POINTER VARIABLES

ADDRESS OF A VARIABLE

- Each location in the memory is made up of 8 bits/1 byte.
- Every location is identified by a unique address.
- In order to access that address, we can use the address operator (&).
- It is a unary operator
- “&” gives the base address of a location i.e the starting address
- Addresses are hexadecimal numbers. Example:

0x7fff56739dd8

DEREFERENCING ADDRESS

- Given an address, we can also access the value stored in that address using a dereferencing operator (*).
- * operator is also called as indirection operator.
- It is a unary operator.
- It can only be placed before an address.

POINTER VARIABLES

- Any variable that has the capacity to store addresses is called as a pointer variable.
- It is declared as any other variable except with an additional "*" preceding it.
- Syntax:

datatype * pointer_var;

This * is only to indicate that it is a pointer variable. It is NOT an indirection operator in the declaration statement.

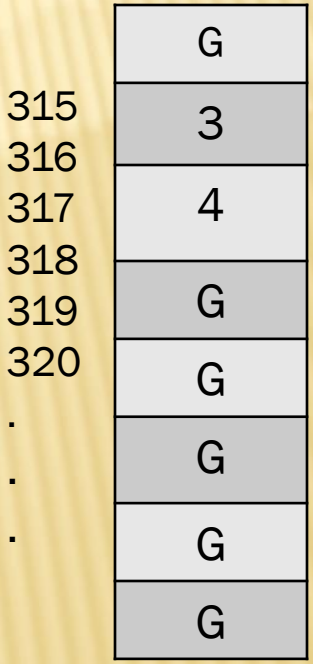
EXAMPLES

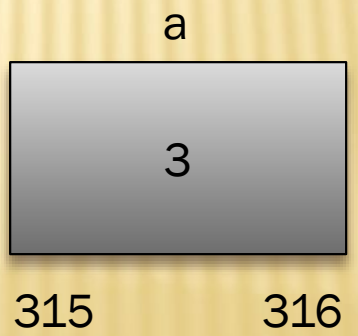
int a = 3;

printf(“%p”, &a); ⟶ 315

printf(“%d”,a); ⟶ 3

printf(“%d”,*(&a)); ⟶ 3





EXAMPLES

```
int a = 3;
int *p;
p=&a;
printf( “%p”, &a);
printf(“%p”,p);
printf( “%d”,*(&a));
printf( “%d”, *p);
printf( “%d”, a);
```

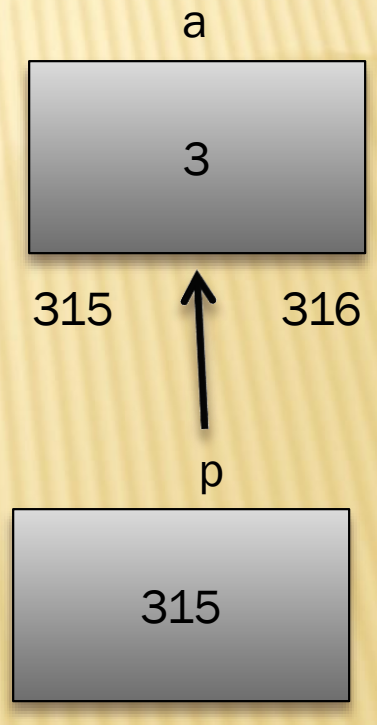
→ 315

→ 315

→ 3

→ 3

→ 3



QUICK EXERCISE

- **Predict the output (Assume address of a is 715):**

```
int a, *p;
a=5;
p=&a;
a=25;
printf("%p", p);   ⟶ 715
printf("%d",*p);   ⟶ 25
*p=34;
printf("%d",a);    ⟶ 34
```

Point to note:
Value of a variable can be changed directly through the variable name or indirectly through a pointer variable which is pointing to it.

QUICK EXERCISE

- **Predict the output (Assume address of a is 715):**

```
int a, *p, *q;
a=5;
p=&a;
q=&a;
printf("%p", p);
printf("%p", q);
*q=25;
printf("%d",*p);
printf("%d",a);
```

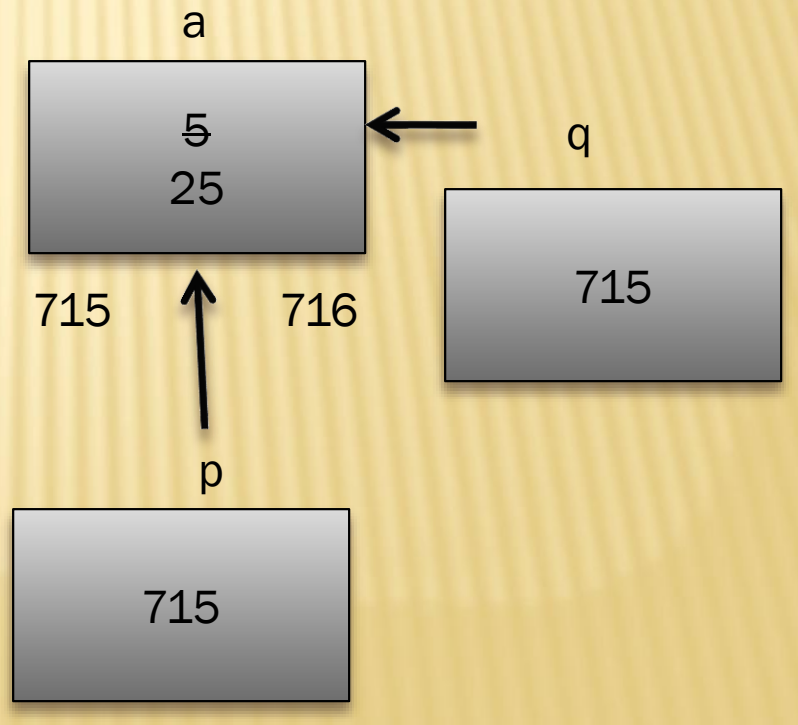
715

715

25

25

Point to note:
Multiple pointer variables can point to the same address location.



QUICK EXERCISE

- **Identify the correct and incorrect statements from below:**

```
int a, *p,q,*zp;
float *xp;
```

p=&a; ⟶ ✓

q=&a; ⟶ X (q is not a pointer variable)

zp=p; ⟶ ✓(One pointer can be assigned to another)

p=715; ⟶ X (You can only assign address using an & operator or another pointer variable)

*p=715; ⟶ ✓(*p is the value at the location. So a constant can be assigned)

xp=&a; ⟶ X (Data type of the pointer variable has to match the variable itself)

QUICK EXERCISE

- Predict the output (Assume address of a and b):

```
int a,b,*p,*q;
a=5, b=10;
q=&b;
p=&a;
*p= *p+*q;
p=q;
*q=*p+*q;
printf("%d %d %d %d",a,b,*p,*q);
```

⟶ 15 20 20 20

QUICK EXERCISE

- **Predict the output:**

```
int i=2, *ip;
float f=3.5, *fp;
ip = &i;
fp=&f;
printf( "%d", *ip);
printf( "%f", *fp);
ip = &f;
fp=&i;
printf ("%d  %d", sizeof (ip), sizeof(fp));
```

→ 2

→ 3.5

→ Error

→ 2 2

Point to note:

- "*" operator accesses "n" number of locations to give value at a location, where "n" depends upon the data type of the pointer variable
- However, addresses are always integers . Thus, size of any pointer variable of any data type will always be 2 bytes (int)