Unit 8. Structure and Union

8.1 Introduction,
8.2 Array of structure,
8.3 Passing structure to function,
8.4 Passing array of structure to function,
8.5 Structure within structure (Nested Structure),
8.6 Union,
8.7 Pointer to structure

Structure in C

- Structure is the collection of variables of different types under a single name for better handling.
- For example: You want to store the information about person about his/her name, citizenship number and salary.
- You can create these information separately but, better approach will be collection of these information under single name because all these information are related to person.

- Unlike an array, structure is user defined data type.

- A structure is heterogeneous data structure where as an array is a homogeneous data structure

- Keyword **struct** is used for creating a structure.

Syntax of structure

```
struct structure_name
{
   data_type member1;
   data_type member2;
   .
   .
   data_type memeber;
};    /* end with semicolon */
```

We can create the structure for a person as mentioned above as:

```
struct person
{
   char name[50];
   int citz_no;
   float salary;
};
```

This declaration above creates the derived data type **struct person**.

Structure variable declaration

- **When a structure is defined, it creates a user-defined type but, no storage is allocated. For the above structure of person, variable can be declared as:**

```
struct person
{
   char name[50];
   int cit_no;
   float salary;
};    /* Above is the declaration of data type  struct person  */
```

- Variables can be decaled for the structure as declaration of other variables of other data types:

```
struct person    p1, p2, p[20];   /* p1,p2 are single structure variables and p is array of structures */
```

- Another way of creating structure variable is:

```
struct person
{
   char name[50];
   int cit_no;
   float salary;
}p1 ,p2 ,p[20];
```

- In both cases, 2 variables *p1*, *p2* and array *p* having 20 elements of type **struct person** are created.

Accessing members of a structure

- There are two types of operators used for accessing members of a structure.
  - Member operator(.)
  - Structure pointer operator(->) (will be discussed in later in chapter pointer to structures)
- Any member of a structure can be accessed as:

  structure_variable_name.member_name
- Suppose, we want to access salary for variable *p2*. Then, it can be accessed as:

  p2.salary

Example: Program to input two distances with feet and inch and find the sum

```
#include <stdio.h>
struct Distance
{
   int feet;
   float inch;
}d1,d2,sum;

int main()
{
   printf("1st distance:\n");
   printf("\nEnter feet: ");
   scanf("%d",&d1.feet);  /* input of feet for structure variable d1 */
   printf("\nEnter inch: ");
   scanf("%f",&d1.inch);  /* input of inch for structure variable d1 */
   printf("\n2nd distance:\n");
   printf("\nEnter feet: ");
   scanf("%d",&d2.feet);  /* input of feet for structure variable d2 */
   printf("\nInch: ");
   scanf("%f",&d2.inch);  /* input of inch for structure variable d2 */
   sum.feet=d1.feet+d2.feet;
   sum.inch=d1.inch+d2.inch;
   if (sum.inch>12)
                  {  //If inch is greater than 12, changing it to feet.
      ++sum.feet;
      sum.inch=sum.inch-12;
   }
   printf("Sum of distances=%d\'-%.1f\"",sum.feet,sum.inch);
/* printing sum of distance d1 and d2 */
   return 0;
}
```

Out put of above program

```
1st distance:

Enter feet: 3

Enter inch: 10

2nd distance:

Enter feet: 4

 Inch: 8
Sum of distances=8'-6.0"
```

Keyword typedef while using structure

- Programmer generally use typedef while using structure in C language.
- Typedef in C is a keyword in C which is used to give the new name to the existing data type For example:

```
typedef  int  integer;    /* gives another name integer to int */
```

- After typedef, we can declare integer variable as :
  - Integer   x,y;   /* look data type here used is integer in place of word int */
- When we define structure, two words struct and structure Name together represent our new data type. To give a single name to this name typedef can be used similarly.

```
typedef struct complex{
  int imag;
  float real;
}comp;
```

- Here struct complex is named as comp which can be used to define structure variable later on
- Now declaration: **comp c1,c2;**   **/* variable declaration for structure*/**
- Here, typedef keyword is used in creating a type *comp*(which is of type as **struct complex**).
- Then, two structure variables *c1* and *c2* are created by this *comp* type.

Structures within structures:Nested Structures

- Structures can be nested within other structures in C programming.

```
struct complex         /* Structure  :  struct complex   */
{
        int imag_value;
        float real_value;
};

struct number                /*  Structure:  struct number
{
  struct complex c1;     / *  struct complex   variable  c1 as member of struct number */
  int real;
}n1,n2;            /* declaration of variable n1 and n2 for struct number */
```

- Suppose you want to access *imag_value* for *n2* structure variable then, structure member ***n1.c1.imag_value*** is used.

- Pointers can be accessed along with structures. A pointer variable of structure can be created as below:

Array of Structures

- Similarly to array of variables of other data type, array of structures can be defined that holds the record of similar structure type.
- Declaration of array of structure:
  - **struct struct_name   var[SIZE];**
  - e.g.   **Struct student s[100];**   declares array of 100 students of defined structure type struct student.
- To access the data member of each variable, array index is used similar to other array.
- e.g.: s[0].fname,   s[0].lname for accessing members fname , lname of first student.
- In general, s[i].member_name is used to access member of

Example: Array of Structure

```
#include<stdio.h>
struct student
{
        char fname[20];
        char lname [20];
        int rollno;
};

main()
{
        struct student s[10]; /* array of structure */
        int i,n;
        printf("\nHow many Student Max 10:");
        scanf("%d",&n);
        printf("\nInput students details:");
        for(i=0;i<n;i++)
        {
                printf("#Student %d:",i+1);
                printf("First Name: ");
                scanf("%s",s[i].fname);
                printf("Last Name:");
                scanf("%s",s[i].lname);
                printf("Roll NO: ");
                scanf("%d",&s[i].rollno);
        }
        printf("\nDetails of Students:\n");
        printf("S.No\tName\t\t\tRoll No\n");
        for(i=0;i<n;i++)
        {
                printf("%d\t",i+1);
                printf("%s %s\t\t",s[i].fname,s[i].lname);
                printf("%d\n",s[i].rollno);
        }
        return 0;
}
```

Output:

```
How many Student Max 10:5

Input students details:#Student 1:First Name: Ram
Last Name:Thapa
Roll NO: 2
#Student 2:First Name: Hari
Last Name:Bist
Roll NO: 4
#Student 3:First Name: Gita
Last Name:Paudel
Roll NO: 5
#Student 4:First Name: Sita
Last Name:Giri
Roll NO: 6
#Student 5:First Name: Kapil
Last Name:KC
Roll NO: 9

Details of Students:
S.No   Name              Roll No
1      Ram Thapa          2
2      Hari Bist          4
3      Gita Paudel        5
4      Sita Giri          6
5      Kapil KC           9
```

Passing Structure to Function

- A structure can be passed to the function argument as other type variable as
  - Passing by value : Structure name is passed to the function as argument.
  - Passing by Address: Address of structure variable is passed to the function as argument.
- Below is the example of Passing structure to function.

```c
/* Passing Structure to Function */
#include<stdio.h>
struct student {
			char fname[20];
			char lname [20];
			int rollno;
};
void getStudentInfo(struct student *s)  {
			printf("First Name: ");
			scanf("%s",s->fname);
			printf("Last Name:");
			scanf("%s",s->lname);
			printf("Roll NO: ");
			scanf("%d",&s->rollno);
}
void showStudentInfo(struct student s)  {
			printf("\nName:%s %s",s.fname,s.lname);
			printf("\nRoll No: %d",s.rollno);
}
main()  {
			struct student s1,s2; /* array of structure */
			printf("\nInput students details for S1:");
			getStudentInfo(&s1);  /*Passing by Address */
			printf("\nInput students details for S2:");
			getStudentInfo(&s2);
			printf("\nStudent Details S1:\n");
			showStudentInfo(s1); /*Passing by Value */
			printf("\nStudent Details S2:\n");
			showStudentInfo(s2);
			return 0;
}
```

OUTPUT:

```
Input students details for S1:First Name: Ram
Last Name:Thapa
Roll NO: 2

Input students details for S2:First Name: Hari
Last Name:Silwal
Roll NO: 5

Student Details S1:

Name:Ram Thapa
Roll No: 2
Student Details S2:

Name:Hari Silwal
Roll No: 5
```

Passing array of structure to function,

```
#include<stdio.h>
struct student
{
		char fname[20];
		char lname [20];
		int rollno;
};
void getStudentInfo(struct student s[],int n)
{
		int i;
		for (i=0;i<n;i++)
		{
				printf("Student %d#\n",i+1);
				printf("First Name: ");
				scanf("%s",&s[i].fname);
				printf("Last Name:");
				scanf("%s",&s[i].lname);
				printf("Roll NO: ");
				scanf("%d",&s[i].rollno);
		}
}
```

```
void showStudentInfo(struct student s[],int n)
{
        int i;
        printf("S.No\tName\t\t\tRoll NO\n ");
        for(i=0;i<n;i++)
        {
                printf("%d\t",i+1);
                printf("%s %s\t\t",s[i].fname,s[i].lname);
                printf("%d\n",s[i].rollno);
        }
}
main()
{
        struct student s[100]; /* array of structure */
        int n;
        printf("How many Students:");
        scanf("%d",&n);
        printf("\nInput details for %d Students:\n",n);
        getStudentInfo(s,n);  /*Passing Address of array */
        printf("Student Details:\n");
        showStudentInfo(s,n); /*Passing by Value */
        return 0;
}
```

Union

- Unions are quite similar to the structures in C. Union is also a derived type as structure.
- Union can be defined in same manner as structures just the keyword used in defining union in **union** where keyword used in defining structure was **struct.**

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

- Union variables can be created in similar manner as structure variable.

```
union car
{
           char name[50];
           int price;
}c1, c2, *c3;
```

OR;

```
 union car
{
            char name[50];
           int price;
};      /* declaration of Union  Car */
```

- After declaration of card , variables also can be defined similar to structure as:

```
 union car  c1,c2;
```

Difference between union and structure

- Though unions are similar to structure in so many ways, the difference between them is crucial to understand.
- This can be demonstrated by the example :

```
#include <stdio.h>
union job
{       //defining a union
  char name[32];
  float salary;
  int worker_no;
}u;
struct job1
{
  char name[32];
  float salary;
  int worker_no;
}s;
int main()
{
  printf("size of union = %d",sizeof(u));
  printf("\nsize of structure = %d", sizeof(s));
  return 0;
}
```

```
Output:
size of union = 32
size of structure = 40
```

Difference between Structure and Union

- There is difference in memory allocation between union and structure as suggested in above example.
- The amount of memory required to store a structure variables is the sum of memory size of all members.

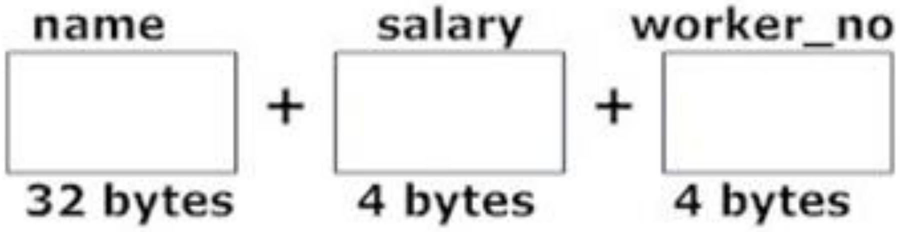

Fig: Memory allocation in case of structure

Difference between Structure and Union

- But, the memory required to store a union variable is the memory required for largest element of an union.
- In above largest member is name which requires 32 bytes of storage.

name
32 bytes

Fig: Memory allocation in case of union