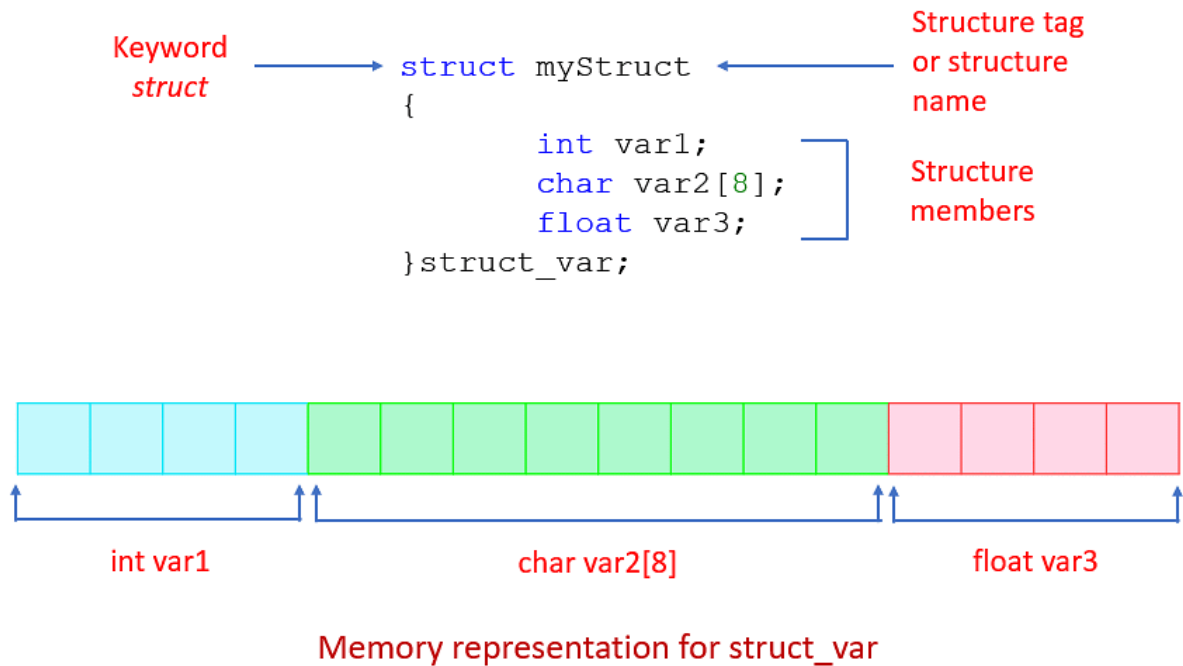


# Structure:



**Def:** Structures (also called structs) are a way to group several related variables into one place. Each variable in the structure is known as a member of the structure.

**Def:** A structure creates a data type that can be used to group items of possibly different types into a single type.

a structure is a collection of variables (can be of different types) under a single name.

## How to Define a Structure?

->We can define the structure using the “struct” key word.

Syntax:

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

How to create a structure ?

->we can create a structure by using the struct keyword and declare each of its members inside curly braces.

->we need to create the variables inside the struct key .

Ex: struct address

```
{  
    char name[50];  
    char street[100];  
    char city[50];  
    char state[20];  
    int pin;  
};
```

->Another way of declaring variables in structure are

```
1. struct Person {  
    // code or variables;  
};
```

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

```
2. struct Person {  
    // code  
} person1, person2, p[20];
```

->Here person1 and person2 are struct Person variables.

->p[ ] is a struct person array of size 20.

How to access the structure members ?

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

1. (".") - Member operator.
2. ("->") - Structure pointer operator.

```
Ex: 1. struct myStructure {
    int myNum;
    char myLetter;
};

int main() {
    struct myStructure s1; // Create a structure variable of myStructure called s1. ( "//" is
comments).

    s1.myNum = 13; // Assign values to members of s1.
    s1.myLetter = 'B';

    printf("My number: %d\n", s1.myNum); // Print values
    printf("My letter: %c\n", s1.myLetter);

    return 0;
}
```

Output : My number: 13

My letter: B

2. struct Person { // create struct with person1 variable

```
    char name[50];
```

```

        int citNo;

        float salary;
    } person1;

int main() {

    strcpy(person1.name, "George Orwell"); // assign value to name of person1

    person1.citNo = 1984; // assign values to other person1 variables
    person1.salary = 2500;

    printf("Name: %s\n", person1.name); // print struct variables
    printf("Citizenship No.: %d\n", person1.citNo);
    printf("Salary: %.2f", person1.salary);

    return 0;
}

```

Output :

```

Name: George Orwell
Citizenship No.: 1984
Salary: 2500.00.

```

What is an array of structures ?

Ex :

```

#include<stdio.h>

struct Point
{
    int x, y;
}

```

```
};
```

```
int main()
```

```
{
```

```
    struct Point arr[10]; // Create an array of structures
```

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

```
    arr[0].y = 20;
```

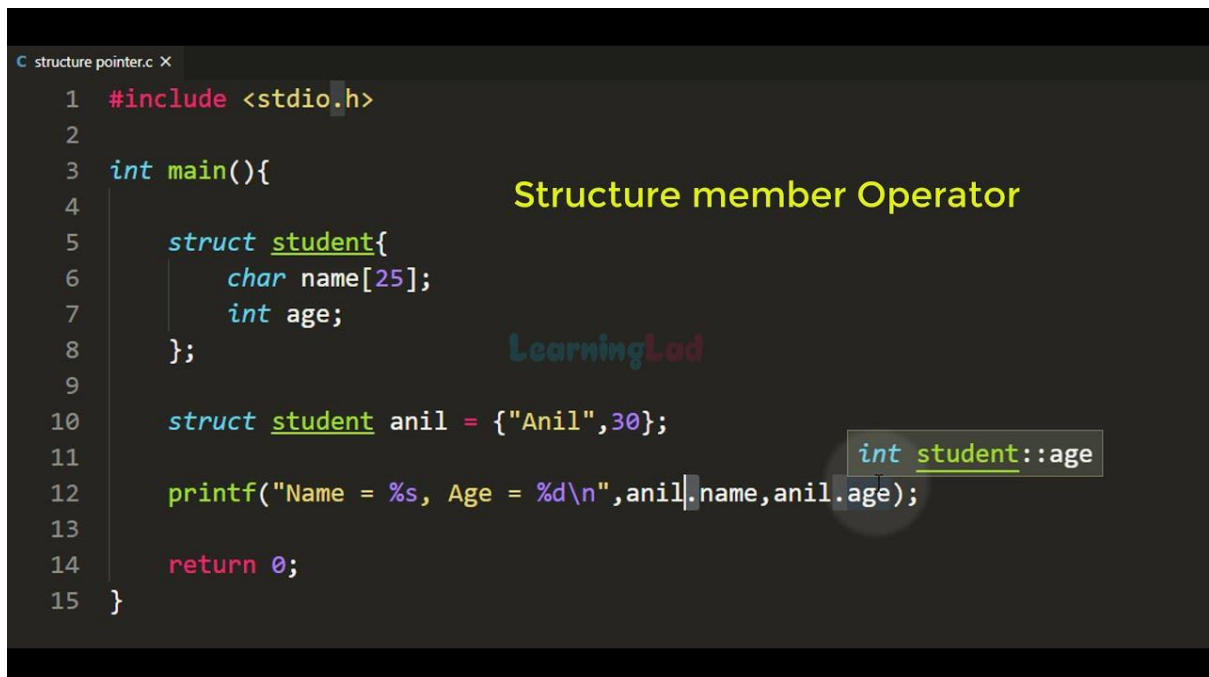
```
    printf("%d %d", arr[0].x, arr[0].y);
```

```
    return 0;
```

```
}
```

Output: 10 20.

## What is a structure pointer ?



```
C structure pointer.c X
1  #include <stdio.h>
2
3  int main(){
4
5      struct student{
6          char name[25];
7          int age;
8      };
9
10     struct student anil = {"Anil",30};
11
12     printf("Name = %s, Age = %d\n",anil.name,anil.age);
13
14     return 0;
15 }
```

Structure member Operator

int student::age

-> If we have a pointer to structure, members are accessed using arrow ( -> ) operator.

Ex : struct person

```
{  
    int age;  
    float weight;  
};
```

```
int main()
```

```
{  
    struct person *personPtr, person1;  
    personPtr = &person1;  
  
    printf("Enter age: ");  
    scanf("%d", &personPtr->age);  
  
    printf("Enter weight: ");  
    scanf("%f", &personPtr->weight);  
  
    printf("Displaying:\n");  
    printf("Age: %d\n", personPtr->age);  
    printf("weight: %f", personPtr->weight);  
  
    return 0;  
}
```

# Pointer to Structure Variable

```

struct abc {
    int x;
    int y;
};

int main() {
    struct abc a = {0, 1};
    struct abc *ptr = &a;

    printf("%d %d", ptr->x, ptr->y);
    return 0;
}

```

ptr is a pointer to some variable of type **struct abc**.

**157 C Programming**

-> In this example, the address of person1 is stored in the personPtr pointer using personPtr = &person1;.

Now, you can access the members of person1 using the personPtr pointer.

-> By the way,

personPtr->age is equivalent to (\*personPtr).age.

personPtr->weight is equivalent to (\*personPtr).weigh.

## How to passing the structs to functions ?

```

struct student {
    char id[15];
    char firstname[64];
    char lastname[64];
    float points;
};

struct student std[3];
displayDetail(std);

void displayDetail(struct student *ptr) {
    // some code goes here...
}

```

std[0].id std[0].firstname std[0].lastname std[0].points



1000...1014 1015 ... 1078 1079 ... 1142 1143 ... 1146

ptr  
↑

std[1].id std[1].firstname std[1].lastname std[1].points



1147...1161 1162 ... 1225 1226 ... 1289 1290 ... 1293

std[2].id std[2].firstname std[2].lastname std[2].points



1294...1308 1309 ... 1372 1373 ... 1436 1437 ... 1440

8000

dyclassroom.com

-> #include <stdio.h>

```
struct student {  
    char name[50];  
    int age;  
};  
  
// function prototype  
void display(struct student s);  
  
int main() {  
    struct student s1;  
  
    printf("Enter name: ");  
  
    // read string input from the user until \n is entered  
    // \n is discarded  
    scanf("%[^\\n]*c", s1.name);  
  
    printf("Enter age: ");  
    scanf("%d", &s1.age);  
  
    display(s1); // passing struct as an argument  
  
    return 0;  
}  
  
void display(struct student s) {  
    printf("\\nDisplaying information\\n");  
    printf("Name: %s", s.name);  
    printf("\\nAge: %d", s.age);  
}
```



Output : Enter name: Bond

Enter age: 13

Displaying information

Name: Bond

Age: 13 .

->Here, a struct variable s1 of type struct student is created. The variable is passed to the display() function using display(s1); statement.

Return struct from a function :

Ex : #include <stdio.h>

```
struct student
```

```
{
```

```
    char name[50];
```

```
    int age;
```

```
};
```

```
struct student getInformation(); // function prototype.
```

```
int main()
```

```
{
```

```
    struct student s;
```

```
    s = getInformation();
```

```
    printf("\nDisplaying information\n");
```

```
    printf("Name: %s", s.name);
```

```
    printf("\nRoll: %d", s.age);
```

```
    return 0;
```

```

}

struct student getInformation()
{
    struct student s1;

    printf("Enter name: ");
    scanf ("%[^\\n]*c", s1.name);

    printf("Enter age: ");
    scanf ("%d", &s1.age);

    return s1;
}

```

-> Here, the getInformation() function is called using s = getInformation(); statement. The function returns a structure of type struct student. The returned structure is displayed from the main() function.

Notice that, the return type of getInformation() is also struct student.

## Passing struct by reference:

->You can also pass structs by reference (in a similar way like you pass variables of built-in type by reference). We suggest you to read pass by reference tutorial before you proceed.

During pass by reference, the memory addresses of struct variables are passed to the function.

```

typedef struct Complex
{
    float real;
    float imag;
} complex;

```

```
void addNumbers(complex c1, complex c2, complex *result);
```

```
int main()
```

```
{
```

```
    complex c1, c2, result;
```

```
    printf("For first number,\n");
```

```
    printf("Enter real part: ");
```

```
    scanf("%f", &c1.real);
```

```
    printf("Enter imaginary part: ");
```

```
    scanf("%f", &c1.imag);
```

```
    printf("For second number, \n");
```

```
    printf("Enter real part: ");
```

```
    scanf("%f", &c2.real);
```

```
    printf("Enter imaginary part: ");
```

```
    scanf("%f", &c2.imag);
```

```
    addNumbers(c1, c2, &result);
```

```
    printf("\nresult.real = %.1f\n", result.real);
```

```
    printf("result.imag = %.1f", result.imag);
```

```
    return 0;
```

```
}
```

```
void addNumbers(complex c1, complex c2, complex *result)
```

```
{
```

```
    result->real = c1.real + c2.real;
```

```
    result->imag = c1.imag + c2.imag;
```

```
}
```

Run Code

Output :

For first number,

Enter real part: 1.1

Enter imaginary part: -2.4

For second number,

Enter real part: 3.4

Enter imaginary part: -3.2

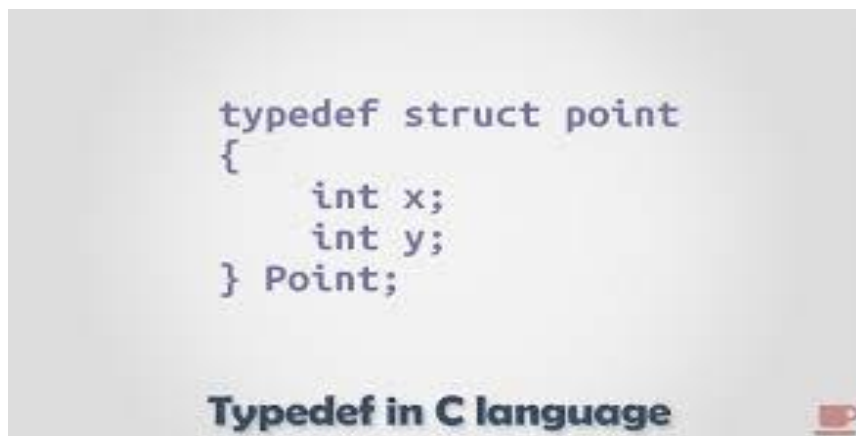
result.real = 4.5

result.imag = -5.6

-> In the above program, three structure variables c1, c2 and the address of result is passed to the addNumbers() function. Here, result is passed by reference.

When the result variable inside the addNumbers() is altered, the result variable inside the main() function is also altered accordingly.

## 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.

->1. typedef struct Distance {

int feet;

float inch;

```
} distances;
```

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

```
2.typedef struct Person {
```

```
    char name[50];
```

```
    int citNo;
```

```
    float salary;
```

```
} person;
```

```
int main() {
```

```
    // create Person variable
```

```
    person p1;
```

```
    // assign value to name of p1
```

```
    strcpy(p1.name, "George Orwell");
```

```
    // assign values to other p1 variables
```

```
    p1.citNo = 1984;
```

```
    p1.salary = 2500;
```

```
    // print struct variables
```

```
    printf("Name: %s\n", p1.name);
```

```
    printf("Citizenship No.: %d\n", p1.citNo);
```

```
    printf("Salary: %.2f", p1.salary);
```

```
    return 0;
```

```
}
```

Output : Name: George Orwell

Citizenship No.: 1984

Salary: 2500.00.

## 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, citiNo 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.

## 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

## Limitations of Structures in C

- The standard C does not allow the struct data type to be treated like built-in types.
- They do not permit data hiding.
- Structure members can be directly accessed by the structure variables by any function anywhere in their scope.