POINTERS :

→ Pointer is a derived data type in C

→ It is built from one of the fundamental datatypes available in C.

→ The pointers are basically a variable same as any other variable.

→ However, what is the difference about them is that instead of containing actual data, they contain the memory location where the actual information is stored.

Definition : " A pointer variable is variable which will store the address of another variable."

→ Pointers are one of the most distinct and exciting feature of C language. It has added power and flexibility to the language

Advantages of using pointer :

Pointers are used frequently in C, as they offer a number of benifits to the programmers. They include :

(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 permits 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 allows C to support dynamic memory management

(6) pointers provide an efficient tool for manipulating dynamic data structures such as linked lists, queue, stack,

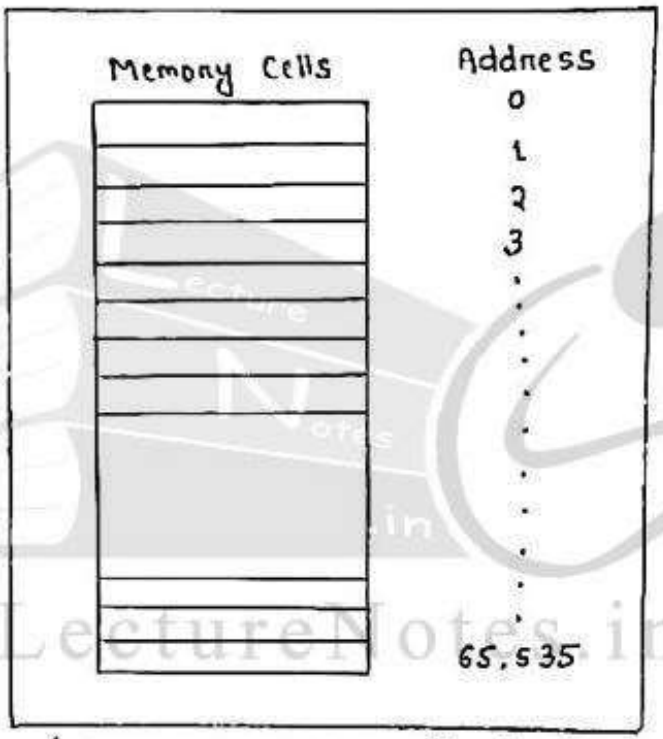
structures and trees.

(7) pointers reduce length and complexity of the program.

(8) pointers increase the execution speed and thus reduce the program execution time.

Understanding pointers :

→ The computer's memory is a sequential collection of 'storage cells' as in figure below.



(Memory Organization)

→ Each cell commonly known as a byte, has a number called address associated with it.

→ Typically, the addresses are numbered consecutively starting from zero. The last address depends on the memory size. A computer system having 64K (64 x 1024) memory will have its last address as 65,535.

→ whenever, we declare a variable, the system allocates, somewhere in the memory, an appropriate location to hold the value of the variable.

→ Since, every byte has a unique address number, this location will have its own address number.

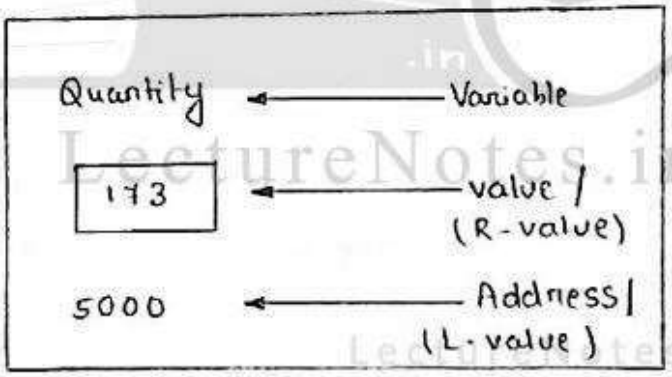
Example :

Consider the statement :

```
int quantity = 173 ;
```

→ This statement instructs the system to find a location for the integer variable quantity and puts the value 173 in that location.

→ let us assume that the system has choosen the address location 5000 for quantity. we may represent this as ;



(Representation of a variable)

→ During execution of the program, the system always associates the name quantity with the address 5000.

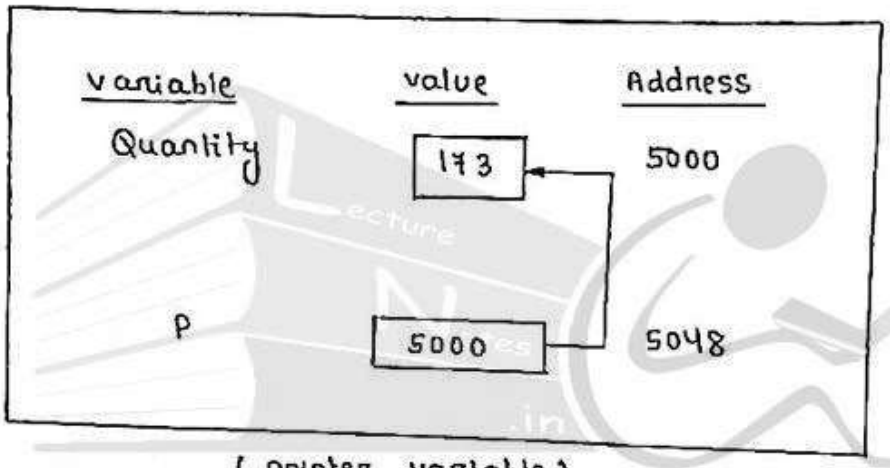
→ we may have access to the value 173 by using either the name 'quantity' or the address '5000'.

→ Since, memory addresses are simply numbers, they can be assigned to some variables, which can be stored in memory, like

any other variable. Such variables that hold memory addresses are called pointer variables.

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

→ Since, a pointer is a variable, its value is also stored in memory in another location. Suppose we assign the address of quantity to a variable p. The link between the variable p and quantity can be visualized as:



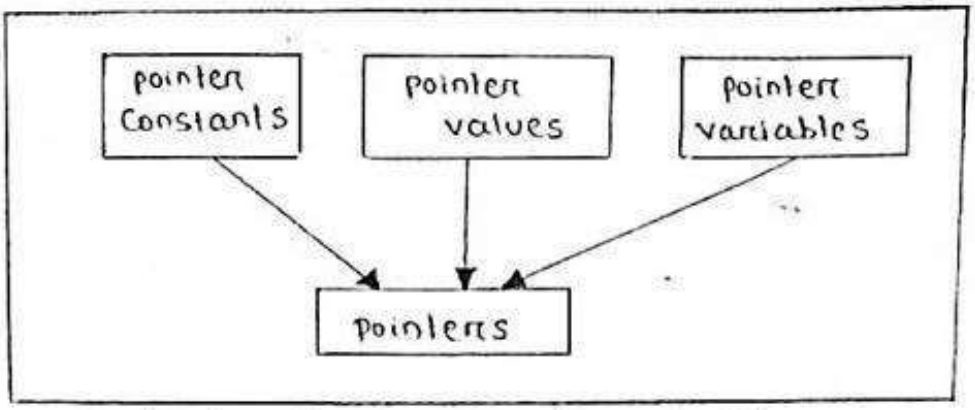
(pointer variable)

→ Since the value of the variable p is the address of the variable quantity, we may access the value of quantity by using the value of p and therefore, we say that the variable p points to the variable quantity. Thus p gets the name pointer.

Underlying Concepts of pointers:

Pointers are built on the three underlying concepts

1. pointer constants
2. pointer values
3. pointer variables.



(Underlying Concepts of pointers)

→ Memory addresses within a computer are referred to as pointer constants. We cannot change them; we can only use them to store data values.

→ We cannot save the value of a memory address directly. We can only obtain the value through the variable store there. Using the address operator (&). The value thus obtained is known as pointer value. The pointer value (i.e. the address of a variable) may change from one run of the program to another

→ Once we have a pointer value, it can be stored into another variable. The variable that contains a pointer value is called a pointer variable.

Accessing the address of a variable :

→ The actual location of the variable in memory is system dependent and therefore, the address of a variable is not known to us immediately

→ How can we then determine the address of a variable ?

→ This can be done with the help of the operator & available in C.

→ The operator & immediately preceding a variable returns the address of the variable associated with it.

→ For example :

```
p = &quantity;
```

would assign the address of 5000 (the location of quantity) to the variable p. The & operator can be remembered as 'address of'.

→ The & operator can be used only with a simple variable or an array element. The following are illegal use of address operator.

1. &125 (pointing at constants)
2. int x[10];
   &x (pointing at array names)
3. &(x+Y) (pointing at expressions)

→ If x is an array, then expression such as &x[0] and &x[i+3] are valid and represents the address of 0th and (i+3)th element of x

Example :

```
/* write a program to print the address of a variable along
   with its value */

#include <stdio.h>
#include <conio.h>
void main()
{
   char a;
   int x;
   float p, q;
   clrscr();
   a = 'A';
   x = 125;
```

```
p = 10.25 ;
q = 18.76 ;
printf (" %c is stored at address %u\n", a, &a);
printf (" %d is stored at addn %u\n", x, &x);
printf (" %f is stored at addn %u\n", p, &p);
printf (" %f is stored at addn %u\n", q, &q);
getch() ;
}
```

output : A is stored at address 4436
125 is stored at addn 4434.
10.25 is stored at addn 4442
18.76 is stored at addn 4438.

Declaring pointer variables :

→ The general syntax for declaring a pointer variable is :

```
data-type * pointer-name ;
```

N.B. [The data-type should match with the variable data type.]

→ This tells the compiler three things about the variable pointer-name.

1. The asterisk (*) tells that the variable pointer-name is a pointer variable
2. pointer-name needs a memory location
3. pointer-name points to a variable of type data-type

→ For example, int *p ; declares the variable p as a pointer variable that points to an integer data-type

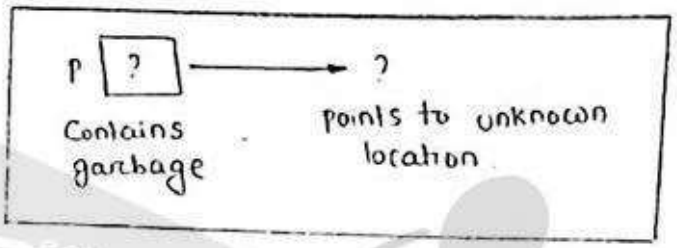
Similarly, the statement ;

float * x ; declares the variable x as a pointer variable that points to a floating point variable

→ These declarations cause the compiler to allocate memory locations for the pointer variable p and x.

→ Since, the memory locations have not been assigned any values, these locations may contain some unknown values in them and therefore, they point to unknown locations.

int * p ;



Pointer Declaration Style :

Pointer variables are declared similar to normal variables except for the addition of the unary * operator. This symbol can appear anywhere between the type name and the pointer variable name. programmers use the following styles :

(1) int * p /* style 1 */

(2) int *p /* style 2 */

(3) int * p /* style 3 */

However, the style 2 is becoming increasingly popular due to the following reasons :

a) This style is convenient to have multiple declarations in the same statements.

Example : int *p, x, *q ;

b) This style matches with the format used for accessing the target values.

Example:
```
int *p, x, Y;
x = 10;
p = &x;
Y = *p; /* accessing through p */
*p = 20; /* assigning 20 to x */
```

Initialization of pointer variable:

→ The process of assigning the address of a variable to a pointer variable is known as initialization.

→ Once the pointer variable is declared, we can use the assignment operator to initialize the variable.

→ Syntax:
```
data-type variable-name;
data-type * pointer-name; /* declaration */
pointer-name = & variable-name; /* initialization */
```

or
```
data-type * pointer-name = & variable-name;
```

→ Example:
```
int quantity;
int *p;      /* declaration */
p = &quantity; /* initialization */
```

or `int *p = &quantity;`

→ Example:
```
float a, b;
int x, *p;
p = &a;   /* wrong */
b = *p;
```

→ It is also possible to combine the declaration of data variable, the declaration of pointer variable and the initialization of pointer variable in one step.

Example : `int x, *p = &x ;` /* three in one */

It declares x as an integer variable and p as a pointer variable and then initializes p to the address of x.

`int *p = &x, x ;` /* wrong */

→ We could also define a pointer variable with an initial value of NULL or 0 (zero).

Example :

```
int *p = NULL ;
int *p = 0 ;
```

but `int *p = 5360 ;` /* wrong */

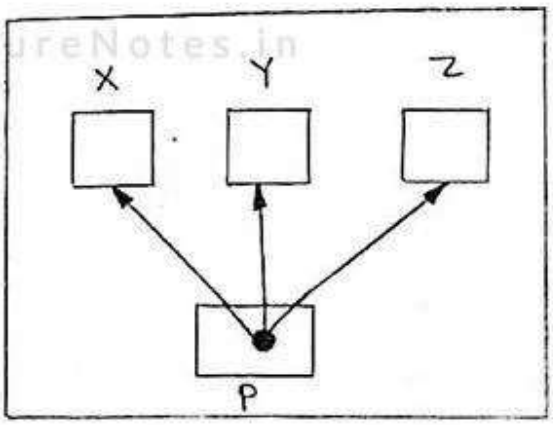
Pointer Flexibility :

Pointers are flexible. Because :

(1) We can make the same pointer to point to different data variables in different statements.

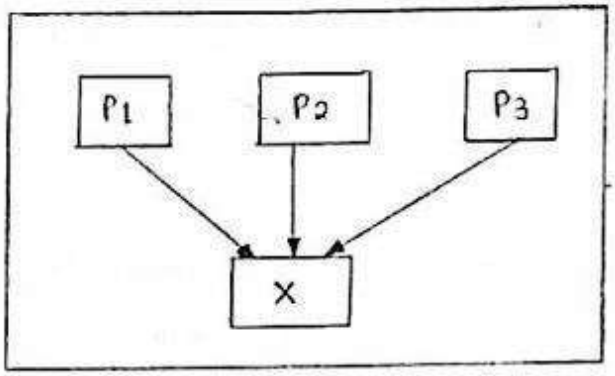
Example :

```
int x, Y, z, *p ;
---------
p = &x ;
---------
p = &Y ;
---------
p = &z ;
---------
```



(2) We can also use different pointers to point the same data variable.

Example:
```
int x ;
int *p1 = &x ;
int *p2 = &x ;
int *p3 = &x ;
. . . . . .
. . . . . .
```



Accessing a variable through its pointer :

→ Once the 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 or dereferencing operator.

→ Example:
```
int quantity, *p, n ;
quantity = 173 ;
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 173 to quantity

→ The third line assigns the address of quantity to the pointer variable p.

→ The fourth line contains the indirection operator *. When the operator * is placed before a pointer variable in an expression (on the right-hand side of the equal sign), the pointer 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 173.

→ The two statements

```
p = &quantity;
n = *p;
```

are equivalent to

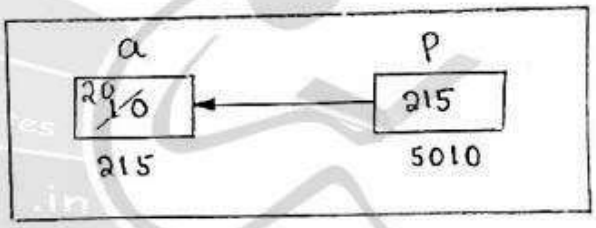
```
n = *&quantity;
```

which in turn equivalent to

```
n = quantity;
```

Example:

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int a=10, *p;
    clrscr();
    p = &a;
printf("%d", a);
printf("%d", *p);
printf("%d", *(&a));
printf("%u", &a);
printf("%u", p);
printf("%u", &p);
    *p = 20;
printf("%d", a);
printf("%d", *p);
}
```

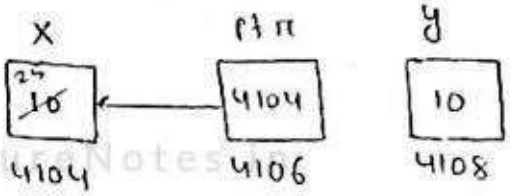


output:

10 10 10 215 215 5010 20 20

Example :

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int x,y;
    int *ptr;
    clrscr();
    ptr = &x;
    y = *ptr;
printf("value of x = %d\n", x);
printf("%d is stored at addr %u\n", x, &x);
printf("%d is stored at addr %u\n", *&x, &x);
printf("%d is stored at addr %u\n", *ptr, ptr);
printf("%d is stored at addr %u\n", ptr, &ptr);
printf("%d is stored at addr %u\n", y, &y);
*ptr = 25;
printf("Now x = %d\n", x);
getch();
}
```



Output :

value of x = 10
10 is stored at addr 4104
10 is stored at addr 4104
10 is stored at addr 4104
4104 is stored at addr 4106
10 is stored at addr 4108

Now x = 25

Pointer to Pointer or Chain of pointers :

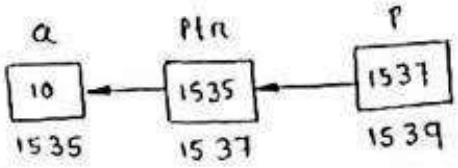
→ The general syntax for pointer to pointer is ;

data-type ** pointer-variable ;

→ Example : int **p ; we can also define int ***p & so on.

Here p is a pointer to pointer, which can store address of a pointer which holds the address of a float variable.

Example :




```
#include <stdio.h>
#include <conio.h>
void main()
{
    int a=10, *ptr, **p;
    clrscr();
    ptr = &a;
     p = &ptr;
printf("%u", p);
printf("%u", *p);
printf("%u", **p);
printf("%u", *ptr);
printf("%u", *(&ptr));
 printf("%u", *(&p));
 printf("%u", **(&p));
 printf("%u", ***(&p));
 getch();
}
```

output : 1537 1535 10 10 1535 1537 1535 10

Pointer Expression :

→ Pointer variables can be used in expression like other variables.

→ C allows us to add integers to or subtract integers from pointers, as well as to subtract one pointer from another pointer. p_1+4, p_2-2 and p_1-p_2 are all allowed. If p_1 and p_2 are both pointers to the same array, then p_2-p_1 gives the number of elements between p1 and p2.

→ We may also use short-hand operations with the pointers

```
p1++ ;
--p2 ;
sum+= *p2 etc.
```

→ In addition to arithmetic operations, pointers can also be compared using the relational operators. The expression such as $p_1 > p_2$, $p_1 == p_2$ and $p_1 != p_2$ are allowed. However, any comparison of pointers that refers to separate and unrelated variables makes no sense. Comparisons can be used meaningfully in handling arrays and strings.

→ We may not use pointers in division or multiplication. For example, $p1/p2$, $p1 * p2$, $p1/3$ are not allowed. Two pointers cannot be added. That is $p1+p2$ is illegal.

→ Example :

Ptr++ → 112

Ptr-- → 108

Ptr+2 → 110 + (2×2) = 114

Ptr+3 → 110 + (2×3) = 116

$$(Ptr + i) = ptr + i * sizeof(data\text{-}type)$$

→ The few following special aspects of pointer expressions.

* pointer Assignments
* pointer Arithmetic
* pointer Comparisons.

Pointer Assignments :

→ In pointer assignment, a pointer variable can be used in the right hand side of the assignment statement to assign its value to another pointer.

→ let ptr1 and ptr2 are two pointer variable of same type, then the statement ptr1 = ptr2 is valid.

Example : /* use of pointer assignment */

```
#include <stdio.h>
#include <conio.h>
void main()
{
   int x = 25;
   int *ptr1, *ptr2;
      clrscr();
      ptr1 = &x;
      ptr2 = ptr1;
  printf("value of variable x is %d\n", x);
  printf("Addn of variable x is %u\n", &x);
  printf("value of pointer variable ptr1 is %u\n", ptr1);
  printf("value of pointer variable ptr2 is %u\n", ptr2);
  getch();
}
```

output : value of variable x is 25
Address of variable x is 50122
value of pointer variable ptr1 is 50122
value of pointer variable ptr2 is 50122

Pointer Arithmetic :

If ptr is a pointer then ptr+2, ptr++, ptr-- are valid.
ptr1 - ptr2 is also valid if ptr1 & ptr2 are 2 pointers

```
# include <stdio.h>
# include <conio.h>
void main()
{
   int x = 25 ;
   char c = 'A' ;
   float f = 2.31 ;
   int *iptr;
   char *cptr ;
   float *fptr;
   clrscr();
   iptr = &x ;
   cptr = &c ;
   fptr = &f ;
printf ( " value of iptr is %u\n", iptr);
printf ( " value of cptr is %u\n", cptr);
printf ( " value of fptr is %u\n", fptr);
   iptr++ ;
   cptr++ ;
   fptr++ ;
printf ( " value of iptr = %u\n", iptr);
printf ( " value of cptr = %u\n", cptr);
printf ( " value of fptr = %u\n", fptr);
getch();
}
```

Output : value of iptr is 5012.
value of cptr is 5016.
value of fptr is 5020.
value of iptr = 5014
value of cptr = 5017
value of fptr = 5024

Pointer Comparisons :

→ In addition to pointer arithmetic, two pointers can be compared with each other by using relational operator.

→ For example, two pointers could be compared to test for end of array or memory area could be compared to see if pointers reference to same object

→ Example ; if (ptr1 > ptr2) /* reached end */
---
if (ptr1 == ptr2) /* same object ? */

Examples of pointer expressions :

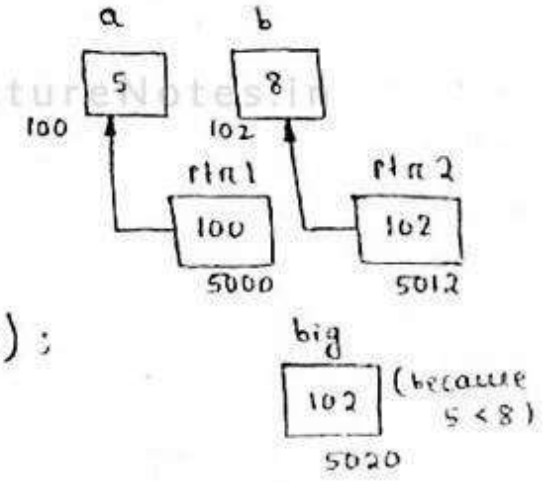
```
/* program to illustrate pointer expression */
#include <stdio.h>
#include <conio.h>
void main()
 {
    int a,b,c,d,*ptr1,*ptr2;
    clrscr();
    a=5,
    b=10;
    ptr1=&a,
    ptr2=&b,
   c=*ptr1+*ptr2;
   d=*ptr2/*ptr1*7;
```

```
    printf (" Address of a = %u\n", ptr1);
    printf (" Address of b = %u\n", ptr2);
    printf (" value of ptr1 & ptr2 are : %d %d", *ptr1, *ptr2);
    printf (" c = %d  d = %d ", c, d);
    getch();
}
```

Output : Address of a = 8432
Address of b = 8436
value of ptr1 and ptr2 are : 8432 8436
c = 15 d = 14

/* program to find the biggest number between two given numbers using pointer */

```
# include <stdio.h>
# include <conio.h>
void main()
{
    int a, b, *big, *ptr1, *ptr2;
    clrscr();
    printf (" Enter two numbers : ");
    scanf (" %d %d", &a, &b);
     ptr1 = &a;
     ptr2 = &b;
  if ( *ptr1 > *ptr2)
        big = ptr1;
    else
        big = ptr2;
  printf (" Biggest no is %d", *big);
  getch();
}
```



Output : Enter two numbers : 5 8
Biggest no. is 8

*big = 8
↓
value at 102 = 8