CHAPTER 10

POINTERS

Intro:

Pointers are another powerful and important tool of C language. Reasons why pointers are used are,

1) **A pointer enables us to access a variable that is defined outside 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 exec. Speed**
5) **The use of pointer array to character strings results in saving a data storage space in memory.**

What is an address?

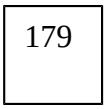
It refers to the location where you are staying with some H.No.

In 64k of computer's memory there are 0-65535 address.

Ex:

int qty = 179;

H.name = qty = variable_name



H.No. = 2002 = address

Here we can access the value of 179 by using either H.No ie address 2002 or by H.Name ie by variable name.

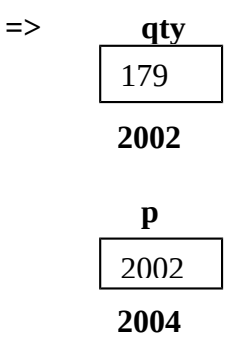
Defination :

As memory address are simply numbers, they can be assigned to some variables which can be stored in memory, like any other variable. Such variables that holds memory addresses are called pointers.

A pointer is, therefore, nothing but a variable that contains an address which is a location of another variable.

Suppose we assign a pointer to a variable like,

P = &qty;



***) WAP to print the address and value of a variable I-e char,int,float.**

Declaring and initialization of pointers:

Syntax:

Data_type *ptr;

Indicates,

1) *** ie asterisk tells that the variable ptr is a pointer variable.**
2) **Ptr needs a memory location**
3) **Ptr points to variable of type datatype**

Ex: int *p;

Initialization:

P = &qty;

Note: before a pointer is initialized, it should not be used.

<u>Accessing a variable through its pointer</u>

```
Ex:     int qty;
            int *ptr;
                int n;
                    qty = 179;
                        ptr = &qty;
                            n = *ptr;
        *prt = 229;
```

here when *ptr = 229 is encountered the value of qty is automatically changed to 229.

Like other variables, pointer variablescan be used in expressions.
Ex:

```
Y = *ptr1 * *ptr2;
Sum = sum + *ptr1;
```

Prog)

```
main()
{
        int a, b, *p1, *p2, x, y;
        a = 12,b = 20;
        p1 = &a;
        p2 = &b;
        x = *p1 * p2 – 6;
        y = 4 - p2/p1 + 10;
        printf("a=%d, b= %d, p1 = %d, p2=%d, x=%d,y=%d",
                a,b,*p1,*p2,x,y);
}
```

Pointer Increments and Decrements:

Ex:

```
int p;
p++;
p--;
```

suppose p is at 2002, p = p +1 will point to 2004 and again p—will point to 2002 back.

Pointers and Arrays:

```
int x[3]={1,2,3};
printf("%u",a);          => 65520
printf("%u",&a[0]);      => 65520
```

The name x is defined as constant pointer pointing to first element , x[0].
If p is integer pointer then we can make pointer p to point to the array x by,

```
p =  x;        => p = &x[0];
so,  p = p+1   => x[1]
     p = p +2  => x[2]
```

***) wap using pointers to compute sum of all elements stored in array.**

```
Main()
{
        int *p, sum ,I;
        int x[5] = {1,2,3,4,5};
        I=0,p=x;
        Printf("Element     value           address\n");
        While(I < 5){
        Printf("x[%d]         %d              %u\n",I,*p,p);
        Sum = sum + *p;
        I++;
        P++;
        }
        printf("\n Sum = %d\n x[0]=%u\n p = %u\n",sum,&x[0],p);
}
```

Pointers and Character Strings:-

***)wap using pointer to determine the length of character string.**

```
Main()
{
        char *name, *ptr;
         int length;
        name = "delhi";
        ptr = name;
        while(*ptr !='\0'){
                printf("%c is stored at address%u\n",*name,ptr);
                ptr++;
                }
        length = ptr – name;
        printf("\nLength of the given string is %d\n",length);
}
```

Ptr to fns :-

A fn, like a variable has an add location in the memory. It is possible to declare a ptr to a fn is declared as an argument in another fn. A ptr to a fn is declared as :

```
Type (*fptr) ( );
```

→ **The compiler that fptr is a ptr to a fn which returns type value. The parentheses around *fptr are necessary.**

But, type *qptr( );

⟶ Qptr as a fn returning a ptr to type we can make a fn ptr to point to we can make a fn by simply assigning the name of the fn to the ptr.

Ex :- double (*Pl) ( ), mul( );

Pl = mul

Declare PI as ptr to fns & mul as a fns & then make PI to pt to the fun mul. To call the fns mul, we may now use the ptr PI with the list of parameters ie

(*PI) (x,y)= => mul(x,y)}

***)WAP that uses fn ptr as a fn arrangements**

```
#define PI  3+4^2
main() {
double y(), cos(), table();
printf("\n Table of y(x)=2*x*x-x+1\x\n");
table(y,0.0,2.0,0.5);
printf("\n table of cos(x) \n\n");
table(cos,0.0,PI,0.5);
}/*EOM*/
double table(f, min. max, step)
double(*f)( ), min, max,  step;
    {
      double a, value;
      for(a=min; a <=max; a+=step)
         {value=(*f)(a);
          printf("%5.2f %10.4f\n"a,value);
    }
}
double y(double x){
return(2*x*x-x+1);}
```

o/p table of y(x)=2*x*x-x+1

0.00	1.00
0.5	1.00
1.0	2.0
1.5	4.00
2.0	7.00

table of cos(x)

0.00	1.000	1.0	0.5403
0.5	0.8776	1.5	0.0707

PTR & STRUCTURES

Ex:- struct inventory { char name[30]
Int number;
Float price;
}product[2],*ptr;

- **declares product as an array of 2elements,eachy of struct involuvantry. & Ptr as a ptr to data objects of the of the type struct involuvantry**

.'. ptr =product;
=>Would assign the zeroth element of product ptr. Ie ptr will now pt to product[0] .its means can be accessed using;->' operator.
Ptr->name
Ptr->number, ptr->price
-> is called Arrow operator
when ptr is incriminated by one, it is made to point to next record, ie product[1]the following statement will print the values of member of all the elements of the product array.
i.e. for(ptr=product; ptr <product+2; ptr++)
printf("%s %d %f\n, ptr->name,ptr->number, ptr->price);}
we can also use(*ptr).number to access the mean number.