class B; //声明

class A {

public:

weak\_ptr<B> pb\_;

~A() {

cout << "A delete\n";

}

};

class B {

public:

shared\_ptr<A> pa\_;

~B() {

cout << "B delete\n";

}

};

void fun() {

shared\_ptr<B> pb(new B());

shared\_ptr<A> pa(new A());

cout << pb.use\_count() << endl; //1

cout << pa.use\_count() << endl; //1

pb->pa\_ = pa;

pa->pb\_ = pb;

cout << pb.use\_count() << endl; //2

cout << pa.use\_count() << endl; //2

}

int main() {

fun();

return 0;

}

class B; //声明

class A {

public:

weak\_ptr<B> pb\_;

~A() {

cout << "A delete\n";

}

};

class B {

public:

shared\_ptr<A> pa\_;

~B() {

cout << "B delete\n";

}

};

void fun() {

shared\_ptr<B> pb(new B());

shared\_ptr<A> pa(new A());

cout << pb.use\_count() << endl; //1

cout << pa.use\_count() << endl; //1

pb->pa\_ = pa;

pa->pb\_ = pb;

cout << pb.use\_count() << endl; //1

cout << pa.use\_count() << endl; //2

//B delete

//A delete

}

int main() {

fun();

return 0;

}

手动实现shared\_ptr, weak\_ptr

template <class T>

class WeakPtr; //为了用weak\_ptr的lock()，来生成share\_ptr用，需要拷贝构造用

template <class T>

class SharePtr {

public:

SharePtr(T\* p = 0): \_ptr(p) {

cnt = new Counter();

if(p)

cnt->s = 1;

cout << "in construct " << cnt->s << endl;

}

~SharePtr() {

release();

}

SharePtr(SharePtr<T> const& s) {

cout << "in copy con" << endl;

\_ptr = s.\_ptr;

(s.cnt)->s++;

cout << "copy construct" << (s.cnt)->s << endl;

cnt = s.cnt;

}

SharePtr(WeakPtr<T> const& w) //为了用weak\_ptr的lock()，来生成share\_ptr用，需要拷贝构造用

{

cout << "in w copy con " << endl;

\_ptr = w.\_ptr;

(w.cnt)->s++;

cout << "copy w construct" << (w.cnt)->s << endl;

cnt = w.cnt;

}

SharePtr<T>& operator=(SharePtr<T>& s) {

if(this != &s) {

release();

(s.cnt)->s++;

cout << "assign construct " << (s.cnt)->s << endl;

cnt = s.cnt;

\_ptr = s.\_ptr;

}

return \*this;

}

T& operator\*() {

return \*\_ptr;

}

T\* operator->() {

return \_ptr;

}

friend class WeakPtr<T>; //方便weak\_ptr与share\_ptr设置引用计数和赋值

protected:

void release() {

cnt->s--;

cout << "release " << cnt->s << endl;

if(cnt->s < 1) {

delete \_ptr;

if(cnt->w < 1) {

delete cnt;

cnt = NULL;

}

}

}

private:

T\* \_ptr;

Counter\* cnt;

};

template <class T>

class WeakPtr {

public: //给出默认构造和拷贝构造，其中拷贝构造不能有从原始指针进行构造

WeakPtr() {

\_ptr = 0;

cnt = 0;

}

WeakPtr(SharePtr<T>& s): \_ptr(s.\_ptr), cnt(s.cnt) {

cout << "w con s" << endl;

cnt->w++;

}

WeakPtr(WeakPtr<T>& w): \_ptr(w.\_ptr), cnt(w.cnt) {

cnt->w++;

}

~WeakPtr() {

release();

}

WeakPtr<T>& operator=(WeakPtr<T>& w) {

if(this != &w) {

release();

cnt = w.cnt;

cnt->w++;

\_ptr = w.\_ptr;

}

return \*this;

}

WeakPtr<T>& operator=(SharePtr<T>& s) {

cout << "w = s" << endl;

release();

cnt = s.cnt;

cnt->w++;

\_ptr = s.\_ptr;

return \*this;

}

SharePtr<T> lock() {

return SharePtr<T>(\*this);

}

bool expired() {

if(cnt) {

if(cnt->s > 0) {

cout << "empty" << cnt->s << endl;

return false;

}

}

return true;

}

friend class SharePtr<T>; //方便weak\_ptr与share\_ptr设置引用计数和赋值

protected:

void release() {

if(cnt) {

cnt->w--;

cout << "weakptr release" << cnt->w << endl;

if(cnt->w < 1 && cnt->s < 1) {

//delete cnt;

cnt = NULL;

}

}

}

private:

T\* \_ptr;

Counter\* cnt;

};