



Structures in C

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.



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 with structure declaration.
struct Point
{
   int x, y;
} p1; // The variable p1 is declared with 'Point'

// 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
}
```

Note: In C++, the struct keyword is optional before in declaration of a variable. In C, it is mandatory.



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
};
```

The reason for above error is simple, when a datatype is declared, no memory is allocated for it. Memory is allocated only when variables are created.



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.

```
 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 designated Initialization?

Designated Initialization allows structure members to be initialized in any order. This feature has been added in [C99 standard](#).



```
struct Point
{
    int x, y, z;
};

int main()
{
    // Examples of initialization using designated initialization
    struct Point p1 = {.y = 0, .z = 1, .x = 2};
    struct Point p2 = {.x = 20};

    printf ("x = %d, y = %d, z = %d\n", p1.x, p1.y, p1.z);
    printf ("x = %d", p2.x);
    return 0;
}
```




Output:

```
x = 2, y = 0, z = 1
x = 20
```

This feature is not available in C++ and works only in C.


What is an array of structures?

Like other primitive data types, we can create an array of structures.



```
struct Point
{
    int x, y;
};
```

```

 int main()
{
    // Create an array of structures
    struct Point arr[10];

    // Access array members
    arr[0].x = 10;
    arr[0].y = 20;

    printf("%d %d", arr[0].x, arr[0].y);
    return 0;
}

```





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.

```

 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
```

Operations on struct variables in C

In C, the only operation that can be applied to *struct* variables is assignment. Any other operation (e.g. equality check) is not allowed on *struct* variables.

For example, program 1 works without any error and program 2 fails in compilation.

Program 1

```
#include <stdio.h>

struct Point {
    int x;
    int y;
};

int main()
{
    struct Point p1 = {10, 20};
    struct Point p2 = p1; // works: contents of p1 are copied to p1
    printf(" p2.x = %d, p2.y = %d", p2.x, p2.y);
    getchar();
    return 0;
}
```

> p2.x = 10, p2.y = 20

Program 2

```
#include <stdio.h>

struct Point {
    int x;
    int y;
};

int main()
{
    struct Point p1 = {10, 20};
    struct Point p2 = p1; // works: contents of p1 are copied to p1
    if (p1 == p2) // compiler error: cannot do equality check for
                  // whole structures
    {
        printf("p1 and p2 are same ");
    }
    getchar();
    return 0;
}
```