Module 5

Structures and Pointers

STRUCTURES

Ques 1) What is structure in C? How is it declared? How are its members accessed? Give examples.

Or

How the members of a structure variable are assigned initial values? How is a structure member accessed?

Or

How do we declare structure in C? How do we initialize the structures? How do we access the structure elements? Explain with examples.

Or

Explain the rules of initializing structures.

Or

Explain the dot (.) operator in C language with proper example.

Ans: Structure

A structure is a user defined data type. Structure is the collection of variables of different types under a single name for better handling.

For example, user wants to store the information about person about his/her name, citizenship number and salary. He/she can create this information separately but, better approach will be collection of this information under single name because all these information are related to person.

Declaring a Structure

Declaring a structure requires the **struct** keyword, followed by a name. Then collection of variables is declared between a pair of curly brackets.

Syntax:

```
struct struct_name
{
    type member1;
    type member2;
};
```

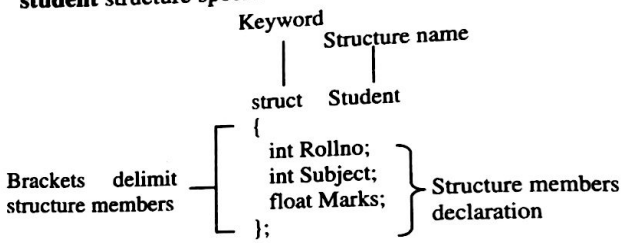
Structure declaration always starts with **struct** keyword. Here, **struct_name** is known as tag. The struct declaration is enclosed within a pair of curly braces.

Using **struct** and **tag** user can declare structure members like member1, member2 and so on. These are the members of the structure. Variables of the corresponding data type are created as given below:

struct struct_name m1, m2;

Here, m1, m2 are variables of structure struct_name.

The declaration defines the structure but this process doesn't allocate memory. The memory allocation takes places only when variables are declared. **For example,** the **student** structure specifier is illustrated in **figure below:**



Initialisation of Structure Variable /Assigning Values to Structure

The syntax for initialising structures is similar to that for initialising arrays. A structure variable in a declaration can be initialised by following it with an equal sign and a list of constants contained within braces. If not enough values are used to assign all the members of the structure, the remaining members are assigned the value zero by default.

Some **examples** are as following:

1) Card c = {13, 'h'}; /* the king of hearts */

2)
```
Complex a[3][3] = {
    {{1.0, – 0.1}, {2.0, 0.2}, {3.0, 0.3}},
    {{4.0, – 0.4}, {5.0, 0.5}, {6.0, 0.6}},
    };   /* a[2][] is assigned zeroes */
```

3) struct fruit frt = {"plum", 150};

4)
```
struct home_address
{
    char *street;
    char *city_and_state;
    long zip_code;
} address = {"87 West Street", "Aspen, Colorado",
80526};
```

5) struct home_address previous_address = {0};

The last **example** illustrates a convenient way to initialise all members of a structure to have value zero.

It causes pointer members to be initialised with the pointer value NULL and array members to have their elements initialised to zero.

Rules for Initializing Structures

1) One cannot initialize individual members inside the structure template.
2) The order of values enclosed in braces must match the order of members in the structure definition.
3) It is permitted to have a partial initialization. We can initialize only the first few members and leave the remaining blank.

   The uninitialized members should be only at the end of the list.
4) The uninitialized members will be assigned default values as follows:
   i) Zero for integer and floating point numbers.
   ii) '\0' for characters and strings.

Accessing Members

Each element in a structure can be accessed and manipulated using operators and expressions.

The C language uses hierarchical naming conventions that first use the structure variable-identifier and the element-identifier.

Syntax:

variable_identifier.element-identifier;

Or

structure_var.member_variable;

The structure variable-identifier is separated from element-identifier by a dot (.).

The dot here is known as the **direct selection operator**, which is a postfix operator at precedence level 1 in the precedence table.

For example, consider the following statement:

part2.modelnumer=3884;

The first component of above expression involving the dot operator is the name of the specific structure variable (part2), not the name of the structure definition (part).

The variable name must be used to distinguish one variable from another, such as part1, part2 and so on, as shown in **figure 5.1**.

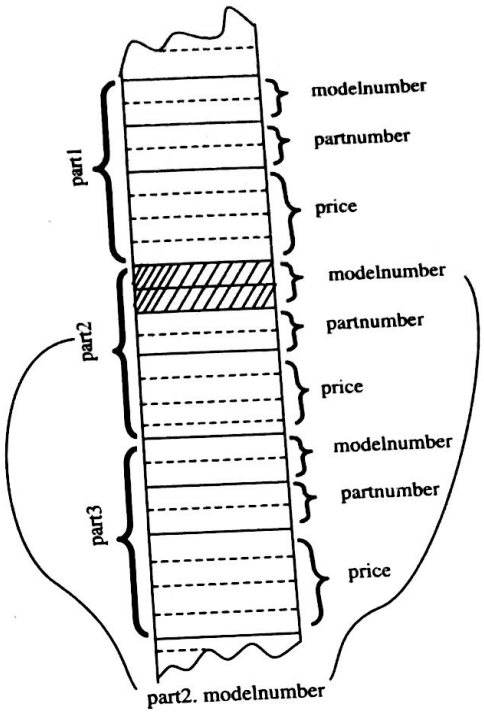


Figure 5.1: Dot Operator

Ques 2) Explain typedef and its use in structure declarations.

Or

What is purpose of the typedef feature? How is this feature used in Structure declaration? What is its need? Give example.

Ans: typedef and Purpose of typedef Feature

The typedef is a keyword used to assign alternative names to existing types. It's mostly used with user defined data types, when names of data types get slightly complicated.

Programmer can use typedef declarations to construct shorter or more meaningful names for types already defined by C or for types that programmer have declared.

typedef names allow programmer to encapsulate implementation details that may change. A typedef declaration is interpreted in the same way as a variable or function declaration, but the identifier, instead of assuming the type specified by the declaration, becomes a synonym for the type.

Once a user-defined data type has been established, then new variables, arrays, structures and so on, can be declared in terms of this new data type.

Syntax:

typedef existing_type new_type;

Where **existing_type** refers to an existing data type, either a standard data type, or a previous user-defined data type, and **new_type** refers to the new user-defined data type.

Needs of typedef

The purpose of typedef is to assign alternative names to existing types, most often those whose standard declaration is cumbersome, potentially confusing, or likely to vary from one implementation to another.

Use of typedef in Structure Declarations

typedef can be used to give a name to user defined data type as well.

Syntax:

```
typedef struct
{
 type member1;
 type member2;
 type member3;
} type_name ;
```

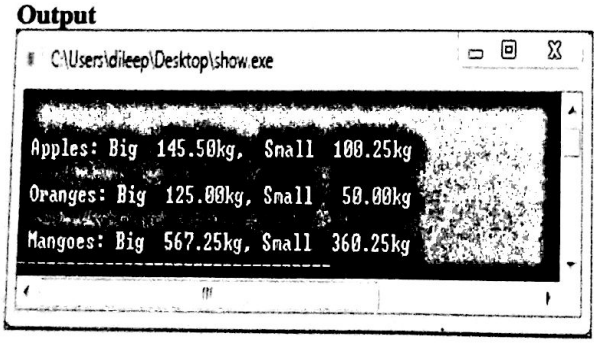
Here **type_name** represents the stucture definition associated with it. Now this type_name can be used to declare a variable of this stucture type:

```
type_name t1, t2 ;
```

For example, let consider the following program:

```
#include<stdio.h>
typedef struct fruits
{
    float big;
    float small;
}weight;

int main()
{
    weight apples = {145.5,100.25};
    weight oranges = {125,50};
    weight mangoes = {567.25, 360.25};
    printf("\n\n Apples: Big %7.2fkg,
    Small %7.2fkg",apples.big, apples.small);
    printf("\n\n Oranges: Big %7.2fkg, Small %7.2fkg",
    oranges.big, oranges.small);
    printf("\n\n Mangoes: Big %7.2fkg, Small %7.2fkg",
    mangoes.big, mangoes.small);
    return 0;
}
```

Output



Ques 3) Explain Array of Structure with example, the syntax and usage of the following in C program.

Ans: Array of Structure

Arrays of structures mean collection of structures, in other word array storing different type of structure member variables.

Consider the following **figure 5.2**:

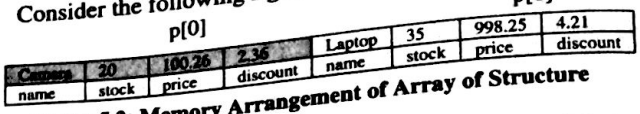


Figure 5.2: Memory Arrangement of Array of Structure

In **above figure**, cell size does not represent actual byte taken by variables. Same color cells represent structure whereas each cell represents each member variable of structure.

Syntax:

```
struct struct-name
{
    datatype var1;
    datatype var2;
    ----------
    ----------
    datatype varN;
};

struct struct-name obj [ size ];
```

For example, let's take following code to understanding the above concept. In this program, there is a **product** structure containing same member variables:

```
struct product
{
  char name[30];
  int stock;
  float price, discount;
};
Product p[2];
```

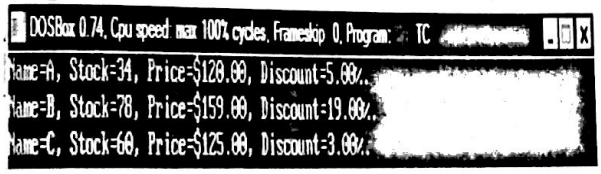
Product structure consists of four member variables **name, stock, price** and **discount**. So products p is an array of four elements each.

Program: Illustrating Array of Structure

```
#include<stdio.h>
#include<conio.h>
struct product
{
    char name[30];
    int stock;
    float price, discount;
};
void main()
{
    struct product p[3];
    int i;
    float temp;
    clrscr();
```

```
    for(i=0;i<3;i++) {
        printf("Enter product name :");
        gets(p[i].name);

        printf("\nEnter Stock :");
        scanf("%d",&p[i].stock);

        printf("\nEnter Price :");
        scanf("%f",&temp);

        p[i].price = temp;
        printf("\nEnter Discount :");
        scanf("%f",&temp);

        p[i].discount = temp;
        printf("\n\n");
        fflush(stdin); }
    clrscr();
    for(i=0;i<3;i++){
        printf("Name=%s,    Stock=%d,    Price=$%.2f,
        Discount=%.2f%.\n",    p[i].name,    p[i].stock,
        p[i].price,p[i].discount);}
    getch();
}
```

Output



Explanation: In the above program we store information about 3 different products. Product information contains product name, stock, price and discount on purchase. From line no. 4 –9 declares product structure. Line no 14 declares an array of structure which can store 3 products you increase or decrease its capacity by replacing 3 by any positive integer. And Line no. 13 – 37 accepts data for products from user and stores it to structure. Now we understand that how to store values in array of structure **(figure 5.3)**.

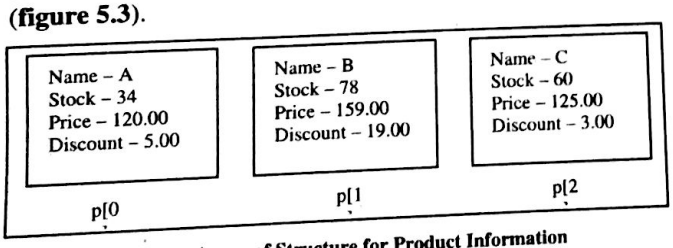


Figure 5.3: Array of Structure for Product Information

In line no. 14 we declared product structure variable p[3]. This means we have three structures in an array p[0], p[1] and p[2]. Above figure represents the same. So to store data in first index of structure variable (i.e. p[0]) we can write following lines:

```
gets(p[0].name);
scanf("%d",&p[0].stock);
scanf("%f",&p[0].price);
scanf("%f",&p[0].discount);
```

Above style for storing data is good for storing information of 1 or2 products but if you have 100s of product then you should use **for loop** because it is very convenient and you don't have to write lengthy code to do the same job.

Now we display stored data in p structure variable. Here also we used **for loop** which loops for 3 times. So to display data in first index of structure variable (i.e. p[0]) we can write following lines:

```
printf("%s", p[0].name);
printf("%d", p[0].stock);
printf("%.2f", p[0].price);
printf("%.2f", p[0].discount);
```

Ques 4) Give the difference between Structures and Arrays.

Ans: Difference between Structures and Arrays

Table below shows the main difference between structure and arrays:

Basis	Array	Structure
Data Collection	Array is a collection of homogeneous data.	Structure is a collection of heterogeneous data.
Element Reference	Array elements are referred by subscript.	Structure elements are referred by its unique name.
Access Method	Array elements are accessed by its position or subscript.	Structure elements are accessed by its object as '.' operator.
Data type	Array is a derived data type.	Structure is user defined data type.
Syntax	<data_type> array_name[size];	struct struct_name { structure member 1; structure member 2; --------- --------- structure member n; }struct_var_nm;
Example	int rno[5];	struct item_mst { int rno; char nm[50]; }it;

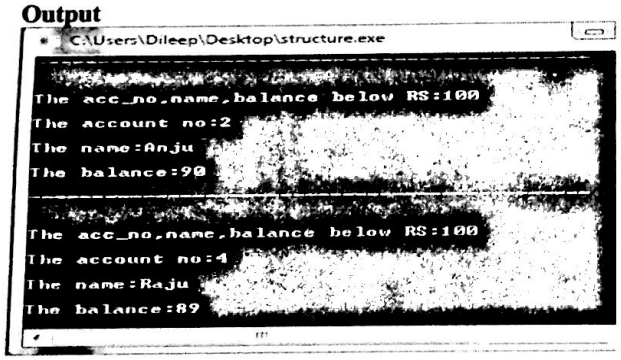
Ques 5) Create a structure to specify data of customers in a bank. The data to be stored is: account number, name, balance in account. Assume maximum of 200 customers in the bank. Write a program to print the account number and name of each customer with balance below rupees 100.

Ans: Program using Structure to Display the Details of Customers of a Bank Whose Balance is Below 100

```
#include<stdio.h>
#include<conio.h>
int main()
{
    struct bank{
        int acc_no;
        char name[20];
```

```
        int bal;
    }b[5];
    int i;
    printf(" \nEnter the acc_no,name,balance\n");
    for(i=0;i<5;i++){
        printf("\nEnter the account no:");
        scanf("%d",&b[i].acc_no);
        printf("\nEnter the name:");
        scanf("%s",b[i].name);
        printf("\nEnter the balance:");
        scanf("%d",&b[i].bal);}
    for(i=0;i<5;i++)
    {
        if(b[i].bal<100){
            printf("\n----------
            \n");
            printf("\n\nThe acc_no,name,balance below
            RS:100\n");
            printf("\nThe account no:%d\n",b[i].acc_no);
            printf("\nThe name:%s\n",b[i].name);
            printf("\nThe balance:%d\n",b[i].bal);}
    }
    getch();
    return 0;
}
```

Output

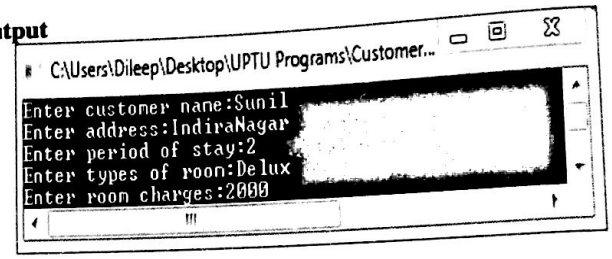


Ques 6) Write a program to store the records in a hotel as customer name, address, period of stay, room allotted and room charges.

Ans: Program to Store the Records in a Hotel

```
#include<stdio.h>
#include<conio.h>
int main()
{
    struct hotel{
        char customer_name[20];
        char address[20];
        int period_of_stay;
        char types_of_room[10];
        int room_charges;
    }h[3];

    int i;
    for(i=0;i<3;i++){
        printf("Enter customer name:");
        scanf("%s",&h[i].customer_name);
        printf("Enter address:");
        scanf("%s",&h[i].address);
        printf("Enter period of stay:");
        scanf("%d",&h[i].period_of_stay);
        printf("Enter types of room:");
        scanf("%s",&h[i].types_of_room);
        printf("Enter room charges:");
        scanf("%d",&h[i].room_charges);}
    for(i=0;i<3;i++){
        printf("\n Customer name
        is %s",h[i].customer_name);
        printf("\n Address = %s",h[i].address);
        printf("\n Period of stay=%d",h[i].period_of_stay);
        printf(" \n Types of stay=%s",h[i].types_of_room);
        printf("\n Room charges=%d",h[i].room_charges);
    }
    getch();
    return 0;
}
```

Output



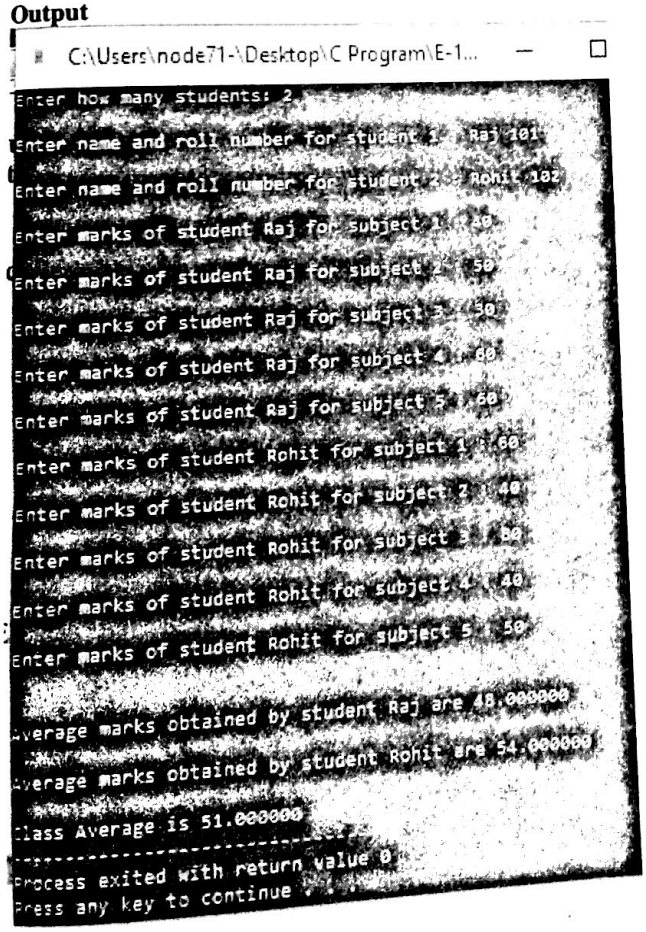
Ques 7) In a class there are fifty students, each student is enrolled for five subjects. Write a program to enter the marks of the students for different subjects and give the average marks of a student obtained by him in the subjects and overall average of the class.

Ans: Program Illustrating Structure for Student's Subjects and Marks and Displaying the Average of Class

```
#include<stdio.h>
#define SIZE 50

struct student {
  char name[30];
  int rollno;
  int sub[5];};

int main() {
  int i, j, max, count, total,  n, a[SIZE], ni;
  float avg,stu_avg=0, class_avg;
  struct student st[SIZE];

  printf("Enter how many students: ");
  scanf("%d", &n);

  /* for loop to read the names and roll numbers*/
  for (i = 0; i < n; i++) {
    printf("\nEnter name and roll number for student %d : ",
i+1);
    scanf("%s", &st[i].name);
    scanf("%d", &st[i].rollno);}
```

```
  /* for loop to read ith student's jth subject*/
  for (i = 0; i < n; i++) {
    for (j = 0; j <5; j++) {
      printf("\nEnter marks of student %s for subject %d : ",
st[i].name, j+1);
      scanf("%d", &st[i].sub[j]);
    }

   /* (i) for loop to calculate average marks obtained by each
student*/

   for (i = 0; i < n; i++) {
     total = 0;
     for (j = 0; j < 5; j++) {
       total = total + st[i].sub[j];
     }

     avg=float (total/5);
     stu_avg+=avg;
     printf("\n\nAverage marks obtained by student %s
are %f", st[i].name,avg);
     //a[i] = total;
   }

   /* (ii) for loop to find class average*/
  class_avg=stu_avg/n;
  printf("\n\nClass Average is %f", class_avg);
   return 0;
}
```

Output

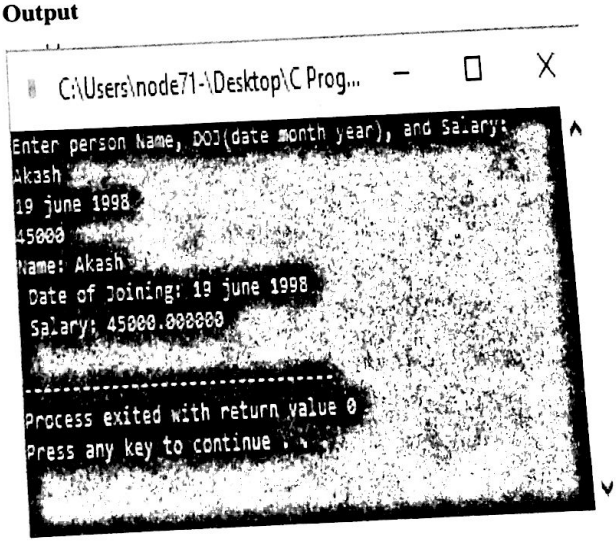


Ques 8) Define a structure type, struct Personal that would contain person name, date of joining and salary. Using this structure, write a program to read this information for one person from the key board and print same on the screen.

Ans: Structure of Personal Information

```
#include <stdio.h>
struct personal
{
     char name[20];
     int day;
     char month[10];
     int year;
     float salary;
};
int main()
{
     struct personal person;
     printf("Enter person Name, DOJ(date month year),
     and Salary:\n");
     scanf("%s %d %s %d %f",person.name,&person.day,
     person.month,&person.year,&person.salary);
     printf("Name: %s \n Date of Joining: %d %s %d \n
     Salary: %f\n",person.name,person.day,person.month,p
     erson.year,person.salary);
     return 0;
}
```

Output



UNION

Ques 9) Define union. Explain its uses. Write the syntax for the declaration of structure and union.

Or

Explain how to access union members?

Or

What is use of union? When is it useful?

Ans: Union

A union is a special data type that enables one to store different data types in the same memory location. Unions provide an efficient way of using the same memory location for multi-purpose. Unions contain two or more data types. Only one member, and thus one data type, can be referenced at a time.

A union is declared with the **union keyword** in the same format as a structure.

Syntax:

```
union union_name
{
  <data_type> member1
  <data_type> member2
  .
  .
  .
  <data_type> membern
}
```

For example, following is a union declaration:

```
union sample
{
    int p;
    float q;
};
```

This indicates that **sample** is a union type with members' **int p** and **float q**. The union definition normally precedes the main() in a program so the definition can be used to declare variables in all the program's functions.

Only a value with same type of the first union member can be used to initialize union in declaration part. Memory space required for defining a variable of the union is max(sizeof(member1), sizeof(member2)... sizeof(membern),).

Accessing Union Members

To access any member of a union, we use the **member access operator (.)**. The member access operator is coded as a period between the union variable name and the union member that one wishes to access. One would use the keyword **union** to define variables of union type.

For example, let consider the following code:

```
union car
{
    char name[50];
    int price;
};
-------Inside Function----------
union car c1, c2;
```

The member of unions can be accessed in similar manner as that structure. Suppose, one wants to access **price** for union variable **c1** in above example, it can be accessed as **c1.price**.

Usage of Union

1) On most computers, the size of a pointer and an int are usually the same- this is because both usually fit into a register in the CPU. So if one wants to do a quick and dirty cast of a pointer to an int or the other way, declare a union.
2) Union can be used in a command or message protocol where different size messages are sent and received. Each message type will hold different information but each will have a fixed part (probably a struct) and a variable part bit.
3) Unions are the same size, it makes sense to only send the meaningful data and not wasted space.
4) A union is a collection of variables of different types, just like a structure. However, with unions, one can only store information in one field at any one time.
5) One can picture a union as like a chunk of memory that is used to store variables of different types. Once a new value is assigned to a field, the existing data is wiped over with the new data.
6) A union can also be viewed as a variable type that can contain many different variables (like a structure), but only actually holds one of them at a time (not like a structure). This can save memory if anyone has a group of data where only one of the types is used at a time. The size of a union is equal to the size of its largest data member.

Ques 10) Differentiate between structure and union.

Or

Compare structure with union.

Ans: Difference between Structure and Union

The following **table below** illustrating the difference between structure and union:

Table

Basis	Structure	Union
Definition	A structure contains an ordered group of data objects.	A union is an object similar to a structure except that all of its members start at the same location in memory.
Access Members	One can access all the members of structure at any time.	Only one member of union can be accessed at any time.

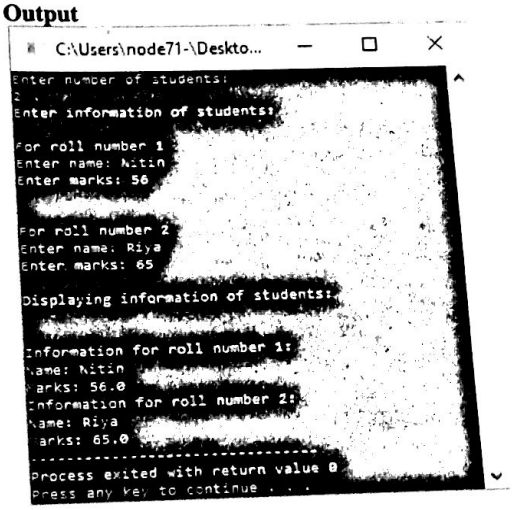
Memory Allocation	Memory is allocated for all variables.	Allocates memory for variable which variable require more memory.
Initialization	All members of structure can be initialized	Only the first member of a union can be initialized.
Keyword	'struct' keyword is used to declare structure.	'union' keyword is used to declare union.
Syntax	struct struct_name { structure member 1; structure member 2; --------- --------- structure member n; }struct_var_nm;	union union_name { union member 1; union member 2; --------- --------- union member n; }union_var_nm;
Example	struct item_mst { int rno; char nm[50]; }it;	union item_mst { int rno; char nm[50]; }it;

Ques 11) Write a C program to store the student details using union.

Program: Storing the Student Details Using Union

```
#include <stdio.h>
union student
{
  char name[50];
  int roll;
  float marks;
};
int main()
{
    union student s[10];
    int i,n;
    printf("Enter number of students:\n");
    scanf("%d", &n);
    printf("Enter information of students:\n");
    for(i=0;i<n;++i)
    {
      s[i].roll=i+1;
      printf("\nFor roll number %d\n",s[i].roll);
      printf("Enter name: ");
      scanf("%s",s[i].name);
      printf("Enter marks: ");
      scanf("%f",&s[i].marks);
      printf("\n");
    }

    printf("Displaying information of students:\n\n");
    for(i=0;i<n;++i)
    {
        printf("\nInformation for roll number %d:\n",i+1);
        printf("Name: ");
        puts(s[i].name);
        printf("Marks: %.1f",s[i].marks);
    }
    return 0;
}
```

Output



POINTERS

Ques 12) What is pointer? How is a pointer initialized and declared? How is variable accessed through its pointer? Give example of it.

Or

Explain the concept and declaration of Pointers with a suitable example in 'C'. Also discuss the accessing a variable through its pointer.

Ans: Pointers

Pointers are widely used in programming; they are used to refer to memory location of another variable without using variable identifier itself. They are mainly used in linked lists and call by reference functions.

With pointer one can perform lot many operations by using memory addresses. It is possible to access and display the memory location of a variable using '&' operator.

A pointer variable holds the memory address of any type of variable. The pointer variable and normal variable should be of the same type. The pointer is denoted by '*' symbol.

A pointer holds an address rather than a value, it has two parts. The pointer itself holds the address. That addresses points to a value.

Declaration of Pointer

C programming language supports pointers to different data types – int, float, double, char, etc. Every pointer can point to either a single value or to multiple values.

Whether a pointer is to point to a single value or to multiple values (in the case of a pointer to an array) is decided by its specific use with the associated constructs of the program. In C a pointer variable is declared by using the pointer (*) operator.

Syntax:

```
type *var_name;
```

Here, **type** is the pointer's base type; it must be a valid C data type and **var-name** is the name of the pointer variable.

For example, let consider the following code:

```
int *pi;      /* pi is an integer pointer/pointer to an
              integer */
float *pf;    /* pf is a float pointer */
char *pc;     /* pc is a character pointer */
```

Here, the pointer variables pi, pf and pc contain a pointer to the corresponding data item. Consider the declaration of int *pi; Here, pi is a pointer variable which is capable of holding the address of the memory location containing an integer. When the compiler comes across this statement, it allocates a location that can store the address of an integer location.

Initialisation of Pointer

Pointer variables can be initialized in their definitions, just like many other variables in C.

For example, the following two statements allocate storage for the two cells iresult and piresult:

1) int iresult;
2) int *piresult = &iresult;

The variable iresult is an ordinary integer variable, and piresult is a pointer to an integer.

Additionally, the code initializes the pointer variable piresult to the address of iresult. We are not initializing *piresult (which would have to be an integer value) but piresult (which must be an address to an integer).

Second statement in the preceding listing can be translated into the following two equivalent statements:

```
int *piresult;
piresult = &iresult;
```

Accessing a Variable through Its Pointer

The indirection operator (*) is used to access the value of a variable by its ptr.

Once a pointer has been assigned the address of a variable, the question remains as to how to access the value of the variable using the pointer. This is done by using another unary operator * (asterisk), usually known as the **indirection operator**. Another name for the indirection operator is thef **dereferencing operator**. Consider the following statements:

```
int quantity, *p, n:
quantity = 179:
p = &quantity:
n= *p:
```

The first line declares quantity and n as integer variables and p as a pointer variable pointing to an integer. The second line assigns the value 179 to quantity and the third line assigns the address of quantity to the pointer variable p. The fourth line contains the indirection operatot *. When the operator* is placed returns the value of the variable of which the pointer value is the address. In this case, *p returns the value of the variable quantity, because p is the address of quantity. The * can be remembered as 'value at address'. Thus the value of n would be 179. The two statements

```
p = &quantity:
n = *p:
```

are equevalent to

```
n = *&quantity:
```

which in turn is equivalent to

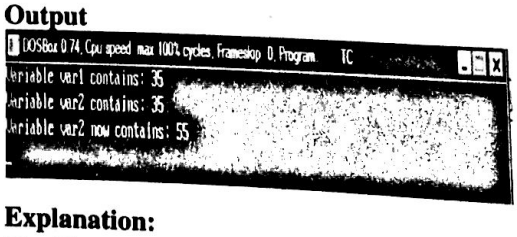
```
n = quantity:
```

In C, the assignment of pointers and addresses is always done symbolically. by means of symbolic names. You cannot access the value stored at the address 5368 by writing *5368. It will not work.

Program: Illustrating Declaration, Initialisation and Accessing a variable of Pointer

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int *iptr, var1, var2;
        iptr = &var1;       /* initialize pointer iptr */
        *iptr = 25;         /* dereference iptr, var1 = 25,
        */
        *iptr += 10;        /* varl = varl + 10; */
        /* next statement prints var1, it should be 35 */
        clrscr();
        printf("Variable var1 contains: %i\n", var1);
        var2 = *iptr;       /* same as var2 = var1; */
        /* next statement prints var2, it also contains 35
        */
        printf("Variable var2 contains: %i\n", var2);
        iptr = &var2; /* iptr now points to var2 */
        *iptr += 20;  /* increase contents of var2 by 20,
        */
        /* next statement prints var2, it now contains 55
        */
        printf("Variable var2 now contains: %i\n", var2);
}
```

Output



Explanation:

1) The asterisk (*) used as an indirection operator has a different meaning from the asterisk used to declare pointer variables:
   int *ptr; /* This is a pointer declaration */

```
*ptr = 100;  /* The LHS does pointer indirection */
```

2) Indirection allows the contents of a variable to be accessed and manipulated without using the name of the variable.
3) All variables that can be accessed directly (by their names) can also be accessed indirectly by means of pointers. The power of pointers becomes evident in situations where indirect access is the only way to access variables in memory.

Ques 13) Why pointers are needed? Also discuss the applications and limitations of pointers.

Ans: Need of Pointer

One generally uses a pointer for the following purposes:

1) Randomly accessing any memory cell of a variable.
2) Implementation of function pointer.
3) Dynamic memory allocation.
4) Implementation of an array.
5) Read/write directly into hardware.
6) Functional calling convention "Pass by Reference".

Applications and Uses of Pointers

1) Pointers are more efficient in handling arrays and data tables.
2) Pointers can be used to return multiple values from a function via function arguments.
3) Pointers permit references to functions and thereby facilitating passing of functions as arguments to other functions.
4) The use of pointer arrays to character strings results in saving of data storage space in memory.
5) Pointers allow C to support dynamic memory management.
6) Pointers provide an efficient tool for manipulating dynamic data structures such as structures, linked lists, queues, stacks and trees.
7) Pointers reduce length & complexity of programs.
8) They increase the execution speed & thus reduce the program execution time.

Limitations of Pointers

1) Uninitialised pointers might cause segmentation fault.
2) Dynamically allocated block needs to be freed explicitly. Otherwise, it would lead to memory leak.
3) Pointers are slower than normal variables.
4) If pointers are updated with incorrect values, it might lead to memory corruption.
5) If the programmer is not careful and consistent with the use of pointers, the program may crash.

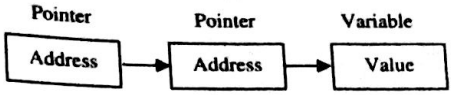
Ques 14) What is chain of pointer? Discuss with suitable examples.

Or

Discuss the concept of chain of pointer with proper syntax example.

Ans: Chain of Pointer

A pointer to a pointer is a form of multiple indirection, or a chain of pointers. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.



A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name. **For example, the** following declaration declares a pointer to a pointer of type int:

```
int **var;
```

When a target value is indirectly pointed to by a pointer to a pointer, accessing that value requires that the asterisk operator be applied twice, as is shown below in the program:

```
#include <stdio.h>

int main () {

  int  var;
  int  *ptr;
  int  **pptr;

  var = 3000;

  /* take the address of var */
  ptr = &var;

  /* take the address of ptr using address of operator & */
  pptr = &ptr;

  /* take the value using pptr */
  printf("Value of var = %d\n", var );
  printf("Value available at *ptr = %d\n", *ptr );
  printf("Value available at **pptr = %d\n", **pptr);

  return 0;
}
```

Output

```
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000
```

Ques 15) Discuss the pointer expression with suitable syntax and examples.

Or

Write a program to illustrate the pointer expression with pointer arithmetic.

Ans: Pointer Expression

Pointer expression is a linear combination of pointer variables, variables and operators (+, -, ++, --). The pointer expression gives either numerical output or address output.

Pointer Assignment
The general form of pointer assignment is
variable = pointer expression

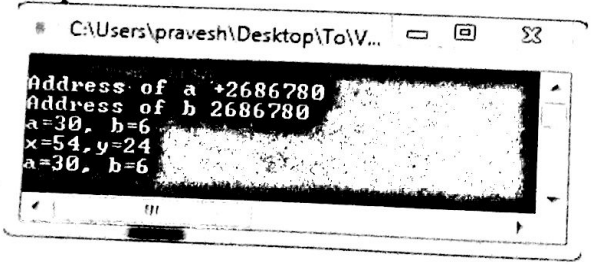
Example
```
1)  int x=5,y;
2)  int *p, *q;
3)  p = &x;
4)  q = &y;
5)  *q = *p +10 => pointer assignment.
```

Example
```
1)  y = *p1 * *p2;
2)  sum = sum + *p1;
3)  z = 5* - *p2/p1;
4)  *p2 = *p2 + 10;
```

C language allows to programmers to add integers, to subtract integers from pointers as well as to subtract one pointer from the other. He/she can also use short hand operators with the pointers p1+=; sum+=*p2; etc., and can also compare pointers by using relational operators the expressions such as p1 > p2, p1= = p2 and p1!=p2 are allowed.

Program: To illustrate the pointer expression with pointer arithmetic.
```
#include<stdio.h>
#include<conio.h>

main()
{
int *ptr1, *ptr2;
int a,b,x,y,z;
a=30;b=6;

ptr1= &a;
ptr2= &a;

x=*ptr1+*ptr2-6;
y=6*-*ptr1 / *ptr2+30;

printf("\nAddress of a +%u",ptr1);
printf("\nAddress of b %u",ptr2);
printf("\na=%d, b=%d",a,b);
printf("\nx=%d,y=%d",x,y);

ptr1=ptr1 + 70;
ptr2= ptr2;

printf("\na=%d, b=%d",a,b);
getch();}
```

Output



Ques 16) Explain operation on pointers in brief.
Or
Explain the increment, decrement and comparison operation of a pointer.
Or
Discuss the pointer increment and scale factor with suitable example.

Ans: Operation on Pointers

1) **Incrementing a Pointer:** The following program increments the variable pointer to access each succeeding element of the array:
```
#include <stdio.h>
const int MAX = 3;
int main ()
{ int  var[] = {10, 100, 200};
  int  i, *ptr;
  /* let us have array address in pointer */
  ptr = var;
  for ( i = 0; i < MAX; i++)
  { printf("Address of var[%d] = %x\n", i, ptr );
    printf("Value of var[%d] = %d\n", i, *ptr );
    /* move to the next location */
    ptr++;  }
  return 0;}
```

Output
```
Address of var[0] = bf882b30
Value of var[0] = 10
Address of var[1] = bf882b34
Value of var[1] = 100
Address of var[2] = bf882b38
Value of var[2] = 200
```

2) **Decrementing a Pointer:** The same considerations apply to decrementing a pointer, which decreases its value by the number of bytes of its data type as shown below:
```
#include <stdio.h>
const int MAX = 3;
int main ()
{ int  var[] = {10, 100, 200};
  int  i, *ptr;

  /* let us have array address in pointer */
  ptr = &var[MAX-1];
  for ( i = MAX; i > 0; i--)
  { printf("Address of var[%d] = %x\n", i, ptr );
    printf("Value of var[%d] = %d\n", i, *ptr );
    /* move to the previous location */
    ptr--;  }  return 0;}
```

Output
```
Address of var[3] = bfedbcd8
Value of var[3] = 200
Address of var[2] = bfedbcd4
Value of var[2] = 100
Address of var[1] = bfedbcd0
Value of var[1] = 10
```

3) **Pointer Comparisons:** Pointers may be compared by using relational operators, such as ==, <, and >. If p1 and p2 point to variables that are related to each other, such as elements of the same array, then p1 and p2 can be meaningfully compared. The following program illustrates the pointer comparison.

```
#include <stdio.h>
const int MAX = 3;
int main ()
{ int  var[] = {10, 100, 200};
  int  i, *ptr;
  /* let us have address of the first element in pointer
*/
  ptr = var;
  i = 0;
  while ( ptr <= &var[MAX - 1] )
  {
    printf("Address of var[%d] = %x\n", i, ptr );
    printf("Value of var[%d] = %d\n", i, *ptr );
    /* point to the previous location */
    ptr++;     i++;   }
  return 0;
}
```

Output

```
Address of var[0] = bfdbcb20
Value of var[0] = 10
Address of var[1] = bfdbcb24
Value of var[1] = 100
Address of var[2] = bfdbcb28
Value of var[2] = 200
```

Scalar Factor

When we increment a pointer, its value is increased by the length of the data type. This length is called scale factors. **For example,** let p1 is an integer pointer with an intial value says-2800, then after operation p1=p1+1; the value of it will be 2802, and not 2801.

The length of various data types are as follows:

Character	1 byte
integer	2 bytes
float	4 bytes
long integer	4 bytes
double	8 bytes

Program below illustrates the pointer increment nd scale factor:

```
void main()
{
int i=4, *j, *k;
clrscr();
j=&i
printf("the value of j:%d\n", j);
j=j+1;
printf("the value of j:%d\n", j)
k=j+3;
printf("the value of k:%d\n", k)
getch();
}
```

Ques 17) Discuss the concept of pointers and arrays with suitable program.

Or

Discuss the concept of pointers and arrays with suitable example.

Ans: Pointers and Arrays

Pointer and arrays are inseparably related, but they are not synonymous. To illustrate this, consider the following declarations:

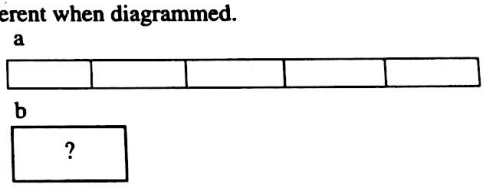
```
int a[5];
int *b;
```

Can a and b be used interchangeably? Both have pointer values, and you can use indirection and subscripts on either one. They are, nevertheless, quite different.

Declaring an array sets aside space in memory for the indicated number of elements, and then creates the array name whose value is a constant that points to the beginning of this space.

When an array is declared, compiler allocates the base address and sufficient amount of storage to contain all elements in contiguous memory locations.

Declaring a pointer variable reserves space for the pointer itself, but that is all. Space is not allocated for any integers. Furthermore, the pointer variable is not initialized to point to any existing space. If it is an automatic variable it is not initialized at all. These two declarations look very different when diagrammed.



Thus, after the declarations the expression *a is perfectly legal, but the expression *b is not. *b will access some indeterminate location in memory or cause the program to terminate.

On the other hand, the expression b++ will compile, but a++ will not because the value of a is a constant.

Example: Let in a[5] = {1, 2, 3, 4, 5} and base address starts from 1000

Then, array of a will be as:

Base address	→	1000	1002	1004	1006	1008
Value	→	1	2	3	4	5
Elements	→	a[0]	a[1]	a[2]	a[3]	a[4]

If, we want to access the base address of a[0], then we can access by pointer 'b' as:

```
int*b;
b=a;
b=& a[0]; // This will access base address of a[0].
```

For accessing other values of 'a' by using pointer 'b'. We can do so by using b++ to move from one element to other.

So, b=& a[0]; // accesses base address 1000

b+1=& a[1]; // accesses base address 1002

:

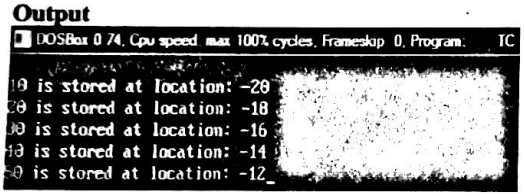
:

b+4=& a[4]; // accesses base address 1008

Program: To process the array using pointers.

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

main()
{
int a[5] = {10, 20, 30, 40, 50};
int i, *ptr;
/* Starting address of array is assigned */
ptr=&a[0];
clrscr();
for(i=0; i<5; i++)
printf("\n%d is stored at location: %d", *(ptr+i), (ptr+i));
}
```

Output



Ques 18) Explain pointers to functions and array of pointers.

Ans: Pointer to Functions

C programming language allows programmers to pass a pointer to a function. To do so, simply declare the function parameter as a pointer type. A function can have a reference with the pointer data type. A pointer to a function should be declared as a pointer to the type which is returned by the function such as void, int, float, etc. When the pointer is declared, the argument of the function must also be specified.

Syntax:

return_type(*variables) (list of parameters);

For example, let consider the following:

```
void (*ptr) (int, int, float)
float (*ptr) (int, float, char);
void (*ptr) (float, float, int)
```

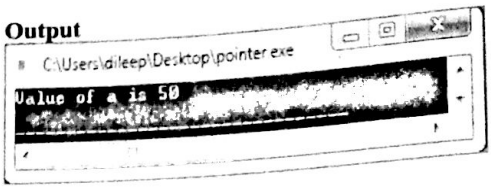
When a pointer to a function has been declared, the address of the function must be assigned to a pointer as:

```
ptr = &add;
```

Program: Illustrating Pointer to Function

```
#include <stdio.h>
// A normal function with an int parameter and void return type
void fun(int a)
{   printf("Value of a is %d\n", a);
}
int main()
{  // fun_ptr is a pointer to function fun()
   void (*fun_ptr)(int) = &fun;
   // Invoking fun() using fun_ptr
   (*fun_ptr)(50);
    return 0; }
```

Output



Array of Pointer

An array of pointer is an array of pointer variables. The pointer variables in such an array are all pointers of the same type.

For example, one can declare an array of pointers to characters or an array of pointers to integers, even an array of pointers to functions. The pointers can hold the addresses of variables or the address of the first element in an array. Arrays of pointers are often used to reference a set of character arrays.

Syntax:

<data_type> *<name>[<number-of-elements];

For example, let consider the following code:

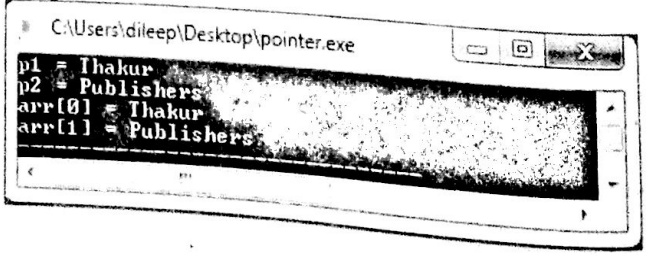
```
char *ptr[3];
```

The above line declares an array of three character pointers.

Program: Illustrating Array of Pointer

```
#include<stdio.h>
int main(void)
{
  char *p1 = "Thakur";
  char *p2 = "Publishers";
  char *arr[3];
  arr[0] = p1;
  arr[1] = p2;
  printf("p1 = %s",p1);
  printf("\np2 = %s",p2);
  printf("\narr[0] = %s",arr[0]);
  printf("\narr[1] = %s",arr[1]);
  return 0;
}
```

Output



Explanation: In the above code, there are two pointers pointing to two strings. Then we declared an array that can contain two pointers. We assigned the pointers 'p' and 'p2' to the 0, 1 and 2 index of array.

Ques 19) Give the differences between Array and Pointer. Write a program to process the array using pointers.

Ans: Differences between Array and Pointer

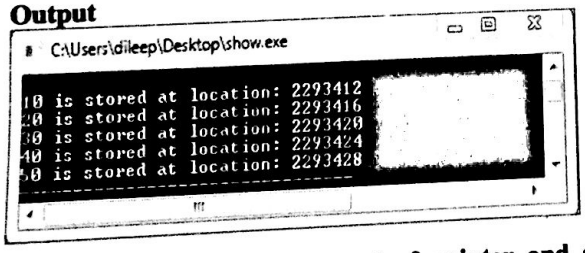
Table: Differences between Array and Pointer

Array	Pointer
Holds data.	Holds the address of data.
Data is accessed directly, so for a[i] simply retrieve the contents of the location.	Data is accessed indirectly, so you first retrieve the contents of the pointer, load that as an address, then retrieve its contents.
Commonly used for holding a fixed number of elements of the same type of data.	Commonly used for dynamic data structures.
Implicitly allocated and deallocated.	Commonly used with malloc(), free().
It is a named variable in its own right.	It typically points to anonymous data.
Size of (array-name) gives the number of bytes occupied by the array.	Size of (p) returns number of bytes used to store the pointer variable p.

Program: Illustrating Processing the Array Using Pointers

```
#include<stdio.h>
#include<conio.h>
void main()
{   int a[5] = {10, 20, 30, 40, 50};
    int i, *ptr;
    /* Starting address of array is assigned */
    ptr =&a[0];
    for(i=0; i<5; i++)
    printf("\n%d is stored at location: %d", *(ptr+i), (ptr+i));
}
```

Output



Ques 20) Explain the concept of pointer and string with suitable example.

Ans: Pointer and String

String is a collection of characters and it can also be called character array. The pointers are beneficial in character-based applications also. **For example,** character pointer variable is declared and assigned to the variable as follows:

```
char *ptr;
```

Here *ptr is a pointer variable, which points to the array of characters.

The string value can be assigned to the variable as below:

```
char name[] = "Karthikeyan";
```

The starting address (base address) of the string is taken from any one of the following ways with the help of the above declaration:

```
name (or) &name[0]
```

/* Both are points to the starting address of the string */

The statements:

```
char *ptr;
char name[] = "Karthi";
```

are declarative statements and the assignment statement is,

```
ptr = name;
```

Here, the starting address of the character array variable or string variable name is assigned to the pointer variable ptr.

Now both name and ptr point to the same memory location. The **figure 5.4** illustrates the above:

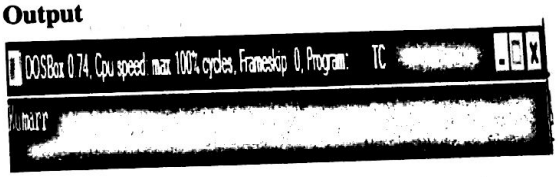
K	a	r	t	h	i	\0	
1000	1001	1002	1003	1004	1005	1006	1007

Figure 5.4

Program: Accessing String Values Using Pointers.

```
main()
{char *ptr, name[] = "Kumarr";
int i, l;
clrscr();
ptr = name;
/*Starting address is assigned to ptr*/
l=strlen(name);
for(i=0; i<l; i++)
printf("%c", *(ptr+i)); }
```

Output



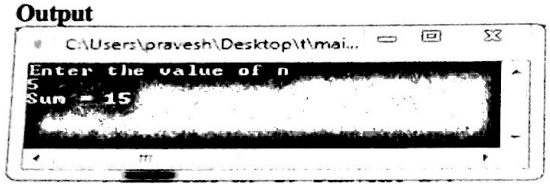
EXAMPLE PROGRAMS

Ques 21) Program to add 1 + 2 +...+ 100 using a pointer to a function method.

Ans: Program to Add Series Using Pointer to a Function

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

main()
{    int n, a, sum(int n);
     int(*ptr)(int n);
     printf("Enter the value of n\n");
     scanf("%d", &n);
     ptr = &sum;
     a = (*ptr)(n);
     printf("Sum = %d\n", a);
     getch(); }
int sum(int n)
{    int i, j;
     j = 0;
     for(i=1; i<=n; i++)
     { j += i;}
     return(j);}
```

Output

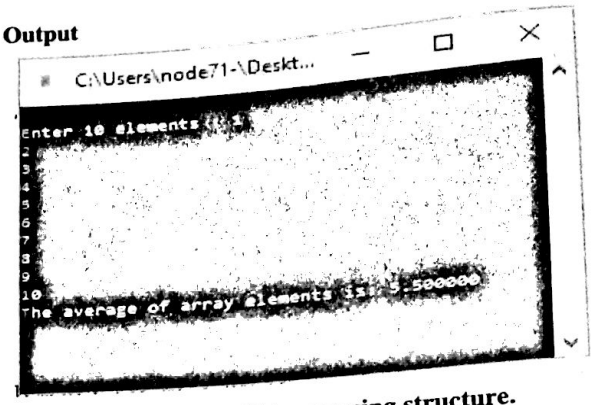


Ques 22) Write a program in C-language to find the average of elements of a given one dimensional array using pointers.

Ans: C Program to Find the Average of Elements of Given One Dimensional Array using Pointers

```
#include<stdio.h>
#include<conio.h>
void main() {
  int numArray[10];
  int i, sum = 0, avg;
  int *ptr;

  printf("\nEnter 10 elements : ");

  for (i = 0; i < 10; i++)
    scanf("%d", &numArray[i]);

  ptr = numArray; /* a=&a[0] */

  for (i = 0; i < 10; i++) {
    sum = sum + *ptr;
    ptr++;   }
     avg = sum/10;
printf("The average of array elements is: %d", avg);

getch();}
```

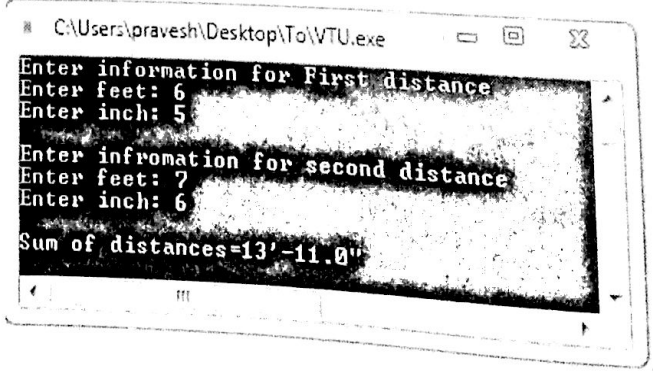
Output



Program: To add two distance using structure.

```
#include <stdio.h>
#include <conio.h>
struct Distance{
   int feet;
   float inch;
}distance1,distance2,sum;

main(){
   printf("Enter information for First distance\n");
   printf("Enter feet: ");
   scanf("%d",&distance1.feet);
   printf("Enter inch: ");
   scanf("%f",&distance1.inch);
   printf("\nEnter infromation for second distance\n");
   printf("Enter feet: ");
   scanf("%d",&distance2.feet);
   printf("Enter inch: ");
   scanf("%f",&distance2.inch);
   sum.feet=distance1.feet+distance2.feet;
   sum.inch=distance1.inch+distance2.inch;

/* If inch is greater than 12, changing it to feet. */
   if (sum.inch>12.0)
   {
      sum.inch=sum.inch-12.0;
      ++sum.feet;
   }
   printf("\nSum of distances=%d\'-
%.1f\"",sum.feet,sum.inch);
   getch();}
```

Output

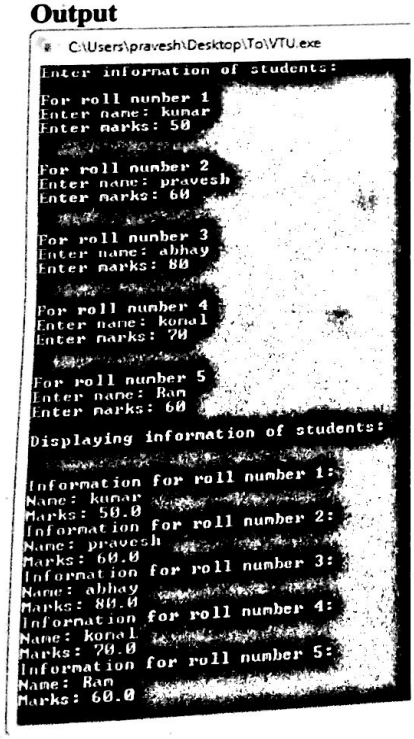


Explanation: In this program, a structure(Distance) is defined with inch and feet as its members. Then, three variables(distance1, distance2 and sum) of struct Distance type is created. Two variables(distance1 and distance2) are used for taking distance from user and the sum of two distance is stored in variable sum and then, displayed.

Program: To store information of 5 students using structure.

```
#include <stdio.h>
#include <conio.h>
struct student{
   char name[50];
   int roll;
   float marks;
};
main(){
   struct student s[5];
   int i;
   printf("Enter information of students:\n");
   for(i=0;i<5;++i) {
      s[i].roll=i+1;
      printf("\nFor roll number %d\n",s[i].roll);
      printf("Enter name: ");
      scanf("%s",s[i].name);
      printf("Enter marks: ");
      scanf("%f",&s[i].marks);
      printf("\n"); }
  printf("Displaying information of students:\n\n");
  for(i=0;i<5;++i) {
   printf("\nInformation for roll number %d:\n",i+1);
   printf("Name: ");
   puts(s[i].name);
   printf("Marks: %.1f",s[i].marks);   }
 getch();}
```

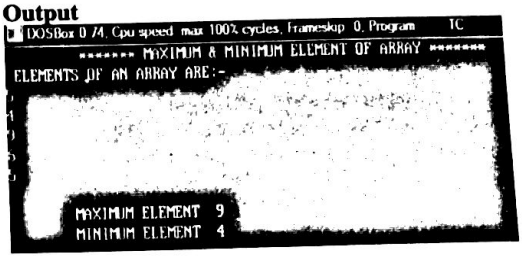
Output



Program: Find highest and lowest element in a single dimension array.

```
#include<stdio.h>
#include<conio.h>
void main() {
     int array[5],x,max,min;
     clrscr();
     printf("\t ******* MAXIMUM & MINIMUM
     ELEMENT OF ARRAY *******\n");
     printf(" ELEMENTS OF AN ARRAY ARE:- \n");
     for(x=0;x<5;x++) {
          scanf("%d",&array[x]); }
     max=array[0];
     min=array[0];
     for(x=0;x<5;x++) {
          if(max<array[x]) {
                    max=array[x]; }
                    if(min>array[x]) {
                    min=array[x]; }}

     printf("\n\tMAXIMUM ELEMENT %d",max);
     printf("\n\tMINIMUM ELEMENT %d",min);
     getch(); }
```

Output

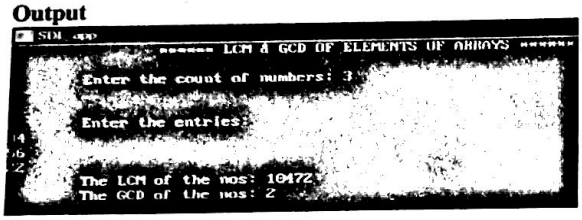


Program: Finding LCM and GCD.

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

void main() {
  int a[20], n, i, j, c, max, min;
  unsigned long prod;
  clrscr();
  printf("\t\t****** LCM & GCD OF ELEMENTS OF ARRAYS
******\n\n");
  printf("\tEnter the count of numbers: ");
  scanf("%d",&n);
  printf("\n\n\tEnter the entries: \n");
  for(i=0;i<n;i++) {
   scanf("%d",&c);
   if(c>0)
    a[i]=c;
   else {
    printf("Invalid Entry");
    return; } }

  max=a[0];
   for(i=0;i<n;i++)
    if(a[i]>=max)
     max=a[i];
   min=a[0];
   for(i=0;i<n;i++)
    if(a[i]<min)
     min=a[i];
```

```
    for(i=0,prod=1;i<n;i++)
      prod=prod*a[i];
    for(i=max;i<=prod;i+=max) {
      c=0;
      for(j=0;j<n;j++)
        if(i%a[j]==0)
          c+=1;
      if(c==n) {
        printf("\tThe LCM of the nos: %d\n",i);
        break;  } }

    for(i=min;i>0;i--) {
      if (min%i==0) {
        c=0;
        for(j=0;j<n;j++)
          if(a[j]%i==0)
            c+=1;  }
      if(c==n)
      {
        printf("\tThe GCD of the nos: %d\n",i);
        break;  } }
    getch();}
```

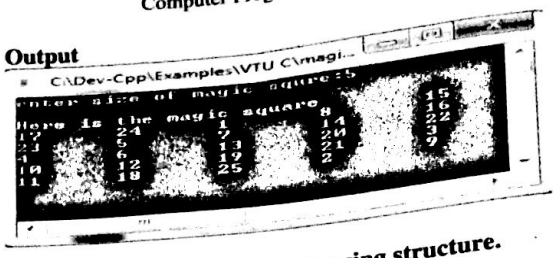
Output



Program: Magic Square.

```
#include<stdio.h>
#include<conio.h>
int main()
{
  int sqr[25][25]={(0)},n,i,j,k,p,q;
  //clrscr();
  printf("enter size of magic squre:");
  label:
  scanf("%d",&n);
  if(n%2!=1||n>=25) {
        printf("Enter the odd value in the range of 25\n");
        goto label; }
 i=0;
 j=(n)/2;
 sqr[i][j]=1;
 for(k=2;k<=(n*n);k++) {
       p=i-1<0?n-1 : i-1;
       q=(j+1)%n;
       if(sqr[p][q]==0) {
            i=p;
            j=q; }
      else
           i++;
      sqr[i][j]=k; }
 printf("\nHere is the magic square\n");
 for(i=0;i<n;i++) {
    for(j=0;j<n;j++)
      printf("%d\t",sqr[i][j]);
    printf("\n"); }
getch();
return 0;}
```

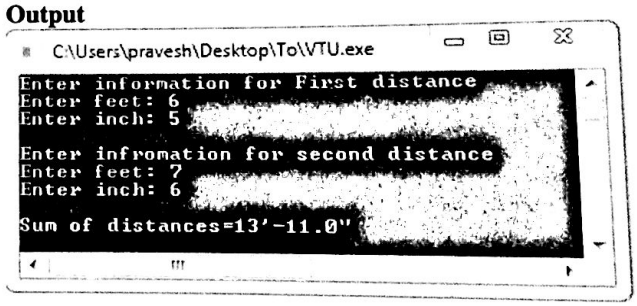
Output



Program: To add two distance using structure.

```
#include <stdio.h>
#include <conio.h>
struct Distance{
   int feet;
   float inch;
}distance1,distance2,sum;
main(){
   printf("Enter information for First distance\n");
   printf("Enter feet: ");
   scanf("%d",&distance1.feet);
   printf("Enter inch: ");
   scanf("%f",&distance1.inch);
   printf("\nEnter infromation for second distance\n");
   printf("Enter feet: ");
   scanf("%d",&distance2.feet);
   printf("Enter inch: ");
   scanf("%f",&distance2.inch);
   sum.feet=distance1.feet+distance2.feet;
   sum.inch=distance1.inch+distance2.inch;

/* If inch is greater than 12, changing it to feet. */
   if (sum.inch>12.0)
   {
     sum.inch=sum.inch-12.0;
     ++sum.feet;
   }
   printf("\nSum of distances=%d\'-%.1f\"",sum.feet,sum.inch);
   getch();}
```

Output



Explanation: In this program, a structure(Distance) is defined with inch and feet as its members. Then, three variables(distance1, distance2 and sum) of struct Distance type is created. Two variables(distance1 and distance2) are used for taking distance from user and the sum of two distance is stored in variable sum and then, displayed.