Structures as function arguments in C

In this tutorial, you will master to pass the structure as function arguments both by call by value and call by reference method. Also, you will learn to return structures to a function with the aid of simple and easy examples.

We assume that you have the basic knowledge about both structures and functions in C before starting this tutorial. If not you can refer to our Functions in C and Structures in C tutorial for a better understanding of this topic.

We can use structures as function arguments just like the general variables do. Structures support both call by value and call by reference procedures. So basically C provides 3 different ways to its users to pass structures to a function.

- The first way is to pass the structure members to the function.
- Secondly, we can pass the entire structure to the function(call by value).
- Thirdly we can pass the structure as a pointer(call by reference)

How to pass structure members as function arguments

Let's begin by trying to understand the below given simple program.

```
// How to pass structure members to a function
#include<stdio.h>
struct employee
{
    char name[20];
    int id_no;
    int salary;
};
// function declaration
void display_structure(char name[], int id_no, int salary);

int main()
{
    struct employee emp = {"Tom", 1001, 70000};
    display_structure(emp.name, emp.id_no, emp.salary);// function call

    return 0;
}
// function definition
void display_structure(char name[], int id_no, int salary)
{
    printf("Name of Employee: %s\n", name);
    printf("Id No of Employee: %d\n", id_no);
    printf("Salary of Employee: %d\n\n", salary);
    printf("\n");
}
```

In the above code snippet, we have created a structure named `employee` with 3 members namely `name`, `id_no`, and `salary`. Since the structure is always declared outside the main function, the structure has a global scope that means the life of the structure exists throughout the program.

Just below the structure, we have given the function declaration which gives information to the compiler that it can expect a function in the program. Here the function declaration is not returning any value as its type here given is `void`. The name of the function is `display_structure` which actually prints the values of structure members that are passed as arguments.

Inside the main function the structure variable `emp` is declared and initialised thereby allocates some space in the memory. Later the function is called where the actual parameters are passed. Now the formal parameters take the value of actual parameters.

In the end, we have defined our function that prints the structure. The Output will be :

```
Name of Employee: Tom
Id No of Employee: 1001
Salary of Employee: 70000
```

The downside of this way of passing structure members to a function is that it is helpful in cases where the number of members in the structure is limited.

How to pass structure to a function by value

Another way that c supports is passing the entire structure to the function by passing the structure variable as its argument. This method is also known as call by value method because any changes made to the structure member variables will not reflect in the original structure.

You can witness this by examining below program.

```
// Passing entire structure to function as call by value
#include<stdio.h>
#include<string.h>

struct employee
{
    char name[20];
    int salary;
};

void update(struct employee E);// formal argument
{
    E.salary = 80000;
    printf("Salary of %s updated to %d\n",E.name,E.salary);
}

int main()
{
    struct employee emp;
    strcpy(emp.name,"Tom");
    emp.salary = 70000;
    printf("Salary of %s in main function is given as %d\n",emp.name,emp.salary);

    update(emp);// actual argument
    printf("Salary of %s after function call is %d\n",emp.name,emp.salary);

    return 0;
}
```

Let's scrutinize the program to understand the concept of call by value clearly. We all know that all programs start with the main() function and so at first we have created a structure variable `emp` using the below statement.

```
struct employee emp;
```

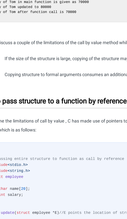
Whenever a structure variable is created, in memory some space will be allocated according to the size of structure members. In our case `emp` will allocate 14 bytes in the memory. The memory allocation of `emp` will be like this :

With the help of `printf` function, the values of structure members are displayed.

In the next line, we have called the user-defined function as follows:

```
update(emp);
```

Here, we are passing the entire structure `emp` (actual argument)to the function definition on functional call. Now the control shifts to the function definition `update()` where the formal argument `E` takes the value of actual arguments, i.e the entire structure itself. Below shows the memory allocation.



When comparing the two images you can see the difference in their memory location. This states the fact that both structures are residing in different locations and are entirely independent of each other.

```
E.salary = 80000;
```

So here in `update()` function, `E.salary` is assigned with a new salary, `80000`, and is printed inside that function. When the function is over the control moves back to the main function where we again print the value of salary, which gives you the output `70000`

From this, we can conclude that any changes made on formal arguments do not reflect back on actual arguments in the case of the call by value method.

The output of the above program will be something like this:

```
Salary of Tom in main function is given as 70000
Salary of Tom updated to 80000
Salary of Tom after function call is 70000
```

Now let's discuss a couple of the limitations of the call by value method while passing structure to a function.

1. If the size of the structure is large, copying of the structure may take some time thereby affecting the efficiency of the program.
2. Copying structure to formal arguments consumes an additional storage space in memory.

How to pass structure to a function by reference

To overcome the limitations of call by value , C has made use of pointers to pass the structures. The above program is slightly modified to achieve the method of call by reference which is as follows.

```
// Passing entire structure to function as call by reference
#include<stdio.h>
#include<string.h>
struct employee
{
    char name[20];
    int salary;
};

void update(struct employee *E);// E points the location of structure
{
    E->salary = 80000;
    printf("Salary of %s updated to %d\n",E->name,E->salary);
}

int main()
{
    struct employee emp;
    strcpy(emp.name,"Tom");
    emp.salary = 70000;
    printf("Salary of %s in main function is given as %d\n",emp.name,emp.salary);

    update(&emp);// passing the address of structure emp
    printf("Salary of %s after function call is %d\n",emp.name,emp.salary);

    return 0;
}
```

In this code fragment, on function call instead of passing the structure here, we have passed the address of the structure `emp`. The formal argument E will take the address and point to the memory location of the structure `emp` itself.

Now both structures `E` and `emp` are residing in the same location and hence any change made on formal argument will flawlessly reflect on the actual argument.

The following depiction clarifies the concept:

Another change we made in our program is instead of using the dot operator to access member variables we used the arrow operator (->) to get the values held by the structure variables.

The output of the program will be :

```
Salary of Tom in main function is given as 70000
Salary of Tom updated to 80000
Salary of Tom after function call is 80000
```

Unlike call by value, here the salary modified in update() function is reflected inside the main function also this is because the value modification actually takes place in the original structure itself.

How to pass an array of structures as function arguments

It is also possible to pass an array of structures to a function, just like passing an array of integers.
The following example illustrates the way of passing an array of structures to a function.

```
#include<stdio.h>
#include<string.h>
struct employee
{
    char name[20];
    int id_no;
    int salary;
};


int main()
{
    struct employee emp[3]={
        {"Tom",1001,70000},
        {"JERRY",1002,75000},
        {"GILL",1003,80000},
    };
    display_structure(emp); //function call

    return 0;
}
// function definition
void display_structure(struct employee E[])
{
    for(int i=0;i<3;i++){
    printf("Employee %d\n",i+1);
    printf("Name: %s\n",E[i].name);
    printf("Id_no: %d\n",E[i].id_no);
    printf("Salary: %d\n",E[i].salary);
    printf("\n");
    }

}
```

Output:

```
Employee 1
Name: Tom
Id_no: 10001
Salary: 70000

Employee 2
Name: JERRY
Id_no: 10002
Salary: 75000

Employee 3
Name: GILL
Id_no: 10003
Salary: 80000
```

Passing an array of structures to a function is not at all a difficult task. We need to properly use the array prototype in the function and use a for loop to traverse through each element in the array.

How to return structures into function

So far we have seen how to pass structures to a function, now it's time to learn the returning of structures to a function.

Here is given an example to calculate the sum of two complex numbers. We know that a complex number contains a real part and an imaginary part. The addition of two complex numbers means the addition of two real parts and the addition of two imaginary parts as shown in the below program.

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

struct complex
{
    float real;
    float imag;
};

struct complex add(struct complex, struct complex);
int main()
{

    struct complex c1,c2,c3;
    printf("Enter the first complex number:");
    scanf("%f%f",&c1.real,&c1.imag);
    printf("Enter the second complex number:");
    scanf("%f%f",&c2.real,&c2.imag);
    c3= add(c1,c2);
    if(c3.imag<0)
      printf("The sum of two complex numbers is %.1f - %.1fi",c3.real,-c3.imag);
    else
      printf("The sum of two complex numbers is %.1f + %.1fi",c3.real,c3.imag);
    return 0;
}

struct complex add(struct complex x,struct complex y)
{
    struct complex z;
    z.real=x.real+y.real;
    z.imag=x.imag+y.imag;
    return z;
}
```

In the above code snippet, the structure is complex with two members real and image of float data type.

```
struct complex add(struct complex, struct complex);
```

This statement is the function declaration where the function name is `add` and it returns a structure data type and is also passing two structures.