POINTER



**Pointers** store address of variables or a memory location. A **pointer** is a variable that stores the address of another variable. Unlike other variables that hold values of a certain type, pointer holds the address of a variable. For example, an integer variable holds (or you can say stores) an integer value, however an integer pointer holds the address of a integer variable

// General syntax

**datatype \*var\_name;**

// An example pointer "ptr" that holds

// address of an integer variable or holds

// address of a memory whose value(s) can

// be accessed as integer values through "ptr"

**Example: int \*ptr;**

**Using a Pointer:**



**To use pointers in C, we must understand below two operators**.

1)To access address of a variable to a pointer, we use the unary operator **& (ampersand**) that returns the address of that variable. For example &x gives us address of variable x.

2)One more operator is **unary \*** (Asterisk) which is used for two things :

 a)To declare a pointer variable: When a pointer variable is declared in C/C++, there must be a \* before its name.

|  |
| --- |
|  |

b)To access the value stored in the address we use the unary operator (\*) that returns the value of the variable located at the address specified by its operand. This is also called **Dereferencing**.

#include <stdio.h>

int main()

{

 int var = 5;

 printf("var: %d\n", var);

 // Notice the use of & before var

 printf("address of var: %p", &var);

 return 0;

}

**Output**

var: 5

address of var: 2686778

**Example of Pointer demonstrating the use of & and \***

#include <stdio.h>

int main()

{

 /\* Pointer of integer type, this can hold the

 \* address of a integer type variable.

 \*/

 int \*p;

 int var = 10;

 /\* Assigning the address of variable var to the pointer

 \* p. The p can hold the address of var because var is

 \* an integer type variable.

 \*/

 p= &var;

 printf("Value of variable var is: %d", var);

 printf("\nValue of variable var is: %d", \*p);

 printf("\nAddress of variable var is: %p", &var);

 printf("\nAddress of variable var is: %p", p);

 printf("\nAddress of pointer p is: %p", &p);

 return 0;

}

Output:

Value of variable var is: 10

Value of variable var is: 10

Address of variable var is: 0x7fff5ed98c4c

Address of variable var is: 0x7fff5ed98c4c

Address of pointer p is: 0x7fff5ed98c50



**Another example**

#include <stdio.h>

int main()

{

 int x = 10;

 // 1) Since there is \* in declaration, ptr

 // becomes a pointer variable (a variable

 // that stores address of another variable)

 // 2) Since there is int before \*, ptr is

 // pointer to an integer type variable

 int \*ptr;

 // & operator before x is used to get address

 // of x. The address of x is assigned to ptr.

 ptr = &x;

 printf("address of x =%u \n",&x);

 printf("address of x=%u \n",ptr);

 printf("adress of ptr=%u \n",&ptr);

 printf("value of ptr=%u \n",ptr);

 printf("value of x=%d \n",x);

 printf("value of x=%d \n",\*(&x));

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

 return 0;

 }

**Output:**

**Address of x=2293324**

**Address of x=2293324**

**Address of ptr=2293312**

**Value of ptr=2293324**

**Value of x=10**

**Value of x=10**

**Value of x=10**

**Picture representation of above program**

 **Ptr x**

  22jjjjjjjjjjjjjjj

10

2293324

 2293312 2293324

**Double pointer**:

We can declare a double pointer variable also,which can store the memory address of any pointer variable of same data type i.e **pointer to pointer**. When a pointer holds the address of another pointer then such type of pointer is known as **pointer-to-pointer** or **double pointer**.

For example:

Int \*ptr,\*\*dptr;

Here dptr is a double pointer with data type int and it can store the memory address of any single pointer with data type int as shown below.

 dptr=&ptr;

#include<stdio.h>

int main()

{

 int p,\*ptr,\*\*dptr;

 p=10;

 ptr=&p;

 dptr=&ptr;

 printf("\n the address of p=%u",&p);

 printf("\n value in p=%u",p);

 printf("\n the address of ptr=%u",&ptr);

 printf("\n value in ptr=%u",ptr);

 printf("\n the address of dptr=%u",&dptr);

 printf("\n value in dptr=%u",dptr);

 printf("\n value in p=%u",\*ptr);

 printf("\n value in ptr=%u",\*dptr);

}

Output:

The address of p=1154

Value in p=10

The address of ptr=2256

Value in ptr=1154

The address of dptr=6234

Value in dptr=2256

Value in p=10

Value in ptr=1154

Above program representation

 1154 2256 6234

2256

1154

 10

 **P ptr dptr**

**Pointer Expressions and Pointer Arithmetic**
A limited set of arithmetic operations can be performed on pointers. A pointer may be:

* incremented ( ++ )
* decremented ( — )
* an integer may be added to a pointer ( + or += )
* an integer may be subtracted from a pointer ( – or -= )

Pointer arithmetic is meaningless unless performed on an array.

**Note** : Pointers contain addresses. Adding two addresses makes no sense, because there is no idea what it would point to. Subtracting two addresses lets you compute the offset between these two addresses.

|  |
| --- |
| **example****#include<stdio.h>**int main(){    // Declare an array    int v[3] = {10, 100, 200};  // Declare pointer variable    int \*ptr;  // Assign the address of v[0] to ptr    ptr = v; int i;  for (i=0;i<3;i++)    {        printf("Value of \*ptr = %d\n", \*ptr);        printf("Value of ptr = %p\n\n", ptr); // Increment pointer ptr by 1        ptr++;    }} |

Output:Value of \*ptr = 10

Value of ptr = 0x7ffcae30c710

Value of \*ptr = 100

Value of ptr = 0x7ffcae30c714

Value of \*ptr = 200

Value of ptr = 0x7ffcae30c718

