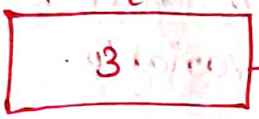
Bunch - 7

pointers

int i = 3

tells a compiler to

1) Reserve space (2 bytes) in the memory
2) Associate the name i with this mem. loc
3) store the val. 3 in that location.



Print

```
main()
{
    int i = 3;
    Printf("value of i = %d", i);
    Printf("Addr. of i = %d", &i);
    Printf("value of i = %d", *(&i));
}
```

(address of) optr returns the address.

o/p:

Addr. of i = 65524
value of i = 3
value of i = 3

→ use %u, %lu, %llu.

%p → pointer.

Scanf() → uses & optr.

(*) → value at address optr

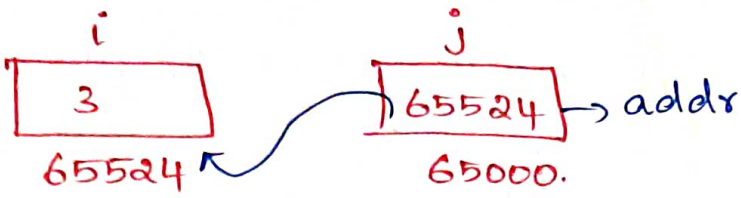
↓

also called indirection optr.
de-refrencing

*(&i) → i same

Now that address can be collected in a var.

$j = \&i$.



i's value = 3

j's value = i's address.

Thus pointer is a variable that stores the address of another variable.

Decl of Ptr var — Repr.

datatype * pointer var name

`int *j` → pointer variable. (decl).

↓ `j = &i;` → initialization

tells the compiler that var. j is used to store the address (not value) of the integer variable.

j → points to the integer

*j → says the value at address stored in j.

take initialization. → back

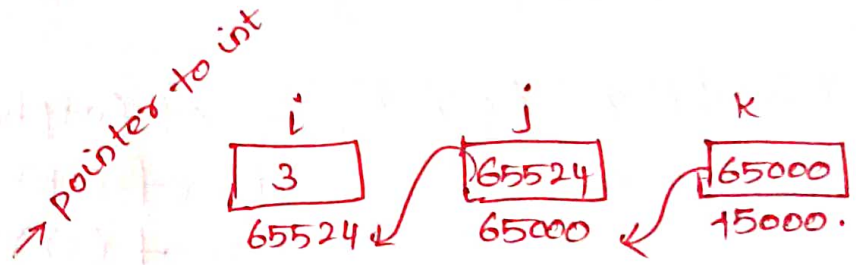
```
main()
{
    int i = 3;
    int *j;
    j = &i;
    Printf ("Addr. of i = %dn", &i);      65524
    Printf ("Addr. of i = %d", j);        65524
    Printf ("Addr. of j = %d", &j);       65000
    Printf ("value of j = %d", j);        65524
```

```
Printf ("value of i = %d", i);          3
Printf ("value of i = %d", *(&i));      3
Printf ("value of i = %d", *j);         3.
}.
```

int * j = int * j = int *j;

Preferable.

Program 2:-




```
main()
{
  int i=3, *j, **K;
  j = &i;
  K = &j;
  Print Addr of i  %u  &i;        65524
  Print Addr of j  %u  &j;        65524
  Print Addr of i  %u  *K;        65524
  Print Addr of j  %u  &j;        65000
  Print Addr of j  %u  K;         65000
  Print Addr of K  %u  &K;        45000
  Print value of j = j            65524
  Print value of K                65000.
  Print value of i                3
  Print value of i (*(&i));       3
  Print " of i*j;                 3
  Print val. of i**K;             3.
}.
```

→ pointer to the integer pointer.

It can go on till any no: of time.

But usually: it is pointer to a pointer.

Sizes:-

```
int *alpha;
char *ch;
float *s;
```

> Var, holding addrs of int, char, float vars

```
Printf("%d %d %d", sizeof(alpha),
                   sizeof(ch),
                   sizeof(s));
```

O/P: (4 4 4).

size of ptr = 2 bytes (16 bit)
4 bytes (32 bit)
8 bytes (64 bit)

↙

not dependent on the datatype the pointer points to.

All pointers regardless of the type of data type, have the same size based on system Architecture.

call by value & call by reference.

↓

value of each actual arg is copied into corresponding formal arguments.

Hence changes made to the formal Parameters has no effect on actual arguments