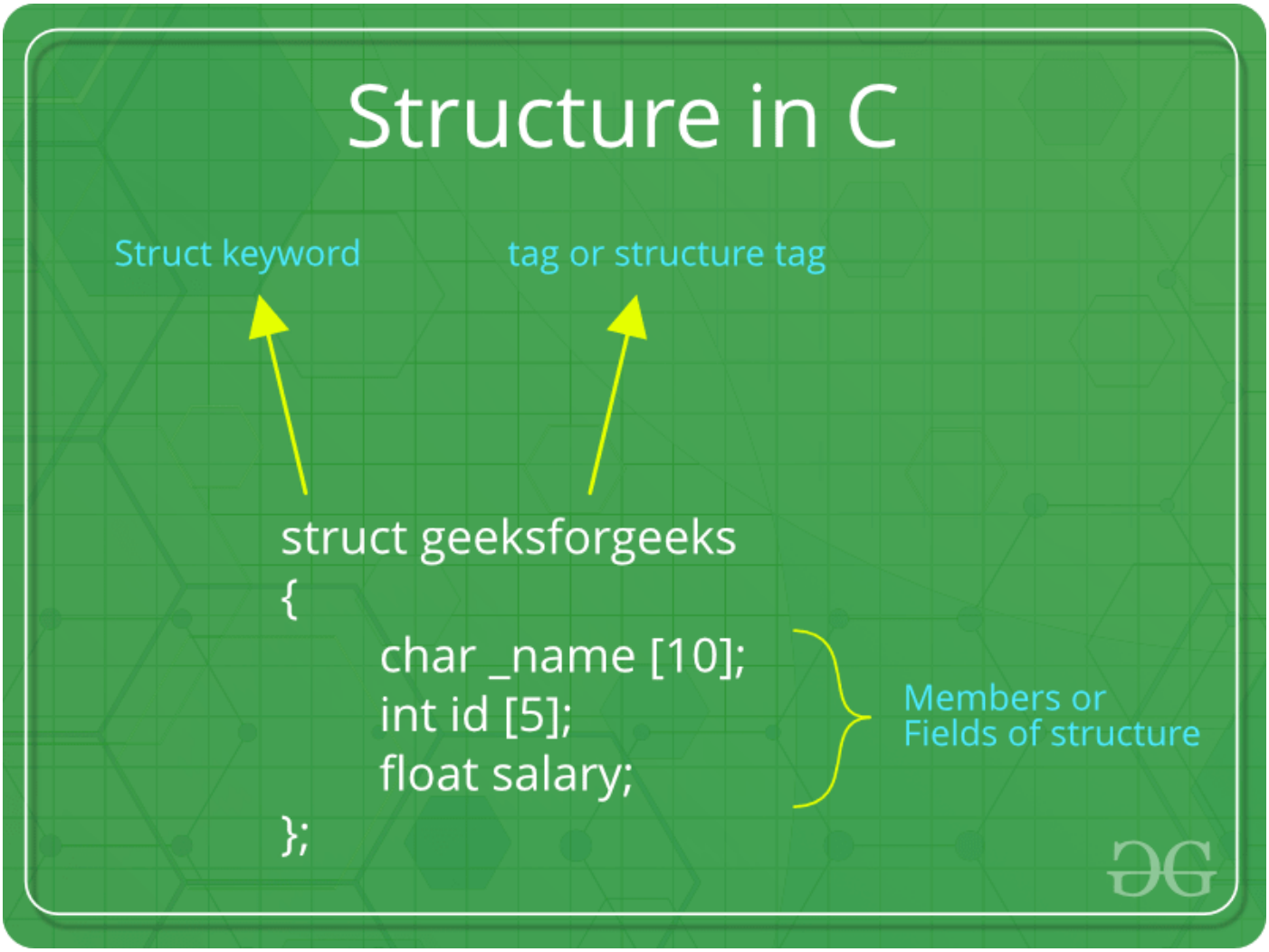
Structures in C

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 define structures?

Before you can create structure variables, you need to define its data type. To define a struct, the `struct` keyword is used.

Syntax of struct

```
struct structureName
{
   dataType member1;
   dataType member2;
   ...
};
```

Here is an example:

```
struct Person
{
   char name[50];
   int citNo;
   float salary;
};
```

Here, a derived type `struct Person` is defined. Now, you can create variables of this type.

Create struct variables

When a struct type is declared, no storage or memory is allocated. To allocate memory of a given structure type and work with it, we need to create variables.
Here's how we create structure variables:

```
struct Person
{
   char name[50];
   int citNo;
   float salary;
};

int main()
{
   struct Person person1, person2, p[20];
   return 0;
}
```

Another way of creating a struct variable is:

```
struct Person
{
   char name[50];
   int citNo;
   float salary;
} person1, person2, p[20];
```

In both cases, two variables `person1`, `person2`, and an array variable `p` having 20 elements of type `struct Person` are created.

Access members of a structure

There are two types of operators used for accessing members of a structure.

1. . - Member operator
2. -> - Structure pointer operator (will be discussed in the next tutorial)

Suppose, you want to access the `salary` of `person2`. Here's how you can do it.

```
person2.salary
```

Example: Add two distances

```c
// Program to add two distances (feet-inch)
#include <stdio.h>
struct Distance
{
   int feet;
   float inch;
} dist1, dist2, sum;

int main()
{
   printf("1st distance\n");
   printf("Enter feet: ");
   scanf("%d", &dist1.feet);

   printf("Enter inch: ");
   scanf("%f", &dist1.inch);
   printf("2nd distance\n");

   printf("Enter feet: ");
   scanf("%d", &dist2.feet);

   printf("Enter inch: ");
   scanf("%f", &dist2.inch);

   // adding feet
   sum.feet = dist1.feet + dist2.feet;
   // adding inches
   sum.inch = dist1.inch + dist2.inch;

   // changing to feet if inch is greater than 12
   while (sum.inch >= 12)
   {
      ++sum.feet;
      sum.inch = sum.inch - 12;
   }

   printf("Sum of distances = %d\'-%.1f\"", sum.feet, sum.inch);
   return 0;
}
```

Output

```
1st distance
Enter feet: 12
Enter inch: 7.9
2nd distance
Enter feet: 2
Enter inch: 9.8
```

```
Sum of distances = 15'-5.7"
```

Keyword typedef

We use the `typedef` keyword to create an alias name for data types. It is commonly used with structures to simplify the syntax of declaring variables.

This code

```
struct Distance{
   int feet;
   float inch;
};

int main() {
   structure Distance d1, d2;
}
```

is equivalent to

```
typedef struct Distance{
   int feet;
   float inch;
} distances;

int main() {
   distances d1, d2;
}
```

Nested Structures

You can create structures within a structure in C programming. For example,

```
struct complex
{
 int imag;
 float real;
};

struct number
{
  struct complex comp;
  int integers;
} num1, num2;
```

Suppose, you want to set `imag` of `num2` variable to 11. Here's how you can do it:

```
num2.comp.imag = 11;
```

Why structs in C?

Suppose, you want to store information about a person: his/her name, citizenship number, and salary. You can create different variables `name`, `citNo` and `salary` to store this information.

What if you need to store information of more than one person? Now, you need to create different variables for each information per person: `name1`, `citNo1`, `salary1`, `name2`, `citNo2`, `salary2`, etc.
A better approach would be to have a collection of all related information under a single name `Person` structure and use it for every person.

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 variable
}
```

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.

```
#include<stdio.h>
```

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

int main()
{
  struct Point p1 = {0, 1};

  // Accessing 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.

```
#include<stdio.h>

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.

```
#include<stdio.h>

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.

```
#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
```

What is structure member alignment?

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 have some limitations.

- The C structure does not allow the struct data type to be treated like built-in data types:
- We cannot use operators like +,- etc. on Structure variables. For example, consider the following code:

```
struct number
{
   float x;
};
int main()
{
   struct number n1,n2,n3;
   n1.x=4;
   n2.x=3;
   n3=n1+n2;
   return 0;
}

/*Output:

prog.c: In function 'main':
prog.c:10:7: error:
invalid operands to binary + (have 'struct number' and 'struct number')
  n3=n1+n2;
```

*/

- **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 Structure
- **Static Members:** C Structures cannot have static members inside their body
- **Access Modifiers:** C Programming language do not support access modifiers. So they cannot be used in C Structures.
- **Construction creation in Structure:** Structures in C cannot have constructor inside Structures.