Dangling, Void, Null and Wild Pointers in C++

Dangling pointer

Dangling pointer is a pointer pointing to a memory location that has been freed (or deleted). There are different ways where Pointer acts as dangling pointer

Function Call

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

```
int *show(void) {
   int n = 76; /* ... */ return &n;
}
```

Output

```
Output of this program will be garbage address.
```

De-allocation of memory

```
int main() {
   float *p = (float *)malloc(sizeof(float));
   //dynamic memory allocation.
   free(p);
   //after calling free()
   p becomes a dangling pointer p = NULL;
   //now p no more a dangling pointer.
}
```

Variable goes out of scope

```
int main() {
   int *p //some code// {
      int c; p=&c;
   }
   //some code//
   //p is dangling pointer here.
}
```

Void pointer

Void pointer is a pointer which is not associate with any data types. It points to some data location in storage means points to the address of variables. It is also called general purpose pointer.

It has some limitations

- Pointer arithmetic is not possible of void pointer due to its concrete size.
- It can’t be used as dereferenced.

Example

```
#include<stdlib.h>
#include<iostream>
using namespace std;
int main() {
   int a = 7;
   float b = 7.6;
   void *p;
   p = &a;
   cout<<*( (int*) p)<<endl ;
   p = &b;
   cout<< *( (float*) p) ;
   return 0;
}
```

Output

```
7
7.600000
```

Algorithm

```
Begin
   Initialize a variable a with integer value and variable b with float value.
   Declare a void pointer p.
   (int*)p = type casting of void.
   p = &b mean void pointer p is now float.
   (float*)p = type casting of void.
   Print the value.
End.
```

Null Pointer

Null pointer is a pointer which points nothing.

Some uses of null pointer are

- To initialize a pointer variable when that pointer variable isn’t assigned any valid memory address yet.
- To pass a null pointer to a function argument if we don’t want to pass any valid memory address.

- To check for null pointer before accessing any pointer variable. So that, we can perform error handling in pointer related code e.g. dereference pointer variable only if it’s not NULL.

Example

Live Demo

```
#include <iostream>
using namespace std;
int main() {
   int *p= NULL; //initialize the pointer as null.
   cout<<"The value of pointer is ";
   cout<<p;
   return 0;
}
```

Output

```
The value of pointer is 0.
```

Wild Pointer

Wild pointers are pointers those are point to some arbitrary memory location. (not even NULL)

```
int main() {
   int *ptr; //wild pointer
   *ptr = 5;
}
```