;
            }
        }
    return ret;
    }
```

```
atoint(str)
char str[ ];
/* convert9 a" string to a" integer
      range is -9999 to +9999.
      returns ERROR (= 32767) if a" error*/
      {
      int len;
      int ret=0;
      int sign = 0;
      int cnt=0;
      int error=0;
      int i=0;
      int j;
      /* eliminate initial spaces */
      while (str[i]==' ')
            {
            i++;
            }
      /* check for a + or a = */
      if (str[i]=='-')
            {
            sign=1;
            i++;
            }
      else if (str[i]=='+')
           {
           i++;
           }
      for (j=0;j<4;j++)
            {
            /* check for a digit */
            /* only accept 4 digits */
            if (str[i]=='\0') break;
            if ((str[i]<'0') || (str[i]>'9'))
               {
               error = 1;
               break;
               }
            /* found a digit, determine new value */
            ret = ret*10 + (str[i]-'0');
            i++;
            }
      /* if not at end of string, then error */
      if (str[i]!='\0') error=1;
      if (sign) ret = -ret;
      if (error) ret=-32767;
      return ret;
      }
```

4) ฟังก์ชัน midstr ใช้สร้างสตริงย่อย (substring) หรือเรียกดึงบางส่วนของสตริงออกมาใช้งานเช่น ดึงเฉพาะบ้านเลขที่และชื่อมาพิมพ์ ดึงเฉพาะรายการข้อมูล (field)ของบันทึก (record) มาใช้หรือคำนวณขอให้สังเกตการเรียกใช้ฟังก์ชั่น strlen( ) ในฟังก์ชั่น match และ midstr

```
char *midstr(dest,string,start,length)
char dest[ ], string[ ];
int start, length;
    {
    int i,j;
    j=0;
    /* check to see if start is beyond string */
    if (start>strlen(string))
        {
        dest[0] = '\0';
        }
    else
        {
        for (i=0;i<length;i++)
                {
                dest[i] = string[i + start];
                /* it at end of string, stop  ●  l
                it (string[i + start] = = '\0') break;
                }
        dest[i] = '\0';
        }
    return dest;
    }
```

5) ฟังก์ชัน atoint ใช้เปลี่ยนสตริงเป็นเลขจำนวนเต็ม เพื่อนำไปใช้วิเคราะห์ เพราะการเก็บข้อมูลเป็นสตริงหรืออัลฟานิวเมอริคทำให้ประหยัดพื้นที่ เมื่อจะนำข้อมูลไปใช้ต้องเปลี่ยนคืนเป็นจำนวนเต็มก่อน

5.6 ตัวอย่างโปรแกรม

โปรแกรมนี้ใช้โต้ตอบกับผู้ใช้ว่าวันที่ที่ผู้ใช้ถามเข้ามานั้นเป็นวันหยุด หรือวันสำคัญของทางราชการ หรือวันนักขัตฤกษ์หรือไม่ ข้อมูลวันสำคัญตามตัวอย่างมีเพียง 5 วัน ขอให้สังเกตการใช้งานของตัวแปรโครงสร้างด้วย

```
struct date {
        char month[10];
        int day;
        int year;
        };
main ( )
/* Determines if date is special */
        {
        int ret,i;
        struct date aday;

        static struct date special[ ] = {
                {"October",25,1925},
                {"November",4,1916},

        {"October",8,1941},
        {"March",29,1916},
        {"February",27,1984}
        };
int size = sizeof (special)/sizeof (struct date);

/* Input the date to be checked */
printf("\nMonth? ");
scanf("%10s",aday.month);
printf("\nDay? ");
scanf("%d",&aday.day);
printf("\nYear? ");
scanf("%d",&aday.year);

/* Check it */
ret = 0;
for (i = 0;i<size;i++)
        {
```

```
        if ( special[i].year = = aday.year
            && special[i].day = = aday.day
            && strcmp(special[i].month,
            aday.month) = = 0 )
            {
            ret = 1;
            break;
            }
        }
if (ret= =0) printf("\n%s %d, %d is not special",
        aday.month,aday.day,aday.year);
else printf("A special day!! %s %d %d",
        aday.month,aday.day,aday.year);
exit(0);
}
```