SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

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 ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

COURSE NAME : 23ITT101- PROBLEM SOLVING & C PROGRAMMING

I YEAR /I SEMESTER

Unit IV – FUNCTIONS AND POINTERS

Topic : Pointer - Arithmetic

Topics Covered

- Pointer Arithmetic
  - ✓ Increment and Decrement of a Pointer
  - ✓ Addition and Subtraction of Integer to Pointer
  - ✓ Subtraction of Pointers
  - ✓ Comparison of Pointers

Increment and 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.

  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 will point 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**.

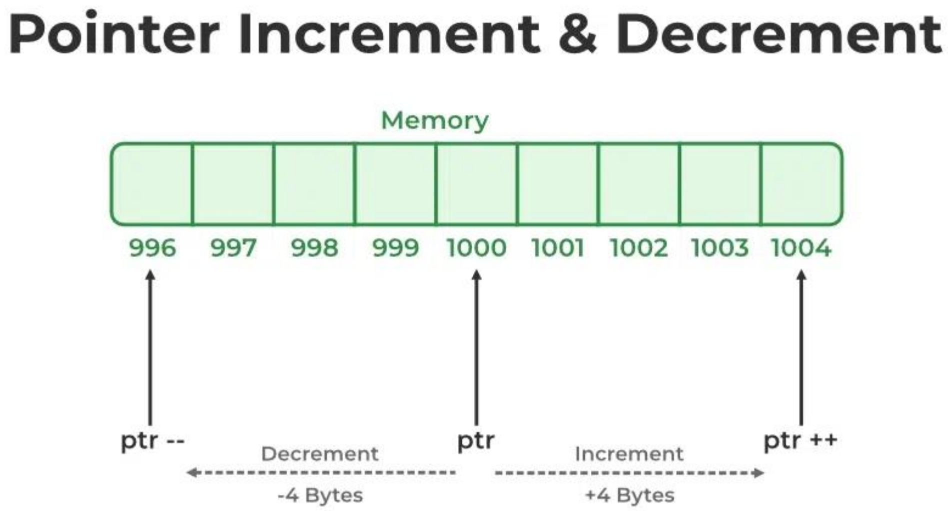
Increment and Decrement of a Pointer

- **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.

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 will point 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**.

Increment and Decrement of a Pointer



Increment and Decrement of a Pointer

```c
#include <stdio.h>
// pointer increment and decrement
//pointers are incremented and decremented by the size of the data type they point to
int main()
{
    int a = 22;
    int *p = &a;
    printf("p = %u\n", p);      // p = 6422288
    p++;
    printf("p++ = %u\n", p);    //p++ = 6422292       // + 4 bytes
    p--;
    printf("p-- = %u\n", p);    //p-- = 6422288        // -4 bytes

    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;
}
```

Output:

```
p = 1441900792
p++ = 1441900796
p-- = 1441900792
q = 1441900796
q++ = 1441900800
q-- = 1441900796
r = 1441900791
r++ = 1441900792
r-- = 1441900791
```

Addition of Integer to Pointer

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

Example:

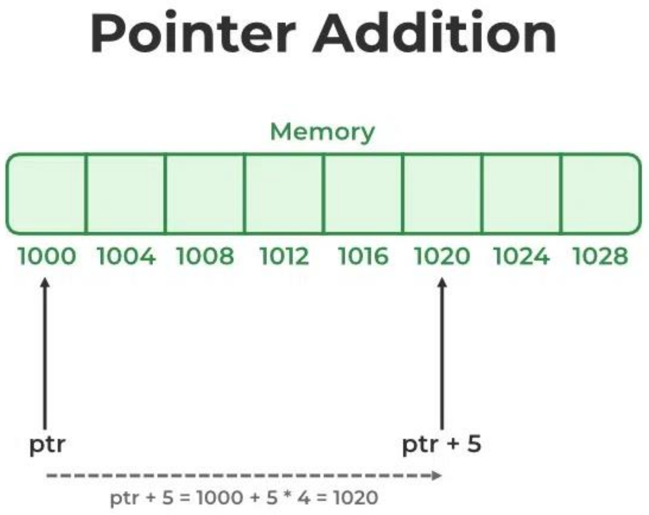
Consider the same example as above where the **ptr** is an **integer pointer** that stores **1000** as an address. If we add integer 5 to it using the expression, **ptr = ptr + 5,** then, the final address stored in the ptr will be **ptr = 1000 + sizeof(int) * 5 = 1020.**

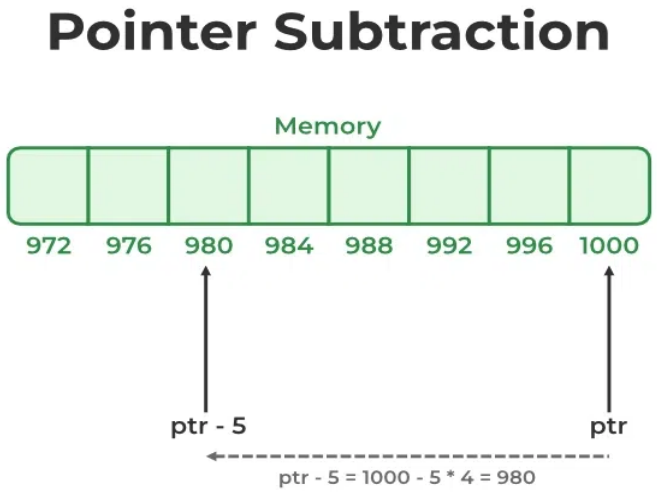
Subtraction of Integer to Pointer

- When a pointer is subtracted with an integer value, the value is first multiplied by the size of the data type and then subtracted from the pointer similar to addition.

Example:

Consider the same example as above where the ptr is an integer pointer that stores 1000 as an address. If we subtract integer 5 from it using the expression, ptr = ptr – 5, then, the final address stored in the ptr will be ptr = 1000 – sizeof(int) * 5 = 980.


Pointer Addition
Memory
1000
1004
1008
1012
1016
1020
1024
1028
ptr
ptr + 5
ptr + 5 = 1000 + 5 * 4 = 1020


Pointer Subtraction
Memory
972
976
980
984
988
992
996
1000
ptr - 5
ptr
ptr - 5 = 1000 - 5 * 4 = 980

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.

Example:

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

Subtraction of Two Pointers

If ptr1 and ptr2 are pointers to elements of the same array:

$$\text{ptr1} - \text{ptr2} = \frac{(\text{address of ptr1}) - (\text{address of ptr2})}{\text{sizeof(type)}}$$

Subtraction of Two Pointers

```
#include <stdio.h>
#include <stddef.h>

int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int *ptr1 = &arr[2];  // Points to 3rd element (value 30)
    int *ptr2 = &arr[0];  // Points to 1st element (value 10)

    ptrdiff_t diff = ptr1 - ptr2;

    printf("Difference between pointers: %td\n", diff); // Output: 2
    return 0;
}
```

Comparison of Pointers

Compare the two pointers by using the comparison operators in C 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.

Comparison to NULL

A pointer can be compared or assigned a NULL value irrespective of what is the pointer type. Such pointers are called NULL pointers and are used in various pointer-related error-handling methods

```
// C Program to demonstrate the pointer comparison with NULL  value
#include <stdio.h>
int main()
{
    int* ptr = NULL;
    if (ptr == NULL) {
        printf("The pointer is NULL");
    }
    else {
        printf("The pointer is not NULL");
    }
    return 0;
}
```

Output:

```
The pointer is NULL
```

Comparison operators on Pointers using an array

Step 1: First, declare the length of an array and array elements.

Step 2: Declare the pointer variable and point it to the first element of an array.

Step 3: Initialize the count_even and count_odd. Iterate the for loop and check the conditions for the number of odd elements and even elements in an array

Step 4: Increment the pointer location ptr++ to the next element in an array for further iteration.

Step 5: Print the result.

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.

Example: if an array is named **arr** then **arr and &arr[0]** can be used to reference the array as a pointer.

Pointer Arithmetic on Arrays

```
// C program to illustrate the array traversal using pointers

#include <stdio.h>
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++;
    }
}
```

Output:

```
1 2 3 4 5
```

Summary

- Pointer is a derived data type and important feature in C.
- Computer's memory is a sequential collection of storage cells, each cell commonly known as a byte, has a number called address associated with it.
- Pointers support dynamic memory management
    - Pointer constants - only we can use them to store data values
    - Pointer values may change from one run of the program to another.
- The variable that contains a pointer value called pointer variable.

Thank
you