(2013)

Stroustrup ch 18

(1)

ch. 18 Vectors & Arrays

objects - always have fixed sizeof (size_t)
- one specific address
- memcpy not effective
- aux. variable data on heap.

18.2 Copying

- pointer copy
- shallow copy (memcpy) = move
- deep (recursive) copy

```
T a;
T b(a); // direct ctor
T b = a; // copy ctor
f(b) // copy ctor by value
return b; // copy ctor
         or move ctor
```

b = a; copy oper= unless a is rvalue. then move oper=

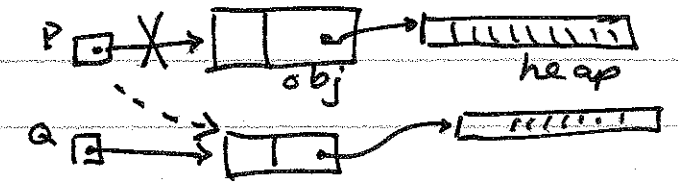
Copy ctor - makes deep copy of object
- default just uses memcpy

Move ctor - uses memcpy to move object
- steals value
- nulls out moved-from object

Stroustrup ch 18

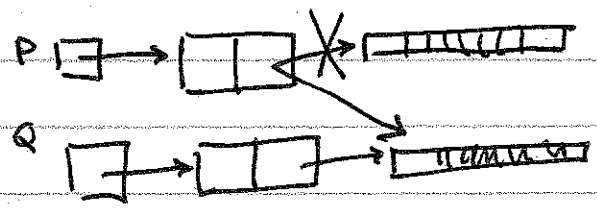
(2)

pointer copy



p = Q;

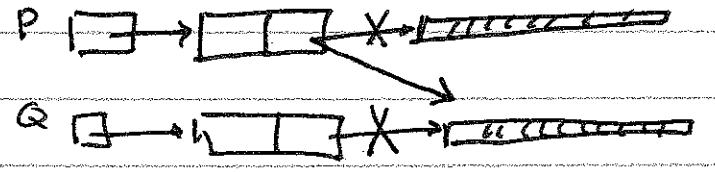
shallow (memcpy) copy



*p = *Q

now if delete Q then p is dangling

shallow move op =



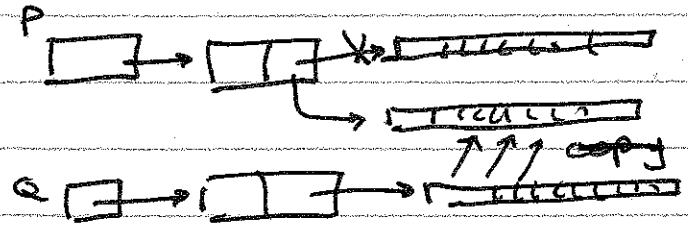
*p = move(*Q);

≡ delete p → heap

p → xp = Q → xp

Q → xp = NULL

deep copy (copy ctor or copy op=)



free p → xxx

alloc new p → xxx

copy from Q → xxxx

Stroustrup ch18

(3)

18.3 Essential ~~f~~operations

default ctor
copy ctor
move ctor (C++11)
copy op=
move op= (C++11)
dtor

other ctors [explicit / implicit]
other op=
~~explicit~~
mutators
accessors (const fns)

Review: syntax big 6

```
foo();
foo(const foo&)
foo(foo&&) // c++11
foo &operator=(const foo&)
foo &operator=(foo&&) //c++11
~foo()
```

18.3.1 Explicit ctors & default parans

~~class complex {~~

```
class vector {
explicit vector(int n);
```

prevents f(3) for f(vector v)

// can't mean f(vector(3))

c++11 apply to all ctors if ∃ list initializers

Stroustrup ed 8
4

```
class complex {
private: double real; double imag;
public:
    complex (double re=0, double im=0):
                real(re), imag(im) { }
    // 3 ctors
    // default ctor
    //   complex c ≡ complex c(0,0)
    //   complex(x) ≡ complex(x,0)
```

if fn f(complex x) then
f(6) means f(complex(6))
≡ f(complex(6,0))

since no ptrs - default (shallow) members OK

```
static const complex I(0,1);
```

---

for many arith make += basic & used by +

```
complex operator+=(const complex &t){
        real += t.real
        imag += t.imag
}
complex operator+(const complex &t){
        complex r(*this);
        r+=t;
        return r; //NOTE: by value
}
```

Stroustrup
5 ch 18

let ++ leverage +=

```
complex & operator ++() {
        *this += 1
}
//conversion ↗ int → double → complex
complex operator ++ (int) {
        complex r(*this)
        ++*this
        return r;
}
```

Fields - all primitive or objects — default ctor etc OK

- any field pointer — suppress all default members
  - write explicitly or = delete
- reference fields — suppress defaults
  - must be init by initializers
  - ~~C~~ C++ assign = in body of ctor

Stroustrup 6 ch18

18.4 Access Vector

```
vector {
    size_t size_
    double *data
    double &operator[](int i){return &data[i]}
    double operator[](int i) const
        {ret data[i]}
```

can't use on const vector

```
v[i] = a //uses first
a = v[i] //uses second for const vector
```

Note: non const return reference

problems with arrays

- hard to initialize
- don't know their own size
- just blocks of storage
- implicit convert to pointers

int a[10] a is same as &a[0]

a[i] ≡ *(a+i)

pointer arithmetic

"___" is a char array not a string

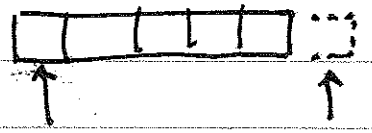
but

string::string(const char *const)

is a ctor

Stroustrup
7@ ch.18

Iterators work like ptrs



```
for (i = v.begin(); i != v.end; ++i) f(*i)
```

```
for(p = a; p != a + n; ++p) f(*p)
```

a is an array say int []
n is its len
i is an int *

operators on ptrs

```
int *p; int i
```

p + i } ⇒ int *
i + p }

p - p produces ptrdiff_t
an unsigned integer

p+p ⟶ error

p[i] ≡ *(p + i)

ex:

```
size_t strlen(const char *s){
  const char *t = s;
  while(*t) ++t;
  return t - s;
}
```

~~const char *const char *v;~~

Stroustrup 8 ch. 18

```
const char *const char *const var;
```

- the chars are const
- the array = const
- var is const

18.5.4 Pointer problems

- dereference NULL
- use uninitialized ptrs
- off either end of array
- dangling: access deleted object
- ptr to object out of scope
- memory leak: failure to free

ex:

```
ispalindrome (const char *s) {
    size_t i = 0;
    size_t e = strlen(s) - 1;
    while (i<e) {
        if( s[i++] != s[e--]) return false
    }
    return true
```