Unit 8
Structure and Union

Introduction : Structure

In some programming contexts, we need to access multiple data types under a single name for easier data manipulation; for example we want to store data about a book. We might want to store its name (a string), its price (a float) and number of pages in its (an int) C supports **structure** which allows us to wrap one or more variables with different data types. A structure can contain any valid data types like **int, char, float** even arrays or even other structures. Each variable in structure is called a structure member.

Structure is a collection of heterogeneous data types in a continuous memory location under a common name called structure tag or structure name. A structure is a collection of variables under a single name. These variables can be of different types, and each has a name which is used to select it from the structure. A structure is a convenient way of grouping several pieces of related information together. It can use other structures, arrays or pointers as some of its members, though this can get complicated unless you are careful.

Uses of structures:

The structures are very much used in database management. Besides this structures are used for a variety of purposes like:

- Changing the size of cursor
- Placing the cursor at the appropriate position on screen.
- Interacting with the mouse.
- Drawing any graphics shape on the screen.
- Changing the contents of the screen.
- Formatting a floppy.
- Hiding a file from the directory.
- Receiving key from the keyboard.
- Sending the output to the printer.
- Checking the memory size of the computer etc.

Declaring a structure:

A structure is a user-defined data type that serves to define its data properties. The keyword ***struct*** is used to declare a structure followed by an identifier, which is also known as *tag name*. The general syntax and example of a structure is defined in following section.

Syntax:

```
struct structure_name
{
data_type element1;
data_type element2;
……..……..
……..…….
……..
data_type elementn;
};
```

Example:

```
struct employee
{
char name[50];
int age,emp_id;
float salary;
};
```

In the above example, the name of structure is *employee*, which contains four members: ***name[50]*** as string (collection of 50 characters), ***age*** and ***emp_id*** as integer, and ***salary*** as float..

Declaration of structure variables:

In a structure the individual members can be of any valid data types of ordinary variables, arrays, pointers or even other structures. All the member names within a particular structure must be unique from each others. Once the new structure data has been defined, one or more variables can be declared to be of that type. Let us consider, the variables **e1, e2,** e3 can be declared to be of the type **struct employee.** The keyword ***struct*** is used to declare a structure and its variables or identifiers are declared by structure name separated by comma as:
struct employee e1, e2, e3;

Structure declarations with its variable declarations are described in following methods.
Method 1:

```
struct employee
{
char name[50];
int age,emp_id;
float salary;
};

struct employee e1, e2, e3;
```

Method 2:

```
struct employee
{
char name[50];
int age,emp_id;
float salary;
};e1, e2, e3;
```

Method 3:

```
struct
{
char name[50];
int age,emp_id;
float salary;
};e1, e2, e3;
```

All of above methods are same. We can use any one of above method for declaring structure variables.

Initialization of structure variables:

Like primary variables and arrays, structure variables can also be initialized where they are declared. The general syntax and initialization process of structure variable is illustrated as following:
Syntax:

```
struct structure_name
{
data_type element1;
data_type element2;
…….
……
data_type elementn;
};
struct structure_variables = {Elements};
```

Example:

```
struct employee
{
char name[50];
int age,emp_id;
float salary;
};
```

```
struct employee e1 = {"Krishna", 17, 12, 12000};
struct employee e2 = {"Srijana", 18, 13, 15000};
struct employee e3 = {"Prakash", 20, 14, 14500};
```

In above example, there are three structure variables: **e1, e2,** and **e3.** The variable makes available space to hold all the elements in the structure. Here each variable occupies **58 bytes** (50 bytes for *name*, 2 bytes for *age*, and 2 bytes for *emp_id* and 4 bytes for *salary)* and the structure *employee* occupy total **174 bytes** (58 bytes each variable). These bytes are always in adjacent memory locations. It is illustrated in following figure. Let the initial address of the memory location would be 1000.

Accessing Members of a Structure

Each member of a structure can be used just like a normal variable, but its name will be a bit longer. Structure can be accessed in two ways and they are:

a) Using normal structure variable (dot operator)
b) Using pointer variable (arrow operator)

Using dot operator:

To access structure members we can use **dot** operator (**.**) between structure name and structure member name as follows:

structure_variable_name.structure_member

Now above structure **employee**, if we want to access a member **age** of variable **e1**, we have to use as:

e1.age

Similarly, to refer ***salary*** of employee third (i.e. **e3),** would use:

e3.salary

How structure elements are stored:

```
#include<stdio.h>
#include<conio.h>
void main()
{
struct employee
   {
   int emp_id;
   char name[50];
   float salary;
   };

struct employee e1={100,"Bharat",20000.00};
clrscr();
printf("\n Address of Employee ID = %u",&e1.emp_id);
printf("\n Address of Name = %u",&e1.name);
printf("\n Address of Salary = %u",&e1.salary);
printf("\n Size of structure is %d bytes", sizeof(e1));
getch();
}
```

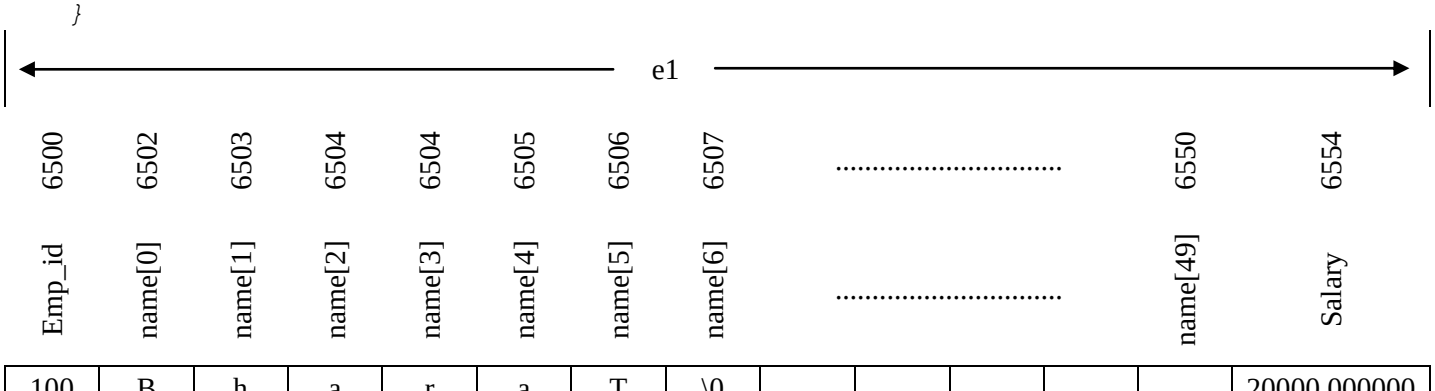


Fig: Memory Management

Nested type of Structure:

Sometimes a member of a structure needs to have multiple values of different types. For such situations the member should be variable of another structure type. C program provides such facility which is known as *nesting* of structure. In general term, if a structure encounter within another structure as a member, then such complex data structure is known as ***nested structure.*** Using this facility, we can create complex data types.

We can declare nested structure in two ways:

Alternate 1:

```
struct date
    {
    int date;
    int month;
    int year;
    };

struct Employee
    {
    char name[50];
    struct date d;
    }e;
```

Alternate 2:

```
struct Employee
    {
    char name[50];
    struct date
        {
        int date;
        int month;
        int year;
        }d;
    }e;
```

The following program illustrates the working mechanism of nested structure.

```
#include<stdio.h>
#include<conio.h>
void main()
{
struct address
    {
    int phone;
    char city[50];
    int street;
    };

struct employee
    {
    char name[50];
    struct address add;
    };

struct employee e1;
clrscr();

printf("\n Input Name:- ");
scanf("%s",&e1.name);
```

```
   printf("\n Input Phone No: - ");
   scanf("%d",&e1.add.phone);
   fflush(stdin);
   printf("\n Input City:- ");
   gets(e1.add.city);
   printf("\n Input Street:- ");
   scanf("%d",&e1.add.street);

   printf("\n %s\t %d\t%s\t%d",e1.name,e1.add.phone,e1.add.city,e1.add.street);
   getch();
   }
```

Output:

```
Input Name: - Ashok
Input Phone No: - 521312
Input City: - Kailali
Input Street: - 4

Ashok     521312    Kailali    4
```

Arrays of Structure:

Structure is used to store the information of a particular object but if we need to store such multiple objects then array of Structure is used. Array of structure means collection of multiple structure variables in a continuous memory location under a common name. The concept of arrays of structure is shown in following program.

```
#include<stdio.h>
#include<conio.h>
struct book
      {
      char name[50];
      int page, price;
      };

void main()
{
struct book b[50];
int i,n;
clrscr();
printf("\n How many books:- ");
scanf("%d", &n);
printf("\n Input information of %d books", n);
for(i=0;i<n;i++)
      {
      fflush(stdin);
      printf("\n Name of book:- ");
      gets(b[i].name);
      printf("\n Page:- ");
      scanf("%d", &b[i].page);
      printf("\n Price:- ");
      scanf("%d", &b[i].price);
      }
printf("\n Name \t Page \t Price");
for(i=0;i<n;i++)
      printf("\n %s \t %d \t %d", b[i].name,b[i].page,b[i].price);
getch();
}
```

Structure and Pointers:

A structure's member can be accessed through pointer in two ways:

- Referencing pointer to another address to access memory
- Using dynamic memory allocation

a) Referencing pointer to another address to access memory

```
#include<stdio.h>
#include<conio.h>
struct book
   {
   char name[50];
   int page;
   };

void main()
{
struct book *ptr;
clrscr();
printf("\n Name of book:- ");
gets(ptr->name);
printf("\n Page:- ");
scanf("%d", &ptr->page);
printf("\n ------ Book Details ------");
printf("\n Name :- %s",ptr->name);
printf("\n Page :- %d",ptr->page);
getch();
}
```

b) Using dynamic memory allocation

To access structure member using pointers, memory can be allocated dynamically using *mallo()*. It can be demonstrated in following program.

```
#include<stdio.h>
#include<conio.h>
struct book
   {
   char name[50],author[50];
   int page;
   };

void main()
{
struct book b, *ptr;
int i,n;
clrscr();
printf("\n How many books:- ");
scanf("%d" ,&n);
ptr = (struct book*) malloc(n*sizeof(struct book));
for(i=0;i<n;i++)
   {
   fflush(stdin);
   printf("\n Name of book:- ");
   gets((ptr+i)->name);
   fflush(stdin);
   printf("\n Author Name:- ");
   gets((ptr+i)->author);
   printf("\n Page:- ");
   scanf("%d", &(ptr+i)->page);
   }
```

```
printf("\n ------- Book Details ---------");
for(i=0;i<n;i++)
   {
   printf("\n Name   :- %s",(ptr+i)->name);
   printf("\n Author :- %s",(ptr+i)->author);
   printf("\n Page   :- %d",(ptr+i)->page);
   }
getch();
}
```

Structure and Function:

A structure can be passed as a function argument just like any other variable. There are two methods through we can pass structure within a function and they are:

- Individual structure elements at a time
- Entire structure variables at one go.

Both approaches are illustrated in following programs.

a) Individual structure elements at a time

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

struct book
   {
   char name[50],author[50];
   int page;
   };

void main()
{
struct book b;
int i,n;
clrscr();

fflush(stdin);
printf("\n Name of book:- ");
gets(b.name);
fflush(stdin);
printf("\n Author Name:- ");
gets(b.author);
printf("\n Page:- ");
scanf("%d", &b.page);
display(b.name,b.author,b.page);
getch();
}

display(char *n, char *a, int p)
{
printf("\n ------- Book Details ---------");
printf("\n Name   :- %s",n);
printf("\n Author :- %s",a);
printf("\n Page   :- %d",p);
}
```

b) Entire structure variables at one go

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

```
struct book
   {
   char name[50],author[50];
   int page;
   };

void main()
{
struct book b;
int i,n;
clrscr();

fflush(stdin);
printf("\n Name of book:- ");
gets(b.name);
fflush(stdin);
printf("\n Author Name:- ");
gets(b.author);
printf("\n Page:- ");
scanf("%d", &b.page);

display(b);
getch();
}

display(struct book b)
{
printf("\n ------- Book Details ---------");
printf("\n Name   :- %s", b.name);
printf("\n Author :- %s", b.author);
printf("\n Page   :- %d", b.page);
}
```

Passing address of a structure to a function:

We can also pass the address of a structure variable to a function. It is demonstrated in following program.

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

struct book
      {
      char name[50],author[50];
      int page;
      };

void main()
{
struct book b;
int i,n;
clrscr();

fflush(stdin);
printf("\n Name of book:- ");
gets(b.name);
fflush(stdin);
printf("\n Author Name:- ");
gets(b.author);
printf("\n Page:- ");
scanf("%d", &b.page);
```

```
display(&b);
getch();
}

display(struct book *b)
{
printf("\n ------- Book Details ---------");
printf("\n Name   :- %s", b->name);
printf("\n Author :- %s", b->author);
printf("\n Page   :- %d", b->page);
}
```

Introduction: Union

Unions are the concept borrowed from structure and therefore are almost similar to the structures and follows same syntax as structure. Like structure the keyword ***union*** is used to declare union. It is described in following section.

Syntax:

```
union union_name
   {
   data_type element1;
   data_type element2;
   ……..……..
   ……..
   ……..
   data_type elementn;
   }varialbes;
```

Example:

```
union employee
   {
   char name;
   int age;
   float salary;
   }uv;
```

This declares a variable **uv** of type union **employee**. The union contains three members with different data type. However, we can use only of them at a time because; only one location is allocated for a union variable.

The compiler allocates a piece of storage that is large enough to hold the largest variable type in the union. In above declaration, the member *salary* requires 4 bytes which the largest among the members. Above figure shows how all three variables share the same memory address.

Difference between structure and union:

The syntax used in the structure and union are similar but there is major distinction between them in terms of storage and access on the members. It is described in following section with appropriate programs.

Memory allocation: In structures each members has its own storage location, where as all the members of a union use the same location. The size of a structure depends upon the number of structure variables and the members i.e. the size of the structure is equivalent to the sum of individual members multiplied by the number of structure variables. It requires a lot of memory spaces. The size of union doesn't depend upon the number of union variables. It is equivalent to the size of a largest union member of a union. So it requires less memory space. It is illustrated in following program.

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

```
#include <conio.h>
void main()
{
struct stud_record
   {
   char name[50];
   int roll;
   float fee;
   }sv;

union stud_record1
   {
   char name[50];
   int roll;
   float fee;
   }uv;
clrscr();
printf("\n Size occupied by a structure variable is %d bytes", sizeof(sv));
printf("\n Size occupied by a union variable is %d bytes", sizeof(uv));

getch();
}
```

Output:

```
Size occupied by a structure variable is 56 bytes
Size occupied by a union variable is 50 bytes
```

Operation on members: Multiple members can be accessed at given time in structure where as union can handle only one member at a time. This mechanism can be illustrated in the following program.

```
#include <stdio.h>
#include <conio.h>
struct stud_record
   {
   char name[50];
   int roll;
   long int fee;
   };

union stud_record1
   {
   char name[50];
   int roll;
   long int fee;
   };

int main()
{
struct stud_record s;
union stud_record1 u;
printf("\n Input Name, Roll number and Fee:- ");
gets(s.name);
scanf("%d%ld",&s.roll,&s.fee);
printf("\n Name:- %s, Roll No:- %d, Fee:- %ld", s.name, s.roll, s.fee);
fflush(stdin);
printf("\n Input Name:- ");
gets(u.name);
printf("\n Name:- %s", u.name);
printf("\n Roll No:- ");
```

```
scanf("%d", &u.roll);
printf("\n Roll No:- %d", u.roll);
printf("\n Fee:- ");
scanf("%ld",&u.fee);
printf("\n Fee:- %ld", u.fee);
getch();
}
```

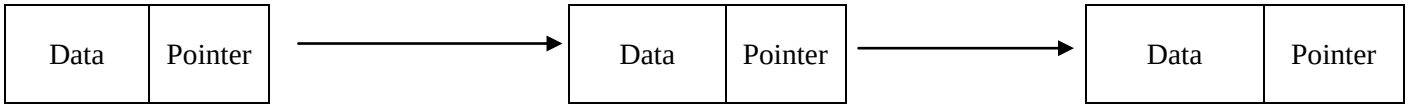
Hence the union members are carefully handled.

Self-referential Structure:

It is essentially a structure definition which includes at least one member that is a pointer to the structure of same type. It is used to create data structure like linked lists, stacks, etc. A chain of such structures can thus be expressed as follows.

```
struct structure_tag
    {
    data_type member 1;
    data_type member 2;
    ……….
    data_type member N;
    struct structure_name *pointer;
    };
```

It can be illustrated in following figure



Node 1 Node 2 Node 3

A node consists of two logical segments: data / information and pointer.

Structure Related Programs

1. WAP to read name, roll number and marks of a student and display those information's using structure.

```
#include<stdio.h>
#include<conio.h>
struct student
{
char name[30];
int roll, marks;
};

void main()
{
struct student s;
clrscr();
printf("\n Input following information:- ");
printf("\n Name:- ");
gets(s.name);
printf("\n Roll Number:- ");
scanf("%d", &s.roll);
printf("\n Total Marks:- ");
scanf("%d", &s.marks);
printf("\n The information is:");
```

```
printf("\n Name = %s\t Roll No. = %d\t Total Marks = %d", s.name, s.roll,
s.marks);
getch();
}
```

2. **WAP to read name, roll number and marks of 'n' students and display all those information's using structure.**

```
#include<stdio.h>
#include<conio.h>
struct student
{
char name[30];
int roll, marks;
};

void main()
{
struct student s[50];
int i,n;
clrscr();
printf("\n How many Students:- ");
scanf("%d", &n);
printf("\n Input following information of %d Students:- ",n);
for(i=0;i<n;i++)
   {
   fflush(stdin);
   printf("\n Name:- ");
   gets(s[i].name);
   printf("\n Roll Number:- ");
   scanf("%d", &s[i].roll);
   printf("\n Total Marks:- ");
   scanf("%d", &s[i].marks);
   }
printf("\n Name \t Roll No. \t Total Marks");
for(i=0;i<n;i++)
   printf("\n %s \t %d \t %d", s[i].name, s[i].roll, s[i].marks);
getch();
}
```

3. **Write a program to input name and address of 'n' number of students and sort them by name using structure variable.**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct student
   {
   char name[50];
   char address[50];
   };
struct student s[100];

void main()
{
int i, j, n;
char temp[50];
clrscr();
printf("\n How many students:- ");
scanf("%d", &n);
```

```
printf("\n Input name and address of %d students:- ", n);
for(i=0;i<n;i++)
   {
   fflush(stdin);
   printf("\n Name[%d]:- ",i+1);
   gets(s[i].name);
   printf("\n Address[%d]:- ", i+1);
   gets(s[i].address);
   }
for(i=0;i<n;i++)
   {
   for(j=i+1;j<n;j++)
         {
         if(strcmpi(s[i].name,s[j].name)>0)
               {
               strcpy(temp, s[i].name);
               strcpy(s[i].name, s[j].name);
               strcpy(s[j].name, temp);
               strcpy(temp, s[i].address);
               strcpy(s[i].address, s[j].address);
               strcpy(s[j].a ddress, temp);
               }
         }
   }
printf("\n Sorted List:-\n");
printf("\n Name\t Address\n");
for(i=0;i<n;i++)
   printf("\n%s \t %s", s[i].name, s[i].address);
getch();
}
```

4. In a bank there are 'n' customers with attributes name, account number and balance. Write a program to find the information of the customer who has highest balance using structure.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct customer
   {
   char name[50];
   int acc_no, balance;
   };
struct customer c[100];

void main()
{
int i, n, max;
clrscr();
printf("\n How many customers:- ");
scanf("%d", &n);
printf("\n Input following information of %d customers:- ", n);
for(i=0;i<n;i++)
   {
   fflush(stdin);
   printf("\n Name[%d]:- ",i+1);
   gets(c[i].name);
   printf("\n Account number[%d]:- ", i+1);
   scanf("%d", &c[i].acc_no);
   printf("\n Balance[%d]:- ", i+1);
```

```
   scanf("%d", &c[i].balance);
   }

max=c[0].balance;
for(i=0;i<n;i++)
   {
   if(c[i].balance>max)
         max = c[i].balance;
   }

for(i=0;i<n;i++)
   {
   if(c[i].balance==max)
         printf("\n%s\t%d\t%d",c[i].name,c[i].acc_no, c[i].balance);
   }
getch();
}
```

5. Write a program to input the following records of any 50 employees using structure and display them properly.

Name	Address	Post	Salary	Date of Appointment		
				Month	Day	Year

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

struct employee
   {
   char name[30], address[40],post[25];
   long int salary;
   char doj[25];
   };

void main()
{
int i;
struct employee e[50];
clrscr();
printf("\n Input following information's of 50 employees:- ");
for(i=0;i<50;i++)
   {
   fflush(stdin);
   printf("\n Name:- ");
   gets(e[i].name);
   fflush(stdin);
   printf("\n Address:- ");
   gets(e[i].address);
   printf("\n Post:- ");
   gets(e[i].post);
   printf("\n Salary:- ");
   scanf("%ld",&e[i].salary);
   fflush(stdin);
   printf("\n Date of Join (mm/dd/yy):- ");
   gets(e[i].doj);
   }
clrscr();
```

```
printf("\n The Information's of 50 employees are:- \n");
printf("\n Name \t Address \t Post \t Salary \t Date of Join");
for(i=0;i<50;i++)
   printf("\n%s  \t  %s  \t  %s  \t  %ld  \t  %s",  e[i].name,  e[i].address,
   e[i].post, e[i].salary, e[i].doj);
getch();
}
```

6. Write a program to read subject name and marks obtained in each subject and display those information's and total marks obtained using structure dynamically.

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

struct student
   {
   char subject[40];
   int marks;
   };

void main()
{
int i, n, total=0;
struct student *ptr;
clrscr();
printf("\n How many subjects:- ");
scanf("%d", &n);
ptr = (struct student*)malloc(n*sizeof(struct student));
printf("\n Input Subject Name and Marks obtained:- ");
for(i=0;i<n;i++)
   {
   fflush(stdin);
   printf("\n Suject Name:- ");
   gets((ptr+i)->subject);
   printf("\n Marks Obtained:- ");
   scanf("%d",&(ptr+i)->marks);
   total+=(ptr+i)->marks;
   }
printf("\n Subject Name \t Marks");
for(i=0;i<n;i++)
   printf("\n %s \t %d",(ptr+i)->subject,(ptr+i)->marks);
printf("\n Total Marks:- %5d",total);
getch();
}
```