UNIT – VII

POINTERS:

One of the powerful features of C is ability to access the memory variables by their memory address. This can be done by using Pointers. The real power of C lies in the proper use of Pointers.
A pointer is a variable that can store an address of a variable (i.e., 112300).We say that a pointer points to a variable that is stored at that address. A pointer itself usually occupies 4 bytes of memory (then it can address cells from 0 to 232-1).

Advantages of Pointers:

1. A pointer enables us to access a variable that is defined out side the function.
2. Pointers are more efficient in handling the data tables.
3. Pointers reduce the length and complexity of a program.
4. They increase the execution speed.

Definition :

A variable that holds a physical memory address is called a pointer variable or Pointer.

Declaration :

Datatype * Variable-name;

Eg:- int *ad; /* pointer to int */

char *s; /* pointer to char */ float *fp; /* pointer

to float */ char **s; /* pointer to variable that is a pointer

to char */

A pointer is a variable that contains an address which is a location of another variable in memory.

Consider the Statement
p=&i; Here „&‟ is
called address of a variable.
„p‟ contains the address of a variable i.

The operator <u>&</u> returns the memory address of variable on which it is operated, this is called <u>Referencing</u>.
The * operator is called an indirection operator or dereferencing operator which is used to display the contents of the Pointer Variable.

Consider the following Statements :

```
int *p,x;
x =5;
p= &x;
```

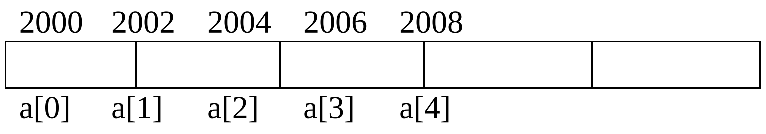
Assume that x is stored at the memory address 2000. Then the output for the following printf statements is :

	Output
Printf("The Value of x is %d",x);	5
Printf("The Address of x is %u",&x);	2000
Printf("The Address of x is %u",p);	2000
Printf("The Value of x is %d",*p);	5
Printf("The Value of x is %d",*(&x));	5

POINTERS WITH ARRAYS :

When an array is declared, elements of array are stored in contiguous locations. The address of the first element of an array is called its base address.

Consider the array



The name of the array is called its base address.

i.e., a and k& a[20] are equal.

Now both a and a[0] points to location 2000. If we declare p as an integer pointer, then we can make the pointer P to point to the array a by following assignment

P = a;

We can access every value of array a by moving P from one element to another.

i.e.,	P	points to 0th element
	P+1	points to 1st element
	P+2	points to 2nd element
	P+3	points to 3rd element
	P +4	points to 4th element

Reading and Printing an array using Pointers :

```
        main()
        {
                int *a,i;
  printf("Enter five elements:");   for
(i=0;i<5;i++)
                        scanf("%d",a+i);
                        printf("The array elements are:");
                for (i=0;i<5;i++)
                                printf("%d", *(a+i));
        }
```

In one dimensional array, a[i] element is referred by

(a+i) is the address of ith element.

* (a+i) is the value at the ith element.

In two-dimensional array, a[i][j] element is represented as

((a+i)+j)

STRUCTURES :

A Structure is a collection of elements of dissimilar data types. Structures provide the ability to create user defined data types and also to represent real world data.

Suppose if we want to store the information about a book, we need to store its name (String), its price (float) and number of pages in it(int). We have to store the above three items as a group then we can use a structure variable which collectively store the information as a book.

Structures can be used to store the real world data like employee, student, person etc.

Declaration :

The declaration of the Structure starts with a Key Word called Struct and ends with ; . The Structure elements can be any built in types.

```
        struct <Structure name>
        {
                Structure element 1;
                Structure element 2;
-
-
-
                Structure element n;
        };
```

Then the Structure Variables are declared as

```
                struct < Structure name > <Var1,Var2>;

                Eg:-    struct emp
                        {
                                int empno.
                        char ename[20];
                                float sal;
                        };
                struct emp  e1,e2,e3;
```

The above Structure can also be declared as :

```
        struct emp                                      struct
        {                                               {
                int empno;                              int empno;
                char ename[20];         (or)            char ename[20];
                float sal;                              float sal;
        }e1,e2,e3;                                      }e1,e2,e3;
```

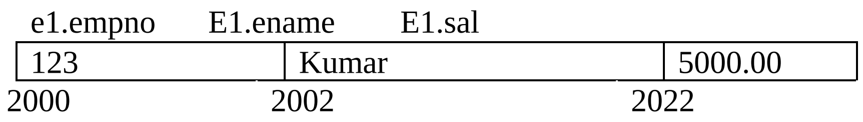
Initialization :

Structure Variables can also be initialised where they are declared.

```
struct emp
          {
                    int empno;
char ename[20];
                    float sal;
          };
          struct emp e1 = { 123,"Kumar",5000.00};
```

To access the Structure elements we use the .(dot) operator.
To refer empno we should use e1.empno
To refer sal we whould use e1.sal

Structure elements are stored in contiguous memory locations as shown below. The above Structure occupies totally 26 bytes.



1. Program to illustrate the usage of a Structure.

```
               main()
               {
                    struct emp
                    {
                              int empno;
                              char ename[20];
                              float sal;
                    };
                    struct emp e;   printf ("  Enter
          Employee number: \n");
                    scanf("%d",&e.empno);
                    printf ("  Enter Employee Name: \n");
                    scanf("%s",&e.empname);
                    printf ("  Enter the Salary: \n");
                    scanf("%f",&e.sal);
                    printf (" Employee No = %d", e.empno);
                    printf ("\n Emp Name = %s", e.empname);
                    printf ("\n Salary  = %f", e.sal);
               }
```

/* Program to read Student Details and Calculate total and average using structures */

```
          #include<stdio.h>
main()
          {
                    struct stud
                    {
                              int rno;
                              char sname[20];
                              int m1,m2,m3;
                    };
                    struct stud s;
          int tot;
float avg;

                    printf("Enter the student roll number: \n");
                    scanf("%d",&s.rno);
                    printf("Enter the student name: \n");
                    scanf("%s",&s.sname);
                    printf("Enter the three subjects marks: \n");
                    scanf("%d%d%d",&s.m1,&s.m2,&s.m3);

                    tot = s.m1 + s.m2 +s.m3;
                    avg = tot/3.0;



                    printf("Roll Number : %d\n",s.rno);
printf("Student Name: %s\n",s.sname);                    printf("Total
Marks : %d\n",tot);
                    printf("Average : %f\n",avg);
          }
```

/* Program to read Item Details and Calculate Total Amount of Items*/

```
          #include<stdio.h>
main()
          {
                    struct item
                    {
                              int itemno;
          char itemname[20];
float rate;                              int qty;
                    };
                    struct item i;
                    float tot_amt;
                    printf(“Enter the Item Number \n”);
scanf(“%d”,&i.itemno);                    printf(“Enter
the Item Name \n”);
scanf(“%s”,&i.itemname);
                    printf(“Enter the Rate of the Item \n”);
scanf(“%f”,&i.rate);
                    printf(“Enter the number of %s purchased ”,i.itemname);     scanf(“%d”,&i.qty);

                    tot_amt = i.rate * i.qty;

                    printf(“Item Number: %d\n”,i.itemno);
          printf(“Item Name: %s\n”,i.itemname);
printf(“Rate: %f\n”,i.rate);                printf(“Number of
Items: %d\n”,i.qty);
                    printf(“Total Amount: %f”,tot_amt);
          }
```

ARRAY OF STRUCTURES :

To store more number of Structures, we can use array of Structures. In array of Structures all elements of the array are stored in adjacent memory location.

/* Program to illustrate the usage of Array of Structures.

```
                    main()
                    {
                              struct book
                              {
                                        char name[20];
                              float price;
          int pages;
                              };
                              struct book b[10];
                              int i;
                              for (i=0;i<10;i++)
                              {
                                        print(“\n Enter Name, Price and Pages”);
          scanf(“%s%f%d”, b[i].name,&b[i].price,&b[i].pages);
                              }
                              for (i i=0;i<10;i++)
                                        printf(“ \n%s%f%d”, b[i].name,b[i].price,b[i].pages);
                    }
```

UNIONS :

Union, similar to Structures, are collection of elements of different data types. However, the members within a union all share the same storage area within the computer‟s memory, whereas each member within a Structure is assigned its own unique Storage area.

Structure enables us to treat a member of different variables stored at different places in memory, a union enables us to treat the same space in memory as a number of different variables. That is, a union offers a way for a section of memory to be treated as a variable of one type on one occasion and as a different variable of a different type on another occasion.

Unions are used to conserve memory. Unions are useful for applications involving multiple members, where values need not be assigned to all of the members at any given time. An attempt to access the wrong type of information will produce meaningless results.

The union elements are accessed exactly the same way in which the structure elements are accessed using dot(.) operator.

The difference between the structure and union in the memory representation is as follows.

```
struct cx

{

        int i;

        char ch[2];

};

struct cx s1;
```

The Variable s1 will occupy 4 bytes in memory and is represented as

□ ---------- s.i -------□		□ s.ch[0]□	□ s.ch[1]□

4002 4003 4004 4005

The above datatype, if declared as union then

```
    union ex
    {
        int i;
char ch[2];

    }
    union ex u;
```

□--------u.i ---------------□	
□ u.ch[0]□	□ u.ch[0]□

/* Program to illustrate the use of unions */

```
main()
{

            union example

            {

                        int i;

char ch[2];


            };
            union exaple u;



            u.i = 412;



            print(“\n u.i = %d”,u.i); print(“\n

            u.ch*0+ = %d”,u.ch*0+);

            print(“\n u.ch*1+ =

            %d”,u.ch*1+);



            u.ch[0] = 50; /* Assign a new value */



            print(“\n\n u.i = %d”,u.i); print(“\n

            u.ch*0+ = %d”,u.ch[0]); print(“\n

            u.ch*1+ = %d”,u.ch*1+);
}
```

Output :

```
        u.i  = 512

        u.ch[0] = 0

        u.ch[1] =2

        u.i  = 562

        u.ch[0] = 50

        u.ch[1] =2
```

A union may be a member of a structure, and a structure may be a member of a union. And also structures and unions may be freely mixed with arrays.

/* Program to illustrate union with in a Structure */ main()

```
{

        union id

        {

                char color;

        int size;

        };
```

```
            struct {


                        char supplier[20];

            float cost;

union id desc;



 }pant,       shirt;              printf("%d\n",

sizeof(union id));  shirt.desc.color = 'w';


            printf("%c %d\n", shirt.desc.color, shirt.desc.size);

            shirt.desc.size = 12; printf("%c %d\n",

            shirt.desc.color, shirt.desc.size);


}
```