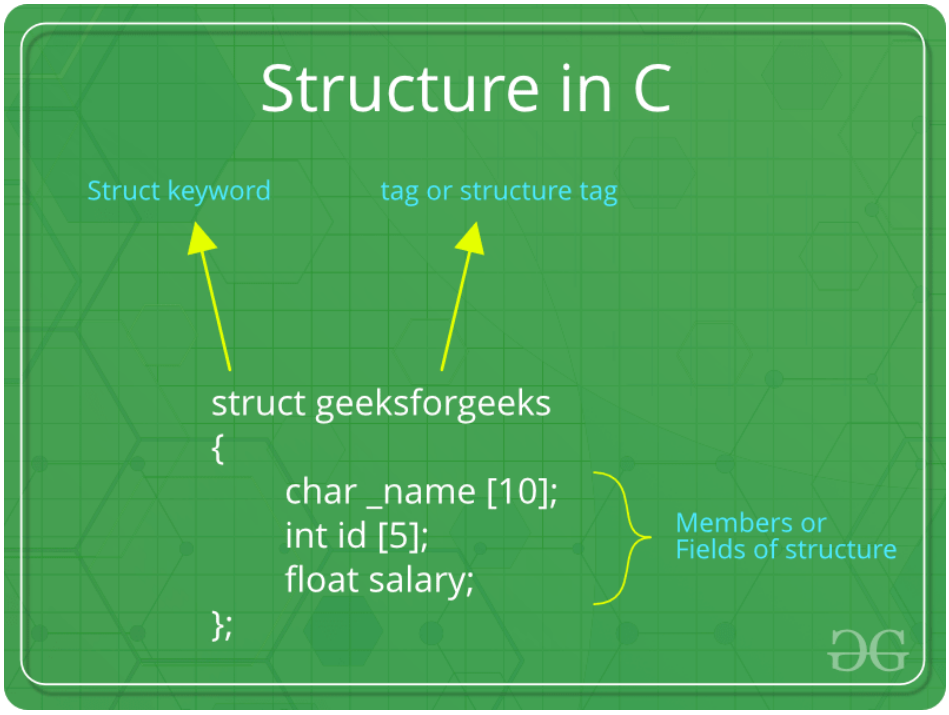
C Structures

- The structure in C is a user-defined data type that can be used to group items of possibly different types into a single type. The **struct keyword** is used to define the structure in the C programming language. The items in the structure are called its **member** and they can be of any valid data type.



C Structure Declaration

We have to declare structure in C before using it in our program. In structure declaration, we specify its member variables along with their datatype. We can use the struct keyword to declare the structure in C using the following syntax:

Syntax

```
struct structure_name {
    data_type member_name1;
    data_type member_name1;
    ....
    ....
};
```

The above syntax is also called a structure template or structure prototype and no memory is allocated to the structure in the declaration.

C Structure Definition

To use structure in our program, we have to define its instance. We can do that by creating variables of the structure type. We can define structure variables using two methods:

1. Structure Variable Declaration with Structure Template

```
struct structure_name {
    data_type member_name1;
    data_type member_name1;
    ....
    ....
}variable1, varaible2, ...;
```

2. Structure Variable Declaration after Structure Template

```
// structure declared beforehand
struct structure_name variable1, variable2, .......;
```

Access Structure Members

We can access structure members by using the **(.) dot operator.**

Syntax

```
structure_name.member1;
strcuture_name.member2;
```

In the case where we have a pointer to the structure, we can also use the arrow operator to access the members.

Initialize Structure Members

Structure members **cannot be** initialized with the declaration. For example, the following C program fails in the compilation.

```
struct Point
{
   int x = 0;  // COMPILER ERROR:  cannot initialize members here
   int y = 0;  // COMPILER ERROR:  cannot initialize members here
};
```

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

We can initialize structure members in 3 ways which are as follows:

1. Using Assignment Operator.

2. Using Initializer List.
3. Using Designated Initializer List.

1. Initialization using Assignment Operator

```
struct structure_name str;
str.member1 = value1;
str.member2 = value2;
str.member3 = value3;
.
.
.
```

2. Initialization using Initializer List

```
struct structure_name str = { value1, value2, value3 };
```
In this type of initialization, the values are assigned in sequential order as they are declared in the structure template.

3. Initialization using Designated Initializer List

Designated Initialization allows structure members to be initialized in any order. This feature has been added in the C99 standard.
```
struct structure_name str = { .member1 = value1, .member2 = value2,
.member3 = value3 };
```
The Designated Initialization is only supported in C but not in C++.

Example of Structure in C

The following C program shows how to use structures

- C

```
// C program to illustrate the use of structures

#include <stdio.h>

  // declaring structure with name str1

struct str1 {

    int i;
```

```
    char c;
    float f;
    char s[30];
};


// declaring structure with name str2
struct str2 {
    int ii;
    char cc;
    float ff;
} var; // variable declaration with structure template
  // Driver code
int main()
{
    // variable declaration after structure template
    // initialization with initializer list and designated
    //    initializer list
    struct str1 var1 = { 1, 'A', 1.00, "GeeksforGeeks" },
                var2;
    struct str2 var3 = { .ff = 5.00, .ii = 5, .cc = 'a' };
```

```
    // copying structure using assignment operator

    var2 = var1;

    printf("Struct 1:\n\ti = %d, c = %c, f = %f, s = %s\n",

           var1.i, var1.c, var1.f, var1.s);

    printf("Struct 2:\n\ti = %d, c = %c, f = %f, s = %s\n",

           var2.i, var2.c, var2.f, var2.s);

    printf("Struct 3\n\ti = %d, c = %c, f = %f\n", var3.ii,

           var3.cc, var3.ff);

    return 0;

}
```

Output

```
Struct 1:
    i = 1, c = A, f = 1.000000, s = GeeksforGeeks
Struct 2:
    i = 1, c = A, f = 1.000000, s = GeeksforGeeks
Struct 3
    i = 5, c = a, f = 5.000000
```

typedef for Structures

The typedef keyword is used to define an alias for the already existing datatype. In structures, we have to use the struct keyword along with the structure name to define the variables. Sometimes, this increases the length and complexity of the code. We can use the typedef to define some new shorter name for the structure.

Example

```
// C Program to illustrate the use of typedef with

// structures

#include <stdio.h>

  // defining structure

struct str1 {

    int a;

};

 // defining new name for str1

typedef struct str1 str1;

  // another way of using typedef with structures

typedef struct str2 {

    int x;

} str2;



int main()

{
```

```
    // creating structure variables using new names

    str1 var1 = { 20 };

    str2 var2 = { 314 };



    printf("var1.a = %d\n", var1.a);

    printf("var2.x = %d", var2.x);



    return 0;

}
```

Output

```
var1.a = 20

var2.x = 314
```

One thing to note here is that the declaration of the structure should always be present before its definition as a structure member. For example, the **declaration below is invalid** as the struct mem is not defined when it is declared inside the parent structure.

```
struct parent {

    struct mem a;

};


struct mem {

    int var;

};
```

Uses of Structure in C

C structures are used for the following:

1. The structure can be used to define the custom data types that can be used to create some complex data types such as dates, time, complex numbers, etc. which are not present in the language.

2. It can also be used in data organization where a large amount of data can be stored in different fields.
3. Structures are used to create data structures such as trees, linked lists, etc.
4. They can also be used for returning multiple values from a function.

Limitations of C Structures

In C language, structures provide a method for packing together data of different types. A Structure is a helpful tool to handle a group of logically related data items. However, C structures also have some limitations.

- **Higher Memory Consumption:** It is due to structure padding.
- **No Data Hiding:** C Structures do not permit data hiding. Structure members can be accessed by any function, anywhere in the scope of the structure.
- **Functions inside Structure:** C structures do not permit functions inside the structure so we cannot provide the associated functions.
- **Static Members:** C Structure cannot have static members inside its body.
- **Construction creation in Structure:** Structures in C cannot have a constructor inside Structures.