**Example: How Pointer Works?**

#include <stdio.h>

int main()

{

int\* pc, c;

c = 22;

printf("Address of c: %u\n", &c);

printf("Value of c: %d\n\n", c);

pc = &c;

printf("Address of pointer pc: %u\n", pc);

printf("Content of pointer pc: %d\n\n", \*pc);

c = 11;

printf("Address of pointer pc: %u\n", pc);

printf("Content of pointer pc: %d\n\n", \*pc);

\*pc = 2;

printf("Address of c: %u\n", &c);

printf("Value of c: %d\n\n", c);

return 0;

}

**Output**

Address of c: 2686784

Value of c: 22

Address of pointer pc: 2686784

Content of pointer pc: 22

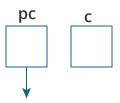
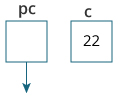
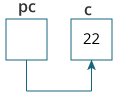
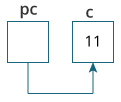
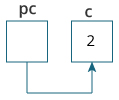
Address of pointer pc: 2686784

Content of pointer pc: 11

Address of c: 2686784

Value of c: 2

**Explanation of the program**

1. int\* pc, c;  
     
   Here, a pointer pc and a normal variable c, both of type int, is created.  
   Since pc and c are not initialized at first, pointer pc points to either no address or a random address. And, variable c has an address but contains a random garbage value.
2. c = 22;  
     
   This assigns 22 to the variable c, i.e., 22 is stored in the memory location of variable c.  
   Note that, when printing &c (address of c), we use %u rather than %d since address is usually expressed as an unsigned integer (always positive).
3. pc = &c;  
     
   This assigns the address of variable c to the pointer pc.  
   You see the value of pc is same as the address of c and the content of pc is 22 as well.
4. c = 11;  
     
   This assigns 11 to variable c.  
   Since, pointer pc points to the same address as c, value pointed by pointer pc is 11 as well.
5. \*pc = 2;  
     
   This change the value at the memory location pointed by pointer pc to 2.  
   Since the address of the pointer pc is same as the address of c, value of c is also changed to 2.

**Common mistakes when working with pointers**

Suppose, you want pointer pc to point to the address of c. Then,

int c, \*pc;

// Wrong! pc is address whereas,

// c is not an address.

pc = c;

// Wrong! \*pc is the value pointed by address whereas,

// &c is an address.

\*pc = &c;

// Correct! pc is an address and,

// &c is also an address.

pc = &c;

// Correct! \*pc is the value pointed by address and,

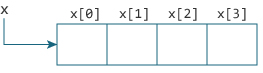
// c is also a value (not address).

\*pc = c;

## Relation between Arrays and Pointers

Consider an array:

int x[4];



From the above example, it's clear that x and &x[0] both contains the same address. Hence, &x[0] is equivalent to x.

And, x[0] is equivalent to \*x.

Similarly,

* &x[1] is equivalent to x+1 and x[1] is equivalent to \*(x+1).
* &x[2] is equivalent to x+2 and x[2] is equivalent to \*(x+2).
* ...
* Basically, &x[i] is equivalent to x+i and x[i] is equivalent to \*(x+i).

### Example 1: Pointers and Arrays

#include <stdio.h>

int main()

{

int i, x[6], sum = 0;

printf("Enter 6 numbers: ");

for(i = 0; i < 6; ++i)

{

scanf("%d", x+i);

sum += \*(x+i);

}

printf("Sum = %d", sum);

return 0;

}

When you run the program, the output will be:

Enter 6 numbers: 2

3

4

4

12

4

Sum = 29

In most contexts, array names "decays" to pointers. In simple words, array names are converted to pointers. That's the reason why you can use pointer with the same name as array to manipulate elements of the array. However, you should remember that **pointers and arrays are not same**.

There are few cases where array name doesn't decay into a pointer. To learn more, visit: [When does array name doesn't decay into a pointer?](https://stackoverflow.com/questions/17752978/exceptions-to-array-decaying-into-a-pointer)

### Example 2: Arrays and Pointers

#include <stdio.h>

int main()

{

int x[5] = {1, 2, 3, 4, 5};

int\* ptr;

ptr = &x[2];

printf("\*ptr = %d \n", \*ptr);

printf("\*ptr+1 = %d \n", \*ptr+1);

printf("\*ptr-1 = %d", \*ptr-1);

return 0;

}

When you run the program, the output will be:

\*ptr = 3

\*ptr+1 = 4

\*ptr-1 = 2

In this example, &x[2] (address of the third element of array x) is assigned to the pointer ptr. Hence, 3 was displayed when we printed \*ptr.

And, printing \*ptr+1 gives us the fourth element. Similarly, printing \*ptr-1 gives us the second element.