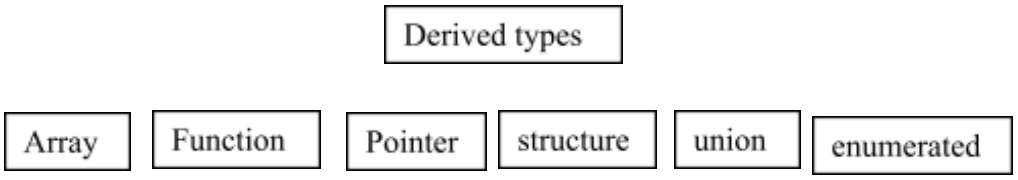
Derived types : These types are derived from existing types. There are six derived types in C



Q. **What is structure and write the general format for declaring and accessing members.**
Structure: A structure is a collection of related elements possibly of different types, having a single name. Each element in a structure is called a field or a member.
Difference between array and structure : Another derived data type that holds group of elements is an array. The difference between an array and a structure is that all elements in an array must be of the **same type**, while the elements in a structure can be of the **same or different types.**

Structure declaration :
There are two ways to declare a structure.

1. **Tagged structure**
2. **Type defined structure**

Tagged structure:

The general format (syntax) of tagged structure definition is as follows

```
structtagname
{
        datatype member1;
        datatype member2;
                ...     ...
                ...     ...
};
```

Eg:

```
struct student
            {
                        char name[50];
                int m1,me,m3;
                        floatavg;
                        char grade;
             };
```

- In the above delcationstruct is a keyword.
- tagname is the name given to the structure
- datatype can be int, float, char or any other type.
- member1, member2,… are individual members of a structure.
- Individual members of a structure can be variables or pointers or arrays or even be structures.

- Individual members cannot be initialized within the structure declaration.
- After closing bracket semicolon must be specified.
- After declaration of a structure, variables are declared as follows
  structtagname v1, v2,…vn;
- Members of a structure do not occupy any space. Memory is allocated only after declaring structure variables.

Type defined structure :

structres can also be declared as

```
typedefstruct
{
        datatype member1;
        datatype member2;
                ...     ...
                ...     ...
}type (or) identifier;
```

Example:

```
typedefstruct
            {
                        char name[50];
                int m1,m2,m3;
                        floatavg;
                        char grade;
        }student;
```

Structure variable declaration :

After a structure has been declared, we can declare variables using it. Generally structure is declared in the global area of a program to make it visible to all functions. The variables will be declared inside the functions**.**

```
structtagname
{
        datatype member1;
        datatype member2;
};

sturcttagname v1, v2,…vn;
```

Eg declaration:

```
        struct student
            {
                        char name[50];
                int m1,m2,m3;
                        floatavg;
```

```
        char grade;
};
    struct student s1,s2;  // s1 and s2 are variables of struct student type.
                           // memory is allocated for all the fields of s1 and s2
```

structure variables can also be declared as

```
typedefstruct
        {
                char name[50];
    int m1,m2,m3;
                floatavg;
                char grade;
    }student;
student s1,s2; // student is the type s1 and s2 are variables
```

Structure is used to group logically related data items. The members of a structure themselves are not variables. They do not occupy any memory until they are associated with the structure variable.

Initialization of Structures :

We can initialize a structure ie we can give values to a structure at the time of declaration. The rules for structure initialization are similar to the rules for array initialization. The values are enclosed in braces and separated by commas. They must match their corresponding types in the structure definition.

```
    struct student
        {       char name[50];
                int m1,m2,m3;
                floatavg;
                char grade;
        };
```

struct student s1={"xxxxxx",12,16,18};
// if values are not given default value is 0 for numeric and \0 for characters

Accessing a Structure variable: The member of a structure variable can be accessed using the member operator(.) which is also known as dot operator.
For the above example first we need to define variables to struct student.
struct student s1, s2, s3;

Memory will be allocated for three structure variables s1,s2 and s3. Now each variable s1, s2, s3 has members namely name,m1,m2,m3,avg and grade. If we want to assign a name to student s1 then it is assigned as
strcpy(s1.name, "rama");
(or)
gets(s1.name);
To assign values to m1 and m2 we can use s1.m1 = 15; and s1.m2 = 20;
Similar is the case with remaining members of remaining variables.

Assigning values: one way of assigning values is at the time of variable declaration.
struct student s ={24, “ramu”, 90.0};
Another way is to assign values to each member individually.
strcpy(s.name,”ramu”);
s.m1=15;
s.m2=20;

Advantages of structures:

- Using a structures we can group elements of different data types where an array is used to group elements of same type.
- In a structure it is not necessary to know the position of a particular element in order to access it.
- Once a new structure has been declared, we can declare any number of variables of that type.
- One structure can be copied or assigned to other structure. It decreases the burden of assigning individual elements like array.
- It is possible to initialize all the fields of a structure at the same time of its declaration.
- A structure provides an efficient way of insertion and deletion of elements

Array of structures: It contains collection of structures. In an array of structures, each element of an array represent a structure variable. For example

```
struct student
    {
            char name[50];
            int m1,m2,m3;
            floatavg;
            char grade;
}s[10];
```

The above example defines an array **s** that consists of 20 elements. Each element of the array is structure of type student

Arrays within structures: C permits the use of arrays as structure members. we can use single or multi dimensionalarrays.For example

```
structstu
{
    intrno;
    char name[23]; // character array
    int m[3]; // int array
};
```

Nested structures (Structures within structures): Structures within structures mean nesting of structures. Nesting of structures is permitted in C. Here, the individual members of a structure can again be structures. There are two ways of declaring such a structure. They are given below.

```
typedefstruct
{
    intday,month,year;
```

```
}date;

typedefstruct
{
        inthrs,mnts,secs;

}time;

struct student
{
        char name[50];
datebday; // date is a structure type
timebtime; // time is a structure type , already declared
};
```

In the above example struct date and struct time are to be defined before it is used in student structure. This is because only legal data types are allowed in structure declaration. Only those data types which are declared till that point are considered as legal data types. The same can be achieved using the following mechanism.

```
struct student
{       char name[50];
        struct date
        {
                intday,month,year;
}bday;
        struct time
        {
                inthrs,mnts,secs;
        }btime;
};
struct student s1;
```

The members day ,month and year are accessing as follows:
s1.bday.day, s1.bday,month, s1.bday.year

Structure &Functions :- we can pass structure variables as parameters(arguments) to functions.

There are **3 methods** by which the values of a structure can be transferred from one function to another function.

1. The first method is to pass each member of the structure as an actual argument of the function-call. The actual arguments are then treated independently like ordinary variables.
   Syntax for function call :
   function-name(structure variablename.member1, structurevariable name.member2--- structurevariablename.member n);

2. The second method involves a copy the entire structure to the called function. Since the function is working on a copy of the structure, any changes to structure members within the function are not reflected in the original structure (in the calling function). It is call by value. Syntax for function call :

function-name(structurevariable name); // call by value
ex :- printdata(s);

3. Pointers to pass the structure as an argument. In this case, the address of the structure is passed to the function.

Syntax for function call :

function-name(&structurevariable name); // call by reference
ex :- readdata(&s);

POINTERS to Structure :-

Pointer is a variable that holds the address of another variable of basic data types such as int, float, etc., In the same way we can also define pointer to structure. Such pointers are called **Structure Pointers**. Pointers along with structures are used to make linked lists and trees.

->operator is used to access pointer with member.

```
Syntax :    Struct Tag-name
            {
            data-type   member-1;
            data-type   member-2;
                        ---
            data-type   member-n;
            };
For ex.,    struct  student
            {
            introllno;
            char name[10];
            int age;
            }*x;
```

where *x is pointer to structure student.
To access member with structure pointer is
x->rollno, x->name, x->age etc

Self referential Structures :

If a member of structure is a pointer to its own type then that structure is known as a self referential structure. These structures are used to create linked lists, trees for dynamic memory allocatin.

Declaration of a self referentialstructure :

```
typedefstruct
{
int data;
node *next;  // pointer to node type
}node;
```

//Program for create and display the linked list

```
#include <stdio.h>
typedefstruct
{
int data;
node *next;  // pointer to node type
```

```
}node;

main()
{
node *head=NULL,*prev=NULL,*temp;
int n;
charans;
do
    {    printf("enter a number  : ");
         scanf("%d",&n);
         temp=(node*) malloc(sizeof(node)); // create memory for a node
         temp->data = n;      // place n in data field
         temp->next = NULL;
         if (head==NULL) // if head is null temp becomes head and prev
         {
         head=temp;
         prev=temp;
         }
else     // else attach temp to last node
         {
         prev->next=temp;
         prev=temp;
         }
         printf("\nAny more data (y/n) ");
         scanf("%c",&ans);
}while(ans=='y');
printf("\nThe elements of linked list are \n");
for(temp=head;temp!=NULL;temp=temp->next)
printf("%d->",temp->data);
}
```

typedef and its use in Structure Declarations :-

The 'typedef' is one of user-defined data type. **It is used to create a new data type** for an existing data type. No new data type is introduced, but an alternative(alias) name is given to a known data type.

typedef existing data type new data type;

The typedef statement simply defines a new type. It can be defined either at before main() (or) under declaration part of main().
The following examples show the use of typedef.

Ex :typedefint integer;

In the above example, integer is the new data type name given to data type int. Therfore, the following statements

integer x, y ;

Mean that x and y are variable names that are declared to hold 'int' data type.

When 'typedef' is used to name a structure, the structure tag name is not necessary such an example follows.

```
typedefstruct      /* no structure tag name used */
{
float real;
float imaginary;
} complex;                    /*   complex number  */
complexu,v;
```

In the above example declares ‘u’ and ‘v’ complex numbers having a real part and an imaginary part. The following are some examples involving structures and ‘typedef’.

```
typedefint integer;  // integer is the new type which is same as int
typedef float real;  // real is the new type which is same as float
typedefstruct stud  student;  // student is the new type
we can declare variables with the new type as follows
integer n1,n2;
realx,y;
student s1,s2;  all  these statements are valid.
```

Unions:

Unions are similar to structures and follow the same syntax as that of structures. The major difference of structures and unions is **in terms of storage**. In structures, each member has its own storage location; where as all the members of a union uses the same location. Although union has many members of different types, it can handle only one member at a time. Like structures, a union can be declared using the keyword union as follows.

```
unionemp
{
        int x;
        float y;
        char z;
}e1; //  e1 is name of variable
```

The above example declares a variable e1 of type union emp. The union contains 3 members each with a different data type. We can only use one of them at a time. The compiler allocates storage that is larger enough to hold the largest variable type in the union.

Eg program using unions:

```
unionemp
{
        int x;
        float y;
        char z;
}e1;
main()
{
  e1.x = 10;
printf(“x= %d “,e1.x);
  e1.y=20.5;
printf(“y= %d “,e1.y);
```

```
printf("x= %d ",e1.x);  // garbage value will be printed, since union contains y value.
}
```

Enumerated types :

Enumerated type is a user defined type based on the standard integer type. In enumerated type, each integer value is given an identifier called an enumeration constant. We can use these constants as symbolic names which makes programs much more readable.

Declaring an enumerated type :

enumtypename { identifier list};

eg :enum color { RED,BLUE,GREEN,WHITE};

color becomes the enumerated with four possible colors. Variables can be declared with color type as follows.

```
enum color x,y,z;
x=BLUE;
y=WHITE;
z=PURPLE; // is wrong, compiler gives error
```

color variable can have only 4 given colors. The constants are initialized with values from 0. Ie for above example RED is 0, BLUE is 1 etc.

eg 2 : enum months {JAN,FEB,MARCH,APRIL,MAY,JUNE};

enum months m;//m is a variable

we can use these in a switch statement as follows

```
switch(m)
{
case JAN :  printf("JANUARY");
break;
case FEB :  printf("FEBRUARY");
break;
    …….
}
```

Two enumerated variables can be compared as x==y or x!=y.

We can initialize the enum constants.

enum months {JAN=1,FEB,MARCH,APRIL,MAY,JUNE};

the value given to JAN is 1, FEB is 2, MARCH is 3 and so on.

Assignment Questions :

1. Define structure and write the general format for declaring and accessing members.
2. Compare arrays, structures and unions
3. Explain the parameter passing techniques in detail
4. How user defined functions can be declared and defined and called explain with example
5. How to pass structures and to functions