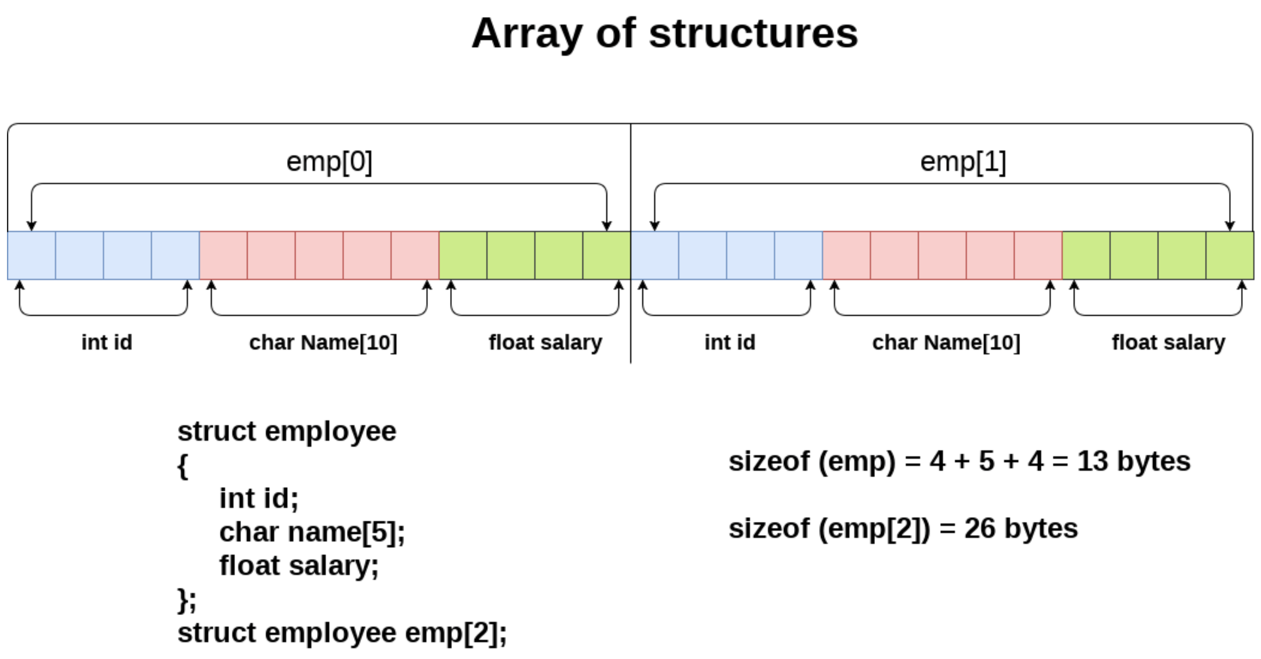
Array of Structures in C

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.



```
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

Let's see an example of an array of structures that stores information of 5 students and prints it.

```
#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;
}
```

• Union in C

Union can be defined as a user-defined data type which is a collection of different variables of different data types in the same memory location. The union can also be defined as many members, but only one member can contain a value at a particular point in time.

Union is a user-defined data type, but unlike structures, they share the same memory location.

Let's understand this through an example.

```
struct abc
{
   int a;
   char b;
}
```

The above code is the user-defined structure that consists of two members, i.e., 'a' of type **int** and 'b' of type **character**. When we check the addresses of 'a' and 'b', we found that their addresses are different. Therefore, we conclude that the members in the structure do not share the same memory location.

Deciding the size of the union

The size of the union is based on the size of the largest member of the union.

Let's understand through an example.

```
union abc{
```

```
int a;

char b;

float c;

double d;

};

int main()

{

  printf("Size of union abc is %d", sizeof(union abc));

  return 0;

}
```

As we know, the size of int is 4 bytes, size of char is 1 byte, size of float is 4 bytes, and the size of double is 8 bytes. Since the double variable occupies the largest memory among all the four variables, so total 8 bytes will be allocated in the memory. Therefore, the output of the above program would be 8 bytes.

Accessing members of union using pointers

We can access the members of the union through pointers by using the (->) arrow operator.

Let's understand through an example.

```
#include <stdio.h>
union abc
{
    int a;
    char b;
};
int main()
{
    union abc *ptr; // pointer variable declaration
    union abc var;
    var.a= 90;
    ptr = &var;
    printf("The value of a is : %d", ptr->a);
    return 0;
}
```

Difference Between Structure and Union in C

Structure	**Union**
A user can deploy the keyword **struct** to define a Structure.	A user can deploy the keyword **union** to define a Union.
The implementation of Structure in C occurs internally- because it contains separate memory locations allotted to every input member.	In the case of a Union, the memory allocation occurs for only one member with the largest size among all the input variables. It shares the same location among all these members/objects.
A user can access individual members at a given time.	A user can access only one member at a given time.
The Syntax of declaring a Structure in C is: struct [structure name] { type element_1; type element_2; . . } variable_1, variable_2, …;	The Syntax of declaring a Union in C is: union [union name] { type element_1; type element_2; . . } variable_1, variable_2, …;
A Structure does not have a shared location for all of its members. It makes the size of a Structure to be greater than or equal to the sum of the size of its data members.	A Union does not have a separate location for every member in it. It makes its size equal to the size of the largest member among all the data members.
Altering the values of a single member does not affect the other members of a Structure.	When you alter the values of a single member, it affects the values of other members.
In the case of a Structure, there is a specific memory location for every input data member. Thus, it can store multiple values of the various members.	In the case of a Union, there is an allocation of only one shared memory for all the input data members. Thus, it stores one value at a time for all of its members.
In the case of a Structure, a user can initialize multiple members at the same time.	In the case of a Union, a user can only initiate the first member at a time.