Unit V - Structure : LEARNING PLAN

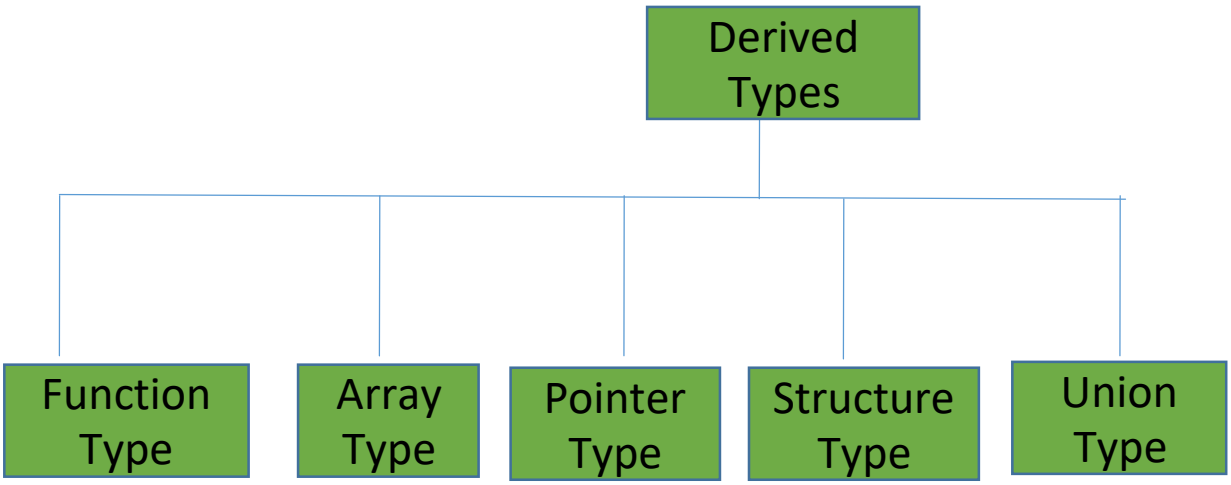
5.0 Introduction to structures–

C Data Types:

Primary data types

Derived data types

User-defined data types



Array – Collection of one or more related variables of similar data type grouped under a single name

Structure – Collection of one or more related variables of different data types, grouped under a single name

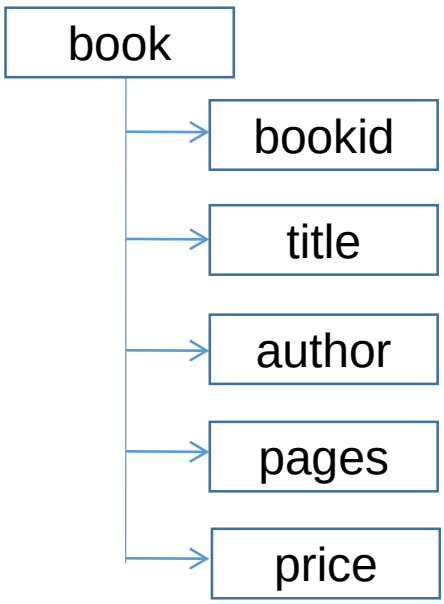
Need of structures

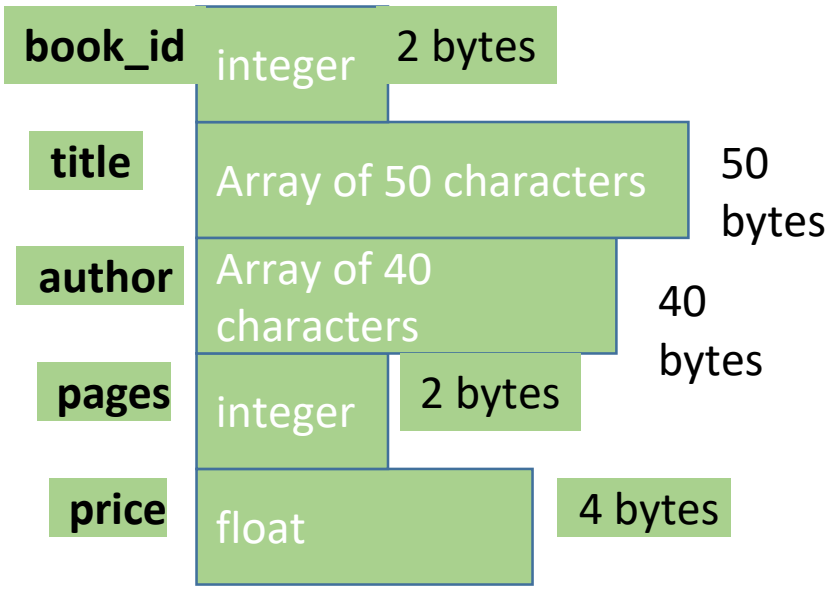
In a Library, each book is an **object**, and its **characteristics** like title, author, no of pages, price are grouped and represented by one **record**.

The characteristics are different types and grouped under a aggregate variable of different types.

A **record** is group of **fields** and each field represents one characteristic. In C, a record is implemented with a derived data type called **structure**. The characteristics of record are called the **members** of the structure.

Book-1	Book-2	Book-3
BookID: 1211 Title : C Primer Plus Author : Stephen Prata Pages : 984 Price : Rs. 585.00	BookID: 1212 Title : The ANSI C Progg. Author : Dennis Ritchie Pages : 214 Price : Rs. 125.00	BookID: 1213 Title : C By Example Author : Greg Perry Pages : 498 Price : Rs. 305.00





Memory occupied by a Structure variable

STRUCTURE- BOOK

```
struct   book  {
        int book_id ;
        char title[50] ;
        char author[40] ;
        int pages ;
        float price ;
};
```

Structure tag

- A ***Structure*** is defined to be a collection of different data items, that are stored under a common name.
- A structure is same as that of records. It stores related information about an entity. Structure is basically a user defined data type that can store related information (even of different data types) together.

Declaration of structures

- A structure is declared using the keyword struct followed by a structure name. All the variables of the structures are declared within the structure. A structure type is defined by using the given syntax.
- By declaring a stucture type

```
struct struct-name {
        data_type var-name;
        data_type var-name;
        ...};
```

By declaring a structure variable

```
struct stru-name  Sv1,Sv2,Sv3;
```

(or)

```
struct stru-name {
     data_type var-name;
     data_type var-name;
} sv1, sv2 , sv3;
```

Example :

```
struct student {
        int r_no;
        char name[20];
        char course[20];
        float fees; };
```

The structure definition does not allocates any memory. It just gives a template that conveys to the C compiler how the structure is laid out in memory and gives details of the member names. Memory is allocated for the structure when we declare a variable of the structure. For ex., we can define a variable of student by writing as :

```
struct student stud1;
```

Here, struct student is a data type and stud1 is a variable. Look at another way of declaring variables. In the following syntax, the variables are declared at the time of structure declaration.

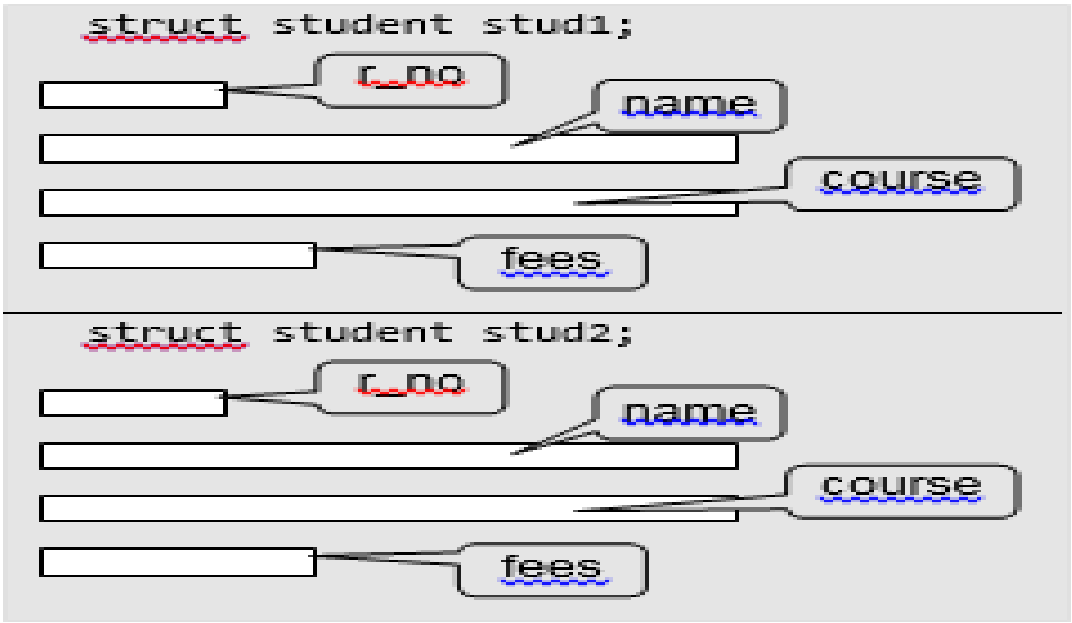
struct student{

int r_no;

char name[20]; char course[20]; float fees;

} stud1, stud2;

In this declaration we declare two variables stud1 and stud2 of the structure student. So if you want to declare more than one variable of the structure, then separate the variables using a comma. When we declare variables of the structure, separate memory is allocated for each variable. This is shown in Fig.



last but not the least, structure member names and names of the structure follow the same rules as laid down for the names of ordinary variables. However, care should be taken to ensure that the name of structure and the name of a structure member should not be the same. Moreover, structure name and its variable name should also be different.

Note: Structure type and variable declaration of a structure can be either local or global depending on their placement in the code.

Type def declarations

The typedef (derived from type definition) keyword enables the programmer to create a new data type name by using an existing data type. By using typedef, no new data is created, rather an alternate name is given to a known data type. The general syntax of using the typedef keyword is given as:

typedef existing_data_type new_data_type;

Note that typedef statement does not occupy any memory; it simply defines a new type. For example, if we write

typedef int INTEGER;

then INTEGER is the new name of data type int. To declare variables using the new data type name, precede the variable name with the data

type name (new). Therefore, to define an integer variable, we may now write

INTEGER num=5;

When we precede a struct name with typedef keyword, then the struct becomes a new type. It is used to make the construct shorter with more meaningful names for types already defined by C or for types that you have declared. With a typedef declaration, becomes a synonym for the type.

For example, writing

```
typedef struct student{
        int r_no;
        char name[20];
        char course[20];
        float fees;};
```

Now that you have preceded the structure's name with the keyword typedef, the student becomes a new data type. Therefore, now you can straight away declare variables of this new data type as you declare variables of type int, float, char, double, etc. to declare a variable of structure student you will just write,

student stud1;

Note that we have not written struct student stud1.

NOTE: Do not forget to place a semicolon after the declaration of structures and unions.

Accessing the 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. A structure member variable is generally accessed using a '.' (dot operator).

The syntax of accessing a structure a member of a structure is:

struct_var.member_name

stud1.r_no

The dot operator is used to select a particular member of the structure. For example, to assign values to the individual data members of the structure variable studl, we may write

```
stud1.r_no = 01;
stud1.name = "Rahul";
stud1.course = "BCA";
stud1.fees = 45000;
```

To input values for data members of the structure variable stud1, we may write

```
scanf("%d", &stud1.r_no);
scanf("%s", stud1.name);
```

Similarly, to print the values of structure variable stud1, we may write

```
printf("%s", stud1.course);
printf("%f", stud1.fees);
```

Memory is allocated only when we declare the variables of the structure. In other words, the memory is allocated only when we instantiate the structure. In the absence of any variable, structure definition is just a template that will be used to reserve memory when a variable of type struct is declared.

Once the variables of a structure are defined, we can perform a few operations on them. For example, we can use the assignment operator (=) to assign the values of one variable to another.

NOTE: Of all the operators –>, . , ( ), and [] have the highest priority. This is evident from the following statement

stud1.fees++ will be interpreted as (stud1.fees)++.

Initialization of structures

- Initializing a structure means assigning some constants to the members of the structure.
- When the user does not explicitly initializes the structure then C automatically does that. For int and float members, the values are initialized to zero and char and string members are initialized to the '\0' by default.
- The initializers are enclosed in braces and are separated by commas. Note that initializers match their corresponding types in the structure definition.
- The general syntax to initialize a structure variable is given as follows.

```
struct struct_name
{     data_type member_name1;
      data_type member_name2;
      .....
}struct_var = {constant1, constant2, constant 3,...};
OR
struct struct_name
{     data_type member_name1;
      data_type member_name2;
      .....
};
struct struct_name struct_var = {constant1, constant2, ….};
```

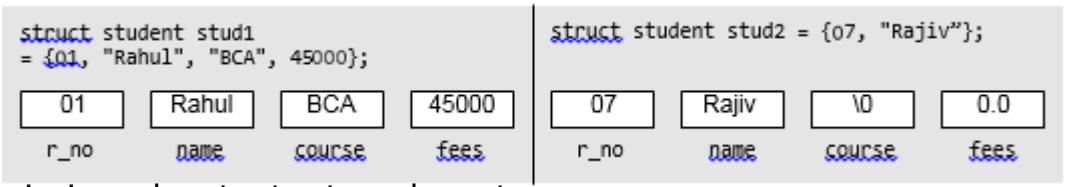
For example, we can initialize a student structure by writing,

```
struct student
{int r_no;
char name[20]; char course[20]; float fees;
}stud1 = {01, "Rahul", "BCA", 45000};
```

Or, by writing,

```
struct student stud1 = {01, "Rahul", "BCA", 45000};
```

Figure illustrates how the values will be assigned to individual fields of the structure.



Assigning values to structure elements

When all the members of a structure are not initialized, it is called partial initialization. In case of partial initialization, first few members of the structure are initialized and those that are uninitialized are assigned default values

To Initialize or assign of structure variable **while declaration**

```
struct student stud1= {01, "Rahul", "BCA", 45000} ;
```

To initialize or assign value **to the individual data members** of the structure variable Rahul, we may write,

```
stud1.r_no = 01;
strcpy(stud1.name, "Rahul");
stud1.course = "BCA";
stud1.fees = 45000;
```

Reading **values to members at runtime:**

```
struct student stud3;
printf("\nEnter the roll no");
scanf("%d",&stud3.r_no);
printf("\nEnter the name");
scanf("%s", stud3.name);
printf("\nEnter the course");
scanf("%s", stud3.course);
printf("\nEnter the fees");
scanf("%d",&stud3.fees);
```

We can initialize / assign a structure to another structure of the same type. For ex, if we have two structure variables stu1 and stud2 of type struct student given as

```
struct student stud1 = {01, "Rahul", "BCA", 45000};
struct student stud2;
```

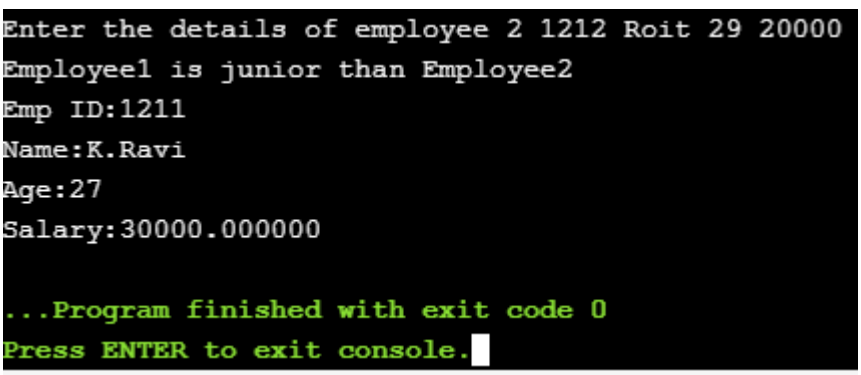
Then to assign one structure variable to another we will write,

```
stud2 = stud1;
```

Example Program 1: Write a program using structures to read and display the information about a student

```
#include <stdio.h>
#include <string.h>
struct employee {
        int empid;
        char name[35];
        int age;
        float salary;};
int main() {
    struct employee emp1 ;
    printf("Enter the details of employee 1 : ");
    scanf("%d %s %d %f" , &emp1.empid, emp1.name, &emp1.age, &emp1.salary);
    printf("Emp ID:%d\nName:%s\n Age:%d\n Salary:%f",emp1.empid,
    emp1.name, emp1.age,emp1.salary);}
```

Output :



Example program 2: Write a program using structures to read student 3 marks and display the total and average of the student.

```
#include<stdio.h>
#include<conio.h>
struct stud
  {
        int regno;
        char name[10];
        int m1;
        int m2;
        int m3;
  };
struct stud s;
void main() {
  float tot,avg;
  printf("\nEnter the student regno,name,m1,m2,m3:");
  scanf("%d%s%d%d%d",&s.regno,&s.name,&s.m1,&s.m2,&s.m3);
  tot=s.m1+s.m2+s.m3;
  avg=tot/3;
  printf("\nThe student Details are:");
  printf("\n%d\t%s\t%f\t%f",s.regno,s.name,tot,avg);
}
```

Output :

```
Enter the student regno,name,m1,m2,m3:100
aaa
87
98
78

The student Details are:
100     aaa     263.000000      87.666664
```

GUIDED ACTIVITY – Here is the guided activity for you on (**Implementing a Structure – declaration, initialization, accessing for an employee DB**)

```
#include <stdio.h>
#include <string.h>
struct employee {
      int empid;
      char name[35];
      int age;
      float salary;
};
int main() {
     struct employee emp1,emp2 ;

     struct employee emp3 = { 1213 , "S.Murali" , 31 , 32000.00 } ;
     emp1.empid=1211;
     strcpy(emp1.name, "K.Ravi");
     emp1.age = 27;
     emp1.salary=30000.00;
     printf("Enter the details of employee 2");
     scanf("%d %s %d %f" , &emp2.empid, emp2.name,
   &emp2.salary);
     if(emp1.age > emp2.age)
           printf("Employee1 is senior than Employee2\n" );
     else
           printf("Employee1 is junior than Employee2\n");

     printf("Emp ID:%d\n Name:%s\nAge:%d\n Salary:%f",
   emp1.empid,emp1.name,emp1.age,emp1.salary);
}
```

Declaration of Structure Type

Declaration of Structure variables

Declaration and initialization of Structure variable

Initialization of Structure members individually

Reading values to members of Structure

Accessing members of Structure

Output:

```
Enter the details of employee 2 1212 Roit 29 20000
Employee1 is junior than Employee2
Emp ID:1211
Name:K.Ravi
Age:27
Salary:30000.000000

...Program finished with exit code 0
Press ENTER to exit console.
```

Copying and Comparing Structures

We can assign a structure to another structure of the same type. For example, if we have two structure variables stud1 and stud2 of type struct student given as

struct student stud1 = {01, "Rahul", "BCA", 45000};

struct student stud2;

Then to assign one structure variable to another, we will write

stud2 = stud1;

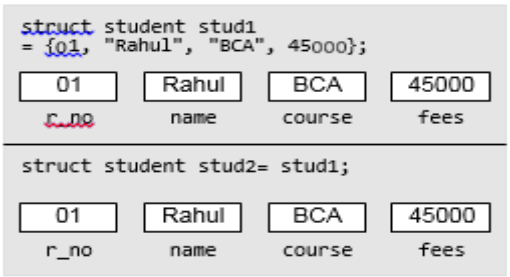


Figure Values of structure variables

This statement initializes the members of stud2 with the values of members of stud1. Therefore, now the values of stud1 and stud2 can be given as shown in Fig.

C does not permit comparison of one structure variable with another. However, individual members of one structure can be compared with individual members of another structure. When we compare one structure member with another structure's member, the comparison will behave like any other ordinary variable comparison.

For example, to compare the fees of two students, we will write

if(stud1.fees > stud2.fees) //to check if fees of stud1 is greater than stud2

Note: An error will be generated if you try to compare two structure variables.

Nested Structures – Array of Structures – Structures and functions – Passing an entire structure –Passing Structures Through Pointers, Self referential structure

Topic. 5.1, 5.2, 5.3, 5.4

5.1 NESTED STRUCTURES

A structure can be placed within another structure. That is, a structure may contain another structure as its member. Such a structure that contains another structure as its member is called a nested structure.

Let us now see how we declare nested structures. Although it is possible to declare a nested structure with one declaration, it is not recommended. The easier and clearer way isto declare the structures separately and then group them in the higher level structure. When you do this, take care to check that nesting must be done from inside out (from lowest level to the most inclusive level), i.e., declare the innermost structure, then the next level structure, working towards the outer (most inclusive) structure.

```
typedef struct {
char first_name[20];
char mid_name[20];
char last_name[20];
} NAME;
typedef struct {
int dd;
int mm;
int yy;
} DATE;
typedef struct {
int r_no;
NAME name;
char course[20];
DATE DOB;
float fees;
} student;
```

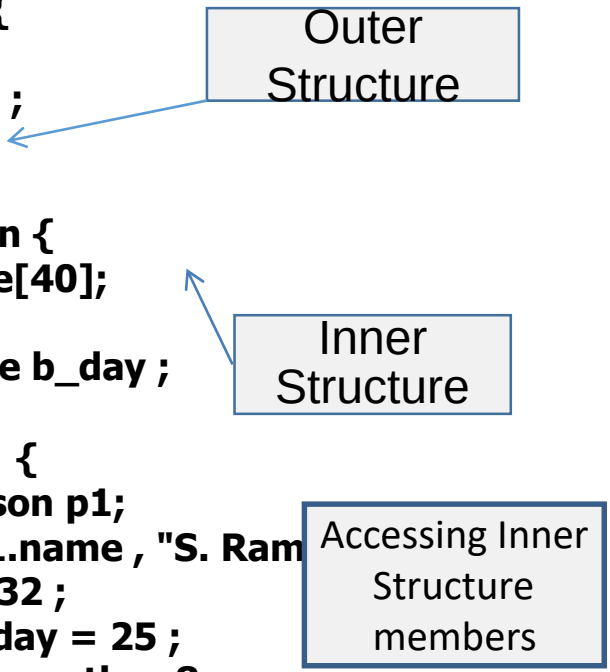
In this example, we see that the structure student contains two other structures, NAME and DATE. Both these structures have their own fields. The structure NAME has three fields: first_name, mid_name, and last_name. The structure DATE also has three fields: dd, mm, and yy, which specify the day, month, and year of the date. Now, to assign values to the structure fields, we will write

```
struct student stud1;
stud1.name.first_name = "Janak";
stud1.name.mid_name = "Raj";
stud1.name.last_name = "Thareja";
stud1.course = "BCA";
stud1.DOB.dd = 15;
stud1.DOB.mm = 09;
stud1.DOB.yy = 1990;
stud1.fees = 45000;
```

In case of nested structures, we use the dot operator in conjunction with the structure variables to access the members of the innermost as well as the outermost structures.

Guided activity on nested structures

```
#include<stdio.h>
#include<string.h>
struct date {
    int day ;
    int month ;
    int year ;
} ;
struct person {
    char name[40];
    int age ;
    struct date b_day ;
};
int  main( ) {
   struct person p1;
   strcpy ( p1.name , "S. Ram
   p1. age = 32 ;
   p1.b_day.day = 25 ;
   p1.b_day. month = 8 ;
   p1.b_day. year = 1978 ;
}
```



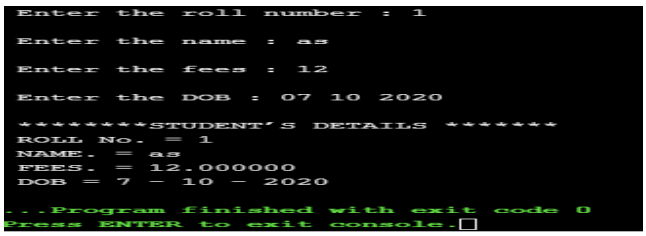
OUTPUT:

No output since there is no print statment

Write a program to read and display information of a student using structure within a structure

```
#include<stdio.h>
int main(){ struct DOB  {
                        int day;
                        int month;
                        int year;     };
            struct student              {
                            int roll_no;
                        char name[100];
                        float fees;
                        struct DOB date;          };
            struct student stud1;
            printf("\n Enter the roll number : ");
            scanf("%d", &stud1.roll_no);
            printf("\n Enter the name : ");
            scanf("%s", stud1.name);
            printf("\n Enter the fees : ");
            scanf("%f", &stud1.fees);
            printf("\n Enter the DOB : ");
            scanf("%d %d %d", &stud1.date.day, &stud1.date.month, &stud1.date.year);
            printf("\n ********STUDENT'S DETAILS *******");
            printf("\n ROLL No. = %d", stud1.roll_no);
            printf("\n NAME. = %s", stud1.name);
            printf("\n FEES. = %f", stud1.fees);
            printf("\n DOB = %d - %d - %d", stud1.date.day, stud1.date.month,
stud1.date.year);
}
```

OUTPUT:



5.2 Arrays Of Structures

In the above examples, we have seen how to declare a structure and assign values to its data members. Now, we will discuss how an array of structures is declared. For this purpose, let us first analyse where we would need an array of structures.

In a class, we do not have just one student. But there may be at least 30 students. So, the same definition of the structure can be used for all the 30 students. This would be possible when we make an array of structures. An array of structures is declared in the same way as we declare an array of a built-in data type.

Another example where an array of structures is desirable is in case of an organization. An organization has a number of employees. So, defining a separate structure for every employee is not a viable solution. So, here we can have a common structure definition for all the employees. This can again be done by declaring an array of structure employee.
The general syntax for declaring an array of structure can be given as,
struct struct_name struct_var[index];

Consider the given structure definition.

```
struct student{

int r_no;

char name[20]; char course[20]; float fees;};
```

A student array can be declared by writing,
struct student stud[30];

Now, to assign values to the i^{th} student of the class, we will write,

```
stud[i].r_no = 09;
stud[i].name = "RASHI";
stud[i].course = "MCA";
stud[i].fees = 60000;
```

In order to initialize the array of structure variables at the time of declaration, we can write as follows:

```
struct student stud[3] = {{01, "Aman", "BCA", 45000},{02, "Aryan", "BCA", 60000},
{03,"John", "BCA", 45000}};
```

Write a program to read and display information of all the students in the class (using Array of structure)

```
#include<stdio.h>
int main()
{       struct student
        {
                int roll_no;
                char name[80];
                float fees;
                char DOB[80];
        };
        struct student stud[50];
        int n, i;
        printf("\n Enter the number of students : ");
        scanf("%d", &n);
        for(i=0;i<n;i++)
        {       printf("Enter the roll number : ");
                scanf("%d", &stud[i].roll_no);
                printf("Enter the name : ");
                scanf("%s", stud[i].name);
                printf("Enter the fees : ");
                scanf("%f", stud[i].fees);
                printf("Enter the DOB : ");
                scanf("%s", stud[i].DOB);
        }
        for(i=0;i<n;i++)
        {       printf("\n*DETAILS OF %dth STUDENT*", i+1);
                printf("\n ROLL No. = %d", stud[i].roll_no);
                printf("\n NAME. = %s", stud[i].name);
                printf("\n ROLL No. = %f", stud[i].fees);
                printf("\n ROLL No. = %s", stud[i].DOB);
        }
}
```

OUTPUT:

```
 Enter the number of students : 2
Enter the roll number : 1
Enter the name : ashik
Enter the fees : 3500
Enter the DOB : 12-12-1978
Enter the roll number : 2
Enter the name : asmi
Enter the fees : 4500
Enter the DOB : 12-12-1990

*DETAILS OF 1th STUDENT*
 ROLL No. = 1
 NAME. = ashik
 ROLL No. = 3500.000000
 ROLL No. = 12-12-1978
*DETAILS OF 2th STUDENT*
 ROLL No. = 2
 NAME. = asmi
 ROLL No. = 4500.000000
 ROLL No. = 12-12-1990
```

...Program finished with exit code 0

GUIDED ACTIVITY – Here is the activity you on (arrays and structures)

```
struct student
{
   int sub[3] ;
   int total ;
} ;


int main( )  {
  struct student s[3];
  int i,j;
  for(i=0;i<3;i++) {
      printf("\n\nEnter student %d marks:",i+1);
      for(j=0;j<3;j++) {
          scanf("%d",&s[i].sub[j]);
       }
    }
    for(i=0;i<3;i++) {
         s[i].total =0;
         for(j=0;j<3;j++) {
                s[i].total +=s[i].sub[j];
       }
       printf("\nTotal marks of student %d is: %d",
                        i+1,s[i].total );
    }
}
```

OUTPUT:

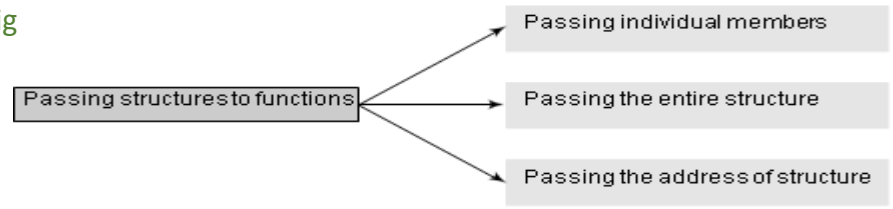
OUTPUT:

```
 Enter student 1 marks: 60 60 60
 Enter student 2 marks: 70 70 70
 Enter student 3 marks: 90 90 90

Total marks of student 1 is: 180
Total marks of student 2 is: 240
Total marks of  student 3 is: 270
```

5.3 Structure and Functions

For structures to be fully useful, we must have a mechanism to pass them to functions and return them. A function may access the members of a structure in three ways as shown in Fig



Passing Individual Structure Members to a Function

To pass any individual member of the structure to a function we must use the direct selection operator to refer to the individual members for the actual parameters. The called program does not know if the two variables are ordinary variables or structure members.

```
#include<stdio.h>
typedef struct
{
        int x;
        int y;
}POINT;
void display(int, int);
main()
{
        POINT p1 = {2, 3};
        display(p1.x, p1.y);
        return 0;
}
void display( int a, int b)
{
        printf(" The coordinates of the point are: %d %d", a, b);
}
```

OUTPUT:

The coordinates of the point are: 2 3

PASSING A STRUCTURE TO A FUNCTION

- When a structure is passed as an argument, **it is passed using call by value method. That is a copy of each member of the structure is made. No doubt, this is a very inefficient method especially when the structure is very big or the function is called frequently.** Therefore, in such a situation passing and working with pointers may be more efficient.
- The general syntax for passing a structure to a function and returning a structure can be given as, struct struct_name func_name(struct struct_name struct_var);
- The code given below passes a structure to the function using call-by-value method.

```
#include<stdio.h>
typedef struct
{
        int x;
        int y;
}POINT;
void display(POINT);
main()
{
        POINT p1 = {2, 3};
        display(p1);
        return 0;
}
void display( POINT p)
{
        printf(" The coordinates of the point are: %d %d", p.x, p.y);
}
```

OUTPUT:

The coordinates of the point are: 2 3

Guided activity on structures and functions

```
struct   fraction {
     int numerator ;
     int denominator ;
};

void show ( struct fraction f )
{
   printf ( " %d / %d ", f.numerator,
                    f.denominator ) ;
}

int main ( )  {
     struct fraction f1 = { 7, 12 } ;
     show ( f1 ) ;
}
```

OUTPUT:

7 / 12

PASSING STRUCTURES THROUGH POINTERS

C allows to crerate a pointer to a structure. Like in other cases, a pointer to a structure **is never itself a structure, but merely a variable that holds the address of a structure.** The syntax to declare a pointer to a structure can be given as

```
struct struct_name
{
   data_type member_name1;
   data_type member_name2;
   ....................................
}*ptr;
```

OR

```
struct struct_name *ptr;
```

For our student structure we **can declare a pointer variable** by writing

```
struct student *ptr_stud, stud;
```

The next step is to **assign the address of stud to the pointer using the address operator (&).** So to assign the address, we will write

```
ptr_stud = &stud;
```

To **access the members of the structure**, one way is to write

```
/* get the structure, then select a member */
(*ptr_stud).roll_no;
```

An alternative to the above statement can be used by using 'pointing-to' operator (->) as shown below.

```
/* the roll_no in the structure ptr_stud points to */
ptr_stud->roll_no = 01;
```

The selection operator (->) is a single token, so do not place any white space between them.

Write a program using pointer to structure to initialize the members in the structure

```
#include<stdio.h>
#include<string.h>
struct student
{
        int r_no;
        char name[20];
        char course[20];
        float fees;
};
main()
{       struct student stud1, *ptr_stud1;
        ptr_stud1 = &stud1;
        ptr_stud1->r_no = 01;
        strcpy(ptr_stud1->name, "Rahul");
        strcpy(ptr_stud1->course, "BCA");
        ptr_stud1->fees = 45000;
        printf("\n DETAILS OF STUDENT");
        printf("\n ---------------------------------------- ");
        printf("\n ROLL NUMBER = %d", ptr_stud1->r_no);
        printf("\n NAME =", puts(ptr_stud1->name));
        printf("\n COURSE = ", puts(ptr_stud1->course));
        printf("\n FEES = %f", ptr_stud1->fees);
}
```

OUTPUT:

```
DETAILS OF STUDENT
----------------------------------------------------
ROLL NUMBER = 1
NAME = Rahul
COURSE =  BCA
FEES = 45000.000000
```

Guided activity on Pointer to a structure

```
struct product
{
   int prodid;
   char name[20];
};
int main()
{
   struct product inventory[3];
   struct product  *ptr;
   printf("Read Product Details : \n");
   for(ptr = inventory;ptr<inventory +3;ptr++) {
     scanf("%d %s", &ptr->prodid, ptr->name);
   }
   printf("\noutput\n");
   for(ptr=inventory;ptr<inventory+3;ptr++)
   {
     printf("\n\nProduct ID:%5d",ptr->prodid);
     printf("\nName              :  %s",ptr->name);
   }
}
```

Accessing structure members through pointer :

i) Using . (dot) operator :

```
( *ptr ) . prodid = 111 ;
strcpy ( ( *ptr ) . Name, "Pen") ;
```

ii) Using - > (arrow) operator :

```
ptr - > prodid = 111 ;
strcpy( ptr->name , "Pencil") ;
```

Read Product Details :

111 Pen
112 Pencil
113 Book

Print Product Details :

Product ID : 111
Name : Pen
Product ID : 112
Name : Pencil
Product ID : 113
Name : Book

5.4 SELF REFERENTIAL STRUCTURES

A self referential structure is one that includes at least one member which is a pointer to the same structure type. With self referential structures, we can create very useful data structures such as linked -lists, trees and graphs.

Self referential structures are those structures that contain a reference to data of its same type. That is, a self referential structure in addition to other data contains a pointer to a data that is of the same type as that of the structure. For example, consider the structure node given below.

```
struct node
{
        int val;
        struct node *next;
};
```

Here the structure node will contain two types of data- an integer val and next that is a pointer to a node. You must be wondering why do we need such a structure? Actually, self-referential structure is the foundation of other data structures.

Self referential structures

```
struct student_node   {
    int roll_no ;
    char name [25] ;
    struct student_node  *next ;
} ;
int  main( )
 {
   struct  student_node  s1 ;
   struct  student_node  s2 = { 1111, "B.Mahesh", NULL } ;
   s1. roll_no  = 1234 ;
   strcpy ( s1.name , "P.Kiran " ) ;

    s1. next  =  & s2 ;

    printf ( " %s ", s1. name ) ;

    printf ( " %s " , s1.next - > name ) ;
}
```

s2 node is linked to s1 node

Prints P.Kiran

Prints B.Mahesh

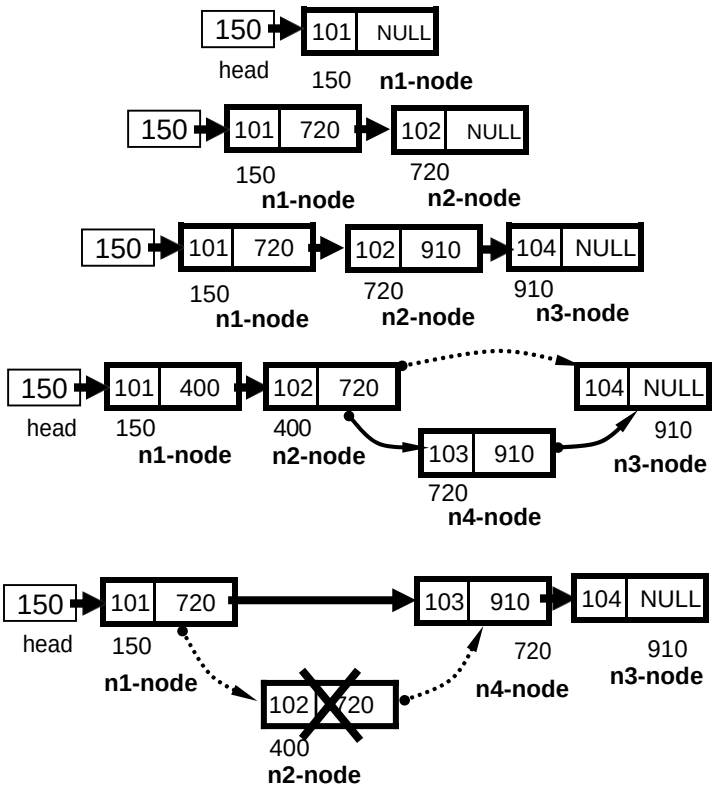
OUTPUT:

GUIDED ACTIVITY – Here is the activity you on (self referential structure – foundation for linked list)

```
struct node {
  int rollno; struct node *next;
};
int main() {
    struct node *head,*n1,*n2,*n3,*n4;
        /* creating a new node */
   n1=(struct node *) malloc(sizeof(struct node));
   n1->rollno=101;
   n1->next = NULL;
     /* referencing the first node to head pointer */
   head = n1;
      /* creating a new node */
   n2=(struct node *)malloc(sizeof(struct node));
   n2->rollno=102;
   n2->next = NULL;
     /* linking the second node after first node */
   n1->next = n2;
     /* creating a new node * /
   n3=(struct node *)malloc(sizeof(struct node));
n3->rollno=104;
   n3->next=NULL;
      /* linking the third node after second node */
   n2->next = n3;
      /* creating a new node */
   n4=(struct node *)malloc (sizeof (struct node));
   n4->rollno=103;
   n4->next=NULL;
       /* inserting the new node between
          second node and third node */
   n2->next = n4;
   n4->next = n3;
```

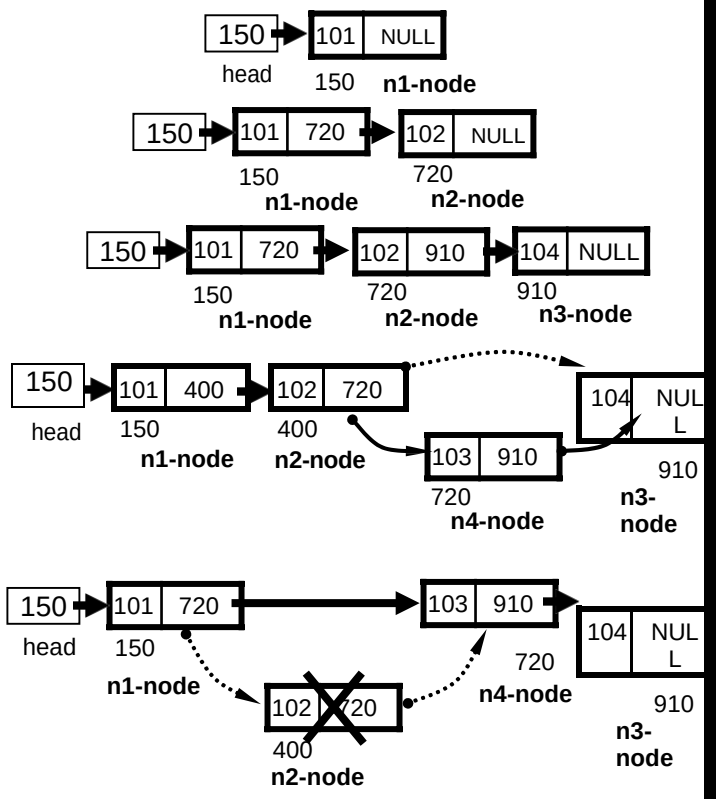
Creating a Singly Linked List

```
      /* deleting n2 node */
    n1->next = n4;
    free(n2);
  }
```



Implementing Singly Linked List

```
struct node *head=NULL;
```




```
#include<stdio.h>
#include<stdlib.h>
struct node {
  int data;
  struct node *next;
};
struct node *createnode() {
   struct node *new;
   new = (struct node *) malloc(sizeof(struct node));
   //printf("\nEnter the data : ");
   //scanf("%d",&new->data);
   new->data=101; //102, 104
   new->next=NULL;
   return new;
}
void append(struct node **h) {
   struct node *new,*temp;
   new = createnode();
   if(*h == NULL) {
      *h = new;
      return;
   }
   temp = *h;
   while(temp->next!=NULL)
   temp = temp->next;
   temp->next = new;
}
void display(struct node *p) {
     printf("\nContents of the List : \n\n");
     while(p!=NULL)  {
        printf("\t%d",p->data);
        p = p->next; } }
 int main() {
   struct node *head=NULL;
   append(&head);
   display(head);
   append(&head);
   display(head);
   append(&head);
   display(head);
   }
```

Implementing Singly Linked List

```
struct node {
  int data;
  struct node *next;
};
struct node *createnode() {
   struct node *new;
   new = (struct node *)malloc(sizeof(struct node));
   printf("\nEnter the data : ");
   scanf("%d",&new->data);
   new->next = NULL;
   return new;
}
void append(struct node **h) {
   struct node *new,*temp;
   new = createnode();
   if(*h == NULL) {
      *h = new;
      return;
   }
   temp = *h;
   while(temp->next!=NULL) temp = temp->next;
   temp->next = new;
}
void display(struct node *p) {
     printf("\nContents of the List : \n\n");
     while(p!=NULL)  {
         printf("\t%d",p->data);
         p = p->next; } }
```

```
int main() {
  struct node *head=NULL;
  int ch;
  while(1) {
     printf("\n1.Append");
     printf("\n2.Display All");

     printf("\n8.Exit program");
     printf("\n\n\tEnter your choice : ");
     scanf("%d",&ch);
     switch(ch)    {
        case 1:append(&head);break;
        case 2:display(head);break;
;
        case 8:exit(0);break;
        default :
         printf( "Wrong Choice, Enter correct one : ");
     }
  }
}
```

Compute the age of a person using structure and functions (passing a structure to a function) –Compute the number of days an employee came late to the office by considering his arrival time for 30 days (Use array of structures and functions)

Topic. 5.5

5.5 Exercise programs:
Compute the age of a person using structure and .fuCnctions (passing a structure to a function) – Compute the number of days an employee came late to the office by considering his arrival time for 30 days (Use array of structures and functions)

GUIDED ACTIVITY – Here is the activity you on (compare dates in C)

```
#include <stdio.h>
struct date {
int dd, mm, yy;} ;

int date_cmp(struct date d1, struct date d2);
void date_print(struct date d);

int main(){
    struct date d1 = {7, 3, 2005};
    struct date d2 = {24, 10, 2005};
    date_print(d1);
    int cmp = date_cmp(d1, d2);
    if (cmp == 0)
       printf(" is equal to");
    else if (cmp > 0)
       printf(" is greater i.e. later than ");
    else printf(" is smaller i.e. earlier than");
       date_print(d2);
    return 0;}

/* compare given dates d1 and d2 */
int date_cmp(struct date d1, struct date d2){
    if (d1.dd == d2.dd && d1.mm == d2.mm && d1.yy ==d2.yy)
       return 0;
    else if (d1.yy > d2.yy || d1.yy == d2.yy && d1.mm > d2.mm || d1.yy == d2.yy
&& d1.mm == d2.mm && d1.dd > d2.dd)
    return 1;
    else return -1;}

/* print a given date */
    void date_print(struct date d)    {
       printf("%d/%d/%d", d.dd, d.mm, d.yy);}
```

Compute the age of a person using structure and functions (passing a structure to a function) – **Age Calculator**: This program will read your date of birth and print the current age. The logic is behind to implement this program - Program will compare given date with the current date and print how old are you?

```
/*Age Calculator (C program to calculate age).*/
#include<stdio.h>
int date_diff(struct date dt1, struct date dt2);

struct date {
int day, month, year; };


int main() {
    struct date dt1 = {05, 10, 2020};
    struct date dt2 = {17, 05, 2004};
    int cmp = date_diff(dt1, dt2);
return cmp;}

int date_diff(struct date dt1, struct date dt2){
  int years,months,days;
  if(dt2.year>dt1.year) {
    years=0;    months=0;    days=0;
    printf("\n I can't travel in time");}
  else if(dt2.year==dt1.year){
    years=0;
    if(dt2.month>dt1.month){
      months=0;      days=0;
      printf("\n I can't travel in time");}
    else if(dt2.month==dt1.month){ months=0;
      if(dt2.day>dt1.day){
        days=0;
        printf("\n I can't travel in time");}
      else if(dt2.day==dt1.day){        days=0;
        printf("\n Welcome to Earth");}
      else
        days=dt1.day-dt2.day;}
    else{
    months=dt1.month-dt2.month;
    if(dt2.day>dt1.day) {      months--;
      days=30-dt2.day+dt1.day; }
    else
      days=dt1.day-dt2.day;} }
  else {
    years=dt1.year-dt2.year;
    if(dt2.month>dt1.month) {
      years--;
      months=12-dt2.month+dt1.month;
      days=30-dt2.day+dt1.day;}
    else {
      months=dt1.month-dt2.month;
      if(dt2.day>dt1.day) {
        months--;
        days=30-dt2.day+dt1.day; }
      else
        days=dt1.day-dt2.day;} }
printf("\n Your age is %d years, %d months, %d days",years,months,days);}
```

Your age is 16 years, 4 months, 18 days

GUIDED ACTIVITY – Here is the activity you on (Time in C)

```
#include  <stdio.h>
#include  <stdlib.h>
#include <time.h>
// Print current date and time in C
int main(void){
        // variables to store date and time components
        int hours, minutes, seconds, day, month, year;
        // time_t is arithmetic time type
        time_t now;
        // Obtain current time
        // time() returns the current time of the system as a time_t value
        time(&now);
        // Convert to local time format and print to stdout
        printf("Today is : %s", ctime(&now));
        // localtime converts a time_t value to calendar time and
        // returns a pointer to a tm structure with its members
        // filled with the corresponding values
        struct tm *local = localtime(&now);

        hours = local->tm_hour;          // get hours since midnight (0-23)
        minutes = local->tm_min;         // get minutes passed after the hour (0-59)
        seconds = local->tm_sec;         // get seconds passed after minute (0-59)

        day = local->tm_mday;            // get day of month (1 to 31)
        month = local->tm_mon + 1;       // get month of year (0 to 11)
        year = local->tm_year + 1900;    // get year since 1900

        // print local time
        if (hours < 12)          // before midday
                printf("Time is : %02d:%02d:%02d am\n", hours, minutes, seconds);

        else          // after midday
                printf("Time is : %02d:%02d:%02d pm\n", hours - 12, minutes, seconds);

        // print current date
        printf("Date is : %02d/%02d/%d\n", day, month, year);

        return 0;
}
```

```
Today is : Fri Oct  9 04:37:08 2020
Time is : 04:37:08 am
Date is : 09/10/2020


...Program finished with exit code 0
Press ENTER to exit console.
```

Compute the number of days an employee came late to the office by considering his arrival time for 30 days (Use array of structures and functions)

```
#include <stdio.h>
#include <time.h>
struct student{
                    char lastName[100];
                    char firstName[100];
                    char *date;
                    int age;
                    int id;};
int main(){
int n=1;
struct student s[n];
int x;
do{
printf("main menu :\n1.add\n2.delete\n3.diplay\n4.exit\n");
scanf("%d",&x);
switch(x){
                    case 1:
                    for(int i=0;i<n;i++){
                    printf("Enter first name\n");
                    scanf("%s",s[i].firstName);
                    printf("Enter last name\n");
                    scanf("%s",s[i].lastName);
                    printf("Enter your id\n");
                    scanf("%d",&s[i].time);
                    printf("Enter your age\n");
                    scanf("%d",&s[i].age);
                    time_t timer;
                    timer=time(NULL);
                     s[i].date = asctime(localtime(&timer));
                     //s[i].time=time(&now);
            }
                    for(int i=0;i<n;i++){
                    printf("id\tfirstName\tlastName\tage\tdate\n%d\t%s\t%s\t%d\t%s",s[i].id,s[i].firstNa
me,s[i].lastName,s[i].age,s[i].date);
                    }
                    break;
                    case 2:
                    break;
                    case 3:
   break;
   case 4:
   break;
   default:
                    printf("wrong choice");
   break;
}
}while(x!=4);
return 0;
}
Note:     time_t t;
          time(&t);
          printf("\n current time is : %s",ctime(&t));
```

Test Yourself –5.1 to 5.6 Topics (quiz)

1) A data structure that can store related information together is called
(a) Array (b) String (c) Structure (d) All of these
2) A data structure that can store related information of different datatypes together is called
(a) Array (b) String (c) Structure (d) All of these
3) Memory for a structure is allocated at the time of
Structure definition Structure variable declaration
Structure declaration Function declaration
4) A structure member variable is generally accessed using
(a) Address operator (b) Dot operator
(c) Comma operator (d) Ternary operator
5) A structure that can be placed within another structure is known as
Self-referential structure Nested structure
Parallel structure Pointer to structure
6) A union member variable is generally accessed using the
(a) Address operator (b) Dot operator
(c) Comma operator (d) Ternary operator
7) typedef can be used with which of these data types?
(a) struct (b) union
(c) enum (d) all of these

Assignment Questions

CO 1	**Develop C program solutions to simple computational problems**		
1.	Declare a structure that represents the following hierarchical information. Student Roll Number Name First name Middle Name Last Name Sex Date of Birth Day Month Year Marks English Mathematics Computer Science	K2	CO1
2.	Define a structure date containing three integers— day, month, and year. Write a program using functions to read data, to validate the date entered by the user and then print the date on the screen. For example, if you enter 29,2,2010 then that is an invalid date as 2010 is not a leap year. Similarly 31,6,2007 is invalid as June does not have 31 days.	K2	CO1
3.	Write a program to define a union and a structure both having exactly the same members. Using the sizeof operator, print the size of structure variable as well as union variable and comment on the result.	K2	CO1

Part A

1. What is Structure? Write the syntax for structure.	**K3**	**CO 3**	**S**
2. How the members of structure object is accessed?	K3	CO 3	S
3. What is a nested structure?	K3	CO 3	S
4. How typedef is used in structure?	K3	CO 3	A
5. What is meant by Self-referential structures?	K3	CO 3	A
6. Develop a structure namely Book and create array of Book structure with size of ten.	K2	CO 3	S
7. Invent the application of size of operator to this structure. Consider the declaration: **struct** **{** **char name;** **int num;** **} student;**	K2	CO 3	S
8. List the use of typedef.	K2	CO 3	A
9. Differentiate between Structure and Array.	K2	CO 3	A
10. Define the meaning of Array structure.	K2	CO 3	A

Part B

1. Describe about the functions and structures. (13)	**K3**	**CO3**	**S**
2. Explain about the structures and its operations with example programs	K3	CO3	S
3. Explain about array of structures and nested structures with example.(13)	K3	CO3	A
4. Write a C program using structures to prepare the students mark statement. (13)	K3	CO3	A
5. Write a C program using structures to prepare the employee payroll. (13)	K3	CO3	A
6. Compute the number of days an employee came late to the office by considering his arrival time for 30 days (Use array of structures and function)	K3	CO3	S