UNIT- IV
Structures & unions

Structures – Introduction

Structure is a compound data type. It is used to store different types of data items. Sometimes a structure is called as a record. The individual structure element is referred to as "members". A structure is a convenient method of handling a group of related data items of different data types.

Differences between Array and structure

Array	Structure
Array is a collection of homogeneous data.	Structure is a collection of heterogeneous data.
Array elements are referred by subscript.	Structure elements are referred by its unique name.
Array elements are accessed by it's position or subscript.	Structure elements are accessed by its object as '.' operator.
Array is a derived data type.	Structure is user defined data type.

Defining a structure

The general format to define a structure is as follows**.**

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

…
};
```

Here 'struct' is the keyword to declare a structure. tag _ name can be any user-defined identifier. The structure declaration should always end with ';'.

Example

Consider a library database consisting of title of the book, autor name, number of pages and price. A structure used to define the above is

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

The above example declares a structure "lib_books" with 4 members : title, author, pages and price.

Variable Declaration

The compiler does not reserve any memory space when a structure is declared. Memory space is reserved when only a variable of this type is defined.

Structure variable is defined anywhere in the program by using the tag-name. For example, the statement

```
struct lib_books c, java, basic;
```

defines c, java, basic as variables of type struct lib_books..

The syntax for creating the structure variable is

```
struct tag_name varaibe_name1, variable_name2,…….
```

The memory allocated for a structure variable is equal to the sum of memory allocated for all its members.
For the above example 41 byte are allocated, for each variable, which is calculated as follows. title 20 bytes (20 * 1) author- 15 bytes pages - 2 bytes price - 4 bytes
It is also possible to combine both definition and variable declaration in the same statement. For example

```
struct lib_books
{
char title[20];
char author[15];
int pages;
float price;
} c, java, basic;
```

is also valid. The use of tag-name is optional when declaring the variables during structure definition.

Initialization of a structure

The members of a structure can be initialized to constant values by enclosing the values to be assigned within the braces after the structure definition. Thus the declaration

```
struct date
{
int date;
int month;
int year;
} republic = {26,1,1950};
```

initializes the member variables date, month, year of republic to 26,1,1950 respectively.

Struct date Gandhi_jay = {2,10,1869) initializes the member variable date, month and year of the structure variable Gandhi_jay to 2,10,1869 respectively.

If number of elements placed in an initialization is less than the member of members present, then the remaining members are initialized to zero, or NULL.

Arrays of structures

Arrays of structures are commonly used when a large number of similar records are required to be processed together.
Assume, a library has 1000 books. The data related to the above books can be organized in an array of structures. The idea is very simple. First create a structure template. (i.e., declare a structure).
Then define an array with this structure for a specified size.
Consider the following example.

```
struct lib_books
{
char title[20];
int pages;
} ;
```

Then array of structure used is struct lib_books library {1000].

In the above statement library is an array containing 1000 elements of the type struct lib_books. The advantage is that all these 1000 records can be easily manipulated with the help of loops.
The first record is referred as library[0], second record is referred by library [1] and so on. The first record's title and pages fields are referred as library [0].title and library[0].page. Hence array of structures is used to manage large number of records easily.

Arrays within structures

A structure can have an array within it. Consider the above example 'lib-books'. Here the first field name is an array which is declared as

```
char title[20];
```

Like this a structure can have number of arrays as members.
Example

```
struct student
{
char name[20];
int semester;
int year;
char branch[10];
int mark[3];
};
```

In the above example there are 5 members namely name, semester, year, branch and mark. Out of these 5 fields, name and branch are character arrays and mark is an integer array. The member mark contains three elements mark[0], mark[1] and mark[2]. If the structure is used to create a variable, then the total amount of memory allocated is 45 bytes (25+2+2+10+2*3) .
Now let us see how to refer the various elements of the array mark. Consider the following definition.

```
main()
{ struc student DCT;
____
}
```

The first subject mark is referred as DCT. mark[0]. And second subject mark as DCT.mark[1] and so on. This is how arrays can be placed inside structure in order to declare an user defined data type that is representing a complex real world entity.

Unions

Introduction

Union is another compound date type like structure. Union is used to minimize memory utilization. In structure, each member has the separate storage location. But in union, all the members share the common place of memory. Therefore, a union can handle only one member at a time.

Unions are useful for application involving multiple members, where values need not be assigned to all the members at any one time.

Declaration of union

Like structures union can be declared using the keyword union. The general format for declaring union is,

```
union tag_name
{
data_type member_1;
data_type member_2;
```

```
--------
data_type member_n;
};
```

where union is the key word. tag-name is any user defined data name.

Example

```
union Bio
{
char name[10];
int age ;
float height;
};
```

In the above example 'union Bio' has 3 members. First member is a character array 'name' having 10 characters (i.e., 10 bytes). Second member is an integer 'age' that requires 2 bytes. Third member is a float 'height' requiring 4 bytes. All the three members are different data types.

Here all these 3 members are allocated with common memory. They all share the same memory. The compiler allocates a place of storage that is large enough to hold the largest type in the union. In the declaration above, the member "name" requires 10 bytes, which is the largest among the members.

The total memory allotted is also different from structures. In case of structure it will be 10+2+4 = 16. But here it is only 10.

Union variables are created just like structure variables. For example to create a variable for above 'union Bio' , the following code is used.

```
main()
{
union Bio studentBio;
---------
}
```

Initializing Unions

Unions must always be initialized with its first field only. For example to initialize the above 'Union Bio' the statement used is

```
main()
{
union Bio studentBio = {"Rama"};;
---------
}
```

Here first element is a character array. Hence the variable studentBio is initialized with "Rama" which is a character constant. The following are erronous initialization for the 'union Bio'.

```
union Bio studentBio = {32};
union Bio studentBio = {172.3};
```

Advantages of Union

- It is used to minimize memory utilization.
- It is used to convert data from one type to another type.
- It is used to write a record into a file as a character.

Differences between Structure and Unions

Structures	Unions
Every member has its own memory space	All members use the same memory space
Can handle all members as required at a time	Can handle only one member at a time
All members can be initialized	Only first member may be initialized
Difference Interpretations for the same memory location is not possible	Different Interpretations for the same memory location are possible
More storage space required	Conservation of memory is possible