Introduction

In C programming, a struct (or structure) is a collection of variables (can be of different types) under a single name. 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.

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
- Book ID
- price

Syntax of struct

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

Example 1:

```
struct book
{
   char title[50];
   char author[50];
   int book_id;
   float price;
};
```

Example 2:

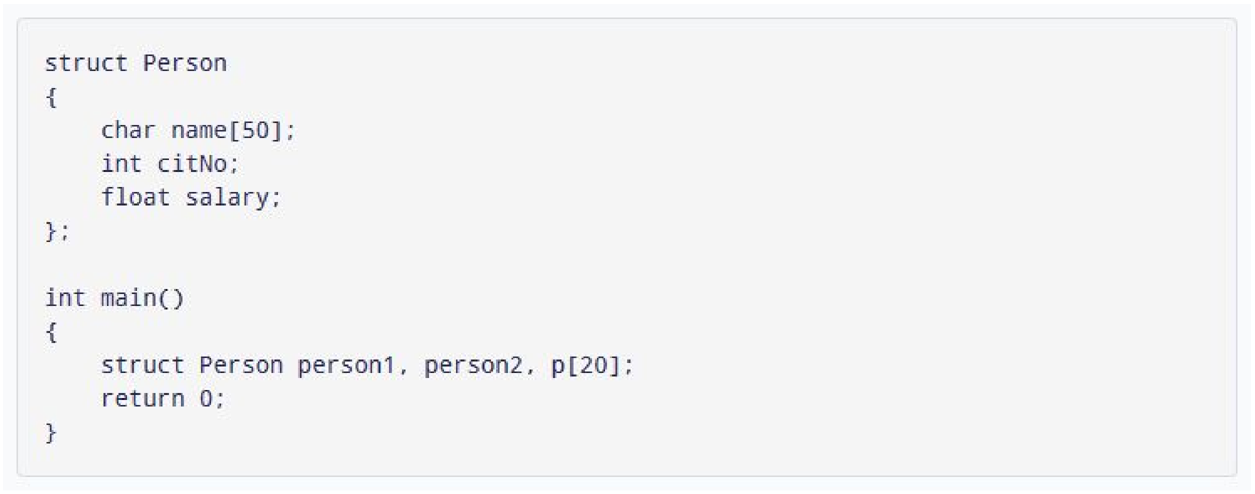
```
struct Person
{
   char name[50];
   int citNo;
```

```
    float salary;
};
```

Create struct variables

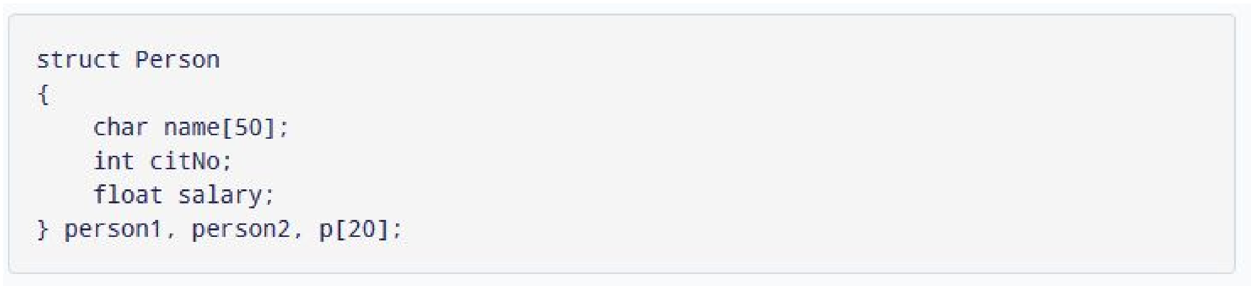
When a struct type is declared, no storage or memory is allocated. To allocate memory of a given structure type and work with it, we need to create variables.

Here's how we create structure variables:



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

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

Another way of creating a struct variable is:



```
struct Person
{
    char name[50];
    int citNo;
    float salary;
} person1, person2, p[20];
```

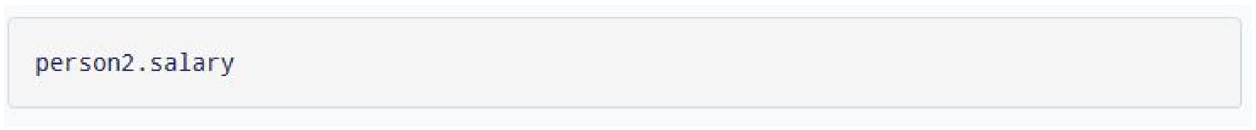
In both cases, two variables person1, person2, and an array variable p having 20 elements of type struct Person are created.

Access members of a structure

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

- Member operator (.)
- Structure pointer operator (->)

Suppose, you want to access the salary of person2. Here's how you can do it.



```
person2.salary
```

Example: Simple example of structure to store information of 2 student.

```
#include<stdio.h>
int main()
{
   struct studentData{
         char name[30];
         int rollNo;
         int age;
   } student1, student2;

   printf("Enter Data for student1:\n");
   printf("Enter name:");
   fgets(student1.name,20,stdin);
   printf("Enter rollNo:");
   scanf("%d",&student1.rollNo);
   printf("Enter age:");
   scanf("%d",&student1.age);

   printf("\nEnter Data for student2:\n");
   printf("Enter name:");
   fflush(stdin);
   fgets(student2.name,20,stdin);
   printf("Enter rollNo:");
   scanf("%d",&student2.rollNo);
```

```c
    printf("Enter age:");
    scanf("%d",&student2.age);

    printf("Displaying data of student1:\n");
    printf("Name=%s\n",student1.name);
    printf("RollNo=%d\n",student1.rollNo);
    printf("Age=%d\n",student1.age);

    printf("\nDisplaying data of student1:\n");
    printf("Name=%s\n",student2.name);
    printf("RollNo=%d\n",student2.rollNo);
    printf("Age=%d\n",student2.age);
    return 0;
}
```

Output:

```
Enter Data for student1:
Enter name:Dipen Karki
Enter rollNo:23
Enter age:23

Enter Data for student2:
Enter name:Hari Karki
Enter rollNo:32
Enter age:23

Displaying data of student1:
Name=Dipen Karki
RollNo=23
Age=23

Displaying data of student1:
Name=Hari Karki
RollNo=32
Age=23
```

```
}
```

Example 2:Create a structure named student that has name,roll,marks and remarks as members. Assume appropriate types and sizes of members. Use this structure to read and display records of N students.

```
#include<stdio.h>
#define n 5
int main()
{
   struct student
   {
         char name[30];
         int roll;
         float marks;
         char remarks[30];
   } s[n];

   int i;
   for(i=0;i<n;i++)
   {
         printf("\nEnter information of student %d\n",i+1);
         printf("Enter name:");
         fgets(s[i].name,30,stdin);
         printf("Enter rollNo:");
         scanf("%d",&s[i].roll);
         printf("Enter marks:");
         scanf("%f",&s[i].marks);
         fflush(stdin);
         printf("Enter remarks:");
         fgets(s[i].remarks,30,stdin);
   }

   printf("\nDisplaying Detail Information:\n");
   printf("Student Name\tRoll No\tMarks\tRemarks");
   printf("\n-------------------------------------------------------------\n");
   for(i=0;i<n;i++)
   {
         printf("%s\t\t%d\t%f\t%s",s[i].name,s[i].roll,s[i].marks,s[i].remarks);
   }
   return 0;
```

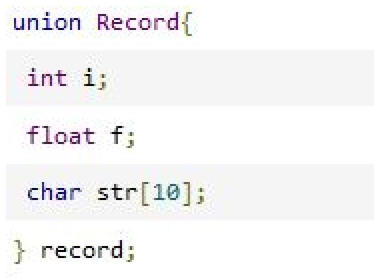
}

Example 3: Create a structure named student which has members:symbol,name and obtained percentage. Read symbol,name and percentage of N students and print the records of students who have passed in distinction.

Union in C:

Union is a data type in C programming that allows different data types to be stored in the same memory locations. Union provides an efficient way of reusing the memory location, as only one of its members can be accessed at a time. A union is used almost in the same way you would declare and use a structure. Let's see this as an example.

Here's how it works with syntax.



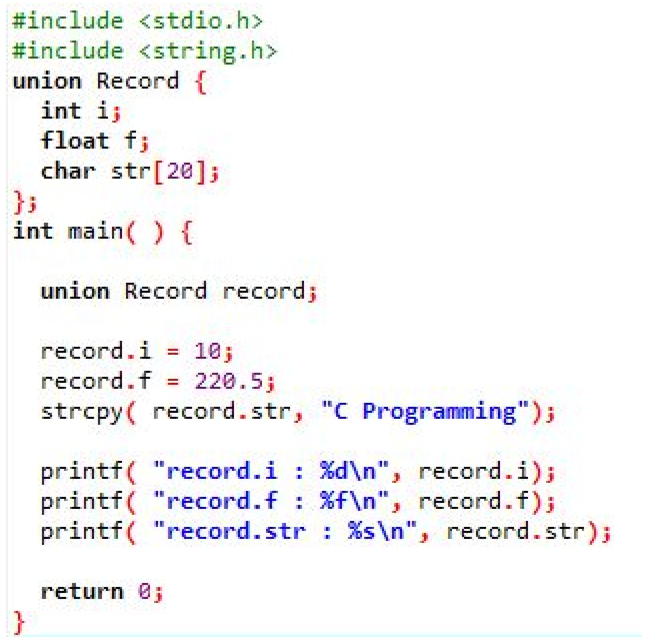
```
union Record{
 int i;
 float f;
 char str[10];
} record;
```

In this example, we initialized a union named Record which had three data members. A reference to this union is created, which is called as record.

In the above declared union, the memory occupied by the union will be the memory required for the largest member of the union. In this case the union will occupy 10 bytes of memory space, as the character string occupies the maximum space. The following shows the total memory size. If you print sizeof(record), this will give you the size of the union, which in this case will be 10.

Example

In this example, we will use the aforementioned union called *record*. We can access the data members of the union using the '.' operator.

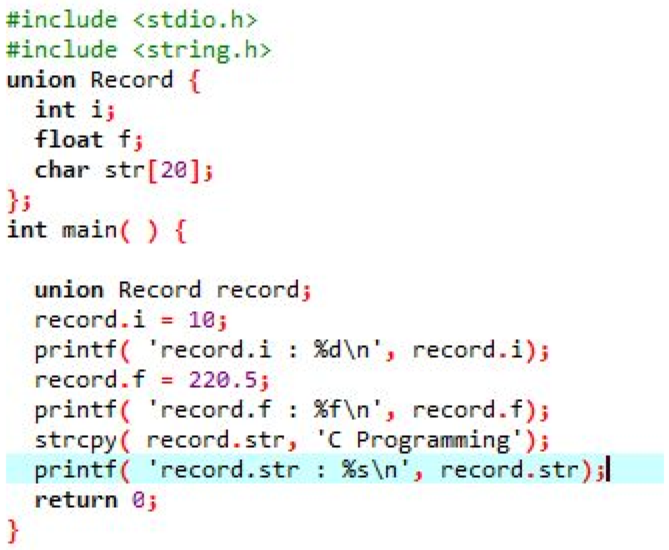


```c
#include <stdio.h>
#include <string.h>
union Record {
  int i;
  float f;
  char str[20];
};
int main( ) {

  union Record record;

  record.i = 10;
  record.f = 220.5;
  strcpy( record.str, "C Programming");

  printf( "record.i : %d\n", record.i);
  printf( "record.f : %f\n", record.f);
  printf( "record.str : %s\n", record.str);

  return 0;
}
```

Output:

```
record.i : 1917853763
record.f : 4122360580327794860452759994368.000000
record.str : C Programming
```

As we see from the output, the value of *record.i* and *record.f* is corrupted. This is because the final memory assignment was done to *record.str*, which is why on printing *record.str*, it gives the correct value. In order to print the correct values of each data member, it's better to use the *printf* statement one at a time as follows:



```
#include <stdio.h>
#include <string.h>
union Record {
  int i;
  float f;
  char str[20];
};
int main( ) {

  union Record record;
  record.i = 10;
  printf( 'record.i : %d\n', record.i);
  record.f = 220.5;
  printf( 'record.f : %f\n', record.f);
  strcpy( record.str, 'C Programming');
  printf( 'record.str : %s\n', record.str);
  return 0;
}
```

Difference between Structure and Union

STRUCTURE	UNION
Every memory has its own memory sapce.	All the members use the same memory space to store the values.
It can handle all the members(or) a few as required at a time.	It can handle only one member at a time,as all the members use the same space.
Keyword struct is used.	Keyword union is used.
It may be initialized with all its members.	Only its first member may be initialized.
Any member can be accessed at any time without the loss of data.	Only one member can be accessed at any time with the loss of previous data.
Different interpretation for the same memory location are not possible.	Different interpretations for the same
More storage space is required.	Minimum storage space is required.

Syntax:
structure strut-name
{
}var1..varn;
	Syntax:
	union union-name
	{
	}var1..varn;