Programs for topic: smart pointers

Contents

Unique pointers

Program:

```cpp
// ============== Unique pointer =================

#include <iostream>
#include <memory>

using namespace std;

int main()
{
    cout << "======================================" << endl;
    unique_ptr<int> ptr1  {new int(56)};
    cout << *ptr1 << endl;
    cout << "======================================" << endl;
    unique_ptr<int> ptr2;
    // ptr2 = ptr1; // copying not allowed with unique pointer
    ptr2 = move(ptr1); // moving
    cout << *ptr2 << endl;
    cout << "======================================" << endl;

    return 0;
}
```

Result:

```
========================================
56
========================================
56
========================================
```

Shared pointers

Program:

```cpp
// ============= Shared pointer =================

#include <iostream>
#include <memory>

using namespace std;

int main()
{

    shared_ptr<int> ptr1 {new int(56)};
    cout << *ptr1 << endl;

    shared_ptr<int> ptr2 {ptr1};
    cout << *ptr2 << endl;

    shared_ptr<int> ptr3;
    ptr3 = ptr2;
    cout << *ptr3 << endl;

    cout << "========================================" << endl;
    cout << "use count " << ptr1.use_count() << endl;
    cout << "use count " << ptr2.use_count() << endl;
    cout << "use count " << ptr3.use_count() << endl;
    cout << "========================================" << endl;
    ptr2.reset();
    cout << "use count " << ptr1.use_count() << endl;
    cout << "use count " << ptr2.use_count() << endl;
    cout << "use count " << ptr3.use_count() << endl;
    cout << "========================================" << endl;
    ptr3.reset();
    cout << "use count " << ptr1.use_count() << endl;
    cout << "use count " << ptr2.use_count() << endl;
```

```
    cout << "use count " << ptr3.use_count() << endl;
    cout << "=====================================" << endl;

    return 0;
}
```

Result:

```
56
56
56
=====================================
use count 3
use count 3
use count 3
=====================================
use count 2
use count 0
use count 2
=====================================
use count 1
use count 0
use count 0
=====================================
```

shared pointer drawback with circular reference

Program:

```
// === shared pointer drawback with circular reference =====

#include <iostream>
#include <memory>

using namespace std;

class B;
class A{
    shared_ptr<B> b_ptr;
    public:
        void set_B(std::shared_ptr<B> &b){
            cout << "set B" << endl;
            b_ptr = b;
        }
        A(){ cout << "A constructor" << endl; }
```

```
        ~A(){ cout << "A destructor" << endl; }
};

class B{
    shared_ptr<A> a_ptr;
    public:
        void set_A(std::shared_ptr<A> &a){
            cout << "set A" << endl;
            a_ptr = a;
        }
        B(){ cout << "B constructor" << endl; }
        ~B(){ cout << "B destructor" << endl; }
};

int main()
{
    cout << "=======================================" << endl;
    shared_ptr<A> ptr1 = make_shared<A>();
    shared_ptr<B> ptr2 = make_shared<B>();
    cout << "=======================================" << endl;
    ptr1-> set_B(ptr2);
    ptr2-> set_A(ptr1);
    cout << "=======================================" << endl;

    return 0;
}
```

Result:

```
=======================================
A constructor
B constructor
=======================================
set B
set A
=======================================
```

weak pointer with circular reference

program:

```
// === weak pointer with circular reference =====

#include <iostream>
#include <memory>

using namespace std;
```

```
class B;
class A{
    shared_ptr<B> b_ptr;
    public:
        void set_B(std::shared_ptr<B> &b){
            cout << "set B" << endl;
            b_ptr = b;
        }
        A(){ cout << "A constructor" << endl; }
        ~A(){ cout << "A destructor" << endl; }
};

class B{
    weak_ptr<A> a_ptr;
    public:
        void set_A(std::shared_ptr<A> &a){
            cout << "set A" << endl;
            a_ptr = a;
        }
        B(){ cout << "B constructor" << endl; }
        ~B(){ cout << "B destructor" << endl; }
};

int main()
{
    cout << "=======================================" << endl;
    shared_ptr<A> ptr1 = make_shared<A>();
    shared_ptr<B> ptr2 = make_shared<B>();
    cout << "=======================================" << endl;
    ptr1-> set_B(ptr2);
    ptr2-> set_A(ptr1);
    cout << "=======================================" << endl;

    return 0;
}
```

Result:

```
=======================================
A constructor
B constructor
=======================================
set B
set A
=======================================
A destructor
B destructor
```