Advanced C Programming
UNIT-2: Structures and Unions

Basic of Structures

C supports a constructed data type known as structures, a mechanism for packing data of different types. A structure is a convenient tool for handling a group of logically related data items. For example, it can be used to represent a set of attributes, such as student_name, roll_number and marks. The concept of a structure is analogous to that of a 'record' in many other languages. More examples of such structures are:

Time	:	seconds, minutes, hours
Date	:	day, month, year
Book	:	author, title, price, year
City	:	name, country, population

Structures help to organize complex data in a more meaningful way. It is a powerful concept that we may often need to use in our program design.

Defining a Structure

Let us use an example to illustrate the process of structure definition and the creation of structure variables. Consider a book database consisting of book name, author, number of pages, and price. We can define a structure to hold this information as follows:

```
struct book_bank
{
        char title[20];
        char author[15];
        int pages;
        float price;
};
```

The keyword struct declares a structure to hold the details of four data fields, namely **title**, **auther**, **pages** and **price**. These fields are called structure elements or members. Each member may belong to a different type of data. **book_bank**is the name of the structure and is called the structure tag. The tag name may be used subsequently to declare variables that have the tag's structure.

The general format of a structure definition is as follows:

```
struct tag_name
{
        data_type  member1;
        data_type  member2;
        ….      …….     ………..
        ….      …….     ………..
};
```

In defining a structure you may note the following syntax:

1. The template is terminated with a semicolon.
2. While the entire definition is considered as a statement, each member is declared independently for its name and type in a separate statement inside the template.
3. The tag name such as **book_bank** can be used to declare structure variables of its type, later in the program.

Array V/s Structures

1. An array is a collection of related data elements of same type. Structure can have elements of different types.
2. An array is derived data type whereas a structure is a programmer-defined one.
3. Any array behaves like a built-in data type. All we have to do is to declare an array variable and use it. But in the case of a structure, first we have to design and declare a data structure before the variables of that type are declared and used.

Declaring Structure Variables

After defining a structure format we can declare variables of that type. A structure variable declaration is similar to the declaration of variables of any other data types. **It includes the following elements.**

1. The keyword **struct**
2. The structure tag name
3. List of variables names separated by commas
4. A terminating semicolon

For example, the statement

struct book_bank, book1, book2, book3;

declares **book1**,**book2** and **book3** as variables of type **struct book_bank.**
Each one of these variables has four members as specified by the template. The complete declaration might look like this:

```
struct book_bank
{
        char title[20];
        char author[15];
        int pages;
        float price;
};
struct book_bank, book1, book2, book3;
```

Accessing Structure Members

We can access and assign values to the members of a structure in a number of ways. The members of structure should be linked to the structure variables in order to make them meaningful members. For example, the word **title** has no meaning whereas the phrase ‘title of book3’ has a meaning. The link between a member and a variable is established using the member operator ‘.’ Which is also known as ‘dot operator’ or ‘period operator’? For example,

```
book1.price
```

is the variable representing the price of book1 and can be treated like any other ordinary variable. Here is how we would assign values to the members of book1:

```
strcpy(book1.title,”BASIC”);
strcpy(book1.author, “Balaguruswamy”);
book1.pages = 250;
book1.price = 120.50;
```

We can also use scanf to give the values through the keyboard.

```
scanf(“%s\n”,book1.title);
scanf(“%d\n”,&book1.pages);
```

are valid input statements.

Structure Initialization

Like any other data type, a structure variable can be initialized at compile time.

```
struct st_record
{
        int weight;
        float height;
} student1 = {6-,180.75};
void main()
{
        struct st_record student2 = {53, 170.60};
}
```

C language does not permit the initialization of individual structure members within the template. The initialization must be done only in the declaration of the actual variable.

Note that the compile-time initialization of a structure variable must have the following elements:

1. The keyword struct
2. The structure tag name
3. The name of the variable to be declared
4. The assignment operator =
5. A set of values for the members of the structure variable, separated by commas and enclosed in braces
6. A terminating semicolon

Rules for initializing structures

There are a few rules to keep in mind while initializing structure variables at compile-time.

1. We cannot initialize individual members inside the structure template.
2. The order of values enclosed in braces must match the order of members in the structure definition
3. It is permitted to have a partial initialization. We can initialize only the first few members and leave the remaining blank. The uninitialized members should be only at the end of the list.
4. The uninitialized members will be assigned default values as follows:
    a. Zero for integer and floating point numbers
    b. '\0' for character and strings

typedef (Type definition)

'C' supports a feature known as "type definition" that allows user to define an identifier that would represent an existing data type. Later on this identifier can be used for variable declaration. 'typedef' cannot create new type. The advantage of typedef is that you can use appropriate format for variable declaration which increase the readability of your program. The format of the "type definition" is:

typedef data-type identifier;

Here the ‘data-type’ refers to basic data-type used in ‘C’ program and ‘identifier’ is the name given by the user.For example in below example we have created typedef ‘marks’ for integer variable. Now you can use the ‘marks’ to declare the variable of integer type. Now you can use ‘int’ or ‘marks’ for integer type declaration. Here ‘sub1’, ‘sub2’, ‘sub3’ are variables of type int declared using typedef ‘marks’.

```
typedef int marks;
void main()
{
        marks sub1, sub2, sub3;
        ……………………………
}
```

Arrays of Structures

We use structures to be describing the format of a number of related variables. For example, in analyzing the marks obtained by a class of students, we may use a template to describe student name and marks obtained in various subjects and then declare all the students as structure variables. In such cases, we may declare an array of structures, each element of the array representing a structure variable. for example,

struct class student[100];

defines an array called student, that consists of 100 elements. Each element is defined to be of the type struct class. Consider the following declaration:

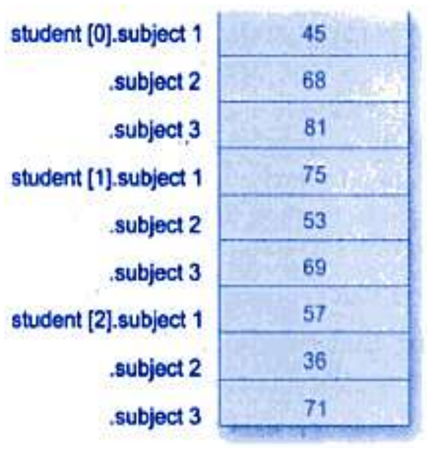
```
struct marks
{
        int subject1;
         int subject2;
        int subject3;
};
void main()
{
        struct marks student[3]={{45,68,81},{75,53,69},{57,36,71}
     };
```

This declares the student as an array of three elements student[0], student[1] and student[2] and initializes their members as follows:

```
student[0].subject1=45;
student[0].subject2=65;
        ………..
        ………..
student[2].subject3=71;
```

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. Remember, each element of student array is a structure variable with three members.

Any array of structures is stored inside the memory in the same way as a multi-dimensional array. The array student actually looks as show in below figure.



Arrays within Structures

C permits the use of arrays as structure members. We have already used arrays of characters inside a structure. Similarly, we can use single-or multi-dimensional arrays of type int or float. For example, the following structure declaration is valid:

```
struct marks
{
        int number;
        float subject[3];
} student[2];
```

Here, the member subject contains three elements, subject[0], subject[1] and subject[2]. These elements can be accessed using appropriate subscripts. For example, the name

```
student[1].subject[2];
```

would refer to the marks obtained in the third subject by the second student.

Structures within structures (Nested Structures)

Structure within structure means nesting of structure. We can define the new structure inside one structure or we can declare the variable of one structure inside another structure. For example in following example we have defined two structures for Address and Student.

```
struct address
{
        char society[100];
        char area[100];
        char city[50];
        int pincode;
};

struct student
{
        int rollno;
        char name[50];
        struct address add;           // Structure inside structure
};
```

In above example we have defined one structure having member variable society, area, city, and pincode number. Then we have defined another structure having member variable roll number, name, and structure variable of structure address.

We can also define above structure by following way. Any method is valid for C program.

```
struct student
{
        int rollno;
        char name[50];

        // Structure address defined inside the structure
        struct address
        {
                char society[100];
                char area[100];
                char city[50];
                int pincode;
        }add;
};
```

We can access the members of nested structure in program as under:

```
void main( )
{
        struct student s;
        printf("Enter student detail\n");
        scanf("%d", &s.rollno);
        gets(s.name);
        // Accessing the member of nested structure
        gets(s.add.society);
        gets(s.add.area);
        gets(s.add.city);
        scanf("%d",&s.add.pincode);
}
```

Structures and Functions

We can pass the structure to the user defined function with many ways. The one 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. It is, therefore, necessary for the function to return the entire structure back to the calling function.

PASSING STRCUTRE VARIABLE INSIDE FUNCTION

As like as simple variable and array variable we can also pass the structure variable inside the function. Following example demonstrate passing structure variable inside the function.

```
struct student
{
        int rollno;
        char name[100];
};
// Function declaration
void print(struct student);
void main( )
{
        struct student s;
        printf("Enter Roll No. and Name of student : ");
        scanf("%d",&s.rollno);
        gets(s.name);
        // Call a function
        print(s);
}
// Function definition
void print(struct student s)
{
        print("Roll No. = %d\n",s.rollno);
        printf("Name = %s\n",s.name);     }
```

In above example we have defined one structure template for the student, which contain roll number and name as structure member. We have used one function print( ) to print the member of structure and we have passed that variable as argument inside function.

RETURNING STRUCTURE VARIABLE FROM FUNCTION

As like passing the structure variable inside structure we can return structure variable from function. But it is supported only when our compiler supports the assignment operation over structure.

```
struct student
{
int rollno;
char name[100];
};

// Function declaration

struct student read( );

void main( )
{
        struct student s;
        // Call function
        s = read( );
}



// Function definition
struct student read( )
{
        struct student t;
        printf("Enter Roll No. and Name of student\n");
        scanf("%d",&t.rollno);
        gets(t.name);
        return( t );   // Returning structure variable
}
```

In above example we have defined one structure template for the student, which contain roll number and name as structure member. We have used one function read( ) to read the data of structure. This function return the value of type struct student and then it is assigned to variable used inside main( ) function.

Important Points

1. The called function must be declared for its type, appropriate to the data type it is expected to return. For example, if it is returning a copy of the entire structure, then it must be declared as struct with an appropriate tag name.
2. The structure variable used as the actual argument and the corresponding formal argument in the called function must be of the same struct type.
3. The return statement is necessary only when the function is returning some data back to the calling function. The expression may be any simple variable or structure variable to an expression using simple variable.
4. When a function returns a structure, it must be assigned to a structure of identical type in the calling function.
5. The called functions must be declared in the calling function appropriately.

Unions

Unions are a concept borrowed from structure and therefore follow the same syntax as structures. However, there is major distinction between them in terms of storage. In structures, each member has its own storage location, whereas all the members of a union use the same location. This implies that, although a union may contain many members of different types, it can handle only one member at a time. Union can be declared using the keyword union as follows:

```
union item
{
        int m;
        float x;
        char c;
 } code;
```

This declares variable code of type union item. The union contains three members, each with a different data type. However, we can use only of them at a time. This is due to the fact that only one location is allocated for a union variable, irrespective of its size.

The complier allocates a piece of storage that is large enough to hold the largest variable type in the union. To access a union member, we can use the same syntax that we use of structure members. That is,

```
code.m
code.x
code.c
```

All are valid members variables. During accessing, we should make sure that we are accessing the member whose value is currently store.

```
code.m = 379;
code.x = 7859.36;
prinft("%d",code.m);
```

Would produce erroneous output(which is machine independent).

A union creates a storage location that can be used by any one of its members at a time. When different member is assigned a new value, the new value supersedes the previous member a value.

Unions may be used in all places where a structure is allowed. The notation for accessing a union member which is nested inside a structure remains the same as for the nested structures.

Difference between Structure and Union

	Unit-2
	MCQ
1	A _____________ is a collection of data items under one name in which the items share the same storage. (a) structure (b) array **(c) union** (d) none
2	A _____________ is a collection of different data items. **(a) structure** (b) array (c) char (d) none
3	The name of a structure is referred to as ____________. (a) Label **(b) Tag Name** (c) Index (d) none
4	______ operator connects the structure name to its member. (a) underscore **(b) dot** (c) ! (d) none
5	Which of the following cannot be a structure member? (a) Another structure **(b) Function** (c) Array (d) none

6	Number of bytes in memory taken by the below structure is ______
	struct test
	{
	        int k;
	        char c;
	};
	(a) Multiple of integer size **(b) integer size+character size**
	(c) Depends on the platform (d) Multiple of word size

7	Size of a union is determined by size of the. (a) First member in the union (b) Last member in the union **(c) Biggest member in the union** (d) Sum of the sizes of all members
8	Members of a union are accessed as________________. **(a) union-name.member** (b) union-pointer->member (c) Both a & b (d) none
9	Which of the following share a similarity in syntax? (a) Union (b) Structure (c) Arrays **(d) Both a &b**
10	The variables declared in a structure definition are called as its__________. (a) objects **(b) members** (c) record (d) none

	Unit-2
	Short Questions
1	Write and explain syntax for structure definition.
2	Write and explain syntax for structure variable declaration.
3	Which elements are required for compile time initialization of a structure variable?
4	Which are the ways to access structure members?
5	Differentiate structure and union.
	Unit-2
	Long Questions
1	Explain structure definition with syntax example.
2	Explain declaring structure variable with syntax and example.
3	Explain how we can access structure members with example.
4	Explain structure initialization with example.
5	Explain array of structure with example.
6	Explain array within structure with example.
7	Explain structure within structure with example.
8	Which different methods for transferring structure from one function to another function? Explain any one method with example.
9	Explain union with syntax and example.
10	Differentiate structure and union.