MyAnatomy

Dangling pointer

A dangling pointer is a pointer that points to a deleted (or freed) memory location.

Example:

```
int* ptr = new int(12);
cout << *ptr;

delete ptr; // After deleting ptr memory, ptr becomes a dangling pointer.
```

void pointer

Void pointer is a specific pointer type - void * - a pointer that points to an unspecified data location in storage. The type is referred to as void. Basically, the data that it points to can be anything. If we assign a char data type address to a void pointer, it will become a char pointer, an int data type address will become an int pointer, and so on. Because any pointer type can be converted to a void pointer, it can point to any value.

Example:

```
int i = 4;
float f = 5.5;
char c = 'A';

void *ptr;          //ptr is declared as void pointer
ptr = &i;   // assigning integer value to void type pointer

// (int*)ptr - does type casting of void
// *((int*)ptr) extracting value of pointer
cout << "Integer variable is = " << *((int*) ptr);

ptr = &f;   // assigning float value to void type pointer
cout << "\nFloat variable is= " << *((float*) ptr);

ptr = &c;   // assigning char value to void type pointer
cout << "\nChar variable is= " << *((char*) ptr);
```

null pointer

While declaring a pointer variable, if it is not assigned to anything then it contains garbage value. And that can lead to unexpected errors in your program. Hence, it is recommended to assign a NULL value to it.

When we assign NULL to a pointer, it means that it does not point to any valid address. NULL denotes the value 'zero'.

A pointer that is assigned a NULL value is called a NULL pointer in C++.

We can give a pointer a null value by assigning it zero to it.

```
int *ptr = 0;
```

We can also use the NULL keyword, which is nothing but a predefined constant for null pointer.

```
int *ptr = NULL;
```

wild pointer

A wild pointer is one that has not been initialised to anything (not even NULL). The pointer may be set to a non-NULL garbage value that is not a valid address.

Example:

```
int *ptr_int;
float *ptr_float
char *pre_char;
```

Memory leak in C++ and How to avoid it?

Memory leakage occurs in C++ when programmers allocate memory by using new keyword and forgets to deallocate the memory by using delete() function or delete[] operator.

The delete operator should be used to free a single allocated memory space, whereas the delete [] operator should be used to free an array of data values.

Disadvantage with memory leakage:
If a program has memory leaks, then its memory usage is satirically increasing since all systems have limited amount of memory and memory is costly. Hence it will create problems.

Example1: Memory Leak

```
#include <iostream>
using namespace std;
void func_to_show_memory_leak() {
	int* ptr = new int(5);
}
int main(void) {
	func_to_show_memory_leak();
}
```

Example2: Memory Leak handled

```
#include <iostream>
using namespace std;
void func_to_handle_memory_leak() {
	int* ptr = new int(5);
	delete ptr;
}
int main(void) {
	func_to_handle_memory_leak();
}
```

Note: Always write delete pointer for matching of new pointer in C++ and always write code between these new and delete as explained in above example. In above example, no memory is wasted because when we are coming out from the function, we are deallocating the memory by using delete function.