

Module 8

Structure/Union

1.1 Structures

Introduction

Arrays are useful to refer separate variables which are the same type. i.e. Homogeneous referred by a single name. But, we may have situations where there is a need for us to refer to different types of data (Heterogeneous) in order to derive meaningful information.

Let us consider the details of an employee of an organization. His details include employer's number (Integer type), employee's name (Character type), basic pay (Integer type) and total salary (Float data type). All these details seem to be of different data types and if we group them together, they will result in giving useful information about employee of the organization.

In above said situations, C provides a special data types called Structure, which is highly helpful to organize different types of variables under a single name.

Definition of Structure

A group of one or more variables of different data types organized together under a single name is called **Structure**.

Or

A collection of heterogeneous (dissimilar) types of data grouped together under a single name is called a **Structure**.

A structure can be defined to be a group of logically related data items, which may be of different types, stored in contiguous memory locations, sharing a common name, but distinguished by its members.

Hence a structure can be viewed as a heterogeneous user-defined data type. It can be used to create variables, which can be manipulated in the same way as variables of built-in data types. It helps better organization and management of data in a program.

When a structure is defined the entire group is referenced through the structure name. The individual components present in the structure are called structure members and those can be accessed and processed separately.

Structure Declaration

The declaration of a structure specifies the grouping of various data items into a single unit without assigning any resources to them. The syntax for declaring a structure in C is as follows:

```
struct Structure Name
{
    Data Type member-1; Data
    Type member-2;
    ....
    Data Type member-n;
};
```

The structure declaration starts with the structure header, which consists of the keyword '**struct**' followed by a tag. The tag serves as a structure name, which can be used for creating structure variables. The individual members of the structure are enclosed between the curly braces and they can be of the similar or dissimilar data types. The data type of each variable is specified in the individual member declarations.

Example:

Let us consider an employee database consisting of employee number, name, and salary. A structure declaration to hold this information is shown below:

```
struct employee
{
```

```

        int eno;
        char name [80];

        float sal;

    };

```

The data items enclosed between curly braces in the above structure declaration are called structure elements or structure members.

Employee is the name of the structure and is called structure tag. Note that, some members of employee structure are integer type and some are character array type.

The individual members of a structure can be variables of built – in data types (int, char, float etc.), pointers, arrays, or even other structures. All member names within a particular structure must be different. However, member names may be the same as those of variables declared outside the structure. The individual members cannot be initialized inside the structure declaration.

Note

Normally, structure declarations appear at the beginning of the program file, before any variables or functions are declared.

They may also appear before the main (), along with macro definitions, such as #define.

In such cases, the declaration is global and can be used by other functions as well.

Structure Variables

Similar to other types of variables, the structure data type variables can be declared using structure definition.

```

struct
{
    int    rollno;

    char name[20];

    float average; a, b;

}

```

In the above structure definition, a and b are said to be structure type variables. 'a' is a structure type variable containing rollno, name average as members, which are of different data types. Similarly 'b' is also a structure type variable with the same members of 'a'.

Structure Initialization

The members of the structure can be initialized like other variables. This can be done at the time of declaration.

Example 1

```

struct
{
    int    day;
    int    month;
    int    year;
}
date = { 25,06,2012};
i.e
date. day = 25

date. month = 06

date. year = 2012

```

Example 2

```

struct address
{char    name [20];

```

```

char    design [10];
char    place [10];
};
struct address my-add={ 'Sree', 'AKM', 'RREDDY' };

```

i.e

```
my-add . name = 'Sree'
```

```
my-add . design = AKM
```

```
my-add . place = RREDDY
```

As seen above, the initial values for structure members must be enclosed with in a pair of curly braces. The values to be assigned to members must be placed in the same order as they are specified in structure definition, separated by commas. If some of the members of the structure are not initialized, then the c compiler automatically assigns a value 'zero' to them.

1.2 Accessing Structure Elements

As seen earlier, the structure can be individually identified using the period operator (.). After identification, we can access them by means of assigning some values to them as well as obtaining the stored values in structure members. The following program illustrates the accessing of the structure members.

Example: Write a C program, using structure definition to accept the time and display it.

```

/* Program to accept time and display it */ #
include<stdio.h>

main( )
{
    struct
    {
        int hour, min;
        float seconds;
    } time;
    printf ( "Enter time in Hours, min and Seconds\n");

    scanf ( "%d %d %f", &time . hour, &time . min, &time . seconds); printf ( "The
    accepted time is %d %d %f", time . hour, time . min, time
    . seconds ");
}

```

Nested Structures

The structure is going to contain certain number of elements / members of different data types. If the members of a structure are of structure data type, it can be termed as structure with structure or nested structure.

Example

```

struct
{
    int    rollno; char
    name[20]; float
    avgmarks; struct
    {

```

```

        int day, mon, year;
    } dob;
} student;

```

In the above declaration, student is a variable of structure type consisting of the members namely rollno, name, avgmarks and the structure variable dob.

The dob structure is within another structure **student** and thus structure is nested. In this type of definitions, the elements of the require structure can be referenced by specifying appropriate qualifications to it, using the period operator (.).

For example, **student.dob.day** refers to the element day of the inner structure dob.

1.3 Way of Storage of Structure Element

Whatever be the elements of a structure, they are always stored in contiguous memory locations. The following program would illustrate this:

```

/* Memory map of structure elements */
main( )
{
    struct book
    {
        char  name ;
        float price ;
        int   pages ;
    };
    struct book  b1 = { 'B', 130.00, 550 };

    printf ( "\nAddress of name = %u", &b1.name ) ;
    printf ( "\nAddress of price = %u", &b1.price ) ;
    printf ( "\nAddress of pages = %u", &b1.pages ) ;
};

```

Output: Here is the output of the program...

Address of name = 65518

Address of price = 65519

Address of pages = 65523

Structure memory allocation

1. how structure members are stored in memory?
2. What is structure padding?

1. HOW STRUCTURE MEMBERS ARE STORED IN MEMORY?

Always, contiguous(adjacent) memory locations are used to store structure members in memory. Consider below example to understand how memory is allocated for structures.

EXAMPLE PROGRAM FOR MEMORY ALLOCATION IN C STRUCTURE:

```

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

```

```

struct student

```

```

{
    int id1;
    int id2;
    char a;
    char b;
    float percentage;
};

int 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);

    return 0;
}
#include <stdio.h>
#include <string.h>

```

OUTPUT:

```

size of structure in bytes : 16
Address of id1 = 675376768
Address of id2 = 675376772
Address of a = 675376776
Address of b = 675376777
Address of percentage = 675376780

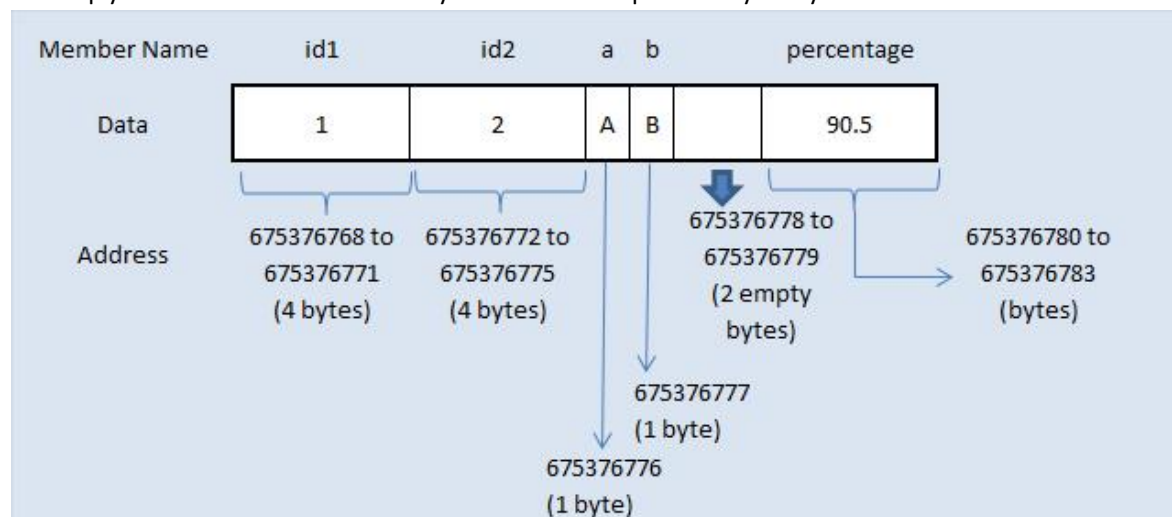
```

There are 5 members declared for structure in above program. In 32 bit compiler, 4 bytes of memory is occupied by int datatype. 1 byte of memory is occupied by char datatype and 4 bytes of memory is occupied by float datatype.

Please refer below table to know from where to where memory is allocated for each datatype in contiguous (adjacent) location in memory.

Datatype	Memory allocation in C (32 bit compiler)		
	From Address	To Address	Total bytes
int id1	675376768	675376771	4
int id2	675376772	675376775	4
char a	675376776		1
char b	675376777		1
Addresses 675376778 and 675376779 are left empty. (Do you know why? Please refer below, the next topic C - Structure padding)			2
float percentage	675376780	675376783	4

The pictorial representation of above structure memory allocation is given below. This diagram will help you to understand the memory allocation concept in C very easily.



2. What is Structure Padding?

- In order to align the data in memory, one or more empty bytes (addresses) are inserted (or left empty) between memory addresses which are allocated for other structure members while memory allocation. This concept is called structure padding.
- Architecture of a computer processor is such a way that it can read 1 word (4 byte in 32 bit processor) from memory at a time.
- To make use of this advantage of processor, data are always aligned as 4 bytes package which leads to insert empty addresses between other member's address.
- Because of this structure padding concept in C, size of the structure is always not same as what we think.

1.4 Defining Structure and Array of Structure

Structures and Arrays

Array is group of identical stored in consecutive memory locations with a single / common variable name. This concept can be used in connection with the structure in the following ways.

- a. Array of structures
- b. structures containing arrays (or) arrays within a structure
- c. Arrays of structures contain arrays.

Array of Structures

Student details in a class can be stored using structure data type and the student details of entire class can be seen as an array of structure.

Example

```
struct student
{
    int rollno;

    int year;

    int tmarks;
}
struct student class[40];
```

In the above class [40] is structure variable accommodating a structure type student up to 40.

The above type of array of structure can be initialized as under

```
struct student class [2] = { {001,2011,786},{ 002, 2012, 710}};
```

```
i.e          class[0] . rollno = 001
              class[0] . year = 2011
              class[0] . tmarks = 777
              and
              class[1] . rollno = 002
              class[1] . year = 2012
              class[1] . tmarks = 777
```

Structures containing Arrays

A structure data type can hold an array type variable as its member or members. We can declare the member of a structure as array data type similar to int, float or char.

Example

```
struct employee
{
    char ename [20]; int
    eno;
};
```

In above, the structure variable employee contains character array type ename as its member. The initialization of this type can be done as usual.

```
struct employee = { 'Rajashekar', 7777};
```

Arrays of Structures Contain Arrays

Arrays of structures can be defined and in that type of structure variables of array type can be used as members.

Example

```

struct rk
{
    int empno;
    char ename[20];

    flat salary;

} mark[50];

```

In the above, mark is an array of 50 elements and such element in the array is of structure type rk. The structure type rk, in turn contains ename as array type which is a member of the structure. Thus mark is an array of structures and these structures in turn hold character names in array ename.

The initialization of the above type can be done as:

```

{
    7777, 'Prasad', 56800.00}
};

```

i.e mark[0] . empno = 7777; mark[0] .

```

    eame = 'Prasad'; mark[0] . salary =
    56800.00

```

Program

Write a C program to accept the student name, rollno, average marks present in the class of student and to print the name of students whose average marks are greater than 40 by using structure concept with arrays.

```

#include<stdio.h>

main()
{
    int i, n, struct
    {
        char name [20];
        int rollno;
        flat avgmarks;
    }
    class [40];
    printf("Enter the no. of students in the class\n"); scanf( "%d",
    & n);

    for ( i = 0, i < n, i++)
    {
        print ( " Enter students name, rollno, avgmarks\n");
        scanf( " %s %d", &class[i].name, class[i].rollno, &class[i].avgmarks)'
    }
    printf("The name of the students whose average"); printf
    ( " marks is greater than 40\n");

    for ( i = 0, i < n, i++)
    if ( class[i].avgmarks > 40) printf
    ( " %s", class[i].name);
}

```


Advantages of Structure Type over Array Type Variables

1. Using structures, we can group items of different types within a single entity, which is not possible with arrays, as arrays store similar elements.
2. The position of a particular structure type variable within a group is not needed in order to access it, whereas the position of an array member in the group is required, in order to refer to it.
3. In order to store the data about a particular entity such as a 'Book', using an array type, we need three arrays, one for storing the 'name', another for storing the 'price' and a third one for storing the 'number of pages' etc., hence, the overhead is high. This overhead can be reduced by using structure type variable.
4. Once a new structure has been defined, one or more variables can be declared to be of that type.
5. A structure type variable can be used as a normal variable for accepting the user's input, for displaying the output etc.,
6. The assignment of one 'struct' variable to another, reduces the burden of the programmer in filling the variable's fields again and again.
7. It is possible to initialize some or all fields of a structure variable at once, when it is declared.
8. Structure type allows the efficient insertion and deletion of elements but arrays cause the inefficiency.
9. For random array accessing, large hash tables are needed. Hence, large storage space and costs are required.
10. When structure variable is created, all of the member variables are created automatically and are grouped under the given variable's name.

Structure Contains Pointers

A pointer variable can also be used as a member in the structure.

In the above, *rr is a pointer variable of structure type which holds inside it another two pointer variables p1 and p2 as its members.

```
#include <stdio.h>

main()
{
    struct
    {
        int *p1, *p2;
    } *rr;

    int a, b ;

    a = 70;

    b = 100;
    rr -> p1 = &a; rr ->
    p2 = &b;

    printf( " The contents of pointer variables");
    printf( " Present in the structure as members are \n"); printf
    ( '%d %d', *rr -> p1, *rr -> p2);

}
```

In the above program, two pointer variables p1 and p2 are declared as members of the structure and their contents/variables are printed after assignment in the program.

1.5 Basic Definition of Union

Introduction

A Union is a collection of heterogeneous elements. That is, it is a group of elements; each element is of different type. They are similar to structures. However, there is a difference in the way the structures members and union members are stored. Each member within a structure is assigned its own memory location. But the union members all share the common memory location. Thus, unions are used to save memory. Unions are chosen for applications involving multiple members, where values need to be assigned to all of the members at any one time.

Union is a data type through which objects of different types and sizes can be stored at different times.

The general form of union type variable declaration is Union name

Definition of Union

```
{
    data type member-1;
        data type
    member-2; data type
    member-3;
        .....
        .....
    data type member-n;
}
```

The declaration includes a key word Union to declare the union data type. It is followed by user defined name, followed by curly braces which includes the members of the union

Example

```
union      value
{
    int
    no;
    float
    sal;
    char
    sex;
};
```

Characteristics of Union

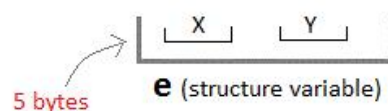
1. Union stores values of different types in a single location in memory
2. A union may contain one of many different types of values but only one is stored at a time.
3. The union only holds a value for one data type. If a new assignment is made the previous value has no validity.
4. Any number of union members can be present. But, union type variable takes the largest memory occupied by its members.

1.6 Comparison between Structure and Union with example

Structure	Union
1. Struct StructureName <pre>{ datatype member-1; datatype member-2; datatype member-n; };</pre>	1. Union name { <pre>datatype member-1; datatype member-2; datatype member-n; };</pre>
2. Every structure member is allocated memory when a structure variable is defined	2. The memory equivalent to the largest item is allocated commonly for all members
3. All the members can be assigned values at a time	3. Values assigned to one member may cause the change in value of other members.
4. All members of a structure can be initialized at the same time	4. Only one union member can be initialized at a time
5. value assigned to one member will not cause the change in other members	5. Value assigned to one member may cause the change in value of other members.
6. The usage of structure is efficient when all members are actively used in the program	6. The usage of union is efficient when members of it are not required to be accessed at the same time.

Structure

```
struct Emp
{
char X; // size 1 byte
float Y; // size 4 byte
} e;
```



Unions

```
union Emp
{
char X;
float Y;
} e;
```

