Structure and Union

C Structure is a collection of different data types which are grouped together

Difference between C variable, C array and C structure:

- A normal C variable can hold only one data of one data type at a time.
- An array can hold group of data of same data type.
- A structure can hold group of data of different data types and Data types can be int, char, float, double and long double etc.

Syntax:

```
struct    name
{
data type      var_name1;
data type      var_name2;
data type      var_name3;
};
```

Example:

```
struct student
{
   int  mark;
  char name[10];
  float average;
};
```

```
struct Books
{
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
} ;
```

```
struct database
{
  int id_number;
  int age;
  float salary;
};
```

```
struct date
{
    int dd;
    int mm;
    int yy;
}
```

Declaring structure using normal variable:

struct student;

Initializing structure using normal variable:

struct student   s1 = {100, "abc", 99.5};

Accessing structure members using normal variable:

To access any member of a structure, we use the member access operator (.).
s1.mark;
s1.name;
s1.average;

Following program demonstrate a structure example.

```
#include <stdio.h>
#include <string.h>
 struct student
{
        int id;
        char name[20];
        float percentage;
};
void main()
{
        struct student  s1  ;
         s1.id=1;
        strcpy(s1.name, "Raju");
        s1.percentage = 86.5;

        printf(" Id is: %d \n",s1 .id);
        printf(" Name is: %s \n", s1.name);
        printf(" Percentage is: %f \n", s1.percentage);
}
```

Output:

```
Id                      is:                     1
Name                    is:                  Raju
Percentage is: 86.500000
```

Array of Structures

array of structures is nothing but collection of structures.

```
#include <stdio.h>
#include <string.h>
 struct student
{
    int id;
    char name[30];
    float percentage;
};
void  main()
{
    int i;
    struct student s1[2];         // s1[2] is array of structure
     // 1st student's s1
    s1[0].id=1;
    strcpy(s1[0].name, "aa");
    s1[0].percentage = 86.5;

    // 2nd student's s1
    s1[1].id=2;
    strcpy(s1[1].name, "bb");
    s1[1].percentage = 90.5;

     // 3rd student's s1
    s1[2].id=3;
    strcpy(s1[2].name, "cc");
    s1[2].percentage = 81.5;

    for(i=0; i<3; i++)
    {
       printf("     S1s of STUDENT : %d \n", i+1);
       printf(" Id is: %d \n", s1[i].id);
       printf(" Name is: %s \n", s1[i].name);
       printf(" Percentage is: %f\n\n",s1[i].percentage);
    }
}
```

Nested Structure in C

When a structure contains another structure, it is called nested structure. It is also called as structure within a structure.

```
struct structure1
        {
            - - - - - - - - - -
            - - - - - - - - - -
        };

        struct structure2
        {
            - - - - - - - - - -
            - - - - - - - - - -
            struct structure1 obj;
        };
```

e.g.

```
struct Address
    {
        char HouseNo[25];
        char City[25];
        char PinCode[25];
    };

    struct Employee
    {
      int Id;
      char Name[25];
      float Salary;
      struct Address Add;
    };
```

Union

A union is a special data type available in C that allows to store different data types in the same memory location.

You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple-purpose.

Syntax:
```
union    name
{
data type      var_name1;
data type      var_name2;
data type      var_name3;
};
```

Example:
```
union     student
{
int  mark;
char name[10];
float average;
};
```

```
union Data

 {

   int i;

   float f;

   char str[20];

}
```

Accessing Union Members

To access any member of a union, we use the member access operator (.).

Now, a variable of Data type can store an integer, a floating-point number, or a string of characters. It means a single variable, i.e., same memory location, can be used to store multiple types of data. You can use any built-in or user defined data types inside a union based on your requirement.

The memory occupied by a union will be large enough to hold the largest member of the union. For example, in the above example, Data type will occupy 20 bytes of memory space because this is the maximum space which can be occupied by a character string.

Difference Between structure and union

structure	union
Keyword struct defines a structure.	Keyword union defines a union.

Example structure declaration:
struct s_tag
{
  int ival;
  float fval;
  char *cptr;
}s;
	Example union declaration:
	union u_tag
	{
	  int ival;
	  float fval;
	  char *cptr;
	}u;

Within a structure all members gets memory allocated	For a union compiler allocates the memory for the largest of all members
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
One or more members of a structure can be initialized at once.	A union may only be initialized with a value of the type of its first member

Bit Field in C

Bit field can be used to reduce memory consumption when it is known that only some bits would be used for a variable.

As we know, integer takes two bytes(16-bits) in memory. Some times we need to store value that takes less then 2-bytes. In such cases, there is wastages of memory. For example, if we use a variable ***temp*** to store value either 0 or 1. In this case only one bit of memory will be used rather then 16-bits. By using bit field, we can **save lot of memory**.

Syntax:

```
struct struct-name
        {
            datatype var1 : size of bits;
            datatype var2 : size of bits;
            - - - - - - - - - -
            - - - - - - - - - -
            datatype varN : size of bits;
        };
```

Example:

```
struct info2
    {
        int num : 1;
    };
```

Now it will use only 1 bit of memory instead of 16 bits (2 bytes).

Self Referential structure

When a structure has at least one member that is a pointer to the *structure* of its own kind, is called as self referential structure.

e.g.

```
struct list
{
  int info;
  struct list *link;
};
```

A *self referential structure* is used to create data structures like linked lists, stacks, etc.

typedef

The C programming language provides a keyword called **typedef**, which you can use to give a type, a new name

e.g. **typedef char xxx;**

now we can declare character variable using xxx type.

xxx ch;

here ch is defined as a character variable.

Function Pointer

A *function pointer* is a variable that stores the address of a *function. We can then call the function using that pointer.*

e.g

1) void show()
{
-------- 	
}
	Pointer to fun show
	
	void *p1 ();
		2) void add (int a, int b)
		{
		------ 	
		}
			Pointer to fun add
			
			void *p2 (int,int);

Call function show as

p1();

Call function add as

p2(10,20);