UNIT-4(POINTERS)

The pointer is a special type of variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer.

Declaring a pointer:→

The pointer in c language can be declared using * (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.

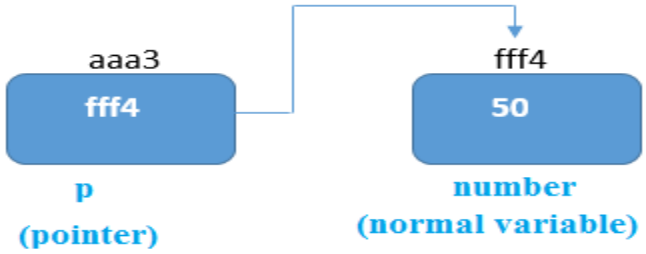
General syntax:➔

Data type * pointer name = value;

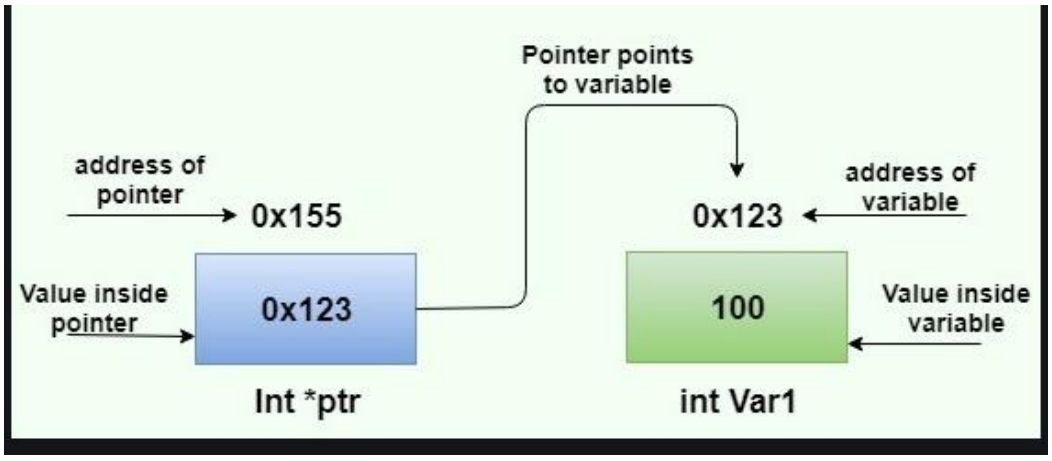
1. **int** *a;//pointer to int
2. **char** *c;//pointer to char

Pointer Example

An example of using pointers to print the address and value is given below.



Parts of pointer:→



```c
#include<stdio.h>
int main(){
int number=50;
int *p;
p=&number;//stores the address of number variable
printf("Address of p variable is %x \n",p); // p contains the address of the number therefore printing p gives the address of nun
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
return 0;
}
```

Output

```
Address of number variable is fff4
Address of p variable is fff4
Value of p variable is 50
```

Advantage of pointer

- Manage the memory more efficiently
- Can be used for efficient handling of 2D and 3D arrays.
- Can be used for dynamic allocation and de-allocation of memory.
- Pointers save the memory.
- Pointers execute at faster rate.
- Can be used to write compact program code.
- Pointers can pass information forward and backward b/w function and its reference.

Usage of pointer

There are many applications of pointers in c language.

1) Dynamic memory allocation

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

2) Arrays, Functions, and Structures

Pointers in c language are widely used in arrays, functions, and structures. It reduces the code and improves the performance.

Address (&) Operator

The address of operator '&' returns the address of a variable. But, we need to use %u to display the address of a variable.

```
#include<stdio.h>
int main(){
int number=50;
printf("value of number is %d, address of number is %u",number,&number);
return 0;
}
```

Output

```
value of number is 50, address of number is fff4
```

Types of addressing:→1)Direct:→In it we use variable name directly

2)Indirect:→We refer to variable using pointer

Indirection Operator:→The symbol * (asterisk) is indirection operator. It is use to access value of variable .Also called as pointer operator.

```
void main(){
int a,*b;
a=5;
b&a;
printf(“”a=%d b=%d”,a,b);
getch();}
```

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;
```

Pointer Program to swap two numbers without using the 3rd variable.

```
1. #include<stdio.h>
2. int main(){
3. int a=10,b=20,*p1=&a,*p2=&b;
```

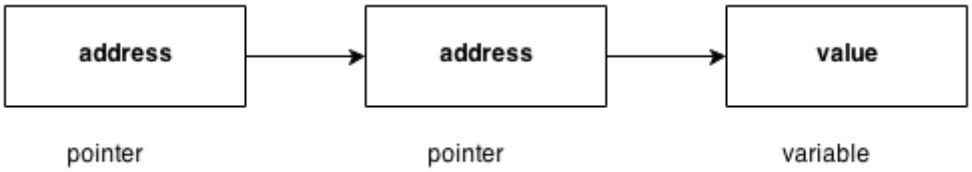
```
4.
5.  printf("Before swap: *p1=%d *p2=%d",*p1,*p2);
6.  *p1=*p1+*p2;
7.  *p2=*p1-*p2;
8.  *p1=*p1-*p2;
9.  printf("\nAfter swap: *p1=%d *p2=%d",*p1,*p2);
10.
11. return 0;
12. }
```

Output

```
Before swap: *p1=10 *p2=20
After swap: *p1=20 *p2=10
```

C Double Pointer (Pointer to 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. Let's understand it by the diagram given below.



The syntax of declaring a double pointer is given below.

Data type **pointer name;.

```
1.  #include<stdio.h>
2.  void main ()
3.  {
4.      int a = 10;
5.      int *p;
6.      int **pp;
7.      p = &a; // pointer p is pointing to the address of a
8.      pp = &p; // pointer pp is a double pointer pointing to the address of pointer p
9.      printf("address of a: %x\n",p); // Address of a will be printed
10.     printf("address of p: %x\n",pp); // Address of p will be printed
```

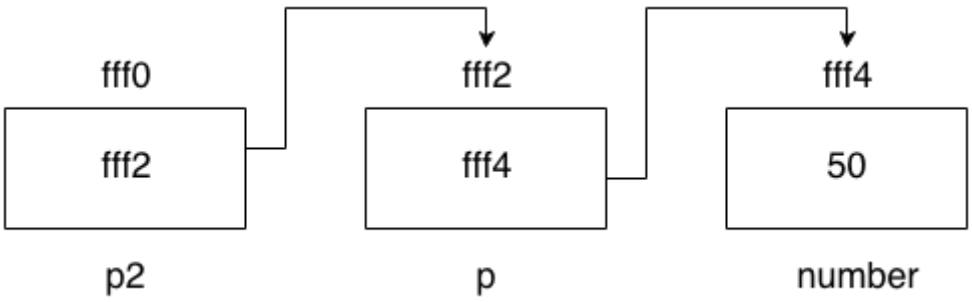
```
11.     printf("value stored at p: %d\n",*p); // value stoted at the address contained
        by p i.e. 10 will be printed
12.     printf("value stored at pp: %d\n",**pp); // value stored at the address contain
        ed by the pointer stoyred at pp
13. }
```

Output

```
address of a: d26a8734
address of p: d26a8738
value stored at p: 10
value stored at pp: 10
```

C double pointer example

Let's see an example where one pointer points to the address of another pointer.



As you can see in the above figure, p2 contains the address of p (fff2), and p contains the address of number variable (fff4).

```
1.  #include<stdio.h>
2.  int main(){
3.  int number=50;
4.  int *p;//pointer to int
5.  int **p2;//pointer to pointer
6.  p=&number;//stores the address of number variable
7.  p2=&p;
8.  printf("Address of number variable is %x \n",&number);
9.  printf("Address of p variable is %x \n",p);
10. printf("Value of *p variable is %d \n",*p);
11. printf("Address of p2 variable is %x \n",p2);
12. printf("Value of **p2 variable is %d \n",*p);
13. return 0;
14. }
```

OUTPUT

```
Address of number variable is fff4
Address of p variable is fff4
Value of *p variable is 50
Address of p2 variable is fff2
Value of **p variable is 50
```

Pointer Arithmetic in C

We can perform arithmetic operations on the pointers like addition, subtraction, etc. However, as we know that pointer contains the address, the result of an arithmetic operation performed on the pointer will also be a pointer if the other operand is of type integer. 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

Incrementing Pointer in C

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:

1. new_address= current_address + i * size_of(data type)

   Where i is the number by which the pointer get increased.

32-bit

For 32-bit int variable, it will be incremented by 2 bytes.

64-bit

For 64-bit int variable, it will be incremented by 4 bytes.

Let's see the example of incrementing pointer variable on 64-bit architecture.

```
1. #include<stdio.h>
2. int main(){
3. int number=50;
4. int *p;//pointer to int
5. p=&number;//stores the address of number variable
6. printf("Address of p variable is %u \n",p);
7. p=p+1;
8. printf("After increment: Address of p variable is %u \n",p); // in our case, p will get
    incremented by 4 bytes.
9. return 0;
10. }
```

Output

```
Address of p variable is 3214864300
After increment: Address of p variable is 3214864304
```

Traversing an array by using pointer

```
1. #include<stdio.h>
2. void main ()
3. {
4.    int arr[5] = {1, 2, 3, 4, 5};
5.    int *p = arr;
6.    int i;
7.    printf("printing array elements...\n");
8.    for(i = 0; i< 5; i++)
9.    {
10.      printf("%d  ",*(p+i));
11.   }
12. }
```

Output

```
printing array elements...
1  2  3  4  5
```

Decrementing Pointer in C

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:

1. new_address= current_address - i * size_of(data type)

32-bit

For 32-bit int variable, it will be decremented by 2 bytes.

64-bit

For 64-bit int variable, it will be decremented by 4 bytes.

Let's see the example of decrementing pointer variable on 64-bit OS.

```
#include <stdio.h>
void 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 decrement: Address of p variable is %u \n",p); // P will now point to th
e immidiate previous location.
}
```

Output

```
Address of p variable is 3214864300
After decrement: Address of p variable is 3214864296
```

C Pointer Addition

We can add a value to the pointer variable. The formula of adding value to pointer is given below:

1. new_address= current_address + (number * size_of(data type))

32-bit

For 32-bit int variable, it will add 2 * number.

64-bit

For 64-bit int variable, it will add 4 * number.

Let's see the example of adding value to pointer variable on 64-bit architecture.

```c
#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+3;   //adding 3 to pointer variable
printf("After adding 3: Address of p variable is %u \n",p);
return 0;
}
```

Output

```
Address of p variable is 3214864300
After adding 3: Address of p variable is 3214864312
```

As you can see, the address of p is 3214864300. But after adding 3 with p variable, it is 3214864312, i.e., 4*3=12 increment. Since we are using 64-bit architecture, it increments 12. But if we were using 32-bit architecture, it was incrementing to 6 only, i.e., 2*3=6. As integer value occupies 2-byte memory in 32-bit OS.

C Pointer Subtraction

Like pointer addition, we can subtract a value from the pointer variable. Subtracting any number from a pointer will give an address. The formula of subtracting value from the pointer variable is given below:

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

32-bit

For 32-bit int variable, it will subtract 2 * number.

64-bit

For 64-bit int variable, it will subtract 4 * number.

Let's see the example of subtracting value from the pointer variable on 64-bit architecture.

```c
#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-3; //subtracting 3 from pointer variable
printf("After subtracting 3: Address of p variable is %u \n",p);
```

```
9.  return 0;
10. }
```

Output

```
Address of p variable is 3214864300
After subtracting 3: Address of p variable is 3214864288
```

You can see after subtracting 3 from the pointer variable, it is 12 (4*3) less than the previous address value.

However, instead of subtracting a number, we can also subtract an address from another address (pointer). This will result in a number. It will not be a simple arithmetic operation, but it will follow the following rule.

If two pointers are of the same type,

1. Address2 -
   Address1 = (Subtraction of two addresses)/size of data type which pointer points

Consider the following example to subtract one pointer from an another.

```
1.  #include<stdio.h>
2.  void main ()
3.  {
4.      int i = 100;
5.      int *p = &i;
6.      int *temp;
7.      temp = p;
8.      p = p + 3;
9.      printf("Pointer Subtraction: %d - %d = %d",p, temp, p-temp);
10. }
```

Output

```
Pointer Subtraction: 1030585080 - 1030585068 = 3
```

Illegal arithmetic with pointers

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, for example, addition, and multiplication. A list of such operations is given below.

- Address + Address = illegal(addresses of two pointers cannot be added)
- Address * Address = illegal(addresses of two pointers cannot be multiply)
- Address % Address = illegal
- Address / Address = illegal

- Address & Address = illegal
- Address ^ Address = illegal
- Address | Address = illegal
- ~Address = illegal