**Pointers :**

* A pointer can be used to store the memory address of other variables, functions, or even other pointers. The use of pointers allows low-level memory access, dynamic memory allocation, and many other functionality in C.

**Syntax :**

datatype **\*** ***ptr***;

The use of pointers in C can be divided into three steps:

1. **Pointer Declaration**
2. **Pointer Initialization**
3. **Pointer Dereferencing**

* Pointer does not belong to any kind of datatype. It is just a special variable that holds the address

The ptr special variable is pointing to integer datatype. Pointers should always be initialized.

The size of the pointers is 4bytes/8bytes. The size of the pointers is same for all data types.

The value of the pointer is always positive so we use %u

4types of Pointer

1. Null pointer

2. Void pointer

3. Wild pointer

4. Dangling pointer (situation)

Void \*ptr 🡪 void pointer/generic pointer. It can point to any kind of datatype

Int \*ptr = NULL; 🡪 null pointer. It points to nothing

Int \*ptr 🡪 wild pointer. As it is not initialized. It can point to any kind of data

ptr 🡪 gives the address of the variable pointing to

\*ptr 🡪 gives the value(dereferencing using \*)

&ptr 🡪 gives the address of pointer

**Arrays and Pointers:**

* Arrays are static pointers can be dynamic.

int a[3] ={1,2,3};

int \*ptr= NULL;

ptr= &a[0];

printf (“%d”, ptr[0]); // array notation

printf (“%d”, ptr[1]);

printf (“%d”, ptr[2]);

printf (“%d”, \*(ptr+0)); // pointer notation

printf (“%d”, \*(ptr+1));

printf (“%d”, \*(ptr+2));

1. &\* 🡪 nullify each other

2. Op[ ] 🡪 \*op

\*op 🡪 op[ ]

We can directly give the name of array to pointer

ptr = &b[0] ==== ptr =b;

……………………………………………………………………………………………..

/\*

Ponter Demo -

\*/

#include <stdio.h>

int main()

{

int a=10;

float b=20.2;

void \*ptr = NULL;

printf("\nAddress of a=%u and its value=%d",&a,a);

printf("\nAddress of b=%u and its value=%f",&b,b);

printf("\nAddress of ptr=%u and its value=%u",&ptr,ptr);

printf("\nSize of a=%d",sizeof(a));

printf("\nSize of b=%d",sizeof(b));

printf("\nSize of ptr=%d",sizeof(ptr));

printf("\n\n");

return 0;

}

O/P

Address of a=1766022732 and its value=10

Address of b=1766022728 and its value=20.200001

Address of ptr=1766022720 and its value=0

Size of a=4

Size of b=4

Size of ptr=8

If changed like this

printf("\nSize of b=%d",sizeof(b));

printf("\nSize of ptr=%d",sizeof(ptr));

ptr = &a;

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

printf("\n\n");

return 0;

then

Address of a=1364800476 and its value=10

Address of b=1364800472 and its value=20.200001

Address of ptr=1364800464 and its value=0

Size of a=4

Size of b=4

Size of ptr=8

ptr value = 1364800476

* + Dereferencing can be done by using \*ptr , once after doing this we can get the content or the value stored in that address.

Printf("\nValue pointed by ptr = %d",\*ptr);

Then it results in

error: invalid use of void expression

* + This is because generic pointers must be explicitly converted, it cannot do on its own

If void \*ptr = NULL;

So, printf("\nValue pointed by ptr = %d", \*(int \*)ptr);

Then, Value pointed by ptr = 10

Note : The above error occurs only when generic pointers are used

Also if we need conversion between similar datatype then no issues

If

int \*ptr1 = NULL; and printf("\nValue pointed by ptr1 = %d", \*ptr1);

then

Value pointed by ptr1 = 10

NO ERROR