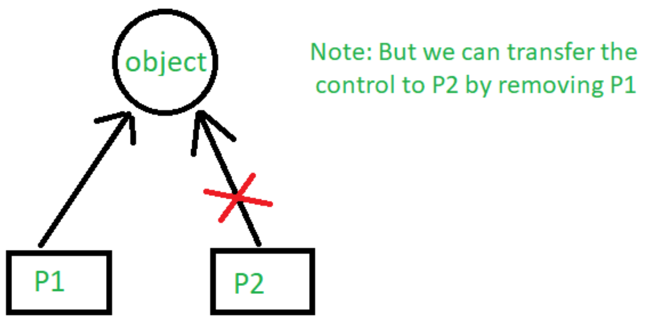
unique_ptr, shared_ptr and weak_ptr in C++

unique_ptr

unique_ptr stores one pointer only. We can assign a different object by removing the current object from the pointer. Notice the code below. First, the *unique_pointer* is pointing to *P1*. But, then we remove *P1* and assign *P2* so the pointer now points to *P2*.



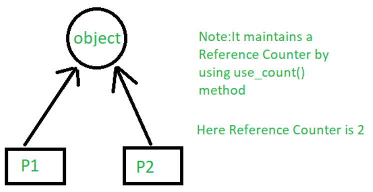
```
#include <iostream>
using namespace std;
#include <memory>

class Rectangle {
    int length;
    int breadth;

public:
    Rectangle(int l, int b){
        length = l;
        breadth = b;
    }

    int area(){
        return length * breadth;
    }
};

int main(){

    unique_ptr<Rectangle> P1(new Rectangle(10, 5));
    cout << P1->area() << endl; // This'll print 50

    // unique_ptr<Rectangle> P2(P1);
    unique_ptr<Rectangle> P2;
    P2 = move(P1);

    // This'll print 50
    cout << P2->area() << endl;

    // cout<<P1->area()<<endl;
    return 0;
}
```

Output

```
 50
50
```

shared_ptr

By using *shared_ptr* more than one pointer can point to this one object at a time and it'll maintain a **Reference Counter** using ***use_count()* method.**


object
P1
P2
Note:It maintains a Reference Counter by using use_count() method
Here Reference Counter is 2

```
#include <iostream>
using namespace std;
#include <memory>
```

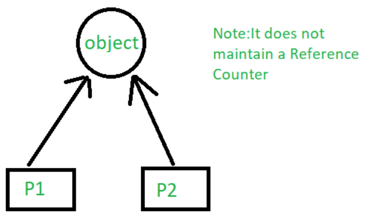
```
class Rectangle {
    int length;
    int breadth;

public:
    Rectangle(int l, int b)
    {
        length = l;
        breadth = b;
    }

    int area()
    {
        return length * breadth;
    }
};

int main()
{

    shared_ptr<Rectangle> P1(new Rectangle(10, 5));
    // This'll print 50
    cout << P1->area() << endl;

    shared_ptr<Rectangle> P2;
    P2 = P1;

    // This'll print 50
    cout << P2->area() << endl;

    // This'll now not give an error,
    cout << P1->area() << endl;

    // This'll also print 50 now
    // This'll print 2 as Reference Counter is 2
    cout << P1.use_count() << endl;
    return 0;
}
```

Output

```
 50
50
50
2
```

weak_ptr

It’s much more similar to shared_ptr except it’ll not maintain a **Reference Counter**. In this case, a pointer will not have a stronghold on the object. The reason is if suppose pointers are holding the object and requesting for other objects then they may form a **Deadlock.**


object
Note:It does not maintain a Reference Counter
P1
P2