How to implement user defined Shared Pointers in C++

Difficulty Level : Hard • Last Updated : 17 Oct, 2022

Shared Pointers :

A std::shared_ptr is a container for raw pointers. It is a reference counting ownership model i.e. it maintains the reference count of its contained pointer in cooperation with all copies of the std::shared_ptr. So, the counter is incremented each time a new pointer points to the resource and decremented when destructor of the object is called.

Reference Counting :

It is a technique of storing the number of references, pointers or handles to a resource such as an object, block of memory, disk space or other resources.

An object referenced by the contained raw pointer will not be destroyed until reference count is greater than zero i.e. until all copies of std::shared_ptr have been deleted.

When to use: We should use shared_ptr when we want to assign one raw pointer to multiple owners.

For more information and details about shared and other smart pointers, please read here.

User Defined Implementation of Shared pointers:

Program:

CPP

```
#include <iostream>
```

```
// Class representing a reference counter class
class Counter
{
public:
    // Constructor
    Counter()
        : m_counter(0){};

    Counter(const Counter&) = delete;
    Counter& operator=(const Counter&) = delete;

    // Destructor
    ~Counter() {}

    void reset()
    {
      m_counter = 0;
    }

    unsigned int get()
    {
      return m_counter;
    }

    // Overload post/pre increment
    void operator++()
    {
      m_counter++;
    }

    void operator++(int)
    {
      m_counter++;
    }

    // Overload post/pre decrement
    void operator--()
    {
      m_counter--;
    }
    void operator--(int)
    {
      m_counter--;
    }

    // Overloading << operator
    friend ostream& operator<<(ostream& os,
```

```
            << endl;
        return os;
    }

private:
    unsigned int m_counter{};
};

// Class representing a shared pointer
template <typename T>

class Shared_ptr
{
public:
    // Constructor
    explicit Shared_ptr(T* ptr = nullptr)
    {
        m_ptr = ptr;
        m_counter = new Counter();
        (*m_counter)++;
    }

    // Copy constructor
    Shared_ptr(Shared_ptr<T>& sp)
    {
        m_ptr = sp.m_ptr;
        m_counter = sp.m_counter;
        (*m_counter)++;
    }

    // Reference count
    unsigned int use_count()
    {
      return m_counter->get();
    }

    // Get the pointer
    T* get()
    {
      return m_ptr;
    }

    // Overload * operator
    T& operator*()
    {
      return *m_ptr;
    }
```

```
        {
          return m_ptr;
        }

        // Destructor
        ~Shared_ptr()
        {
            (*m_counter)--;
            if (m_counter->get() == 0)
            {
                delete m_counter;
                delete m_ptr;
            }
        }

        friend ostream& operator<<(ostream& os,
                                   Shared_ptr<T>& sp)
        {
            os << "Address pointed : " << sp.get() << endl;
            os << *(sp.m_counter) << endl;
            return os;
        }

    private:
        // Reference counter
        Counter* m_counter;

        // Shared pointer
        T* m_ptr;
    };

    int main()
    {
        // ptr1 pointing to an integer.
        Shared_ptr<int> ptr1(new int(151));
        cout << "--- Shared pointers ptr1 ---\n";
        *ptr1 = 100;
        cout << " ptr1's value now: " << *ptr1 << endl;
        cout << ptr1;

        {
            // ptr2 pointing to same integer
            // which ptr1 is pointing to
            // Shared pointer reference counter
            // should have increased now to 2.
            Shared_ptr<int> ptr2 = ptr1;
            cout << "--- Shared pointers ptr1, ptr2 ---\n";
```

```
        {
            // ptr3 pointing to same integer
            // which ptr1 and ptr2 are pointing to.
            // Shared pointer reference counter
            // should have increased now to 3.
            Shared_ptr<int> ptr3(ptr2);
            cout << "--- Shared pointers ptr1, ptr2, ptr3 "
                    "---\n";
            cout << ptr1;
            cout << ptr2;
            cout << ptr3;
        }

        // ptr3 is out of scope.
        // It would have been destructed.
        // So shared pointer reference counter
        // should have decreased now to 2.
        cout << "--- Shared pointers ptr1, ptr2 ---\n";
        cout << ptr1;
        cout << ptr2;
    }

    // ptr2 is out of scope.
    // It would have been destructed.
    // So shared pointer reference counter
    // should have decreased now to 1.
    cout << "--- Shared pointers ptr1 ---\n";
    cout << ptr1;

    return 0;
}
```

Output:

```
--- Shared pointers ptr1 ---
Address pointed : 0x1cbde70
Counter Value : 1

--- Shared pointers ptr1, ptr2 ---
Address pointed : 0x1cbde70
Counter Value : 2

Address pointed : 0x1cbde70
```

```
--- Shared pointers ptr1, ptr2, ptr3 ---
Address pointed : 0x1cbde70
Counter Value : 3

Address pointed : 0x1cbde70
Counter Value : 3

Address pointed : 0x1cbde70
Counter Value : 3

--- Shared pointers ptr1, ptr2 ---
Address pointed : 0x1cbde70
Counter Value : 2

Address pointed : 0x1cbde70
Counter Value : 2

--- Shared pointers ptr1 ---
Address pointed : 0x1cbde70
Counter Value : 1
```


Previous Next

1. Difference between constant pointer, pointers to constant, and constant pointers to constants
2. Function Interposition in C with an example of user defined malloc()
3. User Defined Literals in C++
4. Multi-set for user defined data type
5. User defined Data Types in C++
6. How to create an unordered_map of user defined class in C++?
7. How to create an unordered_set of user defined class or struct in C++?
8. 2D Vector In C++ With User Defined Size
9. Chat application between two processes using signals and shared memory
10. What are Wild Pointers? How can we avoid?

Article Contributed By :

mayank prasad
@mayank prasad

Vote for difficulty

Current difficulty : Hard

Easy | Normal | Medium | Hard | Expert

Improved By : prakashgibbi, vipugpta, leonlinzw

Article Tags : C Basics, cpp-pointer, C++

Practice Tags : CPP

Improve Article | Report Issue

Finance

Lifestyle

Knowledge

C#

SQL

Kotlin

Web Development

Web Tutorials

Django Tutorial

HTML

JavaScript

Bootstrap

ReactJS

NodeJS

Contribute

Write an Article

Improve an Article

Pick Topics to Write

Write Interview Experience

Internships

Video Internship