***What is a structure?***  
A structure is a user defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type.

Arrays allow to define type of variables that can hold several data items of the same kind. Similarly **structure** is another user defined data type available in C that allows to combine data items of different kinds.

Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book −

* Title
* Author
* Subject
* Book ID

***How to create a structure?***  
‘struct’ keyword is used to create a structure. Following is an example.

|  |
| --- |
| struct address  {     char name[50];     char street[100];     char city[50];     char state[20];     int pin;  }; |

***How to declare structure variables?***  
A structure variable can either be declared with structure declaration or as a separate declaration like basic types.

// A variable declaration like basic data types

struct Point

{

   int x, y;

};

int main()

{

   struct Point p1;  // The variable p1 is declared like a normal variable

}

***How to initialize structure members?***  
Structure members **cannot be** initialized with declaration. For example the following C program fails in compilation.

struct Point

{

   int x = 0;  // COMPILER ERROR:  cannot initialize members here

   int y = 0;  // COMPILER ERROR:  cannot initialize members here

};

Structure members **can be** initialized using curly braces ‘{}’. For example, following is a valid initialization

struct Point

{

   int x, y;

};

int main()

{

   // A valid initialization. member x gets value 0 and y

   // gets value 1.  The order of declaration is followed.

   struct Point p1 = {0, 1};

}

***How to access structure elements?***  
Structure members are accessed using dot (.) operator.

|  |
| --- |
| #include<stdio.h>    struct Point  {     int x, y;  };    int main()  {     struct Point p1 = {0, 1};       // Accesing members of point p1     p1.x = 20;     printf ("x = %d, y = %d", p1.x, p1.y);       return 0;  } |

**Output:**

x = 20, y = 1

***What is an array of structures?***  
Like other primitive data types, we can create an array of structures.

|  |
| --- |
| #include<stdio.h>    struct Point  {     int x, y;  };    void main()  {     // Create an array of structures     struct Point arr[10];       // Access array members     arr[0].x = 10;     arr[0].y = 20;  //arr[1].x=34;  //arr[1].y=56;       printf("%d %d", arr[0].x, arr[0].y);     //printf("%d %d", arr[1].x, arr[1].y);    } |

**Output:**

10 20

***What is a structure pointer?***  
Like primitive types, we can have pointer to a structure. If we have a pointer to structure, members are accessed using arrow ( -> ) operator.

|  |
| --- |
| #include<stdio.h>    struct Point  {     int x, y;  };    int main()  {     struct Point p1 = {1, 2};       // p2 is a pointer to structure p1     struct Point \*p2 = &p1;       // Accessing structure members using structure pointer     printf("%d %d", p2->x, p2->y);     return 0;  } |

**Output:**

1 2

**Udf with structures**

1. #include <stdio.h>
2. struct student
3. {
4. char name[50];
5. int age;
6. };
7. // function prototype
8. void display(struct student s);
9. int main()
10. {
11. struct student s1;
12. printf("Enter name: ");
13. scanf("%[^\n]%\*c", s1.name);
14. printf("Enter age: ");
15. scanf("%d", &s1.age);
17. display(s1); // passing struct as an argument
19. return 0;
20. }
21. void display(struct student s)
22. {
23. printf("\nDisplaying information\n");
24. printf("Name: %s", s.name);
25. printf("\nAge: %d", s.age);
26. }

**Output**

Enter name: Bond

Enter age: 13

Displaying information

Name: Bond

Age: 13

# Union in C

Like [Structures](https://www.geeksforgeeks.org/structures-c/), union is a user defined data type. In union, all members share the same memory location.

## How to define a union?

We use the union keyword to define unions. Here's an example:

1. union car
2. {
3. char name[50];
4. int price;
5. };

## Create union variables

When a union is defined, it creates a user-defined type. However, no memory is allocated. To allocate memory for a given union type and work with it, we need to create variables.

Here's how we create union variables.

1. union car
2. {
3. char name[50];
4. int price;
5. };
6. int main()
7. {
8. union car car1, car2, \*car3;
9. return 0;
10. }

For example in the following C program, both x and y share the same location. If we change x, we can see the changes being reflected in y.

#include <stdio.h>

// Declaration of union is same as structures

union test {

int x, y;

};

int main()

{

// A union variable t

union test t;

t.x = 2; // t.y also gets value 2

printf("After making x = 2:\n x = %d, y = %d\n\n", t.x, t.y);

t.y = 10; // t.x is also updated to 10

printf("After making y = 10:\n x = %d, y = %d\n\n", t.x, t.y);

return 0;

}

After making x = 2:

x = 2, y = 2

After making y = 10:

x = 10, y = 10

**Structure vs. Union**

|  |  |
| --- | --- |
| **Structure** | **Union** |
| Struct keyword is used to define a structure. | Union keyword is used to define a union. |
| Members do not share memory in a structure. | Members share the memory space in a union. |
| Any member can be retrieved at any time in a structure. | Only one member can be accessed at a time in a union. |
| Several members of a structure can be initialized at once. | Only the first member can be initialized. |
| Size of the structure is equal to the sum of size of the each member. | Size of the union is equal to the size of the largest member. |
| Altering value of one member will not affect the value of another. | Change in value of one member will affect other member values. |
| Stores different values for all the members. | Stores same value for all the members. |