INTRODUCTION TO POINTERS

A pointer is defined as a derived data type that can store the address of other C variables or a memory location. We can access and manipulate the data stored in that memory location using pointers. As the pointers in C store the memory addresses, their size is independent of the type of data they are pointing to.

```
 datatype * ptr;
int var = 10;
int * ptr;
ptr = &var;
```

```
// C program to illustrate Pointers

#include <stdio.h>

void introptr()
{
    int var = 10;

    // declare pointer variable
    int* ptr;

    // note that data type of ptr and var must be same
    ptr = &var;

    // assign the address of a variable to a pointer
    printf("Value at ptr = %p \n", ptr);
    printf("Value at var = %d \n", var);
    printf("Value at *ptr = %d \n", *ptr);
}

// Driver program
int main()
{
    introptr();
    return 0;
}
```

# Array of Pointers in C

In C, a pointer array is a homogeneous collection of indexed pointer variables that are references to a memory location. It is generally used in C Programming when we want to point at multiple memory locations of a similar data type in our C program. We can access the data by dereferencing the pointer pointing to it.

**Syntax:**

```
pointer_type *array_name [array_size];
```

Here,

- **pointer_type:** Type of data the pointer is pointing to.
- **array_name:** Name of the array of pointers.
- **array_size:** Size of the array of pointers.

***Note:*** *It is important to keep in mind the operator precedence and associativity in the array of pointers declarations of different type as a single change will mean the whole different thing. For example, enclosing *array_name in the parenthesis will mean that array_name is a pointer to an array.*

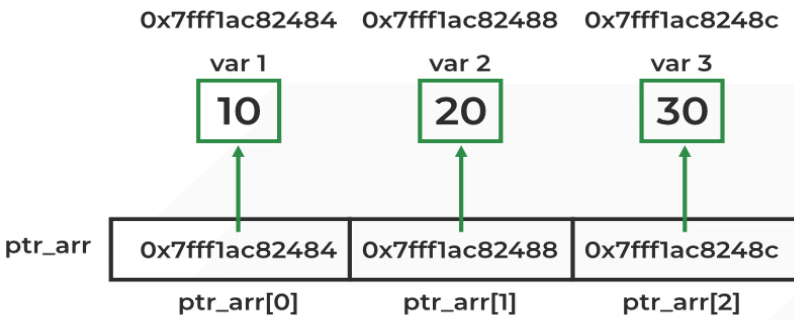
**Example:**

```c
// C program to demonstrate the use of array of pointers
#include <stdio.h>

int main()
{
    // declaring some temp variables
    int var1 = 10;
    int var2 = 20;
    int var3 = 30;

    // array of pointers to integers
    int* ptr_arr[3] = { &var1, &var2, &var3 };

    // traversing using loop
    for (int i = 0; i < 3; i++) {
        printf("Value of var%d: %d\tAddress: %p\n", i + 1, *ptr_arr[i],
ptr_arr[i]);
    }

    return 0;
}
```

**Output**

```
Value of var1: 10    Address: 0x7fff1ac82484
```

```
Value of var2: 20    Address: 0x7fff1ac82488
Value of var3: 30    Address: 0x7fff1ac8248c
```




### *Array of Pointers to Different Types*

*Not only we can define the array of pointers for basic data types like int, char, float, etc. but we can also define them for derived and user-defined data types such as arrays, structures, etc. Let's consider the below example where we create an array of pointers pointing to a function for performing the different operations.*

### Example:

```c
// C program to illustrate the use of array of pointers to
// function
#include <stdio.h>

// some basic arithmetic operations
void add(int a, int b) {
  printf("Sum : %d\n", a + b);
}

void subtract(int a, int b) {
    printf("Difference : %d\n", a - b);
}

void multiply(int a, int b) {
    printf("Product : %d\n", a * b);
}

void divide(int a, int b) {
    printf("Quotient : %d", a / b);
}

int main() {

    int x = 50, y = 5;
```

```
    // array of pointers to function of return type int
    void (*arr[4])(int, int)
        = { &add, &subtract, &multiply, &divide };
    for (int i = 0; i < 4; i++) {
        arr[i](x, y);
    }
    return 0;
}
```