C - Pointer to Pointer (Double Pointer)

In C, **double pointers** are those pointers which stores the address of another pointer. The first pointer is used to store the address of the variable, and the second pointer is used to store the address of the first pointer. That is why they are also known as a **pointer to pointer**.

Syntax of Double Pointer

The syntax to use double pointer can be divided into three parts:

Declaration

A double pointer can be declared similar to a single pointer. The difference is we have to place an additional **'*'** before the name of the pointer.

type **name;

Above is the declaration of the double pointer with some **name** to the given **type**.

Initialization

The double pointer stores the address of another pointer to the same type.

***name = &single_ptr;** // After declaration*

***type **name = &single_ptr;** // With declaration*

Dereferencing

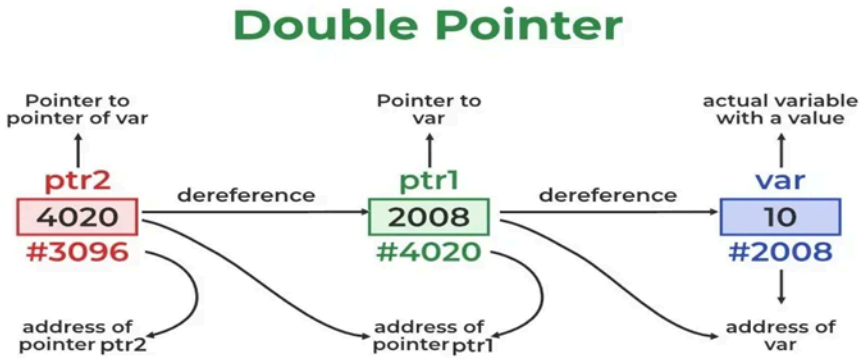
To access the value pointed by double pointer, we have to use the dereference operator * two times.

***name;** // Gives you the address of the single pointer*

***name;** // Gives you the value of the variable it points to*

In C, a double pointer behaves similarly to a normal pointer. So, the **size of the double-pointer** variable is always equal to the normal pointers i.e. only depending on the operating system and CPU architecture.

- **8 bytes** for a **64-bit System**
- **4 bytes** for a **32-bit System**



Examples of Double Pointer

The below examples demonstrate the use of double pointers for different applications in C:

Find the Size of Double Pointer

```c
#include <stdio.h>

int main() {

    // Defining single and double pointers
    int a = 5;
    int* ptr = &a;
    int** d_ptr = &ptr;

    // Size of double pointer
    printf("%d bytes", sizeof(d_ptr));

    return 0;
}
```

Output

```
8 bytes
```

Note: *The output of the above code also depends on the type of machine which is being used.*

Create a Dynamic 2D Array

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int m = 2, n = 3;

    // Create a double pointer
    int** arr;

    // Allocate memory for rows
    arr = (int**)malloc(m * sizeof(int*));

    // Allocate memory for each row
    for (int i = 0; i < m; i++)
        arr[i] = (int*)malloc(n * sizeof(int));

    // Initialize with some values
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            arr[i][j] = i * n + j + 1;
        }
    }
```

```
    // Print the array
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }

    // Free allocated memory
    for (int i = 0; i < m; i++) {
        free(arr[i]);
    }
    free(arr);

    return 0;
}
```

Output

```
1 2 3

4 5 6
```

Explanation: A 2D array is an array where each element is essentially a 1D array. This can be implemented using double pointers. The double pointer points to the first element of the 2D array, and each pointer it references points to a dynamically allocated 1D array using malloc().

Pass Array of Strings to Function

```
#include <stdio.h>

// Function that takes array of strings as argument
void print(char** arr, int n) {
    for (int i = 0; i < n; i++)
        printf("%s\n", *(arr + i));
}

int main() {
    char* arr[10] = {"Good", "Better",
                     "Best"};
    print(arr, 3);
    return 0;
}
```

Output

```
Good

Better

Best
```

Explanation: Array of strings are generally stored as array of pointer to strings. This can be passed using double pointers to function.

Application of Double Pointers in C

Following are the main uses of pointer to pointers in C:

- They are used in the dynamic memory allocation of multidimensional arrays.
- They can be used to store multilevel data such as the text document paragraph, sentences, and word semantics.
- They are used in data structures to directly manipulate the address of the nodes without copying.
- They can be used as function arguments to manipulate the address stored in the local pointer.

```
#include <stdio.h>

int main() {

    // A variable
    int var = 10;

    // Pointer to int
    int *ptr1 = &var;

    // Pointer to pointer (double pointer)
    int **ptr2 = &ptr1;

    printf("var: %d\n", var);
    printf("*ptr1: %d\n", *ptr1);
    printf("**ptr2: %d", **ptr2);

    return 0;
}
```

Output

```
var: 10

*ptr1: 10

**ptr2: 10
```

Explanation: In this code, **ptr1** is a pointer that stores the address of the integer variable **var**. **ptr2** is a double pointer that stores the address of the pointer **ptr1**. ****ptr2** dereferences **ptr2** to get the value of **ptr1** (which is the address of **var**) and then dereferences that address to get the value of **var** itself.