

# Pointers in C

---



## Pointer

Pointer is a special type of variable that is used to store address of a variable. When we declare a variable it is preceded by **asterisk (\*)**. While storing address of variable a variable is preceded by **ampersand (&)**.

A pointer can be incremented / decremented, i.e., to point to the next / previous memory location. The purpose of pointer is to save memory space and achieve faster execution time.

- 1) Normal variable stores the value whereas pointer variable stores the address of the variable.
- 2) The content of the C pointer always be a whole number i.e. address.
- 3) Always C pointer is initialized to null, i.e. `int *p = null`.
- 4) The value of null pointer is 0.
- 5) `&` symbol is used to get the address of the variable.
- 6) `*` symbol is used to get the value of the variable that the pointer is pointing to.
- 7) If a pointer in C is assigned to NULL, it means it is pointing to nothing.

- 8) Two pointers can be subtracted to know how many elements are available between these two pointers.
- 9) But, Pointer addition, multiplication, division are not allowed.
- 10) The size of any pointer is 4 byte (for 64 bit compiler).

**NOTE:** Using pointer we can find address of a variable as well as value of variable.

## Pointer

### Declaration

```
Data_Type *Pointer_Name;
```

### Example:

```
int *ptr;
```

## Pointer

### Storing Address

```
int x;
```

```
ptr = &x;
```

```
int x=10;
```

```
int *ptr = &x;
```

## Pointer

### Example - 1

```
#include<stdio.h>
int main()
{
    int a = 20;
    printf("Address of a = %u\n",&a);
    printf("Value of a = %d",a);
    return 0;
}
```

### OUTPUT

```
Address of a = 648758
Value of a = 20
```

NOTE: Address without pointer

## Pointer

### Example - 2

```
#include <stdio.h>
int main()
{
    int *ptr, q;
    q = 50;
    ptr = &q;
    printf("Value of q = %d\n", q);
    printf("Address of q = %d\n", ptr);
    printf("Value of q by pointer = %d", *ptr);
    return 0;
}
```

### OUTPUT

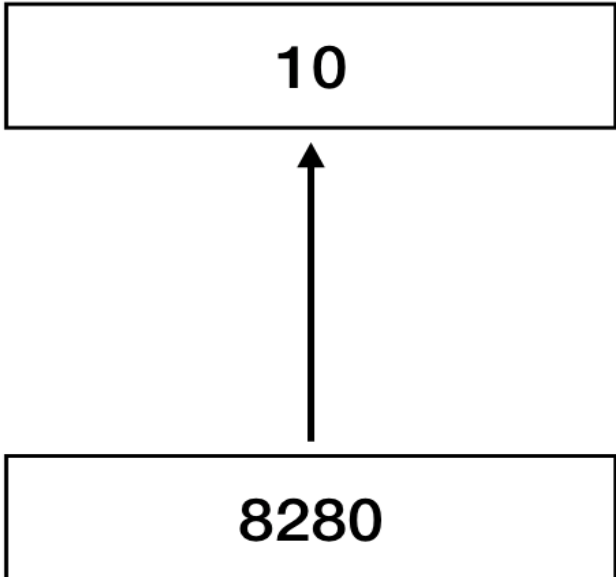
```
Value of q = 50
Address of q = 648758
Value of q by pointer = 50
```

# Pointer

## Graphical

**We can change the value of variable using address.**

Code	Variable	Value	Address
<code>int num = 10;</code>	num	10	8280
<code>int *ptr = &amp;num;</code>	ptr	8280	8272





## Pointer

### Example - 2

```
#include <stdio.h>
int main()
{
    int num = 10;
    int *ptr = &num;
    printf("value of num: %d\n", num);
    printf("value of num: (using pointer): %d\n", *ptr);
    *ptr = 20;
    printf("value of num: %d\n", num);
    printf("value of num (using pointer): %d\n", *ptr);
    return 0;
}
```

### OUTPUT

```
Value of num = 10
Value of num (using pointer = 10
Value of num = 20
Value of num (using pointer = 20
```

There are eight different types of pointers which are as follows –

- ✓ Null pointer
- ✓ Void pointer
- ✓ Wild pointer
- ✓ Dangling pointer

## Pointer

### Null Pointer

You create a null pointer by assigning the null value at the time of pointer declaration.

This method is useful when you do not assign any address to the pointer.  
A null pointer always contains value 0.

## Null Pointer

### Example - 1

```
#include <stdio.h>

int main()
{
    int *ptr = NULL; //null pointer
    printf("The value of ptr is: %d",ptr);
    return 0;
}
```

### OUTPUT

The value of ptr is: 0

## Pointer

### Void Pointer

It is a pointer that has no associated data type with it. A void pointer can hold addresses of any type and can be typecast to any type.

It is also called a generic pointer and does not have any standard data type. It is created by using the keyword **void**.

## Void Pointer

### Example - 1

```
#include<stdio.h>
int main()
{
    int a=2;
    char b='A';
    float f = 3.5f;
    void *ptr;
    ptr= &a;
    printf("Typecasting a = %d\n", *(int *)ptr);
    ptr= &b;
    printf("Typecasting b = %c\n", *(char *)ptr);
    ptr = &f;
    printf("Typecasting f = %f", *(float *)ptr);
    return 0;
}
```

### OUTPUT

```
Typecasting a = 2
Typecasting b = A
Typecasting f = 3.500
```

## Pointer

## Wild Pointer

If a pointer isn't initialized to anything, it's called a wild pointer. Wild pointers are also called uninitialized pointers.

## Wild Pointer

### Example - 1

```
#include <stdio.h>

int main()
{
    int *p; //wild pointer
    printf("%d", *p) ;
    return 0;
}
```

### OUTPUT

Segmentation fault



## Pointer

### Dangling Pointer

A dangling pointer is a pointer that refers to a memory location that has been released or deleted.

# Dangling Pointer

## Example - 1

```
#include <stdlib.h>
#include <stdio.h>
int main()
{
    int *ptr = (int* )malloc(sizeof(int));
    printf("Size of *ptr is : %d\n",sizeof(ptr));
    free(ptr);
    printf("Value of *ptr is %d ", *ptr);
    ptr = NULL;
    return 0;
}
```

### OUTPUT

Segmentation fault

## Call by value and call by reference

Functions can be invoked in two ways: **Call by Value** or **Call by Reference**. These two ways are generally differentiated by the type of values passed to them as parameters.

The parameters passed to function are called ***actual parameters*** whereas the parameters received by function are called ***formal parameters***.

## Call By Value

In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of the caller.

## Call By Value

### Example - 1

```
#include <stdio.h>

void swap(int x, int y)
{
    int t;
    t = x;
    x = y;
    y = t;
    printf("x = %d\n", x);
    printf("y = %d\n", y);
}
```

### OUTPUT

```
x = 20      y = 10
a = 10      b = 20
```

```
int main()
{
    int a = 10, b = 20;
    printf("a = %d\n", a);
    printf("b = %d\n", b);
    swap(a, b);
    return 0;
}
```

## Call By Reference

Both the actual and formal parameters refer to the same locations, so any changes made inside the function are actually reflected in actual parameters of the caller.

# Call By Reference

## Example - 1

```
#include <stdio.h>

void swap(int *x, int *y)
{
    int t;
    t = *x;
    *x = *y;
    *y = t;
    printf("x = %d\n", x);
    printf("y = %d\n", y);
}
```

Formal Parameter

### OUTPUT

```
x = 20      y = 10
a = 20      b = 10
```

```
int main()
{
    int a = 10, b = 20;
    printf("a = %d\n", a);
    printf("b = %d\n", b);
    swap(&a, &b);
    return 0;
}
```

We can perform arithmetic operations on the pointers like addition, subtraction, etc. In pointer-from-pointer subtraction, the result will be an integer value. Following arithmetic operations are possible on the pointer in C language:

- ✓ Increment
- ✓ Decrement
- ✓ Addition
- ✓ Subtraction
- ✓ Comparison



## Pointer

### Incrementing

If we increment a pointer by 1, the pointer will start pointing to the immediate next location. This is somewhat different from the general arithmetic since the value of the pointer will get increased by the size of the data type to which the pointer is pointing.

The Rule to increment the pointer is given below:

$$\text{new\_address} = \text{current\_address} + i * \text{size\_of}(\text{data type})$$

NOTE: Where  $i$  is the number by which the pointer get increased.

# Pointer Incrementing

## Example - 1

```
#include<stdio.h>
int main()
{
    int number=50;
    int *p;
    p=&number;
    printf("Address of p = %u",p) ;
    p = p + 1;
    printf("After increment:") ;
    printf("Address of p = %u \n",p) ;
    return 0;
}
```

### OUTPUT

Address of p = 32100  
After increment:  
Address of p = 32104

## Pointer

### Decrementing

Like increment, we can decrement a pointer variable. If we decrement a pointer, it will start pointing to the previous location. The formula of decrementing the pointer is given below:

$$\text{new\_address} = \text{current\_address} - i * \text{size\_of}(\text{data type})$$

NOTE: Where  $i$  is the number by which the pointer get decreased.

## Pointer Decrementing

### Example - 1

```
#include<stdio.h>
int main()
{
    int number=50;
    int *p;
    p=&number;
    printf("Address of p = %u",p) ;
    p = p - 1;
    printf("After decrement:") ;
    printf("Address of p = %u \n",p) ;
    return 0;
}
```

### OUTPUT

Address of p = 32104  
After decrement:  
Address of p = 32100

## Pointer

### Addition

Like increment and decrement to a pointer variable we can add value to the pointer variable.

**`new_address = current_address + (number * size_of(data type))`**

# Pointer Addition

## Example - 1

```
#include<stdio.h>
int main()
{
    int number=50;
    int *p;
    p=&number;
    printf("Address of p = %u",p) ;
    p = p + 3;
    printf("After Addition:");
    printf("Address of p = %u \n",p) ;
    return 0;
}
```

### OUTPUT

Address of p = 32104  
After Addition:  
Address of p = 32116

There are various operations which can not be performed on pointers. Since, pointer stores address hence we must ignore the operations which may lead to an illegal address.

- ✓  $\text{Address} + \text{Address} = \text{illegal}$
- ✓  $\text{Address} * \text{Address} = \text{illegal}$
- ✓  $\text{Address} \% \text{Address} = \text{illegal}$
- ✓  $\text{Address} / \text{Address} = \text{illegal}$
- ✓  $\text{Address} \& \text{Address} = \text{illegal}$
- ✓  $\text{Address} \wedge \text{Address} = \text{illegal}$
- ✓  $\text{Address} | \text{Address} = \text{illegal}$
- ✓  $\sim \text{Address} = \text{illegal}$

## Pointer

### Pointer to Pointers (Double Pointer)

As we know that, a pointer is used to store the address of a variable in C. Pointer reduces the access time of a variable. However, In C, we can also define a pointer to store the address of another pointer. Such pointer is known as a double pointer (pointer to pointer). The first pointer is used to store the address of a variable whereas the second pointer is used to store the address of the first pointer.



# Pointer

## Syntax

**Data\_Type \*\*Pointer\_Name;**

**Example: int \*\*Ptr;**



## Pointer

### Example - 1

```
#include<stdio.h>
int main ()
{
    int a = 10;
    int *p;
    int **pp;
    p = &a;
    pp = &p;
    printf("Address of a: %x\n",p);
    printf("Address of p: %x\n",pp);
    printf("value of p: %d\n",*p);
    printf("value of pp: %d\n",**pp);
    return 0;
}
```

### OUTPUT

```
Address of a: 10234
Address of p: 20345
Value of p: 10
Value of pp: 10
```

*QUIZ*

## Quiz - 1

```
# include <stdio.h>
void fun(int x)
{
    x = 30;
}

int main()
{
    int y = 20;
    fun(y);
    printf("%d", y);
    return 0;
}
```

OUTPUT

20

[Click here to see code](#)

## Quiz - 2

```
#include <stdio.h>
void fun(int *ptr)
{
    *ptr = 30;
}
int main()
{
    int y = 20;
    fun(&y);
    printf("%d", y);
    return 0;
}
```

OUTPUT

30

[Click here to see code](#)

## Quiz - 3

```
#include <stdio.h>
void fun(int *ptr)
{
    *ptr = 30;
}
int main()
{
    int y = 20;
    fun(&y);
    printf("%d", y);
    return 0;
}
```

OUTPUT

30

[Click here to see code](#)

# *Coding Questions*

## Question - 1

Write a program to take two input from the user and store the address of variable into pointer and find the sum of the numbers using pointer.

[Click here to see code](#)



## Question - 2

Write a program to take two input from the user and store the address of variable into pointer and find the sum of the numbers using **call by reference**.

[Click here to see code](#)

## Coding Questions

1. Write a C program to find cube of any number using function.
2. Write a C program to find diameter, circumference and area of circle using functions.
3. Write a C program to find maximum and minimum between two numbers using functions.
4. Write a C program to check whether a number is even or odd using functions.
5. Write a C program to check whether a number is prime, Armstrong or perfect number using functions.
6. Write a C program to find all prime numbers between given interval using functions.
7. Write a C program to print all strong numbers between given interval using functions.
8. Write a C program to print all Armstrong numbers between given interval using functions.
9. Write a C program to print all perfect numbers between given interval using functions.
10. Write a C program to find power of any number using recursion.

## Coding Questions

11. Write a C program to calculate area of square using return function.
12. [Write a program to calculate area and parameter of rectangle.](#)
13. [Write a program to check if entered character is alphabet or not using return function with argument.](#)
14. [Write a C program to check entered number is even or odd using return with argument function.](#)
- 15.