

Chapter 8

Structures

A structure is a collection of one or more than one variable, possibly of different data type, grouped together under a single name for convenient handling. Thus, a structure might contain integer elements, floating-point elements and character elements. Structure helps to organize data especially in large programs, because they provide group of variables of different data type to be treated as a single unit. For making, usefulness of structure more clear let us consider an example. In an organization an employee's details (i.e. name, address, designation, salary etc.) have to be maintained.

One possible method would be to create a multi dimensional array to contain all employee details. But this is not possible because the variables are of different data type i.e. name is of type char, salary is of type float and so on. Hence, the simple solution is the structure. We should be clear about array and structures.

Structure declarations are some what more complicated than array declarations, since a structure must be defined in terms of its individual members. In general terms, the composition of a structure may be defined

```
struct tag{  
    member 1;  
    member 2;  
    .....  
    member n;  
};
```

Where, **struct** is a required keyword;

tag is a name that identifies structure of this type;

and member 1, member 2, ..., member n are individual member declarations.

Note: There is no formal distinction between a structure definition and a structure declaration; the terms are used interchangeably.

The individual members can be ordinary variables, pointers, arrays, or other structures. The member names within a particular structure must be distinct from one another, though a member name can be the same as the name of variable that is defined outside of the structure. A storage class, however cannot be assigned to an individual member, and individual members cannot be initialized within a structure type declaration.

Declaration of a Structure variable

A structure variable is a variable of a structure type. Once the structure has been defined, individual **structure-type** variable can be declared as follows:

```
storage_class struct tag variable 1, variable 2, ....., variable n;
```

Where, *storage_class* is an optional storage class specifier, **struct** is a required keyword, **tag** is the name of the structure & variable 1, variable 2, ..., variable n are the structure variables of type tag.

Example#1:

```

struct account{
    int acc_no;
    char acc_type;
    char name[80];
    float balance;
};
struct account old_customer, new_customer;

```

The structure is named **account** (i.e. the tag is account). It contains four members: an integer quantity(acc_no), a single character(acc_type), an 80 element character array(name[80]), and a floating point quantity(balance).

It is possible to combine the declaration of structure with that of structure variables as shown below:

```

storage_class struct tag{
    member 1;
    member 2;
    .....
    member n;
}variable1, variable2, ..., variable n;

```

The **tag** is optional in this situation.

Example#2:

```

struct account{
    int acc_no;
    char acc_type;
    char name[80];
    float balance;
}old_customer, new_customer;

```

OR

```

struct{
    int acc_no;
    char acc_type;
    char name[80];
    float balance;
}old_customer, new_customer;

```

When a structure variable such as new_customer is declared, the compiler automatically allocates sufficient memory to accommodate all of its members.

Declaration of a Structure & Accessing Structure Element

Note the following points while declaring a structure type:

- The closing brace in the structure type declaration must be followed by a semicolon.
- It is important to understand that a structure type declaration does not tell the compiler to reserve any space in memory. All a structure declaration does is, it defines the form of the structure.
- Usually structure type declaration appears at the top of the source code file, before any variables or functions are defined. In very large programs they are usually put in a separate header file, and the file is included (using the preprocessor directive #include) in whichever program we want to use this structure type.

Accessing of Structure Elements

Structure use a dot(.) operator to access individual elements. The structure variable name followed by a period and the member name references that individual member. The general form for accessing a member of a structure is

`structure_name.member_name`

So to refer to pages of the structure defined in our sample program we have to use: **b1.pages**

Similarly, to refer to price we should use: **b1.price**

So, to print the name of book we can write: **printf("The book's name is %s",b1.name);**

Initializing Structures

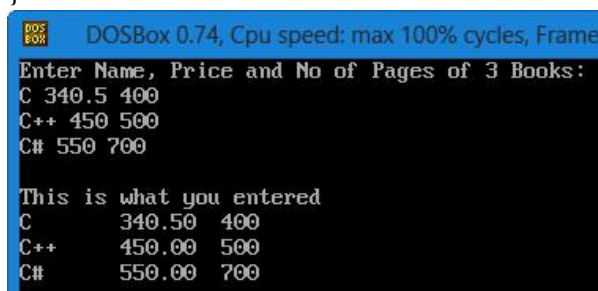
Like primary variables and arrays, structure variables can also be initialized where they are declared. The format used is quite similar to that used to initialize arrays.

Example#3:

```
struct book{
    char name[20]; float price; int pages;
};
struct book b1={"Basic",130.00,550};
struct book b2={"Math",150.50,800};
```

Example#4:

```
#include<stdio.h>
#include<conio.h>
struct book{
    char name[20]; float price; int pages;
};
struct book b1,b2,b3;
void main( ){
    printf("Enter Name, Price and No of Pages of 3 Books:\n");
    scanf("%s%f%d",b1.name,&b1.price,&b1.pages);
    scanf("%s%f%d",b2.name,&b2.price,&b2.pages);
    scanf("%s%f%d",b3.name,&b3.price,&b3.pages);
    printf("\nThis is what you entered");
    printf("\n%s\t%.2f\t%d",b1.name,b1.price,b1.pages);
    printf("\n%s\t%.2f\t%d",b2.name,b2.price,b2.pages);
    printf("\n%s\t%.2f\t%d",b3.name,b3.price,b3.pages); getch();
}
```



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frame
Enter Name, Price and No of Pages of 3 Books:
C 340.5 400
C++ 450 500
C# 550 700

This is what you entered
C 340.50 400
C++ 450.00 500
C# 550.00 700
```

This program demonstrates two fundamental aspects of structures: *Declaration of a structure & Accessing of a structure elements*

Structure Assignment

The information contained in one structure may be assigned to another structure of the same type using a single assignment statement. That is, you do not need to assign the value of each member separately. The following program illustrates structure assignments:

Example#5:

```
#include<stdio.h>
#include<conio.h>
struct{
    int a;
    int b;
}x,y;
void main(){
    x.a=10;
    x.b=20;
    y=x; //Assign one structure to another
    printf("%d",y.a);
    printf("%d",y.b);
    getch();
}
```

Nested Structures

Nested structures are structures within structure. Let us construct two different structures and consolidates it into single nested structures.

Example#6:

```
struct employee{
    char name[20];
    int emp_code;
    char designation[20];
    float salary;
} emp1, emp2;

struct emp_address{
    int no;
    char street[15];
    char area[20];
} address1;
```

These two structures can be consolidated as

```
struct employee{
    char name[20];
    struct emp_address{
        int no;
        char street[15];
        char area[20];
    } address1;
    int emp_code;
    char designation[20];
    float salary;
} emp1, emp2;
```

Example#7: We can also tag names to define inner structures

```

struct date{
    int month;
    int day;
    int year;
};
struct account{
    int acc_no;
    char acc_type;
    char name[80];
    float balance;
    struct date last_payment;
}customers;

```

If a structure member is itself a structure, then a member of the embedded structure can be accessed by writing variable.member.sub_member

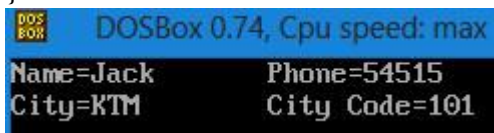
e.g. **customers.last_payment.month**

Example#8: A sample to demonstrate nesting of structure

```

#include<stdio.h>
#include<conio.h>
struct address{
    char phone[15];
    char city[25];
    int city_code;
};
struct emp{
    char name[25];
    struct address a;
};
void main(){
    int i;
    struct emp e={"Jack","54515","KTM",101};
    printf("\nName=%s\tPhone=%s",e.name,e.a.phone);
    printf("\nCity=%s\tCity Code=%d",e.a.city,e.a.city_code);
    getch();
}

```



Example#9:

```

#include <stdio.h>
#include <conio.h>
struct add{
    int door_no;
    char street[20];
    char place[30];
    int pin;
};
struct stdent{
    char name[30];
    int roll_no;
    struct add address;
};
void main(){
    struct stdent std;
    clrscr();
    printf("Enter the name of the stdent:");
    scanf("%s",std.name);
    printf("Enter the roll no. of the stdent:");
    scanf("%d",&std.roll_no);
    printf("Enter the door no.:");
    scanf("%d",&std.address.door_no);
    printf("Enter the street name:");
    scanf("%s",std.address.street);
    printf("Enter the area name:");
    scanf("%s",std.address.place);
    printf("\nThe stdent's details are...\n");
    printf("\nThe name is %s",std.name);
    printf("\nThe roll no is %d",std.roll_no);
    printf("\nThe door no is %d",std.address.door_no);
    printf("\nThe area name is %s",std.address.place);
    printf("\nThe street name is %s",std.address.street);
    getch();
}

```

```

DOSBox 0.74, Cpu speed: max 100%
Enter the name of the stdent:Roshna
Enter the roll no. of the stdent:35
Enter the door no.:301
Enter the street name:Garudkundal
Enter the area name:Libali

The stdent's details are....

The name is Roshna
The roll no is 35
The door no is 301
The area name is Libali
The street name is Garudkundal

```

It is also permissible to nest more than one type of structures.

```

struct personal_record{
    struct name_part name;
    struct address_part address;
    struct date_aprt dateofbirth;
    .....
};
struct personal_record personal;

```

The first member of this structure is name which is of the type name_part. Similarly other members have their own structure types.

Array of Structures

Since we can create an array of any valid type, it is possible to define an array of structure also; i.e. an array in which each element is a structure. To create the array of structures, you must first define a structure and then declare an array variable of that type. For example, to declare a 100-element array of structures of type books, you can write

```

struct book b1[100];

```

This creates 100 sets of variables that are organized as defined in the structure book. To access a specific structure, index the structure name. For example, to print the price of structure 3, write

```

printf("%f",b1[2].price);

```

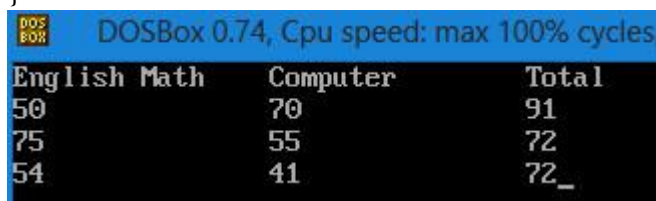
Like all array variables, arrays of structures begin indexing at 0.

Example#10: To demonstrate arrays of structures

```

#include<stdio.h>
#include<conio.h>
struct marks{
    int english;
    int math;
    int computer;
    int total;
};
void main(){
    int i;
    static struct marks student[3]={ { 50,70,91,0},{ 75,55,72,0},{ 54,41,72,0} };
    printf("English\tMath\tComputer\tTotal");
    for(i=0;i<3;i++){
        student[i].total=student[i].english+student[i].math+student[i].computer;
        printf("\n%d\t%d\t%d\t%d",student[i].english,student[i].math,student[i].computer, student[i].total);
    }
    getch();
}

```



English	Math	Computer	Total
50	70	91	0
75	55	72	0
54	41	72	0

Example#11: Storing book information using structure

```

#include<stdio.h>
#include<conio.h>
struct book{
    char name[20];
    int price;
    int pages;
}b[10];
void main(){
    int i;
    clrscr();
    for(i=0;i<10;i++){
        printf("\nEnter Name, Price, and Pages: ");
        scanf("%s%d%d",&b[i].name,&b[i].price,&b[i].pages);
    }
    for(i=0;i<10;i++){
        printf("\n%s %d %d",b[i].name,b[i].price,b[i].pages);
    }
    getch();
}

```

This provides space in memory for 100 structures of the type structure book. In an array of structures all elements of the array are stored in adjacent memory locations. Since each element of this array is a structure, and since all structure elements are always stored in adjacent locations you can visualize the arrangement of array of structures in memory.

e.g. b[0]'s name, price, and pages in memory would be immediately followed by b[1]'s name, price, and pages, and so on.

Note that the array is declared just as it would have been, with any other array. Since student is an array, we use the usual array-accessing methods to access individual elements and then the member operator to access members.

Example#12:

```

struct employee{
    char name[20];
    char roll_no[5];
    int salary;
} emp [5];

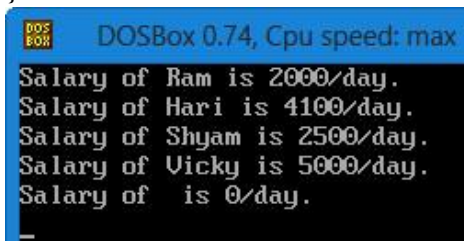
```

The above declaration defines five instances of the variable emp of the type struct employee.

The initialization can be done as below:

```
#include<stdio.h>
#include<conio.h>
struct employee{
    char name[20];
    char roll_no[5];
    int salary;
} emp [5] = {
    {"Ram","e01", 2000},
    {"Hari","e02", 4100},
    {"Shyam","e03", 2500},
    {"Vicky","e04", 5000}
};

void main(){
    int i;
    clrscr();
    for(i=0;i<5;i++){
        printf("Salary of %s is %d/day.\n",emp[i].name,emp[i].salary);
    }
    getch();
}
```



Actually size need not to be specified, because the initialization by itself will determine the size. In above initialization part, every particular employee's details have been enclosed in braces. This is not necessary, but it improves the clarity of code. In case a member is not be initialized, the value for it can be omitted. Assume the name and salary of the employee is not known; the declaration will be as follows.

```
{ , "e06", }
```

In case any member of structure can be accessed using the index number along with the structure name. For e.g., if he/she wants to print the name of the first employee,

```
printf ("The name of the first employee is %s", emp [0].name);
```

Typedef Structure

C language provides the opportunity to define new datatype equivalent to the existing system using the typedef statement. Let us take an example. The declaration would be

```
typedef struct add{
    int dd;
    int mm;
    int yyyy;
}dob;
dob emp_date_of_birth;
```

The above declaration variable emp_date_of_birth has now become the type of structure add.

Structures and Functions

C supports the passing of structure values as arguments of functions. There are three methods by which the values of a structure can be transferred from one function to another. The first method is to pass each member of the structure as an actual argument of the function call. The actual arguments are then treated independently like ordinary variables. This is the most elementary method and becomes unmanageable and inefficient when the structure size is large.

The second method involves passing of a copy of the entire structure to the called function. Since the function is working on a copy of the structure, any changes to structure members within the function are not reflected in the original structure(in the calling function). It is, therefore, necessary for the function to return the entire structure back to the calling function. All compilers may not support this method of passing the entire structure as a parameter.

The third method employs a concept called pointer to pass the structure as an argument. In this case, the address location of the structure is passed to the called function. The function can access indirectly the entire structure and work on it. This is similar to the way arrays are passed to the functions. This method is more efficient as compared to the second one.

1. Passing Structure Member to a Function

When you pass a member of a structure to a function, you are actually passing the value of that member to the function.

Example#13:

```
#include<stdio.h>
#include<conio.h>
struct{
    char g;
    int y;
    float z;
    char s[10];
}mike;
void func(char);
void main(){
    mike.g='M';
    clrscr();
    func(mike.g);
    getch();
}
```

```
void func(char a){
    printf("%c\n", a);
}
```

Here are examples of each member being passed to function. e.g.

```
func(mike.g); //Passes character value of x
function1(mike.y); //Passes integer value of y
function2(mike.z); //Passes float value of z
function3(mike.s) //Passes address of string s
function4(mike.s[2]); //Passes character value of s[2]
```

If you wish to pass the address of an individual structure member, put the & operator before the structure name. e.g.

```
func(&mike.g); //Passes address of character x
function1(&mike.y); //Passes address of integer y
function2(&mike.z); //Passes address of float z
function3(mike.s) //Passes address of string s
function4(&mike.s[2]); //Passes address of character s[2]
```

Remember that & operator precedes the structure name, but not the individual member name.

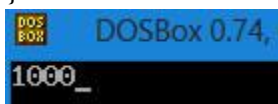
2. Passing Entire Structures to Functions

When a structure is used as an argument to a function, the entire structure is passed using the standard call-by-value method. This means that any changes made to the contents of the structure inside the function to which it is passed do not affect the structure used as an argument.

When using a structure as a parameter, remember that the type of the argument must match the type of the parameter. For example, in the following program both the argument arg and the parameter parm are declared as the same type of structure.

Example#14:

```
#include<stdio.h>
#include<conio.h>
struct test{
    int a,b;
    char ch;
};
void function(struct test);
void main(){
    struct test arg;
    arg.a=1000;
    function(arg);
    getch();
}
void function(struct test parm) {
    printf("%d",parm.a);
}
```



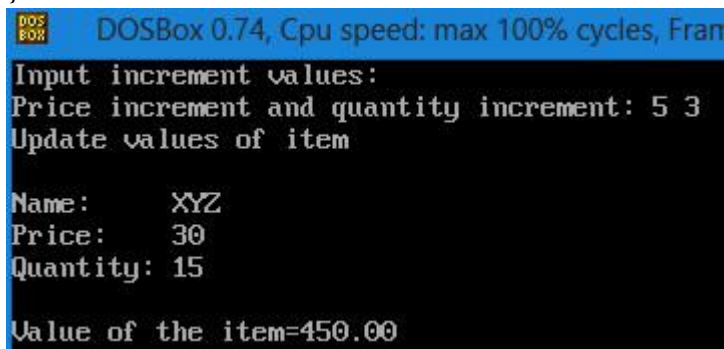
Example#15:

```

#include<stdio.h>
#include<conio.h>
typedef struct{
    char name[20];
    int price;
    int quantity;
}stores;

stores update(stores product, int p, int q){
    product.price+=p;
    product.quantity+=q;
    return(product);
}
float mul(stores stock){
    return(stock.price*stock.quantity);
}
void main(){
    float value;
    int p_inc,q_inc;
    static stores item={"XYZ",25,12};
    clrscr();
    printf("Input increment values:\n");
    printf("Price increment and quantity increment: ");
    scanf("%d%d",&p_inc,&q_inc);
    item=update(item,p_inc,q_inc);
    printf("Update values of item\n\n");
    printf("Name:   %s\n",item.name);
    printf("Price:   %d\n",item.price);
    printf("Quantity: %d\n",item.quantity);
    value=mul(item);
    printf("\nValue of the item=%.2f\n",value);
    getch();
}

```



```

DOSBox 0.74, Cpu speed: max 100% cycles, Fram
Input increment values:
Price increment and quantity increment: 5 3
Update values of item

Name:   XYZ
Price:   30
Quantity: 15

Value of the item=450.00

```

Structure and Pointer

C language allows declaring pointer to structure just like pointer to other ordinary variables. The declaration can be done in the following manner.

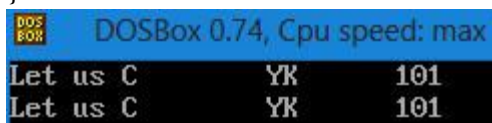
```
struct employee{
    char *name;
    char *roll_no;
    int salary;
}*emp1;
```

Above declaration make pointer variable *emp1 of type struct employee. Now, here *emp1 is the pointer to the structure. So while accessing structure members we can use either (*emp1).name or emp1->name. Here, -> is termed as the member access operator for pointers.

Same arithmetic operation can be performed with pointer for structure as for any other datatype. Pointer to structure can perform all the operation that an ordinary pointer can do variables of different datatype.

Example#16:

```
#include<stdio.h>
#include<conio.h>
void main(){
    struct book{
        char name[25];
        char author[25];
        int call_no;
    };
    struct book b1={"Let us C","YK",101};
    struct book *ptr;
    clrscr();
    ptr=&b1;
    printf("%s\t%s\t%d\n",b1.name,b1.author,b1.call_no);
    printf("%s\t%s\t%d\n",ptr->name, ptr->author, ptr->call_no);
    getch();
}
```



```
DOSBox 0.74, Cpu speed: max
Let us C      YK      101
Let us C      YK      101
```

Example#17:

```

#include<stdio.h>
#include<conio.h>
void main(){
    struct std_record{
        char name[25]; char reg_no[10]; int average; char grade;
    }std[50],*ptr; /* ptr is a pointer of type structure std_record */
    int i,n; clrscr();
    printf("Number of stds grades to be computed?"); scanf("%d",&n);
    for(i=0;i<n;i++){
        printf("\nStudent[%d]'s information:\n",i+1);
        printf("Name? "); scanf("%s",std[i].name);
        printf("Register number? "); scanf("%s",std[i].reg_no);
        printf("Average Score? "); scanf("%d",&std[i].average);
        printf("%s\t%s%8.2d\n",std[i].name,std[i].reg_no,std[i].average);
    }
    ptr=std; /* pointer 'ptr' points to std[0] */
    /* Assigning grades to 'n' stds*/
    for(ptr=std;ptr<std+n;ptr++){
        if(ptr->average<30.0){
            ptr->grade='D';
        }else if(ptr->average<50.0){
            ptr->grade='C';
        }else if(ptr->average<70.0){
            ptr->grade='B';
        }else{
            ptr->grade='A';
        }
    }
    /*Displaying Student Records*/
    printf("\nNAME\tREG_NUMBER\tAVERAGE\tGRADE\n");
    for(ptr=std;ptr<std+n;ptr++){
        printf("%-10s%-10s%10.2d\t%c\n",ptr->name,ptr->reg_no,ptr->average,ptr->grade);
    }
    getch();
}

```

```

DOSBox 0.74, Cpu speed: max 100% cycles
Number of stds grades to be computed?2

Student[1]'s information:
Name? Ramesh
Register number? 074BCT15
Average Score? 77
Ramesh  074BCT15      77

Student[2]'s information:
Name? Rashila
Register number? 074BCT29
Average Score? 83
Rashila 074BCT29      83

NAME      REG_NUMBER      AVERAGE  GRADE
Ramesh    074BCT15         77        A
Rashila   074BCT29         83        A

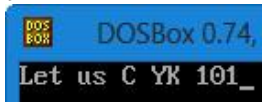
```

Passing Structure to Function using Reference

Structures may be passed to function either by value or by reference. However, they are usually passed by reference, i.e., the address of the structure is passed to the function. Let us consider the employee for instance.

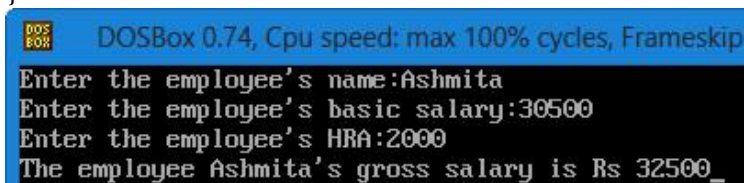
Example#18:

```
#include<stdio.h>
#include<conio.h>
struct book{
    char name[25];
    char author[25];
    int call_no;
};
void display(struct book *);
void main(){
    struct book b1={"Let us C","YK",101}; clrscr();
    display(&b1); getch();
}
void display(struct book *ptr){
    printf("%s %s %d",ptr->name,ptr->author,ptr->call_no);
}
```



Example#19:

```
#include<stdio.h>
#include<conio.h>
float gross_calc(struct employee *);
struct employee{
    char name[20]; float basic; float HRA;
}emp;
void main(){
    float gross; clrscr();
    printf("Enter the employee's name:"); scanf("%s",emp.name); fflush(stdin);
    printf("Enter the employee's basic salary:"); scanf("%f",&emp.basic); fflush(stdin);
    printf("Enter the employee's HRA:"); scanf("%f",&emp.HRA); fflush(stdin);
    gross=gross_calc(&emp);
    printf("The employee %s's gross salary is Rs %0.0f",emp.name,gross);
    getch();
}
float gross_calc(struct employee *emp){ /*emp is a pointer of type struct employee*/
    /*this function adds employee's basic salary & HRA, and returns the gross salary*/
    return(emp->basic+emp->HRA);
}
```



How is the structure different from an array?

- An array can hold multiple elements of same data type whereas a structure can hold multiple elements of different data types
- The component element of an array is referenced by the array name and the index value e.g. name[3], a[5], etc. whereas component element of a structure is referenced by the structure name and the element name e.g. book.id, stu.name etc.