In the last example, we discussed the size of the address, each data type has a different value of size but their address has the same size.

Answers: Space to store address os variable is 8 bytes.

It may vary to 4 and 16 based on the configuration of the system.

For example, there is an apartment with 1, 2, 3 and 4 BHK flats, but on the name address plate, each flat has the same size to write there flat number and name.

Int x, y;

I want that, z should be capable of storing the address of a block whose data type is int.

Int \*z;

What is int \*z?

We declare the variable z is data type is **( int \* )**

What z can store?

After removing the \* whatever is left whose data types address z can be stored means int.

For example, I have Ramesh's address. Technically we do not have the address of Ramesh we have the address of Ramesh's house.

In the same way, Z will not store the x it will store the address of the x.

Lets take another example

We have declare 1 variable

**int \*\*\*k**;

What is data type of k? **(int \*\*\*)**

What k is capable of store? **( address of int\*\* )**

**X is capable of storing int type of value**

**\*z is capable of storing an address of int type block.**

Sensible example of a point of view.

We have a tiffin box, ideally, we should put food in the lunch box.

It can be capable of storing baby shoes. But we will not put shoes in the lunch box because as we know shoes is not food material.

BUT dot eats shoes. So according to the dog's point of view shoes are eatable.

But we are human, so are not going to put shoes in our lunch box because human point of view, because here human point of view matters.

What should we keep things in a container for which this container is made for, that is sensible.

According to Compiler point of view if we have written any number like 10, 20, -100.

Its default data type would be int.

So

#include<stdio.h>

int main(){

int x,y;

int \*z;

x = 30; // type int value int -> **Sensible**

y = 100; // type int value int -> **Sensible**

z = 50; // type int \* value int -> **inSensible**

Z = x; // type int \* value int -> **inSensible**

// we know address of x is 700

z = 700; // // type int \* value int -> **inSensible**

//Even the value of the address is 700

// but we are giving the address to it will be taken as int

z = &x; // type int \* value int \* -> **Sensible**

x = &z; // type int value **int \*\*** -> **inSensible**

}

What is int \*\*?

We know x = 10; x has a value and let's assume the address of x is 700.

Int \*z;

Z = &x; z has the value of the address of x i.e. 700 let's assume the address of z is 600.

So if we want to store the address of z to some variable

What should be the data type of that variable?

z contains the address of x so its type of int \*.

The new variable will contain the address of z that has a value of int \* so its data type should be **int \*\*.**

**Example**

Int x;

Int \*z;

X = 30; // we are talking about x

Z = &x; // we are talking about z

\*z = 575; **// we are talking about \*z -> z me jiska address he, uski bat ki.**

\*z -> if we are writing \*z and assigning any value to it.

That means we are talking about the variable whose address is stored in the z.

Which is x here in this example.

So after

\*z = 575;

X value will change from 30 to 575.

**We want no one can manipulate anything from z.**

**Zero is not allocated to anyone.**

**But**

Z = 0; // again same issue z have data type int \* and 0 is int.

So we have a term called null.

NULL means zero but all 8 data bites

If we write 0 it means 1 sign but and 7 data bits.

Z = NULL; // correct

It means z is not pointing to anyone.

# Pointers

A variable capable of storing an address is called a pointer or pointer variable.

Int \*z; -> z is a pointer which can point to an int-type block.

Z = &x - z start point to x (block reserved for x)

\*z = 575; -> Assigning to whoever z is pointing to.

#include<stdio.h>

int main(){

int x,y;

int \*z;

x = 30;

z = &x;

\*z = 575;

printf("%d\n", x);

x = 300;

printf("%d\n", \*z);

z = &y;

\*z = 959;

printf("%d\n", y);

y = 400;

printf("%d\n", \*z);

z = NULL;

return 0;

}

575

300

959

400

Example to understand

#include<stdio.h>

int add(int p, int q){

return p+q;

}

int subtract(int p, int q){

return p-q;

}

int main(){

int x,y,a,b;

x = 10;

y = 20;

a = add(x,y);

b = subtract(x,y);

return 0;

}

Now i want to perform add and subtract both functions with one variable. ?

We know function can not return more than one value.

So we can give the address of the variable in which we want to store the value.

Lets see a magic.

#include<stdio.h>

void lmn(int p, int q, int \*r, int \*s){

\*r = p+q;

\*s = p-q;

}

int main(){

int x,y,a,b;

x = 10;

y = 20;

lmn(x,y,&a,&b);

printf("Sum is %d\n", a);

printf("Differnce is %d\n", b);

return 0;

}

Output:

Sum is 30

Differnce is -10

**Call by value**

**Call by reference**

A = add(x,y) -> call by value

lmn(x, y, &a, &b) -> call by reference. (when address is passing)