智能指针：shared_ptr

内存泄漏：当我们动态地申请了一片内存以后，在程序中没有主动去释放它的时候，导致他在程序中再也访问不到了。

所谓泄露的内存：进程中，属于进程，但是进程却访问不到的那片内存。

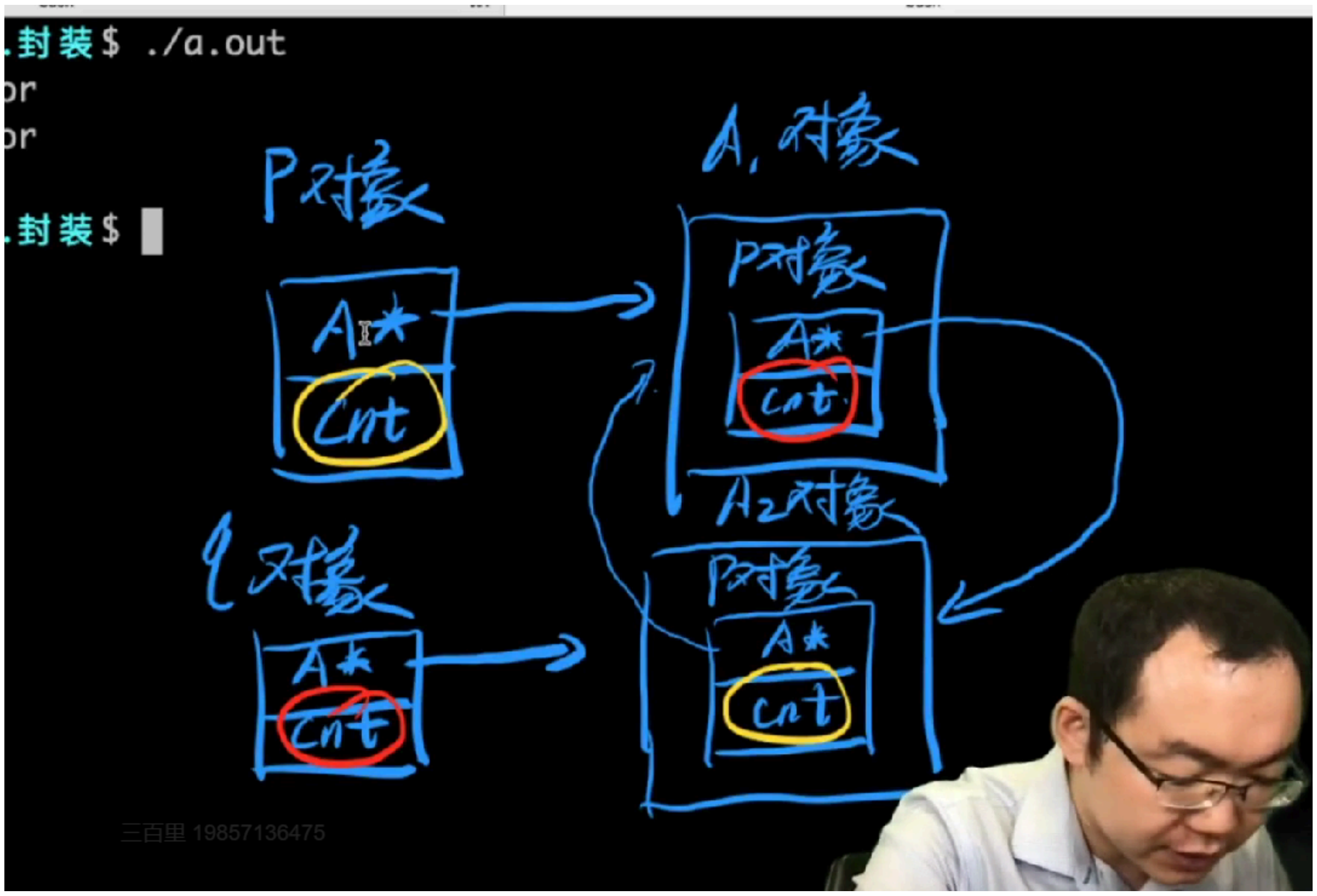
智能指针存在的经典问题：**环形引用**

环形引用

```
class A {
public:
    A() {
        cout << "default constructor" << endl;
    }
    ~A() {
        cout << "destructor" << endl;
    }
    shared_ptr<A> p;
};

int main() {
    shared_ptr<A> p = make_shared<A>();
    shared_ptr<A> q = make_shared<A>();
    p->p = q;
    q->p = p;
    p = nullptr;
    q = nullptr;

    return 0;
}
```



```
.封装$ ./a.out
or
or

.封装$ 
```

13.shared_ptr.cpp

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

#define BEGINS(x) namespace x {
#define ENDS(x) }

BEGINS(test1)
class A {
public:
    A() {
        cout << "default constructor" << endl;
    }
    ~A() {
        cout << "destructor" << endl;
    }
};

int main() {
    // 原生指针:
    //A *p = new A();
    //delete p; //因为默认析构没了，所以得加上这行才会调用析构。

    // 智能指针:
    shared_ptr<A> p = make_shared<A>();
    cout << p.use_count() << endl;
    shared_ptr<A> q = p;
    cout << p.use_count() << endl;
    cout << q.use_count() << endl;
    p = nullptr;
    cout << q.use_count() << endl;
    q = nullptr;
    cout << "end main" << endl;

    return 0;
}

ENDS(test1)


BEGINS(test2)
class A {
public:
    A() {
        cout << "default constructor" << endl;
    }
    ~A() {
        cout << "destructor" << endl;
    }
    shared_ptr<A> p;
};

int main() {
    shared_ptr<A> p = make_shared<A>();
    shared_ptr<A> q = make_shared<A>();
    p->p = q;
    q->p = p;
    p = nullptr;
    q = nullptr;

    return 0;
}
ENDS(test2)


BEGINS(test3)
class A {
public:
    A(int x = 3, int y = 4) : x(x), y(y) {
        cout << "default constructor" << endl;
    }
    int x, y;
    ~A() {
        cout << "destructor" << endl;
    }
};
ostream &operator<<(ostream &out, const A &a) {
    out << "class A : "<< a.x << "," << a.y;
    return out;
}
class ptr_data {
public:
    ptr_data(A *ptr, int *cnt = nullptr) :
    ptr(ptr), cnt(cnt) {
        if (cnt == nullptr) this->cnt = new int(1);
    }
    void increase_one() {
        *cnt += 1;
        return ;
    }
    void decrease_one() {
        *cnt -= 1;
        if (*cnt == 0) delete ptr;
        return ;
    }
    bool operator==(const ptr_data &p) const {
        return ptr == p.ptr && cnt == p.cnt;
    }

    A *ptr;
    int *cnt;// 存储一个引用计数指针
};

class shared_ptr {
public:
    shared_ptr(A *ptr) : p(ptr) {}
    shared_ptr(const shared_ptr &p) : p(p.p) {
        this->p.increase_one();
    }
    shared_ptr &operator=(const shared_ptr &obj) {
        if (obj.p == p) return *this;//判断是否指向同一对象
        p.decrease_one();
        p = obj.p;
        p.increase_one();
    }
    A *operator->() { return p.ptr; }
    A &operator*() { return *(p.ptr); }
    int use_count() { return *p.cnt; }
    ~shared_ptr() {
        p.decrease_one();
    }
private:
    ptr_data p;
};
shared_ptr make_shared() {
    return shared_ptr(new A());
}
int main() {
    shared_ptr p = make_shared(); //RVO返回值优化
    p->x = 3, p->y = 4;
    cout << *p << endl;
    cout << p.use_count() << endl;
    shared_ptr q = p;
    cout << p.use_count() << endl;
    cout << q.use_count() << endl;
    p = nullptr;
    cout << q.use_count() << endl;
    q = nullptr;
    return 0;
}

ENDS(test3)


int main() {
    // test1::main();
    // test2::main();
    test3::main();
    return 0;
}
```

迭代器本质上就是一个指针对象。

关于大小的比较，可以都用重载小于号来实现：

```
bool operator<(const RandomIter &iter) const {
    return ptr < iter.ptr;
}
bool operator>(const RandomIter &iter) const {
    return iter < *this;
}
bool operator<=(const RandomIter &iter) const {
    return !(iter < *this);
}
bool operator>=(const RandomIter &iter) const {
    return !(*this < iter);
}
bool operator==(const RandomIter &iter) const {
    return !(*this < iter) && !(iter < *this);
}
bool operator!=(const RandomIter &iter) const {
    return !(*this == iter);
}
```

14.sort.cpp

```
#include<iostream>
#include<algorithm>
#include<functional>
using namespace std;
#define BEGINS(x) namespace x {
#define ENDS(x) }

BEGINS(test1)
bool cmp1(int a, int b) {
    return a > b;
}
void output(int *first, int *last, const char *msg) {
    cout << msg;
    while (first != last) {
        cout << *first << " ";
        ++first;
    }
    cout << endl;
    return ;
}
class CMP {
public:
    bool operator()(int a, int b) {
        return a < b;
    }
};
int main() {
    int arr[100], n;
    cin >> n;
    for (int i = 0; i < n; i ++) cin >> arr[i];
    sort(arr, arr + n);
    output(arr, arr + n, "none : ");

    sort(arr, arr + n, cmp1);//普通函数
    output(arr, arr + n, "cmp1 : ");

    CMP cmp2;//函数对象
    sort(arr, arr + n, cmp2);
    output(arr, arr + n, "cmp2 : ");
    return 0;
}
ENDS(test1)


BEGINS(test2)
bool cmp1(int a, int b) {
    return a > b;
}
void output(int *first, int *last, const char *msg) {
    cout << msg;
    while (first != last) {
        cout << *first << " ";
        ++first;
    }
    cout << endl;
    return ;
}
class CMP {
public:
    bool operator()(int a, int b) {
        return a < b;
    }
};
//普通版实现
/*
void sort(int *first, int *last, function<bool(int, int)> cmp = less<int>()) {
    if (first >= last) return ;
    int *x = first, *y = last - 1, z = *first;
    while (x < y) {
        while (x < y && cmp(z, *y)) --y;
        if (x < y) *(x++) = *y;
        while (x < y && cmp(*x, z)) ++x;
        if (x < y) *(y--) = *x;
    }
    *x = z;
    sort(first, x, cmp);
    sort(x + 1, last, cmp);
    return ;
}
*/
const int threshold = 16;

void intro_loop(int *first, int *last, function<bool(int, int)> cmp = less<int>()) {
    while (last - first > threshold) {
        int *x = first, *y = last - 1, z = *first;
        do {
            while (cmp(*x, z)) ++x;
            while (cmp(z, *y)) --y;
            if (x <= y) {
                swap(*x, *y);
                ++x; --y;
            }
        } while (x <= y);
        intro_loop(x, last, cmp);
        last = y + 1;
        return ;
    }
}
void insertion_sort(int *first, int *last, function<bool(int, int)>cmp = less<int>()) {
    int *ind = first;
    for (int *x = first + 1; x < last; ++x) {
        if (cmp(*x, *ind)) ind = x;
    }
    while (ind != first) {
        swap(*ind, *(ind - 1));
        --ind;
    }
    for (int *x = first + 2; x < last; ++x) {
        int *j = x;
        while (cmp(*j, *(j - 1))) {
            swap(*j, *(j - 1));
            --j;
        }
    }
    return ;
}
void sort(int *first, int *last, function<bool(int, int)>cmp = less<int>()) {
    intro_loop(first, last, cmp);
    insertion_sort(first, last, cmp);
    return ;
}
int main() {
    int arr[100], n;
    cin >> n;
    for (int i = 0; i < n; i ++) cin >> arr[i];
    sort(arr, arr + n);
    output(arr, arr + n, "none : ");

    sort(arr, arr + n, cmp1);//普通函数
    output(arr, arr + n, "cmp1 : ");

    CMP cmp2;//函数对象
    sort(arr, arr + n, cmp2);
    output(arr, arr + n, "cmp2 : ");
    return 0;
}
ENDS(test2)
```

```cpp
BEGINS(test3)
bool cmp1(int a, int b) {
    return a > b;
}
void output(int *first, int *last, const char *msg) {
    cout << msg;
    while (first != last) {
        cout << *first << " ";
        ++first;
    }
    cout << endl;
    return ;
}
class CMP {
public:
    bool operator()(int a, int b) {
        return a < b;
    }
};
const int threshold = 16;
class RandomIter {
public:
    RandomIter(int *a) : ptr(a) {}
    int operator-(const RandomIter &a) { return ptr-a.ptr; }
    RandomIter operator-(int n) { return RandomIter(ptr - n); }
    RandomIter operator+(int n) { return RandomIter(ptr + n); }
    int &operator*() { return *ptr; }
    RandomIter &operator++() { ++ptr; return *this; }
    RandomIter &operator--() { --ptr; return *this; }
    bool operator<(const RandomIter &a) const { return ptr < a.ptr; }
    bool operator>(const RandomIter &a) const { return a < *this; }
    bool operator<=(const RandomIter &a) const { return !(a < *this); }
    bool operator>=(const RandomIter &a) const { return !(*this < a); }
    bool operator==(const RandomIter &a) { return !(*this < a) && !(a < *this); }
    bool operator!=(const RandomIter &a) { return !(*this == a); }

private:
    int *ptr;
};

void intro_loop(RandomIter first, RandomIter last, function<bool(int, int)> cmp = less<int>()) {
    while (last - first > threshold) {
        RandomIter x = first, y = last - 1;
        int z = *first;
        do {
            while (cmp(*x, z)) ++x;
            while (cmp(z, *y)) --y;
            if (x <= y) {
                swap(*x, *y);
                ++x; --y;
            }
        } while (x <= y);
        intro_loop(x, last, cmp);
        last = y + 1;
        return ;
    }
}
void insertion_sort(RandomIter first, RandomIter last, function<bool(int, int)>cmp = less<int>())
    RandomIter ind = first;
    for (RandomIter x = first + 1; x < last; ++x) {
        if (cmp(*x, *ind)) ind = x;
    }
    while (ind != first) {
        swap(*ind, *(ind - 1));
        --ind;
    }
    for (RandomIter x = first + 2; x < last; ++x) {
        RandomIter j = x;
        while (cmp(*j, *(j - 1))) {
            swap(*j, *(j - 1));
            --j;
        }
    }
    return ;
}
void sort(RandomIter first, RandomIter last, function<bool(int, int)>cmp = less<int>()) {
    intro_loop(first, last, cmp);
    insertion_sort(first, last, cmp);
    return ;
}
int main() {
    int arr[100], n;
    cin >> n;
    for (int i = 0; i < n; i ++) cin >> arr[i];
    sort(arr, arr + n);
    output(arr, arr + n, "none : ");

    sort(arr, arr + n, cmp1);//普通函数
    output(arr, arr + n, "cmp1 : ");

    CMP cmp2;//函数对象
```

```cpp
    sort(arr, arr + n, cmp2);
    output(arr, arr + n, "cmp2 : ");
    return 0;
}
ENDS(test3)

int main() {
    // test1::main();
    // test2::main();
    test3::main();
    return 0;
}
```