Pointer

A pointer is a variable that contains the address of another variable, A pointer can contain the address of any basic data type, of array, structures, unions etc, Pointers contain addresses and not the value.

Advantages of using pointers

1. It is more convenient or flexible to work with the address of a variable than the actual variable.

2. With the use of pointers program execution becomes faster.

3. Using a pointer, your program can access any memory location in the computer's memory.

4. Pointers allow a function to pass back more than one value.

5. Using pointers, arrays and structures can be handled in more efficient way.

6. It will become impossible to create complex data structures without pointers. They are link list, tree, graphs, etc.

The Pointer and the Indirection Operator

The two fundamental operators used with pointers are

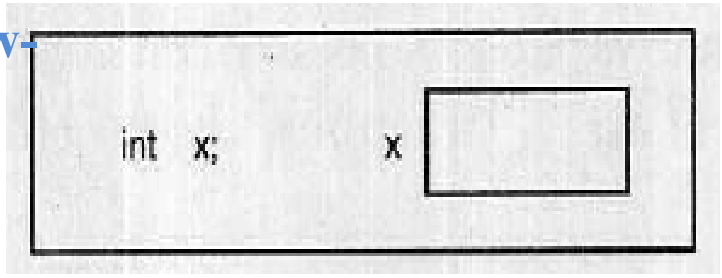
1. The address operator &

2. The indirection operator *

The address operator returns the address of a variable. The indirection operator returns the value of the variable that the pointer is pointing to.

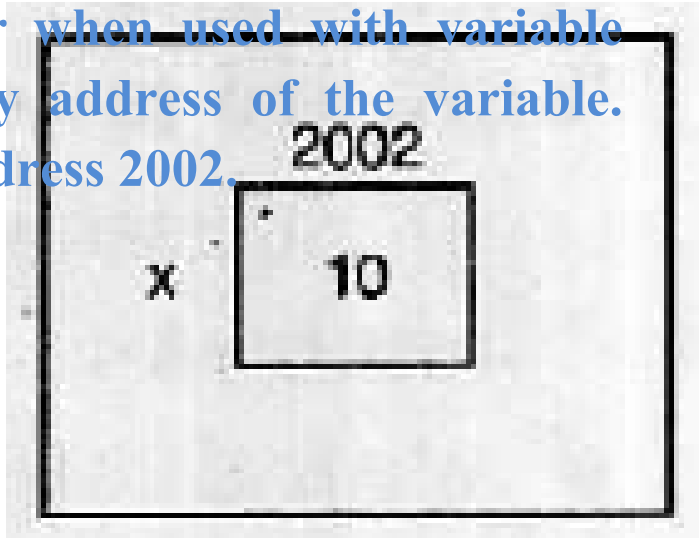
The '&' operator:-

When **a variable x is declared in a program, a storage location of main memory is made available by the compiler as shown below-**



x is the name associated by the compiler to a location in the memory of the computer.

Let the physical address of x (memory location) be 2002. For programmer it is viewed as variable x and by the operating system as an address 2002. The address of variable x can be obtained by &, an address of operator. This operator when used with variable gives the physical memory address of the variable. Thus & will provide the address 2002.



Ex:

```
int x = 10;
printf (" n value of x = %d", x);
printf (" n Address of x = %d", &x);
```

Output

Value of x = 10

Address of x = 2002 (assumed value)

The "" operator:-*

The **'*' is an indirection operator or 'value at address operator'. '*' operator provides a way of accessing the contents of the variable.**

Ex.

```
x= 10;
printf (" \n Value of x = %d", x);   10
printf (" n Address of x = %d", &x);  2002
printf (" n Value of address %d = %d", &x, * (&x));
```

Output

Value of x = 10

Address of x = 2002

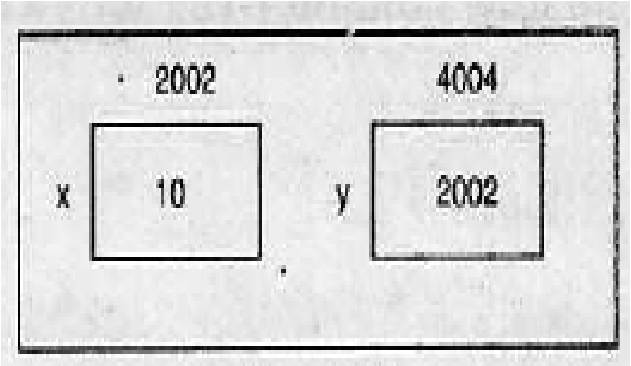
Value at address 2002 = 10

Pointer Variable Declaration and Accessing Values

An address of a variable can be stored in a special variable called pointer variable. A pointer variable is a variable that contains the address of another variable.

Let us assume that y is such a variable.

The address of variable x can be assigned to y by the



statement:

int x, *y //declaration

y = &x;
int x = 10;
y = &x;
&(&x) 4004 // &y

y is the variable that contains the address (i.e. 2002) of another variable x, whereas the address of y itself is 4004 (assumed value). In other words, we can say that y is a pointer to variable x. x=10 y=&x
*y=x

```
printf (" n Value of x = %d", x);
printf (" n Address of x = %d", &x);
printf (" \n Value of x = %d", *y);
printf (" n Address of x = %d", y);
printf (" \n Address of y = %d", &y); //&(&x)
```

Output-
Value of x = 10
Address of x = 2002
Value of x = 10
Address of x = 2002
Address of y = 4004

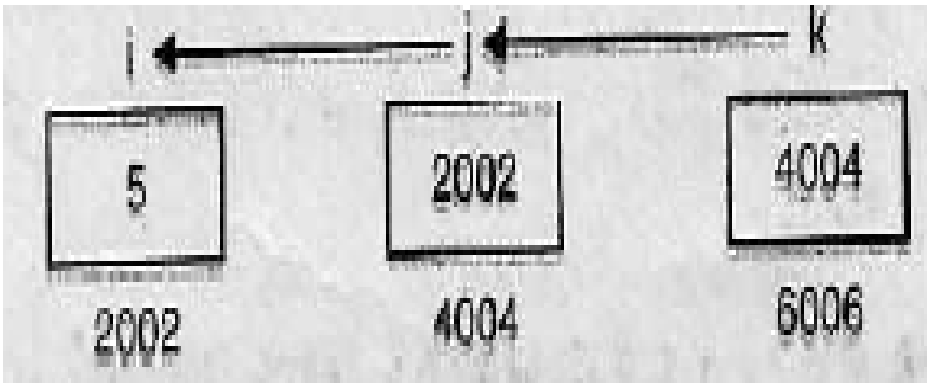
So this is how a pointer variable say y can be declared
int *y;
This declaration means that y is a pointer to a variable of type int

char *p;

This declaration means that p is a pointer to a variable of type char.

Pointer to Pointer

Every variable declared in a program is assigned an address, so a pointer variable also has an address. Pointer to pointer means that a variable can be declared which can store address of a pointer variable.



e.g,

```
int i, *j, **k;
i= 5;
j=&i;
k = &j;
```

i is an integer variable, j is a pointer variable which is pointing to i and k is pointer to pointer i.e, pointer to j.

To access a variable i through double pointer k, we can use **k. **k means value at (value at address k)
value at address k = 2002
So value at (value at address k i.e. 2002) = 5

Program to print the value of i using pointer to pointer.

```
void main( )
{
   int i = 5, *j, **k;
   j = &i;
```

```
  k = &j;
  printf ("\n Address of i = %u %u", &i, j);
  printf ("\n Address of j = %u %u, &j, k);
  printf ("\n Value of i = %d %d %d", i, *j, **k);
}
```

Output

Address of i = 2002 2002

Address of j = 4004 4004

Value of i = 5 5 5

Pointer arithmetic

Pointer can also be incremented to an immediately next location of its type e.g. if the contents of a pointer p of type integer is 5006 then the content of p++ will be 5008 instead of 5007.

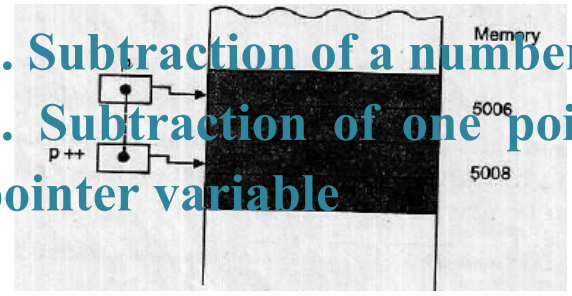
An int is always of 2 bytes size and therefore stored in two memory locations

The following operations are permitted on pointers

1. Addition of a number to a pointer variable

2. Subtraction of a number from a pointer variable

3. Subtraction of one pointer variable from another pointer variable



e.g. Suppose p1 and p2 are two pointers

k = pl + 5; /* Addition of a number to a pointer variable */

k = p2-4; /* Subtraction of a number from a pointer variable */

m = pl-p2; /* Subtraction of a pointer from another pointer */

p1 ++; 2000 4002 m=p2-p1 400-2000

p2 --;

S + = *p1; s=s+*p1

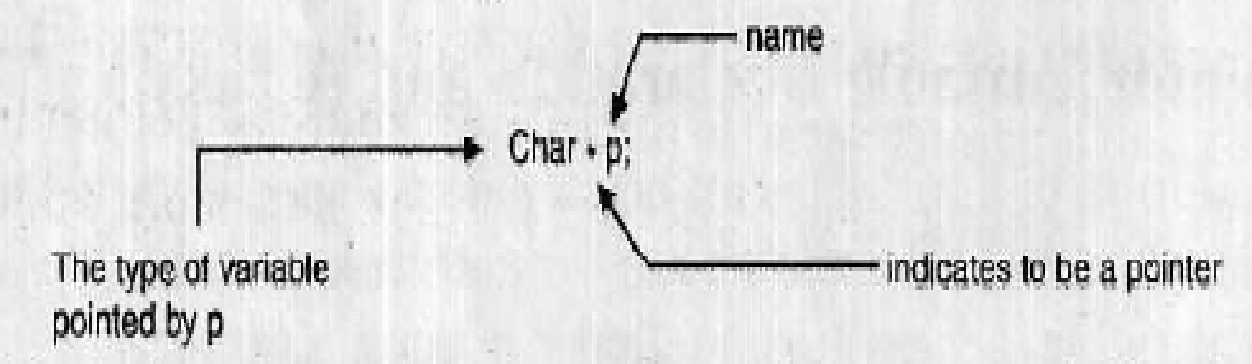
Above all operations are valid operations with pointers.

The following operations are not permitted on pointers

1. Addition of two pointer variables

2. Multiplication of a pointer variable by a number

3. Division of a pointer variable by a number:



e.g. pl+p2; /* Addition of two pointer variables */

p 1 * 5; /* Multiplication of a pointer variable by a number */

pl * p2; /* Multiplication of two pointer variables */

p2/ 4; /* Division of a pointer variable by a number */

p2/p1; /* Division of two pointer variables */

All these operations are invalid with pointers.

Pointer variables can also be compared provided they point to the same data types.

e.g. pl > = p2

p 1 == p2

p1 != p2

p1 <= p2

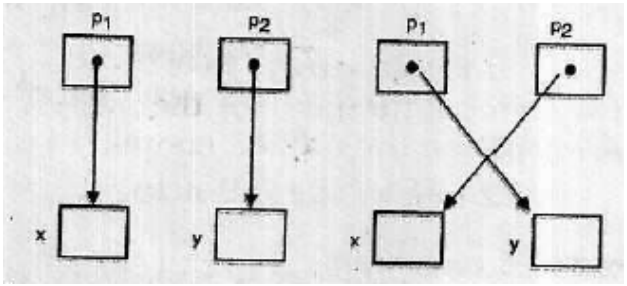
p1 > p2

p 1 <p2

All these comparisons are valid if pl and p2 are pointing to same data type i.e. int or float or char etc.

Write a program that reads two variables x and y of type integer. It prints the exchanged contents of these variables with the help of pointers without altering the variables.

```
/* Exchange the contents of two variables */
#include <stdio.h>
#include <conio.h>
void main( )
{
  int x, y, *pl, *P2, *P3;
  clrscr( );
       printf("  \nEnter  two
integers");
  scanf ("%d %d", &x, &y);
  /* Assign addresses of x and y to p1 and p2 */
  p1 = &x;
  p2 = &y;
  /* exchange the pointers */
```



```
    p3 = pl;
    p1= p2;
    p2 = p3;
    /* Print the contents through exchanged pointers */
    printf(" \n The exchanged contents are \ n”);
    printf("%d \t %d", *p2);
    getch( );
}
```