Module 5

Structure:

1. Structure is a user-defined data type in C which allows combining different data types to store a particular type of record.
2. Structure is a collection of the data of different data types.
3. The only difference is that array is used to store collection of similar data types while structure can store collection of any type of data.
4. struct keyword is used to define a structure.
5. Structure declaration syntax:

   ```
   struct structure-name
   {
   Data-type structure-element1;
   Data-type structure-element2;
   Data-type structure-element3;
   ......
   } ;
   ```

   Note: The closing brace in the structure type declaration must be followed by a semicolon.
6. Structure variables declaration: After the structure is defined, structure variable is declared in order to use it. Syntax:

   ```
   struct structure-name variable1,variable2,variable3;
   ```
7. Example:

   ```
   struct Book
   {
   char name[100];
   float price;
   int pages;
   };

   struct book b1, b2, b3;
   ```
8. Accessing Structure Members: Structure members can be accessed and assigned values using dot (.) operator; also called as period or member access operator. Syntax:

   ```
   Structure-variable.structure-element;
   Example:
   b1.price=200;     //b1 is variable of structure 'Book' and price is element of Book
   ```

Array of Structure:

1. It is possible to define a array of structures when there is a need for multiple structure variables.
2. Each element of an array represents a structure variable.
3. Syntax: struct structure-name array-name[array-size];
4. Example: To maintain information of 10 books, we need to use an array instead of 10 structure variables.

   ```
   struct Book arr[10];
   ```

Nested Structures:

1. If one of the members of structure is also a structure, then such a structure is called as a nested structure.
2. A structure may be defined as a member of another structure.
3. Syntax:

   ```
   struct structure-name {
   Data-type structure-element1;
   struct internal-structure-name {
       Data-type structure-element2;
       Data-type structure-element3;
       ….
   ```

```
    } internal-structure-variable-name;
    ….
    };
```

4. Example:

```
struct student {
char name[30];
int age;
struct address {
    char area[50];
    char city[50];
    int pincode;
    };
};
```

Structure as function arguments:

1. Structure can be passed as a function argument in similar way as any other variable or array.
2. Example:

```
#include<stdio.h>
struct student {
char name[10];
int roll;
};
void show(struct);
void main() {
struct student std;
printf("\nEnter student record\n");
printf("\nstudent name\t");
scanf("%s",std.name);
printf("\nEnter student roll\t");
scanf("%d",&std.roll);
show(std);
getch();
}
void show(struct student st) {
printf("\nstudent name is %s",st.name);
printf("\nroll is %d",st.roll);
}
```

Union

1. A union is a special data type available in C that allows storing different data types in the same memory location.
2. A union can be defined with many members, but only one member can contain a value at any given time.
3. Hence, unions provide an efficient way of using the same memory location for multiple-purpose.
4. union keyword is used to define a structure.
5. Union declaration syntax:

```
union union-name
{
Data-type union-element1;
Data-type union-element2;
Data-type union-element3;
.....
};
```

6. Union variables declaration: After the union is defined, union variable is declared in order to use it. Syntax: union union-name variable1,variable2,variable3;

7. Example:

```
union Data {
int i;
char str[20];
};

union Data data;
```

8. **Note:** The memory occupied by a union will be large enough to hold the largest member of the union. In the above example, Data type will occupy 20 bytes of memory space because it is the maximum space which can be occupied by a character string.
9. Union members can be accessed and assigned values using dot (.) operator;
10. Program:

```
#include<stdio.h>
#include<string.h>
union Data {
int i;
char str[20];
};

void main( ) {
union Data data;
data.i = 10;
strcpy(data.str, "C Programming");
printf("data.i : %d\n", data.i);
printf("data.str : %s\n", data.str);
}
```

When the above code is compiled and executed, it produces the following result:

```
data.i : 1917853763
data.str : C Programming
```

Here, the value of 'i' got corrupted because the final value assigned to the variable has occupied the memory location, i.e., the value of 'str' member is getting printed very well.

11. Program:

```
#include<stdio.h>
#include<string.h>
union Data {
int i;
char str[20];
};
void main( ) {
union Data data;
data.i = 10;
printf("data.i : %d\n", data.i);
strcpy(data.str, "C Programming");
printf("data.str : %s\n", data.str);
}
```

When the above code is compiled and executed, it produces the following result:

```
data.i : 10
data.str : C Programming
```

Here, all the members are getting printed very well because one member is being used at a time.

No.	Structure	Union
1.	Keyword struct defines a structure.	Keyword union defines a union.
2.	The members of a structure all begin at different offsets from the base of the structure.	In a union all members have offset zero from the base
3.	Within a structure all members gets memory allocated; therefore any member can be retrieved at any time.	While retrieving data from a union the type that is being retrieved must be the type most recently stored.
4.	One or more members of a structure can be initialized at once.	A union may only be initialized with currently required type.
5.	Data is more secured in structure.	Data can be corrupted in a union.
6.	Memory allotted to structure is equal to the space require collectively by all the members of the structure.	Memory allotted for a union is equal to the space required by the largest memory of that union
7.	Structure provides ease of programming.	Unions are difficult for programming.
8.	Structures require more memory.	Unions require less memory.
9.	Structure must be used when information of all the member elements of a structure are to be stored.	Unions must be used when only one of the member elements of the union is to be stored.
10.	struct s_name { int ival; float fval; }s;	union u_name { int ival; float fval; }u;