

CHAPTER 14

Structure, Union



LEARNING OBJECTIVE

Today's modern applications need complex data structures to support them. A structure is a collection of related data items of different data types. It is useful for applications that need a lot more features than those provided by the primitive data types. In this chapter, we will learn about structures.

A structure extends the concept of arrays by storing related information of heterogeneous data types together under a single umbrella. So, in this chapter we will see how a structure is defined, declared, and accessed using the C language.

14.1 INTRODUCTION

A structure is similar to records. It stores related information about an entity. Structure is basically a user-defined data type that can store related information (even of different data types) together. The major difference between a structure and an array is that, an array contains related information of the same data type.

A structure is, therefore, a collection of variables under a single name. The variables within a structure are of different data types and each has a name that is used to select it from the structure.

14.1.1 Structure Declaration

A structure is declared using the keyword `struct` followed by a structure name. All the variables of the structure are declared within the structure. A structure type is generally declared by using the following syntax:

```
struct struct-name
{
   data_type var-name;
   data_type var-name;
   ...
};
```

Programming Tip: Do not forget to place a semicolon after the definition of structures and unions.

For example, if we have to define a structure for a student, then its related information probably would be: `roll_number`, `name`, `course`, and `fees`. This structure can be declared as,

```
struct student
{
   int r_no;
   char name[20];
   char course[20];
   float fees;
};
```

Now, the structure has become a user-defined data type. Each `var-name` declared within a structure is called a member of the structure. The structure declaration, however, does not allocate any memory or consume

storage space. It just gives a template that conveys to the C compiler how the structure is laid out in memory and gives details of the member names. Similar to any other data type, memory is allocated for the structure when we declare a variable of the structure. For example, we can define a variable `student` by writing, `struct student stud1;`

Here, `struct student` is a data type and `stud1` is a variable. Look at another way of declaring variables. In the following syntax, the variable is declared at the time of structure declaration.

```
struct student
{
   int r_no;
   char name[20];
   char course[20];
   float fees;
}stud1, stud2;
```

In this declaration, we declare two variables `stud1` and `stud2` of the structure `student`. So, if you want to declare more than one variable of the structure, then separate the variables using a comma.

When we declare variables of the structure, separate memory is allocated for each variable. This is shown in Figure 14.1.

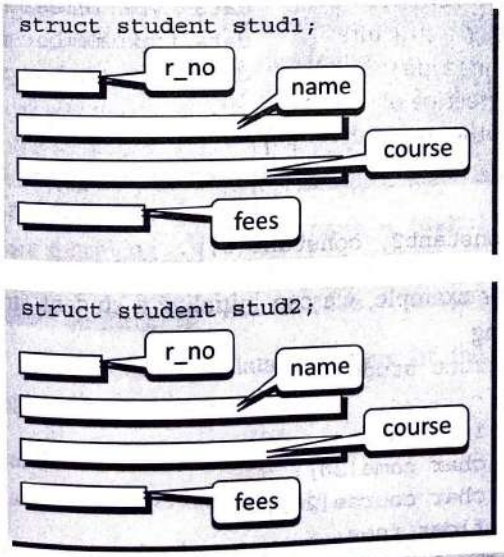


Figure 14.1 Memory allocation for a structure variable

Let us see some more structure declarations.

Example 14.1 Declare a structure to store information about a point in the coordinate system.

Solution

```
struct point
{
   int x,y;
};
```

Example 14.2 Declare a structure to store customer information.

Solution

```
struct customer
{
   int cust_id;
   char name[20];
   char address[20];
   long int phone_num;
   int DOB;
};
```

Example 14.3 Declare a structure to store information of a particular date.

Solution

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

Example 14.4 Declare a structure to store information of a particular book.

Solution

```
struct BOOK
{
   char title[20];
   char author[20];
   int pages;
   float price;
   int yr_of_publication;
};
```

Programming Tip: Use different member names for different structures for clarity.

Example 14.5 Declare a5 structure to create an inventory record.

Solution

```
struct inventory
{
   char prod_name[20];
   float price;
   int stock;
};
```

Note: Structure type and variable declaration of a structure can be either local or global depending on their placement in the code.

Last but not the least, structure member names and names of the structure follow the same rules as laid down for the names of ordinary variables. However, care should be taken to ensure that the name of the structure and the name of a structure member should not be the same. Moreover, as structure name and its variable name should also be different.

14.1.2 Typedef Declarations

The `typedef` (derived from type definition) keyword enables the programmer to create a new data type name for an existing data type. By using `typedef`, no new data is created, rather an alternate name is given to a known data type.

Programming Tip: C does not allow declaration of variables at the time of creating a typedef definition. So variables must be declared in an independent statement.

The general syntax of using the `typedef` keyword is given as:

```
typedef existing data type
new data_type
```

Note that the `typedef` statement does not occupy any memory, it simply defines a new type. For example, if we write

```
typedef int INTEGER;
```

then `INTEGER` is the new name of data type `int`. To declare variables using the new data type name, precede the variable name with the data type name (new). Therefore, to define an integer variable, we may now write

```
INTEGER num = 5;
```

When we precede a `struct` name with the `typedef` keyword, then the `struct` becomes a new type. It is used to make the construct shorter with more meaningful names for types already defined by C or for types that you have declared. A `typedef` declaration is a synonym for the type. For example, writing

```
typedef struct student
{
   int r_no;
   char name[20];
   char course[20];
   float fees;
};
```

Now that you have preceded the structure's name with the keyword `typedef`, the student becomes a new data type. Therefore, now you can straightaway declare variables of this new data type as you declare variables of type `int`, `float`, `char`, `double`, etc., to declare a variable of structure student you will just write,

```
student stud1;
```

Note that we have not written `struct student stud1`.

14.1.3 Initialization of Structures

A structure can be initialized in the same way as other data types are initialized. *Initializing a structure* means assigning some constants to the members of the structure. When the user does not explicitly initialize the structure, then C automatically does that. For `int` and `float` members, the values are initialized to zero and `char` and string members are initialized to the `'\0'` by default (in the absence of any initialization done by the user).

The initializers are enclosed in braces and are separated by commas. However, care must be taken to see that the initializers match their corresponding types in the structure definition.

The general syntax to initialize a structure variable is given as follows:

```
struct struct_name
{
   data_type member_name1;
   data_type member_name2;
   data_type member_name3;
   ...
}struct_var = {constant1, constant2,
constant 3,...};
```

or

Programming Tip: It is an error to assign a structure of one type to a structure of another type.

```
struct struct_name
{
  data_type member_name1;
  data_type member_name2;
  data_type member_name3;
  ...
};
struct struct_name
struct_var = {constant1,
constant2, constant 3,...};
```

For example, we can initialize a student structure by writing

```
struct student
{
  int r_no;
  char name[20];
  char course[20];
  float fees;
}stud1 = {01, "Rahul", "BCA", 45000};
```

or by writing

```
struct student stud1 = {01, "Rahul", "BCA",
45000};
```

Figure 14.2 illustrates how the values will be assigned to the individual fields of the structure.

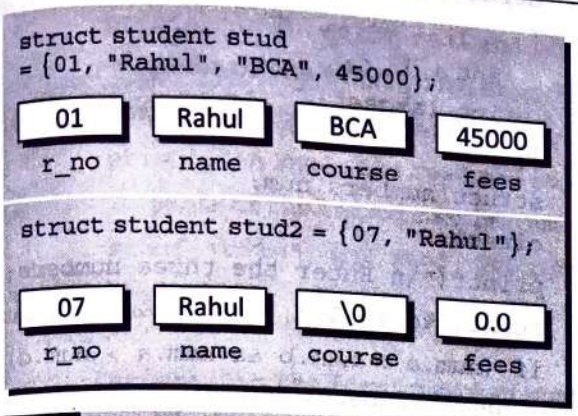


Figure 14.2 Assigning values to structure elements

When all the members of a structure are not initialized, it is called partial initialization. In case of partial initialization, first few members of the structure are initialized and those that are uninitialized are assigned default values.

14.1.4 Accessing the Members of a Structure

Each member of a structure can be used just like a normal variable, but its name will be a bit longer. A structure member variable is generally accessed using a '.' (dot) operator. The syntax of accessing a structure or a member of a structure can be given as:

```
struct_var.member_name
```

Programming Tip:
A member of the structure cannot be accessed directly using its name. Rather you must use the structure name followed by the dot operator before specifying the member name.

The dot operator is used to select a particular member of the structure. For example, to assign value to the individual data members of the structure variable `stud1`, we may write,

```
stud1.r_no = 01;
stud1.name = "Rahul";
stud1.course = "BCA";
stud1.fees = 45000;
```

To input values for data members of the structure variable `stud1`, we may write,

```
scanf("%d", &stud1.r_no);
scanf("%s", &stud1.name);
```

Similarly, to print the values of structure variable `stud1`, we may write,

```
printf("%s", stud1.course);
printf("%f", stud1.fees);
```

Memory is allocated only when we declare variables of the structure. In other words, memory is allocated only when we instantiate the structure. In the absence of any variable, structure definition is just a template that will be used to reserve memory when a variable of type `struct` is declared.

Once the variables of a structure are defined, we can perform a few operations on them. For example, we can use the assignment operator '=' to assign the values of one variable to another.

Note Of all the operators, –>, ., (), and [] have the highest priority. This is evident from the following statement: `stud1.fees++` will be interpreted as `(stud1.fees)++`

14.1.5 Copying and Comparing Structures

We can assign a structure to another structure of the same type. For example, if we have two structure variables `stud1` and `stud2` of type `struct student` given as,

```
struct student stud1 = {01,
"Rahul", "BCA", 45000};
struct student stud2;
```

Programming Tip:
An error will be generated if you try to compare two structure variables.

Then to assign one structure variable to another we will write,

```
stud2 = stud1;
```

This statement initializes the members of `stud2` with the values of members of `stud1`. Therefore, now the values of `stud1` and `stud2` can be given as shown in Figure 14.3.

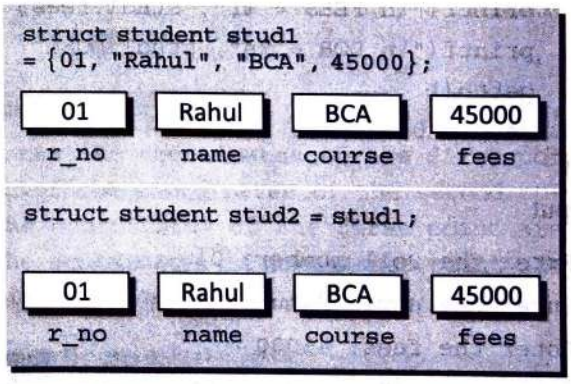


Figure 14.3 Values of structure variables

C does not permit comparison of one structure variable with another. However, individual members of one structure can be compared with individual members of another structure. When we compare one structure member with another structure's member, the comparison will behave like any other ordinary variable comparison. For example, to compare the fees of two students, we will write,

```
if(stud1.fees > stud2.fees)
   Fees of stud1 is greater than stud2
```

1. Write a program using structures to read and display the information about a student.

```
#include <stdio.h>
#include <conio.h>
int main()
{
   struct student
   {
      int roll_no;
      char name[80];
      float fees;
      char DOB[80];
   };
   struct student stud1;
   clrscr();
   printf("\n Enter the roll number: ");
   scanf("%d", &stud1.roll_no);
   printf("\n Enter the name: ");
   scanf("%s", stud1.name);
   printf("\n Enter the fees: ");
   scanf("%f", &stud1.fees);
   printf("\n Enter the DOB: ");
   scanf("%s", stud1.DOB);
   printf("\n ********STUDENT'S DETAILS
   ********");
   printf("\n ROLL No. = %d", stud1.roll_no);
   printf("\n NAME = %s", stud1.name);
   printf("\n FEES = %f", stud1.fees);
   printf("\n DOB = %s", stud1.DOB);
   getch();
   return 0;
}
```

Output

```
Enter the roll number: 01
Enter the name: Rahul
Enter the fees: 45000
Enter the DOB: 25-09-1991
********STUDENT'S DETAILS *******
ROLL No. = 01
NAME = Rahul
FEES = 45000.00
DOB = 25-09-1991
```

2. Write a program, using structures, to find the biggest of three numbers.

```
#include <stdio.h>
#include <conio.h>
int main()
{
struct numbers
{
   int a, b, c;
   int largest;
};
struct numbers num;
clrscr();
printf("\n Enter the three numbers: ");
scanf("%d %d %d", &num.a, &num.b, &num.c);
if (num.a > num.b && num.a > num.c)
   num.largest = num.a;
if (num.b > num.a && num.b > num.c)
   num.largest = num.b;
if (num.c > num.a && num.c > num.b)
   num.largest = num.c;
printf("\n The largest number is: %d",
num.largest);
getch();
return 0;
}
```

Output

```
Enter the three numbers: 7 9 1
The largest number is: 9
```

3. Write a program to read, display, add, and subtract two complex numbers.

```
#include <stdio.h>
#include <conio.h>
int main()
{
   typedef struct complex
   {
      int real;
      int imag;
   }COMPLEX;
   COMPLEX c1, c2, sum_c, sub_c;
   int option;
   clrscr();
   do
   {
      printf("\n *** MAIN MENU ***");
      printf("\n 1. Read the complex nos.");
      printf("\n 2. Display the complex nos.");
      printf("\n 3. Add the complex nos.");
      printf("\n 4. Subtract the complex nos.");
      printf("\n 5. EXIT");
      printf("\n Enter your option: ");
      scanf("%d", &option);
      switch(option)
      {
```

```
        case 1:
            printf("\n Enter the real and imaginary
               parts of the first complex number: ");
            scanf("%d %d", &c1.real, &c1.imag);
            printf("\n Enter the real and imaginary
               parts of the second complex number: ");
            scanf("%d %d", &c2.real, &c2.imag);
            break;
        case 2:
            printf("\n The first complex number is:
               %d + %di", c1.real, c1.imag);
            printf("\n The second complex number is:
               %d + %di", c2.real, c2.imag);
            break;
        case 3:
            sum_c.real = c1.real + c2.real;
            sum_c.imag = c1.imag + c2.imag;
            printf("\n The sum of two complex
            numbers
               is: %d + %di", sum_c.real, sum_c.imag);
            break;
        case 4:
            sub_c.real = c1.real - c2.real;
            sub_c.imag = c1.imag - c2.imag;
            printf("\n The difference between two
               complex numbers is: %d + %di",
               sub_c.real, sub_c.imag);
            break;
        }
    }while(option != 5);
    getch();
    return 0;
}
```

Output

```
******** MAIN MENU *********
1. Read the complex nos.
2. Display the complex nos.
3. Add the complex nos.
4. Subtract the complex nos.
5. EXIT
Enter your option: 1
Enter the real and imaginary parts of the
first complex number:  2 3
Enter the real and imaginary parts of the
second complex number:  4 5
******** MAIN MENU *********
1. Read the complex nos.
2. Display the complex nos.
3. Add the complex nos.
4. Subtract the complex nos.
5. EXIT
Enter your option: 3
The sum of two complex numbers is: 6 + 8i
```

4. Write a program to enter two points and then calculate the distance between them.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int main()
{
    typedef struct point
    {
        int x, y;
    }POINT;
    POINT p1, p2;
    float distance;
    clrscr();
    printf("\n Enter the coordinates of the
    first point: ");
    scanf("%d %d", &p1.x, &p1.y);
    printf("\n Enter the coordinates of the
    second point: ");
    scanf("%d %d", &p2.x, &p2.y);
    distance = sqrt(pow((p1.x - p2.x), 2) +
    pow((p1.y - p2.y), 2));
    printf("\n The coordinates of the first
    point are: %dx %dy", p1.x, p1.y);
    printf("\n The coordinates of the second
    point are: %dx %dy", p2.x, p2.y);
    printf("\n Distance between p1 and p2 =
    %f", distance);
    getch();
    return 0;
}
```

Output

```
Enter the coordinates of the first point: 2 3
Enter the coordinates of the second point: 9 9
The coordinates of the first point are: 2x 3y
The coordinates of the second point are: 9x 9y
Distance between p1 and p2 = 9.219544
```

14.2 NESTED STRUCTURES

A structure can be placed within another structure, i.e., a structure may contain another structure as its member. A structure that contains another structure as its member is called a *nested structure*.

Let us now see how we declare nested structures or structures that contain structures. Although it is possible to declare a nested structure with one declaration, it is not recommended. The easier and clearer way is to declare the structures separately and then group them in a high-level structure. When you do this, take care to check that nesting must be done from inside out (from lowest level to the most inclusive level), i.e., to say, declare the innermost

structure, then the next level structure, working towards the outer (most inclusive) structure.

```
typedef struct
{
   char first_name[20];
   char mid_name[20];
   char last_name[20];
}NAME;

typedef struct
{
   int dd;
   int mm;
   int yy;
}DATE;

typedef struct student
{
   int r_no;
   NAME name;
   char course[20];
   DATE DOB;
   float fees;
};
```

In this example, we see that the structure student in turn contains two other structures—NAME and DATE. Both these structures have their own fields. The structure NAME has three fields: first_name, mid_name, and last_name. The structure DATE also has three fields: dd, mm, and yy, which specify the day, month, and year of the date. To assign values to the structure fields, we will write,

```
student stud1;
stud1.name.first_name = "Janak";
stud1.name.mid_name = "Raj";
stud1.name.last_name = "Thareja";
stud1.course = "BCA";
stud1.DOB.dd = 15;
stud1.DOB.mm = 09;
stud1.DOB.yy = 1990;
stud1.fees = 45000;
```

In case of nested structures, we use the dot operator in conjunction with the structure variables to access the members of the innermost as well as the outermost structures. The use of nested structures is illustrated in the following program:

5. Write a program to read and display information of a student, using a structure within a structure.

```
#include <stdio.h>
#include <conio.h>
int main()
{
   struct DOB
   {
      int day;
      int month;
      int year;
   };
   struct student
   {
      int roll_no;
      char name[100];
      float fees;
      struct DOB date;
   };
   struct student stud1;
   clrscr();
   printf("\n Enter the roll number: ");
   scanf("%d", &stud1.roll_no);
   printf("\n Enter the name: ");
   scanf("%s", stud1.name);
   printf("\n Enter the fees: ");
   scanf("%f", &stud1.fees);
   printf("\n Enter the DOB: ");
   scanf("%d %d %d", &stud1.date.day,
   &stud1.date.month, &stud1.date.year);
   printf("\n *** STUDENT'S DETAILS ***");
   printf("\n ROLL No. = %d", stud1.roll_no);
   printf("\n NAME = %s", stud1.name);
   printf("\n FEES = %f", stud1.fees);
   printf("\n DOB = %d - %d - %d", stud1.
   date.day, stud1.date.month, stud1.date.
   year);
   getch();
   return 0;
}
```

Output

```
Enter the roll number: 01
Enter the name: Rahul
Enter the fees: 45000
Enter the DOB: 25 09 1991
********STUDENT'S DETAILS *******
ROLL No. = 01
NAME = Rahul
FEES = 45000.00
DOB = 25-09-1991
```

14.3 ARRAYS OF STRUCTURES

In this examples, we have seen how to declare a structure and assign values to its data members. Now we will discuss how to declare an array of a structure. For this purpose, let us first analyse, where we would need array of structures.

Programming Tip: It is an error to omit array subscripts when referring to individual structures of an array of structures.

In a class, we do not have just one student, but there may be at least 30 students. So, the same definition of the structure can be used for all the 30 students. This would be possible when we make an array of the structure. An array of a structure is declared in the same way as we had declared an array of a built-in data type.

Another example, where an array of structures is desirable, is in case of an organization. An organization has a number of employees. Hence, defining a separate structure for every employee is not a viable solution. Thus here we can have a common structure definition for all the employees. This can again be done by declaring an array of the structure employee.

The general syntax for declaring an array of a structure can be given as,

```
struct struct_name
{
  data_type member_name1;
  data_type member_name2;
  data_type member_name3;
  ...
};
struct struct_name struct_var[index];
```

Consider the given structure definition.

```
struct student
{
  int r_no;
  char name[20];
  char course[20];
  float fees;
};
```

A student array can be declared simply by writing,

```
student stud[30];
```

Now, to assign values to the i^{th} student of the class, we will write,

```
stud[i].r_no = 09;
stud[i].name = "RASHI";
stud[i].course = "MCA";
stud[i].fees = 60000;
```

In order to initialize the array of structure variables at the time of declaration, you should write as follows:

```
student stud[3] = {{01, "Aman", "BCA",
45000},{02, "Aryan", "MCA", 60000}, {03,
"John", "BCA", 45000}};
```

6. Write a program to read and display the information of all the students in the class.

```
#include <stdio.h>
#include <conio.h>
int main()
{
   struct student
   {
      int roll_no;
      char name[80];
      int fees;
      char DOB[80];
   };
   struct student stud[50];
   int n, i;
   clrscr();
   printf("\n Enter the number of students: ");
   scanf("%d", &n);
   for(i=0; i<n; i++)
   {
      printf("\n Enter the roll number: ");
      scanf("%d", &stud[i].roll_no);
      fflush(stdin);
      printf("\n Enter the name: ");
      gets(stud[i].name);
      fflush(stdin);
      printf("\n Enter the fees: ");
      scanf("%d", &stud[i].fees);
      fflush(stdin);
      printf("\n Enter the DOB: ");
      gets(stud[i].DOB);
      fflush(stdin);
   }
   for(i=0; i<n; i++)
   {
      printf("\n ********DETAILS OF STUDENT
   %d ******", i+1);
      printf("\n ROLL No. = %d", stud[i].
   roll_no);
      printf("\n NAME = %s", stud[i].name);
      printf("\n FEES = %d", stud[i].fees);
      printf("\n DOB = %s", stud[i].DOB);
   }
   getch();
   return 0;
}
```

Output

```
Enter the number of students: 2
Enter the roll number: 1
Enter the name: kirti
Enter the fees: 5678
Enter the DOB: 9 9 91
Enter the roll number: 2
Enter the name: kangana
Enter the fees: 5678
Enter the DOB: 27 8 91
```

	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.