C Pointers

The pointer in C language is a variable which stores the address of another variable.

The size of the pointer depends on the architecture. However, in 32-bit architecture the size of a pointer is 2 byte.

1. **int** n = 10;
2. **int**\* p = &n; // Variable p of type pointer is pointing to the address of the variable n of type integer.

#include<stdio.h>

int main(){

int number=50;

int \*p;

p=&number;//stores the address of number variable

printf("Address of p variable is %p \n",p); // p contains the address of the number therefore printing p gives the address of number.

printf("Value of p variable is %d \n",\*p); // As we know that \* is used to dereference a pointer therefore if we print \*p, we will get the value stored at the address contained by p.

return 0;

}

//

Address of p variable is 000000000061FE14

Value of p variable is 50

### **Pointer to array**

1. **int** arr[10];
2. **int** \*p[10]=&arr; // Variable p of type pointer is pointing to the address of an integer array arr.

### **Pointer to structure**

1. **struct** st {
2. **int** i;
3. **float** f;
4. }ref;
5. **struct** st \*p = &ref;

## **Advantage of pointer**

1) Pointer **reduces the code** and **improves the performance**,

**1) Dynamic memory allocation**

In c language, we can dynamically allocate memory using malloc() and calloc() functions where the pointer is used.

2)used in arrays, functions, and structures. It reduces the code and improves the performance.

## **NULL Pointer**

A pointer that is not assigned any value but NULL is known as the NULL pointer.

If you don't have any address to be specified in the pointer at the time of declaration, you can assign NULL value.

It will provide a better approach.

int \*p=NULL;

#include<stdio.h>

void main()

{

    int a=10;

    int \*iptr=&a;

    printf("size a=%u\n",sizeof(a));

    printf("size iptr=%u\n",sizeof(iptr));

    float b=1.2;

    float \*fptr=&b;

    printf("size b=%u\n",sizeof(b));

    printf("size fptr=%u\n",sizeof(fptr));

    char ch='A';

    char \*cptr=&ch;

    printf("size ch=%u\n",sizeof(ch));

    printf("size cptr=%u\n",sizeof(cptr));

}

//

size a=4

size iptr=8

size b=4

size fptr=8

size ch=1

size cptr=8

#include<stdio.h>

void main()

{

    int a=259;

    char \*iptr=&a;

    char ch='A';

    char \*cptr=&ch;

    printf("iptr = %d\n",\*iptr);

    printf("cptr=%c\n",\*cptr);

    printf("a=%d\n",a);

    printf("ch=%c\n",ch);

}

//

 warning: initialization of 'char \*' from incompatible pointer type 'int \*' [-Wincompatible-pointer-types]

  char \*iptr=&a;

             ^

iptr = 3

cptr=A

a=259

ch=A

# C Double Pointer (Pointer to Pointer)

we can also define a pointer to store the address of another 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. Let's understand it by the diagram given below.



#include<stdio.h>

void main()

{

    int a=10;

    int \*ptr1=&a;

    int \*\*ptr2=&ptr1;

    printf(" %d\n",\*ptr1);

    printf("%d\n",\*\*ptr2);

}

//

//10

//10

# Pointer Arithmetic in C

We can perform arithmetic operations on the pointers like addition, subtraction, etc.

Following arithmetic operations are possible on the pointer in C language:

* Increment
* Decrement
* Addition
* Subtraction
* Comparison

## **Incrementing Pointer in C**

If we increment a pointer by 1, the pointer will start pointing to the immediate next location.

the pointer will get increased by the size of the data type to which the pointer is pointing.

#include<stdio.h>

int main(){

int number=50;

int \*p;//pointer to int

p=&number;//stores the address of number variable

printf("Address of p variable is %u \n",p);

p=p+1;

printf("After increment: Address of p variable is %u \n",p); // in our case, p will get incremented by 4 bytes.

return 0;

}

Output

Address of p variable is 3214864300

After increment: Address of p variable is 3214864304

#include<stdio.h>

void main ()

{

    int arr[5] = {1, 2, 3, 4, 5};

    int \*p = arr;

    int i;

    printf("printing array elements...\n");

    for(i = 0; i< 5; i++)

    {

        printf("%d  ",\*(p+i));

    }

}

Output

printing array elements...

1  2  3  4  5

## 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.

#include <stdio.h>

int main(){

   int \*ptr = NULL; //null pointer

   printf("The value inside variable ptr is:\n%d",ptr);

   return 0;

}

The value inside variable ptr is:

0

## Void Pointer

It is a pointer that has no associated data type with it

It is also called a generic pointer and does not have any standard data type.

It is created by using the keyword void.

#include <stdio.h>

int main()

{

    int a= 10;

    void \*p = &a; //void pointer

   printf("%d",\*(int\*)p);

   return 0;

}

## Wild Pointer

Wild pointers are also called uninitialized pointers.

#include <stdio.h>

int main(){

   int \*p; //wild pointer

   printf("\n%d",\*p);

   return 0;

}

# Dangling,

A pointer pointing to a memory location that has been deleted (or freed) is called dangling pointer.

**De-allocation of memory**

// Deallocating a memory pointed by ptr causes

// dangling pointer

#include <stdlib.h>

#include <stdio.h>

int main()

{

    int \*ptr = (int \*)malloc(sizeof(int));

    // After below free call, ptr becomes a

    // dangling pointer

    free(ptr);

    // No more a dangling pointer

    ptr = NULL;

}

#include <stdio.h>

int\* fun()

{

    int b=123;

    return &b;

}

int main()

{

    int \*a = fun();

    printf("a=%d\n",\*a);

}

**Pointer to a constant.**

#include <stdio.h>

int main(void)

{

    int i = 10;

    int j = 20;

    const int \*ptr = &i; /\* ptr is pointer to constant \*/

    printf("ptr: %d\n", \*ptr);

    /\* error: assignment of read-only location '\*ptr' \*/

    //\*ptr = 100;

    ptr = &j;          /\* valid \*/

    printf("ptr: %d\n", \*ptr);

    return 0;

}

//ptr: 10

ptr: 20

**2) Constant pointer**

#include <stdio.h>

int main(void)

{

   int i = 10;

   int j = 20;

   int \*const ptr = &i; /\* constant pointer to integer \*/

   printf("ptr: %d\n", \*ptr);

   \*ptr = 100;    /\* valid \*/

   printf("ptr: %d\n", \*ptr);

 /\* error: assignment of read-only location 'ptr' \*

   //ptr = &j;        /\* error \*/

   return 0;

}

//

ptr: 10

ptr: 100

#include <stdio.h>

int main(void)

{

    int a= 10;

    int b= 20;

   int \*ptr =&a;

   int \*ptr1 = &b;

   printf("%u\n",&a);

   printf("%u\n",&b);

   int res = ptr-ptr1;

   printf("res = %d",res);

}