SNS COLLEGE OF TECHNOLOGY

Coimbatore – 35

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

PROGRAMMING IN C AND DATA STRUCTURES

I YEAR - II SEM

UNIT III - ARRAYS AND INTRODUCTION TO DATA STRUCTURES

TOPIC – POINTER ARITHMETIC

Introduction

- Pointers variables are also known as **address data types** because they are used to store the address of another variable.
- The address is the memory location that is assigned to the variable. It doesn't store any value.
- 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.
- Hence, there are only a few operations that are allowed to perform on Pointers in C language. The operations are slightly different from the ones that we generally use for mathematical calculations.

Operations used on Pointers

- Increment/Decrement of a Pointer

  Rule for Increment: **new_address= current_address + i * size_of(data type)**

  Rule for Decrement: **new_address= current_address - i * size_of(data type)**

- Addition of integer to a pointer

  Rule for Addition: **new_address= current_address + (number * size_of(data type))**

- Subtraction of integer to a pointer

  Rule for Subtraction: **new_address= current_address - (number * size_of(data type))**

- Subtracting two pointers of the same type

  Rule for Subtraction of two pointer of same type:

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

- Comparison of pointers of the same type

  Rule for Comparing pointers: **pointer 1 <comparison operator> pointer 2**

Increment / Decrement of a pointer

Increment: It is a condition that also comes under addition. When a pointer is incremented, it actually increments by the number equal to the size of the data type for which it is a pointer.

For Example:

If an integer pointer that stores **address 1000** is incremented, then it will increment by 4(**size of an int**) and the new address it will points to **1004**. While if a float type pointer is incremented then it will increment by 4(**size of a float**) and the new address will be **1004**.

Decrement: It is a condition that also comes under subtraction. When a pointer is decremented, it actually decrements by the number equal to the size of the data type for which it is a pointer.

For Example:

If an integer pointer that stores **address 1000** is decremented, then it will decrement by 4(**size of an int**) and the new address it will points to **996**. While if a float type pointer is decremented then it will decrement by 4(**size of a float**) and the new address will be **996**.

Example

```
#include <stdio.h>
int main()
{
            int a = 22;
            int *p = &a;
            printf("p = %u\n", p);                  // p = 6422288
            p++;
            printf("p++ = %u\n", p);                //p++ = 6422292          +4        // 4 bytes
            p--;
            printf("p-- = %u\n", p);                //p-- = 6422288          -4        // restored to original value

            float b = 22.22;
            float *q = &b;
            printf("q = %u\n", q);                    //q = 6422284
            q++;
            printf("q++ = %u\n", q);                //q++ = 6422288          +4        // 4 bytes
            q--;
            printf("q-- = %u\n", q);                //q-- = 6422284          -4        // restored to original value

            char c = 'a';
            char *r = &c;
            printf("r = %u\n", r);                  //r = 6422283
            r++;
            printf("r++ = %u\n", r);                //r++ = 6422284          +1        // 1 byte
            r--;
            printf("r-- = %u\n", r);                //r-- = 6422283          -1        // restored to original value

            return 0;   }
```

- Here %u – unsigned int
- We can also use %x for hexa decimal values

Addition

When a pointer is added with a value, the value is first multiplied by the size of data type and then added to the pointer.

```c
// C program to illustrate pointer Addition
#include <stdio.h>

// Driver Code
int main()
{
    // Integer variable
    int N = 4;

    // Pointer to an integer
    int *ptr1, *ptr2;

    // Pointer stores the address of N
    ptr1 = &N;
    ptr2 = &N;

    printf("Pointer ptr2 before Addition: ");
    printf("%p \n", ptr2);

    // Addition of 3 to ptr2
    ptr2 = ptr2 + 3;
    printf("Pointer ptr2 after Addition: ");
    printf("%p \n", ptr2);

    return 0;
}
```

```
Pointer ptr2 before Addition: 0x7ffca373da9c
Pointer ptr2 after Addition: 0x7ffca373daa8
```

Subtraction

When a pointer is subtracted with a value, the value is first multiplied by the size of the data type and then subtracted from the pointer.

```c
// C program to illustrate pointer Subtraction
#include <stdio.h>

// Driver Code
int main()
{
    // Integer variable
    int N = 4;

    // Pointer to an integer
    int *ptr1, *ptr2;

    // Pointer stores the address of N
    ptr1 = &N;
    ptr2 = &N;

    printf("Pointer ptr2 before Subtraction: ");
    printf("%p \n", ptr2);

    // Subtraction of 3 to ptr2
    ptr2 = ptr2 - 3;
    printf("Pointer ptr2 after Subtraction: ");
    printf("%p \n", ptr2);

    return 0;
}
```

```
Pointer ptr2 before Subtraction: 0x7ffd718ffebc
Pointer ptr2 after Subtraction: 0x7ffd718ffeb0
```

Subtraction of Two pointers

- ✓ The subtraction of two pointers is possible only when they have the same data type.
- ✓ The result is generated by calculating the difference between the addresses of the two pointers and calculating how many bits of data it is according to the pointer data type.
- ✓ The subtraction of two pointers gives the increments between the two pointers.
- ✓ **For Example:**

  Two integer pointers say **ptr1(address:1000)** and **ptr2(address:1004)** are subtracted. The difference between address is 4 bytes. Since the size of int is 4 bytes, therefore the **increment between ptr1 and ptr2** is given by **(4/4) = 1**.

```c
// C program to illustrate Subtraction
// of two pointers
#include <stdio.h>

// Driver Code
int main()
{
    int x = 6;   // Integer variable declaration
    int N = 4;

    // Pointer declaration
    int *ptr1, *ptr2;

    ptr1 = &N;  // stores address of N
    ptr2 = &x;   // stores address of x

    printf(" ptr1 = %u, ptr2 = %u\n", ptr1, ptr2);
  // %p gives an hexa-decimal value,
  //We convert it into an unsigned int value by using %u

    // Subtraction of ptr2 and ptr1
    x = ptr1 - ptr2;

    // Print x to get the Increment
    // between ptr1 and ptr2
    printf("Subtraction of ptr1 "
           "& ptr2 is %d\n",
           x);

    return 0;
}
```

```
 ptr1 = 2715594428, ptr2 = 2715594424
Subtraction of ptr1 & ptr2 is 1
```

Comparison of Pointers

We can compare the two pointers by using the comparison operators in C.

We can implement this by using all operators in C **>, >=, <, <=, ==, !=.**

It returns true for the valid condition and returns false for the unsatisfied condition.

Step 1 : Initialize the integer values and point these integer values to the pointer.

Step 2 : Now, check the condition by using comparison or relational operators on pointer variables.

Step 3 : Display the output.

Example

```c
#include <stdio.h>

int main() {

    // code
    int num1=5,num2=6,num3=5; //integer input
    int *p1=&num1;// addressing the integer input to pointer
    int *p2=&num2;
    int *p3=&num3;
    //comparing the pointer variables.
   if(*p1<*p2)
   {
       printf("\n%d less than %d",*p1,*p2);
   }
   if(*p2>*p1)
   {
     printf("\n%d greater than %d",*p2,*p1);
   }
   if(*p3==*p1)
   {
      printf("\nBoth the values are equal");
   }
   if(*p3!=*p2)
   {
      printf("\nBoth the values are not equal");
   }


    return 0;
}
```

```
5 less than 6
6 greater than 5
Both the values are equal
Both the values are not equal
```

Pointer Arithmetic on Arrays

- 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 the two addresses.
- An array name acts like a pointer constant.
- The value of this pointer constant is the address of the first element.
- **For Example:** if an array named arr then arr and &arr[0] can be used to reference array as a pointer.

Example

```c
// C program to illustrate the array
// traversal using pointers
#include <stdio.h>

// Driver Code
int main()
{

    int N = 5;

    // An array
    int arr[] = { 1, 2, 3, 4, 5 };

    // Declare pointer variable
    int* ptr;

    // Point the pointer to first
    // element in array arr[]
    ptr = arr;

    // Traverse array using ptr
    for (int i = 0; i < N; i++) {

        // Print element at which
        // ptr points
        printf("%d ", ptr[0]);
        ptr++;
    }
}
```

```
1 2 3 4 5
```

Illegal arithmetic with pointers

Address + Address = illegal

Address * Address = illegal

Address % Address = illegal

Address / Address = illegal

Address & Address = illegal

Address ^ Address = illegal

Address | Address = illegal

~Address = illegal