Smart Pointer Introduction

A *Smart Pointer* is a wrapper class over a pointer with an operator like * and -> overloaded. The objects of the smart pointer class look like normal pointers. But, unlike *Normal Pointers* it can deallocate and free destroyed object memory.

The idea is to take a class with a pointer, destructor and overloaded operators like * and ->. Since the destructor is automatically called when an object goes out of scope, the dynamically allocated memory would automatically be deleted (or reference count can be decremented). Consider the following simple *SmartPtr* class.

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

class SmartPtr {
    int* ptr; // Actual pointer
public:
    // Constructor: Refer https:// www.geeksforgeeks.org/g-fa
    // for use of explicit keyword
    explicit SmartPtr(int* p = NULL) { ptr = p; }

    // Destructor
    ~SmartPtr() { delete (ptr); }

    // Overloading dereferencing operator
    int& operator*() { return *ptr; }
};

int main()
{
    SmartPtr ptr(new int());
    *ptr = 20;
    cout << *ptr;

    // We don't need to call delete ptr: when the object
    // ptr goes out of scope, the destructor for it is automa
    // called and destructor does delete ptr.
```

```
    return 0;
}
```

Output

```
20
```

This only works for *int*. So, we'll have to create Smart Pointer for every object? No, there's a solution, *Template*. In the code below as you can see *T* can be of any type.

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

// A generic smart pointer class
template <class T>
class SmartPtr {
    T* ptr; // Actual pointer
public:
    // Constructor
    explicit SmartPtr(T* p = NULL) { ptr = p; }

    // Destructor
    ~SmartPtr() { delete (ptr); }

    // Overloading dereferencing operator
    T& operator*() { return *ptr; }

    // Overloading arrow operator so that
    // members of T can be accessed
    // like a pointer (useful if T represents
    // a class or struct or union type)
    T* operator->() { return ptr; }
};

int main()
{
    SmartPtr<int> ptr(new int());
```

```
    *ptr = 20;
    cout << *ptr;
    return 0;
}
```

Output

```
20
```

Note: Smart pointers are also useful in the management of resources, such as file handles or network sockets.

Types of Smart Pointers

1. **unique_ptr**
2. **shared_ptr**
3. **weak_ptr**