```
        Enter String
        welcome
        Uppercase String :    WELCOME
```

STRUCTURES

Arrays are used for storing a group of SIMILAR data items. In order to store a group of data items, we need structures. Structure is a constructed data type for packing different types of data that are logically related. The structure is analogous to the "record" of a database. Structures are used for organizing complex data in a simple and meaningful way.

Example for structures:

Student	:	regno, student_name, age, address
Book	:	bookid, bookname, author, price, edition, publisher, year
Employee	:	employeeid, employee_name, age, sex, dateofbirth, basicpay
Customer	:	custid, cust_name, cust_address, cust_phone

Structure Definition

Structures are defined first and then it is used for declaring structure variables. Let us see how to define a structure using simple example given below:

```
struct book

{

        int bookid;

        char bookname[20];

        char author[20];

        float price;

        int year;

        int  pages;

        char publisher[25];

};
```

The keyword “struct” is used for declaring a structure. In this example, book is the name of the structure or the structure tag that is defined by the struct keyword. The book structure has six fields and they are known as structure elements or structure members. Remember each structure member may be of a different data type. The structure tag name or the structure name can be used to declare variables of the structure data type.

The syntax for structure definition is given below:

```
struct tagname

{

     Data_type  member1;

     Data_type  member2;

     …………….

     ……………

};
```

Note:

1. To mark the completion of the template, semicolon is used at the end of the template.
2. Each structure member is declared in a separate line.

Declaring Structure Variables

First, the structure format is defined. Then the variables can be declared of that structure type. A structure can be declared in the same way as the variables are declared. There are two ways for declaring a structure variable.

1) Declaration of structure variable at the time of defining the structure (i.e structure definition and structure variable declaration are combined)

```
struct book
{
           int bookid;
           char bookname[20];
           char author[20];
           float price;
           int year;
           int  pages;
           char publisher[25];
}  b1,b2,b3;
```

The b1, b2, and b3 are structure variables of type struct book.

2) Declaration of structure variable after defining the structure

```
struct book
{
          int bookid;
          char bookname[20];
          char author[20];
          float price;
          int year;
          int pages;
          char publisher[25];
};

struct book b1, b2, b3;
```

NOTE:

- Structure tag name is optional.

```
E.g.

   struct
   {
   int bookid;
   char bookname[20];
   char author[20];
   float price;
   int year;
   int pages;
   char publisher[25];
   }b1, b2, b3;
```

Declaration of structure variable at a later time is not possible with this type of declaration. It is a drawback in this method. So the second method can be preferred.

- Structure members are not variables. They don't occupy memory until they are associated with a structure variable.

Accessing Structure Members

There are many ways for storing values into structure variables. The members of a structure can be accessed using a “dot operator” or “period operator”.

E.g. `b1.author` -> b1 is a structure variable and author is a structure member.

Syntax

STRUCTURE_Variable.STRUCTURE_Members

The different ways for storing values into structure variable is given below:

Method 1: Using Simple Assignment Statement

```
b1.pages = 786;

b1.price = 786.50;
```

Method 2: Using strcpy function

```
strcpy(b1.title, “Programming in C”);

strcpy(b1.author, “John”);
```

Method 3: Using scanf function

```
scanf(“%s \n”, b1.title);

scanf(“%d \n”, &b1.pages);
```

Example

```
#include<stdio.h>

#include<conio.h>

struct book

      {

      int bookid;
      char bookname[20];
      char author[20];
      float price;
      int year;
      int pages;
      char publisher[25];
```

```
        };
        struct book b1, b2, b3;
    main()
        {
        struct book b1;
        clrscr();
        printf("Enter the Book Id:  ");
        scanf("%d", &b1.bookid);
        printf("Enter the Book Name:  ");
        scanf("%s", b1.bookname);
        printf("Enter the Author Name:  ");
        scanf("%s", b1.author);
        printf("Enter the Price:  ");
        scanf("%f", &b1.price);
        printf("Enter the Year:  ");
        scanf("%d", &b1.year);
        printf("Enter the Total No. of Pages:  ");
        scanf("%d", &b1.pages);
        printf("Enter the Publisher Name:  ");
        scanf("%s", b1.publisher);
        printf("%d %s %d %f %d %d %s", b1.bookid, b1.bookname,
      b1.author, b1.price, b1.year, b1.pages, b1.publisher);
        getch();
        }
```

Output

```
Enter the Book Id:  786
Enter the Book Name:  Programming
Enter the Author Name:  John
Enter the Price:  123.50
Enter the Year:  2015
Enter the Total No. of Pages:  649
Enter the Publisher Name:  Tata McGraw
786   Programming     2118  123.500000      2015  649   Tata
```

Structure Initialization

Like variables, structures can also be initialized at the compile time.

Example

```
main()
   {
     struct
     {
          int  rollno;
          int attendance;
     }
   s1={786, 98};
   }
```

The above example assigns 786 to the rollno and 98 to the attendance.

Structure variable can be initialized outside the function also.

Example

```
   main()
   {
     struct student
     {
          int  rollno;
          int attendance;
     };
     struct student s1={786, 98};
     struct student s2={123, 97};
   }
```

Note:

Individual structure members cannot be initialized within the template. Initialization is possible only with the declaration of structure members.

Nested Structures or Structures within Structures

Structures can also be nested. i.e A structure can be defined inside another structure.

Example

```
   struct  employee
   {
     int empid;
     char empname[20];
```

```
    int basicpay;
    int da;
    int hra;
    int cca;
  } e1;
```

In the above structure, salary details can be grouped together and defined as a separate structure.

Example

```
  struct  employee
  {
    int empid;
    char empname[20];
               struct
    {
         int basicpay;
         int da;
         int hra;
         int cca;
    } salary;
  } e1;
```

The structure employee contains a member named salary which itself is another structure that contains four structure members. The members inside salary structure can be referred as below:

```
    e1.salary.basicpay
    e1.salary.da;
    e1.salary.hra;
    e1.salary.cca;
```

However, the inner structure member cannot be accessed without the inner structure variable.

Example

```
    e1.basicpay
    e1.da
    e1.hra
    e1.cca
```

are invalid statements

Moreover, when the inner structure variable is used, it must refer to its inner structure member. If it doesn't refer to the inner structure member then it will be considered as an error.

Example

`e1.salary` (salary is not referring to any inner structure member. Hence it is wrong)

Note: C permits 15 levels of nesting and C99 permits 63 levels of nesting.

Array of Structures

A Structure variable can hold information of one particular record. For example, single record of student or employee. Suppose, if multiple records are to be maintained, it is impractical to create multiple structure variables. It is like the relationship between a variable and an array. Why do we go for an array? Because we don't want to declare multiple variables and it is practically impossible. Assume that you want to store 1000 values. Do you declare 1000 variables like a1, a2, a3…. Upto a1000? Is it easy to maintain such code ? Is it a good coding? No. It is not. Therefore, we go for Arrays. With a single name, with a single variable, we can store 1000 values. Similarly, to store 1000 records, we cannot declare 1000 structure variables. But we need "Array of Structures".

An array of structure is a group of structure elements under the same structure variables.

```
struct student s1[1000];
```

The above code creates 1000 elements of structure type student. Each element will be structure data type called student. The values can be stored into the array of structures as follows:

```
s1[0].student_age = 19;
```

Example

```
#include<stdio.h>
#include<conio.h>
struct book
{
        int bookid;
        char bookname[20];
        char author[20];
};


Struct b1[5];

main()
{
int i;
clrscr();
for (i=0;i<5;i++)
{
printf("Enter the Book Id:  ");
scanf("%d", &b1[i].bookid);
printf("Enter the Book Name:  ");
scanf("%s", b1[i].bookname);
printf("Enter the Author Name:  ");
scanf("%s", b1[i].author);
}
for (i=0;i<5;i++)
{
printf("%d \t %s \t %s \n", b1[i].bookid, b1[i].bookname,
b1[i].author);
}
getch();
}
```

Output:

```
Enter the Book Id:  786
Enter the Book Name:  Programming
Enter the Author Name:  Dennis Ritchie
Enter the Book Id:  101
Enter the Book Name:  Java Complete Reference
Enter the Author Name:   Herbert Schildt
Enter the Book Id:  008
Enter the Book Name:  Computer Graphics
Enter the Author Name:   Hearn and Baker


786   Programming   Dennis Ritchie
101   Java Complete Reference   Herbert Schildt

008   Computer Graphics   Hearn and Baker
```

Structure as Function Argument

Example

```
struct sample
{
      int no;
      float avg;
} a;
void main( )
{
      a.no=75;
      a.avg=90.25;
      fun(a);
}

void fun(struct sample p)
{
      printf("The no is=%d   Average is %f",p.no , p.avg);
}
```

Output

```
The no is 75    Average is 90.25
```

Function that returns Structure

The members of a structure can be passed to a function. If a structure is to be passed to a called function , we can use any one of the following method.

Method 1 :- Individual member of the structure is passed as an actual argument of the function call. The actual arguments are treated independently. This method is not suitable if a structure is very large structure.

Method 2:- Entire structure is passed to the called function. Since the structure declared as the argument of the function, it is local to the function only. The members are valid for the