UNIT - 8
Structure and Unions

1. Introduction to Structure

Structure is a convenient tool for handling a group of logically related heterogeneous data types. Structure helps to organize data especially in large programs, because they provide group of variables of different data type to be treated as a single unit. It is most convenient way to keep related data under one roof.

- A structure is usually used when we need to store dissimilar or heterogeneous data together.
- The structure elements are stored in contiguous memory location as array.
- Structure elements can be accessed through a structure variable using a dot (.) operator.
- Structure elements can be accessed through a pointer to a structure using the arrow (->) operator.
- All the elements of one structure variable can be assigned to another structure variable using the assignment (=) operator.
- It is possible to pass a structure variable to a function either by value or by reference.
- It is possible to create an array of structure i.e. similar type of structure is placed in a common variable name. For example: we need to store the detail information of individual student in a class.

❖ Ordinary variable
The ordinary variable hold one piece of data type.
For example:

```
int  a  =  5;
char  c  =  'a';
```

❖ Array variable
The array variable holds a collection of similar data type in contiguous memory location.
For example:

```
int  a[3]  =  {1,2,3 };
```

❖ Structure variable
The structure variable holds a collection of different data type.
For example:

```
struct  student
{
        char  name[20];
        int  roll_no;
        float  percentage;
};
```

2. Rules for Defining Structure

- ➢ The keyword **struct** is used for structure declaration.
- ➢ Generally the structures are globally defined.
- ➢ The closing brace in structure type declaration must be followed by a semi colon.
- ➢ The structure type declaration does not tell the compiler to reserve any space in memory.

3. Declaration of structure

```
struct  structure_name
{
        data_type  member-1;
        data_type  member-2;
        data_type  member-n;
}structure variable;
```

OR

```
struct  structure_name
{
        data_type  member-1;
        data_type  member-2;
        data_type  member-n;
};
struct  structure_name structure_variable;
```

4. Accessing structure elements:

We can access the individual elements of structure using the dot (.) operator .

structure_variable . structure_member

Example of Structure

WAP to create structure of book having name, pages and price. Provide values and display them.

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

struct book
{
        char name[20];
        int pages;
        float price;
};
void main()
{
        struct book b1;
        b1.pages=500;
        b1.price=815.5;
        strcpy(b1.name, “C Programming”);
        printf(“\nName=%s, pages=%d and price=%f”,b1.name,b1.pages,b1.price);
getch();
clrscr();
}
```

5. Feature of structure

5.1 Structure Assignment

The value of the structure variable can be assigned to another structure variable of the same type using the assignment (=) operator.

For example-1:

```
#include<stdio.h>
#include<conio.h>
struct  sample
{
        int  a;
        float  b;
}x,  y;
void  main()
{
        x.a  =  10;
        x.b  =  15.8;
        y=x;
        printf("The  value  is  %d  and  %f",  y.a,  y.b);
        getch();
}
```

Output:

```
The  value  is  10  and  15.80000
```

For example-2:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
struct  employee
{
char  name[20];
char  gender;
int  age;
};
void  main()
{
struct  employee  e1  =  {"Hari",  'M',  35};
struct  employee  e2,  e3;
strcpy  (e2.name,  e1.name);
e2.gender  =  e1.gender;
e2.age  =  e1.age;
e3=e2;
printf("%s  %c  %d  \n",e1.name,  e1.gender,  e1.age);
printf("%s  %c  %d  \n",e2.name,  e2.gender,  e2.age);
printf("%s  %c  %d  \n",e3.name,  e3.gender,  e3.age);
getch();
}
```

Output:

```
Hari  M  35
Hari  M  35
Hari  M  35
```

5.2 Array of structure

An array is a group of identical (similar) data which are stored in contiguous memory location in a common variable name. A similar type of structure placed in a common variable name is called array of structure.

For example:

WAP to create structure “employee” having name, gender and age. Read the values for 10 employees and display them.

```
#include  <stdio.h>
#include  <conio.h>
struct  employee
{
        char  name[20];
        char  gender;
        int  age;
}em[10];
void  main()
{
        int  i;
        for  (i=0;  i<10;  i++)
        {
                printf("Enter  name,  gender  and  age  of  a  student:  \n");
                scanf("%s  %c  %d",  em[i].name,  &em[i].gender,  &em[i].age);
         }
        for  (i=0;  i<10;  i++)
        {
                printf("\nName= %s \t  Gender=%c \t  and  Age=%d",  em[i].name, em[i].gender,  em[i].age);
        }
getch();
clrscr();
}
```

5.3 Passing structure member to function

When we are passing a member of a structure to a function, we are actually passing the value of that member in the form of simple variable.

For example-1:

```
#include <stdio.h>
#include <conio.h>
struct num
{
        int a;
        int b;
}n;
void fun (int, int);
void main()
{
        printf("Enter any two numbers:\n");
        scanf ("%d %d", &n.a, &n.b);
        fun(n.a, n.b);
        getch();
        clrscr();
}
void fun(int x, int y)
{
        int sum;
        sum = x+y;
        printf("The sum is: %d", sum);
}
```

Output:

```
Enter any two numbers:
4
6
The sum is: 10
```

For example-2:

```
        #include <stdio.h>
        #include <conio.h>
        struct book
        {
        char name[25] ;
        char author[25] ;
        int page;
        float price;
        };

        void display (char*, char*, int, float);
        void main()
        {

        struct book b1 = { "C-programming", "XYZ", 101, 350.55 };
        display ( b1.name, b1.author, b1.page, b1.price );
        }
        void display ( char *s, char *t, int n, float m)
        {
        printf ( "\n%s %s %d %f", s, t, n, m) ;
        getch();
        }
```

5.4 Passing entire structure to function

When a structure is used as an argument to a function, the entire structure is passed to the function using the standard call by value method.

For example-1:

```
#include <stdio.h>
#include <conio.h>
struct num
{
        int a;
        int b;
}n;
void fun (struct num);
void main()
{
        printf("Enter any two numbers:\n");
        scanf ("%d %d", &n.a, &n.b);
        fun(n);
        getch();
}

void fun(struct num x)
{
        int sum;
        sum = x.a+x.b;
        printf("The sum is: %d", sum);
}
```

Output:

```
Enter any two numbers:
4
6
The sum is: 10
```

For example-2:

```
#include <stdio.h>
#include <conio.h>
struct book
{
        char name[25] ;
        char author[25] ;
        int callno ;
} ;
void display (struct book);
void main( )
{
        struct book b1 = { "C-programming", "xyz", 101 } ;
        display ( b1 ) ;
}
void display ( struct book b )
{
        printf ( "\n%s %s %d", b.name, b.author, b.callno ) ;
        getch();
        clrscr();
}
```

Output:

```
C-programming XYZ 101
```

5.5 Nested structure

If we can define a structure within structure then it is known as nested type structure. Nested structures are structures within structure.

Declaration format-1

```
struct  employee
{
          char  name[10];
          char  address[20];
          char  post[10];
          struct  student
          {
          char  name[10];
          char  address[20];
          int  class;
          }  s;
}e;
```

For example:

```
#include  <stdio.h>
#include  <conio.h>
struct  employee
{
          char  name[20];
          float  salary;
 };
          struct  address
          {
          char  city[20];
          struct  employee  e;
          };
void  main()
{
struct  address a  =  {"Pokhara",  "Ram",  3500};
printf("\n  City  =  %s",  a.city);
printf("\n  Name  =  %s",  a.e.name);
printf("\n  Salary  =  %f",  a.e.salary);
getch();
}
```

Output:

```
City  =  Pokhara
Name  =  Ram
Salary  =  3500
```

5.6 Structure and pointer

When structure is used as a pointer variable then instead of the dot (.) operator used in the ordinary structure, the arrow (->) operator is used to access members defined inside structure.

For example-1:

```
#include  <stdio.h>
#include  <conio.h>
struct  book
{
char  name[25]  ;
char  author[25]  ;
int  pages  ;
}  ;
void  main(  )
{
struct  book  b1  =  {  "C-Programming",  "xyz",  101  }  ;
struct  book  *ptr  ;
ptr = &b1 ;
printf ( "\n%s %s %d", b1.name, b1.author, b1.pages ) ;
printf ( "\n%s %s %d", ptr->name, ptr->author, ptr->pages ) ;
getch();
clrscr();
}
```

Output:

```
C-Programming xyz 101
C-Programming xyz 101
```

For example-2:

```
#include <stdio.h>
#include <conio.h>
struct student
{
char name[20];
int cls;
float percent;
}st;
void main()
{
struct student *s;
s = &st;
printf("Enter name, class and percentage of a student:\n");
scanf("%s %d %f", s->name, &s->cls, &s->percent);
printf("\n Name = %s", s->name);
printf("\n Class = %d", s->cls);
printf("\n Percentage = %f", s->percent);
getch();
clrscr();
}
```

Output*:*

```
Enter name, class and percentage of a student:
Ram, 7, 89.7
Name = Ram
Class = 7
Percentage = 89.7
```

6. Unions

Both structures and unions are used to group a number of variables of different data type together, called a derived data type. The union follows the same syntax as structures but union differs from structure in storage and in initialization. The structure enables us to reserve a separate place in memory for every individual member of the structure. Whereas with union, it enables to reserve the definite place (memory space of a data type that has highest precision) in memory and use it for all members of the unions.

Example:

```
#include <stdio.h>
#include <conio.h>
union data
{
int a;
float b;
};
void   main()
{
union data x;
x.a=5;
printf("\nx.a=%d",x.a);
x.b=10.5;
printf("\nx.b=%f",x.b);
getch();
}
```

6.1 Similarities Between Structure and Union

- Both are derived data types.
- Both are used to group different data types together.
- The elements of both the structure and union can be accessed using dot(.) operator.

6.2 Difference Between Structure and Union

Structure	Union
It is declared using "struct" keyword	It is declared using "union" keyword
The elements of a structure occupy separate memory location i.e. takes large space.	The elements of a union share common memory location i.e. takes small space.

Syntax:
struct structure_name
{
datatype member1;
datatype member2;
 	
datatype membern;
};
	Syntax:
	union union_name
	{
	datatype member1;
	datatype member2;
	
	datatype membern;
	};

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 to store a union variable is equal to the size required by the largest member of the union.
All the members of the structure can be processed at a given time.	Only one member of the union can be processed at a time.
These are widely used in programming.	These are not widely used as much as structures.

7. Self-Referential Structures

Self-referential structure is the structure with pointer member that refer to the structure containing them. Such data structures are called dynamic data structure. Self-Referential Structures are one of the most useful features. They allow us to create data structures that contain references to data of the same type as themselves. Self-referential Structure is used in data structure such as binary tree, linked list, stack, Queue etc.

Syntax

```
struct  structure_name
{
Datatype  membername;
struct structure_name  *  pointer_name;
};
```

Example: Single linked list

```
#include<stdio.h>
#include<conio.h>
struct  list  {
int  data;
struct  list  *next;
};
void main()
{
struct  list  a,  b,  c;
a.data  =1;
b.data  =2;
c.data  =3;
printf("\na=%d",a.data);
a.next = &b;
b.next = &c;
printf("\nb=%d",a.next  ->  data);
printf("\nb=%d",b.next ->  data );
getch(();
clrscr();
}
```

With the given example, we can store two words in memory. The first word stores the member data and the second word stores the member next. The pointer variable next is called a link. Each structure is linked to a succeeding structure by way of the member next. The pointer variable next contains either an address of the location in memory of the successor list element, or the special value NULL defined as 0. NULL is used to denote the end of the list.