ជំពូកទី ៩

អំពី Structure និង Union

១. លក្ខណៈ:ទូទៅនៃការបង្កើតប្រភេទទិន្នន័យថ្មី

យើងទើបបានសិក្សារួចហើយនូវ array ដែលអាចឲ្យគេកំណត់ប្រភេទអញ្ញាតដែលអាចផ្ទុកទិន្នន័យប្រភេទដូចគ្នាជាច្រើន។ structure មានលក្ខណៈស្រដៀងគ្នានេះដែរ វាជាប្រភេទទិន្នន័យកំណត់ដោយអ្នកប្រើប្រាស់មួយបែបផ្សេងទៀតក្នុងភាសា C ដែលអាចឲ្យគេបន្សំគ្នានូវប្រភេទទិន្នន័យផ្សេងគ្នា។

structure ត្រូវបានប្រើដោយតាងឲ្យ record មួយ។ ឧបមាថា គេចង់រក្សាកំណត់ត្រាសៀវភៅនៅក្នុងបណ្ណាល័យមួយ។ យើងចង់កំណត់ត្រាលក្ខណៈសៀវភៅនិមួយៗដូចខាងក្រោម៖

- Title
- Author
- Subject
- Book ID

២. និយមន័យ structure

ដើម្បីកំណត់នូវ structure មួយ គេត្រូវប្រើឃ្លា struct។ ឃ្លា struct កំណត់នូវប្រភេទថ្មី ដែលមាន member ច្រើនជាងមួយ។ ទម្រង់ទូទៅរបស់ឃ្លា struct មានដូចខាងក្រោម៖

```
struct [structure tag] {
   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

structure tag គឺជាជម្រើសមួយអាចប្រើក៏បាន មិនប្រើក៏បាន ហើយ member definition គឺជាអញ្ញាតធម្មតា ដូចជា int i, ឬ float f; ឬ អញ្ញាតណាមួយផ្សេងទៀត។ នៅខាងចុងនៃ structure ពីមុខសញ្ញា ; យើងអាចកំណត់អញ្ញាត structure មួយឬច្រើន ប៉ុន្តែវាក៏មានលក្ខណៈជាជម្រើសដែរ បានន័យថា ប្រើក៏បាន មិនប្រើក៏បាន។ code ខាងក្រោមនេះគឺជាវិធីដែលគេប្រកាស Book structure ៖

```
struct Books {
   char title[50];
   char author[50];
   char subject[100];
   int book_id;
} book;
```

ឬ

```
struct Books {
   char title[50];
   char author[50];
   char subject[100];
   int book_id;
};
struct Books book;
```

៣. <u>ការចូលប្រើ</u> member <u>របស់</u> structure

ការចូលប្រើ member របស់ structure មួយ យើងប្រើសញ្ញាណនព្វន្ត . (ចំណុច)។ សញ្ញាណនព្វន្តនេះត្រូវបានសរសេរក្នុង code ជាចំណុចរវាងឈ្មោះអញ្ញាត structure និង member របស់ structure ដែលយើងចង់ចូលប្រើ។ យើងនឹងប្រើពាក្យ struct ដើម្បីកំណត់ឲ្យអញ្ញាតរបស់ប្រភេទ structure។ ឧទាហរណ៍ខាងក្រោមនេះបង្ហាញពីរបៀបប្រើ structure មួយក្នុងកម្មវិធី៖

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
struct Books {
   char title[50];
   char author[50];
   char subject[100];
   int  book_id;
};
void main(){
   struct Books Book1;   /* Declare Book1 of type Books */
   struct Books Book2;   /* Declare Book2 of type Books */
```

```
   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Chi Kuong");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 12345678;

   /* book 2 specification */
   strcpy( Book2.title, "Java Programming");
   strcpy( Book2.author, "Kuong Chi");
   strcpy( Book2.subject, "Java 2 Tutorial");
   Book2.book_id = 24681012;

   /* print Book1 info */
   printf( "Book 1 title : %s\n", Book1.title);
   printf( "Book 1 author : %s\n", Book1.author);
   printf( "Book 1 subject : %s\n", Book1.subject);
   printf( "Book 1 book_id : %d\n", Book1.book_id);

   /* print Book2 info */
   printf( "Book 2 title : %s\n", Book2.title);
   printf( "Book 2 author : %s\n", Book2.author);
   printf( "Book 2 subject : %s\n", Book2.subject);
   printf( "Book 2 book_id : %d\n", Book2.book_id);

   getch();
}
```

៤. ការប្រើ structure **ជា** arguments **របស់** function

គេអាចបញ្ជូន structure មួយដូចទៅនឹង argument របស់ function មួយដូចគ្នាទៅនឹងអ្នកបញ្ជូនអញ្ជាតផ្សេងទៀត ឬ pointer ណាមួយ។

ឧទាហរណ៍ទី ១ ៖

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

struct Books {
   char title[50];
   char author[50];
   char subject[100];
```

```
   int book_id;
};
/* function declaration */
void printBook( struct Books book );

void main(){
   struct Books Book1;  /* Declare Book1 of type Book */
   struct Books Book2;  /* Declare Book2 of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Chi Kuong");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 12345678;

   /* book 2 specification */
   strcpy( Book2.title, "Java Programming");
   strcpy( Book2.author, "Kuong Chi");
   strcpy( Book2.subject, "Java 2 Tutorial");
   Book2.book_id = 24681012;

   /* print Book1 info */
   printBook( Book1 );

   /* Print Book2 info */
   printBook( Book2 );
   getch();
}
void printBook( struct Books book ) {
   printf( "Book title : %s\n", book.title);
   printf( "Book author : %s\n", book.author);
   printf( "Book subject : %s\n", book.subject);
   printf( "Book book_id : %d\n", book.book_id);
}
```

ឧទាហរណ៍ទី ២ ៖

```
#include <stdio.h>
#include <conio.h>
typedef struct complex {
```

```
   float real;
   float imag;
} complex;
complex add(complex n1, complex n2);
void main(){
   complex n1, n2, temp;
   printf("For 1st complex number \n");
   printf("Enter real and imaginary part respectively:\n");
   scanf("%f %f", &n1.real, &n1.imag);
   printf("\nFor 2nd complex number \n");
   printf("Enter real and imaginary part respectively:\n");
   scanf("%f %f", &n2.real, &n2.imag);
   temp = add(n1, n2);
   printf("Sum = %.1f + %.1fi", temp.real, temp.imag);
   getch();
}
complex add(complex n1, complex n2){
   complex temp;
   temp.real = n1.real + n2.real;
   temp.imag = n1.imag + n2.imag;
   return(temp);
}
```

៥. អនុគមន៍ឲ្យតម្លៃជា structure

structure គឺជាប្រភេទទិន្នន័យកំណត់ដោយអ្នកប្រើប្រាស់ដូចគ្នានឹង structure ប្រភេទទិន្នន័យដែលមានស្រាប់អាចឲ្យតម្លៃពីអនុគមន៍។

ឧទាហរណ៍ ៖

```
#include <stdio.h>
#include <conio.h>
struct Employee {
   int Id;
```

```
    char Name[25];
    int Age;
    long Salary;
};
Employee Input();    //Statement 1
void main(){
    struct Employee Emp;
    Emp = Input();
    printf("\n\nEmployee Id : %d", Emp.Id);
    printf("\nEmployee Name : %s", Emp.Name);
    printf("\nEmployee Age : %d", Emp.Age);
    printf("\nEmployee Salary : %ld", Emp.Salary);
    getch();
}
Employee Input() {
    struct Employee E;
    printf("\nEnter Employee Id : ");
    scanf("%d", &E.Id);
    printf("\nEnter Employee Name : ");
    scanf("%s", &E.Name);
    printf("\nEnter Employee Age : ");
    scanf("%d", &E.Age);
    printf("\nEnter Employee Salary : ");
    scanf("%ld", &E.Salary);
    return E;    // Statement 2
}
```

៦. ការប្រើ pointer **ទៅលើ** structure

គេអាចកំណត់ pointer ទៅលើ structure ដូចគ្នាទៅនឹងការកំណត់ pointer ទៅលើអញ្ញាតដទៃទៀតដែរ។

ឧទាហរណ៍ ៖

struct Books *struct_pointer;

ឥឡូវនេះ គេអាចផ្តុកអស័យដ្ឋាននៃអញ្ញាត structure នៅក្នុងអញ្ញាត pointer ដែលបានកំណត់ខាងលើ។ ដើម្បីរកអស័យដ្ឋានរបស់អញ្ញាត structure មួយ គេដាក់សញ្ញា & ពីមុខឈ្មោះរបស់ structure ដូចខាងក្រោម៖

struct_pointer = &Book1;

ដើម្បីចូលប្រើ member របស់ structure មួយដោយប្រើ pointer ចង្អុលទៅកាន់ structure នោះ គេត្រូវតែប្រើសញ្ញាណនព្វន្ត → ដូចខាងក្រោមនេះ៖

struct_pointer→title;

ចូរសង្កេតកម្មវិធីខាងក្រោមនេះដែលយើងបានប្រើ pointer ទៅលើ structure ដោយធ្វើការសរសេរប្លែកម្មវិធីខាងលើ។

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
struct Books {
   char title[50];
   char author[50];
   char subject[100];
   int book_id;
};
/* function declaration */
void printBook( struct Books *book );
void main(){
   struct Books Book1;  /* Declare Book1 of type Book */
   struct Books Book2;  /* Declare Book2 of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Chi Kuong");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 12345678;

   /* book 2 specification */
   strcpy( Book2.title, "Java Programming");
```

```
   strcpy( Book2.author, "Kuong Chi");
   strcpy( Book2.subject, "Java 2 Tutorial");
   Book2.book_id = 24681012;

   /* print Book1 info */
   printBook( &Book1 );

   /* Print Book2 info */
   printBook( &Book2 );
   getch();
}
void printBook( struct Books *book ) {
   printf( "Book title : %s\n", book->title);
   printf( "Book author : %s\n", book->author);
   printf( "Book subject : %s\n", book->subject);
   printf( "Book book_id : %d\n", book->book_id);
}
```

៦.១ ការចង្អុលទៅកាន់អស័យដ្ឋានមួយផ្សេងទៀតដើម្បីចូលប្រើ memory

ចូរពិចារណានូវឧទាហរណ៍ខាងក្រោមបង្ហាញពីការចូលប្រើ member របស់ structure តាមរយៈ pointer។

```
#include <stdio.h>
#include <conio.h>
typedef struct {
   int age;
   float weight;
} person;
void main(){
   person *personPtr, person1;
   personPtr = &person1;
        // Referencing pointer to memory address of person1
   printf("Enter Age : ");
   scanf("%d", &(*personPtr).age);
   printf("Enter Weight : ");
   scanf("%f", &(*personPtr).weight);
```

```
    printf("Displaying : ");
    printf("%d\n%f\n",(*personPtr).age,(*personPtr).weight);
    // or
    printf("%d\n%f\n", personPtr->age, personPtr->weight);
    getch();
}
```

៦.២ ការចូលប្រើ members របស់ structure តាមរយៈ pointer ដោយប្រើ dynamic memory allocation

ដើម្បីចូលប្រើ member របស់ structure ដោយប្រើ pointers, memory អាចបង្កើតទីតាំង ដោយប្រើអនុគមន៍ malloc() កំណត់ដោយ header file ឈ្មោះ stdlib.h។

ទម្រង់នៃការប្រើ malloc() ៖

ptr = (cast-type*) malloc(byte-size)

ឧទាហរណ៍ខាងក្រោមនេះបង្ហាញពីការប្រើ member របស់ structure តាមរយៈ pointer ដោយប្រើអនុគមន៍ malloc() ៖

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
struct person {
    int age;
    float weight;
    char name[30];
};
void main() {
    struct person *ptr;
    int i, num;
    printf("Enter number of persons: ");
    scanf("%d", &num);
    ptr = (struct person*) malloc(num * sizeof(struct
person));
```

```
    // Above statement allocates the memory for n structures
    // with pointer personPtr pointing to base address
    for(i = 0; i < num; ++i){
       printf("Enter name, age and weight of the person
respectively:\n");
       scanf("%s%d%f", &(ptr+i)->name, &(ptr+i)->age,
                       &(ptr+i)->weight);
    }
    printf("Displaying Infromation:\n");
    for(i = 0; i < num; ++i)
       printf("%s\t%d\t%.2f\n", (ptr+i)->name, (ptr+i)->age,
                                (ptr+i)->weight);
    getch();
}
```

៧. <u>ការប្រើ</u> <u>structure **មួយនៅក្នុង structure **មួយទៀត**</u>** (Nested structure)

គេអាចប្រើ structure មួយនៅក្នុង structure មួយទៀត បានន័យថា structure មួយអាចប្រកាសនៅខាងក្នុង structure មួយផ្សេងទៀត ដូចដែលយើងប្រកាស members របស់ structure នៅខាងក្នុង structure មួយ។

អញ្ញាត structure អាចជាអញ្ញាត structure ធម្មតា ឬជាអញ្ញាត pointer មួយសម្រាប់ចូលប្រើទិន្នន័យ។ យើងអាចសិក្សានូវទស្សនៈខាងក្រោមនេះ។

<u>៧.១ ការប្រើ</u> <u>structure **មួយនៅក្នុង structure **មួយទៀតដោយប្រើអញ្ញាតធម្មតា**</u>**

កម្មវិធីខាងក្រោមនេះពន្យល់ពីរបៀបប្រើ structure មួយនៅក្នុង structure មួយទៀតដោយប្រើអញ្ញាតធម្មតា។ structure ឈ្មោះ student_college_detail ត្រូវបានប្រកាសនៅខាងក្នុង structure ឈ្មោះ student_detail នៅក្នុងកម្មវិធីខាងក្រោម។ អញ្ញាត structure ទាំងពីរគឺជាអញ្ញាត structure ធម្មតា។

ចូរកត់សម្គាល់ members របស់ structure ឈ្មោះ student_college_detail គឺត្រូវបានចូលប្រើដោយ សញ្ញាណនព្វន្ត (.) ជាមួយនឹង members របស់ structure ឈ្មោះ student_detail។

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

```
struct student_college_detail {
    int college_id;
    char college_name[50];
};
struct student_detail {
    int id;
    char name[20];
    float percentage;
    // structure within structure
    struct student_college_detail clg_data;
} stu_data;
void main(){
    struct student_detail stu_data = {1, "Sopheak", 90.5,
71145, "Royal University of Phnom Penh"};
    printf("Id is: %d \n", stu_data.id);
    printf("Name is: %s \n", stu_data.name);
    printf("Percentage is: %f \n\n", stu_data.percentage);
    printf("College Id is: %d \n",
                    stu_data.clg_data.college_id);
    printf("College Name is: %s \n",
                    stu_data.clg_data.college_name);
    getch();
}
```

លទ្ធផលទទួលបាន គឺ៖

```
Id is: 1
Name is: Sopheak
Percentage is: 90.500000
College Id is: 71145
College Name is: Royal University of Phnom Penh
```

៧.២ ការប្រើ structure មួយនៅក្នុង structure មួយទៀតដោយប្រើអញ្ញាត pointer

កម្មវិធីខាងក្រោមនេះពន្យល់ពីរបៀបប្រើ structure មួយនៅក្នុង structure មួយទៀតដោយប្រើអញ្ញាតធម្មតា។ structure ឈ្មោះ student_college_detail ត្រូវបានប្រកាសនៅខាងក្នុង structure ឈ្មោះ student_detail នៅក្នុងកម្មវិធីខាងក្រោម។ អញ្ញាត structure ធម្មតាមួយនិង អញ្ញាត structure pointer មួយត្រូវបានប្រើក្នុងកម្មវិធីខាងក្រោម។

ចូរកត់សម្គាល់ members របស់ structure ឈ្មោះ student_college_detail គឺត្រូវបានចូល ប្រើដោយ សញ្ញាណនព្វន្ត (.) និង --> ត្រូវបានប្រើ ដើម្បីចូលប្រើ members របស់ structure ដែលប្រកាសនៅខាងក្នុង structure ។

```
#include <stdio.h>
#include <conio.h>
struct student_college_detail {
    int college_id;
    char college_name[50];
};
struct student_detail {
    int id;
    char name[20];
    float percentage;
    // structure within structure
    struct student_college_detail clg_data;
} stu_data, *stu_data_ptr;
void main(){
  struct student_detail stu_data = {1, "Bona", 90.5, 71145,
                        "Royal University of Phnom Penh"};
    stu_data_ptr = &stu_data;
    printf(" Id is: %d \n", stu_data_ptr->id);
    printf(" Name is: %s \n", stu_data_ptr->name);
    printf(" Percentage is: %f \n\n",
```

```
                                stu_data_ptr->percentage);
        printf(" College Id is: %d \n",
                                stu_data_ptr->clg_data.college_id);
        printf(" College Name is: %s \n",
                                stu_data_ptr->clg_data.college_name);
        getch();
    }
```

លទ្ធផលទទួលបាន គឺ៖

Id is: 1

Name is: Bona

Percentage is: 90.500000

College Id is: 71145

College Name is: Royal University of Phnom Penh

៨. ការផ្តុកក្នុងទីតាំង memory របស់ structure (struct memory allocation)

តើអ្នកដឹងទេថា memory ត្រូវផ្តុក members របស់ structure ក្នុងភាសា C យ៉ាងដូចម្តេច? យើងនឹងសិក្សានូវសញ្ញាណខ្លះៗរបស់ភាសា C លើចំណុចមួយចំនួនដូចខាងក្រោម៖

- របៀប members របស់ structure ត្រូវផ្តុកក្នុង memory។
- អ្វីទៅហៅថា structure padding?
- របៀបចៀសវាង structure padding។

៨.១ របៀប members របស់ structure ត្រូវផ្តុកក្នុង memory

ការបង្កើតទីតាំង memory ជាប់ៗគ្នាតែងតែត្រូវបានប្រើសម្រាប់ផ្តុក members របស់ structure ក្នុង memory។ ចូរសង្កេត ឧទាហរណ៍ខាងក្រោមដើម្បីយល់បានរបៀប memory បង្កើតទីតាំងឲ្យ structures។

```
#include <stdio.h>
#include <conio.h>
struct student {
   int id1;
```

```
    int id2;
    char a;
    char b;
    float percentage;
};
void main(){
    int i;
    struct student record1 = {1, 2, 'A', 'B', 90.5};
    printf("size of structure in bytes : %d\n",
            sizeof(record1));
    printf("\nAddress of id1        = %u", &record1.id1);
    printf("\nAddress of id2        = %u", &record1.id2);
    printf("\nAddress of a          = %u", &record1.a);
    printf("\nAddress of b          = %u", &record1.b);
    printf("\nAddress of percentage = %u",
             &record1.percentage);
    printf("\nAddress of i          = %u", &i);
    getch();
}
```

លទ្ធផលទទួលបាន គឺ៖

size of structure in bytes : 14

Address of id1 = 1638208

Address of id2 = 1638212

Address of a = 1638216

Address of b = 1638217

Address of percentage = 1638218

Address of i = 1638224

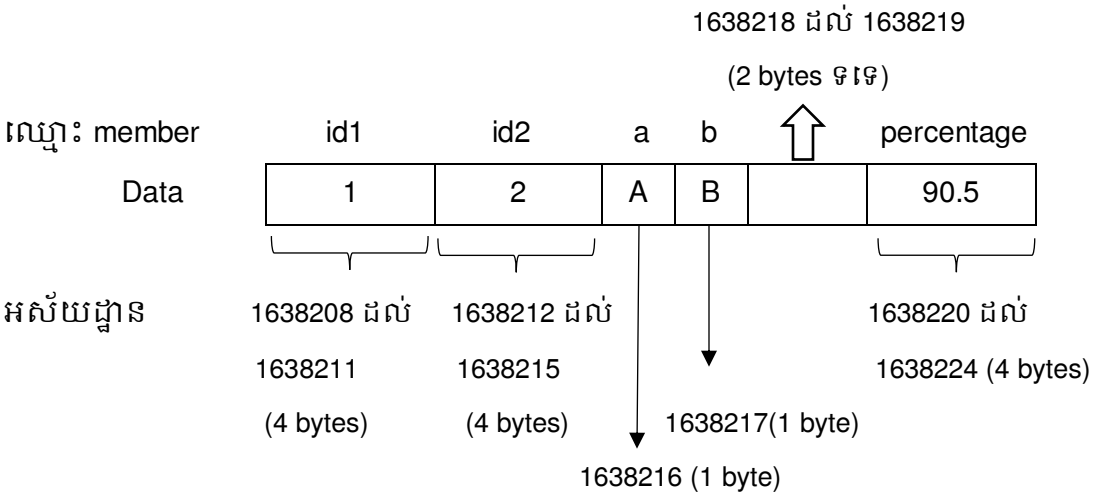
វាមាន members ចំនួន 5 សម្រាប់ structure នៅក្នុងកម្មវិធីខាងលើ។ ចំពោះ compiler ទំហំ 32 bit នោះ memory ទំហំ 4 bytes ត្រូវបានប្រើសម្រាប់ផ្ទុកទិន្នន័យប្រភេទ int , រីឯ memory ទំហំ 1 byte

ត្រូវបានប្រើសម្រាប់ផ្ទុកទិន្នន័យប្រភេទ char និង memory ទំហំ 4 bytes ត្រូវបានប្រើសម្រាប់ផ្ទុកទិន្នន័យប្រភេទ float ។

សូមបញ្ជាក់តារាងខាងក្រោមដើម្បីដឹងពីទីតាំង ដែល memory ត្រូវបានបង្កើតសម្រាប់ប្រភេទទិន្នន័យនិមួយៗនៅទីតាំងជាប់ៗគ្នាក្នុង memory។

ប្រភេទទិន្នន័យ	ទីតាំង memory ក្នុងភាសា C (Compiler ទំហំ 32 bit)		
	ពីអស័យដ្ឋាន	ទៅកាន់អស័យដ្ឋាន	ចំនួន bytes សរុប
int id1	1638208	1638211	4
int id2	1638212	1638215	4
char a	1638216		1
char b	1638217		1
អស័យដ្ឋាន 1638218 និង 1638219 ទុកឲ្យនៅទំនេរ (តើអ្នកដឹងទេ ហេតុអ្វី? យើងសូមបញ្ជាក់នូវផ្នែកខាងក្រោមនេះ អំពី structure padding)			2
float percentage	1638220	1638224	4

ការគាងរូបភាពនៃការបង្កើតទីតាំង memory ឲ្យ structure ខាងលើ គឺបានផ្តល់ឲ្យដូចដ្យាក្រាមខាងក្រោម។ ដ្យាក្រាមនេះនឹងជួយឲ្យអ្នកយល់បាននូវសញ្ញាណទីតាំង memory យ៉ាងងាយ។



៨.២ អំពី structure padding

ដើម្បីរៀបជាជួរនូវទិន្នន័យក្នុង memory, 1 byte ឬច្រើន bytes ទំនេរ (អស័យដ្ឋាន) ត្រូវបានសៀតបញ្ចូល (ឬទុកឲ្យទំនេរ) ចន្លោះអស័យដ្ឋាន memory ដែលបង្កើតទីតាំងឲ្យ members របស់ structure ដទៃទៀត។ គំនិតនេះហៅថា structure padding។

Architecture នៃ processor កុំព្យូទ័រ ដែលជាមធ្យោបាយមួយបែបដែលវាអាចអានមួយពាក្យ (4 byte ក្នុង processor ទំហំ 32 bit) ពី memory នៅក្នុងពេលមួយ។

ដើម្បីប្រើផលប្រយោជន៍ processor នេះ ទិន្នន័យជានិច្ចជាកាលត្រូវដាក់ជាជួរដូចទៅនឹងកញ្ចប់ 4 bytes ដែលឈានទៅសៀតបញ្ចូលនូវ អស័យដ្ឋានទំនេរចន្លោះអស័យដ្ឋានរបស់ member ផ្សេងទៀត។

ដោយសារតែគំនិត structure padding នេះ ទំហំរបស់ structure តែងតែមិនដូចគ្នាដូចអ្វីដែលយើងបានគិតនោះទេ។

ឧទាហរណ៍ ចូរពិចារណានូវ structure ខាងក្រោមដែលមាន members ចំនួន 5។

```
struct student {
        int id1;
        int id2;
        char a;
        char b;
        float percentage;
};
```

តាមគំនិតភាសា C, ប្រភេទទិន្នន័យ int និង float ប្រើ 4 bytes ក្នុងប្រភេទនិមួយៗ ហើយប្រភេទទិន្នន័យ ប្រើ 1 byte សម្រាប់ processor ទំហំ 32 bit។ ហេតុនេះ វាមាន 14 bytes (4+4+1+1+4) នឹងបង្កើតទីតាំងសម្រាប់ structure ខាងលើ។ ក៏ប៉ុន្តែ គំនិតនេះគឺខុស។ អ្នកដឹងទេ ហេតុអ្វី? Architecture នៃ processor កុំព្យូទ័រ ដែលជាមធ្យោបាយមួយបែបដែលវាអាចអានមួយពាក្យ ពី memory នៅក្នុងពេលមួយ។ មួយពាក្យស្មើនឹង 4 bytes ចំពោះ processor ទំហំ 32 bit និង 8 bytes ចំពោះ processor

ទំហំ 64 bit។ ហេតុនេះ processor ទំហំ 32 bit តែងតែអាន 4 bytes ក្នុងពេលតែមួយ និង អាន 8 bytes ក្នុងពេលតែមួយចំពោះ processor ទំហំ 64 bit។ គំនិតនេះមានសារៈសំខាន់ណាស់សម្រាប់បង្កើនល្បឿន processor។ ដើម្បីប្រើផលប្រយោជន៍នេះ memory ត្រូវរៀបចំជាសំណុំ 4 bytes នៅក្នុង processor ទំហំ 32 bit និង 8 bytes នៅក្នុង processor ទំហំ 64 bit។

កម្មវិធីខាងក្រោមនេះ ត្រូវបានធ្វើការ compiled និងប្រតិបត្តិ លើ compiler 32 bit។ ចូរពិនិត្យមើលការបង្កើតទីតាំង memory ឲ្យ structure1 និង structure2 ក្នុងកម្មវិធីខាងក្រោម៖

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
/*  Below structure1 and structure2 are same.
    They differ only in member's allignment */
struct structure1 {
     int id1;
     int id2;
     char name;
     char c;
     float percentage;
};
struct structure2 {
     int id1;
     char name;
     int id2;
     char c;
     float percentage;
};
void main(){
    struct structure1 a;
    struct structure2 b;
    printf("size of structure1 in bytes : %d\n",
sizeof(a));
```

```
        printf("\n Address of id1        = %u", &a.id1 );
        printf("\n Address of id2        = %u", &a.id2 );
        printf("\n Address of name       = %u", &a.name );
        printf("\n Address of c          = %u", &a.c );
        printf("\n Address of percentage = %u", &a.percentage);
        printf("\n\nsize of structure2 in bytes : %d\n",
sizeof(b));
        printf("\n Address of id1        = %u", &b.id1 );
        printf("\n Address of name       = %u", &b.name );
        printf("\n Address of id2        = %u", &b.id2 );
        printf("\n Address of c          = %u", &b.c );
        printf("\n Address of percentage = %u", &b.percentage);
        getch();
    }
```

លទ្ធផលទទួលបាន គឺ៖

size of structure1 in bytes : 14

Address of id1 = 1638212

Address of id2 = 1638216

Address of name = 1638220

Address of c = 1638221

Address of percentage = 1638222

size of structure2 in bytes : 14

Address of id1 = 1638196

Address of name = 1638200

Address of id2 = 1638201

Address of c = 1638205

Address of percentage = 1638206

៨.៣ របៀបចៀសវាង structure padding ក្នុងភាសា C

#pragma pack(1) អាចប្រើបានសម្រាប់រៀប memory ចំពោះ members របស់ structure នៅជាប់ខាងចុងនៃ members របស់ structure ដទៃទៀត។ Visual C++ ផ្តល់នូវលក្ខណៈនេះ ក៏ប៉ុន្តែ compilers ខ្លះ ដូចជា Turbo C/C++ មិនផ្តល់នូវលក្ខណៈនេះឡើយ។ ចូរពិនិត្យកម្មវិធីខាងក្រោមនេះ ៖

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#pragma pack(1)
/* Below structure1 and structure2 are same.
   They differ only in member's allignment */
struct structure1 {
    int id1;
    int id2;
    char name;
    char c;
    float percentage;
};
struct structure2 {
    int id1;
    char name;
    int id2;
    char c;
    float percentage;
};
void main() {
    struct structure1 a;
    struct structure2 b;
    printf("size of structure1 in bytes : %d\n",
sizeof(a));
```

```
    printf ( "\n   Address of id1        = %u", &a.id1 );
    printf ( "\n   Address of id2        = %u", &a.id2 );
    printf ( "\n   Address of name       = %u", &a.name );
    printf ( "\n   Address of c          = %u", &a.c );
    printf ( "\n   Address of percentage = %u",
&a.percentage );
    printf("   \n\nsize of structure2 in bytes : %d\n",
sizeof(b));
    printf ( "\n   Address of id1        = %u", &b.id1 );
    printf ( "\n   Address of name       = %u", &b.name );
    printf ( "\n   Address of id2        = %u", &b.id2 );
    printf ( "\n   Address of c          = %u", &b.c );
    printf ( "\n   Address of percentage = %u",
&b.percentage );
    getch();
}
```

លទ្ធផលទទួលបាន គឺ៖

size of structure1 in bytes : 14

Address of id1 = 1638212

Address of id2 = 1638216

Address of name = 1638220

Address of c = 1638221

Address of percentage = 1638222

size of structure2 in bytes : 14

Address of id1 = 1638196

Address of name = 1638200

Address of id2 = 1638201

Address of c = 1638205

Address of percentage = 1638206

៩. ការប្រើ typedef

typedef គឺជាពាក្យគន្លឹះត្រូវបានប្រើសម្រាប់ផ្តល់នូវឈ្មោះ និមិត្តសញ្ញាថ្មីមួយសម្រាប់ឈ្មោះដែលមានរួចហើយ ក្នុងកម្មវិធី C។ នេះដូចគ្នាទៅនឹងការកំណត់ឈ្មោះថ្មីមួយទៀតសម្រាប់ពាក្យបញ្ជា។

ចូរពិនិត្យ structure ខាងក្រោម ៖

```
struct student {
        int mark [2];
        char name [10];
        float average;
}
```

អញ្ជាតចំពោះ structure ខាងលើ អាចប្រកាសបានតាមពីរវិធី។

វិធីទី ១ ៖

```
struct student record;       /* for normal variable */
struct student *record;      /* for pointer variable */
```

វិធីទី ២ ៖

```
typedef struct student status;
```

នៅពេលដែលយើងប្រើពាក្យគន្លឹះ typedef ពីមុខ struct <tag_name> ដូចខាងលើ បន្ទាប់មកគេអាចប្រើប្រភេទទិន្នន័យដែលកំណត់ឈ្មោះ status ក្នុងកម្មវិធី C សម្រាប់ប្រកាសអញ្ជាត structure។ ឥឡូវនេះ ការប្រកាសអញ្ជាត structure គឺជា "status record"។ នេះស្មើនឹង "struct student record"។ ការកំណត់ប្រភេទទិន្នន័យសម្រាប់ "struct student" គឺជា "status" មានន័យថា "struct student"។

វិធីមួយទៀតសម្រាប់ប្រកាស structure ដោយប្រើ typedef ក្នុងភាសា C :

```
typedef struct student {
    int mark [2];
    char name [10];
```

```
    float average;
} status;
```

ដើម្បីប្រកាសអញ្ញាត structure យើងអាចប្រើឃ្លាដូចខាងក្រោម។

```
status record1;   /* record 1 is structure variable */
status record2;   /* record 2 is structure variable */
```

កម្មវិធីខាងក្រោមជាឧទាហរណ៍នៃការប្រើ typedef ៖

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
typedef struct student {
   int id;
   char name[20];
   float percentage;
} status;
void main() {
   status record;
   record.id = 1;
   strcpy(record.name, "Chi Kuong");
   record.percentage = 89.5;
   printf(" Id is: %d \n", record.id);
   printf(" Name is: %s \n", record.name);
   printf(" Percentage is: %f \n", record.percentage);
   getch();
}
```

typedef អាចប្រើសម្រាប់សម្រួលនូវពាក្យបញ្ជាពិតតាមតម្រូវការរបស់យើង។
ឧទាហរណ៍ ចូរពិចារណាឃ្លាខាងក្រោម៖

```
typedef long int LLI;
```

ឃ្លាខាងលើ LLI គឺជាការកំណត់ប្រភេទទិន្នន័យ សម្រាប់ពាក្យបញ្ជាពិត "long int" ។ យើងអាចប្រើ ការកំណត់ប្រភេទទិន្នន័យ LLI ជំនួសឲ្យការប្រើ "long int" ក្នុងកម្មវិធី C។

```
#include <stdio.h>
#include <conio.h>
void main() {
   typedef long int LLI;
   printf("Storage size for long long int data " \
          "type  : %ld \n", sizeof(LLI));
   getch();
}
```

១០. <u>អំពី</u> <u>array</u> <u>នៃ</u> <u>structure</u>

structure មួយ ក្នុងភាសា C ត្រូវបានប្រើសម្រាប់ផ្ទុកសំណុំប៉ារ៉ាម៉ែត្រ អំពី object /entity មួយ។ ជួនកាលយើងចង់ផ្ទុកអញ្ញាត structure ជាច្រើន សម្រាប់ objects ជាច្រើន ពេលនោះ array នៃ structure ត្រូវបានប្រើ។

ទាំងអញ្ញាត structure ទាំងប្រភេទទិន្នន័យដែលមានស្រាប់ ទទួលបានប្រព្រឹត្តកម្មដូចគ្នាក្នុងភាសា C។ ភាសា C អាចឲ្យយើងបង្កើត array នៃអញ្ញាត structure ដូចយើងបង្កើត array នៃតម្លៃជាចំនួនគត់ ឬ ចំនួនទសភាគ។ ទម្រង់នៃការប្រកាស array នៃ structure, ការចូលប្រើធាតុ array ដោយឡែក និងការដាក់ index array គឺដូចគ្នាទៅនឹង array ប្រភេទទិន្នន័យដែលមានស្រាប់។

ឧទាហរណ៍៖ ការប្រកាស structure array ។

```
struct Employee {
   char name[50];
   int age;
   float salary;
} employees[1000];
```

ឬ

```
struct Employee employees[1000];
```

ការប្រកាសខាងលើ យើងប្រកាស array នៃ employees ចំនួន 1000 ធាតុដែល structure employee និមួយៗ មាន members ឈ្មោះ name, age និង salary។ array employees[0] ផ្ទុកព័ត៌មាន នៃ employee ទី1 , employees[1] ផ្ទុកព័ត៌មាននៃ employee ទី 2, - ល -។

យើងអាចចូលប្រើ members របស់អញ្ញាត structure ដូចជា៖

array_name[index].member_name

<u>ឧទាហរណ៍</u> ៖

employees[5].age

កម្មវិធីខាងក្រោម យើងប្រកាសនូវ structure "employee" ដើម្បីផ្ទុកព័ត៌មានលម្អិតនៃ employee ម្នាក់ ដូចជា name, age, និង salary។ ពេលនោះ យើងប្រកាស array នៃ structure employee មានឈ្មោះថា "employees" ដែលមានទំហំ 10 សម្រាប់ផ្ទុកព័ត៌មានលម្អិតរបស់ employees ជាច្រើន។

```
#include <stdio.h>
#include <conio.h>
struct employee {
   char name[100];
   int age;
   float salary;
};
void main(){
   struct employee employees[10];
   int counter, index, count, totalSalary;
   printf("Enter Number of Employees\n");
   scanf("%d", &count);
   /* Storing employee detaisl in structure array */
   for(counter=0; counter<count; counter++){
       printf("Enter Name, Age and Salary of Employee\n");
       scanf("%s %d %f", &employees[counter].name,
       &employees[counter].age, &employees[counter].salary);
   }
```

```
    /* Calculating average salary of an employee */
    for(totalSalary=0, index=0; index<count; index++){
        totalSalary += employees[index].salary;
    }
    printf("Average Salary of an Employee is %f\n",
        (float)totalSalary/count);
    getch();
}
```

១១. **<u>អំពី</u>** Union

Union គឺជាអញ្ញាតប្រភេទទិន្នន័យពិសេស ដែលអាចឲ្យគេផ្ទុកប្រភេទទិន្នន័យផ្សេងគ្នានៅក្នុងទីតាំង memory ដូចគ្នា។ គេអាចកំណត់ union មួយដោយមាន members ច្រើន ក៏ប៉ុន្តែមានតែ member តែមួយគត់ទេដែលអាចផ្ទុកតម្លៃនៅក្នុងខណៈពេលណាមួយ។ Union ផ្តល់នូវមធ្យោបាយមានប្រសិទ្ធភាពមួយនៃការប្រើទីតាំង memory ដូចគ្នាសម្រាប់គោលបំណងច្រើនយ៉ាង។

១២. **<u>និយមន័យ</u>** Union

ដើម្បីកំណត់ union មួយ គេត្រូវប្រើឃ្លា union តាមវិធីដូចគ្នាទៅនឹងអ្វីដែលយើងបានធ្វើ នៅពេលកំណត់ structure មួយ។ ឃ្លា union កំណត់នូវប្រភេទទិន្នន័យថ្មីមួយដោយមាន member ច្រើនជាងមួយសម្រាប់កម្មវិធី។ ទម្រង់ទូទៅនៃឃ្លា union មានដូចខាងក្រោម៖

```
union [union tag] {
   member definition;
   member definition;
   ...
   member definition;
} [one or more union variables];
```

union tag គឺជាជម្រើសមួយអាចប្រើក៏បាន មិនប្រើក៏បាន ហើយ member definition និមួយៗគឺជាអញ្ញាតធម្មតា ដូចជា int i, ឬ float f; ឬ អញ្ញាតណាមួយផ្សេងទៀត។ នៅខាងចុងនៃ union ពីមុខសញ្ញា ; យើងអាចកំណត់អញ្ញាត union មួយឬច្រើន ប៉ុន្តែវាក៏មានលក្ខណៈជាជម្រើសដែរ បានន័យថា

ប្រើក៏បាន មិនប្រើក៏បាន។ នេះគឺជាវិធីមួយដែលគេនឹងកំណត់ប្រភេទ union ដោយដាក់ឈ្មោះ Data ដែលមាន members ចំនួន 3 គឺ i, f និង str ៖

```
union Data {
  int i;
  float f;
  char str[20];
} data;
```

ពេលនេះ អញ្ញាតមួយរបស់ប្រភេទ Data អាចផ្ទុកចំនួនគត់ integer, ចំនួនទសភាគ float, ឬ តួអក្សរ string។ វាមានអញ្ញាតទោលមួយ បានន័យថា ទីតាំង memory ដូចគ្នា ត្រូវបានប្រើសម្រាប់ផ្ទុកប្រភេទទិន្នន័យជាច្រើន។ គេអាចប្រើប្រភេទទិន្នន័យមានស្រាប់ឬ ប្រភេទទិន្នន័យកំណត់ដោយអ្នកប្រើប្រាស់ នៅក្នុង union មួយផ្អែកលើតម្រូវការរបស់គេ។

memory ជាប់ប្រើដោយ union គឺធំគ្រប់គ្រាន់ ដើម្បីផ្ទុក member របស់ union ធំបំផុត។ ជាក់ស្តែង នៅក្នុងឧទាហរណ៍ខាងលើ ប្រភេទ Data នឹងប្រើចំនួន 20 bytes នៃទំហំ memory ពីព្រោះវាជាទំហំធំបំផុតដែលអាចផ្ទុកតួអក្សរ string។ ឧទាហរណ៍ខាងក្រោមនេះបង្ហាញនូវទំហំ memory សរុបដែលជាប់ប្រើដោយ union ខាងលើ។

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
union Data {
   int i;
   float f;
   char str[20];
};
void main(){
   union Data data;
   printf("Memory size occupied by data : %d\n",sizeof(data));
   getch();
```

```
}
```

លទ្ធផលទទួលបាន គឺ៖

```
Memory size occupied by data : 20
```

១៣. ការចូលប្រើ member របស់ union

ដើម្បីចូលប្រើ member របស់ union យើងប្រើ សញ្ញាណនព្វន្តចូលប្រើ member (.) ។ សញ្ញាណនព្វន្តនេះត្រូវសរសេរជា code ដោយប្រើ . ចន្លោះឈ្មោះអញ្ញាត union និង member ដែលគេចង់ចូលប្រើ។ យើងនឹងប្រើពាក្យ union ដើម្បីកំណត់អញ្ញាតនៃប្រភេទ union។ កម្មវិធីខាងក្រោមនេះជាឧទាហរណ៍បង្ហាញពីការប្រើ union ។

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
union Data {
   int i;
   float f;
   char str[20];
};
void main( ) {
   union Data data;
   data.i = 10;
   data.f = 220.5;
   strcpy( data.str, "C Programming");
   printf( "data.i : %d\n", data.i);
   printf( "data.f : %f\n", data.f);
   printf( "data.str : %s\n", data.str);
   getch();
}
```

លទ្ធផលទទួលបានគឺ ៖

```
data.i : 1917853763
```

data.f : 4122360580327794860000000000000.000000

data.str : C Programming

នៅត្រង់នេះ យើងអាចមើលឃើញថា តម្លៃ member របស់ union គឺ i និង f ទទួលបានលទ្ធផលមិនត្រឹមត្រូវ ពីព្រោះតម្លៃចុងក្រោយកំណត់ទៅឲ្យអញ្ញាតបានជាប់ប្រើក្នុងទីតាំង memory ហើយនេះជាហេតុផលដែលតម្លៃរបស់ member str ត្រូវបោះបញ្ចេញមកក្រៅបានយ៉ាងត្រឹមត្រូវ។

ឥឡូវនេះ ចូរសង្កេតឧទាហរណ៍ដូចគ្នាម្តងទៀត ដែលយើងនឹងប្រើអញ្ញាតម្តងមួយៗក្នុងគោលបំណងសំខាន់ គឺបង្ហាញពីការប្រើ union ៖

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
union Data {
   int i;
   float f;
   char str[20];
};
void main(){
   union Data data;
   data.i = 10;
   printf("data.i : %d\n", data.i);
   data.f = 220.5;
   printf("data.f : %f\n", data.f);
   strcpy(data.str, "C Programming");
   printf("data.str : %s\n", data.str);
   getch();
}
```

លទ្ធផលទទួលបានគឺ ៖

data.i : 10

data.f : 220.500000

data.str : C Programming

នៅពេលនេះ គ្រប់ member ទាំងអស់ត្រូវបង្ហាញមកក្រៅ បានយ៉ាងត្រឹមត្រូវដោយសារ member ត្រូវបានប្រើម្តងមួយៗក្នុងខណៈមួយ។

១៤. ភាពខុសគ្នារវាង Structure និង Union

នៅក្នុងផ្នែកនេះយើងនឹងបង្ហាញភាពខុសគ្នារវាង structure និង union ដោយមានឧទាហរណ៍ និងការវិភាគ។ ចូរសង្កេតលក្ខណៈខុសគ្នារបស់វា ក្នុងតារាងខាងក្រោម ៖

Structure	union
ពាក្យគន្លឹះ struct ត្រូវបានប្រើសម្រាប់ប្រកាស structure	ពាក្យគន្លឹះ union ត្រូវបានប្រើសម្រាប់ប្រកាស union
អញ្ញាត structure នឹងបង្កើតទីតាំង memory សម្រាប់គ្រប់ members របស់ structure ដោយឡែកពីគ្នា	អញ្ញាត union នឹងបង្កើតទីតាំង memory រួមសម្រាប់គ្រប់ members របស់ union

struct Employee{
int age;
char name[50];
float salary;
};
	union Employee{
	int age;
	char name[50];
	float salary;
	};

structure ប្រើទីតាំង memory ច្រើន។ ទំហំ memory = ផលបូកគ្រប់ទំហំ member របស់ structure។ ទំហំ memory = int + char array[50] + float ។ ទំហំ memory = 2 + 50 + 4 bytes = 56 bytes ។	union ប្រើទីតាំង memory តិចជាង បើប្រៀបធៀបទៅនឹង structure។ ទំហំ memory = ទំហំ member របស់ union ធំបំផុត។ តាមឧទាហរណ៍ខាងលើ ទំហំ member របស់ union ធំបំផុត គឺ char array ហេតុនេះ ទំហំ memory = 50 bytes។
វាអាចឲ្យយើងចូលប្រើ member ណាមួយឬ គ្រប់ members ទាំងអស់នៅពេលណាក៏បាន។	វាអាចឲ្យយើងចូលប្រើបាន member តែមួយរបស់ union ក្នុងពេលមួយដង។

កម្មវិធីខាងក្រោមនេះ យើងប្រកាស structure និង union ដោយមាន ប្រភេទទិន្នន័យរបស់ members ដូចគ្នា បន្ទាប់មកយើងនឹងគណនាទំហំរបស់ union និង structure ដោយប្រើអនុគមន៍ sizeof() ។

```
// C Program to find difference between Structure and Union
#include <stdio.h>
#include <conio.h>
struct Employee {
  int age;
  char Name[50];
  char Department[20];
  float Salary;
};
union Person {
  int ag;
  char Nam[50];
  char Departmen[20];
  float Salar;
};
void main() {
  struct Employee emp1;
  union Person person1;
  printf("The Size of Employee Structure = %d\n",
          sizeof (emp1));
  printf("The Size of Person Union = %d\n",
          sizeof (person1));
  getch();
}
```

បើយើងសង្កេតមើលទៅក្នុងកម្មវិធីដែលបាន ប្រកាស structure ឈ្មោះ Employee និង union ឈ្មោះ Person សុទ្ធតែមានប្រភេទទិន្នន័យដូចគ្នានិងទំហំដូចគ្នា។ ក្រៅពីនេះ គេបានបង្កើត អញ្ញាត structure ឈ្មោះ emp1 និង អញ្ញាត union ឈ្មោះ person1 ដូចឃ្លាខាងក្រោម ៖

```
struct Employee emp1;
union Person person1;
```

យើងបានប្រើ sizeof() សម្រាប់គណនា ទំហំនៃ Employee របស់ structure និង Person របស់ union។

sizeof (emp1) = 78 bytes

ពោលគឺ

```
int age ;                  // ត្រូវការទំហំ memory ចំនួន 4 bytes
char Name[50];             // ត្រូវការទំហំ memory ចំនួន 50 bytes
char Department[20];  // ត្រូវការទំហំ memory ចំនួន 20 bytes
float Salary;              // ត្រូវការទំហំ memory ចំនួន 4 bytes
```

ទំហំ memory របស់ structure = 4 + 50 + 20 + 4 = 78 bytes

sizeof (person1) = 50 bytes

member របស់ union ដែលត្រូវការទំហំ memory ធំបំផុតនៅក្នុងកម្មវិធីខាងលើ គឺ Name[50]។ ដូច្នេះ ទំហំ memory របស់ union = 50 bytes។

១៥. <u>តំរូកម្មវិធីមួយចំនួនដាក់ឧទាហរណ៍ការប្រើ</u> structure

កម្មវិធីខាងក្រោមនេះបង្ហាញពីការប្រើ array នៃ structure ដើម្បីរៀបលេខ account តាមលំដាប់កើនឡើង។

```
#include <stdio.h>
#include <conio.h>
/* Define structure */
struct info {
```

```
    long int accntNum;
    char name[25];
    float balance;
};
/* Begin main */
void main(){
    /* Declare variables */
    struct info client[25], temp;
    int i, j, min, numClient;
    /* Display opening message */
    printf("Welcome to Client Account Information
Services\n\n");
    // Prompt for how many clients will have info entered
    printf ("Enter number of clients to be used: ");
    scanf ("%d", &numClient);
    /* Prompt for user to enter each clients info */
    for (i = 0; i < numClient; i++) {
        printf ("\nEnter account number: ", i + 1);
        scanf ("%d", &client[i].accntNum);
        fflush(stdin); /* Remove extraneous characters */
        printf ("Enter last name: ", i + 1);
        gets (client[i].name);
        fflush(stdin); /* Remove extraneous characters */
        printf ("Enter balance: ", i + 1);
        scanf ("%f", &client[i].balance);
    } /* end for */
    /* Sort array in ascending order by account number */
    for (i = 0; i < numClient; i++){
       min = i;
       for (j = i + 1; j < numClient; j++){
          if(client[j].accntNum<client[min].accntNum)
             min = j;
```

```
            }
            temp = client[i];
            client[i] = client[min];
            client[min] = temp;
        }
        /* Display each clients information */
        printf ("\nACCOUNT          LAST NAME          BALANCE");
        for (i = 0; i < numClient; i++) {
            printf ("\n%7d %17s %15.2f", client[i].accntNum,
                                client[i].name, client[i].balance);
        } /* end for */
        printf ("\n");
        getch();
    } /* end main */
```

កម្មវិធីខាងក្រោមនេះបង្ហាញពីការប្រើ array នៃ structure, pointer នៃ structure, និង argument របស់ function ជា structure ដើម្បីរៀបពិន្ទុ(ពីខ្ពស់មកទាប) តាមលំដាប់ថយចុះ។

```
#include <conio.h>
#include <stdio.h>
struct Grade {
  char Name[10];
  int Grade[5];
  int Total;
};
// Definition of Swap function
void swap(struct Grade *px, struct Grade *py){
  struct Grade temp;
  temp = *px;
  *px = *py;
  *py = temp;
```

```
}
void main(){
  int i=0, j, k, m, a, b, c, d, count=1;
  struct Grade classX[4];
  clrscr();
  while(1){
     printf("Enter Name of next student; ");
     scanf("%s", classX[i].Name);
     printf("Enter 5 grades of %s ", classX[i].Name);
     classX[i].Total = 0;
     for ( m = 0; m < 5; m++){
        scanf("%d", &classX[i].Grade[m]);
        classX[i].Total += classX[i].Grade[m];
     }
     i++;
     count++;
     if( count > 4 ) break;
  }
  for (j = 0; j < count-1; j++){
     printf("\n%-15s", classX[j].Name);
     for (k = 0; k < 4; k++)
        printf("%d\t", classX[j].Grade[k]);
     printf("%d", classX[j].Total);
  }
  printf("\nAfter sorting the list is as below.\n");
  for (a = 0; a<count-1; a++){
     for (b = 0; b<count-2; b++)
        if ( classX[b].Total < classX[b+1].Total )
           swap(&classX[b], &classX[b+1]);
  }
  For (c = 0; c < count-1; c++){
     printf("\n%-15s", classX[c].Name);
```

```
      for(d = 0; d < count-1; d++)
         printf("%d\t", classX[c].Grade[d]);
      printf("%d",classX[c].Total );
   }
   printf ("\n");
   getch();
}
```