Dangling, Void, Null and Wild Pointers

Dangling pointer

A pointer pointing to a memory location that has been deleted (or freed) is called dangling pointer.

There are three different ways where Pointer acts as dangling pointer.

[1] De-allocation of memory

```
// deallocating a memory pointed by ptr causes dangling pointer

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

int main()
{
    int *ptr = (int *)malloc(sizeof(int));

    // after below free call, ptr becomes a dangling pointer
    free(ptr);

    // No more a dangling pointer
    ptr = NULL;
}
```

[2] Function Call

```
// The pointer pointing to local variable becomes dangling when local
// variable is not static.

#include<stdio.h>

int *fun()
{
    // x is local variable and goes out of
    // scope after an execution of fun() is over.

    int x = 5; // address 1805N

    return &x;
}

// Driver Code

int main()
{
    int *p = fun();

    // p points to something which is not valid anymore
    printf("%d", *p);
    return 0;
}
```

Output:
A garbage Address

The above problem doesn’t appear (or p doesn’t become dangling) if **x** is a **static** variable.

```
// The pointer pointing to local variable doesn't become dangling when local
// variable is static.

#include<stdio.h>

int *fun()
{
    // x now has scope throughout the program
    static int x = 5;

    return &x;
}

int main()
{
    int *p = fun();

    // Not a dangling pointer as it points to static variable.
    printf("%d",*p);
}
```

Output:
5

[3] Variable goes out of scope

```
void main()
{
   int *ptr = NULL;
   .....
   .....
   {
       int ch;
       ptr = &ch;
   }
   .....
   // Here ptr is dangling pointer
}
```

Void pointer

- **Void** pointer is a specific pointer type – **void *** – a pointer that points to some data location in storage, which doesn't have any specific type.
- **Void** refers to the type. Basically the type of data that it points to is can be any.
- If we assign address of **char** data type to **void** pointer it will become **char** Pointer, if **int** data type then **int** pointer and so on.
- Any pointer type is convertible to a void pointer hence it can point to any value.

Important Points

-**void** pointers cannot be dereferenced. It can however be done using typecasting the void pointer
-Pointer arithmetic is not possible on pointers of **void** due to lack of concrete value and thus size.

Example:

```
#include<stdlib.h>

int main()
{
    int x = 4;
    float y = 5.5;

    //A void pointer
    void *ptr;
    ptr = &x;

    // (int*)ptr - does type casting of void
    // *((int*)ptr) dereferences the typecasted  void pointer variable.
    printf("Integer variable is = %d", *( (int*) ptr) );

    // void pointer is now float
    ptr = &y;
    printf("\nFloat variable is= %f", *( (float*) ptr) );

    return 0;
}
```

Output:

```
Integer variable is = 4
Float variable is= 5.500000
```

NULL Pointer

- NULL Pointer is a pointer which is pointing to nothing.
- In case, if we don't have address to be assigned to a pointer, then we can simply use NULL.

```
#include <stdio.h>
int main()
{
    // Null Pointer
    int *ptr = NULL;

    printf("The value of ptr is %u", ptr);
    return 0;
}
```

Output :

The value of ptr is 0

Important Points

-**NULL** vs Uninitialized pointer

- An uninitialized pointer stores an undefined value.
- A null pointer stores a defined value, but one that is defined by the environment to not be a valid address for any member or object.
-

-**NULL** vs **Void** Pointer – Null pointer is a value, while void pointer is a type

Wild pointer

- A pointer which has not been initialized to anything (not even NULL) is known as wild pointer.
- The pointer may be initialized to a non-NULL garbage value that may not be a valid address.

```
int main()
{
    int *p;  /* wild pointer */

    int x = 10;

    // p is not a wild pointer now
    p = &x;

    return 0;
}
```