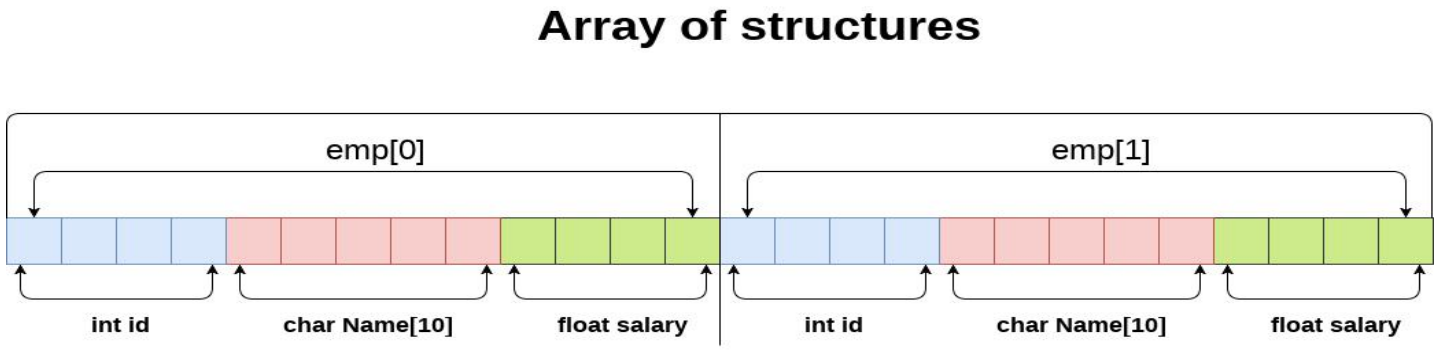
Chapter 7 :- Structure &Unions

Nested Structures

A structure can be nested inside another structure. In other words, the members of a structure can be of any other type including structure. Here is the syntax to create nested structures.




```
struct employee
{
    int id;
    char name[5];
    float salary;
};
struct employee emp[2];
```

sizeof (emp) = 4 + 5 + 4 = 13 bytes

sizeof (emp[2]) = 26 bytes

Syntax:

```
structure tagname_1
{
    member1;
    member2;
    member3;
    ...
    membern;
    structure tagname_2
    {
        member_1;
```

```
 member_2;
        member_3;
        ...
        member_n;
    }, var1

} var2;
```

********** END OF SYNTAX **********

To access the members of the inner structure, we write a variable name of the outer structure, followed by a dot(.) operator, followed by the variable of the inner structure, followed by a dot(.) operator, which is then followed by the name of the member we want to access.

var2.var1.member_1 - refers to the member_1 of structure tagname_2

var2.var1.member_2 - refers to the member_2 of structure tagname_2

and so on.

Array Of Structures

An array of structres in C can be defined as the collection of multiple structures variables where each variable contains information about different entities. The array of structures in C are used to store information about multiple entities of different data types.

The array of structures is also known as the collection of structures.

Ex. :-

```
#include<stdio.h>
#include <string.h>
struct student{
int rollno;
```

```
char name[10];
};
int main(){
int i;
struct student st[5];
printf("Enter Records of 5 students");
for(i=0;i<5;i++){
printf("\nEnter Rollno:");
scanf("%d",&st[i].rollno);
printf("\nEnter Name:");
scanf("%s",&st[i].name);
```

```
}
printf("\nStudent Information List:");
for(i=0;i<5;i++){
printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
}
return 0;
}
```

Output :

```
Enter Records of 5 students
Enter Rollno:1
Enter Name:Sonoo
```

Enter Rollno:2

Enter Name:Ratan

Enter Rollno:3

Enter Name:Vimal

Enter Rollno:4

Enter Name:James

Enter Rollno:5

Enter Name:Sarfraz

Student Information List:

Rollno:1, Name:Sonoo

Rollno:2, Name:Ratan

Rollno:3, Name:Vimal

Rollno:4, Name:James

Rollno:5, Name:Sarfraz

Structure & Functions

we can pass structures as arguments to a function. In fact, we can pass, individual members, structure variables, a pointer to structures etc to the function. Similarly, functions can return either an individual member or structures variable or pointer to the structure.

The following program demonstrates how to pass structure members as arguments to the function -

```
#include<stdio.h>
/*
structure is defined above all functions so it is global.
*/
```

```
struct student
{
    char name[20];
    int roll_no;
    int marks;
};
void print_struct(char name[], int roll_no, int marks);
int main()
{
    struct student stu = {"Tim", 1, 78};
    print_struct(stu.name, stu.roll_no, stu.marks);
```

```
return 0;
}
void print_struct(char name[], int roll_no, int marks)
{
    printf("Name: %s\n", name);
    printf("Roll no: %d\n", roll_no);
    printf("Marks: %d\n", marks);
    printf("\n");
}
```

Expected Output :

Name: icomputercoding

Roll no: 1

Marks: 87

********** **END OF PROGRAM** **********

Passing Structure Variable as Argument to a Function

In the earlier section, we have learned how to pass structure members as arguments to a function. If a structure contains two-three members then we can easily pass them to function but what if there are 9-10 or more

members? Certainly passing 9-10 members is a tiresome and error-prone process. So in such cases instead of passing members individually, we can pass structure variable itself.

The following program demonstrates how we can pass structure variable as an argument to the function :-

```
#include<stdio.h>
/*
structure is defined above all functions so it is global.
*/
```

```
struct student
{
    char name[20];
    int roll_no;
    int marks;
};
void print_struct(struct student stu);
int main()
{
    struct student stu = {"George", 10, 69};
    print_struct(stu);
```

```
 return 0;
}
void print_struct(struct student stu)
{
    printf("Name: %s\n", stu.name);
    printf("Roll no: %d\n", stu.roll_no);
    printf("Marks: %d\n", stu.marks);
    printf("\n");
}
```

Expected Output :

Name: icomputercoding

Roll no: 10

Marks: 66

********** **END OF PROGRAM** **********

Passing Structure Pointers as Argument to a Function

Although passing structure variable as an argument allows us to pass all the members of the structure to a function there are some downsides to this operation. Recall that a copy of the structure is passed to the formal argument. If the structure is large and you are passing structure variables frequently then it can take quite a bit of time which make the program inefficient. Additional memory is required to save every copy of the structure.

The following program demonstrates how to pass structure pointers as arguments to a function :-

```
#include<stdio.h>
/*
structure is defined above all functions so it is global.
*/
struct employee
{
    char name[20];
    int age;
    char doj[10]; // date of joining
    char designation[20];
};
```

```
void print_struct(struct employee *);
int main()
{
    struct employee dev = {"Jane", 25, "25/2/2015", "Developer"};
    print_struct(&dev);
    return 0;
}
void print_struct(struct employee *ptr)
{
    printf("Name: %s\n", ptr->name);
    printf("Age: %d\n", ptr->age);
```

```
printf("Date of joining : %s\n", ptr->doj);
    printf("Designation : %s\n", ptr->designation);
    printf("\n");
}
```

Expected Output :

```
Name: Jin
Age: 25
Date of joining : 25/2/2015
Designation : Developer
```

********** **END OF PROGRAM** **********

Array of Structures as Function Arguments

We have already seen how to pass an array of integers to a function. Similarly, we can pass an array of structures to a function.

The following program demonstrates how we can pass an array of structures to a function :-

```
#include<stdio.h>
/*
structure is defined above all functions so it is global.
*/
struct company
```

```
{
    char name[20];
    char ceo[20];
    float revenue; // in $
    float pps; // price per stock in $
};
void print_struct(const struct company str_arr[]);
int main()
{
   struct company companies[3]= { {"Country Books", "Tim Green",
999999999, 1300 },{"Country Cooks", "Jim Green", 9999999,
700 },{"Country Hooks", "Sim Green", 99999, 300 },};
}
```

```
 print_struct(companies);
return 0;
}
void print_struct(struct company str_arr[])
{
    int i;
    for(i= 0; i<3; i++)
    {
        printf("Name: %s\n", str_arr[i].name);
        printf("CEO: %d\n", str_arr[i].ceo);
        printf("Revenue: %.2f\n", str_arr[i].revenue);
```

```
printf("Price per stock : %.2f\n", str_arr[i].pps);
        printf("\n");
    }
```

Expected Output :

```
Name: Country Books
CEO: 2686660
Revenue: 1000000000.00
Price per stock : 1300.00
```

Name: Country Cooks

CEO: 2686708

Revenue: 9999999.00

Price per stock : 700.00

Name: Country Hooks

CEO: 2686756

Revenue: 99999.00

Price per stock : 300.00

********** **END OF PROGRAM** **********

Difference Between Structure & Union

What is a structure (struct)?

Structure (struct) is a user-defined data type in a programming language that stores different data types' values together. The struct keyword is used to define a structure data type in a program. The struct data type stores one or more than one data element of different kinds in a variable.

1.Id

2.Name

3.Department

4.Email Address

One way to store 4 different data by creating 4 different arrays for each parameter, such as id[], name[], department[], and email[]. Using array id[i] represents the id of the ith employee.

Similarly, name[i] represents the name of ith employee (name). Array element department[i] and email[i] represent the ith employee's department and email address

Ex. :- struct employee

```
{
    int id;
    char name[50];
    string department;
    string email;
};
```

What is a Union?

In "c programming”, union is a user-defined data type that is used to store the different data type's values. However, in the union, one member will occupy the memory at once. In other words, we can say that the size of the union is equal to the size of its largest data member size. Union offers an effective way to use the same memory location several times by each data member. The union keyword is used to define and create a union data type.

Ex.:- union employee

```
{
    string name;
    string department;
    int phone;
    string email;
};
```

Example of Structure and Union showing the difference in C

```
// A simple C program showing differences between Structure and Union
#include <stdio.h>
#include <string.h>
// declaring structure
struct struct_example
{
    int integer;
    float decimal;
    char name[20];
};
```

```
// declaraing union
union union_example
{
    int integer;
    float decimal;
    char name[20];
};
void main()
{
    // creating variable for structure and initializing values difference six
    struct struct_example stru ={5, 15, "John"};
```

```
// creating variable for union and initializing values
    union union_example uni = {5, 15, "John"};
    printf("data of structure:\n integer: %d\n decimal: %.2f\n name: %s\n
", stru.integer, stru.decimal, stru.name);
    printf("\ndata of union:\n integer: %d\n" "decimal: %.2f\n name: %s\
n", uni.integer, uni.decimal, uni.name);


    // difference five
    printf("\nAccessing all members at a time:");
    stru.integer = 163;
    stru.decimal = 75;
```

```
strcpy(stru.name, "John");
    printf("\ndata of structure:\n integer: %d\n " "decimal: %f\n name: %
s\n", stru.integer, stru.decimal, stru.name);
    uni.integer = 163;
    uni.decimal = 75;
    strcpy(uni.name, "John");
    printf("\ndata of union:\n integeer: %d\n " "decimal: %f\n name: %s\
n", uni.integer, uni.decimal, uni.name);

    printf("\nAccessing one member at a time:");
    printf("\ndata of structure:");
```

```
stru.integer = 140;
stru.decimal = 150;
strcpy(stru.name, "Mike");

printf("\ninteger: %d", stru.integer);
printf("\ndecimal: %f", stru.decimal);
printf("\nname: %s", stru.name);

printf("\ndata of union:");
uni.integer = 140;
uni.decimal = 150;
```

```
strcpy(uni.name, "Mike");
    printf("\ninteger: %d", uni.integer);
    printf("\ndecimal: %f", uni.decimal);
    printf("\nname: %s", uni.name);
    //difference four
    printf("\nAltering a member value:\n");
    stru.integer = 512;
    printf("data of structure:\n integer: %d\n decimal: %.2f\n name: %s\n
", stru.inte  ger, stru.decimal, stru.name);
    uni.integer = 512;
    printf("data of union:\n integer: %d\n decimal: %.2f\n name: %s\n", u
ni.integer, uni.decimal, uni.name);
```

```
    // difference two and three
    printf("\nsizeof structure: %d\n", sizeof(stru));
    printf("sizeof union: %d\n", sizeof(uni));
}
```

Output :

1.data of structure:

2. integer: 5

3. decimal: 15.00

4. name: John

5.

6.data of union:

7. integer: 5

8.decimal: 0.00

9. name:

10.

11.Accessing all members at a time:

12.data of structure:

13. integer: 163

14. decimal: 75.000000

15. name: John

16.

17.data of union:
18. integer: 1852337994
19. decimal: 1798376562491225303407185100 8.000000
20. name: John
21.
22.Accessing one member at a time:
23.data of structure:
24.integer: 140
25.decimal: 150.000000
26.name: Mike
27.data of union:
28.integer: 1701538125
29.decimal: 6948116125230294053683 2.000000
30.name: Mike

31.Altering a member value:

32.data of structure:

33. integer: 512

34. decimal: 150.00

35. name: Mike

36.data of union:

37. integer: 512

38. decimal: 0.00

39. name:

40.

41.sizeof structure: 28

42.sizeof union: 20

********** **END OF PROGRAM** **********