Structures and Unions

Structure

A structure is a user defined data type. We know that arrays can be used to represent a group of data items that belong to the same type, such as int or float. However we cannot use an array if we want to represent a collection of data items of different types using a single name. A structure is a convenient tool for handling a group of logically related data items.

Structure is a user defined data type used to represent a group of data items of different types using a single name.

The syntax of structure declaration is

```
struct structure_name
      {
         type element 1;
         type element 2;
         ……………..
         type element n;
      };
```

In structure declaration the keyword **struct** appears first, this followed by structure name. The member of structure should be enclosed between a pair of braces and it defines one by one each ending with a semicolon. It can also be array of structure. There is an enclosing brace at the end of declaration and it end with a semicolon.

We can **declare structure variables** as follows

```
        struct structure_name var1,var2,…..,var n;
```

Example:

To store the names, roll number and total mark of a student you can declare 3 variables. To store this data for more than one student 3 separate arrays may be declared. Another choice is to make a structure. No memory is allocated when a structure is declared. It just defines the “form” of the structure. When a variable is made then memory is allocated. This is equivalent to saying that there's no memory for “int”, but when we declare an integer that is int var; only then memory is allocated. The structure for the above-mentioned case will look like

```
struct student
{
        int rollno;
        char name[25];
        float totalmark;
};
```

We can now declare structure variables stud1, stud2 as follows

```
                struct student stud1,stud2;
```

Thus, the stud1 and stud2 are structure variables of type student. The above structure can hold information of 2 students.

It is possible to combine the declaration of structure combination with that of the structure variables, as shown below.

```
struct structure_name
      {
        type element 1;
        type element 2;
        ……………..
        type element n;
      } var1,var2,…,varn;
```

The following single declaration is equivalent to the two declaration presented in the previous example.

```
struct student
  {
        int rollno;
        char name[25];
        float totalmark;
  } stud1, stud2;
```

Accessing structure Variable

The different variable types stored in a structure are called its members. The structure member can be accessed by using a **dot (.) operator**, so the dot operator is known as structure **member operator**.

Example:

In the above example stud1 is a structure variable of type student. To access the member name, we would write stud1.name. Similarly, stud1's rollno and stud1's totalmark can be accessed by writing stud1.rollno and stud1.totalmark respectively.

Initializing Structure Members

Structure members can be initialized at declaration. This much the same manner as the element of an array; the initial value must appear in the order in which they will be assigned to their corresponding structure members, enclosed in braces and separated by commas.

The **general form** is

struct structure_name var={val1,val2,val3…..};

Example:

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

```
#include<conio.h>
void main()
  {
       struct student
          {
            char *name;
            int rollno;
            float totalmark;
          };
struct student stud1={"Venkat",1,98};
struct student stud3= {"Shweta",3,97};
struct student stud2={"Arpita",2,99};
clrscr();
printf(“STUDENTS DETAILS:\n”);
printf(“\n\n Roll number:%d\n Name:%s\n Total Marks:%f”, stud1.rollno, stud1.name,
stud1.totalmark);
printf(“\n\n Roll number:%d\n Name:%s\n Total Marks:%f”, stud2.rollno, stud2.name,
stud2.totalmark);
printf(“\n\n Roll number:%d\n Name:%s\n Total Marks:%f”, stud3.rollno, stud3.name,
stud3.totalmark);
getch();
}
```

Output

```
STUDENTS DETAILS:
Roll number: 1
Name: Venkat
Total Marks:98.000000

Roll number: 2
Name: Arpita
Total Marks:99.000000

Roll number: 2
Name:Shweta
 Total Marks:99.000000
```

Array of structures:

It is possible to store a structure has an array element. i.e., an array in which each element is a structure. Just as arrays of any basic type of variable are allowed, so are arrays of a given type of structure. Although a structure contains many different types, the compiler never gets to know this information because it is hidden away inside a sealed structure capsule, so it can believe that all the elements in the array have the same type, even though that type is itself made up of lots of different types.

The declaration statement is given below.

```
        struct struct_name
             {
             type element 1;
             type element 2;
             ……………..
             type element n;
             }array name[size];
```

Example:

```
struct student
             {
                 int rollno;
                 char name[25];
                 float totalmark;
             } stud[100];
```

In this declaration stud is a 100-element array of structures. Hence, each element of stud is a separate structure of type student. An array of structure can be assigned initial values just as any other array. So the above structure can hold information of 100 students.

Program to demonstrate use of array of structure

```
#include  <stdio.h>
#include  <conio.h>
void main()
 {
struct student
 {
        int rollno;
        char name[25];
        int totalmark;
 }stud[100];

int n,i;
clrscr();
printf("Enter total number of students\n\n");
scanf("%d",&n);
for(i=0;i<n;i++)
    {
        printf("Enter details of %d-th student\n",i+1);
        printf("Name:\n");
        scanf("%s",&stud[i].name);
        printf("Roll number:\n");
        scanf("%d",&stud[i].rollno);
        printf("Total mark:\n");
        scanf("%d",&stud[i].totalmark);
   }
```

```
    printf("STUDENTS DETAILS:\n");
  for(i=0;i<n;i++)
      {
           printf("\nRoll number:%d\n",stud[i].rollno);
           printf("Name:%s\n",stud[i].name);
          printf("Total mark:%d\n",stud[i].totalmark);
      }
getch();
}
```

OUTPUT

```
Enter total number of students:
3

Enter details of 1-th student
Name:SUBAHAS
Roll number:11
Total mark:589

Enter details of 2-th student
Name:RUKSANA
Roll number:12
Total mark:594
Enter details of 3-th student
Name:SANA
Roll number:13
Total mark:595

STUDENTS DETAILS:
Roll number:11
Name: SUBAHAS
Total mark:589

Roll number:12
Name: RUKSANA
Total mark:594

Roll number:13
Name: SANA
Total mark:595
```

Structure as structure member (Embedded structure):

A structure inside another structure is called an **embedded structure**. A structure can have one or more of its member as another structure, but a structure cannot be member to itself when a structure is used as structure member. In such situation, the declaration of the embedded structure must appear before the declaration of the outer structure. For example

```
#include  <stdio.h>
#include  <conio.h>
void main()
{
          struct dob
           {
                    int day;
                    int month;
                    int year;
          };

          struct student
           {
                    struct dob d;
                    int rollno;
                    char name[25];
                    int totalmark;
           }stud[25];

int n,i;
clrscr();
printf("Enter total number of students");
scanf("%d",&n);
for(i=0;i<n;i++)
          {
          printf("\n\nEnter details of %d student",i+1);
          printf("\nName:");
          scanf("%s",&stud[i].name);
          printf("\nRoll number:");
          scanf("%d",&stud[i].rollno);
          printf("\nTotal mark:");
          scanf("%d",&stud[i].totalmark);
          printf("\nDate of birth (Format:01 06 2010):");
          scanf("%d%d%d",&stud[i].d.day,&stud[i].d.month,&stud[i].d.year);
          }
printf("\nSTUDENTS DETAILS:\n");
for(i=0;i<n;i++)
          {
```

```
        printf("\nRoll number:%d\n",stud[i].rollno);
        printf("Name:%s\n",stud[i].name);
        printf("Total mark:%d\n",stud[i].totalmark);
        printf("Date of birth : %d / %d / %d \n\n",
stud[i].d.day,stud[i].d.month,stud[i].d.year);
        }
getch();
}
```

OUTPUT

```
Enter total number of students 2

Enter details of 1 student
Name: karthik
Roll number:12
Total mark:588
Date of birth (Format:01 06 2010):11 12 1997

Enter details of 2 student
Name: sarita
Roll number:18
Total mark:598
Date of birth (Format:01 06 2010):1 2 1997

STUDENTS DETAILS:
Roll number:12
Name: karthik
Total mark: 588
Date of birth : 11 / 12 / 1997

Roll number:18
Name: sarita
Total mark:598
Date of birth : 1 / 2 / 1997
```

Union

Union is a user created data type similar to structure but in this case all the members share a common memory location. The size of the union corresponds to the length of the largest member. Since the member share a common location they have the same starting address.

The real purpose of unions is to prevent memory fragmentation by arranging for a standard size for data in the memory. By having a standard data size we can guarantee that any hole left when dynamically allocated memory is freed will always be reusable by another instance of the same type of union. This is a natural strategy in system programming where many instances of different kinds of variables with a related purpose and stored dynamically.

A union is declared in the same way as a structure. The **syntax** of union declaration is

```
union union_name
    {
      type element 1;
      type element 2;
      ……………..
      type element n;
    };
```

This declares a type template. **Variables are then declared as**:

```
union union_name x,y,z;
```

For example, the following code declares a union data type called Student and a union variable called stud:

```
union student
        {
        int rollno;
        float totalmark;
        };
union student stud;
```

It is possible to combine the declaration of union combination with that of the union variables, as shown below.

```
union union_name
        {
        type element 1;
        type element 2;
        ……………..
        type element n;
        }var1,var2,…,varn;
```

The following single declaration is equivalent to the two declaration presented in the previous example.

```
union student
{
int rollno;
float totalmark;
}x,y,z;
```

Exercise: Compare structure and Union

The difference between structure and union is,

Structure	**Union**
The amount of memory required to store a structure variable is the sum of the size of all the members.	The amount of memory required is always equal to that required by its largest member.
Each member have their own memory space.	One block is used by all the member of the union.
Keyword struct defines a structure	Keyword union defines a union.

struct s_tag
{
  int ival;
  float fval;
  char *cptr;
}s;
	union u_tag
	{
	  int ival;
	  float fval;
	  char *cptr;
	}u;

Within a structure all members gets memory allocated; therefore any member can be retrieved at any time.	While retrieving data from a union the type that is being retrieved must be the type most recently stored. It is the programmer's responsibility to keep track of which type is currently stored in a union; the results are implementation-dependent if something is stored as one type and extracted as another.
One or more members of a structure can be initialized at once.	A union may only be initialized with a value of the type of its first member; thus union u described above (during example declaration) can only be initialized with an integer value.

Structure:

```
#include<stdio.h>
#include<conio.h>
void main()
{
struct testing
        {
            int a;
            char b;
            float c;
            }var;
clrscr();
printf("\nsizeof(var) is %d",sizeof(var));
printf("\nsizeof(var.a) is %d",sizeof(var.a));
printf("\nsizeof(var.b) is %d",sizeof(var.b));
printf("\nsizeof(var.c) is %d",sizeof(var.c));
var.a=10;
printf("\nvalue of var.a is %d",var.a);
var.b='b';
printf("\nvalue of var.b is %c",var.b);
var.c=15.55;
printf("\nvalue of var.c is %f",var.c);
printf("\nvalue of var.a is %d",var.a);
printf("\nvalue of var.b is %c",var.b);
printf("\nvalue of var.c is %f",var.c);
getch();
}
```

OUTPUT

```
sizeof(var) is 7
sizeof(var.a) is 2
sizeof(var.b) is 1
sizeof(var.c) is 4
value of var.a is 10
value of var.b is b
value of var.c is 15.550000
value of var.a is 10
value of var.b is b
value of var.c is 15.550000
```

Union:

```
#include<stdio.h>
#include<conio.h>
void main()
{
union testing
            {
```

```
            int a;
            char b;
            float c;
            }var;
clrscr();
printf("\nsizeof(var) is %d",sizeof(var));
printf("\nsizeof(var.a) is %d",sizeof(var.a));
printf("\nsizeof(var.b) is %d",sizeof(var.b));
printf("\nsizeof(var.c) is %d",sizeof(var.c));
var.a=10;
printf("\nvalue of var.a is %d",var.a);
var.b='b';
printf("\nvalue of var.b is %c",var.b);
var.c=15.55;
printf("\nvalue of var.a is %f",var.c);
printf("\nvalue of var.a is %d",var.a);
printf("\nvalue of var.b is %c",var.b);
printf("\nvalue of var.c is %f",var.c);
getch();
}
```

OUTPUT

```
sizeof(var) is 4
sizeof(var.a) is 2
sizeof(var.b) is 1
sizeof(var.c) is 4
value of var.a is 10
value of var.b is b
value of var.c is 15.550000
value of var.a is -13458
value of var.b is
value of var.c is 15.550000
```