บทที่ 8 ชนิดข้อมูลและตัวดำเนินการขั้นสูง

ในบทก่อนหน้านี้ เราได้ทำการศึกษาเกี่ยวกับชนิดข้อมูลพื้นฐานของภาษา C อันได้แก่ ตัวอักษร จำนวนเต็ม เลขทศนิยม และอาร์เรย์ เป็นต้น แต่ในการเขียนโปรแกรมบางอย่าง เราจำเป็นที่จะต้องสร้างข้อมูลที่มีโครงสร้างซับซ้อนขึ้น เช่น โปรแกรมจัดการเกี่ยวกับฐานข้อมูล เป็นต้น ซึ่งลำพังชนิดข้อมูลพื้นฐาน หรืออาร์เรย์ ไม่สามารถทำได้ หรืออาจจะทำได้ แต่ไม่สะดวกในการเขียน (ยกตัวอย่าง เช่น การแสดงค่าจำนวนเชิงซ้อนค่าหนึ่งโดยใช้ตัวแปร 2 ตัว เป็นต้น) ด้วยเหตุนี้ จึงทำให้เกิดข้อมูลชนิด structure และ union ขึ้น

ในส่วนของตัวดำเนินการนอกจาก +, -, *, /, % ที่ใช้กับการคำนวณจำนวนเต็มและเลขทศนิยม หรือ ? : สำหรับการกำหนดเงื่อนไข ดังได้ศึกษามาก่อนหน้านี้แล้ว ภาษา C ยังมีตัวดำเนินการที่ใช้กับชนิดข้อมูล structure ดังกล่าวข้างต้น และตัวดำเนินการที่ใช้กับการคำนวณแบบบิต (Bitwise Operation) ซึ่งเป็นการคำนวณโดยคิดทีละ 1 บิต แทนที่จะคำนวณโดยใช้ค่าทั้งจำนวน ความสามารถอันนี้ เป็นจุดเด่นอันหนึ่งที่ทำให้ภาษา C ถูกจัดให้เป็นภาษาที่มีคุณสมบัติ Low-level Language รวมอยู่ด้วย

8.1 Structure

เราใช้อาร์เรย์ในการบันทึกชุดข้อมูลชนิดเดียวกัน แต่ในกรณีที่เราต้องการสร้างชุดข้อมูลที่ประกอบด้วยข้อมูลหลายชนิด ดังตัวอย่างต่อไปนี้

```
Student name              (string 30 characters)
mathematics score               (float)
physics score             (float)
```

เราต้องใช้ข้อมูลชนิดใหม่ที่มีชื่อเรียกว่า *structure* ซึ่งสามารถเก็บข้อมูลหลายชนิดและหลายค่าได้

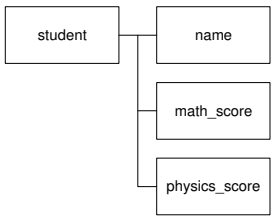
ในอาร์เรย์ ตัวประกอบทั้งหมดจะเป็นข้อมูลชนิดเดียวกัน และจะเรียงตามลำดับ แต่ใน structure ตัวประกอบแต่ละตัวจะมีชื่อเรียกว่า *field* และเป็นข้อมูลต่างชนิดกันได้

โดยทั่วไป การกำหนดนิยามของ *structure* จะทำในลักษณะดังนี้

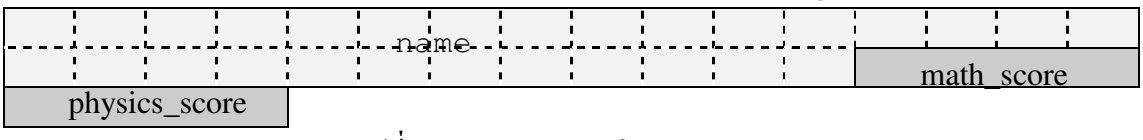
```
struct structure_name {
      field_type1 field_name1 /* comment */
      field_type2 field_name2 /* comment */
      ....
} variable_name ;
```

กล่าวคือ จะใช้คำสงวน (Reserved Word) *struct* แล้วตามด้วยชื่อ *structure* หรือเรียกว่า *tag* จากนั้น ในเครื่องหมาย { } ก็จะเป็นการกำหนดชนิดข้อมูล และชื่อของแต่ละ field ยกตัวอย่างเช่น

```
struct student {
    char name[28] ; /* name of a student */
    float math_score;
      /* mathematics score */
    float   physics_score ;
      /* physics score */
} student1 ;
/* the variable of student type */
```



รูปที่ 8.1 student structure



รูปที่ 8.2 structure student ในหน่วยความจำ

ในการกำหนดนิยามข้างต้น จะเป็นการประกาศให้คอมไพเลอร์ได้รู้เกี่ยวกับโครงสร้างของ Structure *student* ซึ่งเป็นข้อมูลชนิดใหม่ที่เราสร้างขึ้น เพื่อที่จะใช้ในโปรแกรมต่อไป นอกจากนี้ ควบคู่กับการกำหนดนิยามของ Structure ก็ได้ทำการประกาศตัวแปรแบบนี้ที่ชื่อ *student1* ไปด้วย

เราอาจจะใช้ **struct student** ในการ declare ภายหลังได้ โดยทำในลักษณะเช่นเดียวกับการ declare ทั่วไป ดังนี้

```
struct student student2 ; /* the data of student2 */
```

นอกจากนี้ ในการกำหนดนิยามของ structure เราอาจจะไม่กำหนดชื่อของ structure ได้ ดังตัวอย่างต่อไปนี้

```
struct {
       char   name[30] ;          /* student name*/
       float  math_score ; /* mathematics score */
       float  physics_score ;     /* physics score */
} student1 ;
```

ตัวแปรชื่อ *student1* ก็ยังคงถูกประกาศให้เป็นตัวแปรแบบ structure เช่นเคย เพียงแต่ไม่มีชื่อ structure สำหรับ structure แบบนี้เท่านั้น เราเรียก structure ที่ไม่มีชื่อว่า *anonymous structure* ทั้งนี้ เราไม่มีความจำเป็นที่จะต้อง declare ตัวแปรไปพร้อมกับการกำหนดนิยามของ structure กล่าวคือ เรากำหนดนิยามของ Structure ก่อน แล้วค่อยทำการ declare ตัวแปรในภายหลังก็ได้

โดยทั่วไป ในการกำหนดนิยามของ structure (หรือการกำหนดตัวแปร Macro และอื่น ๆ ที่ไว้ใช้ส่วนตัว) เราจะเขียนไว้ใน Header File แล้ว Include เข้ามา และเมื่อจะใช้ เราจะใช้ในลักษณะดังนี้

```
struct student student1 ;
```

การกำหนดค่าเริ่มต้นของ Structure

ตัวอย่างที่ 1:

```
struct student {
       char   name[30] ;
       float  math_score ;
       float  physics_score ;
} ;
struct student student1={"somchai",0,0};
```

ตัวอย่างที่ 2:

```
struct student {
       char   name[30] ;
       float  math_score ;
       float  physics_score ;
} student1 = {"somchai", 0, 0} ;
```

ตัวอย่างที่ 3:: Array of Structures

```
struct student student_list[50] = {
    {"somchai", 0, 0},
    {"somsri", 0, 0},
    {"somsak", 0, 0},
    {"sommai", 0, 0}
} ;
```

8.2 การอ้างอิงถึงฟิลด์ (Field Reference)

ในการอ้างอิงถึงค่าของแต่ละ Field เราจะใช้ Member Operator ซึ่งมีอยู่ด้วยกัน 2 ชนิดใหญ่ ๆ คือ ชนิดที่ใช้กรณีแสดง structure แบบธรรมดา (dot Operator) กับชนิดที่ใช้กรณีที่แสดงโดยใช้ Pointer (Arrow Operator)

ตารางที่ 8.1 Member Operator

ตัวดำเนินการ	ความหมาย
. (dot operator)	Identify field in the case of usage of structure
-> (arrow operator)	Identify field in the case of usage of pointer representation

dot operator

เป็นการอ้างอิงถึงค่าของแต่ละ field โดยมีรูปแบบการใช้ดังนี้

```
variable.field
```

ยกตัวอย่าง เช่น

```
student1.math_score
```

ตัวอย่างที่ 8.1 : ตัวอย่างการใช้ตัวดำเนินการ dot

```
#include <stdio.h>
struct student{                  /* structure "student" definition */
       char   name[30] ;         /* name of a student */
       float  math_score ;       /* the score of mathematics */
       float  physics_score ;    /* the score of physics */
} ;
int main()
{
       struct student student1;/* declaration structure type variable */
       strcpy(student1.name, "somchai") ; /* assignment of field name */
       student1.math_score = 80 ;   /* assignment of field math_score */
       student1.physics_score = 60;
                                  /* assignment of field physics_score */
       /* field reference */
       printf("NAME : %s\n", student1.name) ;
       printf("MATH : %.2f\n", student1.math_score) ;
       printf("PHYSICS : %.2f\n", student1.physics_score) ;
       return 0;
}
```

ผลการรันโปรแกรม

```
NAME : somchai
MATH : 80.00
PHYSICS : 60.00
```

arrow operator

ในกรณีที่ใช้ Pointer ในการแสดง structure ดังตัวอย่างต่อไปนี้

```
struct student student1 /* declaration of structure type variable */
struct student *ptr_student1 = &student1 ;
/* declaration of structure type pointer variable and let ptr_student1 point
to student1 */
```

ถ้าเราจะกำหนดให้ค่าของ *math_score* เป็น 80 ถ้าดูผิวเผินแล้ว อาจจะมองเห็นว่า ประโยคคำสั่งต่อไปนี้เป็นสิ่งที่ถูกต้อง

```
       *ptr_student1.math_score = 80 ;
```

แต่เมื่อพิจารณาถึงกฎลำดับก่อนหลัง (Precedence Rule) ของ Operator แล้ว จะเห็นว่า Priority ของ . (dot) จะสูงกว่า *(dereference) ดังนั้น ประโยคคำสั่งนี้จะมีความหมายเป็น

```
       *(ptr_student1.math_score) = 80 ;
```

ซึ่งต่างจากที่เราต้องการ (เราต้องการให้ค่าของ field math_score ของ structure ที่ ptr_student ชี้อยู่มีค่าเท่ากับ 80) ดังนั้น เราจะต้องทำในลักษณะดังต่อไปนี้

```
       (*ptr_student1).math_score = 80 ;
```

ซึ่งยุ่งยากและสับสน Arrow Operator มีไว้เพื่อขจัดปัญหาความยุ่งยากที่ว่านี้ กล่าวคือ ถ้าเราใช้ Arrow Operator เราจะเขียนประโยคคำสั่งข้างต้นให้อยู่ในรูป

```
       ptr_student1->math_score = 80 ;
```

ซึ่งง่ายต่อการเขียน-อ่านโปรแกรมยิ่งขึ้น

```
Usage :      pointer_to_structure->field
```

ตัวอย่างที่ 8.2 : ตัวอย่างการใช้ตัวดำเนินการ arrow

```
#include <stdio.h>
struct student{
       char   name[30] ;    /* name of a student */
       float  math_score ;  /* the score of mathematics */
       float  physics_score ;     /* the score of physics */
} ;
int main()
{
```

```
        struct student student1 ;
        struct student *ptr_student1 = &student1 ;
                    /* Let ptr_student1 point to student1 */
        strcpy(ptr_student1->name, "somchai") ;
                    /* Usage of arrow operator */
        ptr_student1->math_score = 80 ;
        (*ptr_student1).physics_score = 60 ;
                    /* In the case of usage of dot operator */
        printf("NAME : %s\n", ptr_student->name) ;
        printf("MATH : %.2f\n", ptr_student1->math_score) ;
        printf("PHYSICS : %.2f\n", ptr_student1->physics_score) ;
        return 0;
}
```

ผลการรันโปรแกรม เหมือนกับตัวอย่าง 8.1

8.3 การคัดลอก structure (Copy of structure)

การคัดลอก structure ทำได้ 2 วิธีคือ

1. คัดลอกโดยใช้ structure

```
        struct student student1, student2, student3, slist[50] ;
        ...
        student1 = student2 ;
        student3 = slist[25] ;
        slist[0] = slist[40] ;
```

1. คัดลอกโดยใช้ pointer to structure

```
        struct student student1, slist[50] ;
        struct student *ps1, *ps2 ;
        ps1 = &student1 ;   /* let ps1 point to student1 */
        ps2 = slist ;
        /* let ps2 point to (the first element of) slist */
```

ตัวอย่างที่ 8.3 : ตัวอย่างการคัดลอก structure และอาร์เรย์ของ structure

```
#include <stdio.h>
#include <string.h>
struct student{
        char   name[30] ;    /* name of a student */
        float  math_score ; /* the score of mathematics */
        float  physics_score ;     /* the score of physics */
} ;
int main()
{
        int    i;
        struct student slist[10] ;    /* Array of Structures */
        struct student *sp ;         /* Pointer to structure */
        sp = slist ;        /* sp points to slist (array's first element)*/
        strcpy(sp->name, "somchai") ;     /* First Element */
        sp->math_score = 80 ;
        sp->physics_score = 60 ;
        ++sp ;
        strcpy(sp->name, "somsri") ;      /* Second Element */
        sp->math_score = 55 ;
        sp->physics_score = 70 ;
        slist[2] = slist[1] ;             /* Copy slist[1] to slist[2] */
        slist[3].math_score = -1 ;        /* Mark breakpoint */
        for (i = 0 ; i < 3 ; i++)
              printf("%d : %s %.2f %.2f\n", i, slist[i].name,
                          slist[i].math_score, slist[i].physics_score) ;
        sp = slist ;      /* sp points to first element of array again */
        i = 0 ;
        while (sp->math_score != -1) {
              printf("%d : %s %.2f %.2f\n", i, sp->name,
                                  sp->math_score, sp->physics_score) ;
              ++sp ;  ++i ;
        }
        return 0;
}
```

ผลการรันโปรแกรม

```
0 : somchai 80.00 60.00
1 : somsri 55.00 70.00
2 : somsri 55.00 70.00
0 : somchai 80.00 60.00
1 : somsri 55.00 70.00
2 : somsri 55.00 70.00
```

8.4 การใช้ structure เป็นอาร์กิวเมนต์และค่าส่งคืนของฟังก์ชัน

เช่นเดียวกับข้อมูลชนิดอื่น ๆ เราสามารถใช้ structure เป็นอาร์กิวเมนต์ของฟังก์ชันได้ โดยทำในลักษณะเดียวกับข้อมูลชนิดอื่น ๆ กล่าวคือ

1. Call by value โดยมีรูปแบบดังนี้

```
function_a (... , struct student  abc , ...)
```

2. Call by reference โดยมีรูปแบบดังนี้

```
function_b (..., struct student *p_abc , ...)
```

แต่เนื่องจาก structure เป็นข้อมูลที่มีขนาดใหญ่ ถ้าหากใช้วิธี Call by value อาจจะเสียเวลาในการส่งค่ามาก โดยทั่วไป จึงนิยมใช้วิธี Call by reference ในการรับส่งค่ามากกว่า

ส่วนการส่งคืนค่า structure จาก function ก็ทำวิธีเดียวกับตัวแปรชนิดอื่น ๆ โดยกำหนดให้ชนิดข้อมูลของค่าส่งคืนเป็นข้อมูลแบบ structure

ตัวอย่างที่ 8.4 ตัวอย่างการใช้ structure เป็นอาร์กิวเมนต์

```
#include <stdio.h>
struct student{
       char   name[30] ;    /* name of a student */
       float  math_score ;  /* the score of mathematics */
       float  physics_score ;     /* the score of physics */
} ;
void display1(struct student abc) ;
void display2(struct student *p_abc) ;
int main()
{
       struct student a = {"Somchai", 90, 80} ;
       display1(a) ;
       display2(&a) ;
       return 0;
}
void display1(struct student abc)
{
       printf("Name %s Math %.2f Physics %.2f\n",
                     abc.name, abc.math_score, abc.physics_score) ;
}
void display2(struct student *p_abc)
{
       printf("Name %s Math %.2f Physics %.2f\n",
                     p_abc->name, p_abc->math_score, p_abc->physics_score) ;
}
```

ผลการรันโปรแกรม

```
Name Somchai Math 90.00 Physics 80.00
Name Somchai Math 90.00 Physics 80.00
```

8.5 Structure in Structure

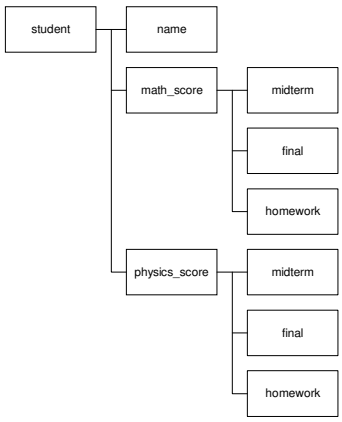
เราสามารถใช้ structure เป็นชนิดของข้อมูลใน field ของ Structure อีกอันหนึ่งได้ ยกตัวอย่าง เช่น ถ้าเราต้องการให้ field math_score และ physics_score สามารถบันทึกข้อมูลได้ละเอียดยิ่งขึ้น โดยสามารถเก็บคะแนน

Midterm, Final และคะแนนการบ้านได้ เราทำได้โดยสร้าง structure ที่สามารถเก็บรายละเอียดที่ว่านี้ แล้วนำไปบรรจุเป็น field หนึ่งใน student

ตัวอย่างที่ 8.5 ตัวอย่างการใช้ structure in structure

```
#include <stdio.h>
#include <string.h>
struct score {
       float  midterm ;
       float  final ;
       float  homework ;
} ;
struct student{
       char   name[30] ;          /* name of a student */
       struct score  math_score ;        /* the score of mathematics */
       struct score  physics_score ;     /* the score of physics */
} ;
int main()
{
       float total ;
       struct student st1, *pt1  = &st1 ;
       /* structure representation */
       strcpy(st1.name, "somchai") ;
       st1.math_score.midterm = 30 ;
       st1.math_score.final = 40 ;
       st1.math_score.homework = 10 ;
       /* pointer representation */
       pt1->physics_score.midterm = 20 ;
       pt1->physics_score.final = 35 ;
       pt1->physics_score.homework = 5 ;
       /* display output */
       printf("student : %s\n", pt1->name) ;
       printf("mathematics score :\tmidterm %.2f\n",pt1->math_score.midterm);
       printf("\t\t\tfinal   %.2f\n", pt1->math_score.final) ;
       printf("\t\t\thomework %.2f\n", pt1->math_score.homework) ;
       total = st1.math_score.midterm + st1.math_score.final +
              st1.math_score.homework ;
       printf("\t\t\ttotal   %.2f\n", total) ;
       printf("physics score : midterm %.2f\n", pt1->physics_score.midterm) ;
       printf("\t\tfinal   %.2f\n", pt1->physics_score.final) ;
       printf("\t\thomework %.2f\n", pt1->physics_score.homework) ;
       total = st1.physics_score.midterm + st1.physics_score.final +
              st1.physics_score.homework ;
       printf("\t\ttotal   %.2f\n", total) ;
       return 0;
}
```

```
struct score {
   float midterm ;
   float final ;
   float homework ;
}
struct student {
   char name[28] ;
   struct score     math_score ;
   struct score     physics_score ;
};
```



รูปที่ 8.3 เวอร์ชันใหม่ของ student structure

ผลการรันโปรแกรม

```
student : somchai
mathematics score :    midterm  30.00
                       final    40.00
                       homework 10.00
                       total    80.00
physics score : midterm 20.00
                final   35.00
                homework 5.00
                total   60.00
```

8.6 คำสั่ง typedef

คำสั่ง **typedef** มีไว้เพื่อให้ผู้เขียนโปรแกรมสามารถกำหนดนิยามของชนิดของข้อมูลเองได้ นอกเหนือจากชนิดของข้อมูลพื้นฐานที่ภาษา C มีให้ (e.g. char, int, float, double etc.) เพื่อให้ง่ายต่อการเขียนโปรแกรมยิ่งขึ้น

```
Usage : typedef type new-type
```

type จะเหมือนกับชนิดของข้อมูลที่ใช้ตอนประกาศตัวแปร (หรือพูดง่าย ๆ คือ จะต้องเป็นชนิดของข้อมูลที่มีในภาษา C ซึ่งรวมไปถึงอาร์เรย์และ Structure) ส่วน *new-type* จะเป็นชื่อชนิดของข้อมูลที่เรากำหนดขึ้น ตัวอย่าง เช่น

```
typedef int boolean ;
```

จะหมายถึง การกำหนดนิยามของชนิดข้อมูลชนิดใหม่โดยใช้ชื่อว่า **boolean** ให้เป็นเหมือนกับ int ดังนั้น การ declare ต่อไปนี้

```
boolean  flag ;
```

จะมีความหมายเหมือนกับ

```
int  flag ;
```

แต่คำที่ใช้ในโปรแกรมจะสื่อความหมายดีขึ้น

ตัวอย่างที่ 8.6 : ตัวอย่างการใช้คำสั่ง typedef

```
#include <stdio.h>
int main()
{
  int   i ;
  typedef int group[10] ;/* Create a new type "group" */
  group totals ;/* Use the new type for a variable */
  for (i = 0 ; i < 10 ; i++) {
    totals[i] = 0 ;
  }
  for (i = 0 ; i < 10 ; i++)
    printf("%d ", totals[i]) ;
  printf("\n") ;
  return 0;
}
```

นอกจากวิธีใช้ที่กล่าวมาแล้ว *typedef* นิยมใช้ในการกำหนดชื่อของชนิดของข้อมูลที่เป็น structure เช่น

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

date จะหมายถึงข้อมูลที่เป็น structure ซึ่งประกอบด้วย 3 field เวลาใช้ เราจะใช้ในลักษณะดังต่อไปนี้

```
date   birthday, holiday[20]  ;
```

หมายเหตุ การกำหนดนิยามเหล่านี้มักจะเขียนไว้ที่ส่วนหัวของโปรแกรม หรือไม่ก็เขียนในไฟล์เฮดเดอร์

typedef VS #define

typedef จะคล้ายกับ **#define** ลองพิจารณาตัวอย่างต่อไปนี้

```
#define INT32 long int     /* 32 bit signed integer type */
typedef long int int32     /* 32 bit signed integer */
```

ตัวอย่างนี้ จะให้ความหมายที่เหมือนกัน แต่ลักษณะการแปลความหมายของคอมไพเลอร์จะต่างกันอย่างสิ้นเชิง โดย #define จะหมายถึงการแทนที่ INT32 ในโปรแกรมให้เป็น long int ในขณะที่ **typedef** เป็นการกำหนดนิยามของชนิดข้อมูลชนิดใหม่โดยให้เหมือนกับ long int ดังนั้น ในการกำหนดนิยามลักษณะนี้ ใช้ **typedef** จะดีกว่า เพราะเป็น*การกำหนดนิยาม* มิใช่เป็นเพียงแต่*การเปลี่ยนคำ* พิจารณาตัวอย่างต่อไปนี้

```
#define INT_PTR int *      /* Define a pointer to integer */
typedef int * int_ptr ;    /* Define a pointer to an integer */
INT_PTR ptr1, ptr2 ;       /* This contains a problem */
int_ptr ptr3, ptr4 ;       /* This does not */
```

ปัญหาจะเกิดขึ้นเมื่อคอมไพเลอร์ทำการแปลความหมายของบรรทัด INT_PTR ptr1, ptr2 จะกลายเป็น

```
/* expanded */
int * ptr1, ptr2 ;          /* This contains a problem */
```

ซึ่งหมายถึงว่า ptr2 จะไม่เป็น pointer เราสามารถขจัดปัญหานี้ไปได้โดยใช้ **typedef** นอกจากจุดนี้แล้ว **typedef** ยังสามารถใช้ในการสร้างชนิดของข้อมูลที่ซับซ้อน ซึ่ง **#define** ไม่สามารถทำได้ เช่น

```
typedef      int  group[10] ;
```

จะหมายถึงชนิดของข้อมูลชื่อ **group** ซึ่งหมายถึงอาร์เรย์ที่มีตัวประกอบเป็น integer 10 ตัว ดังแสดงไว้ในตัวอย่างที่ 8.6

ข้อสรุปเกี่ยวกับ typedef & #define :

- **typedef** ใช้ในการกำหนดนิยามของชนิดของข้อมูล
- **#define** ใช้กำหนดนิยามของ MACRO ที่ไม่ซับซ้อน และเพื่อให้สะดวกต่อการเขียนโปรแกรม

8.7 Self-Referential Structure

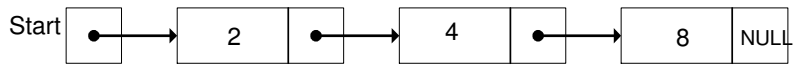
ในบางกรณี การกำหนดให้มี field หนึ่งที่สามารถบันทึก Pointer ที่ชี้ไปยังข้อมูลที่เป็น structure แบบเดียวกันจะเป็นประโยชน์ในการเขียนโปรแกรมมาก ซึ่งเราเรียก structure แบบนี้ว่า Self-Referential Structure โดยทั่วไปจะเขียนในลักษณะดังต่อไปนี้

```
struct  structure_name {
        field_type1   field_name1 ;
        field_type2   field_name2 ;
        ...
        struct structure_name      *ptr_name ;
} ;
```

โดย *ptr_name* จะเป็นชื่อ Pointer ที่ชี้ไปที่ Structure

ตัวอย่างการใช้ Self-Referential Structure

Linked List Linked List เป็น Abstract Data Type ชนิดหนึ่ง ซึ่งนิยมใช้กันมากในการแสดงข้อมูลที่มีความสัมพันธ์ในลักษณะต่อเนื่องกัน โดยมี Pointer สำหรับชี้ไปที่ข้อมูลตัวต่อไปดังรูปที่ 8.4

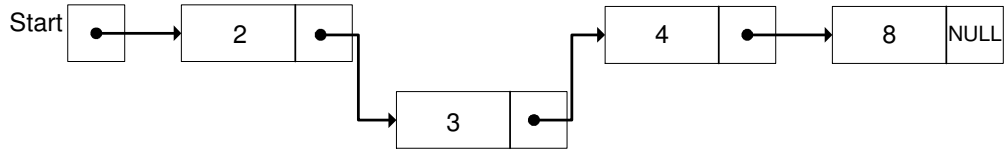


รูปที่ 8.4 Linked List

เราสามารถแสดงข้อมูลตามรูปที่ 8.4 โดยใช้ Self-Referential Structure ในลักษณะดังนี้

```
struct list_element {
        int    value ;
        struct list_element *next ;
} ;
```

ในการ Insert ข้อมูลลงไปใน Linked List เราทำได้ในลักษณะดังรูปที่ 8.5



รูปที่ 8.5 Insert an element into Linked-list

ซึ่งเราสามารถเขียนเป็น Function ได้ดังนี้

```
typedef struct list_element LIST_ELEMENT
void insert_element(LIST_ELEMENT *list, LIST_ELEMENT *element, int position)
{
        LIST_ELEMENT        *tmp ;
        tmp = (list + position)->next ;
```

```
        (list + position)->next = element ;
        element->next = tmp ;
}
```

คำถาม เขียน Function สำหรับการลบ LIST_ELEMENT

เมื่อเปรียบเทียบการแสดงข้อมูลด้วย Linked List กับอาร์เรย์ธรรมดาแล้ว จะเห็นว่า Linked List จะใช้เนื้อที่ในการเก็บบันทึกข้อมูลมากกว่า แต่มีข้อดีคือ สะดวกในการจัดการกับข้อมูล เช่น การเพิ่ม การลบ เป็นต้น และสามารถรวมข้อมูลหลาย ๆ อย่าง ให้เป็นหมวดหมู่ได้ดีกว่าอาร์เรย์ธรรมดา

Doubly Linked List

Doubly Linked List มีลักษณะคล้ายกับ Linked List ต่างกันเพียงแต่ว่า Doubly Linked List จะมี Pointer ชี้ไปยังข้อมูลก่อนหน้าด้วยดังรูปที่ 8.6

รูปที่ 8.6 Doubly Linked List

8.8 union

union มีลักษณะคล้าย structure แต่ลักษณะการใช้เนื้อที่ในหน่วยความจำจะต่างกัน โดย structure จะจองเนื้อที่ไว้สำหรับทุก ๆ field เช่น ถ้าเกิดประกอบด้วย field ที่เป็น integer 2 field ดังตัวอย่างต่อไปนี้

```
struct rectangle {
        float  width ;
        float  length ;
} ;
```

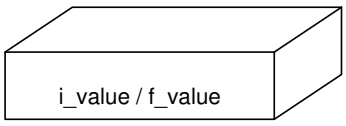
โปรแกรมจะทำการจองเนื้อที่ 8 Bytes สำหรับที่จะบันทึกค่าของ structure นี้ (รูปที่ 8.8) แต่ union จะใช้เนื้อที่ในหน่วยความจำเดียวกัน ซึ่งหมายความว่า จะไม่สามารถบันทึกข้อมูลของแต่ละ field ในเวลาเดียวกันได้ union จะใช้ในกรณีที่ไม่มีความจำเป็นต้องบันทึกค่าของแต่ละ field พร้อมกัน และต้องการประหยัดหน่วยความจำ ส่วน union จะจองเนื้อที่เท่าไรในหน่วยความจำนั้น ขึ้นอยู่กับความยาวสูงสุดของ field ใน union รูปแบบในการ declare union จะเหมือน structure โดยเป็นดังต่อไปนี้

```
union union_name {
        field_type field_name    /* comment */
        field_type field_name    /* comment */
        ....
} variable_name ;
```

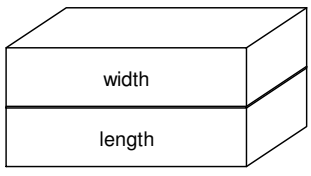
ตัวอย่าง เช่น

```
union value {
        long int i_value ;  /* integer version of value */
        float f_value ;             /* floating version of value */
}
```

หน่วยความจำที่ใช้ในการบันทึกค่า i_value และ f_value จะเป็นส่วนเดียวกัน (รูปที่ 8.7)



รูปที่ 8.7 value Union Layout



รูปที่ 8.8 rectangle Structure Layout

ตัวอย่างที่ 8.7 : ตัวอย่างการใช้ union

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

```
union value {
       long int i_value ;         /* integer version of value */
       float f_value ;            /* floating version of value */
} data ;
int   i ;                   /* Random integer */
float  f ;                  /* Random floating-point number */
int main()
{
       data.f_value = 5.0 ;
       data.i_value = 3 ;         /* data.f_value overwritten */
       i = data.i_value ;         /* legal */
       f = data.f_value ;  /* not legal, will generate unexpected results */
       data.f_value = 5.5 ;/* put something in f_value/clobber i_value */
       i = data.i_value ;  /* not legal, will generate unexpected results */
       return 0;
}
```

8.9 enum Type

enum type ย่อมาจาก enumerated data type ถูกออกแบบสำหรับตัวแปรที่บันทึกข้อมูลที่มีจำนวนจำกัด โดยจะอ้างอิงถึงค่าของตัวแปรโดย name (tag) และคอมไพเลอร์จะกำหนดค่าของแต่ละ tag ให้เป็นเลข integer โดยเรียงตามลำดับ ตัวอย่างเช่น วันใน 1 สัปดาห์ ถ้าเราใช้ **#define** ในการกำหนดค่าสำหรับ week_day ดังนี้

```
#define week_day int /* define the type for week_days */
#define SUNDAY 0
#define MONDAY 1
#define TUESDAY 2
#define WEDNESDAY 3
#define THURSDAY 4
#define FRIDAY 5
#define SATURDAY 6
week_day today = TUESDAY ;
```

เราจะเห็นได้ว่า วิธีนี้มีความยุ่งยากมาก เราสามารถทำให้สะดวกกว่านี้ได้ โดยใช้ **enum** type

```
enum week_day {SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
               FRIDAY, SATURDAY} ;
enum week_day today = TUESDAY ;
```

รูปแบบทั่วไปในการใช้คำสั่ง **enum** จะเป็นดังนี้

```
enum enum-name { tag-1, tag-2,  . . . } variable-name
```

เหมือนกับ structure เราสามารถไม่เขียน *enum-name* หรือ *variable-name* ในการ declare ได้ อนึ่ง *tag* จะใช้ในการอ่านค่า ซึ่งโดยปกติ จะถูกแสดงด้วยตัวอักษรตัวพิมพ์ใหญ่

ข้อดีประการหนึ่งของ **enum** type คือ C จะจำกัดค่าของตัวแปรที่ถูก declare ให้เป็นแบบ **enum** ให้เป็นไปตาม declaration ตัวอย่างต่อไนี้ จะเกิด Error ในการ Compile

```
today = 5 ;   /*  5 is not a week_day */
```

ข้อเสียประการหนึ่งของ **enum** คือไม่สามารถใช้ค่าของตัวแปรแบบ **enum** เป็น index ของอาร์เรย์ได้

```
enum week_day today = TUESDAY ;
char day_names[7][] = {
       "Sunday",    "Monday",    "Tuesday",    "Wednesday",    "Thursday",
       "Friday", "Saturday" } ;
/* The following line generates a warning because today is not an
integer */
printf("Today is %s\n", day_names[today]) ;
```

ในการแก้ปัญหาที่เกิดขึ้นนี้ เราสามารถทำได้โดยบอกให้คอมไพเลอร์ตีความหมายของตัวแปร **today** เป็น integer กล่าวคือใช้ Type Conversion เปลี่ยนชนิดของตัวแปร **today** ให้เป็น integer ดังประโยคคำสั่งต่อไปนี้

```
printf("Today is %s\n", day_names[(int)today]) ;
```

8.10 ตัวดำเนินการขั้นสูง (Advanced Operator)

Bitwise operator

เหตุผลประการหนึ่งที่ทำให้ภาษา C ถูกจัดเป็น Low-Level Language ก็เพราะสามารถจัดการกับข้อมูลในคอมพิวเตอร์ที่เป็นเลขฐานสองในลักษณะเป็น bit ได้ ซึ่งเราเรียกว่า Bitwise Operation ยกตัวอย่าง เช่น ถ้าเป็น int โดยทั่วไป เราจะอ่านค่า 32 bits (4 Bytes) เป็นค่าของตัวแปรตัวนั้น แต่ใน Bitwise operator เราจะทำการอ่านค่าเป็น bit ซึ่งจะได้ค่าเป็น 32 ค่า ที่เป็นอิสระต่อกัน ส่วนในการคำนวณ อย่างเช่น ในการบวกเลขจำนวนเต็มทั่วไป จะทำการบวกค่าของเลขทั้งจำนวน ซึ่งทำให้ต้องใช้ค่าของทุก bit มาใช้ในการคำนวณ แต่การบวกเลขใน Bitwise Operation จะทำการบวก Bit ต่อ Bit โดยไม่คำนึงว่า bit อื่น ๆ จะเป็นอย่างไร

ตัวดำเนินการ	ความหมาย	ตัวอย่าง
&	Bitwise and	a & b
|	Bitwise or	a | b
^	Bitwise exclusive or	a ^ b
~	One's Complement	~a
<<	Shift left	a << 2 (เท่ากับ a * 4)
>>	Shift right	a >> 3 (เท่ากับ a / 8)

ผลที่ได้

Bit1	Bit2	Bit1 & Bit2	Bit1 | Bit2	~Bit1	Bit1 ^ Bit2
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

ตัวอย่าง : สมมุติให้ a = 0x45, b = 0x71

a & b = 0x41 a | b = 0x75 a^b = 0x34

~a = 0xBA a << 1 = 0x8A a >> 2 = 0x11

ตัวอย่างที่ 8.8 : ตัวอย่างการใช้ตัวดำเนินการแบบบิต (bitwise operator)

```
#include <stdio.h>
int main()
{
  char   c1, c2 ;
  c1 = 0x45 ;/* represent character by hexadecimal number */
  c2 = 0x71 ;/* represent character by hexadecimal number */
  printf("Result of %#X & %#X = %#X \n", c1, c2, c1 & c2) ;
  printf("Result of %#X | %#X = %#X \n", c1, c2, c1 | c2) ;
  printf("Result of %#X ^ %#X = %#X \n", c1, c2, c1 ^ c2) ;
  printf("Result of ~%#X = %#X \n", c1, ~c1) ;
  printf("Result of %#X << 1 = %#X\n", c1, c1 << 1) ;
  printf("Result of %#X >> 2 = %#X\n", c1, c1 >> 2) ;
  return 0;
}
```

ผลการรันโปรแกรม

```
Result of 0X45 & 0X71 = 0X41
Result of 0X45 | 0X71 = 0X75
Result of 0X45 ^ 0X71 = 0X34
Result of ~0X45 = 0XFFFFFFBA
Result of 0X45 << 1 = 0X8A
Result of 0X45 >> 2 = 0X11
```

ตัวอย่างที่ 8.9 : ตัวอย่างแสดงความแตกต่างระหว่างตัวดำเนินการ & กับ &&

```
#include <stdio.h>
int main()
{
       int    i1, i2 ;            /* two random integers */
       i1 = 4 ;
       i2 = 2 ;                   /* set values */
       /* Nice way of writing the conditional */
       if ((i1 != 0) && (i2 != 0))
             printf("Both are not zero #1\n") ;
       /* shorthand way of doing the same thing */
       if (i1 && i2)
             printf("Both are not zero #2\n") ;
       /* Incorrect use of bitwise and resulting in an error */
       if (i1 & i2)
             printf("Both are not zero #3\n") ;
       return 0;
}
```

คำถาม : ทดลองรันโปรแกรมนี้ สังเกตผลที่ได้ และพิจารณาสาเหตุ

, (comma) Operator

Comma Operator ช่วยให้เราสามารถเขียนประโยคคำสั่งเป็น group ได้ ดังตัวอย่างต่อไปนี้

```
if (total > 0) {
       printf("You owe nothing\n") ;
       total = 0 ;
}
```

เราสามารถเขียนในลักษณะดังนี้

```
if (total > 0)
       printf("You owe nothing\n") ,  total = 0 ;
```

โดยทั่วไป เราควรจะใช้เครื่องหมายวงเล็บปีกกา ({}) แทนการใช้ Comma Operator มีเพียงกรณีเดียวเท่านั้นที่ใช้ Comma Operator อย่างเป็นประโยชน์ก็คือ ในประโยค **for** กรณีที่ต้องการใช้ตัวแปรในการควบคุมการทำซ้ำมากกว่า 1 ดังตัวอย่างต่อไปนี้

```
for (two = 0, three = 0 ;
      two < 10 ;
      two += 2, three += 3)
       printf("%d %d\n", two, three) ;
```

แบบฝึกหัด

1. จงตอบคำถามเกี่ยวกับ Structure ต่อไปนี้

```
struct movie {
       int   id ;
       char  type[30] ;
       char  name[30] ;
       intt  year ;
} ;
```

a) จงทำการประกาศตัวแปรที่เป็น structure แบบนี้ โดยให้มีชื่อว่า tmp และทำการประกาศอาร์เรย์ของ Structure แบบนี้ ที่มีตัวประกอบได้ 100 ตัว และให้ชื่อว่า list

b) จงบอกค่าของ sizeof(struct animal), sizeof(tmp) และ sizeof(list[0].name) ทั้งนี้ สมมุติให้ตัวแปร int 1 ตัว มีขนาด 4 ไบต์

c) จงแสดงวิธีการกำหนดค่าของ field ใน tmp ให้มีค่าดังนี้ id : 5, type : Comedy, name : The Parent Trap, year : 1998

d) จงเขียนฟังก์ชันที่ใช้สำหรับรับค่าของ field ใน Structure แบบนี้

e) จงเขียนฟังก์ชันสำหรับแสดงค่าของ field ใน Structure แบบนี้

2. จงบอกข้อผิดพลาดในโค้ดต่อไปนี้

```
struct wreck {
       char* flotsam ;
```

```
        char* jetsam ;
        int   number_of_crew_saved ;
    } wreck1 ;
    flotsam = "a cracked flower pot" ;
    jetsam = "two Persian carpets" ;
```

3. จงบอกข้อผิดพลาดในโค้ดต่อไปนี้

```
    typedef struct book {
        char  title[25] ;
        char  author[50] ;
        char  isbn[11] ;
        char  date_of_publication[8] ;
        float price ;
    } BOOK books[200] ;
```

4. จงพิจารณาว่าโค้ดต่อไปนี้ ถูกต้องหรือไม่

```
    typedef struct {
        char  type[20] ;
        char  composer[50] ;
        char  name[50] ;
        int   year ;
        int   number ;
    } composition ;
    ...
    composition brahm1 =
    { "Instrumental", "Johannes Brahms", "Piano Sonata No.1" } ;
```

5. จงเขียนคำสั่งสำหรับสร้างชนิดข้อมูลชนิดใหม่ขึ้นมา โดยให้ชื่อว่า *student_data* และให้สามารถเก็บข้อมูลชื่อ ภูมิลำเนา วันเดือนปีเกิด และอายุได้ โดยในส่วนที่ใช้เก็บวันเดือนปีเกิด ให้ใช้ชนิดข้อมูล date ที่ทำการกำหนดโดยคำสั่งต่อไปนี้

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

6. จงเขียนฟังก์ชันสำหรับแสดงค่าของ field ใน *student_data* ในข้อ 6 โดยให้กำหนด Argument ของฟังก์ชันเป็น Pointer ที่ชี้ไปที่ชนิดข้อมูล *student_data*
7. จงเขียนคำสั่งสำหรับกำหนดนิยามของข้อมูลชนิดใหม่ที่ใช้สำหรับแทนค่าของจำนวนเชิงซ้อน
8. จงเขียนฟังก์ชันสำหรับคำนวณจำนวนเชิงซ้อนต่อไปนี้ โดยให้รับค่า Argument เป็น Structure ที่ใช้แทนค่าจำนวนเชิงซ้อนที่กำหนดนิยามในข้อ 7
    - add_c สำหรับบวกจำนวนเชิงซ้อน 2 จำนวน
    - sub_c สำหรับคำนวณผลลบของจำนวนเชิงซ้อน
    - mul_c สำหรับคำนวณผลคูณของจำนวนเชิงซ้อน
    - div_c สำหรับคำนวณผลหารของจำนวนเชิงซ้อน
    - read_c สำหรับรับค่าจำนวนเชิงซ้อน
    - print_c สำหรับแสดงค่าจำนวนเชิงซ้อน
    - abs_c สำหรับคำนวณหาค่า $|c|$
    - exp_c สำหรับคำนวณหาค่า e^c โดย c เป็นจำนวนเชิงซ้อน
    - sqrt_c สำหรับคำนวณหาค่า Square Root ของ c
9. จงสร้าง Structure สำหรับแสดงจุด ใน Space ที่มี 3 มิติ แล้วสร้างฟังก์ชันต่อไปนี้
    - distance สำหรับคำนวณหาระยะห่างระหว่างจุด 2 จุด

- length สำหรับคำนวณหาความยาวของ Vector *ov* โดยให้ *o* เป็นจุดเริ่มต้น (Origin หรือ (0,0,0)) และ *v* เป็นจุดใดจุดหนึ่งใน Space
- dot สำหรับคำนวณ Dot Product ของ Vector *ov* กับ *ow*
- cross สำหรับคำนวณ Cross Product ของ Vector *ov* กับ *ow*
- angle สำหรับคำนวณมุมระหว่าง Vector *ov* กับ *ow*

10. จงใช้ structure สร้าง Data Structure ที่สามารถเก็บข้อมูลแสดงโครงสร้างของ Directory แล้วทดลองเขียนโปรแกรมจัดการระบบ Directory
11. จงเขียนโปรแกรมสำหรับคำนวณผลบวก ลบ คูณ และหา Inverse Matrix ของ Matrix ที่มีตัวประกอบเป็นจำนวนเชิงซ้อน
12. เขียนโปรแกรมสำหรับเก็บข้อมูลเกี่ยวกับเพื่อน เช่น ชื่อ ที่อยู่ เบอร์โทรศัพท์ วันเกิด เป็นต้น
13. จัดทำข้อมูลในแบบฝึกหัดข้อ 12 ให้อยู่ในรูป Linked-list เพื่อให้ง่ายต่อการเพิ่มหรือลบข้อมูล
14. เขียนโปรแกรมสำหรับเก็บข้อมูลเกี่ยวกับจังหวัดต่าง ๆ ในประเทศไทย
15. เขียนโปรแกรมสำหรับเก็บข้อมูลเกี่ยวกับผลการเรียนของตัวเอง อันประกอบด้วย รหัสวิชา, ชื่อวิชา, หน่วยกิต, เกรดที่ได้, ภาคการศึกษาที่เรียน
16. เขียนโปรแกรมสำหรับจัดเก็บและแสดงตารางสอน ซึ่งประกอบด้วย รหัสวิชา, ชื่อวิชา, วัน, เวลาเริ่ม-เลิก, อาจารย์ผู้สอน, ห้องเรียน