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

1. Increment using pointers

- We know that "++" and "--" are used as the increment and decrement operators in C. They are unary operators, used in prefix or postfix manner with numeric variable operands, and they increment or decrement the value of the variable by one.
- Assume that an integer variable "x" is created at address 1000 in the memory, with 10 as its value. Then, "x++" makes the value of "x" as 11.

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

Address of p variable is 3214864300

After increment: Address of p variable is 3214864304

Traversing an array by using pointer

```
#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

Pointers can be incremented or decremented, which allows for traversing arrays:

1. 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)

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

Example

```
#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 the immidiate pr evious
    location.
}
```

Output

Address of p variable is 3214864300

After decrement: Address of p variable is 3214864296

3) Pointer Addition

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

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

syntax

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

Example

```
#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

4) 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:

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

```
#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);
return 0;
    }
```

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.

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

int *p = arr; // p points to the first element

printf("%d", *p); // Outputs 1

p++; // Moves to the next integer

printf("%d", *p); // Outputs 2
```

Null Pointers

A null pointer is a pointer that does not point to any valid memory location. It's often used to signify that the pointer is not initialized.

```
int *p = NULL;
```

Access and Manipulate Values using Pointer

The value of the variable which is pointed by a pointer can be accessed and manipulated by using the pointer variable. You need to use the asterisk (*) sign with the pointer variable to access and manipulate the variable's value.

Example

In the below example, we are taking an integer variable with its initial value and changing it with the new value.

```c
#include <stdio.h>

int main() {
  int x = 10;

  // Pointer declaration and initialization
  int * ptr = & x;

  // Printing the current value
  printf("Value of x = %d\n", * ptr);

  // Changing the value
  * ptr = 20;

  // Printing the updated value
  printf("Value of x = %d\n", * ptr);

  return 0;
}
```

Output

Value of x = 10

Value of x = 20