

## C Pointers

What is pointer?

Pointer is a variable just like other variables of C but only difference is unlike the other variable it stores the memory address of any other variables of C. This variable may be type of int, char, array, structure, function or any other pointers. The size of the pointer depends on the architecture. However, in 32-bit architecture the size of a pointer is 2 byte.

When we declare any variable, the declaration statement instructs the compiler that create the location and required space for variable and put the value at the location.

Consider following examples

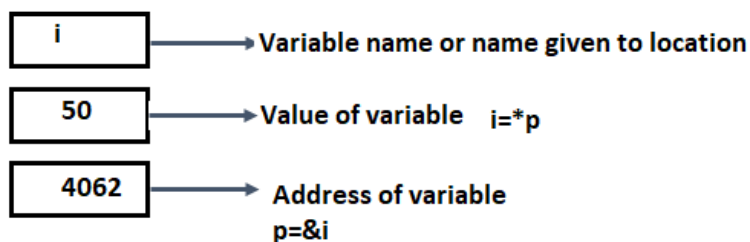
(1) Pointer p which is storing memory address of int type variable:

```
int i=50; //Reserve space in memory, give name i to location and store value 50 at location.
```

```
int *p;
```

```
p=&i;
```

Three blocks shown below shows the meaning of each term in declaration.



(2) Pointer p which is storing memory address of float type variable:

```
float i=50.6;
```

```
float *p;
```

```
p=&i;
```

(3) Pointer p which is storing memory address of char type variable:

```
char i='A';
```

```
char *p;
```

```
p=&i;
```

(4) Pointer p which is storing memory address of an array:

```
int arr[20];
```

```
int *ptr;
```

```
ptr=&arr;
```

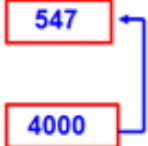
The value of variable can be accessed by using its name or address. The memory address is just integer number and this number can be assigned to another variable and stored at different location as shown in following example.

### EXAMPLE FOR POINTER:

`int X = 547;`

HERE:

Variable name	Contents	Location
X	547	4000
ptr	4000	4036



According to above figure, By the help of `ptr` variable we stored the address of variable `X` in address 4036

Variable name is `x` whose value is 547, `ptr` is the variable used to store the address of `x` which is equal to 4000. We can say `ptr` points to `x` thus `ptr` is pointer variable which can be stored at other location 4063.

Advantages and disadvantages of pointers in c

#### **Benefits (use) of pointers in c:**

- Pointers provide direct access to memory
- Pointers provide a way to return more than one value to the functions
- Reduces the storage space and complexity of the program
- Reduces the execution time of the program
- Provides an alternate way to access array elements
- Pointers can be used to pass information back and forth between the calling function and called function.
- Pointers allow us to perform dynamic memory allocation and deallocation.
- Pointers helps us to build complex data structures like linked list, stack, queues, trees, graphs etc.
- Pointers allow us to resize the dynamically allocated memory block.
- Addresses of objects can be extracted using pointers

#### **Drawbacks of pointers in c:**

- Uninitialized pointers might cause segmentation fault.
- Dynamically allocated block needs to be freed explicitly. Otherwise, it would lead to memory leak.
- Pointers are slower than normal variables.
- If pointers are updated with incorrect values, it might lead to memory corruption.

### Accessing the address of the variable-

The physical memory location of the variable in the memory is system dependent so it cannot be predicted. This can be accessed by using address operator &. The & operator returns the address of the variable. This operator can be used for any type of primary and array variable. Address of the variable or value of pointer variable is unsigned integer so control character used is %u. Following example shows how to access the address.

<pre>#include&lt;stdio.h&gt; int main() {     char ch='z';     int A=23;     float B=12.5,C=26.3;     printf("%c is stored at %u\n",ch,&amp;ch);     printf("%d is stored at %u\n",A,&amp;A);     printf("%f is stored at %u\n",B,&amp;B);     printf("%f is stored at %u\n",C,&amp;C); } /*Output z is stored at 6487583 23 is stored at 6487576 12.500000 is stored at 6487572 26.299999 is stored at 6487568</pre>	<p>Explanation –</p> <p>In the program</p> <p>ch is char type, value of ch is 'z'</p> <p>A is int type, value of A is 23</p> <p>B is float type, value of B is 12.5'</p> <p>C is float type, value of C is 26.5</p> <p>→ Display value of ch and address of ch</p> <p>→ Display value of A and address of A</p> <p>→ Display value of B and address of B</p> <p>→ Display value of C and address of C</p>
---	---

### Declaration of pointer variable

Similar to other variables pointer variable must be declared before its use. The pointer in c language can be declared using \* (asterisk symbol). It is also known as indirection operator or dereference operator. The syntax for declaration of pointer variable is as below.

Data\_type \*pointer\_variable\_name;

Data type of pointer refers to the data type of the variable for which it is used. Following example shows the declaration of pointer variables for different types of variables.

int \*ptr //ptr is pointer variable or integer pointer and holds the address of integer variable.

float \*p //p is pointer variable or float pointer and holds the address of float type variable.

char \*w //w is pointer variable or char pointer and holds the address of char type variable.

### Initialization of pointer variable.

Initialization of pointer variable is done by assigning the address of a variable to it

eg.

```
int X=547
```

```
int *ptr;
```

```
ptr = &X;
```

In above example pointer ptr is initialized to address of X.

### **Accessing variable through its pointer-**

Once the address of the primary variable is assigned to pointer variable the value of the primary variable is accessed by using indirection operator \*. Following code shows how to access the value of primary variable using pointer variable.

```
int quantity, *p, n;
```

```
quantity=779;
```

```
p=&quantity;
```

```
n=*p;
```

In above segment first line declares the quantity and n as integer variables and p as integer pointer. Second line initialize the quantity to 779. Third line assigns the address of quantity to pointer p. Fourth line contains the indirection operator before pointer variable in an expression on right side of the equal to sign. \*p returns the value at p or value of variable quantity. Thus \* is value at address. Above segment can be written in the following format also.

```
int quantity, *p, n;
```

```
quantity=779;
```

```
p=&quantity;
```

```
n=* &quantity;
```

```
#include<stdio.h>
int main()
{
    int x,y,*ptr;           //Declaration of variables
    x=10;                   //initialization of x to 10
    ptr=&x;                  //assign address of x to ptr
    y=*ptr;                 //assign value at ptr to y
    printf("value of x = %d\n",x); //Display value of x
    printf("%d is stored at address %u\n",x,&x); //Display value and address of x
    printf("%d is stored at address %u\n",&x,&x); //Display value and address of x
    printf("%d is stored at address %u\n",*ptr,ptr); //Display value at ptr and address of x
    printf("%d is stored at address %u\n",y,&*ptr); //Display value of y and address of x
    printf("%d is stored at address %u\n",y,&y); //Display value of y and address of y
```

```

    *ptr=25;                                //Modify value at ptr i.e. value of x
    printf("New value of x=%d",x);          //Display modifies value of x
}
/*Output
value of x = 10
10 is stored at address 6487572
10 is stored at address 6487572
10 is stored at address 6487572
10 is stored at address 6487572
10 is stored at address 6487568
New value of x=25

```

### Arithmetic operations on pointer (Pointer expressions)

**1. Operations on values at address-** we can perform operations like arithmetic, relational, assignment, conditional, etc. on pointer variables. which can be performed on values at addresses. Following table shows the different operations on pointer variables.

	Operations	Operator	Example	Result
<b>int a=20,b=6;</b> <b>int *ptr_a, *ptr_b;</b> <b>ptr_a=&amp;a;</b> <b>ptr_b=&amp;b</b>				
<u>Arithmetic Operators</u>	Addition	+	*ptr_a+*ptr_b	26
	Subtraction	-	*ptr_a-*ptr_b	14
	Multiplication	*	*ptr_a* *ptr_b	120
	Division	/ *	*ptr_a/ *ptr_b	3
		(should be space between / and *)		
	Module division	%	*ptr_a%*ptr_b	2

**Note:** While performing division, make sure you put a blank space between '/' and '\*' of the pointer as together it would make a multi-line comment ('/\*').

<u>Relational Operators</u>	*ptr_a>*ptr_b	True	Assignment Operators	*ptr_a=10;	10
	*ptr_a<*ptr_b	False		*ptr_a+=20	30
	*ptr_a== *ptr_b	False		*ptr_a - =*ptr_b;	24
	*ptr_a!= *ptr_b	False		*ptr_a / =*ptr_b;	4

<b>Increment or Decrement</b>	Post increment or decrement	*ptr_a++	Pre increment or decrement	++*ptr_a	
		*ptr_a--		--*ptr_a	

**/\* Demonstration of expressions using pointer variable\*/**

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
    int a,b,*p1,*p2,x,y,z;
```

```
    a=12;
```

```
    b=4;
```

```
    p1=&a;
```

```
    p2=&b;
```

```
    x=*p1+*p2-6;
```

```
    y=4*- *p2/ *p1+10;
```

```
    printf("Address of a=%u\n",p1);
```

```
    printf("Address of b=%u\n",p2);
```

```
    printf("a= %d, b= %d\n",a,b);
```

```
    printf("x= %d, y= %d\n",x,y);
```

```
    *p2+=3;
```

```
    *p1-=*p2;
```

```
    z=*p1* *p2-6;
```

```
    printf("a= %d, b= %d\n",a,b);
```

```
    printf("z= %d ",z);
```

```
}
```

**/\*Output**

```
Address of a=6487552
```

```
Address of b=6487548
```

```
a= 12, b= 4
```

```
x= 10, y= 9
```

```
a= 5, b= 7
```

```
z= 29
```

## 2. Operations on pointer (operations on address)

A pointer in c is an address, which is integer numeric value. Therefore, you can perform arithmetic operations on a pointer. C allows to add or subtract integer from pointer as below.

P1+4 or p2-5;

Depending on type of variable and system used the number is added or subtracted from the pointer. For example on 64 bit system for integer 4 bytes, for float 4 bytes and for character 1 byte. So the addition of 1 for integer is the addition of 4, for float addition of 4 and for character addition of 1. i.e. in addition and subtraction the scale factor is used which is equal to size of variable. In above examples the if P1 and p2 are integer pointers then P1+4 means initial address plus 16 and p2-5 means initial address is subtracted by 20. Most used pointer operations are: ++, --, which will increment/decrement the pointer by the total number of bytes required to store the data pointed by pointer. Increment or decrement on 64-bit system is as below.

- 1) Pointer pointing to int value is incremented/decremented by 4 bytes
- 2) Pointer pointing to char value is incremented / decremented by 1 byte
- 3) Pointer pointing to float value is incremented. decremented by 4 bytes

**Table shows the results of arithmetic operations on different types of pointer variables.**

Data type	Required bytes	Initial address	Operation	Scale factor	Address after operation
int a, *Pa;	4	2056	Pa+3;	4	2068
		2056	Pa-2;		2048
		2056	Pa++;		2060
		2056	Pa--;		2052
float b, *Pb;	4	2056	Pb+3;	4	2068
		2056	Pb-2;		2048
		2056	Pb++;		2060
		2056	Pb--;		2052
char c, *Pc;	1	2056	Pc+3;	1	2059
		2056	Pc-2;		2054
		2056	Pc++;		2057
		2056	Pc--;		2055

```
/* Demonstration of pointer operations*/
```

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
    int a,*pa;
```

```
    float b,*pb;
```

```
    char c,*pc;
```

```
    pa=&a;
```

```
    pb=&b;
```

```

pc=&c;
printf("Initial Address of a=%u\n",pa);
printf("Initial Address of b=%u\n",pb);
printf("Initial Address of c=%u\n",pc);
pa=pa+3;
pb=pb+3;
pc=pc+3;
printf("Address after addition of 3 in int=%u\n",pa);
printf("Address after addition of 3 in float=%u\n",pb);
printf("Address after addition of 3 in char=%u\n",pc);
pa++;
pb--;
pc++;
printf("New int addresses %u\n",pa);
printf("New float addresses %u\n",pb);
printf("New char addresses %u\n",pc);
}
/*Output
Initial Address of a=6487556
Initial Address of b=6487552
Initial Address of c=6487551
Address after addition of 3 in int=6487568
Address after addition of 3 in float=6487564
Address after addition of 3 in char=6487554
New int addresses 6487572
New float addresses 6487560
New char addresses 6487555

```

**Subtraction of pointers**-We can subtract one pointer from other like p1-p4. Subtraction of pointer will give the number of elements between them. Check the following code.

```

#include<stdio.h>
int main()
{
    int a,b,c,d,*p1,*p2,*p3,*p4;
    p1=&a;
    p2=&b;
    p3=&c;
    p4=&d;
    printf("subtraction of pointers=%d",p1-p4);
}
/*Output
addition of pointers=3

```



**Comparison between two pointers or addresses-** Two addresses can be compared using relational operators.

P1>P2; P1==P2 or P1!=P2

```
*Compare two pointers*/
#include<stdio.h>
int main()
{
    int a=25,*p1,*p2;
    p1=&a;
    p2=&a;
    if(p1==p2)
        printf("Two pointers have same address");
    else
        printf("Two pointers have different address");
}
```

*Note-Multiplication, division, Module division and addition of pointers is not allowed*

### **Pointer to an array**

Once the array is declared the compiler allocate the base address and sufficient amount of memory for all elements of array in continuous memory locations. The base address or address of first element (0<sup>th</sup> element) is array name itself. The elements of array together with their address can be displayed by using array name itself. For example.

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

Suppose base address is 1000 and each integer on 64-bit system requires 4 bytes. So above array elements are arranged in the following format.

Elements	X[0]	X[1]	X[2]	X[3]	X[4]
Value	1	2	3	4	5
Address	1000	1004	1008	1012	1016

The name of array X is constant pointer to first (0<sup>th</sup>) element. Thus X=1000 i.e. location where X is stored. Thus we can write

```
X=&X[0];
```

When we declare integer pointer P we can make pointer P to point first element by following expression.

```
P=X; which is equivalent to P=&X[0];
```

**/\*Display array elements with their addresses using array name\*/**

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
    int x[5]={4,6,8,10,23};
```

```
    int k;
```

```
    printf("\nElement no. Element Address ");
```

```
    for(k=0;k<5;k++)
```

```
    {
```

```
        printf("\n x[%d] \t%d \t%u",k,*(x+k),x+k);
```

```
    }
```

```
}
```

**/\*Output**

Element no. Element Address

x[0]        4        6487552

x[1]        6        6487556

x[2]        8        6487560

x[3]        10       6487564

x[4]        23       6487568

**/\* Print characters in string along with memory addresses of each character\*/**

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
    char x[]="Hello";
```

```
    int k;
```

```
    printf("\nElement no. Element Address ");
```

```
    for(k=0;k<5;k++)
```

```
    {
```

```
        printf("\n x[%d] \t%c \t%u",k,*(x+k),x+k);
```

```
    }
```

```
}
```

**/\*Output**

Element no. Element Address

x[0]	H	6487568
x[1]	e	6487569
x[2]	l	6487570
x[3]	l	6487571
x[4]	o	6487572

### 1. Pass by address (call by reference)- Function call by reference-

In this method the address of actual arguments in the calling function are copied in to the formal arguments of called function. by this method we can access the address of the actual argument and we can manipulate them. Also we can return more than one values to the calling function.

```
/*swap values using call by reference*/
```

```
#include<stdio.h>
```

```
void swapr(int *,int *);
```

```
int main()
```

```
{
```

```
    int a=10,b=20;
```

```
    swapr(&a,&b);
```

```
    printf("a= %d",a);
```

```
    printf("\nb= %d",b);
```

```
    return 0;
```

```
}
```

```
void swapr(int *x,int *y)
```

```
{
```

```
    int temp;
```

```
    temp=*x;
```

```
    *x=*y;
```

```
    *y=temp;
```

```
}
```

```
/* Calculate area and perimeter using call by reference*/
```

```
#include<stdio.h>
```

```
void areaperi(int,float *,float *);
```

```
int main()
```

```
{
```

```
    int radius;
```

```
    float area,perimeter;
```

```
    printf("Enter the radius");
```

```
    scanf("%d",&radius);
```

```
    areaperi(radius,&area,&perimeter);
```

```
    printf("Area of circle= %f",area);
```

```
        printf("\nPerimeter of circle= %f",perimeter);  
        return 0;  
    }  
void areaperi(int r,float *a,float *p)  
{  
    *a=3.141*r*r;  
    *p=2*3.141*r;  
}
```