STRUCTURES IN C

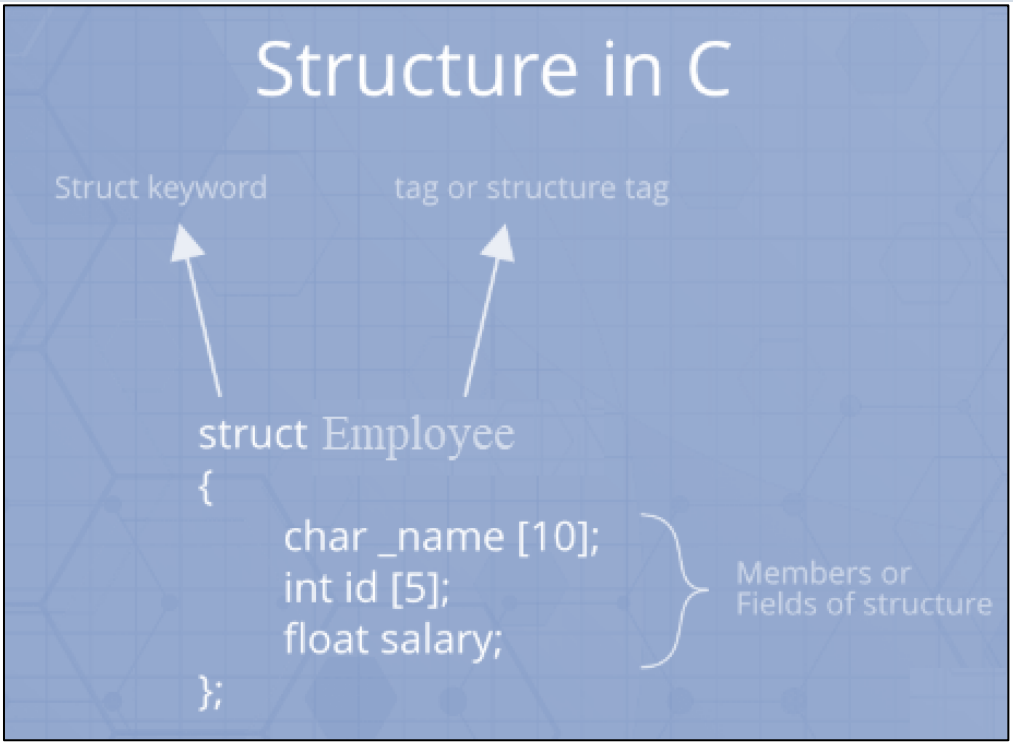
1. In C programming, a struct (or structure) is a collection of variables (can be of different types) under a single name.
2. Arrays allow to define type of variables that can hold several data items of the same kind. Similarly, **structure** is another user defined data type available in C that allows to combine data items of different kinds.
3. Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book −
   - Title
   - Author
   - Subject
   - Book ID

Syntax of struct

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

Here is an example:

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



Declaring Structure Variables

It is possible to declare variables of a **structure**, either along with structure definition or after the structure is defined. **Structure** variable declaration is similar to the declaration of any normal variable of any other datatype. Structure variables can be declared in following two ways:

1) Declaring Structure variables separately

```
struct Student
{
    char name[25];
    int age;
    char branch[10];
    //F for female and M for male
    char gender;
};

struct Student S1, S2;      //declaring variables of struct Student
```

2) Declaring Structure variables with structure definition

```
struct Student
{
    char name[25];
    int age;
    char branch[10];
    //F for female and M for male
    char gender;
}S1, S2;
```

Here S1 and S2 are variables of structure Student. However, this approach is not much recommended.

Accessing Structure Members

Structure members can be accessed and assigned values in a number of ways. Structure members have no meaning individually without the structure. In order to assign a value to any structure member, the member name must be linked with the **structure** variable using a dot . operator also called **period** or **member access** operator.

For example:

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

struct Student
{
    char name[25];
    int age;
    char branch[10];
    //F for female and M for male
    char gender;
};

int main()
{
    struct Student s1;

    /*
        s1 is a variable of Student type and
        age is a member of Student
    */
    s1.age = 18;
    /*
        using string function to add name
    */
    strcpy(s1.name, "ABC");
    /*
```

```
        displaying the stored values
    */
    printf("Name of Student : %s\n", s1.name);
    printf("Age of Student : %d\n", s1.age);

    return 0;
}
```

OUTPUT:

```
Name of Student : ABC

Age of Student : 18
```

NOTE: We can also use scanf() to give values to structure members through terminal.

```
scanf(" %s ", s1.name);

scanf(" %d ", &s1.age);
```

Passing structs to functions:

Here's how you can pass structures to a function

```
#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: ABC
Enter age: 13

Displaying information
Name: ABC
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:

Here's how you can return structure from a function:

```
#include <stdio.h>
struct student
{
    char name[50];
    int age;
};

// function prototype
struct student getInformation();

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`.

Nested Structure

You can use a structure inside another structure, which is fairly possible.

Example of Nested Structure in C Programming

Lets say we have two structure like this:

Structure 1: stu_address

```
struct stu_address
{
     int street;
     char *state;
     char *city;
     char *country;
}
```

Structure 2: stu_data

```
struct stu_data
{
    int stu_id;
    int stu_age;
    char *stu_name;
    struct stu_address stuAddress;
}
```

As you can see here that I have nested a structure inside another structure.

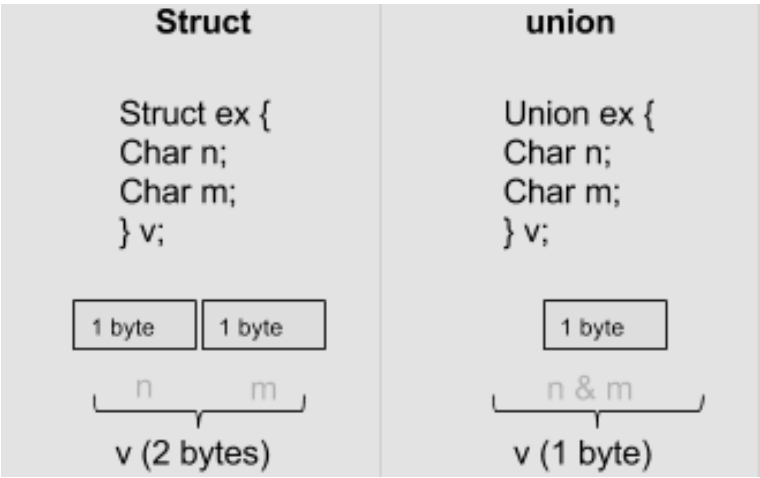
C - Unions

A union is a user-defined type similar to structs in C except for one key difference. Structs allocate enough space to store all its members whereas unions allocate the space to store only the largest member.

How to define a union?

We use the `union` keyword to define unions. Here's an example:

```
union car
{
  char name[50];
  int price;
};
```



Example

```
#include <stdio.h>

union item
{
    int a;
    float b;
    char ch;
};

int main( )
{
    union item it;
    it.a = 12;
    it.b = 20.2;
    it.ch = 'z';

    printf("%d\n", it.a);
    printf("%f\n", it.b);
    printf("%c\n", it.ch);

    return 0;
}
```

OUTPUT:

```
26426
20.1999
z
```